spytek_mikolaj_pd5

May 17, 2021

1 Praca domowa 5

Mikołaj Spytek

Celem tej pracy domowej jest zastosowanie co najmniej dwóch metryk do wyboru odpowiedniej liczby klastrów, dwóch metod klastrowania i wizualizacja wyników.

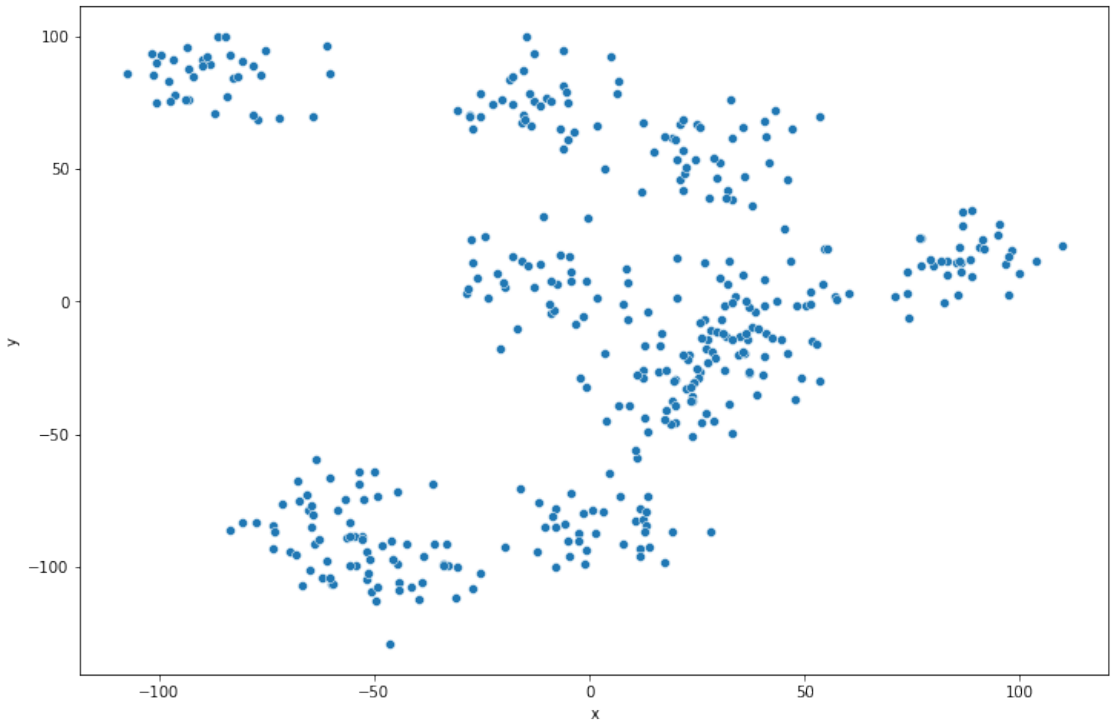
[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df = pd.read_csv("clustering.csv", names=["x","y"])
df.head()
```

[2]:

	x	y
0	41.788907	52.220182
1	-96.586516	90.957033
2	-54.143591	-99.153377
3	19.929231	-45.859779
4	-82.941076	84.099186

[3]:
```
# dzięki temu, że dane są w R^2, możemy po prostu na nie popatrzeć
plt.figure(figsize=(12,8))
sns.scatterplot(x="x", y="y", data=df)
plt.show()
```



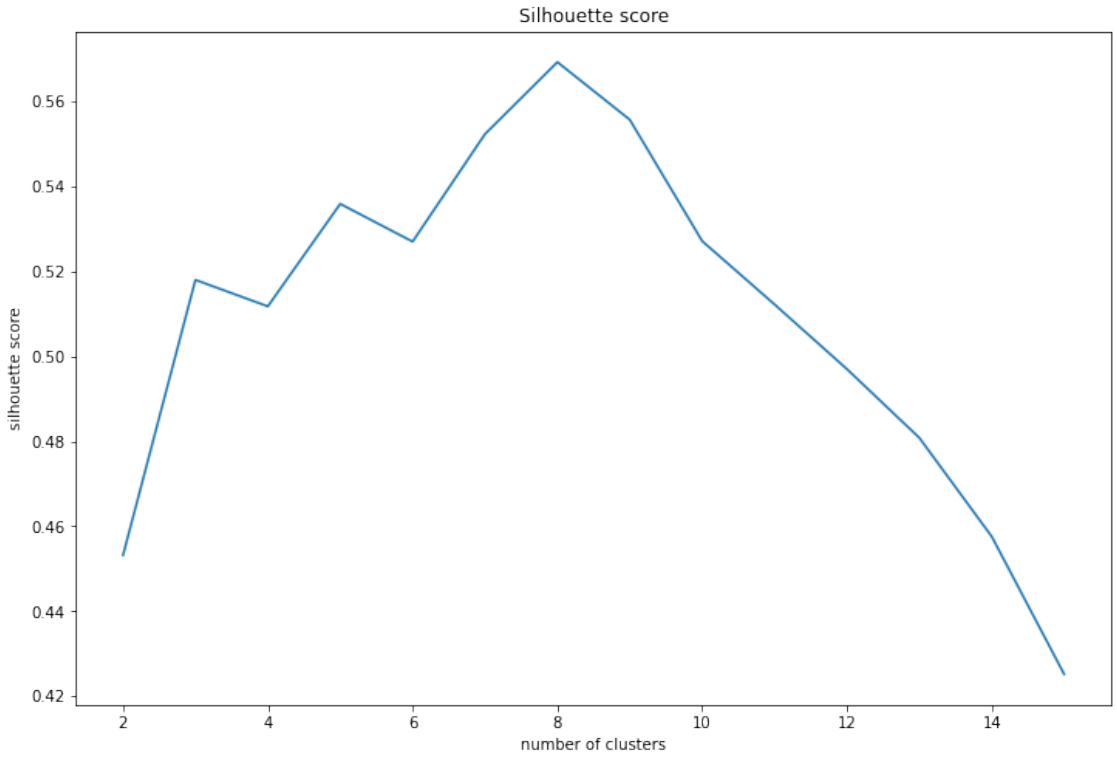
1.1 Metoda 1 - KMeans

Jako pierwszej metody klasteryzacji użyję KMeans. Aby dobrać odpowiednią liczbę klastrów sprawdzę metrykę Silhouette score oraz metodę, która na laboratorium nazwana była metodą łokcia - policzymy sumę kwadratów odległości od centrum klastra dla każdego klastra i optycznie na wykresie ocenimy gdzie występuje największe przegięcie.

Na początku spróbuję dobrać optymalną liczbę klastrów za pomocą silhouette score. Miara ta mierzy średnią odległość próbek w klastrze, oraz średnią odległość między klastrami i wyznacza stosunek różnicy tych dwóch liczb do maksimum z nich. Taka definicja pozwala wnioskować, że najlepsze klastrowanie jest dla tej liczby klastrów, dla której miara ta jest największa.

[4]:
```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

scores = []

for number_of_clusters in range(2, 16):
    kmeans = KMeans(n_clusters=number_of_clusters)
    kmeans.fit(df)
    clusters = kmeans.predict(df)
    scores.append(silhouette_score(df, clusters))

x = [i for i in range(2,16)]
```

```
plt.figure(figsize=(12,8))
plt.plot(x, scores)
plt.xlabel("number of clusters")
plt.ylabel("silhouette score")
plt.title("Silhouette score")
plt.show()
```

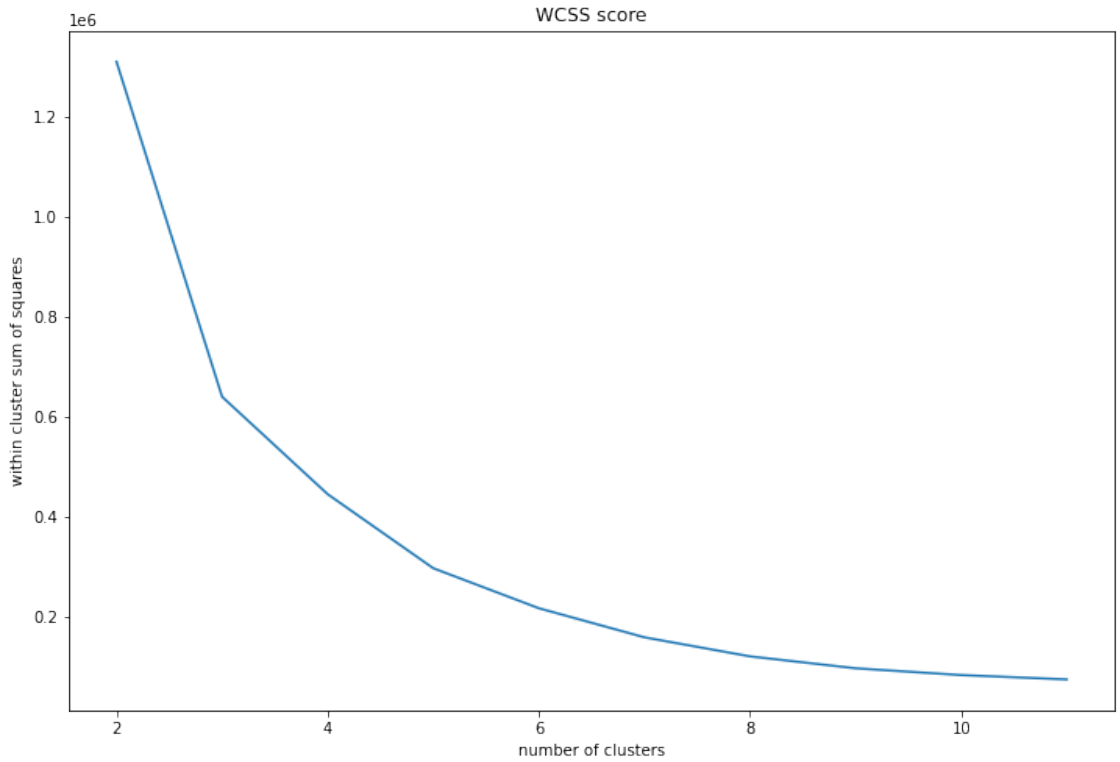


Łatwo zauważyć, że metoda silhouette wskazuje, że optymalną liczbą klastrów będzie 8.

Postanowiłem też sprawdzić, czy taki sam wynik można otrzymać metodą łokcia.

[5]:
```
scores = []

for num_clusters in range(2, 12):
    kmeans = KMeans(n_clusters = num_clusters)
    kmeans.fit(df)
    score = kmeans.score(df) * (-1)
    scores.append(score)

x = [i for i in range(2,12)]

plt.figure(figsize=(12,8))
```

```
plt.plot(x, scores)
plt.xlabel("number of clusters")
plt.ylabel("within cluster sum of squares")
plt.title("WCSS score")
plt.show()
```

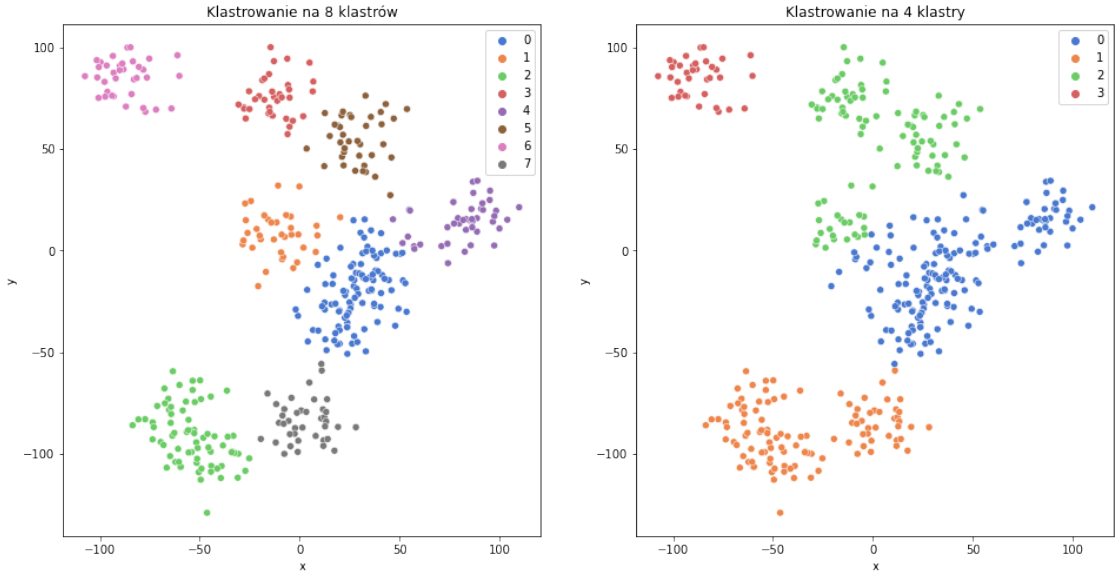


Tutaj nie mamy już wyraźnego wskazania. Jeżeli już miałbym się zdecydować, to powiedziałbym, że największe przegięcie występuje w punkcie 3, a więc optymalną liczbą klastrów byłoby 4, co zupełnie nie pokrywa się ze wskazaniem poprzedniej metody.

Ponieważ mamy ten komfort, że dane są w $\mathbb{R}^2$, możemy je zwizualizować i zobaczyć jaka liczba klastrów jest lepsza:

[6]:
```
kmeans1 = KMeans(n_clusters=8)
kmeans1.fit(df)
kmeanslabels1 = kmeans1.predict(df)

kmeans2 = KMeans(n_clusters=4)
kmeans2.fit(df)
kmeanslabels2 = kmeans2.predict(df)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,8))
```

```
sns.scatterplot(x="x", y="y", data=df, hue=kmeanslabels1, ax = ax1, ␣
 ↪palette=sns.color_palette('muted', n_colors=8))
sns.scatterplot(x="x", y="y", data=df, hue=kmeanslabels2, ax = ax2, ␣
 ↪palette=sns.color_palette('muted', n_colors=4))
ax1.set_title("Klastrowanie na 8 klastrów")
ax2.set_title("Klastrowanie na 4 klastry")

plt.show()
```



Z powyższych wykresów widać, że metoda silhouette zadziałała dużo lepiej. Przy podziale na 4 klastry widać, że punkty, które wizualnie są w oddzielnych klastrach, zostały przydzielone do tego samego.

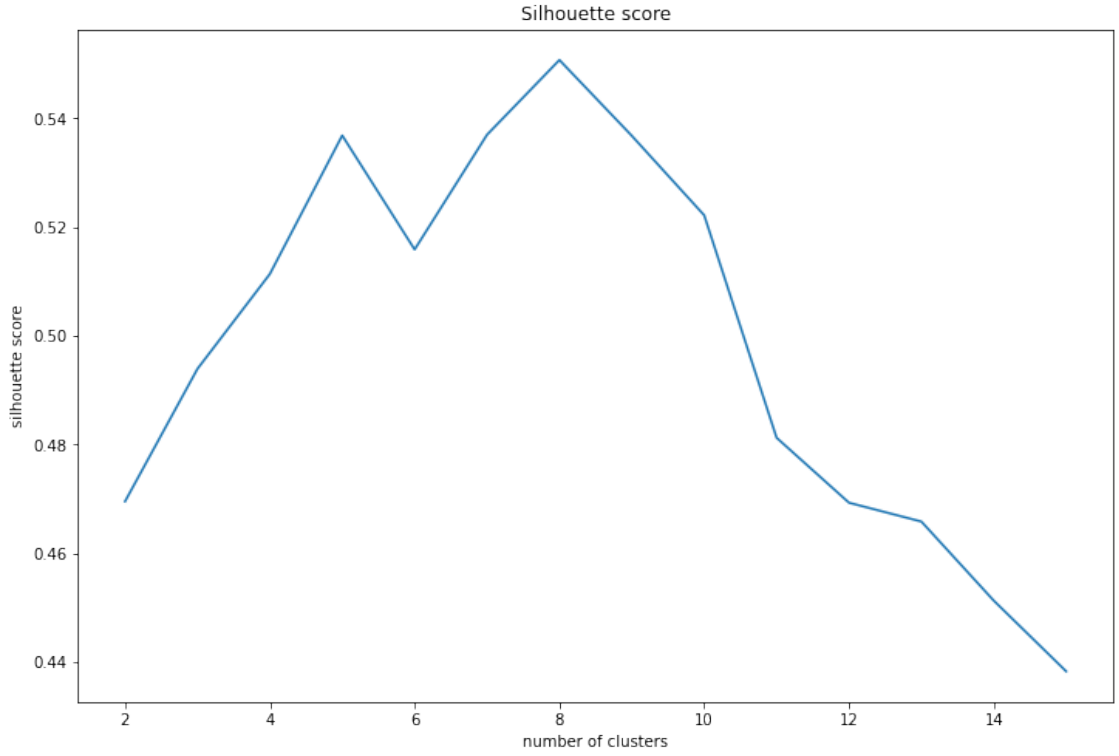
1.2 Metoda 2 - Agglomerative Clustering

Jako drugiej metody klasteryzacji użyję Agglomerative Clustering. Sposób działania tego algorytmu wygląda następująco: każda obserwacja zaczyna w osobnym klastrze, a następnie są one iteracyjnie łączone. Z domyślną metodą łączenia (ward), kryterium połączenia jest takie, aby nowe klastry miały jak najmniejszą wariancję. Proces ten powtarzany jest, aż zostanie tylko żądana liczba klastrów.

Ponieważ ostatnio ta metryka sprawdziła się bardzo dobrze zastosujmy silhouette score jeszcze raz.

[7]:
```
from sklearn.cluster import AgglomerativeClustering


from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
```

```
scores = []

for number_of_clusters in range(2, 16):
    clusterer = AgglomerativeClustering(n_clusters=number_of_clusters)
    clusterer.fit(df)
    clusters = clusterer.labels_
    scores.append(silhouette_score(df, clusters))

x = [i for i in range(2,16)]

plt.figure(figsize=(12,8))
plt.plot(x, scores)
plt.xlabel("number of clusters")
plt.ylabel("silhouette score")
plt.title("Silhouette score")
plt.show()
```

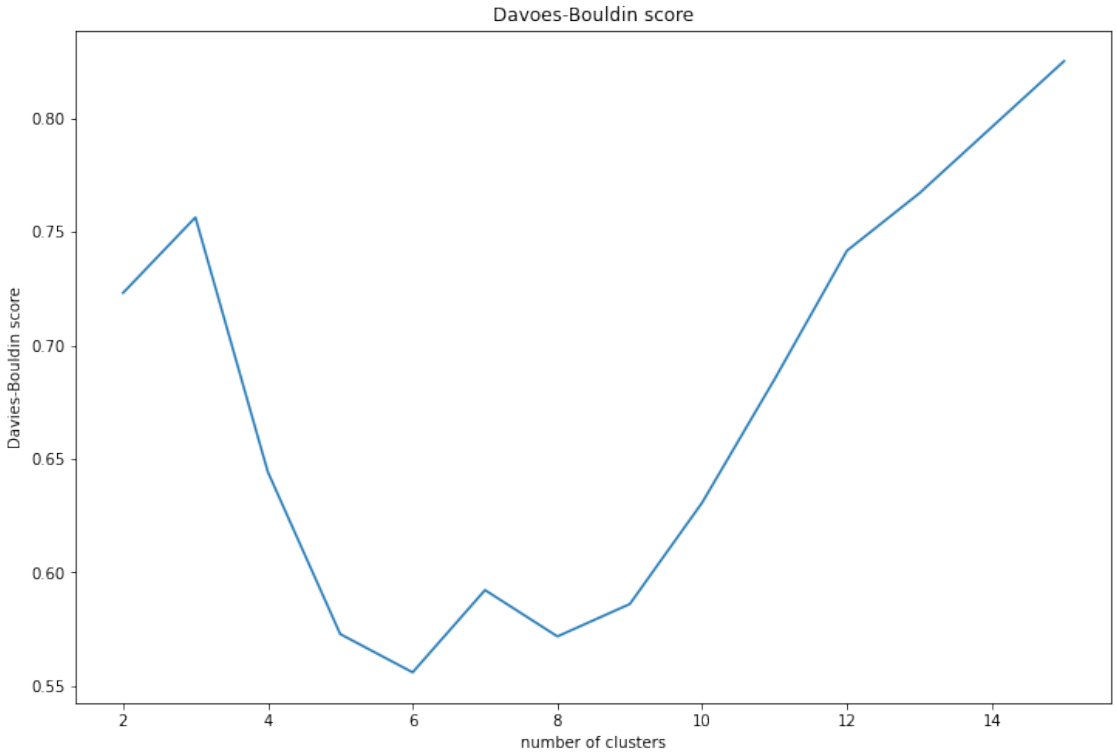


Jak widać, również w przypadku klastrowania hierarchicznego, optymalną liczbą klastrów według tej metody jest 8.

Jako drugiej metody użyłem miary Daviesa-Bouldina. Zdefiniowana jest ona jako średnia podobieństwa każdego klastra, z klastrem najbardziej do niego podobnym, a podobieństwo jest określone jako iloraz średniej odległości wewnątrz klastra do średniej odległości pomiędzy klastrami. Widzimy, że przy takiej definicji interesować nas będą niskie wartości tej miary - chcemy,

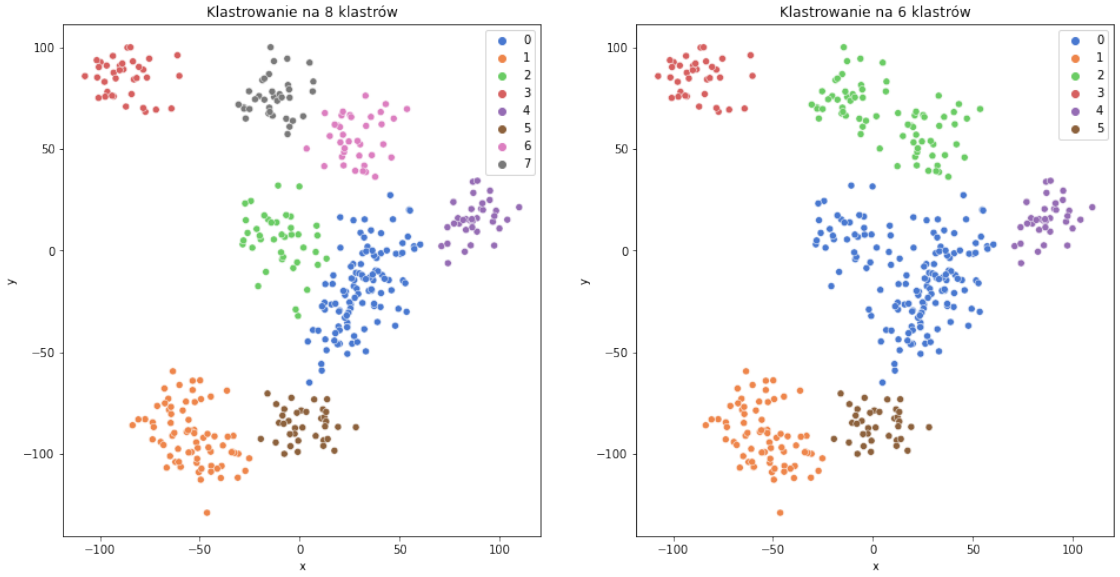
aby klastry były od siebie jak najbardziej różne.

[8]:
```
from sklearn.metrics import davies_bouldin_score
scores = []

for number_of_clusters in range(2, 16):
    clusterer = AgglomerativeClustering(n_clusters=number_of_clusters)
    clusterer.fit(df)
    clusters = clusterer.labels_
    scores.append(davies_bouldin_score(df, clusters))

x = [i for i in range(2,16)]

plt.figure(figsize=(12,8))
plt.plot(x, scores)
plt.xlabel("number of clusters")
plt.ylabel("Davies-Bouldin score")
plt.title("Davoes-Bouldin score")
plt.show()
```



Miara ta wskazuje nam, że optymalną liczbą klastrów będzie 6. Sprawdźmy i porównajmy jak wyglądają klastrowania z 6-cioma i 8-mioma klastrami.

[9]:
```
agg1 = AgglomerativeClustering(n_clusters=8)
agg1.fit(df)
agglabels1 = agg1.labels_

agg2 = AgglomerativeClustering(n_clusters=6)
agg2.fit(df)
agglabels2 = agg2.labels_

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,8))

sns.scatterplot(x="x", y="y", data=df, hue=agglabels1, ax = ax1,  palette=sns.
 ↪color_palette('muted', n_colors=8))
sns.scatterplot(x="x", y="y", data=df, hue=agglabels2, ax = ax2,  palette=sns.
 ↪color_palette('muted', n_colors=6))
ax1.set_title("Klastrowanie na 8 klastrów")
ax2.set_title("Klastrowanie na 6 klastrów")

plt.show()
```

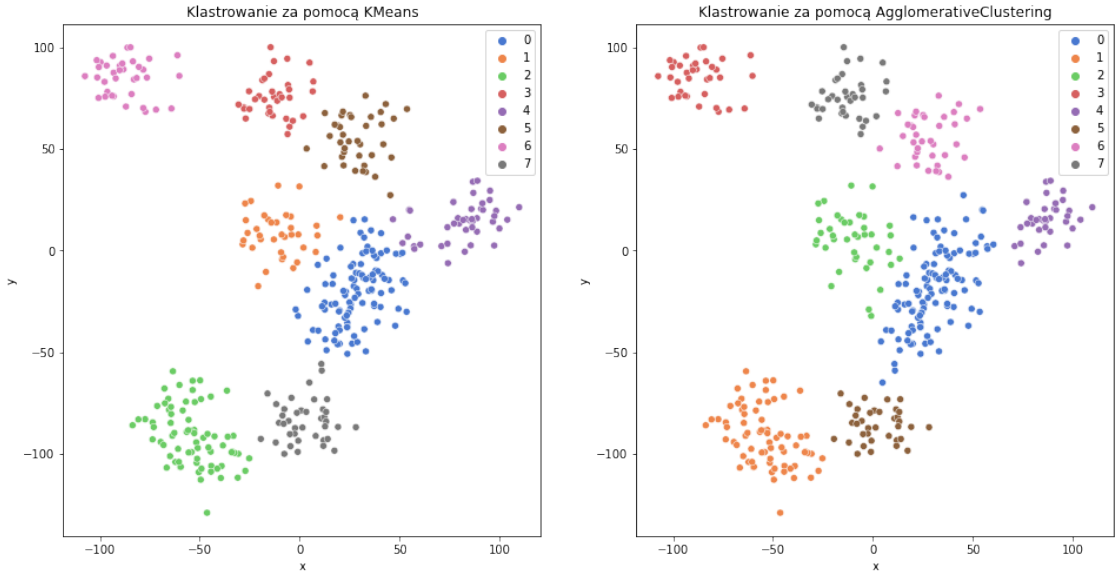


Widać wyraźnie, że 8 klastrów jest lepszą liczbą. Na przykład na prawym rysunku niebieski klaster powinien być rozbity na dwie części, tak samo jak zielony.

Można jeszcze sprawdzić, w jakich punktach optymalne klastrowania tymi dwoma metodami się różnią:

[10]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,8))
```

```
sns.scatterplot(x="x", y="y", data=df, hue=kmeanslabels1, ax = ax1, ␣
↪palette=sns.color_palette('muted', n_colors=8))
sns.scatterplot(x="x", y="y", data=df, hue=agglabels1, ax = ax2,  palette=sns.
↪color_palette('muted', n_colors=8))
ax1.set_title("Klastrowanie za pomocą KMeans")
ax2.set_title("Klastrowanie za pomocą AgglomerativeClustering")

plt.show()
```



Widać, że klastry różnią się tylko obserwacjami, które są na granicy, tak, że nawet człowiekowi ciężko byłoby określić, do którego klastra powinny należeć.