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

# zmiana sposobu wyświetlania danych typu float
pd.options.display.float_format = "{:.2f}".format
```

Najważniejszych członkiem rodziny metod uczneia nienadzorowanego jest grupowanie danych. Polega ono na podziale zbioru danych na klasy, przy wykorzystaniu podobieństwa cech między poszczególnymi obiektami. Podobieństwo to określane jest przy pomocy różnych metryk np. Euklidesowej lub miejskiej - dla zmiennych ciągłych czy Hamminga - dla zmiennych dyskretnych.

Implementacje algorytmów grupowania danych można znaleźć w pakiecie `scipy`, który jest rozbudowaną [biblioteką wykorzystywaną w obliczenia naukowych (https://www.scipy.org/index.html)](https://www.scipy.org/index.html). Na potrzeby grupowania będą z niej importowane jedynie niezbędne procedury.

1. Hierarchiczne grupowanie aglomeracyjne

Grupowanie hierarchiczne polega na iteracyjnym łączeniu w grupy (klastry) obiektów najbardziej do siebie podobnych przy pomocy przyjętej metryki. Struktura połączeń grup jest odwzorowywana w drzewie połączeń - dendrogramie. Wyróżnia się dwa podejścia do budowania hierachii grup:

1. Aglomeracyjne - polegająca na budowaniu drzewa od liści do korzenia. Początkowo każdy obiekt stanowi odrębną grupę. Następnie następuje łączenie grup. W każdej iteracji dwie najbliższe grupy są łączone w jedną, większą.
2. Rozdzielające - polegająca na na budowaniu drzewa od korzenia do liści. Najpierw wszystkie próbki uważane są za jedną grupę, po czym jest ona dzielnona na mniejsze grupy, aż do otrzymania liczby grup równej liczbie obiektów.

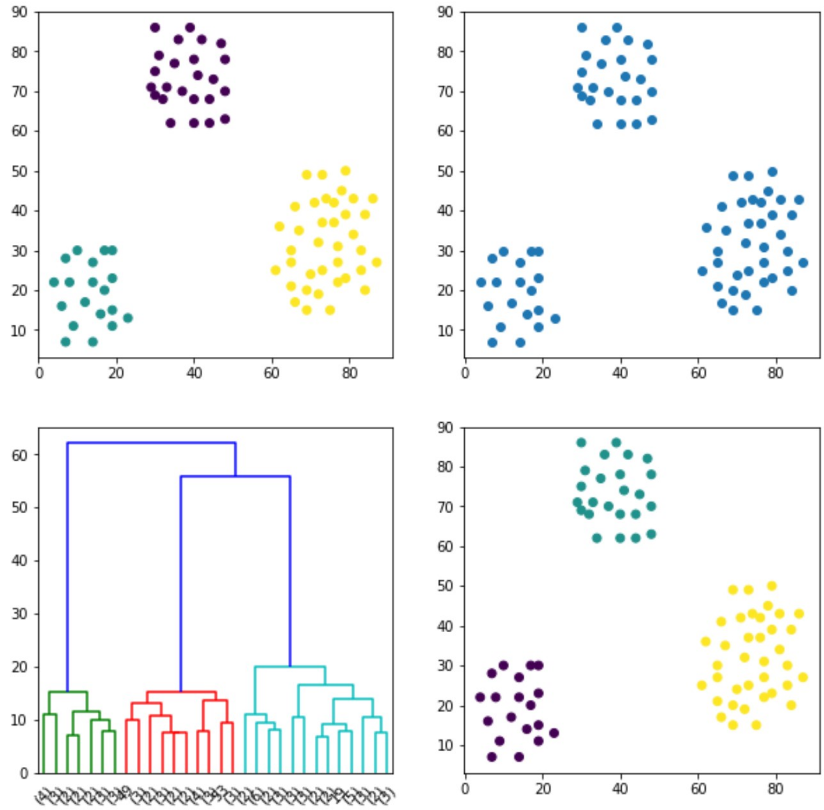
W ćwiczeniu będzie pokazana metoda aglomeracyjna.

In [14]:
```
# import niezbędnych procedur pakietu scipy i scikit.learn
from scipy.cluster.hierarchy import linkage, dendrogram, ward, fcluster
from sklearn.cluster import AgglomerativeClustering
```

Przykład poniżej pokazuje wyniki grupowania aglomeracyjnego dla zbioru `dane1`. Kolejne wykresy pokazują dane oryginalne na wykresie punktowym, dane oryginalne bez uwzględniania atrybutu decyzyjnego (wykres punktowy), dendrogram oraz wynik grupowania na wykresie punktowym. W przykładzie liczba grup jest wyznaczana automatycznie jako procent (zmienna `prog_proc`) największej odległości między grupami przypisanej do węzła dendrogramu (korzenia drzewa).

In [3]:
```
df_org = pd.read_csv('dane1.csv')
# nowa ramka bez kolumny z informacją o klasie
df = df_org.drop(columns = ['klasa'])
grupy = linkage(df, method = 'average', metric = 'euclidean')
prog_proc = 70
prog = prog_proc*max(grupy[:,2])/100
# zamiast linkage(df, method='ward', metric='euclidean') można napisać ward(df)
plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.scatter( x=df['atrybut1'], y=df['atrybut2'], c=df_org['klasa'].astype('category').cat.codes)
plt.subplot(2,2,2)
plt.scatter( x=df_org['atrybut1'], y=df_org['atrybut2'])
plt.subplot(2,2,3)
cl = dendrogram(grupy, truncate_mode='lastp', color_threshold = prog )
df['grupa']=  fcluster(grupy, prog, criterion='distance')
plt.subplot(2,2,4)
plt.scatter( x=df['atrybut1'], y=df['atrybut2'], c=df['grupa'])
```

Out[3]: <matplotlib.collections.PathCollection at 0x268c05e8860>



Zadanie Czy wynik grupowania jest zgodny z przypisaniem punktów do klas w zbiorze oryginalnym ? Zbadaj pozostałe zbiory `dane2` ,... `dane11` . Czy wyniki oryginalnego przypisania do klas i grupowania zawsze są zgodne ? Z czego wynikają różnice ? O czym świadczą ?

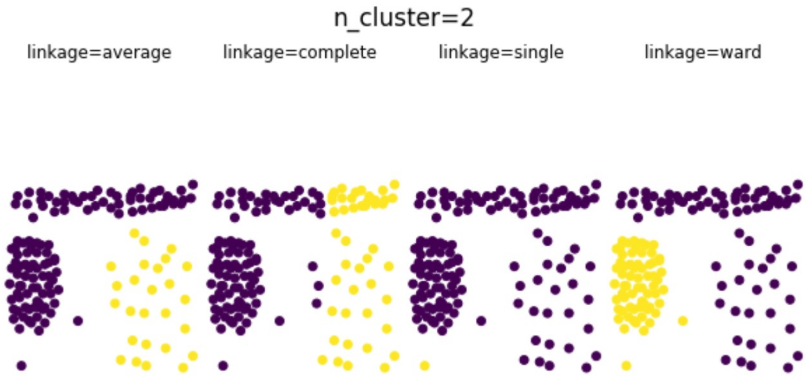
W przykładzie powyżej liczba grup jest wyznaczana automatycznie. Alternatywą jest ręczne wyznaczenie liczby grup na podstawie wiedzy zewnętrznej o zbiorze danych (jeśli takowa jest dostępna), lub na podstawie obserwacji dendrogramu.

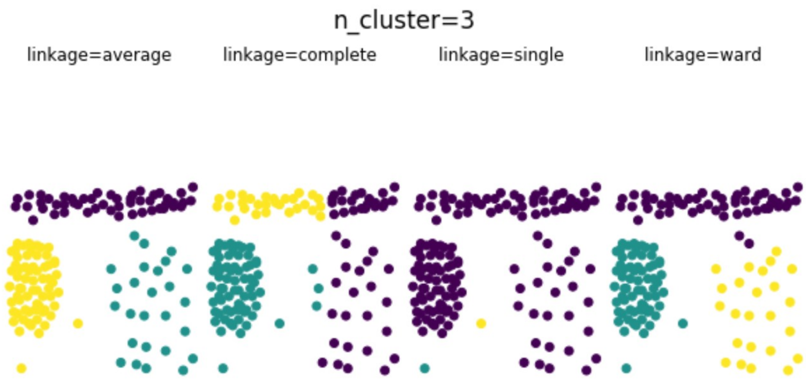
Istotnym parametrem metody jest sposób wyznaczania odległości międzygrupowej ("linkage")

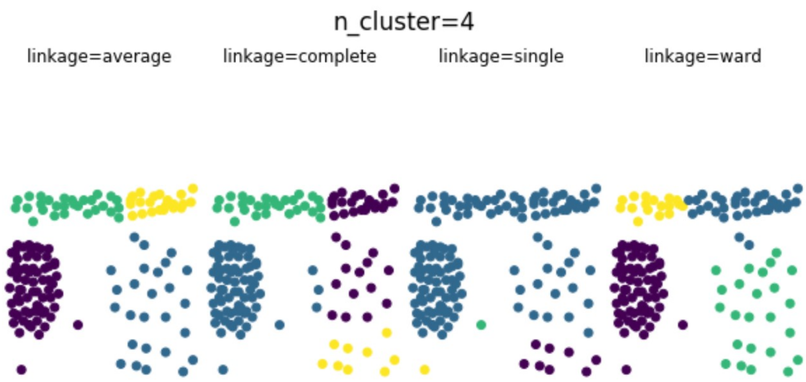
- average - wykorzystuje średnią odległość między wszystkimi obserwacjami w obu grupach
- complete - wykorzystuje maksymalną odległość między wszystkimi obserwacjami obu grup
- single - wykorzystuje minimalną odległość między wszystkimi obserwacjami obu grup
- ward - minimalizuje wariancję między łączonymi grupami (łączone są te dwie grupy, dla których wariancja po połączeniu będzie najmniejsza)

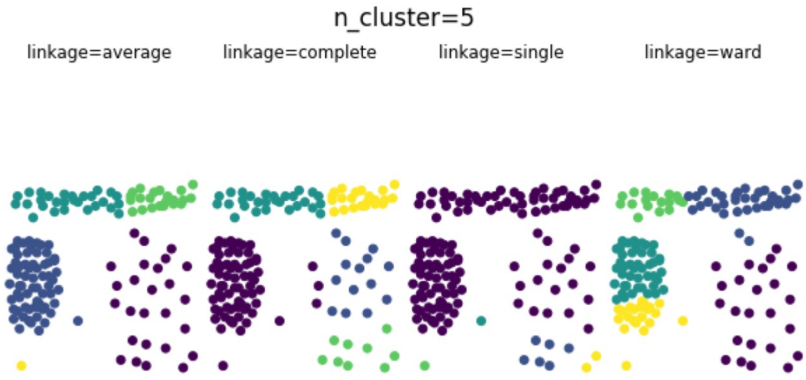
Przykład poniżej pokazuje warianty grupowania dla różnych ustalonych liczb grup i dla różnych metod wyznacznia odległości międzygrupowej.

In [4]:
```
df_org = pd.read_csv('dane6.csv')
# nowa ramka bez kolumny z informacją o klasie
df = df_org.drop(columns = ['klasa'])
for n_cluster in (2,3,4,5):
    plt.figure(figsize=(8, 5))
    for index, linkage in enumerate(('average', 'complete', 'single', 'ward')):
        plt.subplot(1, 4, index + 1)
        model = AgglomerativeClustering(linkage=linkage, n_clusters=n_cluster)
        model.fit(df)
        plt.scatter(df.atrybut1, df.atrybut2, c=model.labels_)
        plt.title('linkage=%s ' % linkage, fontdict=dict(verticalalignment='top'))
        plt.axis('equal')
        plt.axis('off')
        plt.subplots_adjust(bottom=0, top=.89, wspace=0, left=0, right=1)
        plt.suptitle('n_cluster=%i' % n_cluster, size=17)
```


n_cluster=2
linkage=average
linkage=complete
linkage=single
linkage=ward


n_cluster=3
linkage=average
linkage=complete
linkage=single
linkage=ward


n_cluster=4
linkage=average
linkage=complete
linkage=single
linkage=ward



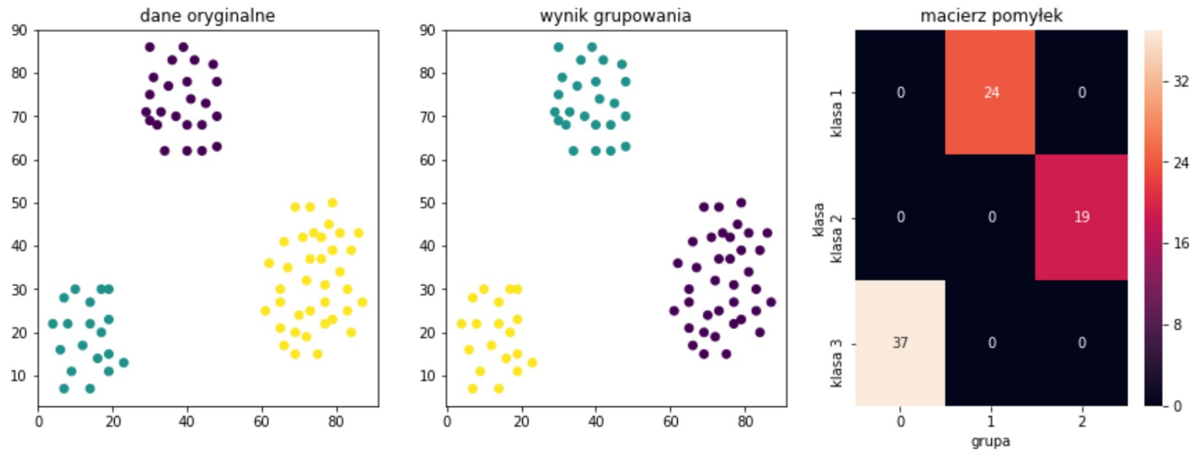
W sytuacji gdy każdemu obiektowi w zbiorze jest przypisana pewna klasa, grupowanie danych może służyć do zbadania czy klasy przypisane arbitralnie obiektom są zgodne z naturalnym przyporzadkowanie obiektów do pewnych grup. Naturalne przyporządkowanie obiektów do grup uzyskujemy przy tym w wybiku grupowania.

Różnicę między naturalnym przyporządkowanie obiektw a klasami można przedstawić w postaci macierzy pomyłek, która pokazuje jak wygląda przyporządkowanie obiektów w poszczególnych klasach konkretnym grupom. W idealnym przypadku macierz taka powinna w kazdym wierszu (kolumnie) zwierać dokładnie jeden element niezerowy. Macierz pomyłek można przedstawić graficznie w postaci mapy ciepła.

In [19]:
```
ile_grup = 3
df_org = pd.read_csv('dane1.csv')
df = df_org.drop(columns = ['klasa'])

model = AgglomerativeClustering(linkage='complete', affinity='euclidean', n_clusters=ile_grup)
model.fit(df)

klasa = df_org['klasa'].astype('category').cat.codes
grupa = model.labels_

plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
plt.scatter( x=df_org['atrybut1'], y=df_org['atrybut2'], c=klasa)
plt.title('dane oryginalne')
plt.subplot(1,3,2)
df['grupa']=  model.labels_
plt.scatter( x=df['atrybut1'], y=df['atrybut2'], c=grupa)
plt.title('wynik grupowania')
plt.subplot(1,3,3)
pomylki = pd.crosstab(df_org['klasa'],df['grupa'])
print(pomylki)
sns.heatmap(pomylki,annot = pomylki)
plt.title('macierz pomyłek')
```

```
grupa     0   1   2
klasa
klasa 1   0  24   0
klasa 2   0   0  19
klasa 3  37   0   0
```

Out[19]:
```
Text(0.5, 1.0, 'macierz pomyłek')
```



> **Zadanie** Sprawdź jak wygląda macierz pomyłek dla pozostałych zbiorów testowych ? Jak mógłbyś ją zinterpretować w każdym przypadku ?

2. Grupowanie k-średnich

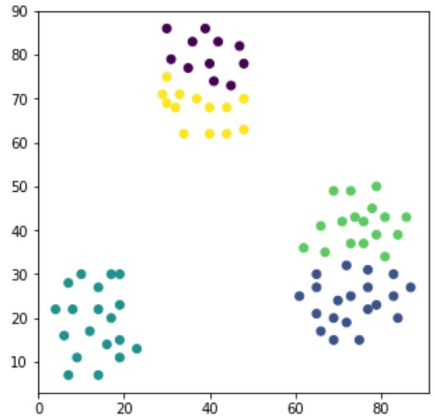
Metoda k-średnich polega na minimalizacji odległości wektorów wartości atrybutów obiektów należących do danego klastra do pewnego punktu charakterystycznego klastra (zwanego jego środkiem lub centroidem), do którego obiekty zostały przyporządkowane. Przyporządkowanie danego obiektu do klastra odbywa się poprzez porównanie jego odległości do wszystkich centroidów. Metoda ta wymaga informacji o liczbie klastrów (grup). Początkowe ich położenia wybierane są losowo albo z użyciem specjalnego algorytmu.

Algorytm grupowania k-średnich znajduje się w pakiecie `scikit-learn`

In [3]:
```
from scipy.spatial.distance import cdist
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
```

In [4]:
```
k = 5
df_org = pd.read_csv('dane1.csv')
df = df_org.drop(columns = ['klasa'])

kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(df)
etykiety_klastrow = kmeans.fit_predict(df)
plt.figure(figsize=(5,5))
plt.scatter(df.atrybut1, df.atrybut2, marker='o', c=etykiety_klastrow)
```

Out[4]: <matplotlib.collections.PathCollection at 0x2b8117c98d0>

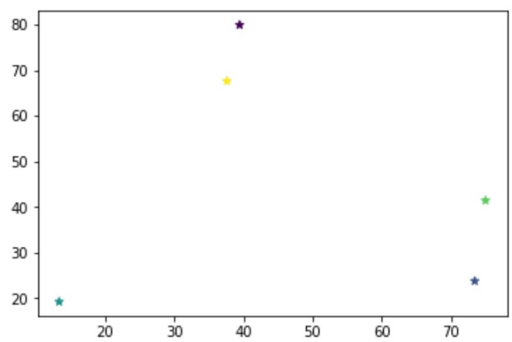


W efekcie działania algorytmu otrzymaliśmy wynik grupowania oraz ostateczne wartości centroidów grup. Ich współrzędne możemy otrzymać przy pomocy metody `cluster_centers` .

In [5]:
```
centroidy = kmeans.cluster_centers_
print ("Współrzędne centroidów: \n", centroidy)
plt.scatter(centroidy[:,0], centroidy[:,1], marker='*', c=np.array(range(k)))
```

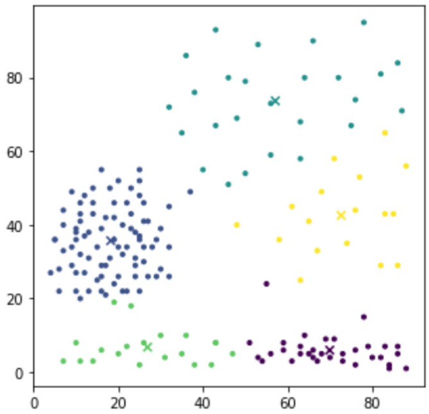
```
Współrzędne centroidów:
 [[39.45454545 79.90909091]
 [73.45       23.75      ]
 [13.36842105 19.21052632]
 [75.         41.41176471]
 [37.61538462 67.61538462]]
```

Out[5]: <matplotlib.collections.PathCollection at 0x2b811c512e8>



In [6]:
```
k = 5
df_org = pd.read_csv('dane5.csv')
df = df_org.drop(columns = ['klasa'])

kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(df)
etykiety_klastrow = kmeans.fit_predict(df)
plt.figure(figsize=(5,5))
plt.scatter(df.atrybut1, df.atrybut2, marker='.', c=etykiety_klastrow)
centroidy = kmeans.cluster_centers_
plt.scatter(centroidy[:,0], centroidy[:,1], marker='x', c=np.array(range(k)))
```

Out[6]:
```
<matplotlib.collections.PathCollection at 0x2b811cb94a8>
```



Zadanie Wykonaj grupowanie dla innej wartości `random_state` (inna wartość początkowych klastra). Proszę porównać otrzymany wykres z wcześniejszym wynikiem. Skąd wzięła się niestabilność algorytmu polegająca na istnieniu różnic w przyporządkowaniu obiektów do grup ?

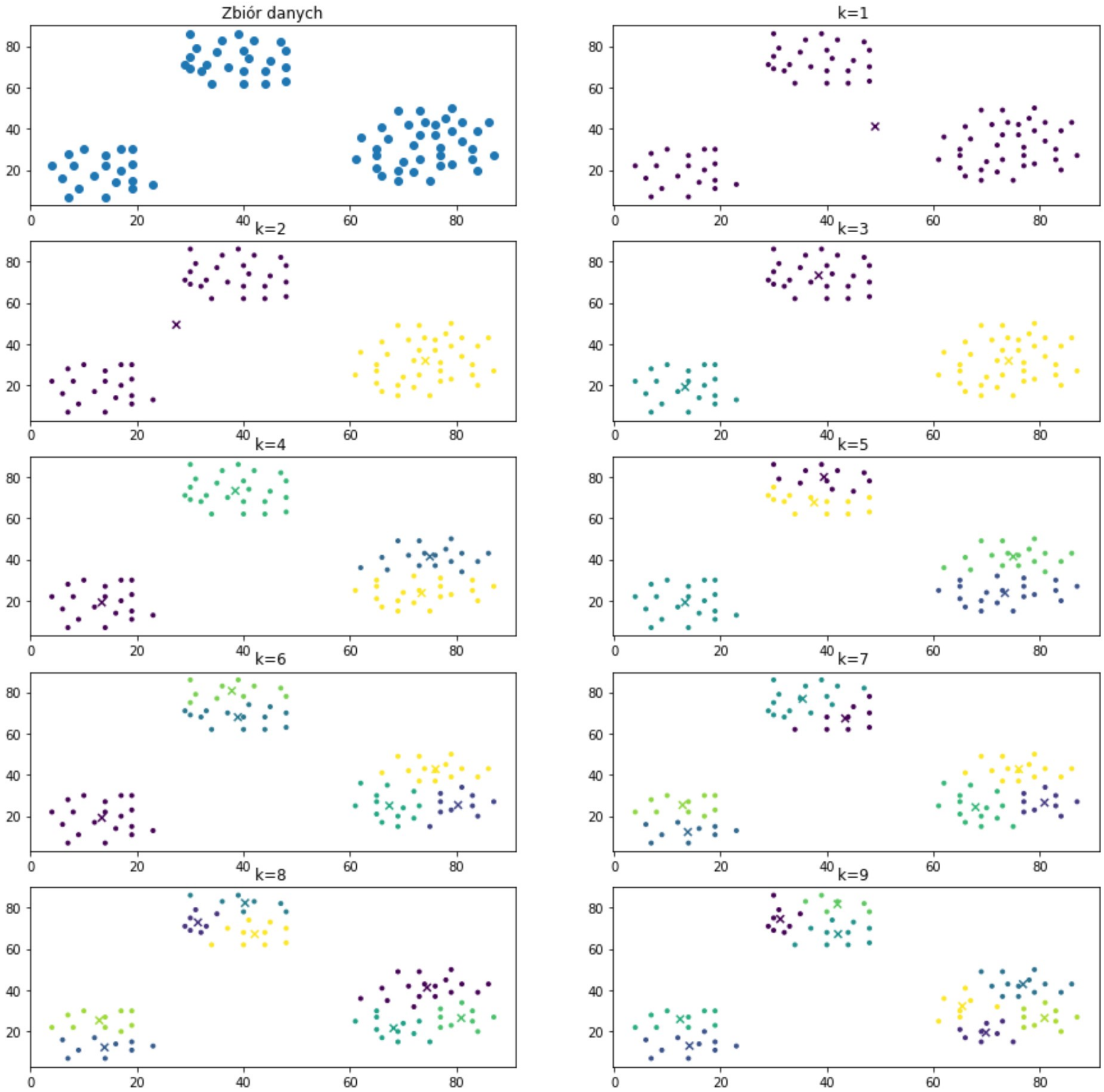
Zadanie Przetestuj metodę dla różnych wartości k oraz różnych zbiorów testowych.

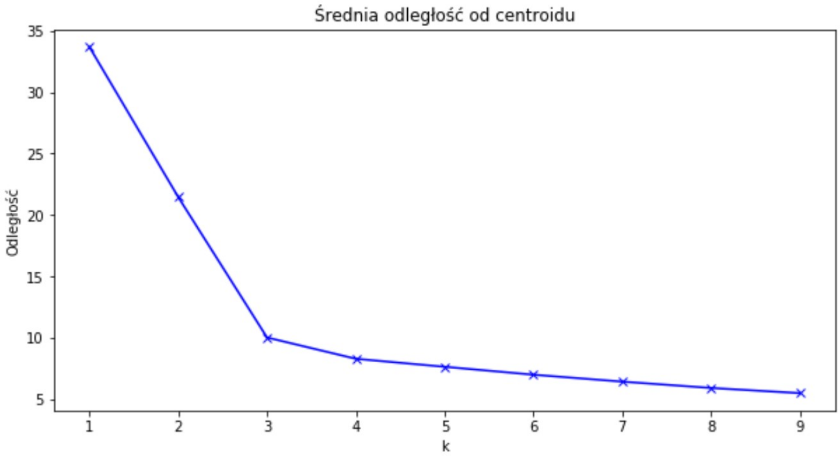
Oprócz braku stabilności, ważnym zagadnieniem związanym z algorytmem k-średnich jest dobór liczby klastrów. Z reguły nie jest ona znana, a algorytm wymaga jej do działania. Jeden ze sposobów wyznaczania optymalnej liczby k jest tzw. matoda łokciowa, polegająca na iteracyjnym grupowaniu zbioru z rosnącą liczbą klastrów. Następnie wyświetlamy wykres sumy odległości punktów od środków centroidów w funkcji liczby klastrów. Wykres takiej funkcji będzie z początku bardzo szybko maleć by później zmniejszyć swój spadek i spokojnie dążyć do 0 (które osiągnie dla liczby klastrów równej liczbie obiektów w macierzy danych). Punkt w którym obserwowane jest zahamowanie spadku nazywamy punktem łokciowym, który określa optymalną ilość grup.

Metoda łokciowa

In [12]:

```
df_org = pd.read_csv('dane1.csv')
df = df_org.drop(columns = ['klasa'])

maks_k = 9
zakres_k = range(1,maks_k)
ile_w_pionie = int(maks_k/2) + 1
plt.figure(figsize=(15,15))
plt.subplot(ile_w_pionie,2,1)
plt.scatter(df['atrybut1'], df['atrybut2'])
plt.title('Zbiór danych')
srednia_odl = []
pozycja = 2
for k in range(1,maks_k+1):
    plt.subplot(ile_w_pionie,2,pozycja)
    pozycja = pozycja + 1
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(df)
    etykiety_klastrow = kmeans.fit_predict(df)
    plt.scatter(df.atrybut1, df.atrybut2, marker='.', c=etykiety_klastrow)
    centroidy = kmeans.cluster_centers_
    plt.scatter(centroidy[:,0], centroidy[:,1], marker='x', c=np.array(range(k)))
    plt.title('k=%s ' % k)
    srednia_odl.append(sum(np.min(cdist(df, centroidy, 'euclidean'), axis=1)) / df.shape[0])

plt.figure(figsize=(10,5))
plt.plot(range(1,maks_k+1), srednia_odl, 'bx-')
plt.xlabel('k')
plt.ylabel('Odległość')
plt.title('Średnia odległość od centroidu')
plt.show()
```



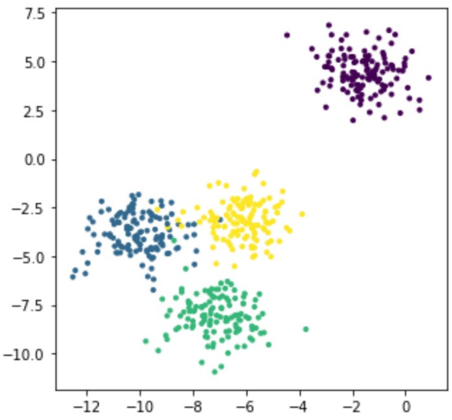


Metoda sylwetki

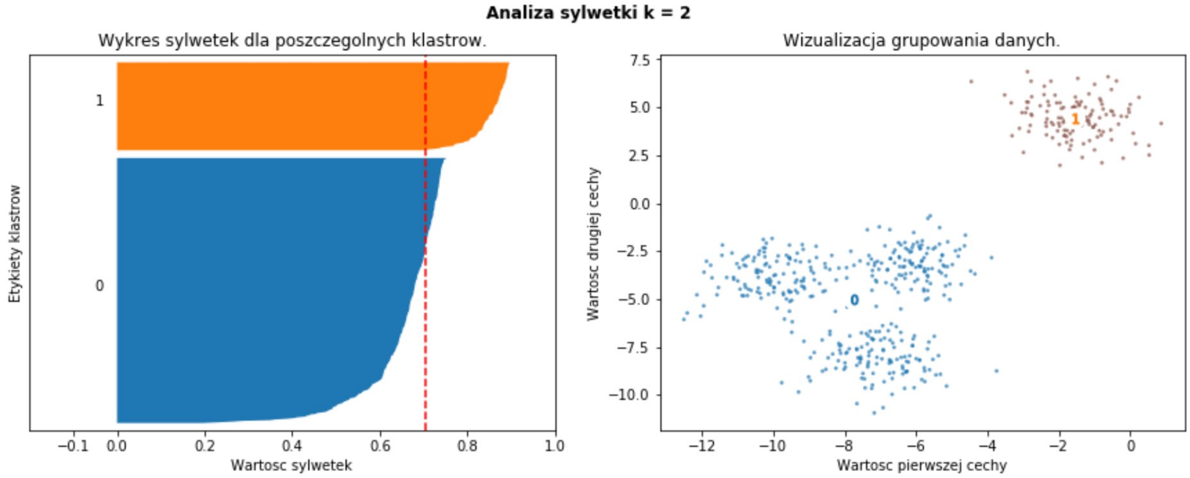
Inna metoda to tzw. metoda sylwetki danych (data silhouette) - wykorzystująca tzw. sylwetkę danych jako miarę dopasowania obiektu do własnego klastra i do klastrów sąsiednich. Miarą sylwetki dla danej obserwacji $s(i)$ nazywamy $s(i) = \frac{b(i)-a(i)}{max\{a(i),b(i)\}}$, gdzie $b(i)$ jest odległością do centroidu najbliższego klastra do którego nie została zaklasyfikowana, a $a(i)$ jest odległością od centroidu do którego została zaklasyfikowana. Liczba grup jest dobierana tak, by wartość średnia miary sylwetki była jak największa tj. jak najwięcej obiektów leżało jak najbliżej własciwych im centroidów. Miara sylwetki dla każdego punktu przyjmuje wartości między -1 a 1.

In [9]:
```
# generujemy nowy zbiór danych

X1, y1 = make_blobs(n_samples=500,
                  n_features=2,
                  centers=4,
                  cluster_std=1,
                  center_box=(-10.0, 10.0),
                  shuffle=True,    # przetasowanie kolejności próbek
                  random_state=1)  # ustawienie momentu startu zmiennej pseudolosowej w celu zapewnieni
a powtarzalności wyników
# Możemy przekształcić nasz zbiór zmiennych opisujących X1 i odpowiadających im klas y1 do jednej macie
rzy danych (pd.DataFrame)
columns = ['feature' + str(x) for x in np.arange(1, X1.shape[1]+1, 1)]
d = {key: values for key, values in zip(columns, X1.T)}
d['label'] = y1
dane1 = pd.DataFrame(d).reindex(columns=columns+['label'])

# Wyświelenie
plt.figure(figsize=(5,5))
plt.scatter(dane1.feature1, dane1.feature2, marker='.', c=dane1.label)
```

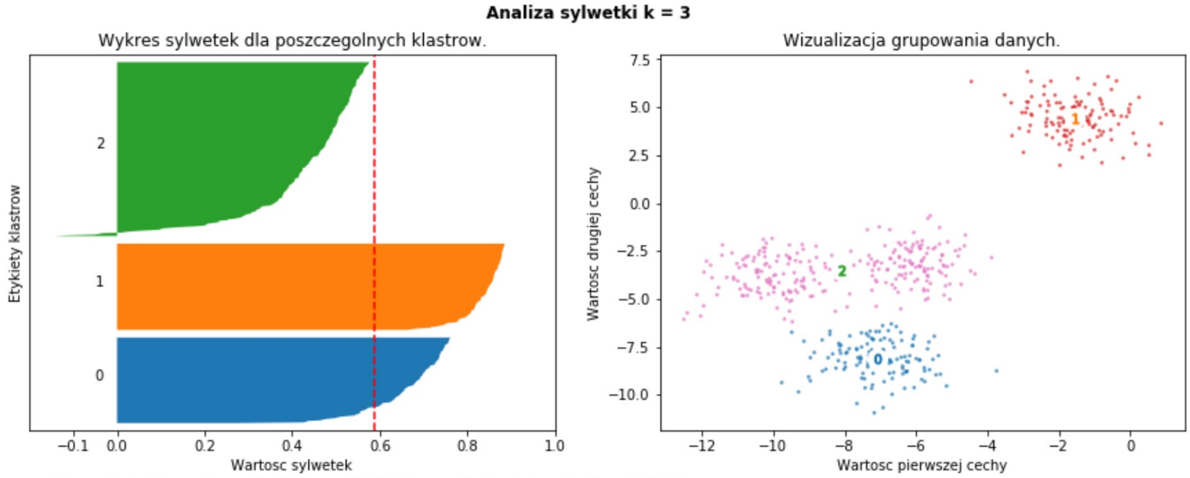
Out[9]: <matplotlib.collections.PathCollection at 0x2b811fba550>



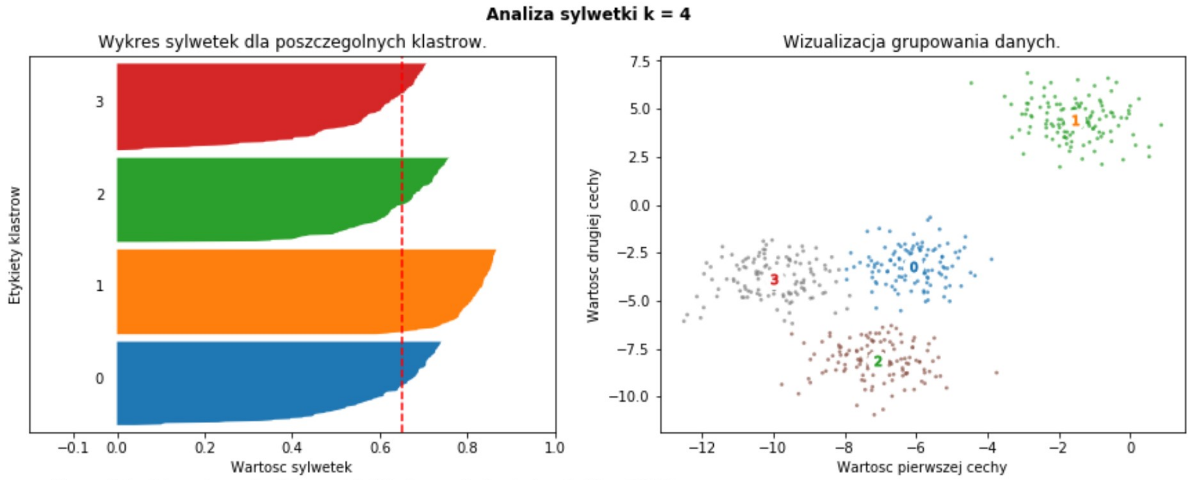
In [10]:
```
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
score = pd.Series()

for n_cluster in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(15, 5)

    ax1.set_xlim([-0.2, 1]) # zakres zmian współczynnika sylwetki
    ax1.set_ylim([0, len(dane1) + (n_cluster + 1) * 10]) # organizacja wydruku

    kmeans = KMeans(n_clusters=n_cluster, random_state=0).fit(X1)
    score = score.append(pd.Series(kmeans.inertia_))
    cluster_labels = kmeans.fit_predict(dane1[['feature1','feature2']])

    silhouette_avg = silhouette_score(dane1[['feature1','feature2']], cluster_labels)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(dane1[['feature1','feature2']], cluster_labels)

    y_lower = 10
    for i in range(n_cluster):
        # Zebranie wyników sylwetek do próbek należących do klastra i ich sortowanie
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.tab10(float(i) / n_cluster)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values)

        # Etykieta sylwetek z numerami klastrów w środku
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Wyliczenie przesunięcia w pionie dla kolejnego wykresu
        y_lower = y_upper + 10  # 10 dla kolejnej próbki

    ax1.set_title("Wykres sylwetek dla poszczegolnych klastrow.")
    ax1.set_xlabel("Wartosc sylwetek")
    ax1.set_ylabel("Etykiety klastrow")

    # Wyrysowanie wartości średniej sylwetki
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Wyczyszczenie etykiety osi Y
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # Drugi wykres będzie przedstawiał klastry
    colors = cm.tab10(cluster_labels.astype(float) / n_cluster)
    ax2.scatter(dane1.feature1, dane1.feature2, marker='.', s=30, lw=0, alpha=0.7,c=colors)

    # Etykietowanie klastrów
    centers = kmeans.cluster_centers_
    # Rysowanie białych kółek w cetroidach
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o', c="white", alpha=1, s=200)

    # Numerowanie centroidów
    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50)

    ax2.set_title("Wizualizacja grupowania danych.")
    ax2.set_xlabel("Wartosc pierwszej cechy")
    ax2.set_ylabel("Wartosc drugiej cechy")

    plt.suptitle(("Analiza sylwetki k = %d" % n_cluster),  fontweight='bold')
    plt.figtext(0.14, 0, ("Dla n = %d, średnia wartosc sylwetki wynosi: %.3f, Suma odleglosci od centro
idow: %.2f"
                  % (n_cluster, silhouette_avg, kmeans.inertia_ ) ))
    plt.show()


plt.plot(range_n_clusters, score,'bo-')
plt.title("Wykres lokciowy", fontsize=14, fontweight='bold')
plt.xlabel("Ilosc klastrow")
plt.ylabel("Suma odleglosci od centroidow")
```



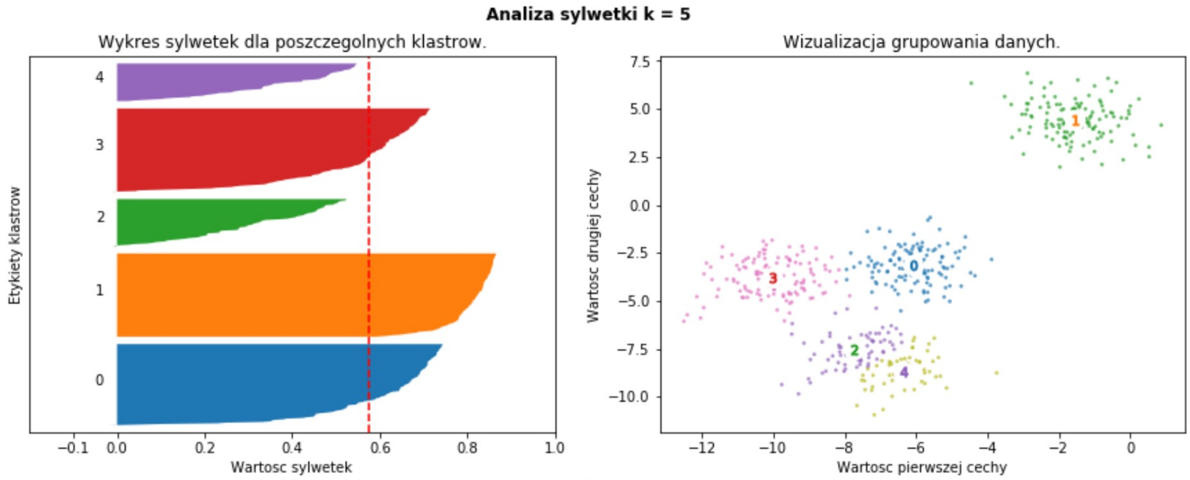
Dla n = 2, średnia wartosc sylwetki wynosi: 0.705, Suma odleglosci od centroidow: 3735.41



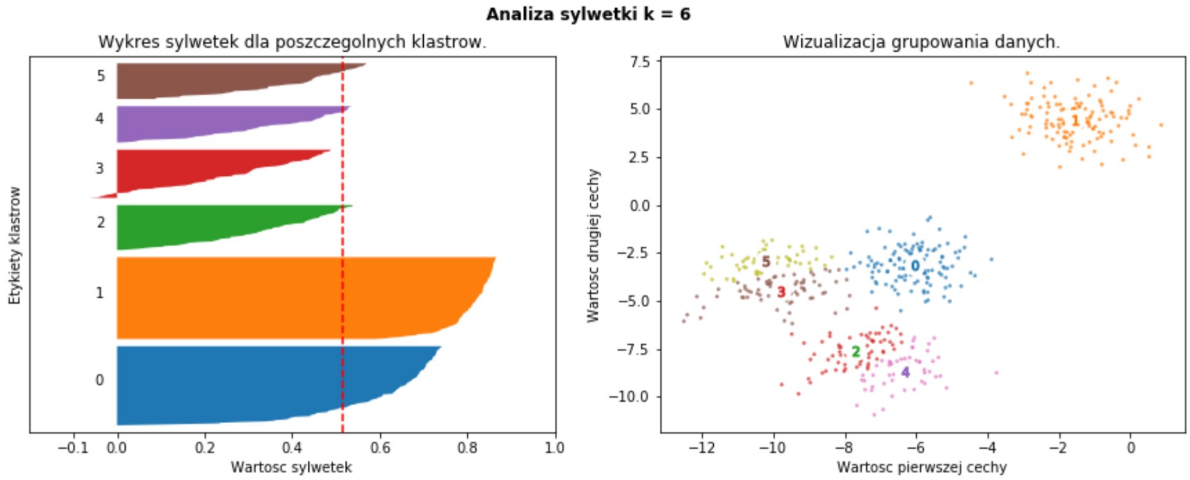
Dla n = 3, średnia wartosc sylwetki wynosi: 0.588, Suma odleglosci od centroidow: 1903.45



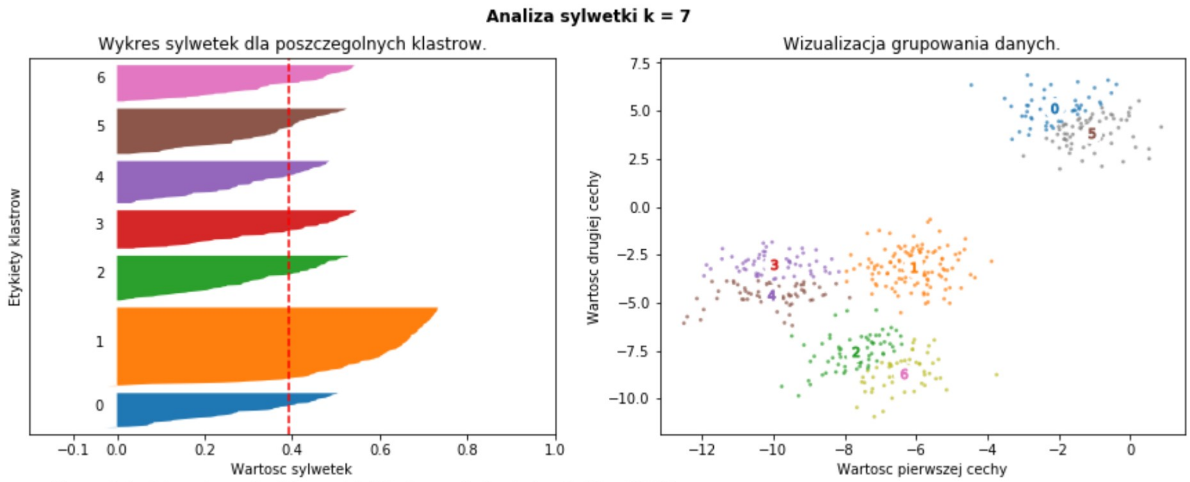
Dla n = 4, średnia wartosc sylwetki wynosi: 0.651, Suma odleglosci od centroidow: 908.39



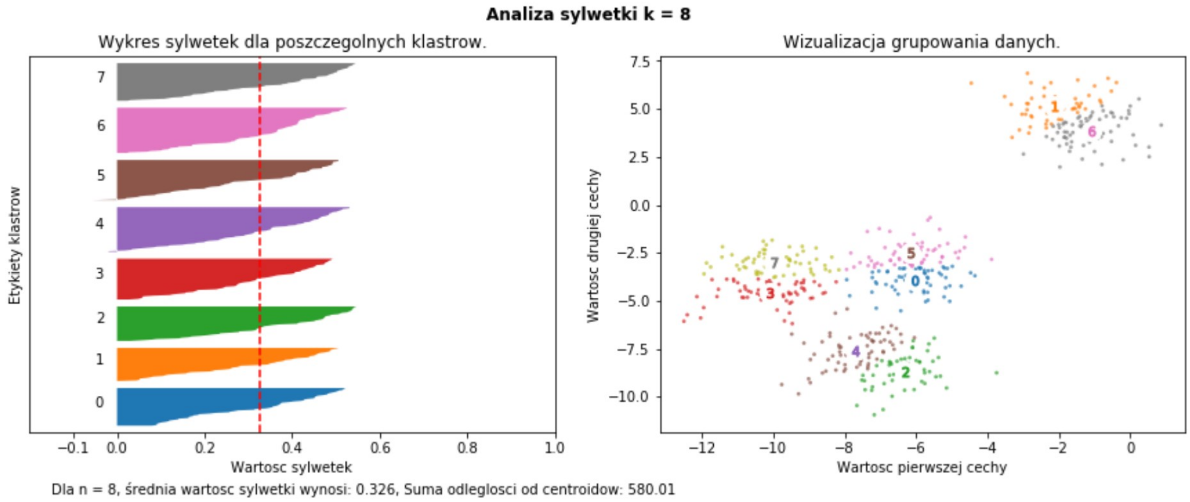
Dla n = 5, średnia wartosc sylwetki wynosi: 0.575, Suma odleglosci od centroidow: 811.08



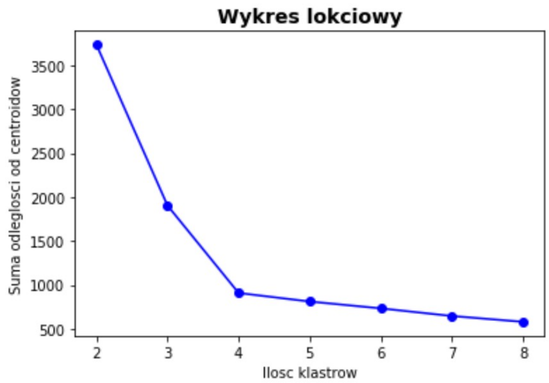
Dla n = 6, średnia wartosc sylwetki wynosi: 0.515, Suma odleglosci od centroidow: 733.15



Dla n = 7, średnia wartosc sylwetki wynosi: 0.392, Suma odleglosci od centroidow: 646.04




```
Out[10]: Text(0, 0.5, 'Suma odleglosci od centroidow')
```



Obie te metody (łokciowa i sylwetki) moga być stosowane także z innymi algorytmami grupowania, m.in. grupowaniem aglomeracyjnym.

> **Zadanie** Wykonaj grupowanie danych dla atrybutów opisujących zbioru `iris` obiema metodami, aglomeracyjną i k-średnich. Określ optymalną liczbę grup. Czy wszystkie trzy odmiany irysów są łatwe do identyfikacji (jako odróżniające się od pozostałych) w przestrzeni artybutów ?

> **Zadanie** Napisz funkcję, która będzie wizualizować metodę sylwetki i łokciową dla dowolnego zbioru danych. Pierwszym zrgumentem tej funkcji powinien być zbiór (ramka) danych, drugim - zakres zmian k, trzecim - nazwy dwóch atrybutów wyświetlanych na wykresie punktwym obok wykresu sylwetki.

3. Analiza koszyka zakupowego

Analiza koszyka zakupowego (analiza asocjacji) polega na znajdowaniu zależności między atrybutami w formie reguł. Algorytmy tej analizy nie zostały zaimplementowane w dotychczas wykorzystywanym pakietach. Znajdują się natomiast w pakiecie [MLxtend (http://rasbt.github.io/mlxtend/)](http://rasbt.github.io/mlxtend/).

In [11]:
```
# pakiet wymaga wcześniejszej instalacji
# w razie problemów z instalacją przez anaconda navigator-a
# należy w Anaconda prompt wpisać:
# pip install mlxtend

from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

Zbiór danych `sklep` zawiera informację o 20 transakcjach zakupowych 7 towarów w sklepie. Wiersze oznaczają koszyki zakupowe, zaś w kolumnach zawarta jest informacja o fakcie obecności (1) lub jej braku (0) danego produktu w koszyku.

In [14]:
```
df = pd.read_excel('sklep.xlsx',header = 1,usecols = range(1,9), index_col = 0)
df
```

Out[14]:

Lp.	Długopis	Ołówek	Zeszyt	Papier	Linijka	Kredki	Blok
1	1	0	1	0	0	0	0
2	0	1	0	1	0	0	0
3	0	0	1	0	0	1	0
4	0	1	1	0	0	1	0
5	0	0	1	0	1	0	0
6	0	0	0	0	0	1	1
7	1	0	0	0	1	1	1
8	1	0	1	0	1	0	0
9	1	0	0	1	0	0	0
10	1	1	0	0	0	1	0
11	0	0	1	0	0	0	0
12	0	0	1	0	0	0	1
13	1	0	1	0	1	1	0
14	0	0	0	0	0	1	1
15	1	0	1	0	0	1	0
16	0	1	0	1	1	0	0
17	1	1	1	0	1	0	0
18	0	0	0	0	1	1	1
19	0	0	0	0	1	1	0
20	1	0	1	0	1	1	0

In [15]:
```
print("Liczba produktów w koszykach: \n", df.sum(axis=1))
print("Częstość atrybutów: \n",df.sum(axis=0))
```

```
Liczba produktów w koszykach: 
  Lp.
1     2
2     2
3     2
4     3
5     2
6     2
7     4
8     3
9     2
10    3
11    1
12    2
13    4
14    2
15    3
16    3
17    4
18    3
19    2
20    4
dtype: int64
Częstość atrybutów: 
 Długopis     9
Ołówek       5
Zeszyt      11
Papier       3
Linijka      9
Kredki      11
Blok         5
dtype: int64
```

Wybieramy zbiory częste (zasada a-priori) o zadanym progu wsparcia (powyżej 0,2).

In [16]:
```
zbiory_czeste = apriori(df, min_support=0.2, use_colnames=True)
zbiory_czeste
```

Out[16]:

	support	itemsets
0	0.45	(Długopis)
1	0.25	(Ołówek)
2	0.55	(Zeszyt)
3	0.45	(Linijka)
4	0.55	(Kredki)
5	0.25	(Blok)
6	0.30	(Zeszyt, Długopis)
7	0.25	(Długopis, Linijka)
8	0.25	(Długopis, Kredki)
9	0.25	(Zeszyt, Linijka)
10	0.25	(Zeszyt, Kredki)
11	0.25	(Linijka, Kredki)
12	0.20	(Blok, Kredki)
13	0.20	(Zeszyt, Długopis, Linijka)

Na podstawie zbiorów częstych generujemy reguły.

In [17]:
```
reguly = association_rules(zbiory_czeste, metric="lift", min_threshold=1)
reguly
```

Out[17]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Zeszyt)	(Długopis)	0.55	0.45	0.30	0.55	1.21	0.05	1.21
1	(Długopis)	(Zeszyt)	0.45	0.55	0.30	0.67	1.21	0.05	1.35
2	(Długopis)	(Linijka)	0.45	0.45	0.25	0.56	1.23	0.05	1.24
3	(Linijka)	(Długopis)	0.45	0.45	0.25	0.56	1.23	0.05	1.24
4	(Długopis)	(Kredki)	0.45	0.55	0.25	0.56	1.01	0.00	1.01
5	(Kredki)	(Długopis)	0.55	0.45	0.25	0.45	1.01	0.00	1.01
6	(Zeszyt)	(Linijka)	0.55	0.45	0.25	0.45	1.01	0.00	1.01
7	(Linijka)	(Zeszyt)	0.45	0.55	0.25	0.56	1.01	0.00	1.01
8	(Linijka)	(Kredki)	0.45	0.55	0.25	0.56	1.01	0.00	1.01
9	(Kredki)	(Linijka)	0.55	0.45	0.25	0.45	1.01	0.00	1.01
10	(Blok)	(Kredki)	0.25	0.55	0.20	0.80	1.45	0.06	2.25
11	(Kredki)	(Blok)	0.55	0.25	0.20	0.36	1.45	0.06	1.18
12	(Zeszyt, Długopis)	(Linijka)	0.30	0.45	0.20	0.67	1.48	0.07	1.65
13	(Zeszyt, Linijka)	(Długopis)	0.25	0.45	0.20	0.80	1.78	0.09	2.75
14	(Długopis, Linijka)	(Zeszyt)	0.25	0.55	0.20	0.80	1.45	0.06	2.25
15	(Zeszyt)	(Długopis, Linijka)	0.55	0.25	0.20	0.36	1.45	0.06	1.18
16	(Długopis)	(Zeszyt, Linijka)	0.45	0.25	0.20	0.44	1.78	0.09	1.35
17	(Linijka)	(Zeszyt, Długopis)	0.45	0.30	0.20	0.44	1.48	0.07	1.26

Wybieramy jedynie te reguły, które spełniają warunek (ufność >= 0,8).

In [18]:
```
reguly[ (reguly['confidence'] >= 0.8) ]
```

Out[18]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
10	(Blok)	(Kredki)	0.25	0.55	0.20	0.80	1.45	0.06	2.25
13	(Zeszyt, Linijka)	(Długopis)	0.25	0.45	0.20	0.80	1.78	0.09	2.75
14	(Długopis, Linijka)	(Zeszyt)	0.25	0.55	0.20	0.80	1.45	0.06	2.25

Zadanie Poeksperymentuj z innymi wartościami parametrów.

> **Zadanie** Wygeneruj losowo inny, większy, zestaw transakcji (kilkaset transakcji) w tym samym sklepie. Zastosuj metodę, dobierz parametry.

Dla dociekliwych

- Porównanie różnych rodzajów odległości międzygrupowej w grupowaniu (https://scikit-learn.org/stable/auto_examples/cluster/plot_linkage_comparison.html)
- Grupowanie k-średnich, poradnik (https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html)
- Grupowanie k-średnich, przypadki szczególne (https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py)
- Market basket analysis (https://pbpython.com/market-basket-analysis.html)