ZAD 1

Proszę pobrać plik customers_mall.csv, zawierający informacje o klientach pewnego centrum handlowego. Pierwsza kolumna przedstawia ich zarobki (w tysiącach), w drugiej zaś znajduje się punktowa ocena wydatków (od 0 do 100) każdego z klientów. Proszę dokonać klasteryzacji zbioru z użyciem algorytmu k-means. Uzyskane wyniki należy zwizualizować, ocenić i opisać. Wskazane jest zarekomendowanie odpowiedniej ilości klastrów.

In [58]:
```
from google.colab import drive
import pandas as pd
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Lab06/cus
print(f'Rows = {df.shape[0]}')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remo
unt, call drive.mount("/content/drive", force_remount=True).
Rows = 200
```

In [59]:
```
df
```

Out[59]:

	Annual Income	Spending Score
0	15	39
1	15	81
2	16	6
3	16	77
4	17	4
...	...	...
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

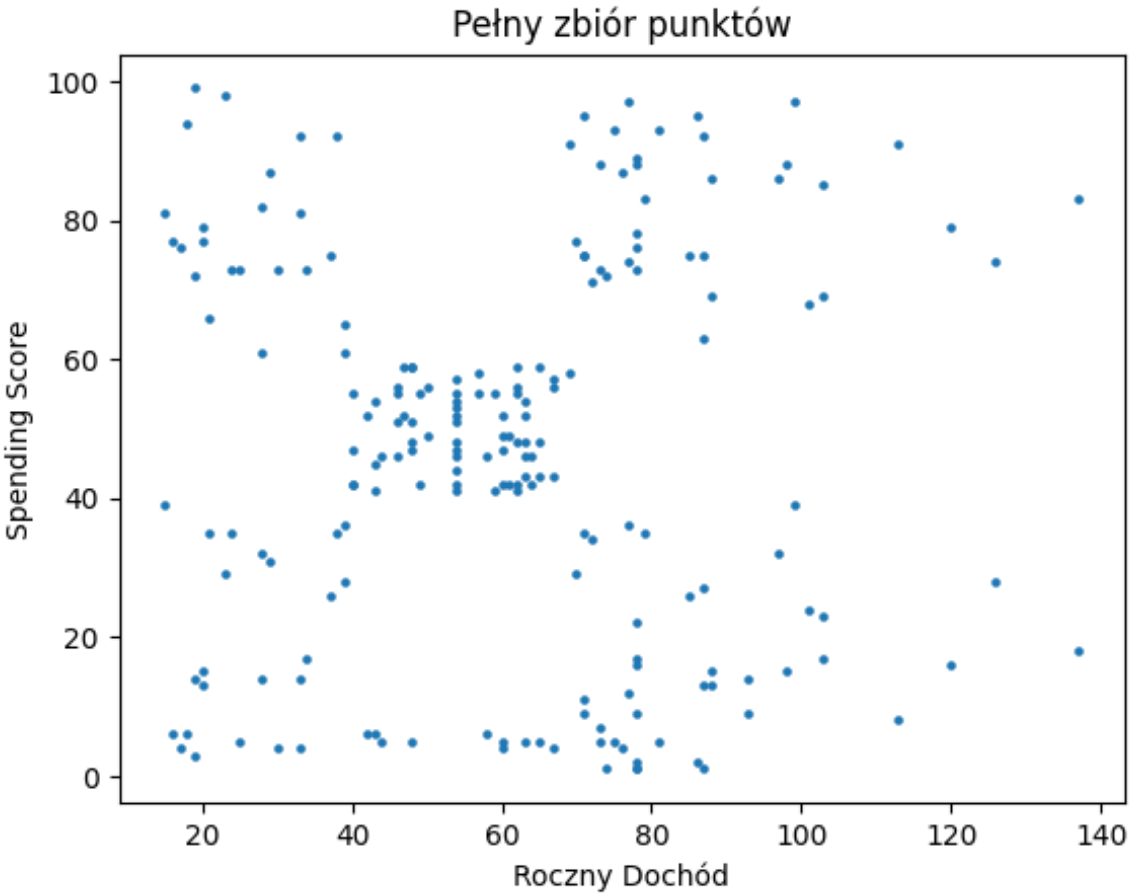
200 rows × 2 columns

In [60]:
```
df.describe()
```

Out[60]:

	Annual Income	Spending Score
count	200.000000	200.000000
mean	60.560000	46.465000
std	26.264721	28.220396
min	15.000000	1.000000
25%	41.500000	21.000000
50%	61.500000	47.500000
75%	78.000000	72.000000
max	137.000000	99.000000

In [61]:
```
import matplotlib.pyplot as plt
plt.scatter(df.iloc[:,0],df.iloc[:,1],s=5)
plt.xlabel('Roczny Dochód')
plt.ylabel('Spending Score')
plt.title(f'Pełny zbiór punktów')
plt.show()
```



In [62]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df),columns = ['Annua
df_scaled.describe()
```

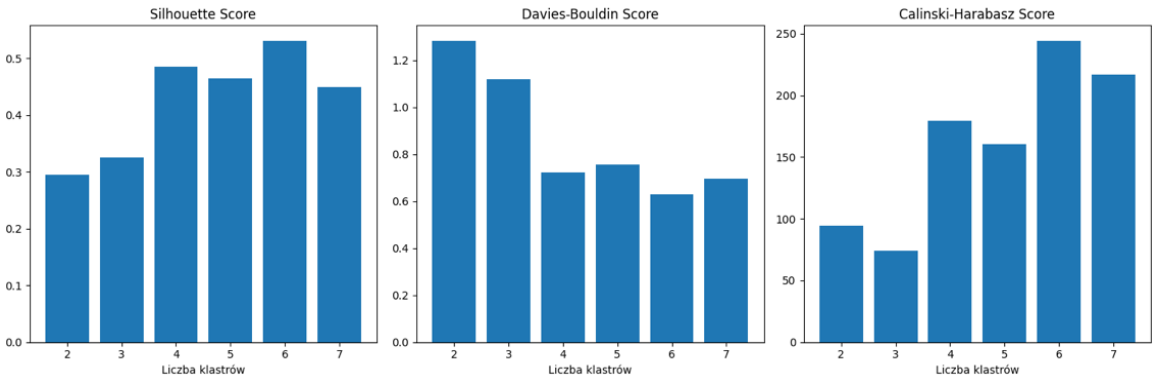
Out[62]:

	Annual Income	Spending Score
count	2.000000e+02	2.000000e+02
mean	-2.131628e-16	-1.487699e-16
std	1.002509e+00	1.002509e+00
min	-1.738999e+00	-1.615112e+00
25%	-7.275093e-01	-9.046260e-01
50%	3.587926e-02	3.676764e-02
75%	6.656748e-01	9.071127e-01
max	2.917671e+00	1.866268e+00

Rekomendacja odpowiedniej ilości klastrów

In [63]:

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

d = {'Silhouette':[], 'DB':[], 'CH':[]}

for i in range(2,8):
  model = KMeans(n_clusters=i)
  model.fit(df_scaled)
  labels = model.labels_
  d['Silhouette'].append(metrics.silhouette_score(df_scaled, labels
  d['DB'].append(metrics.davies_bouldin_score(df_scaled, labels))
  d['CH'].append(metrics.calinski_harabasz_score(df_scaled, labels)

fig, axs = plt.subplots(1, 3, figsize=(15, 5))
x_values = range(2,8)
axs[0].bar(x_values, d['Silhouette'])
axs[0].set_title('Silhouette Score')

axs[1].bar(x_values, d['DB'])
axs[1].set_title('Davies-Bouldin Score')

axs[2].bar(x_values, d['CH'])
axs[2].set_title('Calinski-Harabasz Score')

for i in range(3):
  axs[i].set_xlabel('Liczba klastrów')

plt.tight_layout()
plt.show()
```



Wnioski

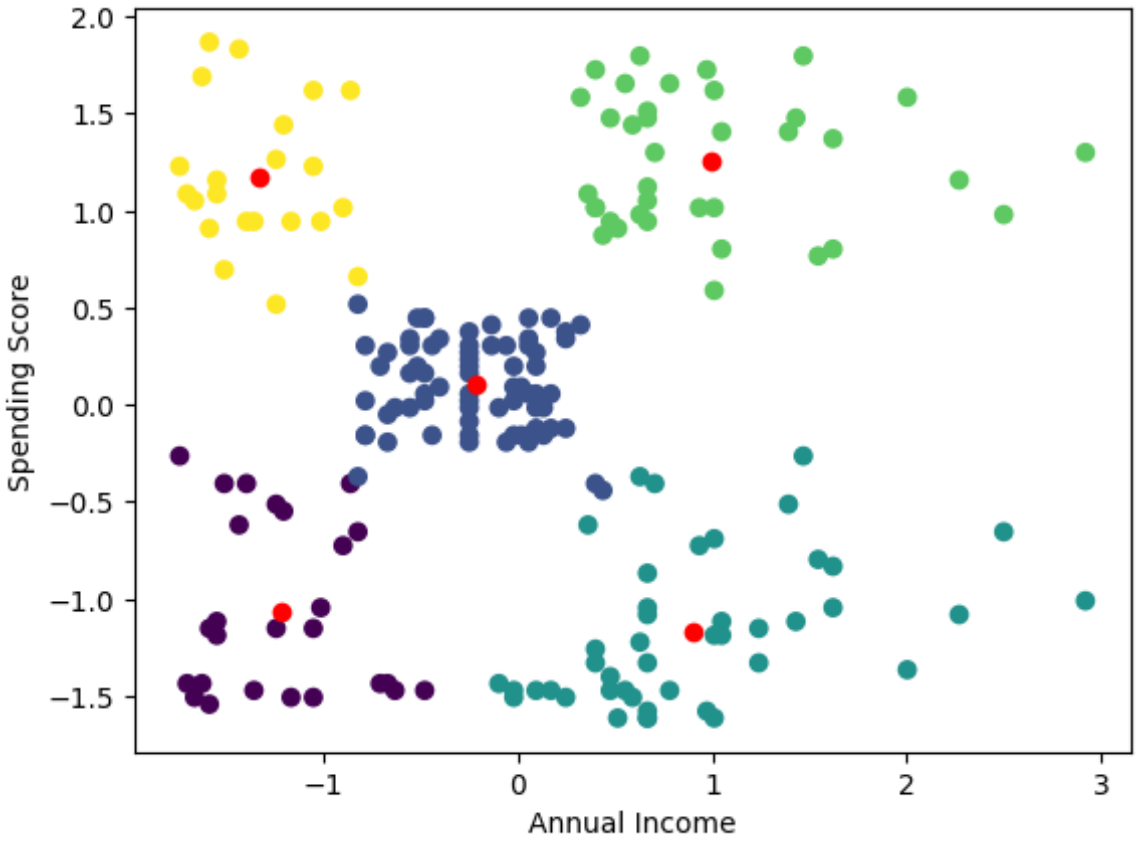
Analiza wyników metryk dla różnych liczb klastrów wskazuje, że optymalną liczbą klastrów może być 5 lub 6. Ostatecznie, wybieram liczbę klastrów 5, ponieważ subiektywnie wydaje się to lepiej oddawać strukturę danych na wykresie powyżej.

Dla takiej liczby klastrów zaprezentuję ostateczny podział danych:

In [64]:
```
model = KMeans(n_clusters=5)
model.fit(df_scaled)

print('Centra klastrów:')
print(scaler.inverse_transform(model.cluster_centers_))
```

```
Centra klastrów:
[[28.57692308 16.57692308]
 [54.81690141 49.25352113]
 [84.25       13.54545455]
 [86.59459459 81.7027027 ]
 [25.72727273 79.36363636]]
```

In [65]:
```
plt.scatter(df_scaled.iloc[:,0],df_scaled.iloc[:,1],c=model.labels_
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```



ZAD 2

W pliku planets.csv mają państwo zbiór 778 spośród ponad 5000 znanych egzoplanet (planet pozasłonecznych) pozyskany z bazy danych NASA (https://exoplanetarchive.ipac.caltech.edu/index.html). Proszę dokonać klasteryzacji tego zbioru kilkoma sposobami, ocenić wyniki za pomocą odpowiednich metryk, wybrać dowolny z wyników i jego rezultaty (czym się charakteryzują klastry).

Poniżej znajduje się opis kolumn do zestawu danych:

```
      pl_name:        Planet Name,
      pl_orbper:      Orbital Period [days]',
      pl_orbsmax:     Orbit Semi-Major Axis [au]),
      pl_rade:        Planet Radius [Earth Radius],
      pl_masse:       Planet Mass [Earth Mass],
      pl_orbeccen:    Eccentricity,
      pl_eqt:         Equilibrium Temperature [K],
      st_teff:        Stellar Effective Temperature
[K],
      st_mass:        Stellar Mass [Solar mass],
      sy_dist:        Distance [pc]
```

In [66]:
```
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Lab06/pla
```

In [67]:
```
df
```

Out[67]:

	pl_name	pl_orbper	pl_orbsmax	pl_rade	pl_masse	pl_orbeccen	p
0	55 Cnc e	0.736544	0.01544	2.080	7.81000	0.061	
1	AU Mic b	8.463000	0.06450	4.070	17.00000	0.000	
2	AU Mic c	18.859019	0.11010	3.240	13.60000	0.000	
3	BD+20 594 b	41.685500	0.24100	2.230	16.30000	0.000	
4	CoRoT-10 b	13.240600	0.10550	10.870	874.00000	0.530	
...	...	...	...	...	...	...	
773	XO-5 b	4.187756	0.05150	12.780	378.20000	0.000	
774	XO-6 b	3.765001	0.08150	23.203	1398.45200	0.000	
775	XO-7 b	2.864142	0.04421	15.390	225.34147	0.038	
776	pi Men c	6.267900	0.06839	2.042	4.82000	0.000	
777	pi Men c	6.268340	0.06702	2.060	4.52000	0.000	

778 rows × 10 columns

In [68]:
```
df.describe()
```

Out[68]:

	pl_orbper	pl_orbsmax	pl_rade	pl_masse	pl_orbeccen	
count	778.000000	778.000000	778.000000	778.000000	778.000000	7
mean	19.318080	0.089892	9.991374	367.210044	0.083592	1
std	147.168764	0.217884	5.872213	685.849443	0.130941	5
min	0.179715	0.005800	0.510000	0.070000	0.000000	1
25%	2.815264	0.038155	3.150000	15.050000	0.000000	1
50%	4.014884	0.050600	11.555500	177.982400	0.030000	12
75%	8.878242	0.080000	14.238750	390.930900	0.112750	16
max	3650.000000	4.500000	23.430000	6388.100000	0.920000	40

In [69]:
```
df.columns[1:]
```

Out[69]:
```
Index(['pl_orbper', 'pl_orbsmax', 'pl_rade', 'pl_masse', 'pl_orbec
cen',
       'pl_eqt', 'st_teff', 'st_mass', 'sy_dist'],
      dtype='object')
```

In [70]:
```
import numpy as np
for col in df.columns[1:]:
  mean = np.mean(df[col])
```

```
  std = np.std(df[col])
  lower_bound = mean - 3*std
  upper_bound = mean + 3*std
  df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

print(f'Liczba rzędów bo usunięciu outlierów: {df.shape[0]}')
```

```
Liczba rzędów bo usunięciu outlierów: 710
```

In [71]:
```
df.shape
```

Out[71]: (710, 10)

In [72]:
```
df.iloc[:,1:]
```

Out[72]:

	pl_orbper	pl_orbsmax	pl_rade	pl_masse	pl_orbeccen	pl_eqt	st_t
0	0.736544	0.01544	2.080	7.81000	0.061	1958	523
1	8.463000	0.06450	4.070	17.00000	0.000	593	370
2	18.859019	0.11010	3.240	13.60000	0.000	454	370
3	41.685500	0.24100	2.230	16.30000	0.000	546	576
5	2.994330	0.04360	16.030	740.51000	0.000	1657	644
...	...	...	...	...	...	...	
773	4.187756	0.05150	12.780	378.20000	0.000	1230	543
774	3.765001	0.08150	23.203	1398.45200	0.000	1577	672
775	2.864142	0.04421	15.390	225.34147	0.038	1743	625
776	6.267900	0.06839	2.042	4.82000	0.000	1170	603
777	6.268340	0.06702	2.060	4.52000	0.000	1147	587

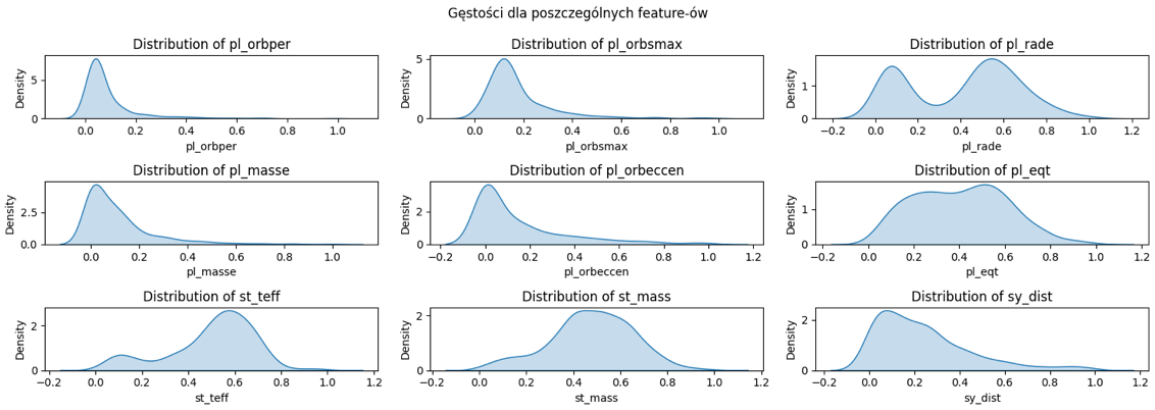
710 rows × 9 columns

In [73]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df.iloc[:,1:]),column
df_scaled.describe()
```

Out[73]:

	pl_orbper	pl_orbsmax	pl_rade	pl_masse	pl_orbeccen	
count	710.000000	710.000000	710.000000	710.000000	710.000000	710.0(
mean	0.092839	0.179351	0.405627	0.116160	0.153643	0.4(
std	0.122986	0.148942	0.258595	0.156437	0.220905	0.2
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	0.033920	0.097624	0.108639	0.005889	0.000000	0.2
50%	0.048525	0.134120	0.478534	0.069322	0.052874	0.4(
75%	0.099833	0.202943	0.599040	0.147785	0.217241	0.5
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.0(

In [74]:
```
from seaborn import kdeplot

fig, axs = plt.subplots(3, 3, figsize=(15, 5))
i = 0
for col in df_scaled.columns:
    row = i // 3
    col_index = i % 3
    kdeplot(data=df_scaled, x=col, fill=True, ax=axs[row, col_index
    i += 1
    axs[row, col_index].set_title(f'Distribution of {col}')

fig.tight_layout()
fig.suptitle('Gęstości dla poszczególnych feature-ów', y=1.05)
plt.show()
```



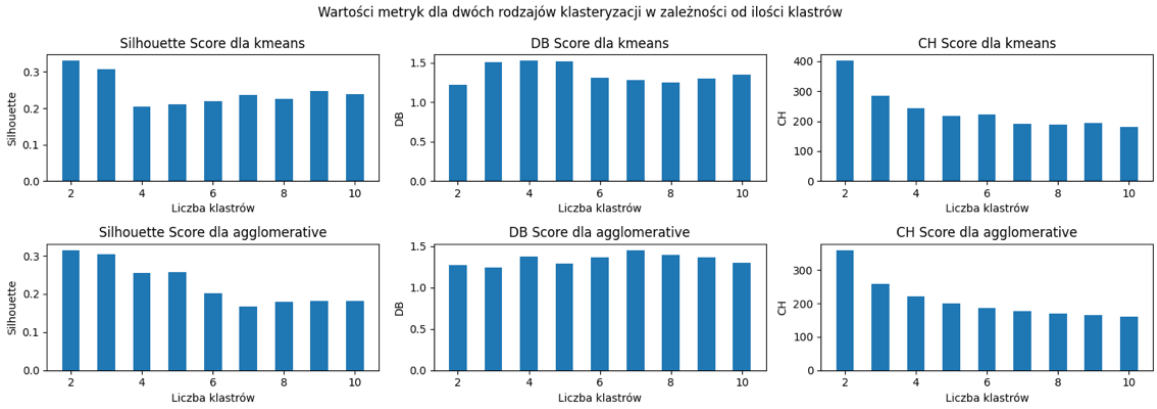
In [75]:
```
#  Lepiej zrobic normalizacje min-max
#  --> jak mamy histogramy: dwa gaussy nalozone na sb albo uciety g
```

In [76]:
```
d = {name : {'Silhouette':[], 'DB':[], 'CH':[] } for name in ['kmea

for nr_clusters in range(2,11):
  model_kmeans = KMeans(n_clusters = nr_clusters)
  model_kmeans.fit(df_scaled)
  labels = model_kmeans.labels_
  d['kmeans']['Silhouette'].append(metrics.silhouette_score(df_scal
  d['kmeans']['DB'].append(metrics.davies_bouldin_score(df_scaled,
```

```
  d['kmeans']['CH'].append(metrics.calinski_harabasz_score(df_scale
```

In [77]:
```
from sklearn.cluster import AgglomerativeClustering

for nr_clusters in range(2,11):
  model_agglomerative = AgglomerativeClustering(n_clusters=nr_clust
  model_agglomerative.fit(df_scaled)
  labels = model_agglomerative.labels_
  d['agglomerative']['Silhouette'].append(metrics.silhouette_score(
  d['agglomerative']['DB'].append(metrics.davies_bouldin_score(df_s
  d['agglomerative']['CH'].append(metrics.calinski_harabasz_score(d
```

In [78]:
```
fig, axs = plt.subplots(2, 3, figsize=(15, 5))
for idx, metric in enumerate(['Silhouette', 'DB', 'CH']):
    axs[0, idx].bar(range(2,11), d['kmeans'][metric], width = 0.5)
    axs[1, idx].bar(range(2,11), d['agglomerative'][metric], width

    axs[0, idx].set_title(f'{metric} Score dla kmeans')
    axs[1, idx].set_title(f'{metric} Score dla agglomerative')
    for row in range(2):
        axs[row, idx].set_xlabel('Liczba klastrów')
        axs[row, idx].set_ylabel(metric)

fig.tight_layout()
fig.suptitle('Wartości metryk dla dwóch rodzajów klasteryzacji w za
plt.show()
```

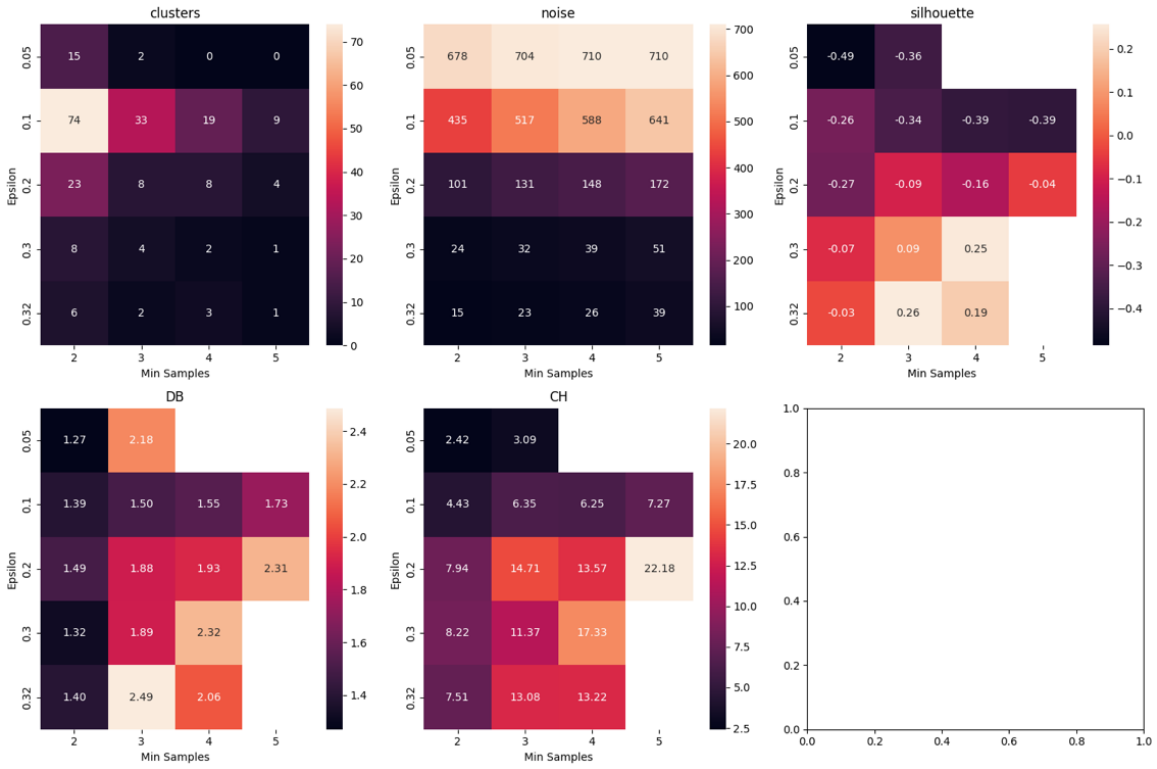


Wnioski

Na wykresie zauważamy, że dla obu metod klasteryzacji: k-średnich oraz aglomeracyjnej, metryki osiągają optymalne wartości dla dwóch klastrów.

In [79]:
```
from sklearn.cluster import DBSCAN
from seaborn import heatmap

epsilons = [0.05, 0.1, 0.2, 0.3, 0.32]
min_samples = [2, 3, 4, 5]
results = {}

for epsilon in epsilons:
    results[epsilon] = {}
```

```
    for min_sample in min_samples:
        model_dbscan = DBSCAN(eps=epsilon, min_samples=min_sample)
        model_dbscan.fit(df_scaled)
        labels = model_dbscan.labels_
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
        n_noise_ = list(labels).count(-1)

        results[epsilon][min_sample] = {'clusters': n_clusters_, 'n

        if n_clusters_ > 1:
            results[epsilon][min_sample]['silhouette'] = metrics.si
            results[epsilon][min_sample]['DB'] = metrics.davies_bou
            results[epsilon][min_sample]['CH'] = metrics.calinski_h
        else:
            results[epsilon][min_sample]['silhouette'] = None
            results[epsilon][min_sample]['DB'] = None
            results[epsilon][min_sample]['CH'] = None

fig, axs = plt.subplots(2, 3, figsize=(15, 10))  # 2 rows, 3 column
metrics_to_plot = ['clusters', 'noise', 'silhouette', 'DB', 'CH']
axs_flat = axs.flatten()

for i, metric in enumerate(metrics_to_plot):
    data = np.zeros((len(epsilons), len(min_samples)))
    for x, epsilon in enumerate(epsilons):
        for y, min_sample in enumerate(min_samples):
            data[x, y] = results[epsilon][min_sample].get(metric, n
    heatmap(data, annot=True, fmt=".2f" if metric in ['silhouette',
    axs_flat[i].set_xlabel('Min Samples')
    axs_flat[i].set_ylabel('Epsilon')
    axs_flat[i].set_title(f'{metric}')

plt.tight_layout()
plt.show()
```



Wnioski

Wyniki opisujące klasteryzację przy pomocy DBSCAN na podstawie wartości epsilon oraz minimalnej liczby elementów potrzebnych do powstania zgrupowania zaprezentowane na heatmapach wskazują, że jest to metoda dużo gorsza dla tych danych. Spośród przedstawionych parametrów, daje ona najlepszy wynik dla : epsilon = 0.32, min_samples = 3. Dla takich parametrów, dzieli dane również na 2 klastry, podobnie jak poprzednie metody.

Spośród przetestowanych metod, optymalna wydaje się klasteryzacja za pomocą kmeans, dla dwóch klastrów. Dla niej uzyskałam najlepsze wartości analizowanych metryk. Wartości metryk dla takiego modelu przentują się następująco:

In [80]:
```
model_kmeans = KMeans(n_clusters = 2)
model_kmeans.fit(df_scaled)
labels = model_kmeans.labels_
print(f'Silhouette: {metrics.silhouette_score(df_scaled, labels)}')
print(f'Davies-Bouldinc Score: {metrics.davies_bouldin_score(df_sca
print(f'Carabasz-Halinski Score: {metrics.calinski_harabasz_score(d
```

```
Silhouette: 0.3308017100105429
Davies-Bouldinc Score: 1.216605318199412
Carabasz-Halinski Score: 402.27765514050327
```

ZAD 3

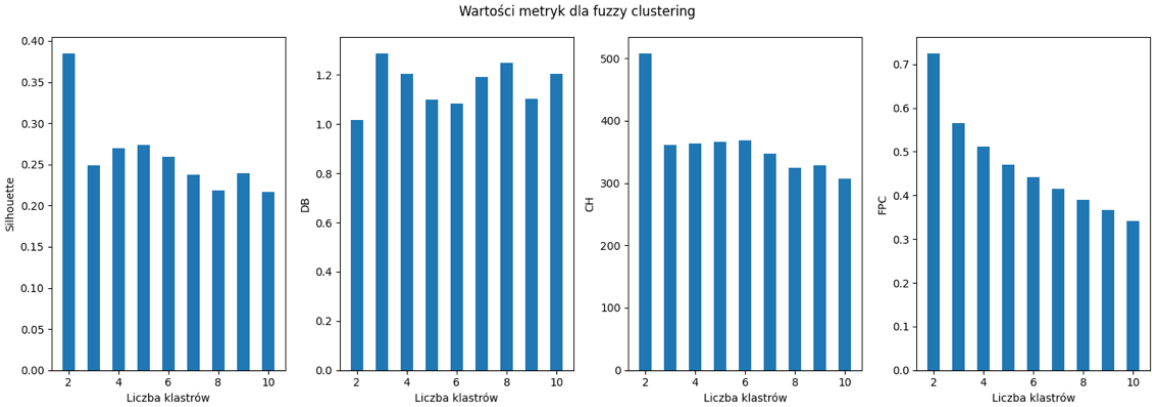
Z użyciem algorytmu fuzzy clustering proszę zaproponować model dla podanego powyżej zestawu danych. Do uczenia należy wybrać wyłącznie połowę jego kolumn.

In [81]:
```
!pip install scikit-fuzzy
```

```
Requirement already satisfied: scikit-fuzzy in /usr/local/lib/python
3.11/dist-packages (0.5.0)
```

In [82]:
```
import skfuzzy as fuzz
half_columns = df_scaled.columns[:][1::2]
X = df_scaled[half_columns]

fuzzy_model_metrics = {'Silhouette':[], 'DB':[], 'CH':[] , 'FPC': [

for nr_clusters in range(2,11):
  cntr, u_orig, u0, d, jm, p, fpc  = fuzz.cluster.cmeans(X.T, nr_cl
  labels = np.argmax(u_orig, axis=0)
  fuzzy_model_metrics['Silhouette'].append(metrics.silhouette_score
  fuzzy_model_metrics['DB'].append(metrics.davies_bouldin_score(X,
  fuzzy_model_metrics['CH'].append(metrics.calinski_harabasz_score(
  fuzzy_model_metrics['FPC'].append(fpc)
```

In [83]:
```
fig, axs = plt.subplots(1, 4, figsize=(15, 5))
for idx, metric in enumerate(['Silhouette', 'DB', 'CH', 'FPC']):
    axs[idx].bar(range(2,11), fuzzy_model_metrics[metric], width =
    axs[idx].set_xlabel('Liczba klastrów')
    axs[idx].set_ylabel(metric)

fig.tight_layout()
fig.suptitle('Wartości metryk dla fuzzy clustering', y=1.05)
plt.show()
```



Wnioski

Najlepsze wyniki dla modelu klastrowania 'miękkiego' zdecydowanie zwraca nam podział na dwa klastry. Wnioski bazujące na wartościach metryk potwierdza również współczynnik Fuzzy Partition Coefficient, który po zmaksymalizowaniu, określa najlepsze opisanie danych.

ZAD 4

Dla zbioru danych circle.csv proszę, wykorzystując wszystkie (za wyjątkiem c-means) poznane do tej pory algorytmy klasteryzacyjne, podjąć kilka prób dopasowania jak najlepszego modelu, za każdym razem oceniając rozwiązanie z użyciem dedykowanych do tego metryk. Który z algorytmów najlepiej radzi sobie z takim układem danych i dlaczego? Proszę przedstawić wyniki również w formie odpowiednich wizualizacji.

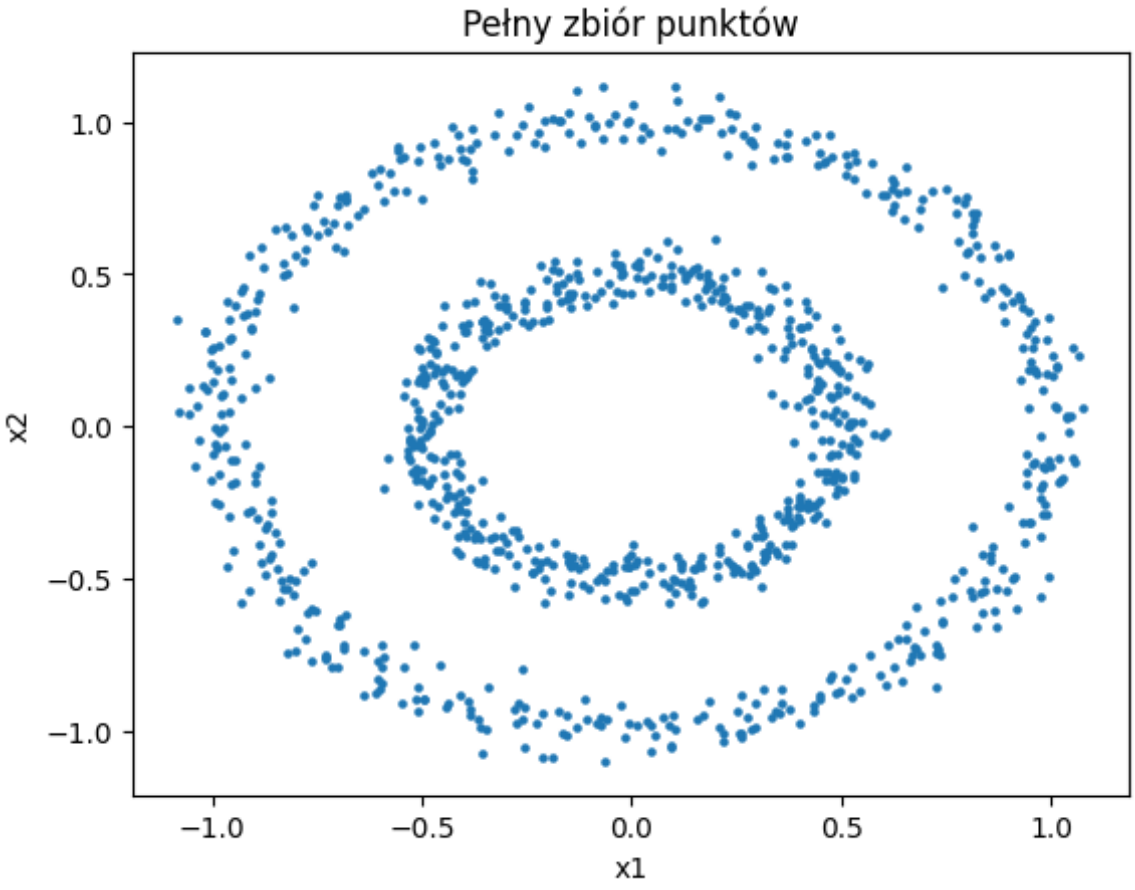
In [84]:
```
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Lab06/cir
df.describe()
```

Out[84]:

	x1	x2
count	1000.000000	1000.000000
mean	0.004336	0.000483
std	0.559414	0.561679
min	-1.081727	-1.099632
25%	-0.439131	-0.448021
50%	0.005845	0.001042
75%	0.440670	0.426588
max	1.078307	1.116832

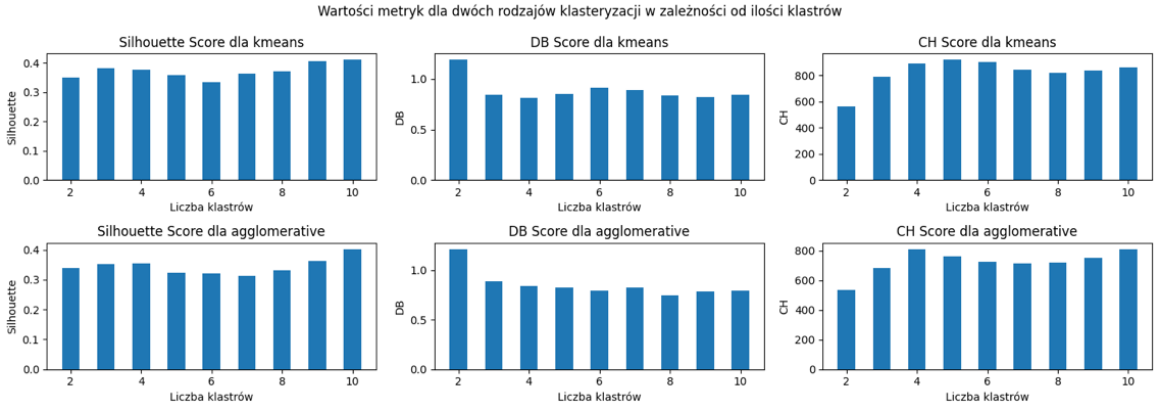
Nie ma potrzeby normalizować danych.

In [85]:
```
plt.scatter(df['x1'],df['x2'],s=5)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title(f'Pełny zbiór punktów')
plt.show()
```



In [86]:
```
d = {name : {'Silhouette':[], 'DB':[], 'CH':[] } for name in ['kmea

for nr_clusters in range(2,11):
  model_kmeans = KMeans(n_clusters = nr_clusters)
```

```
  model_kmeans.fit(df)
  labels = model_kmeans.labels_
  d['kmeans']['Silhouette'].append(metrics.silhouette_score(df, lab
  d['kmeans']['DB'].append(metrics.davies_bouldin_score(df, labels)
  d['kmeans']['CH'].append(metrics.calinski_harabasz_score(df, labe

  model_agglomerative = AgglomerativeClustering(n_clusters=nr_clust
  model_agglomerative.fit(df)
  labels = model_agglomerative.labels_
  d['agglomerative']['Silhouette'].append(metrics.silhouette_score(
  d['agglomerative']['DB'].append(metrics.davies_bouldin_score(df,
  d['agglomerative']['CH'].append(metrics.calinski_harabasz_score(d

fig, axs = plt.subplots(2, 3, figsize=(15, 5))
for idx, metric in enumerate(['Silhouette', 'DB', 'CH']):
    axs[0, idx].bar(range(2,11), d['kmeans'][metric], width = 0.5)
    axs[1, idx].bar(range(2,11), d['agglomerative'][metric], width

    axs[0, idx].set_title(f'{metric} Score dla kmeans')
    axs[1, idx].set_title(f'{metric} Score dla agglomerative')
    for row in range(2):
        axs[row, idx].set_xlabel('Liczba klastrów')
        axs[row, idx].set_ylabel(metric)

fig.tight_layout()
fig.suptitle('Wartości metryk dla dwóch rodzajów klasteryzacji w za
plt.show()
```

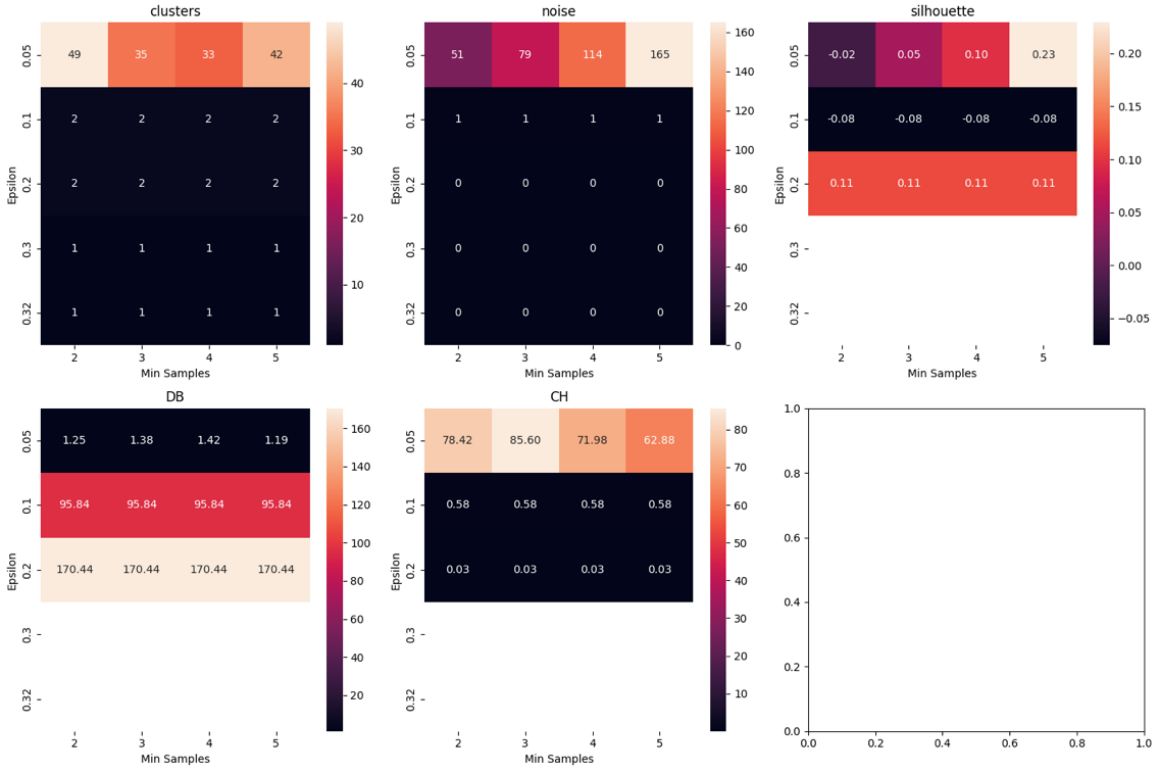


Wnioski

Wartości metryk odznaczają się dla ilości klastrów: 4 albo 10 na tle pozostałych. Są najlepsze.

In [87]:
```
epsilons = [0.05, 0.1, 0.2, 0.3, 0.32]
min_samples = [2, 3, 4, 5]
results = {}

for epsilon in epsilons:
    results[epsilon] = {}
    for min_sample in min_samples:
        model_dbscan = DBSCAN(eps=epsilon, min_samples=min_sample)
        model_dbscan.fit(df)
```

```
        labels = model_dbscan.labels_
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
        n_noise_ = list(labels).count(-1)

        results[epsilon][min_sample] = {'clusters': n_clusters_, 'n

        if n_clusters_ > 1:
            results[epsilon][min_sample]['silhouette'] = metrics.si
            results[epsilon][min_sample]['DB'] = metrics.davies_bou
            results[epsilon][min_sample]['CH'] = metrics.calinski_h
        else:
            results[epsilon][min_sample]['silhouette'] = None
            results[epsilon][min_sample]['DB'] = None
            results[epsilon][min_sample]['CH'] = None

fig, axs = plt.subplots(2, 3, figsize=(15, 10))
metrics_to_plot = ['clusters', 'noise', 'silhouette', 'DB', 'CH']
axs_flat = axs.flatten()

for i, metric in enumerate(metrics_to_plot):
    data = np.zeros((len(epsilons), len(min_samples)))
    for x, epsilon in enumerate(epsilons):
        for y, min_sample in enumerate(min_samples):
            data[x, y] = results[epsilon][min_sample].get(metric, n
    heatmap(data, annot=True, fmt=".2f" if metric in ['silhouette',
    axs_flat[i].set_xlabel('Min Samples')
    axs_flat[i].set_ylabel('Epsilon')
    axs_flat[i].set_title(f'{metric}')

plt.tight_layout()
plt.show()
```

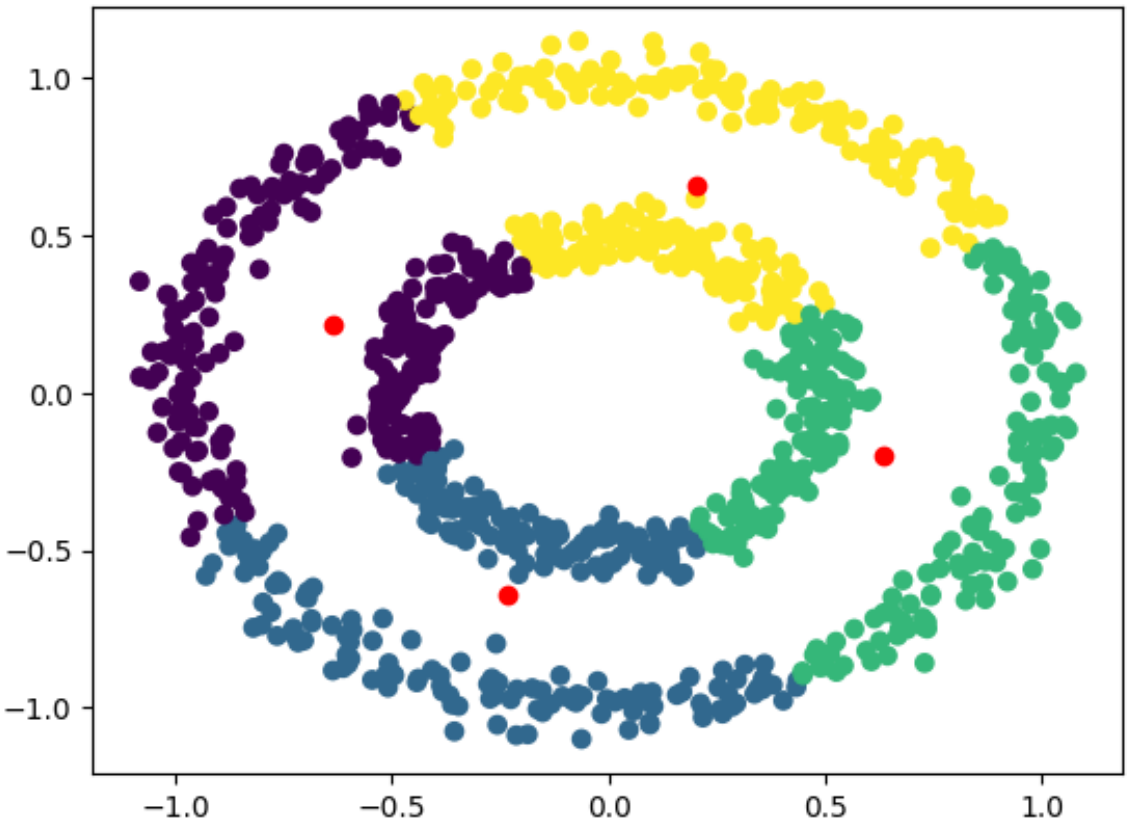


Wnioski

Algorytm DBSCAN gorzej radzi sobie z tym zbiorem danych. Dla wielu kombinacji współczynników wprowadzanych do modelu, realizuje tylko jeden klaster. Realizuje również podział na dwa klastry w wielu przypadkach, jednak on zazwyczaj zwraca niezadowalające wartości metryk.

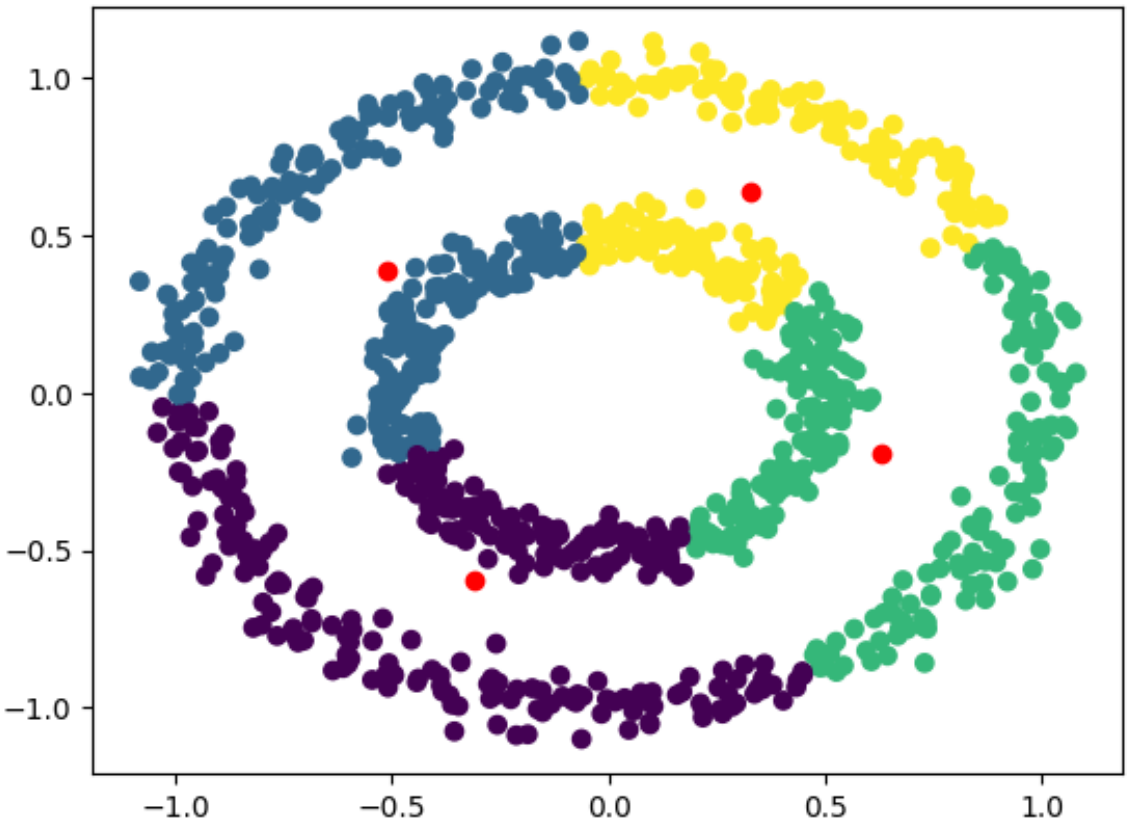
Na podstawie wyników dla różnych rodzajów algorytmów klasteryzacji oraz różnych współczynników rządzących tymi algorytmami, przedstawię podział na 4 klastry za pomocą k-means oraz klastrowania hierarchicznego. Dodatkowo, przedstawię klasteryzację za pomocą dbscan z parametrami eps = 0.05 oraz min_samples = 5, która moim zdaniem najlepiej sobie poradziła, ze względu na wartości metryk, zwracając 42 klastry.

In [88]:
```
model_kmeans = KMeans(n_clusters = 4)
model_kmeans.fit(df)
plt.scatter(df['x1'],df['x2'],c=model_kmeans.labels_)
plt.scatter(model_kmeans.cluster_centers_[:,0], model_kmeans.cluste
plt.show()
```

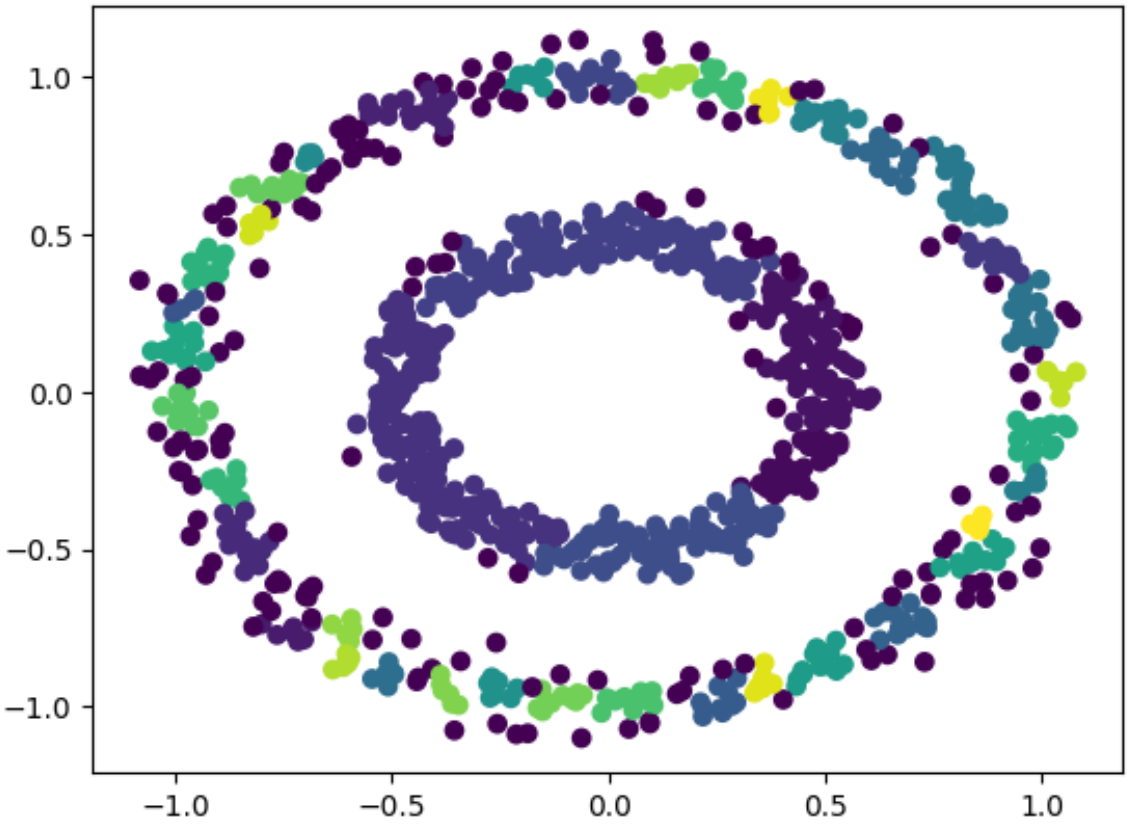


In [89]:
```
model_agglomerative = AgglomerativeClustering(n_clusters=4)
model_agglomerative.fit(df)

cluster_centers = []
for cluster_label in range(4):
    cluster_points = df[model_agglomerative.labels_ == cluster_labe
    cluster_center = cluster_points[['x1', 'x2']].mean(axis=0)
    cluster_centers.append(cluster_center)
cluster_centers = np.array(cluster_centers)

plt.scatter(df['x1'], df['x2'], c=model_agglomerative.labels_)
```

```
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red')
plt.show()
```

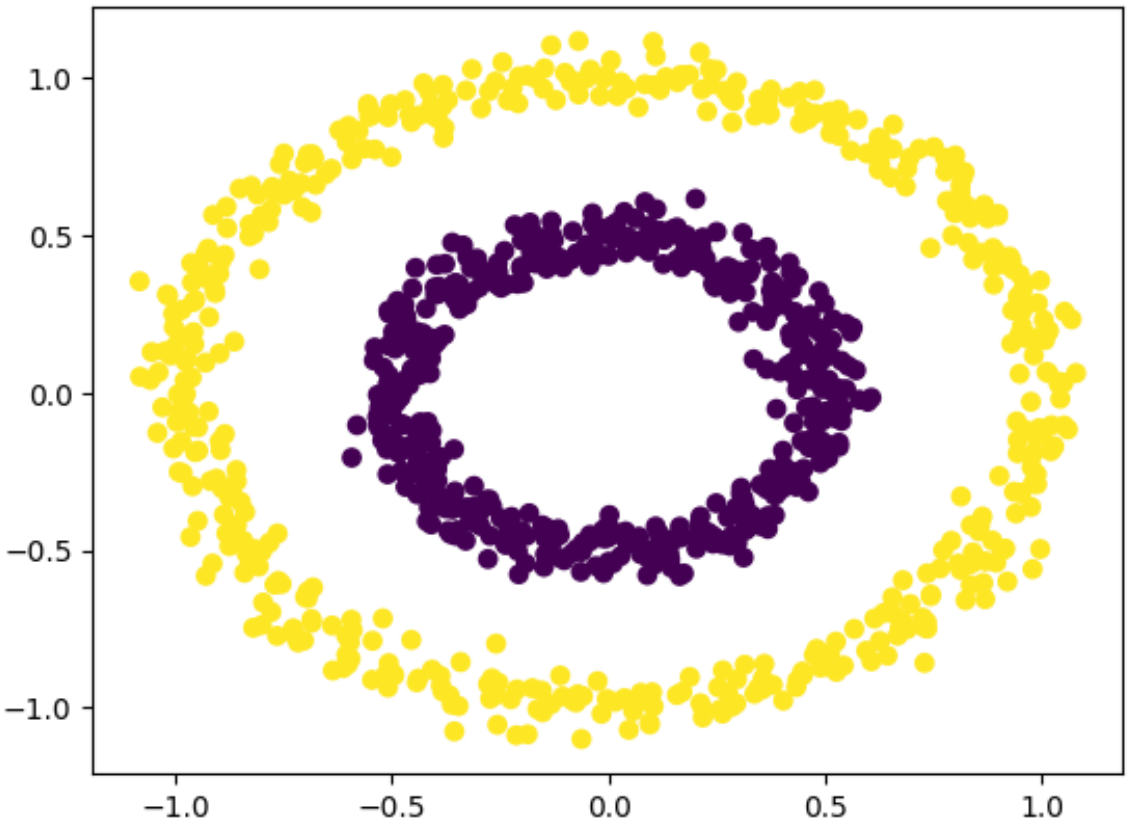


In [90]:
```
model_dbscan = DBSCAN(eps=0.05, min_samples=5)
model_dbscan.fit(df)
plt.scatter(df['x1'],df['x2'],c=model_dbscan.labels_)
plt.show()
```



Mimo najlepszych wartości metryk, ta klasteryzacja nie ma sensu wizualnego, dlatego do wybiorę model, który dzieli zbiór na dwa klastry.

In [91]:
```
model_dbscan = DBSCAN(eps=0.2, min_samples=4)
model_dbscan.fit(df)
plt.scatter(df['x1'],df['x2'],c=model_dbscan.labels_)
plt.show()
```



In [92]:
```
print(f'Podobieństwo kmeans oraz agglomerative: {metrics.rand_score
```

```
Podobieństwo kmeans oraz agglomerative: 0.9175635635635636
```

Wnioski

Klasteryzacje przeprowadzone za pomocą algorytmów k-means oraz klasteryzacji aglomeracyjnej dały bardzo zbliżone rezultaty. Potwierdza to wysoka wartość Indeksu Randa obliczona dla etykiet przypisanych przez te modele.

Jednocześnie, oba te algorytmy podzieliły zbiór danych w zupełnie inny sposób niż algorytm DBSCAN. Pomimo że DBSCAN uzyskał niższe wartości metryk w porównaniu do dwóch pozostałych metod, to sposób podziału, który zaproponował, może być satysfakcjonujący w pewnych sytuacjach. Warto zauważyć, że k-means i klasteryzacja aglomeracyjna nie są w stanie dokonać takiego podziału, który polega na identyfikacji wewnętrznego i zewnętrznego okręgu.

Zatem odpowiedzią na pytanie: który z algorytmów najlepiej radzi sobie z takim układem danych, będzie: to zależy od tego, co chcemy osiągnąć. Np. z algorytmu k-means algo agglomerative skorzystamy, jeżeli będziemy chcieli dokonać segmentacji klientów na podstawie ich preferencji zakupowych, natomiast algorytmu DBSCAN użyjemy, gdy będziemy chcieli zidentyfikować pacjentów z grupą ryzyka, którzy wykazują nieprawidłowe wyniki badań.