Klasteryzacja

May 11, 2023

1 Autorzy

- Mateusz Łopaciński
- Mateusz Mazur

2 Zadanie 1

2.1 Wczytanie danych

W pierwszej kolejności wczytamy dane ze zbiorów danych, na których będziemy pracować

[ ]:
```
import glob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

column_names = ["x", "y", "cluster"]
file_paths = glob.glob("dataset/*.txt")

dfs = {}

for file_path in file_paths:
    file_name = file_path.split("/")[-1].split(".")[0]
    dfs[file_name] = pd.read_csv(file_path, names=column_names)
```

Zobaczmy, czy dane zostały prawidłowo wczytane (czy zgadzają się typy danych)

Zbiór blobs

[ ]:
```
dfs['blobs'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   x        500 non-null    float64
 1   y        500 non-null    float64
 2   cluster  500 non-null    int64
```

```
dtypes: float64(2), int64(1)
memory usage: 11.8 KB
```

Zbiór circles

[ ]:
```
dfs['circles'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   x        500 non-null    float64
 1   y        500 non-null    float64
 2   cluster  500 non-null    int64
dtypes: float64(2), int64(1)
memory usage: 11.8 KB
```

Zbiór ellipses

[ ]:
```
dfs['ellipses'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   x        500 non-null    float64
 1   y        500 non-null    float64
 2   cluster  500 non-null    int64
dtypes: float64(2), int64(1)
memory usage: 11.8 KB
```

Zbiór moons

[ ]:
```
dfs['moons'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   x        500 non-null    float64
 1   y        500 non-null    float64
 2   cluster  500 non-null    int64
dtypes: float64(2), int64(1)
memory usage: 11.8 KB
```

2.2 Normalizacja danych liczbowych

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
Xs = {}

for key, df in dfs.items():
    X = df.drop('cluster', axis=1)
    Xs[key] = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

Zbiór blobs

[ ]:
```
dfs['blobs'].head()
```

[ ]:
```
          x         y  cluster
0  1.103182  4.705777        0
1 -1.932846  3.642251        2
2 -2.034422  1.866002        2
3  1.616402  2.686831        0
4 -0.960010  4.492566        0
```

Zbiór circles

[ ]:
```
dfs['circles'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   x        500 non-null    float64
 1   y        500 non-null    float64
 2   cluster  500 non-null    int64
dtypes: float64(2), int64(1)
memory usage: 11.8 KB
```

Zbiór ellipses

[ ]:
```
dfs['ellipses'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   x        500 non-null    float64
 1   y        500 non-null    float64
 2   cluster  500 non-null    int64
dtypes: float64(2), int64(1)
```

```
memory usage: 11.8 KB
```

Zbiór moons

[ ]:
```
dfs['moons'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   x        500 non-null    float64
 1   y        500 non-null    float64
 2   cluster  500 non-null    int64
dtypes: float64(2), int64(1)
memory usage: 11.8 KB
```

2.3 Prezentacja klastrów na wykresach

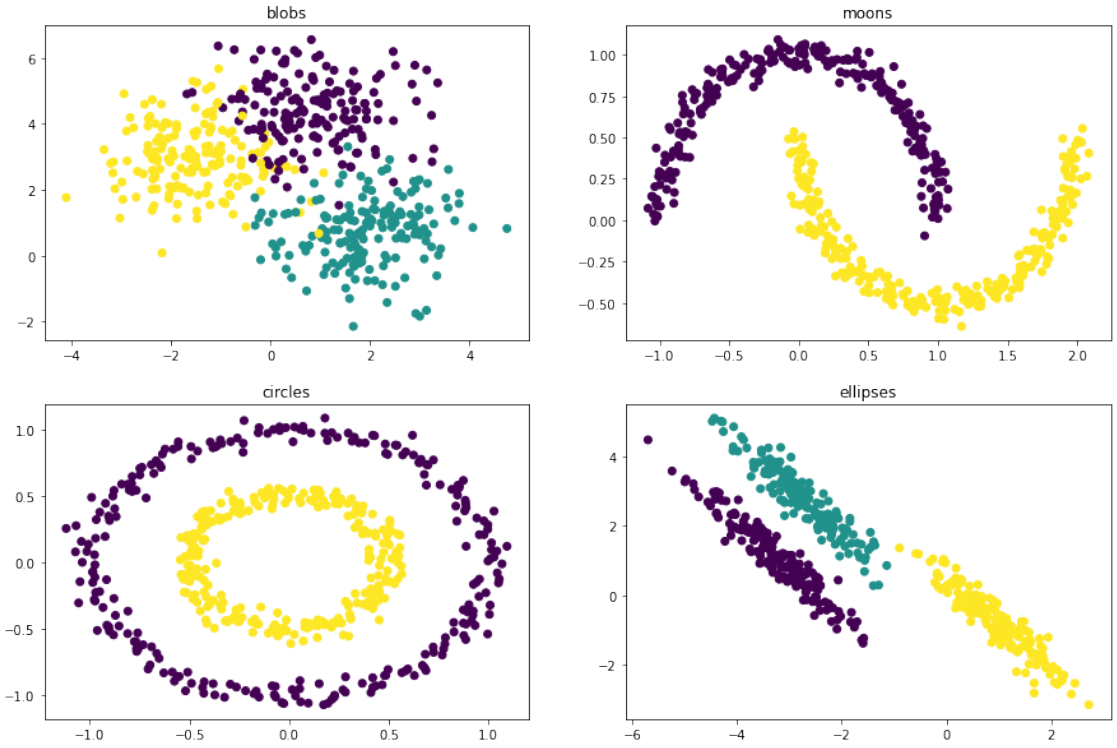
2.3.1 Wykresy klastrów dla poszczególnych zbiorów danych

Pomocnicza funkcja, rysująca wykresy

[ ]:
```
def plot_clusters(df, title):
    plt.figure(figsize=(10, 8))
    plt.scatter(df['x'], df['y'], c=df['cluster'])
    plt.title(title)
    plt.show()
```

Wykresy, przedstawiające rozmieszczenie klastrów dla wszystkich zbiorów

[ ]:
```
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

for i, (key, df) in enumerate(dfs.items()):
    ax = axs[i // 2, i % 2]
    ax.scatter(df['x'], df['y'], c=df['cluster'])
    ax.set_title(key)
```



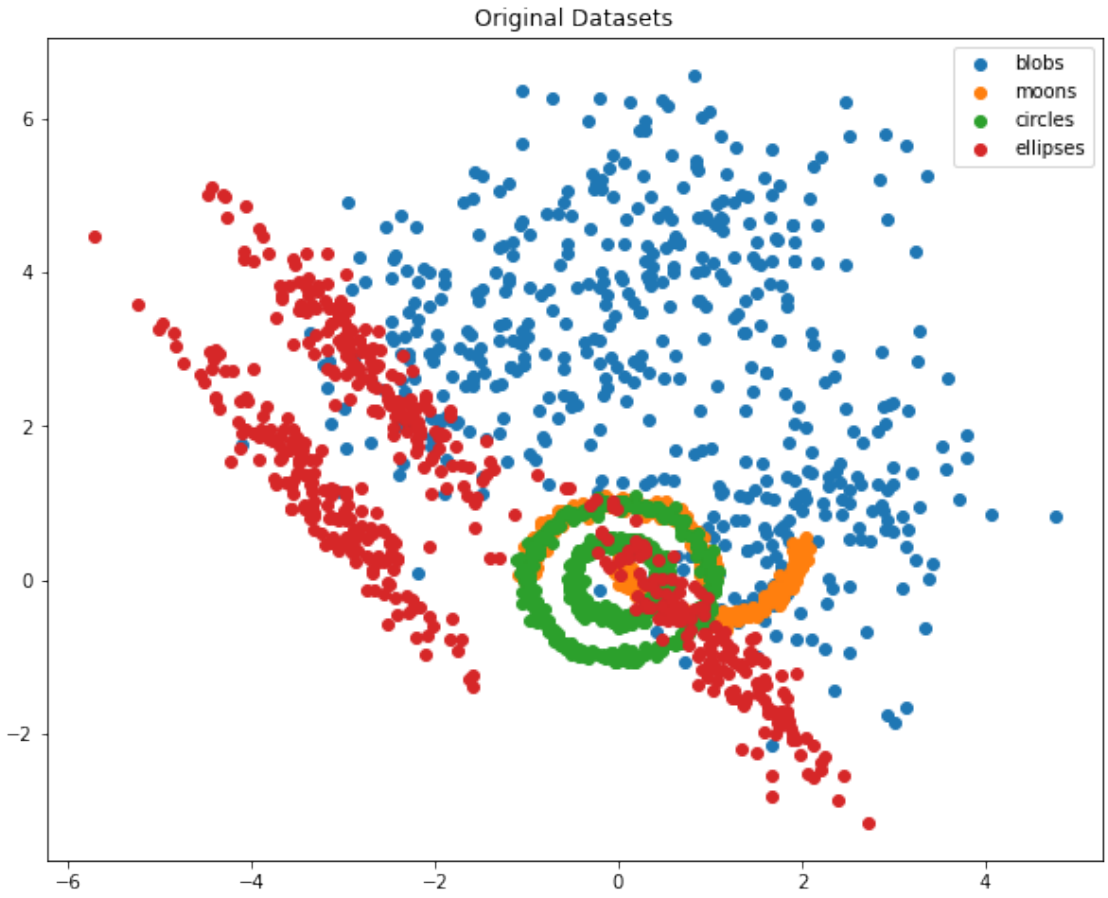
2.3.2 Wspólny wykres klastrów dla wszystkich zbiorów danych

Pomocnicza funkcja, rysująca wszystkie klastry na jednym wykresie

[ ]:
```
def plot_all_clusters(dfs, title):
    plt.figure(figsize=(10, 8))
    for key, df in dfs.items():
        plt.scatter(df['x'], df['y'], label=key)
    plt.title(title)
    plt.legend()
    plt.show()
```

Wspólny wykres

[ ]:
```
plot_all_clusters(dfs, 'Original Datasets')
```



2.4 Analiza algorytmów klasteryzacji

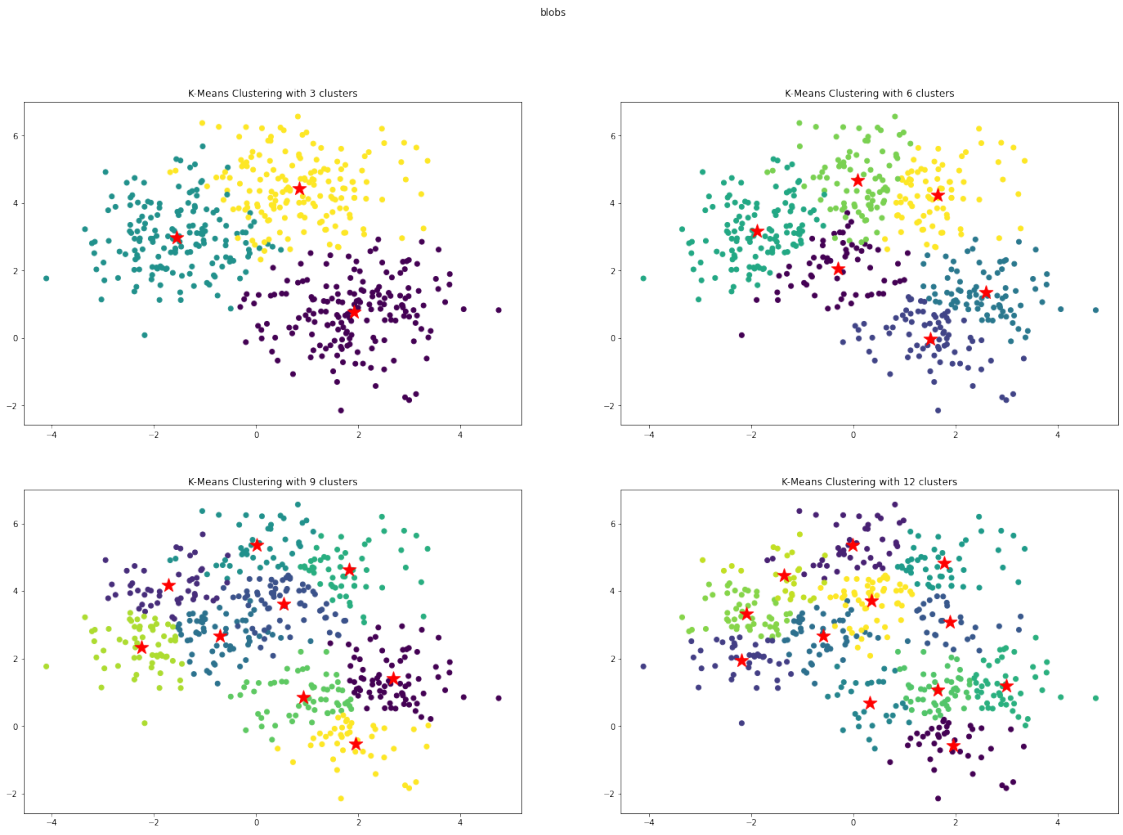
2.4.1 Algorytm k-means

Pomocnicza funkcja, rysująca rezultat klasteryzacji, dokonanej przy pomocy algorytmu k-means

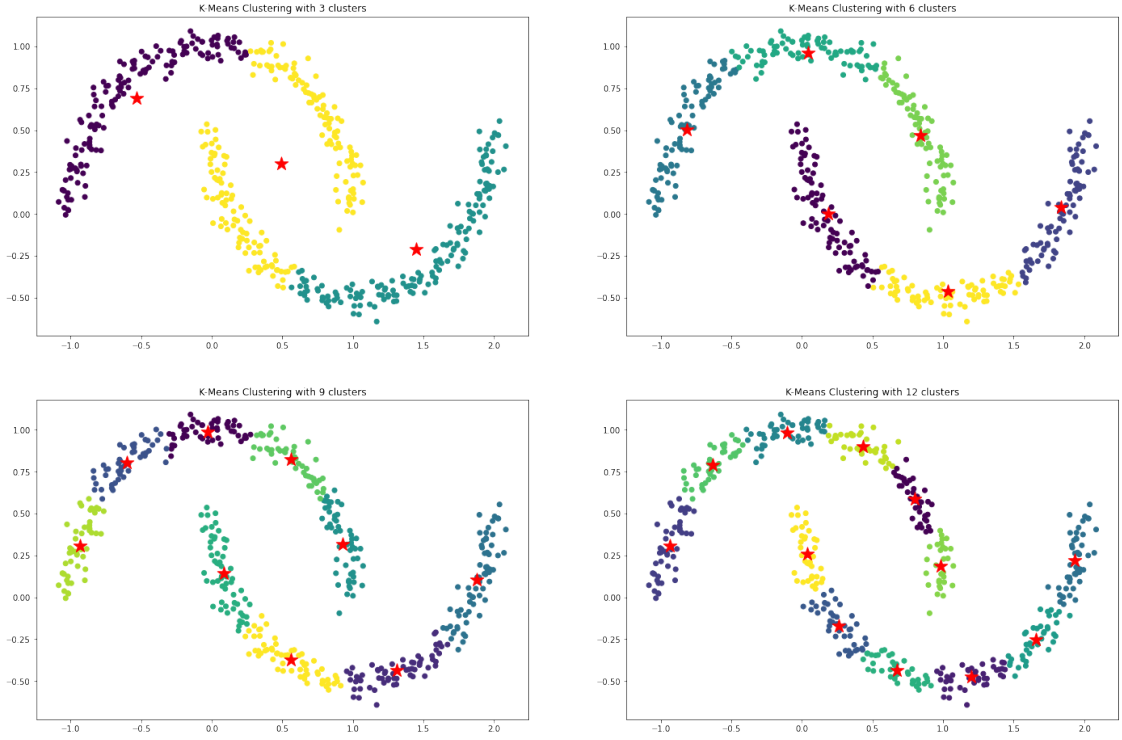
[ ]:
```
from sklearn.cluster import KMeans

def k_means(X, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X)
    labels = kmeans.labels_
    centroids = kmeans.cluster_centers_

    plt.figure(figsize=(10, 8))
    plt.scatter(X['x'], X['y'], c=labels)
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300, c='r')
    plt.title(f'K-Means Clustering with {n_clusters} clusters')
    plt.show()
```

Wykresy przedstawiające rezultaty klasteryzacji dla algorytmu k-means oraz różnych liczb klastrów kolejno dla wszystkich analizowanych w tym zadaniu zbiorów danych

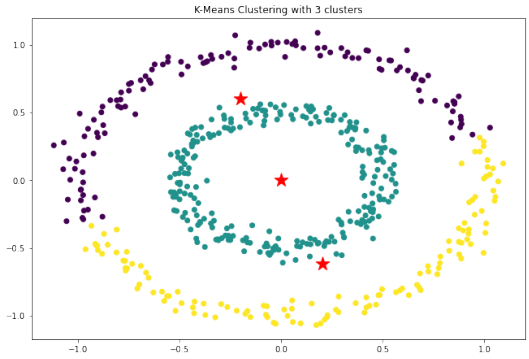
[ ]:
```
clusters = [3, 6, 9, 12]

for i, (key, df) in enumerate(dfs.items()):
    fig, axs = plt.subplots(2, 2, figsize=(24, 16))

    fig.suptitle(key)

    for j, n_clusters in enumerate(clusters):
        ax = axs[j // 2, j % 2]
        kmeans = KMeans(n_clusters=n_clusters, n_init=10)
        kmeans.fit(df)
        labels = kmeans.labels_
        centroids = kmeans.cluster_centers_

        ax.scatter(df['x'], df['y'], c=labels)
        ax.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300, c='r')
        ax.set_title(f'K-Means Clustering with {n_clusters} clusters')
```

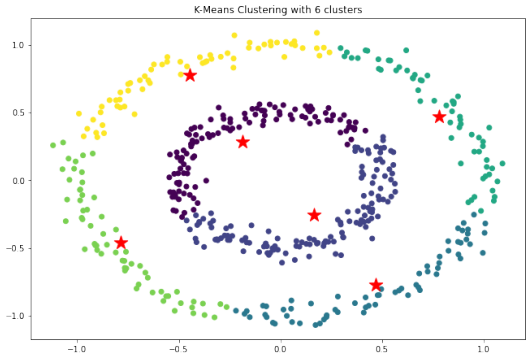


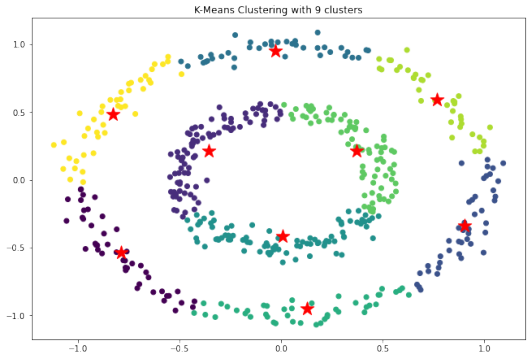
moons

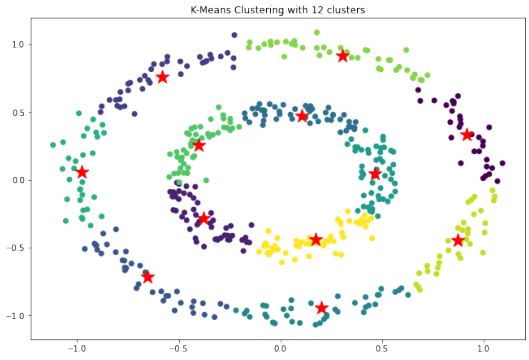

K-Means Clustering with 3 clusters
K-Means Clustering with 6 clusters
K-Means Clustering with 9 clusters
K-Means Clustering with 12 clusters

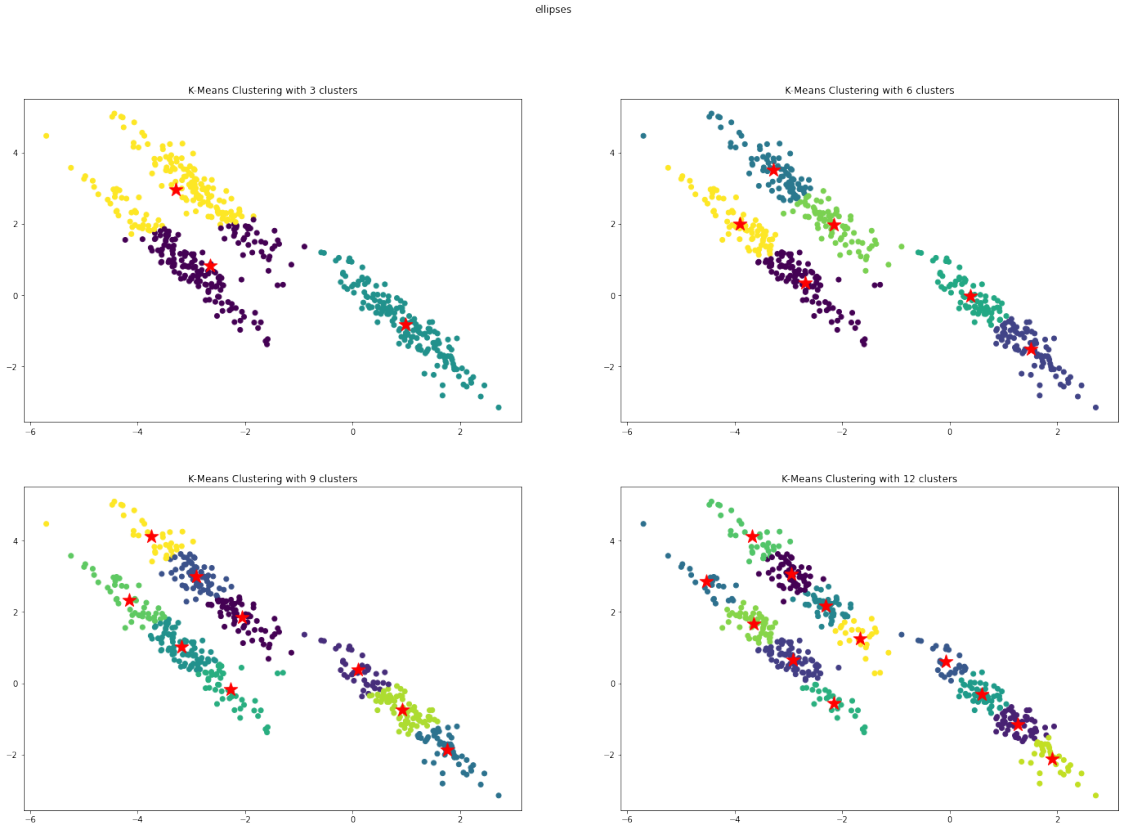
circles


K-Means Clustering with 3 clusters


K-Means Clustering with 6 clusters


K-Means Clustering with 9 clusters


K-Means Clustering with 12 clusters



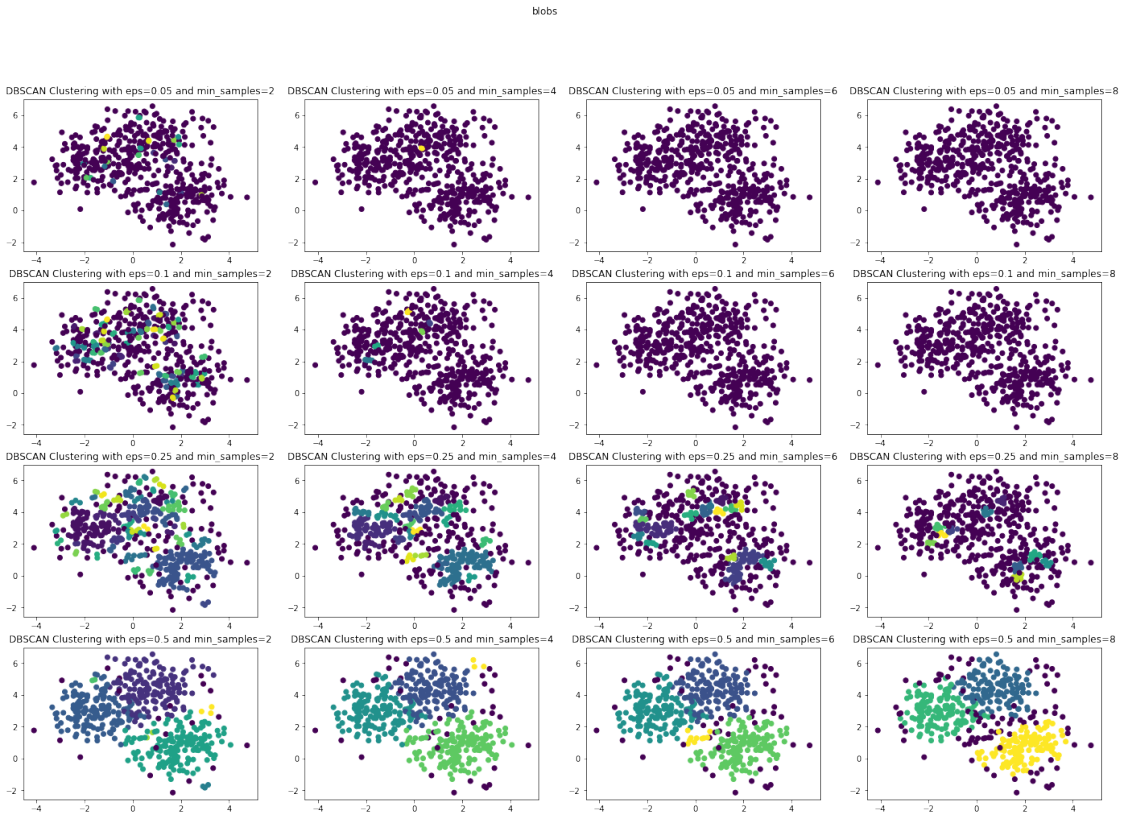
2.4.2 Algorytm DBSCAN

Pomocnicza funkcja, rysująca rezultat klasteryzacji, dokonanej przy pomocy algorytmu DBSCAN

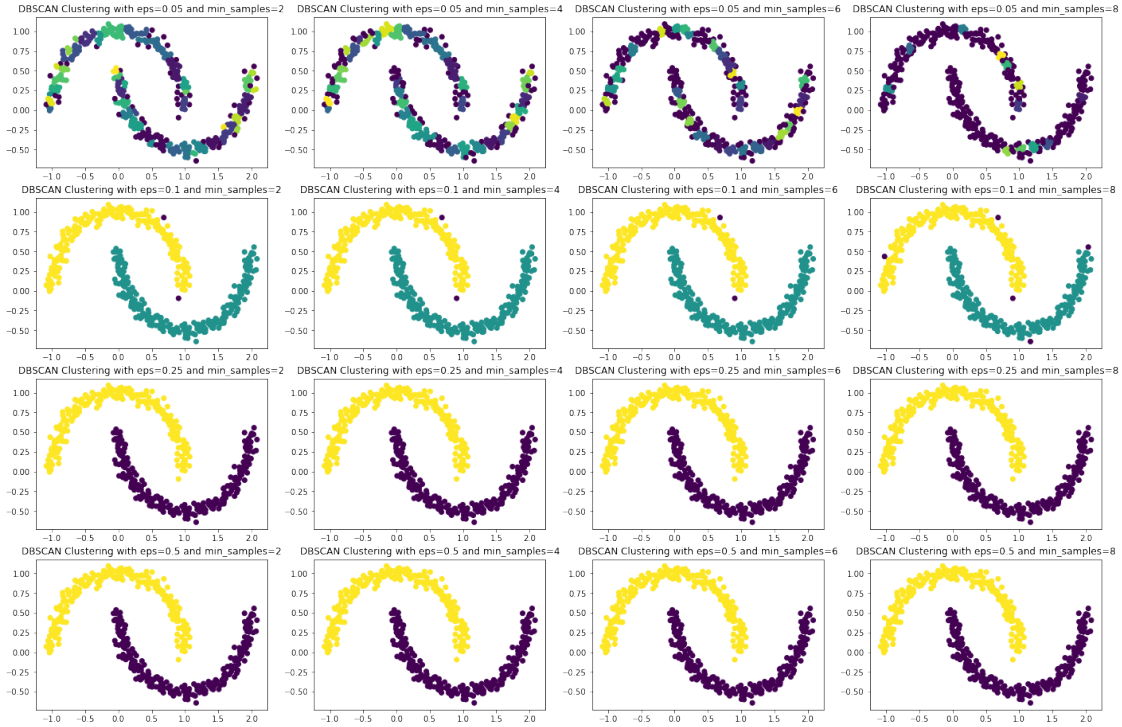
[ ]:
```
from sklearn.cluster import DBSCAN

def dbscan(X, eps, min_samples):
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    dbscan.fit(X)
    labels = dbscan.labels_

    plt.figure(figsize=(10, 8))
    plt.scatter(X['x'], X['y'], c=labels)
    plt.title(f'DBSCAN Clustering with eps={eps} and min_samples={min_samples}')
    plt.show()
```

Wykresy przedstawiające rezultaty klasteryzacji dokonanej, przy pomocy algorytmu DBSCAN, dla różnych wartości parametrów `eps` i `min_samples` kolejno dla wszystkich analizowanych w tym zadaniu zbiorów.

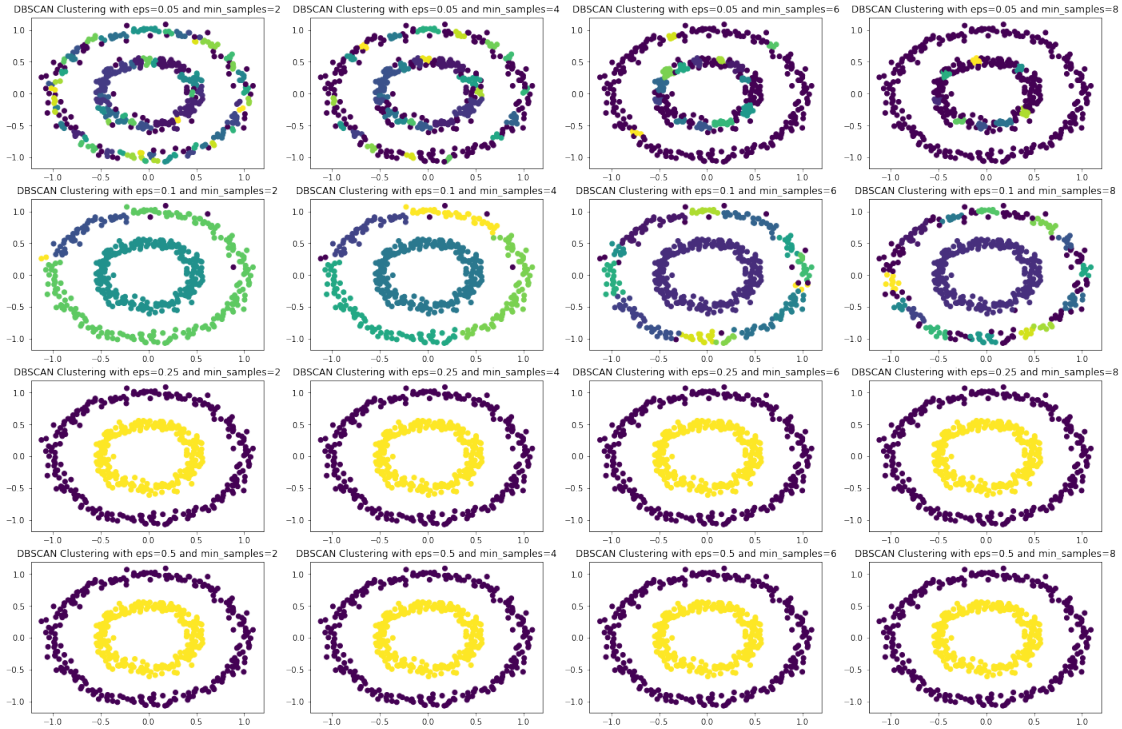
[ ]:
```
eps_list = [0.05, 0.1, 0.25, 0.5]
min_samples_list = [2, 4, 6, 8]

for i, (key, df) in enumerate(dfs.items()):
    fig, axs = plt.subplots(4, 4, figsize=(24, 16))

    fig.suptitle(key)

    for j, eps in enumerate(eps_list):
        for k, min_samples in enumerate(min_samples_list):
            ax = axs[j, k]
            dbscan = DBSCAN(eps=eps, min_samples=min_samples)
            dbscan.fit(df)
            labels = dbscan.labels_

            ax.scatter(df['x'], df['y'], c=labels)
            ax.set_title(f'DBSCAN Clustering with eps={eps} and␣
 ↪min_samples={min_samples}')
```

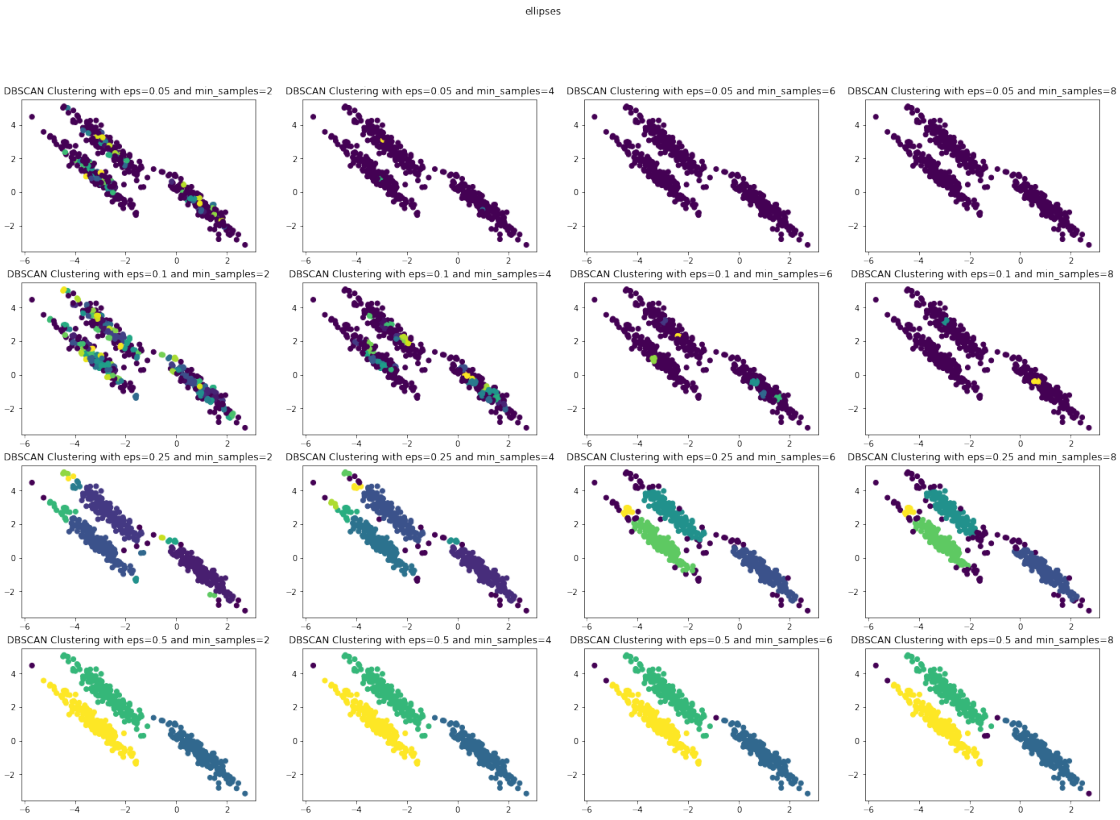


moons


DBSCAN Clustering with eps=0.05 and min_samples=2
DBSCAN Clustering with eps=0.05 and min_samples=4
DBSCAN Clustering with eps=0.05 and min_samples=6
DBSCAN Clustering with eps=0.05 and min_samples=8
DBSCAN Clustering with eps=0.1 and min_samples=2
DBSCAN Clustering with eps=0.1 and min_samples=4
DBSCAN Clustering with eps=0.1 and min_samples=6
DBSCAN Clustering with eps=0.1 and min_samples=8
DBSCAN Clustering with eps=0.25 and min_samples=2
DBSCAN Clustering with eps=0.25 and min_samples=4
DBSCAN Clustering with eps=0.25 and min_samples=6
DBSCAN Clustering with eps=0.25 and min_samples=8
DBSCAN Clustering with eps=0.5 and min_samples=2
DBSCAN Clustering with eps=0.5 and min_samples=4
DBSCAN Clustering with eps=0.5 and min_samples=6
DBSCAN Clustering with eps=0.5 and min_samples=8

circles


DBSCAN Clustering with eps=0.05 and min_samples=2
DBSCAN Clustering with eps=0.05 and min_samples=4
DBSCAN Clustering with eps=0.05 and min_samples=6
DBSCAN Clustering with eps=0.05 and min_samples=8
DBSCAN Clustering with eps=0.1 and min_samples=2
DBSCAN Clustering with eps=0.1 and min_samples=4
DBSCAN Clustering with eps=0.1 and min_samples=6
DBSCAN Clustering with eps=0.1 and min_samples=8
DBSCAN Clustering with eps=0.25 and min_samples=2
DBSCAN Clustering with eps=0.25 and min_samples=4
DBSCAN Clustering with eps=0.25 and min_samples=6
DBSCAN Clustering with eps=0.25 and min_samples=8
DBSCAN Clustering with eps=0.5 and min_samples=2
DBSCAN Clustering with eps=0.5 and min_samples=4
DBSCAN Clustering with eps=0.5 and min_samples=6
DBSCAN Clustering with eps=0.5 and min_samples=8



3 Zadanie 2

3.1 Import bibliotek

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn import metrics
```

3.2 Wczytanie danych

Pomocnicza funkcja, wczytująca i parsująca plik CSV. Funkcja zwraca nazwy kolumn oraz dane zawarte w pliku CSV z danymi

[ ]:
```
def extract_columns_and_data(content: str):
    lines = content.split('\n')

    column_names = []
    data = []
```

```
    for line in lines:
        if line.startswith("@ATTRIBUTE"):
            column = line.split()[1].strip()
            column_names.append(column)
        elif not line.startswith(("@RELATION", "@DATA", "%")) and line != '':
            data.append(list(map(float, line.split(','))))

    return column_names, data
```

Wczytujemy dane, a następnie tworzymy DataFrame

[ ]:
```
data_path = "dataset/banknotes.csv"

with open(data_path, 'r') as f:
    content = f.read()
    column_names, data = extract_columns_and_data(content)

df = pd.DataFrame(data, columns=column_names)
```

Zobaczmy, z jakimi danymi mamy do czynienia

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   counterfeit  200 non-null    float64
 1   Length       200 non-null    float64
 2   Left         200 non-null    float64
 3   Right        200 non-null    float64
 4   Bottom       200 non-null    float64
 5   Top          200 non-null    float64
 6   Diagonal     200 non-null    float64
dtypes: float64(7)
memory usage: 11.1 KB
```

[ ]:
```
df.head()
```

[ ]:

	counterfeit	Length	Left	Right	Bottom	Top	Diagonal
0	0.0	214.8	131.0	131.1	9.0	9.7	141.0
1	0.0	214.6	129.7	129.7	8.1	9.5	141.7
2	0.0	214.8	129.7	129.7	8.7	9.6	142.2
3	0.0	214.8	129.7	129.6	7.5	10.4	142.0
4	0.0	215.0	129.6	129.7	10.4	7.7	141.8

3.3 Normalizacja danych

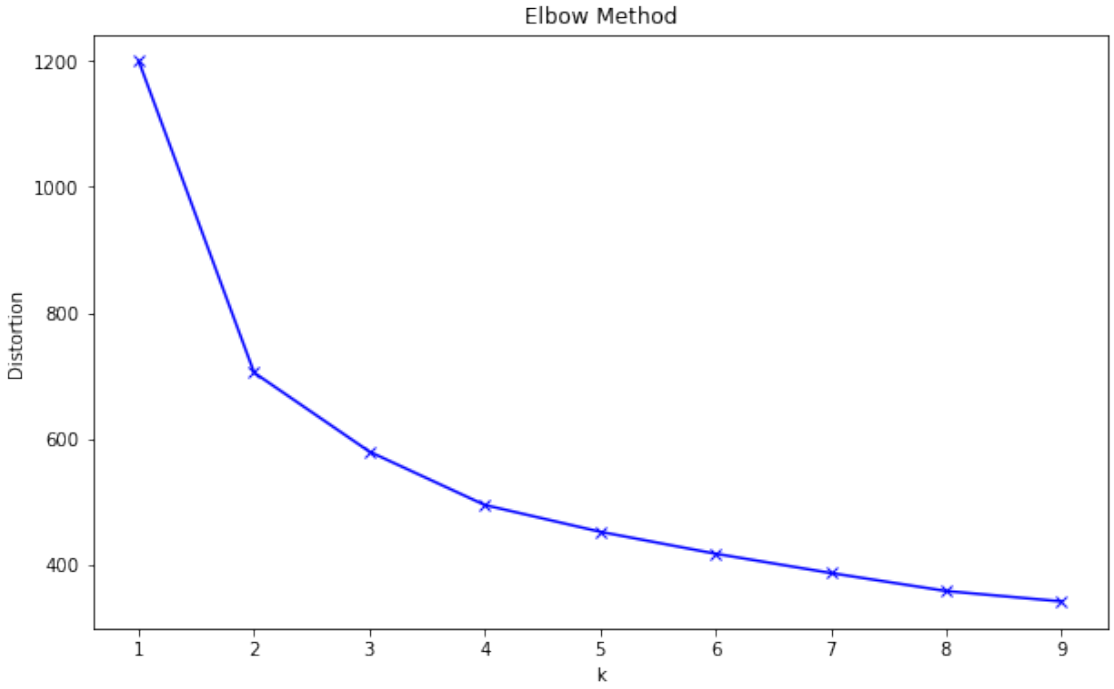
Do normalizacji danych wykorzystamy `StandardScaler`. Z oryginalnego zbioru danych odrzucamy kolumnę `counterfeit`, która oznacza przyporządkowanie banknotu do odpowiedniej klasy (`counterfeit` lub `genuine`). Zostawiamy więc jedynie kolumny zawierające informacje o cechach banknotów.

[ ]:
```
features = ['Length', 'Left', 'Right', 'Bottom', 'Top', 'Diagonal']
X = df[features]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3.4 Oszacowanie optymalnej liczby klastrów dla metody k-means

3.4.1 Metoda *elbow*

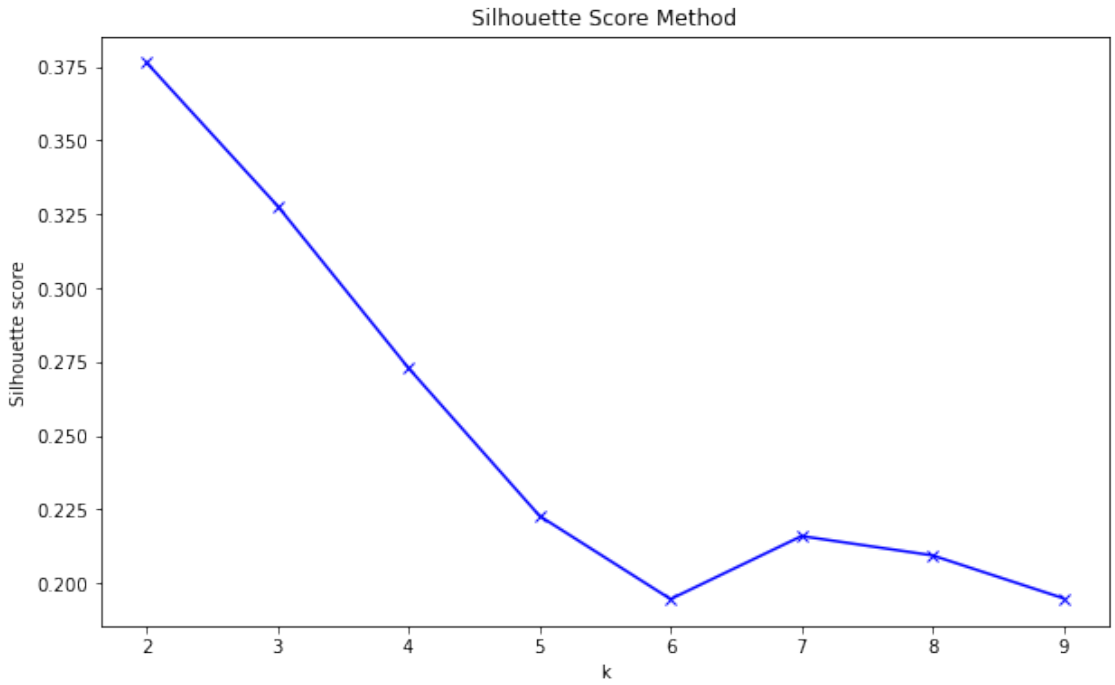
[ ]:
```
import matplotlib.pyplot as plt

distortions = []
K = range(1, 10)
for k in K:
    kmeans_model = KMeans(n_clusters=k, n_init=10)
    kmeans_model.fit(X_scaled)
    distortions.append(kmeans_model.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('Elbow Method')
plt.show()
```



W przypadku metody łokcia (*elbow*), najlepszą konfigurację klastrów dostaniemy dla liczby klastrów, dla której na wykresie obserwujemy największe przegięcie. Dobrze to widać na otrzymanym wyżej wykresie i punktem, dla którego obserwujemy największą zmianę tempa spadku wartości, a więc największe przegięcie jest punkt dla $k = 2$

3.4.2 Metoda *silhouette score*

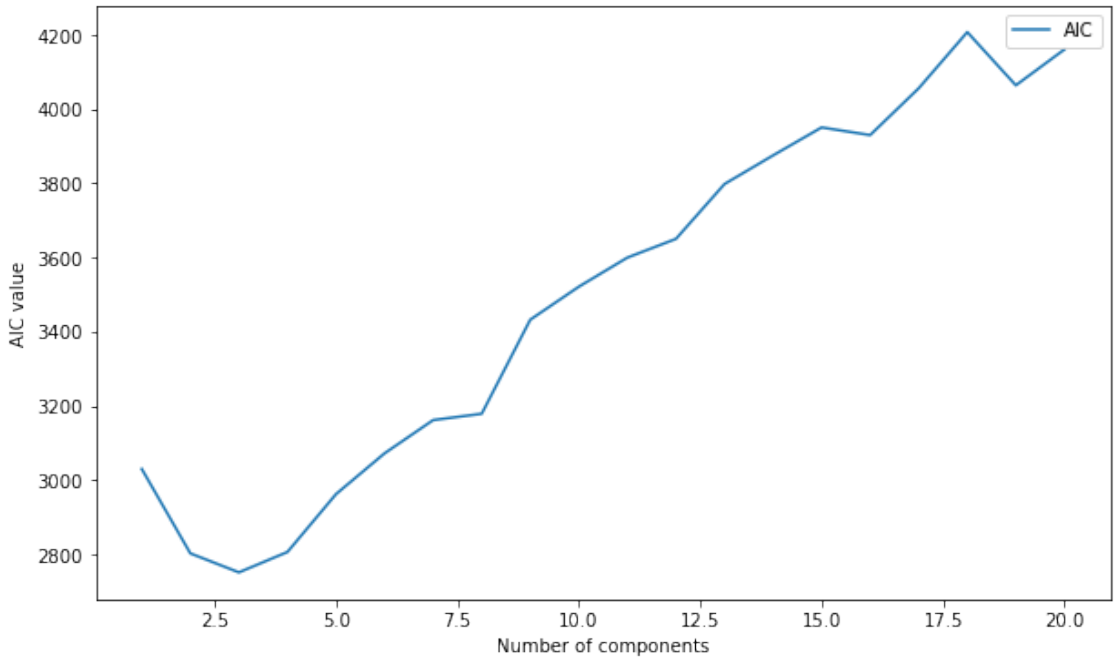
[ ]:
```
silhouette_scores = []
K = range(2, 10)
for k in K:
    kmeans_model = KMeans(n_clusters=k, n_init=10)
    kmeans_model.fit(X_scaled)
    silhouette_scores.append(metrics.silhouette_score(X_scaled, kmeans_model.
 ↪labels_))

plt.figure(figsize=(10, 6))
plt.plot(K, silhouette_scores, 'bx-')
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.title('Silhouette Score Method')
plt.show()
```



W przypadku metody wyniku profilu (*silhouette score*), najlepszą konfigurację klastrów dostaniemy dla najwyższej wartości *silhouette score*. Możemy tę wartość odczytać z powyższego wykresu. Jak możemy zauważyć, najwyższy wynik dostaliśmy dla $k = 2$, dlatego optymalna liczba klastrów to 2

3.4.3 Metoda *Akaike Information Criterion*

[ ]:
```
from sklearn.mixture import GaussianMixture

n_components = np.arange(1, 21)
models = [GaussianMixture(n, covariance_type='full', random_state=0).
 ↪fit(X_scaled) for n in n_components]

plt.figure(figsize=(10, 6))
plt.plot(n_components, [m.bic(X_scaled) for m in models], label='AIC')
plt.xlabel('Number of components')
plt.ylabel('AIC value')
plt.legend(loc='best')
plt.show()

best_n_components = np.argmin([m.bic(X_scaled) for m in models]) + 1

print(f'Best number of components: {best_n_components}')
```




```
Best number of components: 3
```

W przypadku metody kryterium informacyjnego Akaikego (*Akaike Information Criterion*), najlepszą konfigurację klastrów dostaniemy dla liczy klastrów równej 3

3.5 Sprawdzanie poprawności przypisania banknotów do klastrów dla metody DBSCAN

[ ]:
```
dbscan = DBSCAN()
dbscan_labels = dbscan.fit_predict(X_scaled)

print(f'Number of clusters: {len(set(labels))}')
print(f'Number of outliers: {np.sum(labels == -1)}')
```

```
Number of clusters: 2
Number of outliers: 1
```

Jak możemy zauważyć, w przypadku, gdy korzystamy z domyślnych wartości parametrów metody kalsteryzacji DBSCAN (`eps=0.5` i `min_samples=5`), 1 banknot nie został poprawnie zaklasyfikowany

3.6 Ocena jakości otrzymanych klastrów

W przypadku metody k-means wykorzystujemy uzyskaną wcześniej optymalną liczbę klastrów

[ ]:
```

```

```
from sklearn.metrics import homogeneity_score, completeness_score,␣
 ↪v_measure_score

# Klasteryzacja k-means
kmeans = KMeans(n_clusters=2, n_init=10)
kmeans_clusters = kmeans.fit_predict(X_scaled)

# Klasteryzacja DBSCAN
dbscan = DBSCAN()
dbscan_clusters = dbscan.fit_predict(X_scaled)

# Obliczenie metryk
labels_true = df.iloc[:, 0]

homogeneity_kmeans = homogeneity_score(labels_true, kmeans_clusters)
homogeneity_dbscan = homogeneity_score(labels_true, dbscan_clusters)

completeness_kmeans = completeness_score(labels_true, kmeans_clusters)
completeness_dbscan = completeness_score(labels_true, dbscan_clusters)

v_measure_kmeans = v_measure_score(labels_true, kmeans_clusters)
v_measure_dbscan = v_measure_score(labels_true, dbscan_clusters)

# Wypisanie wyników
print("K-means:")
print(f"Homogeneity: {homogeneity_kmeans}")
print(f"Completeness: {completeness_kmeans}")
print(f"V-Measure: {v_measure_kmeans}")

print("\nDBSCAN:")
print(f"Homogeneity: {homogeneity_dbscan}")
print(f"Completeness: {completeness_dbscan}")
print(f"V-Measure: {v_measure_dbscan}")
```

```
K-means:
Homogeneity: 0.7942888437190893
Completeness: 0.7979767420525216
V-Measure: 0.7961285220549091

DBSCAN:
Homogeneity: 0.0
Completeness: 1.0
V-Measure: 0.0
```

Na podstawie otrzymanych wyników wniosków, możemy dojść do następujących wniosków:

1. **K-means:**
   - *Homogenność* wynosi 0.7943, co oznacza, że każdy klaster zawiera tylko elementy z jednej klasy, co jest pożądaną cechą, ale nie osiągnięto maksymalnej wartości (1.0).

- *Kompletność* wynosi 0.7980, co sugeruje, że większość elementów tej samej klasy została zebrana w jednym klastrze, ale jeszcze nie został osiągnięty stan idealny (1.0).
- *Miara V* wynosi 0.7961, co jest średnią harmoniczną homogenności i kompletności. Wynik bliski 1.0 wskazuje na wyższą jakość grupowania. Wartość 0.7961 pokazuje, że metoda K-means ma dość dobrą jakość grupowania.

2. **DBSCAN:**
   - *Homogenność* wynosi 0.0, co oznacza, że każdy klaster zawiera elementy z różnych klas, co oznacza, że metoda DBSCAN nie poradziła sobie z prawidłowym grupowaniem danych.
   - *Kompletność* wynosi 1.0, co oznacza, że wszyscy członkowie tej samej klasy znajdują się w jednym klastrze, ale zważywszy na brak homogenności, wynik ten sugeruje, że metoda próbowania rozwiązała problem przypisując wszystkie elementy do jednego klastra.
   - *Miara V* wynosi 0.0, co oznacza, że jakość grupowania dla metody DBSCAN jest znacznie niższa niż w przypadku metody K-means.

W oparciu o powyższe wyniki analizy, metoda K-means radzi sobie lepiej z dzieleniem na klastry w tym przypadku niż DBSCAN, co zobaczyć można poprzez wyższe wartości dla homogenności, kompletności oraz miary V.