
	Politechnika Bydgoska im. J.J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki		
Przedmiot	Algorytmy i eksploracja danych		
Prowadzący	dr inż. Michał Kruczkowski		
Temat	Uczenie bez nadzoru		
Studenci	Adam Szreiber, Cezary Naskręt		
Nr ćw.	3	Data wykonania	20.10.2023

1. Cel ćwiczenia

Uczenie maszynowe bez nadzoru to obszar uczenia maszynowego, w którym algorytmy są trenowane na danych, które nie mają przypisanych etykiet lub wyników, w przeciwieństwie do uczenia maszynowego nadzorowanego, gdzie algorytmy uczą się na danych z etykietami (wynikami). W uczeniu maszynowym bez nadzoru celem jest odnalezienie ukrytych wzorców, struktury lub zależności w danych, a także grupowanie danych w klastry lub redukcja wymiarów.

2. Przebieg laboratorium

2.1. Zadanie 1

Zbiór danych: <http://archive.ics.uci.edu/ml/machine-learning-databases/00396/> Dane dotyczące transakcji różnych produktów na przestrzeni 51-tygodni zawierają zarówno wartości oryginalne jak i znormalizowane.

2.2. Zadanie 2

Wykonaj analizę skupień danych dla następujących algorytmów:

- K-MEANS
- K-MEANS++
- *K-MEDIOD

Na początku importuje wszystkie biblioteki.

Następnie przygotowuję dane do obróbki.

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Wczytaj dane z pliku CSV do DataFrame
data = pd.read_csv(
    '/home/adam-pc/Documents/mgr/AED-AlgorytmyEksploracjiDanych/lab3/Sales_Transactions_Dataset_Weekly.csv')

# Przetwórz dane, usuń brakujące wartości
data = data.dropna()

values = []

# Utwórz pętlę od 0 do 51 (włącznie)
for i in range(52):
    value = f'W{i}'
    values.append(value)

features = data[values].copy()
```

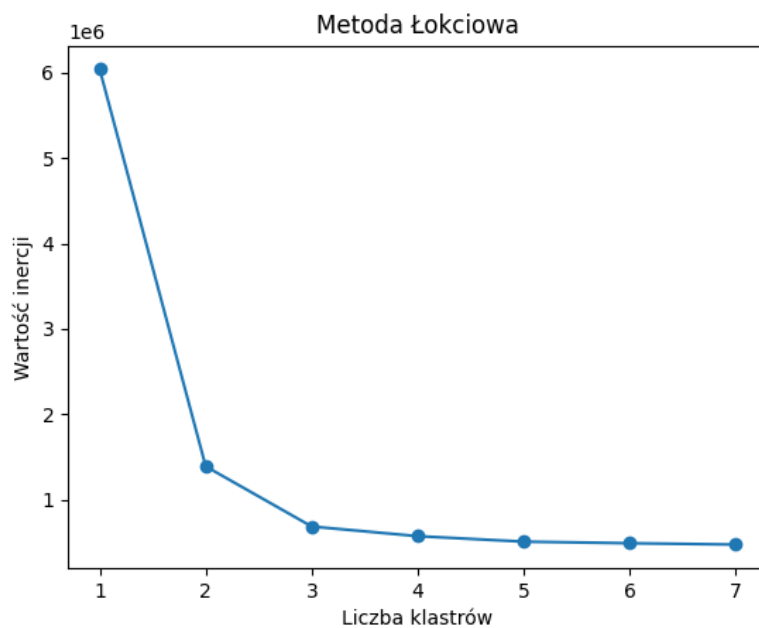
Kod przedstawiający metodę łokciową - służącą w wyznaczaniu ilości klastrow.

```
# Przygotuj listę wartości inercji dla różnej liczby klastrow
inertia_values = []

# Przeprowadź analizę K-means dla różnych liczb klastrow
for n_clusters in range(1, 8):
    kmeans = KMeans(n_clusters=n_clusters, n_init=10)
    kmeans.fit(features)
    inertia_values.append(kmeans.inertia_)

# Wykres metody "łokcia"
plt.plot(range(1, 8), inertia_values, marker='o')
plt.xlabel('Liczba klastrow')
plt.ylabel('Wartość inercji')
plt.title('Metoda łokciowa')
plt.show()
```

Na wykresie poszukaj "łokcia" (punkt zwany "punkt łokcia"), który jest miejscem, w którym spadek wartości funkcji kosztu zaczyna się wyhamowywać. Z wykresu wynika że przy wartości 3 jest wyrównanie na wykresie - co oznacza że to jest nasza ilość klastrow.



Kod służący wczytaniu ilości klastrów od użytkownika, a następnie wykonaniu uczenia modelu sieci KMeans.

Do danych w `features` zostanie dodana kolumna `Cluster` i w niej będą przechowywane etykiety klastra do którego przynależą.

W ostatniej fazie zostaje tworzony wykres w raz z centroidami.

```
# Poprosz użytkownika o wprowadzenie liczby klastrów
n_clusters = int(input("Podaj liczbę klastrów: "))

kmeans = KMeans(n_clusters=n_clusters, n_init=10, init='k-means++')
kmeans.fit(features)

# Dodaj wyniki klastry do DataFrame
features['Cluster'] = kmeans.labels_
print(features)

# Wykres
plt.scatter(features['W2'], features['W3'],
            c=features['Cluster'])
plt.xlabel('W0')
plt.ylabel('W1')
plt.title('Analiza K-means')

# Dodaj centra klastrów na wykres
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], c='red',
            marker='x', s=200, label='Centroids')

# Dodaj etykiety do centrów
for i, txt in enumerate(centroids):
    plt.annotate(f'Centroid {i}', (centroids[i, 0], centroids[i, 1]),
                textcoords="offset points", xytext=(0, 10), ha='center')

plt.legend()
plt.show()
```

Przedstawienie wykresu z pogrupowanymi punktami.



Widok danych wraz z etykietą klastra w ostatniej kolumnie.

[illegible]

W kolejnym podpunkcie należało użyć metody `k-means++`.

Nie nastąpiła żadna większa zmiana w kodzie.

Poniżej przedstawiono fragment kodu zawierającego inicjalizację modelu `KMeans`` lecz parametr `init`` świadczy o tym iż będzie użyty algorytm `k-means++``

```
kmeans = KMeans(n_clusters=n_clusters, n_init=10, init='k-means++')
kmeans.fit(features)
```

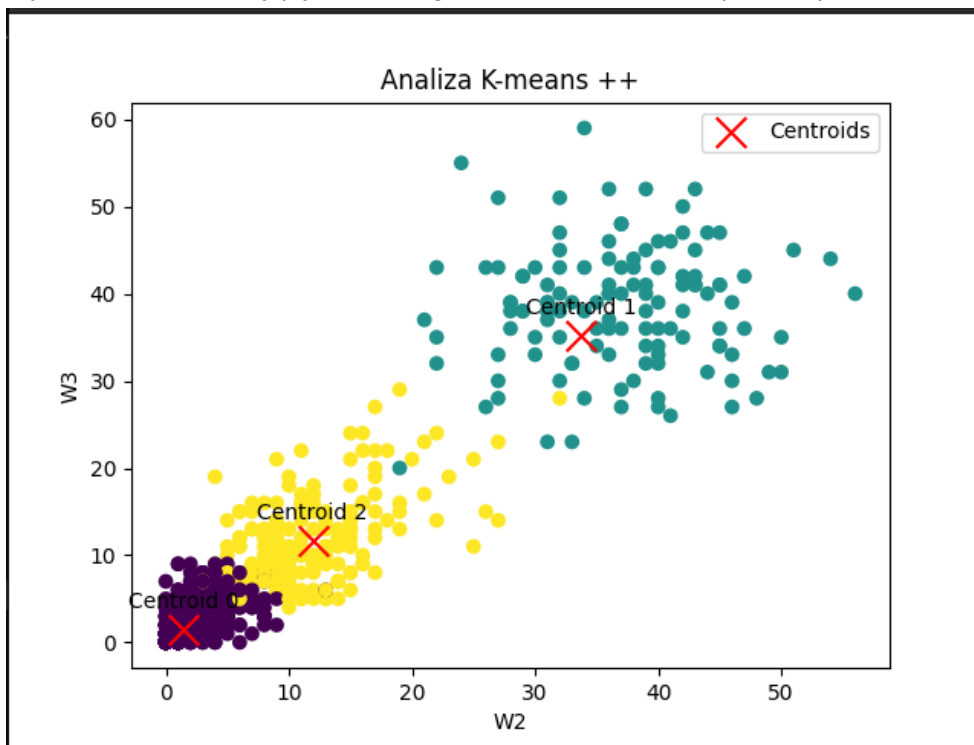
Wyniki z konsoli.

```

Podaj liczbę klastrow: 3
W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14 W15 W16 W17 W18 W19 ... W33 W34 W35 W36 W37 W38 W39 W40 W41 W42 W43 W44 W45 W46 W47 W48 W49 W50 W51 Cluster
0 11 12 10 8 13 12 14 21 6 14 11 14 16 9 9 9 14 9 3 12 ... 6 5 14 10 9 12 17 7 11 4 7 8 10 12 3 7 6 5 10 2
1 7 6 3 2 7 1 6 3 3 3 2 2 6 2 0 6 2 7 7 9 ... 2 3 10 5 2 7 3 2 5 2 4 5 1 1 4 5 1 6 0 0
2 7 11 8 9 10 8 7 13 12 6 14 9 4 7 12 8 7 11 10 7 ... 8 7 9 6 12 12 9 3 5 6 14 5 5 7 8 14 8 7 2
3 12 8 13 5 9 6 9 13 13 11 8 4 5 4 15 7 11 9 15 4 ... 12 5 10 8 6 8 8 12 6 9 10 3 4 6 8 14 8 7 8 2
4 8 5 13 11 6 7 9 14 9 9 11 18 8 4 13 8 10 15 6 13 ... 5 9 7 4 8 8 5 5 8 7 11 7 12 6 6 5 11 8 9 2
...
806 0 0 1 0 0 2 1 0 0 1 0 0 1 0 1 1 3 0 0 0 ... 0 0 0 0 0 0 1 2 1 0 0 1 1 0 0 1 0 0 2 0 0
807 0 1 0 0 1 2 2 6 0 1 0 2 2 0 5 1 1 1 2 3 ... 4 5 4 3 2 4 6 6 7 3 3 4 2 4 5 5 5 6 5 0
808 1 0 0 0 1 1 2 1 1 0 0 1 0 0 0 0 0 0 0 ... 1 0 0 0 0 0 0 1 2 0 0 2 0 0 0 0 4 3 0
809 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 ... 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 2 0 0
810 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 ... 0 0 3 1 1 1 0 1 0 0 0 0 0 0 0 1 0
[811 rows x 53 columns]

```

Wykres przedstawiający dane pogrupowane za pomocą metody `k-means++`



W kolejnym podpunkcie należało wykorzystać metodę KMedoids.

Poniżej przedstawiono kod który realizuje uczenie oraz tworzenie wykres

```

# Przeprowadź analizę K-Medoids na kopii danych
kmedoids = KMedoids(n_clusters=n_clusters) # Określ liczbę klastrow, np. 3
kmedoids.fit(features)

# Dodaj wyniki klastry do kopii DataFrame
features['Cluster'] = kmedoids.labels_

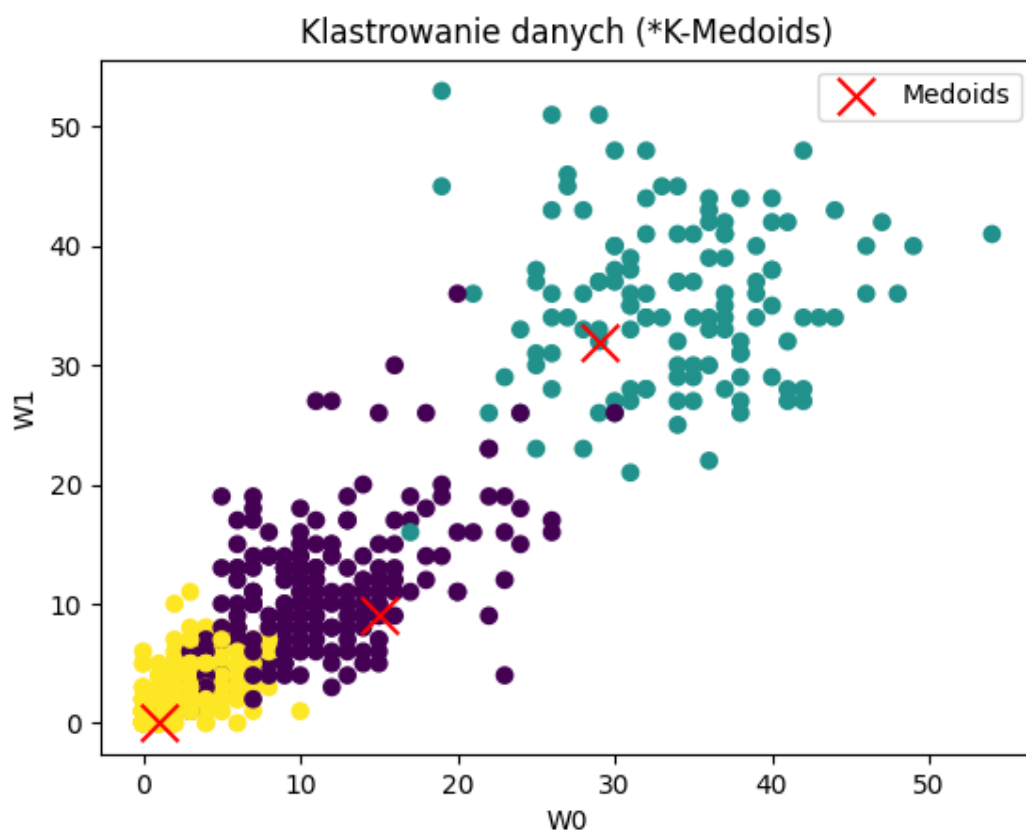
# Wykres klastrow dla całego zestawu danych (K-Medoids)
plt.scatter(features['W0'], features['W1'],
            c=features['Cluster'], cmap='viridis')
plt.xlabel('W0')
plt.ylabel('W1')
plt.title('Klastrowanie danych (*K-Medoids)')

# Dodaj centra klastrow (medoidy) na wykres
medoids = features.iloc[kmedoids.medoid_indices_]
plt.scatter(medoids['W0'], medoids['W1'], c='red',
            marker='x', s=200, label='Medoids')

plt.legend()
plt.show()

```

Wykres grupowania metodą KMedoids z centroidami.



2.3. Zadanie 3

Wykonaj klasteryzację dla tego samego zbioru danych na podstawie algorytmów:

- Agglomerative Clustering z dowolnym typem klasteryzacji, typem wiązania oraz metryką odległości
- DBSCAN z domyślnymi wartościami argumentów

Inicjalizacja modelu AgglomerativeClustering,

```
# Wybierz parametry AgglomerativeClustering
n_clusters = 3 # Określ liczbę klastrow, np. 3

# Przeprowadź analizę AgglomerativeClustering
agg_clustering = AgglomerativeClustering(
    n_clusters=n_clusters, linkage='ward', affinity='euclidean')
agg_clustering.fit(features)

# Dodaj wyniki klastry do kopii DataFrame
features['Cluster'] = agg_clustering.labels_
```

Zliczanie ilości wystąpień elementów danej klasy.

```
# Zakresy przedziałów wartości
n_bins = 10
value_ranges = np.linspace(0, data.max().max(), n_bins + 1)

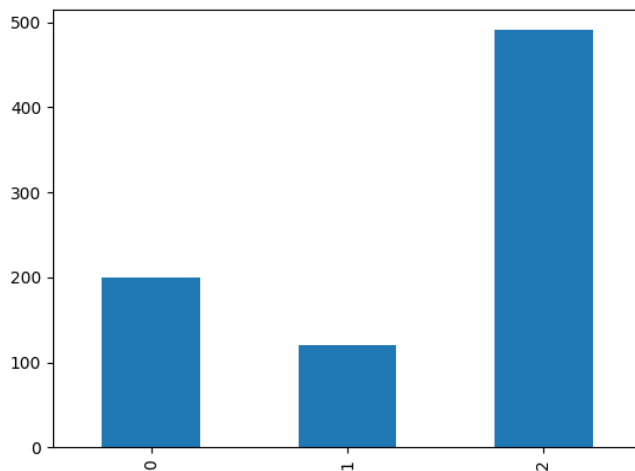
# Inicjalizacja słownika do zliczania klastrow
cluster_counts = {cluster: [] for cluster in range(n_clusters)}

# Iteruj po kolumnie 'Cluster'
for cluster in features['Cluster']:
    for i in range(n_clusters):
        # Dla każdego klastra zapisz wartość 1, jeśli jest to szukane
        cluster_counts[i].append(1 if cluster == i else 0)

# Konwertuj słownik na DataFrame
cluster_counts_df = pd.DataFrame(cluster_counts)

# Wykreśl wykres słupkowy ilości wystąpień klastrow
cluster_counts_df.sum().plot(kind='bar')
```

Wyświetlenie obfitości wystąpień klas.



Następnie przeszedłem do obróbki danych pod `heatmap`

Poniżej tworze 10 zakresów liczbowych, od wartości 0 do maksymalnej wartości występującej w oryginalnej tablicy "data".

Liczmy ilość wystąpień danej wystąpień danych z danego przedziały w każdej z klas

```
# Inicjalizacja słownika do przechowywania wyników
results = {i: [] for i in range(n_clusters)}

# Podziel dane na 10 równych zakresów
n_ranges = 10
range_size = len(features) // n_ranges

for i in range(n_ranges):
    start = i * range_size
    end = (i + 1) * range_size

    # Dla każdego klastra, zlicz wystąpienia w danym zakresie
    for cluster in range(n_clusters):
        counts = (features.iloc[start:end]['Cluster'] == cluster).sum()
        results[cluster].append({'range': f'{start}-{end}', 'count': counts})

# Konwersja danych na ramkę danych
df = pd.DataFrame(results).T

# Tworzenie tablicy z wartościami zakresów "range"
range_values = [item['range'] for item in results[0]]

# Tworzenie tablicy z wartościami pola "count"
v = []

for key, value in results.items():
    counts = [item['count'] for item in value]
    v.append(counts)

# Tworzenie heatmapy
plt.figure(figsize=(10, 6))
sns.heatmap(v, cmap='hot', annot=True, fmt='d', xticklabels=range_values,
            vmin=0, vmax=max(map(max, v)))
plt.xlabel('Range')
plt.ylabel('Cluster')
plt.title('Heatmap')
plt.show()
```


Po obróbce danych możemy wyświetlić heatmap

Interpretacja danych:

Heatmapa przedstawia 3 klastry (oś y).

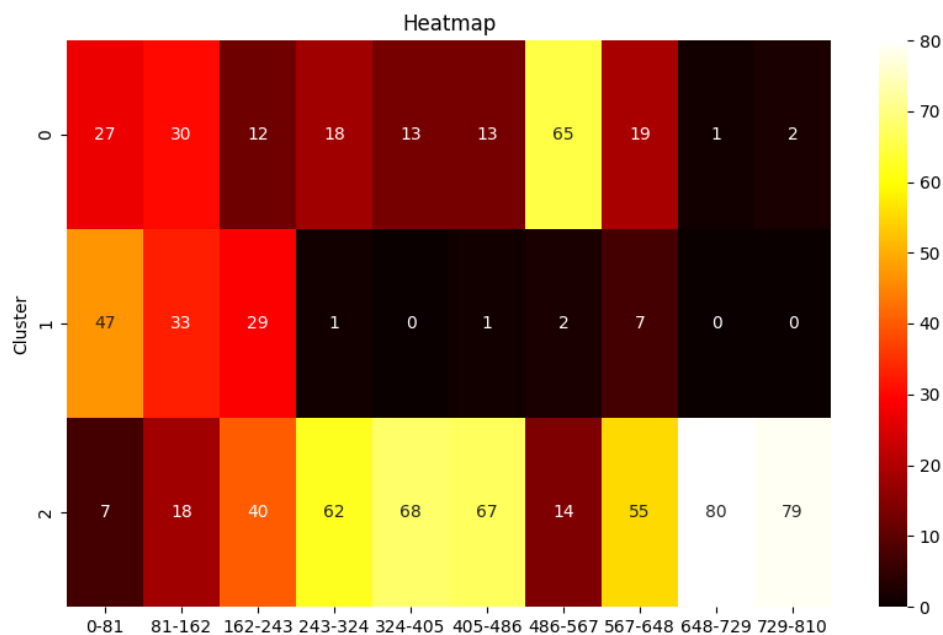
Oś x przedstawia zakresy indeksy danych w tablicy zwróconej poprzez predykcję modelu.

Natomiast nałożone kolory tj. im jaśniejszy kolor tym większa ilość wystąpień.

Z poniższego wykresu możemy zauważyć, że klaster 2 najwięcej występował w pod koniec datasetu, natomiast najmniej na początku.

Zupełnie inaczej wygląda klaster 1, który to można powiedzieć, występował tylko na początkowych 250 pozycjach.

Elementy przynależące do klaster 0, były niemal równomiernie rozłożone po całym dataset, za wyjątkiem zakresów 485-565 gdzie występował pik częstotliwości występowania (65 razy - co stanowi 80% predykcji), 650-end gdzie klaster 0 praktycznie nie występował.



W kolejnym kroku należało przygotować dane do denogramu.

```
result = []

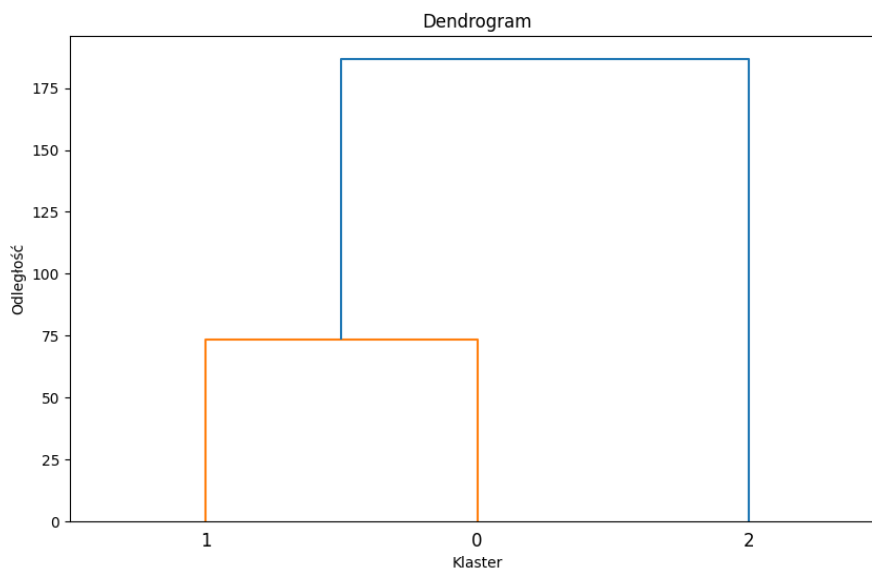
for key, value in results.items():
    counts = [item['count'] for item in value]
    result.append(counts)

# Przygotowanie danych do hierarchicznego klastrowania
X = np.array(result)

# Tworzenie dendrogramu
linked = linkage(X, 'ward')

# Rysowanie dendrogramu
plt.figure(figsize=(10, 6))
dendrogram(linked, orientation='top', labels=list(results.keys()),
            distance_sort='descending', show_leaf_counts=True)
plt.title('Dendrogram')
plt.xlabel('Klaster')
plt.ylabel('Odległość')
plt.show()
```

Wyświetlenie denogramu.



W pierwszej kolejności, po wczytaniu danych, należy ustandaryzować dane. W przypadku, gdy niektóre cechy mają większy wpływ na wyniki analizy niż inne, to może prowadzić to do błędnych wyników lub zniekształcenia struktury klastrów. Standaryzacja pomaga uniknąć takich sytuacji.

```
# Standaryzacja danych
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features)

# Wybierz parametry DBSCAN
eps = 0.2 # Promień sąsiedztwa
min_samples = 5 # Minimalna liczba próbek w sąsiedztwie

# Przeprowadź analizę klastrowania DBSCAN
dbscan = DBSCAN(eps=eps, min_samples=min_samples)
dbscan.fit(X_scaled)

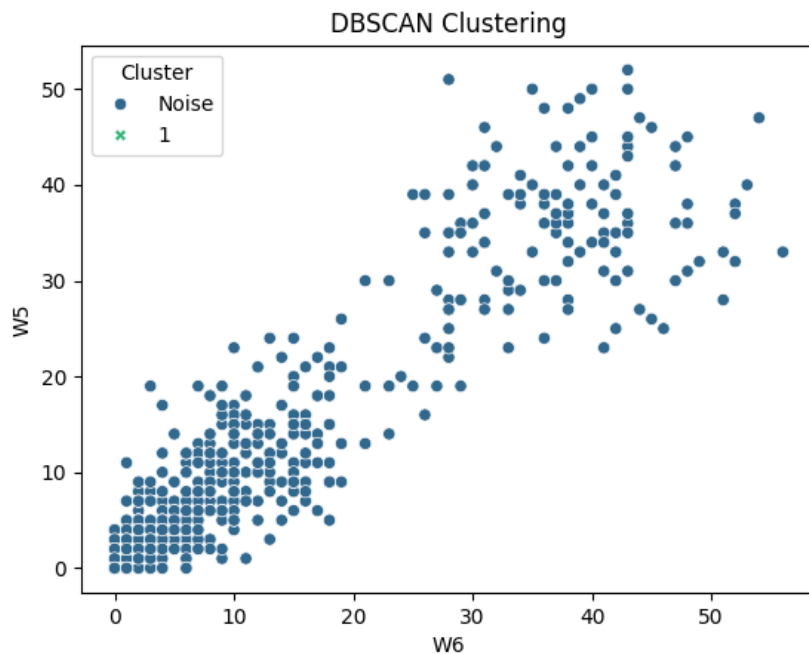
# Przygotuj etykiety klastrów w formie kategorii całkowitych
unique_labels = set(dbscan.labels_)
print(unique_labels)
cluster_labels = {}
for label in unique_labels:
    if label == -1:
        cluster_labels[label] = 'Noise'
    else:
        cluster_labels[label] = len(cluster_labels) + 1

# Dodaj kategorie do DataFrame
features['Cluster'] = [cluster_labels[label] for label in dbscan.labels_]

# Wykres z oznaczeniem klastrów
sns.scatterplot(x=features[item1], y=features[item2],
                hue=features['Cluster'], style=features['Cluster'], palette='viridis')
plt.title('DBSCAN Clustering')

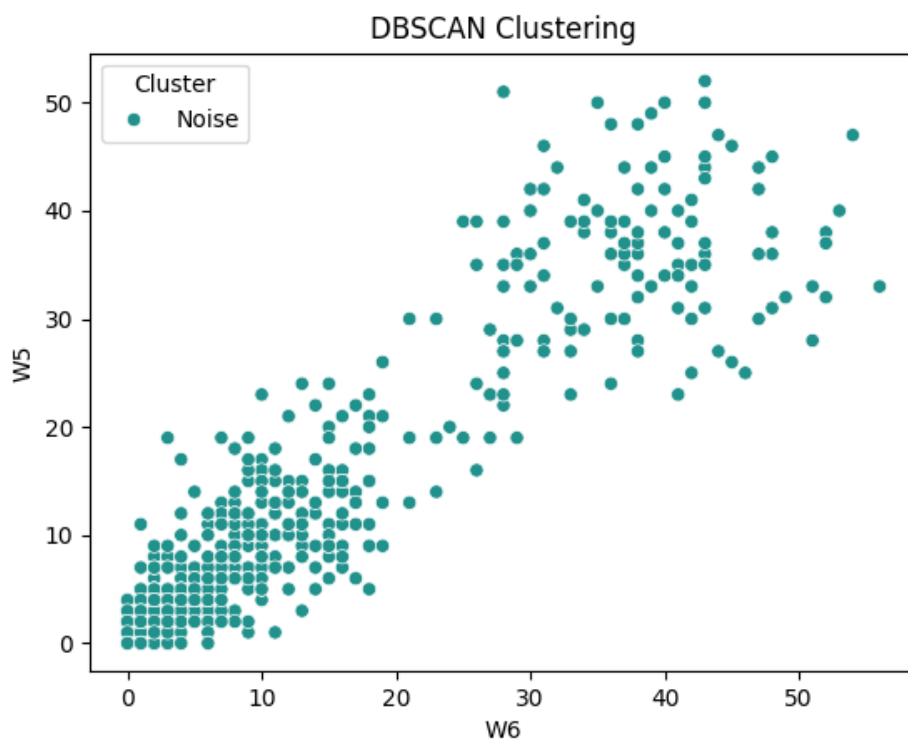
plt.show()
```

Wykres standaryzowanych danych.
Z wykresu wynika, że wszystkie punkty to “szum”.
Zakwalifikowanych danych było



Natomiast zmiana parametrów przyniosła tylko negatywne skutki.
Np dla wartości `min_samples = 5`

```
36  
37 # Wybierz parametry DBSCAN  
38 eps = 0.05 # Promień  
39 min_samples = 5 # Minimalna liczba punktów
```



2.4. Zadanie 4

Korzystając z miary silhouette, a dokładnie średniej miary silhouette dla wszystkich klastrów oblicz i zaprezentuj graficznie optymalne liczby klastrów dla każdego z analizowanych algorytmów w zakresie od 2 do 15 klastrów.

Poniższy kod realizuje wczytanie danych z pliku, inicjalizację podstawowych zmiennych, oraz generowanie danych potrzebnych do utworzenia wykresu silhouette.

```
# Wczytaj dane z pliku CSV do DataFrame
data = pd.read_csv(
    '/home/adam-pc/Documents/mgr/AED-AlgorytmyEksploracjiDanych/lab3/Sales_Transactions_Dataset_Weekly.csv')

# Przetwórz dane, usuń brakujące wartości
data = data.iloc[:, 1:53].dropna()

values = []

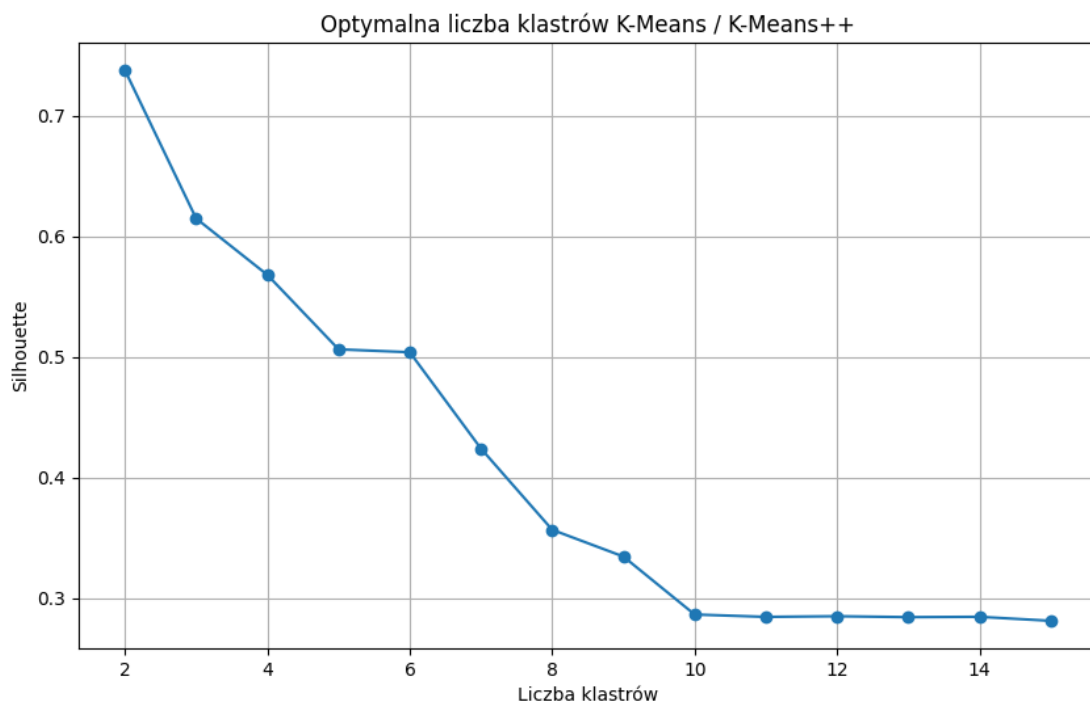
min_clusters = 2
max_clusters = 15
# Pusty słownik do przechowywania wyników średnich miar silhouette
silhouette_scores = {}
# Konwertuj DataFrame do tablicy numpy
data = data.to_numpy()

# Iteruj po liczbie klastrów i oblicz miarę silhouette
for n_clusters in range(min_clusters, max_clusters + 1):
    kmeans = KMeans(n_clusters=n_clusters, random_state=0, n_init='auto')
    cluster_labels = kmeans.fit_predict(data)
    silhouette_avg = silhouette_score(data, cluster_labels)
    silhouette_scores[n_clusters] = silhouette_avg
```

Następnie mając dane mogłem utworzyć wykres

```
# Tworzenie wykresu z wynikami średnich miar silhouette
# Rozdziel dane na oś X i oś Y
x_values = list(silhouette_scores.keys())
y_values = list(silhouette_scores.values())
# Tworzenie wykresu
plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, marker='o', linestyle='-')
plt.xlabel('Liczba klastrów')
plt.ylabel('Silhouette')
plt.title('Optymalna liczba klastrów K-Means / K-Means++')
plt.grid(True)
plt.show()
```

Przedstawienie wykresu który otrzymałem



W kolejnym przeszedłem do realizacji wykresu silhouette dla metody KMedoids, co ukazuje poniższy kod:

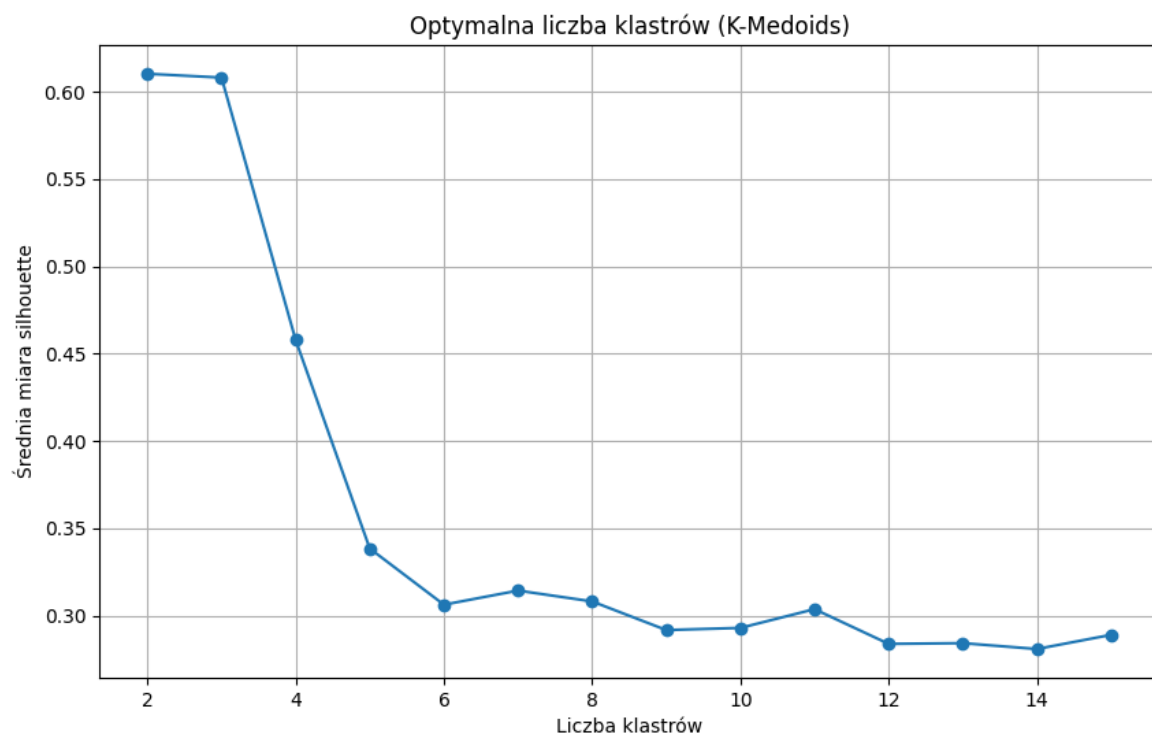
```
# Pusty słownik do przechowywania wyników średnich miar silhouette
silhouette_scores = {}

# Iteruj po liczbie klastrów i oblicz miarę silhouette
for n_clusters in range(min_clusters, max_clusters + 1):
    kmedoids = KMedoids(n_clusters=n_clusters, random_state=0)
    cluster_labels = kmedoids.fit_predict(data)
    silhouette_avg = silhouette_score(data, cluster_labels)
    silhouette_scores[n_clusters] = silhouette_avg
```

Dane zostały zgromadzone, mogą je następnie zaprezentować na wykresie.

```
# Tworzenie wykresu z wynikami średnich miar silhouette
plt.figure(figsize=(10, 6))
plt.plot(list(silhouette_scores.keys()), list(
    silhouette_scores.values()), marker='o', linestyle='-')
plt.xlabel('Liczba klastrów')
plt.ylabel('Średnia miara silhouette')
plt.title('Optymalna liczba klastrów (K-Medoids)')
plt.grid(True)
plt.show()
```

Wykres optymalnej ilości klas dla metody K-Medoids



3. Wnioski

K-means (K-średnich) to popularny algorytm w uczeniu maszynowym, wykorzystywany do klasteryzacji danych. Jest to technika uczenia maszynowego bez nadzoru, która grupuje punkty danych w klastry na podstawie ich podobieństwa. Istnieją techniki, takie jak metoda łokcia (elbow method) do wyboru optymalnej liczby klastrów. Algorytm K-means jest prosty w implementacji, ale jego skuteczność może zależeć od charakterystyki danych i wyboru parametrów.

K-means jest stosowany w wielu dziedzinach, takich jak analiza danych, segmentacja klientów, analiza obrazów, analiza tekstu, przetwarzanie sygnałów i wiele innych.

K-means++ to ulepszona wersja algorytmu K-means, która poprawia sposób inicjalizacji początkowych pozycji centroidów klastrów. Standardowa inicjalizacja w algorytmie K-means polega na losowym wyborze początkowych centroidów, co może prowadzić do wyników zależnych od wylosowanego początkowego stanu. K-means++ wprowadza bardziej inteligentny sposób inicjalizacji, który pomaga zbliżyć się do lepszych wyników klastrowania.

K-medoid, który jest modyfikacją algorytmu K-średnich (K-means) i bazuje na koncepcji medoidów.

K-medoid to algorytm klasteryzacji danych, który jest podobny do K-średnich, ale zamiast używać centroidów klastrów, używa medoidów.

Medoid to punkt danych w klastrze, który minimalizuje sumę odległości między nim a pozostałymi punktami w klastrze. W przeciwieństwie do centroidów, medoid nie jest średnią arytmetyczną punktów w klastrze.

Algorytm K-medoid ma na celu znalezienie k medoidów, reprezentujących klastry w danych.

Agglomerative Clustering to algorytm klasteryzacji hierarchicznej wykorzystywany w uczeniu maszynowym do grupowania danych. Jest to algorytm aglomeracyjny, co oznacza, że zaczyna od pojedynczych punktów i stopniowo łączy je w większe klastry.

Algorytm DBSCAN to metoda klasteryzacji danych, która grupuje punkty na podstawie gęstości punktów w przestrzeni.

Wykorzystuje dwa parametry: promień otoczenia (eps) i minimalną liczbę punktów w otoczeniu (min_samples), aby określić klastry i punkty odstające.

Dendrogram ukazuje hierarchiczną strukturę danych lub obserwacji w kontekście skupień. Skupienia są reprezentowane jako gałęzie drzewa, a odległość między nimi jest interpretowana jako miara podobieństwa lub odmienności.

Wykres Silhouette pomaga on w określeniu optymalnej liczby klastrów poprzez mierzenie odległości między punktami wewnątrz klastrów i między klastrami. Wyższe wartości indeksu Silhouette wskazują na lepszą jakość klastrów.