

```
In [ ]: import numpy as np
import skfuzzy as fuzz
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import cm
from segysak.segy import (
    get_segy_texthead,
    segy_header_scan,
    segy_loader,
)
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

Projekt - interpretacja litofacji wspomagana sztuczną inteligencją

Anna Kaniowska - indeks: 407334

Kontekst

Klasyfikacja litofacji jest istotnym etapem w interpretacji sejsmicznej. Dokładna klasyfikacja próbek pod względem podobieństw prowadzi do lepszego zrozumienia obszarów zainteresowania (prospektów eksploracyjnych). Sygnał sejsmiczny odbity od struktur podpowierzchniowych zawiera cenne informacje przestrzenno-czasowe, które mają znaczenie naukowe i komercyjne, co sprawia, że automatyzacja procesu interpretacji jest istotna. W tradycyjnej, manualnej analizie sejsmicznej ten proces polegał na ręcznym przypisywaniu litofacji przez wyspecjalizowanych interpreterów, na podstawie odpowiedzi amplitudowych. Jest to proces pracochłonny i obarczony dużym błędem. Dzięki zastosowaniu ML możliwe jest obniżenie niepewności analiz oraz zwiększenie tempa przetwarzania danych dzięki automatyzacji.

Zestaw danych Smeaheia to referencyjny zbiór zawierający informacje dotyczące podziemnych warstw, raporty i geomodele związane z oceną potencjalnych miejsc składowania dwutlenku węgla w regionie Smeaheia na platformie Hordaland u wybrzeży Norwegii. Miejsce to było rozważane w kontekście projektu CCS (Carbon Capture and Storage) Northern Lights. Badania były prowadzone etapami, obejmując studia wstępnej wykonalności oraz bardziej szczegółowe analizy. Ostateczny zestaw danych zawiera sejsmiczne dane 2D i 3D, raporty wiertnicze, dane ciśnienia i temperatury, geomodele oraz modele symulacji.

Szczegółowe informacje na temat zbioru danych tutaj:

<https://co2datashare.org/dataset/smeaheia-dataset>

1. Wczytanie danych

```
In [ ]: # wczytanie nagłówka
```

```
header = get_segy_texthead('data/TNE01_Full')
header
```

Out []: **Text Header**

```
C 1 SEGY OUTPUT FROM Petrel 2019.4 Wednesday, October 21 2020 14:19:53
C 2 Name: TNE01_FULL YRealized" 1 Type: 3D seismic
C 3
C 4 First inline: 362 Last inline: 1540
C 5 First xline: 908 Last xline: 3880
C 6 CRS: ST_ED50_UTM31N_P23031_T1133 YStatoil,2100005"
C 7 X min: 531854.62 max: 563388.48 delta: 31533.86
C 8 Y min: 6731674.15 max: 6771202.25 delta: 39528.10
C 9 Time min: -4000.00 max: 4.00 delta: 4004.00
C10 Lat min: 60.42'51.1341"N max: 61.04'22.1371"N delta: 0.21'31.0031"
C11 Long min: 3.35'1.5768"E max: 4.10'28.3122"E delta: 0.35'26.7354"
C12 Trace min: -3998.00 max: 2.00 delta: 4000.00
C13 Seismic (template) min: -304.74 max: 302.36 delta: 607.11
C14 Amplitude (data) min: -304.74 max: 302.36 delta: 607.11
C15 Trace sample format: IEEE floating point
C16 Coordinate scale factor: 10.00000
C17
C18 Binary header locations:
C19 Sample interval : bytes 17-18
C20 Number of samples per trace : bytes 21-22
C21 Trace date format : bytes 25-26
C22
C23 Trace header locations:
C24 Inline number : bytes 5-8
C25 Xline number : bytes 21-24
C26 Coordinate scale factor : bytes 71-72
C27 X coordinate : bytes 73-76
C28 Y coordinate : bytes 77-80
C29 Trace start time/depth : bytes 109-110
C30 Number of samples per trace : bytes 115-116
C31 Sample interval : bytes 117-118
C32
C33
C34
C35
C36
C37
C38
C39
C40 END EBCDIC
```

In []: *# lokalizacja typowych bytów dla inline, xline, cdp, cdp*

```
scan = segy_header_scan('data/TNE01_Full')
scan[scan['std'] > 0]

0%|          | 0.00/1.00k [00:00<?, ? traces/s]
```

Out []:

	byte_loc	count	mean	std	min	
TRACE_SEQUENCE_LINE	1	1000.0	5.005000e+02	288.819436	1.0	
TraceNumber	13	1000.0	5.005000e+02	288.819436	1.0	
CDP	21	1000.0	1.907000e+03	577.638872	908.0	1
SourceX	73	1000.0	5.570523e+06	36537.740701	5507333.0	5538
SourceY	77	1000.0	6.749913e+07	62278.028634	67391426.0	67445
CDP_X	181	1000.0	5.570523e+06	36537.740701	5507333.0	5538
CDP_Y	185	1000.0	6.749913e+07	62278.028634	67391426.0	67445
CROSSLINE_3D	193	1000.0	1.907000e+03	577.638872	908.0	1

Jak widać lokalizacje bytów w pliku są inne niż te, które zeskanowano. Poniżej poprawna wersja wczytywania.

In []:

```
# poprawne wczytanie

V3D = segy_loader("data/TNE01_Full", iline=5, xline=21, cdp_x=73, cdp_y=77, vert_d
V3D

0%|          | 0.00/877k [00:00<?, ? traces/s]
Loading as 3D
Fast direction is TRACE_SEQUENCE_FILE
Converting SEGY: 0%|          | 0.00/877k [00:00<?, ? traces/s]
```

Out []:

xarray.Dataset

► Dimensions:

(iline: 590, xline: 1487, twt: 1001)

▼ Coordinates:

iline	(iline)	uint16	362 364 366 368 ... 1536 1538 ...		
xline	(xline)	uint16	908 910 912 914 ... 3876 3878 ...		
twt	(twt)	float64	-2.0 2.0 ... 3.994e+03 3.998e+03		
cdp_x	(iline, xline)	float32	5.634e+05 5.634e+05 ... 5.319...		
cdp_y	(iline, xline)	float32	6.739e+06 6.739e+06 ... 6.764...		

▼ Data variables:

data	(iline, xline, twt)	float32	0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0		
------	---------------------	---------	-------------------------------------	--	--

► Indexes: (3)

► Attributes: (13)

In []:

```
# pobranie srodkowych inline, xline oraz time slice

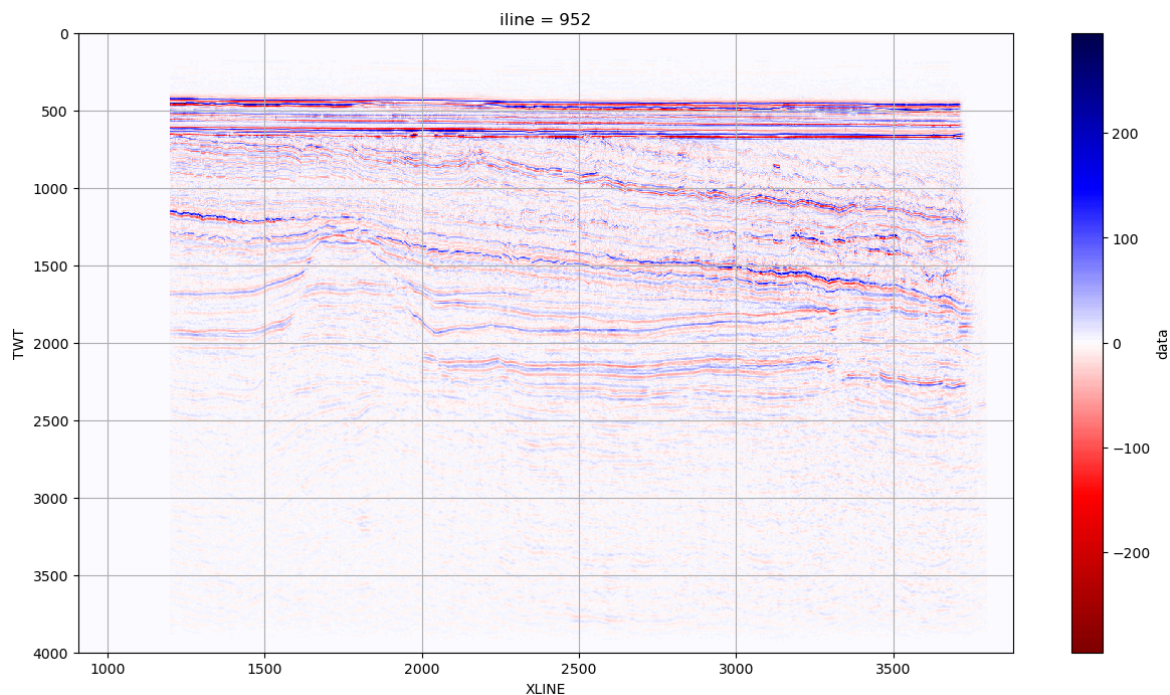
iline_sel = V3D.data['iline'].mean().values
xline_sel = V3D.data['xline'].mean().values
twt_sel = V3D.data['twt'].mean().values

print("iline: ", iline_sel, ", xline: ", xline_sel, ", twt: ", twt_sel)
```

iline: 951.0 , xline: 2394.0 , twt: 1998.0

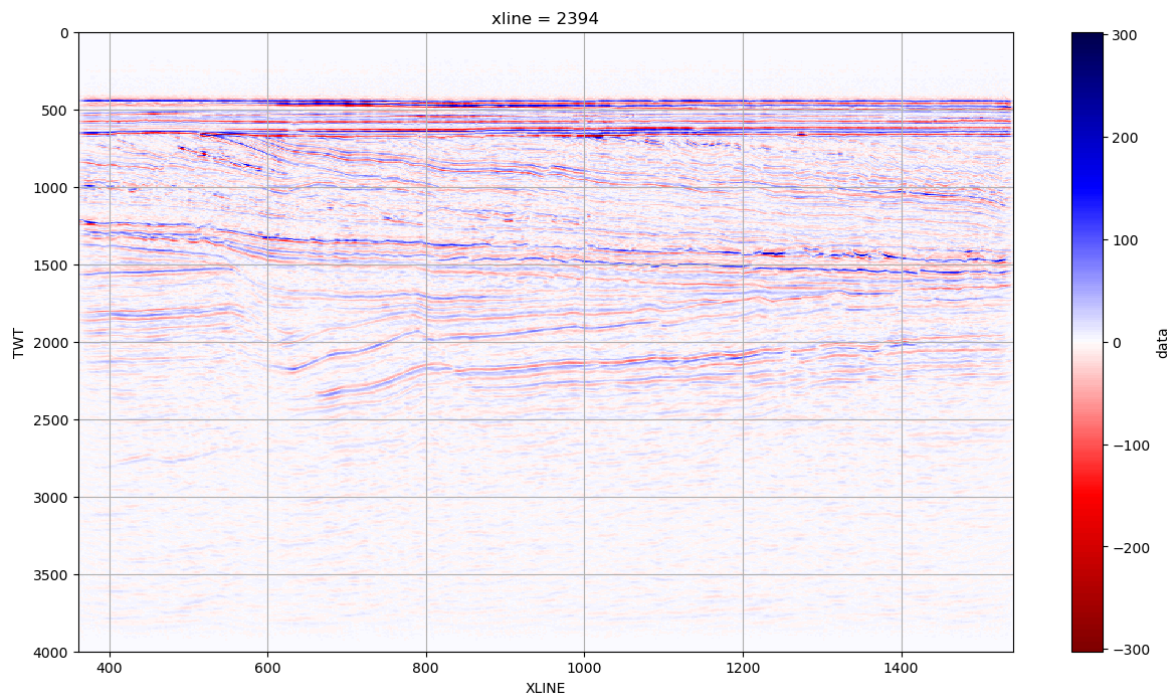
In []: *# wizualizacja środkowego inline*

```
fig, ax1 = plt.subplots(ncols=1, figsize=(15, 8))
V3D.data.transpose("twt", "iline", "xline", transpose_coords=True).sel(iline=ili
plt.grid("grey")
plt.ylabel("TWT")
plt.xlabel("XLINE")
plt.show()
```



In []: *# wizualizacja środkowego xline*

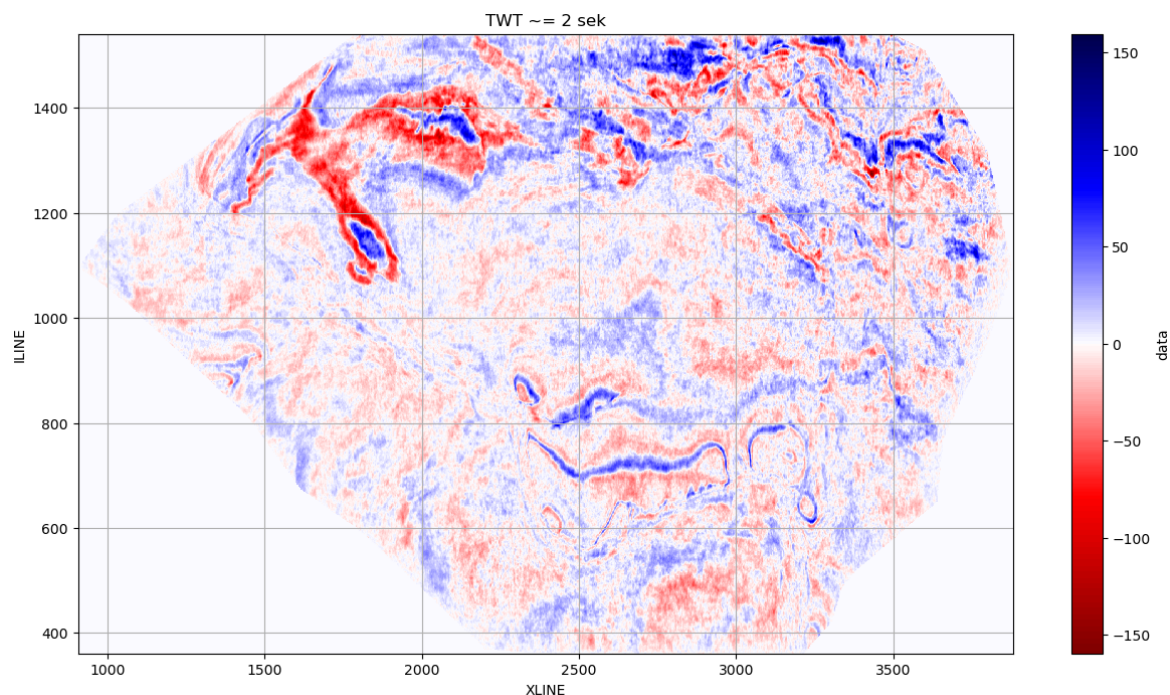
```
fig, ax1 = plt.subplots(ncols=1, figsize=(15, 8))
V3D.data.transpose("twt", "iline", "xline", transpose_coords=True).sel(xline=xli
plt.grid("grey")
plt.ylabel("TWT")
plt.xlabel("XLINE")
plt.show()
```



In []: *# wizualizacja środkowego twt*

```
data_at_twt = V3D.data.sel(twt=twt_sel, method="nearest")

fig, ax1 = plt.subplots(ncols=1, figsize=(15, 8))
data_at_twt.transpose("iline", "xline").plot(ax=ax1, cmap="seismic_r")
plt.grid("grey")
plt.ylabel("ILINE")
plt.xlabel("XLINE")
plt.title("TWT ~= 2 sek")
plt.show()
```



In []: *# ograniczenie cube'a*

```
iline_range = (500, 1400)
xline_range = (2000, 3000)
twt_range = (1900, 2100)
```











```
cube = V3D.sel(
    iline=slice(iline_range[0], iline_range[1]),
    xline=slice(xline_range[0], xline_range[1]),
    twt=slice(twt_range[0], twt_range[1]),
)

cube
```



Out[]: xarray.Dataset

► Dimensions: (iline: 451, xline: 501, twt: 50)

▼ Coordinates:

iline	(iline)	uint16	500 502 504 506 ... 1396 1398 ...	 
xline	(xline)	uint16	2000 2002 2004 ... 2996 2998 ...	 
twt	(twt)	float64	1.902e+03 1.906e+03 ... 2.098...	 
cdp_x	(iline, xline)	float32	5.55e+05 5.55e+05 ... 5.389e+...	 
cdp_y	(iline, xline)	float32	6.75e+06 6.75e+06 ... 6.755e+...	 

▼ Data variables:

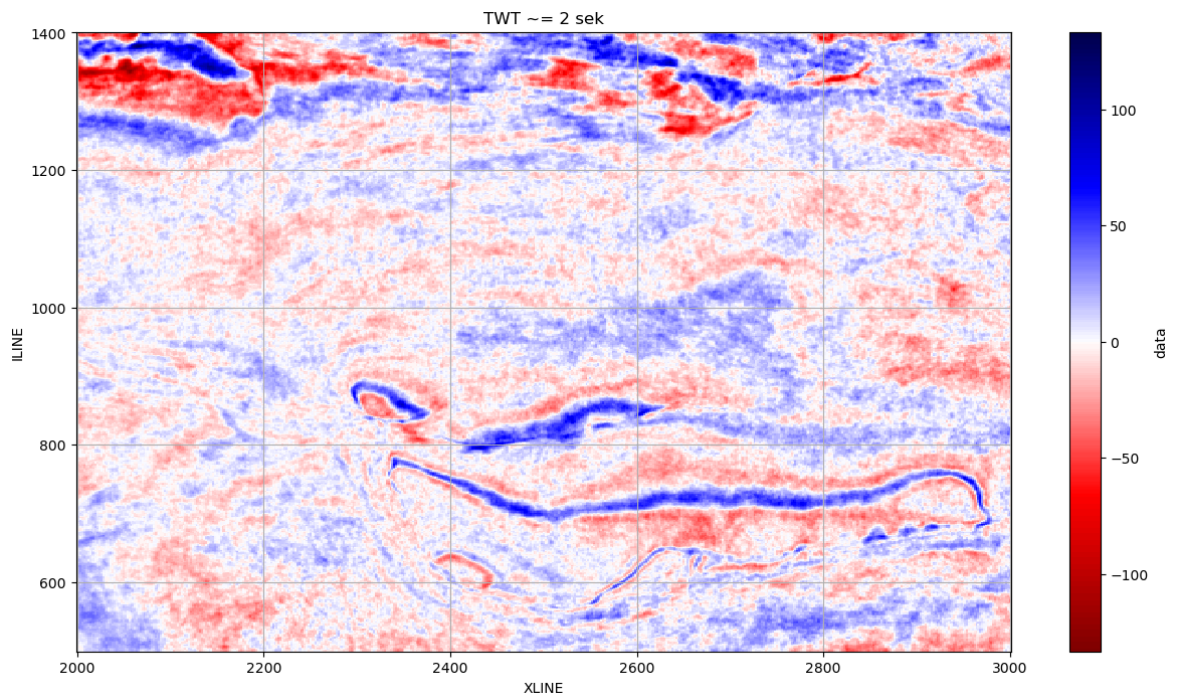
data	(iline, xline, twt)	float32	-4.762 -2.381 0.0 ... 4.762 -14.28	 
-------------	---------------------	---------	------------------------------------	---

► Indexes: (3)

► Attributes: (13)

```
In [ ]: data_at_twt = cube.data.sel(twt=twt_sel, method="nearest")

fig, ax1 = plt.subplots(ncols=1, figsize=(15, 8))
data_at_twt.transpose("iline", "xline").plot(ax=ax1, cmap="seismic_r")
plt.grid("grey")
plt.ylabel("ILINE")
plt.xlabel("XLINE")
plt.title("TWT ~= 2 sek")
plt.show()
```

2. Konwersja xarray do numpy

```
In [ ]: np_cube = cube["data"].values  
  
print(np_cube.shape)  
np_cube
```

(451, 501, 50)

```

Out[ ]: array([[[ -4.7616096, -2.3808048,  0.          , ..., -4.7616096,
                  -7.1424146, -2.3808048],
                 [ -4.7616096, -4.7616096, -2.3808048, ..., -7.1424146,
                  -4.7616096, -4.7616096],
                 [ -7.1424146, -4.7616096, -4.7616096, ..., -9.523219 ,
                  -4.7616096,  0.          ],
                 ...,
                 [  4.7616096, -2.3808048, -4.7616096, ..., -7.1424146,
                  -2.3808048,  0.          ],
                 [  7.1424146,  4.7616096,  0.          , ..., -11.904024 ,
                  -2.3808048,  4.7616096],
                 [  4.7616096,  0.          , -4.7616096, ..., -7.1424146,
                  0.          ,  2.3808048]],

                [[  0.          ,  0.          ,  0.          , ..., -4.7616096,
                  -7.1424146, -4.7616096],
                 [ -2.3808048,  0.          ,  2.3808048, ..., -4.7616096,
                  -2.3808048, -4.7616096],
                 [ -4.7616096, -2.3808048, -2.3808048, ..., -9.523219 ,
                  -4.7616096,  0.          ],
                 ...,
                 [  7.1424146, -2.3808048, -7.1424146, ..., -2.3808048,
                  -4.7616096, -4.7616096],
                 [  4.7616096,  4.7616096,  2.3808048, ..., -4.7616096,
                  -4.7616096, -4.7616096],
                 [  4.7616096,  2.3808048, -2.3808048, ..., -7.1424146,
                  -7.1424146, -4.7616096]],

                [[  2.3808048,  4.7616096,  4.7616096, ..., -2.3808048,
                  -4.7616096, -4.7616096],
                 [  0.          ,  7.1424146,  9.523219 , ...,  0.          ,
                  0.          , -4.7616096],
                 [ -7.1424146,  0.          ,  4.7616096, ..., -7.1424146,
                  -2.3808048,  0.          ],
                 ...,
                 [  4.7616096, -4.7616096, -9.523219 , ..., -7.1424146,
                  -9.523219 , -7.1424146],
                 [  2.3808048,  2.3808048,  0.          , ..., -9.523219 ,
                  -7.1424146, -2.3808048],
                 [  4.7616096,  2.3808048, -2.3808048, ..., -14.284829 ,
                  -9.523219 , -2.3808048]],

                ...,

                [[  0.          ,  0.          , -2.3808048, ..., -4.7616096,
                  -2.3808048,  7.1424146],
                 [ -2.3808048,  0.          ,  2.3808048, ..., -4.7616096,
                  0.          , 14.284829 ],
                 [  4.7616096,  0.          ,  0.          , ..., -4.7616096,
                  2.3808048, 11.904024 ],
                 ...,
                 [  4.7616096, 16.665634 , 14.284829 , ..., 57.139317 ,
                  30.950462 , -7.1424146],
                 [  4.7616096, 14.284829 , 14.284829 , ..., 47.616096 ,
                  21.427242 , -7.1424146],
                 [ 14.284829 , 19.046438 ,  9.523219 , ..., 42.854485 ,
                  14.284829 , -7.1424146]],

                [[  2.3808048,  2.3808048,  2.3808048, ..., -7.1424146,
                  -7.1424146,  4.7616096],

```



```
[ 4.7616096, 7.1424146, 9.523219 , ..., -9.523219 ,
 -2.3808048, 11.904024 ],
[ 11.904024 , 9.523219 , 7.1424146, ..., -4.7616096,
 4.7616096, 16.665634 ],
...,
[ 11.904024 , 21.427242 , 14.284829 , ..., 59.52012 ,
 35.71207 , 2.3808048],
[ 2.3808048, 9.523219 , 14.284829 , ..., 49.996902 ,
 21.427242 , -4.7616096],
[ 7.1424146, 14.284829 , 11.904024 , ..., 42.854485 ,
 19.046438 , -7.1424146]],

[[ 4.7616096, 0. , 2.3808048, ..., -7.1424146,
 -7.1424146, 0. ],
[ 11.904024 , 11.904024 , 9.523219 , ..., -4.7616096,
 -4.7616096, 4.7616096],
[ 11.904024 , 11.904024 , 7.1424146, ..., 0. ,
 2.3808048, 9.523219 ],
...,
[ 19.046438 , 19.046438 , 4.7616096, ..., 61.900925 ,
 33.33127 , 4.7616096],
[ 11.904024 , 11.904024 , 4.7616096, ..., 45.23529 ,
 23.808048 , -2.3808048],
[ 7.1424146, 11.904024 , 7.1424146, ..., 35.71207 ,
 4.7616096, -14.284829 ]]], dtype=float32)
```

3. Klasteryzacja - k-means oraz fuzzy c-means

Różnice między k-means a fuzzy c-means

K-means:

- K-means to algorytm klasteryzacji, który przypisuje każdy punkt danych do jednego konkretnego klastra.
- Algorytm minimalizuje sumę kwadratów odległości punktów danych od środka klastra.
- Każdy punkt danych należy jednoznacznie do jednego klastra.
- Prosty i szybki, ale może nie radzić sobie dobrze z danymi, które mają skomplikowane struktury klastrowe.

Fuzzy C-means (FCM):

- FCM to bardziej zaawansowany algorytm, który pozwala na przypisanie punktów danych do wielu klastrów z różnymi stopniami przynależności (fuzzy membership).
- Minimalizuje ważoną sumę kwadratów odległości punktów danych od środków klastrów, gdzie wagi są stopniami przynależności.
- Każdy punkt danych ma pewną przynależność do każdego klastra, wyrażoną jako liczba w przedziale [0, 1].
- Może lepiej radzić sobie z danymi, które mają rozmyte granice między klastrami, ale jest bardziej złożony obliczeniowo.

```
In [ ]: palette = cm.get_cmap("viridis", 3)
```

C:\Users\Lenovo\AppData\Local\Temp\ipykernel_10148\1154367784.py:1: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.

```
palette = cm.get_cmap("viridis", 3)
```

```
In [ ]: # klasteryzacja 2D

slices = np.linspace(0, 450, 5, dtype=int)

fig, axes = plt.subplots(5, 3, figsize=(20, 15))

for i, sl in enumerate(slices):
    slice_data = np_cube[sl, :, :]
    data_2d = slice_data.reshape(-1, 1)

    kmeans = KMeans(n_clusters=3, random_state=0, n_init="auto").fit(data_2d)
    kmeans_labels = kmeans.labels_.reshape(slice_data.shape)

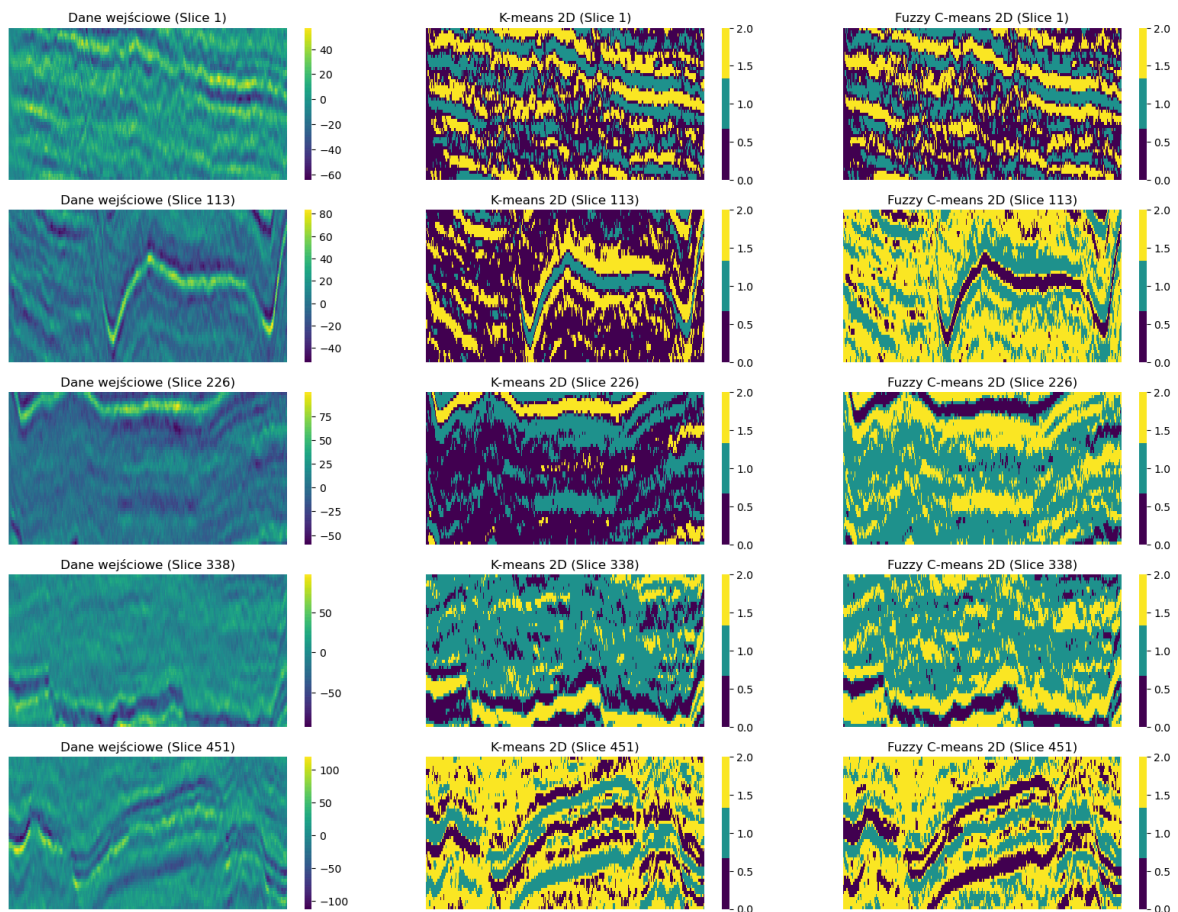
    cntr, u, _, _, _, _ = fuzz.cluster.cmeans(data_2d.T, c=3, m=2, error=0.001)
    fcm_labels = np.argmax(u, axis=0).reshape(slice_data.shape)

    sns.heatmap(slice_data.T, ax=axes[i, 0], cmap='viridis', cbar=True)
    axes[i, 0].set(title=f'Dane wejściowe (Slice {sl+1})', xlabel="ILINE", ylabel="T")
    axes[i, 0].axis('off')

    sns.heatmap(kmeans_labels.T, ax=axes[i, 1], cmap=palette, cbar=True)
    axes[i, 1].set(title=f'K-means 2D (Slice {sl+1})', xlabel="ILINE", ylabel="T")
    axes[i, 1].axis('off')

    sns.heatmap(fcm_labels.T, ax=axes[i, 2], cmap=palette, cbar=True)
    axes[i, 2].set(title=f'Fuzzy C-means 2D (Slice {sl+1})', xlabel="ILINE", ylabel="T")
    axes[i, 2].axis('off')

fig.suptitle(f"Porównanie klasteryzacji 2D metodą K-Means oraz Fuzzy C-Means dla")
plt.show()
```



Powyższy kod realizuje klasteryzację dla 5 slice'ów (1, 113, 226, 338, 451). Główną zaletą klasteryzacji 2D jest wyraźniejsze rozdzielanie różnych warstw danych. Powstałe klastry zachowują ogólny kształt i strukturę danych wejściowych, ale znacznie lepiej uwidaczniają różnice między nimi. Zarówno klasteryzacja metodą K-Means, jak i Fuzzy C-Means dają podobne rezultaty, jednak w zależności od wybranego slice'u, czasami to wyniki K-Means, a czasami Fuzzy C-Means lepiej i bardziej płynnie przedstawiają różne warstwy. Nawet bez specjalistycznej wiedzy można zauważyć, że taka klasyfikacja ma sens, a jej wyniki mogą skutecznie uwidocznić odrębne wzorce.

```
In [ ]: # klasteryzacja 3D

data_3d = np_cube.reshape(-1, 1)

kmeans_3d = KMeans(n_clusters=3, random_state=0, n_init="auto").fit(data_3d)
kmeans_labels_3d = kmeans_3d.labels_.reshape(np_cube.shape)

cntr_3d, u_3d, _, _, _, _ = fuzz.cluster.cmeans(data_3d.T, c=3, m=2, error=0.
fcm_labels_3d = np.argmax(u_3d, axis=0).reshape(np_cube.shape)
```

```
In [ ]: fig, axes = plt.subplots(5, 3, figsize=(20, 25))

for i, sl in enumerate(slices):
    slice_data = np_cube[sl, :, :]
    kmeans_slice = kmeans_labels_3d[sl, :, :]
    fcm_slice = fcm_labels_3d[sl, :, :]
```

```

sns.heatmap(slice_data.T, ax=axes[i, 0], cmap='viridis', cbar=True)
axes[i, 0].set(title=f'Dane wejściowe (Slice {sl+1})', xlabel="ILINE", ylabel="X")
axes[i, 0].axis('off')

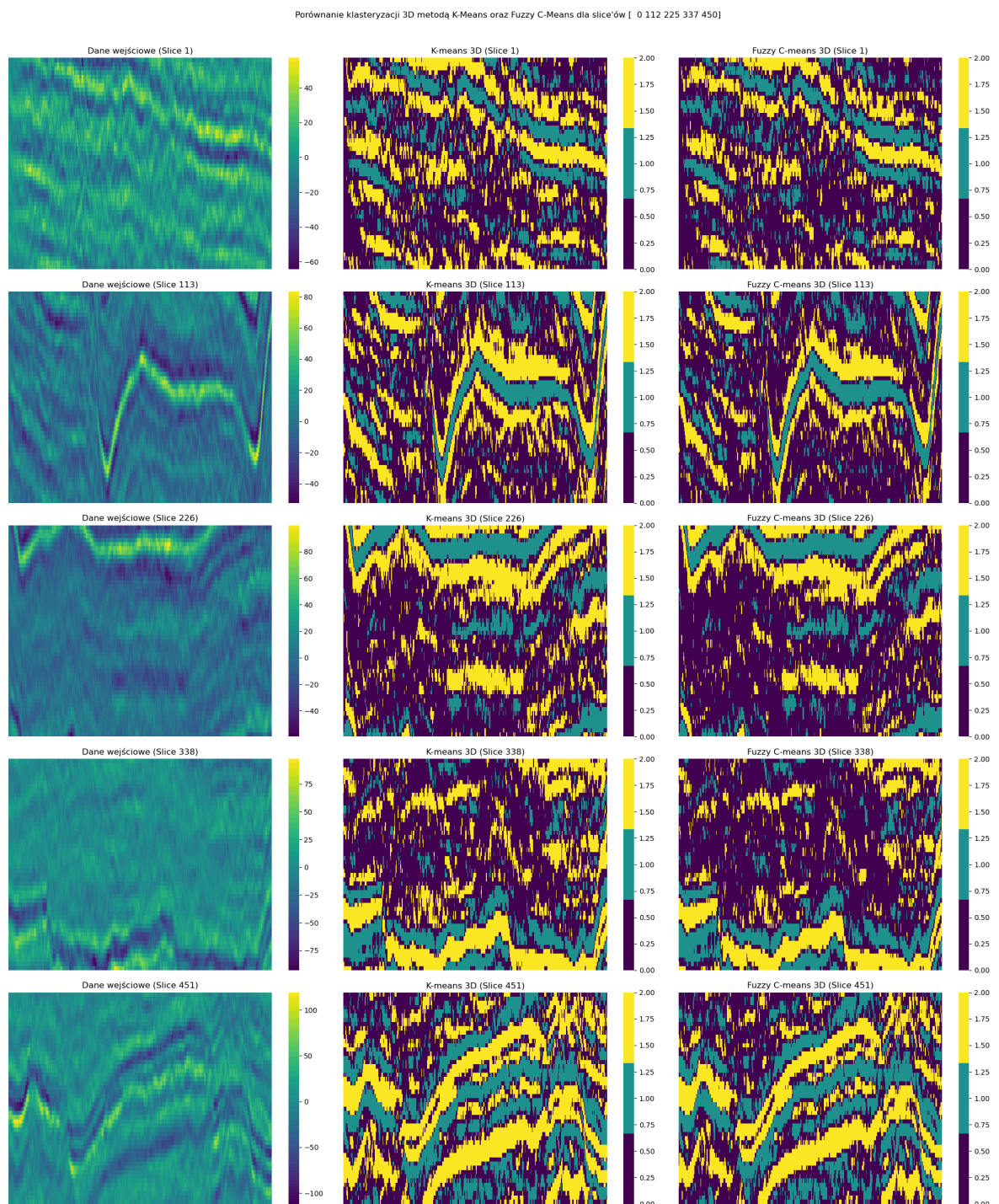
sns.heatmap(kmeans_slice.T, ax=axes[i, 1], cmap=palette, cbar=True)
axes[i, 1].set(title=f'K-means 3D (Slice {sl+1})', xlabel="ILINE", ylabel="X")
axes[i, 1].axis('off')

sns.heatmap(fcm_slice.T, ax=axes[i, 2], cmap=palette, cbar=True)
axes[i, 2].set(title=f'Fuzzy C-means 3D (Slice {sl+1})', xlabel="ILINE", ylabel="X")
axes[i, 2].axis('off')

fig.suptitle(f"Porównanie klasteryzacji 3D metodą K-Means oraz Fuzzy C-Means dla")

plt.tight_layout(rect=[0, 0.03, 1, 0.97])
plt.show()

```



W odróżnieniu od klasteryzacji przeprowadzanej na slice'ach, klasteryzacja w trójwymiarowym kontekście pozwala wyraźniej zauważyć różnice między różnymi algorytmami. Metoda Fuzzy C-Means prezentuje się bardziej jednolicie, co daje wrażenie większej pewności w przyporządkowaniu danych do poszczególnych klas. To odczucie wynika głównie z obserwacji wyników, a nie z konkretnych pomiarów czy wskaźników.

4. Dodatkowe atrybuty

Ze względu na brak odpowiedniej wiedzy domenowej oraz opcjonalność tej sekcji autorka zdecydowała się nie realizować tej części. Dodatkowo wyniki klasteryzacji są na tyle zadowalające, że inne parametry nie są wcale konieczne potrzebne.

5. Optymalna ilość klastrów

W wymienionym w poleceniu artykule w rozdziale 3.5 omówiono trzy różne metody wyboru optymalnej liczby klastrów: metoda Silhouette, indeks Daviesa-Bouldina oraz indeks Calinski-Harabasz. Zaznaczono, że wybór odpowiedniej liczby klastrów nie jest prostym zadaniem i nie istnieje jednoznaczna odpowiedź. Poniżej przedstawiono jedną, wybraną z artykułu metodę doboru optymalnej ilości klastrów - metodę Silhouette. Dodatkowo, jedną z najprostszych metod oceny liczby klastrów jest metoda "Elbow", która nie została wymieniona w tym artykule. Jest to metoda subiektywna i nie zawsze daje jednoznaczne wyniki, dlatego nie cieszy się tak dużą popularnością, jak inne metody. Niemniej jednak jest to sposób najłatwiejszy i najszybszy w użyciu. Poniżej przedstawiono również tę metodę.

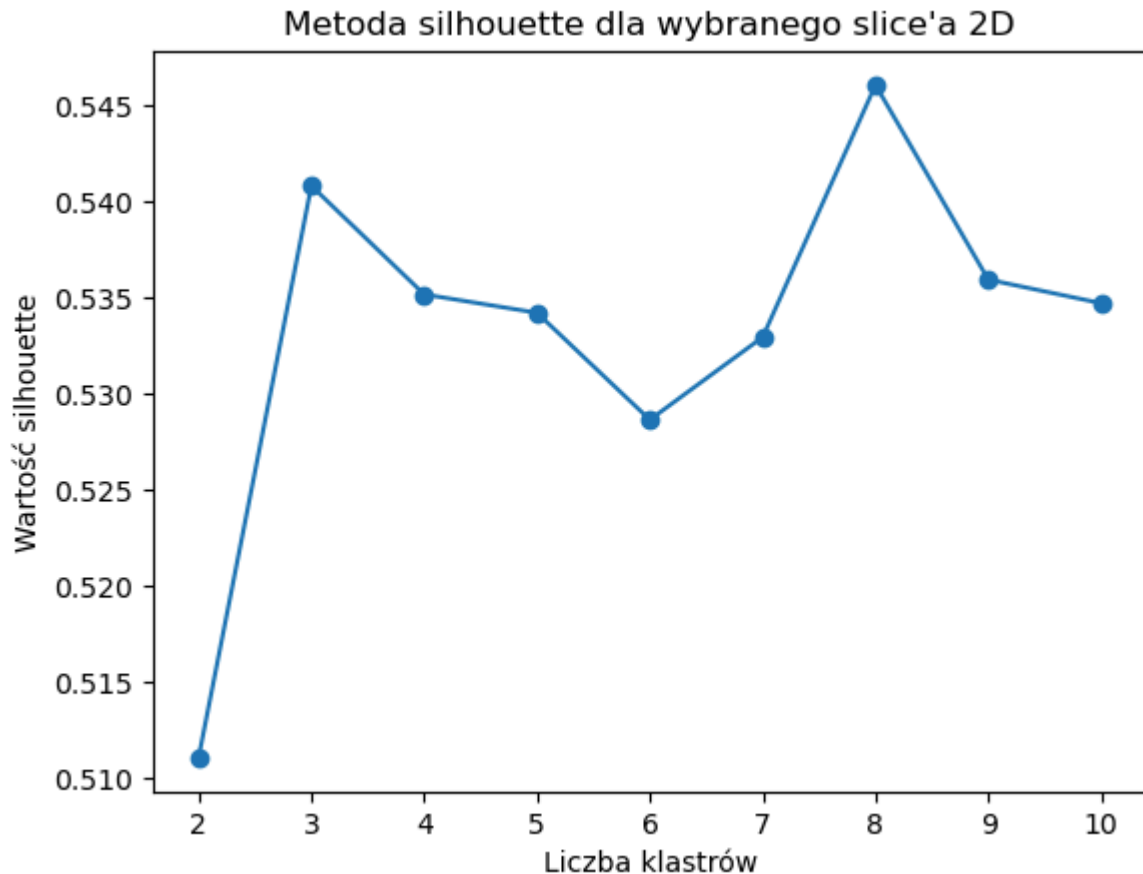
```
In [ ]: # metoda silhouette na danych 2d, na przykładowym slice

slice_data = np_cube[300, :, :]
data_2d = slice_data.reshape(-1, 1)

clusters_range = range(2, 11)
silhouette_scores = []

for n_clusters in clusters_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=0, n_init="auto").fit(data_2d)
    labels = kmeans.labels_
    silhouette_avg = silhouette_score(data_2d, labels)
    silhouette_scores.append(silhouette_avg)

plt.plot(clusters_range, silhouette_scores, marker='o')
plt.xlabel('Liczba klastrów')
plt.ylabel('Wartość silhouette')
plt.title('Metoda silhouette dla wybranego slice\'a 2D')
plt.show()
```



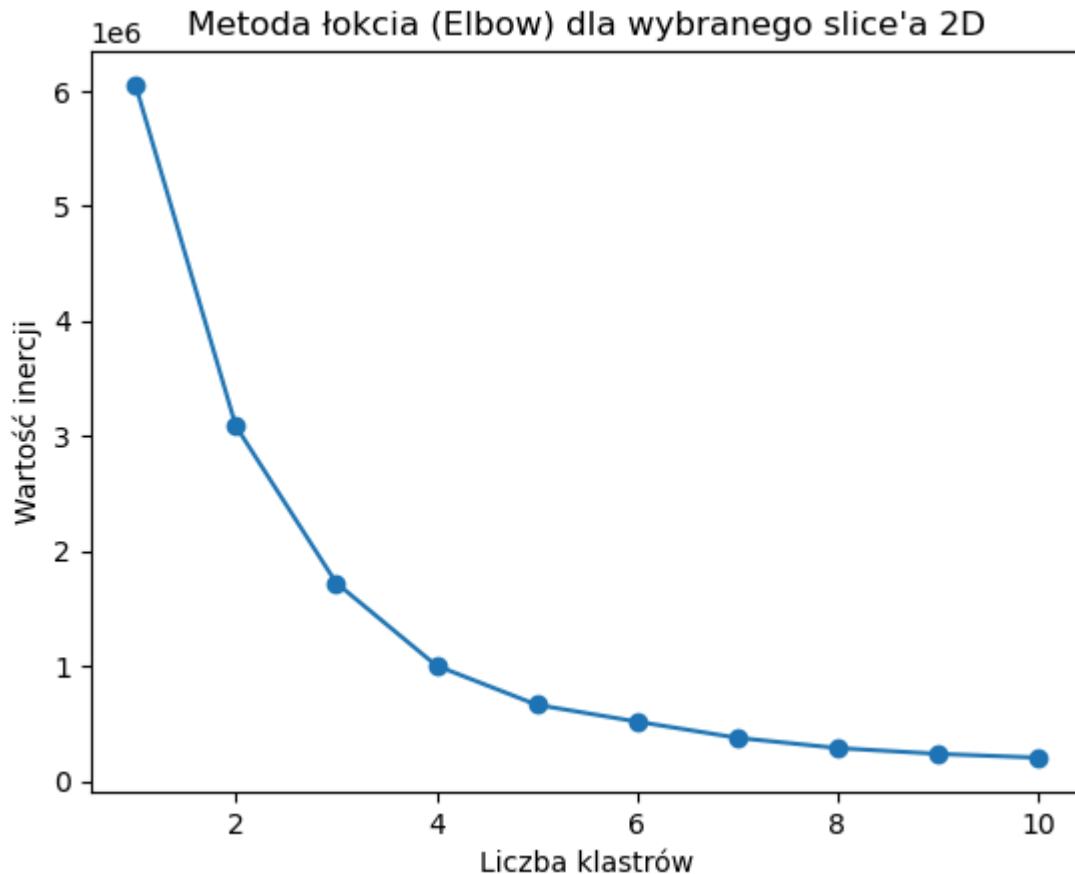
Na wykresie metody silhouette, wartość silhouette jest przedstawiana dla każdej liczby klastrow. Optymalna liczba klastrow będzie tą, dla której wartość silhouette jest największa. Jest to punkt, w którym wykres osiąga szczyt lub najbliższy szczytu, a następnie zaczyna maleć lub stabilizować się. Dla analizowanych danych widać 2 piki - dla 3 skupień oraz 8 skupień. Wybór lepszej opcji zostanie dokonany po sprawdzeniu jakie wyniki da metoda Elbow.

```
In [ ]: # metoda elbow na danych 2d, na przykładowym slice

inertia_values = []

clusters_range = range(1, 11)
for n_clusters in clusters_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=0, n_init="auto").fit(data)
    inertia_values.append(kmeans.inertia_)

plt.plot(clusters_range, inertia_values, marker='o')
plt.xlabel('Liczba klastrow')
plt.ylabel('Wartość inercji')
plt.title('Metoda łokcia (Elbow) dla wybranego slice\'a 2D')
plt.show()
```

Optymalna liczba klastrów w metodzie Elbow to ten punkt, gdzie "łokieć" jest najbardziej widoczny. Innymi słowy, jest to punkt, w którym dalsze zwiększanie liczby klastrów nie powoduje już znaczącego zmniejszenia inercji, a wykres staje się bardziej płaski. Dla analizowanych danych będzie to wartość 3 lub 4. Biorąc pod uwagę wyniki uzyskane metodą Silhouette dla tego typu danych optymalną ilością klastrów będzie 3. W tym momencie autorka podmienia w poprzednich analizach ilość klastrów na 3 (wcześniej sprawdzane opcje wynosiły 4 i 5).

Podsumowanie

W ramach projektu przeprowadzono klasteryzację sejsmicznych danych 3D z regionu Smeaheia, wykorzystując algorytmy K-Means oraz Fuzzy C-Means. Analizy przeprowadzono na wybranych slice'ach (1, 113, 226, 338, 451) oraz na całości danych, aby uzyskać wgląd w strukturę warstw podpowierzchniowych oraz ocenić różnice w wynikach obu metod.

Klasteryzacja 2D

Klasteryzacja na wybranych slice'ach wykazała, że zarówno K-Means, jak i Fuzzy C-Means dobrze zachowują ogólny kształt i strukturę danych wejściowych, uwidaczniając różnice między warstwami. W zależności od slice'a, jeden z algorytmów może lepiej podkreślać pewne cechy:

- K-Means często daje bardziej zdefiniowane granice między klastrami.

- Fuzzy C-Means tworzy bardziej płynne przejścia, co może być korzystne w kontekście geologicznych analiz warstw.

Klasteryzacja 3D

Przeprowadzenie klasteryzacji na całym wolumenie danych pozwoliło na bardziej spójne oznaczanie klastrów w poszczególnych slice'ach, co ułatwia analizę wielu przekrojów jednocześnie. W trójwymiarowej klasteryzacji zauważalne są wyraźniejsze różnice między algorytmami:

- K-Means może lepiej uwidaczniać różnice między bardziej kontrastującymi strukturami.
- Fuzzy C-Means dostarcza bardziej jednorodne klastry, co daje wrażenie większej pewności przypisania danych do poszczególnych klas.

Analiza Optymalnej Liczby Klastrów

Do określenia optymalnej liczby klastrów wykorzystano dwie metody: "Elbow" oraz "Silhouette". Metoda "Elbow" polega na wizualnej identyfikacji punktu, gdzie dalsze zwiększanie liczby klastrów przestaje znacząco redukować sumę kwadratów odległości wewnątrz klastrów. Metoda "Silhouette" ocenia, jak dobrze obiekty pasują do swoich klastrów, mierząc średnią odległość do obiektów w tym samym klastrze w porównaniu do średniej odległości do obiektów w sąsiednich klastrach. Obydwie metody pozwalają na empiryczne podejście do wyboru liczby klastrów, zwiększając pewność i dokładność wyników klasteryzacji. Po przeanalizowaniu wyników obu metod wysunięto wniosek, że optymalna ilość klastrów dla analizowanych danych to 3.

Zastosowanie w Analizach Litofacji

Automatyzacja procesu interpretacji danych sejsmicznych za pomocą algorytmów klasteryzacji jest szczególnie cenna w kontekście klasyfikacji litofacji. Tradycyjnie, ręczne przypisywanie litofacji przez wyspecjalizowanych interpreterów było procesem czasochłonnym i podatnym na błędy. Zastosowanie algorytmów uczenia maszynowego, takich jak K-Means i Fuzzy C-Means, pozwala na szybsze i bardziej precyzyjne analizy, co jest kluczowe w projektach takich jak CCS (Carbon Capture and Storage).