Sprawozdanie

Z projektu - interpolacja litofacji wspomagana sztuczną inteligencją

Autor: Ewa Szewczyk, nr indeksu: 406923

Celem projektu jest użycie metod sztucznej inteligencji do poprawnej klasyfikacji litofacji na podstawie danych 3D.

Ustawienia notatnika

In [ ]:
```
from segysak.segy import (
    get_segy_texthead,
    segy_header_scan,
    segy_loader,
)

import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import cm

pal = cm.get_cmap("viridis", 5)

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import skfuzzy as fuzz
```

```
C:\Users\ewa.szewczyk\AppData\Local\Temp\ipykernel_8652\1277359974.py:13: MatplotlibDeprecationWarning: The get_
cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.co
lormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  pal = cm.get_cmap("viridis", 5)
```

1. Wgranie danych przy użyciu biblioteki SEGYSAK

1.1. Wczytanie nagłówku tekstowego

In [ ]:
```
ebcdic = get_segy_texthead("TNE01_Full")

print(type(ebcdic))
ebcdic
```

```
<class 'segysak._richstr.rich_texthead'>
```

Out[ ]:

Text Header

```
C 1 SEGY OUTPUT FROM Petrel 2019.4 Wednesday, October 21 2020 14:19:53
C 2 Name: TNE01_FULL ÝRealized¨ 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 ÝStatoil,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
```

1.2 Sprawdzenie lokalizacji typowych bytów

In [ ]:
```
scan = segy_header_scan("TNE01_Full")

scan[scan["std"] > 0]
```

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

Out[ ]:

	byte_loc	count	mean	std	min	25%	50%	75%	
TRACE_SEQUENCE_LINE	1	1000.0	5.005000e+02	288.819436	1.0	250.75	500.5	750.25	10
TraceNumber	13	1000.0	5.005000e+02	288.819436	1.0	250.75	500.5	750.25	10
CDP	21	1000.0	1.907000e+03	577.638872	908.0	1407.50	1907.0	2406.50	29
SourceX	73	1000.0	5.570523e+06	36537.740701	5507333.0	5538928.25	5570523.5	5602118.50	56337
SourceY	77	1000.0	6.749913e+07	62278.028634	67391426.0	67445279.25	67499133.0	67552986.00	676068
CDP_X	181	1000.0	5.570523e+06	36537.740701	5507333.0	5538928.25	5570523.5	5602118.50	56337
CDP_Y	185	1000.0	6.749913e+07	62278.028634	67391426.0	67445279.25	67499133.0	67552986.00	676068
CROSSLINE_3D	193	1000.0	1.907000e+03	577.638872	908.0	1407.50	1907.0	2406.50	29

Lokalizacja bytów w używanym pliku (plik EBCDIC) jest inna niż te, które są domyślne (powyżej). Należy je poprawnie ustawić przy wczytywaniu.

1.3 Wczytanie pliku

In [ ]:
```
V3D = segy_loader("TNE01_Full", iline=5, xline=21, cdpx=73, cdpy=77, vert_domain="TWT")

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:

▾ Data variables:

▸ Indexes: (3)

▸ Attributes: (13)

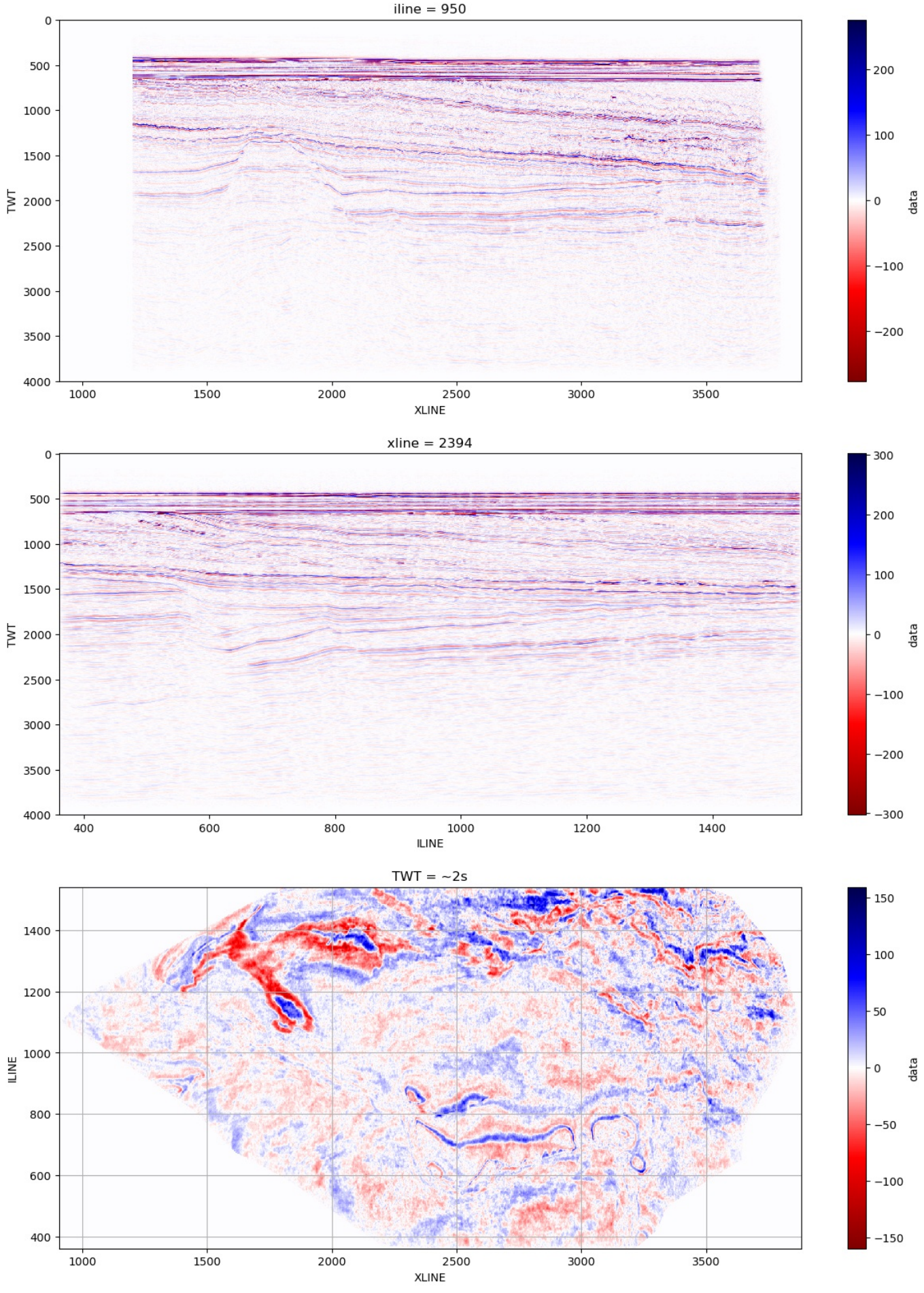
1.4 Wizualizacja środkowego inline, xline oraz time slice

In [ ]:
```
# Sprawdzanie środkowych wartości

print("iline:", V3D.data.iline.median().values)
print("xline:", V3D.data.xline.median().values)
print("twt:", V3D.data.twt.median().values)
```

```
iline: 951.0
xline: 2394.0
twt: 1998.0
```

In [ ]:
```
fig, ax = plt.subplots(ncols=1, nrows=3, figsize=(15, 20))
iline_sel = 950  # dla 951 nie ma
V3D.data.transpose("twt", "iline", "xline", transpose_coords=True).sel(
    iline=iline_sel
).plot(yincrease=False, cmap="seismic_r", ax=ax[0])
plt.grid("gray")
ax[0].set(ylabel="TWT", xlabel="XLINE")

xline_sel = 2394
V3D.data.transpose("twt", "iline", "xline", transpose_coords=True).sel(
    xline=xline_sel
).plot(yincrease=False, cmap="seismic_r", ax=ax[1])
plt.grid("gray")
ax[1].set(ylabel="TWT", xlabel="ILINE")

twt_value = 1998
data_at_twt = V3D.data.sel(twt=twt_value, method="nearest")
```

```
data_at_twt.transpose("iline", "xline").plot(ax=ax[2], cmap="seismic_r")
plt.grid("gray")
ax[2].set(ylabel="ILINE", xlabel="XLINE", title="TWT = ~2s")

plt.show()
```



1.5 Ograniczenie zakresów

In [ ]:
```
iline_range = (500, 1400)
```

```
xline_range = (2000, 3000)
twt_range = (1900, 2100)

smaller_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]),
)

smaller_cube
```

Out[ ]: xarray.Dataset

▸ Dimensions:

(**iline**: 451, **xline**: 501, **twt**: 50)

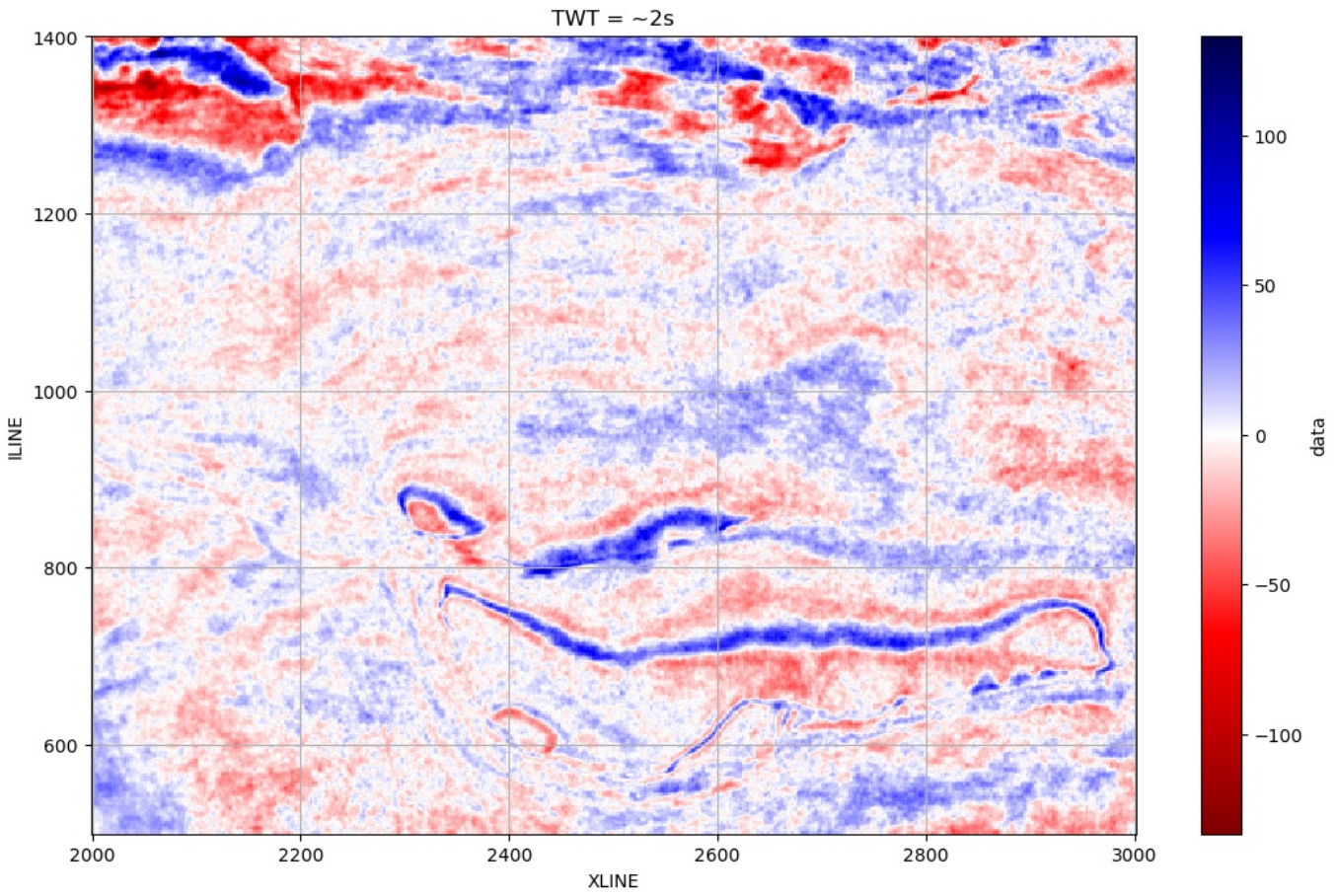
▾ Coordinates:

▾ Data variables:

▸ Indexes: (3)

▸ Attributes: (13)

In [ ]:
```
fig, ax = plt.subplots(1, 1, figsize=(13, 8))

twt_value = 1998
data_at_twt = smaller_cube.data.sel(twt=twt_value, method="nearest")
data_at_twt.transpose("iline", "xline").plot(ax=ax, cmap="seismic_r")
plt.grid("gray")
ax.set(ylabel="ILINE", xlabel="XLINE", title="TWT = ~2s")

plt.show()
```



2. Konwersja XArray do Numpy

In [ ]:
```
np_cube = smaller_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 przy użyciu algorytmu k-means oraz fuzzy c-means

Oba algorytmy k-means oraz fuzzy c-means służą do klasteryzacji, czyli podziału danych na wyróżniające się wspólnymi cechami grupy, ale różnią się w sposobie przypisywania punktów do klastrów. Zasadniczą różnicą jest wykorzystanie logiki klasycznej i teorii zbiorów w przypadku algorytmu k-means (obiekt należy lub nienależy do zbioru) oraz logiki trójwartościowej i teorii zbiorów rozmytych w przypadku fuzzy c-means.

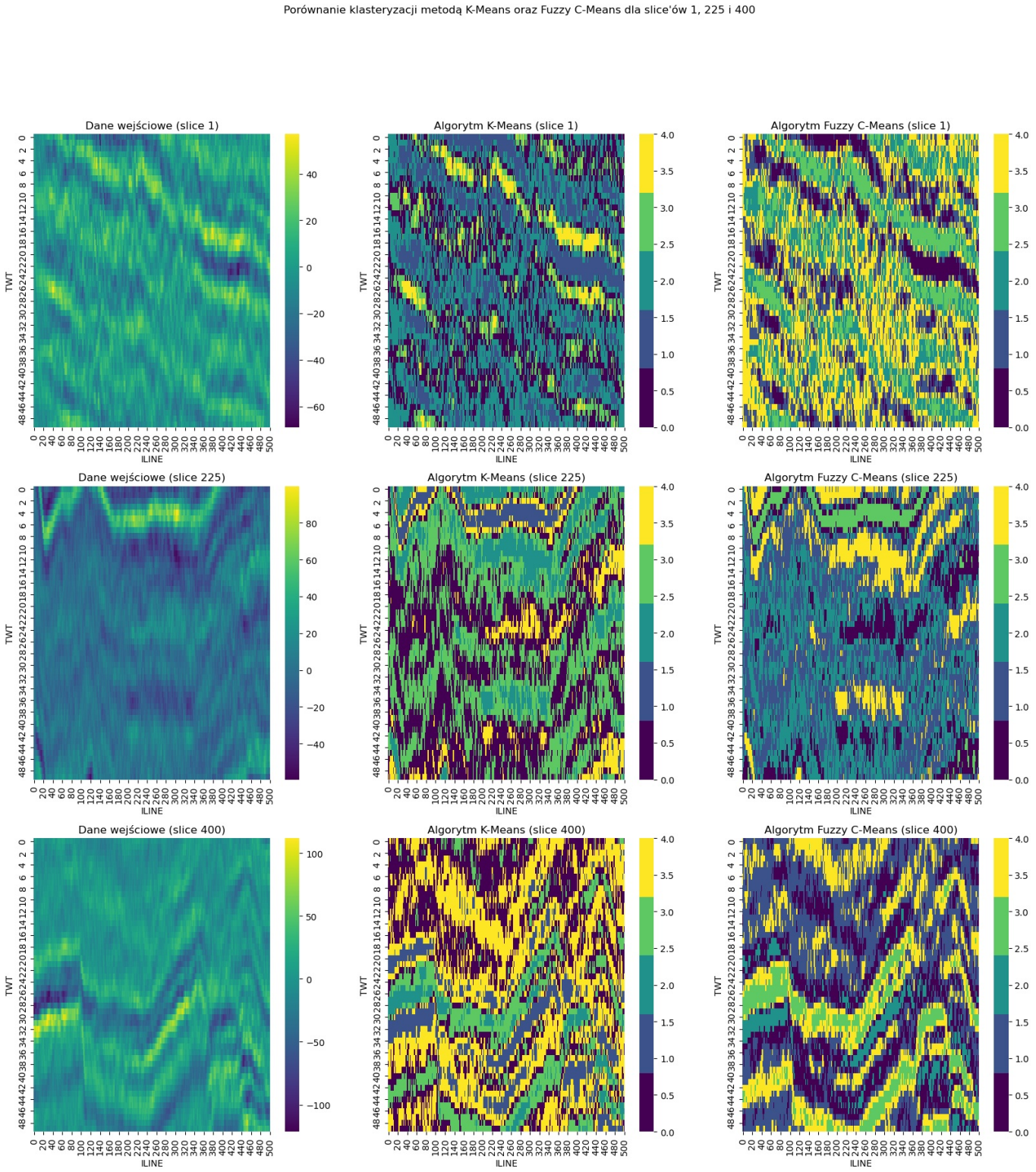
Główne różnice między oboma algorytmami przedstawiono w poniższej tabeli: | Kryterium | K-means | Fuzzy c-means | |--- |--- |--- | | **Typ przypisania klastrów** | Twarde przypisanie (obiekt należy lub nie) | Miękkie przypisanie (obiekt należy w pewnym stopniu) | | **Przynależność do klastra** | Każdy punkt należy do jednego klastra | Punkt może należeć do wielu klastrów | | **Centroidy klastrów** | Centroidy to średnie punktów (wartości ich cech) w klastrze | Centroidy to ważone średnie punktów w klastrze | | **Elastyczność** | Mniej elastyczny, trudniejszy w przypadkach nakładających się klastrów | Bardziej elastyczny, lepiej radzi sobie z nakładającymi się klastrami | | **Szybkość** | Zazwyczaj szybszy | Zazwyczaj wolniejszy |

3.1 Klasteryzacja na slice

Stworzono klasteryzację dla 3 slice'ów - 1, 225 i 400.

In [ ]:
```
# Wyciągnięcie slice'ów
slice_1 = np_cube[1, :, :]
slice_225 = np_cube[225, :, :]
slice_400 = np_cube[400, :, :]


# Reshape
slice_1_reshaped = slice_1.reshape(-1, 1)
slice_225_reshaped = slice_225.reshape(-1, 1)
slice_400_reshaped = slice_400.reshape(-1, 1)

# Kmeans
kmeans = KMeans(n_clusters=5, random_state=44, n_init="auto").fit(slice_1_reshaped)
clustered_slice_1 = kmeans.labels_.reshape(slice_1.shape)

kmeans = KMeans(n_clusters=5, random_state=44, n_init="auto").fit(slice_225_reshaped)
clustered_slice_225 = kmeans.labels_.reshape(slice_225.shape)

kmeans = KMeans(n_clusters=5, random_state=44, n_init="auto").fit(slice_400_reshaped)
clustered_slice_400 = kmeans.labels_.reshape(slice_400.shape)

# Fuzzy
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    slice_1_reshaped.T, c=5, m=2, error=0.005, maxiter=1000, init=None, seed=44
)
clustered_slice_fuzzy_1 = np.argmax(u, axis=0).reshape(slice_1.shape)

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    slice_225_reshaped.T, c=5, m=2, error=0.005, maxiter=1000, init=None, seed=44
)
clustered_slice_fuzzy_225 = np.argmax(u, axis=0).reshape(slice_225.shape)

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    slice_400_reshaped.T, c=5, m=2, error=0.005, maxiter=1000, init=None, seed=44
)
clustered_slice_fuzzy_400 = np.argmax(u, axis=0).reshape(slice_400.shape)

# Wykresy
fig, axes = plt.subplots(3, 3, figsize=(20, 20))
sns.heatmap(slice_1.T, ax=axes[0, 0], cmap="viridis")
sns.heatmap(clustered_slice_1.T, ax=axes[0, 1], cmap=pal)
sns.heatmap(clustered_slice_fuzzy_1.T, ax=axes[0, 2], cmap=pal)

sns.heatmap(slice_225.T, ax=axes[1, 0], cmap="viridis")
sns.heatmap(clustered_slice_225.T, ax=axes[1, 1], cmap=pal)
sns.heatmap(clustered_slice_fuzzy_225.T, ax=axes[1, 2], cmap=pal)

sns.heatmap(slice_400.T, ax=axes[2, 0], cmap="viridis")
sns.heatmap(clustered_slice_400.T, ax=axes[2, 1], cmap=pal)
sns.heatmap(clustered_slice_fuzzy_400.T, ax=axes[2, 2], cmap=pal)

axes[0, 0].set(title="Dane wejściowe (slice 1)", ylabel="TWT", xlabel="ILINE")
axes[0, 1].set(title="Algorytm K-Means (slice 1)", ylabel="TWT", xlabel="ILINE")
axes[0, 2].set(title="Algorytm Fuzzy C-Means (slice 1)", ylabel="TWT", xlabel="ILINE")

axes[1, 0].set(title="Dane wejściowe (slice 225)", ylabel="TWT", xlabel="ILINE")
axes[1, 1].set(title="Algorytm K-Means (slice 225)", ylabel="TWT", xlabel="ILINE")
axes[1, 2].set(title="Algorytm Fuzzy C-Means (slice 225)", ylabel="TWT", xlabel="ILINE")

axes[2, 0].set(title="Dane wejściowe (slice 400)", ylabel="TWT", xlabel="ILINE")
```

```
axes[2, 1].set(title="Algorytm K-Means (slice 400)", ylabel="TWT", xlabel="ILINE")
axes[2, 2].set(title="Algorytm Fuzzy C-Means (slice 400)", ylabel="TWT", xlabel="ILINE")


fig.suptitle(
    "Porównanie klasteryzacji metodą K-Means oraz Fuzzy C-Means dla slice'ów 1, 225 i 400"
)
plt.show()
```



Podstawową zaletą klasteryzacji jest wyraźniejsze rozróżnienie poszczególnych warstw. Wzory które wyłaniają się po klasteryzacji pasują kształtem/wzorcem do danych wejściowych, ale znacznie wyraźniej pokazują różnice między nimi. Wyniki klasteryzacji oboma metodami dają podobne wyniki, przy czym w zależności od slice'u czasami wyniki K-Means, a czasami wyniki Fuzzy C-Means lepiej "gładziej" pokazują różne warstwy. Pomimo braku potrzebnej wiedzy dziedzinowej z całą pewnością można stwierdzić, że taka klasyfikacja ma sens, a jej rezultaty potrafią pokazać odrębne wzorce.

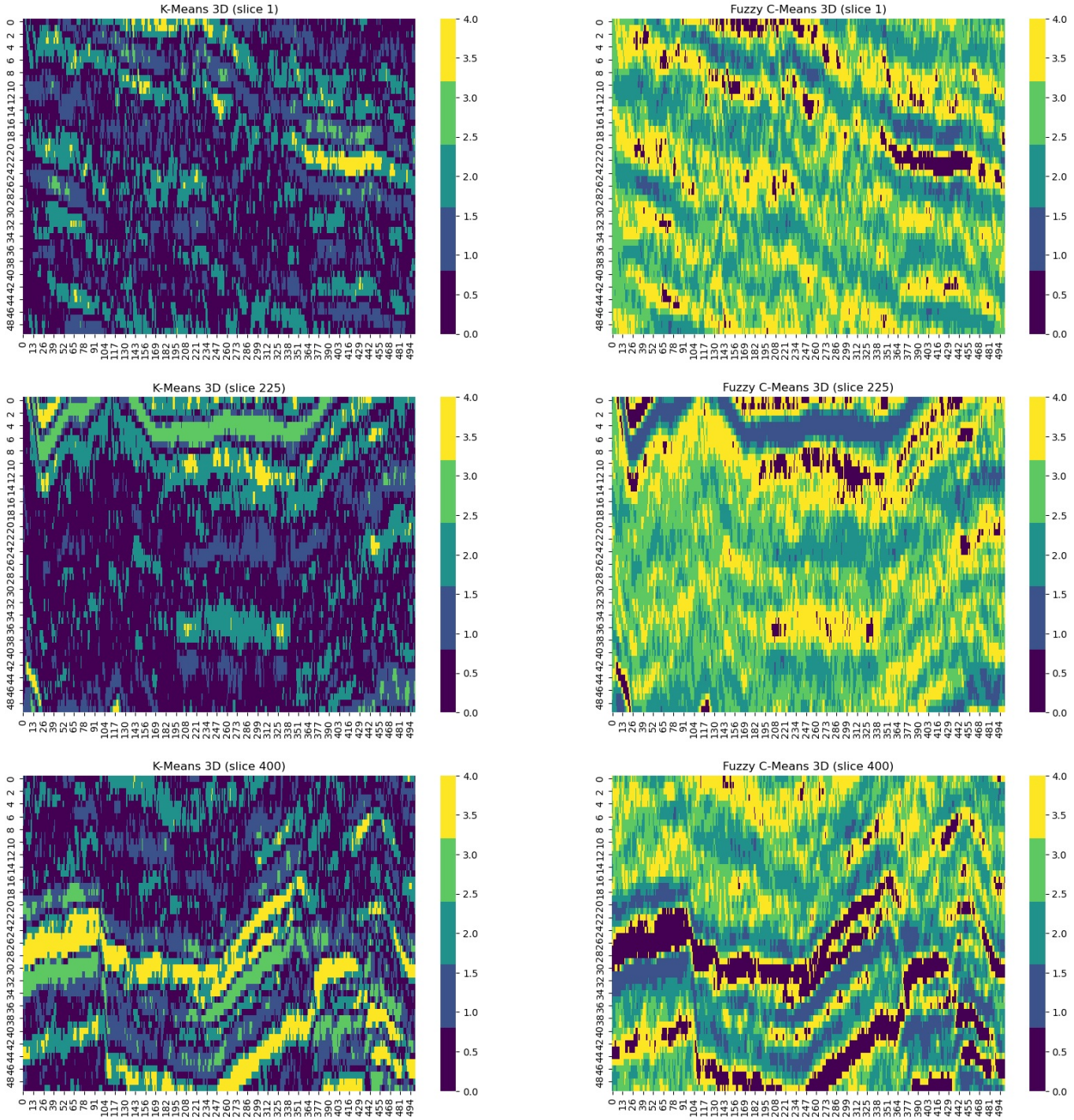
3.2 Klasteryzacja 3D

In [ ]:
```
## K-Means
np_cube_reshaped = np_cube.reshape(-1, 1)
kmeans_all = KMeans(n_clusters=5, random_state=0, n_init="auto").fit(np_cube_reshaped)
clustered_cube = kmeans_all.labels_.reshape(np_cube.shape)
```

```
## FCM
cntr_all, u_all, u0_all, d_all, jm_all, p_all, fpc_all = fuzz.cluster.cmeans(
    np_cube_reshaped.T, c=5, m=2, error=0.005, maxiter=1000, init=None
)
clustered_cube_fuzzy = np.argmax(u_all, axis=0).reshape(np_cube.shape)
```

In [ ]:

```
fig, axes = plt.subplots(3, 2, figsize=(20, 20))

sns.heatmap(clustered_cube[1, :, :].T, ax=axes[0, 0], cmap=pal)
sns.heatmap(clustered_cube_fuzzy[1, :, :].T, ax=axes[0, 1], cmap=pal)

sns.heatmap(clustered_cube[225, :, :].T, ax=axes[1, 0], cmap=pal)
sns.heatmap(clustered_cube_fuzzy[225, :, :].T, ax=axes[1, 1], cmap=pal)

sns.heatmap(clustered_cube[400, :, :].T, ax=axes[2, 0], cmap=pal)
sns.heatmap(clustered_cube_fuzzy[400, :, :].T, ax=axes[2, 1], cmap=pal)

axes[0, 0].set(title="K-Means 3D (slice 1)")
axes[0, 1].set(title="Fuzzy C-Means 3D (slice 1)")

axes[1, 0].set(title="K-Means 3D (slice 225)")
axes[1, 1].set(title="Fuzzy C-Means 3D (slice 225)")

axes[2, 0].set(title="K-Means 3D (slice 400)")
axes[2, 1].set(title="Fuzzy C-Means 3D (slice 400)")

fig.suptitle(
    "Porównanie wyników klasteryzacji na danych 3D algorytmem K-Means oraz Fuzzy C-Means"
)
plt.show()
```

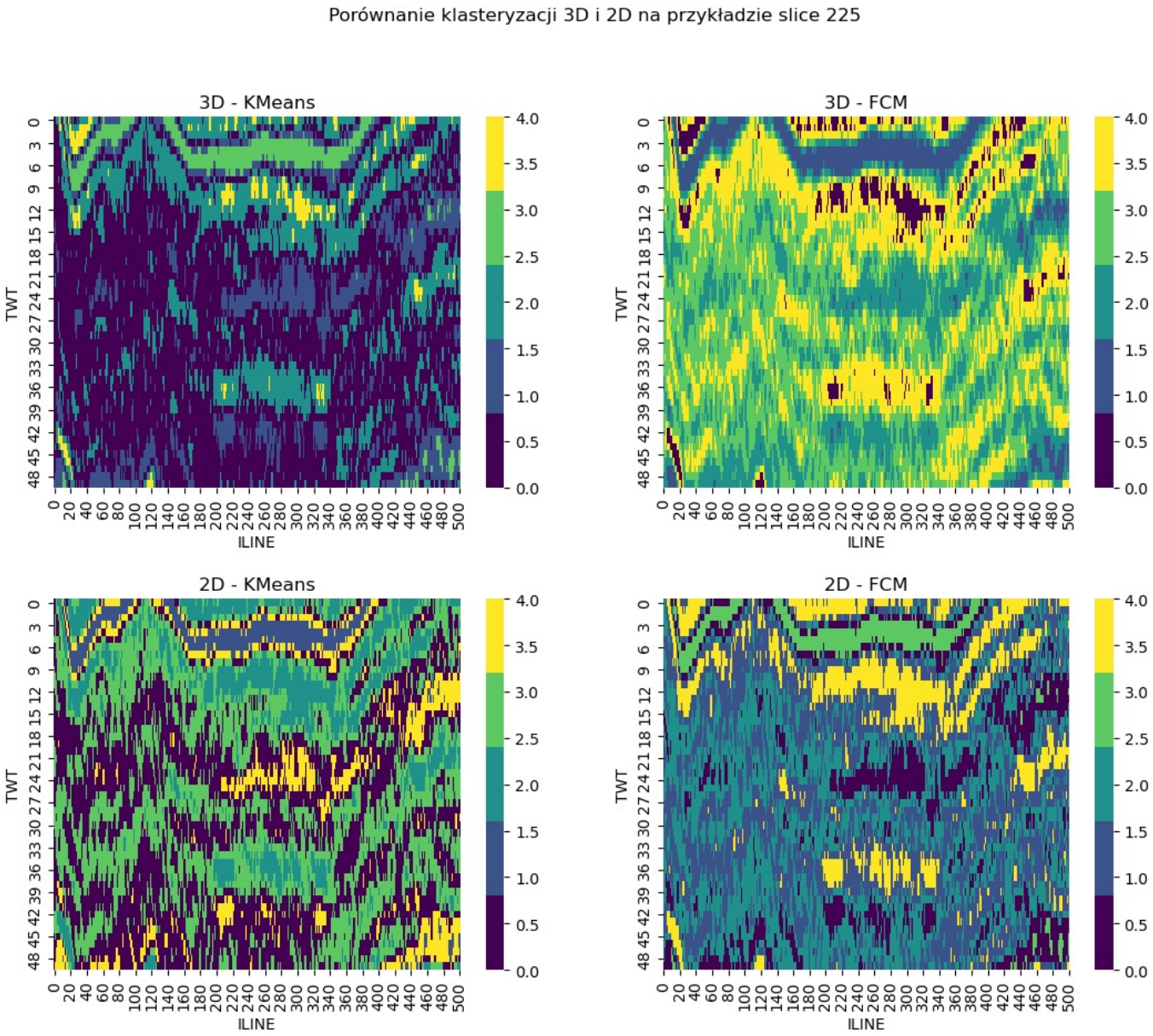


Jedną z trywialnych zalet klasteryzacji 3D jest fakt, że na poszczególnych slice'ach poszczególne klasy będą oznaczone tym samym kodem, co umożliwia szybką analizę wielu wycinków naraz (nie jest potrzebna dokładniejsza analiza, która klasa łączy się z którą). W przeciwieństwie do klasteryzacji na slice'ach, w tym przypadku widać wyraźniejsze różnice pomiędzy oboma algorytmami. Fuzzy C-Means jest wyraźnie "gładsze" i daje (czysto empiryczne) poczucie większej pewności przy dopasowaniu do klas.

3.3 Porównanie klasteryzacji 2D i 3D

In [ ]:
```
fig, axes = plt.subplots(2, 2, figsize=(13, 10))

sns.heatmap(clustered_cube[225, :, :].T, ax=axes[0, 0], cmap=pal)
sns.heatmap(clustered_cube_fuzzy[225, :, :].T, ax=axes[0, 1], cmap=pal)

sns.heatmap(clustered_slice_225.T, ax=axes[1, 0], cmap=pal)
sns.heatmap(clustered_slice_fuzzy_225.T, ax=axes[1, 1], cmap=pal)

axes[0, 0].set(title="3D - KMeans", xlabel="ILINE", ylabel="TWT")
axes[0, 1].set(title="3D - FCM", xlabel="ILINE", ylabel="TWT")

axes[1, 0].set(title="2D - KMeans", xlabel="ILINE", ylabel="TWT")
```

```
axes[1, 1].set(title="2D - FCM", xlabel="ILINE", ylabel="TWT")

fig.suptitle("Porównanie klasteryzacji 3D i 2D na przykładzie slice 225")

plt.subplots_adjust(hspace=0.3)
plt.show()
```



Na powyższej figurze można zobaczyć porównanie klasteryzacji na całym cube 3D (pierwszy rząd) oraz na osobnym slice 2D metodami K-Means oraz Fuzzy C-Means na przykładzie slice'u nr 225. Różnice nie są duże, jednak można zauważyć że:

- Klasteryzacja metodą FCM daje wyniki mniej poszarpane, klastry są bardziej spójne i mniej mieszjaą się między sobą
- Klasteryzacja na danych 3D daje wyniki jeszcze spójniejsze, możliwe że zbyt stałe (niektóre wzorce mogą zostać pominięte)

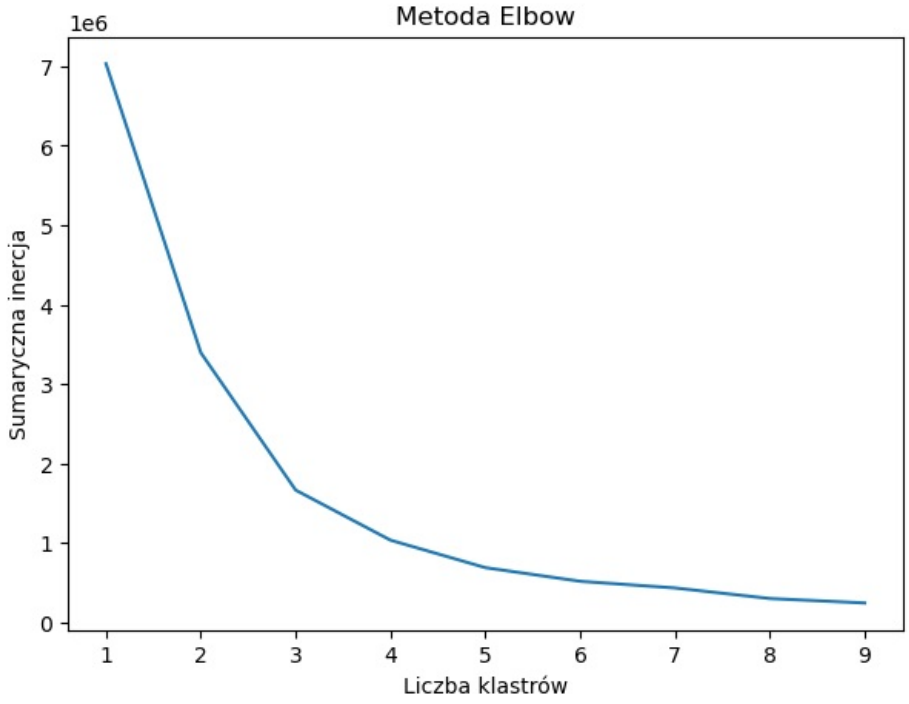
5. Dobór odpowiedniej liczby klastrów

W [artykule]!(https://www.mdpi.com/1996-1073/16/1/493), w rozdziale 3.5 przedstawiono 3 różne metody doboru optymalnej ilości klastrów: Silhouette, Davies-Bouldin Index oraz Calinski-Harabasz Index. Jak wspomniano w tym artykule, dobór liczby klastrów nie jest zadaniem trywialnym i nie ma swojej jednoznacznej odpowiedzi. Jedną z najprostszych metod determinacji liczby klastrów jest metoda "Elbow", która nie została wymieniona w tym artykule. Jest to metoda subiektywna i nie zawsze daje jednoznaczne wyniki, dlatego nie jest ona zbyt popularna, jest jednak metodą najłatwiejszą i najszybszą w użytku. Poniżej sprawdzono ile klastrów należy użyć przy użyciu metody Elbow.

5.1 Metoda Elbow

In [ ]:
```
res = []

for i in range(1, 10):
    kmeans = KMeans(n_clusters=i, random_state=44, n_init="auto").fit(
        slice_225_reshaped
    )
```

```
    res.append(kmeans.inertia_)


fig, ax = plt.subplots(1, 1, figsize=(7,  5))
sns.lineplot(x=list(range(1, 10)), y=res, ax=ax)
ax.set(title="Metoda Elbow", xlabel="Liczba klastrów", ylabel="Sumaryczna inercja")

plt.show()
```



Jak widać metoda Elbow nie daje jednoznacznej liczby klastrów która powinna być użyta. Kandydaci to 3 i 4. Użycie innych metryk mogłoby dać większa pewność co do liczby klastrów która powinna być użyta, ale wychodzi to poza scope tego projektu.