Sheet

KM 3

```
import warnings
warnings.filterwarnings('ignore')
```

Alternatywna ramka danych

Po sprawdzeniu wyników na wstępnym modelu spostrzegliśmy, że nie zdają sie one być poprawne, dlatego postanowiliśmy stworzyć też alternatywną ramkę danych, która będzie zawierała kluczowe zebrane przez nas informacje. Ramka ta zawiera :

- liczbę wyrazów w każdym rozdziale
- liczbę liter w każdym rozdziale
- liczbę wyrazów z podziałem na części mowy
- polaryzję, czyli liczbę z zakresu <-1,1>, która określa czy tekst jest pozytywny czy negatywny.(-1 oznacza skrajnie negatywny, 1 skrajnie pozytywny).

Polaryzacja została zrobiona na dołączonym pełnym tekście rozdziałów, ponieważ badanie tylko pojedynczych słów nie przyniosłoby oczekiwanych rezultatów. Na tak stworzonej ramce przetestowaliśmy kilka modeli.

Przeprowadziliśmy również ten sam preprocessing jaki w przypadku poprzedniego modelu

```
text = open("Complete_data .txt", "r",encoding="latin1")
text = text.read()
```

```
import re

split_text = re.split("[0-7].[0-9]+\n", text)
split_text.pop(0)
```

```
''
```

Stworzenie polarity

```
from textblob import TextBlob
text_sentiment = []
for i in range (0,590):
    text_sentiment.append(TextBlob(split_text[i]).sentiment)
text_polarity = []
for i in range (0,590):
    text_polarity.append(TextBlob(split_text[i]).sentiment.polarity)
```

Zmiany kosmeteczne

```
import pandas as pd
allBooks = pd.read_csv("AllBooks_baseline_DTM_Unlabelled.csv").rename(columns={'# foolishness': 'fool
allBooks
```

	foolishness	hath	wholesome	takest	feelings	anger	vaivaswata	matrix	kindled	convict	...	erred	thinkest	modern	reigned	spa
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
585	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
586	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
587	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0
588	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
589	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

590 rows × 8266 columns

Obliczenie sumy wyrazow w rozdziale

```
number_of_words = allBooks.sum(axis=1)
```

Obliczenie sumy liter w rozdziale

```
letter_count = allBooks.copy()

for word in letter_count.columns:
    letter_count[word] *= len(word)

number_of_letters = letter_count.sum(axis=1)
```

Usunięcie stopwordów

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
to_be_removed = [word for word in allBooks.columns.values if word in stop_words]
allBooks = allBooks.drop(to_be_removed, axis=1)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/datalore/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Stemming

```
from nltk.stem import LancasterStemmer

ls = LancasterStemmer()
allBooks_stemmed = pd.DataFrame()

for el in allBooks.columns.values:

    col = ls.stem(el)

    if col in allBooks_stemmed.columns.values:
        allBooks_stemmed[col] = allBooks_stemmed[col] + allBooks[el]


    else:
        allBooks_stemmed[col] = allBooks[el]

allBooks = allBooks_stemmed
```

Policzenie wyrazów pogrupowanych według części mowy

```
nltk.download('averaged_perceptron_tagger', quiet=True)

parts_of_speech = nltk.pos_tag(allBooks.columns)
parts_of_speech = pd.DataFrame(parts_of_speech, columns=['Words', 'POS'])
parts_of_speech['POS'] = parts_of_speech.POS.astype('str')
columns= parts_of_speech.POS.unique().astype('str')
```

```
pos_x = nltk.pos_tag(allBooks.columns)
```

```
import numpy as np
data = np.zeros((590,24))
```

```
import pandas as pd
d = pd.DataFrame(data = data,columns= columns)
d
```

	JJ	NN	JJS	RB	FW	IN	VBD	DT	VBP	CC	...	JJR	PRP	VBZ	MD	VBN	WP	WRB	NNP
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
585	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
586	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
587	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
588	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
589	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

590 rows × 24 columns

```
for i in range(0,590):
    x = pos_x[i][1]
    string = pos_x[i][0]
    d[x] += allBooks[string]
```

```
d['Number_of_words'] = number_of_words
d['Number_of_letters'] = number_of_letters
d['Polarity'] = text_polarity
```

```
d
```

	JJ	NN	JJS	RB	FW	IN	VBD	DT	VBP	CC	...	MD	VBN	WP	WRB	NNP	RP	RBS	Numbe
0	18.0	53.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	298.0
1	7.0	28.0	0.0	2.0	0.0	0.0	0.0	0.0	2.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	107.0
2	13.0	64.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	188.0
3	6.0	33.0	0.0	1.0	4.0	0.0	0.0	0.0	3.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	129.0
4	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
585	13.0	29.0	0.0	0.0	0.0	0.0	1.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	237.0
586	12.0	48.0	0.0	0.0	0.0	0.0	3.0	8.0	2.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	378.0
587	7.0	40.0	0.0	2.0	1.0	0.0	2.0	0.0	2.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	246.0
588	10.0	43.0	1.0	1.0	2.0	1.0	7.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	344.0
589	11.0	43.0	1.0	0.0	0.0	0.0	4.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	284.0

590 rows × 27 columns

Stworzenie funkcji wykorzystywanych w predykcji oraz ocenie optymalnej liczby klastrów

```
pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in /opt/python/envs/default/lib/python3.8/site-packages (1
Requirement already satisfied: scikit-learn>=0.20 in /opt/python/envs/default/lib/python3.8/site-pack
Requirement already satisfied: cycler>=0.10.0 in /opt/python/envs/default/lib/python3.8/site-packages
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in /opt/python/envs/default/lib/python3.8/si
Requirement already satisfied: scipy>=1.0.0 in /opt/python/envs/default/lib/python3.8/site-packages (
Requirement already satisfied: numpy<1.20,>=1.16.0 in /opt/python/envs/default/lib/python3.8/site-pac
Requirement already satisfied: six in /opt/python/envs/default/lib/python3.8/site-packages (from cycl
Requirement already satisfied: pillow>=6.2.0 in /opt/python/envs/default/lib/python3.8/site-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/python/envs/default/lib/python3.8/site-packa
Requirement already satisfied: python-dateutil>=2.1 in /opt/python/envs/default/lib/python3.8/site-pa
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /opt/python/envs/default/l
Requirement already satisfied: joblib>=0.11 in /opt/python/envs/default/lib/python3.8/site-packages (
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/python/envs/default/lib/python3.8/site-pa
[33mWARNING: You are using pip version 21.1; however, version 21.1.2 is available.
You should consider upgrading via the '/opt/python/envs/default/bin/python3 -m pip install --upgrade
Note: you may need to restart the kernel to use updated packages.
```

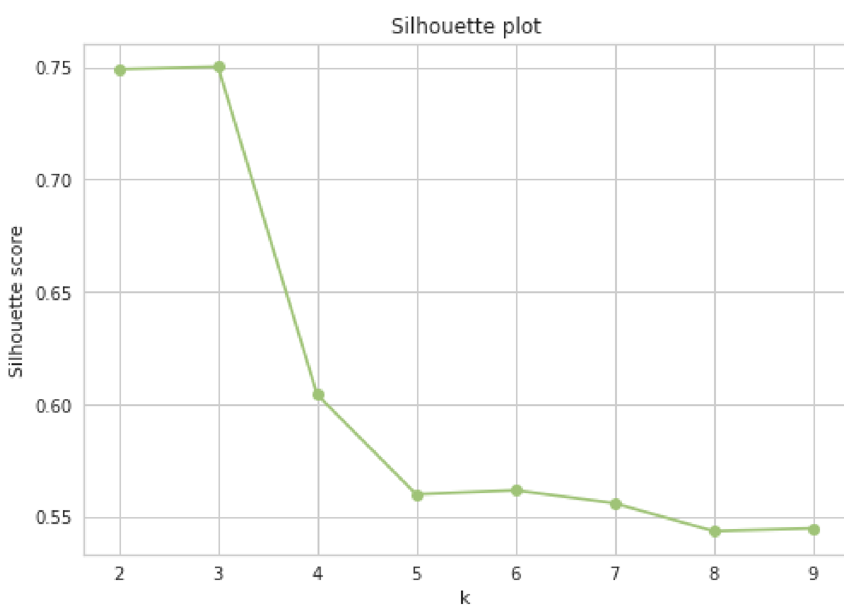
```
from sklearn.cluster import KMeans
from sklearn.cluster import Birch
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

import numpy as np
import seaborn as sns

def scatter(x,colors):
    palette = np.array(sns.color_palette("hls",10))

    f = plt.figure(figsize=(8,8))
    ax = plt.subplot(aspect = 'equal')
    sc = ax.scatter(x[:,0],x[:,1],lw=0,s=40,c=palette[colors.astype(np.int)])

    plt.xlim(-25,25)
    plt.ylim(-25,25)
    ax.axis('off')
    ax.axis('tight')

    txts = []
    for i in range(10):
        xtext,ytext = np.median(x[colors == i, :],axis = 0)
        txt = ax.text(xtext,ytext,str(i), fontsize =24)
        txts.append(txt)

    return f, ax, sc, txts
from sklearn.metrics import silhouette_score
from yellowbrick.cluster import KElbowVisualizer


def silhouette(df, i):
    if i == 1:
        model = KMeans(random_state= 0)
    elif i == 2:
        model = Birch(threshold=5)
    elif i == 3:
        model = AgglomerativeClustering()
    cluster_num_seq = range(2, 10)
    scores = []

    for k in cluster_num_seq:
        model.n_clusters = k
        labels = model.fit_predict(df)
        score = silhouette_score(df, labels)
        scores.append(score)

    plt.plot(cluster_num_seq, scores, 'go-')
    plt.xlabel('k')
    plt.ylabel('Silhouette score')
    plt.title('Silhouette plot')
    plt.show()



def calinski_harabasz(df, i):
    if i == 1:
        visualizer = KElbowVisualizer(
        KMeans(random_state= 0), k=(2,10), metric='calinski_harabasz', timings=False, locate_elbow=Fa
    )
    elif i == 2:
        visualizer = KElbowVisualizer(
        Birch(threshold=5), k=(2,10), metric='calinski_harabasz', timings=False, locate_elbow=False
    )
    elif i == 3:
        visualizer = KElbowVisualizer(
        AgglomerativeClustering() , k=(2,10), metric='calinski_harabasz', timings=False, locate_elbow
    )
    else: return
```

```
    visualizer.fit(df)
    visualizer.show()




def tSNE_function(df, n, i):
    random_state = 10
    tSNE = TSNE(random_state=random_state, verbose=0)
    books_proj = tSNE.fit_transform(df)
    mod = KMeans(n_clusters=n)
    if i == 2:
        mod = Birch(threshold=5, n_clusters=n)
    if i == 3:
        mod = AgglomerativeClustering(n_clusters=n)
    if i == 4:
        mod = DBSCAN(eps=4)
    y = mod.fit_predict(df)

    scatter(books_proj, y)
    plt.show()
```

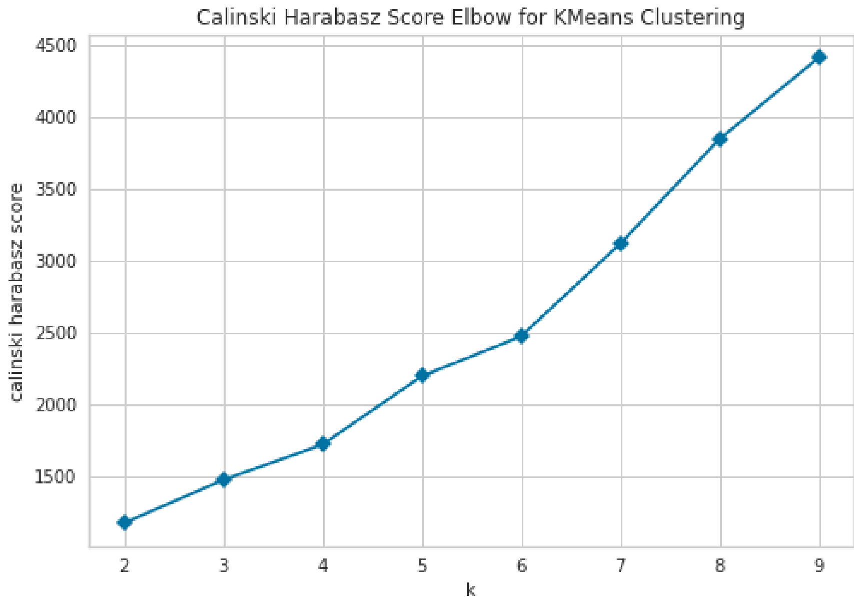
Model w przypadku alternatywnej ramki

KMeans

```
silhouette(d,1);
```

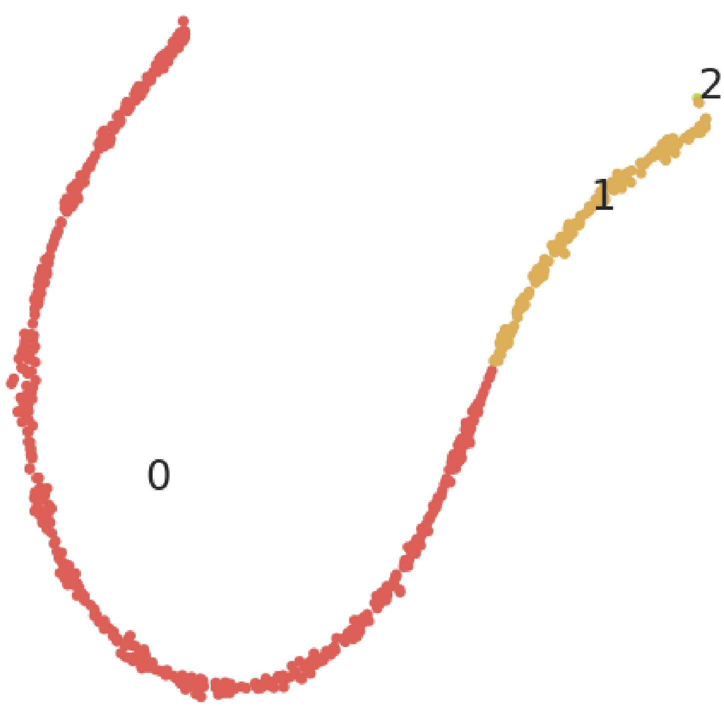



```
calinski_harabasz(d,1)
```



Metoda silhouette wskazała, że optymalną liczbą klastrów, w przypadku metody KMeans, będą 3 klastry, natomiast metoda Calińskiego-Harabasza: 9

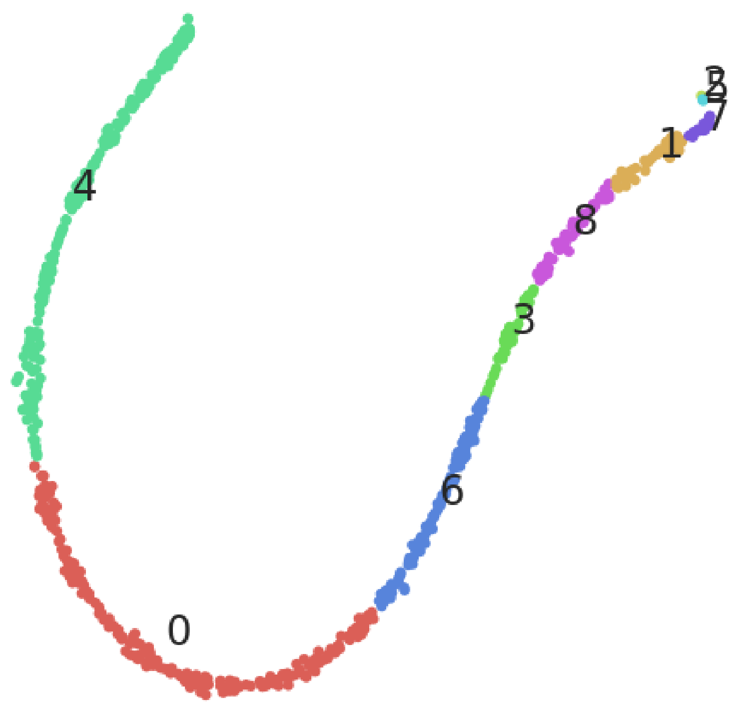
```
tSNE_function(d,3,1);
```




```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

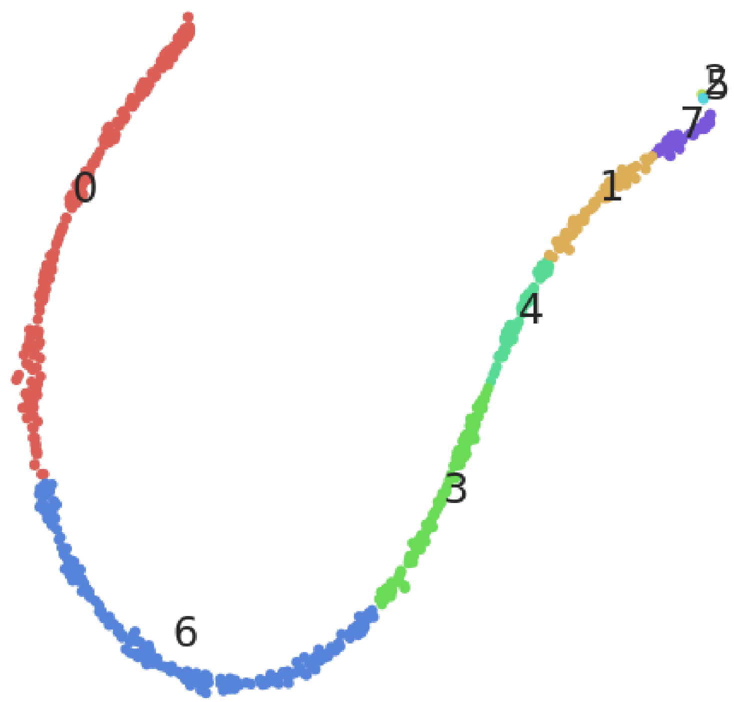
```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
tSNE_function(d,9,1);
```




```
posx and posy should be finite values
posx and posy should be finite values
```

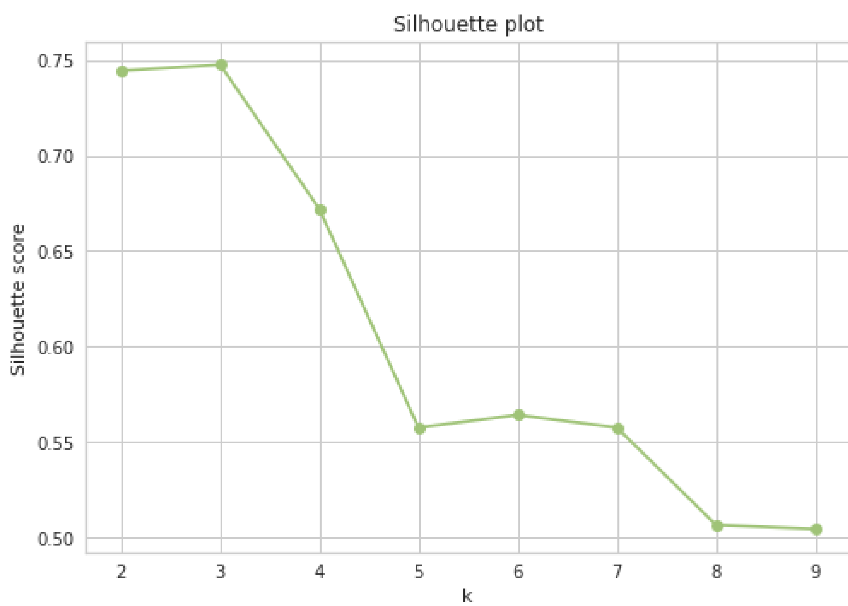
```
tSNE_function(d,8,1)
```



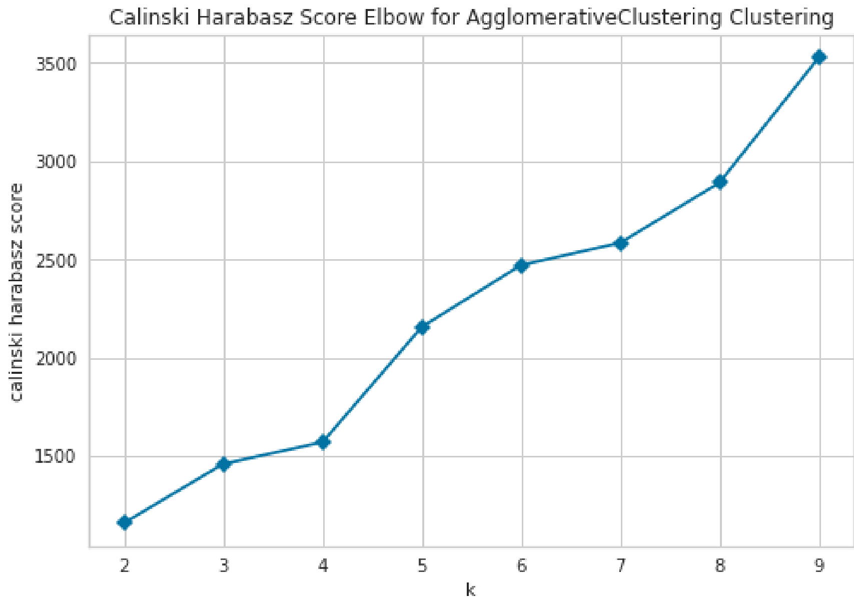

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

AgglomerativeClustering

```
silhouette(d,3)
```

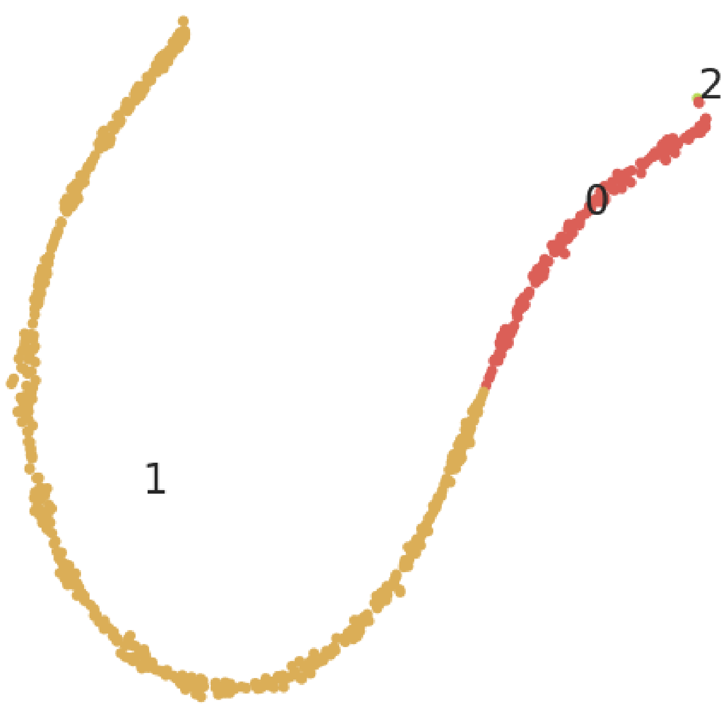



```
calinski_harabasz(d,3)
```



Metoda silhouette wskazała, że optymalną liczbą klastrów, w przypadku metody AgglomerativeClustering, będą 3 klastry, natomiast metoda Calińskiego-Harabasza: 9

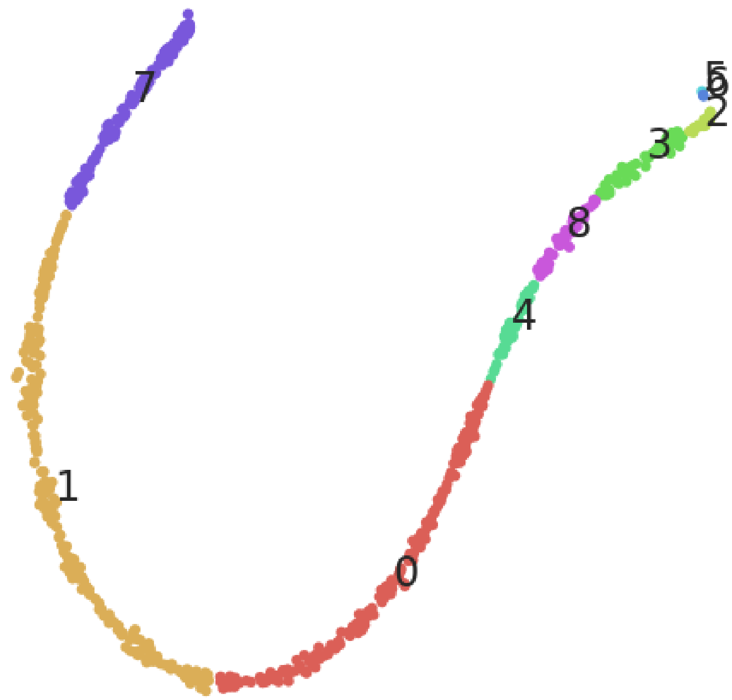
```
tSNE_function(d,3,3)
```




```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

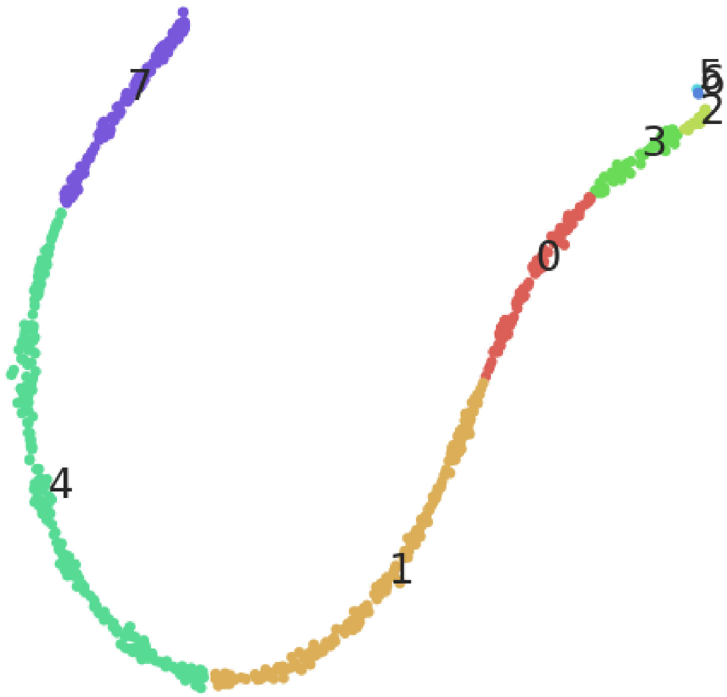
```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
tSNE_function(d,9, 3)
```




```
posx and posy should be finite values
posx and posy should be finite values
```

```
tSNE_function(d,8,3)
```




```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

Word2Vec

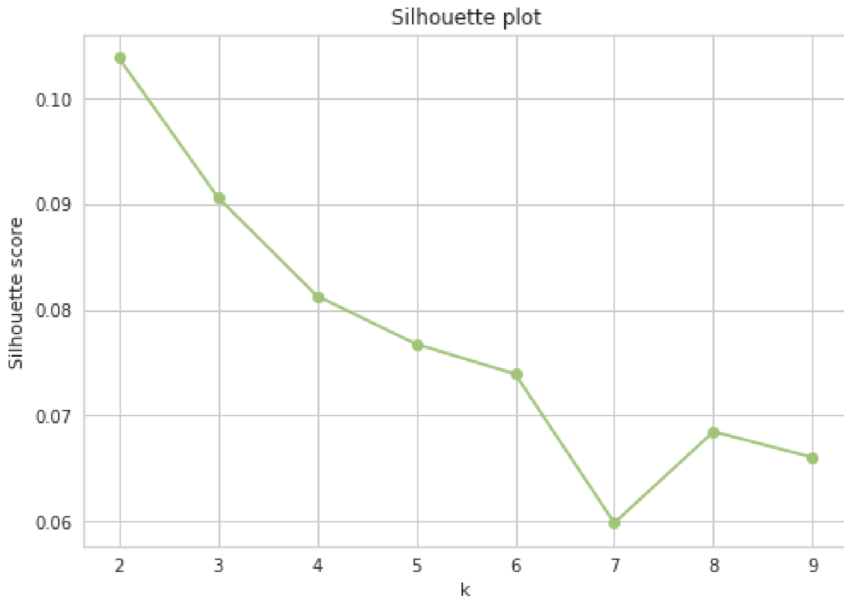
Kolejną naszą cechą, którą rozważyliśmy zrobić było zamienienie rozdziałów na 300 wymiarowe wektory, która miałyby usprawnić działanie naszego modelu. Proces wektoryzacji słów opisany został w osobnym notebooku: word2vec.ipynb.

```
df_vectorized = pd.read_csv(open("df_vectorized.csv", "rb"))
```

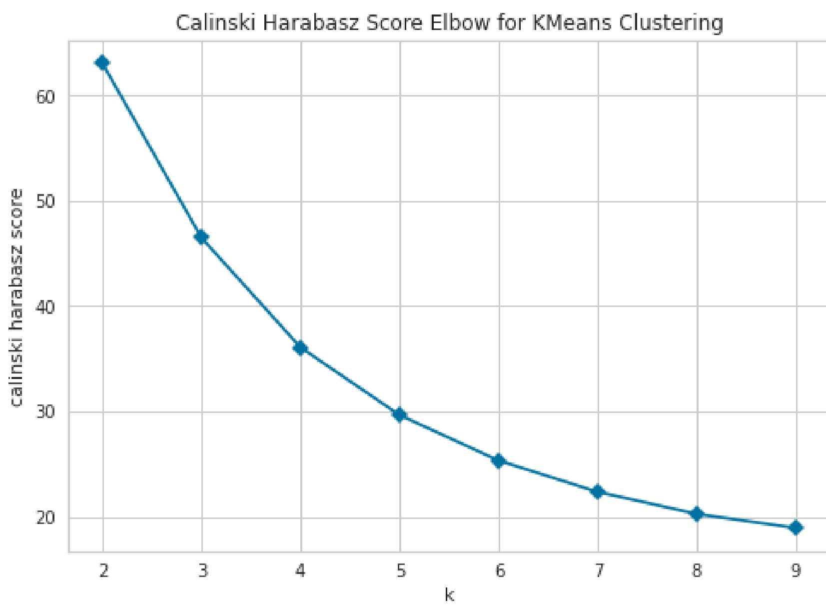
Model w przypadku wektorów słów

KMeans

```
silhouette(df_vectorized, 1)
```

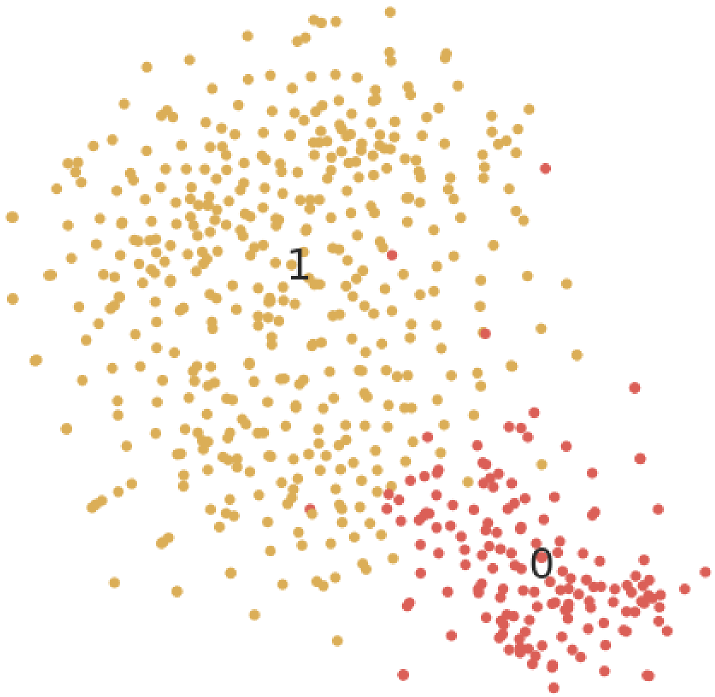



```
calinski_harabasz(df_vectorized, 1)
```



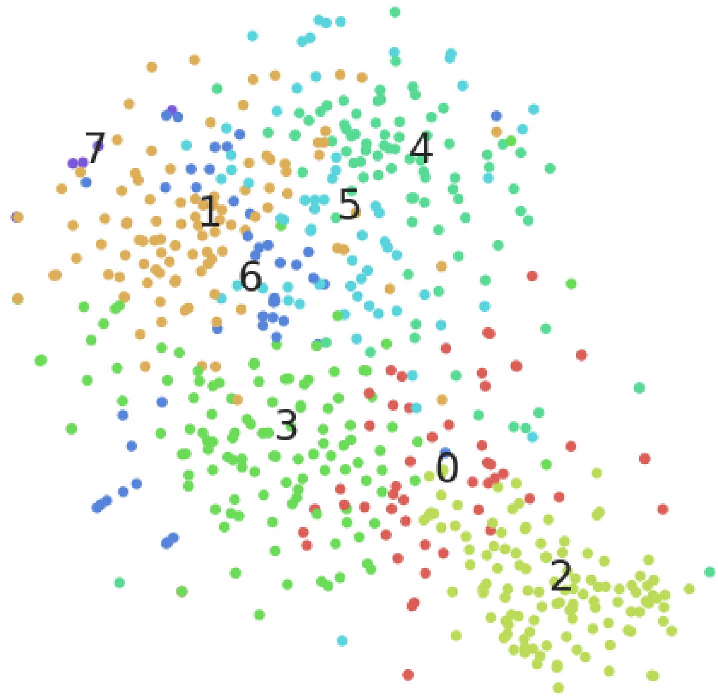
Zarówno metoda silhouette jak i Calińskiego-Harabasza wskazały, że optymalną liczbą klastrów będzie 8

```
tSNE_function(df_vectorized, 2, 1)
```




```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

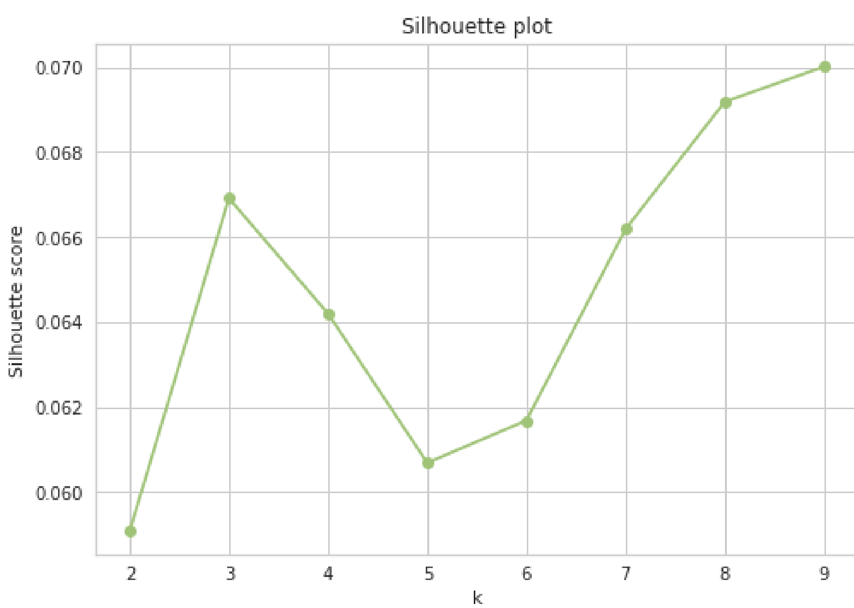
```
tSNE_function(df_vectorized, 8, 1)
```



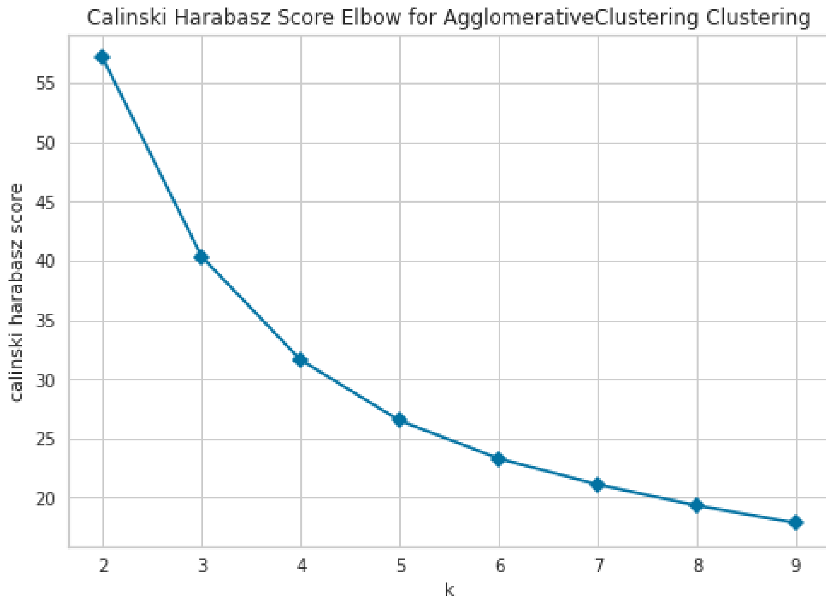
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values

AgglomerativeClustering

```
silhouette(df_vectorized, 3)
```




```
calinski_harabasz(df_vectorized, 3)
```

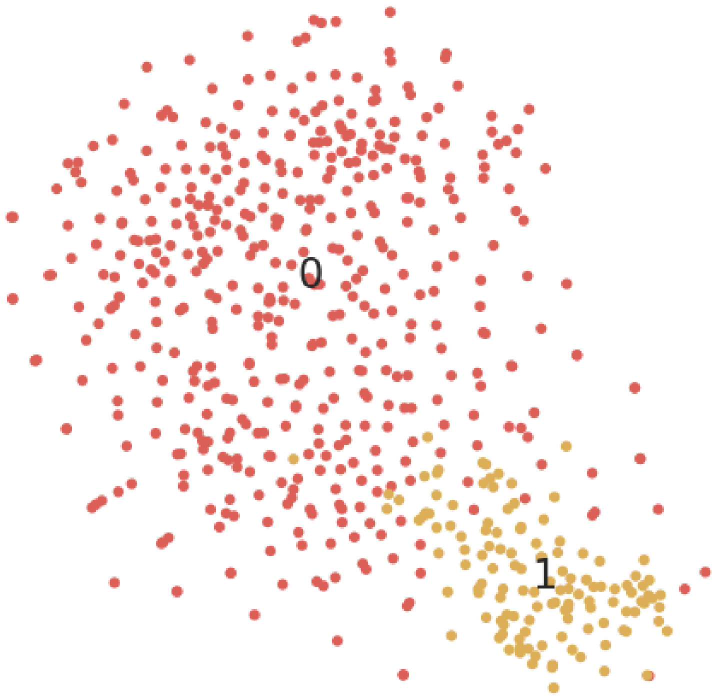


Metoda silhouette wskazała, że optymalną liczbą klastrów, w przypadku metody AgglomerativeClustering, będą 3 klastry, natomiast metoda Calińskiego-Harabasza: 2

```
tSNE_function(df_vectorized, 3, 3)
```

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
tSNE_function(df_vectorized, 2, 3)
```




```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
tSNE_function(df_vectorized, 8, 3)
```

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

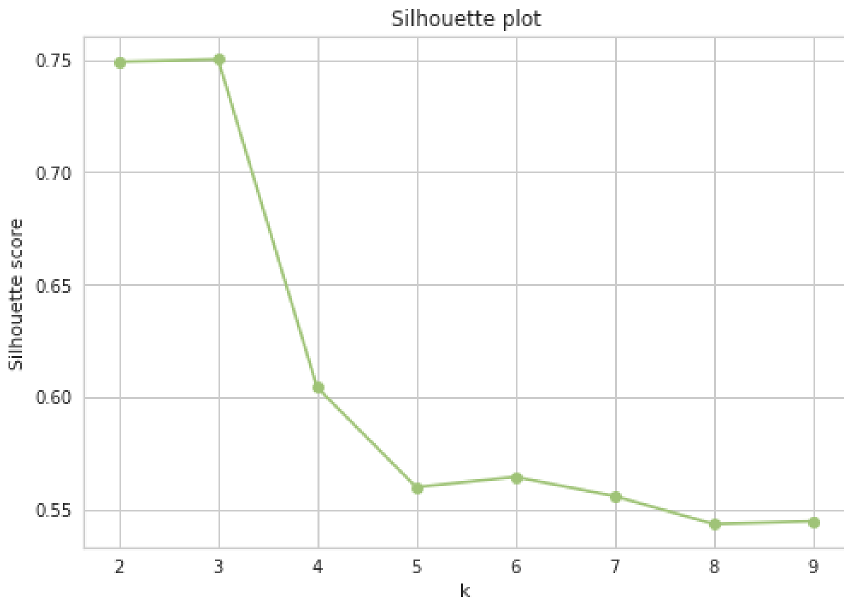
Model w przypadku wektora słów + alternatywnej ramki

Stwierdziliśmy, że możemy również spróbować połaczyć obie ramki danych, zawierające nieco odmienne informacje, w celu poprawienia predykcji

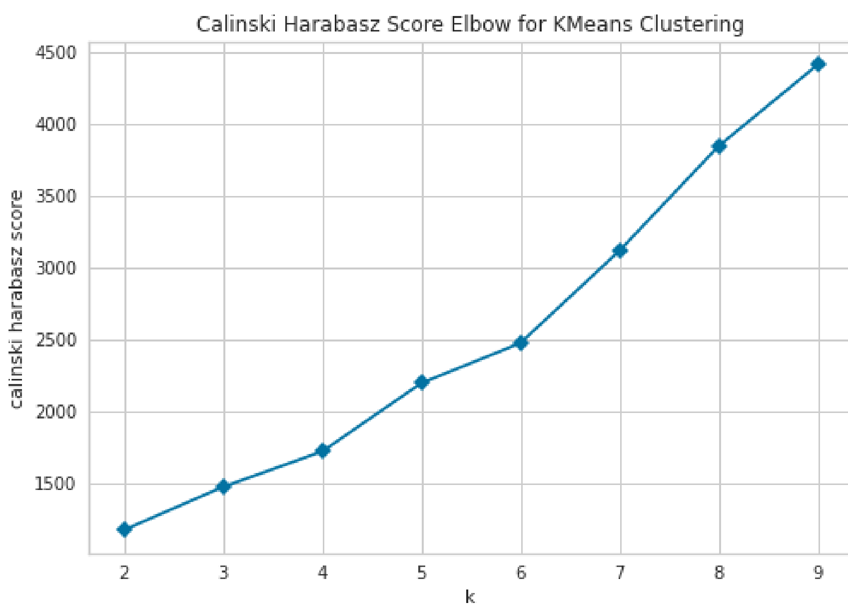
```
df_merged = pd.concat([df_vectorized, d.reindex(df_vectorized.index)], axis=1)
```

KMeans

```
silhouette(df_merged, 1)
```

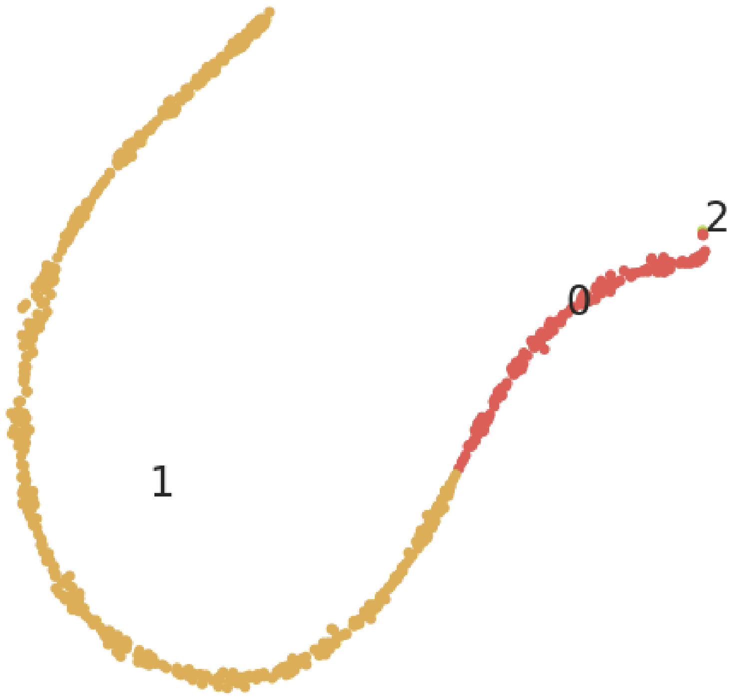



```
calinski_harabasz(df_merged, 1)
```



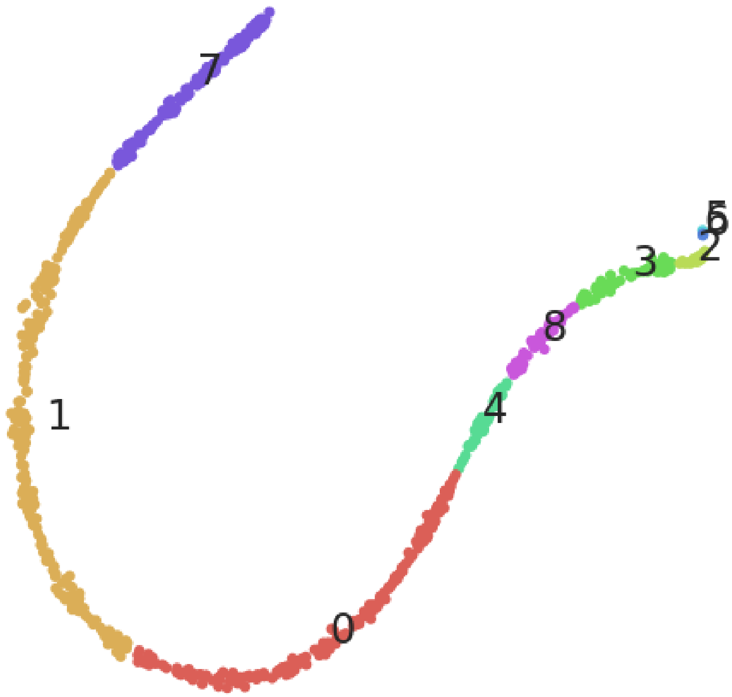
Metoda silhouette wskazała, że optymalną liczbą klastrów, w przypadku metody KMeans, będą 3 klastry, natomiast metoda Calińskiego-Harabasza: 9

```
tSNE_function(df_merged, 3, 3)
```



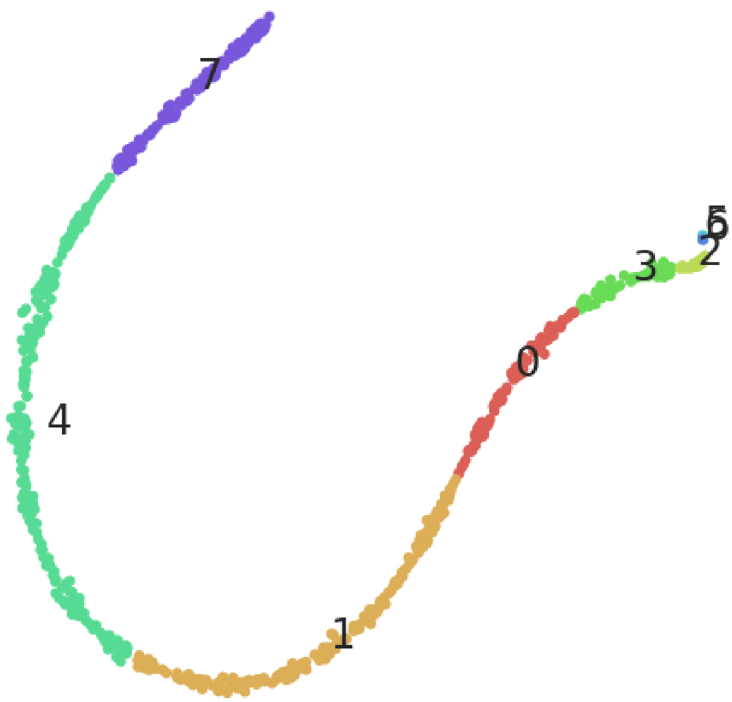

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
tSNE_function(df_merged, 9, 3)
```




```
posx and posy should be finite values
posx and posy should be finite values
```

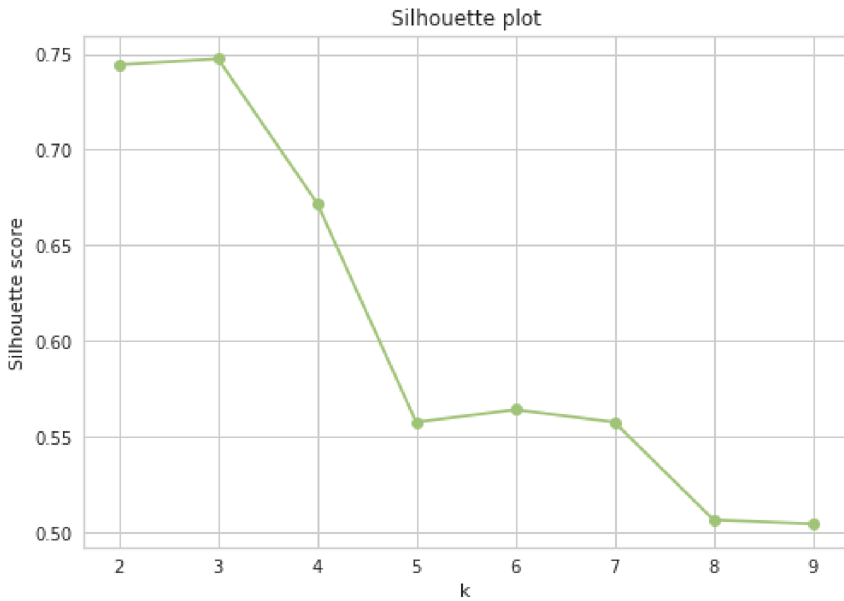
```
tSNE_function(df_merged, 8, 3)
```



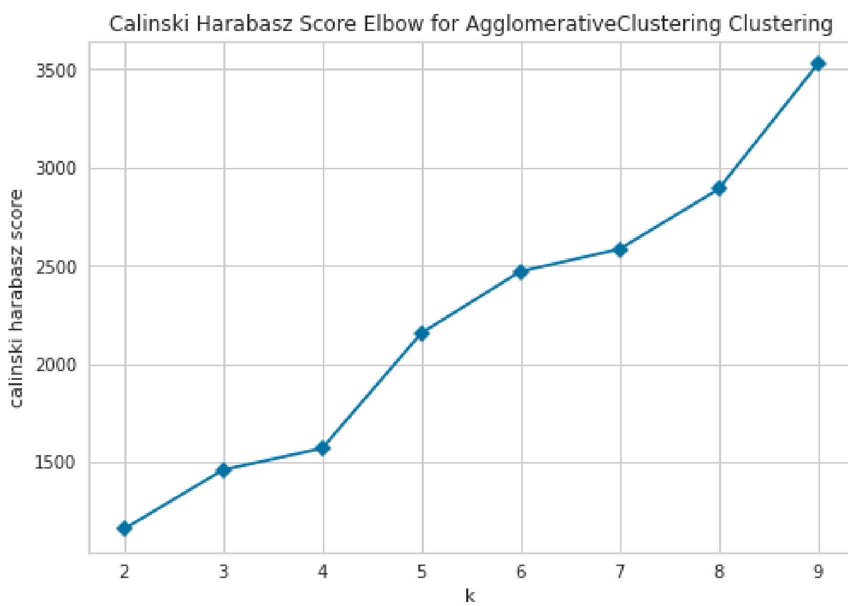

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

AgglomerativeClustering

```
silhouette(df_merged, 3)
```

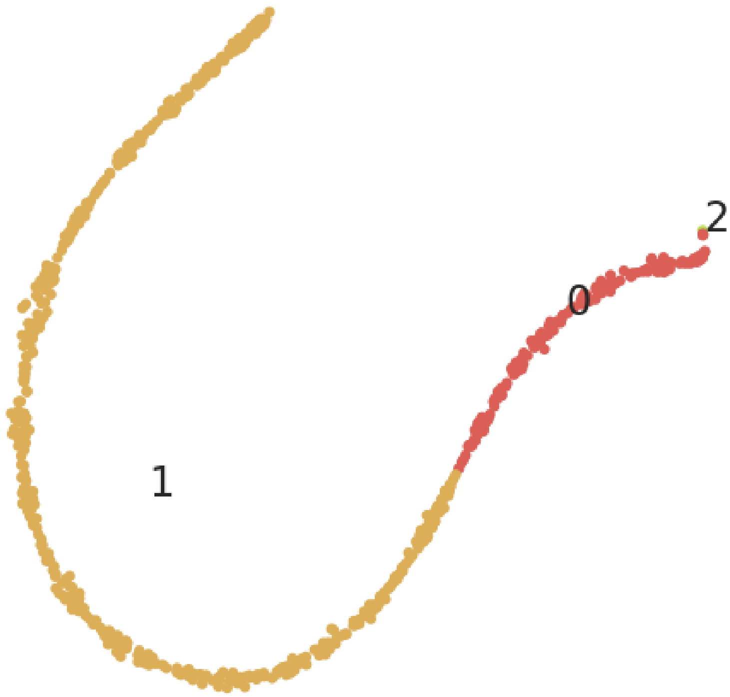



```
calinski_harabasz(df_merged, 3)
```



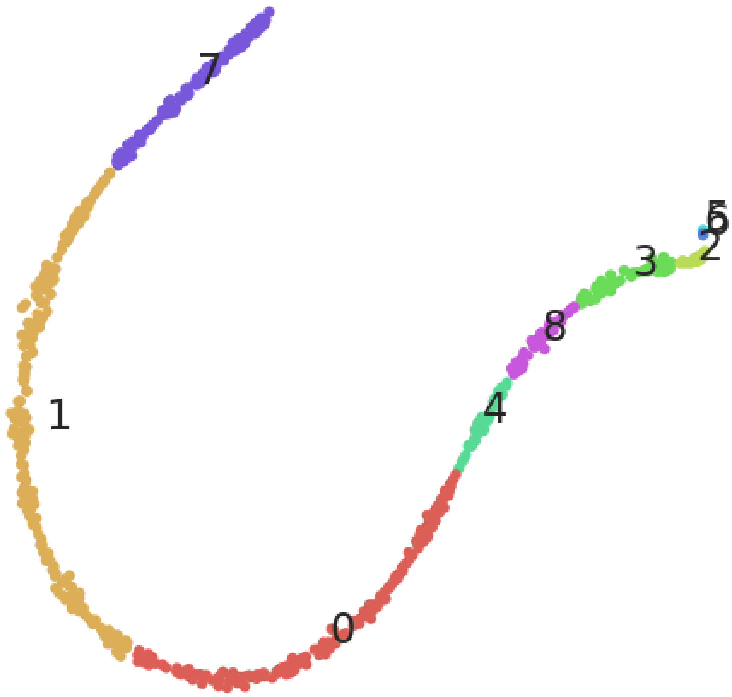
Metoda silhouette wskazała, że optymalną liczbą klastrów, w przypadku metody KMeans, będą 3 klastry, natomiast metoda Calińskiego-Harabasza: 9

```
tSNE_function(df_merged, 3, 3)
```



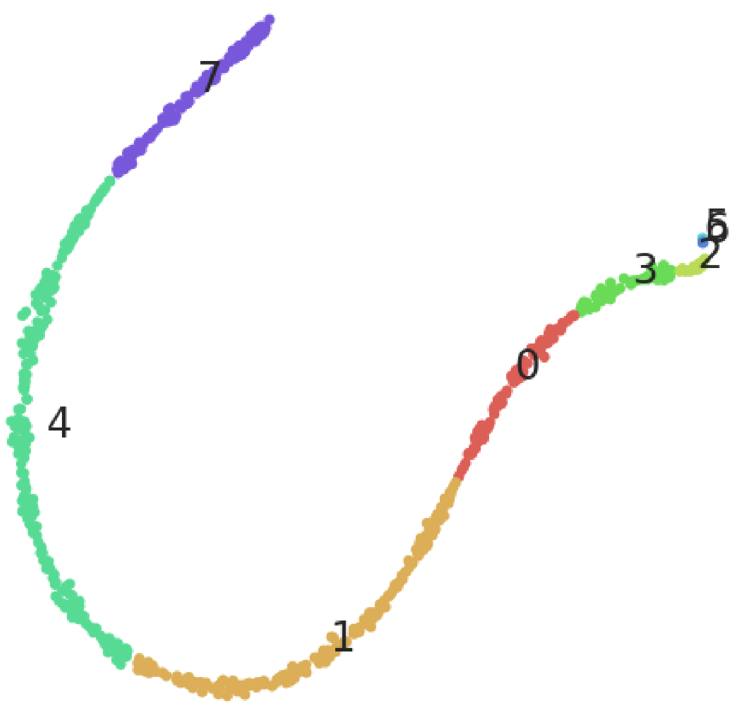

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
tSNE_function(df_merged, 9, 3)
```



posx and posy should be finite values
posx and posy should be finite values

```
tSNE_function(df_merged, 8, 3)
```



posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values

Sprawdzenie wyników

Adjusted Rand Index

Do oceny jakości klasterowania użyliśmy miary ARI. Nasze zadanie wymagało użycia Adjusted Rand Index w zamian za Rand Index, bo o ile liczba klastrów była stała, to liczba elementów w klastrach była zmienna i zależała od modelu i danych. Rand Index jest miarą analogiczną do Accuracy, ale stosowaną w zadaniach klasteryzacji. Wartość 0 miary oznacza całkowicie losowy przydział klastrów, a wartość jeden idealny podział danych na klastry.

Najpierw stworzyliśmy listę poprawnych labelów. W tym celu skorzystaliśmy z drugiej ramki danych (AllBooks_baseline_DTM_Labelled.csv), z której wyekstraktowaliśmmy a następnie odpowiedni zformatowaliśmy kolumnę z etykietami.

```
from sklearn.metrics import adjusted_rand_score
```

```
allBooks_labelled = pd.read_csv(open("AllBooks_baseline_DTM_Labelled.csv", "br"))
```

```
allBooks_labelled = allBooks_labelled.rename(columns={'Unnamed: 0': 'true_label'})
allBooks_labelled["true_label"] = allBooks_labelled["true_label"].map(lambda x: x.partition("_")[0])
allBooks_labelled["true_label"]
```

```
0          Buddhism
1          Buddhism
2          Buddhism
3          Buddhism
4          Buddhism
             ...
585    BookOfWisdom
586    BookOfWisdom
587    BookOfWisdom
588    BookOfWisdom
589    BookOfWisdom
Name: true_label, Length: 590, dtype: object
```

```
from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder()
true_labels = encoder.fit_transform(allBooks_labelled[["true_label"]])
encoder.categories_
```

```
[array(['BookOfEccleasiasticus', 'BookOfEcclesiastes', 'BookOfProverb',
        'BookOfWisdom', 'Buddhism', 'TaoTeChing', 'Upanishad', 'YogaSutra'],
       dtype=object)]
```

```
true_labels = [i[0] for i in true_labels]
```

Pierwotna ramka danych

KMeans

```
df_raw = pd.read_csv("AllBooks_baseline_DTM_Unlabelled.csv").rename(columns={'# foolishness': 'foolis
kmeans = KMeans(n_clusters=8)
y = kmeans.fit_predict(df_raw)
adjusted_rand_score(true_labels, y)
```

```
0.18831143518258045
```

AgglomerativeClustering

```
agglomerative_clustering = AgglomerativeClustering(n_clusters=8)
y = agglomerative_clustering.fit_predict(df_raw)
adjusted_rand_score(true_labels, y)
```

```
0.229089356978241
```

Alternatywna ramka danych

KMeans

```
kmeans = KMeans(n_clusters=8)
y = kmeans.fit_predict(d)
adjusted_rand_score(true_labels, y)
```

```
0.17277497612297507
```

AgglomerativeClustering

```
agglomerative_clustering = AgglomerativeClustering(n_clusters=8)
y = agglomerative_clustering.fit_predict(d)
adjusted_rand_score(true_labels, y)
```

```
0.20435153827970956
```

Zwektoryzowana ramka danych

KMeans

```
kmeans = KMeans(n_clusters=8)
y = kmeans.fit_predict(df_vectorized)
adjusted_rand_score(true_labels, y)
```

```
0.16460032197376923
```

AgglomerativeClustering

```
agglomerative_clustering = AgglomerativeClustering(n_clusters=8)
y = agglomerative_clustering.fit_predict(df_vectorized)
adjusted_rand_score(true_labels, y)
```

```
0.23553917283784076
```

Zmergeowana ramka danych

KMeans

```
kmeans = KMeans(n_clusters=8)
y = kmeans.fit_predict(df_merged)
adjusted_rand_score(true_labels, y)
```

```
0.17277497612297507
```

AgglomerativeClustering

```
agglomerative_clustering = AgglomerativeClustering(n_clusters=8)
y = agglomerative_clustering.fit_predict(df_merged)
adjusted_rand_score(true_labels, y)
```

```
0.20435153827970956
```