Milestone2

June 2, 2021

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1 Wczytanie danych

[6]:
```
#ramka danych ze słowawmi
df = pd.read_csv('data.csv')
df.head()
```

[6]:
```
   foolishness  hath  wholesome  tak  feeling  anger  vaivaswata  matrix  \
0          0.0   0.0        0.0  0.0      1.0    0.0         0.0     0.0
1          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
3          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

   kindle  convict  …  lifeless  postponement  stout  taketh  kettle  \
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
2     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
4     0.0      0.0  …       0.0           0.0    0.0     0.0     0.0

   thinkest  sparingly  visual  thought  attire
0       0.0        0.0     0.0      0.0     0.0
1       0.0        0.0     0.0      0.0     0.0
2       0.0        0.0     0.0      0.0     0.0
3       0.0        0.0     0.0      0.0     0.0
4       0.0        0.0     0.0      0.0     0.0

[5 rows x 6111 columns]
```

1.1 Skalowanie ramki za pomocą TF IDF

[7]:
```
cols = df.columns
texts = [''] * len(df)
for i in range(len(df)):
    t = texts[i]
    tmp_num = np.array(df.iloc[i])
    for j in range(len(tmp_num)):
        w = int(tmp_num[j])
        for k in range(w): t = t + ' ' + cols[j]
    texts[i] = str(t)
    #print(texts[i])
```

[8]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=0.9, min_df=2, use_idf=True,␣
 ↪stop_words='english', token_pattern=r"\b[^\d\W]+\b")

tfidf = tfidf_vectorizer.fit_transform(texts)
tfidf_feature_names = tfidf_vectorizer.get_feature_names()

df_tfidf = pd.DataFrame(tfidf.toarray(), columns=list(tfidf_feature_names))
```

[9]:
```
df_tfidf
```

[9]:
```
     aaron   abandon  abasement  abate   abateth  abhor  abhorreth  abide  \
0      0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0
1      0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0
2      0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0
3      0.0  0.085756        0.0    0.0  0.000000    0.0        0.0    0.0
4      0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0
..     …       …         …     …       …     …        …    …
585    0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0
586    0.0  0.000000        0.0    0.0  0.054215    0.0        0.0    0.0
587    0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0
588    0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0
589    0.0  0.000000        0.0    0.0  0.000000    0.0        0.0    0.0

     abidingplace  ability  …  yellow  yes  yesterday     yield  yieldeth  \
0             0.0      0.0  …     0.0  0.0        0.0  0.000000       0.0
1             0.0      0.0  …     0.0  0.0        0.0  0.000000       0.0
2             0.0      0.0  …     0.0  0.0        0.0  0.000000       0.0
3             0.0      0.0  …     0.0  0.0        0.0  0.000000       0.0
4             0.0      0.0  …     0.0  0.0        0.0  0.000000       0.0
..            …      …  …     …  …        …       …       …
585           0.0      0.0  …     0.0  0.0        0.0  0.000000       0.0
586           0.0      0.0  …     0.0  0.0        0.0  0.000000       0.0
```

```
587           0.0       0.0  ...     0.0  0.0         0.0  0.071308       0.0
588           0.0       0.0  ...     0.0  0.0         0.0  0.000000       0.0
589           0.0       0.0  ...     0.0  0.0         0.0  0.000000       0.0

     yoga  yoke  young  youth  zeal
0     0.0   0.0    0.0    0.0   0.0
1     0.0   0.0    0.0    0.0   0.0
2     0.0   0.0    0.0    0.0   0.0
3     0.0   0.0    0.0    0.0   0.0
4     0.0   0.0    0.0    0.0   0.0
..    ...   ...    ...    ...   ...
585   0.0   0.0    0.0    0.0   0.0
586   0.0   0.0    0.0    0.0   0.0
587   0.0   0.0    0.0    0.0   0.0
588   0.0   0.0    0.0    0.0   0.0
589   0.0   0.0    0.0    0.0   0.0

[590 rows x 3366 columns]
```

1.2 Wczytanie i standaryzacja statystyk tekstów

[2]:
```
#ramka danych ze statystykami tesktów
stats = pd.read_csv('stats_df.csv')
stats = stats.drop(['Unnamed: 0', 'index', 'text'], axis = 1)
stats.head()
```

[2]:
```
    len  words   avg_sen  reading_ease  grade  sentences
0  3631    587  5.031561         38.39   16.0         18
1  1512    265  4.705660         80.01    6.2         16
2  2204    370  4.877333         71.34    7.5         22
3  1584    267  4.823529         62.98    8.6         16
4   216     29  6.448276         56.76    8.9          2
```

[10]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(stats)
stat_scale = scaler.transform(stats)

stats_scale = pd.DataFrame(stat_scale, columns = stats.columns)
stats_scale.head(3)
```

[10]:
```
        len     words   avg_sen  reading_ease     grade  sentences
0  1.832013  1.549162  0.749075      0.162432 -0.298802   0.775681
1  0.208099  0.189544  0.040772      0.928372 -0.808777   0.609403
2  0.738420  0.632898  0.413880      0.768816 -0.741128   1.108236
```

1.3 Stworzenie zbioru do klasteryzacji

[156]:
```
X = pd.merge(stats_scale.reset_index(), df_tfidf.reset_index(), on = 'index').
↪drop('index', axis = 1)
X.head()
```

[156]:
```
        len     words   avg_sen  reading_ease     grade  sentences  aaron  \
0  1.832013  1.549162  0.749075      0.162432 -0.298802   0.775681    0.0
1  0.208099  0.189544  0.040772      0.928372 -0.808777   0.609403    0.0
2  0.738420  0.632898  0.413880      0.768816 -0.741128   1.108236    0.0
3  0.263277  0.197989  0.296945      0.614966 -0.683885   0.609403    0.0
4 -0.785101 -0.806946  3.828118      0.500498 -0.668274  -0.554540    0.0

    abandon  abasement  abate  …  yellow  yes  yesterday  yield  yieldeth  \
0  0.000000        0.0    0.0  …     0.0  0.0        0.0    0.0       0.0
1  0.000000        0.0    0.0  …     0.0  0.0        0.0    0.0       0.0
2  0.000000        0.0    0.0  …     0.0  0.0        0.0    0.0       0.0
3  0.085756        0.0    0.0  …     0.0  0.0        0.0    0.0       0.0
4  0.000000        0.0    0.0  …     0.0  0.0        0.0    0.0       0.0

   yoga  yoke  young  youth  zeal
0   0.0   0.0    0.0    0.0   0.0
1   0.0   0.0    0.0    0.0   0.0
2   0.0   0.0    0.0    0.0   0.0
3   0.0   0.0    0.0    0.0   0.0
4   0.0   0.0    0.0    0.0   0.0

[5 rows x 3372 columns]
```

1.4 Stworzenie ramki z odpowiedziami

[20]:
```
Y = pd.read_csv('AllBooks_baseline_DTM_Labelled.csv')[['Unnamed: 0']]
Y['label'] = Y['Unnamed: 0'].apply(lambda x: x.split('_')[0])


def add_religion(label):
  if label == "Buddhism": return "Buddhism"
  elif label == "TaoTeChing": return "Taoism"
  elif (label == "Upanishad") | (label =="YogaSutra"): return "Hindusim"
  else: return "Old testament"


Y['rel'] = Y['label'].apply(lambda x : add_religion(x))
Y = Y.drop('Unnamed: 0', axis = 1)
Y
```

[20]:

```
             label             rel
0         Buddhism        Buddhism
1         Buddhism        Buddhism
2         Buddhism        Buddhism
3         Buddhism        Buddhism
4         Buddhism        Buddhism
..             …               …
585   BookOfWisdom  Old testament
586   BookOfWisdom  Old testament
587   BookOfWisdom  Old testament
588   BookOfWisdom  Old testament
589   BookOfWisdom  Old testament

[590 rows x 2 columns]
```

2 Klasteryzacja bez redukcji wymiarów

2.1 Wyznaczenie liczby klastrów

[30]:

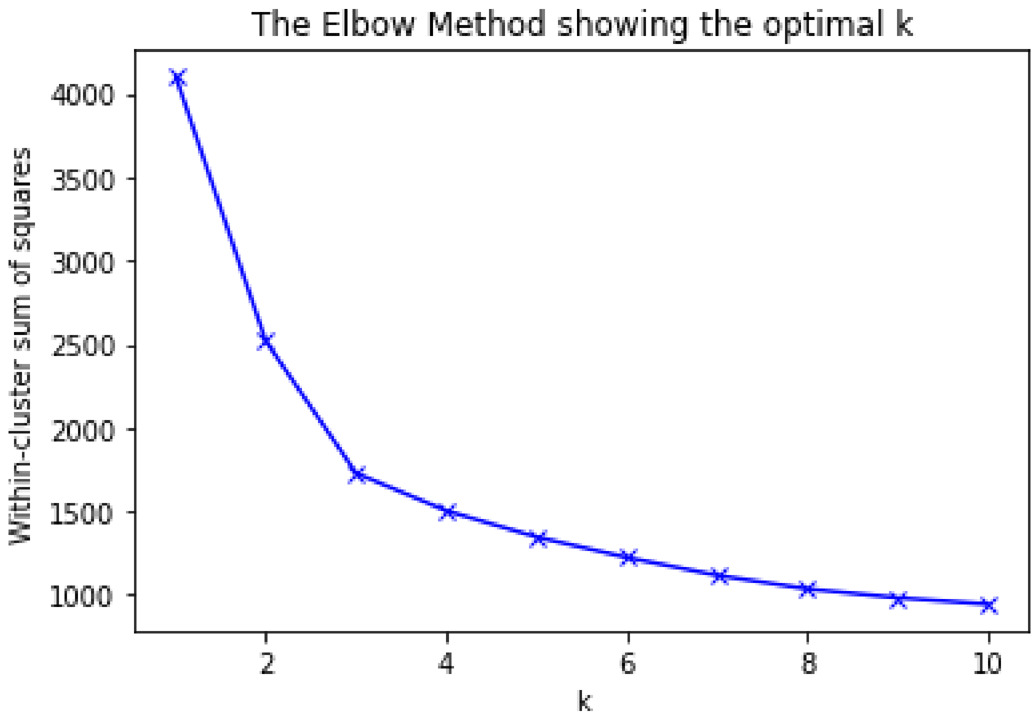
```python
from sklearn.cluster import KMeans, AgglomerativeClustering
```

[44]:

```python
# metdoda łokcia dla KMeans

def KMeansElbow(X, k_max):
    #  WCSS = within-cluster sum of squares
    scores = []
    for k in range(1, k_max+1):
        model = KMeans(n_clusters=k, random_state=0)
        model.fit(X)
        wcss = model.score(X) * -1 # score returns -WCSS
        scores.append(wcss)
    x_ticks = list(range(1, len(scores) + 1))
    plt.plot(x_ticks, scores, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Within-cluster sum of squares')
    plt.title('The Elbow Method showing the optimal k')
    plt.show()
```

[45]:

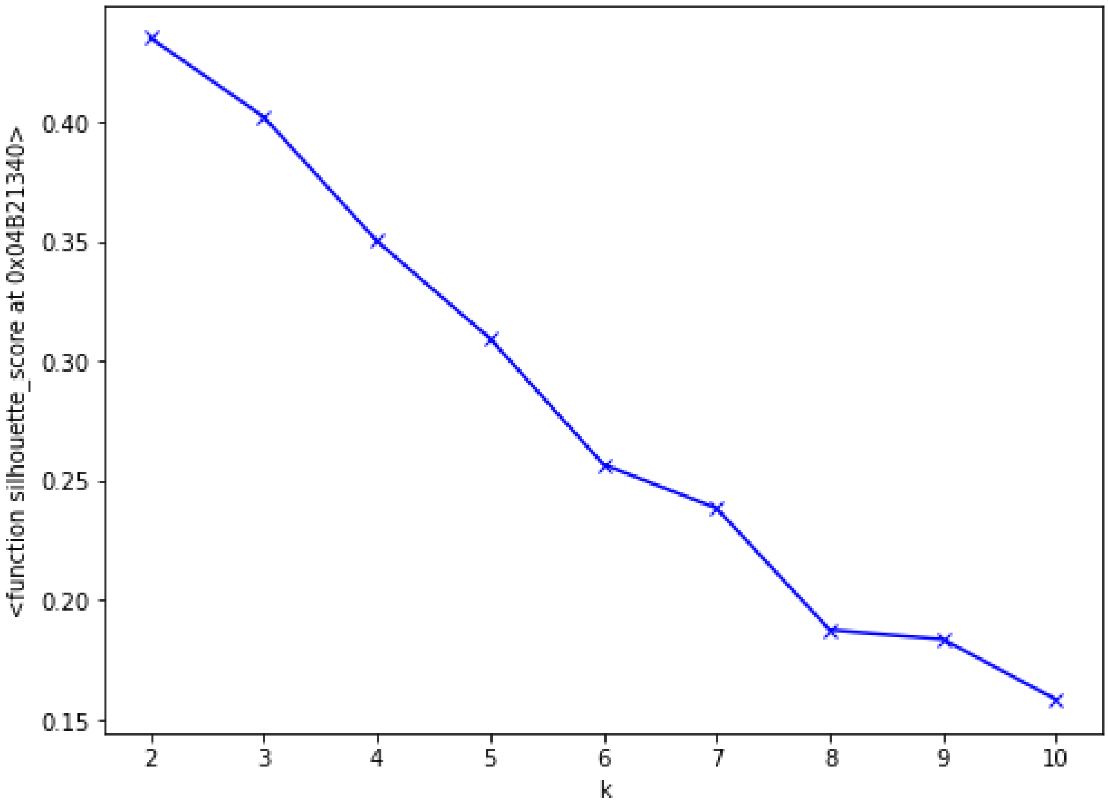
```python
KMeansElbow(X, 10)
```



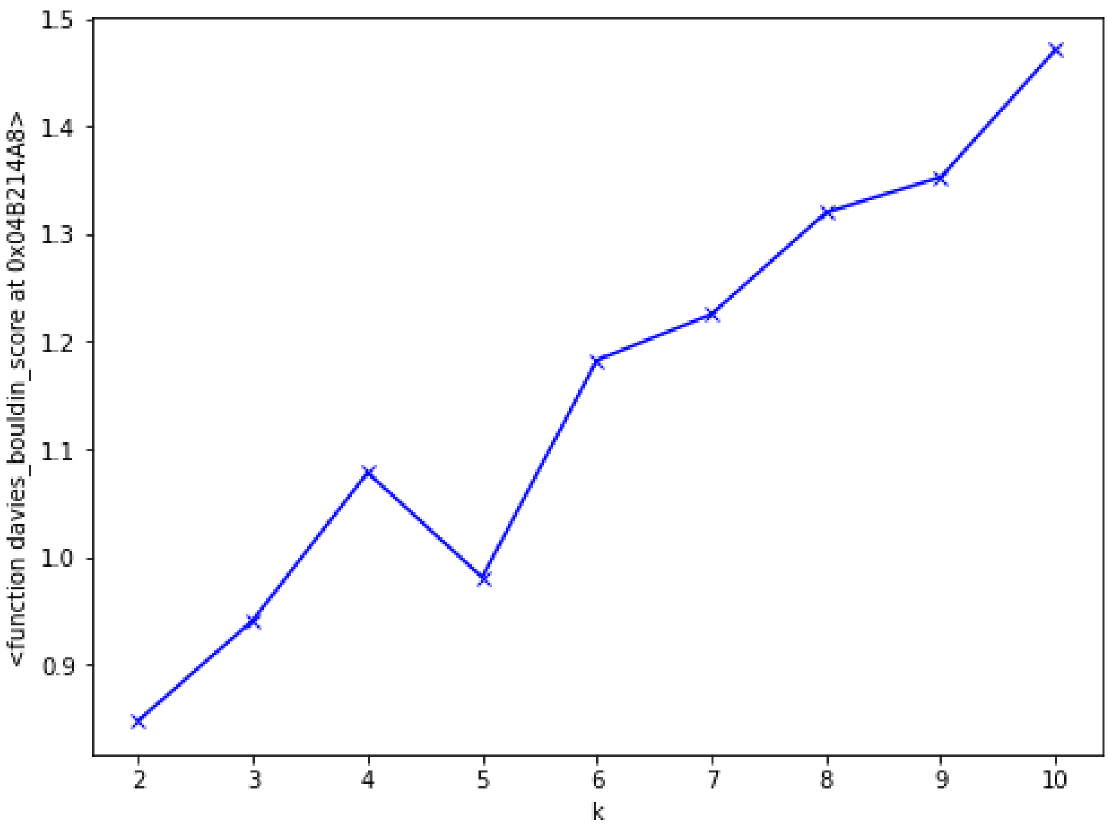
[125]:
```
from sklearn.metrics import silhouette_score, davies_bouldin_score, rand_score,␣
↪adjusted_mutual_info_score, mutual_info_score
```

[69]:
```
#metoda silhouette

def silhouetteClusterNum(X, cluster_num, score_fun):
    scores = []
    for k in range(2, cluster_num+1):
        model_instance = KMeans(n_clusters=k, random_state=0)
        labels = model_instance.fit_predict(X)
        wcss = score_fun(X, labels)
        scores.append(wcss)

    f = plt.figure(figsize=[8, 6])
    plt.plot(range(2, cluster_num+1), scores, 'bx-')
    plt.xlabel('k')
    plt.ylabel(f'{score_fun}')
    plt.show()
```

[70]:
```
silhouetteClusterNum(X, 10, silhouette_score)
#im większy wynik tym lepiej
```



[72]:
```
silhouetteClusterNum(X, 10, davies_bouldin_score)
#im mniejszy wybik tym lepiej
```



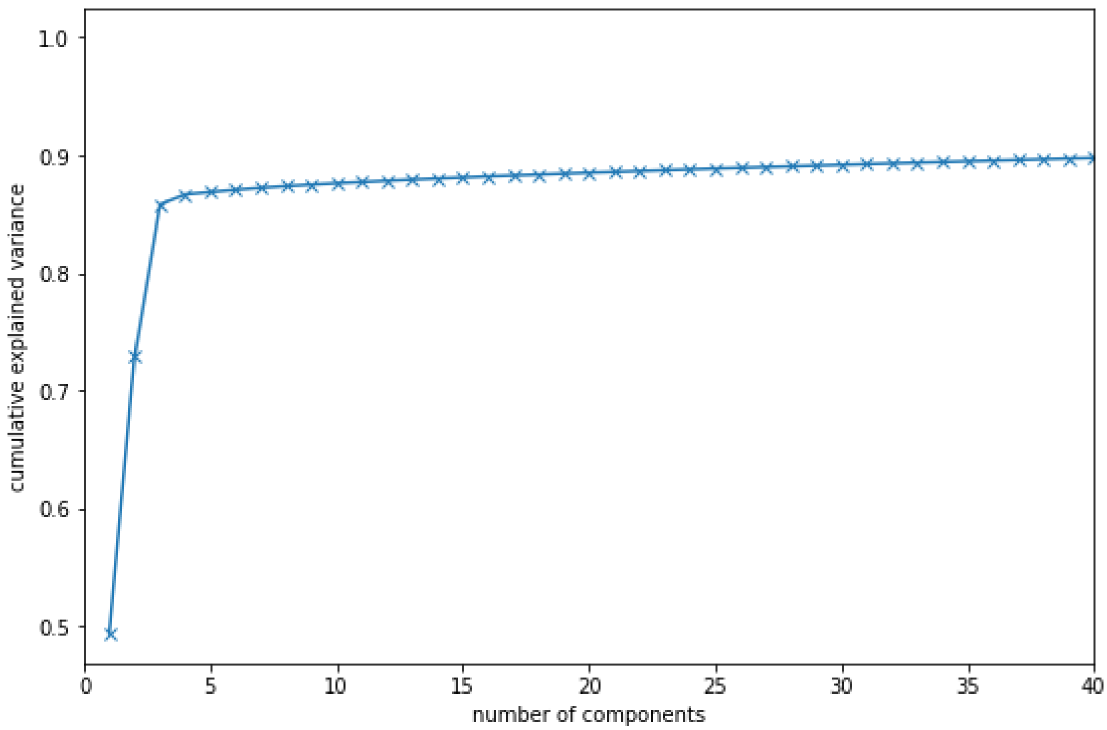
Biorąc pod uwagę wyniki różnych metryk sprawdzimy podział na 2, 3, 4 i 5 klastrów.

2.2 Stworzenie ramek z redukcją wymiarów

2.2.1 PCA

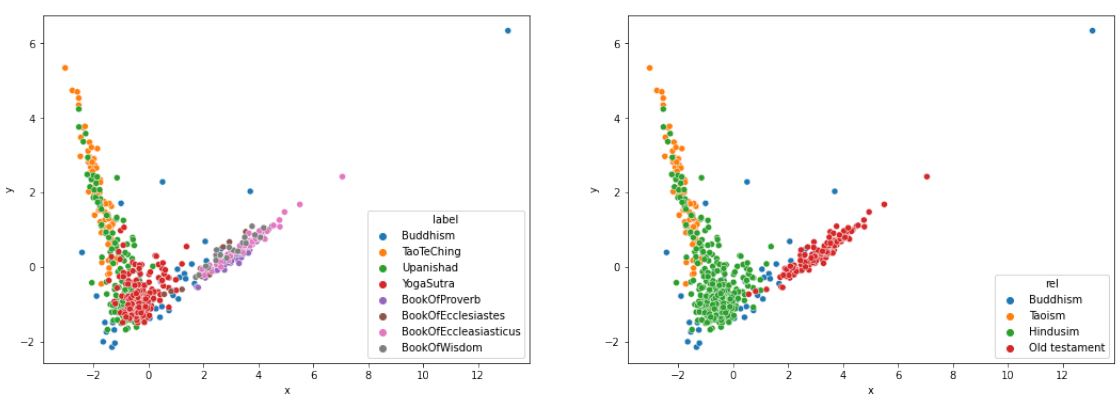
[73]:
```
from sklearn.decomposition import PCA

pca = PCA().fit(X)

plt.figure(figsize=(9,6))
plt.plot(range(1, len(pca.explained_variance_ratio_)+1), np.cumsum(pca.
 ↪explained_variance_ratio_),marker='x')
plt.xlabel('number of components')
plt.xlim(0, 40)
plt.ylabel('cumulative explained variance')
```

[73]:
```
Text(0, 0.5, 'cumulative explained variance')
```



dla 3 zmiennych mamy 85% wariancji

[94]:
```
X_pca2 = PCA(n_components=2).fit_transform(X)
X_pca2 = pd.DataFrame({'x': X_pca2[:, 0], 'y': X_pca2[:, 1], 'label':
 ↪Y['label'], 'rel': Y['rel']})

f, (ax1, ax2) = plt.subplots(1, 2, figsize=[18, 6])
sns.scatterplot(data=X_pca2, x='x', y='y', hue='label', ax = ax1)
sns.scatterplot(data=X_pca2, x='x', y='y', hue='rel', ax = ax2)
plt.show()
```



[101]:
```
import plotly.graph_objs as go
from sklearn import preprocessing

X_pca3 = PCA(n_components=3).fit_transform(X)
le = preprocessing.LabelEncoder()

Scene = dict(xaxis = dict(title  = 'PC1'),yaxis = dict(title  = 'PC2'),zaxis =␣
 ↪dict(title  = 'PC3'))
labels = le.fit_transform(Y['label'])
trace = go.Scatter3d(x=X_pca3[:,0], y=X_pca3[:,1], z=X_pca3[:,2],␣
 ↪mode='markers',marker=dict(color = labels, size = 10, line = dict(color =␣
 ↪'gray',width = 5)))
layout = go.Layout(margin=dict(l=0,r=0),scene = Scene, height = 600,width = 600)
data = [trace]
fig = go.Figure(data = data, layout = layout)
fig.show()
```

[102]:
```
Scene = dict(xaxis = dict(title  = 'PC1'),yaxis = dict(title  = 'PC2'),zaxis =␣
 ↪dict(title  = 'PC3'))
labels = le.fit_transform(Y['rel'])
trace = go.Scatter3d(x=X_pca3[:,0], y=X_pca3[:,1], z=X_pca3[:,2],␣
 ↪mode='markers',marker=dict(color = labels, size = 10, line = dict(color =␣
 ↪'gray',width = 5)))
layout = go.Layout(margin=dict(l=0,r=0),scene = Scene, height = 600,width = 600)
data = [trace]
fig = go.Figure(data = data, layout = layout)
fig.show()
```

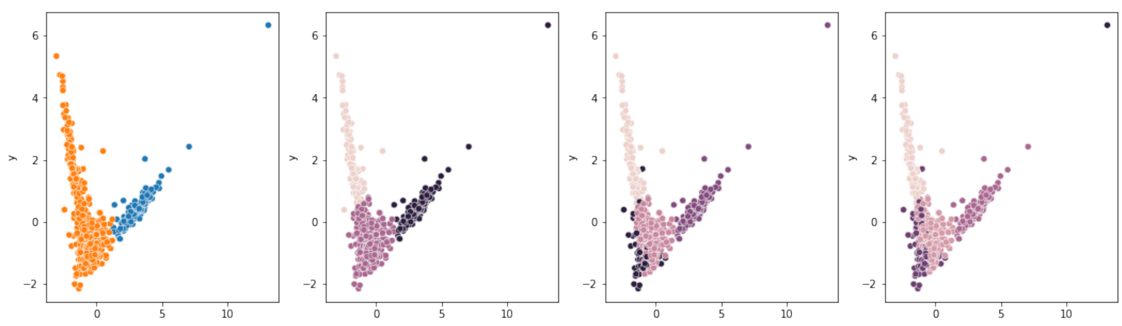
2.3 Klasteryzacja bez redukcji wymiarów, ale zwizualizowana na PCA

[137]:
```
def KMeansClustering(data, reduction, actual_labels):
    results = pd.DataFrame(columns = ['clusters', 'silhouette_score',␣
 ↪'davies_bouldin_score',
                                      'rand_score',␣
 ↪'adjusted_mutual_info_score', 'mutual_info_score'])

    fig, axs = plt.subplots(1, 4, figsize = (18, 5))

    for i in range(2, 6):
        kmeans = KMeans(n_clusters=i, random_state=0)
        kmeans.fit(data)
        y_kmeans = kmeans.predict(data)

        i_results = pd.DataFrame({'clusters':[i],
```

```
                                        'silhouette_score':[silhouette_score(data,␣
↪y_kmeans)],
                                        'davies_bouldin_score':
↪[davies_bouldin_score(data, y_kmeans)],
                                        'rand_score':[rand_score(actual_labels,␣
↪y_kmeans)],
                                        'adjusted_mutual_info_score':
↪[adjusted_mutual_info_score(actual_labels, y_kmeans)],
                                        'mutual_info_score':
↪[mutual_info_score(actual_labels, y_kmeans)]})
        results = pd.concat([results, i_results])

        sns.scatterplot(data = reduction, x = 'x', y = 'y',
                        hue = y_kmeans, legend = False,
                        ax = axs[i-2])
        ax1.set_title(f'{i} clusters')


    plt.show()
    return results
```

[138]:

```
KMeansClustering(X, X_pca2, Y['label'])
```

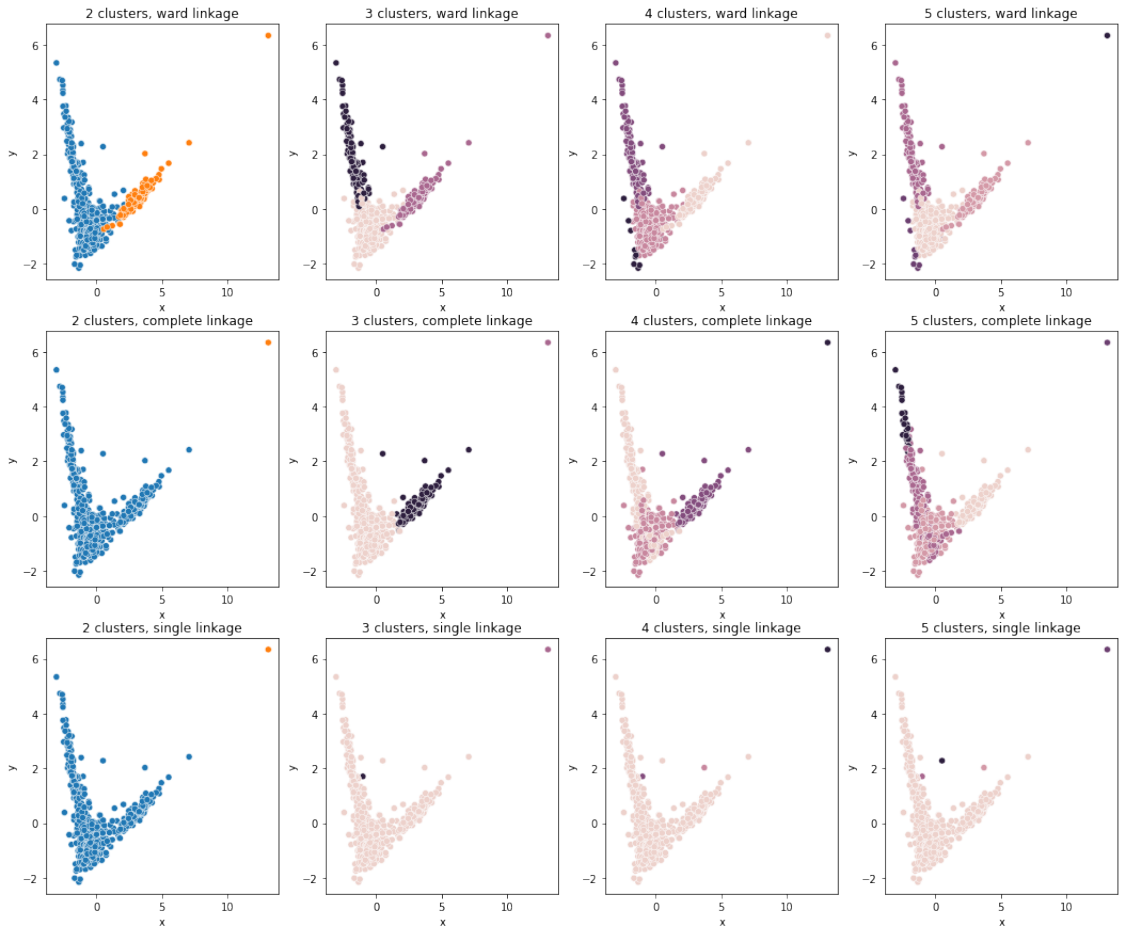


[138]:

```
   clusters  silhouette_score  davies_bouldin_score  rand_score  \
0         2          0.435288              0.847755    0.530972
0         3          0.402285              0.940976    0.690133
0         4          0.350284              1.078130    0.729878
0         5          0.309429              0.980741    0.736819

   adjusted_mutual_info_score  mutual_info_score
0                    0.389753           0.442232
0                    0.475444           0.643050
0                    0.467164           0.690380
0                    0.448753           0.686280
```

[140]:
```
def AggClustering(data, reduction, actual_labels):
    results = pd.DataFrame(columns = ['clusters', 'linkage',␣
↪'silhouette_score', 'davies_bouldin_score',
                                      'rand_score',␣
↪'adjusted_mutual_info_score', 'mutual_info_score'])

    fig, axs = plt.subplots(3, 4, figsize = (18, 15))
    linkage = ['ward', 'complete', 'single']

    for j in range(3):
        for i in range(2, 6):
            aggClus = AgglomerativeClustering(n_clusters = i, linkage =␣
↪linkage[j])
            y_aggClus = aggClus.fit_predict(data)

            i_results = pd.DataFrame({'clusters':[i],
                                     'linkage':[linkage[j]],
                                     'silhouette_score':[silhouette_score(data,␣
↪y_aggClus)],
                                     'davies_bouldin_score':
↪[davies_bouldin_score(data, y_aggClus)],
                                     'rand_score':[rand_score(actual_labels,␣
↪y_aggClus)],
                                     'adjusted_mutual_info_score':
↪[adjusted_mutual_info_score(actual_labels, y_aggClus)],
                                     'mutual_info_score':
↪[mutual_info_score(actual_labels, y_aggClus)]})
            results = pd.concat([results, i_results])

            sns.scatterplot(data = reduction, x = 'x', y = 'y',
                            hue = y_aggClus, legend = False,
                            ax = axs[j, i-2])
            axs[j, i-2].set_title(f'{i} clusters, {linkage[j]} linkage')

    plt.show()
    return results
```

[141]:
```
AggClustering(X, X_pca2, Y['label'])
```



[141]:

	clusters	linkage	silhouette_score	davies_bouldin_score	rand_score
0	2	ward	0.430679	0.838535	0.524284
0	3	ward	0.376586	0.994408	0.703289
0	4	ward	0.376948	0.922601	0.720256
0	5	ward	0.379742	0.758135	0.720894
0	2	complete	0.779682	0.152003	0.215867
0	3	complete	0.430162	0.594429	0.525867
0	4	complete	0.185398	1.317981	0.665800
0	5	complete	0.180182	1.241147	0.684354
0	2	single	0.779682	0.152003	0.215867
0	3	single	0.243739	0.465091	0.218745
0	4	single	0.172431	0.463161	0.221628
0	5	single	0.108968	0.556775	0.224517

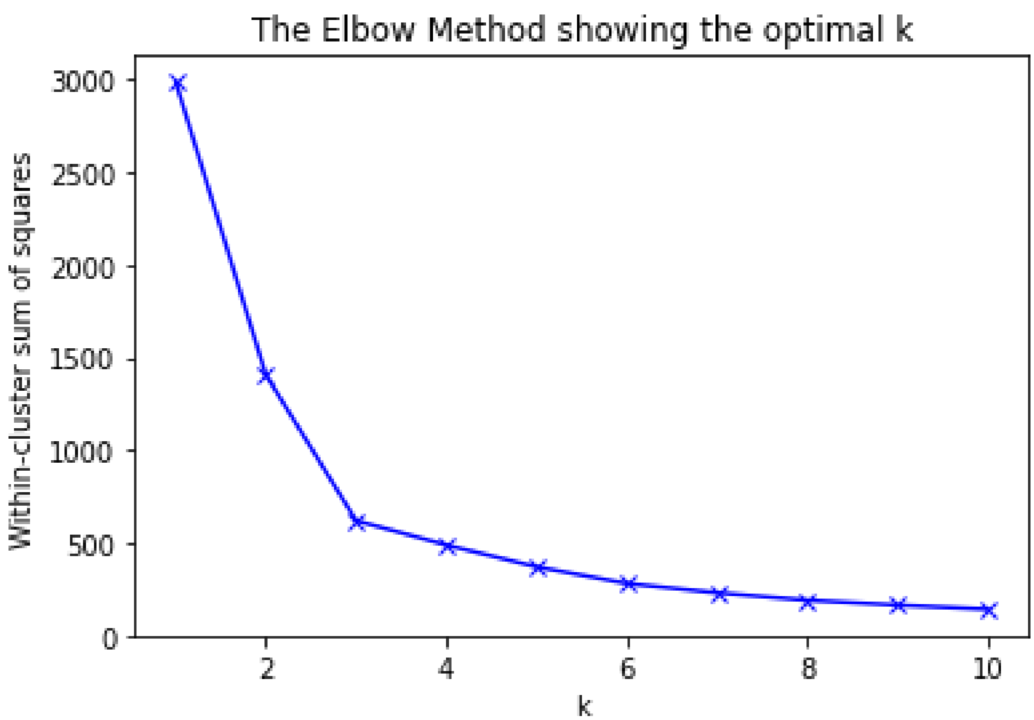
\

	adjusted_mutual_info_score	mutual_info_score
0	0.423434	0.476915
0	0.512000	0.698181
0	0.516778	0.730515

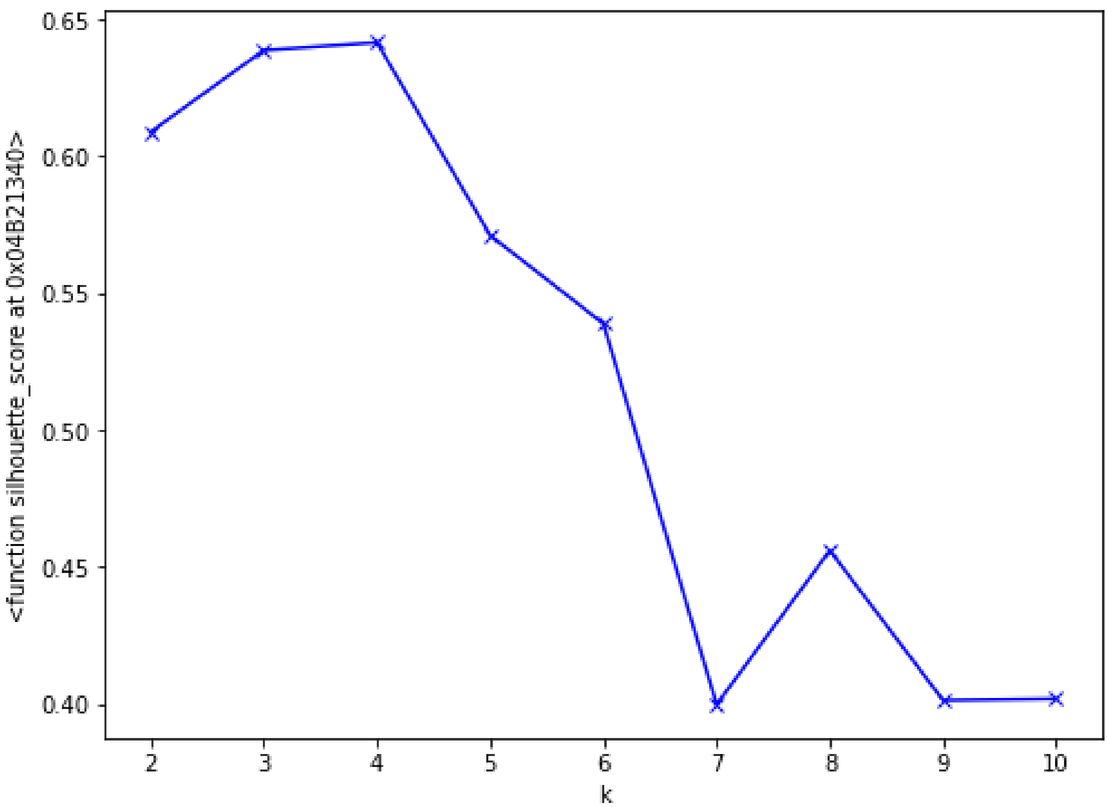
0	0.519257	0.737880
0	0.001569	0.004342
0	0.390862	0.445355
0	0.391383	0.554652
0	0.388643	0.585214
0	0.001569	0.004342
0	0.003166	0.008718
0	0.004791	0.013130
0	0.006446	0.017579

2.4 Klastrowanie po PCA

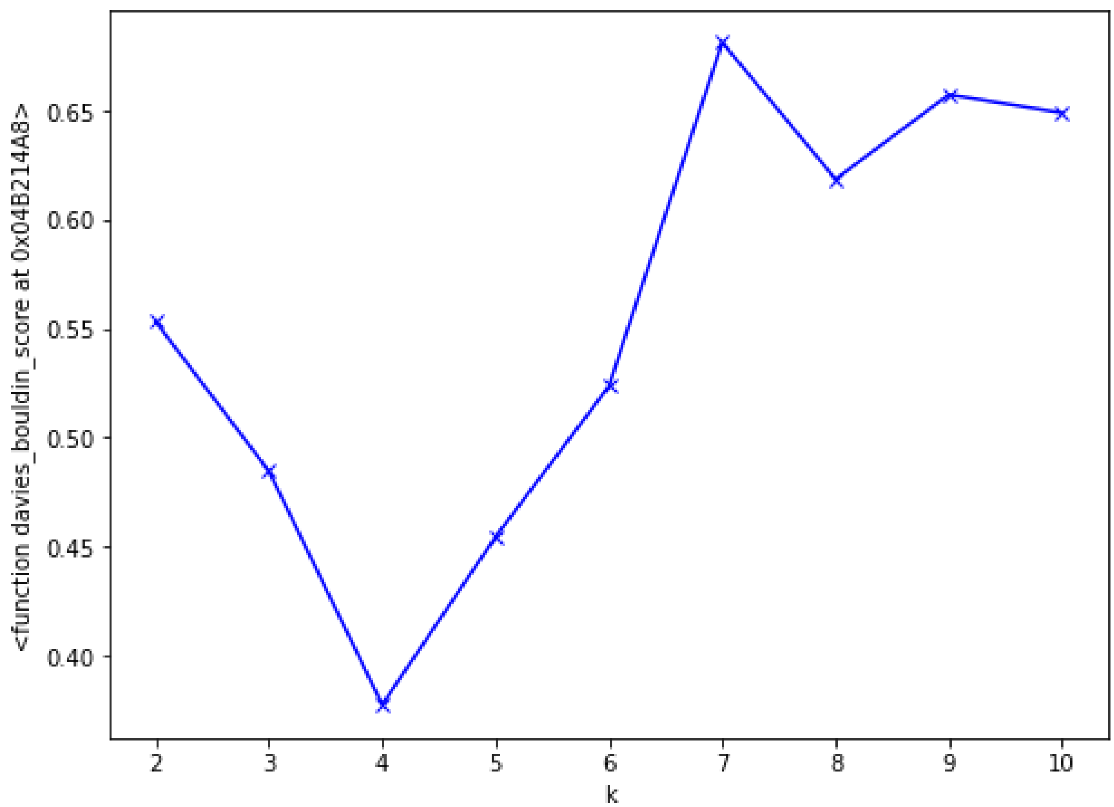
[144]:
```
KMeansElbow(X_pca2[['x','y']], 10)
```



[146]:
```
silhouetteClusterNum(X_pca2[['x','y']], 10, silhouette_score)
```

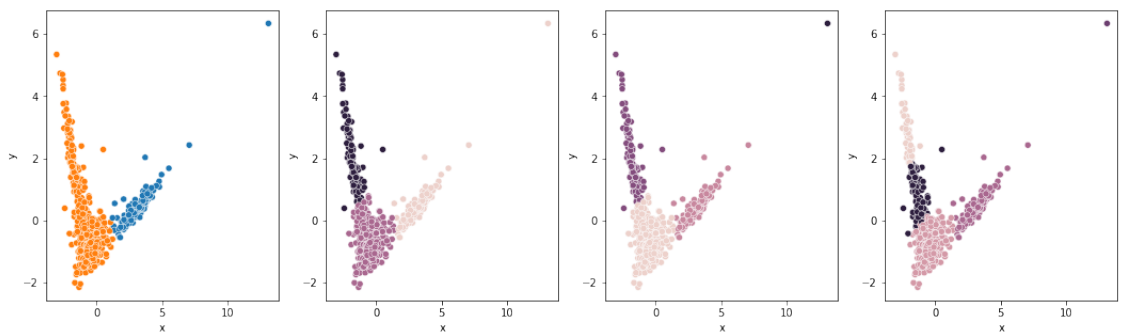


[147]:
```
silhouetteClusterNum(X_pca2[['x','y']], 10, davies_bouldin_score)
```



Po PCA dalej wygląda na to, że będziemy szukac tej samej liczby klastrów

[148]:
```
KMeansClustering(X_pca2[['x', 'y']], X_pca2, Y['label'])
```



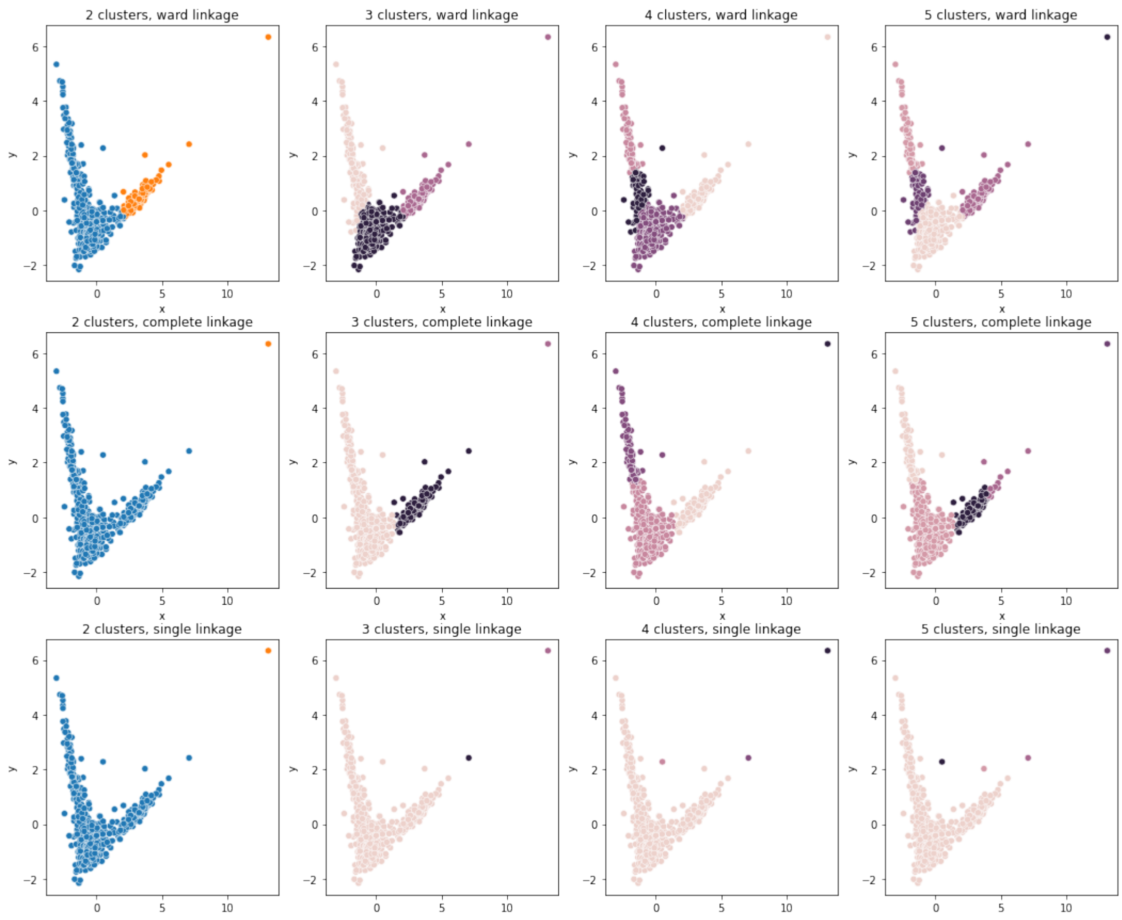
[148]:

	clusters	silhouette_score	davies_bouldin_score	rand_score	\
0	2	0.608793	0.553328	0.530839	
0	3	0.638631	0.484705	0.690133	
0	4	0.641498	0.377479	0.690731	
0	5	0.571064	0.454516	0.718069	

	adjusted_mutual_info_score	mutual_info_score
0	0.384076	0.436321
0	0.475444	0.643050
0	0.476209	0.647944
0	0.475098	0.701041

[149]:

```
AggClustering(X_pca2[['x', 'y']], X_pca2, Y['label'])
```



[149]:

	clusters	linkage	silhouette_score	davies_bouldin_score	rand_score	\
0	2	ward	0.604723	0.510290	0.495445	
0	3	ward	0.585826	0.553548	0.698478	
0	4	ward	0.511152	0.607908	0.702564	
0	5	ward	0.513673	0.495200	0.703105	
0	2	complete	0.817598	0.126704	0.215867	
0	3	complete	0.607592	0.380336	0.526477	
0	4	complete	0.626451	0.360984	0.645380	
0	5	complete	0.569905	0.473947	0.649501	

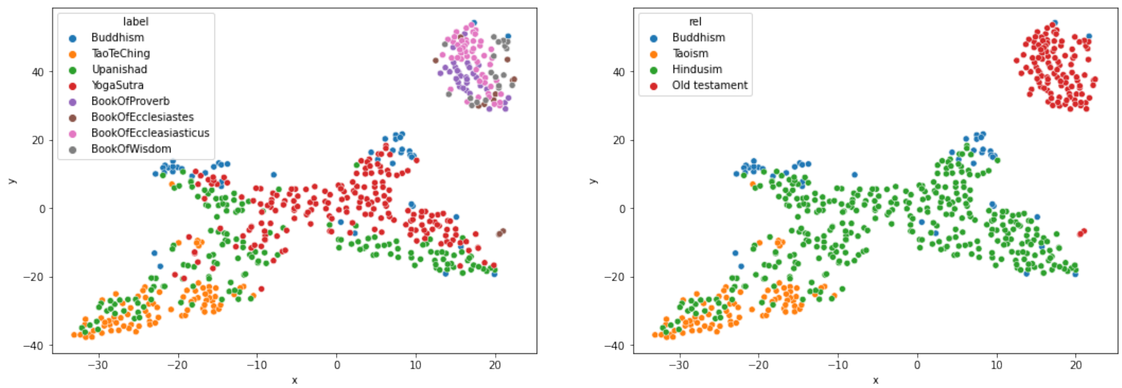
```
0          2     single           0.817598             0.126704     0.215867
0          3     single           0.618018             0.204883     0.218687
0          4     single           0.186662             0.480402     0.221559
0          5     single           0.122767             0.467895     0.224437

   adjusted_mutual_info_score  mutual_info_score
0                    0.332557           0.371286
0                    0.457913           0.627631
0                    0.436472           0.639887
0                    0.437556           0.645189
0                    0.001569           0.004342
0                    0.392522           0.447676
0                    0.441700           0.584799
0                    0.431711           0.595184
0                    0.001569           0.004342
0                    0.002960           0.008538
0                    0.004543           0.012911
0                    0.006154           0.017320
```

3 T-SNE

[150]:
```python
from sklearn.manifold import TSNE

plt.figure(figsize=[10, 8])
tSNE = TSNE(random_state=0, verbose=1)
X_tsne = tSNE.fit_transform(X)
X_tsne = pd.DataFrame({'x': X_tsne[:, 0], 'y': X_tsne[:, 1], 'label':
 ↪Y['label'], 'rel' : Y['rel']})

f, (ax1, ax2) = plt.subplots(1, 2, figsize=[18, 6])
sns.scatterplot(data=X_tsne, x='x', y='y', hue='label', ax = ax1)
sns.scatterplot(data=X_tsne, x='x', y='y', hue='rel', ax = ax2)
plt.show()
```

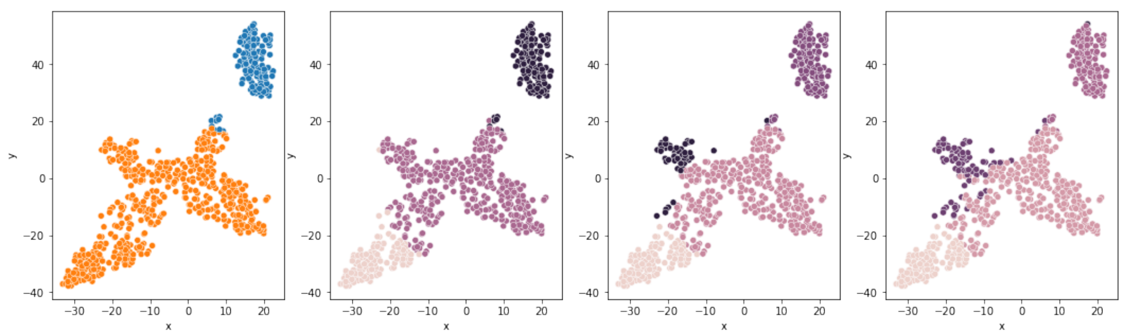
```
[t-SNE] Computing 91 nearest neighbors…
[t-SNE] Indexed 590 samples in 0.010s…
[t-SNE] Computed neighbors for 590 samples in 0.182s…
[t-SNE] Computed conditional probabilities for sample 590 / 590
[t-SNE] Mean sigma: 0.446972
[t-SNE] KL divergence after 250 iterations with early exaggeration: 60.796814
[t-SNE] KL divergence after 1000 iterations: 0.550892
```

```
<Figure size 720x576 with 0 Axes>
```



3.1 Klasteryzacja bez redukcji wymiarów, wizualizacja na T-SNE

[151]:
```
KMeansClustering(X, X_tsne, Y['label'])
```

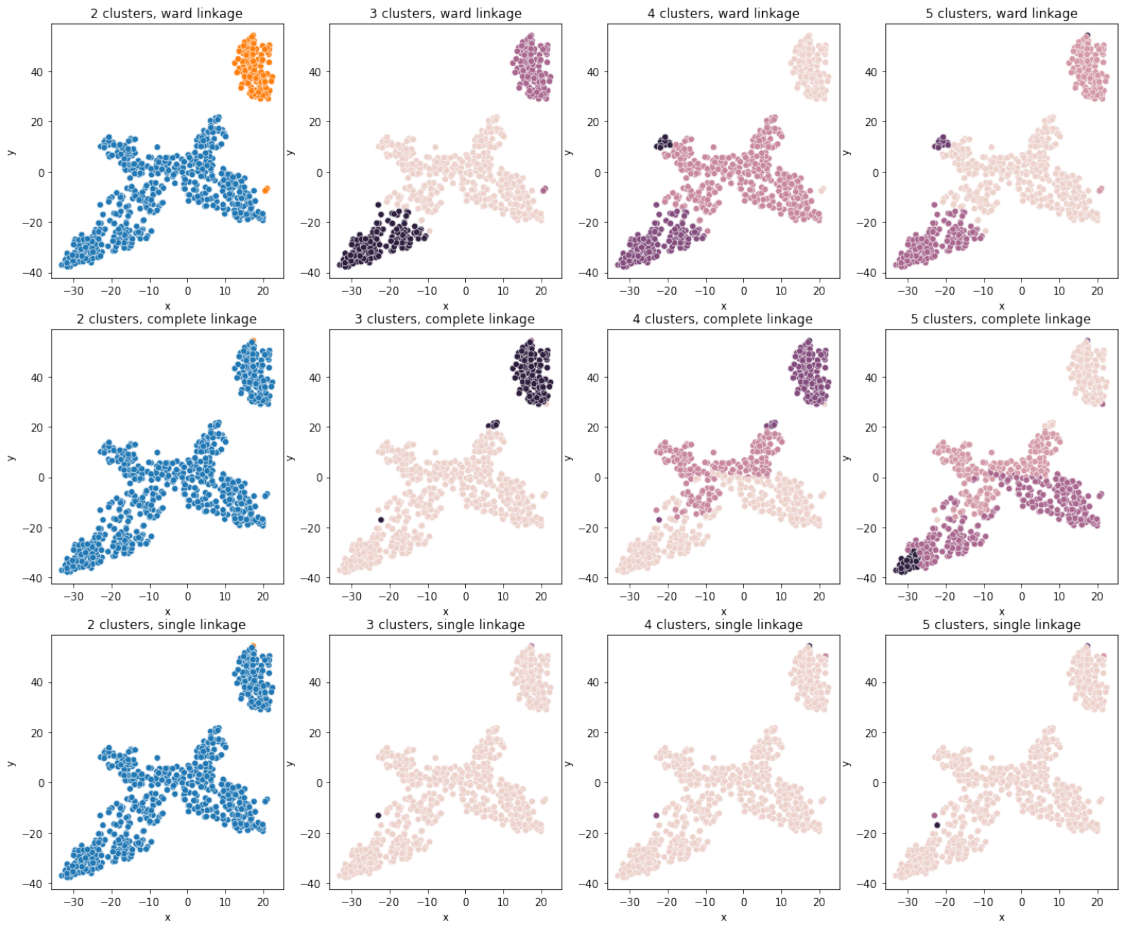


[151]:

	clusters	silhouette_score	davies_bouldin_score	rand_score	\
0	2	0.435288	0.847755	0.530972	
0	3	0.402285	0.940976	0.690133	
0	4	0.350284	1.078130	0.729878	
0	5	0.309429	0.980741	0.736819	

	adjusted_mutual_info_score	mutual_info_score
0	0.389753	0.442232
0	0.475444	0.643050
0	0.467164	0.690380
0	0.448753	0.686280

[152]:
```
AggClustering(X, X_tsne, Y['label'])
```



[152]:

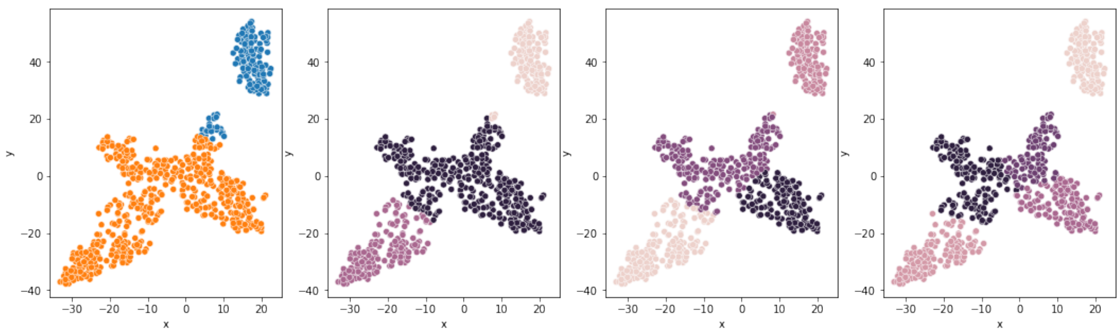
	clusters	linkage	silhouette_score	davies_bouldin_score	rand_score \
0	2	ward	0.430679	0.838535	0.524284
0	3	ward	0.376586	0.994408	0.703289
0	4	ward	0.376948	0.922601	0.720256
0	5	ward	0.379742	0.758135	0.720894
0	2	complete	0.779682	0.152003	0.215867
0	3	complete	0.430162	0.594429	0.525867
0	4	complete	0.185398	1.317981	0.665800
0	5	complete	0.180182	1.241147	0.684354
0	2	single	0.779682	0.152003	0.215867
0	3	single	0.243739	0.465091	0.218745
0	4	single	0.172431	0.463161	0.221628
0	5	single	0.108968	0.556775	0.224517

	adjusted_mutual_info_score	mutual_info_score
0	0.423434	0.476915
0	0.512000	0.698181
0	0.516778	0.730515

```
0                  0.519257            0.737880
0                  0.001569            0.004342
0                  0.390862            0.445355
0                  0.391383            0.554652
0                  0.388643            0.585214
0                  0.001569            0.004342
0                  0.003166            0.008718
0                  0.004791            0.013130
0                  0.006446            0.017579
```

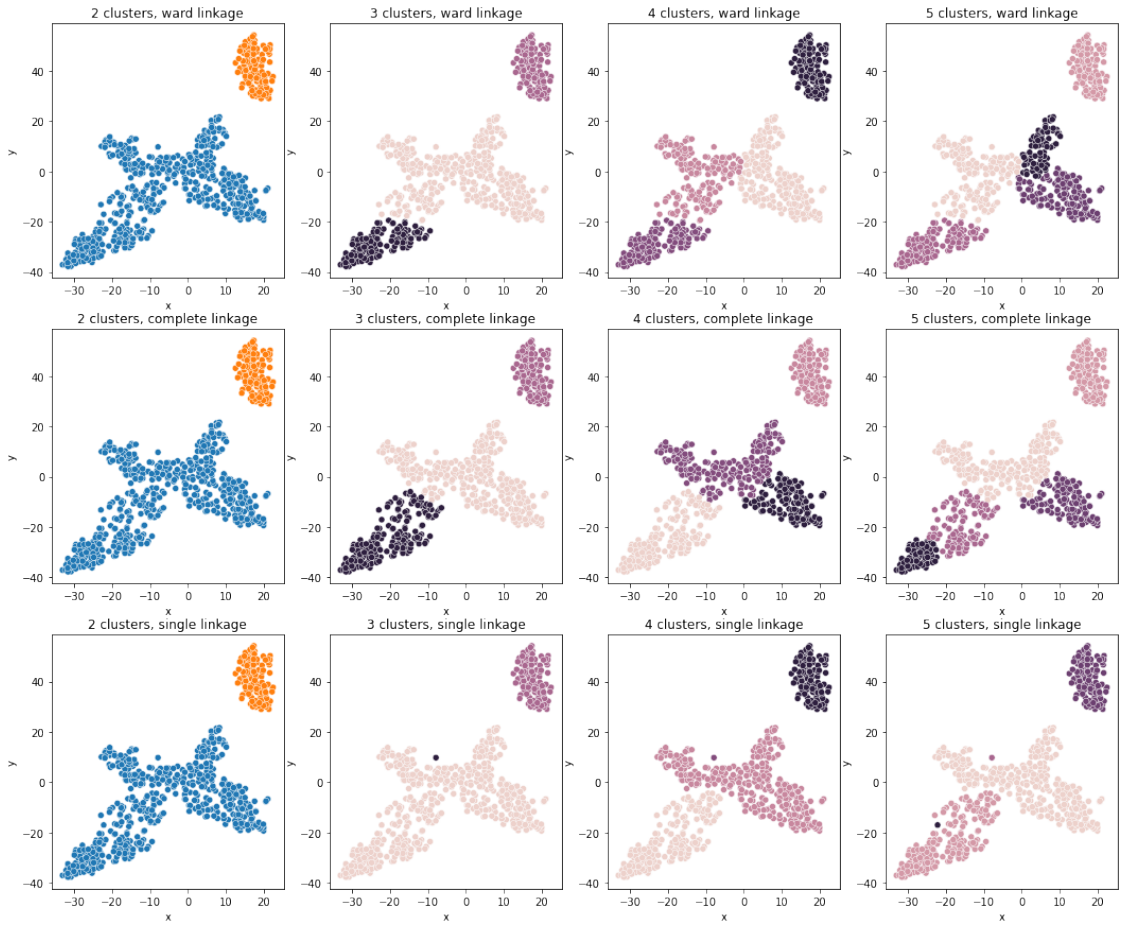
3.2 Klasteryzacja po T-SNE

[153]:
```
KMeansClustering(X_tsne[['x', 'y']], X_tsne, Y['label'])
```



[153]:
```
   clusters  silhouette_score  davies_bouldin_score  rand_score  \
0         2          0.559706              0.536397    0.542154
0         3          0.542620              0.610013    0.715444
0         4          0.544692              0.656311    0.744865
0         5          0.537984              0.666933    0.755834

   adjusted_mutual_info_score  mutual_info_score
0                    0.365356           0.420147
0                    0.502009           0.693618
0                    0.458577           0.722163
0                    0.449276           0.764575
```

[154]:
```
AggClustering(X_tsne[['x', 'y']], X_tsne, Y['label'])
```



[154]:

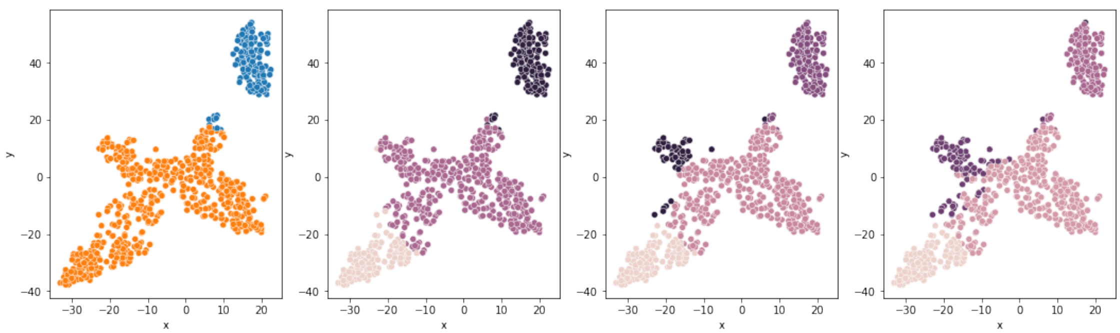
	clusters	linkage	silhouette_score	davies_bouldin_score	rand_score	\
0	2	ward	0.578043	0.455887	0.517671	
0	3	ward	0.532381	0.563287	0.695520	
0	4	ward	0.519822	0.724215	0.737026	
0	5	ward	0.524075	0.633958	0.750908	
0	2	complete	0.578043	0.455887	0.517671	
0	3	complete	0.534083	0.644520	0.710080	
0	4	complete	0.542494	0.641761	0.743599	
0	5	complete	0.498207	0.662969	0.751581	
0	2	single	0.578043	0.455887	0.517671	
0	3	single	0.075305	0.820491	0.519927	
0	4	single	0.267056	0.800274	0.702086	
0	5	single	0.065983	1.154284	0.703122	

	adjusted_mutual_info_score	mutual_info_score
0	0.407878	0.458127
0	0.513918	0.695575
0	0.450797	0.709817

0	0.451712	0.764074
0	0.407878	0.458127
0	0.487483	0.678167
0	0.455785	0.716834
0	0.440178	0.738361
0	0.407878	0.458127
0	0.407724	0.462191
0	0.465491	0.655111
0	0.468494	0.663281

4 Wyniki w porównaniu do religii a nie konkretnych ksiąg

[165]:
```
#bez redukcji wymiarów
KMeansClustering(X, X_tsne, Y['rel'])
```

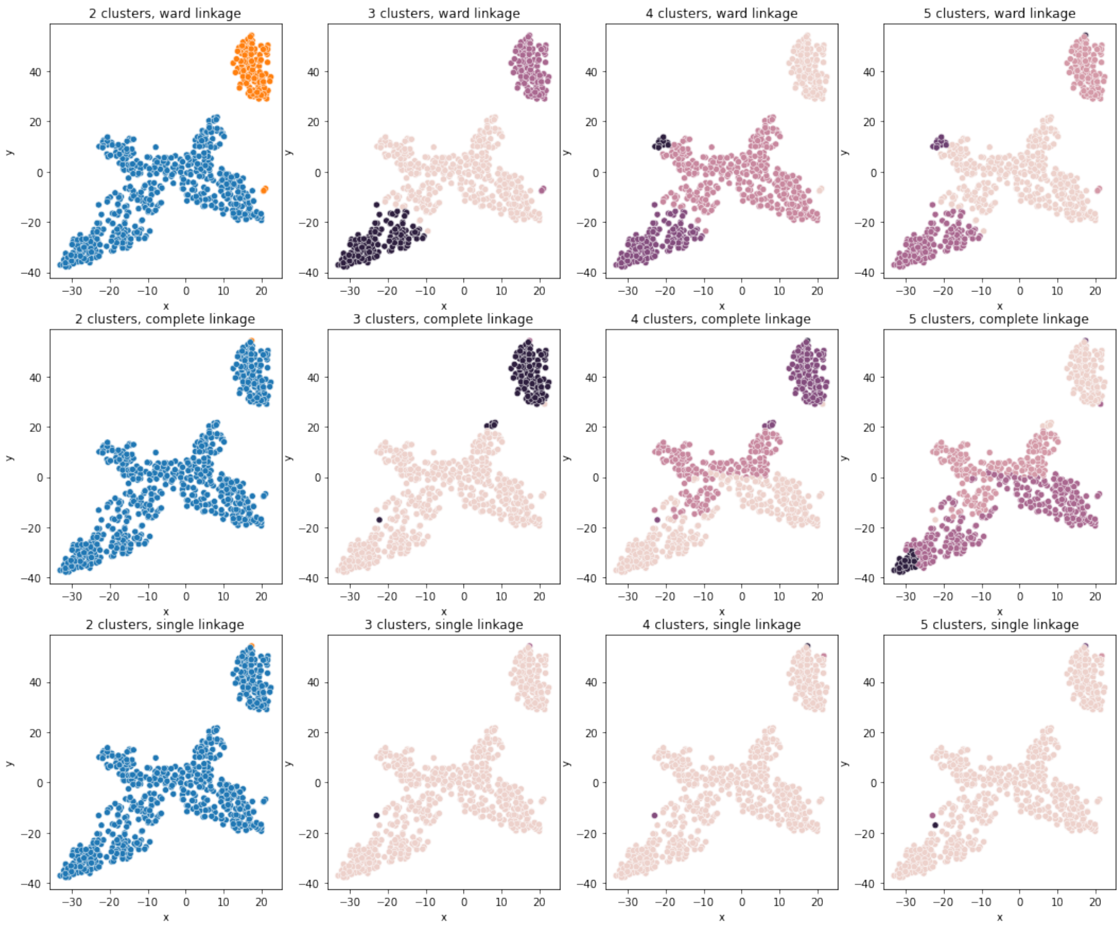


[165]:

	clusters	silhouette_score	davies_bouldin_score	rand_score	\
0	2	0.435288	0.847755	0.726552	
0	3	0.402285	0.940976	0.807856	
0	4	0.350284	1.078130	0.807798	
0	5	0.309429	0.980741	0.780070	

	adjusted_mutual_info_score	mutual_info_score
0	0.534664	0.429196
0	0.581368	0.591890
0	0.561967	0.639466
0	0.536186	0.636204

[166]:
```
#bez redukcji wymiarów
AggClustering(X, X_tsne, Y['rel'])
```



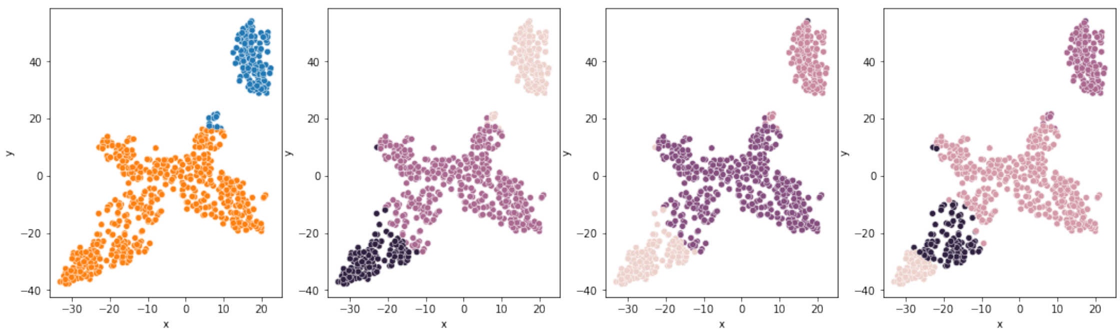
[166]:

	clusters	linkage	silhouette_score	davies_bouldin_score	rand_score	\
0	2	ward	0.430679	0.838535	0.725182	
0	3	ward	0.376586	0.994408	0.800892	
0	4	ward	0.376948	0.922601	0.813600	
0	5	ward	0.379742	0.758135	0.814238	
0	2	complete	0.779682	0.152003	0.416765	
0	3	complete	0.430162	0.594429	0.722448	
0	4	complete	0.185398	1.317981	0.677166	
0	5	complete	0.180182	1.241147	0.688078	
0	2	single	0.779682	0.152003	0.416765	
0	3	single	0.243739	0.465091	0.419643	
0	4	single	0.172431	0.463161	0.422526	
0	5	single	0.108968	0.556775	0.425415	

	adjusted_mutual_info_score	mutual_info_score
0	0.599808	0.476915
0	0.632098	0.651302
0	0.632121	0.680386

0	0.635506	0.687751
0	0.004490	0.004342
0	0.540788	0.435031
0	0.484847	0.521898
0	0.470558	0.544888
0	0.004490	0.004342
0	0.008971	0.008718
0	0.013445	0.013130
0	0.017915	0.017579

[167]:
```
#po PCA
KMeansClustering(X_pca[['x', 'y']], X_tsne, Y['rel'])
```

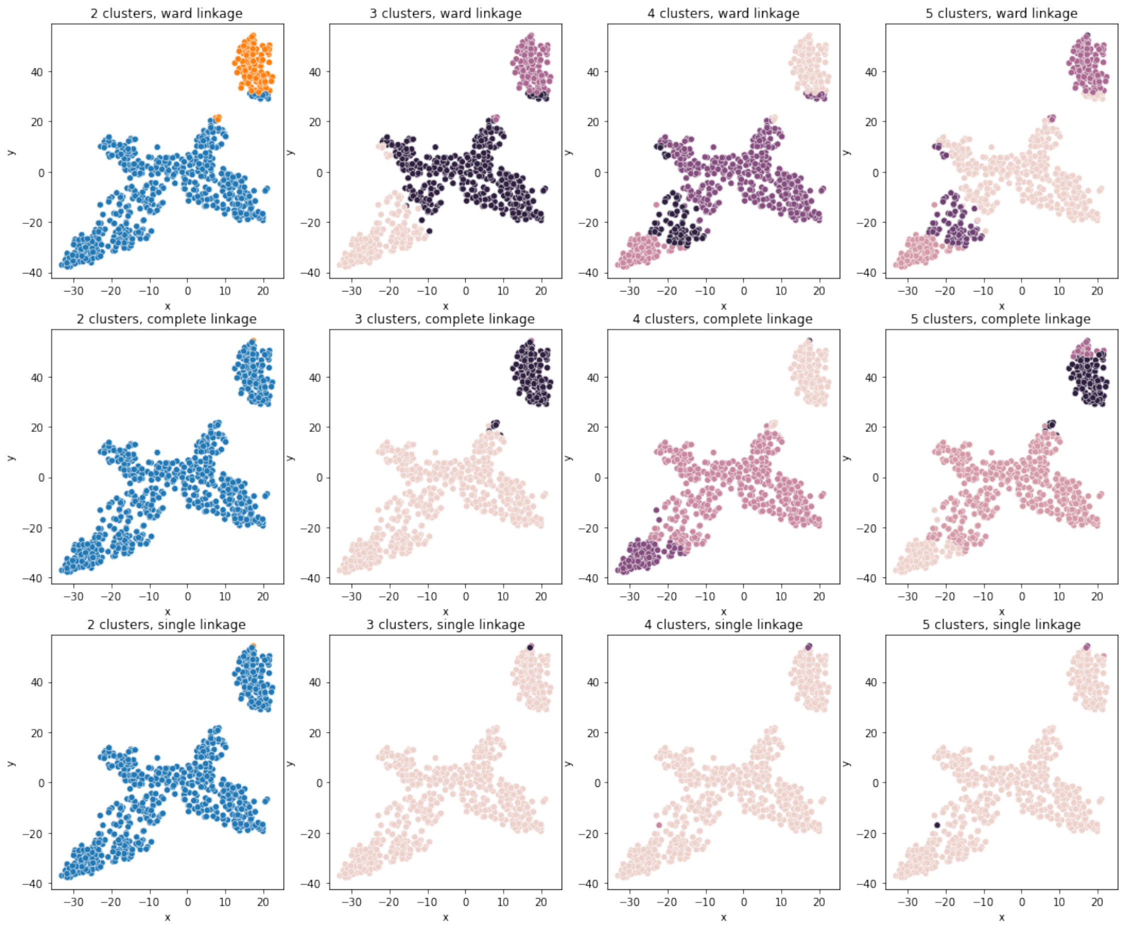


[167]:

	clusters	silhouette_score	davies_bouldin_score	rand_score	\
0	2	0.608793	0.553328	0.724555	
0	3	0.638631	0.484705	0.807856	
0	4	0.641498	0.377479	0.808454	
0	5	0.571064	0.454516	0.791891	

	adjusted_mutual_info_score	mutual_info_score
0	0.525175	0.422229
0	0.581368	0.591890
0	0.582629	0.596783
0	0.557389	0.632426

[168]:
```
#po PCA
AggClustering(X_pca[['x', 'y']], X_tsne, Y['rel'])
```



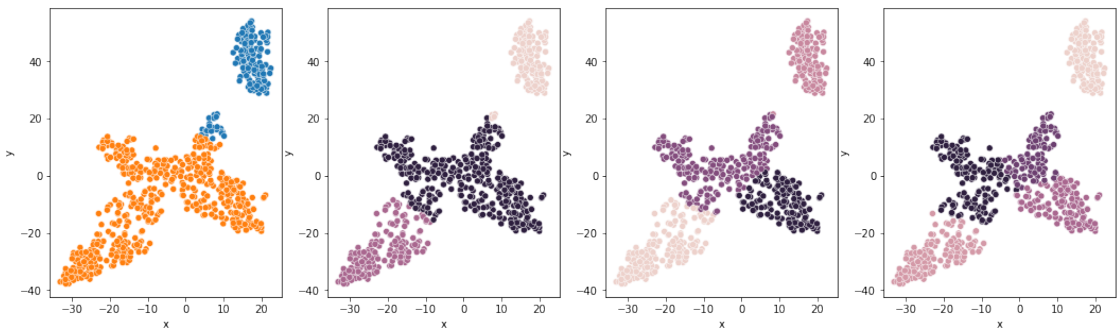
[168]:

	clusters	linkage	silhouette_score	davies_bouldin_score	rand_score	\
0	2	ward	0.604723	0.510290	0.684429	
0	3	ward	0.585826	0.553548	0.765636	
0	4	ward	0.511152	0.607908	0.767949	
0	5	ward	0.513673	0.495200	0.768490	
0	2	complete	0.817598	0.126704	0.416765	
0	3	complete	0.607592	0.380336	0.722057	
0	4	complete	0.626451	0.360984	0.789025	
0	5	complete	0.569905	0.473947	0.780334	
0	2	single	0.817598	0.126704	0.416765	
0	3	single	0.618018	0.204883	0.418871	
0	4	single	0.186662	0.480402	0.421743	
0	5	single	0.122767	0.467895	0.424621	

	adjusted_mutual_info_score	mutual_info_score
0	0.466152	0.364743
0	0.552127	0.571848
0	0.514313	0.578370

0	0.516060	0.583671
0	0.004490	0.004342
0	0.539486	0.434639
0	0.546445	0.538681
0	0.521176	0.539005
0	0.004490	0.004342
0	0.006174	0.007161
0	0.010602	0.011535
0	0.015025	0.015944

[169]:
```
#po t-sne
KMeansClustering(X_tsne[['x', 'y']], X_tsne, Y['rel'])
```

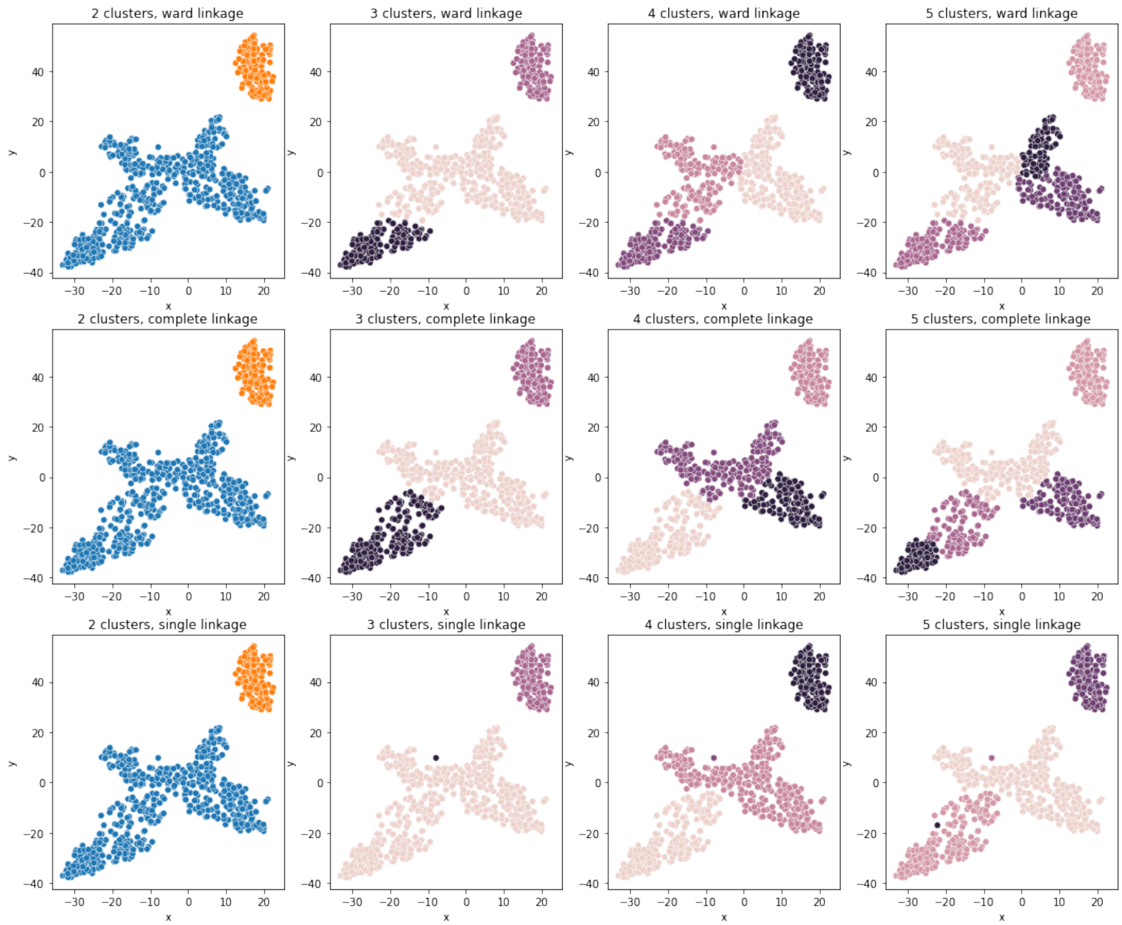


[169]:

	clusters	silhouette_score	davies_bouldin_score	rand_score \
0	2	0.559706	0.536397	0.726546
0	3	0.542620	0.610013	0.780484
0	4	0.544692	0.656311	0.693511
0	5	0.537984	0.666933	0.677857

	adjusted_mutual_info_score	mutual_info_score
0	0.490180	0.400714
0	0.598818	0.627824
0	0.518824	0.641075
0	0.474097	0.643958

[170]:
```
#po t-sne
AggClustering(X_tsne[['x', 'y']], X_tsne, Y['rel'])
```



[170]:

	clusters	linkage	silhouette_score	davies_bouldin_score	rand_score	\
0	2	ward	0.578043	0.455887	0.715116	
0	3	ward	0.532381	0.563287	0.801007	
0	4	ward	0.519822	0.724215	0.712365	
0	5	ward	0.524075	0.633958	0.688228	
0	2	complete	0.578043	0.455887	0.715116	
0	3	complete	0.534083	0.644520	0.749153	
0	4	complete	0.542494	0.641761	0.683647	
0	5	complete	0.498207	0.662969	0.687324	
0	2	single	0.578043	0.455887	0.715116	
0	3	single	0.075305	0.820491	0.717372	
0	4	single	0.267056	0.800274	0.735564	
0	5	single	0.065983	1.154284	0.736600	

	adjusted_mutual_info_score	mutual_info_score
0	0.563513	0.446142
0	0.627700	0.640471
0	0.527938	0.651719

0	0.490666	0.661175
0	0.563513	0.446142
0	0.582802	0.616275
0	0.512381	0.632030
0	0.483539	0.644171
0	0.563513	0.446142
0	0.563295	0.450206
0	0.569239	0.609460
0	0.573358	0.617630