notebook_final2

June 8, 2021

1 Projekt 2 - EDA

Mikołaj Spytek, Artur Żółkowski

W tym projekcie zajmujemy się klasteryzacją danych dotyczących aktywności użytkowników sklepu internetowego.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.cluster import AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.cluster import DBSCAN
from sklearn.metrics import calinski_harabasz_score, silhouette_score,␣
↪davies_bouldin_score, adjusted_mutual_info_score,␣
↪normalized_mutual_info_score
from sklearn.manifold import TSNE
import sklearn
import seaborn as sns
from sklearn.cluster import KMeans

import random
random.seed(42)
```

[2]:
```
data = pd.read_csv("data/online_shoppers_intention.csv")
```

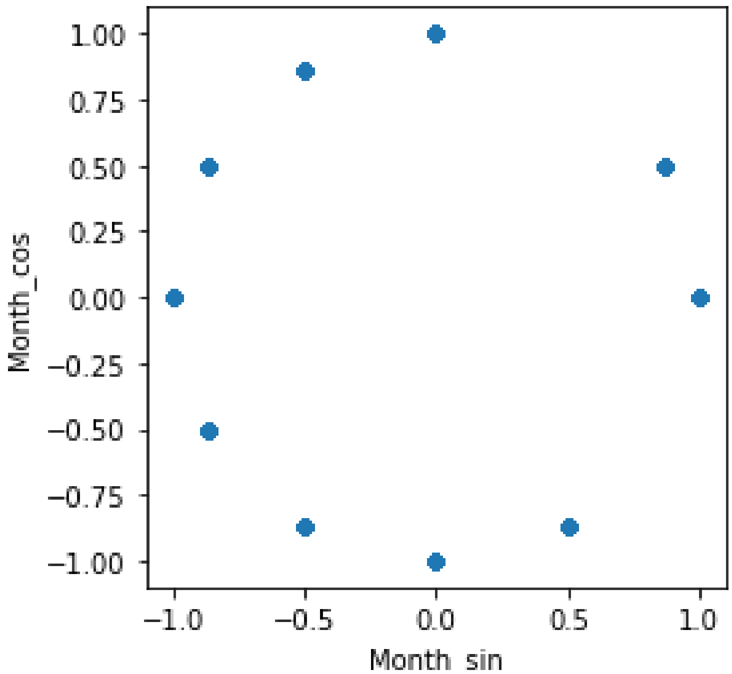
1.0.1 Przygotowanie danych

[3]:
```
def encode(data, col, max_val):
    data[col + '_sin'] = np.sin(2 * np.pi * data[col]/max_val)
    data[col + '_cos'] = np.cos(2 * np.pi * data[col]/max_val)
    return data
```

[4]:
```
months = {"Jan": 1, "Feb": 2, "Mar": 3, "Apr": 4, "May": 5, "June": 6,
          "Jul": 7, "Aug": 8, "Sep": 9, "Oct": 10, "Nov": 11, "Dec": 12}
data["Month"] = data["Month"].map(months)
```

[5]:
```
data = encode(data, 'Month', 12)
```

[6]:
```
ax = data.plot.scatter('Month_sin', 'Month_cos').set_aspect('equal')
```



[7]:
```
num_vars = ["Administrative", "Administrative_Duration", "Informational",␣
 ↪"Informational_Duration", "ProductRelated",
            "ProductRelated_Duration", "BounceRates", "ExitRates",␣
 ↪"PageValues", "SpecialDay", "Month_sin", "Month_cos"]
cat_vars = ["OperatingSystems", "Browser", "Region", "VisitorType", "Weekend",␣
 ↪"TrafficType"]
log_vars = ['Administrative', 'Administrative_Duration', 'Informational',
                'Informational_Duration', 'ProductRelated',␣
 ↪'ProductRelated_Duration',
                'BounceRates', 'ExitRates', 'PageValues']
```

[8]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer, StandardScaler,␣
 ↪OrdinalEncoder

scaler=StandardScaler()

preprocessor = ColumnTransformer(
    transformers= [
        ('log', FunctionTransformer(np.log1p), log_vars),
```

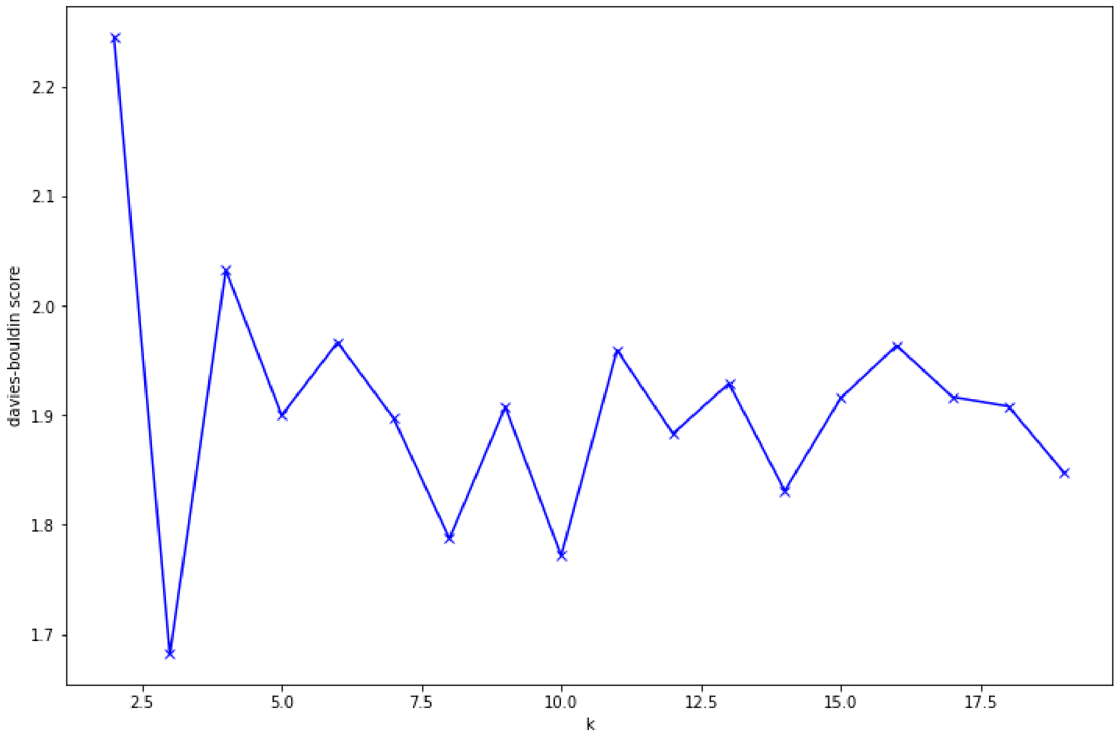
```
        ('cat', OrdinalEncoder(), cat_vars)
    ],
    remainder = 'passthrough'
)
transformed_data = preprocessor.fit_transform(data.drop(['Month', 'Revenue'],␣
↪axis=1))
```

[9]:
```
transformed_data = scaler.fit_transform(transformed_data)
transformed_data = pd.DataFrame(transformed_data, columns = data.drop(['Month',␣
↪'Revenue'], axis=1).columns)
```

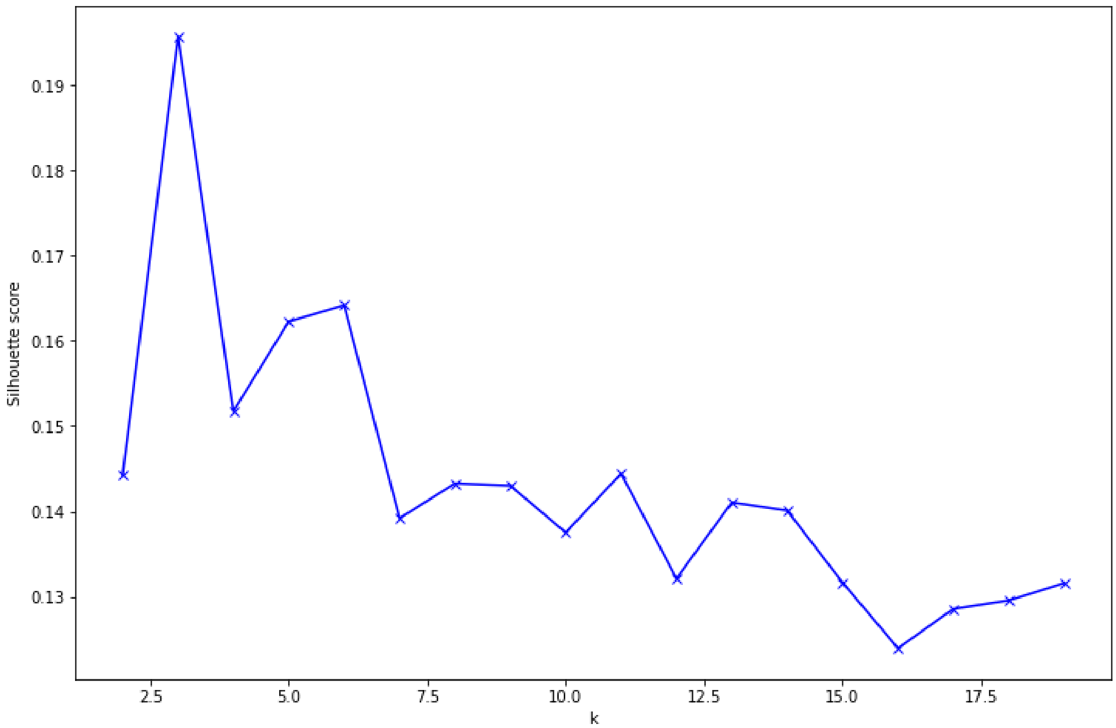
1.0.2 Klastrowania

[10]:
```
def count_clustering_scores(X, cluster_num, model, score_fun):
    if isinstance(cluster_num, int):
        cluster_num_iter = [cluster_num]
    else:
        cluster_num_iter = cluster_num

    scores = []
    for k in cluster_num_iter:
        model_instance = model(n_clusters=k)
        labels = model_instance.fit_predict(X)
        wcss = score_fun(X, labels)
        scores.append(wcss)

    if isinstance(cluster_num, int):
        return scores[0]
    else:
        return scores
```

[11]:
```
cluster_num_seq = range(2, 20)
davies_vec = count_clustering_scores(transformed_data, cluster_num_seq, KMeans,␣
↪davies_bouldin_score)
plt.figure(figsize=(12,8))
plt.plot(cluster_num_seq, davies_vec, 'bx-')
plt.xlabel('k')
plt.ylabel('davies-bouldin score')
plt.show()
```



[12]:
```
cluster_num_seq = range(2, 20)
silhouette_vec = count_clustering_scores(transformed_data, cluster_num_seq,␣
↪KMeans, silhouette_score)
plt.figure(figsize=(12,8))
plt.plot(cluster_num_seq, silhouette_vec, 'bx-')
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.show()
```



Pierwszy przykładowy model

```
[13]: model_km = KMeans(n_clusters = 12, random_state = 42)
      labels_km = model_km.fit_predict(transformed_data)
```

```
[14]: transformed_data["cluster"] = labels_km
      data["cluster"] = labels_km
```

```
[15]: tSNE = TSNE(learning_rate = 300, random_state = 42, verbose = 1)
```

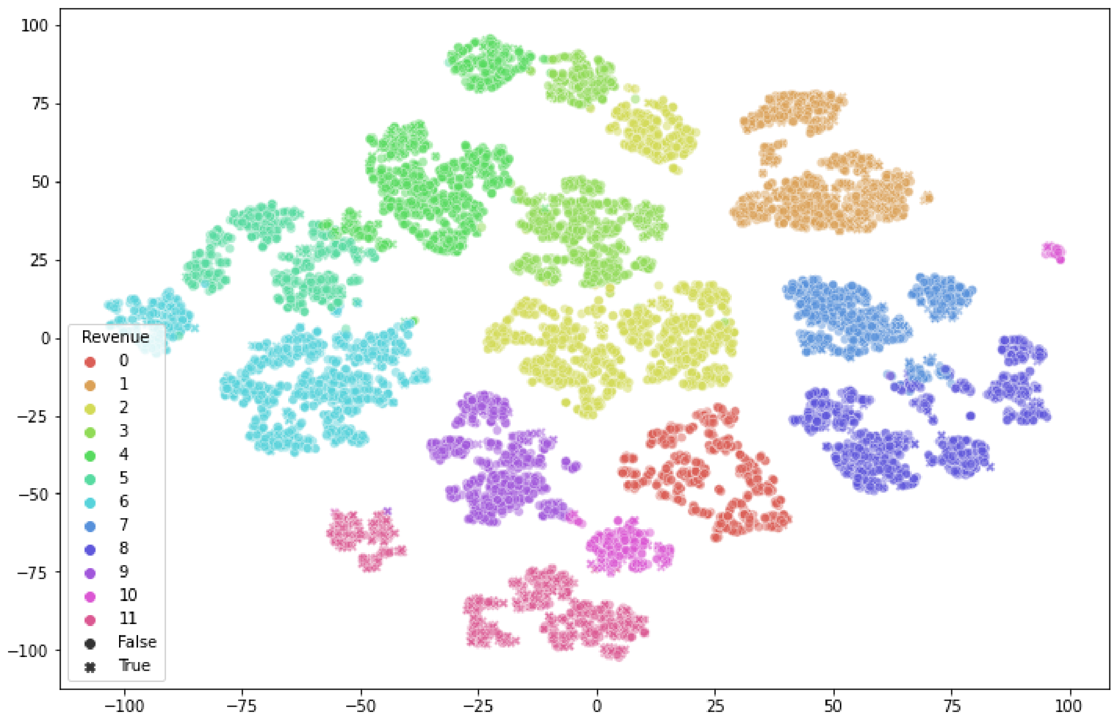
```
[16]: tSNE_td = tSNE.fit_transform(transformed_data)
```

```
[t-SNE] Computing 91 nearest neighbors…
[t-SNE] Indexed 12330 samples in 0.001s…
[t-SNE] Computed neighbors for 12330 samples in 3.583s…
[t-SNE] Computed conditional probabilities for sample 1000 / 12330
[t-SNE] Computed conditional probabilities for sample 2000 / 12330
[t-SNE] Computed conditional probabilities for sample 3000 / 12330
[t-SNE] Computed conditional probabilities for sample 4000 / 12330
[t-SNE] Computed conditional probabilities for sample 5000 / 12330
[t-SNE] Computed conditional probabilities for sample 6000 / 12330
[t-SNE] Computed conditional probabilities for sample 7000 / 12330
[t-SNE] Computed conditional probabilities for sample 8000 / 12330
[t-SNE] Computed conditional probabilities for sample 9000 / 12330
[t-SNE] Computed conditional probabilities for sample 10000 / 12330
```

```
[t-SNE] Computed conditional probabilities for sample 11000 / 12330
[t-SNE] Computed conditional probabilities for sample 12000 / 12330
[t-SNE] Computed conditional probabilities for sample 12330 / 12330
[t-SNE] Mean sigma: 0.918153
[t-SNE] KL divergence after 250 iterations with early exaggeration: 73.616447
[t-SNE] KL divergence after 1000 iterations: 1.176429
```

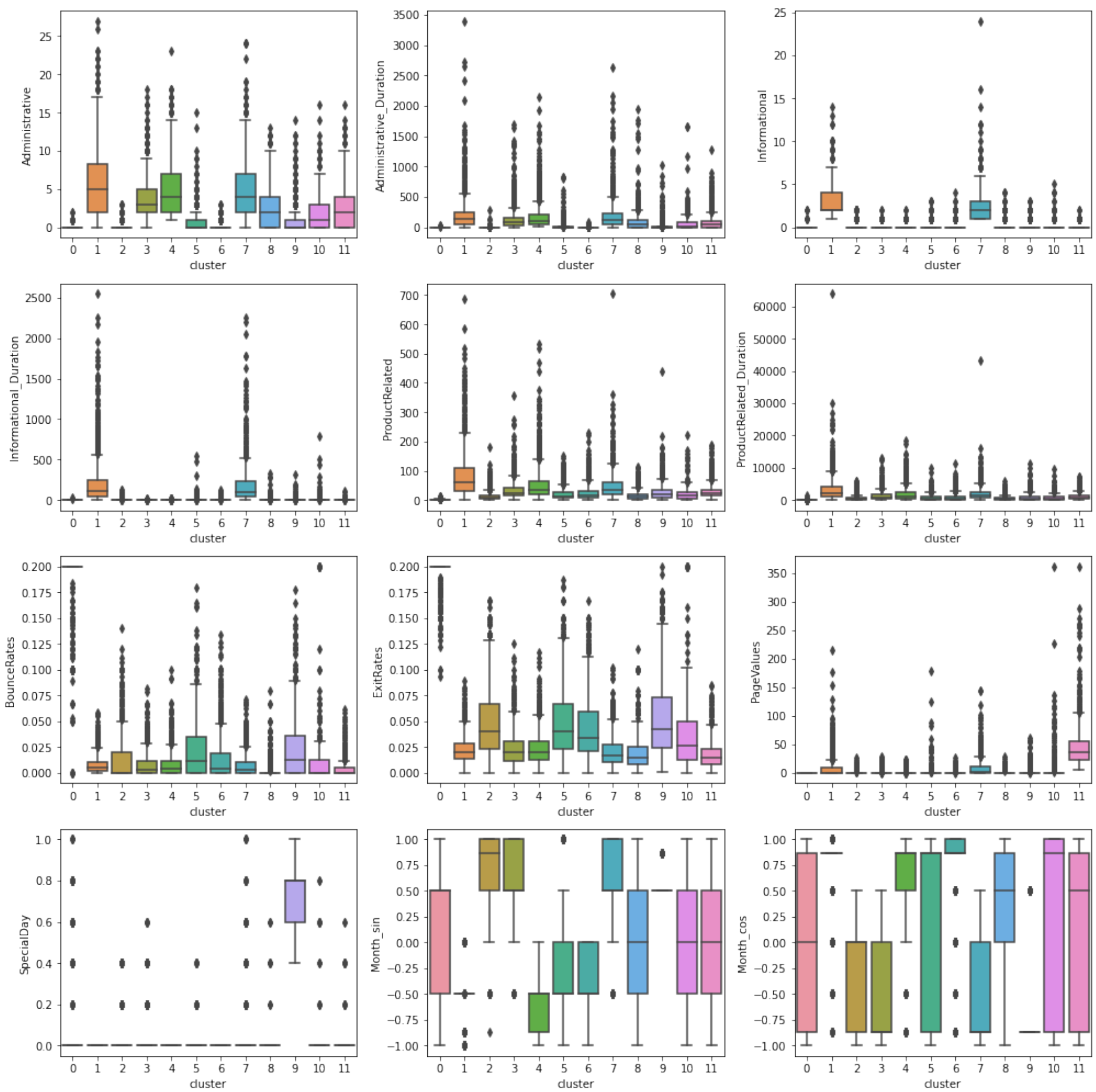
[17]:

```
plt.figure(figsize=(12,8))
sns.scatterplot(x = tSNE_td[:,0],
                y = tSNE_td[:,1],
                hue = labels_km,
                style = data["Revenue"],
                alpha=0.5,
                palette=sns.color_palette("hls", 12),
                legend=True)
plt.show()
```



[18]:

```
fig, ax = plt.subplots(4, 3, figsize=(14, 14))
for i, feature in enumerate(num_vars):
    m, n = divmod(i, 3)
    sns.boxplot(x="cluster", y=feature, data=data, ax = ax[m, n])
plt.tight_layout()
plt.show()
```



[19]:
```python
results = data.groupby("cluster").agg(['sum', 'count'])
results["Revenue"]
```

[19]:

cluster	sum	count
0	4	795
1	307	1112
2	29	1716
3	81	1088
4	197	1281
5	50	725
6	121	1457
7	180	846

```
8          122   1175
9           40    817
10          64    334
11         713    984
```

Porównaniue wyników różnych modeli

[20]:
```
algorithms = {
    "KMeans": KMeans(random_state=42),
    "Agglomerative - ward linkage": AgglomerativeClustering(linkage="ward"),
    "Agglomerative - single linkage": AgglomerativeClustering(linkage="single"),
    "GMM - spherical covariance": GaussianMixture(covariance_type =␣
 ↪"spherical", random_state = 42)
}

# scores = {
#     "Silhouette": silhouette_score(),
#     "Calinski_Harabasz": calinski_harabasz_score(),
#     "Davies_Bouldin": davies_bouldin_score()
# }


silhouette_scores = pd.DataFrame()
calinski_harabasz_scores = pd.DataFrame()
davies_bouldin_scores = pd.DataFrame()
stability_scores= pd.DataFrame()
indices = [k for k in range(len(transformed_data))]


for i in range (2, 13):
    for name in algorithms:
        model = algorithms[name]
        if "KMeans" in name or "Agglomerative" in name:
            model.n_clusters = i
        else:
            model.n_components = i
        labels = model.fit_predict(transformed_data)
        silhouette_scores.loc[name, i] = silhouette_score(transformed_data,␣
 ↪labels)
        calinski_harabasz_scores.loc[name, i] =␣
 ↪calinski_harabasz_score(transformed_data, labels)
        davies_bouldin_scores.loc[name, i] =␣
 ↪davies_bouldin_score(transformed_data, labels)
        stability = []
        for j in range(5):
            resampled = sklearn.utils.resample(indices)
            resampled_pred = model.fit_predict(transformed_data.loc[resampled])
```

```
            stability.append(normalized_mutual_info_score(labels[resampled],
→resampled_pred))
        stability_scores.loc[name,i]  = np.mean(stability)
        print("Doing {} with {} clusters".format(name, i))
```

```
Doing KMeans with 2 clusters
Doing Agglomerative - ward linkage with 2 clusters
Doing Agglomerative - single linkage with 2 clusters
Doing GMM - spherical covariance with 2 clusters
Doing KMeans with 3 clusters
Doing Agglomerative - ward linkage with 3 clusters
Doing Agglomerative - single linkage with 3 clusters
Doing GMM - spherical covariance with 3 clusters
Doing KMeans with 4 clusters
Doing Agglomerative - ward linkage with 4 clusters
Doing Agglomerative - single linkage with 4 clusters
Doing GMM - spherical covariance with 4 clusters
Doing KMeans with 5 clusters
Doing Agglomerative - ward linkage with 5 clusters
Doing Agglomerative - single linkage with 5 clusters
Doing GMM - spherical covariance with 5 clusters
Doing KMeans with 6 clusters
Doing Agglomerative - ward linkage with 6 clusters
Doing Agglomerative - single linkage with 6 clusters
Doing GMM - spherical covariance with 6 clusters
Doing KMeans with 7 clusters
Doing Agglomerative - ward linkage with 7 clusters
Doing Agglomerative - single linkage with 7 clusters
Doing GMM - spherical covariance with 7 clusters
Doing KMeans with 8 clusters
Doing Agglomerative - ward linkage with 8 clusters
Doing Agglomerative - single linkage with 8 clusters
Doing GMM - spherical covariance with 8 clusters
Doing KMeans with 9 clusters
Doing Agglomerative - ward linkage with 9 clusters
Doing Agglomerative - single linkage with 9 clusters
Doing GMM - spherical covariance with 9 clusters
Doing KMeans with 10 clusters
Doing Agglomerative - ward linkage with 10 clusters
Doing Agglomerative - single linkage with 10 clusters
Doing GMM - spherical covariance with 10 clusters
Doing KMeans with 11 clusters
Doing Agglomerative - ward linkage with 11 clusters
Doing Agglomerative - single linkage with 11 clusters
Doing GMM - spherical covariance with 11 clusters
Doing KMeans with 12 clusters
Doing Agglomerative - ward linkage with 12 clusters
```

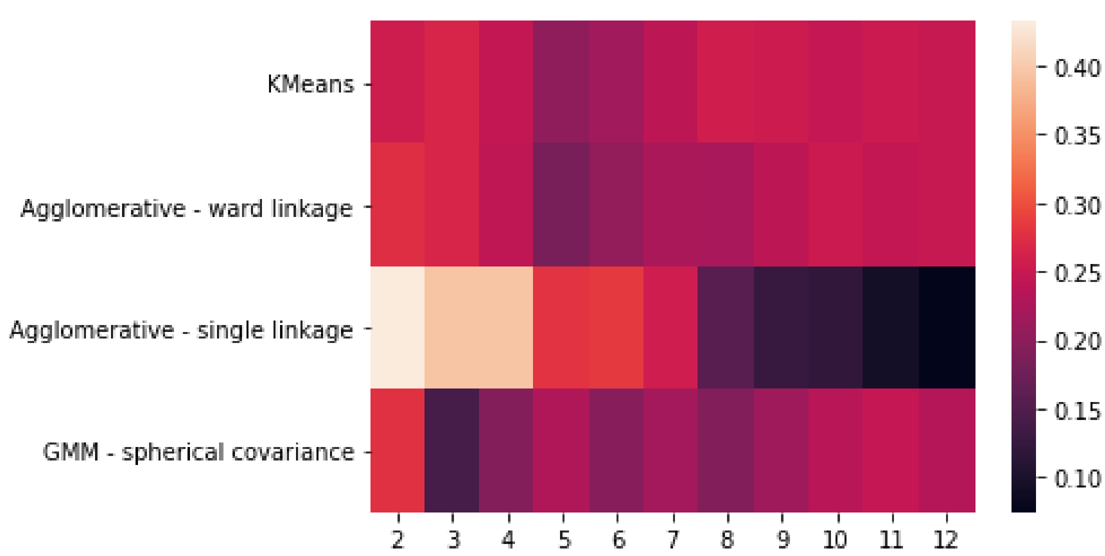
```
Doing Agglomerative - single linkage with 12 clusters
Doing GMM - spherical covariance with 12 clusters
```

[21]:
```
silhouette_scores
```

[21]:
```
                                           2         3         4         5  \
KMeans                              0.255348  0.266297  0.245254  0.203897
Agglomerative - ward linkage        0.274518  0.266342  0.243409  0.184296
Agglomerative - single linkage      0.432522  0.395241  0.395992  0.279146
GMM - spherical covariance          0.278084  0.141519  0.193916  0.229010

                                           6         7         8         9  \
KMeans                              0.218407  0.241893  0.258280  0.254101
Agglomerative - ward linkage        0.206469  0.226734  0.223355  0.241273
Agglomerative - single linkage      0.284495  0.256608  0.156920  0.127339
GMM - spherical covariance          0.197152  0.219387  0.192337  0.214404

                                          10        11        12
KMeans                              0.247850  0.253027  0.249738
Agglomerative - ward linkage        0.253351  0.246361  0.249511
Agglomerative - single linkage      0.120986  0.093372  0.074501
GMM - spherical covariance          0.237004  0.247576  0.233331
```

[22]:
```
sns.heatmap(silhouette_scores)
```

[22]: <AxesSubplot:>



[23]:
```
calinski_harabasz_scores
```

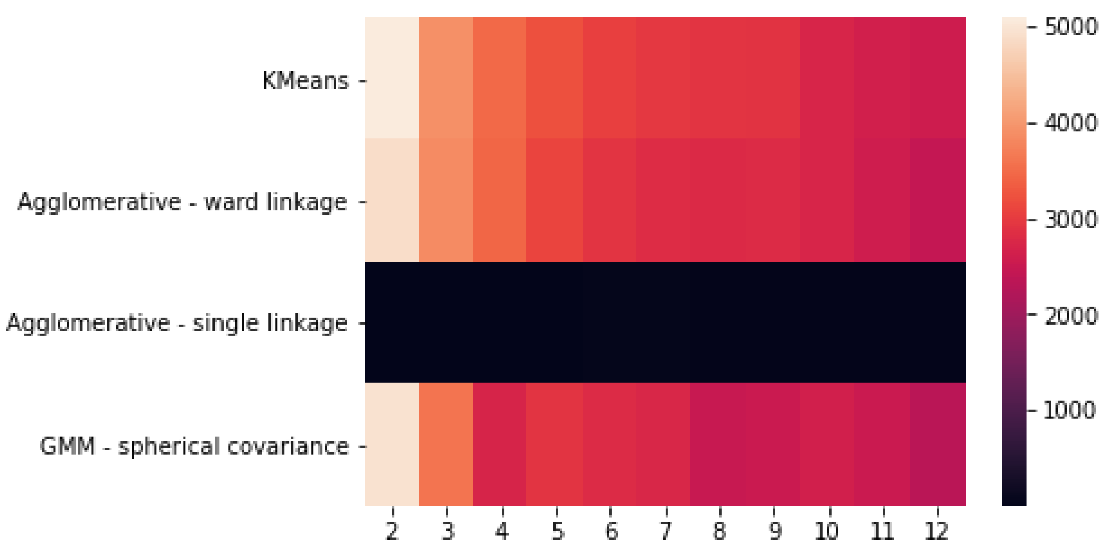
[23]:

```
                                        2            3            4  \
KMeans                        5087.371788  3918.483244  3492.695439
Agglomerative - ward linkage  4907.386965  3874.951727  3458.577867
Agglomerative - single linkage   4.834767     9.004224    19.692637
GMM - spherical covariance    4948.824651  3601.096198  2720.141607

                                        5            6            7  \
KMeans                        3238.734305  3048.447730  2971.450644
Agglomerative - ward linkage  3111.461145  2942.399783  2833.131294
Agglomerative - single linkage  15.442683    54.618631    45.981889
GMM - spherical covariance    2934.964963  2821.506144  2746.769409

                                        8            9           10  \
KMeans                        2926.669918  2906.266775  2730.208396
Agglomerative - ward linkage  2800.095730  2808.529266  2729.904592
Agglomerative - single linkage  39.667657    35.005747    31.420348
GMM - spherical covariance    2487.971080  2537.649394  2626.224560

                                       11           12
KMeans                        2628.124666  2576.475931
Agglomerative - ward linkage  2587.664813  2453.150647
Agglomerative - single linkage  28.535504    26.124138
GMM - spherical covariance    2539.121563  2344.685174
```

[24]:

```python
sns.heatmap(calinski_harabasz_scores)
```

[24]: <AxesSubplot:>

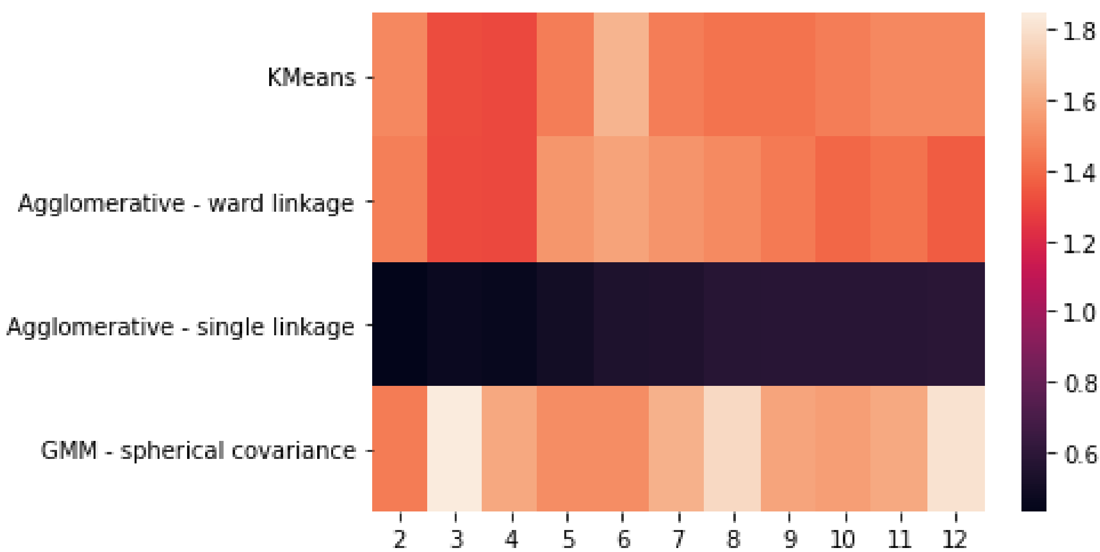


[25]:
```
davies_bouldin_scores
```

[25]:
```
                                       2         3         4         5  \
KMeans                          1.496253  1.318724  1.308293  1.459808
Agglomerative - ward linkage    1.465788  1.314278  1.306258  1.539190
Agglomerative - single linkage  0.431722  0.474953  0.462561  0.502775
GMM - spherical covariance      1.455278  1.845880  1.598681  1.509607

                                       6         7         8         9  \
KMeans                          1.642072  1.460383  1.429281  1.429113
Agglomerative - ward linkage    1.580744  1.535321  1.501923  1.449785
Agglomerative - single linkage  0.544506  0.548674  0.576988  0.582061
GMM - spherical covariance      1.514436  1.635397  1.777552  1.591033

                                      10        11        12
KMeans                          1.464697  1.495592  1.494629
Agglomerative - ward linkage    1.396953  1.428144  1.362351
Agglomerative - single linkage  0.581853  0.583344  0.591060
GMM - spherical covariance      1.569335  1.607668  1.811624
```

[26]:
```
sns.heatmap(davies_bouldin_scores)
```

[26]: <AxesSubplot:>



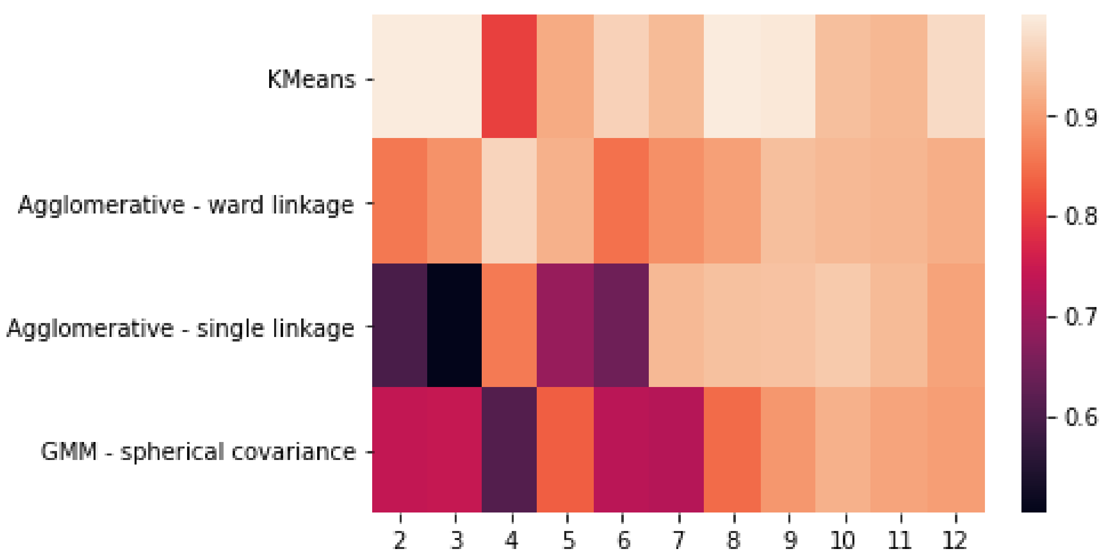
[27]:
```
stability_scores
```

[27]:
```
                                       2         3         4         5  \
KMeans                          0.997010  0.998603  0.802918  0.916176
```

```
Agglomerative - ward linkage    0.859431  0.887994  0.968292  0.923977
Agglomerative - single linkage  0.600000  0.503843  0.859522  0.690673
GMM - spherical covariance      0.741066  0.743817  0.611240  0.831221

                                       6         7         8         9  \
KMeans                          0.964812  0.935656  0.998195  0.992914
Agglomerative - ward linkage    0.850819  0.885467  0.902903  0.942487
Agglomerative - single linkage  0.644099  0.933815  0.942974  0.945501
GMM - spherical covariance      0.730173  0.724903  0.845635  0.893364

                                      10        11        12
KMeans                          0.942443  0.932783  0.977070
Agglomerative - ward linkage    0.933957  0.930194  0.920582
Agglomerative - single linkage  0.954348  0.936235  0.906418
GMM - spherical covariance      0.923431  0.909229  0.900518
```

[28]:
```
sns.heatmap(stability_scores)
```

[28]: <AxesSubplot:>

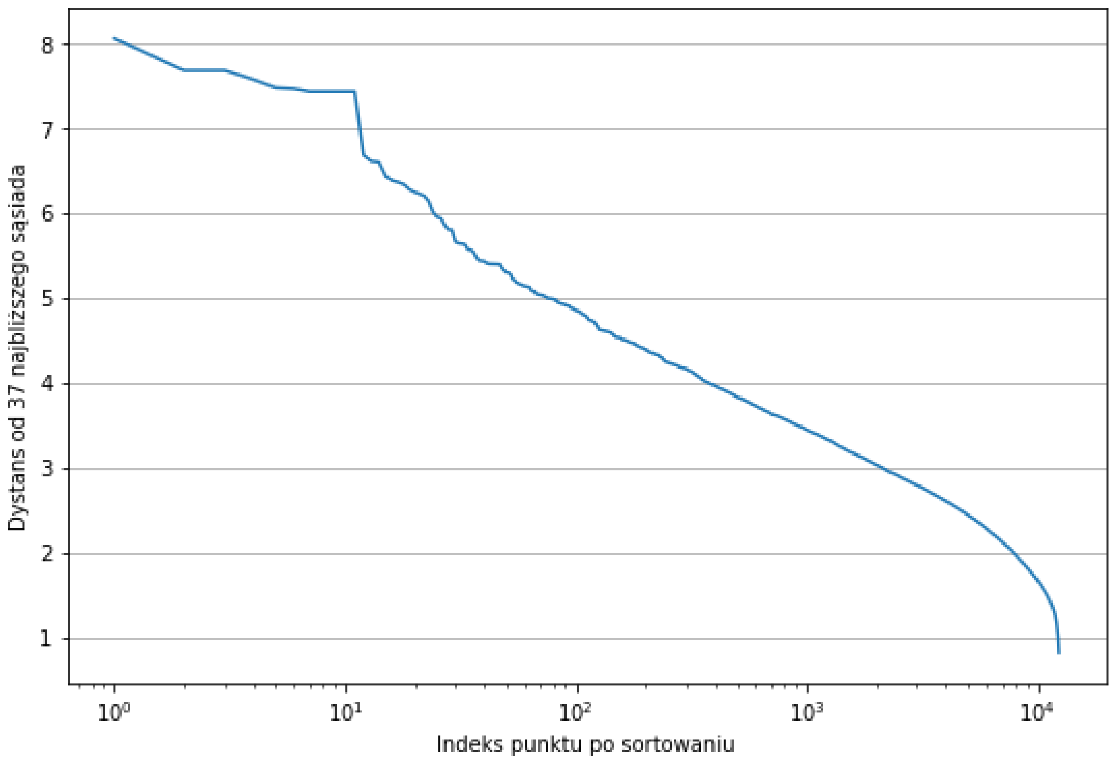


[29]:
```
transformed_data.shape
```

[29]: (12330, 19)

[30]:
```
minPts = 38
nbrs = sklearn.neighbors.NearestNeighbors(n_neighbors=minPts).
 ↪fit(transformed_data)
distances, indices = nbrs.kneighbors(transformed_data)
```

```
distanceDec = sorted(distances[:,minPts-1], reverse=True)
fig = plt.figure(figsize=(9,6))
ax1 = fig.add_subplot()

plt.xlabel('Indeks punktu po sortowaniu')
plt.ylabel('Dystans od 37 najbliższego sąsiada')
ax1.plot(list(range(1,transformed_data.shape[0]+1)), distanceDec)
plt.xscale('log')
plt.grid(axis='y')

plt.show()
```



[31]:
```
db = DBSCAN(eps=4.2, min_samples=38)

db_labels = db.fit_predict(transformed_data)
```

[32]:
```
set(db_labels)
```

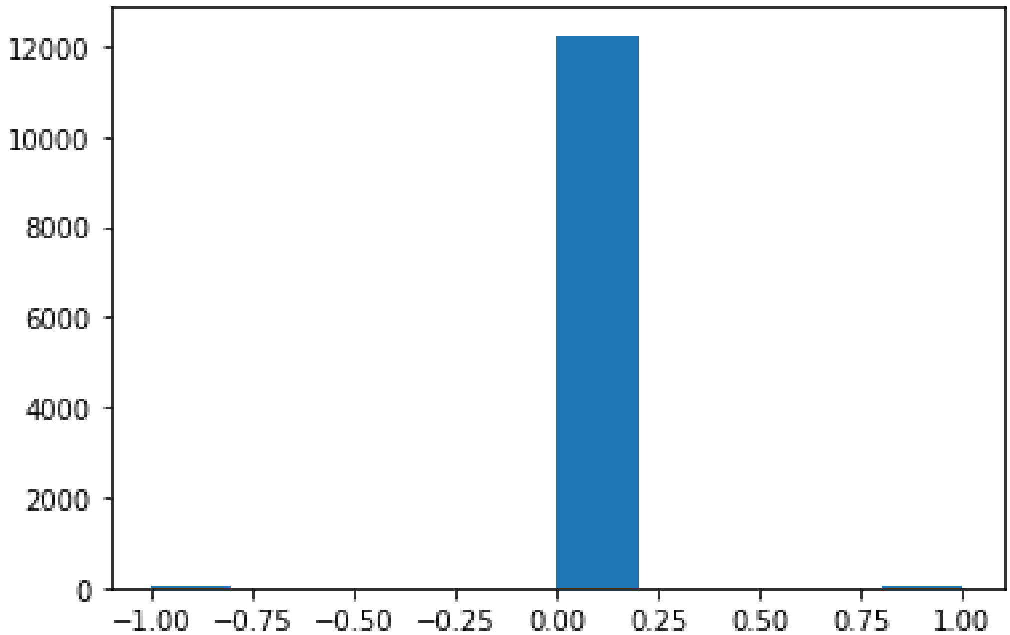
[32]: {-1, 0, 1}

[33]:
```
plt.figure(figsize=(12,8))
sns.scatterplot(x = tSNE_td[:,0],
```

```
                 y = tSNE_td[:,1],
                 hue = db_labels,
                 alpha=0.5,
                 palette=sns.color_palette("Set2", 3),
                 legend=True)
plt.show()
```

[34]:
```
plt.hist(db_labels)
```

[34]:
```
(array([   41.,     0.,     0.,     0.,     0., 12246.,     0.,     0.,
            0.,    43.]),
 array([-1. , -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  1. ]),
 <BarContainer object of 10 artists>)
```

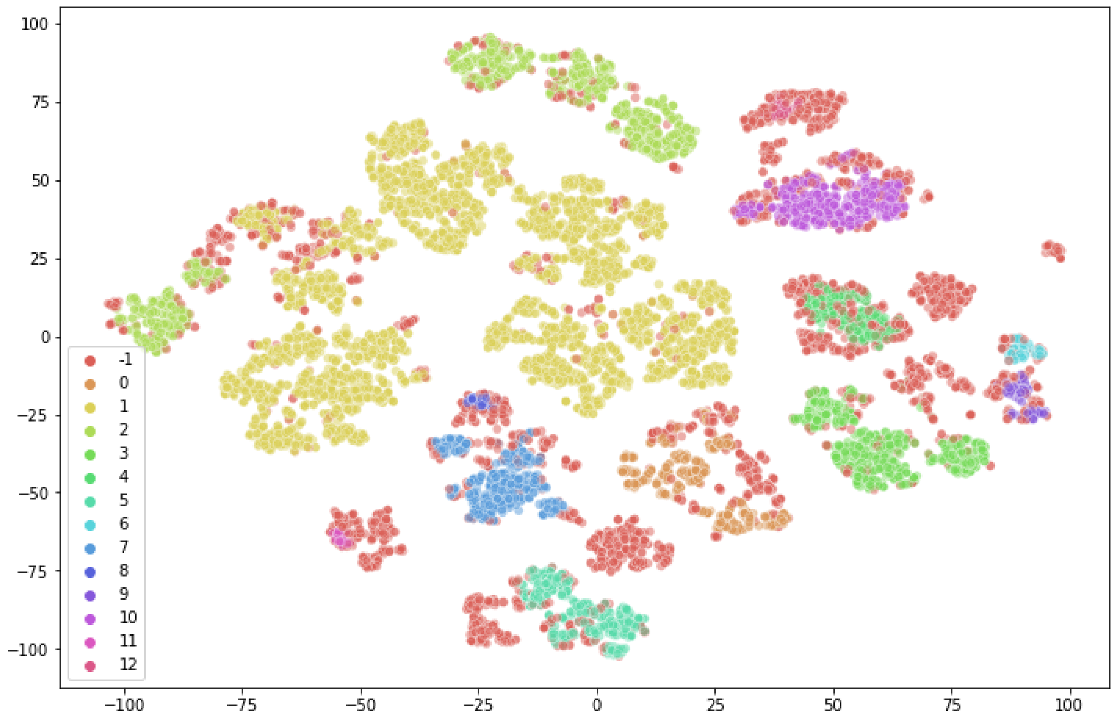


[35]:
```
db = DBSCAN(eps=2.2, min_samples=38)

db_labels = db.fit_predict(transformed_data)
```

[36]:
```
set(db_labels)
```

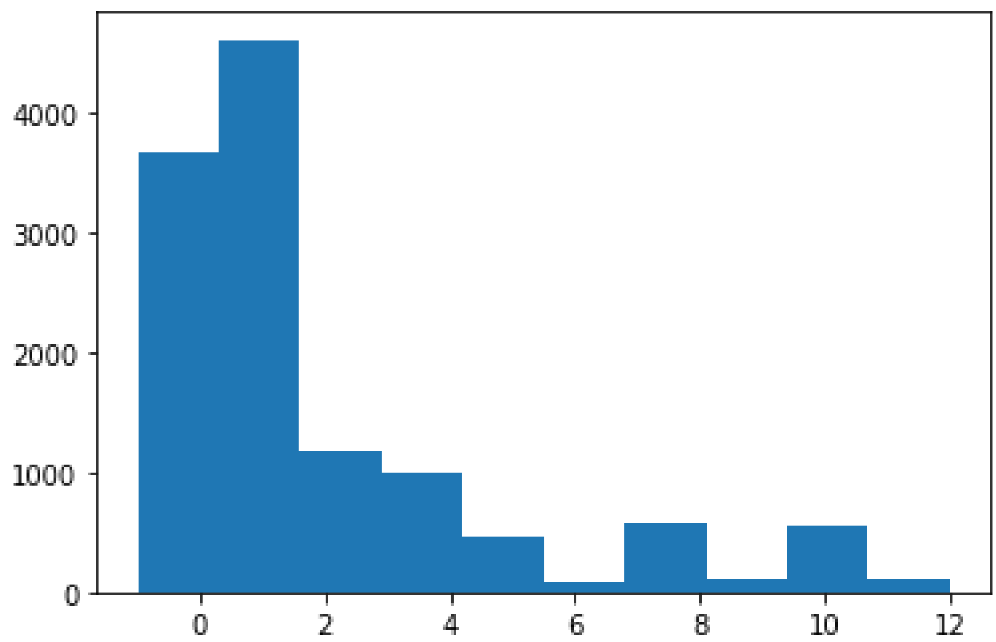
[36]: {-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[37]:
```
plt.figure(figsize=(12,8))
sns.scatterplot(x = tSNE_td[:,0],
                y = tSNE_td[:,1],
                hue = db_labels,
                alpha=0.5,
                palette=sns.color_palette("hls", 14),
                legend=True)
plt.show()
```



[38]:
```
plt.hist(db_labels)
```

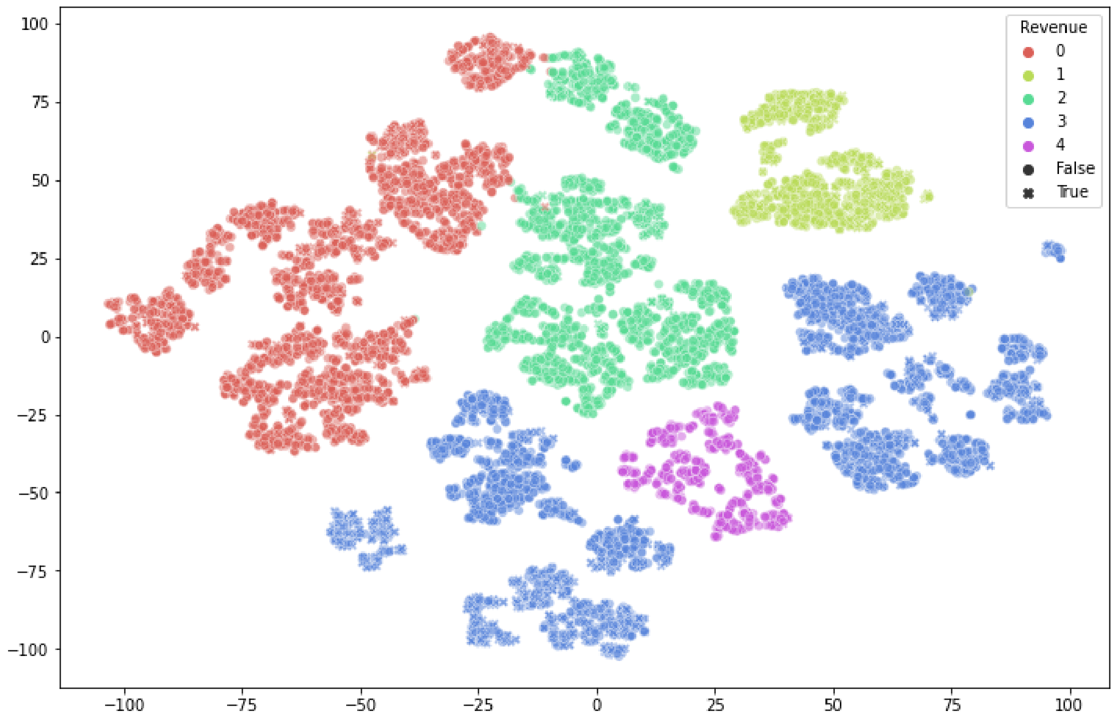
[38]:
```
(array([3676., 4606., 1175.,  990.,  453.,   85.,  577.,  109.,  556.,
          103.]),
 array([-1. ,  0.3,  1.6,  2.9,  4.2,  5.5,  6.8,  8.1,  9.4, 10.7, 12. ]),
 <BarContainer object of 10 artists>)
```



Analiza wybranego modelu

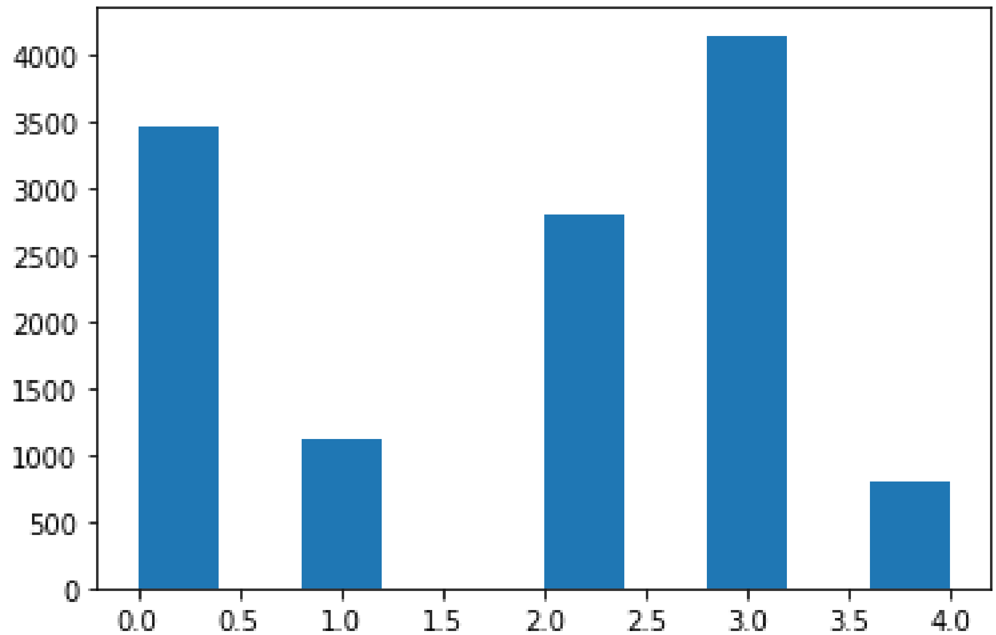
[39]:
```
km = KMeans(n_clusters=5, random_state=42)


labels = km.fit_predict(transformed_data)


transformed_data["cluster"] = labels
data["cluster"] = labels
```

[40]:
```
plt.figure(figsize=(12,8))
sns.scatterplot(x = tSNE_td[:,0],
                y = tSNE_td[:,1],
                hue = labels,
                style = data["Revenue"],
                alpha=0.5,
                palette=sns.color_palette("hls", 5),
                legend=True)
plt.show()
```



[41]:
```
plt.hist(labels)
```

[41]:
```
(array([3462.,    0., 1114.,    0.,    0., 2804.,    0., 4155.,    0.,
         795.]),
 array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
 <BarContainer object of 10 artists>)
```

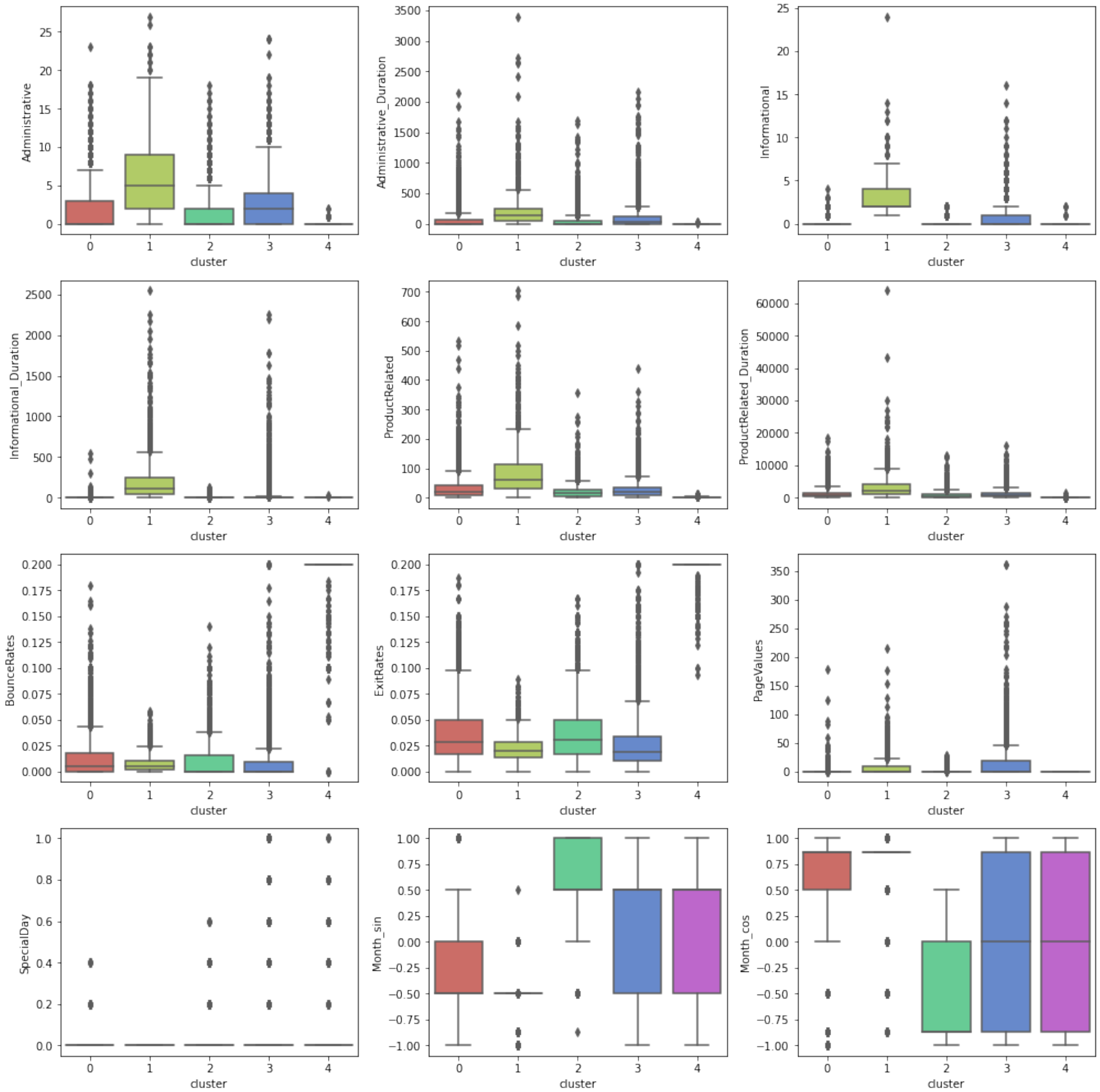


[42]:
```
results = data.groupby("cluster").agg(['sum', 'count'])
results["Revenue"]
```

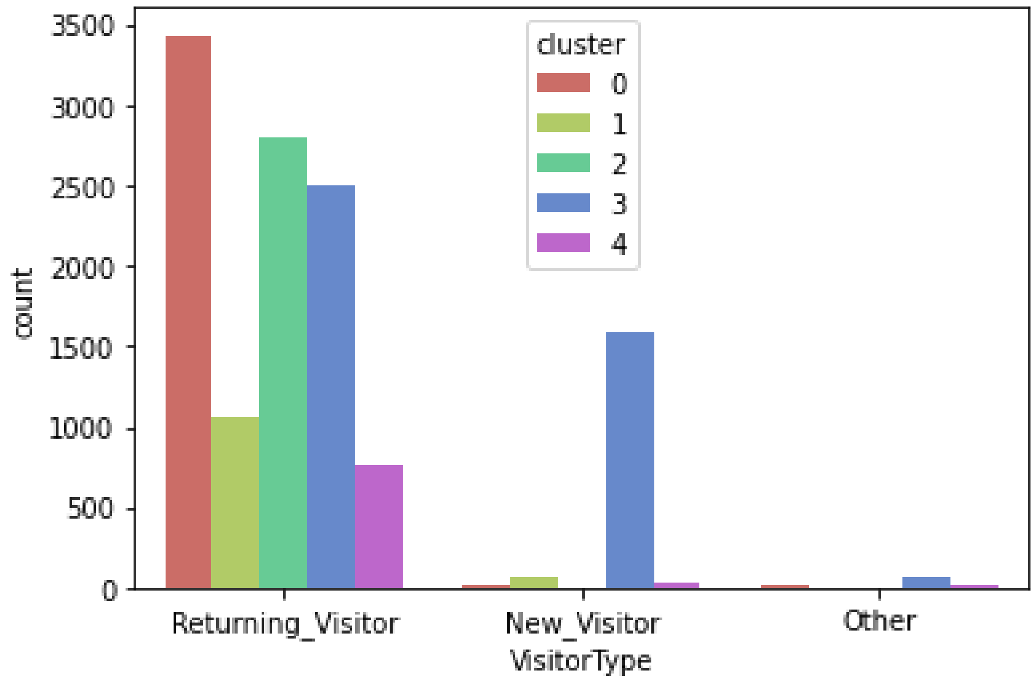
[42]:

cluster	sum	count
0	367	3462
1	308	1114
2	110	2804
3	1119	4155
4	4	795

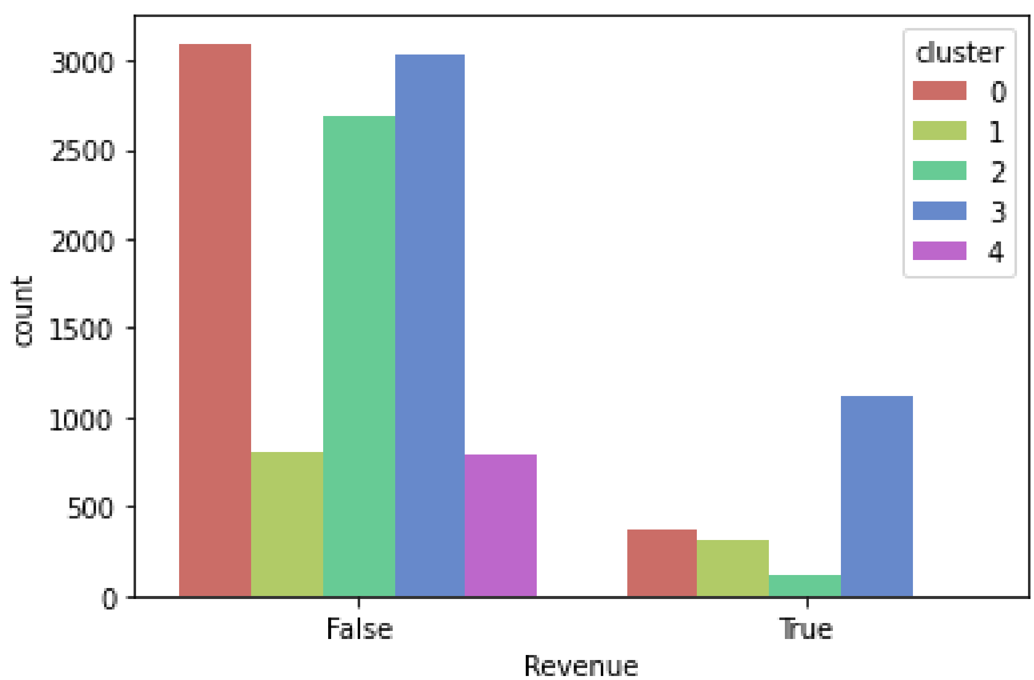
[43]:
```
fig, ax = plt.subplots(4, 3, figsize=(14, 14))
for i, feature in enumerate(num_vars):
    m, n = divmod(i, 3)
    sns.boxplot(x="cluster", y=feature, data=data, ax = ax[m, n], palette=sns.
 ↪color_palette("hls", 5))
plt.tight_layout()
plt.show()
```



[44]:
```
sns.countplot(x="VisitorType", hue="cluster", data=data, palette=sns.
↪color_palette("hls", 5))
plt.show()
```



[45]:
```
sns.countplot(x="Revenue", hue="cluster", data=data, palette=sns.
 ↪color_palette("hls", 5))
plt.show()
```



[46]:
```
from scipy.spatial import distance

def min_interclust_dist(X, label):
    clusters = set(label)
    global_min_dist = np.inf
    for cluster_i in clusters:
        cluster_i_idx = np.where(label == cluster_i)
        for cluster_j in clusters:
            if cluster_i != cluster_j:
                cluster_j_idx = np.where(label == cluster_j)
                interclust_min_dist = np.min(distance.cdist(X[cluster_i_idx],
 ↪X[cluster_j_idx]))
                global_min_dist = np.min([global_min_dist, interclust_min_dist])
    return global_min_dist

def _inclust_mean_dists(X, label):
    clusters = set(label)
    inclust_dist_list = []
    for cluster_i in clusters:
        cluster_i_idx = np.where(label == cluster_i)
        inclust_dist = np.mean(distance.pdist(X[cluster_i_idx]))
        inclust_dist_list.append(inclust_dist)
    return inclust_dist_list

def mean_inclust_dist(X, label):
    inclust_dist_list = _inclust_mean_dists(X, label)
    return np.mean(inclust_dist_list)

def std_dev_of_inclust_dist(X, label):
    inclust_dist_list = _inclust_mean_dists(X, label)
    return np.std(inclust_dist_list)

def mean_dist_to_center(X, label):
    clusters = set(label)
    inclust_dist_list = []
    for cluster_i in clusters:
        cluster_i_idx = np.where(label == cluster_i)
        cluster_i_mean = np.mean(X[cluster_i_idx], axis=0, keepdims=True)
        inclust_dist = np.mean(distance.cdist(X[cluster_i_idx], cluster_i_mean))
        inclust_dist_list.append(inclust_dist)
    return np.mean(inclust_dist_list)
```

[47]:
```
min_interclust_dist(transformed_data.to_numpy(), labels)
```

[47]: 1.1149975556513327

[48]:
```
mean_inclust_dist(transformed_data.to_numpy(), labels)
```

[48]: 4.558032295508201

[49]:
```
std_dev_of_inclust_dist(transformed_data.to_numpy(), labels)
```

[49]: 0.7973024631127239

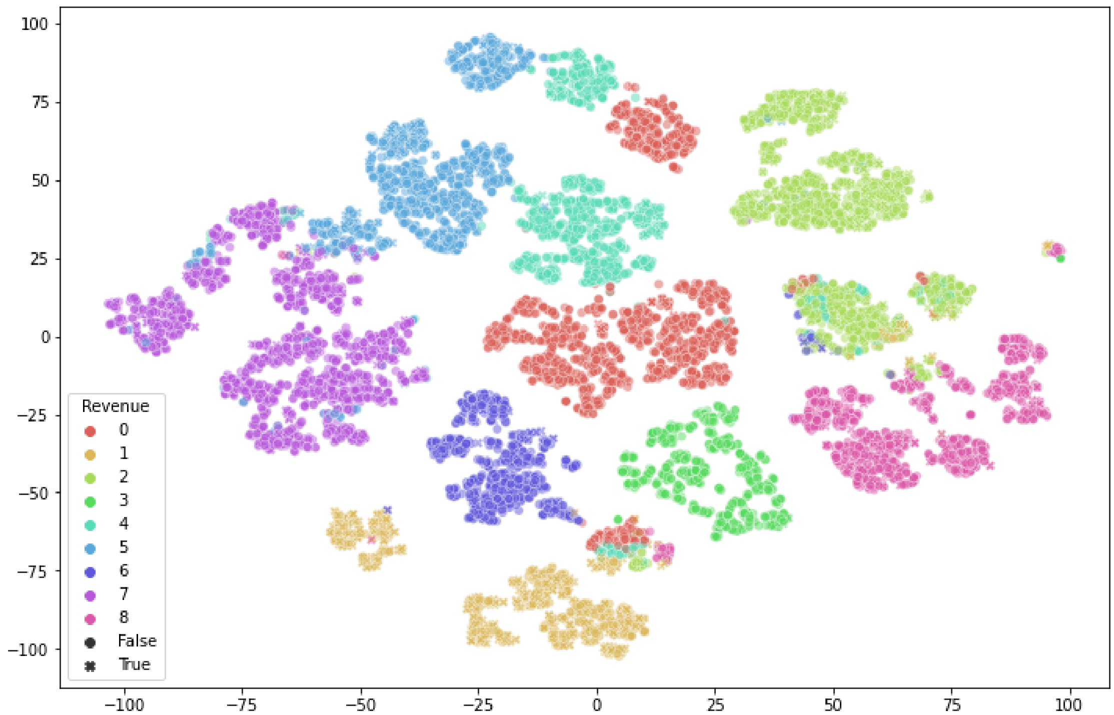
[50]:
```
mean_dist_to_center(transformed_data.to_numpy(), labels)
```

[50]: 3.2280194726003573

1.0.3 9 klastrów - bonus

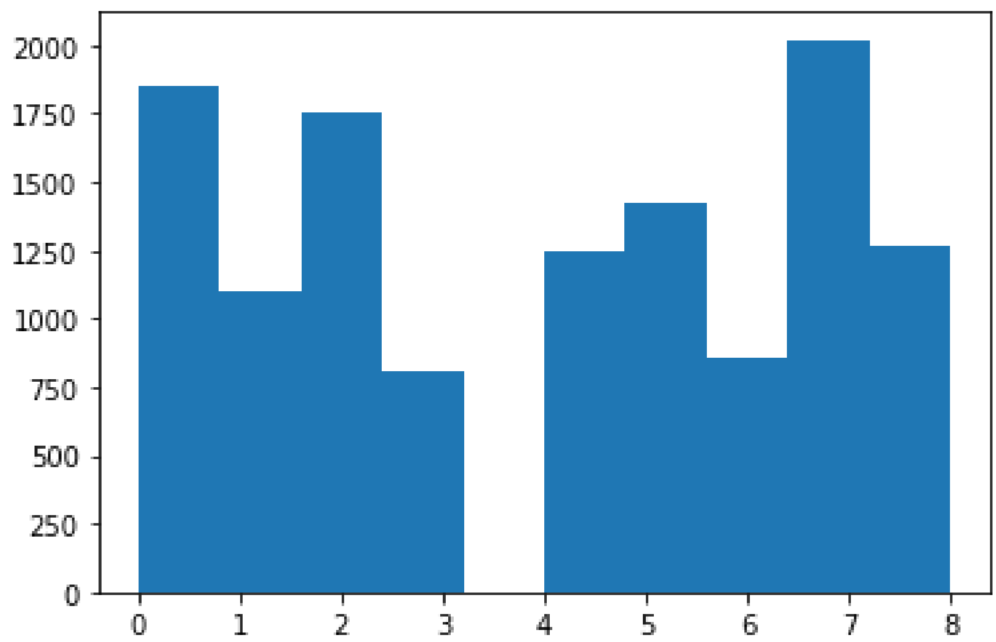
[51]:
```
km = KMeans(n_clusters=9, random_state=42)


labels = km.fit_predict(transformed_data)


transformed_data["cluster"] = labels
data["cluster"] = labels
```

[52]:
```
plt.figure(figsize=(12,8))
sns.scatterplot(x = tSNE_td[:,0],
                y = tSNE_td[:,1],
                hue = labels,
                style = data["Revenue"],
                alpha=0.5,
                palette=sns.color_palette("hls", 9),
                legend=True)
plt.show()
```



[53]:
```
plt.hist(labels)
```

[53]:
```
(array([1853., 1098., 1754.,  805.,    0., 1252., 1424.,  853., 2021.,
        1270.]),
 array([0. , 0.8, 1.6, 2.4, 3.2, 4. , 4.8, 5.6, 6.4, 7.2, 8. ]),
 <BarContainer object of 10 artists>)
```

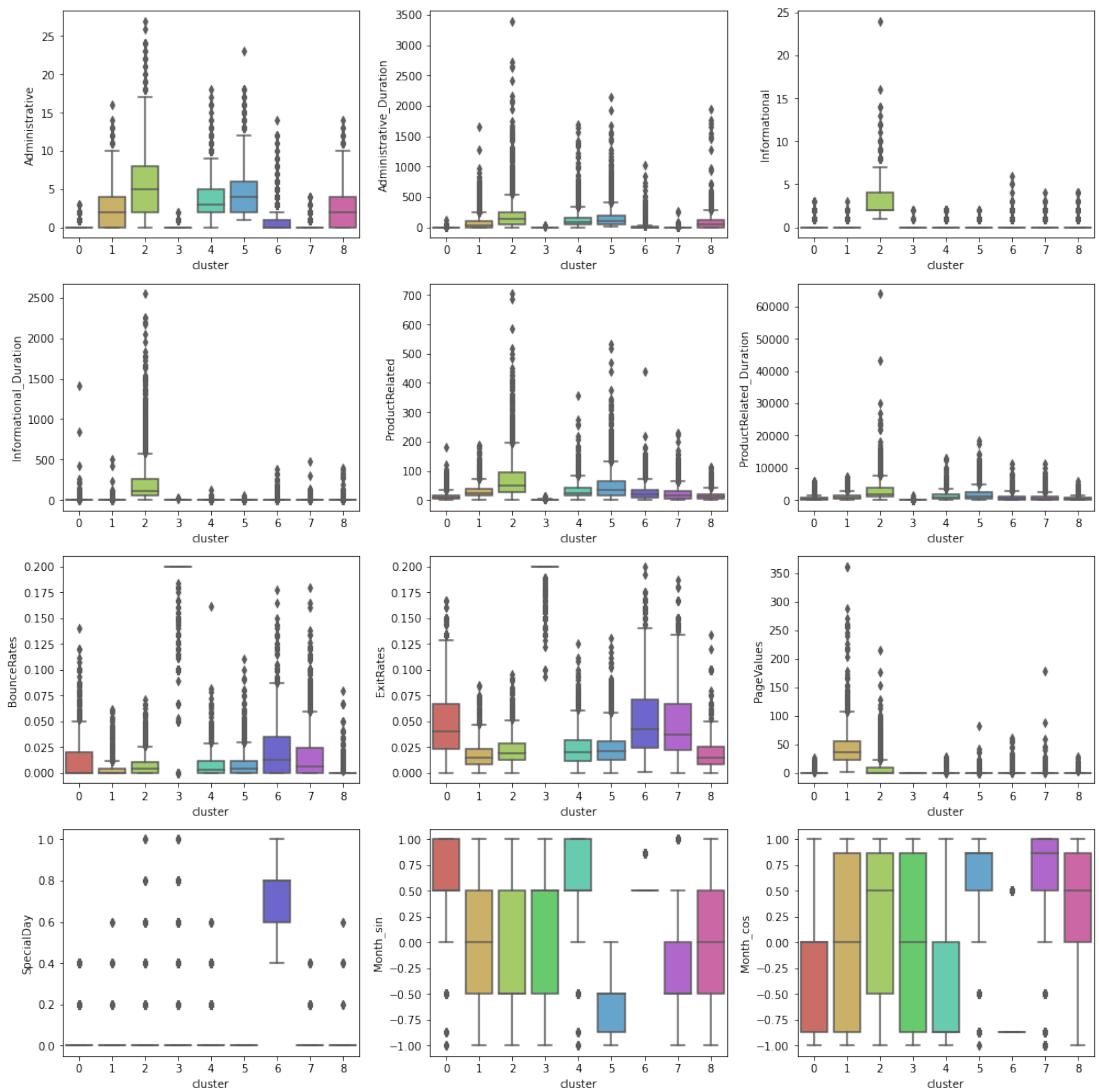


[54]:
```
results = data.groupby("cluster").agg(['sum', 'count'])
results["Revenue"]
```

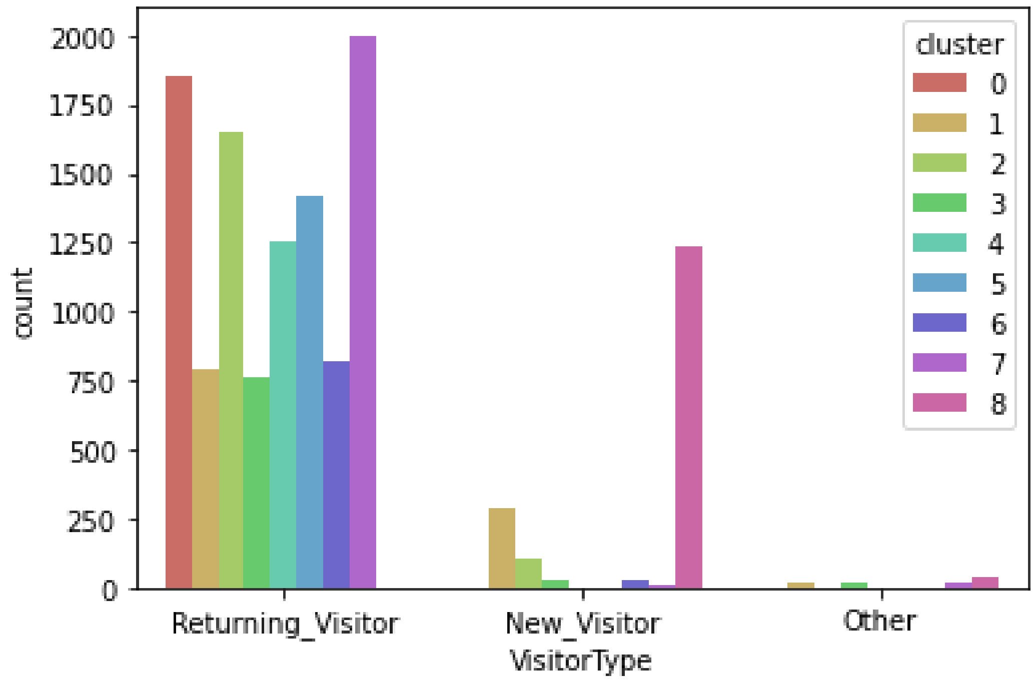
[54]:

cluster	sum	count
0	34	1853
1	800	1098
2	441	1754
3	4	805
4	88	1252
5	213	1424
6	47	853
7	153	2021
8	128	1270

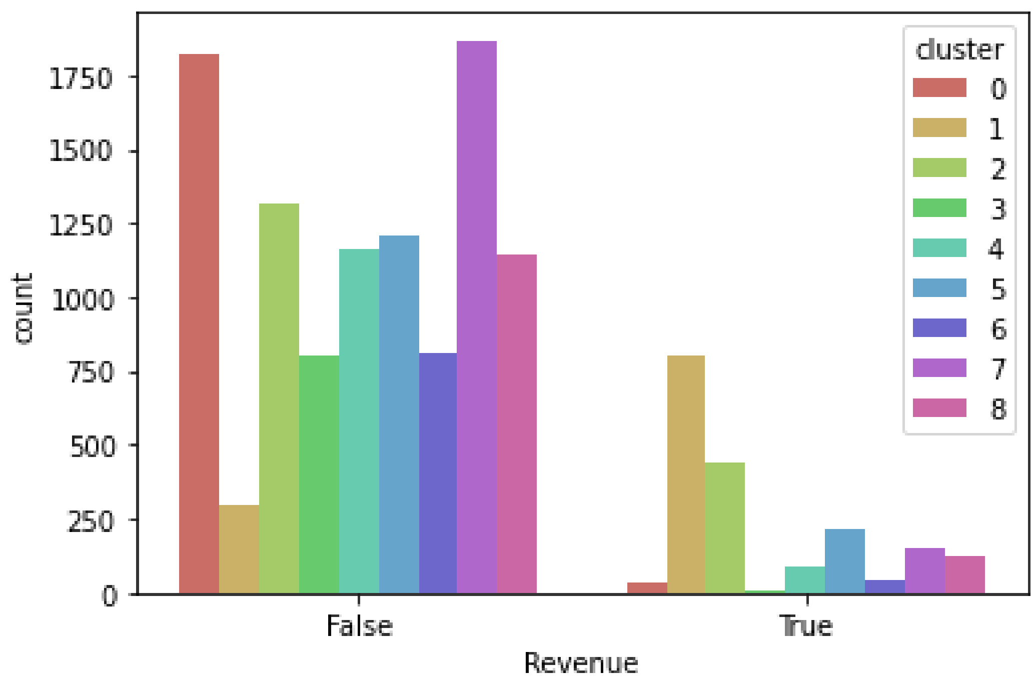
[55]:
```
fig, ax = plt.subplots(4, 3, figsize=(14, 14))
for i, feature in enumerate(num_vars):
    m, n = divmod(i, 3)
    sns.boxplot(x="cluster", y=feature, data=data, ax = ax[m, n], palette=sns.
 ↪color_palette("hls", 9))
plt.tight_layout()
plt.show()
```



[56]:
```
sns.countplot(x="VisitorType", hue="cluster", data=data, palette=sns.
 ↪color_palette("hls", 9))
plt.show()
```



[57]:
```
sns.countplot(x="Revenue", hue="cluster", data=data, palette=sns.
↪color_palette("hls", 9))
plt.show()
```



```
[ ]:
```