Klastrowanie_charakterystki_klastrów

June 16, 2021

[2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import pandas_profiling
import copy
from sklearn.decomposition import PCA
import sklearn.metrics
from sklearn import manifold
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
```

0.1 Dane, klasteryzacja k-means

[3]:
```
df=pd.read_csv("online_shoppers_intention.csv")
df=df.dropna()
df=df.drop(["Revenue"], axis=1)
```

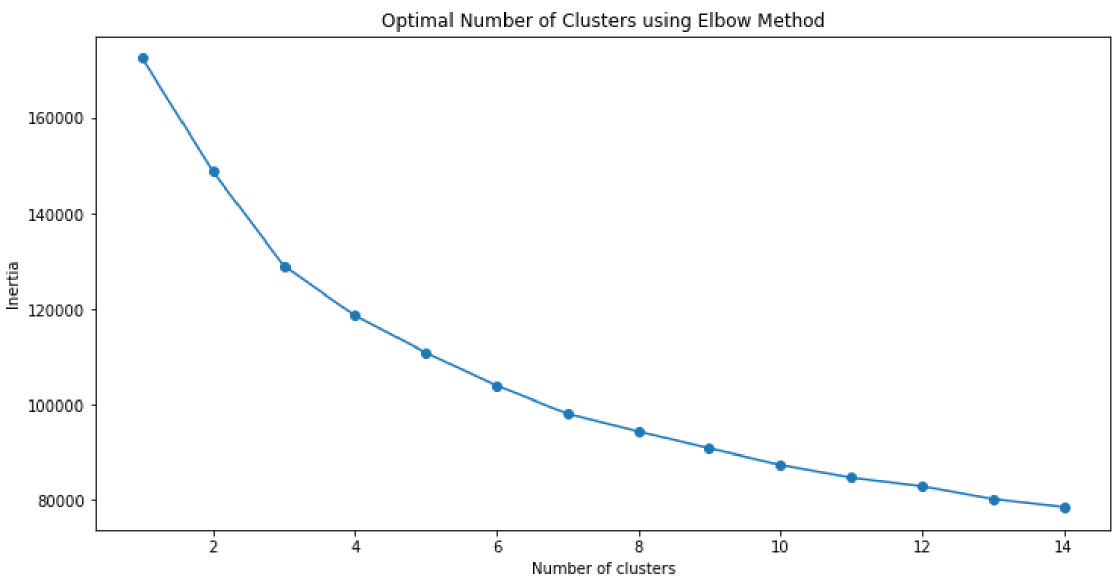
[4]:
```
X=df.copy()
X=pd.get_dummies(X, columns=["Month","OperatingSystems", "Browser", "Region",␣
 ↪"TrafficType", "VisitorType"] )
nums=["Administrative", "Administrative_Duration",
      "Informational", "Informational_Duration",
      "ProductRelated", "ProductRelated_Duration",
     "BounceRates", "ExitRates", "PageValues", "SpecialDay"]
X[nums]=(X[nums]-X[nums].min())/(X[nums].max()-X[nums].min())
df_norm=X.copy()
X=df.copy()
X=pd.get_dummies(X, columns=["Month","OperatingSystems", "Browser", "Region",␣
 ↪"TrafficType", "VisitorType"] )
nums=["Administrative", "Administrative_Duration",
      "Informational", "Informational_Duration",
      "ProductRelated", "ProductRelated_Duration",
     "BounceRates", "ExitRates", "PageValues", "SpecialDay"]
X[nums]=(X[nums]-X[nums].mean())/(X[nums].std())
df_scale=X.copy()
```

[5]:
```
df_scale.head()
```

[5]:
```
   Administrative  Administrative_Duration  Informational  \
0       -0.697553                -0.457458      -0.396615
1       -0.697553                -0.457458      -0.396615
2       -0.697553                -0.463112      -0.396615
3       -0.697553                -0.457458      -0.396615
4       -0.697553                -0.457458      -0.396615

   Informational_Duration  ProductRelated  ProductRelated_Duration  \
0               -0.245029       -0.691473                -0.624767
1               -0.245029       -0.668997                -0.591336
2               -0.252130       -0.691473                -0.625290
3               -0.245029       -0.668997                -0.623374
4               -0.245029       -0.489182                -0.296984

   BounceRates  ExitRates  PageValues  SpecialDay  …  TrafficType_14  \
0     3.672477   3.235240   -0.317363   -0.309001  …               0
1    -0.457439   1.174544   -0.317363   -0.309001  …               0
2     3.672477   3.235240   -0.317363   -0.309001  …               0
3     0.575040   1.998823   -0.317363   -0.309001  …               0
4    -0.044447   0.144196   -0.317363   -0.309001  …               0

   TrafficType_15  TrafficType_16  TrafficType_17  TrafficType_18  \
0               0               0               0               0
1               0               0               0               0
2               0               0               0               0
3               0               0               0               0
4               0               0               0               0

   TrafficType_19  TrafficType_20  VisitorType_New_Visitor  VisitorType_Other  \
0               0               0                        0                  0
1               0               0                        0                  0
2               0               0                        0                  0
3               0               0                        0                  0
4               0               0                        0                  0

   VisitorType_Returning_Visitor
0                              1
1                              1
2                              1
3                              1
4                              1

[5 rows x 74 columns]
```

[6]:
```
sse = []
k_list = range(1, 15)
for k in k_list:
    km = KMeans(n_clusters=k)
    km.fit(df_scale)
    sse.append([k, km.inertia_])

oca_results_scale = pd.DataFrame({'Cluster': range(1,15), 'SSE': sse})
plt.figure(figsize=(12,6))
plt.plot(pd.DataFrame(sse)[0], pd.DataFrame(sse)[1], marker='o')
plt.title('Optimal Number of Clusters using Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

[6]:
```
Text(0, 0.5, 'Inertia')
```



[7]:
```
# A w praktyce wygląda to tak:
def count_clustering_scores(X, cluster_num, model, score_fun):
    # Napiszmy tę funkcje tak ogólnie, jak to możliwe.
    # Zwróćcie uwagę na przekazanie obiektów typu callable: model i 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
```

[8]:
```
from sklearn.metrics import silhouette_score

cluster_num_seq = range(2, 15) # Niektóre metryki nie działają gdy mamy tylko␣
 ↪jeden klaster
silhouette_vec = count_clustering_scores(df_scale, cluster_num_seq, KMeans,␣
 ↪silhouette_score)
plt.plot(cluster_num_seq, silhouette_vec, 'bx-')
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.title("Silhouette score")
plt.show()
```

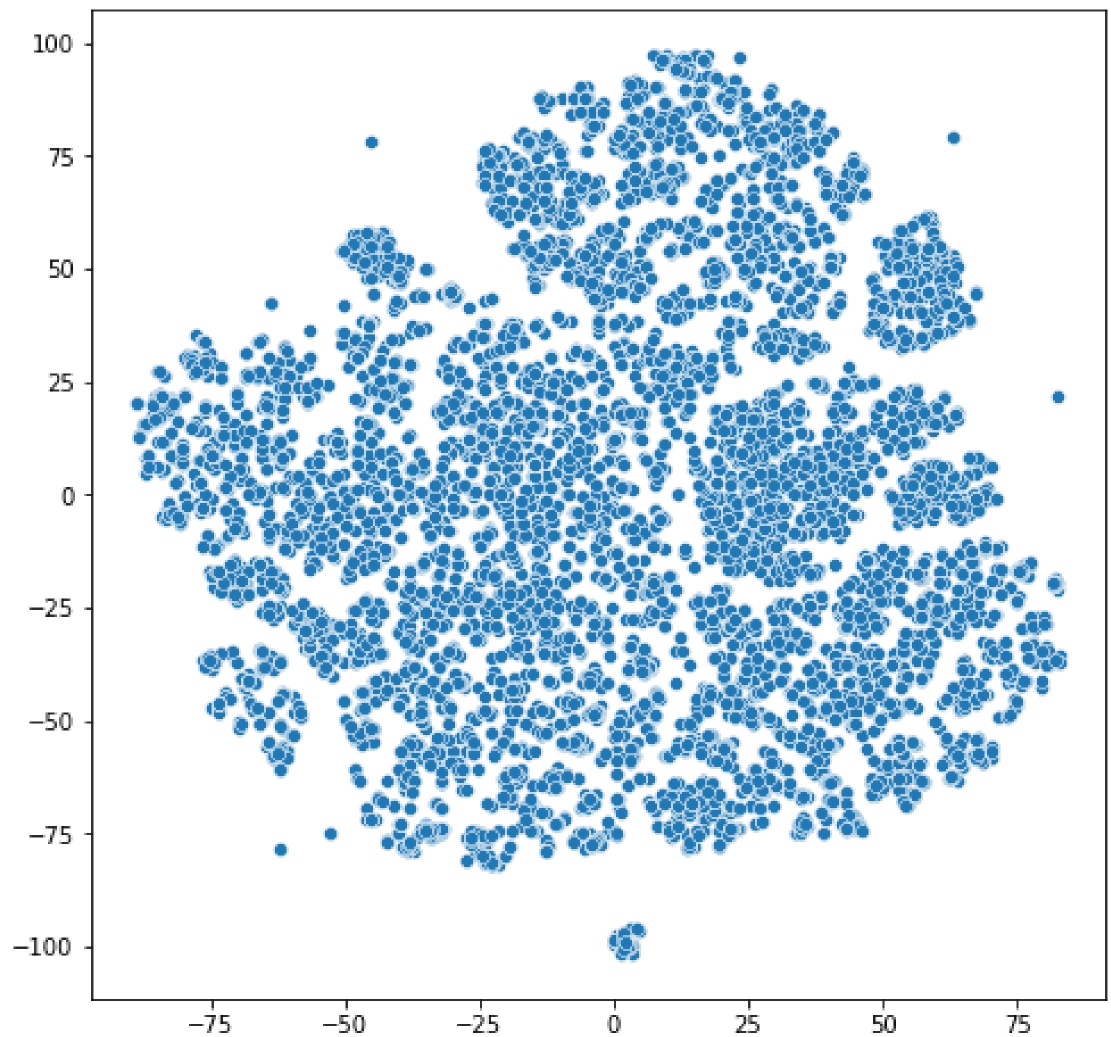
0.2 t-SNE - tylko do wizualizacji

[9]:
```
# A w praktyce wygląda to tak:
def count_clustering_scores(X, cluster_num, model, score_fun):
    # Napiszmy tę funkcje tak ogólnie, jak to możliwe.
    # Zwróćcie uwagę na przekazanie obiektów typu callable: model i 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
```

[10]:
```
tsne = TSNE(n_components=2, perplexity=35, random_state=40)
cords=tsne.fit_transform(df_norm)
plt.figure(figsize=(8,8))
sns.scatterplot(cords[:, 0], cords[:, 1], marker = 'o')
```

```
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

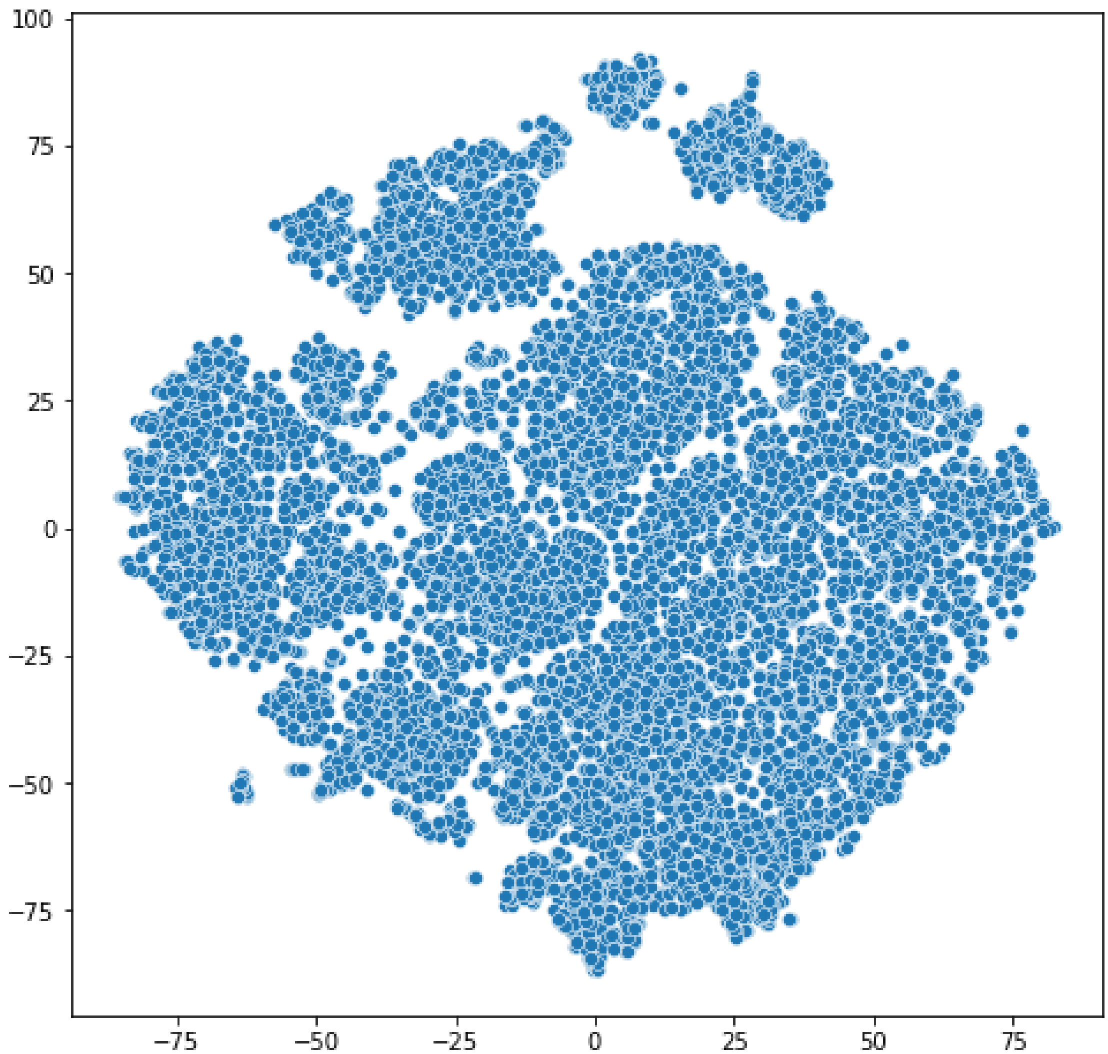
[10]:
```
<AxesSubplot:>
```



[11]:
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, perplexity=30, random_state=1)
coordsFIN=tsne.fit_transform(df_scale)
plt.figure(figsize=(8,8))
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o')
```

```
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[11]: <AxesSubplot:>



0.3 Klasteryzacja k means

[12]:
```
def count_wcss_scores(X, k_max):
    #  WCSS = within-cluster sum of squares
    scores = []
    for k in range(1, k_max+1):
        kmeans = KMeans(n_clusters=k, random_state=0)
        kmeans.fit(X)
        wcss = kmeans.score(X) * -1 # score returns -WCSS
        scores.append(wcss)
    return scores
```

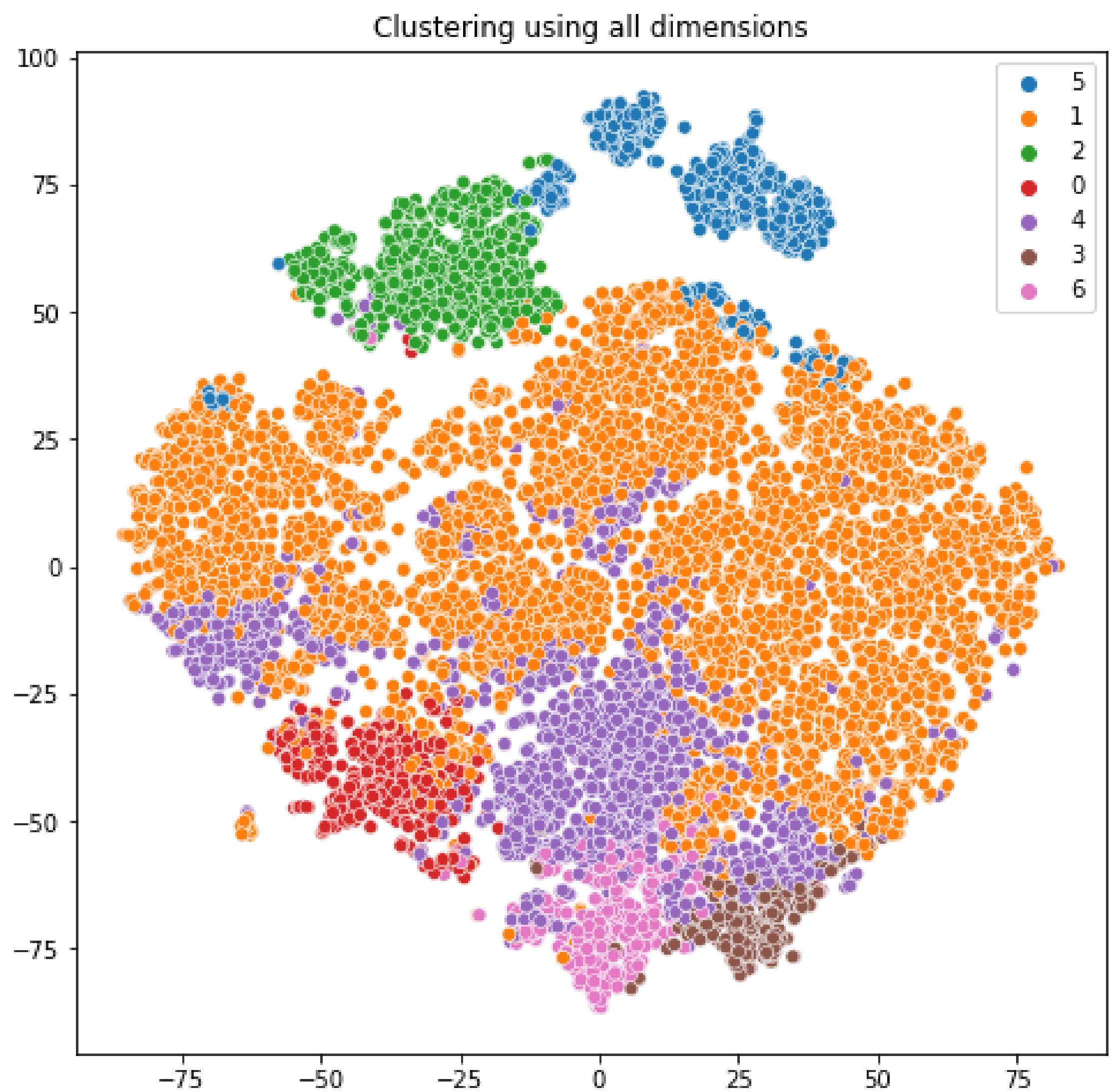
[13]:
```
km=KMeans(n_clusters=7, random_state=40)
arr=km.fit_predict(df_scale)
```

```
lst=[]
for i in range(len(arr)):
    lst.append(str(arr[i]))
```

[14]:
```
plt.figure(figsize=(8,8))
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o', hue=lst).
↪set_title("Clustering using all dimensions")
```

```
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

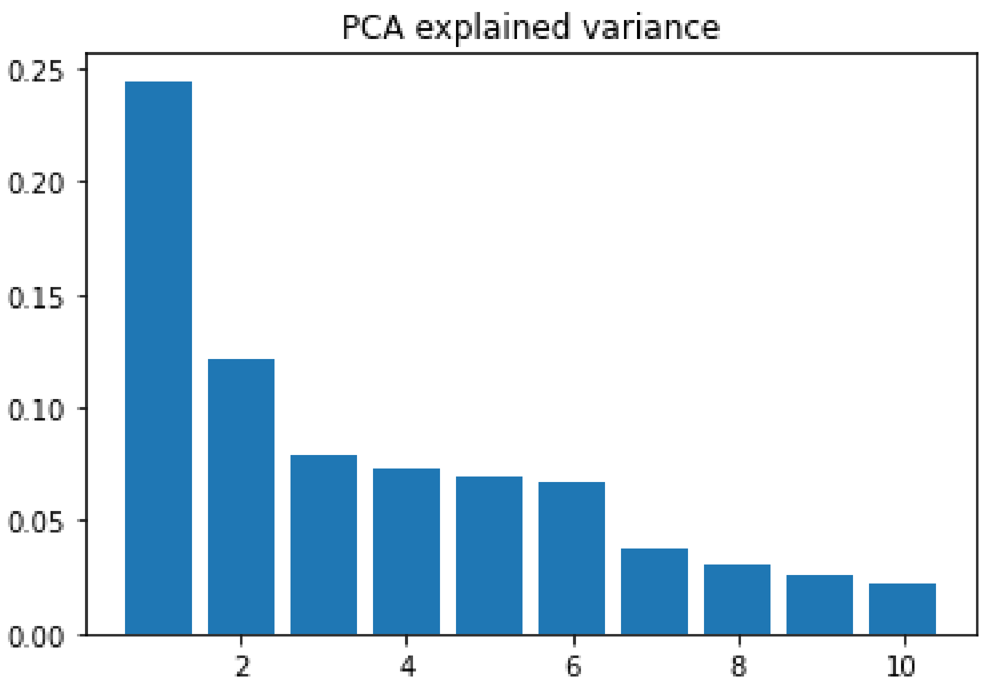
[14]: `Text(0.5, 1.0, 'Clustering using all dimensions')`



0.4 PCA

[15]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
pC = pca.fit_transform(df_scale)
ss=pca.explained_variance_ratio_
x=range(1,11)
plt.bar(x, ss)
plt.title("PCA explained variance")
```

[15]: Text(0.5, 1.0, 'PCA explained variance')

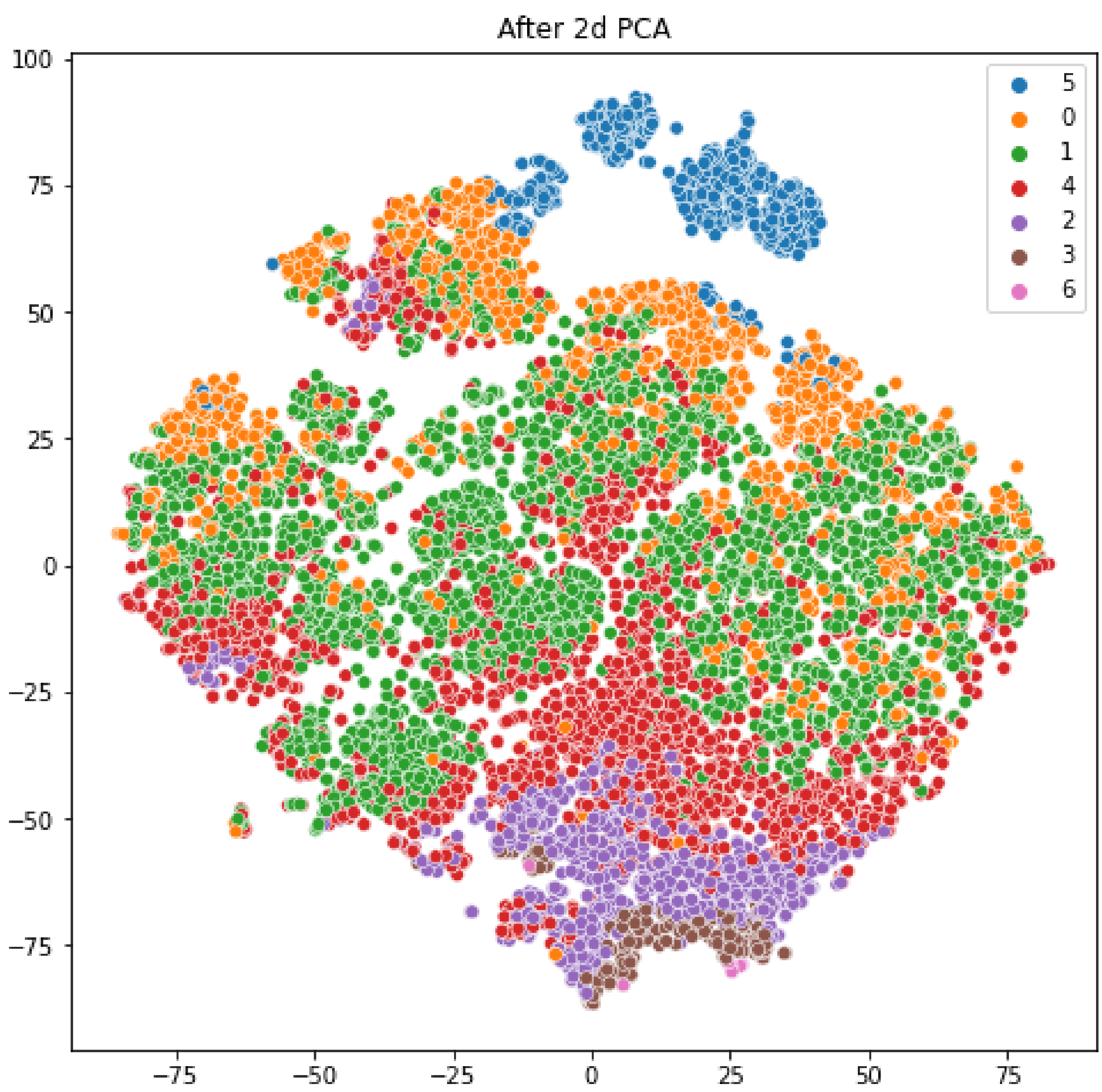


[16]:
```
pca = PCA(n_components=2)
df_scale
pC = pca.fit_transform(df_scale)
km=KMeans(n_clusters=7, random_state=40)
arr=km.fit_predict(pC)
lst=[]
for i in range(len(arr)):
    lst.append(str(arr[i]))
plt.figure(figsize=(8,8))
```

```
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o', hue=lst).
↪set_title("After 2d PCA")
```

```
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[16]: Text(0.5, 1.0, 'After 2d PCA')



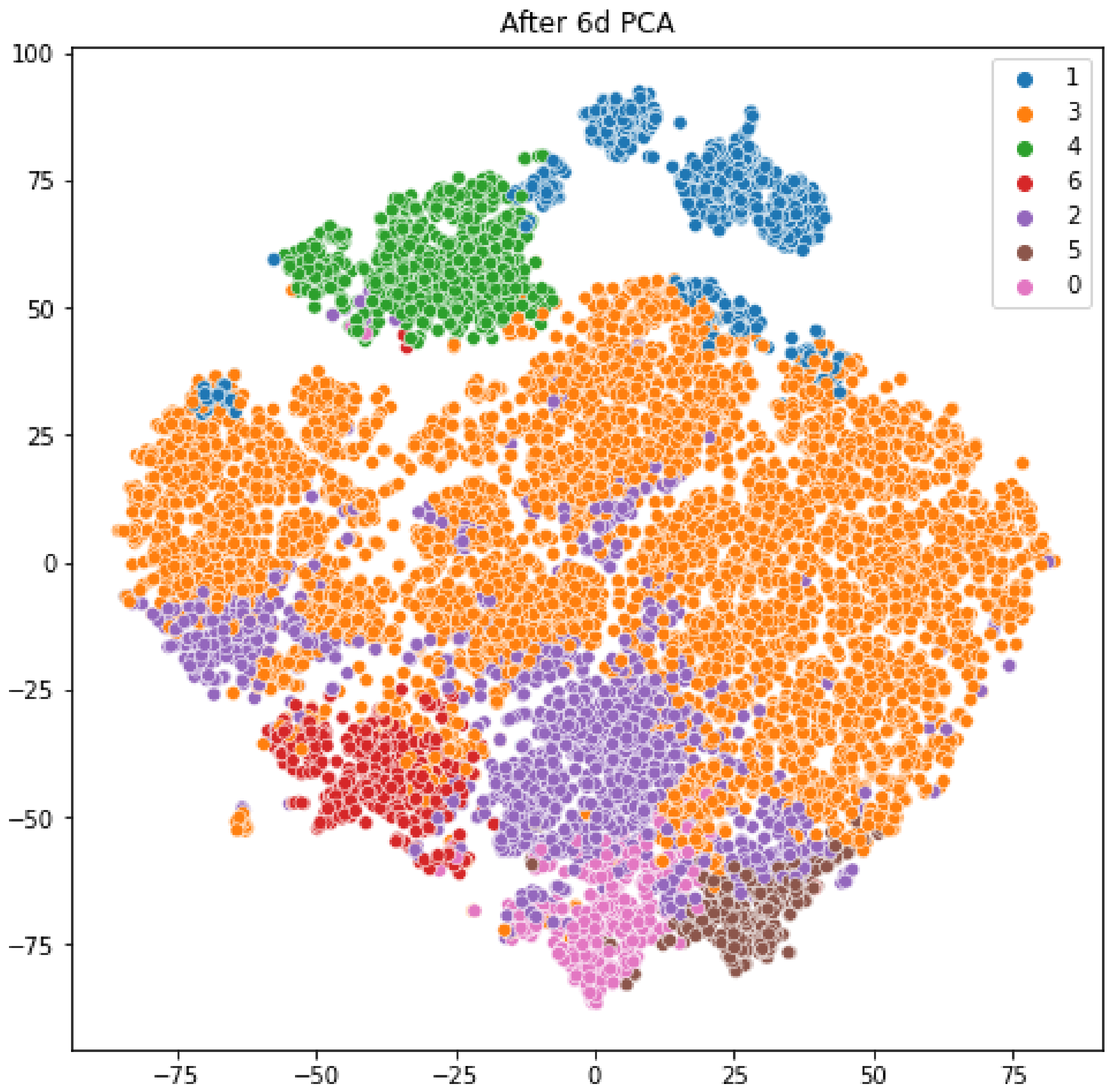
[17]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=6)
df_scale
pC = pca.fit_transform(df_scale)
```

[18]:
```
km=KMeans(n_clusters=7, random_state=40)
arr=km.fit_predict(pC)
lstFin=[]
for i in range(len(arr)):
    lstFin.append(str(arr[i]))
```

[19]:
```
plt.figure(figsize=(8,8))
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o', hue=lstFin).
 ↪set_title("After 6d PCA")
```

```
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[19]:
```
Text(0.5, 1.0, 'After 6d PCA')
```



[20]:
```
from sklearn.metrics import adjusted_mutual_info_score
```

[21]:
```
for i in range(len(df_scale.columns)):
    print(f"{df_scale.columns[i]}: {adjusted_mutual_info_score(df_scale.iloc[:,␣
↪i], lst)}" )
```

```
Administrative: 0.1911714225049339
Administrative_Duration: 0.08357875479756202
Informational: 0.19900853839620336
Informational_Duration: 0.12721766229391115
ProductRelated: 0.1484832761208349
ProductRelated_Duration: 0.03375928952606079
BounceRates: 0.1960404109937648
ExitRates: 0.1529290770498833
```

```
PageValues: 0.03903308724655664
SpecialDay: 0.03465679149476827
Weekend: 0.0019077495844321503
Month_Aug: 0.00043281847545738891
Month_Dec: 0.0013201786813762458
Month_Feb: 0.0059965174586356935
Month_Jul: -0.00012434541256417118
Month_June: 0.0010721039645300598
Month_Mar: 0.0041028422054351235
Month_May: 0.013325994803690044
Month_Nov: 0.011694108974036
Month_Oct: 0.0038509103447635046
Month_Sep: 0.0025558640288833433
OperatingSystems_1: 0.0018967400388558299
OperatingSystems_2: 0.006192047742414939
OperatingSystems_3: 0.005226088316196516
OperatingSystems_4: 0.0009054251071059689
OperatingSystems_5: 5.11214612070386e-05
OperatingSystems_6: -0.0001176003394583869
OperatingSystems_7: -0.0001758544446687089
OperatingSystems_8: 0.001428681474179667
Browser_1: 0.0019170214970306292
Browser_2: 0.0031384612891710464
Browser_3: 0.0007698906504514389
Browser_4: 0.000647880241687398
Browser_5: -9.490192761625973e-05
Browser_6: -0.00010936398219194529
Browser_7: -0.00023750635741884258
Browser_8: 0.001197804100517266
Browser_9: 6.491678839476614e-06
Browser_10: -0.00011812030452019429
Browser_11: 5.11214612070386e-05
Browser_12: 2.2857262405622196e-05
Browser_13: 0.0010735802791310874
Region_1: 0.0002848723945065966
Region_2: -0.0001588067440541682
Region_3: 0.00020220030320159757
Region_4: -0.00024060166687937123
Region_5: -0.00017319269008805635
Region_6: 0.00013021162837568358
Region_7: 0.0007571909728295049
Region_8: 0.00016202759130722752
Region_9: 0.00044147234789494423
TrafficType_1: 0.005766241169550866
TrafficType_2: 0.03716033350705932
TrafficType_3: 0.012724816831965016
TrafficType_4: 0.001185559959374187
TrafficType_5: 0.0019005192587427105
```

```
TrafficType_6: 0.00016667950741810342
TrafficType_7: 0.00024684160203799604
TrafficType_8: 0.00237075261655813
TrafficType_9: 0.00015213729765437166
TrafficType_10: 0.00043107863662806314
TrafficType_11: 0.000507622385613376
TrafficType_12: 3.1652726894261757e-05
TrafficType_13: 0.013509752840179204
TrafficType_14: 0.00020424964345374695
TrafficType_15: 0.001010274235980653
TrafficType_16: -0.00010231845781195034
TrafficType_17: 3.1652726894261757e-05
TrafficType_18: -0.00015010901086953923
TrafficType_19: -7.085641826230404e-05
TrafficType_20: 0.0006264591771941439
VisitorType_New_Visitor: 0.03714991247964353
VisitorType_Other: 0.0012437771428175843
VisitorType_Returning_Visitor: 0.03594368145144588
```

[22]:

```
pca = PCA(n_components=2)
pC2 = pca.fit_transform(df_scale)
km=KMeans(n_clusters=7, random_state=40)
km_pC2_pred=km.fit_predict(pC2)
lst=[]
for i in range(len(km_pC2_pred)):
    lst.append(str(km_pC2_pred[i]))
```

[23]:

```
pca = PCA(n_components=6)
pC6 = pca.fit_transform(df_scale)
km=KMeans(n_clusters=7, random_state=40)
km_pC6_pred=km.fit_predict(pC6)
lst=[]
for i in range(len(km_pC6_pred)):
    lst.append(str(km_pC6_pred[i]))
```

[24]:

```
km=KMeans(n_clusters=7, random_state=40)
km_pred=km.fit_predict(df_scale)
lst=[]
for i in range(len(km_pred)):
    lst.append(str(km_pred[i]))
```

[25]:

```
km=KMeans(n_clusters=7, random_state=50)
km_pred2=km.fit_predict(df_scale)
lst=[]
for i in range(len(km_pred)):
    lst.append(str(km_pred[i]))
```

[26]:
```
adjusted_mutual_info_score(km_pred,km_pred2)
```

[26]: 0.9872288327165795

0.5 Informacja wzajemna klastrowań przed i po PCA

0.5.1 PCA n_component = 6

[27]:
```
adjusted_mutual_info_score(km_pred,km_pC6_pred)
```

[27]: 0.9301404284248578

0.5.2 PCA n_component = 2

[28]:
```
adjusted_mutual_info_score(km_pred,km_pC2_pred)
```

[28]: 0.44937202726312775

0.6 Stabilność(?)

0.6.1 PCA n_component = 6

[29]:
```
pca = PCA(n_components=6)
pC6 = pca.fit_transform(df_scale)
mis = []
for i in range(20):
    km=KMeans(n_clusters=7)
    km_pC6_pred=km.fit_predict(pC6)

    km=KMeans(n_clusters=7)
    km_pred=km.fit_predict(df_scale)

    mis.append(adjusted_mutual_info_score(km_pred,km_pC6_pred))

print(f"Mean mutual info score of random non-PCA / PCA clusters: {np.
 ↪mean(mis)}")
print(f"Std of random non-PCA / PCA clusters: {np.std(mis)}")
```

```
Mean mutual info score of random non-PCA / PCA clusters: 0.9276728645726207
Std of random non-PCA / PCA clusters: 0.011029566912655437
```

0.6.2 PCA n_component = 2

[30]:
```
pca = PCA(n_components=2)
pC6 = pca.fit_transform(df_scale)
mis = []
for i in range(20):
```

```
    km=KMeans(n_clusters=7)
    km_pC2_pred=km.fit_predict(pC2)

    km=KMeans(n_clusters=7)
    km_pred=km.fit_predict(df_scale)

    mis.append(adjusted_mutual_info_score(km_pred,km_pC2_pred))

print(f"Mean mutual info score of random non-PCA / PCA clusters: {np.
 ↪mean(mis)}")
print(f"Std of random non-PCA / PCA clusters: {np.std(mis)}")
```

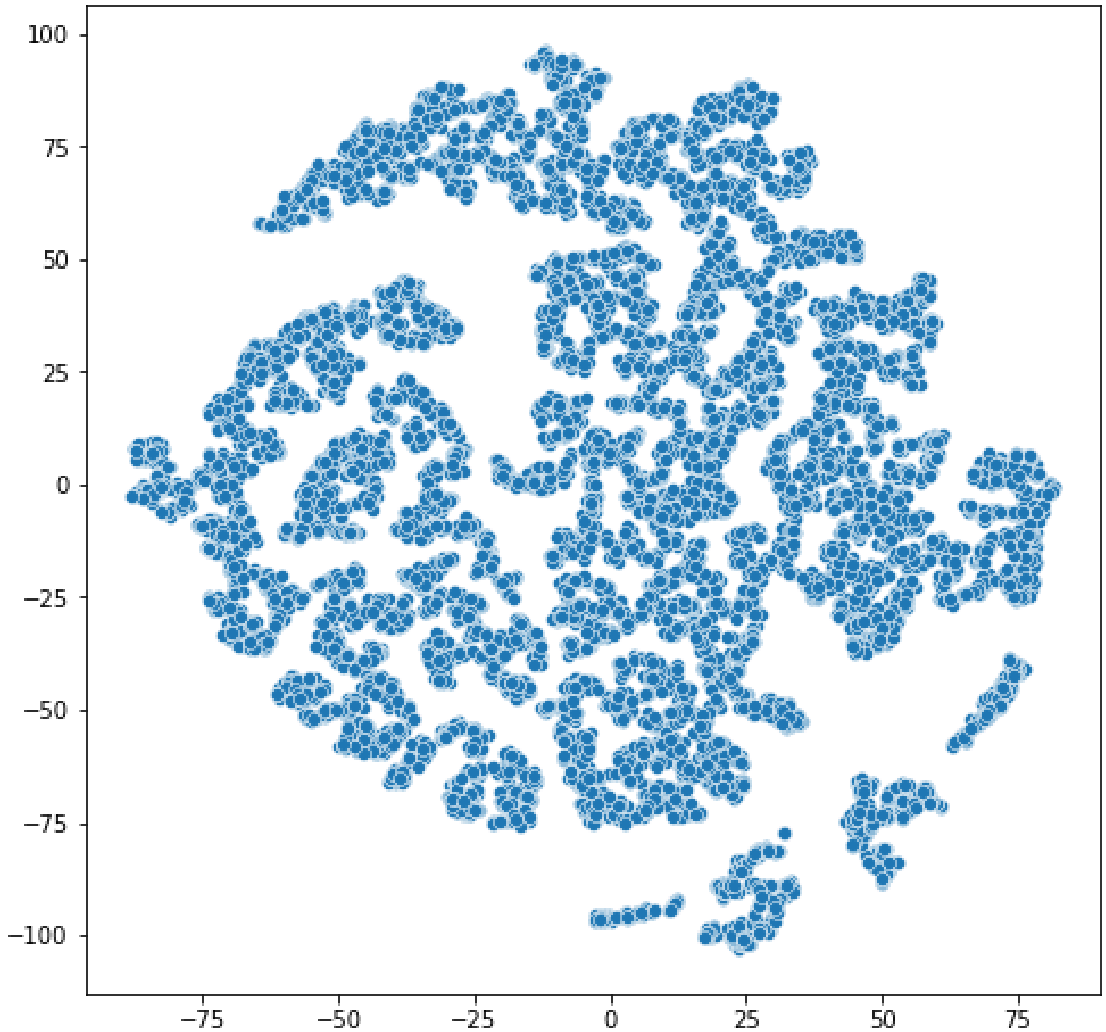
```
Mean mutual info score of random non-PCA / PCA clusters: 0.4473783793300722
Std of random non-PCA / PCA clusters: 0.0017477419495951722
```

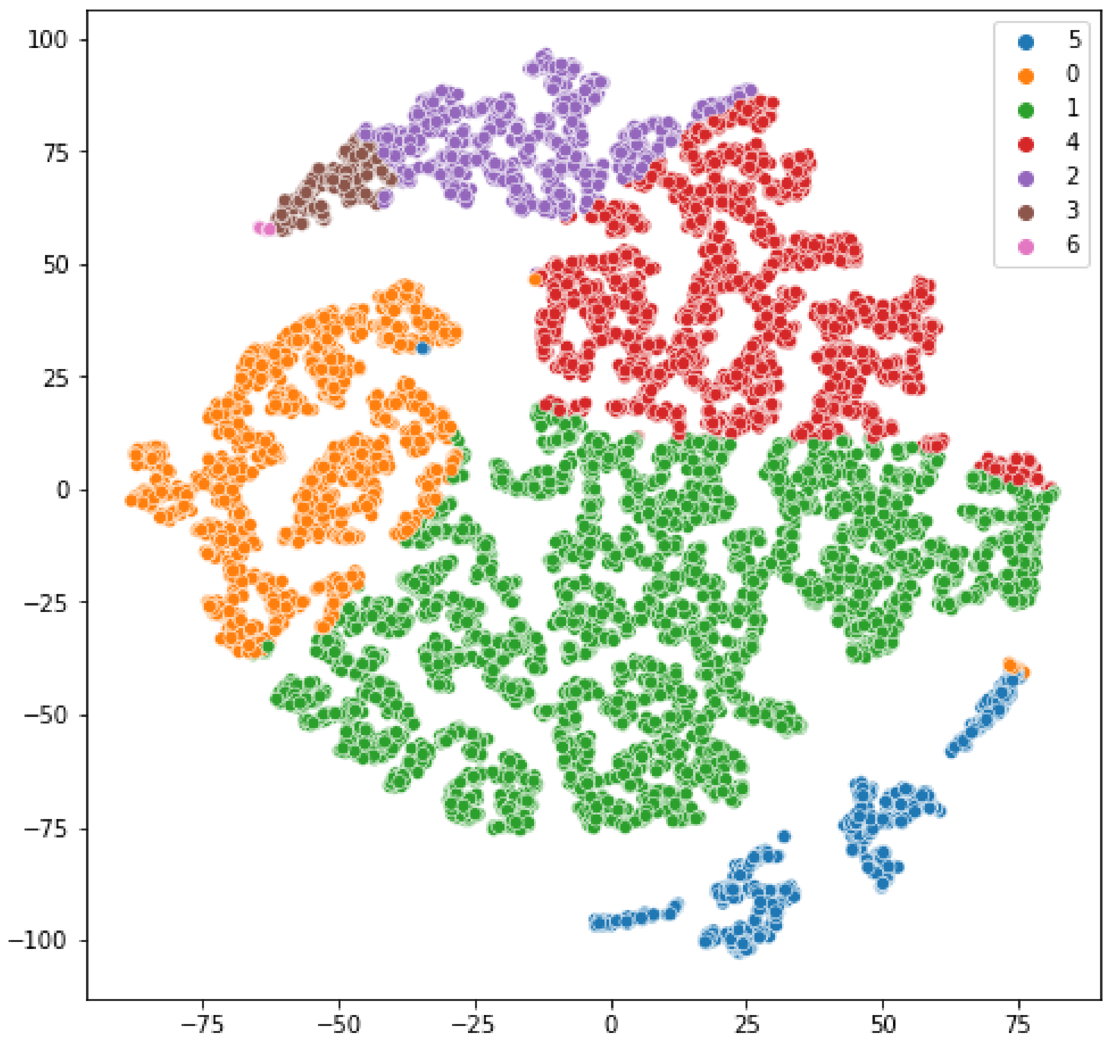
0.7 TSNE na PCA (tylko żeby spróbwać)

[31]:
```
pca = PCA(n_components=2)
pC = pca.fit_transform(df_scale)
tsne = TSNE(n_components=2, perplexity=35, random_state=40)
coordsFIN=tsne.fit_transform(pC)
plt.figure(figsize=(8,8))
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o')
km=KMeans(n_clusters=7, random_state=40)
arr=km.fit_predict(pC)
lst=[]
for i in range(len(arr)):
    lst.append(str(arr[i]))
plt.figure(figsize=(8,8))
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o', hue=lst)
```

```
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[31]: <AxesSubplot:>


100
75
50
25
0
−25
−50
−75
−100
−75
−50
−25
0
25
50
75

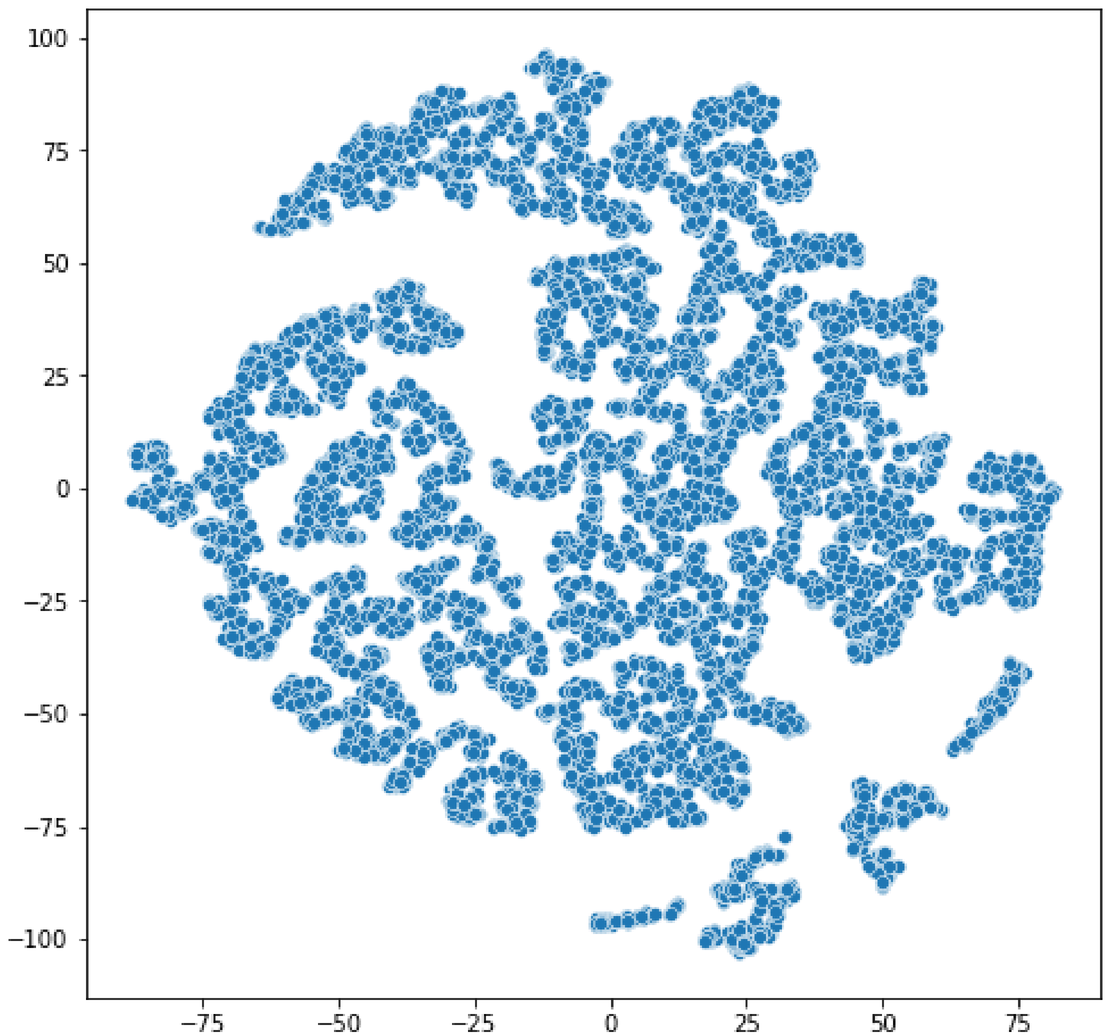


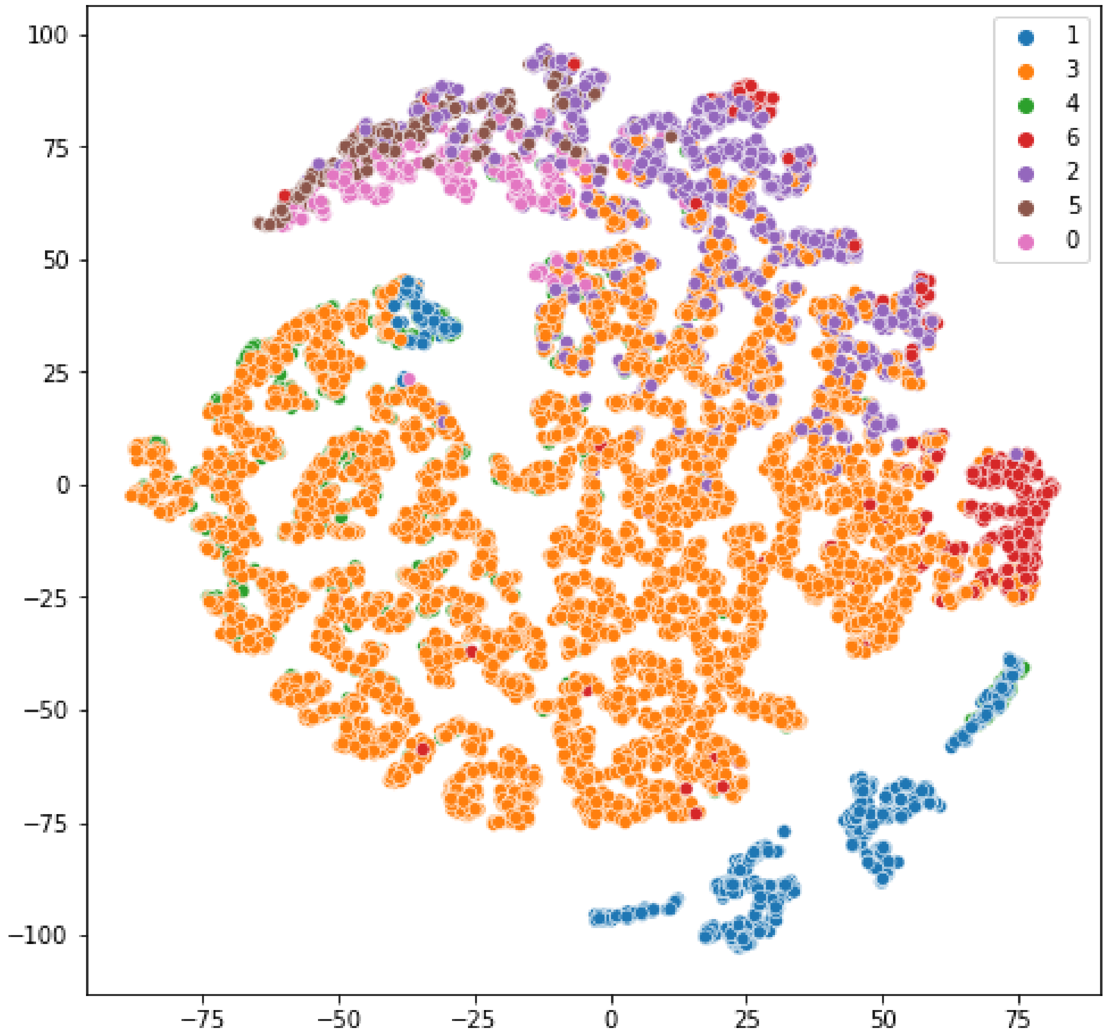
[32]:
```
pca = PCA(n_components=6)
pC = pca.fit_transform(df_scale)
tsne = TSNE(n_components=2, perplexity=35, random_state=40)
#coordsFIN=tsne.fit_transform(pC)
plt.figure(figsize=(8,8))
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o')
km=KMeans(n_clusters=7, random_state=40)
arr=km.fit_predict(pC)
lst=[]
for i in range(len(arr)):
    lst.append(str(arr[i]))
plt.figure(figsize=(8,8))
sns.scatterplot(coordsFIN[:, 0], coordsFIN[:, 1], marker = 'o', hue=lst)
```

```
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
```

```
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[32]: <AxesSubplot:>





0.8 Charakterystyka klastrów

0.9 Korelacje

0.10 Informacja wzajemna - all

[33]:
```
import sklearn.metrics as metrics
def mtr(x, y):
    return metrics.normalized_mutual_info_score(x, y)
```

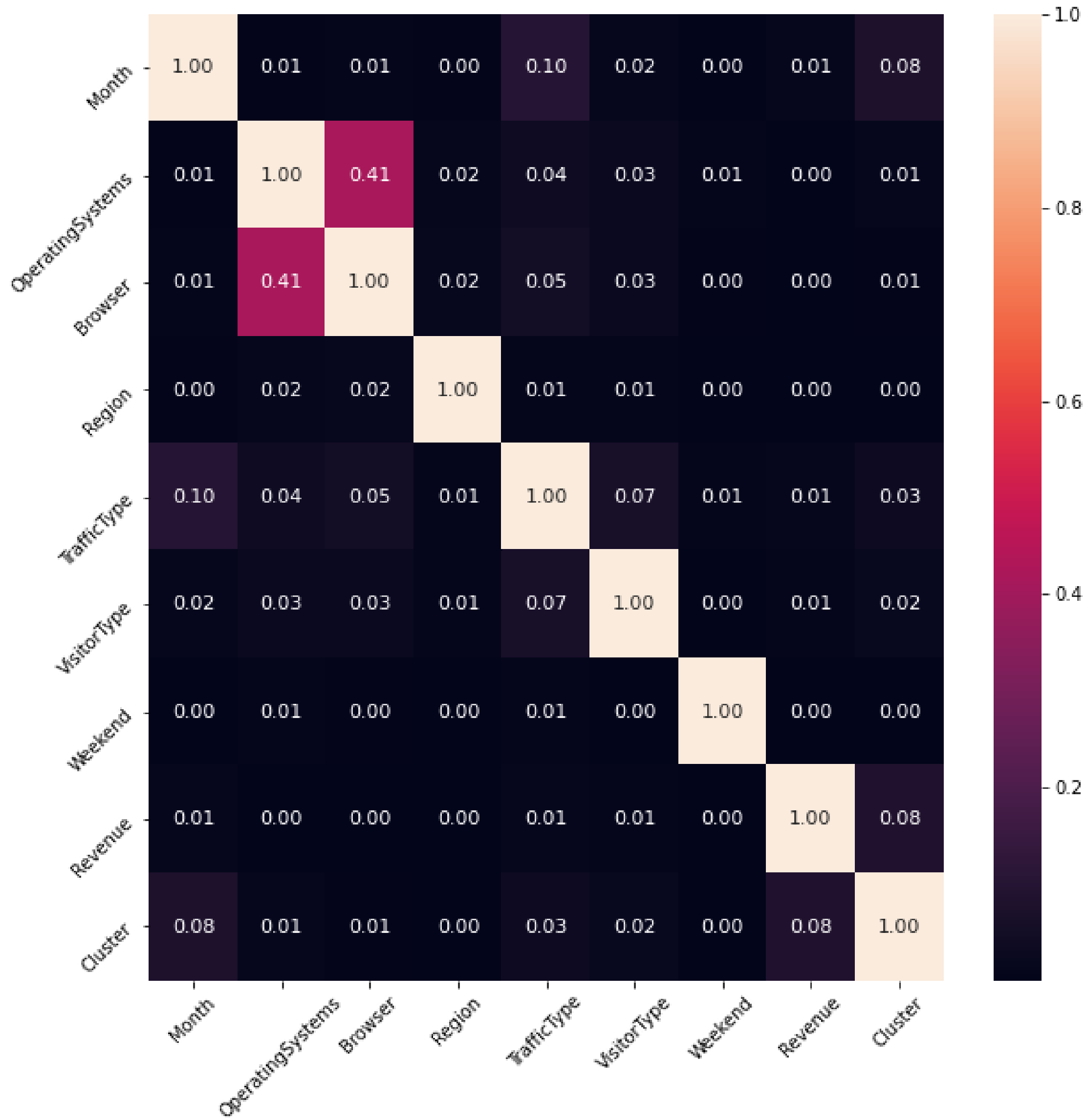
[34]:
```
df=pd.read_csv("online_shoppers_intention.csv")
df=df.dropna()
X=df.copy().drop(nums, axis=1)
cats=["Month", "OperatingSystems" ,"Browser" ,"Region" ,␣
 ↪"TrafficType","VisitorType","Weekend","Revenue"]
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
for n in cats:
    X[n]=le.fit_transform(X[n])
```

[35]:
```
X["Cluster"]=lst
cats.append("Cluster")
dst=metrics.pairwise_distances(X.T, metric=mtr)
```

[36]:
```
plt.figure(figsize=(10, 10))
p=sns.heatmap(dst, annot=True, annot_kws={'size': 11}, fmt='.2f')
p.set_xticklabels(cats, rotation=45)
p.set_yticklabels(cats, rotation=45)
```

[36]:
```
[Text(0, 0.5, 'Month'),
 Text(0, 1.5, 'OperatingSystems'),
 Text(0, 2.5, 'Browser'),
 Text(0, 3.5, 'Region'),
 Text(0, 4.5, 'TrafficType'),
 Text(0, 5.5, 'VisitorType'),
 Text(0, 6.5, 'Weekend'),
 Text(0, 7.5, 'Revenue'),
 Text(0, 8.5, 'Cluster')]
```



0.11 Informacja wzajemna - szczegółowo

[37]:
```
X=df.copy().drop(nums, axis=1)
X["Cluster"]=lst
cats=["Month", "OperatingSystems" ,"Browser" ,"Region" ,␣
↪"TrafficType","VisitorType","Weekend","Revenue"]
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
for n in cats:
    X[n]=le.fit_transform(X[n])
```

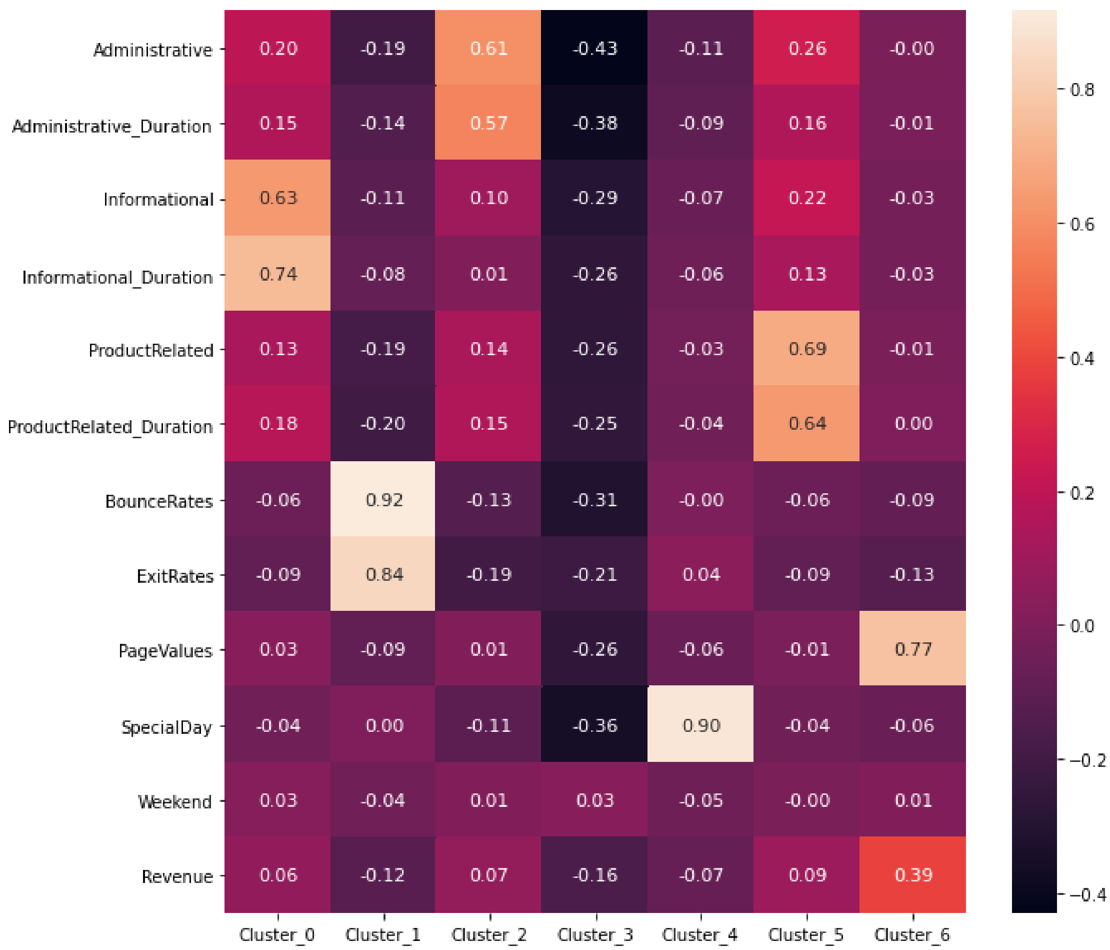
```
X=pd.get_dummies(X, columns=["Cluster"] )
cls=["Cluster_0","Cluster_1", "Cluster_2", "Cluster_3", "Cluster_4",␣
↪"Cluster_5", "Cluster_6" ]


cats2=np.concatenate((cats, cls))
dst=pd.DataFrame(metrics.pairwise_distances(X.T, metric=mtr))
dst["cats"]=cats2
dst=dst.set_index("cats")
dst.columns=cats2
dst=dst[cls]
dst=dst.iloc[dst.index.isin(cats)]

plt.figure(figsize=(10, 10))
sns.heatmap(dst, annot=True, annot_kws={'size': 11}, fmt='.2f')
```

[37]: <AxesSubplot:ylabel='cats'>

0.12 Korelacja

[38]:
```
from scipy import stats
X=df.copy().dropna()
X["Cluster"]=lst
nums=["Administrative", "Administrative_Duration",
      "Informational", "Informational_Duration",
      "ProductRelated", "ProductRelated_Duration",
     "BounceRates", "ExitRates", "PageValues", "SpecialDay"]
times=["Administrative_Duration", "Informational_Duration",␣
 ↪"ProductRelated_Duration"]
X=pd.get_dummies(X, columns=["Cluster"] )
cls=["Cluster_0","Cluster_1", "Cluster_2", "Cluster_3", "Cluster_4",␣
 ↪"Cluster_5", "Cluster_6" ]
X[times] = X[times][X<X.quantile(0.997)]
corr=X.corr()[cls]
nums.append("Revenue")
nums.append("Weekend")
X=X.replace(True, 1)
X=X.replace(False, 0)
corr=corr.iloc[corr.index.isin(nums)]
plt.figure(figsize=(10, 10))
sns.heatmap(corr, annot=True, annot_kws={'size': 11}, fmt='.2f')
```

[38]: <AxesSubplot:>

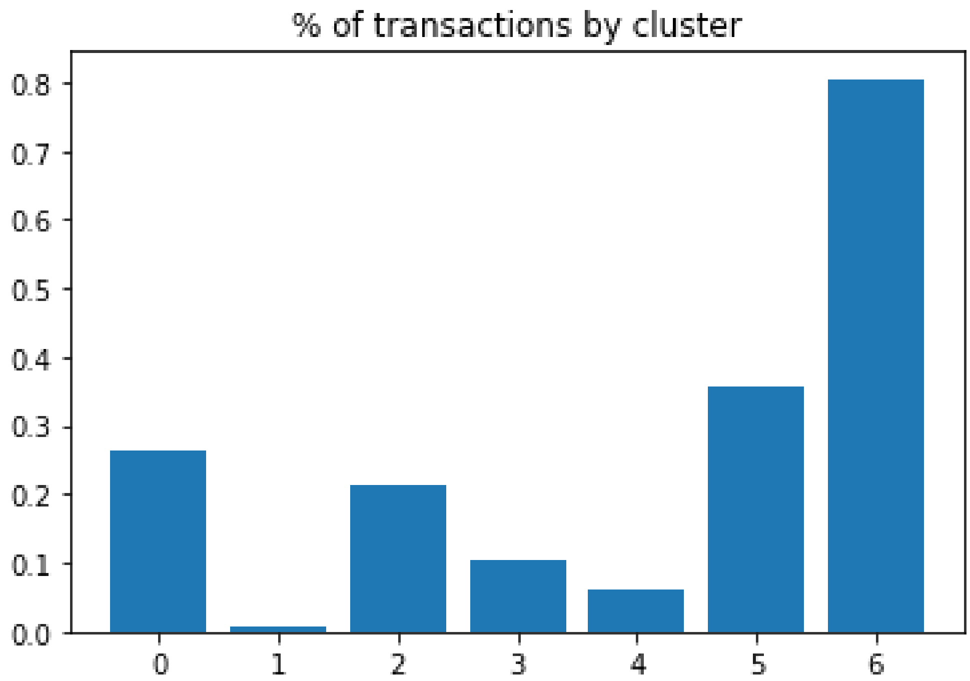


0.13 charakterystyka klastrów

0.13.1 Czas

[39]:
```
df=pd.read_csv("online_shoppers_intention.csv")
df_scale2=df.dropna().copy()
df_scale_lbs=df_scale2.copy()
df_scale_lbs["lbs"]=lst
tmp=pd.DataFrame()
tmp["lab"]=[]
tmp["score"]=[]
tmp=df_scale_lbs.groupby("lbs").agg({'Revenue':['sum', "count"]}).reset_index()
tmp.columns=["lbs", "sum", "count"]
tmp["prc"]=tmp["sum"]/tmp["count"]
plt.bar(tmp["lbs"], tmp["prc"])
plt.title("% of transactions by cluster")
```

[39]: Text(0.5, 1.0, '% of transactions by cluster')



0.13.2 Months

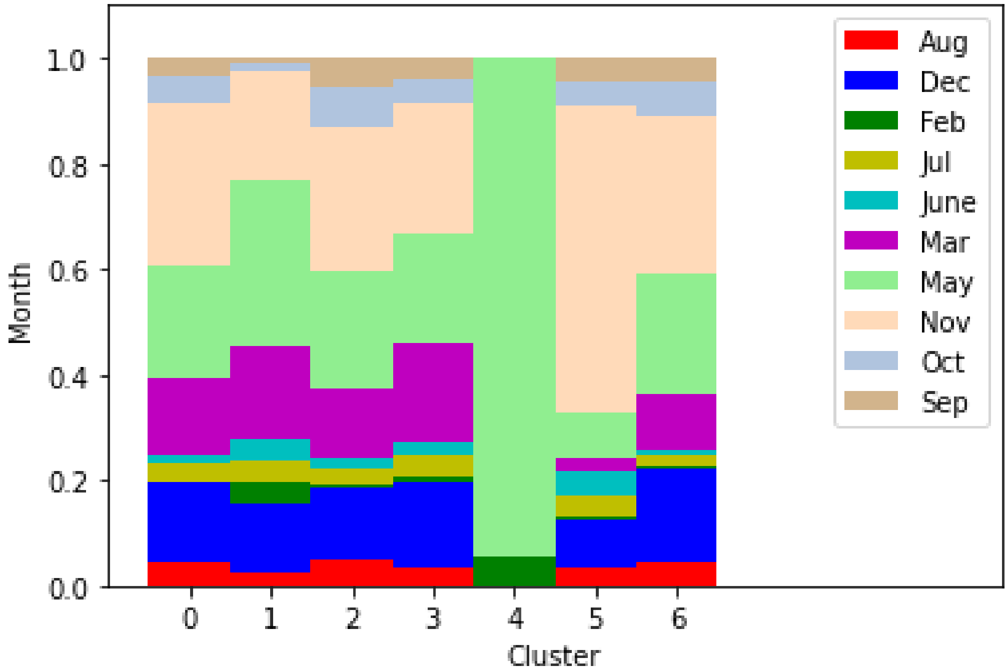
[40]:
```
X=df.dropna().copy()
X["lbs"]=lst
X
X=X[["Month", "lbs"]]
X=X.groupby(["Month", "lbs"]).agg({'Month':["count"]}).reset_index()
X
X.columns=["Month", "lbs", "count"]
X=X.pivot(columns="Month", index="lbs", values="count")
X=X.fillna(0)

X=X.T
for i in X.columns:
    X[i] = X[i]/X[i].sum()
X=X.T
#X.columns=["1",  "2", "3", "4", "5", "6", "7", "8"]
X=X.reset_index().drop(["lbs"], axis=1)
X
```

[40]:
```
Month       Aug       Dec       Feb       Jul      June       Mar       May  \
0      0.042596  0.154158  0.000000  0.036511  0.012170  0.148073  0.210953
```

```
1       0.024044  0.131148  0.042623  0.039344  0.040437  0.178142  0.311475
2       0.048116  0.139130  0.001739  0.032464  0.022029  0.132174  0.221449
3       0.036619  0.158045  0.012115  0.040158  0.025320  0.185543  0.211271
4       0.000000  0.000000  0.054604  0.000000  0.000000  0.000000  0.945396
5       0.034384  0.091691  0.002865  0.040115  0.045845  0.028653  0.083095
6       0.046931  0.176895  0.001805  0.023466  0.009025  0.102888  0.231047

Month        Nov       Oct       Sep
0       0.310345  0.048682  0.036511
1       0.209836  0.013115  0.009836
2       0.272464  0.077101  0.053333
3       0.246937  0.044650  0.039341
4       0.000000  0.000000  0.000000
5       0.581662  0.045845  0.045845
6       0.299639  0.064982  0.043321
```

[41]:

```
fig, ax = plt.subplots()
bot=[0]*7
labs=["0", "1", "2", "3", "4", "5", "6"]
colors=["r", "b", "g", "y", "c", "m", "lightgreen", "peachpuff",␣
 ↪"lightsteelblue", "tan", "violet" ]
width=1
for i in range(len(X.columns)):
    ax.bar(labs, X[X.columns[i]],  width, bottom=bot, color=colors[i], label=X.
 ↪columns[i])
    bot+=X[X.columns[i]]

ax.legend()
ax.set_ylabel('Month')
ax.set_xlabel('Cluster')
ax.set_ylim([0, 1.1])
ax.set_xlim([-1, 10])
plt.show()
```

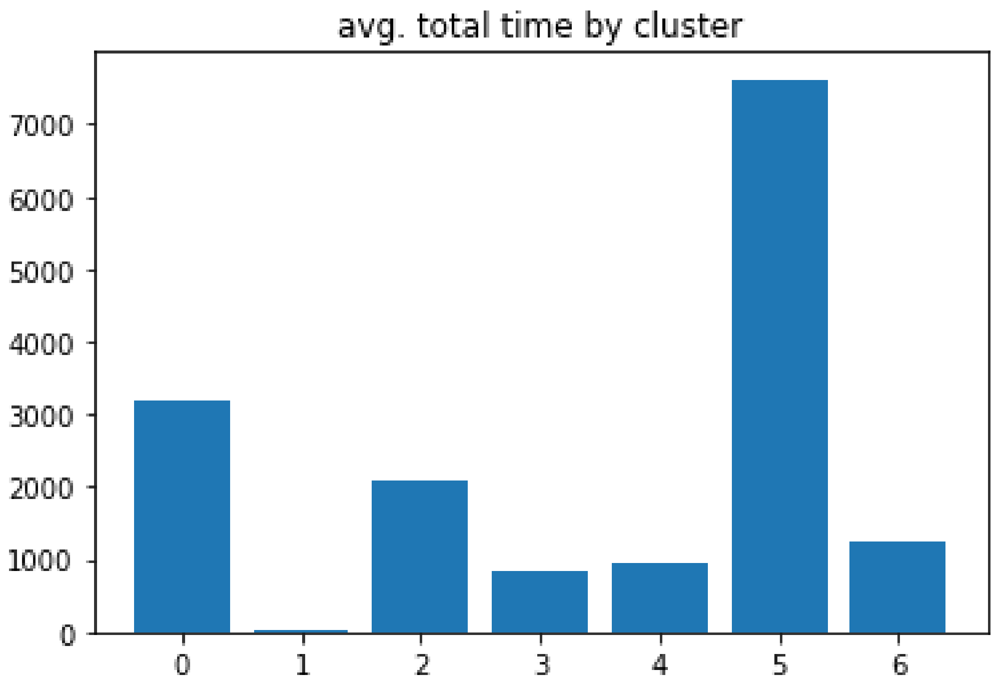


0.14 Czas

[42]:
```
from scipy import stats




df_scale_lbs=df_scale.dropna().copy()
 # without outliers
df_scale_lbs["Time"]=df["ProductRelated_Duration"]+df["Administrative_Duration"]+df["Informati
x = df_scale_lbs["Time"]
ind=(x<x.quantile(.997)).to_numpy()
lst_outliers=np.array(lst)


lst_outliers=lst_outliers[ind]

df_scale_lbs = df_scale_lbs[ind]



df_scale_lbs["lbs"]=lst_outliers
df_scale_lbs[df_scale_lbs["Time"]<0]=0
df_scale_lbs["Time"]=pd.to_numeric(df_scale_lbs["Time"])
tmp=df_scale_lbs.groupby("lbs").agg({'Time':['sum', "count"]}).reset_index()
```

```
tmp.columns=["lbs", "sum", "count"]
tmp["prc"]=tmp["sum"]/tmp["count"]
tmp["lbs"]=pd.to_numeric(tmp["lbs"])
plt.bar(tmp["lbs"], tmp["prc"])
plt.title("avg. total time by cluster")
```

[42]: Text(0.5, 1.0, 'avg. total time by cluster')



[ ]: