EDA

May 11, 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
```

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

Ramka danych składa się z następujących kolumn:

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
```

```
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

[4]:
```
data.head()
```

[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType	Weekend	Revenue
0	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0	0.0	Feb	1	1	1	1	Returning_Visitor	False	False
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	0.0	0.0	Feb	2	2	1	2	Returning_Visitor	False	False
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0	0.0	Feb	4	1	9	3	Returning_Visitor	False	False
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	0.0	0.0	Feb	3	2	2	4	Returning_Visitor	False	False
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	0.0	0.0	Feb	3	3	1	4	Returning_Visitor	True	False

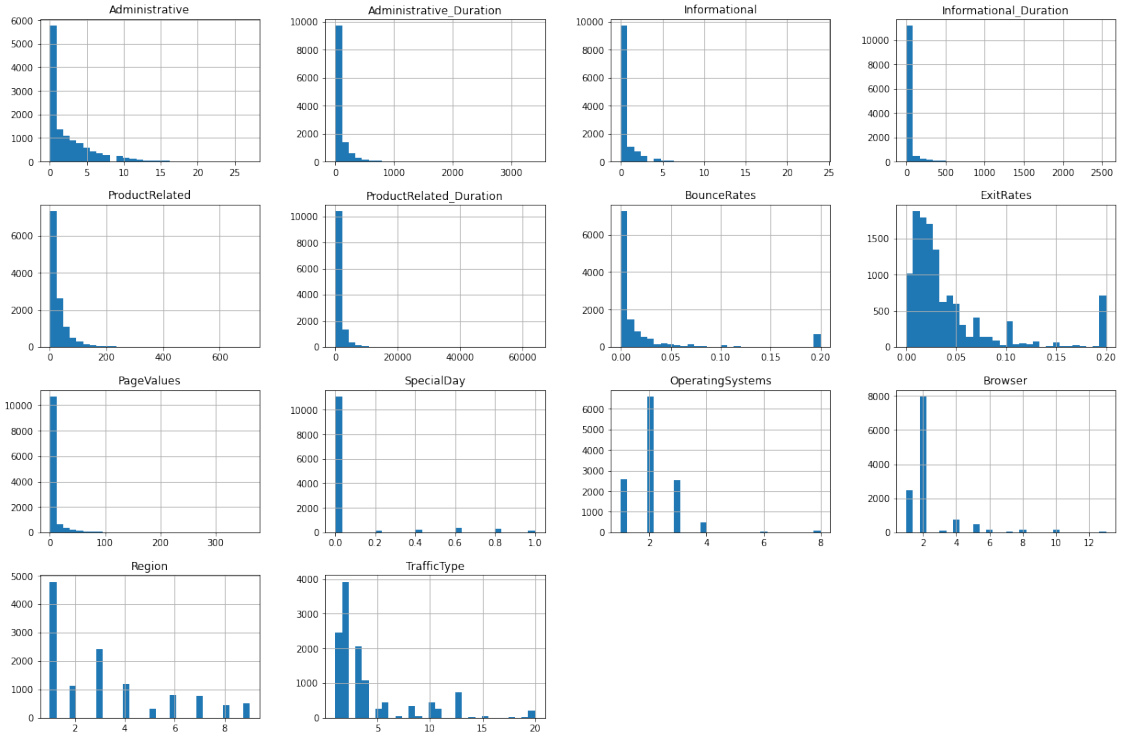
[5]:
```
data.describe()
```

[5]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.315166	80.818611	0.503569	34.472398	31.731468	1194.746220
std	3.321784	176.779107	1.270156	140.749294	44.475503	1913.669288
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	7.000000	184.137500
50%	1.000000	7.500000	0.000000	0.000000	18.000000	598.936905
75%	4.000000	93.256250	0.000000	0.000000	38.000000	1464.157213
max	27.000000	3398.750000	24.000000	2549.375000	705.000000	63973.522230

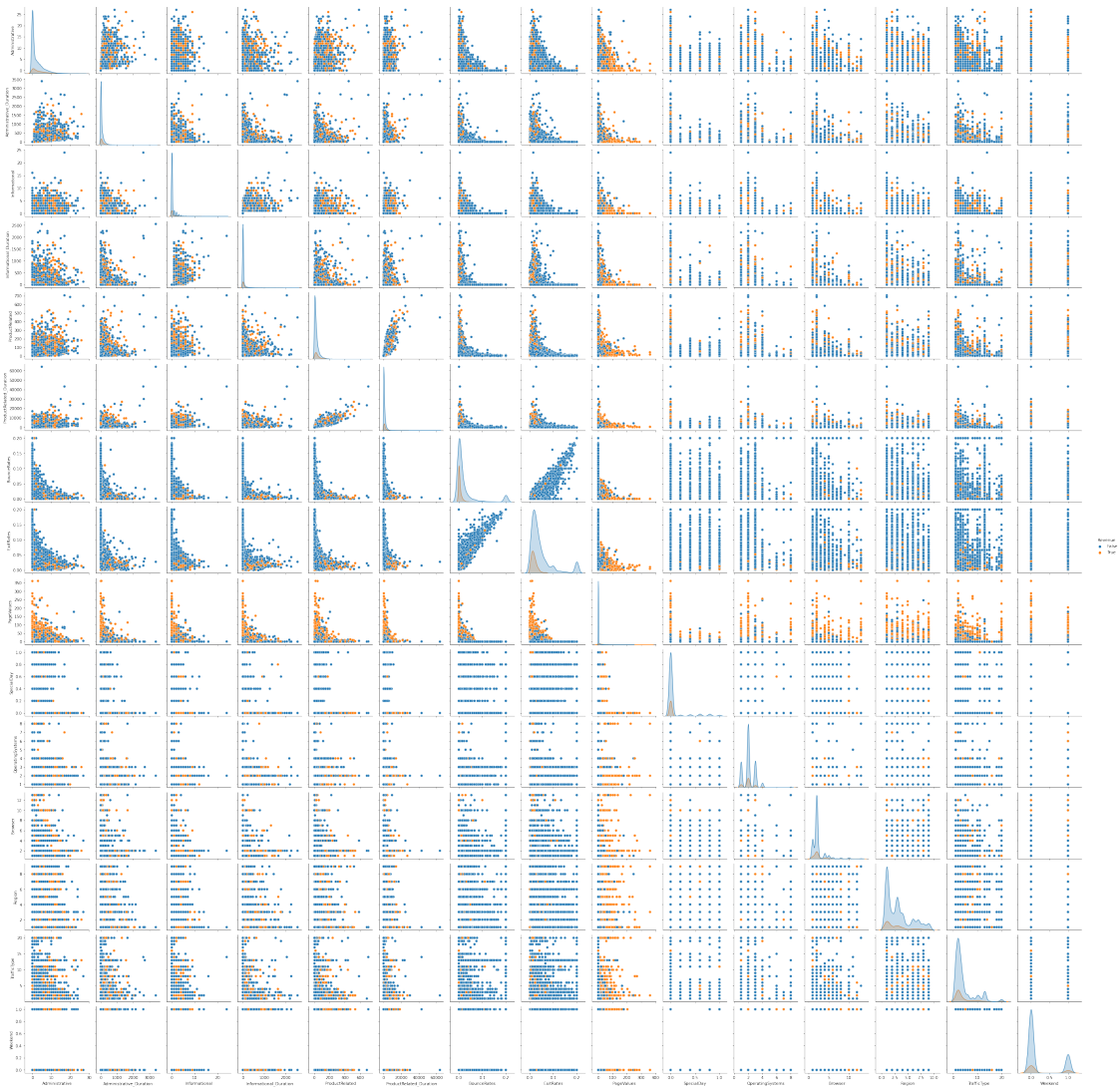
```
mean                34.472398       31.731468             1194.746220
std                140.749294       44.475503             1913.669288
min                  0.000000        0.000000                0.000000
25%                  0.000000        7.000000              184.137500
50%                  0.000000       18.000000              598.936905
75%                  0.000000       38.000000             1464.157214
max               2549.375000      705.000000            63973.522230

        BounceRates     ExitRates    PageValues    SpecialDay  \
count  12330.000000  12330.000000  12330.000000  12330.000000
mean       0.022191      0.043073      5.889258      0.061427
std        0.048488      0.048597     18.568437      0.198917
min        0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.014286      0.000000      0.000000
50%        0.003112      0.025156      0.000000      0.000000
75%        0.016813      0.050000      0.000000      0.000000
max        0.200000      0.200000    361.763742      1.000000

       OperatingSystems       Browser        Region   TrafficType
count      12330.000000  12330.000000  12330.000000  12330.000000
mean           2.124006      2.357097      3.147364      4.069586
std            0.911325      1.717277      2.401591      4.025169
min            1.000000      1.000000      1.000000      1.000000
25%            2.000000      2.000000      1.000000      2.000000
50%            2.000000      2.000000      3.000000      2.000000
75%            3.000000      2.000000      4.000000      4.000000
max            8.000000     13.000000      9.000000     20.000000
```

[6]:
```
data.hist(bins=30, figsize=(21,14))
plt.show()
```

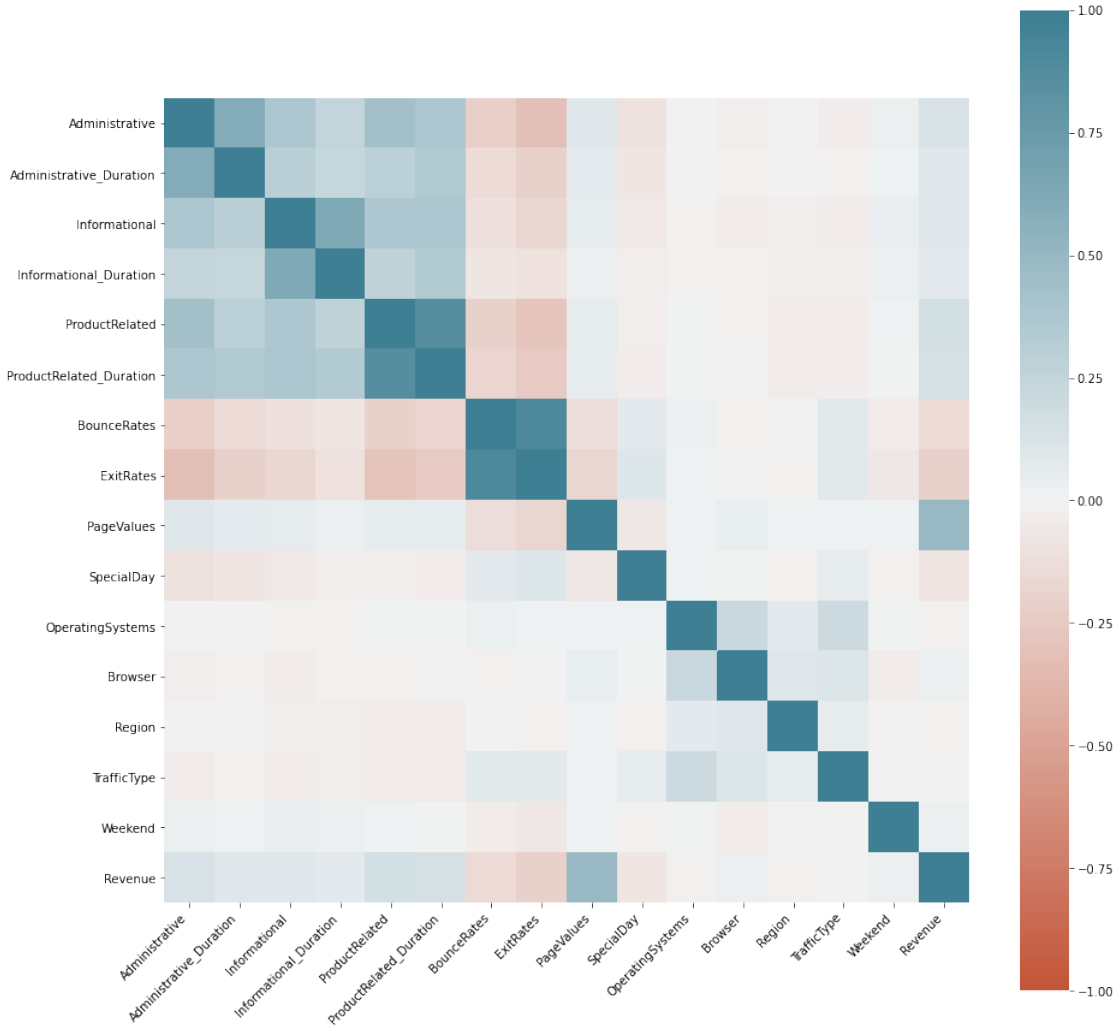


Na histogramach widzimy, że prawie wszystkie zmienne są mocno skośne do zera.

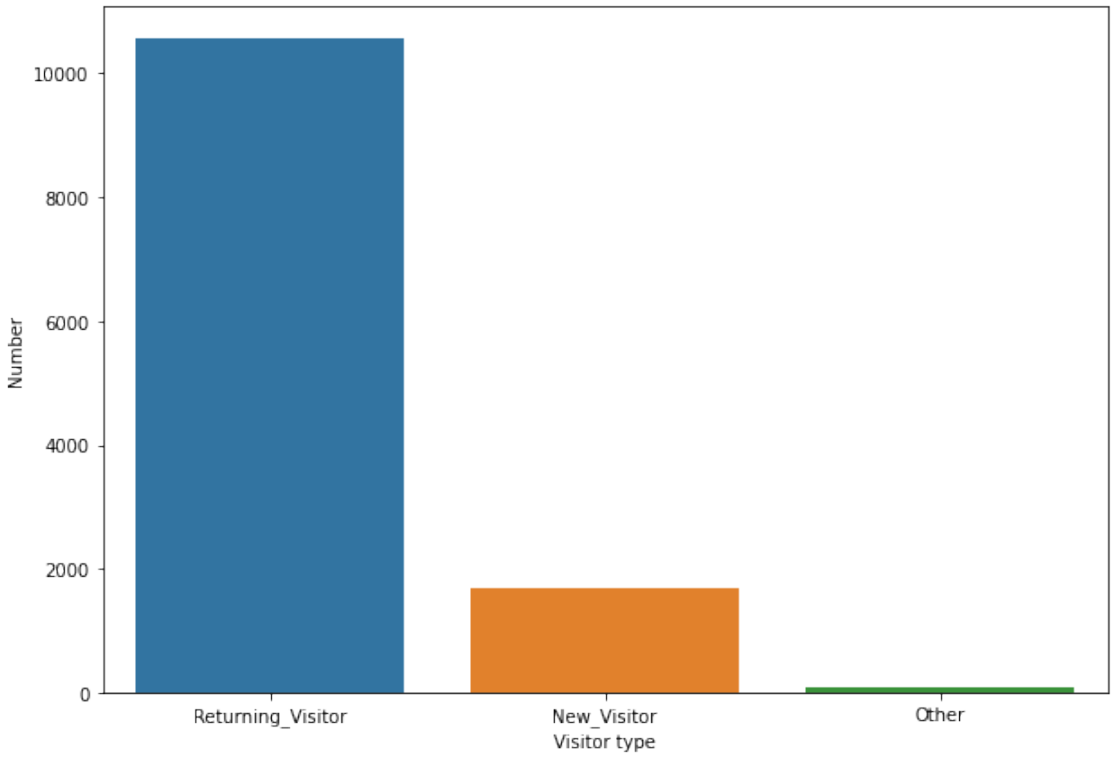
[7]:
```
sns.pairplot(data, hue='Revenue')
plt.show()
```



[8]:
```
corr = data.corr()
f, ax = plt.subplots(figsize=(15, 15))
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
)
plt.show()
```

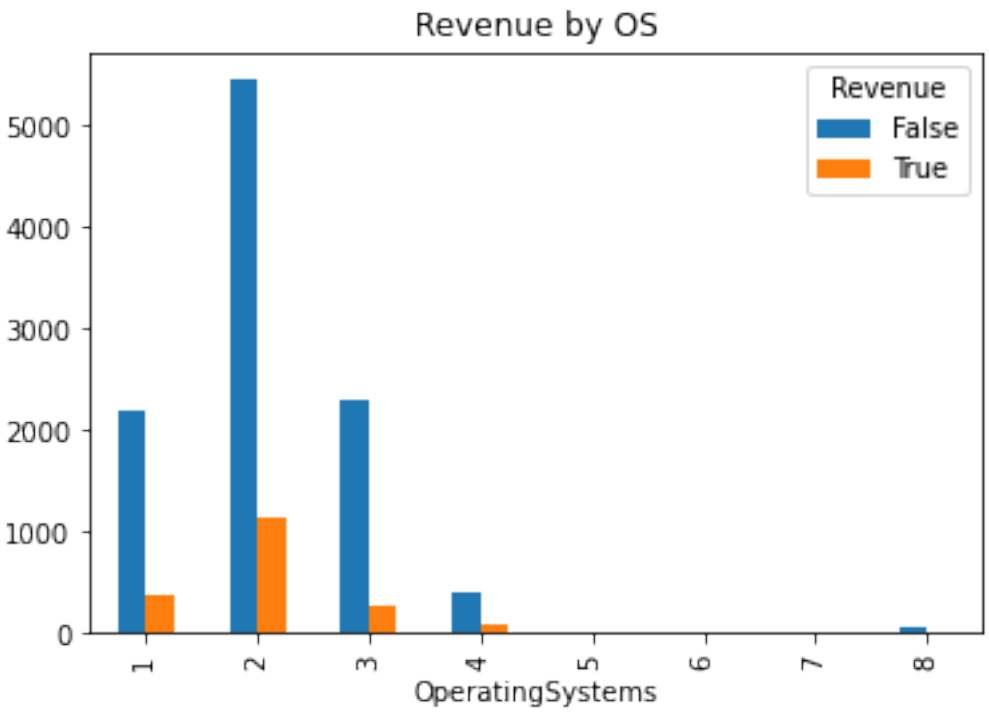


[9]:
```
plt.figure(figsize = (10, 7))
sns.countplot(data = data, x = "VisitorType")
plt.xlabel("Visitor type")
plt.ylabel("Number")
plt.show()
```



[10]:
```
plt.figure(figsize=(20,20))
data.groupby("OperatingSystems")["Revenue"].value_counts().unstack().
 ↪plot(kind="bar")
plt.title("Revenue by OS")
plt.show()

percentage = data.groupby("OperatingSystems")["Revenue"].value_counts().
 ↪unstack()
percentage["procent"] = percentage[True] / (percentage[True] +␣
 ↪percentage[False] )
percentage
```

```
<Figure size 1440x1440 with 0 Axes>
```

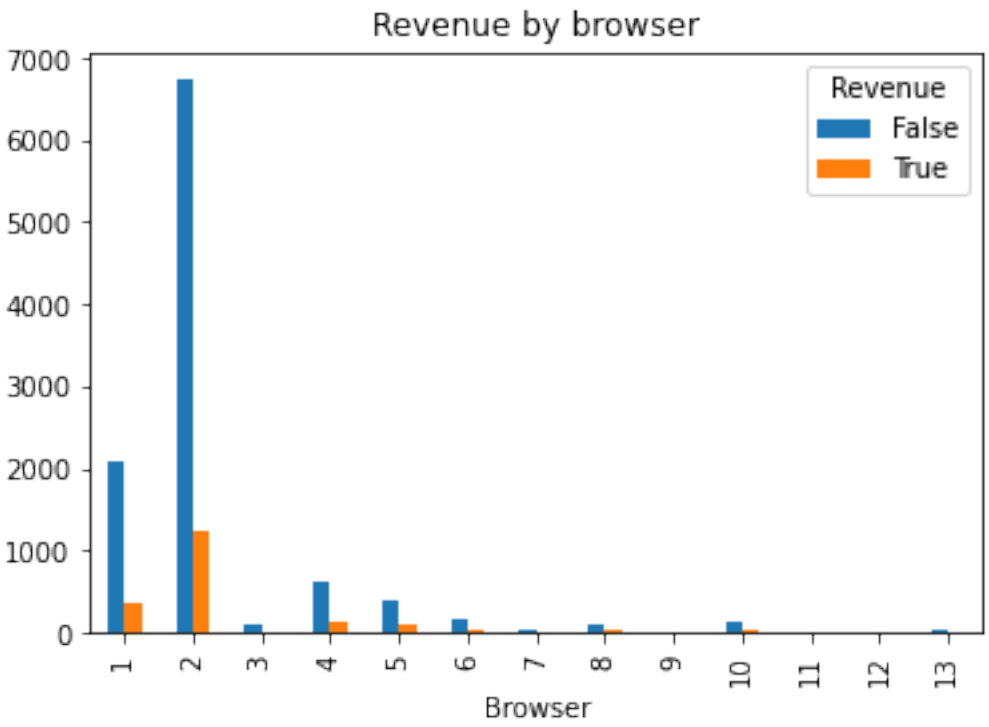


[10]:

Revenue	False	True	procent
OperatingSystems			
1	2206	379	0.146615
2	5446	1155	0.174973
3	2287	268	0.104892
4	393	85	0.177824
5	5	1	0.166667
6	17	2	0.105263
7	6	1	0.142857
8	62	17	0.215190

[11]:
```
plt.figure(figsize = (10,10))
data.groupby("Browser")["Revenue"].value_counts().unstack().plot(kind="bar")
plt.title("Revenue by browser")
plt.show()

percentage = data.groupby("Browser")["Revenue"].value_counts().unstack()


percentage["procent"] = percentage[True] / (percentage[True] +␣
 ↪percentage[False] )
percentage
```

```
<Figure size 720x720 with 0 Axes>
```



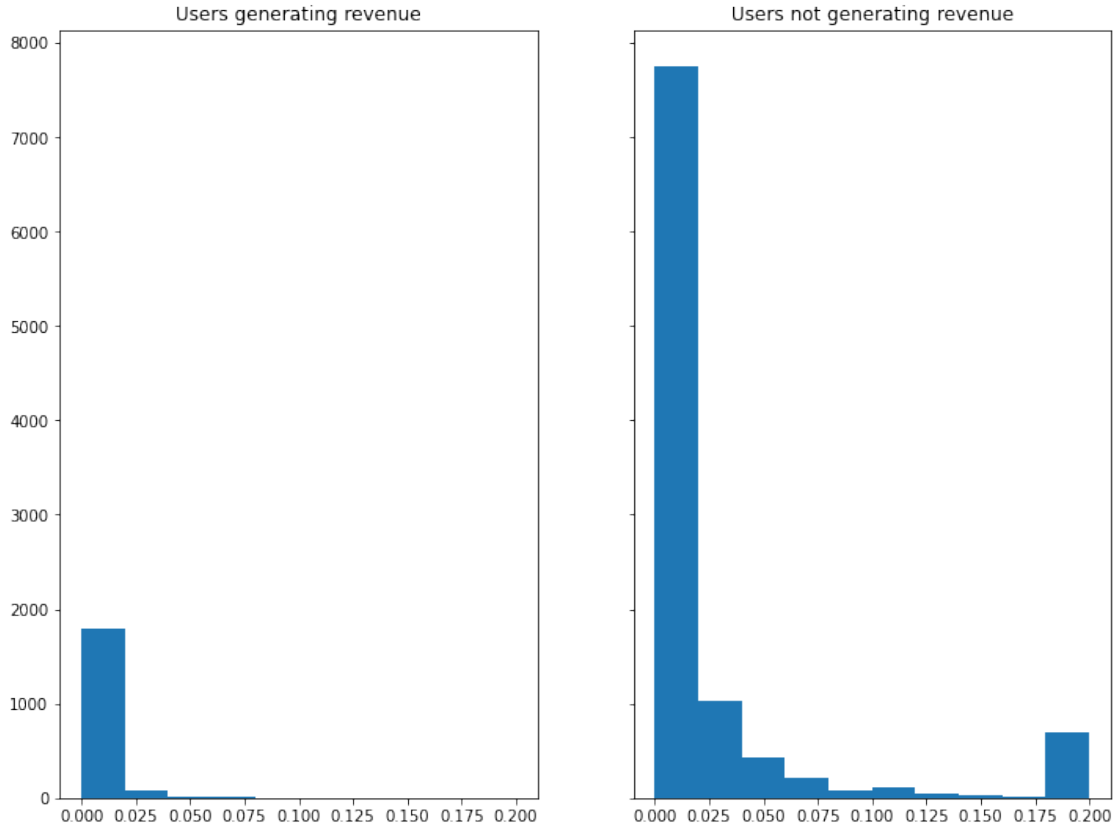
[11]:

Revenue	False	True	procent
Browser			
1	2097.0	365.0	0.148253
2	6738.0	1223.0	0.153624
3	100.0	5.0	0.047619
4	606.0	130.0	0.176630
5	381.0	86.0	0.184154
6	154.0	20.0	0.114943
7	43.0	6.0	0.122449
8	114.0	21.0	0.155556
9	1.0	NaN	NaN
10	131.0	32.0	0.196319
11	5.0	1.0	0.166667
12	7.0	3.0	0.300000
13	45.0	16.0	0.262295

[12]:
```
istrue = data["Revenue"]== True

fig, (ax1, ax2) = plt.subplots(1,2, sharey=True, figsize=(12,9))

ax1.hist(data.loc[istrue, "BounceRates"])
ax1.set_title("Users generating revenue")
ax2.hist(data.loc[istrue==False, "BounceRates"])
```

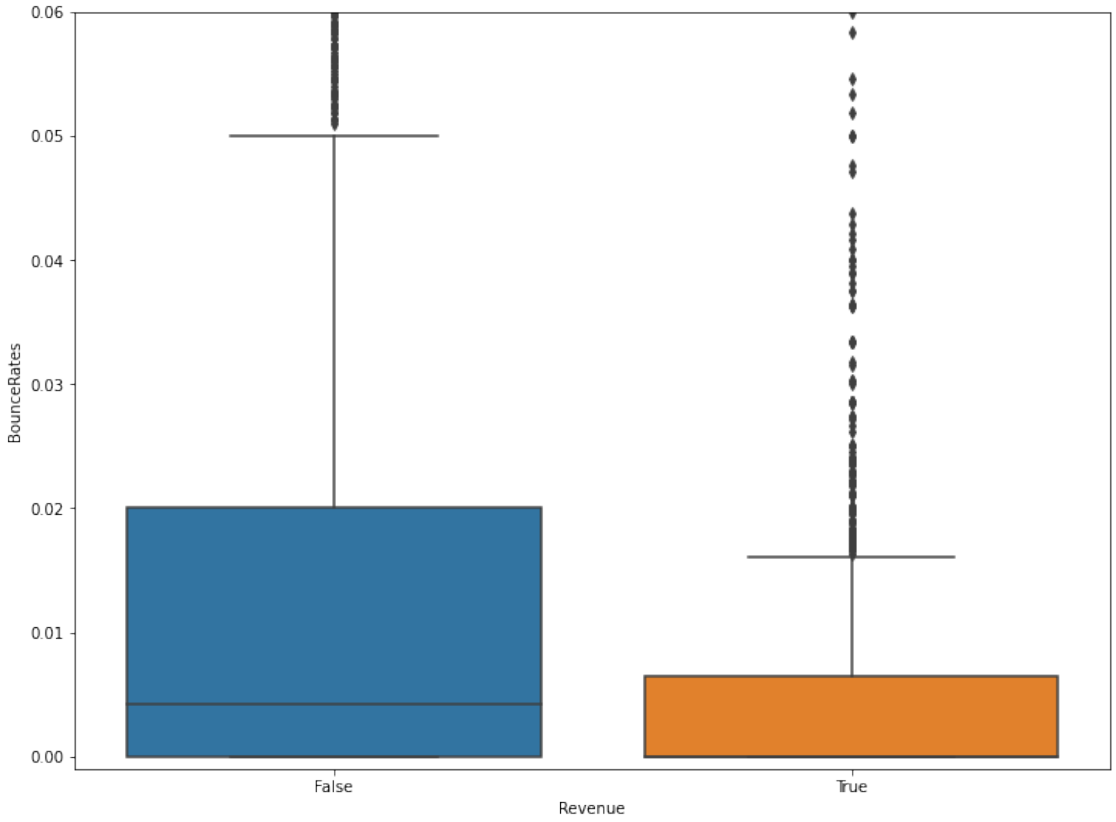
```
ax2.set_title("Users not generating revenue")
```

[12]: Text(0.5, 1.0, 'Users not generating revenue')

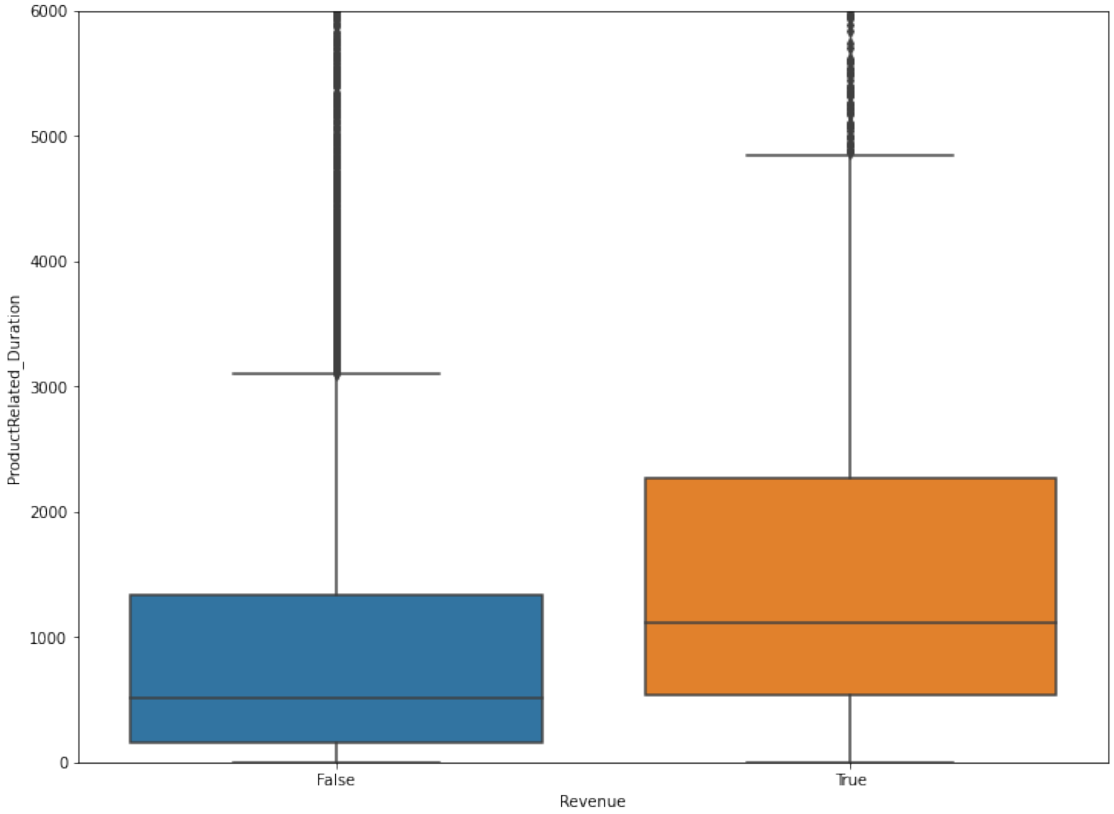


[13]:
```
plt.figure(figsize=(12,9))
sns.boxplot(x=data["Revenue"], y=data["BounceRates"])
plt.ylim([-0.001,0.06])
```

[13]: (-0.001, 0.06)



[14]:
```
plt.figure(figsize=(12,9))
sns.boxplot(x=data["Revenue"], y=data["ProductRelated_Duration"])
plt.ylim([0,6000])
plt.show()
```

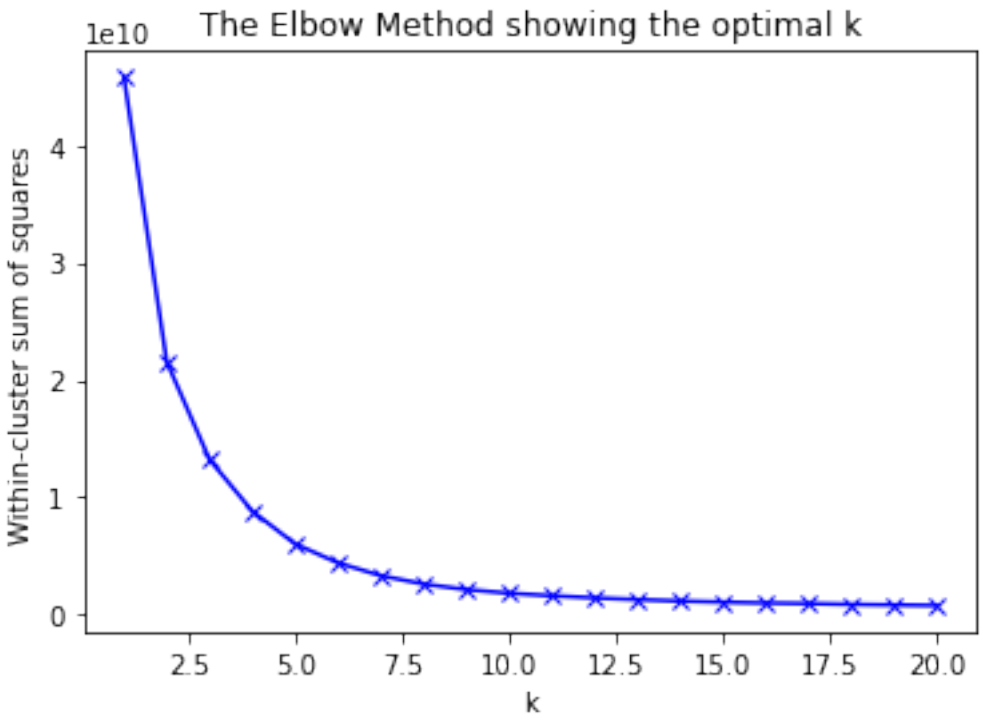


[15]:
```
from sklearn.cluster import KMeans
```

[16]:
```
def count_wcss_scores(X, k_max):
    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
        scores.append(wcss)
    return scores
```

[17]:
```
X = data.drop(['Month', "VisitorType"], axis=1)
```

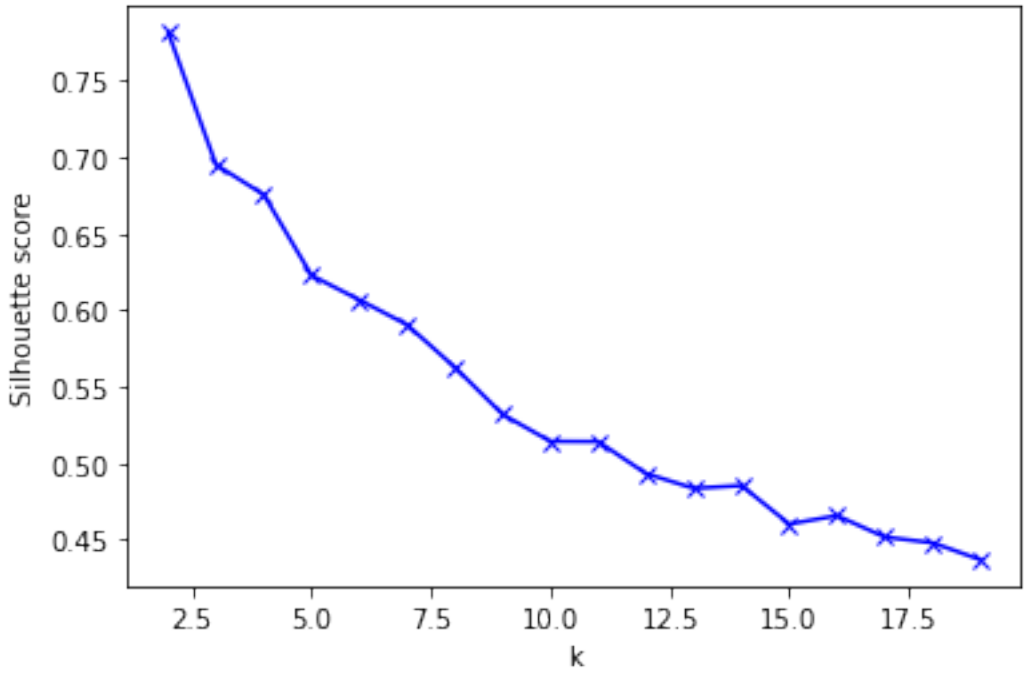
[31]:
```
wcss_vec = count_wcss_scores(X, 20)
x_ticks = list(range(1, len(wcss_vec) + 1))
plt.plot(x_ticks, wcss_vec, 'bx-')
plt.xlabel('k')
plt.ylabel('Within-cluster sum of squares')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



[19]:
```
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
```

[20]:
```
from sklearn.metrics import silhouette_score
cluster_num_seq = range(2, 20)
silhouette_vec = count_clustering_scores(X, cluster_num_seq, KMeans,␣
 ↪silhouette_score)
plt.plot(cluster_num_seq, silhouette_vec, 'bx-')
```

```
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.show()
```



1.0.1 2. KMeans on Original Dataset

[21]:
```
from sklearn.decomposition import PCA
```

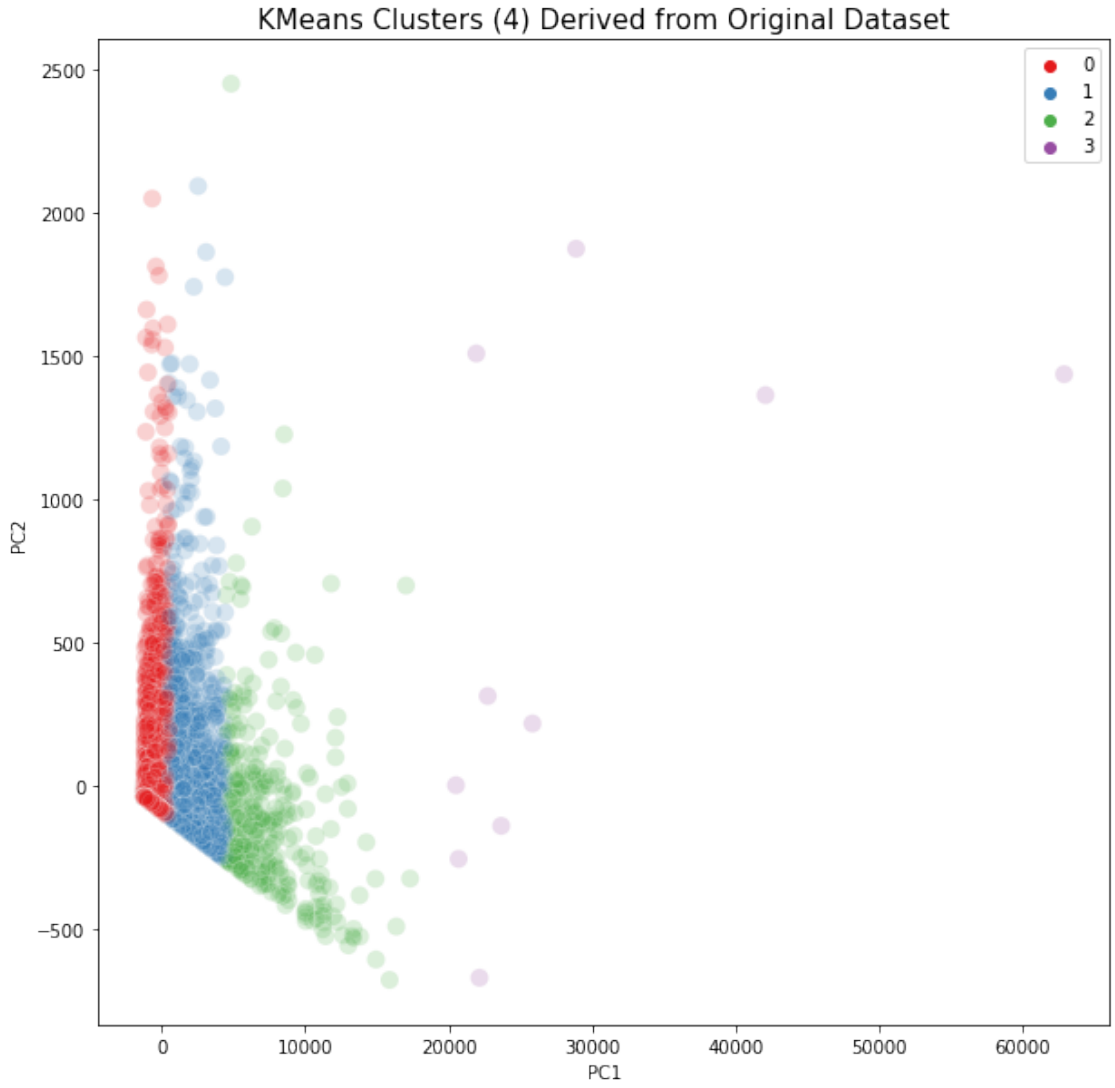
[22]:
```
df_scale2 = X.copy()
kmeans_scale = KMeans(n_clusters=4, n_init=100, max_iter=400, init='k-means++',␣
↪random_state=42).fit(df_scale2)
print('KMeans Scaled Silhouette Score: {}'.format(silhouette_score(df_scale2,␣
↪kmeans_scale.labels_, metric='euclidean')))
labels_scale = kmeans_scale.labels_
clusters_scale = pd.concat([df_scale2, pd.DataFrame({'cluster_scaled':
↪labels_scale})], axis=1)
```

```
KMeans Scaled Silhouette Score: 0.6757718818464974
```

[23]:
```
pca2 = PCA(n_components=3).fit(df_scale2)
pca2d = pca2.transform(df_scale2)
plt.figure(figsize = (10,10))
sns.scatterplot(pca2d[:,0], pca2d[:,1],
                hue=labels_scale,
                palette='Set1',
```

```
                s=100, alpha=0.2).set_title('KMeans Clusters (4) Derived from␣
 ↪Original Dataset', fontsize=15)
plt.legend()
plt.ylabel('PC2')
plt.xlabel('PC1')
plt.show()
```

```
c:\users\mikołaj\appdata\local\programs\python\python38\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(
```



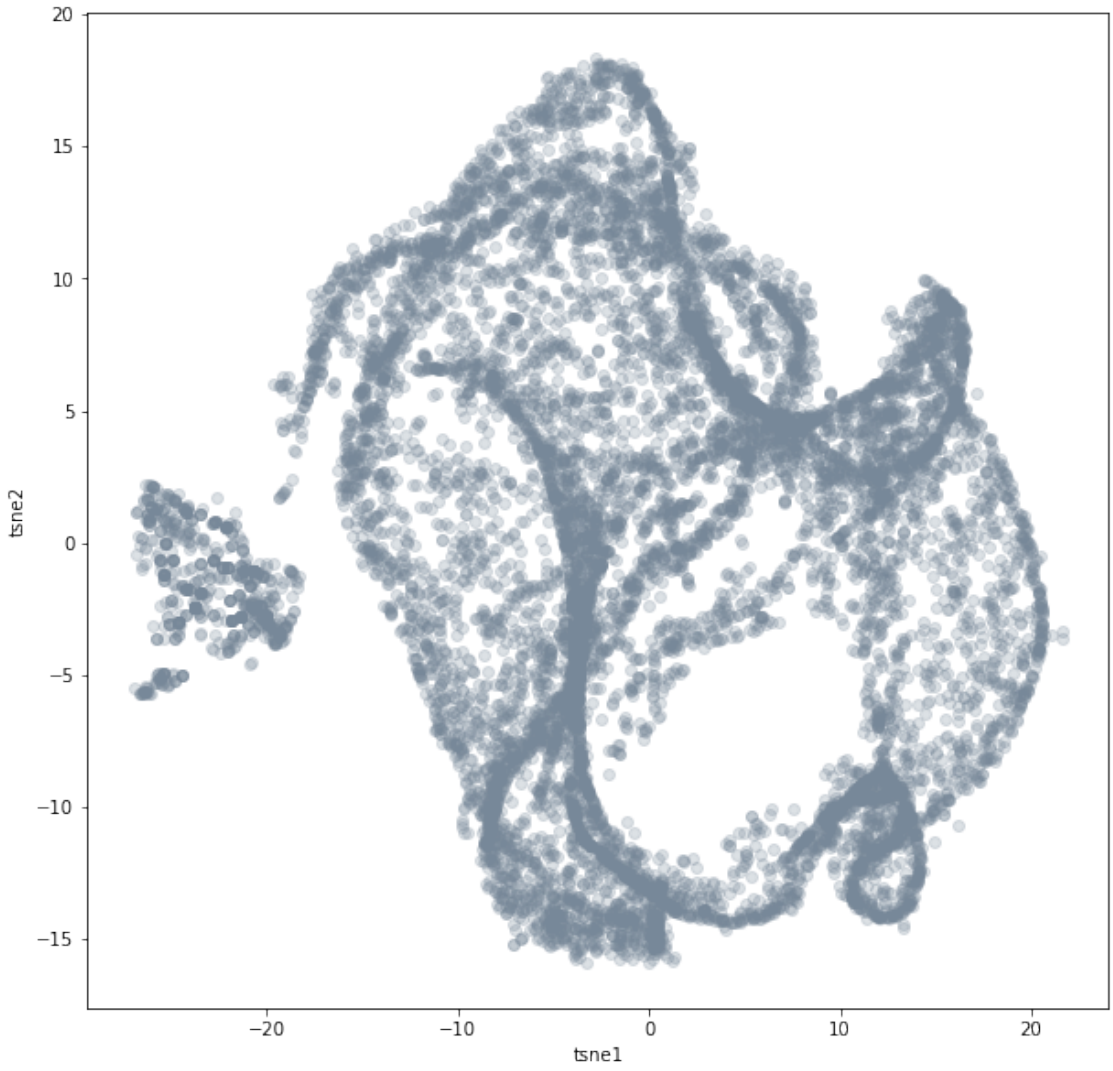
[24]:
```
import plotly.graph_objs as go
```

[25]:
```
Scene = dict(xaxis = dict(title  = 'PC1'),yaxis = dict(title  = 'PC2'),zaxis =␣
↪dict(title  = 'PC3'))
labels = labels_scale
trace = go.Scatter3d(x=pca2d[:,0], y=pca2d[:,1], z=pca2d[:,2],␣
↪mode='markers',marker=dict(color = labels, colorscale='Viridis', size = 10,␣
↪line = dict(color = 'gray',width = 5)))
layout = go.Layout(margin=dict(l=0,r=0),scene = Scene, height = 1000,width =␣
↪1000)
data = [trace]
fig = go.Figure(data = data, layout = layout)
fig.show()
```

[26]:
```
from sklearn.manifold import TSNE
```

[27]:
```
tsne = TSNE(n_components=3, verbose=1, perplexity=80, n_iter=5000,␣
↪learning_rate=200)
tsne_scale_results = tsne.fit_transform(X)
tsne_df_scale = pd.DataFrame(tsne_scale_results, columns=['tsne1', 'tsne2',␣
↪'tsne3'])
plt.figure(figsize = (10,10))
plt.scatter(tsne_df_scale.iloc[:,0],tsne_df_scale.iloc[:,1],alpha=0.25,␣
↪facecolor='lightslategray')
plt.xlabel('tsne1')
plt.ylabel('tsne2')
plt.show()
```

```
[t-SNE] Computing 241 nearest neighbors…
[t-SNE] Indexed 12330 samples in 0.000s…
[t-SNE] Computed neighbors for 12330 samples in 4.961s…
[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: 5.035102
[t-SNE] KL divergence after 250 iterations with early exaggeration: 63.038815
[t-SNE] KL divergence after 5000 iterations: 0.527966
```



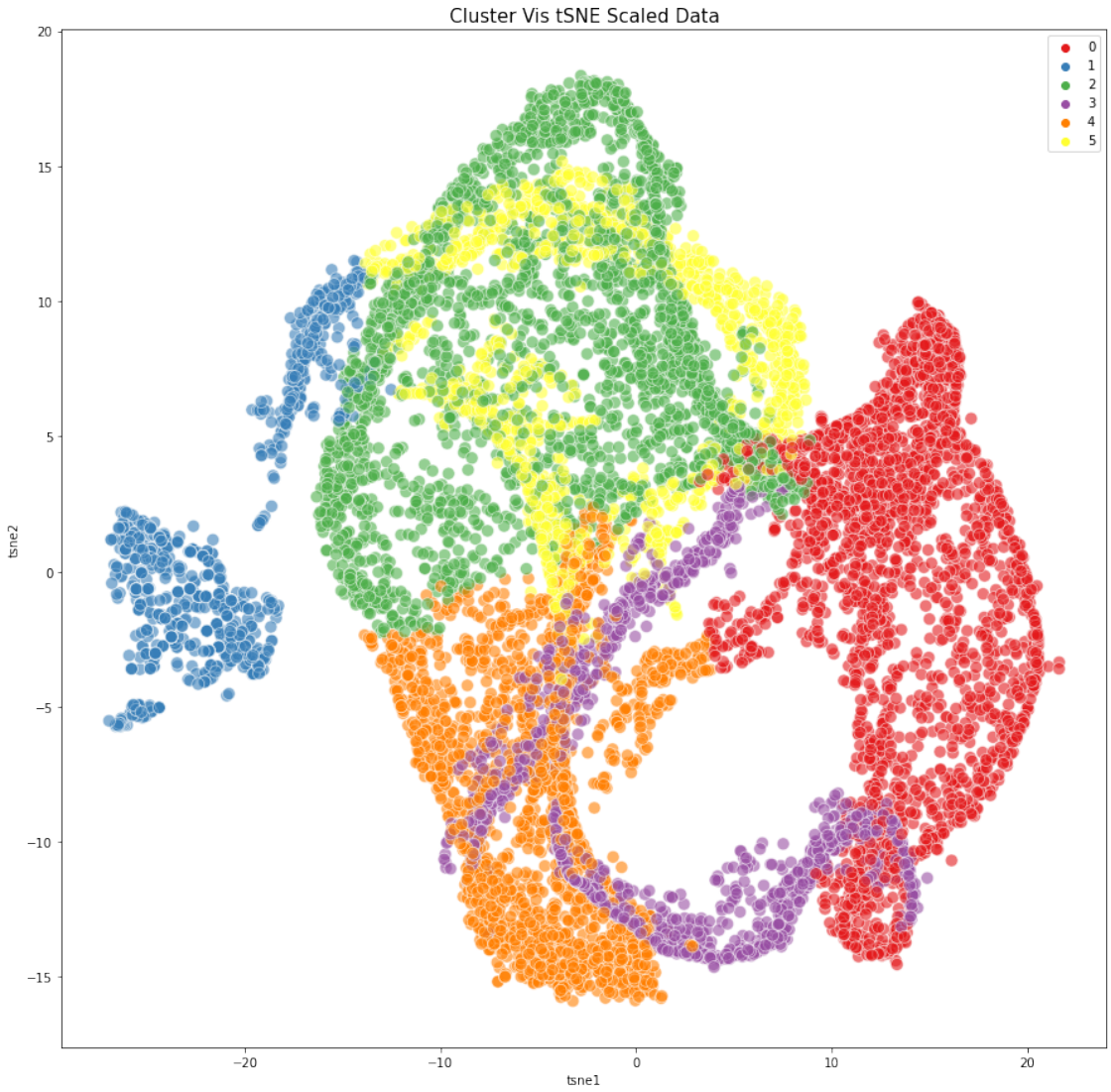
[28]:
```
kmeans_tsne_scale = KMeans(n_clusters=6, n_init=100, max_iter=400,␣
 ↪init='k-means++', random_state=42).fit(tsne_df_scale)
print('KMeans tSNE Scaled Silhouette Score: {}'.
 ↪format(silhouette_score(tsne_df_scale, kmeans_tsne_scale.labels_,␣
 ↪metric='euclidean')))
labels_tsne_scale = kmeans_tsne_scale.labels_
clusters_tsne_scale = pd.concat([tsne_df_scale, pd.DataFrame({'tsne_clusters':
 ↪labels_tsne_scale})], axis=1)
```

```
KMeans tSNE Scaled Silhouette Score: 0.3651507496833801
```

[29]:
```
plt.figure(figsize = (15,15))
```

```
sns.scatterplot(clusters_tsne_scale.iloc[:,0],clusters_tsne_scale.iloc[:
 ↪,1],hue=labels_tsne_scale, palette='Set1', s=100, alpha=0.6).
 ↪set_title('Cluster Vis tSNE Scaled Data', fontsize=15)
plt.legend()
plt.show()
```

```
c:\users\mikołaj\appdata\local\programs\python\python38\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.
```



[30]:
```
Scene = dict(xaxis = dict(title  = 'tsne1'),yaxis = dict(title  =␣
↪'tsne2'),zaxis = dict(title  = 'tsne3'))
labels = labels_tsne_scale
trace = go.Scatter3d(x=clusters_tsne_scale.iloc[:,0], y=clusters_tsne_scale.
↪iloc[:,1], z=clusters_tsne_scale.iloc[:,2], mode='markers',marker=dict(color␣
↪= labels, colorscale='Viridis', size = 10, line = dict(color =␣
↪'yellow',width = 5)))
layout = go.Layout(margin=dict(l=0,r=0),scene = Scene, height = 1000,width =␣
↪1000)
data = [trace]
fig = go.Figure(data = data, layout = layout)
fig.show()
```

[ ]:
```
```