WUM - projekt nr 2 - raport

June 10, 2020

1 Ramka danych

Ramka danych zadana w zadaniu zawiera informacje nt. sesji na stronie sprzedażowej. Zawiera 10 zmiennych numerycznych i 8 kategorycznych. 'Revenue' to zmienna odnosząca się do tego, czy nastąpił zakup. "Administrative", "Administrative Duration", "Informational", "Informational Duration", "Product Related" i "Product Related Duration" reprezentują liczbę różnych rodzajów stron odwiedzanych przez odwiedzającego w tej sesji oraz całkowity czas spędzony w każda z tych kategorii stron. Kolumny "Bounce Rate", "Exit Rate" and "Page Value" reprezentują dane mierzone przez „Google Analytics" dla każdej strony witryny e-commerce. Wartość funkcji "Bounce Rate" dla strony internetowej odnosi się do odsetka odwiedzających, którzy uzyskują dostęp do witryny z tej strony, a następnie wychodzą („odbijają") bez wywoływania innych żądań do serwera analitycznego podczas tej sesji. Wartość kolumny "Exit Rate" dla określonej strony oblicza się tak, jak dla wszystkich odsłon strony, wartość procentową, która była ostatnią sesją. Kolumna "Page Value" reprezentuje średnią wartość strony internetowej odwiedzanej przez użytkownika przed zakończeniem transakcji e-commerce. Funkcja „Dzień specjalny" wskazuje bliskość czasu wizyty w witrynie do określonego dnia specjalnego (np. Dzień Matki, Walentynki), kiedy sesje są bardziej prawdopodobne, że transakcja zostanie sfinalizowana. Wartość tego atrybutu jest ustalana z uwzględnieniem dynamiki handlu elektronicznego, takiego jak czas trwania między datą zamówienia a datą dostawy. Na przykład dla Walentynek ta wartość przyjmuje wartość niezerową między 2 lutego a 12 lutego, zero przed tą datą i po tej dacie, chyba że zbliża się ona do innego specjalnego dnia i jej maksymalna wartość wynosi 1 8 lutego. Zestaw danych obejmuje również system użytkownika, przeglądarkę, region, typ ruchu, typ odwiedzającego jako powracającego lub nowego odwiedzającego, wartość logiczną wskazującą, czy data wizyty to weekend i miesiąc roku.

Chcemy znaleźć najbardziej naturalne podziały tego zbioru, które możnaby jakoś zinterpretować. Przy klasteryzacji usuwamy zmienną 'Revenue', ponieważ taki podział może być jednym z otrzymanych.

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[3]:
```
dataframe = pd.read_csv('online_shoppers_intention.csv')
```

[4]:
```
dataframe
```

[4]:

	Administrative	Administrative_Duration	Informational	\
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	
…	…	…	…	
12325	3	145.0	0	
12326	0	0.0	0	
12327	0	0.0	0	
12328	4	75.0	0	
12329	0	0.0	0	

	Informational_Duration	ProductRelated	ProductRelated_Duration	\
0	0.0	1	0.000000	
1	0.0	2	64.000000	
2	0.0	1	0.000000	
3	0.0	2	2.666667	
4	0.0	10	627.500000	
…	…	…	…	
12325	0.0	53	1783.791667	
12326	0.0	5	465.750000	
12327	0.0	6	184.250000	
12328	0.0	15	346.000000	
12329	0.0	3	21.250000	

	BounceRates	ExitRates	PageValues	SpecialDay	Month	OperatingSystems	\
0	0.200000	0.200000	0.000000	0.0	Feb	1	
1	0.000000	0.100000	0.000000	0.0	Feb	2	
2	0.200000	0.200000	0.000000	0.0	Feb	4	
3	0.050000	0.140000	0.000000	0.0	Feb	3	
4	0.020000	0.050000	0.000000	0.0	Feb	3	
…	…	…	…	…	…	…	
12325	0.007143	0.029031	12.241717	0.0	Dec	4	
12326	0.000000	0.021333	0.000000	0.0	Nov	3	
12327	0.083333	0.086667	0.000000	0.0	Nov	3	
12328	0.000000	0.021053	0.000000	0.0	Nov	2	
12329	0.000000	0.066667	0.000000	0.0	Nov	3	

	Browser	Region	TrafficType	VisitorType	Weekend	Revenue
0	1	1	1	Returning_Visitor	False	False
1	2	1	2	Returning_Visitor	False	False
2	1	9	3	Returning_Visitor	False	False
3	2	2	4	Returning_Visitor	False	False
4	3	1	4	Returning_Visitor	True	False
…	…	…	…	…	…	…
12325	6	1	1	Returning_Visitor	True	False

```
12326        2       1            8  Returning_Visitor     True    False
12327        2       1           13  Returning_Visitor     True    False
12328        2       3           11  Returning_Visitor    False    False
12329        2       1            2        New_Visitor     True    False

[12330 rows x 18 columns]
```

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

[5]:
```
       Administrative  Administrative_Duration  Informational  \
count    12330.000000             12330.000000   12330.000000
mean         2.315166                80.818611       0.503569
std          3.321784               176.779107       1.270156
min          0.000000                 0.000000       0.000000
25%          0.000000                 0.000000       0.000000
50%          1.000000                 7.500000       0.000000
75%          4.000000                93.256250       0.000000
max         27.000000              3398.750000      24.000000

       Informational_Duration  ProductRelated  ProductRelated_Duration  \
count            12330.000000    12330.000000             12330.000000
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              1464.157213
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
```

Procent zakupów

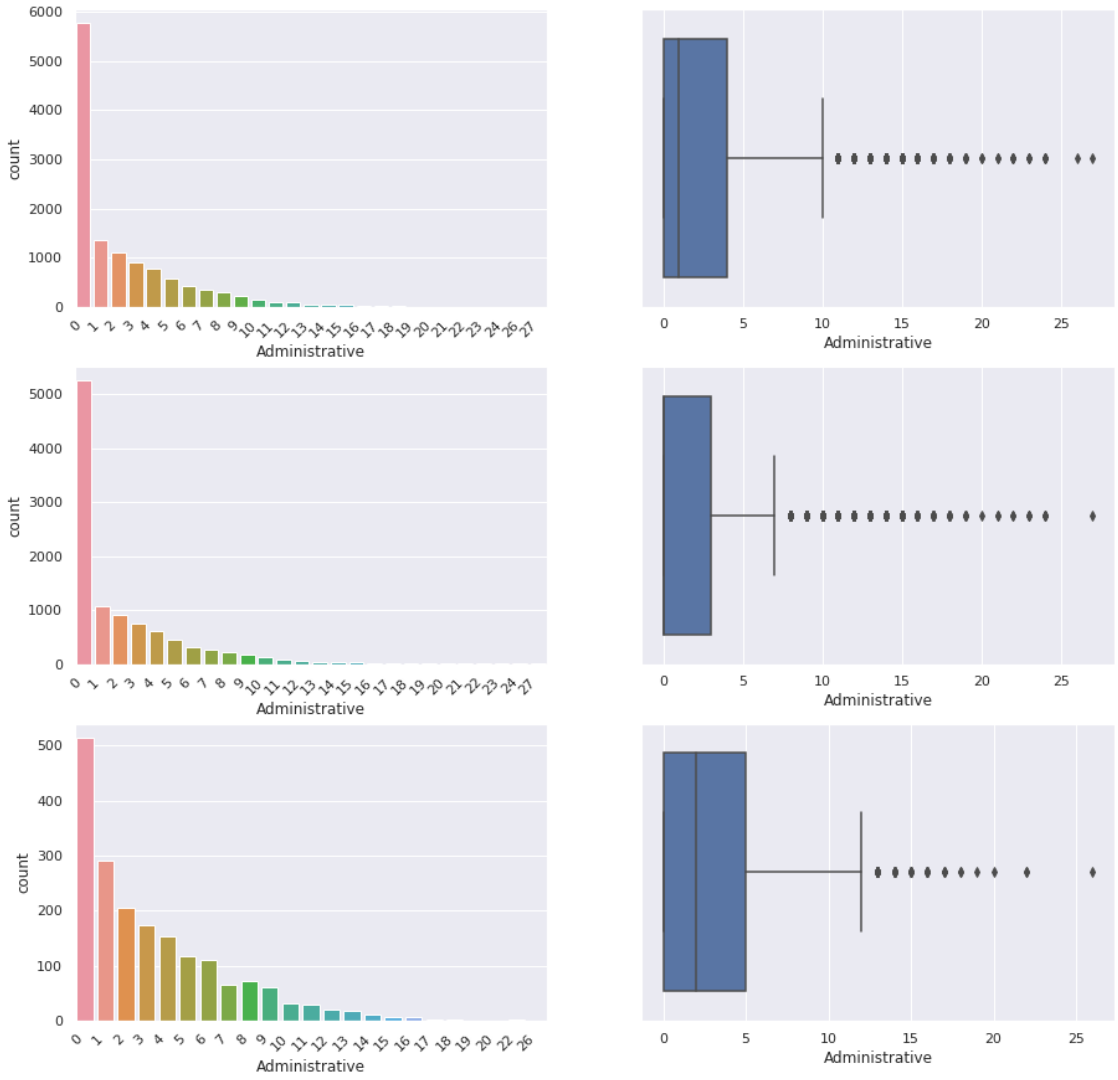
[7]:
```
dataframe.Revenue.mean()
```

[7]:
```
0.15474452554744525
```

2 Eksploracja danych

Administrative

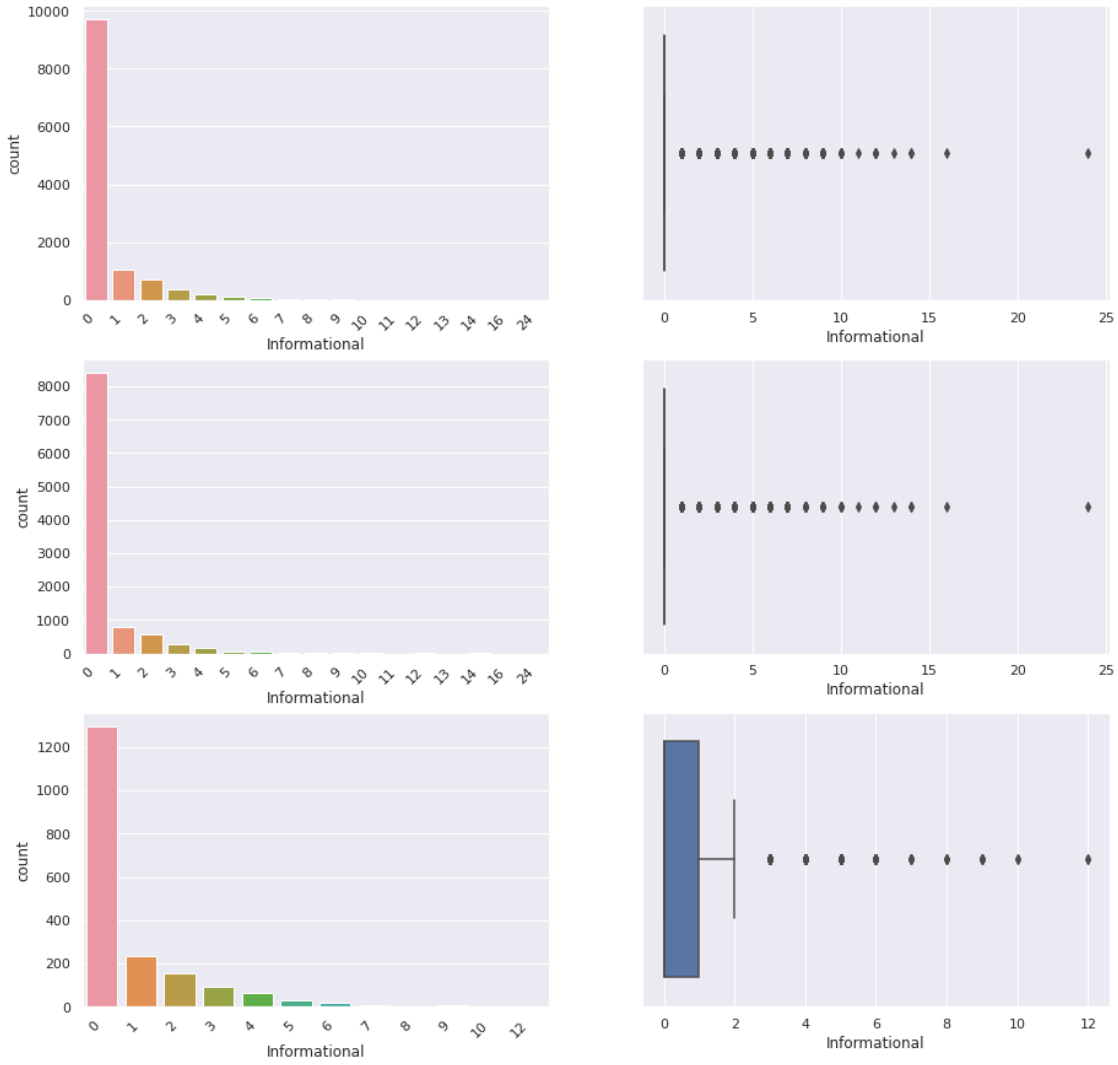
[8]:
```
sns.set(style="darkgrid")

plt.figure(figsize=(15,15))

plt.subplot(3, 2, 1)
ax = sns.countplot(x='Administrative', data=dataframe)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')

plt.subplot(3, 2, 2)
ax = sns.boxplot(x=dataframe['Administrative'])

plt.subplot(3, 2, 3)
ax = sns.countplot(x='Administrative', data=dataframe[dataframe['Revenue'] ==␣
 ↪False])
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')

plt.subplot(3, 2, 4)
ax = sns.boxplot(x=dataframe[dataframe['Revenue'] == False]['Administrative'])

plt.subplot(3, 2, 5)
ax = sns.countplot(x='Administrative', data=dataframe[dataframe['Revenue'] ==␣
 ↪True])
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')

plt.subplot(3, 2, 6)
ax = sns.boxplot(x=dataframe[dataframe['Revenue'] == True]['Administrative'])

plt.show()
```



Informational

[9]:
```
sns.set(style="darkgrid")

plt.figure(figsize=(15,15))

plt.subplot(3, 2, 1)
ax = sns.countplot(x='Informational', data=dataframe)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
↪horizontalalignment='right')

plt.subplot(3, 2, 2)
ax = sns.boxplot(x=dataframe['Informational'])
```

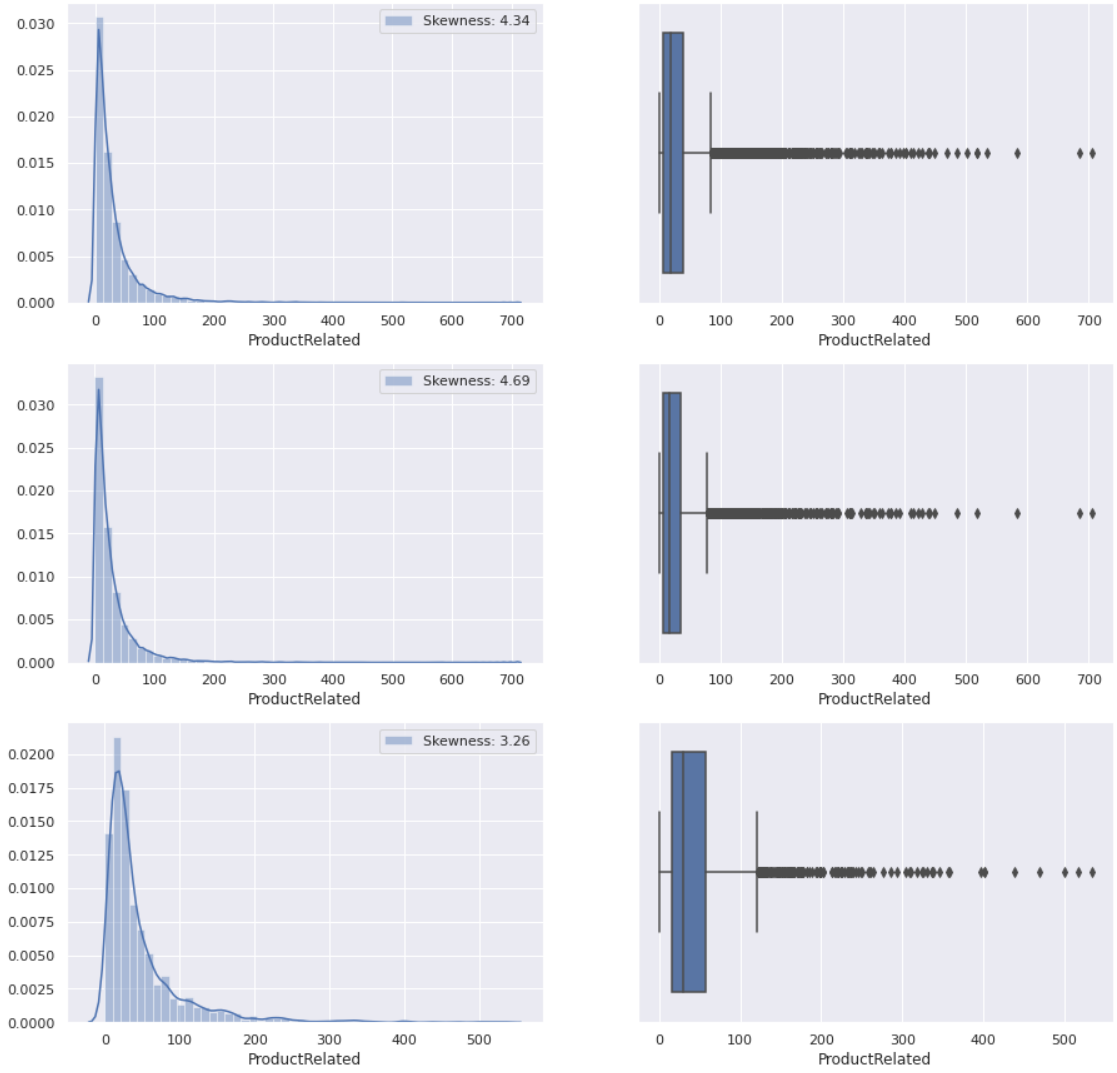
```
plt.subplot(3, 2, 3)
ax = sns.countplot(x='Informational', data=dataframe[dataframe['Revenue'] ==␣
 ↪False])
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')

plt.subplot(3, 2, 4)
ax = sns.boxplot(x=dataframe[dataframe['Revenue'] == False]['Informational'])

plt.subplot(3, 2, 5)
ax = sns.countplot(x='Informational', data=dataframe[dataframe['Revenue'] ==␣
 ↪True])
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')

plt.subplot(3, 2, 6)
ax = sns.boxplot(x=dataframe[dataframe['Revenue'] == True]['Informational'])

plt.show()
```



Product Related

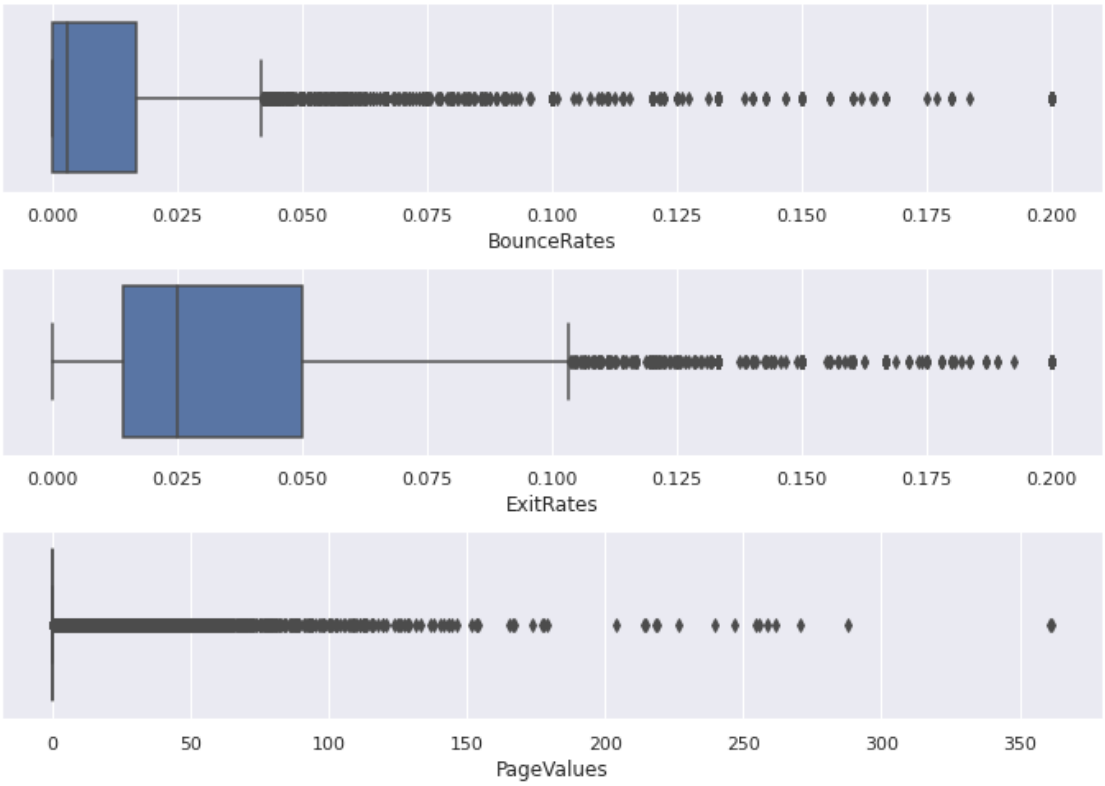
[10]:
```
sns.set(style="darkgrid")

plt.figure(figsize=(15,15))

plt.subplot(3, 2, 1)
sns.distplot(dataframe.ProductRelated, label='Skewness: %.2f' % (dataframe.
 ↪ProductRelated.skew()))
plt.legend()

plt.subplot(3, 2, 2)
ax = sns.boxplot(x=dataframe['ProductRelated'])

plt.subplot(3, 2, 3)
```

```
sns.distplot(dataframe[dataframe['Revenue'] == False]['ProductRelated'],␣
 ↪label='Skewness: %.2f' % (dataframe[dataframe['Revenue'] ==␣
 ↪False]['ProductRelated'].skew()))
plt.legend()

plt.subplot(3, 2, 4)
ax = sns.boxplot(x=dataframe[dataframe['Revenue'] == False]['ProductRelated'])

plt.subplot(3, 2, 5)
sns.distplot(dataframe[dataframe['Revenue'] == True]['ProductRelated'],␣
 ↪label='Skewness: %.2f' % (dataframe[dataframe['Revenue'] ==␣
 ↪True]['ProductRelated'].skew()))
plt.legend()

plt.subplot(3, 2, 6)
ax = sns.boxplot(x=dataframe[dataframe['Revenue'] == True]['ProductRelated'])

plt.show()
```



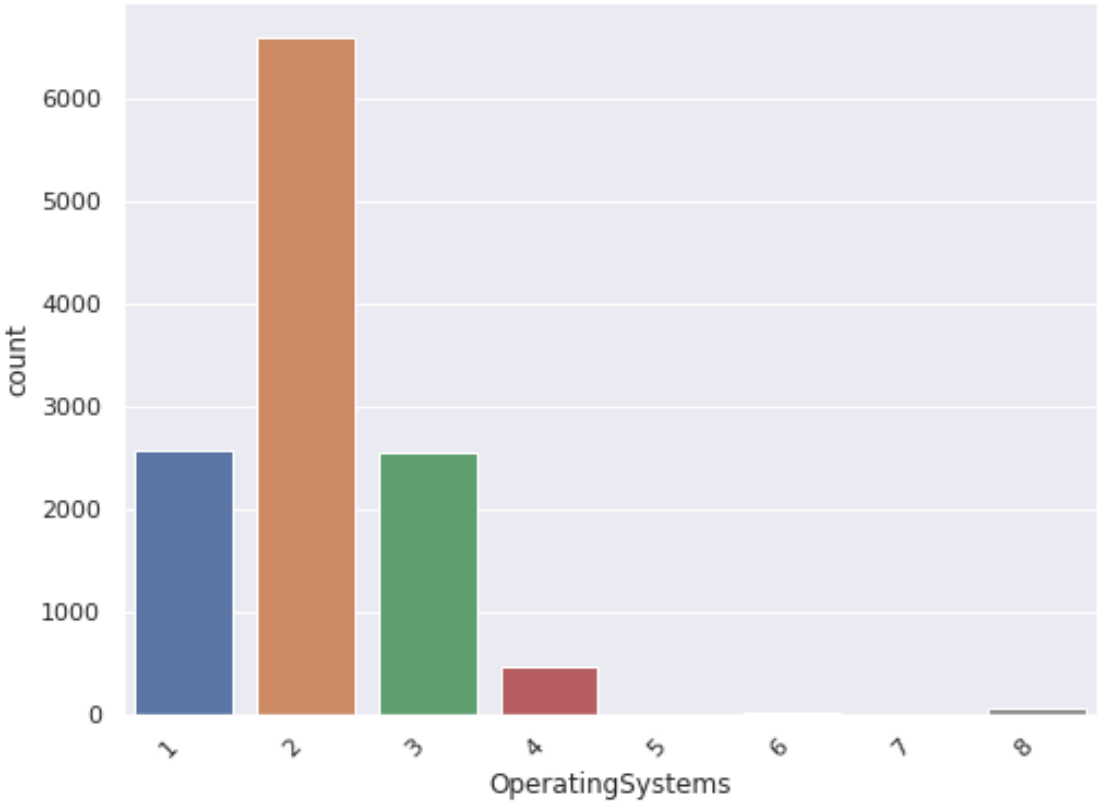
Bounce, Exit, Page Rate

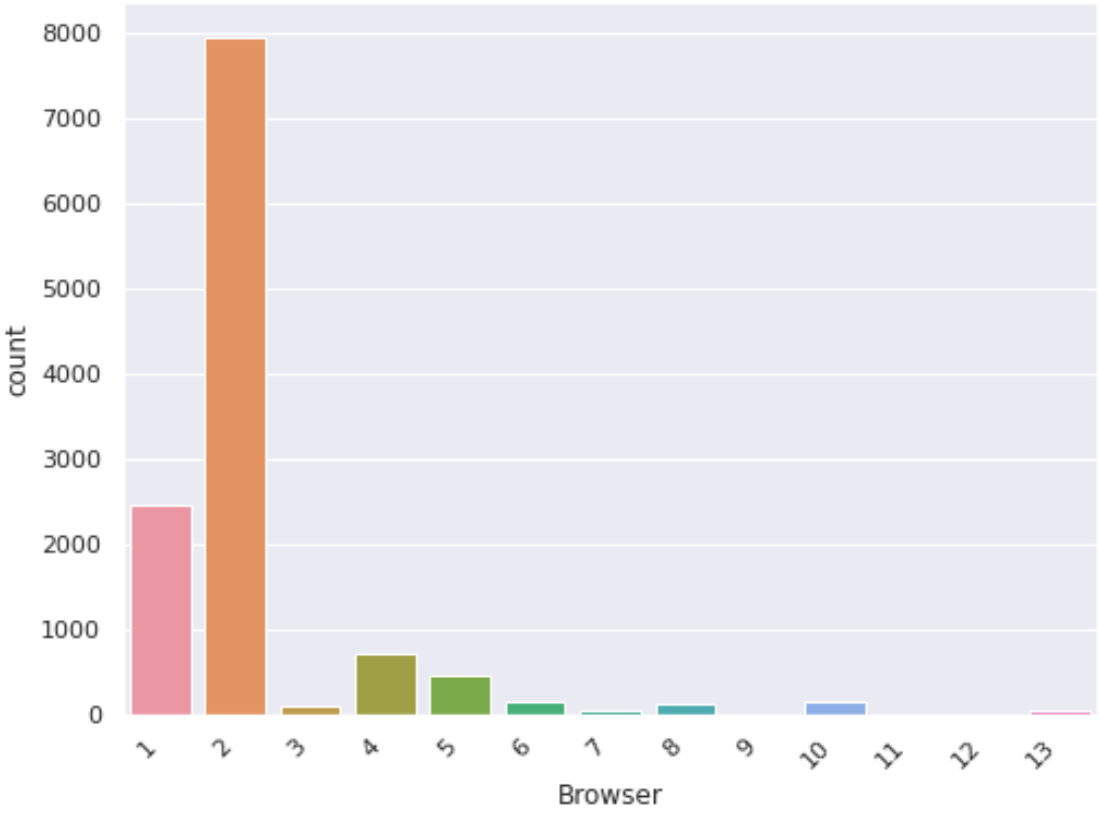
[11]:
```
rates = ['BounceRates', 'ExitRates', 'PageValues']

sns.set(style="darkgrid")

fig = plt.figure(figsize=(12,8))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i in range(1, 4):
    ax = fig.add_subplot(3, 1, i),
    ax = sns.boxplot(x=dataframe[rates[i-1]])
```



Systemy operacyjne i przeglądarki

[12]:
```
sns.set(style="darkgrid")

fig = plt.figure(figsize=(8,6))
ax = sns.countplot(x='OperatingSystems', data=dataframe)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
plt.show()

sns.set(style="darkgrid")

fig = plt.figure(figsize=(8,6))
ax = sns.countplot(x='Browser', data=dataframe)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
plt.show()
```


6000
5000
4000
3000
2000
1000
0
count
1
2
3
4
5
6
7
8
OperatingSystems



3 Feature engineering

[13]:
```
cleanup = {"Month": {"Jan": 1,
                     "Feb": 2,
                     "Mar": 3,
                     "Apr": 4,
                     "May": 5,
                     "June": 6,
                     "Jul": 7,
                     "Aug": 8,
                     "Sep": 9,
                     "Nov": 10,
                     "Oct": 11,
                     "Dec": 12},
           "VisitorType":{
               "Returning_Visitor": 0,
               "New_Visitor": 1,
               "Other": 2
           }}
```

[14]:
```
dataframe.replace(cleanup, inplace=True)
dataframe.head()
```

[14]:

	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	2	1	1	1	1	0	False	False
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	0.0	0.0	2	2	2	1	2	0	False	False
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0	0.0	2	4	1	9	3	0	False	False
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	0.0	0.0	2	3	2	2	4	0	False	False
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	0.0	0.0	2	3	3	1	4	0	True	False

[18]:
```
import math

date = list(dataframe.Month)

date_sin = [math.sin(date[i]/12) for i in range (len(date))]
date_cos = [math.cos(date[i]/12) for i in range (len(date))]
dataframe['date_sin'] = date_sin
dataframe['date_cos'] = date_cos
dataframe = dataframe.drop('Month', axis=1)
```

Zaminiliśmy outliery, które mogły być spowodowane np. zostawieniem włączonego komputera.

[19]:
```
upper_lim_Adm_Dur = dataframe['Administrative_Duration'].mean() +␣
↪dataframe['Administrative_Duration'].std() * 3

dataframe.loc[(dataframe['Administrative_Duration'] > upper_lim_Adm_Dur),
```

```
            'Administrative_Duration'] = upper_lim_Adm_Dur

upper_lim_Inf_Dur = dataframe['Informational_Duration'].mean() +␣
↪dataframe['Informational_Duration'].std() * 3

dataframe.loc[(dataframe['Informational_Duration'] > upper_lim_Inf_Dur),
            'Informational_Duration'] = upper_lim_Inf_Dur

upper_lim_Prd_Dur = dataframe['ProductRelated_Duration'].mean() +␣
↪dataframe['ProductRelated_Duration'].std() * 3

dataframe.loc[(dataframe['ProductRelated_Duration'] > upper_lim_Prd_Dur),
            'ProductRelated_Duration'] = upper_lim_Prd_Dur
```

Użyliśmy StandardScaler aby zestandaryzować dane.

[20]:
```
from sklearn.preprocessing import StandardScaler

to_scale = ['Administrative_Duration', 'Informational_Duration',␣
↪'ProductRelated_Duration',
            'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay']

scaler = StandardScaler()

dataframe[to_scale] = scaler.fit_transform(dataframe[to_scale])
```

Parę prób wstępnego modelowania.

[21]:
```
y = dataframe.Revenue
X = dataframe.drop('Revenue', axis=1)
```

[22]:
```
X
```

[22]:
```
       Administrative  Administrative_Duration  Informational  \
0                   0                -0.614896              0
1                   0                -0.614896              0
2                   0                -0.614896              0
3                   0                -0.614896              0
4                   0                -0.614896              0
…                 …                        …              …
12325               3                 0.673239              0
12326               0                -0.614896              0
12327               0                -0.614896              0
12328               4                 0.051381              0
12329               0                -0.614896              0

       Informational_Duration  ProductRelated  ProductRelated_Duration  \
```

```
0                  -0.352755              1                -0.829946
1                  -0.352755              2                -0.781368
2                  -0.352755              1                -0.829946
3                  -0.352755              2                -0.827922
4                  -0.352755             10                -0.353655
…                        …              …                        …
12325              -0.352755             53                 0.524004
12326              -0.352755              5                -0.476428
12327              -0.352755              6                -0.690095
12328              -0.352755             15                -0.567322
12329              -0.352755              3                -0.813817

       BounceRates  ExitRates  PageValues  SpecialDay  OperatingSystems  \
0         3.667189   3.229316   -0.317178   -0.308821                 1
1        -0.457683   1.171473   -0.317178   -0.308821                 2
2         3.667189   3.229316   -0.317178   -0.308821                 4
3         0.573535   1.994610   -0.317178   -0.308821                 3
4        -0.045196   0.142551   -0.317178   -0.308821                 3
…              …          …          …          …                 …
12325    -0.310366  -0.288966    0.342125   -0.308821                 4
12326    -0.457683  -0.447364   -0.317178   -0.308821                 3
12327     1.261014   0.897093   -0.317178   -0.308821                 3
12328    -0.457683  -0.453140   -0.317178   -0.308821                 2
12329    -0.457683   0.485525   -0.317178   -0.308821                 3

       Browser  Region  TrafficType  VisitorType  Weekend  date_sin  date_cos
0            1       1            1            0    False  0.165896  0.986143
1            2       1            2            0    False  0.165896  0.986143
2            1       9            3            0    False  0.165896  0.986143
3            2       2            4            0    False  0.165896  0.986143
4            3       1            4            0     True  0.165896  0.986143
…          …       …            …          …       …        …         …
12325        6       1            1            0     True  0.841471  0.540302
12326        2       1            8            0     True  0.740177  0.672412
12327        2       1           13            0     True  0.740177  0.672412
12328        2       3           11            0    False  0.740177  0.672412
12329        2       1            2            1     True  0.740177  0.672412

[12330 rows x 18 columns]
```

[23]:
```
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
pred = kmeans.labels_

accuracy_score(y, pred)
```

[23]: 0.8125709651257097

[24]:
```
from sklearn.cluster import AgglomerativeClustering

model = AgglomerativeClustering(n_clusters=2, linkage='single').fit(X)

pred = model.labels_

accuracy_score(y, pred)
```

[24]: 0.8450932684509327

[25]:
```
from sklearn.cluster import AgglomerativeClustering

model = AgglomerativeClustering(n_clusters=2, linkage='ward').fit(X)

pred = model.labels_

accuracy_score(y, pred)
```

[25]: 0.22668288726682886

[26]:
```
from sklearn.cluster import AgglomerativeClustering

model = AgglomerativeClustering(n_clusters=2, linkage='complete').fit(X)

pred = model.labels_

accuracy_score(y, pred)
```

[26]: 0.8451743714517437

[27]:
```
from sklearn.cluster import AgglomerativeClustering

model = AgglomerativeClustering(n_clusters=2, linkage='average').fit(X)

pred = model.labels_

accuracy_score(y, pred)
```

[27]: 0.8450932684509327

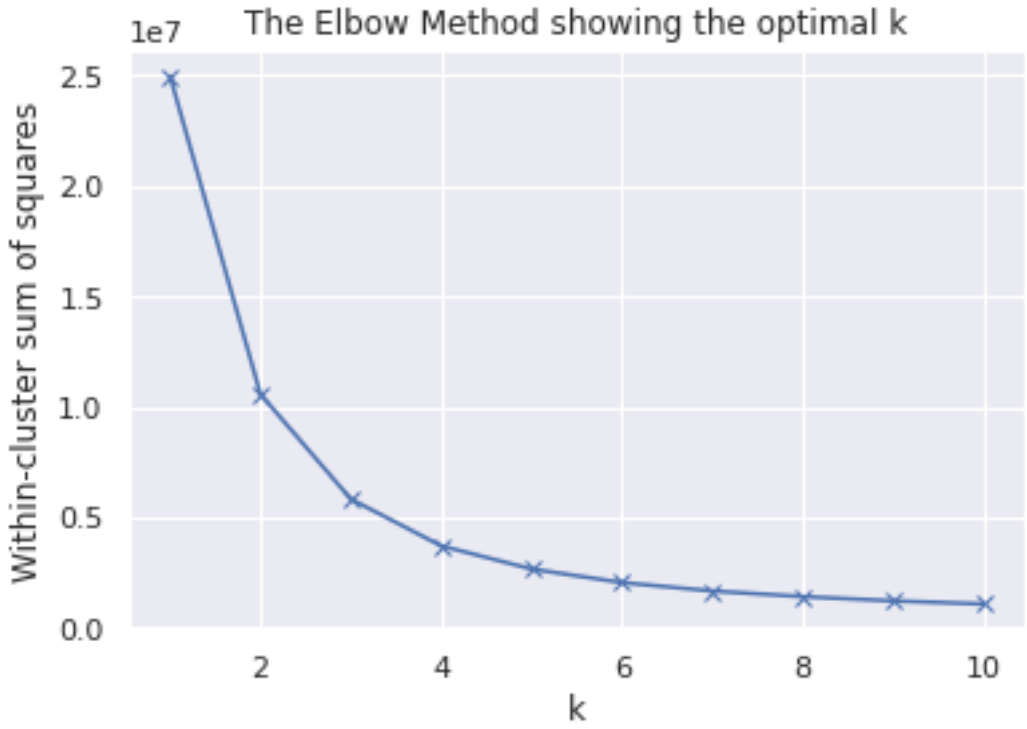
Widzimy, że zakładając dwa klastry (kupujący/niekupujący) osiągamy całkiem dobre accuracy względem danego Revenue.

4 Checkpoint 3

4.1 Szukanie optymalnej ilości klastrów

4.1.1 Elbow method

[28]:
```
scores = []
k_max = 10
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)

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()
```



Z metody łokcia moglibyśmy przyjąć k = 3 lub k = 4

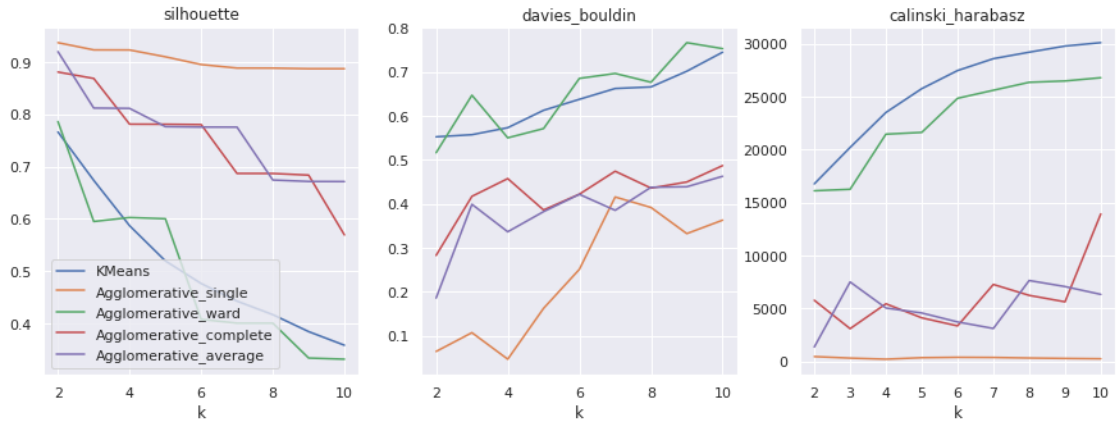
4.1.2 Metryki

[29]:
```
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
↪calinski_harabasz_score
```

[30]:
```
k_max = 10
col = ["k","KMeans","Agglomerative_single","Agglomerative_ward",
       "Agglomerative_complete","Agglomerative_average"]
# silhouette_score
metric_silhouette = pd.DataFrame(columns=col)
for i in range(2,k_max+1):
    metric_silhouette = metric_silhouette.append(
        {"k" : i,
         "KMeans" : silhouette_score(dataframe, KMeans(n_clusters=i).
↪fit_predict(dataframe)),
         "Agglomerative_single" : silhouette_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='single').
↪fit_predict(dataframe)),
         "Agglomerative_ward" : silhouette_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='ward').fit_predict(dataframe)),
         "Agglomerative_complete" : silhouette_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='complete').
↪fit_predict(dataframe)),
         "Agglomerative_average" : silhouette_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='average').
↪fit_predict(dataframe)),},
        ignore_index=True)

# davies_bouldin_score
metric_davies_bouldin = pd.DataFrame(columns=col)
for i in range(2,k_max+1):
    metric_davies_bouldin = metric_davies_bouldin.append(
        {"k" : i,
         "KMeans" : davies_bouldin_score(dataframe, KMeans(n_clusters=i).
↪fit_predict(dataframe)),
         "Agglomerative_single" : davies_bouldin_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='single').
↪fit_predict(dataframe)),
         "Agglomerative_ward" : davies_bouldin_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='ward').fit_predict(dataframe)),
         "Agglomerative_complete" : davies_bouldin_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='complete').
↪fit_predict(dataframe)),
         "Agglomerative_average" : davies_bouldin_score(dataframe,␣
↪AgglomerativeClustering(n_clusters=i,linkage='average').
↪fit_predict(dataframe))},
        ignore_index=True)
```

```
# calinski_harabasz_score
metric_calinski_harabasz = pd.DataFrame(columns=col)
for i in range(2,k_max+1):
    metric_calinski_harabasz = metric_calinski_harabasz.append(
        {"k" : i,
         "KMeans" : calinski_harabasz_score(dataframe, KMeans(n_clusters=i).
 ↪fit_predict(dataframe)),
         "Agglomerative_single" : calinski_harabasz_score(dataframe,␣
 ↪AgglomerativeClustering(n_clusters=i,linkage='single').
 ↪fit_predict(dataframe)),
         "Agglomerative_ward" : calinski_harabasz_score(dataframe,␣
 ↪AgglomerativeClustering(n_clusters=i,linkage='ward').fit_predict(dataframe)),
         "Agglomerative_complete" : calinski_harabasz_score(dataframe,␣
 ↪AgglomerativeClustering(n_clusters=i,linkage='complete').
 ↪fit_predict(dataframe)),
         "Agglomerative_average" : calinski_harabasz_score(dataframe,␣
 ↪AgglomerativeClustering(n_clusters=i,linkage='average').
 ↪fit_predict(dataframe))},
        ignore_index=True)
```

[34]:
```
alg = ["KMeans","Agglomerative_single","Agglomerative_ward",
       "Agglomerative_complete","Agglomerative_average"]
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
plt.xticks(np.arange(1, k_max+1, 1.0))
metric_silhouette.plot(x="k",y=alg,ax=ax[0], title = "silhouette")
metric_davies_bouldin.plot(x="k",y=alg,ax=ax[1],legend=None, title =␣
 ↪"davies_bouldin")
metric_calinski_harabasz.plot(x="k",y=alg,ax=ax[2],legend=None, title =␣
 ↪"calinski_harabasz")
plt.show(fig)
```



[35]:
```
from sklearn import preprocessing
s = metric_silhouette + preprocessing.normalize(metric_calinski_harabasz) +␣
 ↪(1-metric_davies_bouldin)
s['k'] = range(2,k_max+1)
s
```

[35]:

	k	KMeans	Agglomerative_single	Agglomerative_ward	Agglomerative_complete	Agglomerative_average
0	2	1.912789	1.891746	1.941578	1.838765	1.790877
1	3	1.860476	1.827673	1.546723	1.565484	1.689359
2	4	1.733613	1.882808	1.709538	1.490530	1.629409
3	5	1.660789	1.758067	1.662114	1.515554	1.528185
4	6	1.574099	1.654696	1.387984	1.448359	1.453973
5	7	1.510583	1.482625	1.358106	1.398537	1.469827
6	8	1.471595	1.504723	1.374624	1.405135	1.425363
7	9	1.412210	1.561965	1.216033	1.372040	1.405992
8	10	1.312532	1.530294	1.200880	1.405927	1.355893

[36]:
```
print("Clusters number : ",s.drop('k',axis=1).max(axis=1).idxmax()+2)
print("Best algorithm  : ",s.drop('k',axis=1).max(axis=0).idxmax())
```

```
Clusters number :  2
Best algorithm  :  Agglomerative_ward
```

[37]:
```
model = AgglomerativeClustering(n_clusters=2, linkage='ward').
 ↪fit_predict(dataframe)

accuracy_score(y, model)
```

[37]:
```
0.8253041362530413
```

Widzimy, że najlepszy algorytm podzielił zbiór na dwa klastry, tak jak podejrzewaliśmy.

Zauważmy, że jest to bardzo podobny podział do podziału oryginalnego na osoby, które kupiły i nie.