BA476 Practical Assignment 1 - K-Means Clustering for Market Segmentation

The purpose of this assignment is to use K-Means Clustering to better understand customer behavior. In this assignment, we are taking a look at transaction data for a large e-Commerce company and trying to convert that transaction information into customer-level data for actionable insights. The goal of this assignment is to perform market segmentation based on purchase behaviour.

We will use Pandas to manipulate dataframes, Scikit-learn to create the clusters, and Matplotlib for visualization. The deliverable for this assignment is (1) this notebook, (2) a PDF file that you will produce by converting your notebook to html and then printing the html page to pdf. As a reminder your notebook should contain all the code you used to generate your results – try writing concise code and include comments describing what you're doing. Submit your files on gradescope when you're done.

A skeleton is provided to get you started. Good luck!

Acknowledgements: This example makes use of the UCI MLR dataset on online retail (http://archive.ics.uci.edu/ml/datasets/online+retail). Most of the code in this example is based on the Github repo (https://github.com/PacktPublishing/Hands-On-Data-Science-for-Marketing) for "Hands-On Data Science for Marketing" by Packt, and the treatment of that by Mike Nemke (https://www.mktr.ai/applications-and-methods-in-data-science-customer-segmentation/).

Importing and cleaning data

Get started by importing the necessary packages and importing the dataset.

In [170]:
```
# DO NOT EDIT THIS BLOCK OF CODE
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import cluster
from sklearn.cluster import KMeans

#Create the initial dataframe from the UCI repository
df = pd.read_excel("http://archive.ics.uci.edu/ml/machine-learning-datab
ases/00352/Online%20Retail.xlsx")
```

Inspect the dataframe to understand the columns and rows. We have around half a million rows, end each row corresponds to an item purchased by some customer. Notably, a row is *not* a transaction, you'll notice several rows share the same invoice number.

In [171]:
```
print(df.shape)
df.head(10)
```

```
(541909, 8)
```

Out[171]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850.0	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom
8	536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	2010-12-01 08:34:00	1.69	13047.0	United Kingdom

Check the dataframe for null entries in each column.

In [172]:
```
#Check for null values in the dataframe by column
df.isnull().sum()
```

Out[172]:
```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID    135080
Country            0
dtype: int64
```

Cleaning the Data (3 Points)

Remove cancelled orders (quantity<=0), orders with no description (description = ""), and records without a customerID.

In [173]:
```
# (2 points)
# Drop cancelled orders
print(df.loc[df['Quantity'] <= 0].shape)

df.shape
print("geree")
df = df.loc[df['Quantity'] > 0]
df.shape

# Drop blank descriptions since we do not know what the customer ordered
df = df.loc[df["Description"] != ""]
print(df.shape)

# Drop records without CustomerID
df = df[pd.notnull(df["CustomerID"] )]
print(df.shape)
```

```
(10624, 8)
geree
(531285, 8)
(397924, 8)
```

In [174]:
```
df.shape
```

Out[174]:
```
(397924, 8)
```

Add a column with the total revenue (sales price) per row (quantity sold times price per unit).

In [175]:
```
# Calculate total sales from the Quantity and UnitPrice (1 point)
df["salesPrice"] = df["Quantity"]*df["UnitPrice"]

df.head()
```

Out[175]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	sa
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	

Create a dataframe at the invoice level (3 Points)

Ultimately we want to say something about customers but first, let's investigate at the order/invoice level. Aggregate the data so that you have a new dataframe with one row per invoice. Keep track of the value of each transaction, the number of unique items sold, the total number of items sold and the customer who bought it. Remember to set your column names appropriately.

In [176]:
```
order_df_colnames = ['InvoiceNo', 'Sales', 'UniqueItems', 'TotalItems',
'CustomerID']
# Create order_df grouping by InvoiceNo (1 point)
# Use an agg function to create Sales, StockCode, Quantity, and Customer
ID as are described above (2 points)

order_df =  df.groupby('InvoiceNo').agg(
    Sales=('salesPrice', 'sum'),
    UniqueItems =('StockCode', 'nunique'),
    TotalItems=('Quantity', 'sum'),
    CustomerID = ('CustomerID', 'unique')
)

#order_df.columns = order_df_colnames
order_df.head()
```

Out[176]:

	Sales	UniqueItems	TotalItems	CustomerID
InvoiceNo				
536365	139.12	7	40	[17850.0]
536366	22.20	2	12	[17850.0]
536367	278.73	12	83	[13047.0]
536368	70.05	4	15	[13047.0]
536369	17.85	1	3	[13047.0]

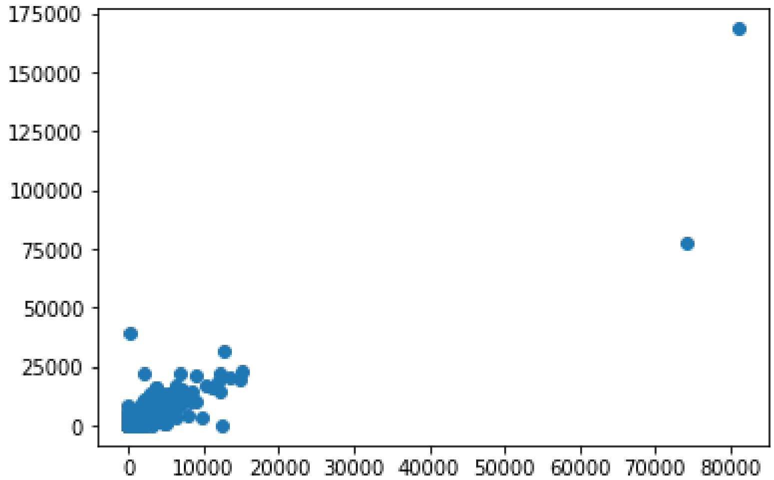
In [177]:
```
order_df.shape
```

Out[177]: (18536, 4)

Visualize the order data and remove excessively large purchases (3 points)

Create a scatter plot of your new dataframe, with total items on the X axis and Sales on the Y axis. You will use this to visualize outliers before removing them. Once you see the plot, you should be able to filter the dataframe on each axis to remove the clear outliers. We chose the Sales and TotalItems axes because we wanted to omit orders where the total sales was extreme or orders with an extreme number of items. The only other column to consider here would be uniqueitems and we do not consider it because the totalitems column already does the same job and intuitively, we want to know total sales and total items.

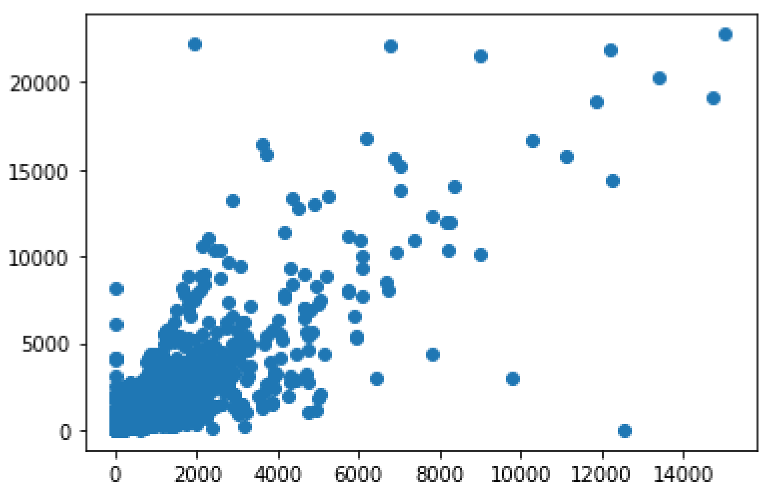
In [178]:
```
#Create a scatter plot (1 point)
fig = plt.scatter(order_df['TotalItems'], order_df['Sales'])
plt.show()
```



Remove outliers witih total items greater than 20000 or sales greater than 30000. Replot (with axis labels) for a better look at the data.

In [179]:
```
#Given the plot above remove outliers on both the totalitems and sales a
xis (1 point each)

#Remove totalitems outliers in order_df using the scatter plot (1 point)
order_df = order_df[order_df["TotalItems"] <= 20000]
# order_df = order_df[order_df["TotalItems"] > 0]
#Remove sales outliers in order_df using the scatter plot (1 point)
order_df = order_df[order_df["Sales"] <= 30000]
# order_df = order_df[order_df["Sales"] > 0]

print(order_df)
# after filtering, my order_df has 18532 rows
fig = plt.scatter(order_df['TotalItems'], order_df['Sales'])
plt.show()
```

```
             Sales  UniqueItems  TotalItems CustomerID
InvoiceNo
536365      139.12            7          40  [17850.0]
536366       22.20            2          12  [17850.0]
536367      278.73           12          83  [13047.0]
536368       70.05            4          15  [13047.0]
536369       17.85            1           3  [13047.0]
...            ...          ...         ...        ...
581583      124.60            2          76  [13777.0]
581584      140.64            2         120  [13777.0]
581585      329.05           21         278  [15804.0]
581586      339.20            4          66  [13113.0]
581587      249.45           15         105  [12680.0]

[18532 rows x 4 columns]
```



Create a customer dataframe and remove outliers using IQR (11 Points)

Create a new dataframe at the customer level using the order dataframe. You can use the invoice dataframe for reference as the process is similar. Columns included should be (1) the total dollar amount of 'sales' across orders, (2) the number of different orders by the customer, (3) the average number of unique items in an order, and (4) the total items ordered across all orders.

In [180]:
```
cust_df_colnames = ['TotalSales', 'OrderCount', 'AvgUniqueItems', 'Total
Items']

#Create the customer dataframe (3 points)
cust_df = df.groupby('CustomerID').agg(
    TotalSales=('salesPrice', 'sum'),
    OrderCount=('InvoiceNo', 'nunique'),
    AvgUniqueItems=('StockCode', 'nunique'),
    TotalItems  =('Quantity','sum'),
)
cust_df.head()

#Add titles to the columns you included (1 point)
#cust_df.columns = ['TotalSales', 'OrderCount', 'AvgUniqueItems', 'Total
Items']
```

Out[180]:

	TotalSales	OrderCount	AvgUniqueItems	TotalItems
CustomerID				
12346.0	77183.60	1	1	74215
12347.0	4310.00	7	103	2458
12348.0	1797.24	4	22	2341
12349.0	1757.55	1	73	631
12350.0	334.40	1	17	197

In [181]:
```
cust_df.shape
```

Out[181]: (4339, 4)

Now add a new column showing each customer's average order value.

In [182]:
```
#Create a new column, AvgOrderValue (total sales divided by # orders per
customer (1 point)
cust_df['AvgOrderValue'] = cust_df['TotalSales'] / cust_df['OrderCount']
cust_df.head()
#print(cust_df)
```

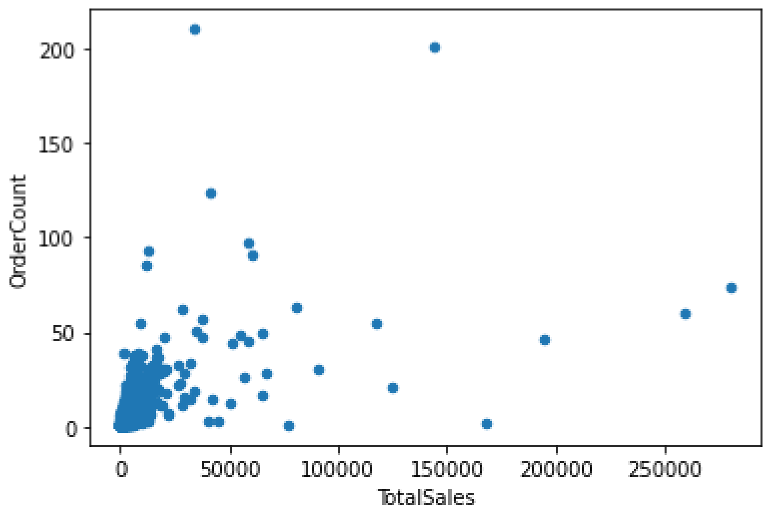
Out[182]:

	TotalSales	OrderCount	AvgUniqueItems	TotalItems	AvgOrderValue
CustomerID					
12346.0	77183.60	1	1	74215	77183.600000
12347.0	4310.00	7	103	2458	615.714286
12348.0	1797.24	4	22	2341	449.310000
12349.0	1757.55	1	73	631	1757.550000
12350.0	334.40	1	17	197	334.400000

Now, create a scatter plot with total sales on the x axis and order count on the y axis to check for outliers

In [183]:
```
#Create a scatter plot of total sales and order count to visualize our c
urrent data
cust_df.plot.scatter(x='TotalSales', y='OrderCount')
print(cust_df.shape)
plt.show()
```

```
(4339, 5)
```



This time, let's calculate the interquartile range (IQR), the difference between the upper and lower quartiles, and use this to compute lower (upper)bounds on the `Sales` column equal to the $Q_1 - 1.5 \times IQR$ (and $Q_3 + 1.5 \times IQR$, respectively).

In [184]:
```
#Instead of using a visual heuristic and boxplot, let's try implementing
IQR to detect our outliers. (2 points)

sales_quartile_75 = np.quantile(cust_df['TotalSales'], 0.75)
sales_quartile_25 = np.quantile(cust_df['TotalSales'], 0.25)
sales_iqr = sales_quartile_75 - sales_quartile_25

print(sales_iqr)

sales_lbound = sales_quartile_25 - 1.5 * sales_iqr
sales_ubound = sales_quartile_75 + 1.5 * sales_iqr

print(sales_lbound, sales_ubound)
```

```
1354.395
-1724.3474999999999 3693.2325
```

Now, repeat the process to compute similar bounds on the number of orders per customer.

In [185]:
```
#Instead of using a visual heuristic and boxplot, let's try implementing
IQR to detect our outliers. (1 point)

order_quartile_75 = np.quantile(cust_df['OrderCount'], 0.75)
order_quartile_25 = np.quantile(cust_df['OrderCount'], 0.25)
order_iqr = order_quartile_75 - order_quartile_25


order_ubound = order_quartile_75 + 1.5 * order_iqr
order_lbound = order_quartile_25 - 1.5 * order_iqr
print(order_lbound, order_ubound)
```

```
-5.0 11.0
```

Now, filter the dateframe to exclude rows where the orders or sales value exceed the bounds you computed.

In [186]:
```
#(1 point)

cust_df = cust_df[cust_df['OrderCount'] > order_lbound]
cust_df = cust_df[cust_df['OrderCount'] < order_ubound]

cust_df = cust_df[cust_df['TotalSales'] > sales_lbound]
cust_df = cust_df[cust_df['TotalSales'] < sales_ubound]
print(cust_df)
```

```
            TotalSales  OrderCount  AvgUniqueItems  TotalItems  AvgOrde
rValue
CustomerID
12348.0        1797.24           4              22        2341      449.
310000
12349.0        1757.55           1              73         631     1757.
550000
12350.0         334.40           1              17         197      334.
400000
12352.0        2506.04           8              59         536      313.
255000
12353.0          89.00           1               4          20       89.
000000
...                ...         ...             ...         ...
...
18278.0         173.90           1               9          66      173.
900000
18280.0         180.60           1              10          45      180.
600000
18281.0          80.82           1               7          54       80.
820000
18282.0         178.05           2              12         103       89.
025000
18287.0        1837.28           3              59        1586      612.
426667

[3840 rows x 5 columns]
```

Finally, create a normalized dataframe, by standardizing each column of cust_df (subtract the mean, scale by the standard deviation). You should be able to do this in one line of code.

In [187]:
```
#Create normalized_df, a normalized version of cust_df (2 points)
# def absolute_maximum_scale(series):
#     return series / series.abs().max()
normalized_df = pd.DataFrame()
for col in cust_df.columns:
  # normalized_df = (rank_df - rank_df.mean()) / rank_df.std()
  normalized_df[col] = (cust_df[col] - cust_df[col].mean()) / cust_df[co
l].std()
    # normalized_df[col] = absolute_maximum_scale(cust_df[col])
print(cust_df)
```

```
            TotalSales  OrderCount  AvgUniqueItems  TotalItems  AvgOrde
rValue
CustomerID
12348.0        1797.24           4              22        2341      449.
310000
12349.0        1757.55           1              73         631     1757.
550000
12350.0         334.40           1              17         197      334.
400000
12352.0        2506.04           8              59         536      313.
255000
12353.0          89.00           1               4          20       89.
000000
...                ...         ...             ...         ...
...
18278.0         173.90           1               9          66      173.
900000
18280.0         180.60           1              10          45      180.
600000
18281.0          80.82           1               7          54       80.
820000
18282.0         178.05           2              12         103       89.
025000
18287.0        1837.28           3              59        1586      612.
426667

[3840 rows x 5 columns]
```

In [188]:
```
normalized_df.shape
```

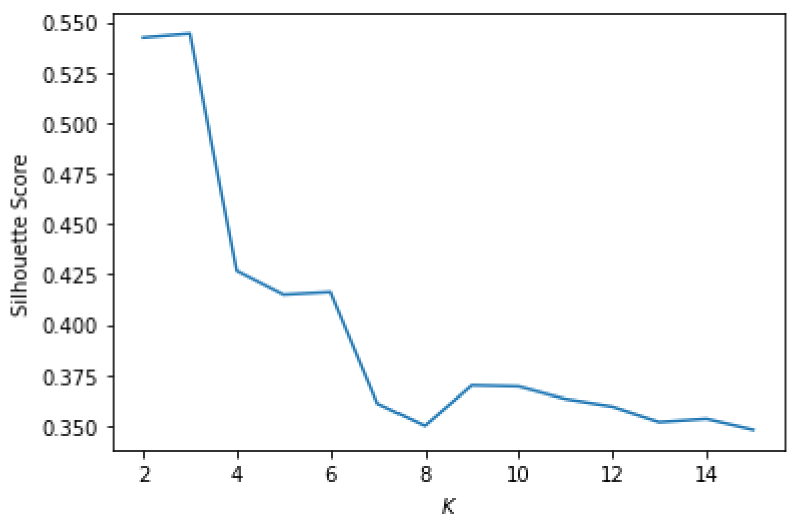
Out[188]:
```
(3840, 5)
```

You'll notice that we were pretty aggressive in removing outliers, we went from 4338 customers to 3841. It'll help us get cleaner clusters later for the purpose of this exercise, but in practice we should be more careful.

K-Means Clustering Algorithm (5 points)

We don't know how many clusters is appropriate, so we will do k-means clustering for $k \in [2, 3, \ldots, 15]$ and use the [silhouette-score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html) to choose the best value of k. Be sure to read the documentation to understand silhouettes, briefly, higher scores are better. Training may take a few seconds. I used `random_state=3` , generally note that since teh starting position is random we may have small differences in results.

In [189]:
```
from numpy.core.fromnumeric import shape
#(5 points)
from sklearn.metrics import silhouette_score

krange = list(range(2,16))
cols = ['TotalSales', 'OrderCount',  'TotalItems', 'AvgOrderValue']
X = normalized_df[cols].values


silhouette = [None for i in range(2,16)]

# Iterate over the range of K values, which denotes the number of cluste
rs
for n in krange:
  # your code here
  kmeans = KMeans(n_clusters=n)
  kmeans.fit(X)
  # I do n-2 here to prevent list index out of range error- on the graph
0 == no. of clusters == 2
  silhouette[n-2] = silhouette_score(X, kmeans.labels_)



kmeans = KMeans(n_clusters=2)
kmeans.fit(X)



plt.plot(krange, silhouette)
plt.xlabel("$K$")
plt.ylabel("Silhouette Score")
plt.show()
```



In [190]:
```
normalized_df.shape
```

Out[190]: (3840, 5)

Investigate the clusters (4 points)

Using the plot above, identify the best choice of k. Run k-means clustering with the chosen k, then create a new dataframe with an additional column showing the cluster of every customer. You should investigate your clusters to make sure a cluster doesn't just consist of one or two outliers.

In [191]:
```
#Set the K value and run the kmeans algorithm on the normalized datafram
e (1 point)
k = 3
kmeans = KMeans(n_clusters=k).fit(normalized_df[cols])

#copy the columns from normalized_df
cluster_df = normalized_df[cols].copy(deep=True)

#Add the labels from the k-means algoithm to the cluster column in clust
er_df (1 point)
cluster_df['Cluster'] =  kmeans.labels_
print(kmeans.cluster_centers_)

#Run groupby to see how many instances are in each cluster
x = cluster_df.groupby('Cluster')
x.head()
```

```
[[-0.50643891 -0.40572141 -0.42098858 -0.2846267 ]
 [ 1.40230995  1.49824384  1.20635622  0.13358636]
 [ 0.94740376 -0.52061014  0.64875706  2.76699231]]
```

Out[191]:

	TotalSales	OrderCount	TotalItems	AvgOrderValue	Cluster
CustomerID					
12348.0	1.188652	0.618487	2.935939	0.469258	1
12349.0	1.138702	-0.824107	0.170819	5.511570	2
12350.0	-0.652326	-0.824107	-0.530972	0.026363	0
12352.0	2.080674	2.541945	0.017201	-0.055135	1
12353.0	-0.961161	-0.824107	-0.817186	-0.919475	0
12354.0	0.285253	-0.824107	0.007499	2.897796	2
12355.0	-0.495014	-0.824107	-0.461439	0.508147	0
12356.0	2.465006	0.137622	1.723167	2.349500	2
12358.0	0.396832	-0.343242	-0.448503	0.988505	0
12360.0	2.277025	0.137622	1.034312	2.157596	2
12361.0	-0.834179	-0.824107	-0.702377	-0.530579	0
12364.0	0.579364	0.618487	1.585719	0.002756	1
12370.0	3.389069	0.618487	2.955343	2.154008	1
12371.0	1.302823	-0.343242	0.106138	2.375850	2
12380.0	2.355995	0.618487	0.972865	1.363034	1

Create a new dataframe with the centroid of each cluster. You can easily access the centroids in your estimators `cluster_centers_` attribute.

In [192]:
```
#(2 points)
centroids  = kmeans.cluster_centers_
#Create a df using the centroids stored in the previous step
cluster_center_df = pd.DataFrame(centroids)
#Rename the columns of your df to the correct names

cluster_center_df.columns = cols
cluster_center_df.head()
```

Out[192]:

	TotalSales	OrderCount	TotalItems	AvgOrderValue
0	-0.506439	-0.405721	-0.420989	-0.284627
1	1.402310	1.498244	1.206356	0.133586
2	0.947404	-0.520610	0.648757	2.766992

Visualize and interpret clusters (8 points)

Create scatter plots to visualize the relationship between your features and clusters. The template iterates over the respective x, y axes of each plot. You should create one pane with four plots using `cluster_df` .

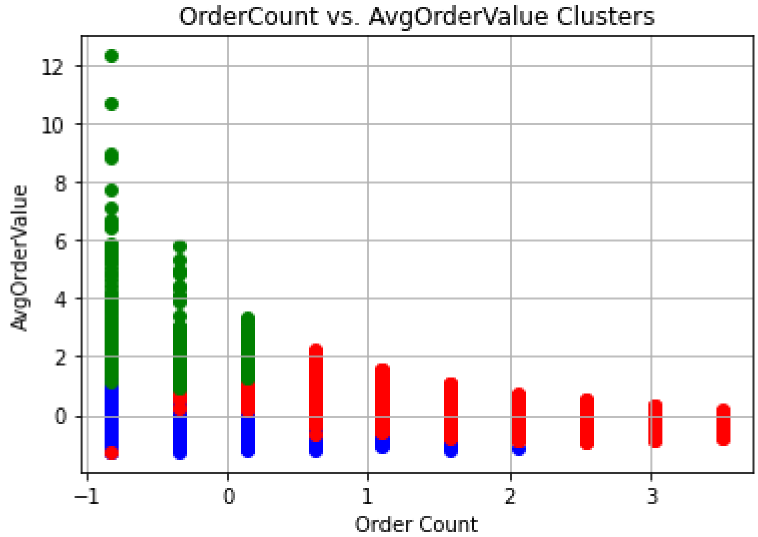
The plots in the second pane were created by first ranking the data in `cluster_df` . You can try to do this if it helps you interpret the clusters, but will not be penalised if you don't.

In [193]:

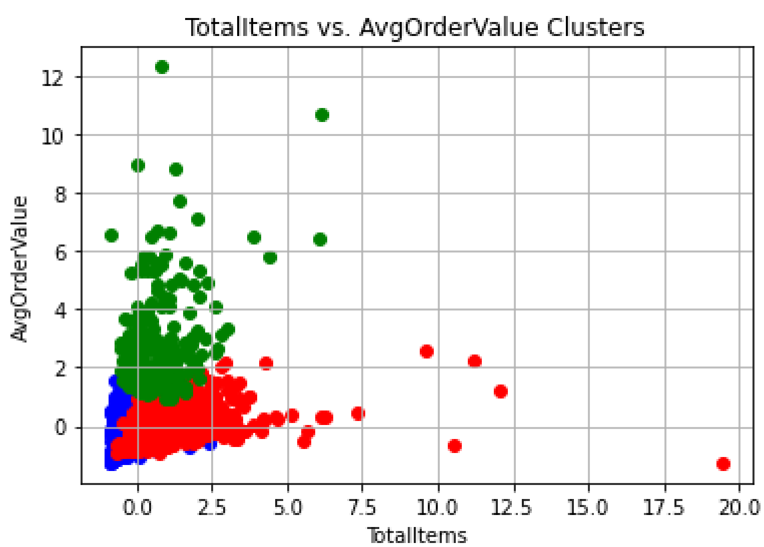
```
#(3 points), (1 bonus point for plotting on rankings)

plots = [('OrderCount', 'AvgOrderValue'),   ('OrderCount', 'TotalItems'
), ('TotalItems', 'AvgOrderValue') , ('TotalSales', 'OrderCount') ]
colors = [ 'blue', 'orange', 'green', 'purple']

plot_df = cluster_df
#fig, axs = plt.subplots(1, len(plots) , figsize=(25, 5))

# for idx, col_pair in zip(range(len(plots)), plots):
#   print(idx)
#   print(col_pair)

#   #Iterate through all the clusters with a different color per cluster
#   for cluster in range(k):

#     #Create a scatter plot based on the X and Y axis in each plot, p u
sing the colors variable (2 point)
#     plt.scatter(
#     plot_df.loc[plot_df['Cluster'] == cluster][col_pair[0]],
#     plot_df.loc[plot_df['Cluster'] == cluster][col_pair[1]],
#     c=colors[cluster]
# )

plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 0]['OrderCount'],
    plot_df.loc[plot_df['Cluster'] == 0]['AvgOrderValue'],
    c='blue'
)

plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 1]['OrderCount'],
    plot_df.loc[plot_df['Cluster'] == 1]['AvgOrderValue'],
    c='red'
)
plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 2]['OrderCount'],
    plot_df.loc[plot_df['Cluster'] == 2]['AvgOrderValue'],
    c='green'
)

plt.title('OrderCount vs. AvgOrderValue Clusters')
plt.xlabel('Order Count')
plt.ylabel('AvgOrderValue')
plt.grid()
plt.show()
  #Set axis labels (1 point)


plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 0]['OrderCount'],
    plot_df.loc[plot_df['Cluster'] == 0]['TotalItems'],
    c='blue'
)

plt.scatter(
```

```
    plot_df.loc[plot_df['Cluster'] == 1]['OrderCount'],
    plot_df.loc[plot_df['Cluster'] == 1]['TotalItems'],
    c='red'
)
plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 2]['OrderCount'],
    plot_df.loc[plot_df['Cluster'] == 2]['TotalItems'],
    c='green'
)


plt.title('OrderCount vs. TotalItems Clusters')
plt.xlabel('Order Count')
plt.ylabel('TotalItems')
plt.grid()
plt.show()


    #Set axis labels (1 point)
plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 0]['TotalItems'],
    plot_df.loc[plot_df['Cluster'] == 0]['AvgOrderValue'],
    c='blue'
)

plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 1]['TotalItems'],
    plot_df.loc[plot_df['Cluster'] == 1]['AvgOrderValue'],
    c='red'
)
plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 2]['TotalItems'],
    plot_df.loc[plot_df['Cluster'] == 2]['AvgOrderValue'],
    c='green'
)


plt.title('TotalItems vs. AvgOrderValue Clusters')
plt.xlabel('TotalItems')
plt.ylabel('AvgOrderValue')
plt.grid()
plt.show()

# Plot 4
plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 0]['TotalSales'],
    plot_df.loc[plot_df['Cluster'] == 0]['OrderCount'],
    c='blue'
)

plt.scatter(
    plot_df.loc[plot_df['Cluster'] == 1]['TotalSales'],
    plot_df.loc[plot_df['Cluster'] == 1]['OrderCount'],
    c='red'
)

plt.scatter(
```

```
    plot_df.loc[plot_df['Cluster'] == 2]['TotalSales'],
    plot_df.loc[plot_df['Cluster'] == 2]['OrderCount'],
    c='green'
)



plt.title('TotalSales vs. OrderCount Clusters')
plt.xlabel('TotalSales')
plt.ylabel('OrderCount')
plt.grid()
plt.show()
```


OrderCount vs. AvgOrderValue Clusters
AvgOrderValue
Order Count


OrderCount vs. TotalItems Clusters
TotalItems
Order Count


TotalItems vs. AvgOrderValue Clusters
AvgOrderValue
TotalItems

Another useful visualization that allows you to compare clusters is a polar plot of the cluster centers. For this we'll use the plotly library.

In [194]:
```
import plotly.express as px
polar_data=cluster_center_df.reset_index()
polar_data=pd.melt(polar_data,id_vars=['index'])
px.line_polar(polar_data, r='value', theta='variable', color='index', li
ne_close=True, height=400,width=400)
```

Thouroughly characterize the types of customers and their purchase behaviour for each cluster. (5 points)

In [195]:
```
# Blue: relatively low number of orders, lowest number of items and lowest avg order value. Low volume/low-med frequency
print("As you can see from visualizations above: Cluster 0 can be characterised as Blue since they have relatively low number of orderCount (cluster 1 has lowest), lowest number of items , and lower average order value among the 3 clusters ")
# Red: Highest average order value, relatively small number of items per order so on average more expensive items. Low volume, high value.
print("As you can see from visualizations above: Cluster 1 can be characterised as Red since they have low number of items per order (Total Items / order Count) and highest average order value among the 3 clusters  "
)

# Green: largest number of orders and large number of items. ordervalue varies, so they are buying a lot, but not necessarily expensive things. high volume + frequency.
print("As you can see from visualizations above: Cluster 2 can be characterised as Green since they have highest number of orderCount (largest number of orders), largest number of Total items and its order value is relatively low compared to other clusters and varies but concentrated around less expensive items ")
```

```
As you can see from visualizations above: Cluster 0 can be characterised as Blue since they have relatively low number of orderCount (cluster 1 has lowest), lowest number of items , and lower average order value among the 3 clusters
As you can see from visualizations above: Cluster 1 can be characterised as Red since they have low number of items per order (Total Items / order Count) and highest average order value among the 3 clusters
As you can see from visualizations above: Cluster 2 can be characterised as Green since they have highest number of orderCount (largest number of orders), largest number of Total items and its order value is relatively low compared to other clusters and varies but concentrated around less expensive items
```

Dive Deeper into the High Value Cluster and display the Top Products (4 points)

Investigate the cluster with the highest order value (on average) further by printing the top 10 best-selling products in the cluster. You will need to use your original dataframe coupled with the cluster number of each customer.

In [196]:
```
high_value_cluster_number = 1  #(1 point)
#cluster Red has highest value customers


# filter to get the customers in the high value cluster
high_value_cluster =  cluster_df.loc[cluster_df['Cluster'] == high_value
_cluster_number]
# print(high_value_cluster)
# identify the most commonly purchased items  (3 point)
custID = high_value_cluster.index
# print(custID)

print(df.shape)
maxBought = df.loc[df['CustomerID'].isin(custID)]
freq = maxBought['Description']
print("Highest value customers top 10 most bought products: ")
print(freq.value_counts()[:10])
print()
print()


print("for sake of further analysis I look for least useful products to
 have in the inventory")
print()
lowest_value_cluster_number = 0
lowest_value_cluster =  cluster_df.loc[cluster_df['Cluster'] == lowest_v
alue_cluster_number]
custIDL = lowest_value_cluster.index

minBought = df.loc[df['CustomerID'].isin(custIDL)]
freqL = minBought['Description']
print("Lowest value customers top 10 most bought products: ")
print(freqL.value_counts()[:10])
```

```
(397924, 9)
Highest value customers top 10 most bought products:
WHITE HANGING HEART T-LIGHT HOLDER    588
ASSORTED COLOUR BIRD ORNAMENT         426
REGENCY CAKESTAND 3 TIER              412
JUMBO BAG RED RETROSPOT               396
LUNCH BAG RED RETROSPOT               375
PARTY BUNTING                         368
LUNCH BAG  BLACK SKULL.               349
NATURAL SLATE HEART CHALKBOARD        344
HEART OF WICKER SMALL                 337
PACK OF 72 RETROSPOT CAKE CASES       321
Name: Description, dtype: int64


for sake of further analysis I look for least useful products to have i
n the inventory

Lowest value customers top 10 most bought products:
WHITE HANGING HEART T-LIGHT HOLDER    567
REGENCY CAKESTAND 3 TIER              437
REX CASH+CARRY JUMBO SHOPPER          424
ASSORTED COLOUR BIRD ORNAMENT         400
PARTY BUNTING                         337
BAKING SET 9 PIECE RETROSPOT          306
PAPER CHAIN KIT 50'S CHRISTMAS        296
HEART OF WICKER SMALL                 289
NATURAL SLATE HEART CHALKBOARD        267
SET OF 3 CAKE TINS PANTRY DESIGN      266
Name: Description, dtype: int64
```

Inform Strategy (6 points)

Now that you have a better understanding of customers' purchase behaviour, how would you change your practices?

Write 1-2 sentences with a business recommendation (this can cover marketing, operations, etc. as long as it refers back to the results of your cluster analysis) for each of the clusters.

Based of the analysis we have conducted I would recommend the business to focus on only high value customers as they are more profitable to us. I checked the least valuable products included in our inventory and would cut my stock of these products while increasing stock of more valuable products (products more frequently bought by cluster 1) and (cut products inventory for products bought by cluster 0). I will then also suggest marketing advertisements that will optimise sales for cluster 2 by offering bundling options at a discounted rate to these customers since they are more frequent buyers.

Collaboration statement (3 points)

Include the names of everyone that helped you with this homework and explain how each person helped. Also include the names of everyone you helped, and explain how. Asking for guidance is perfectly fine, but please do not ask for or share exact solutions. You should leave any discussion of the assignment and go write up your solutions on your own.

If you do not submit a collaboration statement you will receive 0/3 for this section. Even if you did not collaborate with anyone, you still need to write a statement below.

I helped Neeraja Mehta by explaining the concept of kmeans clustering to her so she can better understand the assignment.

Preparing for submission

To convert your notebook to html, change the string below to reflect the location of the notebook in your Google Drive.

In [200]:
```
path_to_file = '/content/drive/My Drive/ColabNotebooks/PA1-market-segmentation.ipynb'
```

Now execute the code cell below. After execution there should be an html file in the same Google Drive folder where this notebook is located. Download the html file, open with your browser and print to pdf, then submit the pdf on the course page along with your notebook.

NOTE: this seems to fail if your path contains spaces - move it to a location without spaces and try again.

In [201]:
```
!apt update
!apt install texlive-xetex texlive-fonts-recommended texlive-generic-rec
ommended

import re, pathlib, shutil

from google.colab import drive
drive.mount('/content/drive')
```

```
Hit:1 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRel
ease
Hit:3 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu bionic-backports InRelease
Ign:5 https://developer.download.nvidia.com/compute/machine-learning/re
pos/ubuntu1804/x86_64  InRelease
Hit:6 http://security.ubuntu.com/ubuntu bionic-security InRelease
Hit:7 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu18
04/x86_64  InRelease
Hit:8 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRel
ease
Hit:9 https://developer.download.nvidia.com/compute/machine-learning/re
pos/ubuntu1804/x86_64  Release
Hit:10 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Hit:11 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Hit:12 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRe
lease
Reading package lists... Done
Building dependency tree
Reading state information... Done
20 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
texlive-fonts-recommended is already the newest version (2017.20180305-
1).
texlive-generic-recommended is already the newest version (2017.2018030
5-1).
texlive-xetex is already the newest version (2017.20180305-1).
The following package was automatically installed and is no longer requ
ired:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 20 not upgraded.
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [199]:
```
nbpath =  pathlib.PosixPath(path_to_file)
!jupyter nbconvert "{nbpath.as_posix()}" --to html --output "{nbpath.stem.replace(" ", "_")}"
```

```
[NbConvertApp] WARNING | pattern '/content/drive/My Drive/ColabNotebook
s/PA1-market-segmentation-solution.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASE
S.

Options
=======
The options below are convenience aliases to configurable class-option
s,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an erro
r and include the error message in the cell output (the default behavio
ur is to abort conversion). This flag is only relevant if '--execute' w
as specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting noteboo
k with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertA
pp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertA
pp.export_format=notebook --FilesWriter.build_directory= --ClearOutputP
```

```
reprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --Temp
lateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --Tem
plateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'E
RROR', 'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebo
ok', 'pdf', 'python', 'rst', 'script', 'slides']
            or a dotted object name that represents the import path for
an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                    results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                    results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a tim
e.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each no
tebook. To recover
                                  previous default behaviour (outputtin
g to the current
                                  working directory) use . as the flag
value.
```

```
        Default: ''
        Equivalent to: [--FilesWriter.build_directory]
    --reveal-prefix=<Unicode>
        The URL prefix for reveal.js (version 3.x).
                This defaults to the reveal CDN, but can be any url pointin
g to a copy
                of reveal.js.
                For speaker notes to work, this must be a relative path to
a local
                copy of reveal.js: e.g., "reveal.js".
                If a relative path is given, it must be a subdirectory of t
he
                current directory (from which the server is run).
                See the usage documentation
                (https://nbconvert.readthedocs.io/en/latest/usage.html#reve
al-js-html-slideshow)
                for more details.
        Default: ''
        Equivalent to: [--SlidesExporter.reveal_url_prefix]
    --nbformat=<Enum>
        The nbformat version to write.
                Use this to downgrade notebooks.
        Choices: any of [1, 2, 3, 4]
        Default: 4
        Equivalent to: [--NotebookExporter.nbformat_version]

    Examples
    --------

        The simplest way to use nbconvert is

                > jupyter nbconvert mynotebook.ipynb

                which will convert mynotebook.ipynb to the default format
(probably HTML).

                You can specify the export format with `--to`.
                Options include ['asciidoc', 'custom', 'html', 'latex', 'ma
rkdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

                > jupyter nbconvert --to latex mynotebook.ipynb

                Both HTML and LaTeX support multiple output templates. LaTe
X includes
                'base', 'article' and 'report'.  HTML includes 'basic' and
'full'. You
                can specify the flavor of the format used.

                > jupyter nbconvert --to html --template basic mynotebook.i
pynb

                You can also pipe the output to stdout, rather than a file

                > jupyter nbconvert mynotebook.ipynb --stdout

                PDF is generated via latex
```

```
            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a co
uple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, con
taining::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```