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 [1]:
```
# 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 [3]:
```
print(df.shape)
df.head(10)
```

```
(541909, 8)
```

Out[3]:

	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 [4]:
```
#Check for null values in the dataframe by column
df.isnull().sum()
```

Out[4]:
```
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 [5]:
```
# (2 points)
df = df[df.Quantity > 0]

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

# Drop records without CustomerID
df.dropna(how = 'any',subset = ['CustomerID'],inplace = True)
```

In [6]:
```
df.shape
```

Out[6]:
```
(397924, 8)
```

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

In [7]:
```
df['Sales'] = df['UnitPrice']*df['Quantity']
df.head()
```

Out[7]:

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

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 [8]:
```
order_df_colnames = ['InvoiceNo', 'Sales', 'UniqueItems', 'TotalItems',
'CustomerID']
order_df = df.groupby('InvoiceNo').agg({'Sales':np.sum,'StockCode':'nuni
que','Quantity':np.sum,'CustomerID':np.unique}).reset_index()
order_df.columns = order_df_colnames
order_df.head()
```

Out[8]:

	InvoiceNo	Sales	UniqueItems	TotalItems	CustomerID
0	536365	139.12	7	40	17850.0
1	536366	22.20	2	12	17850.0
2	536367	278.73	12	83	13047.0
3	536368	70.05	4	15	13047.0
4	536369	17.85	1	3	13047.0

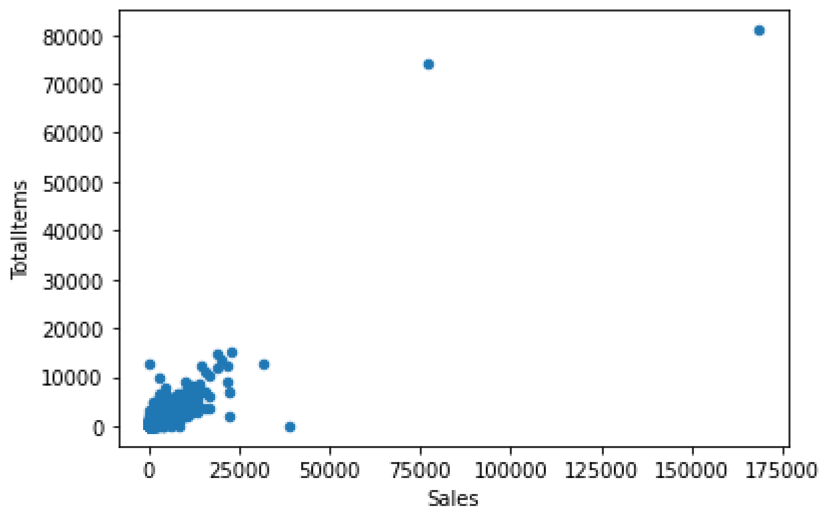
In [9]:
```
order_df.shape
```

Out[9]: (18536, 5)

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.

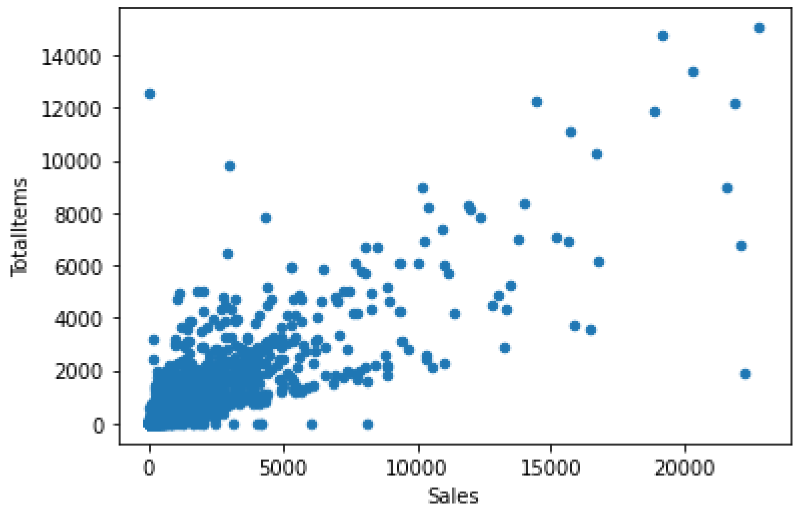
In [10]:
```
order_df.plot.scatter(x = 'Sales', y = 'TotalItems')
plt.show()
```



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 [11]:
```
order_df = order_df[(order_df.Sales <= 30000) & (order_df.TotalItems <= 20000)]
order_df.plot.scatter(x = 'Sales', y = 'TotalItems')
len(order_df)
```

Out[11]: 18532



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 [13]:
```
cust_df = order_df.groupby('CustomerID').agg({'Sales':np.sum,'InvoiceNo':'nunique','UniqueItems':np.mean,'TotalItems':[np.sum]})
cust_df.columns = ['TotalSales', 'OrderCount', 'AvgUniqueItems', 'TotalItems']
cust_df.head(5)
```

Out[13]:

	TotalSales	OrderCount	AvgUniqueItems	TotalItems
CustomerID				
12347.0	4310.00	7	26.000	2458
12348.0	1797.24	4	6.750	2341
12349.0	1757.55	1	73.000	631
12350.0	334.40	1	17.000	197
12352.0	2506.04	8	10.375	536

In [14]:
```
cust_df.shape
```

Out[14]: (4338, 4)

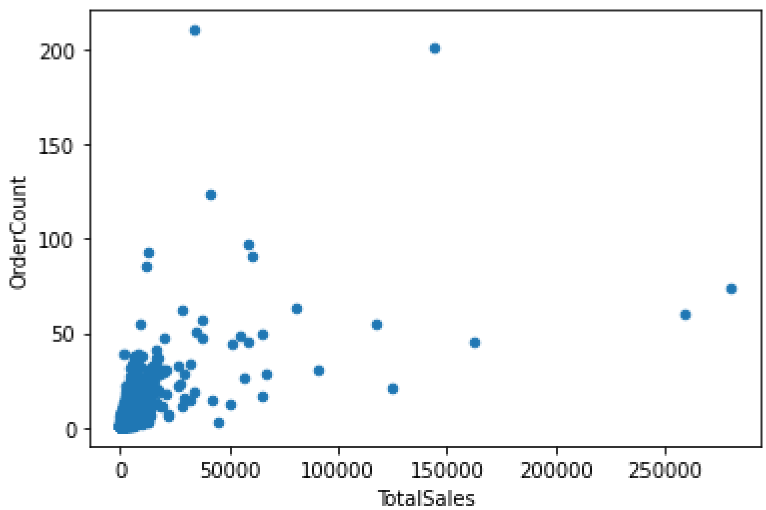
Now add a new column showing each customer's average order value.

In [15]:
```
cust_df['AvgOrderValue'] = cust_df['TotalSales']/cust_df['OrderCount']
```

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

In [16]:
```
#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()
```

```
(4338, 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 [17]:
```
#Instead of using a visual heuristic and boxplot, let's try implementing
IQR to detect our outliers. (2 points)
sales_quartile_75, sales_quartile_25 = np.quantile(cust_df.TotalSales,[
0.75,0.25])

sales_iqr = sales_quartile_75 - sales_quartile_25

sales_ubound, sales_lbound = sales_quartile_75 + 1.5*sales_iqr , sales_q
uartile_25 - 1.5*sales_iqr
print(sales_lbound, sales_ubound)
```

```
-1723.9362500000002 3691.99375
```

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

In [18]:
```
#Instead of using a visual heuristic and boxplot, let's try implementing
IQR to detect our outliers. (1 point)
order_quartile_75, order_quartile_25 = np.quantile(cust_df.OrderCount,[
0.75,0.25])

order_iqr = order_quartile_75 - order_quartile_25

order_ubound, order_lbound = order_quartile_75 + 1.5*order_iqr, order_qu
artile_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 [19]:
```
cust_df = cust_df[(cust_df.OrderCount > order_lbound) & (cust_df.OrderCo
unt < order_ubound) & (cust_df.TotalSales > sales_lbound) & (cust_df.Tot
alSales < sales_ubound)]
cust_df
```

Out[19]:

	TotalSales	OrderCount	AvgUniqueItems	TotalItems	AvgOrderValue
CustomerID					
12348.0	1797.24	4	6.750000	2341	449.310000
12349.0	1757.55	1	73.000000	631	1757.550000
12350.0	334.40	1	17.000000	197	334.400000
12352.0	2506.04	8	10.375000	536	313.255000
12353.0	89.00	1	4.000000	20	89.000000
...	...	...	...	...	...
18278.0	173.90	1	9.000000	66	173.900000
18280.0	180.60	1	10.000000	45	180.600000
18281.0	80.82	1	7.000000	54	80.820000
18282.0	178.05	2	6.000000	103	89.025000
18287.0	1837.28	3	22.666667	1586	612.426667

3841 rows × 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 [20]:
```
normalized_df = (cust_df - cust_df.mean())/cust_df.std()
```

In [21]:
```
normalized_df.shape
```

Out[21]:
```
(3841, 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) 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.

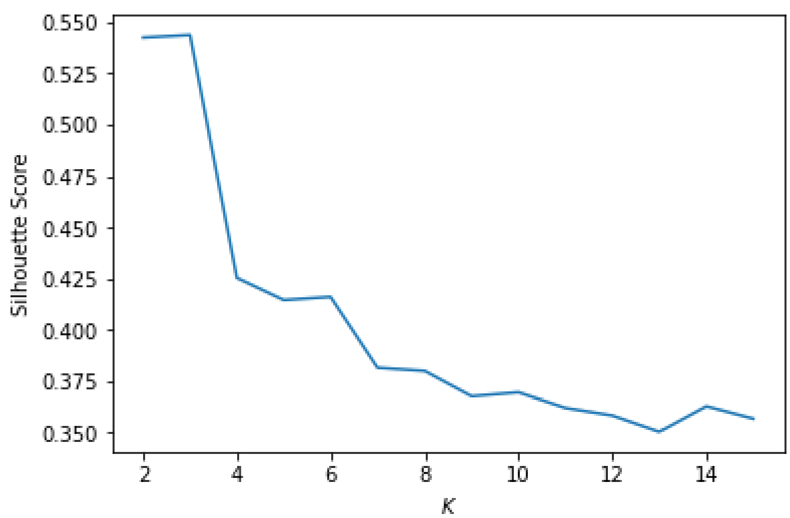
In [22]:
```
#(5 points)
from sklearn.metrics import silhouette_score

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

silhouette = []

# Iterate over the range of K values, which denotes the number of cluste
rs
for n in krange:
  clusterer = KMeans(n_clusters = n, random_state = 3)
  clusterer.fit(X)
  silhouette += [silhouette_score(X, clusterer.labels_)]
silhouette

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



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 [30]:
```
#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][:]

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

#Run groupby to see how many instances are in each cluster
cluster_df.Cluster.value_counts()
```

Out[30]:
```
1    2755
0     842
2     244
Name: Cluster, dtype: int64
```

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

In [31]:
```
#(2 points)
centers = kmeans.cluster_centers_
#Create a df using the centroids stored in the previous step
cluster_center_df = pd.DataFrame(kmeans.cluster_centers_)
#Rename the columns of your df to the correct names
cluster_center_df.columns = ['TotalSales','OrderCount','TotalItems','Avg
OrderValue']
cluster_center_df
```

Out[31]:

	TotalSales	OrderCount	TotalItems	AvgOrderValue
0	1.391095	1.491340	1.194369	0.130675
1	-0.509306	-0.409717	-0.422617	-0.284992
2	0.950144	-0.520240	0.650215	2.766906

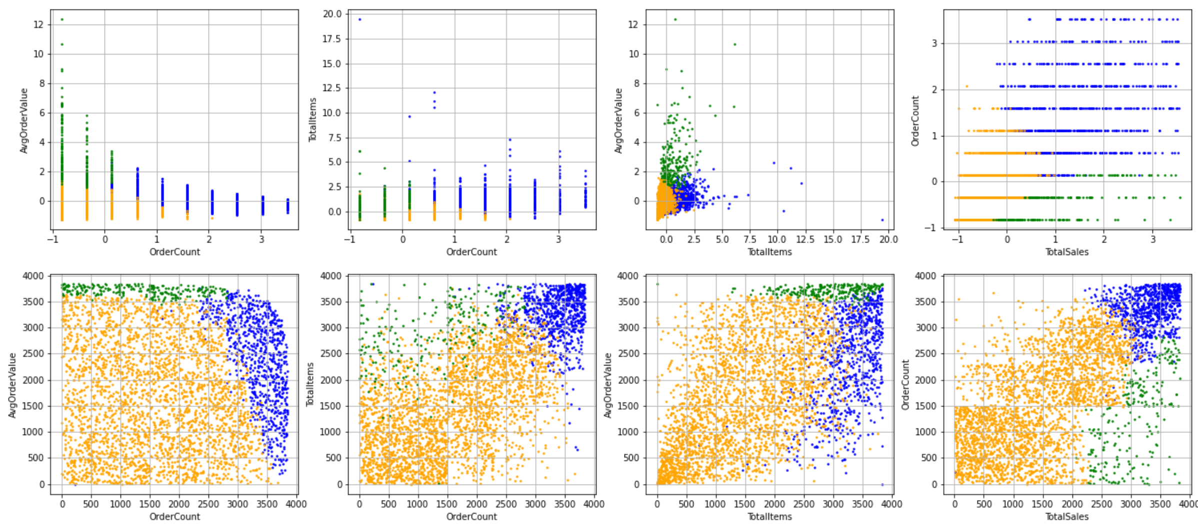
Visualize and interpret clusters (8 points)

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 [32]:
```
plots = [('OrderCount', 'AvgOrderValue'),   ('OrderCount', 'TotalItems'
), ('TotalItems', 'AvgOrderValue') , ('TotalSales', 'OrderCount') ]
colors = [ 'blue', 'orange', 'green', 'purple']

dataset = [cluster_df,cluster_df.rank(method = 'first')]
fig, axs = plt.subplots(len(dataset), len(plots) , figsize=(23, 10))
for dfnum,plot_df in enumerate(dataset):
  plot_df['Cluster'] = kmeans.labels_
  for idx, col_pair in zip(range(len(plots)), plots):
    #Iterate through all the clusters with a different color per cluster
    for cluster in range(k):
      draw = plot_df[plot_df.Cluster == cluster]
      X,y = col_pair
      axs[dfnum][idx].scatter(x = draw[X],y = draw[y],color = colors[clu
ster],s = 2.5)
      axs[dfnum][idx].set_xlabel(X)
      axs[dfnum][idx].set_ylabel(y)
    axs[dfnum][idx].grid()
```



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 [33]:
```
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)

The blue cluster tends to order more frequently, has the highest total sales and total amount of items oredered but a relatively small average order value each time

The green cluster make orders very rarely but has the highest average order value and relatively highe total sales and total items

The red cluster has both low total sales, order counts, average order value and amount of total 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 [35]:
```
cust_df['Cluster'] = kmeans.labels_
high_value_cluster_number =  cust_df.groupby('Cluster')['AvgOrderValue']
.mean().idxmax()
# filter to get the customers in the high value cluster
filtered_cust_df = cust_df[cust_df.Cluster == 2]
# identify the most commonly purchased items  (3 point)
df[df['CustomerID'].isin(filtered_cust_df.index)].groupby('Description')
['Quantity'].sum().nlargest(10)
```

Out[35]:
```
Description
WORLD WAR 2 GLIDERS ASSTD DESIGNS    6336
SMALL POPCORN HOLDER                 4487
JUMBO BAG RED RETROSPOT              2366
RED  HARMONICA IN BOX                1940
PACK OF 72 RETROSPOT CAKE CASES      1637
VINTAGE DOILY JUMBO BAG RED          1581
SMALL CERAMIC TOP STORAGE JAR        1576
60 TEATIME FAIRY CAKE CASES          1241
RED RETROSPOT CHARLOTTE BAG          1182
ASSORTED COLOUR BIRD ORNAMENT        1061
Name: Quantity, 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.

Customer who are in blue cluster could be our loyal retail customer that make frequent purchases in relatively small amount. A loyalty program could be aopted to retain their trust.

Customer who are in green cluster can be a business customer that make rare big purcahse. Frequent communication with them is vital so that our operation depaterment can fulfill their need in time.

More marketing effort should be placed on customer who are in red customer to move them to blue or green cluster.

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.

No one helped me and I did not helped anyone for this 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 [42]:
```
path_to_file = '/content/drive/MyDrive/PA1.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 [45]:
```
#!apt update
#!apt install texlive-xetex texlive-fonts-recommended texlive-generic-re
commended

#import re, pathlib, shutil

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

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [47]:
```
nbpath =  pathlib.PosixPath(path_to_file)
!jupyter nbconvert "{nbpath.as_posix()}" --to html --output "{nbpath.ste
m.replace(" ", "_")}"
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/PA1.ipynb to
html
[NbConvertApp] Writing 669395 bytes to /content/drive/MyDrive/PA1.html
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```