Customer Segmentation On E-Commerce

Technical Stacks:

1. Programming Language: Python.
2. Libraries - Pandas, Matplotlib, Seaborn, Plotly, NLTK, Scikit Learn, Word Cloud.
3. IDE: Jupyter Lab, Kaggle.
4. Version Control: Git, GitHub.

Dataset:

This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail.The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers.
The store is UK based and registered. Non-store online retail means the merchandising of goods by means other than retail shops; merchandising by mail order, vending machines, telephone, door-to-door, etc.
Wholesalers means they are running their own shops, and they are purchasing the gifts from this company in large quantities which will be sold in small quantities in their own store.
NOTE: Per the UCI Machine Learning Repository, this data was made available by Dr Daqing Chen, Director: Public Analytics group. chend '@' lsbu.ac.uk, School of Engineering, London South Bank University, London SE1 0AA, UK.

This dataset contains 8 columns for each entry that correspond to:

- InvoiceNo: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation.
- StockCode: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.
- Description: Product (item) name. Nominal.
- Quantity: The quantities of each product (item) per transaction. Numeric.
- InvoiceDate: Invoice Date and time. Numeric, the day and time when each transaction was generated.
- UnitPrice: Unit price. Numeric, Product price per unit in sterling.
- CustomerID: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.
- Country: Country name. Nominal, the name of the country where each customer resides.

A snapshot of the first 5 rows in the dataset:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850	United Kingdom

The dataset is imported using pandas `read_csv` method. In the method, I used the `encoding=ISO-8859-1` encoding to properly import the data. At the same time I have converted the data types of ***CustomerID*** and **InvoiceNo** to `str` data type to be on the safe side.
After loading the dataset, I copied the data to `df` so that later if needed I can just run this cell instead of loading the whole data again. To make a deep copy, I used `copy(deep=True)`.
The dataset is huge as it contains more than **541K** transaction data.
Furthermore, I checked the column information and found out that there are some missing values as well as the ***InvoiceDate*** column is in string format.
So, I converted the ***InvoiceDate*** to datetime format using the pandas *to_datetime* function and then dropped the missing values using the CustomerID column with `dropna(axis=0, subset=['CustomerID'], inplace=True)`.
Remove duplicate entries from the dataset. Using the *duplicated* method, I can see which row is a duplicate. To find out how many duplicate rows are present, I am summing the previous result. So, the code is like `df.duplicated().sum()`.
Now, I am deleting the duplicate rows using the `drop_duplicates` method and setting `inplace=True` to permanently modify the original dataframe.

Data Exploration:

First, I explore the **country** column in the dataset.
Here I am trying to know how many transactions happened in each country. As this is a UK-based company, most entries will be from the UK but what about other countries? That is what I am finding.
Many transactions are the same i.e. there are more than or equal to 1 row for each transaction **InvoiceNo** as the data shows different products purchased in each transaction. That is why, I am grouping the unique customers with unique invoice numbers and then taking their countries. I am doing this using the `groupby` method.
The code for this looks like

```
df[['CustomerID', 'InvoiceNo', 'Country']].groupby(['CustomerID',
'InvoiceNo', 'Country']).count()
```

Then, extracting the count of countries in this new dataframe. There are total 37 countries present in the dataset and out of them top 5 countries according to no. of transactions are -

```
United Kingdom    19857
Germany             603
France              458
EIRE                319
Belgium             119
```

Using the countries data I have plotted a map chart that shows the no. of transactions with colours. Red represents more transactions whereas blue represents less number of transactions in those countries.
From the UK only I found around 20K transactions.

Now, let's look at the no. of products, customers and total transactions in the dataset.

```
pd.DataFrame([{
    'product': df['StockCode'].nunique(),
    'customer': df['CustomerID'].nunique(),
    'transaction': df['InvoiceNo'].nunique()
}], columns=['product', 'customer', 'transaction'],
index=['quantity'])
```

	product	customer	transaction
quantity	3684	4372	22190

Next, I am counting the number of products per transaction. To do this first I need to group the dataset based on **CustomerID** and **InvoiceNo** to get unique transactions. Then, I need to count **InvoiceDate** as this denotes the time of purchase for each product.

```
nb_products_per_transaction = df.groupby(['CustomerID',
'InvoiceNo'], as_index=False)['InvoiceDate'].count()

nb_products_per_transaction =
nb_products_per_transaction.rename(columns={'InvoiceDate': 'Number
of Products'})

display(nb_products_per_transaction.sort_values('CustomerID')[:5])
```

	CustomerID	InvoiceNo	Number of Products
0	12346	541431	1
1	12346	C541433	1
2	12347	537626	31
3	12347	542237	29
4	12347	549222	24

From the result, I can see that some customers only purchased 1 gift but some customers (like 12347) have purchased a lot of products and also he/she is regular. Another thing to note here is the **C** letter with **541433** Invoice number which means cancelled.

I am going to check the cancelled transactions, the reason I am not trying to remove the cancelled orders is because an order can be cancelled if there exists a previous order. It means one of the previous orders that is valid is going to be negated by another cancelled order. So, essentially I am trying to remove the amount that was cancelled and calculate the correct sales for each transaction.

First I am checking how many cancelled orders are present.

```
nb_products_per_transaction['Order Cancelled'] =
nb_products_per_transaction['InvoiceNo'].apply(lambda x: int('C'
in x))
total_cancelled_orders = nb_products_per_transaction['Order
Cancelled'].sum()
print(f"Percentage of cancelled orders over total orders:
{total_cancelled_orders / nb_products_per_transaction.shape[0]:
.2f}")
```

```
Percentage of cancelled orders over total orders:  0.16
```

Over 16% of orders were cancelled.

I did another experiment to check the cancelled orders but it is proven that it is not always the case that the cancelled orders will have a counterpart with the same quantity(but negative).

If I do believe the hypothesis, that is easily false by the following code result:

```
df_check = df[df['Quantity'] < 0][['CustomerID','Quantity',

'StockCode','Description','UnitPrice']]
for index, col in  df_check.iterrows():
    if df[(df['CustomerID'] == col[0]) & (df['Quantity'] ==
-col[1])
                & (df['Description'] == col[2])].shape[0] == 0:
        print(df_check.loc[index])
```

```
        print(15*'-'+'>'+' HYPOTHESIS NOT FULFILLED')
        break
```

```
CustomerID        14527
Quantity             -1
StockCode             D
Description    Discount
UnitPrice          27.5
Name: 141, dtype: object
---------------> HYPOTHESIS NOT FULFILLED
```

Even removing the Discount product does not work. We get another product that does not follow the hypothesis.

```
154 CustomerID                              15311
Quantity                                   -1
StockCode                              35004C
Description    SET OF 3 COLOURED  FLYING DUCKS
UnitPrice                                4.65
Name: 154, dtype: object
---------------> HYPOTHESIS NOT FULFILLED
```

Finally I am writing the code to store those indexes that contain cancelled orders and at the same time removing their counterparts if they exist. Those entries that does not have any counterpart most probably are ordered before 2010-12-01 as this is the starting date of the dataset.

```
df_cleaned = df.copy(deep = True)

# To store tha cancelled quantity for the counterpart
df_cleaned['QuantityCanceled'] = 0

# lists to store indices of cancelled orders with counterpart and
without counterpart respectively
entry_to_remove = [] ; doubtfull_entry = []

# Go through each row of the dataframe using iterrows method
for index, col in  df.iterrows():
    # Ignore orders with positive quantity and Discount order
    if (col['Quantity'] > 0) or col['Description'] == 'Discount':
continue
    df_test = df[(df['CustomerID'] == col['CustomerID']) &
                         (df['StockCode']  == col['StockCode']) &
                         (df['InvoiceDate'] < col['InvoiceDate'])
```

```
&
                         (df['Quantity']   > 0)].copy()

    # Cancelation WITHOUT counterpart
    if (df_test.shape[0] == 0):
        doubtfull_entry.append(index)

    # Cancelation WITH a counterpart
    elif (df_test.shape[0] == 1):
        index_order = df_test.index[0]
        df_cleaned.loc[index_order, 'QuantityCanceled'] =
-col['Quantity']
        entry_to_remove.append(index)

    # Various counterparts exist in orders: we delete the last
order that is purchased in >= quantity than the cancelled one
    elif (df_test.shape[0] > 1):
        df_test.sort_index(axis=0, ascending=False, inplace =
True)
        for ind, val in df_test.iterrows():
            if val['Quantity'] < -col['Quantity']: continue
            df_cleaned.loc[ind, 'QuantityCanceled'] =
-col['Quantity']
            entry_to_remove.append(index)
            break
```

For the orders that have more than 1 counterpart, I am sorting those entries in descending order and checking the entry that has quantity more or equal to the cancelled order quantity. If I find that entry I append it to `entry_to_remove` and break from the loop.

This code block will take a lot of time as it is going through all 541K rows, but after this code block is successfully executed I can check the number of `entry_to_remove` and `doubtfull_entry`.

```
print("entry_to_remove: {}".format(len(entry_to_remove)))
print("doubtfull_entry: {}".format(len(doubtfull_entry)))
```

```
entry_to_remove: 7521
doubtfull_entry: 1226
```

Now, I can delete these rows.

```
df_cleaned.drop(entry_to_remove, axis = 0, inplace = True)
df_cleaned.drop(doubtfull_entry, axis = 0, inplace = True)
```

As I filtered the entries with -ve quantities with the condition of whether they have counterparts or not, in the 3rd case where they have more than 1 counterpart there could be entries that did not fulfil the condition. So, I am going to check for entries that were still not dropped even though having -ve quantity and not being product ‘D’. Also some cancelled entries could be completely out of our expectation and does not follow the convention we assumed.

```
remaining_entries = df_cleaned[(df_cleaned['Quantity'] < 0) &
(df_cleaned['StockCode'] != 'D')]
print("nb of entries to delete:
{}".format(remaining_entries.shape[0]))
remaining_entries[:5]
```

```
nb of entries to delete: 48
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	QuantityCanceled
77598	C542742	84535B	FAIRY CAKES NOTEBOOK A6 SIZE	-94	2011-01-31 16:26:00	0.65	15358	United Kingdom	0
90444	C544038	22784	LANTERN CREAM GAZEBO	-4	2011-02-15 11:32:00	4.95	14659	United Kingdom	0
111968	C545852	22464	HANGING METAL HEART LANTERN	-5	2011-03-07 13:49:00	1.65	14048	United Kingdom	0
116064	C546191	47566B	TEA TIME PARTY BUNTING	-35	2011-03-10 10:57:00	0.70	16422	United Kingdom	0
132642	C547675	22263	FELT EGG COSY LADYBIRD	-49	2011-03-24 14:07:00	0.66	17754	United Kingdom	0

So we have 48 entries to remove from the dataset along with the ‘D’ product.

```
df_cleaned = df_cleaned[~df_cleaned['Quantity'] < 0]
```

Next, I am calculating the total price of each product purchased using the **UnitPrice**, **Quantity** and **Cancelled Quantity** columns.

```
df_cleaned['TotalPrice'] = df_cleaned['UnitPrice'] *
(df_cleaned['Quantity'] - df_cleaned['QuantityCanceled'])
df_cleaned.sort_values('CustomerID')[:5]
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	QuantityCanceled	TotalPrice
61619	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	2011-01-18 10:01:00	1.04	12346	United Kingdom	74215	0.0
72256	542237	22728	ALARM CLOCK BAKELIKE PINK	4	2011-01-26 14:30:00	3.75	12347	Iceland	0	15.0
72263	542237	47559B	TEA TIME OVEN GLOVE	10	2011-01-26 14:30:00	1.25	12347	Iceland	0	12.5
72264	542237	21154	RED RETROSPOT OVEN GLOVE	10	2011-01-26 14:30:00	1.25	12347	Iceland	0	12.5
14945	537626	22774	RED DRAWER KNOB ACRYLIC EDWARDIAN	12	2010-12-07 14:57:00	1.25	12347	Iceland	0	15.0

Every basket price is for one product and as I have more than one product in each transaction, they are separated. So in this step, I am combining those rows for calculating the total price of one order.

```
t = df_cleaned.groupby(['CustomerID', 'InvoiceNo'],
as_index=False)['TotalPrice'].sum()
basket_price = t.rename(columns={'TotalPrice': 'BasketPrice'})

df_cleaned['InvoiceDateInt'] =
df_cleaned['InvoiceDate'].astype(np.int64)
t = df_cleaned.groupby(['CustomerID', 'InvoiceNo'],
as_index=False)['InvoiceDateInt'].mean()
basket_price['InvoiceDate'] = t['InvoiceDateInt']
basket_price['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
df_cleaned.drop(['InvoiceDateInt'], axis=1, inplace=True)

display(basket_price[:5])
```

	CustomerID	InvoiceNo	BasketPrice	InvoiceDate
0	12346	541431	0.00	2010-12-01 08:26:00
1	12347	537626	711.79	2010-12-01 08:26:00
2	12347	542237	475.39	2010-12-01 08:26:00
3	12348	539318	892.80	2010-12-01 08:26:00
4	12348	541998	227.44	2010-12-01 08:26:00

After getting the basket price for each order, I can see how the purchases are divided according to basket price. I am creating groups for each basket price range for example - 0-50, 50-100, 100-200, … etc. I have a total of 7 groups with price [0, 50, 100, 200, 500, 1000, 5000, 50000]. I am trying to find out how many orders belong to each group and then plot a pie chart showing the exact division of purchases based on basket price. The price ranges are inclusive of the first element and exclusive of the last element like [0, 50). As the highest basket price in the dataset is less than 20K, we are safe to use 50K as the price limit.

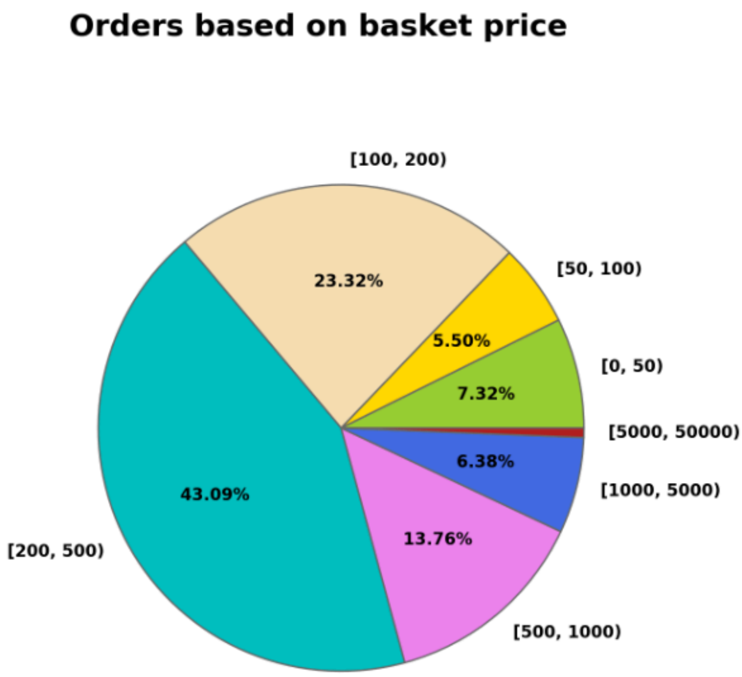
```
prices = [0, 50, 100, 200, 500, 1000, 5000, 50000]
counts = []

for i, price in enumerate(prices):
    if i==0: continue

    t = basket_price[(basket_price['BasketPrice'] >= prices[i-1])
&
                    (basket_price['BasketPrice'] < prices[i])]
    counts.append(t.shape[0])

print(f"Orders in each price range: {counts}\nAll orders are
```

```
counted: \
      {sum(counts)==basket_price.shape[0]}")
```

```
Orders in each price range: [233, 175, 742, 1371, 438, 203, 20]
All orders are counted:       True
```

Finally code to draw the pie chart.

```
font = {'family': 'sans-serif',
        'weight': 'bold'}
plt.rc('font', **font)
f, ax = plt.subplots(figsize=(11, 6), dpi=200)
wedge_colors = ['yellowgreen', 'gold', 'wheat', 'c', 'violet',
'royalblue','firebrick']
wedge_labels = [f"[{prices[i-1]}, {price})" for i, price in
enumerate(prices) if not i==0]
autopct_fn = lambda x: f"{x: .2f}%" if x>1 else ""
ax.pie(counts,
       labels=wedge_labels,
       colors=wedge_colors,
       autopct=autopct_fn)
ax.axis('equal')
f.text(x=0.5, y=1.0, s="Orders based on basket price",
ha='center', fontsize=18)
plt.show()
```



43% purchases were in between 200-500. Relatively most of the purchases are large orders given that around 65% orders are more than 200.

Product Categorization: