fda-project2-1

December 2, 2023

Importing necessary libraries

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

Importing and reading the data

```
[ ]: data = pd.read_csv('/data.csv', encoding='ISO-8859-1')
     DF = pd.read_csv('/data.csv', encoding='ISO-8859-1')
```

Observing the data

```
[ ]: data.head(20)
```

```
[ ]:   InvoiceNo StockCode                          Description  Quantity  \
    0     536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
    1     536365     71053                  WHITE METAL LANTERN         6
    2     536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
    3     536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
    4     536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
    5     536365     22752         SET 7 BABUSHKA NESTING BOXES         2
    6     536365     21730    GLASS STAR FROSTED T-LIGHT HOLDER         6
    7     536366     22633               HAND WARMER UNION JACK         6
    8     536366     22632            HAND WARMER RED POLKA DOT         6
    9     536367     84879        ASSORTED COLOUR BIRD ORNAMENT        32
    10    536367     22745           POPPY'S PLAYHOUSE BEDROOM          6
    11    536367     22748            POPPY'S PLAYHOUSE KITCHEN         6
    12    536367     22749    FELTCRAFT PRINCESS CHARLOTTE DOLL         8
    13    536367     22310              IVORY KNITTED MUG COSY          6
    14    536367     84969   BOX OF 6 ASSORTED COLOUR TEASPOONS         6
    15    536367     22623        BOX OF VINTAGE JIGSAW BLOCKS          3
    16    536367     22622       BOX OF VINTAGE ALPHABET BLOCKS         2
    17    536367     21754             HOME BUILDING BLOCK WORD         3
    18    536367     21755             LOVE BUILDING BLOCK WORD         3
    19    536367     21777          RECIPE BOX WITH METAL HEART         4

           InvoiceDate  UnitPrice  CustomerID         Country
```

```
0   12/1/2010 8:26       2.55     17850.0  United Kingdom
1   12/1/2010 8:26       3.39     17850.0  United Kingdom
2   12/1/2010 8:26       2.75     17850.0  United Kingdom
3   12/1/2010 8:26       3.39     17850.0  United Kingdom
4   12/1/2010 8:26       3.39     17850.0  United Kingdom
5   12/1/2010 8:26       7.65     17850.0  United Kingdom
6   12/1/2010 8:26       4.25     17850.0  United Kingdom
7   12/1/2010 8:28       1.85     17850.0  United Kingdom
8   12/1/2010 8:28       1.85     17850.0  United Kingdom
9   12/1/2010 8:34       1.69     13047.0  United Kingdom
10  12/1/2010 8:34       2.10     13047.0  United Kingdom
11  12/1/2010 8:34       2.10     13047.0  United Kingdom
12  12/1/2010 8:34       3.75     13047.0  United Kingdom
13  12/1/2010 8:34       1.65     13047.0  United Kingdom
14  12/1/2010 8:34       4.25     13047.0  United Kingdom
15  12/1/2010 8:34       4.95     13047.0  United Kingdom
16  12/1/2010 8:34       9.95     13047.0  United Kingdom
17  12/1/2010 8:34       5.95     13047.0  United Kingdom
18  12/1/2010 8:34       5.95     13047.0  United Kingdom
19  12/1/2010 8:34       7.95     13047.0  United Kingdom
```

[ ]:
```
#Data types of various columns
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

[ ]:
```
#Looking at the statistical charecterstics of numeric columns
data[['Quantity','UnitPrice']].describe()
```

[ ]:

	Quantity	UnitPrice
count	541909.000000	541909.000000
mean	9.552250	4.611114
std	218.081158	96.759853

```
min    -80995.000000  -11062.060000
25%         1.000000       1.250000
50%         3.000000       2.080000
75%        10.000000       4.130000
max     80995.000000   38970.000000
```

Data Preprocessing

[ ]:
```
#Handelling missing values
#total number of missing values per column
data.isnull().sum()
```

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

[ ]:
```
#percentage of missing values per column
data.isnull().mean()
```

[ ]:
```
InvoiceNo      0.000000
StockCode      0.000000
Description    0.002683
Quantity       0.000000
InvoiceDate    0.000000
UnitPrice      0.000000
CustomerID     0.000000
Country        0.000000
dtype: float64
```

[ ]:
```
#Dealing with missing customer id's and removing decimal point
#removing decimal point and digits after that
data['CustomerID_new']=data['CustomerID'].astype(str)
data['CustomerID_new']=np.where(data['CustomerID_new'].
 ↪isnull(),'Unavilable',data['CustomerID_new'].str[:-2])
```

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

[ ]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
```

```
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

      InvoiceDate  UnitPrice CustomerID         Country
0  12/1/2010 8:26       2.55      17850  United Kingdom
1  12/1/2010 8:26       3.39      17850  United Kingdom
2  12/1/2010 8:26       2.75      17850  United Kingdom
3  12/1/2010 8:26       3.39      17850  United Kingdom
4  12/1/2010 8:26       3.39      17850  United Kingdom
```

[ ]:
```
#replacing missing customer id's with 'unavailable'
data['CustomerID_new']=np.
 ↪where(data['CustomerID_new']=='n','Unavilable',data['CustomerID_new'])
data['CustomerID']=data['CustomerID_new']
data.drop(['CustomerID_new'],axis=1,inplace=True)
```

[ ]:
```
#sample columns where customer if is unavailable
data[data['CustomerID']=='Unavilable']
```

[ ]:
```
       InvoiceNo StockCode                      Description  Quantity  \
622       536414     22139                              NaN        56
1443      536544     21773  DECORATIVE ROSE BATHROOM BOTTLE         1
1444      536544     21774  DECORATIVE CATS BATHROOM BOTTLE         2
1445      536544     21786               POLKADOT RAIN HAT          4
1446      536544     21787            RAIN PONCHO RETROSPOT         2
…            …        …                              …       …
541536    581498    85099B          JUMBO BAG RED RETROSPOT         5
541537    581498    85099C   JUMBO  BAG BAROQUE BLACK WHITE         4
541538    581498     85150    LADIES & GENTLEMEN METAL SIGN         1
541539    581498     85174                S/4 CACTI CANDLES         1
541540    581498       DOT                   DOTCOM POSTAGE         1

            InvoiceDate  UnitPrice  CustomerID         Country
622     12/1/2010 11:52       0.00  Unavilable  United Kingdom
1443    12/1/2010 14:32       2.51  Unavilable  United Kingdom
1444    12/1/2010 14:32       2.51  Unavilable  United Kingdom
1445    12/1/2010 14:32       0.85  Unavilable  United Kingdom
1446    12/1/2010 14:32       1.66  Unavilable  United Kingdom
…                   …        …           …             …
541536  12/9/2011 10:26       4.13  Unavilable  United Kingdom
541537  12/9/2011 10:26       4.13  Unavilable  United Kingdom
541538  12/9/2011 10:26       4.96  Unavilable  United Kingdom
541539  12/9/2011 10:26      10.79  Unavilable  United Kingdom
541540  12/9/2011 10:26    1714.17  Unavilable  United Kingdom

[135080 rows x 8 columns]
```

[ ]:
```
#Dealing with outliers in quantity and unit price columns
#defining a function which gives us order status wrt order price and order␣
↪quantity
def status_finder(quantity,price):
  if price<0 and quantity <0:
    return('Order cancelled and money given to customer')
  elif price==0 and quantity <0:
    return('Order cancelled and no money charged')
  elif price<0 and quantity >0:
    return('Bad Debt settled')
  elif price==0 and quantity>0:
    return('Order given for free')
  elif price>0 and quantity<0:
    return('Order cancelled')
  elif price>0 and quantity>0:
    return('Order dispatched')
  else:
    pass
```

[ ]:
```
data['Order_Status'] = data.apply(lambda row: status_finder(row['Quantity'],␣
↪row['UnitPrice']), axis=1)
```

[ ]:
```
#different types of order status in our data
data['Order_Status'].value_counts()
```

[ ]:
```
Order dispatched                           530104
Order cancelled                              9288
Order cancelled and no money charged         1336
Order given for free                         1179
Bad Debt settled                                2
Name: Order_Status, dtype: int64
```

[ ]:
```
# Convert InvoiceDate to datetime
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
```

[ ]:
```
#extracting month name, year and time of the day from invoice date
data['Invoice_Month'] = data['InvoiceDate'].dt.strftime('%B')
data['Invoice_Year'] = data['InvoiceDate'].dt.year
data['Invoice_Time'] = data['InvoiceDate'].dt.strftime('%H:%M:%S')
```

[ ]:
```
data.head(10)
```

[ ]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
```

```
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
5    536365     22752         SET 7 BABUSHKA NESTING BOXES         2
6    536365     21730    GLASS STAR FROSTED T-LIGHT HOLDER         6
7    536366     22633               HAND WARMER UNION JACK         6
8    536366     22632            HAND WARMER RED POLKA DOT         6
9    536367     84879        ASSORTED COLOUR BIRD ORNAMENT        32

          InvoiceDate  UnitPrice CustomerID         Country      Order_Status  \
0 2010-12-01 08:26:00       2.55      17850  United Kingdom  Order dispatched
1 2010-12-01 08:26:00       3.39      17850  United Kingdom  Order dispatched
2 2010-12-01 08:26:00       2.75      17850  United Kingdom  Order dispatched
3 2010-12-01 08:26:00       3.39      17850  United Kingdom  Order dispatched
4 2010-12-01 08:26:00       3.39      17850  United Kingdom  Order dispatched
5 2010-12-01 08:26:00       7.65      17850  United Kingdom  Order dispatched
6 2010-12-01 08:26:00       4.25      17850  United Kingdom  Order dispatched
7 2010-12-01 08:28:00       1.85      17850  United Kingdom  Order dispatched
8 2010-12-01 08:28:00       1.85      17850  United Kingdom  Order dispatched
9 2010-12-01 08:34:00       1.69      13047  United Kingdom  Order dispatched

  Invoice_Month  Invoice_Year Invoice_Time
0      December          2010     08:26:00
1      December          2010     08:26:00
2      December          2010     08:26:00
3      December          2010     08:26:00
4      December          2010     08:26:00
5      December          2010     08:26:00
6      December          2010     08:26:00
7      December          2010     08:28:00
8      December          2010     08:28:00
9      December          2010     08:34:00
```

[ ]:
```
#Adding a column Totalcost for Monetary by multiplying total quantity with unit price
data['TotalCost'] = data['Quantity'] * data['UnitPrice']
data['TotalCost']=data['TotalCost'].abs()
```

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

[ ]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice CustomerID         Country      Order_Status  \
0 2010-12-01 08:26:00       2.55      17850  United Kingdom  Order dispatched
```

```
1 2010-12-01 08:26:00       3.39      17850  United Kingdom  Order dispatched
2 2010-12-01 08:26:00       2.75      17850  United Kingdom  Order dispatched
3 2010-12-01 08:26:00       3.39      17850  United Kingdom  Order dispatched
4 2010-12-01 08:26:00       3.39      17850  United Kingdom  Order dispatched

  Invoice_Month  Invoice_Year Invoice_Time  TotalCost
0      December          2010     08:26:00      15.30
1      December          2010     08:26:00      20.34
2      December          2010     08:26:00      22.00
3      December          2010     08:26:00      20.34
4      December          2010     08:26:00      20.34
```

RFM Calculation

[ ]:
```python
#Recency Calculations
from datetime import datetime
#taking todays date
latest_date = datetime(2023, 11, 29)

#grouping the data wrt latest date where order was generated by each customer
df_recency=data.groupby(by='CustomerID', as_index=False)['InvoiceDate'].max()
df_recency.columns = ['CustomerID', 'Latest_Date']

#calculating recency in days by subtracting the two dates
df_recency['Recency_in_days'] = (latest_date - df_recency['Latest_Date']).dt.
↪days
```

[ ]:
```python
df_recency.head()
```

[ ]:
```
   CustomerID         Latest_Date  Recency_in_days
0       12346 2011-01-18 10:17:00             4697
1       12347 2011-12-07 15:52:00             4374
2       12348 2011-09-25 13:13:00             4447
3       12349 2011-11-21 09:51:00             4390
4       12350 2011-02-02 16:01:00             4682
```

[ ]:
```python
df_recency.drop(['Latest_Date'],axis=1,inplace=True)
```

[ ]:
```python
#Frequency Calculations

#grouping data by number of invoices per customer
df_frequency=data.groupby(by='CustomerID', as_index=False)['InvoiceNo'].count()

df_frequency.columns = ['CustomerID', 'Frequency']
```

[ ]:
```python
df_frequency.head()
```

[ ]:

```
  CustomerID  Frequency
0      12346          2
1      12347        182
2      12348         31
3      12349         73
4      12350         17
```

[ ]:

```
#Monetory Calculations

#grouping data wrt total money spend by each customer
df_monetory=data.groupby(by='CustomerID', as_index=False)['TotalCost'].sum()

df_monetory.columns = ['CustomerID', 'total_expense']
```

[ ]:

```
df_monetory.head()
```

[ ]:

```
  CustomerID  total_expense
0      12346      154367.20
1      12347        4310.00
2      12348        1797.24
3      12349        1757.55
4      12350         334.40
```

[ ]:

```
print(df_monetory.shape)
print(df_frequency.shape)
print(df_recency.shape)
```

```
(4373, 2)
(4373, 2)
(4373, 2)
```

[ ]:

```
#merging the above 3 df using left join
merged_df = pd.merge(df_recency, df_monetory, on='CustomerID', how='left')
merged_df = pd.merge(merged_df, df_frequency, on='CustomerID', how='left')
```

[ ]:

```
merged_df.head(10)
```

[ ]:

```
  CustomerID  Recency_in_days  total_expense  Frequency
0      12346             4697      154367.20          2
1      12347             4374        4310.00        182
2      12348             4447        1797.24         31
3      12349             4390        1757.55         73
4      12350             4682         334.40         17
5      12352             4408        3466.67         95
6      12353             4576          89.00          4
7      12354             4604        1079.40         58
8      12355             4586         459.40         13
```

```
9      12356              4394         2811.43          59
```

[ ]:
```
merged_df.shape
```

[ ]:
```
(4373, 4)
```

RFM Segmentation

[ ]:
```
#Providing a score from 1 to 4 for each RFM metric respectively as suggested
merged_df['Recency_Score'] = pd.qcut(merged_df['Recency_in_days'], 4,␣
↪labels=[4, 3, 2, 1])
merged_df['Frequency_Score'] = pd.qcut(merged_df['Frequency'].
↪rank(method='first'), 4, labels=[1, 2, 3, 4])
merged_df['Monetary_Score'] = pd.qcut(merged_df['total_expense'], 4, labels=[1,␣
↪2, 3, 4])
```

[ ]:
```
merged_df.head(5)
```

[ ]:
```
   CustomerID  Recency_in_days  total_expense  Frequency  Recency_Score  \
0       12346             4697      154367.20          2              1
1       12347             4374        4310.00        182              4
2       12348             4447        1797.24         31              2
3       12349             4390        1757.55         73              3
4       12350             4682         334.40         17              1

  Frequency_Score Monetary_Score
0               1              4
1               4              4
2               2              4
3               3              4
4               1              2
```

[ ]:
```
#generating unified RMF score
merged_df['RFM_Score']=merged_df['Recency_Score'].
↪astype(str)+merged_df['Frequency_Score'].
↪astype(str)+merged_df['Monetary_Score'].astype(str)
```

[ ]:
```
merged_df.head()
```

[ ]:
```
   CustomerID  Recency_in_days  total_expense  Frequency  Recency_Score  \
0       12346             4697      154367.20          2              1
1       12347             4374        4310.00        182              4
2       12348             4447        1797.24         31              2
3       12349             4390        1757.55         73              3
4       12350             4682         334.40         17              1

  Frequency_Score Monetary_Score  RFM_Score
```

```
0              1              4         114
1              4              4         444
2              2              4         224
3              3              4         334
4              1              2         112
```

[ ]:
```
#converting rfm score into integer
merged_df['RFM_Score']=merged_df['RFM_Score'].astype(int)
```

[ ]:
```
scored_df=merged_df[['CustomerID','Recency_Score','Frequency_Score','Monetary_Score','RFM_Scor
```

[ ]:
```
scored_df.head()
```

[ ]:
```
  CustomerID Recency_Score Frequency_Score Monetary_Score  RFM_Score
0      12346             1               1              4        114
1      12347             4               4              4        444
2      12348             2               2              4        224
3      12349             3               3              4        334
4      12350             1               1              2        112
```

Customer Segmentation

[ ]:
```
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.cluster import DBSCAN
from scipy.cluster.hierarchy import dendrogram, linkage
```

[ ]:
```
# Standardizing the data so that model can easily process them
df_rmf=scored_df[['Recency_Score','Frequency_Score','Monetary_Score']]
scaler = StandardScaler()
scaled_rfm_data = scaler.fit_transform(df_rmf)

# getting clusters based on 5 popular clustering methods (more detail in␣
 ↪documentation)
# defining a function for getting the clusters basis each of these 5 techniques␣
 ↪with default hyperparameters for now
def perform_clustering(method, data, n_clusters=None):
    if method == 'kmeans':
        model = KMeans(n_clusters=n_clusters, random_state=42)
        labels = model.fit_predict(data)
    elif method == 'hierarchical':
        model = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
        labels = model.fit_predict(data)
    elif method == 'dbscan':
        model = DBSCAN(eps=0.5, min_samples=5)
```

```
        labels = model.fit_predict(data)
    elif method == 'agglomerative':
        model = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
        labels = model.fit_predict(data)
    elif method == 'gmm':
        model = GaussianMixture(n_components=n_clusters, random_state=42)
        labels = model.fit_predict(data)
    else:
        raise ValueError("Invalid clustering method")

    return labels
```

[ ]:

```
#calling the 5 functions and saving the output as a column in our dataframe
scored_df['KMeans_Segment'] = perform_clustering('kmeans', scaled_rfm_data,␣
 ↪n_clusters=5)
scored_df['Hierarchical_Segment'] = perform_clustering('hierarchical',␣
 ↪scaled_rfm_data, n_clusters=5)
scored_df['DBSCAN_Segment'] = perform_clustering('dbscan', scaled_rfm_data)
scored_df['Agglomerative_Segment'] = perform_clustering('agglomerative',␣
 ↪scaled_rfm_data, n_clusters=5)
scored_df['GMM_Segment'] = perform_clustering('gmm', scaled_rfm_data,␣
 ↪n_clusters=5)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
<ipython-input-88-5c54d21eb709>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  scored_df['KMeans_Segment'] = perform_clustering('kmeans', scaled_rfm_data,
n_clusters=5)
<ipython-input-88-5c54d21eb709>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  scored_df['Hierarchical_Segment'] = perform_clustering('hierarchical',
scaled_rfm_data, n_clusters=5)
```

[ ]:

```
scored_df.head()
```

```
[ ]:   CustomerID  Recency_Score  Frequency_Score  Monetary_Score  RFM_Score  \
0       12346              1                1               4        114
1       12347              4                4               4        444
2       12348              2                2               4        224
3       12349              3                3               4        334
4       12350              1                1               2        112

   KMeans_Segment  Hierarchical_Segment  DBSCAN_Segment  \
0               3                     0               0
1               0                     1               1
2               3                     0               2
3               0                     1               3
4               1                     0               4

   Agglomerative_Segment  GMM_Segment
0                      0            2
1                      1            3
2                      0            2
3                      1            1
4                      0            0
```

```
[ ]: data['Order_Status'].value_counts()
```

```
[ ]: Order dispatched                          530104
     Order cancelled                             9288
     Order cancelled and no money charged        1336
     Order given for free                        1179
     Bad Debt settled                               2
     Name: Order_Status, dtype: int64
```

```
[ ]: #grouping the data basis number of orders generated by each customer
     data['total_order']=np.where(data['Order_Status']=='Order dispatched',1,0)
     df_new=data.groupby(by='CustomerID', as_index=False)['total_order'].sum()
```

```
[ ]: #left joining it with our original dataframe
     scored_df = pd.merge(scored_df, df_new, on='CustomerID', how='left')
```

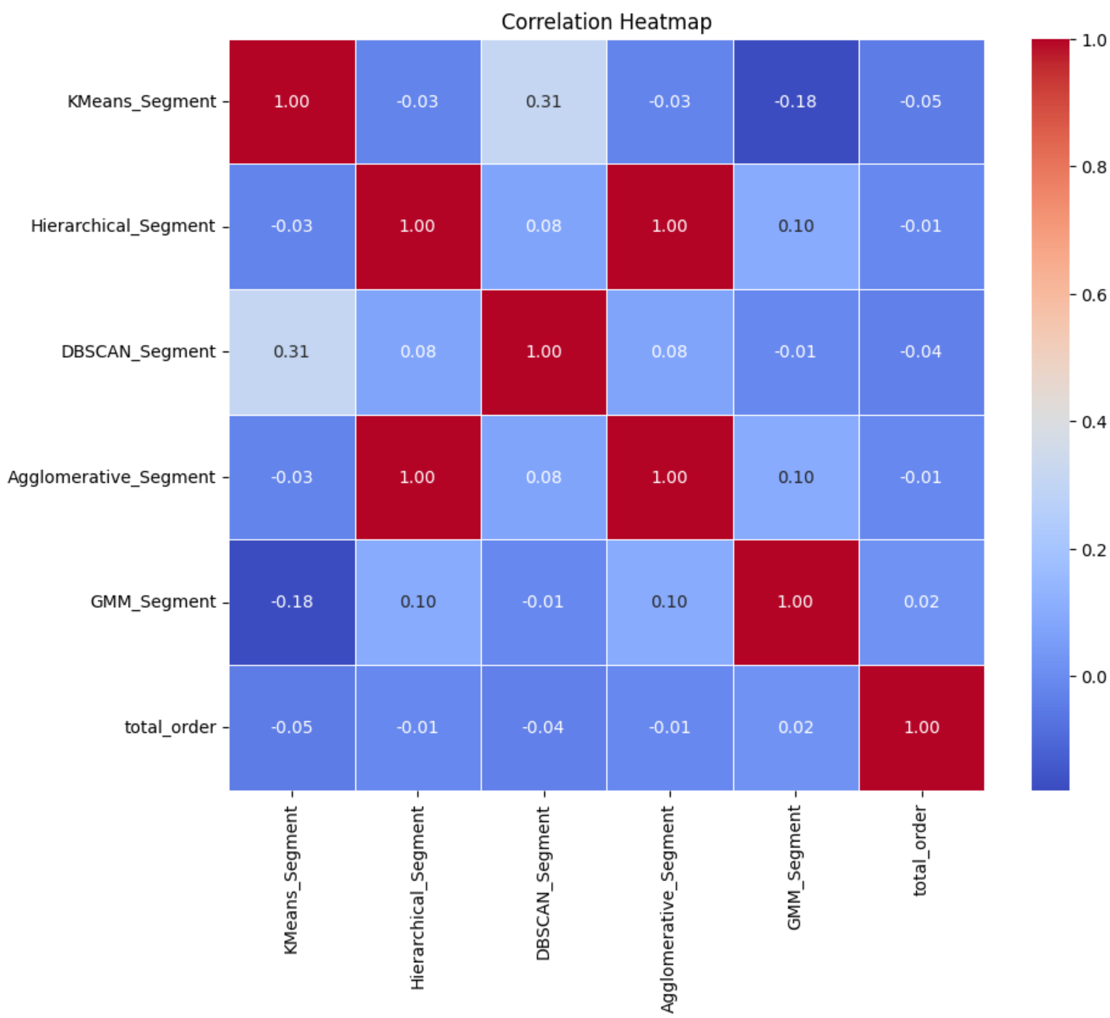
```
[ ]: scored_df.head()
```

```
[ ]:   CustomerID  Recency_Score  Frequency_Score  Monetary_Score  RFM_Score  \
0       12346              1                1               4        114
1       12347              4                4               4        444
2       12348              2                2               4        224
3       12349              3                3               4        334
4       12350              1                1               2        112

   KMeans_Segment  Hierarchical_Segment  DBSCAN_Segment  \
```

```
0                3                      0                0
1                0                      1                1
2                3                      0                2
3                0                      1                3
4                1                      0                4

   Agglomerative_Segment  GMM_Segment  total_order
0                      0            2            1
1                      1            3          182
2                      0            2           31
3                      1            1           73
4                      0            0           17
```

understanding the relationship between the segments generated by the 5 clusturing algorithms and likelyhood of a customer ordering a product

```
[ ]: #correlation between total number of orders generated by each customer and the␣
     ↪sengments generated by the clusturing algorithms.
     columns_to_correlate = ['KMeans_Segment', 'Hierarchical_Segment',␣
     ↪'DBSCAN_Segment', 'Agglomerative_Segment', 'GMM_Segment', 'total_order']
     correlation_data = scored_df[columns_to_correlate]

     correlation_matrix = correlation_data.corr()

     plt.figure(figsize=(10, 8))
     sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
     ↪linewidths=0.5)
     plt.title('Correlation Heatmap')
     plt.show()
```



We observe that standard k means clusturing algorithm shows highest negative correlation, implying that the as the segment number increases the likelyhood of a customer generating an order decreases, which is exactly what we are looking for, hence we will be going ahead with a standard k means clusturing algorthm

[ ]:
```
final_segmented_df=scored_df[['CustomerID','Recency_Score','Frequency_Score','Monetary_Score',
```

Segment Profiling

[ ]:
```
final_segmented_df['KMeans_Segment'].value_counts()
```

[ ]:
```
1     1381
0      995
3      768
2      633
4      596
```

```
Name: KMeans_Segment, dtype: int64
```

[ ]:
```
df_segment_profiling=pd.
 ↪merge(final_segmented_df[['CustomerID','RFM_Score','KMeans_Segment','total_order']],merged_
```

[ ]:
```
df_segment_profiling.head(5)
```

[ ]:
```
   CustomerID  RFM_Score  KMeans_Segment  total_order  Recency_in_days  \
0       12346        114               3            1             4697
1       12347        444               0          182             4374
2       12348        224               3           31             4447
3       12349        334               0           73             4390
4       12350        112               1           17             4682

   Frequency  total_expense
0          2      154367.20
1        182        4310.00
2         31        1797.24
3         73        1757.55
4         17         334.40
```

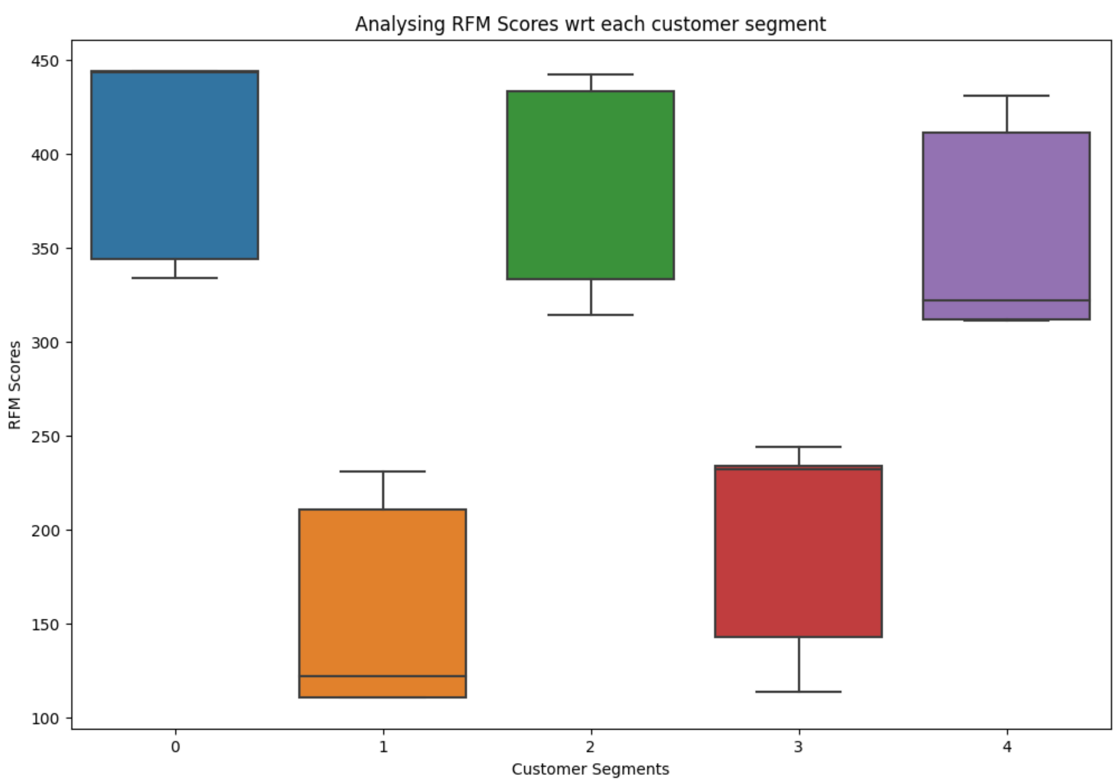
[ ]:
```
all_segments=[0,1,2,3,4]
```

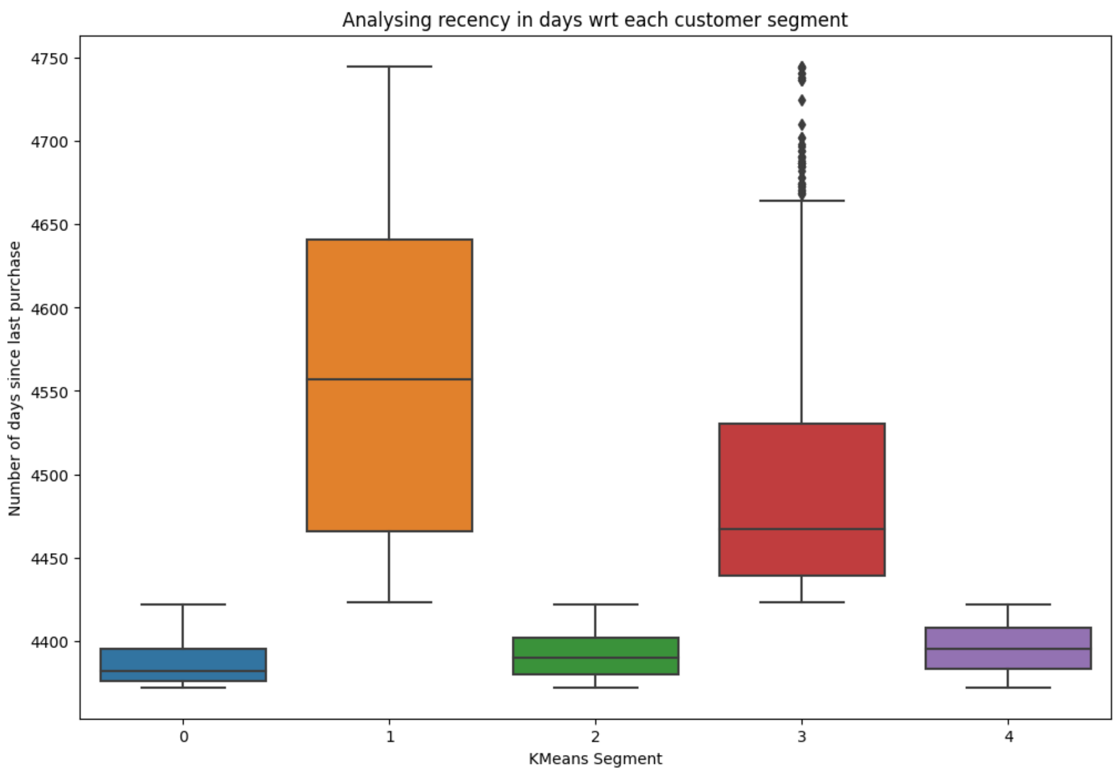
[ ]:
```
#printing the statistical characterstics like mean, std etc for rfm score,␣
 ↪total number of order, number of days elapsed since last order
#, frequency of order and total money spend by customers belonging to each␣
 ↪segment.
for i in all_segments:
  print('Describing the statistical characterestics of customers belonging to␣
 ↪'+str(i)+' segment:')
 ␣
 ↪print(df_segment_profiling[['RFM_Score','total_order','Recency_in_days','Frequency','total_
 ↪describe())
```

```
Describing the statistical characterestics of customers belonging to 0 segment:
        RFM_Score    total_order  Recency_in_days      Frequency  \
count  995.000000     995.000000       995.000000     995.000000
mean   404.704523     395.074372      4386.907538     403.982915
std     48.845475    4205.297223        12.958279    4296.128487
min    334.000000      41.000000      4372.000000      42.000000
25%    344.000000     115.000000      4376.000000     118.000000
50%    443.000000     171.000000      4382.000000     174.000000
75%    444.000000     285.500000      4395.000000     292.000000
max    444.000000  132220.000000      4422.000000  135080.000000

       total_expense
count   9.950000e+02
```

```
mean    8.418169e+03
std     6.753396e+04
min     6.831300e+02
25%     1.901675e+03
50%     2.847950e+03
75%     5.053720e+03
max     2.062871e+06
Describing the statistical characterestics of customers belonging to 1 segment:
         RFM_Score  total_order  Recency_in_days    Frequency  total_expense
count  1381.000000  1381.000000      1381.000000  1381.000000    1381.000000
mean    154.281680    17.199855      4562.779146    17.614048     316.588799
std      49.242843    13.506382        98.795513    13.479380     225.462742
min     111.000000     0.000000      4423.000000     1.000000       1.250000
25%     111.000000     7.000000      4466.000000     8.000000     156.580000
50%     122.000000    14.000000      4557.000000    15.000000     275.280000
75%     211.000000    24.000000      4641.000000    24.000000     416.860000
max     231.000000    88.000000      4745.000000    88.000000    1659.750000
Describing the statistical characterestics of customers belonging to 2 segment:
        RFM_Score  total_order  Recency_in_days   Frequency  total_expense
count  633.000000   633.000000       633.000000  633.000000     633.000000
mean   375.022117    61.200632      4392.238547   62.447077    1639.089828
std     49.748508    30.299808        13.970639   30.086912   13386.390703
min    314.000000     1.000000      4372.000000    1.000000     272.440000
25%    333.000000    42.000000      4380.000000   43.000000     675.270000
50%    333.000000    57.000000      4390.000000   58.000000     942.340000
75%    433.000000    78.000000      4402.000000   79.000000    1290.950000
max    442.000000   218.000000      4422.000000  218.000000  336942.100000
Describing the statistical characterestics of customers belonging to 3 segment:
        RFM_Score  total_order  Recency_in_days   Frequency  total_expense
count  768.000000   768.000000       768.000000  768.000000     768.000000
mean   203.265625    80.697917      4495.700521   82.453125    1987.783022
std     47.082382    61.533213        75.218527   62.263375    6851.576237
min    114.000000     0.000000      4423.000000    1.000000     309.360000
25%    143.000000    42.000000      4439.000000   43.000000     765.497500
50%    232.000000    62.000000      4467.000000   63.000000    1118.900000
75%    234.000000   101.000000      4530.250000  103.000000    1878.425000
max    244.000000   543.000000      4745.000000  548.000000  154367.200000
Describing the statistical characterestics of customers belonging to 4 segment:
        RFM_Score  total_order  Recency_in_days   Frequency  total_expense
count  596.000000   596.000000       596.000000  596.000000     596.000000
mean   351.006711    21.033557      4395.961409   21.422819     349.282651
std     47.573914    14.839156        14.112043   14.866094     233.241689
min    311.000000     0.000000      4372.000000    1.000000       0.000000
25%    312.000000    10.000000      4383.000000   11.000000     186.175000
50%    322.000000    18.000000      4395.000000   19.000000     297.705000
75%    411.000000    29.000000      4408.000000   29.000000     467.970000
max    431.000000    97.000000      4422.000000  101.000000    1692.270000
```

[ ]:
```
#Analysing RFM Score
plt.figure(figsize=(12, 8))
sns.boxplot(x='KMeans_Segment', y='RFM_Score', data=df_segment_profiling)
plt.title('Analysing RFM Scores wrt each customer segment')
plt.xlabel('Customer Segments')
plt.ylabel('RFM Scores')
plt.show()

#Analysing Recency in days
plt.figure(figsize=(12, 8))
sns.boxplot(x='KMeans_Segment', y='Recency_in_days', data=df_segment_profiling)
plt.title('Analysing recency in days wrt each customer segment')
plt.xlabel('KMeans Segment')
plt.ylabel('Number of days since last purchase')
plt.show()
```





Observations: 1. Customers in the 0th segment are the best customers with the highest RFM scores. They perform very well in all metrics except recency, however they have an exceptionally high spending tendency, also, they were frequent buyers who have stopped purchasing due to some reason (will explore in next section)

2. Customers in the 1st segment are the weakest customers. Their average in all metrics is lowest except recency, indicating that they are part of the recent customer wave who isin't spendig a lot.

3. Customers in 2nd and 3rd segment are the best customers in terms of 'increasing sales' due to their fairly decent performance accross all metrics. If suitable steps are taken, they could lead to massive increase in profits

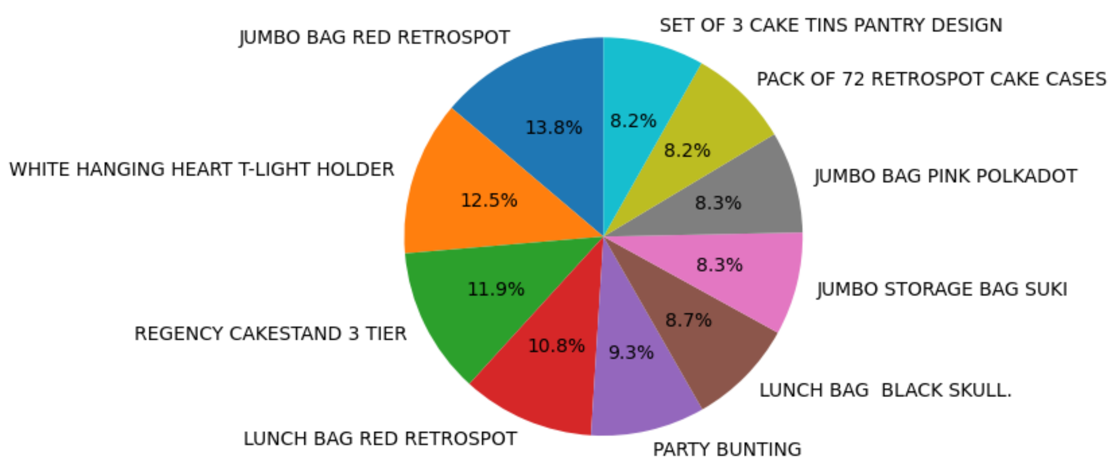
4. Finally, we have 4th segment customers. They are a liability according to me due to them not performing exceptionally well in any metric and will not make or break the company revenue.

Marketing Recomendations

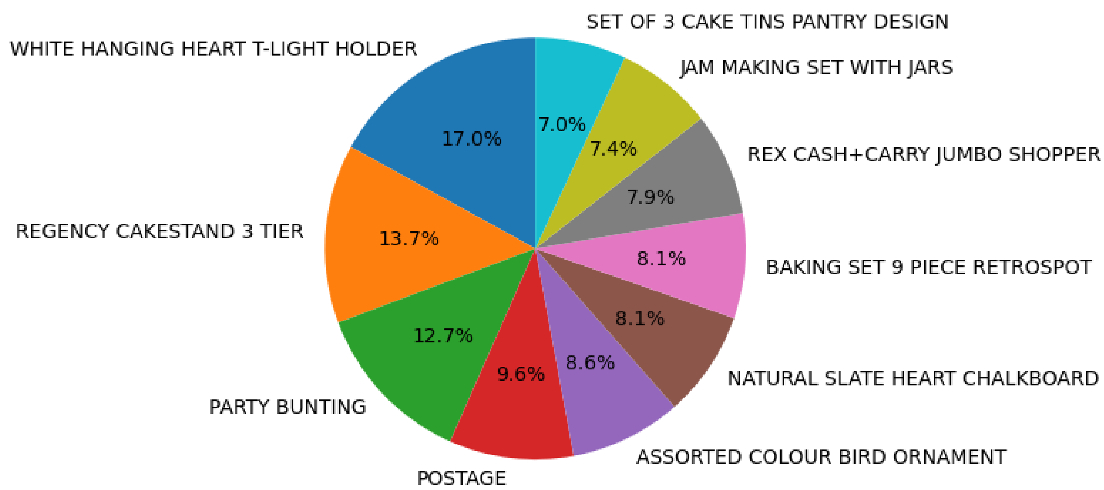
[ ]:
```
#pie chart for various orders made by customers in all the segments
cluster_list=[0,1,2,3,4]
labels=[]
sizes=[]
for i in cluster_list:
  customer_list=[]
```

```
␣
↪customer_list=df_segment_profiling['CustomerID'][df_segment_profiling['KMeans_Segment']==i]
↪to_list()
␣
↪df_temp=data[['CustomerID','Description','Country','Order_Status','Invoice_Month','Invoice_
↪isin(customer_list)]
 labels=list(df_temp['Description'].value_counts().head(10).index)
 sizes=df_temp['Description'].value_counts().head(10).to_list()
 print("Top products choosen by customers in "+str(i)+" :segment")
 plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
 plt.show()
```

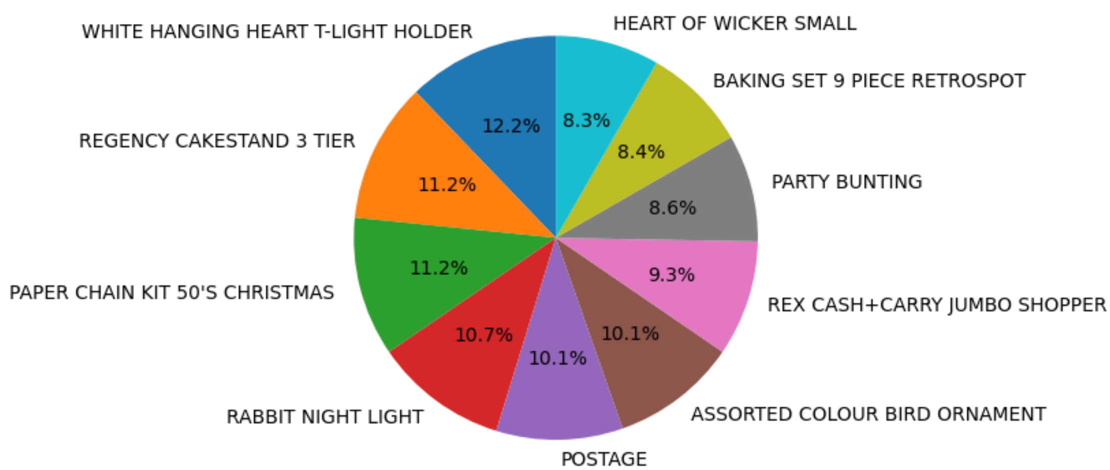
```
Top products choosen by customers in 0 :segment
```



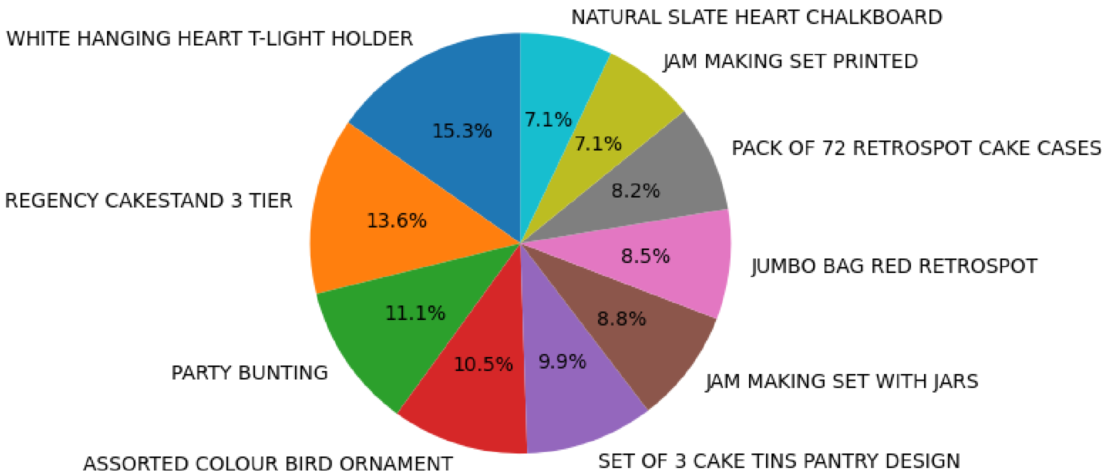

```
Top products choosen by customers in 1 :segment
```



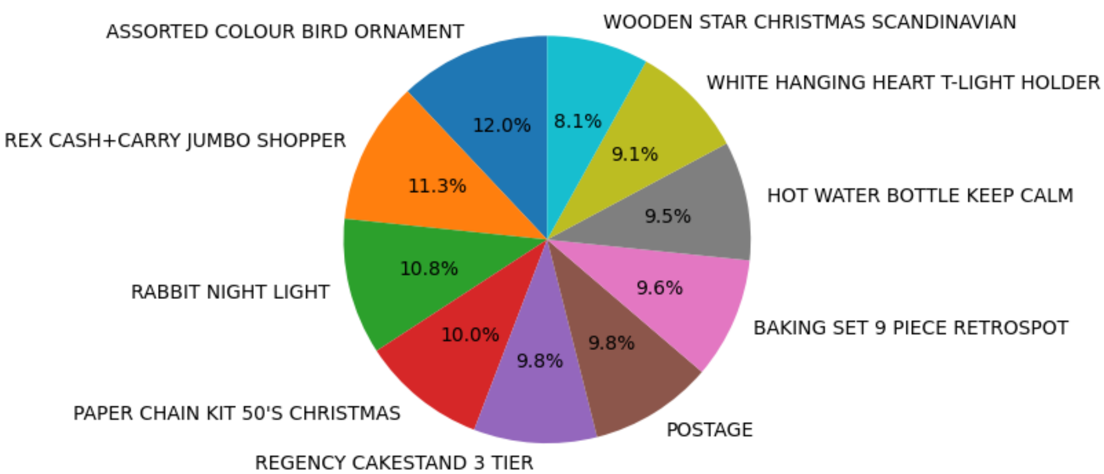
Top products choosen by customers in 2 :segment



Top products choosen by customers in 3 :segment



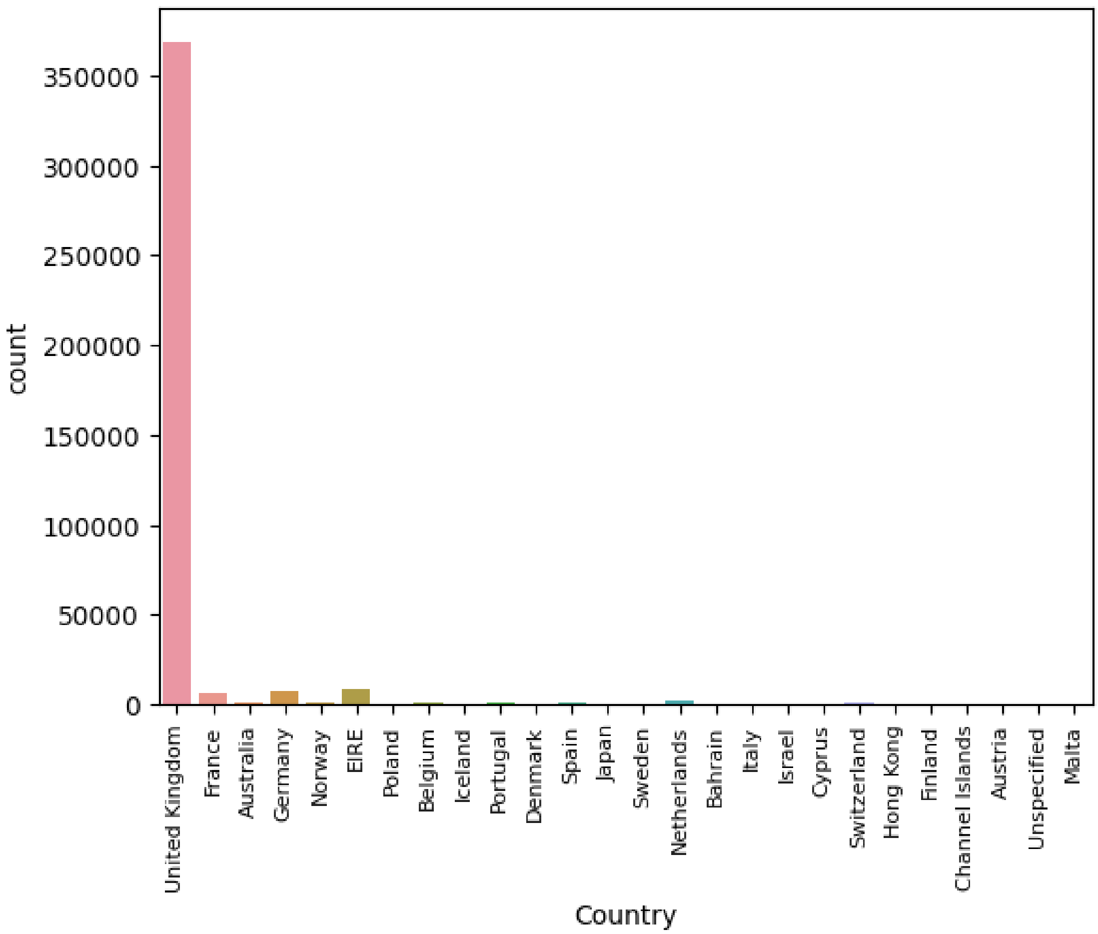

```
Top products choosen by customers in 4 :segment
```



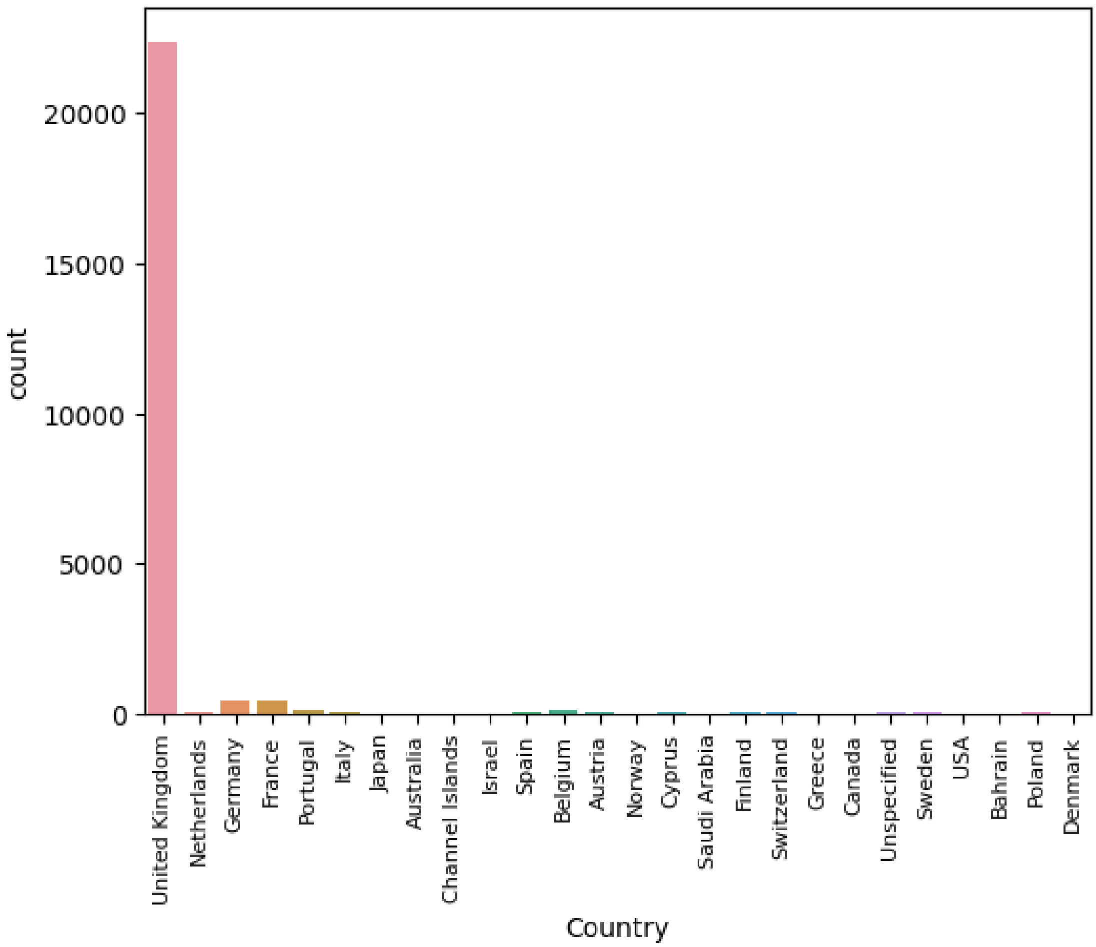
[ ]:
```
#countplots showing top countries the customers belong to
cluster_list=[0,1,2,3,4]
labels=[]
sizes=[]
for i in cluster_list:
  customer_list=[]
 ␣
 ↪customer_list=df_segment_profiling['CustomerID'][df_segment_profiling['KMeans_Segment']==i]
 ↪to_list()
```

```
␣
↪df_temp=data[['CustomerID','Description','Country','Order_Status','Invoice_Month','Invoice_
↪isin(customer_list)]
 print("Top countries with customers in "+str(i)+" :segment")
 sns.countplot(x='Country', data=df_temp)
 plt.xticks(rotation=90, fontsize=8)
 plt.show()
```

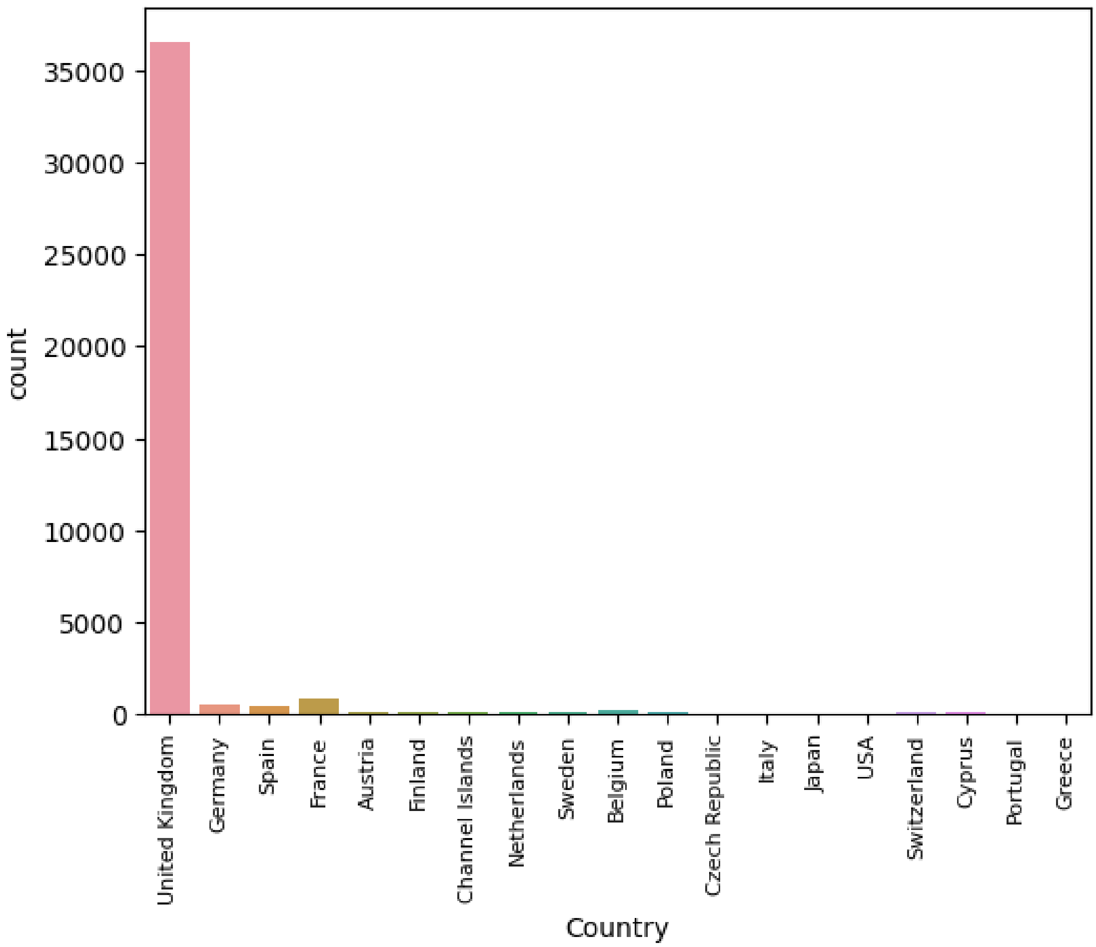
Top countries with customers in 0 :segment



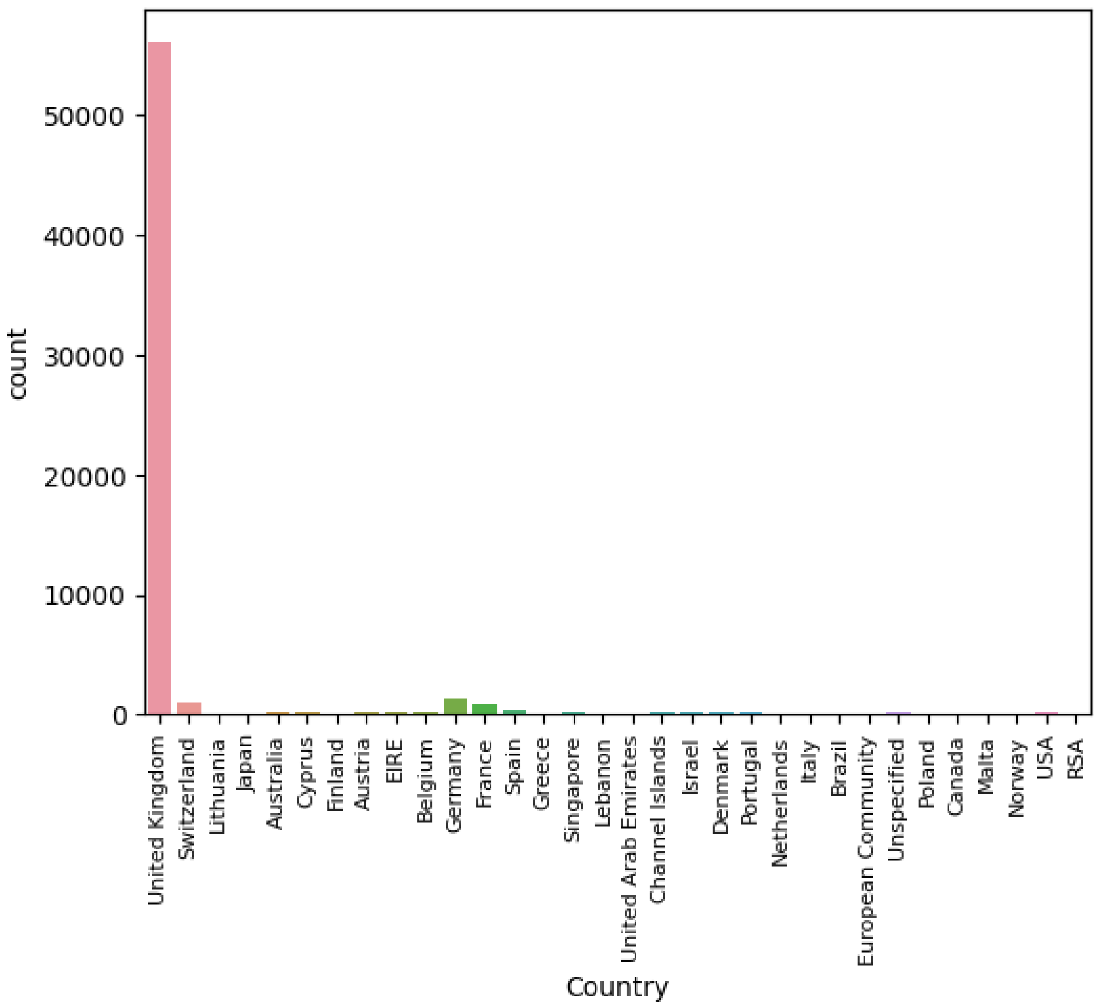
Top countries with customers in 1 :segment



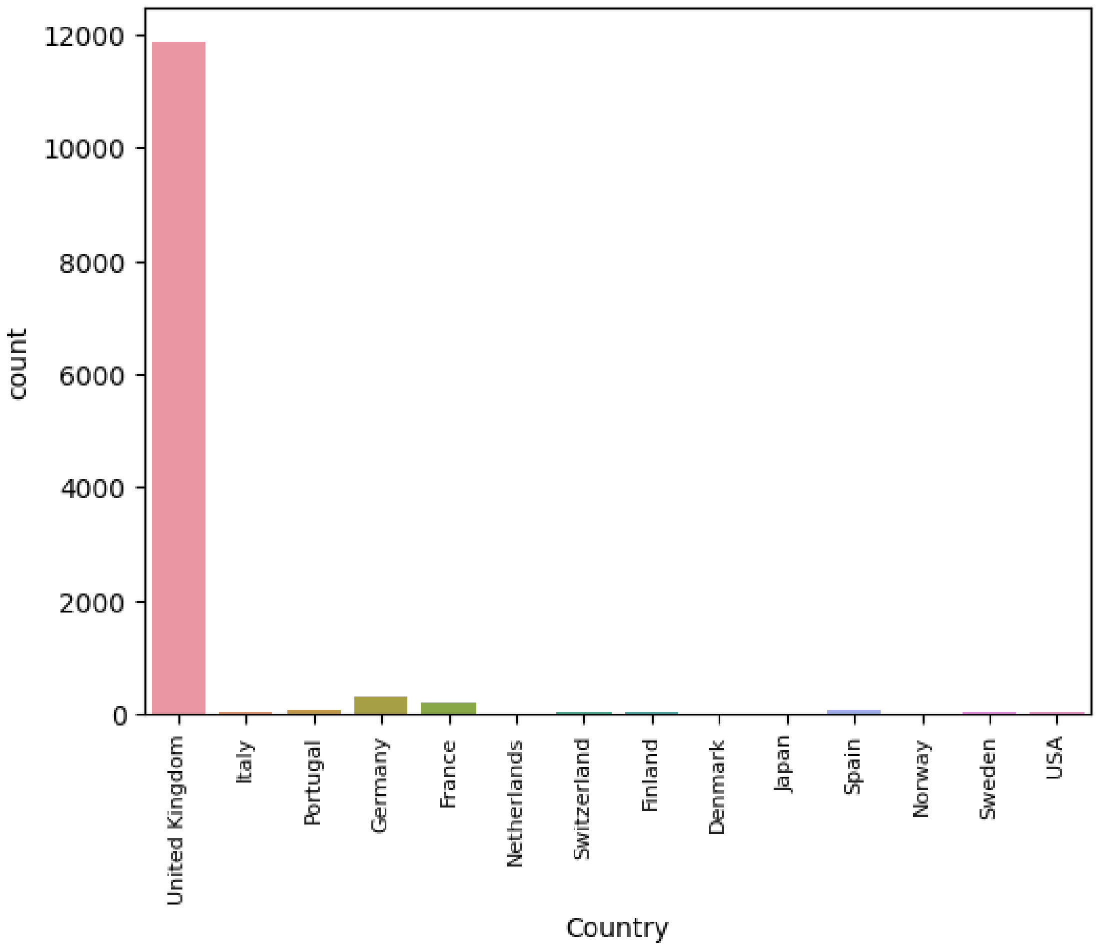
Top countries with customers in 2 :segment




```
Top countries with customers in 3 :segment
```



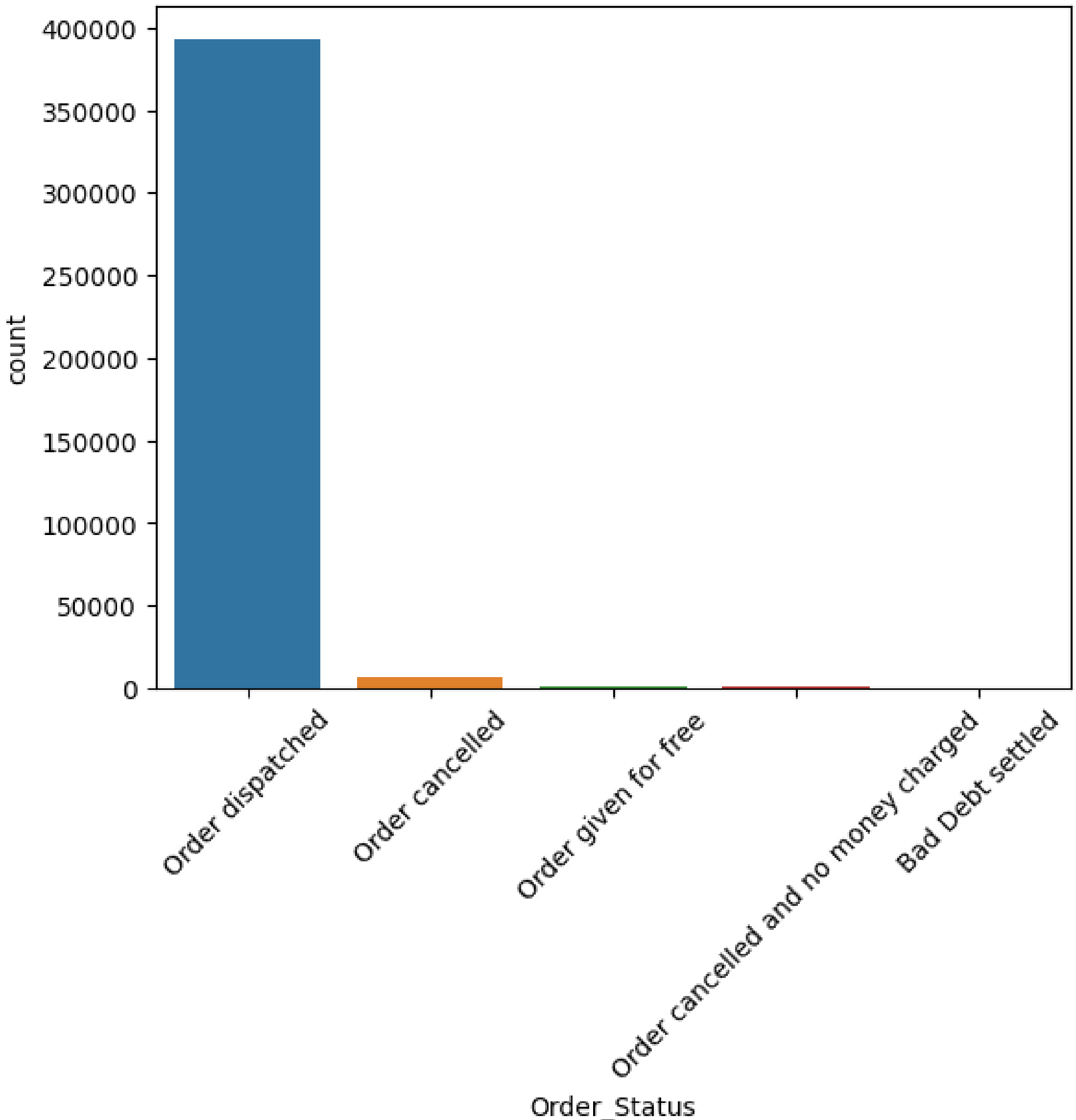

```
Top countries with customers in 4 :segment
```



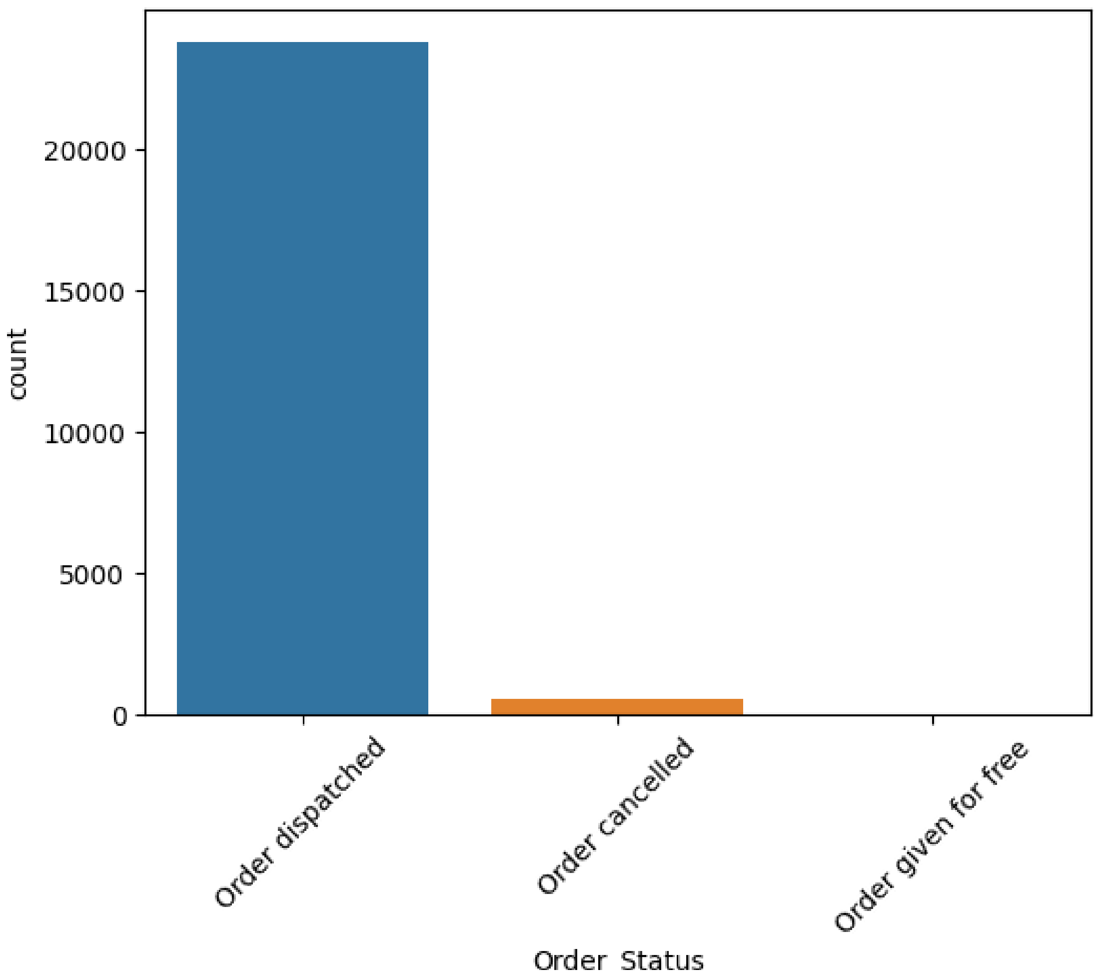
[ ]:

```
#countplots showing order status of customers in each segment
cluster_list=[0,1,2,3,4]
labels=[]
sizes=[]
for i in cluster_list:
  customer_list=[]
 ␣
 ↪customer_list=df_segment_profiling['CustomerID'][df_segment_profiling['KMeans_Segment']==i]
 ↪to_list()
 ␣
 ↪df_temp=data[['CustomerID','Description','Country','Order_Status','Invoice_Month','Invoice_
 ↪isin(customer_list)]
  print("Order status of customers in "+str(i)+" :segment")
  sns.countplot(x='Order_Status', data=df_temp)
  plt.xticks(rotation=45, fontsize=10)
  plt.show()
  plt.show()
```

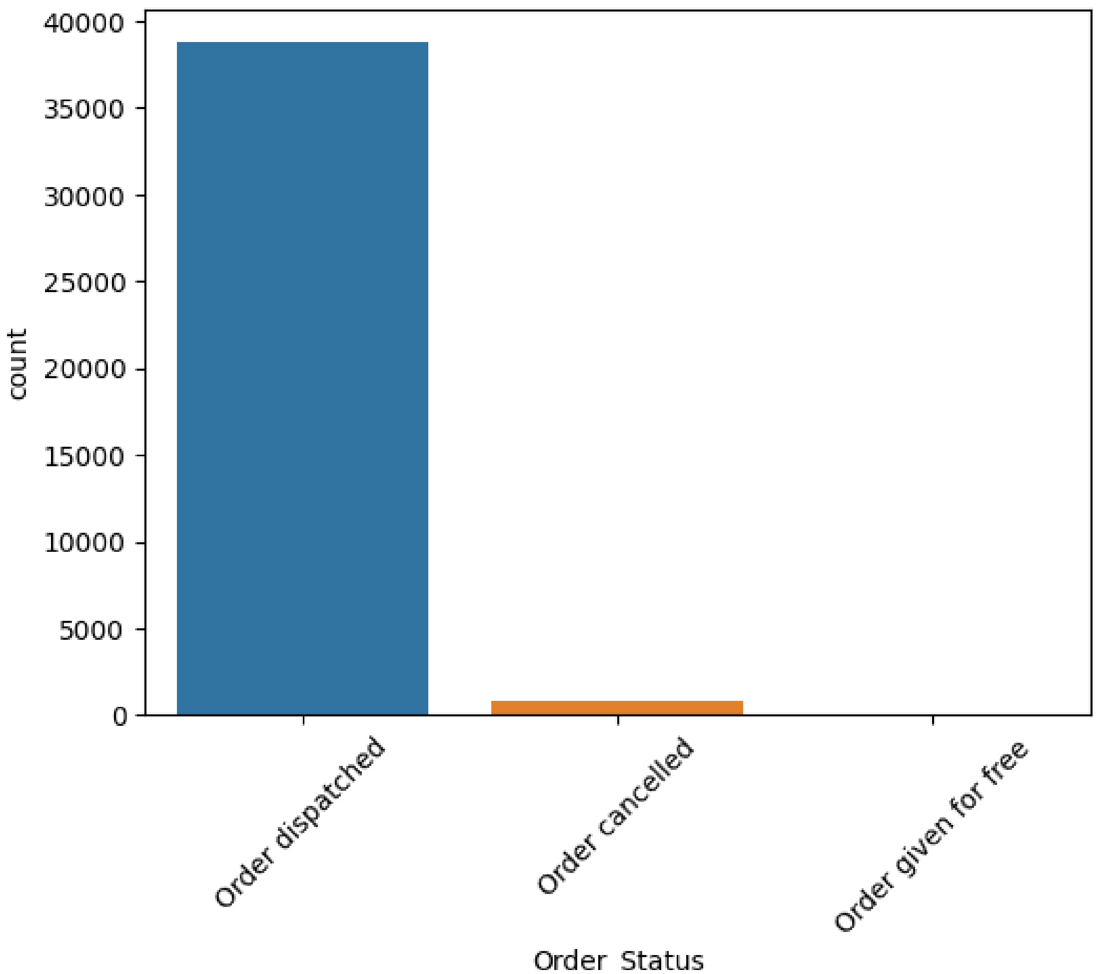
```
Order status of customers in 0 :segment
```



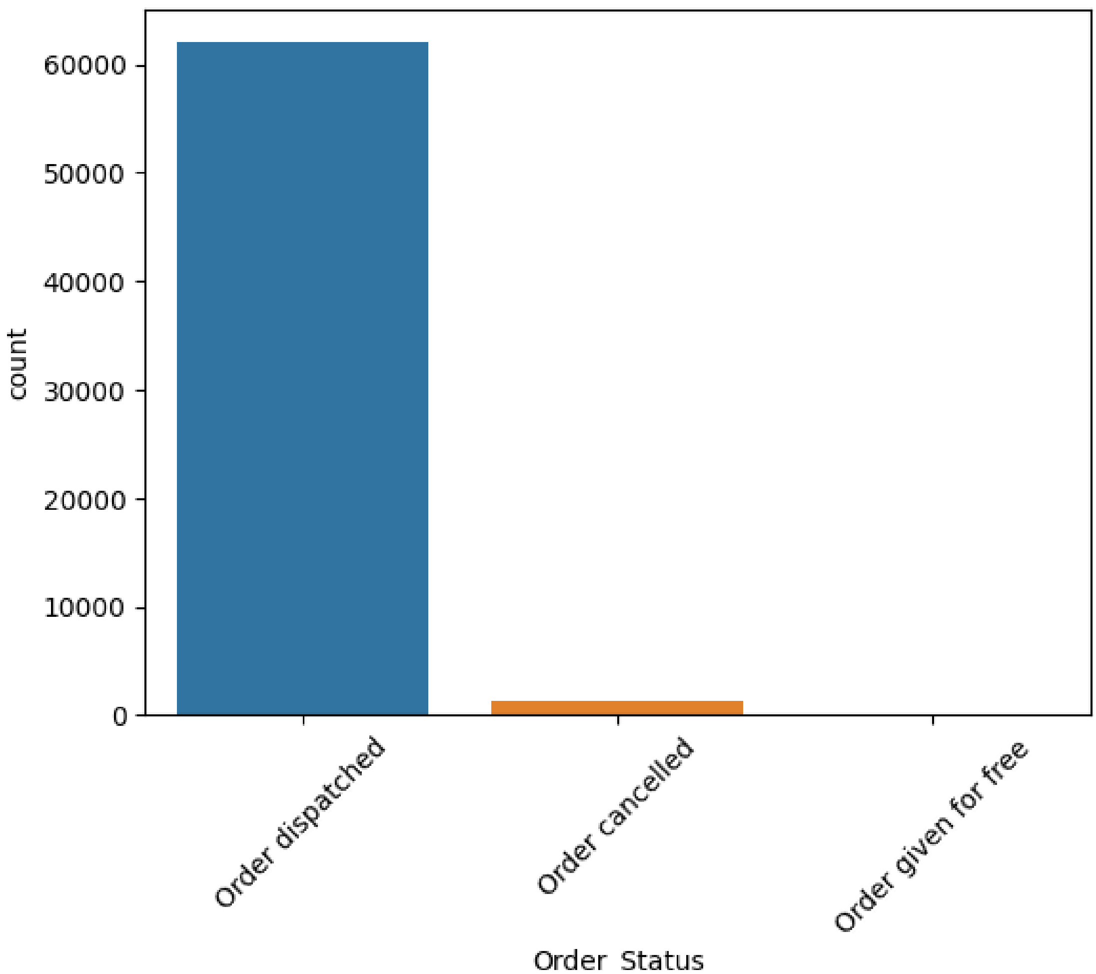

```
Order status of customers in 1 :segment
```



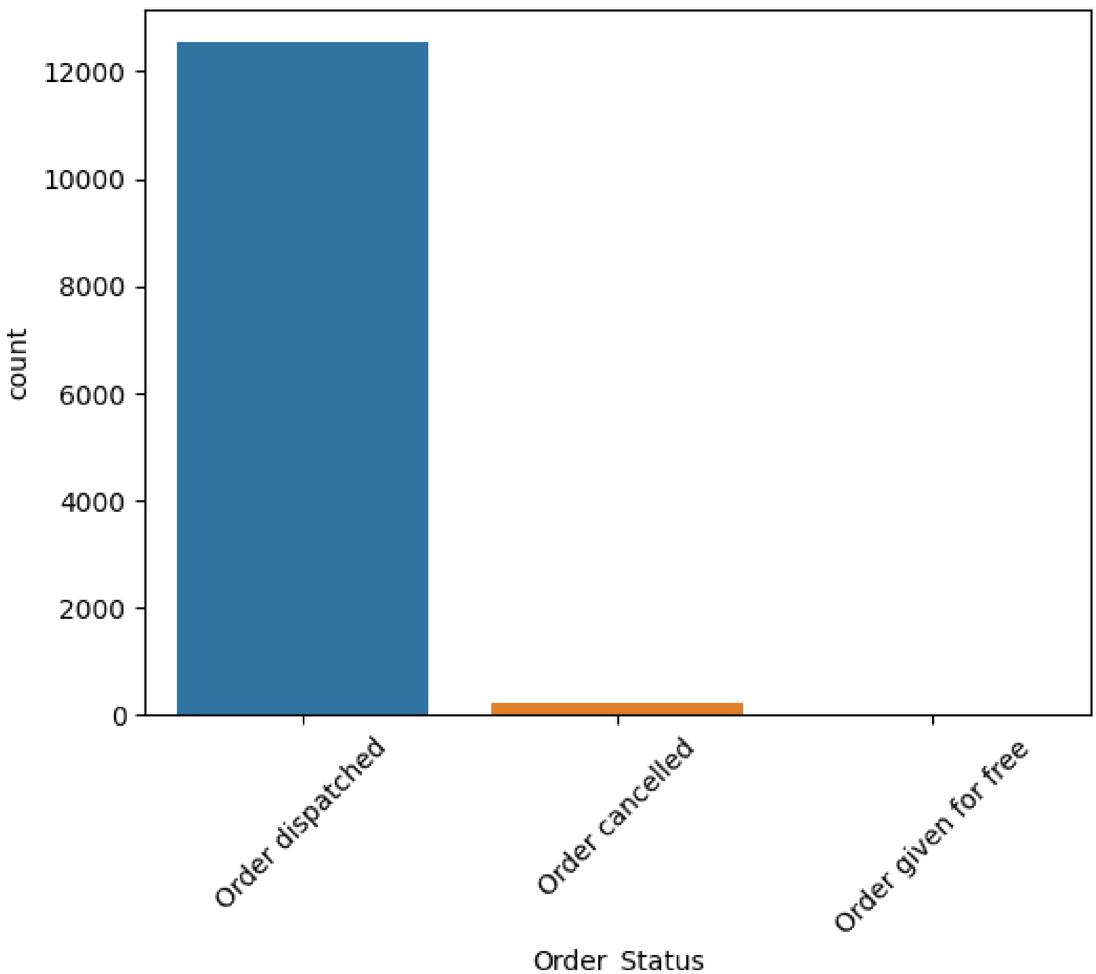
Order status of customers in 2 :segment




```
Order status of customers in 3 :segment
```



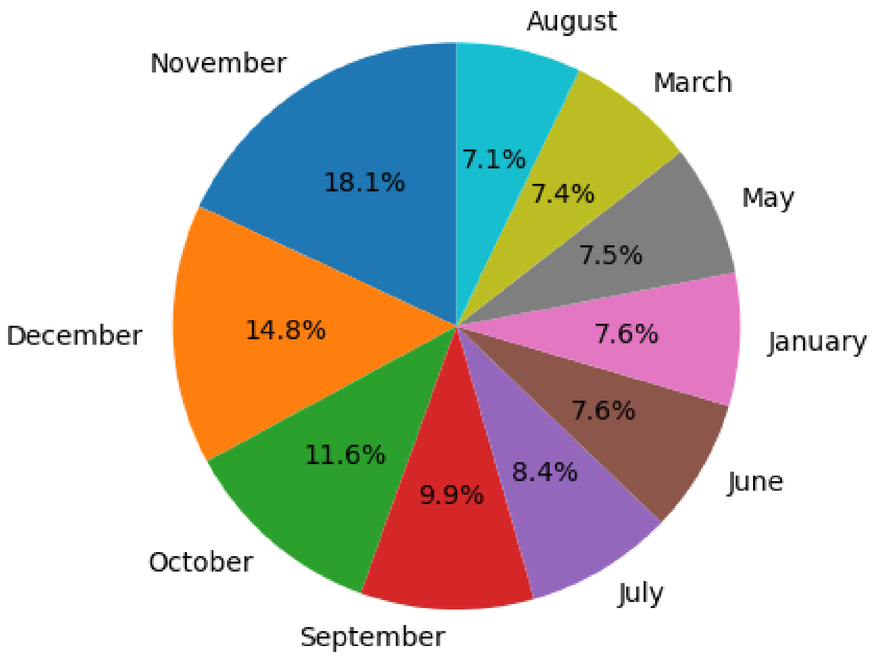
Order status of customers in 4 :segment



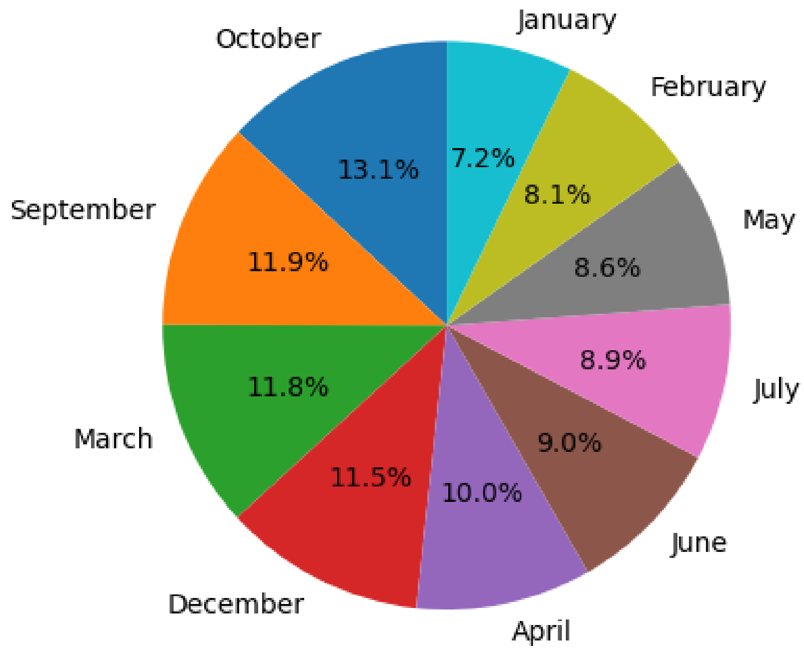
[ ]:
```
#pie charts showing the months where customers have made orders wrt each segment

labels=[]
sizes=[]
for i in cluster_list:
  customer_list=[]
  ␣
↪customer_list=df_segment_profiling['CustomerID'][df_segment_profiling['KMeans_Segment']==i]
↪to_list()
  ␣
↪df_temp=data[['CustomerID','Description','Country','Order_Status','Invoice_Month','Invoice_
↪isin(customer_list)]
  labels=list(df_temp['Invoice_Month'].value_counts().head(10).index)
  sizes=df_temp['Invoice_Month'].value_counts().head(10).to_list()
  print("Top invoice months customers in "+str(i)+" :segment")
  plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
```

```
plt.show()
```

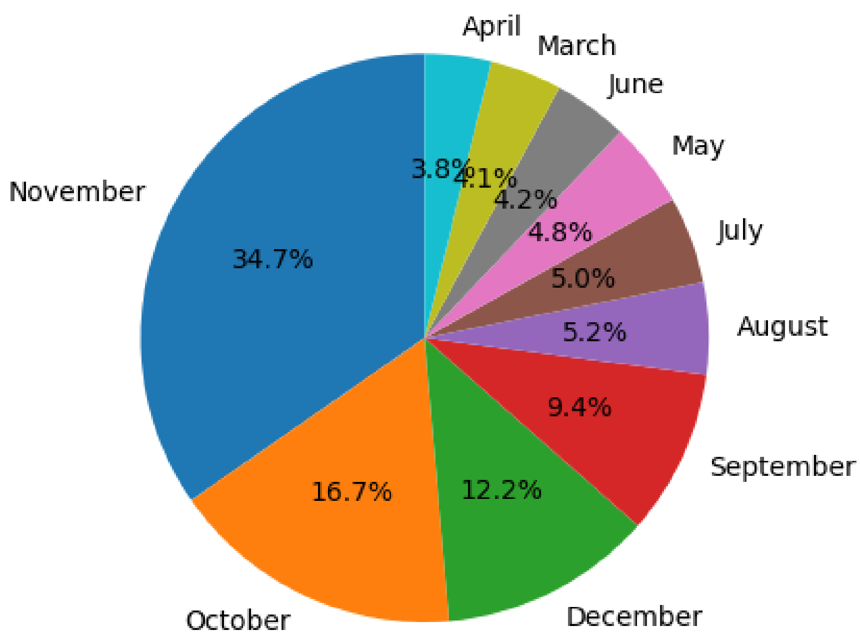
Top invoice months customers in 0 :segment



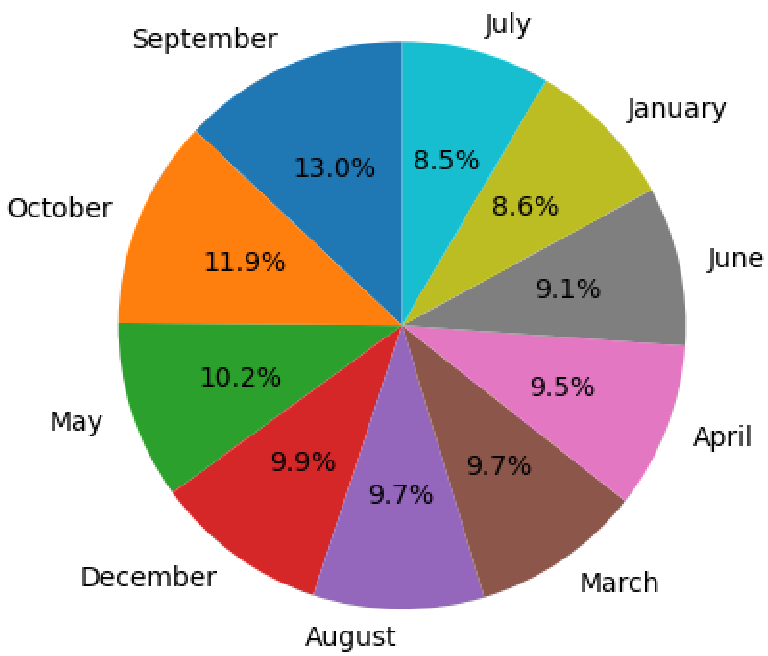
Top invoice months customers in 1 :segment



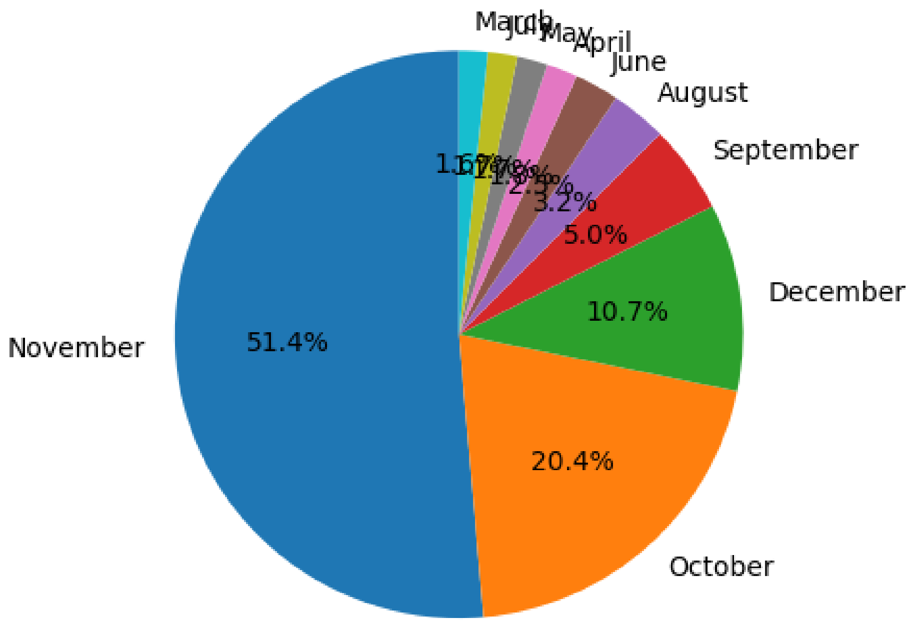
Top invoice months customers in 2 :segment



```
Top invoice months customers in 3 :segment
```




```
Top invoice months customers in 4 :segment
```



Insights: 1. Customers of segment 0 have shown keen interst on jumbo bag retrospot and white hanging heart tree holder. These customers are heavy spenders but have stopped doing so lately, a higher emphasis on improving quality of these products could be usefull. A vast majority of them belong to the UK, hence no use on focusing on that metric as of now. There is a significant portion of them who have cancelled orders. A further look into it could be very helpful. Finally, a vast majority of them happen to shop during november, launhing things like sales and discounts during that period could be very usefull for us.

2. Customers of segment 1 have also shown a hightened interest in white hanging heart tree holder. stratagies to boost this product could be applied on this segment as well. Since they are the customers whi have high recency but low spending tendency, additional sales associated with the former product could be useful. contrary to others, they tend to spend during october, another interesting trend to look into.

3. Customers of grp 2 and 3 have shown massive intersts in white hanging tree holder and cake casktets. coupled with the fact that they have shown most sales during the month of november, which is wedding season in america. An added marketing effort to boost wedding related products coulb be very useful to the company when it comes to increasing profits

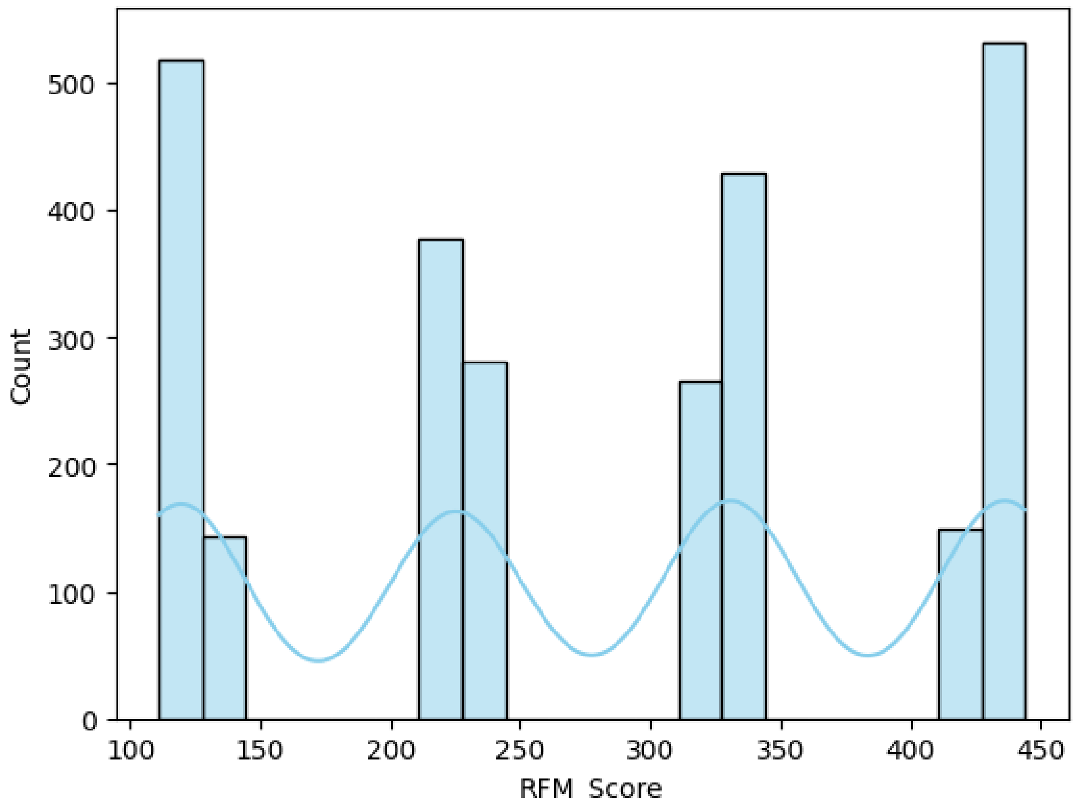
Visualisation

[ ]:
```
#distribution of rmf scores wrt customers belonging to each segment
for i in cluster_list:
  customer_list=[]
```

```
↪customer_list=df_segment_profiling['CustomerID'][df_segment_profiling['KMeans_Segment']==i]
↪to_list()
 df_temp=scored_df[['CustomerID','RFM_Score']][data['CustomerID'].
↪isin(customer_list)]
 print("RFM Score of customers belonging to "+ str(i)+" cluster")
 sns.histplot(df_temp['RFM_Score'], kde=True, bins=20, color='skyblue')
 plt.show()
```

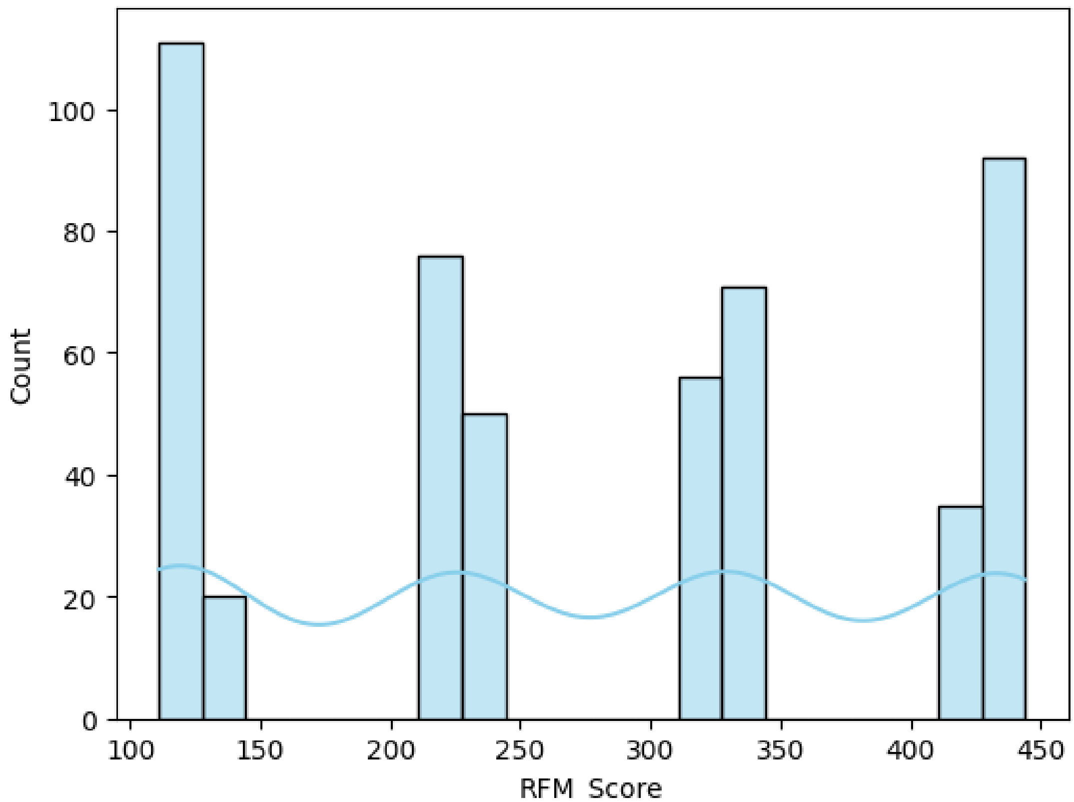
```
<ipython-input-125-999b01e42c02>:4: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  df_temp=scored_df[['CustomerID','RFM_Score']][data['CustomerID'].isin(customer
_list)]
```

```
RFM Score of customers belonging to 0 cluster
```



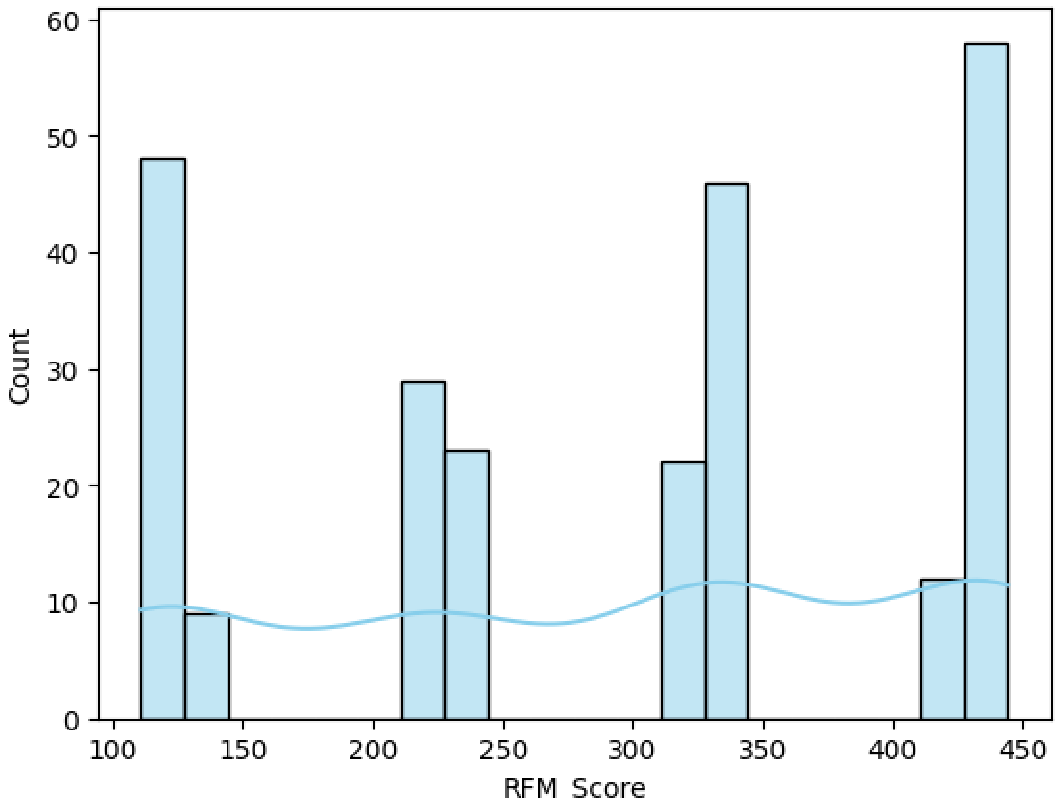

```
<ipython-input-125-999b01e42c02>:4: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  df_temp=scored_df[['CustomerID','RFM_Score']][data['CustomerID'].isin(customer
_list)]
```

```
RFM Score of customers belonging to 1 cluster
```



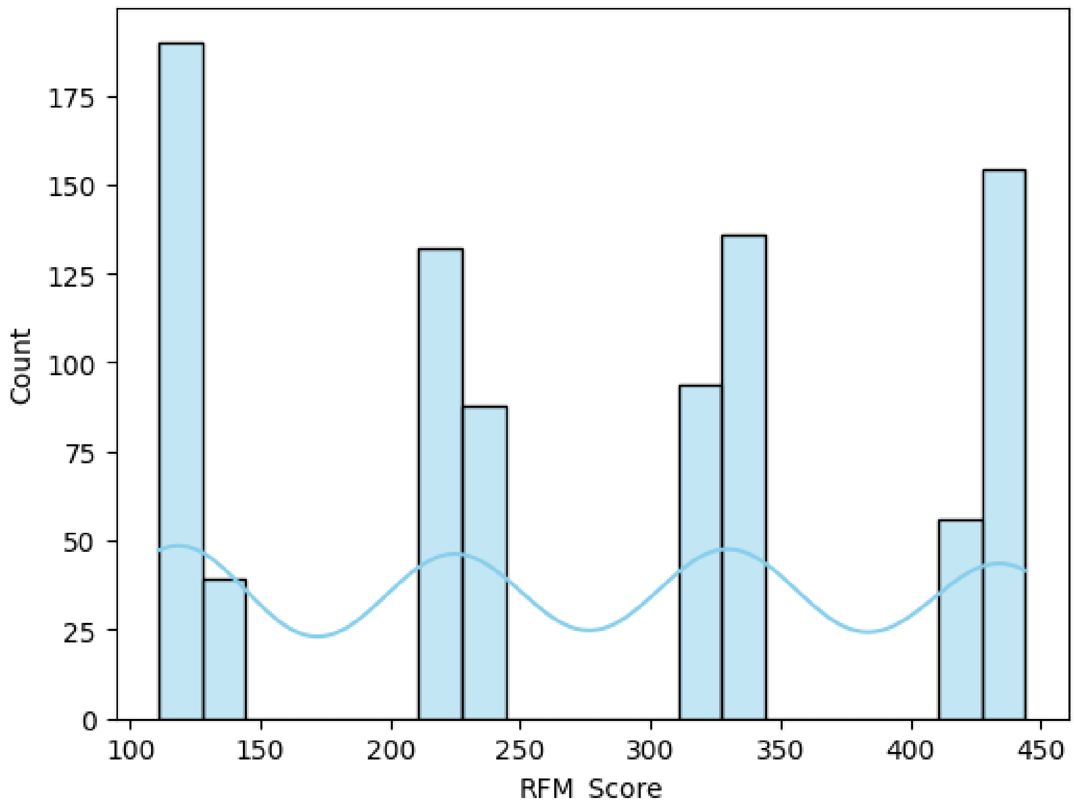

```
<ipython-input-125-999b01e42c02>:4: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  df_temp=scored_df[['CustomerID','RFM_Score']][data['CustomerID'].isin(customer
_list)]
```

```
RFM Score of customers belonging to 2 cluster
```



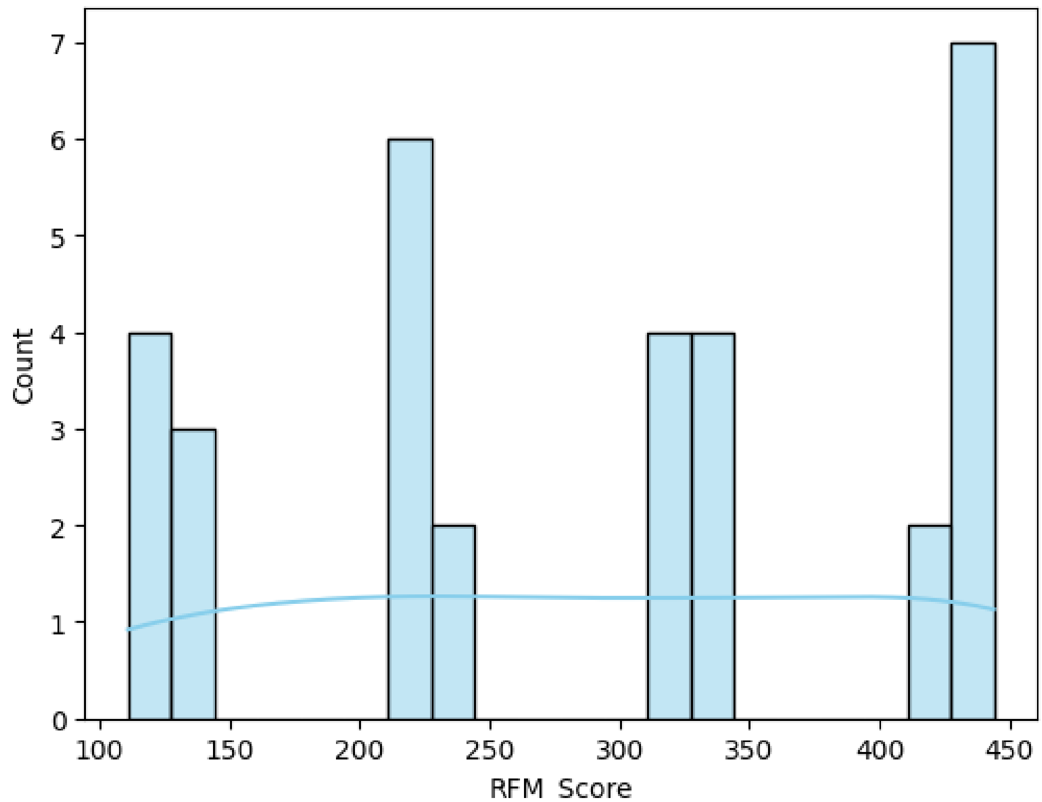

```
<ipython-input-125-999b01e42c02>:4: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  df_temp=scored_df[['CustomerID','RFM_Score']][data['CustomerID'].isin(customer
_list)]
```

```
RFM Score of customers belonging to 3 cluster
```




```
<ipython-input-125-999b01e42c02>:4: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  df_temp=scored_df[['CustomerID','RFM_Score']][data['CustomerID'].isin(customer
_list)]
```

```
RFM Score of customers belonging to 4 cluster
```



Solutions to individual questions

1. Data Overview - Anagha

[ ]:
```
#1
DF.shape
```

[ ]: (541909, 8)

[ ]:
```
#2
DF.describe()
```

[ ]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

[ ]:
```
#2
DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

[136]:
```
#3
#covers a time period from december 2010 to december 2011
print("Start date:", data['InvoiceDate'].min())
print("End date:", data['InvoiceDate'].max())
```

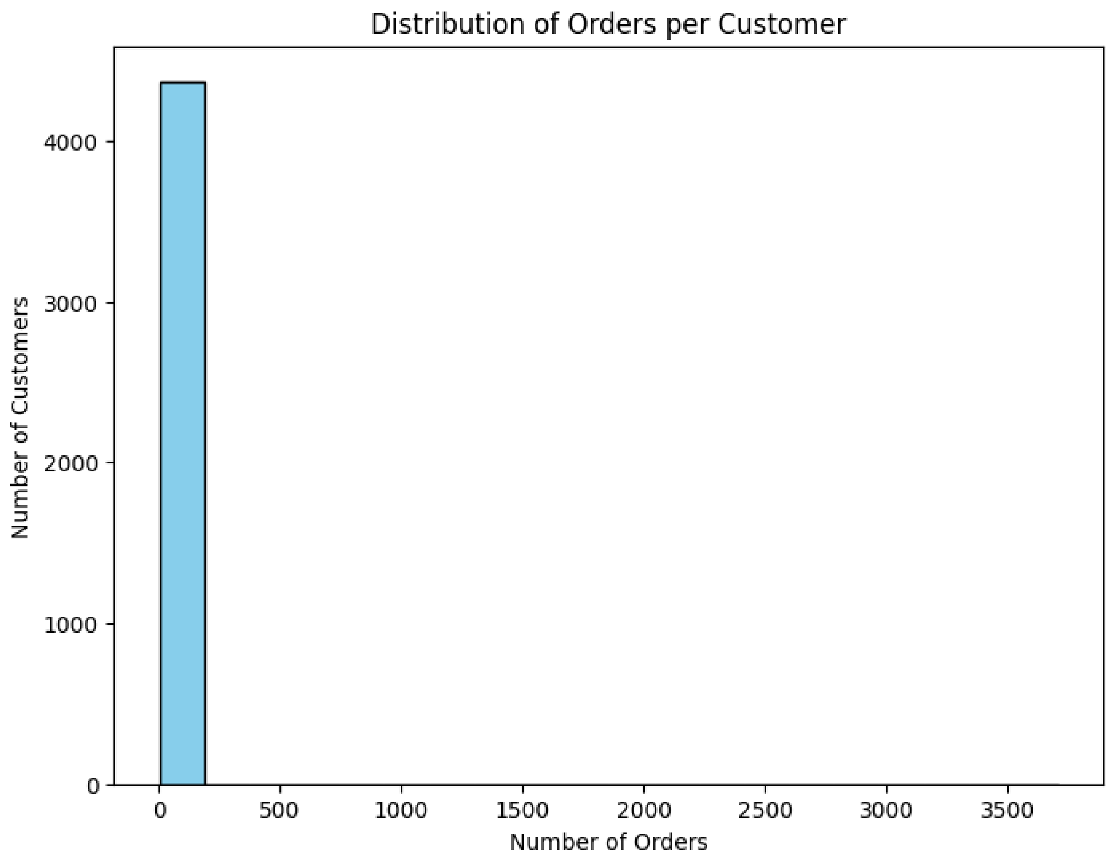
```
Start date: 2010-12-01 08:26:00
End date: 2011-12-09 12:50:00
```

2. Customer Analysis-Jash

[139]:
```
#1
# 4373 unique customers
merged_df.shape
```

[139]:
```
(4373, 8)
```

[176]:
```
#2
orders_per_customer = data.groupby('CustomerID')['InvoiceNo'].nunique()
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.hist(orders_per_customer, bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Number of Orders')
plt.ylabel('Number of Customers')
plt.title('Distribution of Orders per Customer')
plt.show()
```



[177]:
```
orders_per_customer.head()
```

[177]:
```
CustomerID
12346    2
12347    7
12348    4
12349    1
12350    1
Name: InvoiceNo, dtype: int64
```

[142]:
```
#3
df_temp_new = df_frequency.sort_values(by='Frequency', ascending=False)
print(df_temp_new['CustomerID'].head(5))
```

```
4372    Unavilable
4042         17841
1895         14911
1300         14096
330          12748
Name: CustomerID, dtype: object
```

3. Product Analysis - Jash

[144]:
```
#1
data['Description'].value_counts().head(5)
```

[144]:
```
WHITE HANGING HEART T-LIGHT HOLDER    2369
REGENCY CAKESTAND 3 TIER              2200
JUMBO BAG RED RETROSPOT               2159
PARTY BUNTING                         1727
LUNCH BAG RED RETROSPOT               1638
Name: Description, dtype: int64
```

[145]:
```
#2
data['UnitPrice'].mean()
```

[145]: 4.611113626088513

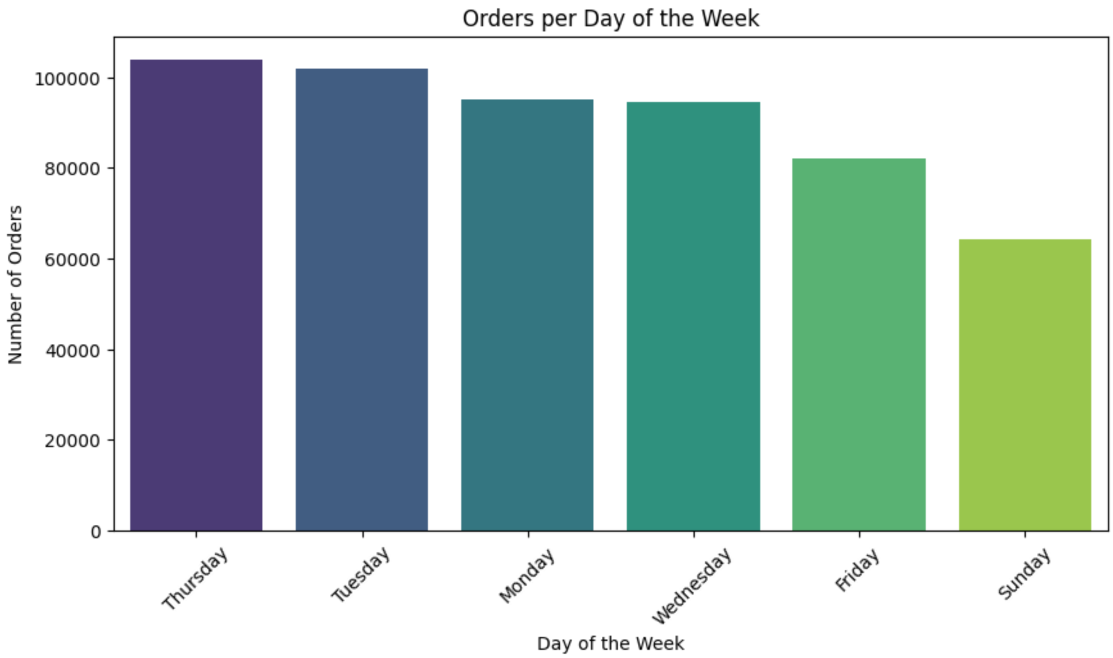
[147]:
```
#3
#catagory generating highest revenue
df_temp_2=data.groupby(by='Description', as_index=False)['TotalCost'].sum()
df_temp_2['Description'][df_temp_2['TotalCost']==df_temp_2['TotalCost'].max()]
```

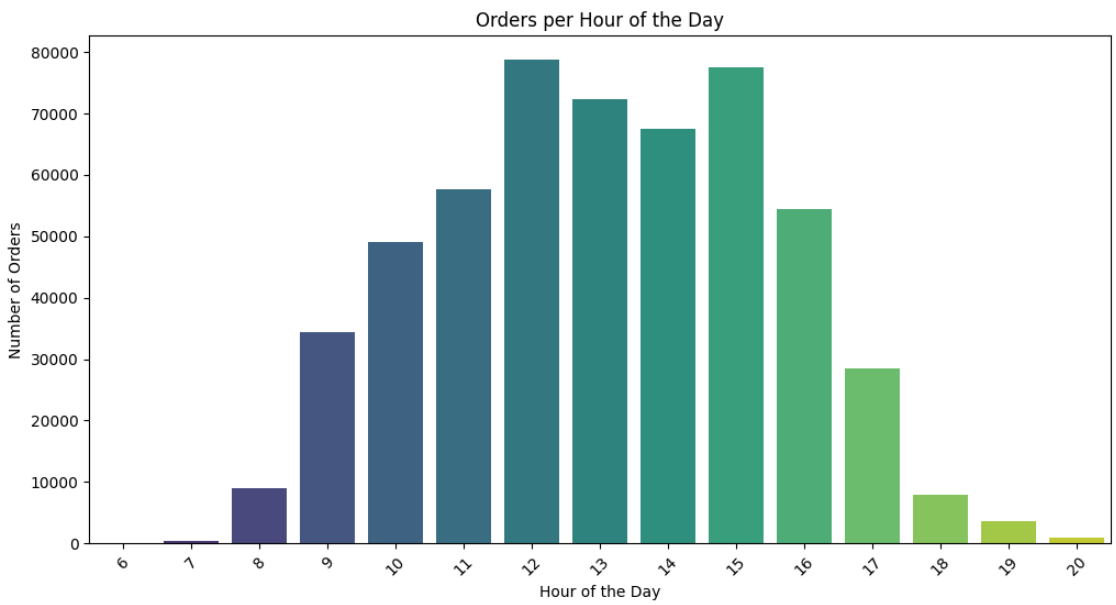
[147]:
```
2445    PAPER CRAFT , LITTLE BIRDIE
Name: Description, dtype: object
```

4. Time Analysis- Rutuja

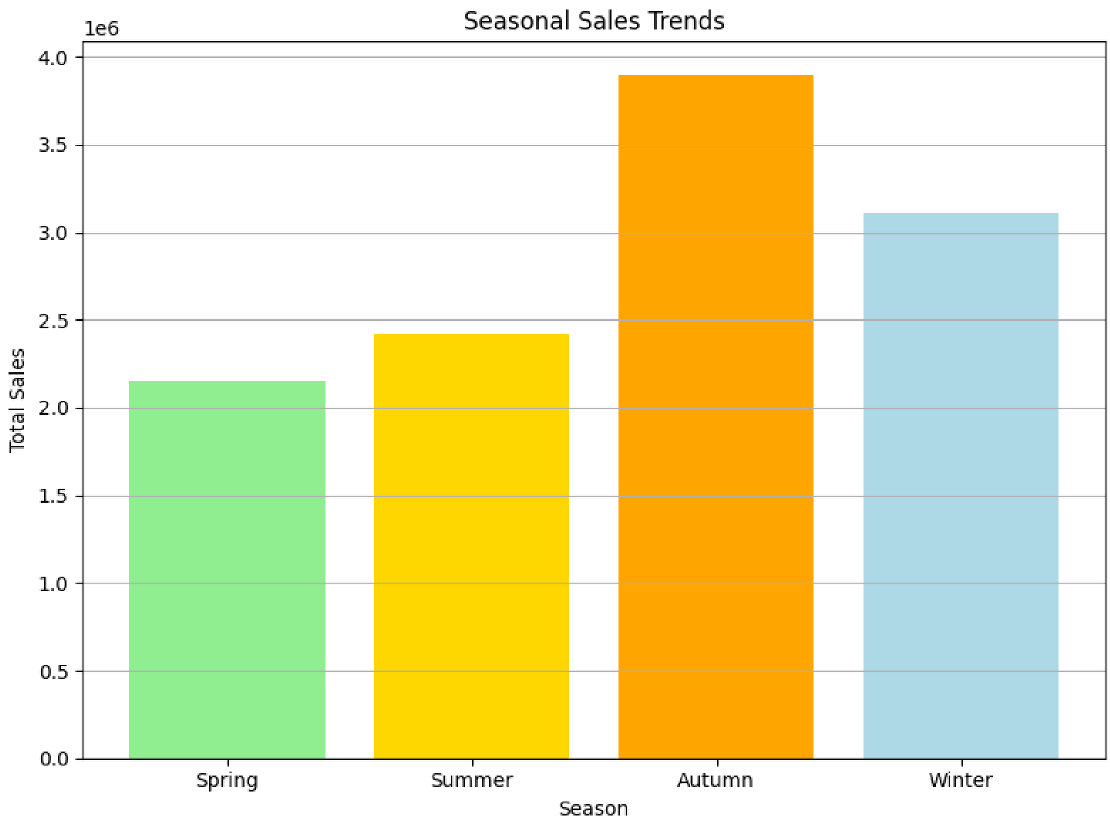
[148]:
```
#1
#graphs showing orders per day of the week and hour of the day
data['DayOfWeek'] = data['InvoiceDate'].dt.day_name()
data['HourOfDay'] = data['InvoiceDate'].dt.hour


orders_per_day = data['DayOfWeek'].value_counts()


orders_per_hour = data['HourOfDay'].value_counts()


plt.figure(figsize=(10, 5))
sns.barplot(x=orders_per_day.index, y=orders_per_day.values, palette='viridis')
plt.title('Orders per Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.show()
```

```
plt.figure(figsize=(12, 6))
sns.barplot(x=orders_per_hour.index, y=orders_per_hour.values,␣
↪palette='viridis')
plt.title('Orders per Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.show()
```





[179]:
```python
#observing seasonal trends
data['Month'] = data['InvoiceDate'].dt.month


seasons = {
    1: 'Winter', 2: 'Winter', 3: 'Spring',
    4: 'Spring', 5: 'Spring', 6: 'Summer',
    7: 'Summer', 8: 'Summer', 9: 'Autumn',
    10: 'Autumn', 11: 'Autumn', 12: 'Winter'
}
data['Season'] = data['Month'].map(seasons)


seasonal_sales = data.groupby('Season')['TotalCost'].sum()


plt.figure(figsize=(8, 6))
seasonal_sales = seasonal_sales.reindex(['Spring', 'Summer', 'Autumn',␣
↪'Winter'])  # Reordering seasons
plt.bar(seasonal_sales.index, seasonal_sales.values, color=['lightgreen',␣
↪'gold', 'orange', 'lightblue'])
plt.title('Seasonal Sales Trends')
plt.xlabel('Season')
plt.ylabel('Total Sales')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



[180]:
```
monthly_sales = data.groupby('Month')['TotalCost'].sum()
print(monthly_sales)
```

```
Month
1      822728.860
2      549201.130
3      752011.640
4      582410.121
5      817738.530
6      832356.680
7      757142.271
8      835596.250
9     1097492.722
10    1239253.930
11    1557236.410
12    1742452.610
Name: TotalCost, dtype: float64
```

5. Geographical Analysis- Saheel

[150]:
```python
#1
top_countries = data['Country'].value_counts().head(5)
print(top_countries)
```

```
United Kingdom    495478
Germany             9495
France              8557
EIRE                8196
Spain               2533
Name: Country, dtype: int64
```

[151]:
```python
#2
#no way to calculate correlation since countries are catagorical and average␣
↪order value is numeric
country_order_values = data.groupby('Country')['TotalCost'].sum()

orders_per_country = data.groupby('Country')['InvoiceNo'].nunique()

average_order_value = country_order_values / orders_per_country

print(average_order_value)
```

```
Country
Australia             2028.483333
Austria                539.107368
Bahrain                239.970000
Belgium                348.585882
Brazil                1143.600000
Canada                 611.063333
Channel Islands        630.745152
Cyprus                 711.723500
Czech Republic         189.152000
Denmark                911.549524
EIRE                   843.419722
European Community     261.750000
Finland                474.279583
France                 481.618915
Germany                391.436269
Greece                 801.753333
Hong Kong             1417.770667
Iceland                615.714286
Israel                 929.188889
Italy                  328.654000
Japan                 1410.432857
Lebanon               1693.880000
Lithuania              415.265000
Malta                  294.571000
```

```
Netherlands             2833.971683
Norway                   929.185500
Poland                   310.673333
Portugal                 537.002535
RSA                     1002.310000
Saudi Arabia              80.335000
Singapore               3343.819000
Spain                    651.234667
Sweden                   873.059783
Switzerland              781.006081
USA                      775.694286
United Arab Emirates     634.093333
United Kingdom           419.793896
Unspecified              365.368462
dtype: float64
```

6. Payment Analysis- Anagha

[ ]:
```
#Data is unavailable
```

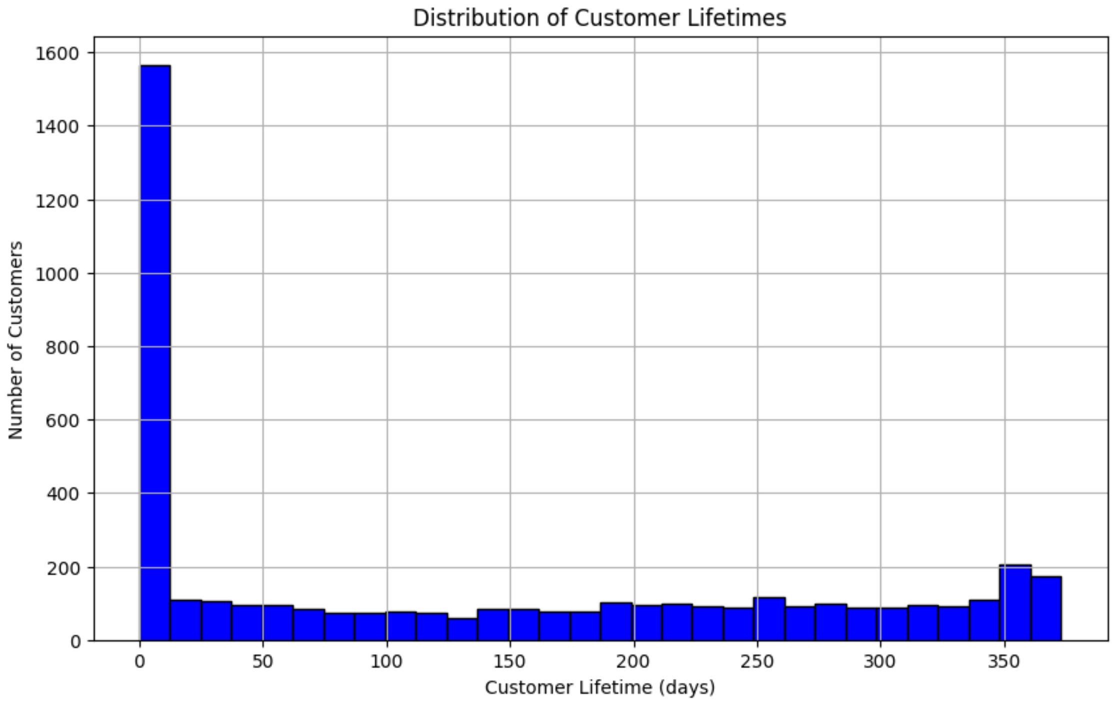
7. Customer Behaviour- Harsh

[152]:
```
#Average customer lifespan
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

customer_lifetime = data.groupby('CustomerID')['InvoiceDate'].agg(['min',␣
 ↪'max'])
customer_lifetime['Lifetime'] = (customer_lifetime['max'] -␣
 ↪customer_lifetime['min']).dt.days

average_lifetime = customer_lifetime['Lifetime'].mean()
print("Average customer lifetime (days):", average_lifetime)
```

```
Average customer lifetime (days): 133.44042991081636
```

[154]:
```
#plotting average customer lifelines

customer_lifetime = data.groupby('CustomerID')['InvoiceDate'].agg(['min',␣
 ↪'max'])
customer_lifetime['Lifetime'] = (customer_lifetime['max'] -␣
 ↪customer_lifetime['min']).dt.days

plt.figure(figsize=(10, 6))
plt.hist(customer_lifetime['Lifetime'], bins=30, color='blue',␣
 ↪edgecolor='black')
plt.title('Distribution of Customer Lifetimes')
plt.xlabel('Customer Lifetime (days)')
plt.ylabel('Number of Customers')
```

```
plt.grid(True)
plt.show()
```



[155]:

```
#Customer Segments based on purchase behavior
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

customer_data = data.groupby('CustomerID').agg({
    'InvoiceDate': 'max',
    'Quantity': 'sum',
    'UnitPrice': 'mean'
}).reset_index()

customer_data['TotalAmount'] = customer_data['Quantity'] *␣
 ↪customer_data['UnitPrice']
features = ['TotalAmount', 'Quantity', 'UnitPrice']
scaler = StandardScaler()
scaled_data = scaler.fit_transform(customer_data[features])

kmeans = KMeans(n_clusters=3, random_state=42)
customer_data['Cluster'] = kmeans.fit_predict(scaled_data)

cluster_stats = customer_data.groupby('Cluster')[features].mean()
print(cluster_stats)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
         TotalAmount      Quantity    UnitPrice
Cluster
0       3.789664e+03    1068.974353     5.100706
1       1.802309e+06  127045.000000    68.088802
2       2.395296e+05      29.500000  6171.705000
```

8. Returns and Refunds- Harsh

[163]:
```
#top 10 products which have been returned
data[data['Quantity']<0]['Description'].value_counts().head(10)
```

[163]:
```
Manual                                 244
REGENCY CAKESTAND 3 TIER               181
POSTAGE                                126
check                                  120
JAM MAKING SET WITH JARS                87
Discount                                77
SET OF 3 CAKE TINS PANTRY DESIGN        74
SAMPLES                                 61
STRAWBERRY CERAMIC TRINKET BOX          55
ROSES REGENCY TEACUP AND SAUCER         54
Name: Description, dtype: int64
```

[ ]:
```
#no way to observe correlation since product catagory is a catagorical column␣
↪and number of returned orders is a numeric column
```

9. Profitability Analysis- Saheel

[165]:
```
data['Profit'] = data['TotalCost'] * (np.random.uniform(-0.1, 0.3, len(data)))
```

[167]:
```
profit_total = data['Profit'].sum()
total_revenue = data['TotalCost'].sum()
print("Total Profit generated: ", profit_total)
print("Profit Percentage: ", (profit_total/total_revenue)*100)
```

```
Total Profit generated:  1102306.8178391655
Profit Percentage:  9.514438657944444
```

[169]:
```
#Top 5 products with highest profit margins
df_product = data.groupby('Description').agg({
    'UnitPrice': 'mean',
    'TotalCost': 'sum',
    'Profit': 'sum'
}).reset_index()
```

```
df_product.columns= ['Product','Average Unit Price','Total Revenue','Total␣
↪Profit']
df_product.sort_values(by='Total Profit', ascending=False, inplace=True)
df_product.head()
```

[169]:

```
                            Product  Average Unit Price  Total Revenue  \
2246                         Manual          374.914266      224897.28
171                      AMAZON FEE         7324.784706      249042.68
2915          REGENCY CAKESTAND 3 TIER           13.800277      184207.29
1098                  DOTCOM POSTAGE          290.905585      206252.06
3918  WHITE HANGING HEART T-LIGHT HOLDER            3.204251      112917.07

      Total Profit
2246  24288.074259
171   24019.893559
2915  19396.498494
1098  19232.542881
3918  11009.580230
```

10. Customer Satisfaction - Rutuja

[171]:

```
data['Customer Satisfaction'] = np.random.randint(1, 6, len(data))
```

[173]:

```
customer_satisfaction = data.groupby('Description')['Customer Satisfaction'].
↪mean().reset_index()
customer_satisfaction.columns=["Product","Average Product Customer␣
↪Satisfaction"]
customer_satisfaction
```

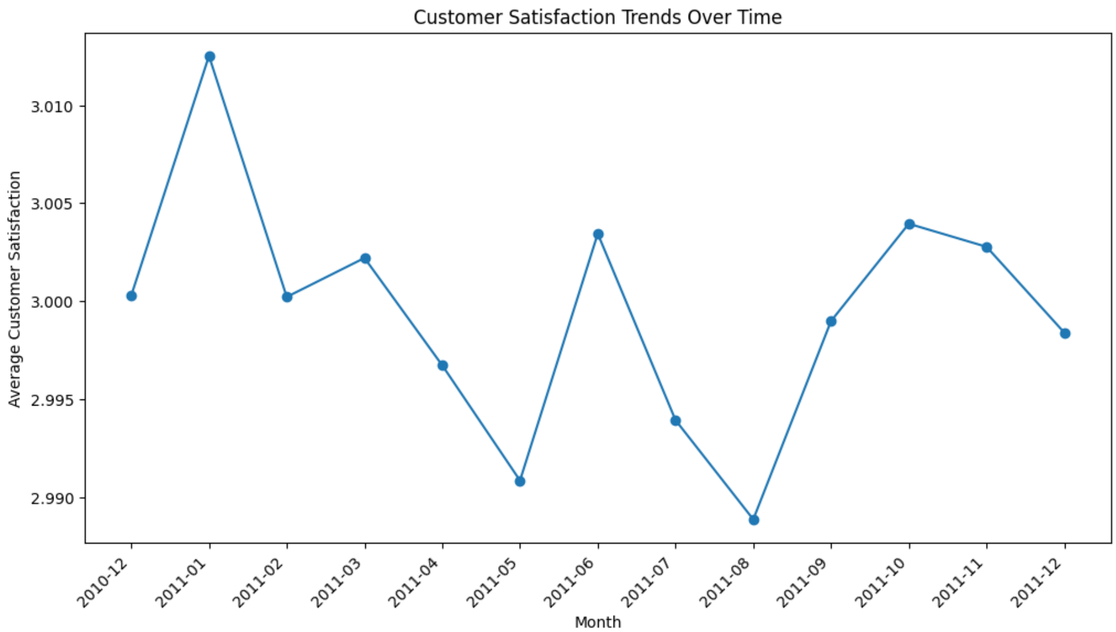
[173]:

```
                           Product  Average Product Customer Satisfaction
0       4 PURPLE FLOCK DINNER CANDLES                               3.146341
1       50'S CHRISTMAS GIFT BAG LARGE                               3.169231
2                   DOLLY GIRL BEAKER                               2.812155
3         I LOVE LONDON MINI BACKPACK                               3.193182
4         I LOVE LONDON MINI RUCKSACK                               2.000000
…                                 …                                    …
4218        wrongly marked carton 22804                               3.000000
4219       wrongly marked. 23343 in box                               4.000000
4220       wrongly sold (22719) barcode                               1.000000
4221               wrongly sold as sets                               4.000000
4222                  wrongly sold sets                               2.000000

[4223 rows x 2 columns]
```

[175]:

```
satisfaction_trends = data.groupby(data['InvoiceDate'].dt.
↪to_period("M"))['Customer Satisfaction'].mean()
```

```
plt.figure(figsize=(12, 6))
plt.plot(satisfaction_trends.index.astype(str), satisfaction_trends.values,␣
↪marker='o')
plt.title('Customer Satisfaction Trends Over Time')
plt.xlabel('Month')
plt.ylabel('Average Customer Satisfaction')
plt.xticks(rotation=45, ha='right')
plt.show()
```



[ ]:
```
#Code ends here
```