Online Retail - CP3- Final

June 9, 2024

[1]:
```python
import numpy as np
import pandas as pd
data = pd.read_excel('Online Retail.xlsx')
```

[2]:
```python
data.head()
```

[2]:
```
  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 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

Gives a preview of outliers. Any column whose mean is not around 50% is an outlier

[3]:
```python
data.describe()
```

[3]:
```
            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
```

Gives us an idea of the type of data and the missing values

[4]:
```python
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  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

[5]:
```
typecol = ['Categorical', 'Categorical','Categorical', 'Discrete', 'Date',␣
↪'Continuous' , 'Categorical' , 'Categorical' ]
missingdf = pd.DataFrame({'Columns' : data.columns.to_list(), 'Type of data' :␣
↪typecol, 'No of missing data' : data.isna().sum()})
def highlight_max(s):
    is_max = s
    return ['background-color: pink' if v else '' for v in is_max]
missingdf.style.apply(highlight_max, subset = ['No of missing data'])
missingdf.style.hide_index()
```

```
/tmp/ipykernel_253/3468921664.py:7: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis="index")`
  missingdf.style.hide_index()
```

[5]: <pandas.io.formats.style.Styler at 0x7fb446f1bf40>

[6]:
```
print ("No of records before dropping customer ID column")
print (len(data))
data.drop(data[data['CustomerID'].isna()].index, inplace = True)
data.reset_index(drop=True)
print ("No of records after dropping customer ID column")
print (len(data))
print ("Is there any missing data in Description column after dropping the Null␣
↪Customer ID columns")
print (any(data['Description'].isna()==True))
missingdf = pd.DataFrame({'Columns' : data.columns.to_list(), 'No of missing␣
↪data after cleaning' : data.isna().sum()})
missingdf.style.hide_index()
```

```
No of records before dropping customer ID column
541909
No of records after dropping customer ID column
```

```
406829
Is there any missing data in Description column after dropping the Null Customer
ID columns
False
```

```
/tmp/ipykernel_253/1142630559.py:10: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis="index")`
  missingdf.style.hide_index()
```

[6]:
```
<pandas.io.formats.style.Styler at 0x7fb44716fbb0>
```

0.0.1 b. Remove duplicate data records.

[7]:
```
print ("No of records before dropping duplicate records")
print (len(data))
data.drop_duplicates(inplace=True)
data.reset_index(drop=True)
print ("No of records after dropping duplicate records")
print (len(data))
```

```
No of records before dropping duplicate records
406829
No of records after dropping duplicate records
401604
```

0.0.2 c. Remove the transactions of the last month in year 2011 as they have only data for 9 days

[8]:
```
# Define a function that will parse the date
import datetime
def get_month(x):
    return datetime.datetime(x.year,x.month,x.day)
print ("No of records before dropping the transactions of the last month")
print (len(data))

# Create InvoiceMonth column
data['InvoiceMonth'] = data['InvoiceDate'].apply(get_month)
data[data['InvoiceMonth'] > datetime.datetime(2011,11,30)]
data.drop(data[data['InvoiceMonth'] > datetime.datetime(2011,11,30)].index,␣
 ↪inplace = True)

data.reset_index(drop=True)
print ("No of records after dropping the transactions of the last month")
print (len(data))
```

```
No of records before dropping the transactions of the last month
401604
```

```
No of records after dropping the transactions of the last month
384222
```

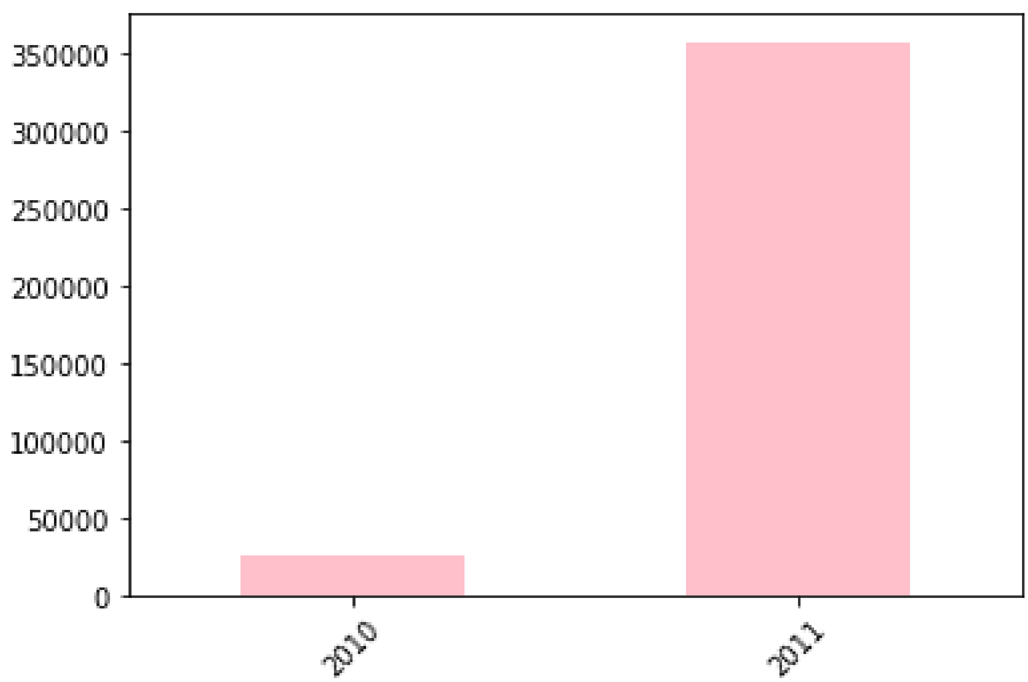
0.0.3 d. Perform descriptive analytics on the given data

Observe the countries that have most of the customers residing

[9]:
```python
data.Country.value_counts(normalize=True).head(10).mul(100).round(2).
 ↪astype(str) + ' %'
```

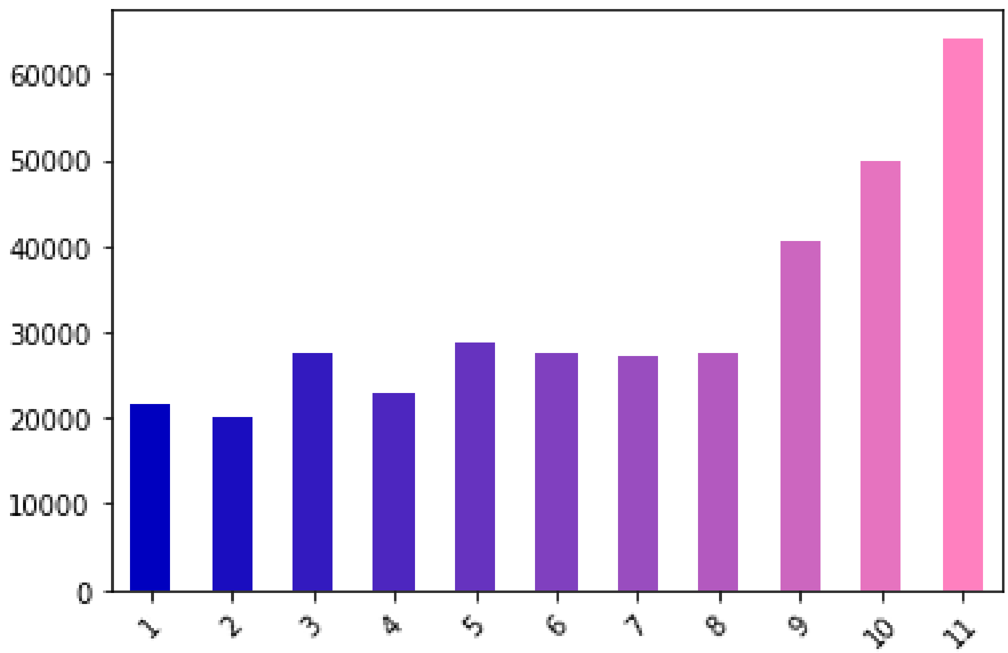
[9]:
```
United Kingdom    88.73 %
Germany            2.38 %
France             2.12 %
EIRE               1.86 %
Spain              0.64 %
Netherlands        0.59 %
Belgium            0.51 %
Switzerland        0.49 %
Portugal           0.36 %
Australia          0.33 %
Name: Country, dtype: object
```

[10]:
```python
data.InvoiceDate.dt.year.value_counts(sort=False).plot(kind='bar', rot=45,␣
 ↪color='pink');
```



Let us visualize the customer trend on a monthly basis in the year 2011

[11]:
```
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(data[data.InvoiceDate.dt.
 ↪year==2011].InvoiceDate.dt.month.value_counts(sort=False)))]
data[data.InvoiceDate.dt.year==2011].InvoiceDate.dt.month.
 ↪value_counts(sort=False).plot(kind='bar', rot=45, color=my_colors);
```

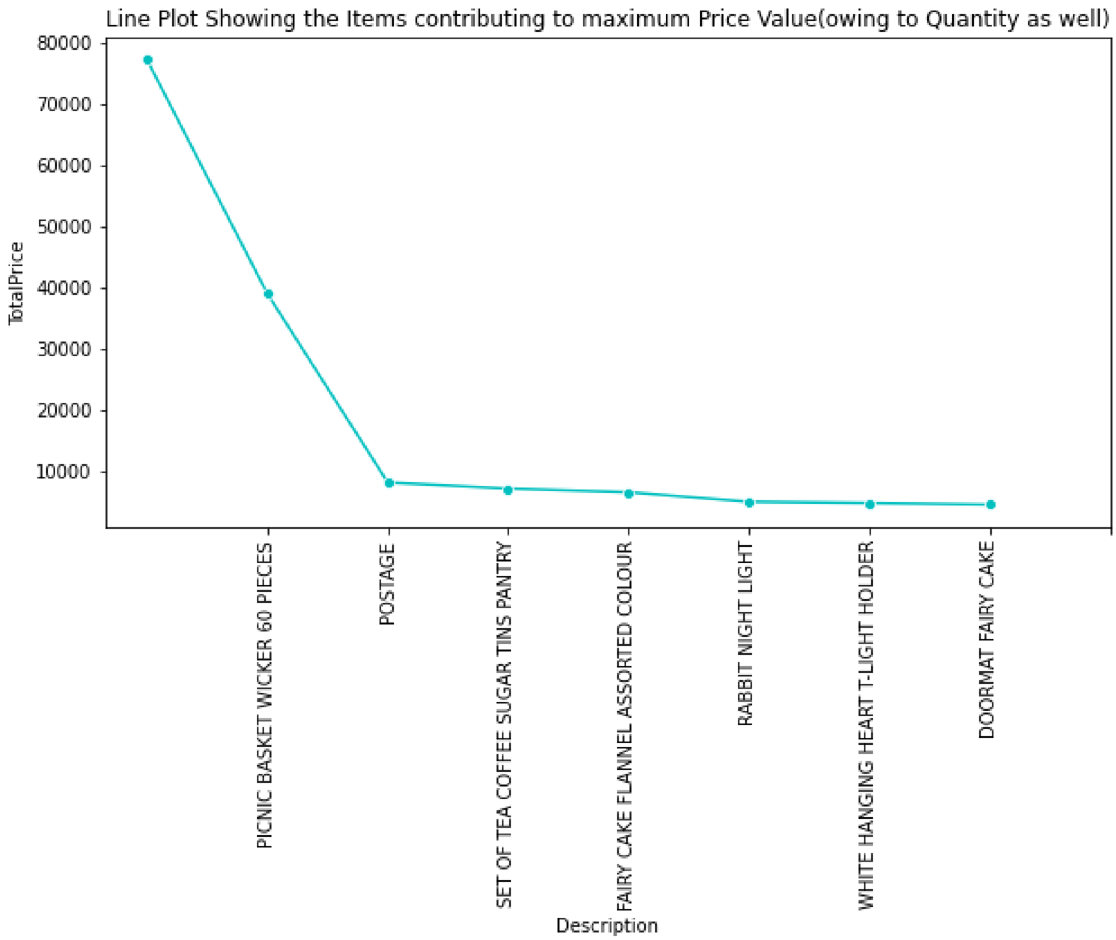


Visualize the Items contributing to maximum Price Value

[12]:
```
data['TotalPrice'] = data.Quantity * data.UnitPrice
df =data.TotalPrice.sort_values(ascending=False).head(10).to_frame().style.
 ↪hide_index()
desc = data.sort_values(by='TotalPrice', ascending=False)['Description'].
 ↪head(10)
price = data.sort_values(by='TotalPrice', ascending=False)['TotalPrice'].
 ↪head(10)
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
sns.lineplot(y=price,x=desc, marker='o', color='c',).set_title('Line Plot␣
 ↪Showing the Items contributing to maximum Price Value(owing to Quantity as␣
 ↪well)')
plt.xticks(range(1,9), rotation=90)
plt.show();
```

```
/tmp/ipykernel_253/2347732985.py:2: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis="index")`
  df =data.TotalPrice.sort_values(ascending=False).head(10).to_frame().style.hid
```

```
e_index()
```



Let us explore the data some more!

[13]:
```
print ("First business transaction date is {}".format(data.InvoiceDate.min()))
print ("Last business transaction  date is {}".format(data.InvoiceDate.max()))
monthly_gross =data[data.InvoiceDate.dt.year==2011].groupby(data.InvoiceDate.dt.
  ↪month).TotalPrice.sum()
df = pd.DataFrame(monthly_gross)
df.index.name = 'Invoice Month'
df
```

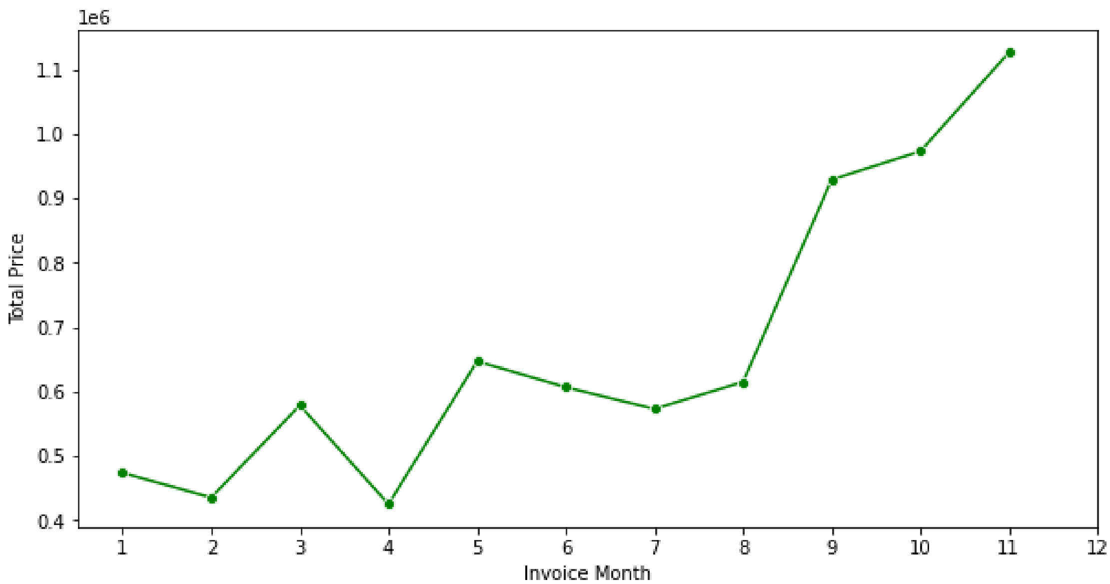
```
First business transaction date is 2010-12-01 08:26:00
Last business transaction  date is 2011-11-30 17:42:00
```

[13]:

	TotalPrice
Invoice Month	
1	473731.900

```
2              435534.070
3              578576.210
4              425222.671
5              647011.670
6              606862.520
7              573112.321
8              615078.090
9              929356.232
10             973306.380
11            1126815.070
```

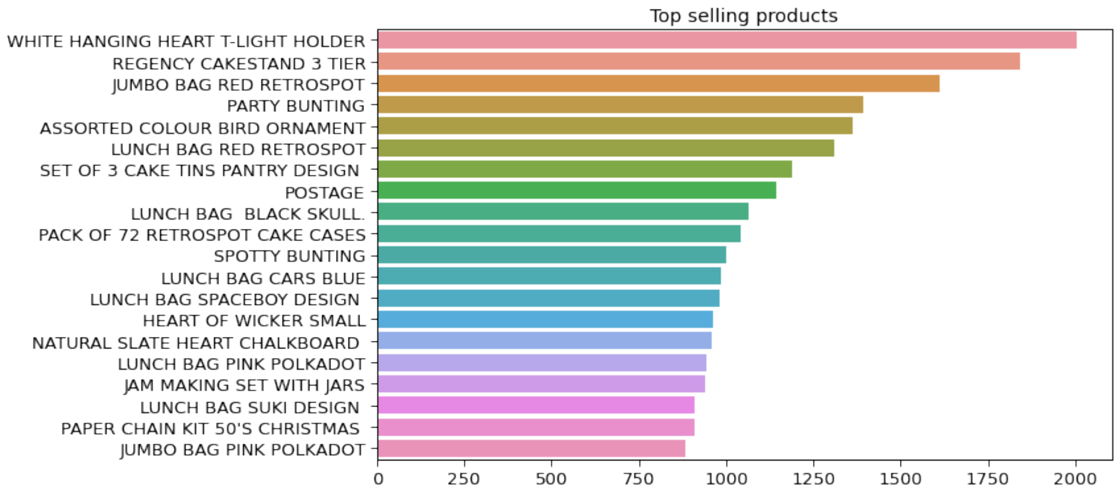
[14]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
sns.lineplot(y=monthly_gross.values,x=monthly_gross.index, marker='o',␣
 ↪color='g');
plt.xlabel('Invoice Month')
plt.ylabel('Total Price')
plt.xticks(range(1,13))
plt.show();
```



Let's visualize some top products from the whole range.

[15]:
```
top_products = data['Description'].value_counts()[:20]
plt.figure(figsize=(10,6))
sns.set_context("paper", font_scale=1.5)
sns.barplot(y = top_products.index,
            x = top_products.values)
```

```
plt.title("Top selling products")
plt.show();
```



[16]:
```
pd.DataFrame(data['Description'].value_counts())
```

[16]:
```
                                    Description
WHITE HANGING HEART T-LIGHT HOLDER         2005
REGENCY CAKESTAND 3 TIER                   1843
JUMBO BAG RED RETROSPOT                    1613
PARTY BUNTING                              1391
ASSORTED COLOUR BIRD ORNAMENT              1363
…                                           …
PINK FLOCK PHOTO FRAME                        1
BLING KEY RING STAND                          1
LASER CUT MULTI STRAND NECKLACE               1
WHITE ROSEBUD & PEARL NECKLACE                1
SILVER AND BLACK ORBIT NECKLACE               1

[3887 rows x 1 columns]
```

[17]:
```
def outlierDetection(datacolumn):
    #Sort the data in ascending order
    sorted(datacolumn)

    #GET Q1 and Q3
    Q1,Q3 = np.percentile(datacolumn, [25,75])

    #Calc IQR
    IQR = Q3 - Q1
```

```
    #Calc LowerRange
    lr = Q1 - (1.5 * IQR)

    #Calc Upper Range
    ur = Q3 + (1.5 * IQR)

    return lr,ur

#Outliers detection are considered only for numeric columns.ie Quantity , Unit␣
 ↪Price and Total Price

def outlier_treatment(drop_col = False):
    for col in data.columns[[3,5,8]]:
        lowerRange,upperRange = outlierDetection(data[col])
        if not data[(data[col] > upperRange) | (data[col] < lowerRange)].empty:
            print ("Detected  outliers for this column %r " % col)
            #hdataUpdated.drop(hdataUpdated[(hdataUpdated[col] > upperRange) |␣
 ↪(hdataUpdated[col] < lowerRange)].index , inplace=drop_col)
```

[18]:
```
cohort =data.copy()
cohort
```

[18]:
```
        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
…             …         …                                    …       …
516379    C579886     22197                       POPCORN HOLDER        -1
516380    C579886     23146       TRIPLE HOOK ANTIQUE IVORY ROSE        -1
516381    C579887     84946         ANTIQUE SILVER T-LIGHT GLASS        -1
516382    C579887     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        -1
516383    C579887     23490      T-LIGHT HOLDER HANGING LOVE BIRD        -3

                InvoiceDate  UnitPrice  CustomerID         Country  \
0       2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1       2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2       2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3       2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4       2010-12-01 08:26:00       3.39     17850.0  United Kingdom
…                         …        …          …              …
516379  2011-11-30 17:39:00       0.85     15676.0  United Kingdom
516380  2011-11-30 17:39:00       3.29     15676.0  United Kingdom
516381  2011-11-30 17:42:00       1.25     16717.0  United Kingdom
516382  2011-11-30 17:42:00       7.95     16717.0  United Kingdom
516383  2011-11-30 17:42:00       3.75     16717.0  United Kingdom
```

```
       InvoiceMonth  TotalPrice
0        2010-12-01       15.30
1        2010-12-01       20.34
2        2010-12-01       22.00
3        2010-12-01       20.34
4        2010-12-01       20.34
…               …          …
516379   2011-11-30       -0.85
516380   2011-11-30       -3.29
516381   2011-11-30       -1.25
516382   2011-11-30       -7.95
516383   2011-11-30      -11.25

[384222 rows x 10 columns]
```

[19]:
```python
# Define a function that will parse the date
import datetime
def get_month(x):
    return datetime.datetime(x.year,x.month,1)

# Create InvoiceMonth column
cohort['InvoiceMonth'] = cohort['InvoiceDate'].apply(get_month)

# Group by CustomerID and select the InvoiceMonth value
grouping = cohort.groupby('CustomerID')['InvoiceMonth']

# Assign a minimum InvoiceMonth value to the dataset
cohort['CohortMonth'] = grouping.transform('min')
```

[20]:
```python
cohort
#grouping
```

[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
…            …        …                                   …       …
516379   C579886     22197                       POPCORN HOLDER        -1
516380   C579886     23146       TRIPLE HOOK ANTIQUE IVORY ROSE        -1
516381   C579887     84946         ANTIQUE SILVER T-LIGHT GLASS        -1
516382   C579887     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        -1
516383   C579887     23490     T-LIGHT HOLDER HANGING LOVE BIRD        -3

               InvoiceDate  UnitPrice  CustomerID         Country  \
```

```
0      2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2      2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
…                     …         …         …             …
516379 2011-11-30 17:39:00       0.85     15676.0  United Kingdom
516380 2011-11-30 17:39:00       3.29     15676.0  United Kingdom
516381 2011-11-30 17:42:00       1.25     16717.0  United Kingdom
516382 2011-11-30 17:42:00       7.95     16717.0  United Kingdom
516383 2011-11-30 17:42:00       3.75     16717.0  United Kingdom

       InvoiceMonth  TotalPrice  CohortMonth
0        2010-12-01       15.30   2010-12-01
1        2010-12-01       20.34   2010-12-01
2        2010-12-01       22.00   2010-12-01
3        2010-12-01       20.34   2010-12-01
4        2010-12-01       20.34   2010-12-01
…             …           …          …
516379   2011-11-01       -0.85   2011-03-01
516380   2011-11-01       -3.29   2011-03-01
516381   2011-11-01       -1.25   2010-12-01
516382   2011-11-01       -7.95   2010-12-01
516383   2011-11-01      -11.25   2010-12-01

[384222 rows x 11 columns]
```

[21]:
```python
def get_date_int(df, column):
    year = df[column].dt.year
    month = df[column].dt.month
    return year, month

# Get the integers for date parts from the `InvoiceMonth` column
invoice_year, invoice_month = get_date_int(cohort,'InvoiceMonth')

# Get the integers for date parts from the `CohortMonth` column
cohort_year, cohort_month = get_date_int(cohort,'CohortMonth')
```

[22]:
```python
print ("Unique terms for Cohort Year is {} " .format(cohort_year.unique()))
print ("Unique terms for Cohort Month is {} " .format(cohort_month.unique()))
print ("Unique terms for Invoice Year is {} " .format(invoice_year.unique()))
print ("Unique terms for Invoice Year is {} " .format(invoice_month.unique()))
```

```
Unique terms for Cohort Year is [2010 2011]
Unique terms for Cohort Month is [12  1  2  3  4  5  6  7  8  9 10 11]
Unique terms for Invoice Year is [2010 2011]
Unique terms for Invoice Year is [12  1  2  3  4  5  6  7  8  9 10 11]
```

[23]:
```
# Calculate difference in years
years_diff = invoice_year - cohort_year

# Calculate difference in months
months_diff = invoice_month - cohort_month

# Extract the difference in months from all previous values
cohort['CohortIndex'] = years_diff * 12 + months_diff + 1
```

[24]:
```
#THis Cohort Index will give us an idea on the time difference in months␣
↪between the customer's first purchase and the customer's current purchase
cohort['CohortIndex'].unique()
cohort
```

[24]:
```
       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
…            …         …                                    …       …
516379   C579886     22197                       POPCORN HOLDER        -1
516380   C579886     23146       TRIPLE HOOK ANTIQUE IVORY ROSE        -1
516381   C579887     84946         ANTIQUE SILVER T-LIGHT GLASS        -1
516382   C579887     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        -1
516383   C579887     23490     T-LIGHT HOLDER HANGING LOVE BIRD        -3

               InvoiceDate  UnitPrice  CustomerID         Country  \
0      2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2      2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
…                      …         …         …             …
516379 2011-11-30 17:39:00       0.85     15676.0  United Kingdom
516380 2011-11-30 17:39:00       3.29     15676.0  United Kingdom
516381 2011-11-30 17:42:00       1.25     16717.0  United Kingdom
516382 2011-11-30 17:42:00       7.95     16717.0  United Kingdom
516383 2011-11-30 17:42:00       3.75     16717.0  United Kingdom

       InvoiceMonth  TotalPrice CohortMonth  CohortIndex
0        2010-12-01       15.30  2010-12-01            1
1        2010-12-01       20.34  2010-12-01            1
2        2010-12-01       22.00  2010-12-01            1
3        2010-12-01       20.34  2010-12-01            1
4        2010-12-01       20.34  2010-12-01            1
…             …           …          …          …
```

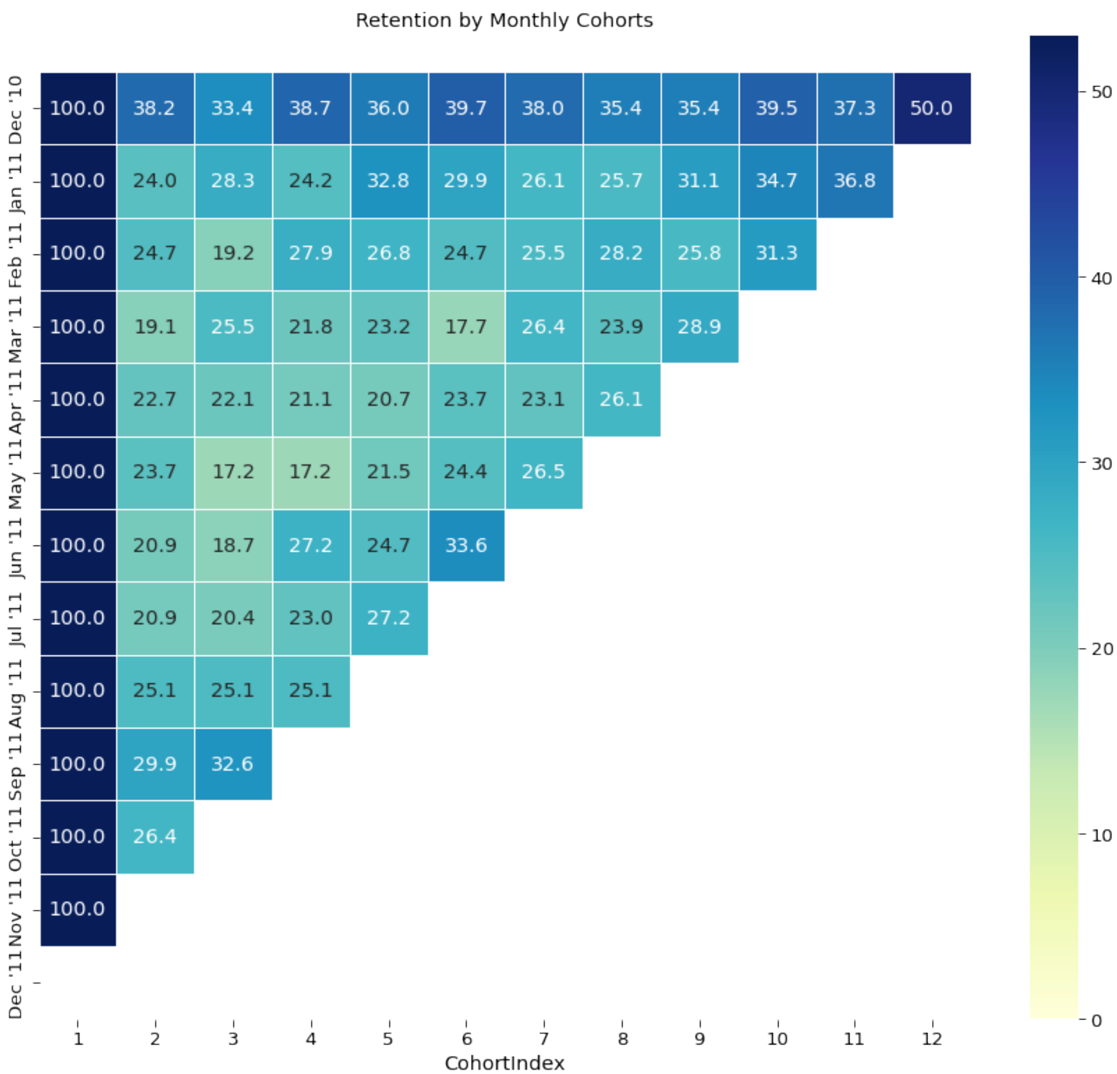
```
516379   2011-11-01       -0.85   2011-03-01           9
516380   2011-11-01       -3.29   2011-03-01           9
516381   2011-11-01       -1.25   2010-12-01          12
516382   2011-11-01       -7.95   2010-12-01          12
516383   2011-11-01      -11.25   2010-12-01          12

[384222 rows x 12 columns]
```

[25]:
```
grouping = cohort.groupby(['CohortMonth', 'CohortIndex'])
```

[26]:
```
# Count the number of unique values per customer ID
#cohort_data = grouping['CustomerID'].apply(pd.Series.nunique).reset_index()
cohort_data = grouping['CustomerID'].apply(pd.Series.nunique).reset_index()

# Create a pivot
cohort_counts = cohort_data.pivot(index='CohortMonth', columns='CohortIndex',␣
 ↪values='CustomerID')

# Select the first column and store it to cohort_sizes
cohort_sizes = cohort_counts.iloc[:,0]

# Divide the cohort count by cohort sizes along the rows
retention = cohort_counts.divide(cohort_sizes, axis=0)*100
#print (cohort[cohort['CohortMonth']=='2011-12-01']['CustomerID'].nunique())␣
 ↪#Verifies 41 against this month
#cohort_sizes
retention.index = retention.index.date
```

[27]:
```
month_list = ["Dec '10", "Jan '11", "Feb '11", "Mar '11", "Apr '11",\
              "May '11", "Jun '11", "Jul '11", "Aug '11", "Sep '11", \
              "Oct '11", "Nov '11", "Dec '11"]

# Initialize inches plot figure
plt.figure(figsize=(15,13))

# Add a title
plt.title('Retention by Monthly Cohorts')

# Create the heatmap
ax = sns.heatmap(data=retention,
            annot = True,
            cmap = "YlGnBu",
            vmin = 0.0,
            vmax = list(retention.max().sort_values(ascending = False))[1]+3,
            fmt = '.1f',
            linewidth = 0.9,
            yticklabels=month_list)
```

```
# plot of the data
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
fig = plt.figure()
plt.show();
```




```
<Figure size 432x288 with 0 Axes>
```

Calculate average price per cohort Now we will calculate the average price metric and analyze if there are any differences in shopping patterns across time and across cohorts

[28]:
```
# Create a groupby object and pass the monthly cohort and cohort index as a list
grouping = cohort.groupby(['CohortMonth', 'CohortIndex'])
```

```
# Calculate the average of the unit price column
cohort_data = grouping['UnitPrice'].mean()

# Reset the index of cohort_data
cohort_data = cohort_data.reset_index()

# Create a pivot
average_price = cohort_data.pivot(index='CohortMonth', columns='CohortIndex',␣
↪values='UnitPrice')
#average_price.round(1)
#average_price.index = average_price.index.date
average_price
#cohort_data
#cohort
```

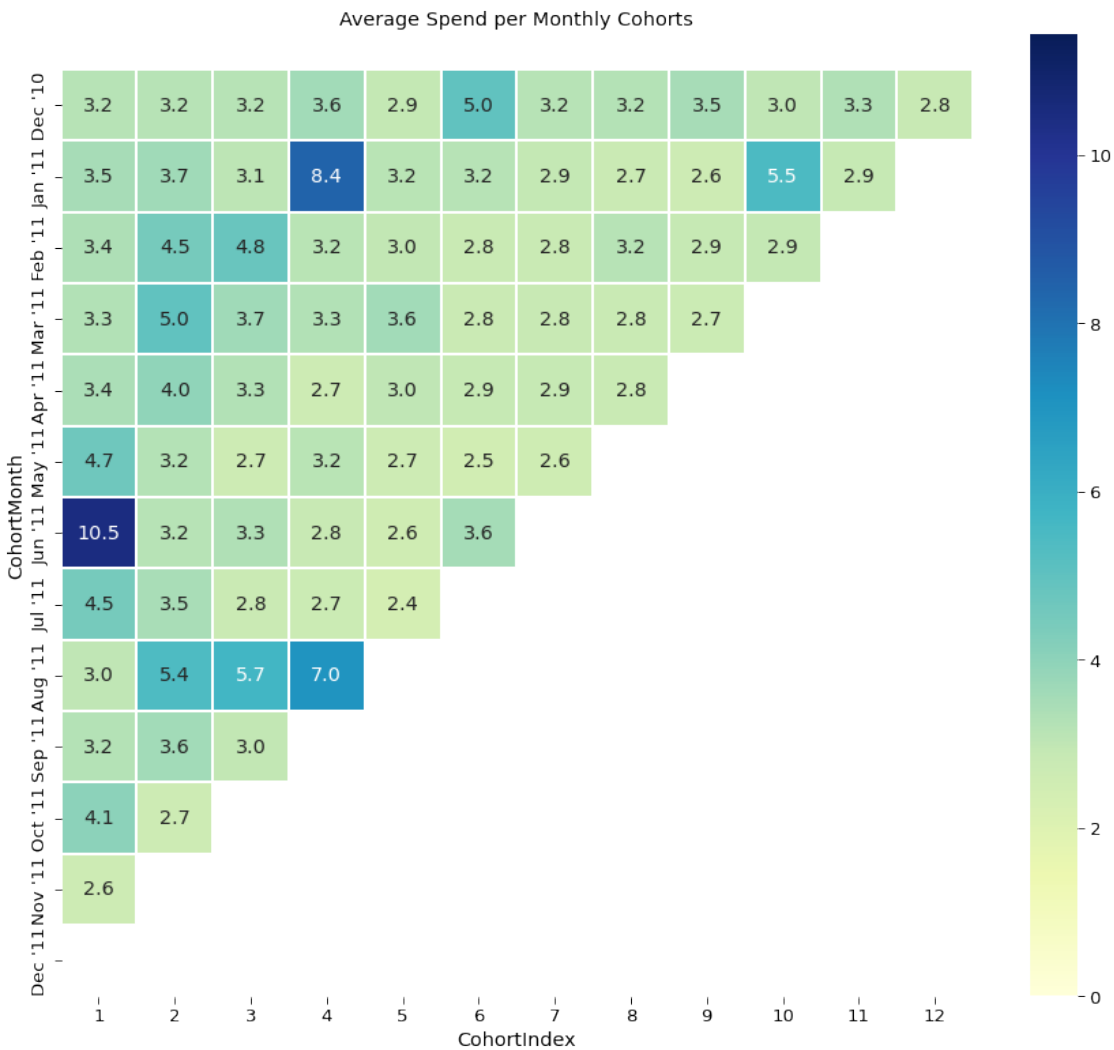
[28]:

```
CohortIndex         1         2         3         4         5         6  \
CohortMonth
2010-12-01   3.216682  3.182040  3.207467  3.603758  2.937803  4.996508
2011-01-01   3.505492  3.653572  3.069534  8.439024  3.157803  3.172919
2011-02-01   3.355968  4.469638  4.824106  3.150045  2.987616  2.792577
2011-03-01   3.302802  4.990095  3.655094  3.289768  3.616562  2.758381
2011-04-01   3.431172  3.958074  3.300128  2.673439  3.028297  2.867185
2011-05-01   4.662054  3.243691  2.652761  3.167391  2.667158  2.495751
2011-06-01  10.490030  3.205283  3.343994  2.835952  2.553037  3.550657
2011-07-01   4.493676  3.480495  2.752121  2.701985  2.403989       NaN
2011-08-01   3.028246  5.425904  5.714033  7.046410       NaN       NaN
2011-09-01   3.235116  3.584834  2.957893       NaN       NaN       NaN
2011-10-01   4.053162  2.678140       NaN       NaN       NaN       NaN
2011-11-01   2.641554       NaN       NaN       NaN       NaN       NaN

CohortIndex        7         8         9        10        11        12
CohortMonth
2010-12-01  3.184572  3.235695  3.511560  3.035982  3.309705  2.835557
2011-01-01  2.918498  2.749649  2.641686  5.489040  2.886220       NaN
2011-02-01  2.812985  3.214380  2.894988  2.946092       NaN       NaN
2011-03-01  2.843273  2.809136  2.707846       NaN       NaN       NaN
2011-04-01  2.902668  2.812492       NaN       NaN       NaN       NaN
2011-05-01  2.615408       NaN       NaN       NaN       NaN       NaN
2011-06-01       NaN       NaN       NaN       NaN       NaN       NaN
2011-07-01       NaN       NaN       NaN       NaN       NaN       NaN
2011-08-01       NaN       NaN       NaN       NaN       NaN       NaN
2011-09-01       NaN       NaN       NaN       NaN       NaN       NaN
2011-10-01       NaN       NaN       NaN       NaN       NaN       NaN
2011-11-01       NaN       NaN       NaN       NaN       NaN       NaN
```

[29]:

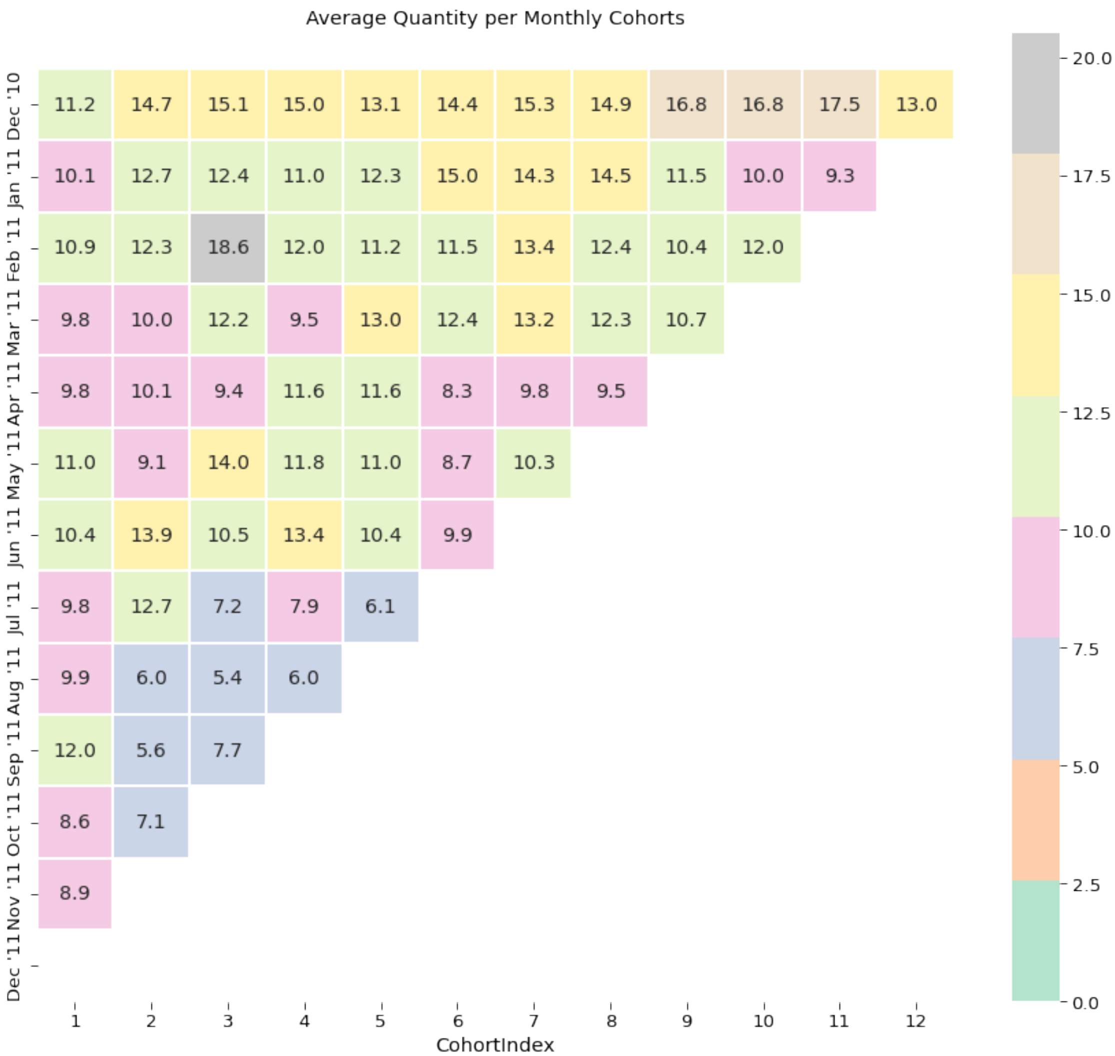
```
# Initialize plot figure
plt.figure(figsize=(15, 13))
```

```
plt.title('Average Spend per Monthly Cohorts')
# Create the heatmap
ax = sns.heatmap(data = average_price,
            annot=True,
            vmin = 0.0,
#             vmax =20,
            cmap='YlGnBu',
            vmax = list(average_price.max().sort_values(ascending = False))[1]+3,
            fmt = '.1f',
            linewidth = 0.3,
            yticklabels=month_list)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show();
```



[30]:
```python
# Create a groupby object and pass the monthly cohort and cohort index as a list
grouping = cohort.groupby(['CohortMonth', 'CohortIndex'])

# Calculate the average of the Quantity column
cohort_data = grouping['Quantity'].mean()

# Reset the index of cohort_data
cohort_data = cohort_data.reset_index()

# Create a pivot
average_quantity = cohort_data.pivot(index='CohortMonth',␣
↪columns='CohortIndex', values='Quantity')
average_quantity.round(1)
average_quantity.index = average_quantity.index.date
```

[31]:
```python
# Initialize plot figure
plt.figure(figsize=(15, 13))

# Add a title
plt.title('Average Quantity per Monthly Cohorts')

# Create the heatmap
ax  = sns.heatmap(data = average_quantity,
            annot=True,
            vmin = 0.0,
            cmap='Pastel2',
            vmax = list(average_quantity.max().sort_values(ascending =␣
↪False))[1]+3,
            fmt = '.1f',
            linewidth = 0.3,
            yticklabels=month_list)

bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show();
```



[32]:
```python
Segment =  ['Platinum Customers',
                     'Big Spenders',
                     'High Spend New Customers',
                     'Lowest-Spending Active Loyal Customers',
                     'Recent Customers',
                     'Good Customers Almost Lost',
                     'Churned Best Customers',
                     'Lost Cheap Customers ']
RFM = [
               ['444', '443'],
               ['114', '124', '134', '144', '214', '224', '234', '244', '314',
 ↪'324', '334', '344', '414', '424', '434', '444'],
               ['413', '314', '313', '414'],
               ['331', '341', '431', '441'],
```

```
                 ['422', '423', '424', '432', '433', '434', '442', '443', '444'],
                 ['244', '234', '243', '233'],
                 ['144', '134', '143', '133'],
                 ['122', '111', '121', '112', '221', '212', '211']
                 ]
# Create a dictionary for each segment to map them against each customer
Description = ['Customers who bought most recently, most often and spend the␣
 ↪most',
                 'Customers who spend the most',
                 'New Customers who spend the most',
                 'Active Customers who buy very often but spend less ',
                 'Customers who have purchased recently',
                 'Customers who were frequent and good spenders who are becoming␣
 ↪very inactive',
                 'Customers who were frequent and good spenders who are lost␣
 ↪contributing to attrition',
                 'Customers who purchased long ago , less frequent and very␣
 ↪little']

Marketing = ['No price incentives, New products and Loyalty Programs',
                        'Market your most expensive products',
                        'Price Incentives',
                        'Promote economical cost effective products in daily use',
                        'Discounts and promote a variety of product sells',
                        'Aggressive Price Incentives',
                        'Monitor close communication with customers with constant␣
 ↪feedback and rework ',
                        'Dont spend too much time to re-acquire',
                        ]
rfm_segments = pd.DataFrame({'Segment': Segment , 'RFM' : RFM , 'Description':␣
 ↪Description, 'Marketing': Marketing})
rfm_segments
```

[32]:

```
                                      Segment  \
0                          Platinum Customers
1                               Big Spenders
2                     High Spend New Customers
3  Lowest-Spending Active Loyal Customers
4                           Recent Customers
5                   Good Customers Almost Lost
6                       Churned Best Customers
7                         Lost Cheap Customers

                                                   RFM  \
0                                           [444, 443]
1  [114, 124, 134, 144, 214, 224, 234, 244, 314, …
```

```
2                               [413, 314, 313, 414]
3                               [331, 341, 431, 441]
4      [422, 423, 424, 432, 433, 434, 442, 443, 444]
5                               [244, 234, 243, 233]
6                               [144, 134, 143, 133]
7                [122, 111, 121, 112, 221, 212, 211]

                                         Description  \
0  Customers who bought most recently, most often…
1                     Customers who spend the most
2                 New Customers who spend the most
3  Active Customers who buy very often but spend …
4             Customers who have purchased recently
5  Customers who were frequent and good spenders …
6  Customers who were frequent and good spenders …
7  Customers who purchased long ago , less freque…

                                           Marketing
0  No price incentives, New products and Loyalty …
1               Market your most expensive products
2                                  Price Incentives
3  Promote economical cost effective products in …
4   Discounts and promote a variety of product sells
5                        Aggressive Price Incentives
6  Monitor close communication with customers wit…
7             Dont spend too much time to re-acquire
```

[33]:
```python
#last date available in our dataset
import datetime as dt
data['InvoiceDate'].max()
```

[33]:
```
Timestamp('2011-11-30 17:42:00')
```

[34]:
```python
# Lets set this date as the today's date for further analysis
current_date = dt.date(2011,11,30)
current_date
```

[34]:
```
datetime.date(2011, 11, 30)
```

[35]:
```python
# Lets create a date column for date values only
data['Purchase_Date'] = data.InvoiceDate.dt.date
```

[36]:
```python
recency = data.groupby('CustomerID')['Purchase_Date'].max().reset_index()
recency
```

[36]:
```
      CustomerID Purchase_Date
0        12346.0    2011-01-18
```

```
1        12347.0    2011-10-31
2        12348.0    2011-09-25
3        12349.0    2011-11-21
4        12350.0    2011-02-02
…          …          …
4326     18280.0    2011-03-07
4327     18281.0    2011-06-12
4328     18282.0    2011-08-09
4329     18283.0    2011-11-30
4330     18287.0    2011-10-28

[4331 rows x 2 columns]
```

[37]:
```
# Create a separate column for this date.
recency = recency.assign(Current_Date = current_date)
recency
```

[37]:
```
      CustomerID Purchase_Date Current_Date
0        12346.0    2011-01-18   2011-11-30
1        12347.0    2011-10-31   2011-11-30
2        12348.0    2011-09-25   2011-11-30
3        12349.0    2011-11-21   2011-11-30
4        12350.0    2011-02-02   2011-11-30
…          …          …            …
4326     18280.0    2011-03-07   2011-11-30
4327     18281.0    2011-06-12   2011-11-30
4328     18282.0    2011-08-09   2011-11-30
4329     18283.0    2011-11-30   2011-11-30
4330     18287.0    2011-10-28   2011-11-30

[4331 rows x 3 columns]
```

[38]:
```
# Compute the number of days since last purchase
recency['Recency'] = recency.Purchase_Date.apply(lambda x: (current_date - x).
 ↪days)
current_date
```

[38]:
```
datetime.date(2011, 11, 30)
```

[39]:
```
recency.head()
```

[39]:
```
   CustomerID Purchase_Date Current_Date  Recency
0     12346.0    2011-01-18   2011-11-30      316
1     12347.0    2011-10-31   2011-11-30       30
2     12348.0    2011-09-25   2011-11-30       66
3     12349.0    2011-11-21   2011-11-30        9
4     12350.0    2011-02-02   2011-11-30      301
```

[40]:
```
# Drop the irrelevant Date columns
recency.drop(['Purchase_Date','Current_Date'], axis=1, inplace=True)
recency
```

[40]:
```
      CustomerID  Recency
0        12346.0      316
1        12347.0       30
2        12348.0       66
3        12349.0        9
4        12350.0      301
…            …        …
4326     18280.0      268
4327     18281.0      171
4328     18282.0      113
4329     18283.0        0
4330     18287.0       33

[4331 rows x 2 columns]
```

[41]:
```
frequency = data.groupby('CustomerID').InvoiceNo.nunique().reset_index().
 ↪rename(columns={'InvoiceNo':'Frequency'})
frequency.max()
```

[41]:
```
CustomerID    18287.0
Frequency       238.0
dtype: float64
```

[42]:
```
# Create a separate column for Total Cost of Unit purchased
data['Total_cost'] = data.Quantity * data.UnitPrice
data
```

[42]:
```
       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
…            …         …                                    …       …
516379   C579886     22197                       POPCORN HOLDER        -1
516380   C579886     23146       TRIPLE HOOK ANTIQUE IVORY ROSE        -1
516381   C579887     84946         ANTIQUE SILVER T-LIGHT GLASS        -1
516382   C579887     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        -1
516383   C579887     23490      T-LIGHT HOLDER HANGING LOVE BIRD        -3

               InvoiceDate  UnitPrice  CustomerID         Country  \
0      2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

```
2      2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
…                      …       …        …             …
516379 2011-11-30 17:39:00       0.85     15676.0  United Kingdom
516380 2011-11-30 17:39:00       3.29     15676.0  United Kingdom
516381 2011-11-30 17:42:00       1.25     16717.0  United Kingdom
516382 2011-11-30 17:42:00       7.95     16717.0  United Kingdom
516383 2011-11-30 17:42:00       3.75     16717.0  United Kingdom

       InvoiceMonth  TotalPrice Purchase_Date  Total_cost
0        2010-12-01       15.30    2010-12-01       15.30
1        2010-12-01       20.34    2010-12-01       20.34
2        2010-12-01       22.00    2010-12-01       22.00
3        2010-12-01       20.34    2010-12-01       20.34
4        2010-12-01       20.34    2010-12-01       20.34
…             …          …           …          …
516379   2011-11-30       -0.85    2011-11-30       -0.85
516380   2011-11-30       -3.29    2011-11-30       -3.29
516381   2011-11-30       -1.25    2011-11-30       -1.25
516382   2011-11-30       -7.95    2011-11-30       -7.95
516383   2011-11-30      -11.25    2011-11-30      -11.25

[384222 rows x 12 columns]
```

[43]:
```python
monetary = data.groupby('CustomerID').Total_cost.sum().reset_index().
 ↪rename(columns={'Total_cost':'Monetary'})
monetary.head()
```

[43]:
```
   CustomerID  Monetary
0     12346.0      0.00
1     12347.0   4085.18
2     12348.0   1797.24
3     12349.0   1757.55
4     12350.0    334.40
```

Now Combine all three to form an aggregated RFM Table

[44]:
```python
rf = recency.merge(frequency, on='CustomerID')
rfm_table = rf.merge(monetary, on='CustomerID')
```

[45]:
```python
rfm_table.set_index('CustomerID',inplace=True)
rfm_table.head()
#rfm_table.Monetary.max()
```

[45]:
```
            Recency  Frequency  Monetary
CustomerID
```

```
12346.0         316           2      0.00
12347.0          30           6   4085.18
12348.0          66           4   1797.24
12349.0           9           1   1757.55
12350.0         301           1    334.40
```

[46]:
```
rfm_table.index[1]
```

[46]:
```
12347.0
```

[47]:
```
# Fetch the records corresponding to the first customer id in above table
data[data.CustomerID == rfm_table.index[1]]
```

[47]:
```
       InvoiceNo StockCode                          Description  Quantity  \
14938     537626     85116      BLACK CANDELABRA T-LIGHT HOLDER        12
14939     537626     22375     AIRLINE BAG VINTAGE JET SET BROWN         4
14940     537626     71477     COLOUR GLASS. STAR T-LIGHT HOLDER        12
14941     537626     22492               MINI PAINT SET VINTAGE         36
14942     537626     22771   CLEAR DRAWER KNOB ACRYLIC EDWARDIAN        12
…            …        …                                    …        …
428999    573511     22196          SMALL HEART MEASURING SPOONS        24
429000    573511     22195          LARGE HEART MEASURING SPOONS        24
429001    573511     20719               WOODLAND CHARLOTTE BAG        10
429002    573511     23162                  REGENCY TEA STRAINER         8
429003    573511     22131      FOOD CONTAINER SET 3 LOVE HEART          6

               InvoiceDate  UnitPrice  CustomerID  Country InvoiceMonth  \
14938  2010-12-07 14:57:00       2.10     12347.0  Iceland   2010-12-07
14939  2010-12-07 14:57:00       4.25     12347.0  Iceland   2010-12-07
14940  2010-12-07 14:57:00       3.25     12347.0  Iceland   2010-12-07
14941  2010-12-07 14:57:00       0.65     12347.0  Iceland   2010-12-07
14942  2010-12-07 14:57:00       1.25     12347.0  Iceland   2010-12-07
…                      …        …          …       …           …
428999 2011-10-31 12:25:00       0.85     12347.0  Iceland   2011-10-31
429000 2011-10-31 12:25:00       1.65     12347.0  Iceland   2011-10-31
429001 2011-10-31 12:25:00       0.85     12347.0  Iceland   2011-10-31
429002 2011-10-31 12:25:00       3.75     12347.0  Iceland   2011-10-31
429003 2011-10-31 12:25:00       1.95     12347.0  Iceland   2011-10-31

        TotalPrice Purchase_Date  Total_cost
14938         25.2    2010-12-07        25.2
14939         17.0    2010-12-07        17.0
14940         39.0    2010-12-07        39.0
14941         23.4    2010-12-07        23.4
14942         15.0    2010-12-07        15.0
…            …           …         …
428999        20.4    2011-10-31        20.4
```

```
429000         39.6    2011-10-31        39.6
429001          8.5    2011-10-31         8.5
429002         30.0    2011-10-31        30.0
429003         11.7    2011-10-31        11.7

[171 rows x 12 columns]
```

[48]:
```
# Check if the number difference of days from the purchase date in original␣
↪record is same as shown in rfm table.
(current_date - data[data.CustomerID == rfm_table.index[0]].iloc[0].
↪Purchase_Date).days == rfm_table.iloc[0,0]
```

[48]:
```
True
```

[49]:
```
# RFM Quantiles
quantiles = rfm_table.quantile(q=[0.25,0.5,0.75])
quantiles
```

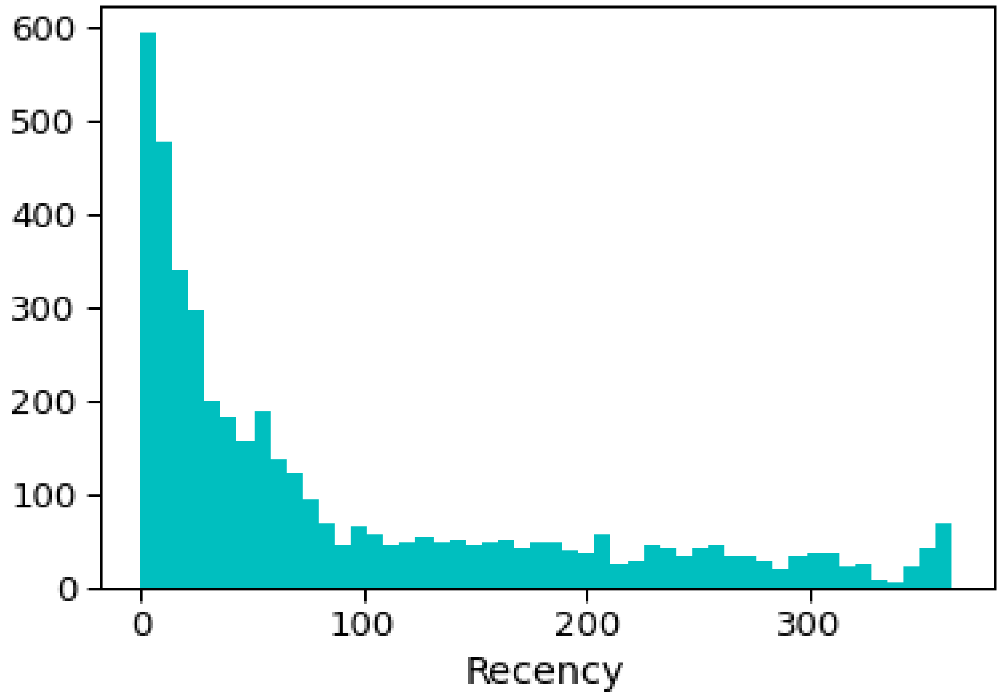
[49]:
```
      Recency  Frequency  Monetary
0.25     15.0        1.0   288.755
0.50     48.0        3.0   628.780
0.75    144.0        5.0  1545.905
```

[50]:
```
# Let's convert quartile information into a dictionary so that cutoffs can be␣
↪picked up.
quantiles=quantiles.to_dict()
quantiles
rfm_table
```

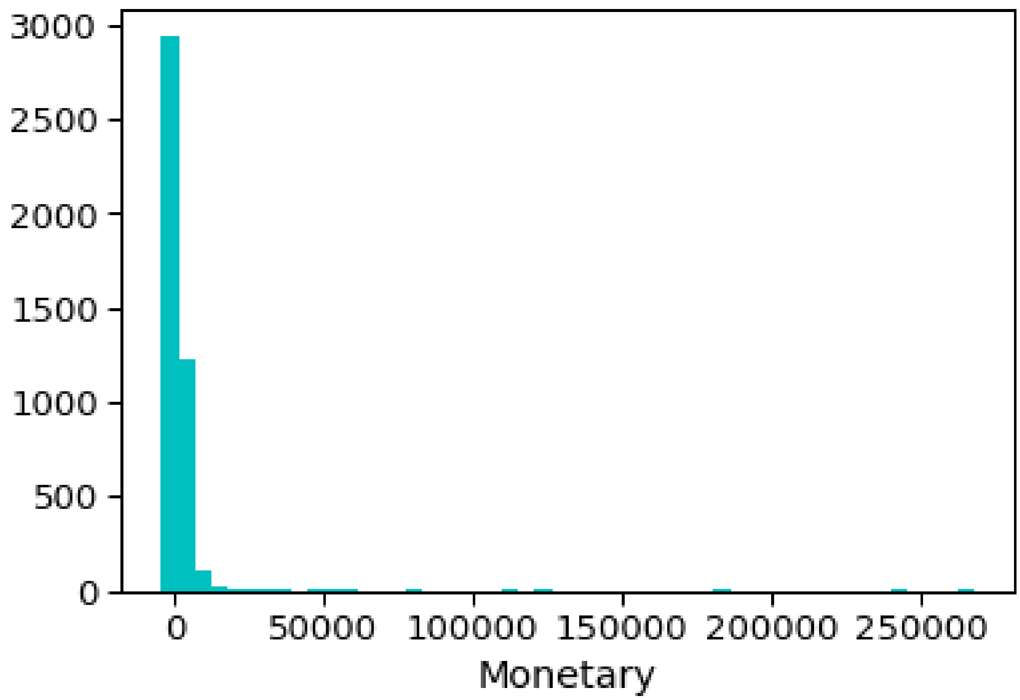
[50]:
```
            Recency  Frequency  Monetary
CustomerID
12346.0         316          2      0.00
12347.0          30          6   4085.18
12348.0          66          4   1797.24
12349.0           9          1   1757.55
12350.0         301          1    334.40
…             …        …        …
18280.0         268          1    180.60
18281.0         171          1     80.82
18282.0         113          2     98.76
18283.0           0         15   1837.53
18287.0          33          3   1837.28

[4331 rows x 3 columns]
```

[51]:
```
#Let us visualize the histogram charts for Recency, Frequency and Monetary
plt.hist(rfm_table.Recency, bins = 50, color='c')
```

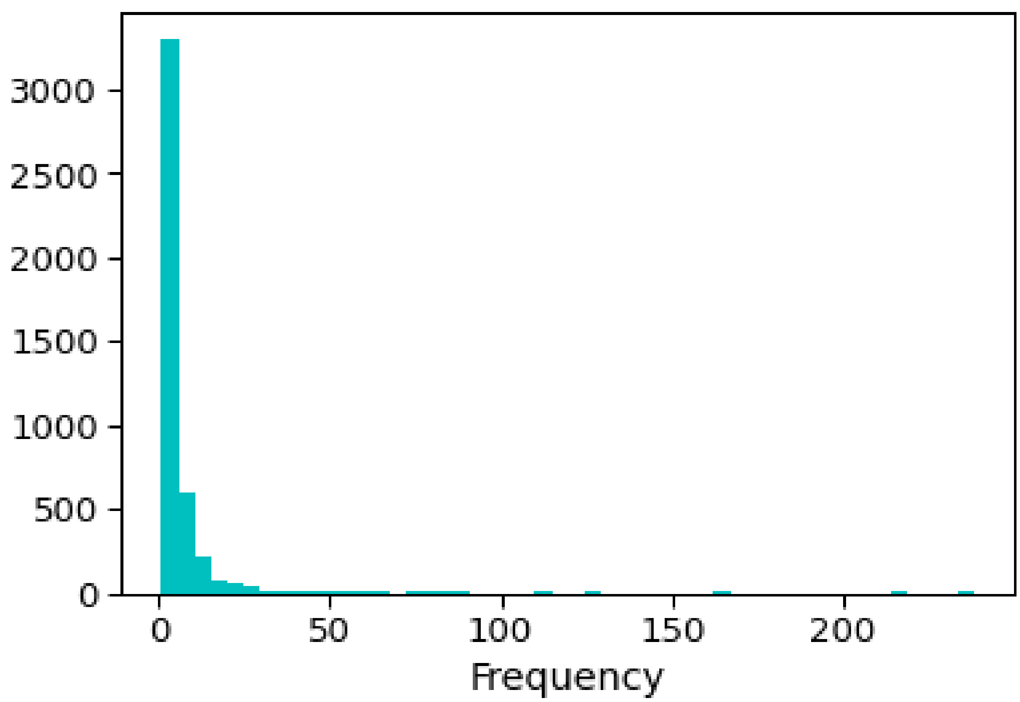
```
plt.xlabel('Recency')
plt.show()
```



[52]:
```
plt.hist(rfm_table.Monetary, bins = 50, color='c')
plt.xlabel('Monetary')
plt.show()
```



[53]:
```
plt.hist(rfm_table.Frequency, bins = 50, color='c')
plt.xlabel('Frequency')
plt.show()
```



Creation of RFM Segments We will create two segmentation classes since, high recency is bad, while high frequency and monetary value is good

[54]:
```
# Arguments (x = value, p = recency, monetary_value, frequency, d = quantiles␣
↪dict)
def RScore(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
# Arguments (x = value, p = recency, monetary_value, frequency, k = quantiles␣
↪dict)
def FMScore(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
rfm_segment = rfm_table.copy()
rfm_segment['R_Quartile'] = rfm_segment['Recency'].apply(RScore,␣
↪args=('Recency',quantiles,))
rfm_segment['F_Quartile'] = rfm_segment['Frequency'].apply(FMScore,␣
↪args=('Frequency',quantiles,))
rfm_segment['M_Quartile'] = rfm_segment['Monetary'].apply(FMScore,␣
↪args=('Monetary',quantiles,))
```

[55]:
```
rfm_segment['RFMScore'] = rfm_segment.R_Quartile.map(str) \
                            + rfm_segment.F_Quartile.map(str) \
                            + rfm_segment.M_Quartile.map(str)
rfm_segment.head()
```

[55]:
```
            Recency  Frequency  Monetary  R_Quartile  F_Quartile  M_Quartile  \
CustomerID
12346.0         316          2      0.00           1           2           1
12347.0          30          6   4085.18           3           4           4
12348.0          66          4   1797.24           2           3           4
12349.0           9          1   1757.55           4           1           4
12350.0         301          1    334.40           1           1           2
```

```
            RFMScore
CustomerID
12346.0          121
12347.0          344
12348.0          234
12349.0          414
12350.0          112
```

[56]:
```
# Reset the index to create a customer_ID column
rfm_segment.reset_index(inplace=True)
```

[57]:
```
import itertools

# Highest frequency as well as monetary value with least recencycy
platinum_customers = ['444', '443']
print ("Platinum Customers                     : {}".format(platinum_customers))

# Get all combinations of [1, 2, 3,4] and length 2
big_spenders_comb =  itertools.product([1, 2, 3,4],repeat = 2)
# Print the obtained combinations
big_spenders = []
for i in list(big_spenders_comb):
    item = (list(i))
    item.append(4)
    big_spenders.append( ("".join(map(str,item))))
print ("Big Spenders                           : {}".format(big_spenders))

#High-spending New Customers - This group consists of those customers in 1-4-1␣
 ↪and 1-4-2.
#These are customers who transacted only once, but very recently and they spent␣
 ↪a lot

high_spend_new_customers = ['413', '314' ,'313','414']
print ("High Spend New Customers               : {}".
 ↪format(high_spend_new_customers))


lowest_spending_active_loyal_customers_comb =  itertools.product([ 3,4], repeat␣
 ↪= 2)
lowest_spending_active_loyal_customers = []
for i in list(lowest_spending_active_loyal_customers_comb):
    item = (list(i))
    item.append(1)
    lowest_spending_active_loyal_customers.append( ("".join(map(str,item))))
print ("Lowest Spending Active Loyal Customers : {}".
 ↪format(lowest_spending_active_loyal_customers))
```

```
recent_customers_comb =  itertools.product([ 2,3,4], repeat = 2)
recent_customers = []
for i in list(recent_customers_comb):
    item = (list(i))
    item.insert(0,4)
    recent_customers.append( ("".join(map(str,item))))
print ("Recent Customers                       : {}".format(recent_customers))




almost_lost = ['244', '234', '243', '233']        #  Low R - Customer's␣
↪shopping less often now who used to shop a lot
print ("Good Customers Almost Lost             : {}".format(almost_lost))

churned_best_customers = ['144', '134' ,'143','133']
print ("Churned Best Customers                 : {}".
↪format(churned_best_customers))


lost_cheap_customers = ['122','111' ,'121','112','221','212' ,'211'] #␣
↪Customer's shopped long ago but with less frequency and monetary value
print ("Lost Cheap Customers                   : {}".
↪format(lost_cheap_customers))
```

```
Platinum Customers                       : ['444', '443']
Big Spenders                             : ['114', '124', '134', '144', '214',
'224', '234', '244', '314', '324', '334', '344', '414', '424', '434', '444']
High Spend New Customers                 : ['413', '314', '313', '414']
Lowest Spending Active Loyal Customers : ['331', '341', '431', '441']
Recent Customers                         : ['422', '423', '424', '432', '433',
'434', '442', '443', '444']
Good Customers Almost Lost               : ['244', '234', '243', '233']
Churned Best Customers                   : ['144', '134', '143', '133']
Lost Cheap Customers                     : ['122', '111', '121', '112', '221',
'212', '211']
```

[58]:
```
# Create a dictionary for each segment to map them against each customer
segment_dict = {
    'Platinum Customers':platinum_customers,
    'Big Spenders':      big_spenders,
    'High Spend New Customers':high_spend_new_customers,
    'Lowest-Spending Active Loyal Customers' :␣
↪lowest_spending_active_loyal_customers ,
    'Recent Customers': recent_customers,
    'Good Customers Almost Lost':almost_lost,
```

```
    'Churned Best Customers':   churned_best_customers,
    'Lost Cheap Customers ': lost_cheap_customers,
}
```

[59]:
```
# Allocate segments to each customer as per the RFM score mapping
def find_key(value):
    for k, v in segment_dict.items():
        if value in v:
            return k
rfm_segment['Segment'] = rfm_segment.RFMScore.map(find_key)

# Allocate all remaining customers to others segment category
rfm_segment.Segment.fillna('others', inplace=True)
rfm_segment.sample(10)
```

[59]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile
4115	17986.0	47	1	20.80	3	1
1458	14339.0	225	4	289.91	1	3
1552	14467.0	8	4	596.26	4	3
1397	14246.0	101	3	1474.06	2	2
372	12823.0	65	5	1759.50	2	3
3235	16767.0	21	11	5575.56	3	4
2108	15227.0	27	5	1219.40	3	3
1204	13983.0	20	13	2590.46	3	4
1314	14133.0	120	5	590.34	2	3
2296	15483.0	152	4	225.26	1	3

	M_Quartile	RFMScore	Segment
4115	1	311	others
1458	2	132	others
1552	2	432	Recent Customers
1397	3	223	others
372	4	234	Big Spenders
3235	4	344	Big Spenders
2108	3	333	others
1204	4	344	Big Spenders
1314	2	232	others
2296	1	131	others

[60]:
```
# Best Customers who's recency, frequency as well as monetary attribute is␣
↪highest.
rfm_segment[rfm_segment.RFMScore=='444'].sort_values('Monetary',␣
↪ascending=False).head()
```

[60]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile
1685	14646.0	7	74	267761.00	4	4
4193	18102.0	2	59	244952.95	4	4

```
3722     17450.0        1        54  185759.77           4           4
1876     14911.0        0       238  125482.36           4           4
54       12415.0       15        26  123725.45           4           4

      M_Quartile RFMScore             Segment
1685           4      444  Platinum Customers
4193           4      444  Platinum Customers
3722           4      444  Platinum Customers
1876           4      444  Platinum Customers
54             4      444  Platinum Customers
```

[61]:

```
# Biggest spenders
rfm_segment[rfm_segment.RFMScore=='334'].sort_values('Monetary',␣
 ↪ascending=False).head()
```

[61]:

```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
2765     16126.0       20          4   6287.77           3           3
12       12359.0       48          5   6274.23           3           3
727      13316.0       28          5   5570.69           3           3
2894     16303.0       16          4   5305.83           3           3
2868     16258.0       36          5   5203.51           3           3

      M_Quartile RFMScore       Segment
2765           4      334  Big Spenders
12             4      334  Big Spenders
727            4      334  Big Spenders
2894           4      334  Big Spenders
2868           4      334  Big Spenders
```

[62]:

```
# customers that you must retain are those whose monetary and frequency was␣
 ↪high but recency reduced quite a lot recently
rfm_segment[rfm_segment.RFMScore=='244'].sort_values('Monetary',␣
 ↪ascending=False).head()
```

[62]:

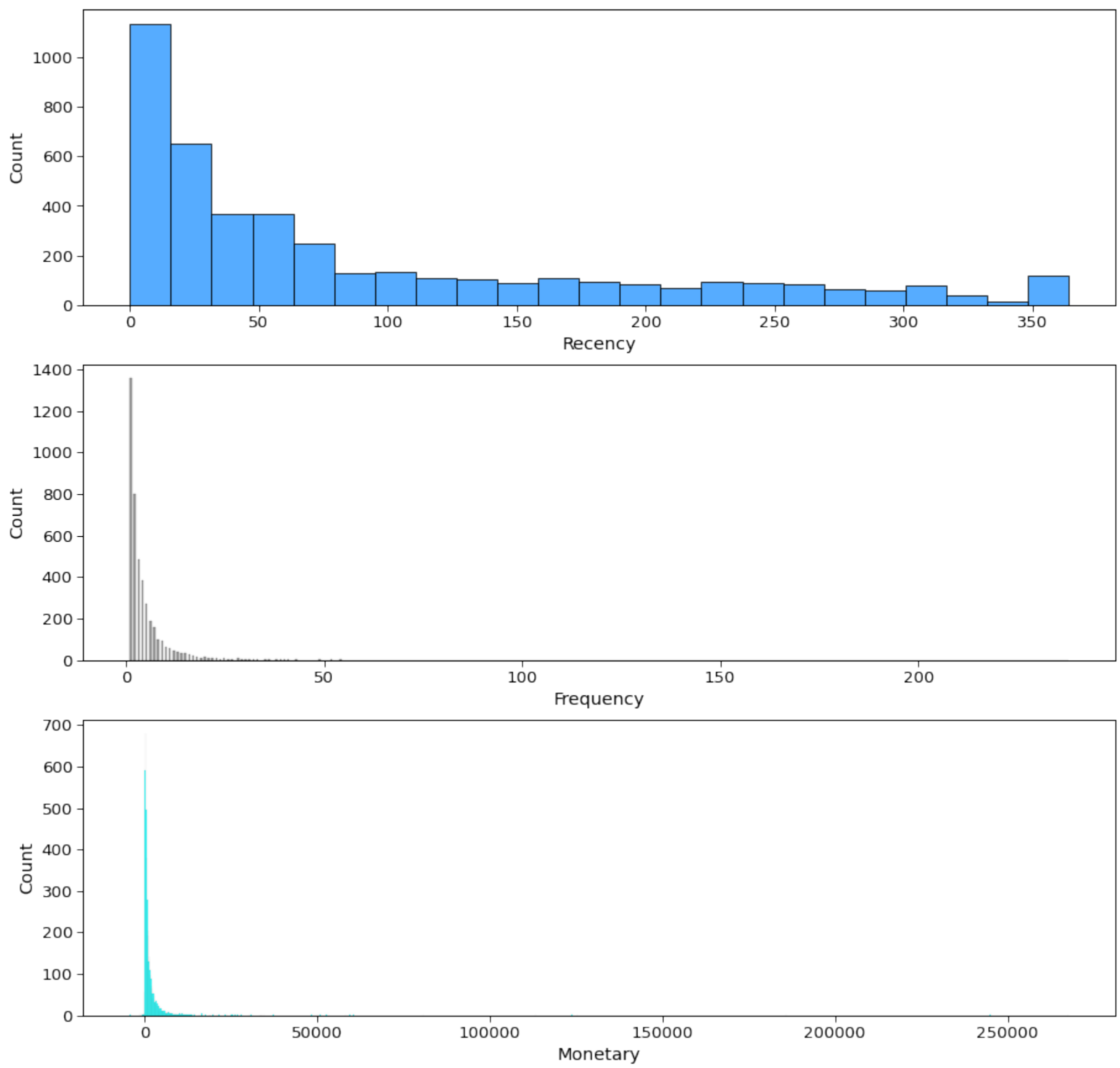
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
457      12939.0       55          8  11581.80           2           4
49       12409.0       69          7  11056.93           2           4
2807     16180.0       91         10  10217.48           2           4
1776     14769.0       68          9  10041.86           2           4
3215     16745.0       77         18   7157.10           2           4

      M_Quartile RFMScore       Segment
457            4      244  Big Spenders
49             4      244  Big Spenders
2807           4      244  Big Spenders
1776           4      244  Big Spenders
3215           4      244  Big Spenders
```

[63]:
```
rfm_segment.to_csv('RFM Segment.csv')
```

[64]:
```
rfm_segment.Segment.value_counts()
rfm_segment.Recency
```

[64]:
```
0       316
1        30
2        66
3         9
4       301
       …
4326    268
4327    171
4328    113
4329      0
4330     33
Name: Recency, Length: 4331, dtype: int64
```

[65]:
```
# Distribution plot
fig, axes = plt.subplots(3, 1, figsize=(15, 15))
sns.histplot(rfm_table.Recency , color="dodgerblue", ax=axes[0],␣
 ↪label='Recency')
sns.histplot(rfm_table.Frequency , color="grey", ax=axes[1], label='Frequency')
sns.histplot(rfm_table.Monetary , color="cyan", ax=axes[2], label='Monetary')
plt.show();
```



[66]:
```python
# Let's describe the table to see if there are any negative values
rfm_table.describe()
```

[66]:

	Recency	Frequency	Monetary
count	4331.000000	4331.000000	4331.000000
mean	90.277303	4.910875	1832.597551
std	99.389069	9.025901	7944.283177
min	0.000000	1.000000	-4287.630000
25%	15.000000	1.000000	288.755000
50%	48.000000	3.000000	628.780000
75%	144.000000	5.000000	1545.905000
max	364.000000	238.000000	267761.000000

[67]:
```python
# Create a copy of rfm table
rfm_table_scaled = rfm_table.copy()

# Shift all values in the column by adding absolute of minimum value to each␣
↪value, thereby making each value positive.
rfm_table_scaled.Monetary = rfm_table_scaled.Monetary + abs(rfm_table_scaled.
↪Monetary.min()) + 1
rfm_table_scaled.Recency = rfm_table_scaled.Recency + abs(rfm_table_scaled.
↪Recency.min()) + 1

# Check the summary of new values
rfm_table_scaled.describe()
```

[67]:

	Recency	Frequency	Monetary
count	4331.000000	4331.000000	4331.000000
mean	91.277303	4.910875	6121.227551
std	99.389069	9.025901	7944.283177
min	1.000000	1.000000	1.000000
25%	16.000000	1.000000	4577.385000
50%	49.000000	3.000000	4917.410000
75%	145.000000	5.000000	5834.535000
max	365.000000	238.000000	272049.630000

[68]:
```python
# Transform the data before K-Means clustering
from sklearn.preprocessing import StandardScaler

# Taking log first because normalization forces data for negative values
log_df = np.log(rfm_table_scaled)

# Normalize the data for uniform averages and means in the distribution.
scaler = StandardScaler()
normal_df = scaler.fit_transform(log_df)
normal_df = pd.DataFrame(data=normal_df, index=rfm_table.index,␣
↪columns=rfm_table.columns)
```

[69]:
```python
normal_df
```

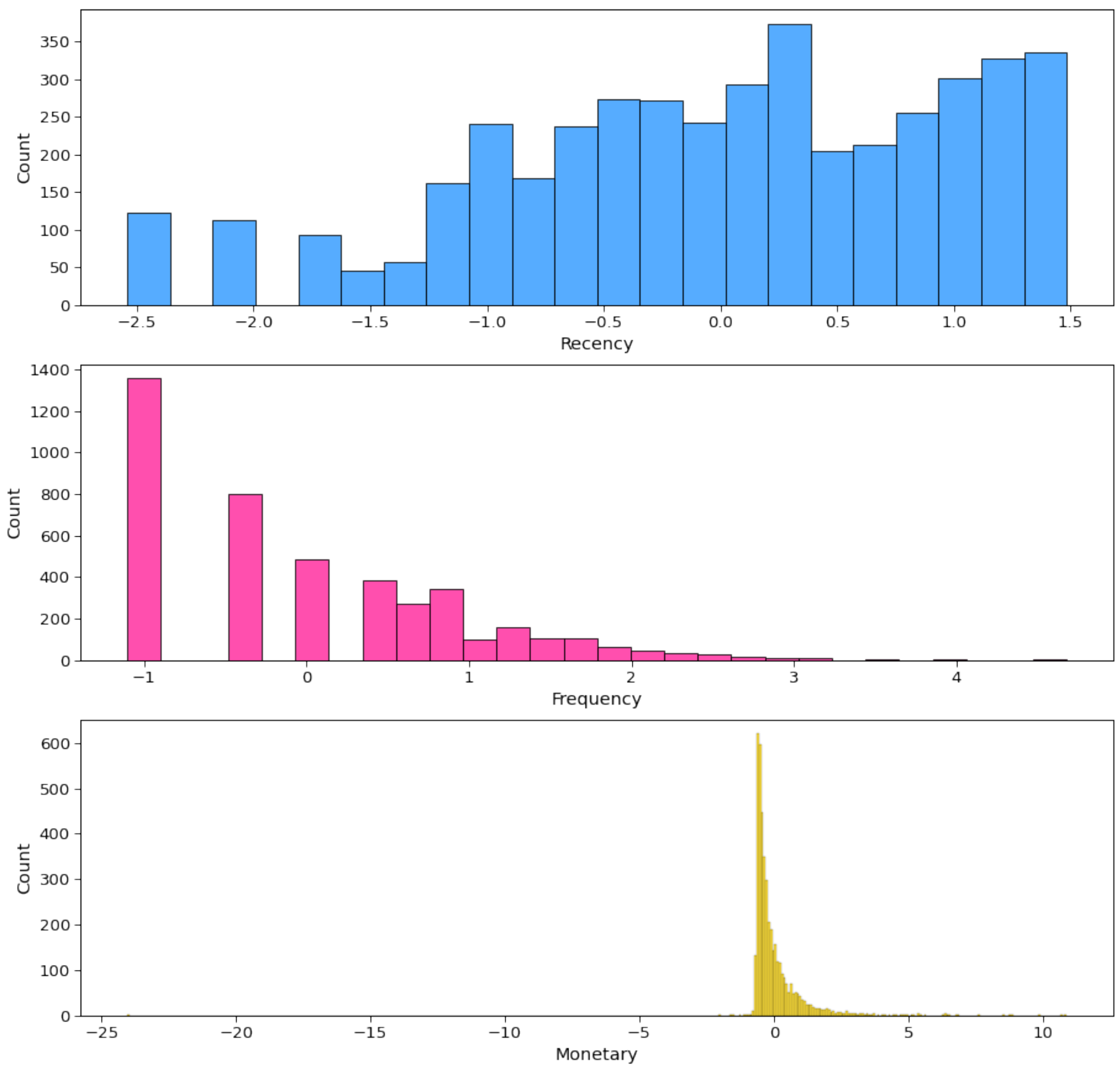
[69]:

	Recency	Frequency	Monetary
CustomerID			
12346.0	1.386976	-0.369465	-0.687546
12347.0	-0.198501	0.790665	1.180610
12348.0	0.327082	0.362496	0.289615
12349.0	-0.970062	-1.101426	0.271348
12350.0	1.353919	-1.101426	-0.477924
…	…	…	…
18280.0	1.275007	-1.101426	-0.572384
18281.0	0.970027	-1.101426	-0.635422

```
18282.0     0.689543  -0.369465 -0.623983
18283.0    -2.540311   1.758265  0.308037
18287.0    -0.135507   0.058705  0.307923

[4331 rows x 3 columns]
```

Visualize the data after applying logarithmic transformation on scaled data. Observe that the skewness is reduced

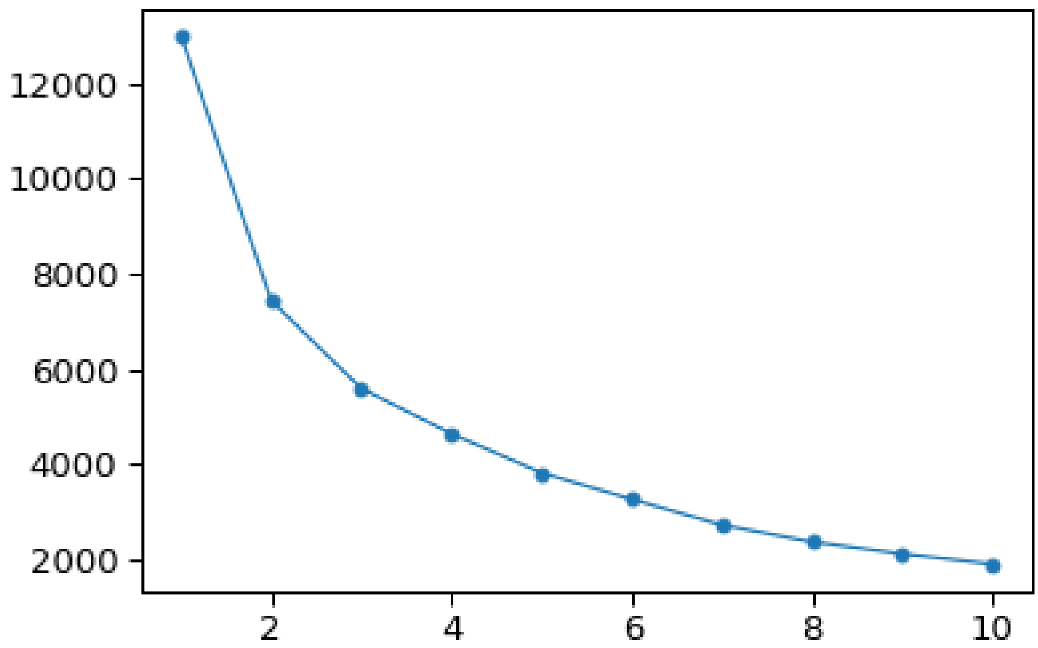
[70]:
```
# Distribution plot
fig, axes = plt.subplots(3, 1, figsize=(15, 15))
sns.histplot(normal_df.Recency , color="dodgerblue", ax=axes[0],␣
 ↪label='Recency')
sns.histplot(normal_df.Frequency , color="deeppink", ax=axes[1],␣
 ↪label='Frequency')
sns.histplot(normal_df.Monetary , color="gold", ax=axes[2], label='Monetary')
# plt.xlim(50,75);
plt.show();
```



[71]:
```python
# find WCSS
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++')
    kmeans.fit(normal_df)
    wcss.append(kmeans.inertia_)

# plot elbow graph
plt.plot(range(1,11),wcss,marker='o');
```

```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
```

```
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
```



Let us copy this data to an Excel sheet. This will be used in Tableau to determine the elbow Plot.

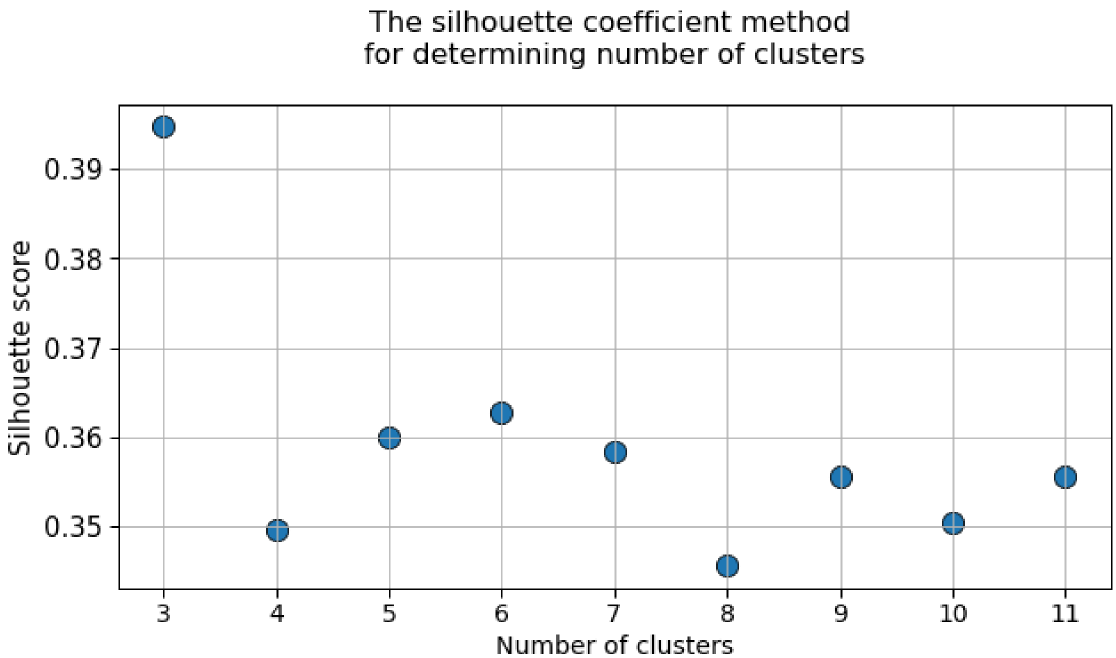
[72]:
```
ElbowPlot = pd.DataFrame({'Cluster': range(1,11) , 'SSE': wcss})
ElbowPlot.to_csv('Elbow Plot Data.csv')
```

[73]:
```
from sklearn.metrics import silhouette_score
wcss_silhouette = []
for i in range(3,12):
    km = KMeans(n_clusters=i, random_state=0,init='k-means++').fit(normal_df)
    preds = km.predict(normal_df)
    silhouette = silhouette_score(normal_df,preds)
    wcss_silhouette.append(silhouette)
    print("Silhouette score for number of cluster(s) {}: {}".
 ↪format(i,silhouette))

plt.figure(figsize=(10,5))
plt.title("The silhouette coefficient method \nfor determining number of␣
 ↪clusters\n",fontsize=16)
plt.scatter(x=[i for i in range(3,12)],y=wcss_silhouette,s=150,edgecolor='k')
plt.grid(True)
plt.xlabel("Number of clusters",fontsize=14)
plt.ylabel("Silhouette score",fontsize=15)
plt.xticks([i for i in range(3,12)],fontsize=14)
plt.yticks(fontsize=15)
plt.show()
```

```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 3: 0.3947454614514733

/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 4: 0.3496682694835896

/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 5: 0.3600416387004347

/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 6: 0.36266394645573724

/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 7: 0.35838280468222317

/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 8: 0.34562891059349565

/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 9: 0.35553057886310274

/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette score for number of cluster(s) 10: 0.35047985506205726
```

```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
```

```
Silhouette score for number of cluster(s) 11: 0.3555508957391139
```



Here we can clearly see that optimum number of cluster should be 4 not 2 or 3. Because that is the only point after which the mean cluster distance looks to be plateaued after a steep downfall. So we will assume the 4 number of clusters as best for grouping of customer segments.

Now let's apply K-Means on 4 clusters to segregate the customer base

[74]:
```
kmeans = KMeans(n_clusters=4, random_state=1, init='k-means++')
kmeans.fit(normal_df)
cluster_labels = kmeans.labels_
```

```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
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
  super()._check_params_vs_input(X, default_n_init=10)
```

[75]:
```
kmeans
```

[75]:
```
KMeans(n_clusters=4, random_state=1)
```

[76]:
```python
print(f"Shape of cluster label array is {cluster_labels.shape}")
print(f"Shape of RFM segment dataframe is {rfm_segment.shape}")
```

```
Shape of cluster label array is (4331,)
Shape of RFM segment dataframe is (4331, 9)
```

[77]:
```python
# Assign the clusters as column to each customer
Cluster_table = rfm_segment.assign(Cluster = cluster_labels)
```

[78]:
```python
# Check counts of records assigned to different clusters
Cluster_table.Cluster.value_counts()
```

[78]:
```
0    1951
3    1173
2    1034
1     173
Name: Cluster, dtype: int64
```

Here we see that most of the customers belong to 0 ,2 and 3 cluster, whereas very less number of customers assigned to 1 cluster, may be possible that those are some of the best customers out of the pool or worst customer, lets checkout the pattern

[79]:
```python
Cluster_table.sample(10)
print ("Platinum customers belong to cluster                      : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='Platinum␣
 ↪Customers']['Cluster'].unique()))
print ("Big Spenders belong to cluster                            : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='Big Spenders']['Cluster'].
 ↪unique()))
print ("High Spend new Customers belong to cluster                : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='High Spend New␣
 ↪Customers']['Cluster'].unique()))
print ("Lowest-Spending Active Loyal Customers belong to cluster  : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='Lowest-Spending Active Loyal␣
 ↪Customers']['Cluster'].unique()))
print ("Recent Customers belong to cluster                        : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='Recent␣
 ↪Customers']['Cluster'].unique()))
print ("Good Customers Almost Lost belong to cluster              : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='Good Customers Almost␣
 ↪Lost']['Cluster'].unique()))
print ("Churned Best Customers belong to cluster                  : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='Churned Best␣
 ↪Customers']['Cluster'].unique()))
print ("Lost Cheap customers belong to cluster                    : {} ".
 ↪format(Cluster_table[Cluster_table['Segment']=='Lost Cheap Customers␣
 ↪']['Cluster'].unique()))
```

```
Platinum customers belong to cluster                              : [3 1]
Big Spenders belong to cluster                                    : [3 2 0 1]
High Spend new Customers belong to cluster                        : [2 0]
Lowest-Spending Active Loyal Customers belong to cluster          : [2 3]
Recent Customers belong to cluster                                : [2 3]
Good Customers Almost Lost belong to cluster                      : [0 3 2]
Churned Best Customers belong to cluster                          : [0 3]
Lost Cheap customers belong to cluster                            : [0 2]
```

Here we can observe that RFM score is very low for customers in 0 & 3 cluster. Comparetivey, customers in 1&2 clusters have high RFM scores along with above average Recency and frequency values.

Let's checkout customers in each cluster more closely

[80]:
```
Cluster_table[Cluster_table.Cluster == 3].sample(5)
```

[80]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScore	Segment	Cluster
2384	15602.0	13	14	902.79	4	4	3	443	Platinum Customers	3
465	12949.0	21	10	4143.02	3	4	4	344	Big Spenders	3
3929	17725.0	7	14	3371.13	4	4	4	444	Platinum Customers	3
3667	17377.0	14	21	3883.32	4	4	4	444	Platinum Customers	3
3951	17758.0	10	8	3408.20	4	4	4	444	Platinum Customers	3

Here it can be seen that the RFM score for Cluster 3 customers with low recency, good frequency and high monetary value, These are the loyal customers to the firm.

[81]:
```
Cluster_table[Cluster_table.Cluster == 2].sample(5)
```

[81]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScore	Segment	Cluster
823	13455.0	15	4	1007.35	4	3	3	433	Recent Customers	2
3375	16956.0	0	2	292.34	4	2	2	422	Recent Customers	2
3429	17038.0	21	1	112.40	3	1	1	311	others	2
1083	13816.0	14	4	555.04	4	3	2	432	Recent Customers	2
4040	17885.0	3	1	156.34	4	1	1	411	others	2

Cluster 2 contains the highest number of customers who accounts for lowest value to the firm because there RFM values are lowest. Most of them are in the lost segment or on the verge of churning out

[82]:
```
Cluster_table[Cluster_table.Cluster == 1].sample(5)
```

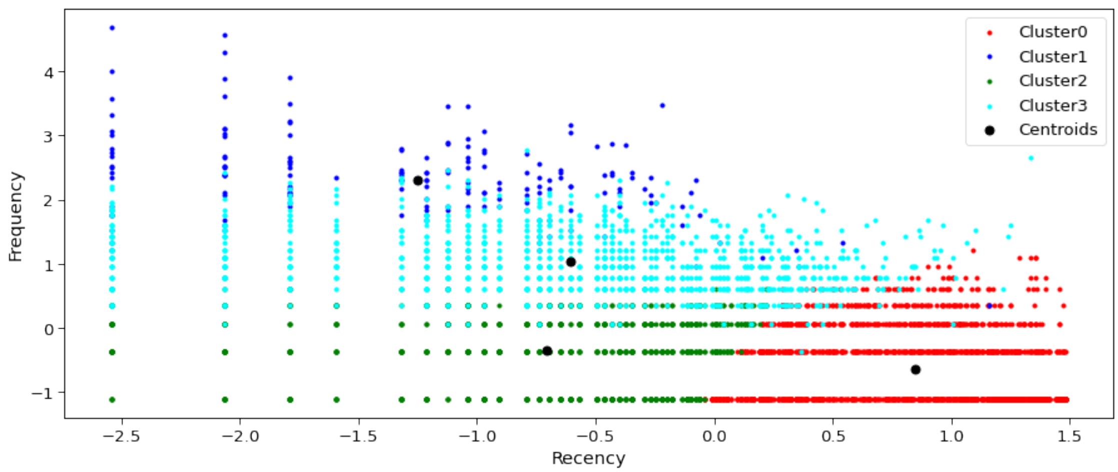
[82]:
```
      CustomerID  Recency  Frequency   Monetary  R_Quartile  F_Quartile  \
2497     15749.0      226          4   21535.90           1           3
1110     13854.0       15         28    7722.74           4           4
3574     17243.0        5         28    7556.17           4           4
1331     14156.0        0         66  113214.59           4           4
65       12428.0       16         12    7877.20           3           4

      M_Quartile RFMScore             Segment  Cluster
2497           4      134        Big Spenders        1
1110           4      444  Platinum Customers        1
3574           4      444  Platinum Customers        1
1331           4      444  Platinum Customers        1
65             4      344        Big Spenders        1
```

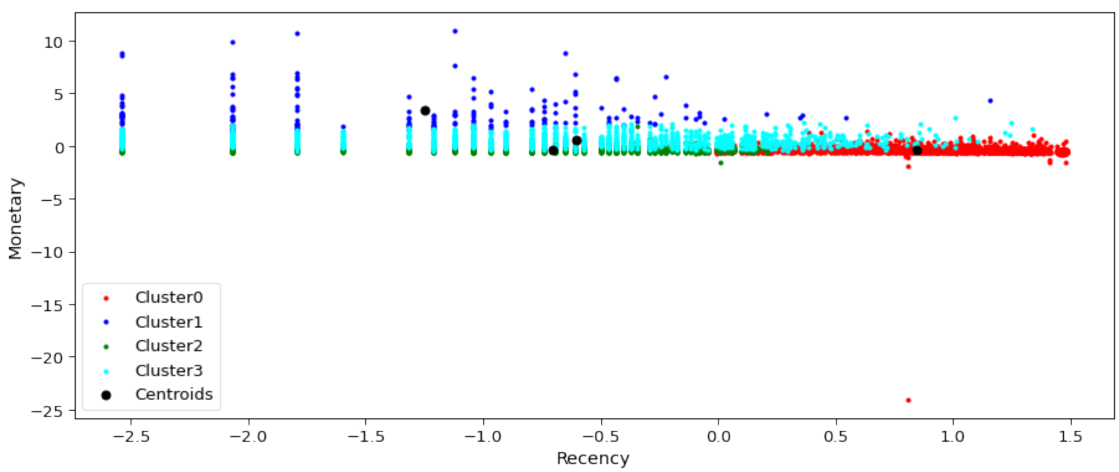
[83]:
```
Cluster_table[Cluster_table.Cluster == 0].sample(5)
```

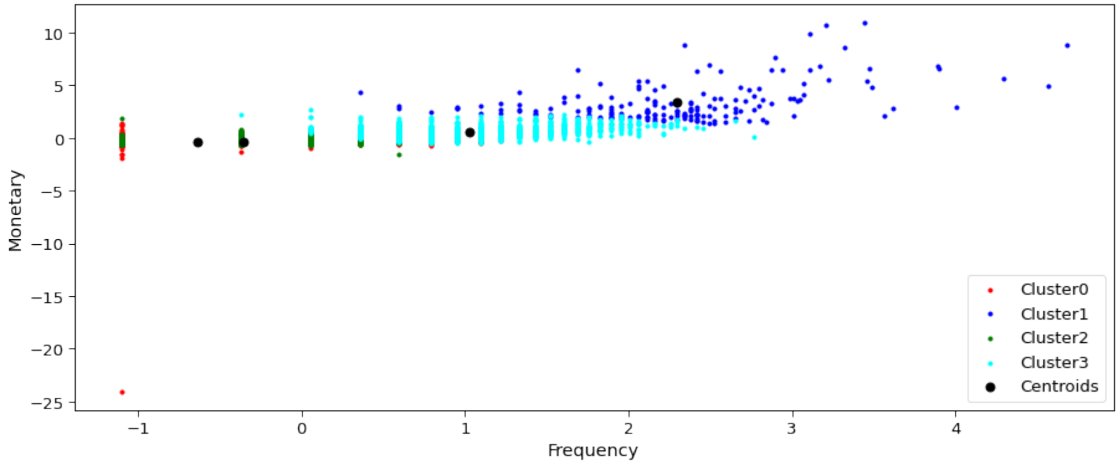
[83]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
2599     15888.0       70          2    141.27           2           2
1008     13715.0      272          8   1051.99           1           4
235      12639.0      229          1    486.10           1           1
2919     16339.0      275          1     94.05           1           1
2799     16171.0       50          1     73.20           2           1

      M_Quartile RFMScore                 Segment  Cluster
2599           1      221     Lost Cheap Customers        0
1008           3      143    Churned Best Customers        0
235            2      112     Lost Cheap Customers        0
2919           1      111     Lost Cheap Customers        0
2799           1      211     Lost Cheap Customers        0
```

[84]:
```
# Plotting two dimesional plots of each attributes respectively.
X = normal_df.iloc[:,0:3].values
count=X.shape[1]
for i in range(0,count):
    for j in range(i+1,count):
        plt.figure(figsize=(15,6));
        plt.scatter(X[cluster_labels == 0, i], X[cluster_labels == 0, j], s =␣
 ↪10, c = 'red', label = 'Cluster0')
        plt.scatter(X[cluster_labels == 1, i], X[cluster_labels == 1, j], s =␣
 ↪10, c = 'blue', label = 'Cluster1')
```

```
        plt.scatter(X[cluster_labels == 2, i], X[cluster_labels == 2, j], s =
↪10, c = 'green', label = 'Cluster2')
        plt.scatter(X[cluster_labels == 3, i], X[cluster_labels == 3, j], s =
↪10, c = 'cyan', label = 'Cluster3')
        plt.scatter(kmeans.cluster_centers_[:,i], kmeans.cluster_centers_[:,j],
↪s = 50, c = 'black', label = 'Centroids')
        plt.xlabel(normal_df.columns[i])
        plt.ylabel(normal_df.columns[j])
        plt.legend()
        plt.show();
```





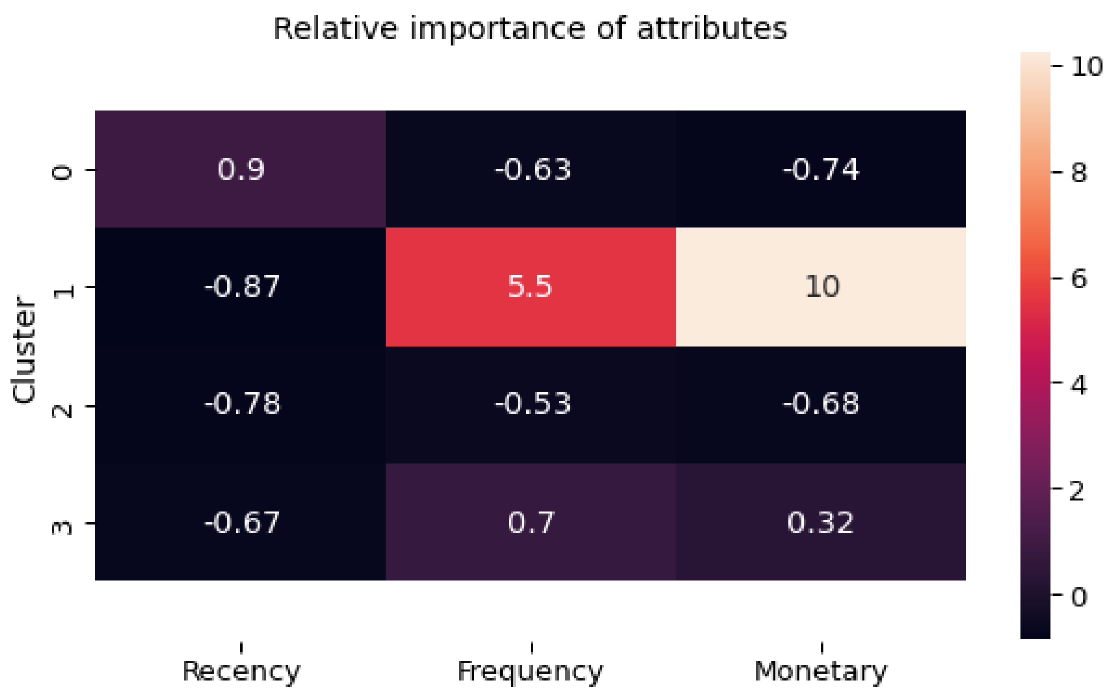


[85]:
```
Cluster_table.to_excel('RFMSegment.xlsx')
```

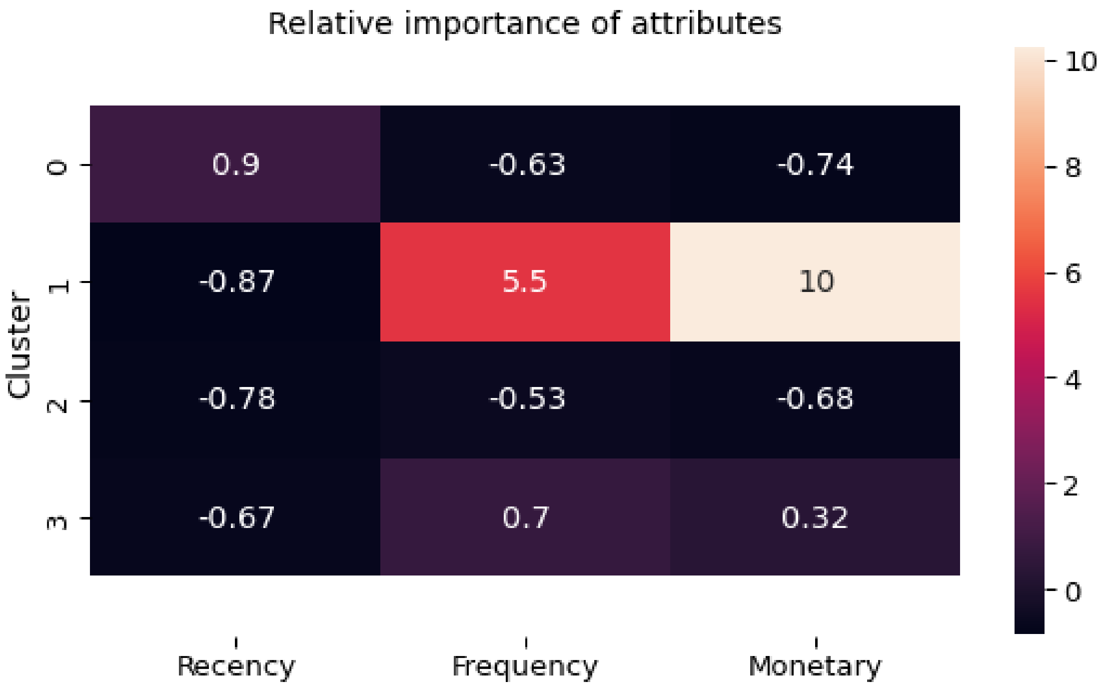
Let's try to visualize this pattern through the help Clusters

[86]:
```
# Assign Cluster labels to RFM table
rfm_table_cluster = rfm_table.assign(Cluster = cluster_labels)

# Average attributes for each cluster
cluster_avg = rfm_table_cluster.groupby(['Cluster']).mean()

# Calculate the population average
population_avg = rfm_table.mean()

# Calculate relative importance of attributes by
relative_imp = cluster_avg / population_avg - 1
```

[87]:
```
plt.figure(figsize=(9, 5))
plt.title('Relative importance of attributes')

ax = sns.heatmap(relative_imp, annot=True) #notation: "annot" not "annote"
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
#plt.tight_layout()
#plt.gcf().subplots_adjust(bottom=0.15)
plt.show();
```



[88]:
```
plt.figure(figsize=(9, 5))
plt.title('Relative importance of attributes')

ax = sns.heatmap(relative_imp, annot=True) #notation: "annot" not "annote"
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
#plt.tight_layout()
#plt.gcf().subplots_adjust(bottom=0.15)
plt.show();
```



[89]:
```
data.to_csv('TableauSource.csv')
```

[ ]:
```
```