Retail_Capstone

October 17, 2022

[ ]:
```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline

# Supress Scientific notation in python
pd.set_option('display.float_format', lambda x: '%.2f' % x)


pd.set_option('display.max_columns', None)

import time
import datetime as dt

warnings.filterwarnings("ignore")
```

[2]:
```
train = pd.read_excel('Online Retail.xlsx', parse_dates=['InvoiceDate'])
train.head()
```

[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.00	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.00	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.00	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.00	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6				

```
4 2010-12-01 08:26:00       3.39    17850.00  United Kingdom
```

[3]:
```
train.shape
```

[3]:
```
(541909, 8)
```

[4]:
```
print(f'Duplicate items in train dataset is {train.duplicated().sum()}')
```

```
Duplicate items in train dataset is 5268
```

[6]:
```
# Remove duplicate items
train = train[~train.duplicated()]
```

[8]:
```
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()
        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(columns = {0 :␣
↪'Missing Values', 1 : '% of Total Values'})
        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns =␣
↪mis_val_table_ren_columns[mis_val_table_ren_columns.iloc[:,1] != 0].
↪sort_values('% of Total Values', ascending=False).round(1)
        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.
↪\n"+"There are " + str(mis_val_table_ren_columns.shape[0]) +" columns that␣
↪have missing values.")
        # Return the dataframe with missing information
        return mis_val_table_ren_columns
```

[9]:
```
missing_values_table(train)
```

```
Your selected dataframe has 8 columns.
There are 2 columns that have missing values.
```

[9]:

	Missing Values	% of Total Values
CustomerID	135037	25.20
Description	1454	0.30

[10]:
```
Invoice_list = train[train.CustomerID.isnull()]['InvoiceNo'].tolist()
```

[11]:
```
len(train[train.InvoiceNo.isin(Invoice_list)])
```

[11]:
```
135037
```

[12]:
```
rfm_train = train[train.CustomerID.notnull()].copy()
```

[13]:
```
rfm_train.CustomerID = (rfm_train.CustomerID).astype(int)
```

[14]:
```
missing_values_table(rfm_train) # Train
```

```
Your selected dataframe has 8 columns.
There are 0 columns that have missing values.
```

[14]:
```
Empty DataFrame
Columns: [Missing Values, % of Total Values]
Index: []
```

[15]:
```
desc_df = rfm_train[~rfm_train.InvoiceNo.str.contains('C', na=False)]
```

[16]:
```
desc_df['Total_cost'] = rfm_train.Quantity * rfm_train.UnitPrice
```

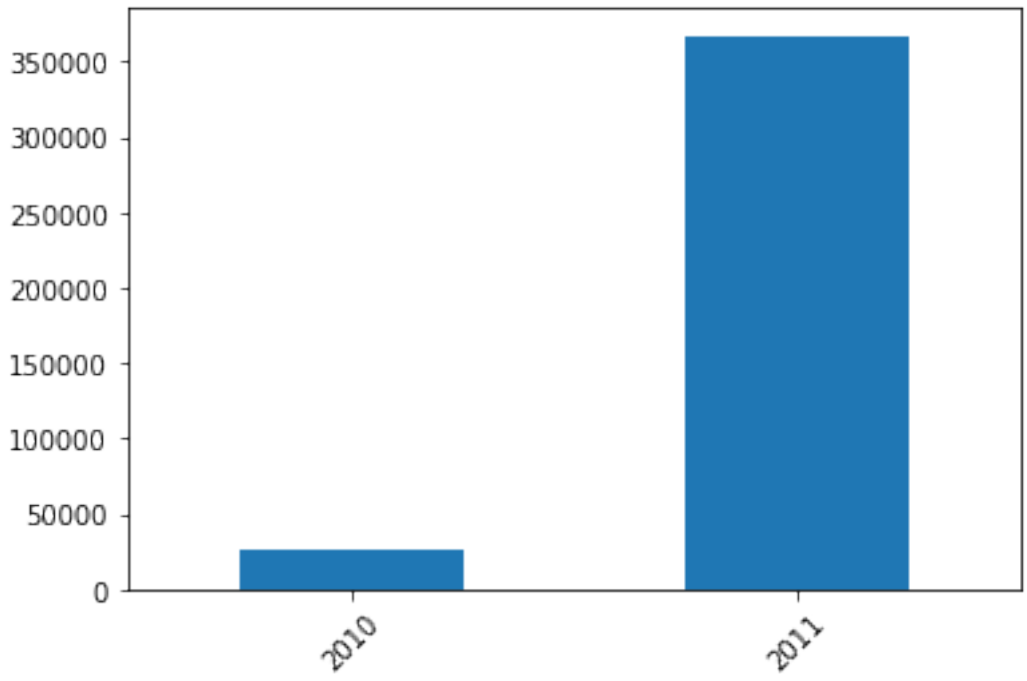
[17]:
```
# Check the oldest and latest date in the dataset.
print(f'Oldest date is - {desc_df.InvoiceDate.min()}\n')
print(f'Latest date is - {desc_df.InvoiceDate.max()}')
```

```
Oldest date is - 2010-12-01 08:26:00

Latest date is - 2011-12-09 12:50:00
```

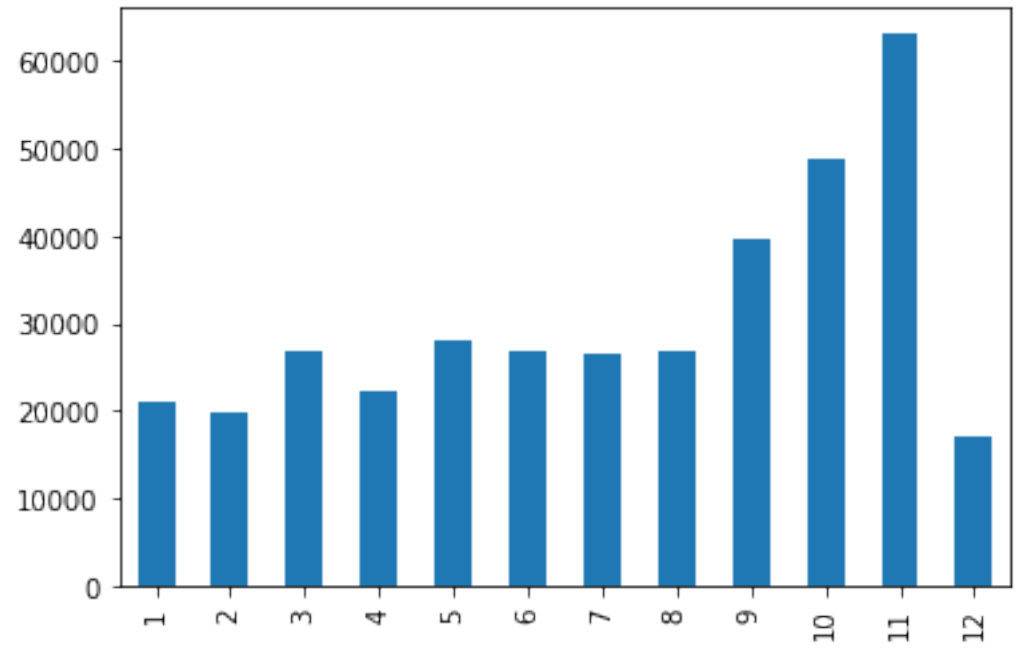
[18]:
```
desc_df.Country.value_counts(normalize=True).head(10).mul(100).round(1).
↪astype(str) + '%'
```

[18]:
```
United Kingdom    88.9%
Germany            2.3%
France             2.1%
EIRE               1.8%
Spain              0.6%
Netherlands        0.6%
Belgium            0.5%
Switzerland        0.5%
Portugal           0.4%
Australia          0.3%
Name: Country, dtype: object
```

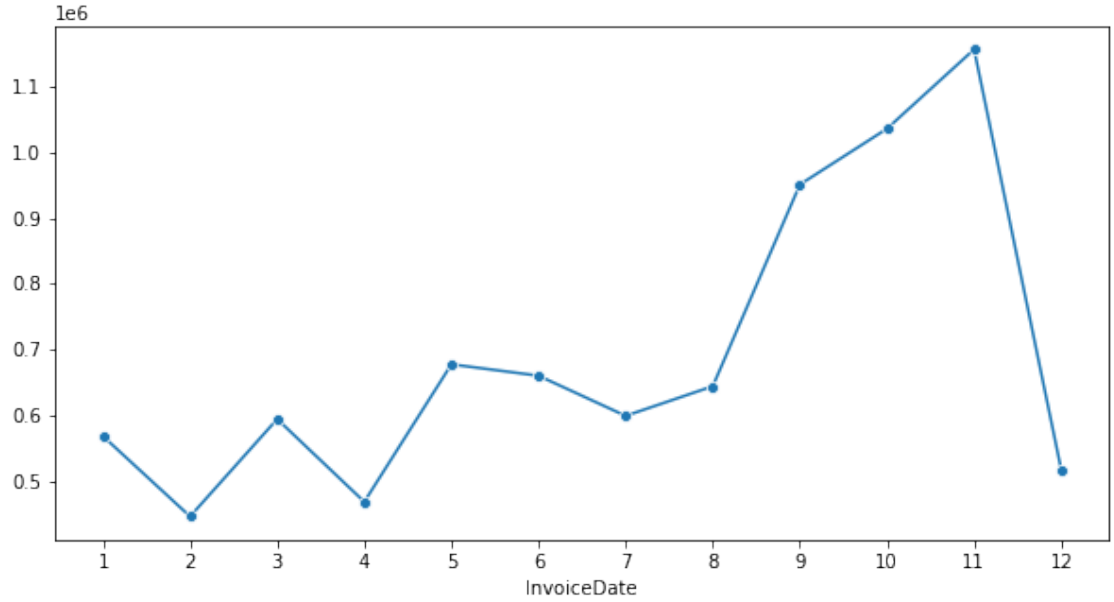
[19]:
```
desc_df.InvoiceDate.dt.year.value_counts(sort=False).plot(kind='bar', rot=45);
```



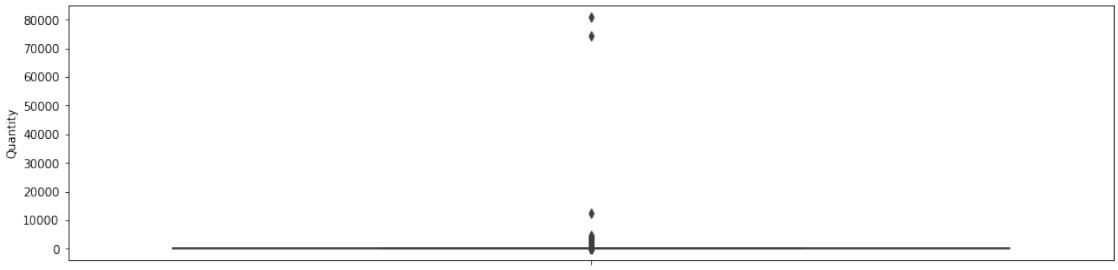
[20]:
```
desc_df[desc_df.InvoiceDate.dt.year==2011].InvoiceDate.dt.month.
↪value_counts(sort=False).plot(kind='bar');
```



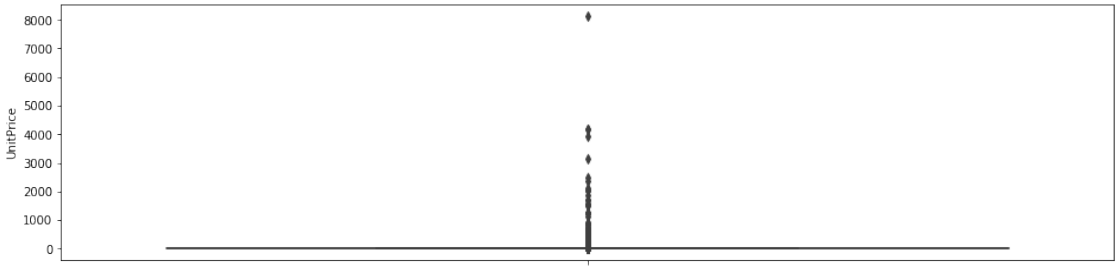
[21]:
```
monthly_gross = desc_df[desc_df.InvoiceDate.dt.year==2011].groupby(desc_df.
↪InvoiceDate.dt.month).Total_cost.sum()
plt.figure(figsize=(10,5))
sns.lineplot(y=monthly_gross.values,x=monthly_gross.index, marker='o');
plt.xticks(range(1,13))
plt.show();
```



[22]:
```
plt.figure(figsize=(16,4))
sns.boxplot(y='Quantity', data=desc_df, orient='h');
```



[23]:
```
plt.figure(figsize=(16,4))
sns.boxplot(y='UnitPrice', data=desc_df, orient='h');
```

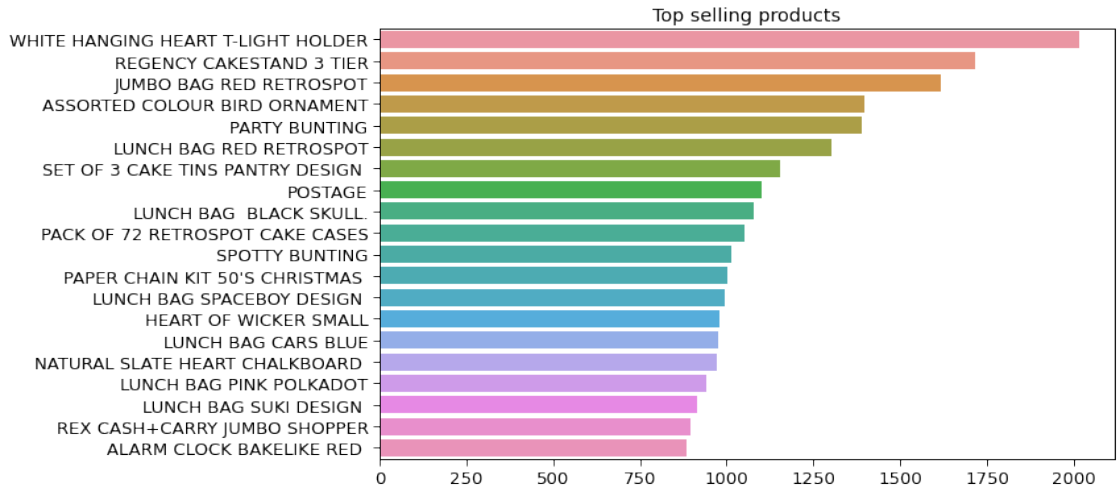


[24]:
```
desc_df.head()
```

[24]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_cost
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom	15.30
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	20.34

[25]:
```
# Let's visualize some top products from the whole range.
top_products = desc_df['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();
```



[26]:
```
%%html
<div class='tableauPlaceholder' id='viz1574249006038' style='position:␣
↪relative'><noscript><a href='#'><img alt=' ' src='https:&#47;&#47;public.
↪tableau.com&#47;static&#47;images&#47;Mu&#47;
↪Multinationonlineretailstore&#47;OnlineStoreDashboard&#47;1_rss.png'␣
↪style='border: none' /></a></noscript><object class='tableauViz' ␣
↪style='display:none;'><param name='host_url' value='https%3A%2F%2Fpublic.
↪tableau.com%2F' /> <param name='embed_code_version' value='3' /> <param␣
↪name='site_root' value='' /><param name='name'␣
↪value='Multinationonlineretailstore&#47;OnlineStoreDashboard' /><param␣
↪name='tabs' value='no' /><param name='toolbar' value='yes' /><param␣
↪name='static_image' value='https:&#47;&#47;public.tableau.com&#47;static&#47;
↪images&#47;Mu&#47;Multinationonlineretailstore&#47;OnlineStoreDashboard&#47;
↪1.png' /> <param name='animate_transition' value='yes' /><param␣
↪name='display_static_image' value='yes' /><param name='display_spinner'␣
↪value='yes' /><param name='display_overlay' value='yes' /><param␣
↪name='display_count' value='yes' /></object></div>                <script␣
↪type='text/
↪javascript'>                    var divElement = document.getElementById('viz1574
```

```
<IPython.core.display.HTML object>
```

[27]:
```
cohort = rfm_train.copy()
```

[28]:
```
def get_month(x):
    return dt.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')
```

[29]:
```
def get_date_int(df, column):
    year = df[column].dt.year
    month = df[column].dt.month
    return year, month
```

[30]:
```
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')
```

[31]:
```
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
```

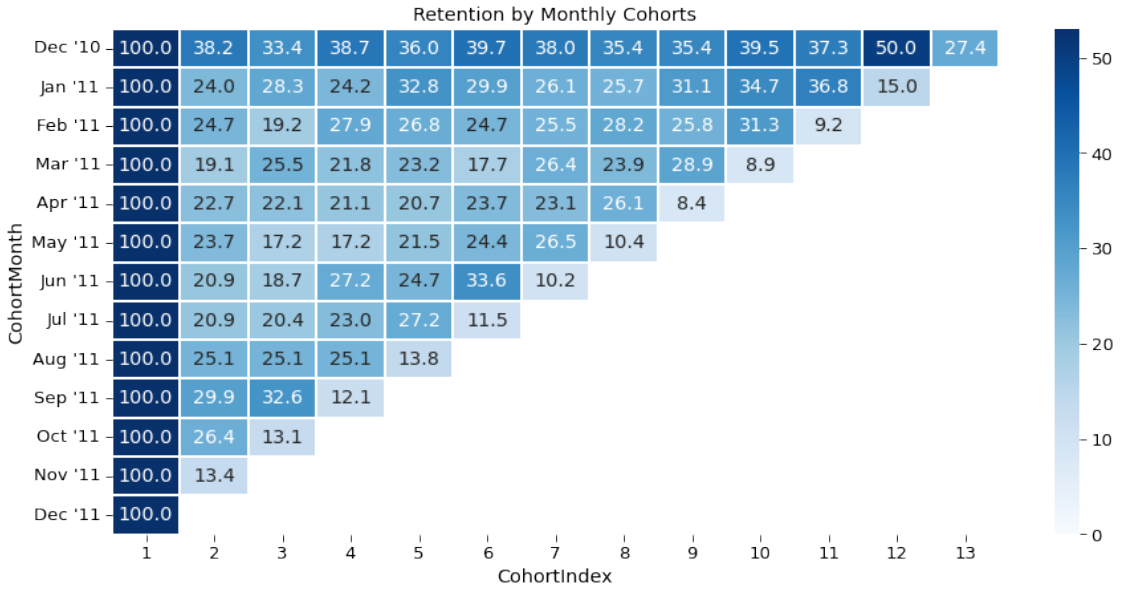
[32]:
```
cohort.head()
```

[32]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth	CohortIndex
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom	2010-12-01	2010-12-01	1
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12-01	1
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom	2010-12-01	2010-12-01	1
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12-01	1
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12-01	1

[33]:
```
grouping = cohort.groupby(['CohortMonth', 'CohortIndex'])
```

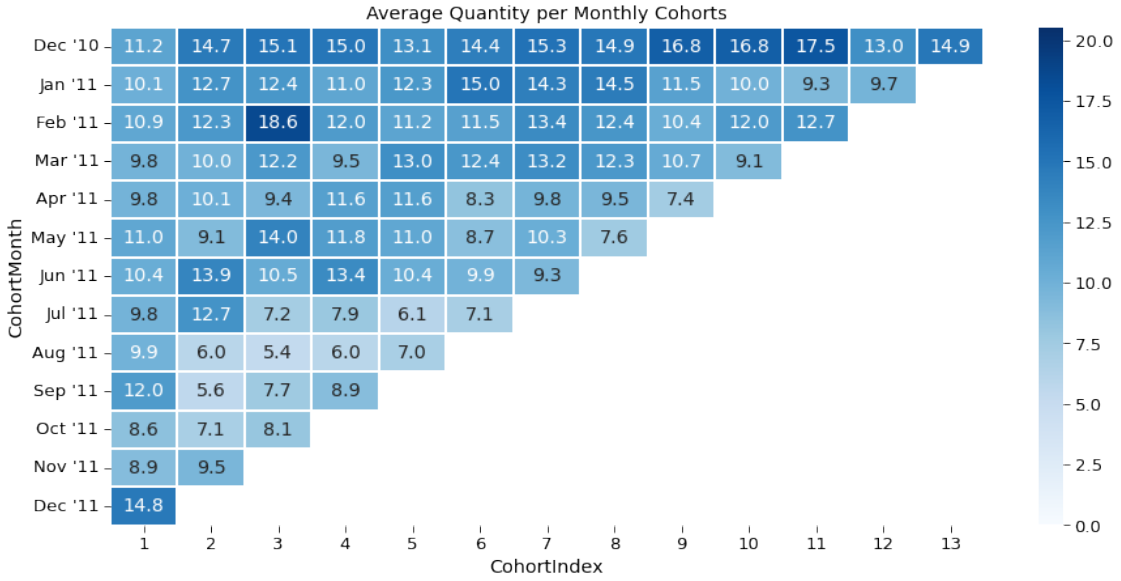
[34]:
```
# Count the number of unique values per customer ID
cohort_data = grouping['CustomerID'].apply(pd.Series.nunique).reset_index()

# 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
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=(15,7))

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

# Create the heatmap
sns.heatmap(data=retention,
            annot = True,
            cmap = "Blues",
            vmin = 0.0,
#             vmax = 0.5,
            vmax = list(retention.max().sort_values(ascending = False))[1]+3,
            fmt = '.1f',
            linewidth = 0.3,
            yticklabels=month_list)

plt.show();
```



[35]:
```
# 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
```

[36]:
```
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)
```

[37]:
```
plt.figure(figsize=(15, 7))

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

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



[38]:
```
rfm_train['InvoiceDate'].max()
```

[38]:
```
Timestamp('2011-12-09 12:50:00')
```

[39]:
```
current_date = dt.date(2011,12,9)
```

[40]:
```
rfm_train['Purchase_Date'] = rfm_train.InvoiceDate.dt.date
```

[41]:
```
recency = rfm_train.groupby('CustomerID')['Purchase_Date'].max().reset_index()
```

[42]:
```
recency = recency.assign(Current_Date = current_date)
```

[43]:
```
recency['Recency'] = recency.Purchase_Date.apply(lambda x: (current_date - x).
↪days)
```

[44]:
```
recency.head()
```

[44]:

	CustomerID	Purchase_Date	Current_Date	Recency
0	12346	2011-01-18	2011-12-09	325
1	12347	2011-12-07	2011-12-09	2
2	12348	2011-09-25	2011-12-09	75
3	12349	2011-11-21	2011-12-09	18
4	12350	2011-02-02	2011-12-09	310

[45]:
```
recency.drop(['Purchase_Date','Current_Date'], axis=1, inplace=True)
```

[46]:
```
frequency = rfm_train.groupby('CustomerID').InvoiceNo.nunique().reset_index().
↪rename(columns={'InvoiceNo':'Frequency'})
```

[47]:
```
frequency.head()
```

[47]:

	CustomerID	Frequency
0	12346	2
1	12347	7
2	12348	4
3	12349	1
4	12350	1

[48]:
```
rfm_train['Total_cost'] = rfm_train.Quantity * rfm_train.UnitPrice
```

[49]:
```
monetary = rfm_train.groupby('CustomerID').Total_cost.sum().reset_index().
↪rename(columns={'Total_cost':'Monetary'})
```

[50]:
```
monetary.head()
```

[50]:

	CustomerID	Monetary
0	12346	0.00
1	12347	4310.00
2	12348	1797.24
3	12349	1757.55
4	12350	334.40

[51]:
```
temp_ = recency.merge(frequency, on='CustomerID')
rfm_table = temp_.merge(monetary, on='CustomerID')
```

[52]:
```
rfm_table.set_index('CustomerID',inplace=True)
rfm_table.head()
```

[52]:
```
            Recency  Frequency  Monetary
CustomerID
12346           325          2      0.00
12347             2          7   4310.00
12348            75          4   1797.24
12349            18          1   1757.55
12350           310          1    334.40
```

[53]:
```
rfm_train[rfm_train.CustomerID == rfm_table.index[0]]
```

[53]:
```
      InvoiceNo StockCode                      Description  Quantity  \
61619    541431     23166  MEDIUM CERAMIC TOP STORAGE JAR     74215
61624   C541433     23166  MEDIUM CERAMIC TOP STORAGE JAR    -74215

              InvoiceDate  UnitPrice  CustomerID         Country  \
61619 2011-01-18 10:01:00       1.04       12346  United Kingdom
61624 2011-01-18 10:17:00       1.04       12346  United Kingdom

      Purchase_Date  Total_cost
61619    2011-01-18    77183.60
61624    2011-01-18   -77183.60
```

[54]:
```
(current_date - rfm_train[rfm_train.CustomerID == rfm_table.index[0]].iloc[0].
↪Purchase_Date).days == rfm_table.iloc[0,0]
```

[54]:
```
True
```

[55]:
```
# RFM Quantiles
quantiles = rfm_table.quantile(q=[0.25,0.5,0.75])
quantiles
```

[55]:
```
      Recency  Frequency  Monetary
0.25    16.00       1.00    291.79
0.50    50.00       3.00    644.07
0.75   143.00       5.00   1608.34
```

[56]:
```
quantiles=quantiles.to_dict()
quantiles
```

[56]:
```
{'Recency': {0.25: 16.0, 0.5: 50.0, 0.75: 143.0},
 'Frequency': {0.25: 1.0, 0.5: 3.0, 0.75: 5.0},
 'Monetary': {0.25: 291.79499999999996,
  0.5: 644.0700000000002,
  0.75: 1608.335}}
```

[57]:
```
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
```

[58]:
```
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
```

[59]:
```
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,))
```

[60]:
```
rfm_segment.head()
```

[60]:

CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile
12346	325	2	0.00	1	2	1
12347	2	7	4310.00	4	4	4
12348	75	4	1797.24	2	3	4
12349	18	1	1757.55	3	1	4
12350	310	1	334.40	1	1	2

[61]:
```
rfm_segment['RFMScore'] = rfm_segment.R_Quartile.map(str) \
                          + rfm_segment.F_Quartile.map(str) \
                          + rfm_segment.M_Quartile.map(str)
rfm_segment.head()
```

[61]:

CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	\
12346	325	2	0.00	1	2	1	
12347	2	7	4310.00	4	4	4	
12348	75	4	1797.24	2	3	4	
12349	18	1	1757.55	3	1	4	

```
12350           310          1    334.40           1            1            2

           RFMScore
CustomerID
12346           121
12347           444
12348           234
12349           314
12350           112
```

[62]:
```
# Reset the index to create a customer_ID column
rfm_segment.reset_index(inplace=True)
```

[63]:
```
segment_dict = {
    'Best Customers':'444',      # Highest frequency as well as monetary value␣
↪with least recency
    'Loyal Customers':'344',     # High frequency as well as monetary value␣
↪with good recency
    'Big Spenders':'334',        # High monetary value but good recency and␣
↪frequency values
    'Almost Lost':'244',         # Customer's shopping less often now who used␣
↪to shop a lot
    'Lost Customers':'144',      # Customer's shopped long ago who used to shop␣
↪a lot.
    'Recent Customers':'443',    # Customer's who recently started shopping a␣
↪lot but with less monetary value
    'Lost Cheap Customers':'122' # Customer's shopped long ago but with less␣
↪frequency and monetary value
}
```

[64]:
```
dict_segment = dict(zip(segment_dict.values(),segment_dict.keys()))
```

[65]:
```
rfm_segment['Segment'] = rfm_segment.RFMScore.map(lambda x: dict_segment.get(x))
```

[66]:
```
rfm_segment.Segment.fillna('others', inplace=True)
```

[67]:
```
rfm_segment.sample(10)
```

[67]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
4257       18135       29          5    681.91           3           3
2760       16084      299          1    436.18           1           1
2003       15066       31          2    760.59           3           2
1380       14209       10          7   2067.13           4           4
1768       14733        9         15   9451.54           4           4
3286       16790        3          8   1520.12           4           4
183        12574      315          1    218.45           1           1
```

```
3849       17574      242          1    185.65           1           1
2786       16117      189          1    232.21           1           1
3320       16834      227          1    413.46           1           1

      M_Quartile RFMScore           Segment
4257           3      333            others
2760           2      112            others
2003           3      323            others
1380           4      444    Best Customers
1768           4      444    Best Customers
3286           3      443  Recent Customers
183            1      111            others
3849           1      111            others
2786           1      111            others
3320           2      112            others
```

[68]:
```
%%html
<div class='tableauPlaceholder' id='viz1574249157493' style='position:␣
↪relative'><noscript><a href='#'><img alt=' ' src='https:&#47;&#47;public.
↪tableau.com&#47;static&#47;images&#47;RF&#47;RFM_Analysis_15741611609370&#47;
↪RFMAnalysis&#47;1_rss.png' style='border: none' /></a></noscript><object␣
↪class='tableauViz'  style='display:none;'><param name='host_url'␣
↪value='https%3A%2F%2Fpublic.tableau.com%2F' /> <param␣
↪name='embed_code_version' value='3' /> <param name='site_root' value='' /
↪><param name='name' value='RFM_Analysis_15741611609370&#47;RFMAnalysis' /
↪><param name='tabs' value='no' /><param name='toolbar' value='yes' /><param␣
↪name='static_image' value='https:&#47;&#47;public.tableau.com&#47;static&#47;
↪images&#47;RF&#47;RFM_Analysis_15741611609370&#47;RFMAnalysis&#47;1.png' />␣
↪<param name='animate_transition' value='yes' /><param␣
↪name='display_static_image' value='yes' /><param name='display_spinner'␣
↪value='yes' /><param name='display_overlay' value='yes' /><param␣
↪name='display_count' value='yes' /></object></div>                <script␣
↪type='text/
↪javascript'>                    var divElement = document.getElementById('viz1574
```

```
<IPython.core.display.HTML object>
```

[69]:
```
rfm_segment[rfm_segment.RFMScore=='444'].sort_values('Monetary',␣
↪ascending=False).head()
```

[69]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
1703       14646        1         77 279489.02           4           4
4233       18102        0         62 256438.49           4           4
3758       17450        8         55 187322.17           4           4
1895       14911        1        248 132458.73           4           4
1345       14156        9         66 113214.59           4           4
```

```
      M_Quartile RFMScore         Segment
1703           4      444  Best Customers
4233           4      444  Best Customers
3758           4      444  Best Customers
1895           4      444  Best Customers
1345           4      444  Best Customers
```

[70]:
```
rfm_segment[rfm_segment.RFMScore=='334'].sort_values('Monetary',␣
↪ascending=False).head()
```

[70]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
2794       16126       29          4   6287.77           3           3
737        13316       37          5   5570.69           3           3
2923       16303       25          4   5305.83           3           3
2897       16258       45          5   5203.51           3           3
70         12432       42          5   5059.32           3           3

      M_Quartile RFMScore       Segment
2794           4      334  Big Spenders
737            4      334  Big Spenders
2923           4      334  Big Spenders
2897           4      334  Big Spenders
70             4      334  Big Spenders
```

[71]:
```
rfm_segment[rfm_segment.RFMScore=='244'].sort_values('Monetary',␣
↪ascending=False).head()
```

[71]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
464        12939       64          8  11581.80           2           4
50         12409       78          7  11056.93           2           4
2836       16180      100         10  10217.48           2           4
328        12744       51         10   9120.39           2           4
3248       16745       86         18   7157.10           2           4

      M_Quartile RFMScore      Segment
464            4      244  Almost Lost
50             4      244  Almost Lost
2836           4      244  Almost Lost
328            4      244  Almost Lost
3248           4      244  Almost Lost
```

[72]:
```
rfm_segment[rfm_segment.RFMScore=='122'].sort_values('Monetary',␣
↪ascending=False).head()
```

[72]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
1578       14481      164          2    636.51           1           2
```

```
2246       15384       169           3    635.76           1           2
1230       14000       206           2    633.71           1           2
1985       15045       151           3    633.66           1           2
1391       14220       247           2    632.40           1           2

      M_Quartile RFMScore               Segment
1578           2      122  Lost Cheap Customers
2246           2      122  Lost Cheap Customers
1230           2      122  Lost Cheap Customers
1985           2      122  Lost Cheap Customers
1391           2      122  Lost Cheap Customers
```

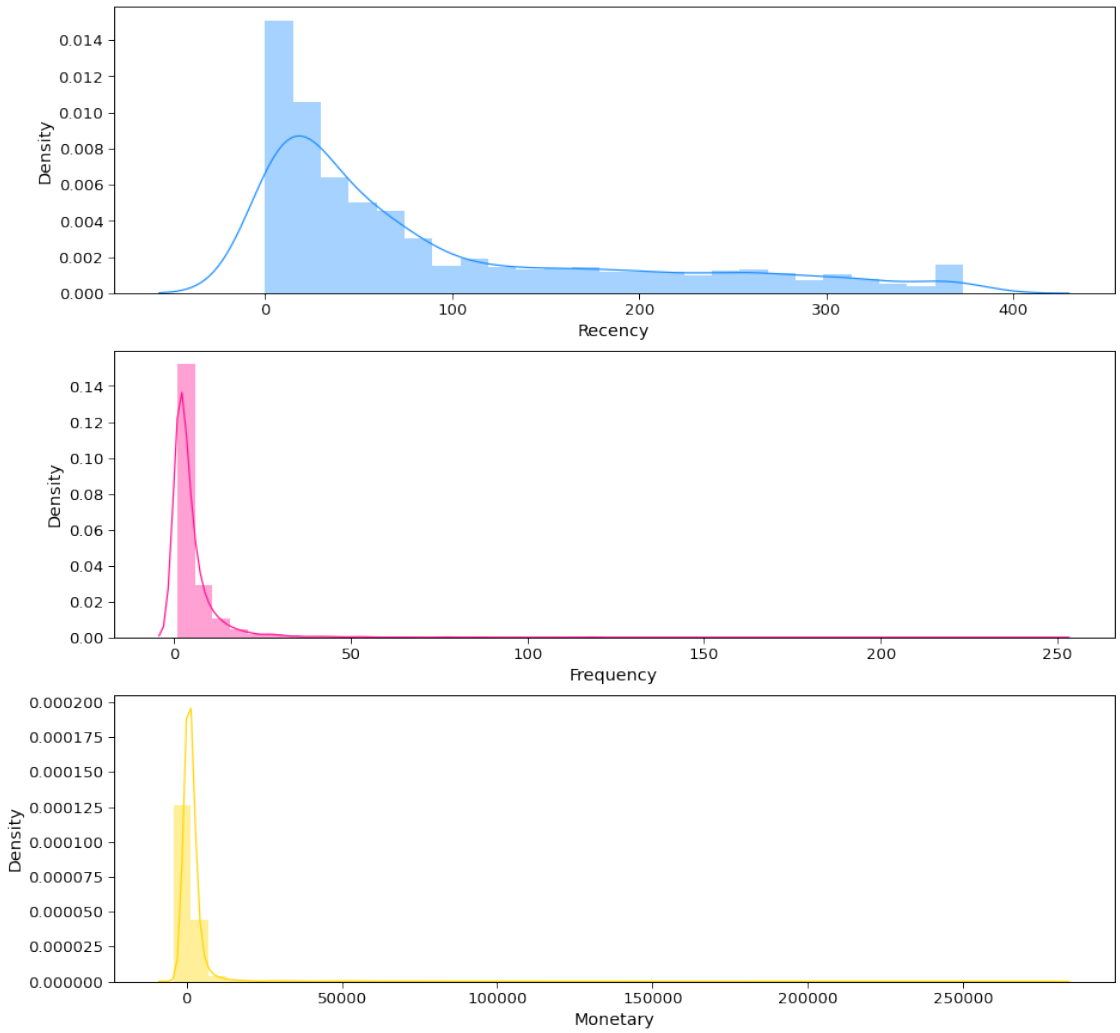
[73]:
```
rfm_segment[rfm_segment.RFMScore=='344'].sort_values('Monetary',␣
↪ascending=False).head()
```

[73]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
55         12415       24         26 123725.45           3           4
2722       16029       38         76  53168.69           3           4
3014       16422       17         75  33805.69           3           4
458        12931       21         20  33462.81           3           4
1728       14680       25         23  26932.34           3           4

      M_Quartile RFMScore          Segment
55             4      344  Loyal Customers
2722           4      344  Loyal Customers
3014           4      344  Loyal Customers
458            4      344  Loyal Customers
1728           4      344  Loyal Customers
```

[74]:
```
rfm_segment[rfm_segment.RFMScore=='244'].sort_values('Monetary',␣
↪ascending=False).head()
```

[74]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
464        12939       64          8  11581.80           2           4
50         12409       78          7  11056.93           2           4
2836       16180      100         10  10217.48           2           4
328        12744       51         10   9120.39           2           4
3248       16745       86         18   7157.10           2           4

      M_Quartile RFMScore      Segment
464            4      244  Almost Lost
50             4      244  Almost Lost
2836           4      244  Almost Lost
328            4      244  Almost Lost
3248           4      244  Almost Lost
```

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



[76]:
```
rfm_table.describe()
```

[76]:

	Recency	Frequency	Monetary
count	4372.00	4372.00	4372.00
mean	91.58	5.08	1893.53

```
std     100.77       9.34   8218.70
min       0.00       1.00  -4287.63
25%      16.00       1.00    291.79
50%      50.00       3.00    644.07
75%     143.00       5.00   1608.34
max     373.00     248.00 279489.02
```

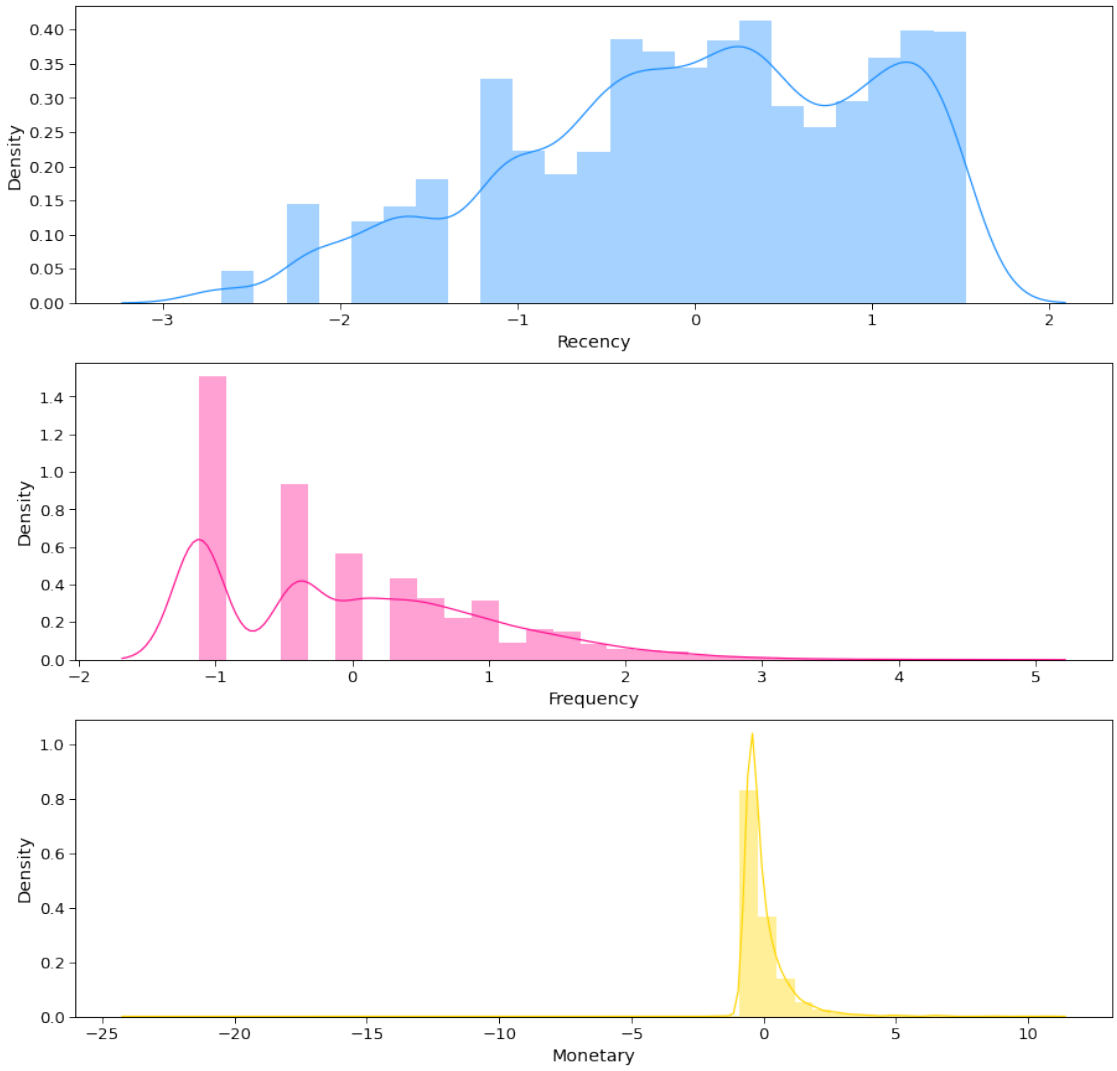
[77]:
```
rfm_table_scaled = rfm_table.copy()

# 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()
```

[77]:
```
       Recency  Frequency  Monetary
count  4372.00    4372.00   4372.00
mean     92.58       5.08   6182.16
std     100.77       9.34   8218.70
min       1.00       1.00      1.00
25%      17.00       1.00   4580.43
50%      51.00       3.00   4932.70
75%     144.00       5.00   5896.97
max     374.00     248.00 283777.65
```

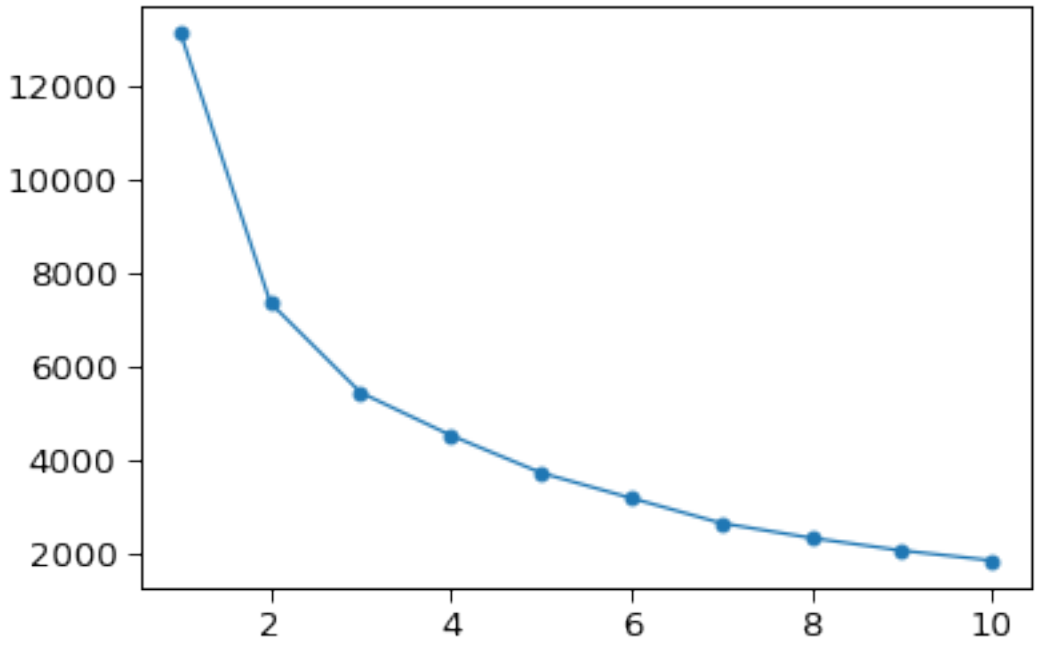
[78]:
```
from sklearn.preprocessing import StandardScaler

# 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)
```

[79]:
```
# plot again on the transformed RFM data
fig, axes = plt.subplots(3, 1, figsize=(15, 15))
sns.distplot(normal_df.Recency , color="dodgerblue", ax=axes[0],␣
 ↪axlabel='Recency')
sns.distplot(normal_df.Frequency , color="deeppink", ax=axes[1],␣
 ↪axlabel='Frequency')
sns.distplot(normal_df.Monetary , color="gold", ax=axes[2], axlabel='Monetary')
```

```
plt.show();
```



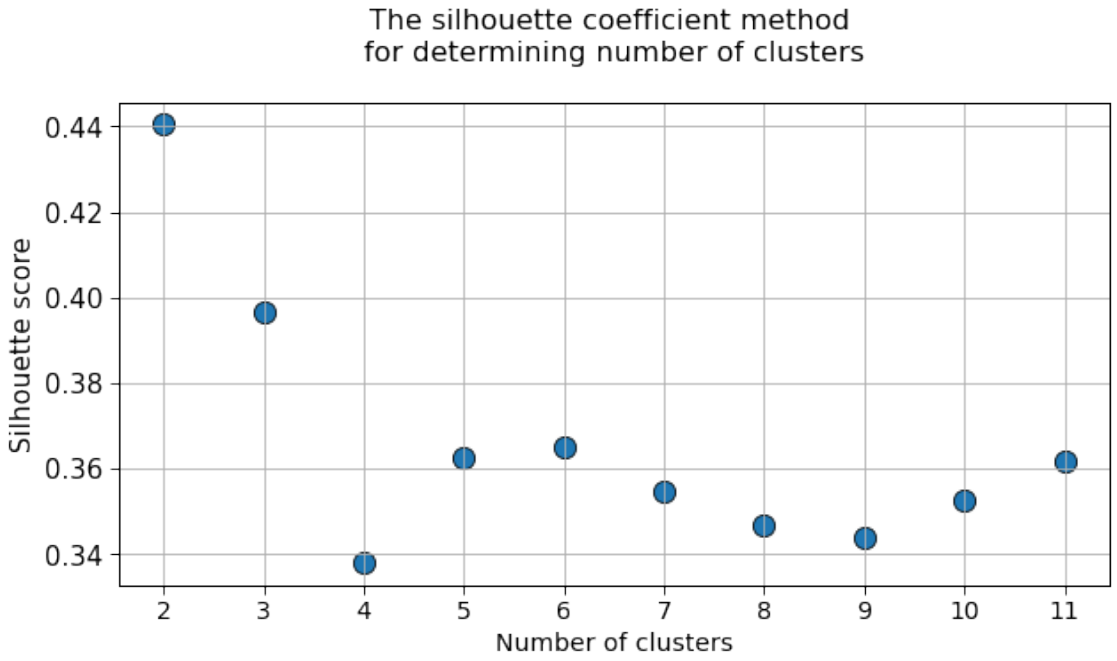
[81]:
```
# find WCSS
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++')
    kmeans.fit(normal_df)
    wcss.append(kmeans.inertia_)

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



[82]:
```
from sklearn.metrics import silhouette_score
wcss_silhouette = []
for i in range(2,12):
    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(2,12)],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(2,12)],fontsize=14)
plt.yticks(fontsize=15)
plt.show()
```

```
Silhouette score for number of cluster(s) 2: 0.4405297656150766
Silhouette score for number of cluster(s) 3: 0.39677649451050456
Silhouette score for number of cluster(s) 4: 0.3380133401024639
Silhouette score for number of cluster(s) 5: 0.3626494830847667
```

```
Silhouette score for number of cluster(s) 6: 0.3649032445702833
Silhouette score for number of cluster(s) 7: 0.354671831939756
Silhouette score for number of cluster(s) 8: 0.3466498591411801
Silhouette score for number of cluster(s) 9: 0.3439470526219597
Silhouette score for number of cluster(s) 10: 0.35264410417389935
Silhouette score for number of cluster(s) 11: 0.3618099356284099
```



[83]:
```
kmeans = KMeans(n_clusters=4, random_state=1, init='k-means++')
kmeans.fit(normal_df)
cluster_labels = kmeans.labels_
```

[84]:
```
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 (4372,)
Shape of RFM segment dataframe is (4372, 9)
```

[85]:
```
Cluster_table = rfm_segment.assign(Cluster = cluster_labels)
```

[86]:
```
Cluster_table.Cluster.value_counts()
```

[86]:
```
0    1786
3    1499
2     963
1     124
Name: Cluster, dtype: int64
```

[87]:
```
Cluster_table.sample(10)
```

[87]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScore	Segment	Cluster
650	13196	11	4	1797.78	4	3	4	434	others	3
2537	15769	7	29	51823.72	4	4	4	444	Best Customers	1
2108	15203	25	6	1827.80	3	4	4	344	Loyal Customers	2
3363	16898	26	4	436.68	3	3	2	332	others	3
2094	15182	154	2	622.85	1	2	2	122	Lost Cheap Customers	0
2746	16062	9	4	1153.62	4	3	3	433	others	3
3225	16716	266	3	319.80	1	2	2	122	Lost Cheap Customers	0
1823	14810	40	11	2085.33	3	4	4	344	Loyal Customers	2
4090	17899	159	1	154.55	1	1	1	111	others	0
1618	14530	25	6	2862.11	3	4	4	344	Loyal Customers	2

[88]:
```
Cluster_table[Cluster_table.Cluster == 3].sample(5)
```

[88]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScore	Segment	Cluster
4145	17974	24	3	703.99	3	2	3	323	others	3
838	13461	38	3	1445.00	3	2	3	323	others	3
1971	15028	8	4	627.13	4	3	2	432	others	3
2395	15582	24	3	682.91	3	2	3	323	others	3
846	13473	60	3	417.54	2	2	2	222	others	3

[89]:
```
Cluster_table[Cluster_table.Cluster == 2].sample(5)
```

[89]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile
3552	17162	28	11	1707.21	3	4
1250	14030	18	8	2358.84	3	4

```
3423       16979         3           8   1809.05           4           4
1564       14462        63          15   1942.45           2           4
3491       17071         8          14   2385.48           4           4

      M_Quartile RFMScore          Segment  Cluster
3552           4      344  Loyal Customers        2
1250           4      344  Loyal Customers        2
3423           4      444   Best Customers        2
1564           4      244      Almost Lost        2
3491           4      444   Best Customers        2
```

[90]:
```
Cluster_table[Cluster_table.Cluster == 1].sample(5)
```

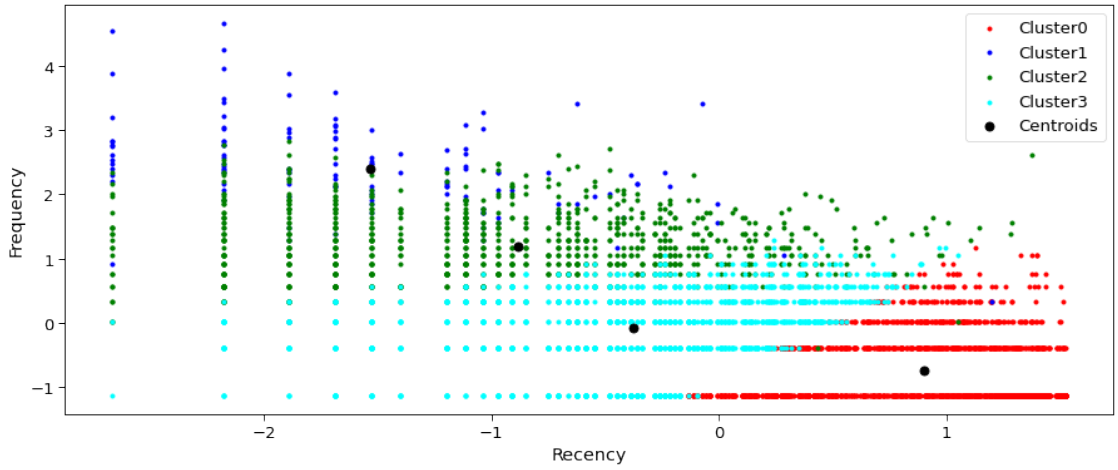
[90]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
464        12939       64          8  11581.80           2           4
1861       14866       10         13  14197.45           4           4
2725       16033        5         27   8690.03           4           4
276        12681       14         27  13677.59           4           4
3014       16422       17         75  33805.69           3           4

      M_Quartile RFMScore          Segment  Cluster
464            4      244      Almost Lost        1
1861           4      444   Best Customers        1
2725           4      444   Best Customers        1
276            4      444   Best Customers        1
3014           4      344  Loyal Customers        1
```

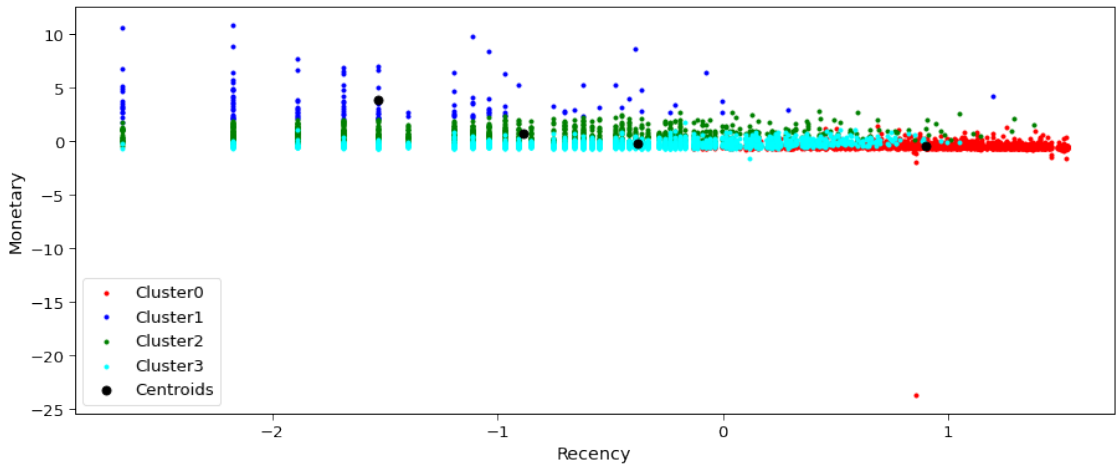
[91]:
```
Cluster_table[Cluster_table.Cluster == 0].sample(5)
```

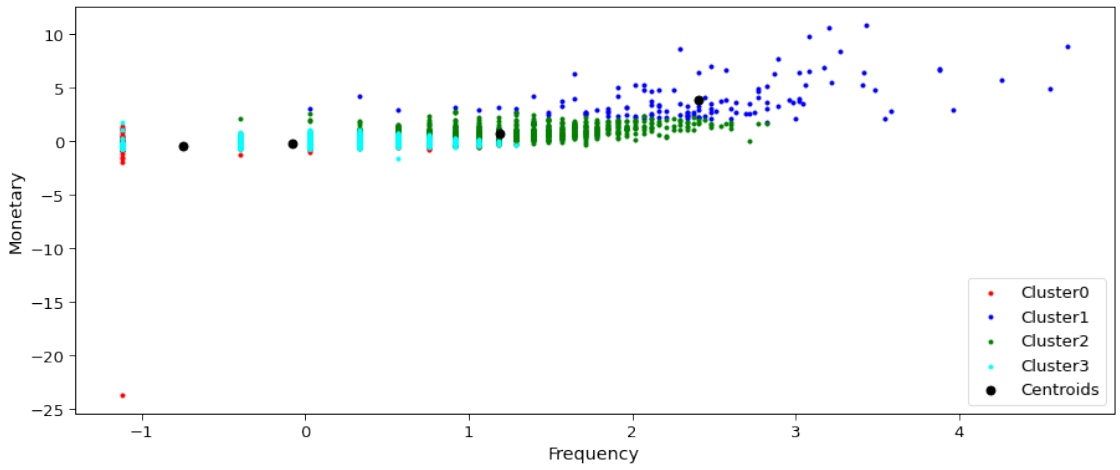
[91]:
```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile  \
1161       13900      183          3    740.95           1           2
821        13434       74          2    534.24           2           2
414        12868      185          6   1607.06           1           4
3701       17376       70          2    203.20           2           2
715        13284      322          2    196.15           1           2

      M_Quartile RFMScore Segment  Cluster
1161           3      123  others        0
821            2      222  others        0
414            3      143  others        0
3701           1      221  others        0
715            1      121  others        0
```

[92]:
```
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();
```





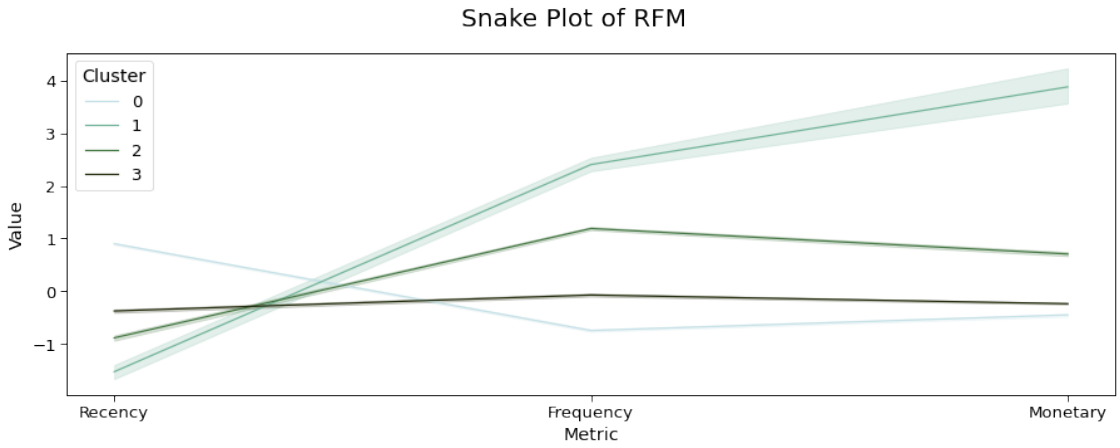


[93]:
```
normal_df = normal_df.assign(Cluster = cluster_labels)

# Melt normalized dataframe into long form to have all metric in same column
normal_melt = pd.melt(normal_df.reset_index(),
                      id_vars=['CustomerID','Cluster'],
                      value_vars=['Recency', 'Frequency', 'Monetary'],
                      var_name='Metric',
                      value_name='Value')
normal_melt.head()
```

[93]:

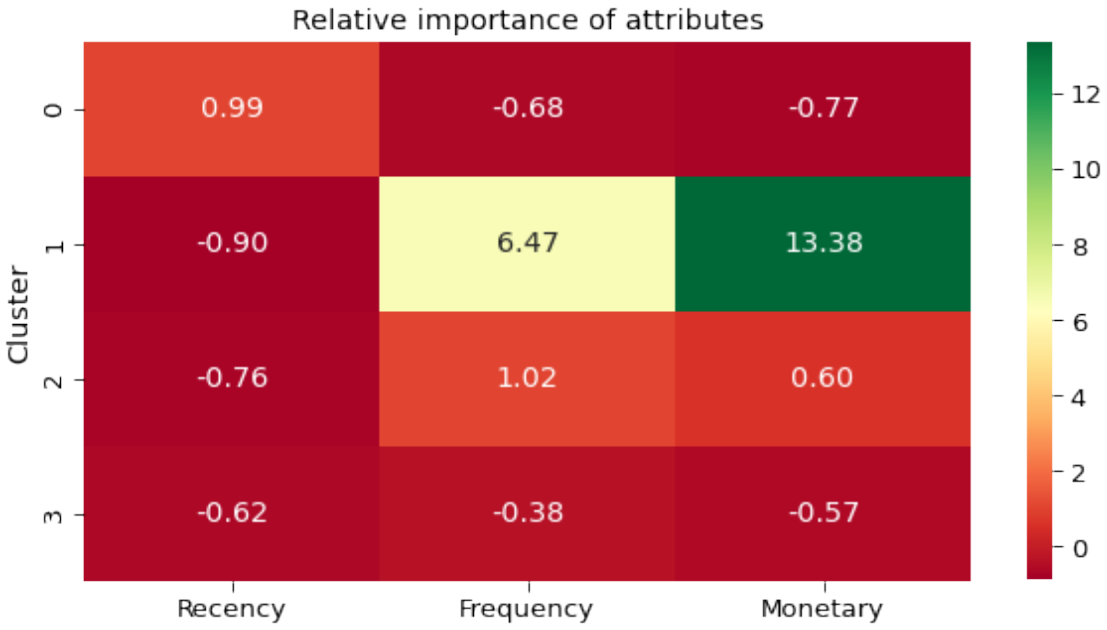
	CustomerID	Cluster	Metric	Value
0	12346	0	Recency	1.43
1	12347	2	Recency	-1.89
2	12348	3	Recency	0.40
3	12349	3	Recency	-0.58
4	12350	0	Recency	1.40

[94]:
```
plt.figure(figsize=(15,5))
palette = sns.color_palette("mako_r", 4)
sns.lineplot(x = 'Metric',
             y = 'Value',
             hue = 'Cluster',
             data = normal_melt,
             palette = "ch:4.4,.44")

plt.suptitle("Snake Plot of RFM",fontsize=20)
plt.show();
```



[95]:
```
# 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
```

[96]:
```
plt.figure(figsize=(10, 5))
plt.title('Relative importance of attributes')
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='RdYlGn')
plt.show();
```


Relative importance of attributes
Cluster
0
1
2
3
0.99
-0.68
-0.77
-0.90
6.47
13.38
-0.76
1.02
0.60
-0.62
-0.38
-0.57
Recency
Frequency
Monetary
12
10
8
6
4
2
0

[ ]: