ONLINE RETAIL CAPSTONE PROJECT

DESCRIPTION

Problem Statement

It is a critical requirement for business to understand the value derived from a customer. RFM is a method used for analyzing customer value. Customer segmentation is the practice of segregating the customer base into groups of individuals based on some common characteristics such as age, gender, interests, and spending habits Perform customer segmentation using RFM analysis. The resulting segments can be ordered from most valuable (highest recency, frequency, and value) to least valuable (lowest recency, frequency, and value).

Dataset Description

This is a transnational data set which contains all the transactions that occurred between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The company mainly sells unique and all-occasion gifts.

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

Load the necessary libraries.

In [92]:
```
import numpy as np
import pandas as pd
data = pd.read_excel('Online Retail.xlsx')
```

Take a brief preview of the first few rows of the data set

In [93]:
```
1 data.head()
```

Out[93]:

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

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

In [94]:
```
1
2 data.describe()
```

Out[94]:

	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

In [95]:
```
1 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
```

a. Check for missing data and formulate an apt strategy to treat them.

Observe the missing data in this UK retail sheet.

1. Customer ID and description having missing values

In [96]:
```
1 typecol = ['Categorical', 'Categorical','Categorical', 'Discrete', 'Date'
2 missingdf = pd.DataFrame({'Columns' : data.columns.to_list(), 'Type of da
3 def highlight_max(s):
4     is_max = s
5     return ['background-color: pink' if v else '' for v in is_max]
6 missingdf.style.apply(highlight_max, subset = ['No of missing data'])
7 missingdf.style.hide_index()
```

```
C:\Users\ANANT KALEKAR\AppData\Local\Temp\ipykernel_10808\3468921664.py:7: F
utureWarning: this method is deprecated in favour of `Styler.hide(axis='inde
x')`
  missingdf.style.hide_index()
```

Out[96]:

Columns	Type of data	No of missing data
InvoiceNo	Categorical	0
StockCode	Categorical	0
Description	Categorical	1454
Quantity	Discrete	0
InvoiceDate	Date	0
UnitPrice	Continuous	0
CustomerID	Categorical	135080
Country	Categorical	0

Handling missing data Treatment.

1. We will drop all the records where the customer ID is Null as it has no significance 2. The records where Description is Null will be automatically treated when discarding records with

missing values of customer ID 3. Customer ID and description now do not having missing values after treatment

In [97]:
```
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 th
print (any(data['Description'].isna()==True))
missingdf = pd.DataFrame({'Columns' : data.columns.to_list(), 'No of miss
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 Cust
omer ID columns
False

C:\Users\ANANT KALEKAR\AppData\Local\Temp\ipykernel_10808\1142630559.py:10:
FutureWarning: this method is deprecated in favour of `Styler.hide(axis='ind
ex')`
  missingdf.style.hide_index()
```

Out[97]:

Columns	No of missing data after cleaning
InvoiceNo	0
StockCode	0
Description	0
Quantity	0
InvoiceDate	0
UnitPrice	0
CustomerID	0
Country	0

b. Remove duplicate data records.

In [98]:
```
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
```

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

In [99]:
```
# 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)].inde

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
```

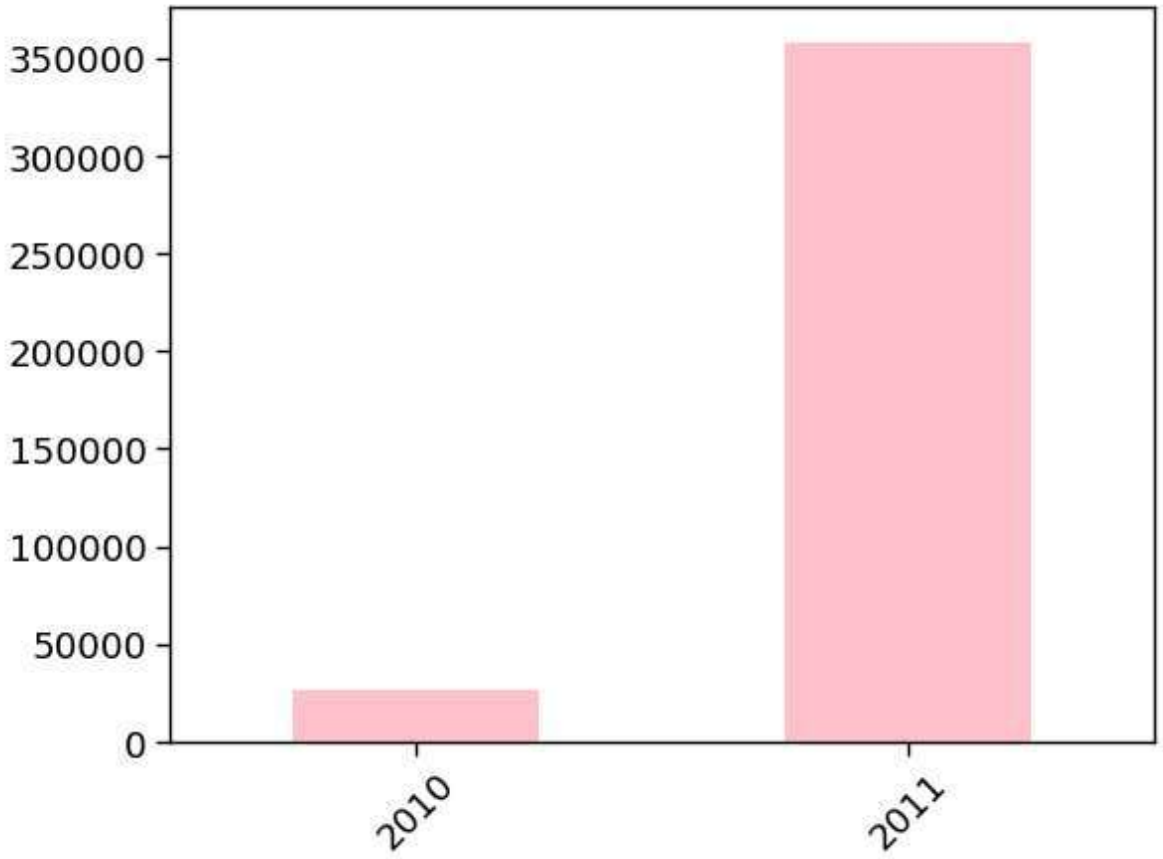
d. Perform descriptive analytics on the given data

Observe the countries that have most of the customers residing

In [100]:
```
data.Country.value_counts(normalize=True).head(10).mul(100).round(2).asty
```

Out[100]:
```
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
```

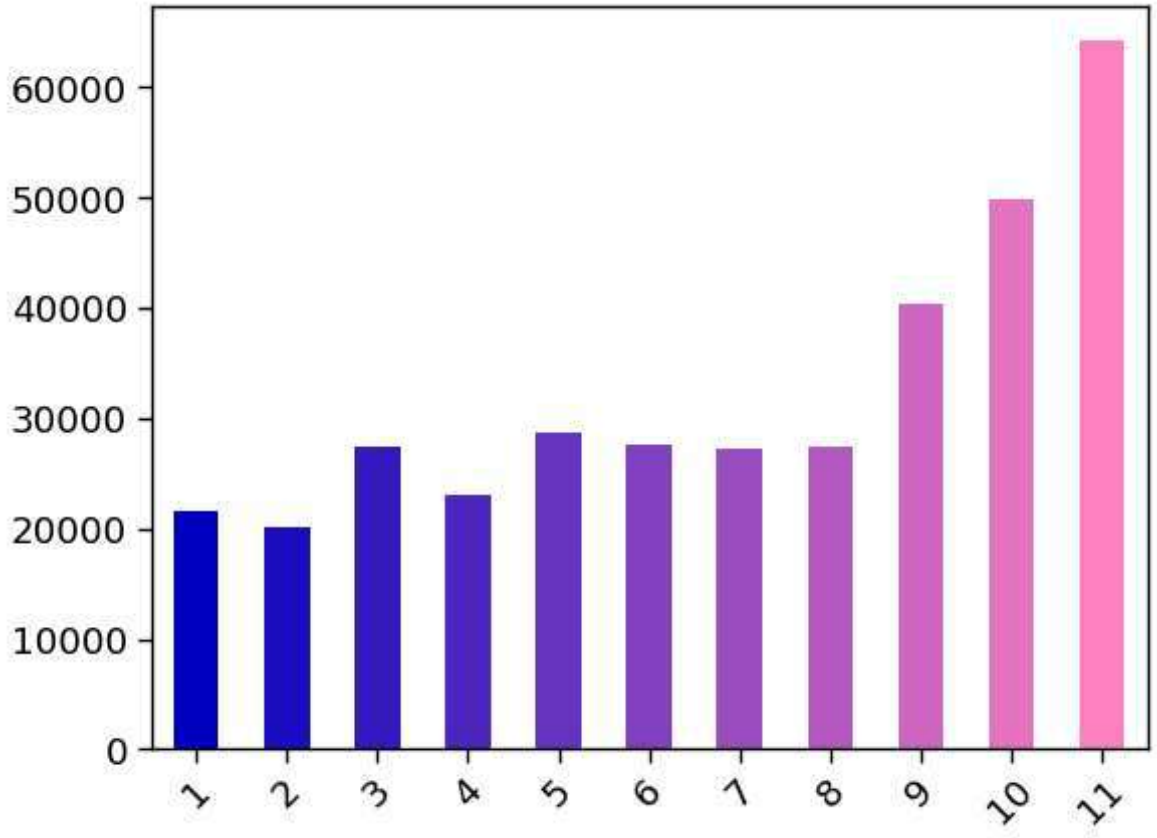
In [101]:
```
1 data.InvoiceDate.dt.year.value_counts(sort=False).plot(kind='bar', rot=45
2
```



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

In [102]:

```
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(data[data.InvoiceD
data[data.InvoiceDate.dt.year==2011].InvoiceDate.dt.month.value_counts(so
```

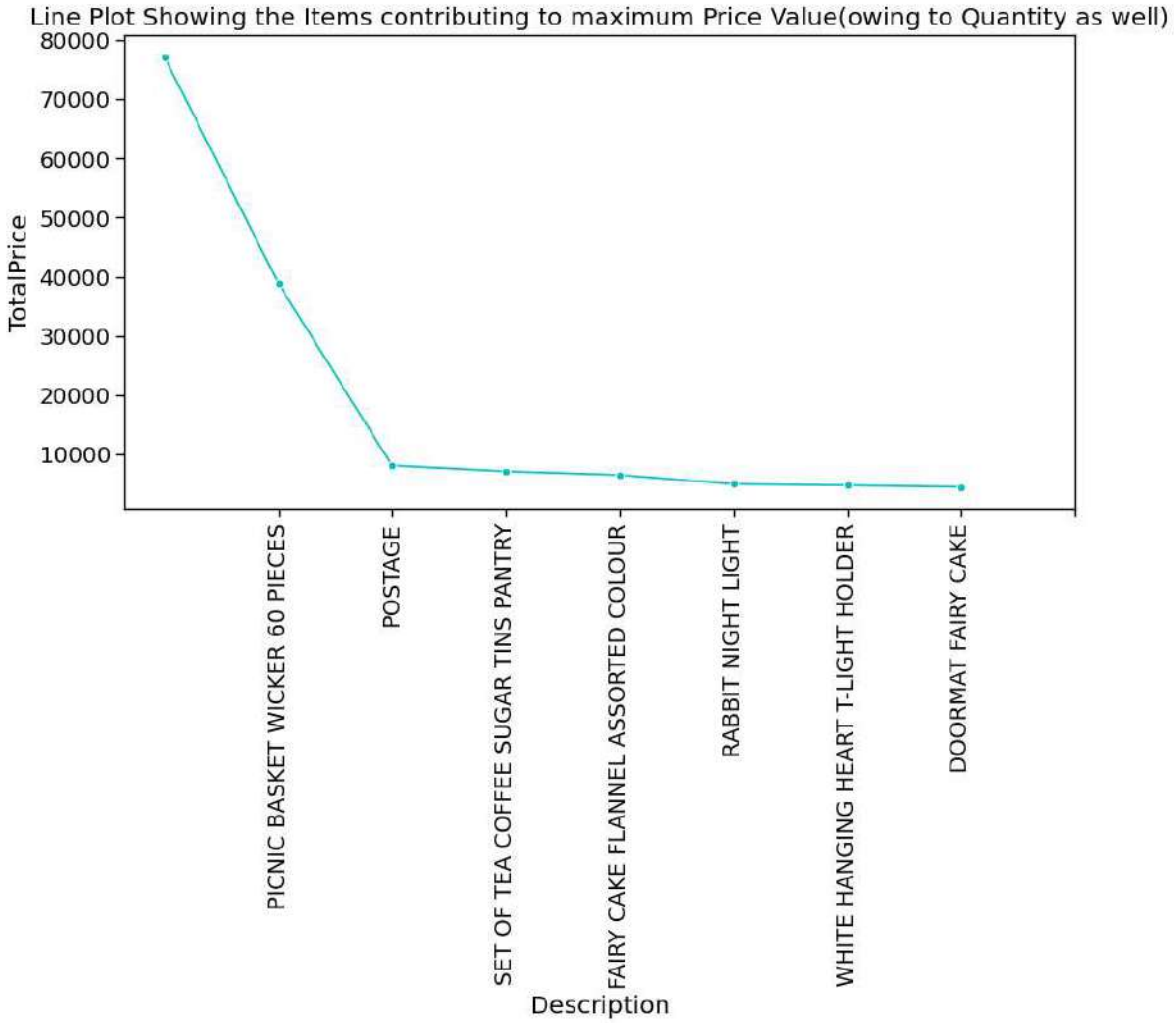


Visualize the Items contributing to maximum Price Value

In [103]:

```
data['TotalPrice'] = data.Quantity * data.UnitPrice
df =data.TotalPrice.sort_values(ascending=False).head(10).to_frame().styl
desc = data.sort_values(by='TotalPrice', ascending=False)['Description'].
price = data.sort_values(by='TotalPrice', ascending=False)['TotalPrice'].
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
plt.xticks(range(1,9), rotation=90)
plt.show();
```

```
C:\Users\ANANT KALEKAR\AppData\Local\Temp\ipykernel_10808\2347732985.py:2: F
utureWarning: this method is deprecated in favour of `Styler.hide(axis='inde
x')`
  df =data.TotalPrice.sort_values(ascending=False).head(10).to_frame().styl
e.hide_index()
```



Let us explore the data some more!

In [104]:

```
print ("First business transaction date is {}".format(data.InvoiceDate.mi
print ("Last business transaction  date is {}".format(data.InvoiceDate.ma
monthly_gross =data[data.InvoiceDate.dt.year==2011].groupby(data.InvoiceD
df = pd.DataFrame(monthly_gross)
df.index.name = 'Invoice Month'
df

```

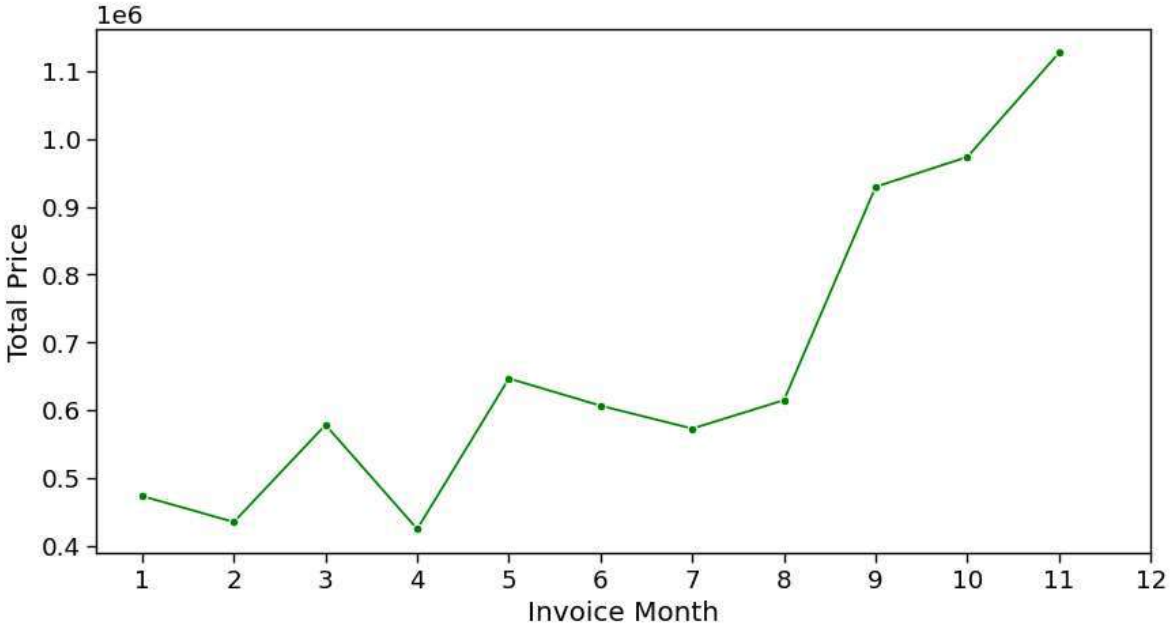
```
First business transaction date is 2010-12-01 08:26:00
Last business transaction  date is 2011-11-30 17:42:00
```

Out[104]:

Invoice Month	TotalPrice
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

In [105]:

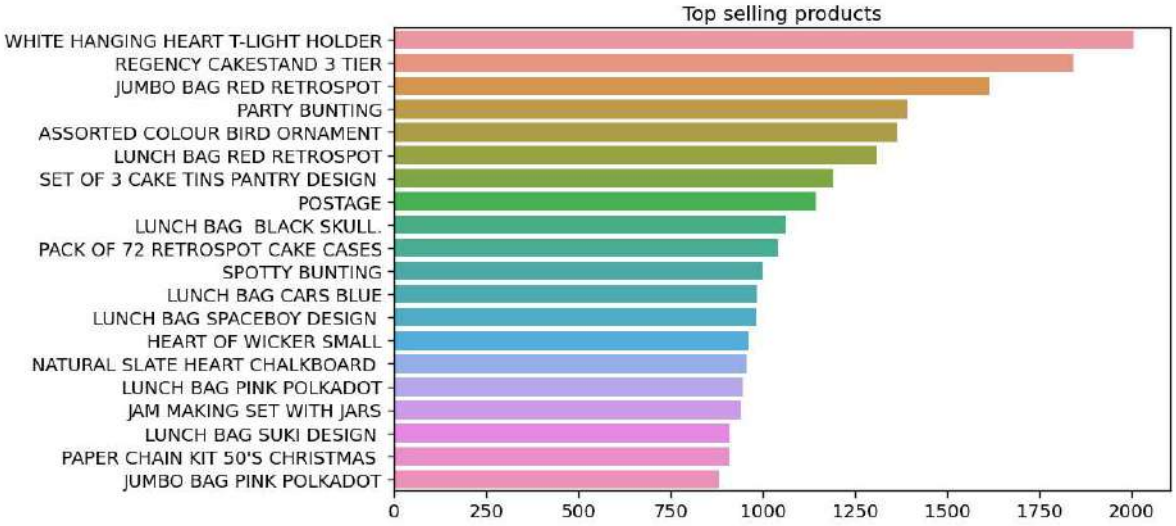
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
sns.lineplot(y=monthly_gross.values,x=monthly_gross.index, marker='o', co
plt.xlabel('Invoice Month')
plt.ylabel('Total Price')
plt.xticks(range(1,13))
plt.show();

```



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

In [106]:

```
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();
```



In [107]:
```
1 pd.DataFrame(data['Description'].value_counts())
```

Out[107]:

	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 × 1 columns

Observe Outliers using the 1.5 Interquartile Rule

Though it's not often affected much by them, the interquartile range can be used to detect outliers. This is done using these steps:

```
1. Calculate the interquartile range for the data.
2. Multiply the interquartile range (IQR) by 1.5 (a constant used to
discern outliers).
3. Add 1.5 x (IQR) to the third quartile. Any number greater than thi
s is a suspected outlier.
4. Subtract 1.5 x (IQR) from the first quartile. Any number less than
this is a suspected outlier.
```

Remember that the interquartile rule is only a rule of thumb that generally holds but does not apply to every case. In general, you should always follow up your outlier analysis by studying the resulting outliers to see if they make sense. Any potential outlier obtained by the interquartile method should be examined in the context of the entire set of data.

In [182]:

```
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 ,

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)].
            print ("Detected  outliers for this column %r " % col)
            #hdataUpdated.drop(hdataUpdated[(hdataUpdated[col] > upperRan
```

2. Perform cohort analysis (a cohort is a group of subjects that share a defining characteristic). Observe how a cohort behaves across time and compare it to other cohorts.

a. Create month cohorts and analyze active customers for each cohort.

COHORT ANALYSIS

A cohort is a group of subjects who share a defining characteristic. We can observe how a cohort behaves across time and compare it to other cohorts.

Types of cohorts:

* Time Cohorts:

They are customers who signed up for a product or service during a particular time frame. Analyzing these cohorts shows the customers' behavior depending on the time they started using the company's products or services. The time may be monthly or quarterly even daily.

* Behaviour Cohorts:

They are customers who purchased a product or subscribed to a service in the past. It groups customers by the type of product or service they signed up. Customers who signed up for basic level services might have different needs than those who signed up for advanced services. Understaning the needs of the various cohorts can help a company design custom-made services or products for particular segments.

* Size Cohorts:

Size cohorts refer to the various sizes of customers who purchase company's products or services. This categorization can be based on the amount of spending in some periodic time after acquisition or the product type that the customer spent most of their order amount in some period of time.

For cohort analysis, there are a few labels that we have to create:

Invoice period - A string representation of the year and month of a single transaction/invoice. Cohort group - A string representation of the the year and month of a customer's first purchase. This label is common across all invoices for a particular customer. Cohort period/Index- A integer representation a customer's stage in its "lifetime". The number represents the number of months passed since the first purchase.

In [109]:
```
cohort =data.copy()
cohort
```

Out[109]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Un Kingc
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Un Kingc
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
...	...	...	...	...	...	...	...	
516379	C579886	22197	POPCORN HOLDER	-1	2011-11-30 17:39:00	0.85	15676.0	Un Kingc
516380	C579886	23146	TRIPLE HOOK ANTIQUE IVORY ROSE	-1	2011-11-30 17:39:00	3.29	15676.0	Un Kingc
516381	C579887	84946	ANTIQUE SILVER T-LIGHT GLASS	-1	2011-11-30 17:42:00	1.25	16717.0	Un Kingc
516382	C579887	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	-1	2011-11-30 17:42:00	7.95	16717.0	Un Kingc
516383	C579887	23490	T-LIGHT HOLDER HANGING LOVE BIRD	-3	2011-11-30 17:42:00	3.75	16717.0	Un Kingc

384222 rows × 10 columns

Assign monthly acquisition cohort Defining a cohort is the first step to cohort analysis. We will now create monthly cohorts based on the month each customer has made their first transactio

In [110]:

```
# 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')
```

In [111]:
```
cohort
#grouping
```

Out[111]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Un Kingc
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Un Kingc
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
...	...	...	...	...	...	...	...	
516379	C579886	22197	POPCORN HOLDER	-1	2011-11-30 17:39:00	0.85	15676.0	Un Kingc
516380	C579886	23146	TRIPLE HOOK ANTIQUE IVORY ROSE	-1	2011-11-30 17:39:00	3.29	15676.0	Un Kingc
516381	C579887	84946	ANTIQUE SILVER T-LIGHT GLASS	-1	2011-11-30 17:42:00	1.25	16717.0	Un Kingc
516382	C579887	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	-1	2011-11-30 17:42:00	7.95	16717.0	Un Kingc
516383	C579887	23490	T-LIGHT HOLDER HANGING LOVE BIRD	-3	2011-11-30 17:42:00	3.75	16717.0	Un Kingc

384222 rows × 11 columns

Calculate time offset in months
Calculating time offset for each transaction allows you to report the metrics for each cohort in a comparable fashion.

First, we will create some variables that capture the integer value of years and months for Invoice and Cohort Date using the get_date_int() function

In [112]:

```
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')
```

In [113]:

```
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.uniqu

```

```
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]
```

In [114]:

```
# 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
```

In [115]:
```
#THis Cohort Index will give us an idea on the time difference in months
cohort['CohortIndex'].unique()
cohort
```

Out[115]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Un Kingc
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Un Kingc
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
...	...	...	...	...	...	...	...	
516379	C579886	22197	POPCORN HOLDER	-1	2011-11-30 17:39:00	0.85	15676.0	Un Kingc
516380	C579886	23146	TRIPLE HOOK ANTIQUE IVORY ROSE	-1	2011-11-30 17:39:00	3.29	15676.0	Un Kingc
516381	C579887	84946	ANTIQUE SILVER T-LIGHT GLASS	-1	2011-11-30 17:42:00	1.25	16717.0	Un Kingc
516382	C579887	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	-1	2011-11-30 17:42:00	7.95	16717.0	Un Kingc
516383	C579887	23490	T-LIGHT HOLDER HANGING LOVE BIRD	-3	2011-11-30 17:42:00	3.75	16717.0	Un Kingc

384222 rows × 12 columns

Calculate retention rate Customer retention is a very useful metric to understand how many of all the customers are still active. It gives you the percentage of active customers compared to the total number of customers

In [116]:

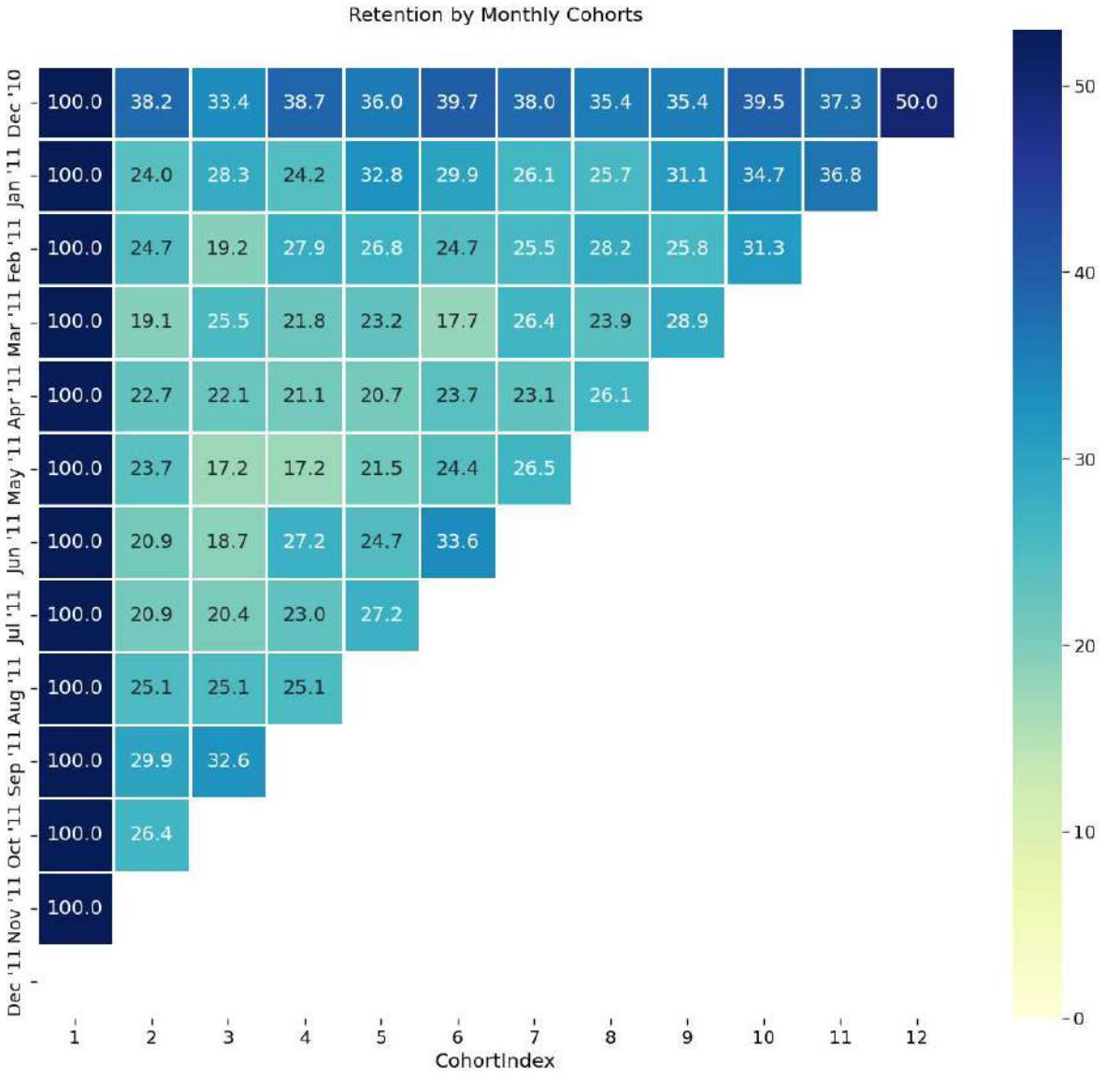
```
grouping = cohort.groupby(['CohortMonth', 'CohortIndex'])

```

In [117]:

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

# Create a pivot
cohort_counts = cohort_data.pivot(index='CohortMonth', columns='CohortInd

# 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
#cohort_sizes
retention.index = retention.index.date

```

In [118]:

```
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
            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 640x480 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

In [119]:

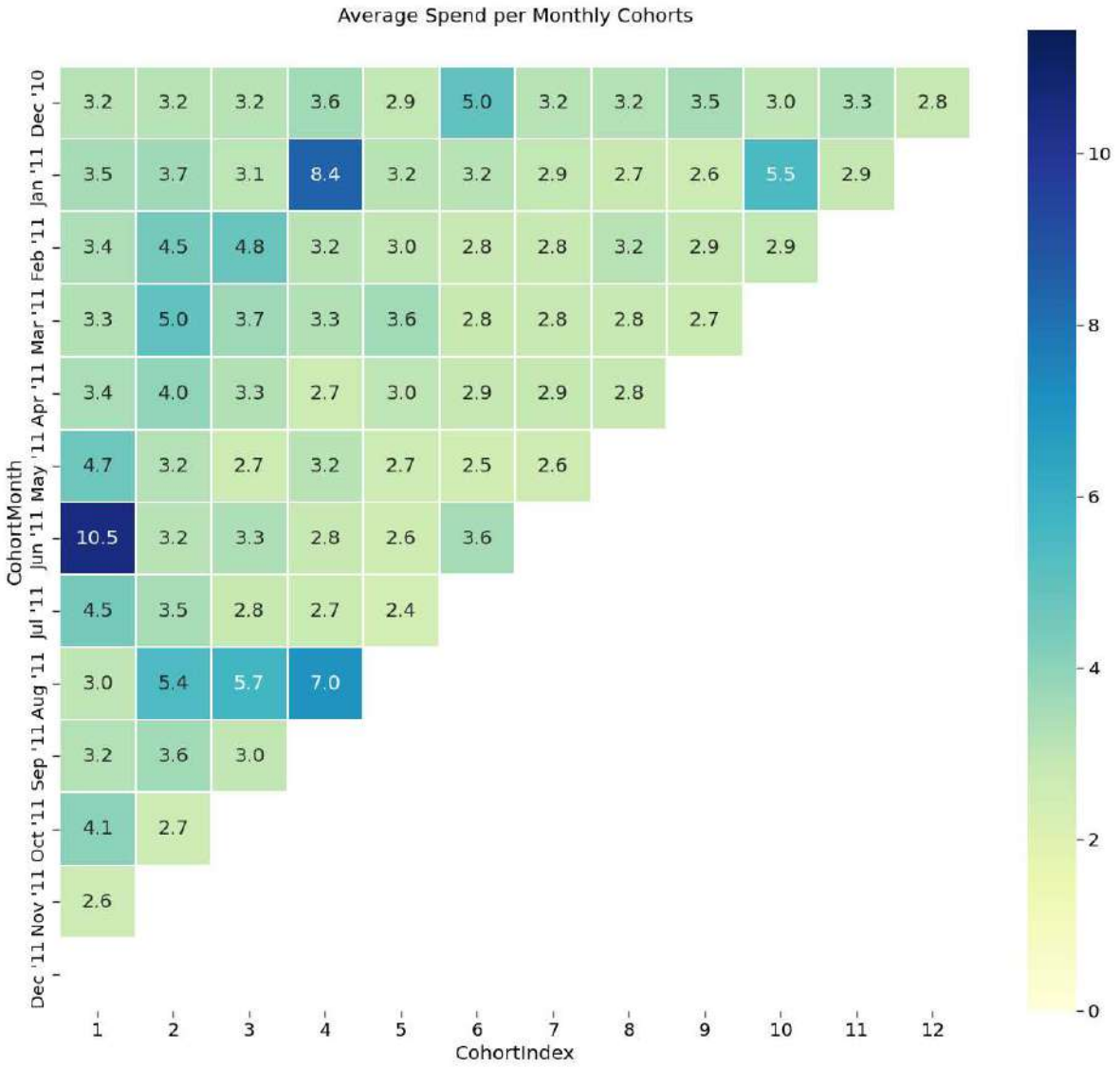
```
# Create a groupby object and pass the monthly cohort and cohort index as
grouping = cohort.groupby(['CohortMonth', 'CohortIndex'])

# 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='CohortInd
#average_price.round(1)
#average_price.index = average_price.index.date
average_price
#cohort_data
#cohort
```

Out[119]:

CohortIndex	1	2	3	4	5	6	7	8
CohortMonth								
2010-12-01	3.216682	3.182040	3.207467	3.603758	2.937803	4.996508	3.184572	3.235695
2011-01-01	3.505492	3.653572	3.069534	8.439024	3.157803	3.172919	2.918498	2.749649
2011-02-01	3.355968	4.469638	4.824106	3.150045	2.987616	2.792577	2.812985	3.214380
2011-03-01	3.302802	4.990095	3.655094	3.289768	3.616562	2.758381	2.843273	2.809136
2011-04-01	3.431172	3.958074	3.300128	2.673439	3.028297	2.867185	2.902668	2.812492
2011-05-01	4.662054	3.243691	2.652761	3.167391	2.667158	2.495751	2.615408	NaN
2011-06-01	10.490030	3.205283	3.343994	2.835952	2.553037	3.550657	NaN	NaN
2011-07-01	4.493676	3.480495	2.752121	2.701985	2.403989	NaN	NaN	NaN
2011-08-01	3.028246	5.425904	5.714033	7.046410	NaN	NaN	NaN	NaN
2011-09-01	3.235116	3.584834	2.957893	NaN	NaN	NaN	NaN	NaN
2011-10-01	4.053162	2.678140	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	2.641554	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [120]:

```
# 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
                 fmt = '.1f',
                 linewidth = 0.3,
                 yticklabels=month_list)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show();
```



Calculate average quantity per cohort

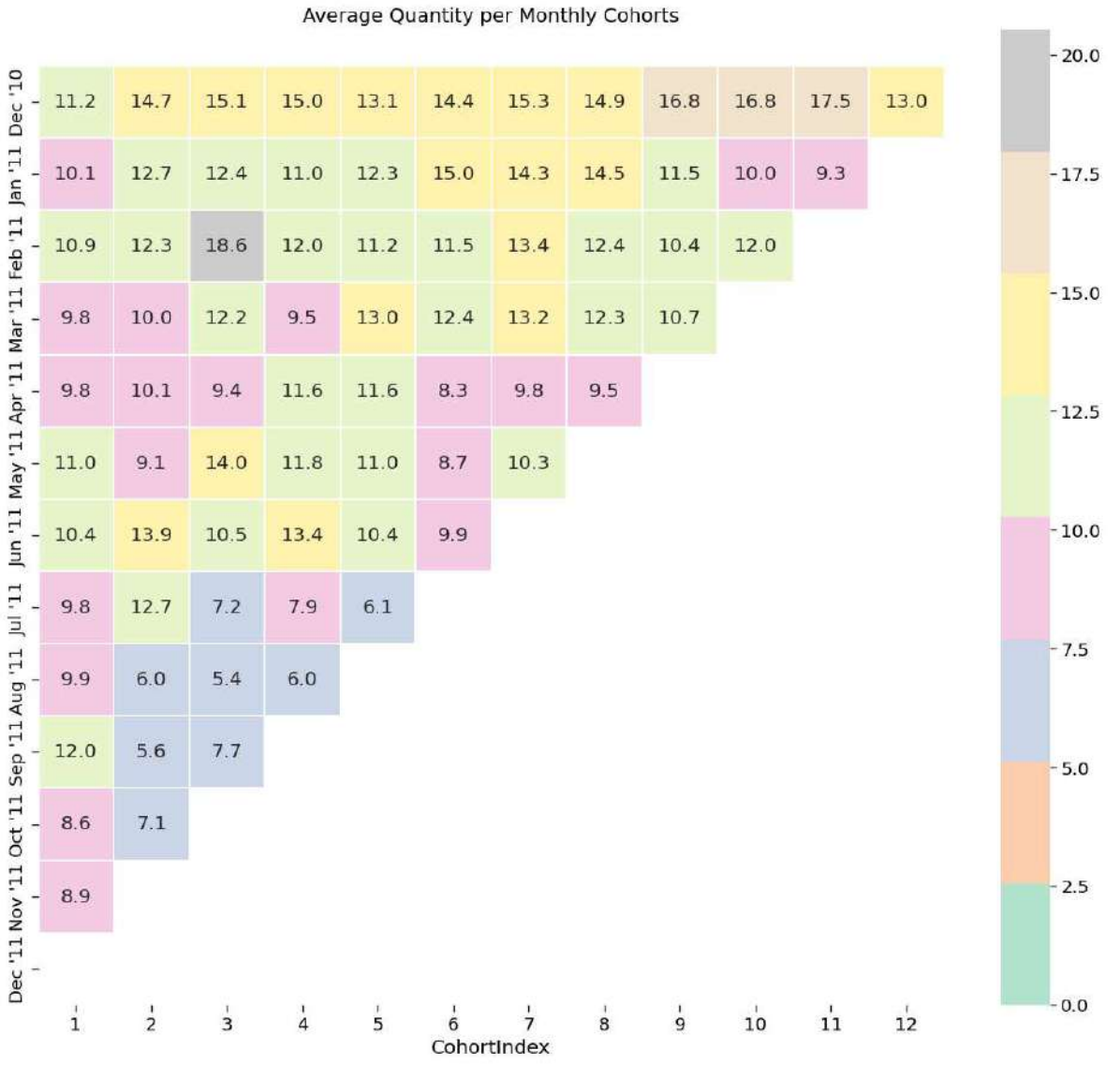
Now we will calculate the average quantity metric and analyze if there are any differences in shopping patterns across time and across cohorts.

In [121]:

```
# Create a groupby object and pass the monthly cohort and cohort index as
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='Cohort
average_quantity.round(1)
average_quantity.index = average_quantity.index.date

```

In [122]:

```
# 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 = Fa
                  fmt = '.1f',
                  linewidth = 0.3,
                  yticklabels=month_list)

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



Project Task: Week 2

Data Modeling :

1. Build a RFM (Recency Frequency Monetary) model. Recency means the number of days since a customer made the last purchase. Frequency is the number of purchase in a given period. It could be 3 months, 6 months or 1 year. Monetary is the total amount of money a customer spent in that given period. Therefore, big spenders will be differentiated among other customers such as MVP (Minimum Viable Product) or VIP.

2. Calculate RFM metrics.

3. Build RFM Segments. Give recency, frequency, and monetary scores individually by dividing them into quartiles.

   b1. Combine three ratings to get a RFM segment (as strings).

   b2. Get the RFM score by adding up the three ratings.

   b3. Analyze the RFM segments by summarizing them and comment on the findings.

Note: Rate “recency" for customer who has been active more recently higher than the less recent customer, because each company wants its customers to be recent.

Note: Rate “frequency" and “monetary" higher, because the company wants the customer to visit more often and spend more money

RFM Analysis

RFM analysis is a customer segmentation technique that uses past purchase behavior to divide customers into groups. RFM helps divide customers into various categories or clusters to identify customers who are more likely to respond to promotions and also for future personalization services.

Recency (R):

Time since last purchase

Frequency (F):

Total number of purchases

Monetary (M):

Total purchase value

Benefits of RFM analysis

Increased customer retention Increased response rate Increased conversion rate Increased revenue

To perform RFM analysis, we divide customers into four equal groups according to the distribution of values for recency, frequency, and monetary value. Four equal groups across three variables create 64 (4x4x4) different customer segments, which is a manageable number.

For example, let’s look at a customer who: is within the group who purchased most recently (R=4), is within the group who purchased most quantity (F=4), is within the group who spent the most (M=4) This customer belongs to RFM segment 4-4-4 (Best Customers), (R=4, F=4, M=4)

In [123]:

```
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', '
                 ['413', '314', '313', '414'],
                 ['331', '341', '431', '441'],
                 ['422', '423', '424', '432', '433', '434', '442', '443', '
                 ['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
                 '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 bec
                 'Customers who were frequent and good spenders who are los
                 'Customers who purchased long ago , less frequent and very

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

Out[123]:

	Segment	RFM	Description	Marketing
0	Platinum Customers	[444, 443]	Customers who bought most recently, most often...	No price incentives, New products and Loyalty ...
1	Big Spenders	[114, 124, 134, 144, 214, 224, 234, 244, 314, ...	Customers who spend the most	Market your most expensive products
2	High Spend New Customers	[413, 314, 313, 414]	New Customers who spend the most	Price Incentives
3	Lowest-Spending Active Loyal Customers	[331, 341, 431, 441]	Active Customers who buy very often but spend ...	Promote economical cost effective products in ...
4	Recent Customers	[422, 423, 424, 432, 433, 434, 442, 443, 444]	Customers who have purchased recently	Discounts and promote a variety of product sells
5	Good Customers Almost Lost	[244, 234, 243, 233]	Customers who were frequent and good spenders ...	Aggressive Price Incentives
6	Churned Best Customers	[144, 134, 143, 133]	Customers who were frequent and good spenders ...	Monitor close communication with customers wit...
7	Lost Cheap Customers	[122, 111, 121, 112, 221, 212, 211]	Customers who purchased long ago , less freque...	Dont spend too much time to re-acquire

Recency

Recency is about when was the last order of a customer. It means the number of days since a customer made the last purchase. If it's a case for a website or an app, this could be interpreted as the last visit day or the last login time.

In [124]:
```
#last date available in our dataset
import datetime as dt
data['InvoiceDate'].max()
```

Out[124]:
```
Timestamp('2011-11-30 17:42:00')
```

In [125]:
```
# Lets set this date as the today's date for further analysis
current_date = dt.date(2011,11,30)
current_date
```

Out[125]:
```
datetime.date(2011, 11, 30)
```

In [126]:
```
# Lets create a date column for date values only
data['Purchase_Date'] = data.InvoiceDate.dt.date
```

In [127]:
```
recency = data.groupby('CustomerID')['Purchase_Date'].max().reset_index()
recency
```

Out[127]:

	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 × 2 columns

In [128]:
```
# Create a separate column for this date.
recency = recency.assign(Current_Date = current_date)
recency
```

Out[128]:

	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 × 3 columns

In [129]:
```
# Compute the number of days since last purchase
recency['Recency'] = recency.Purchase_Date.apply(lambda x: (current_date
current_date
```

Out[129]:
```
datetime.date(2011, 11, 30)
```

In [130]:
```
1 recency.head()
```

Out[130]:

	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

In [131]:
```
1 # Drop the irrelevant Date columns
2 recency.drop(['Purchase_Date','Current_Date'], axis=1, inplace=True)
3 recency
```

Out[131]:

	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 × 2 columns

Frequency

Frequency is about the number of purchase in a given period. It could be 3 months, 6 months or 1 year. So we can understand this value as for how often or how many a customer used the product of a company. The bigger the value is, the more engaged the customers are. Could we say them as our VIP? Not necessary. Cause we also have to think about how much they actually paid for each purchase, which means monetary value

In [132]:
```
1 frequency = data.groupby('CustomerID').InvoiceNo.nunique().reset_index().
2 frequency.max()
```

Out[132]:
```
CustomerID    18287.0
Frequency       238.0
dtype: float64
```

Monetary

Monetary is the total amount of money a customer spent in that given period. Therefore big spenders will be differentiated with other customers such as MVP or VIP.

In [133]:
```
# Create a separate column for Total Cost of Unit purchased
data['Total_cost'] = data.Quantity * data.UnitPrice
data
```

Out[133]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Un Kingc
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Un Kingc
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
...	...	...	...	...	...	...	...	
516379	C579886	22197	POPCORN HOLDER	-1	2011-11-30 17:39:00	0.85	15676.0	Un Kingc
516380	C579886	23146	TRIPLE HOOK ANTIQUE IVORY ROSE	-1	2011-11-30 17:39:00	3.29	15676.0	Un Kingc
516381	C579887	84946	ANTIQUE SILVER T-LIGHT GLASS	-1	2011-11-30 17:42:00	1.25	16717.0	Un Kingc
516382	C579887	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	-1	2011-11-30 17:42:00	7.95	16717.0	Un Kingc
516383	C579887	23490	T-LIGHT HOLDER HANGING LOVE BIRD	-3	2011-11-30 17:42:00	3.75	16717.0	Un Kingc

384222 rows × 12 columns

In [134]:
```
monetary = data.groupby('CustomerID').Total_cost.sum().reset_index().rena
monetary.head()
```

Out[134]:

	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

In [135]:
```
rf = recency.merge(frequency, on='CustomerID')
rfm_table = rf.merge(monetary, on='CustomerID')
```

In [136]:
```
rfm_table.set_index('CustomerID',inplace=True)
rfm_table.head()
#rfm_table.Monetary.max()
```

Out[136]:

	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

RFM Table integrity Check

Let's check whether the RFM table attributes are in conjunction with the original values

In [137]:
```
rfm_table.index[1]
```

Out[137]:
```
12347.0
```

In [138]:
```
# Fetch the records corresponding to the first customer id in above table
data[data.CustomerID == rfm_table.index[1]]
```

Out[138]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
14938	537626	85116	BLACK CANDELABRA T-LIGHT HOLDER	12	2010-12-07 14:57:00	2.10	12347.0	Ic
14939	537626	22375	AIRLINE BAG VINTAGE JET SET BROWN	4	2010-12-07 14:57:00	4.25	12347.0	Ic
14940	537626	71477	COLOUR GLASS. STAR T-LIGHT HOLDER	12	2010-12-07 14:57:00	3.25	12347.0	Ic
14941	537626	22492	MINI PAINT SET VINTAGE	36	2010-12-07 14:57:00	0.65	12347.0	Ic
14942	537626	22771	CLEAR DRAWER KNOB ACRYLIC EDWARDIAN	12	2010-12-07 14:57:00	1.25	12347.0	Ic
...	...	...	...	...	...	...	...	
428999	573511	22196	SMALL HEART MEASURING SPOONS	24	2011-10-31 12:25:00	0.85	12347.0	Ic
429000	573511	22195	LARGE HEART MEASURING SPOONS	24	2011-10-31 12:25:00	1.65	12347.0	Ic
429001	573511	20719	WOODLAND CHARLOTTE BAG	10	2011-10-31 12:25:00	0.85	12347.0	Ic
429002	573511	23162	REGENCY TEA STRAINER	8	2011-10-31 12:25:00	3.75	12347.0	Ic
429003	573511	22131	FOOD CONTAINER SET 3 LOVE HEART	6	2011-10-31 12:25:00	1.95	12347.0	Ic

171 rows × 12 columns

In [139]:
```
# Check if the number difference of days from the purchase date in origin
(current_date - data[data.CustomerID == rfm_table.index[0]].iloc[0].Purch


```

Out[139]: True

Customer segments with RFM Model

The simplest way to create customers segments from RFM Model is to use Quantiles. We assign a score from 1 to 4 to Recency, Frequency and Monetary. Four is the best/highest value, and one is the lowest/worst value. A final RFM score is calculated simply by combining

individual RFM score numbers.

In [140]:
```
# RFM Quantiles
quantiles = rfm_table.quantile(q=[0.25,0.5,0.75])
quantiles
```

Out[140]:

	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

In [141]:
```
# Let's convert quartile information into a dictionary so that cutoffs ca
quantiles=quantiles.to_dict()
quantiles
rfm_table
```

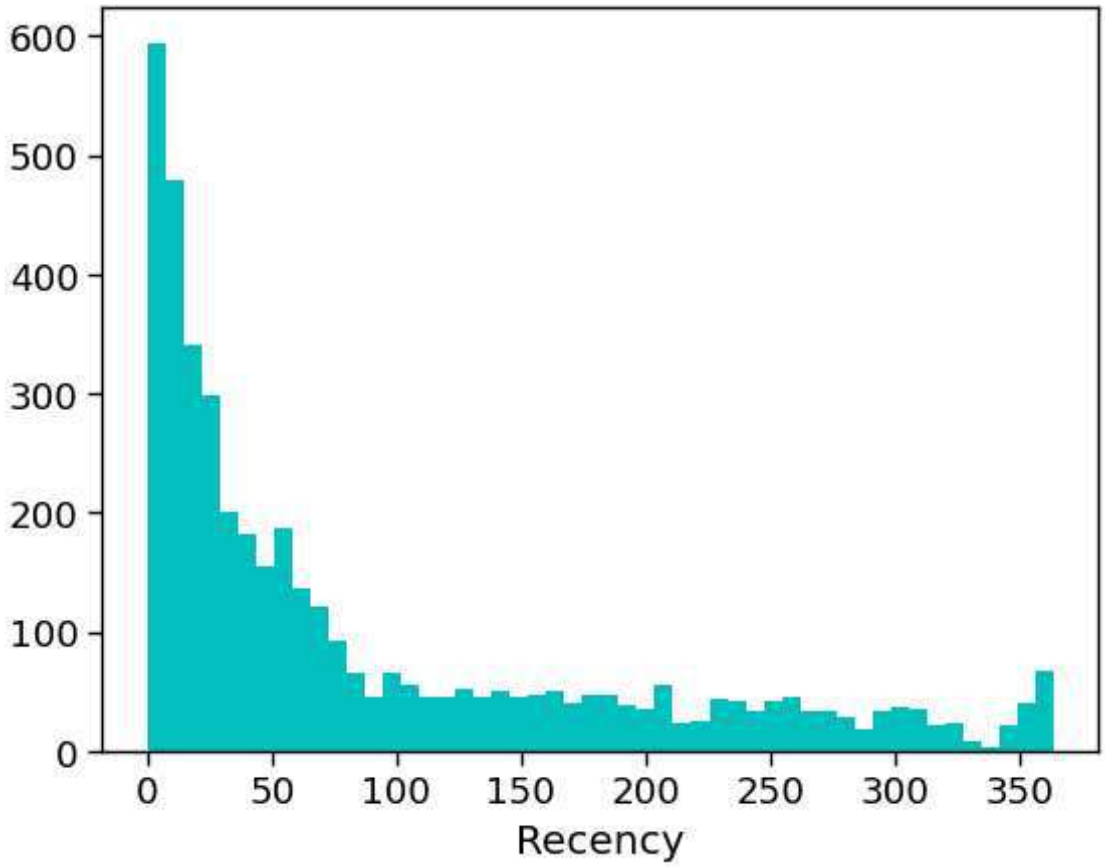
Out[141]:

	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

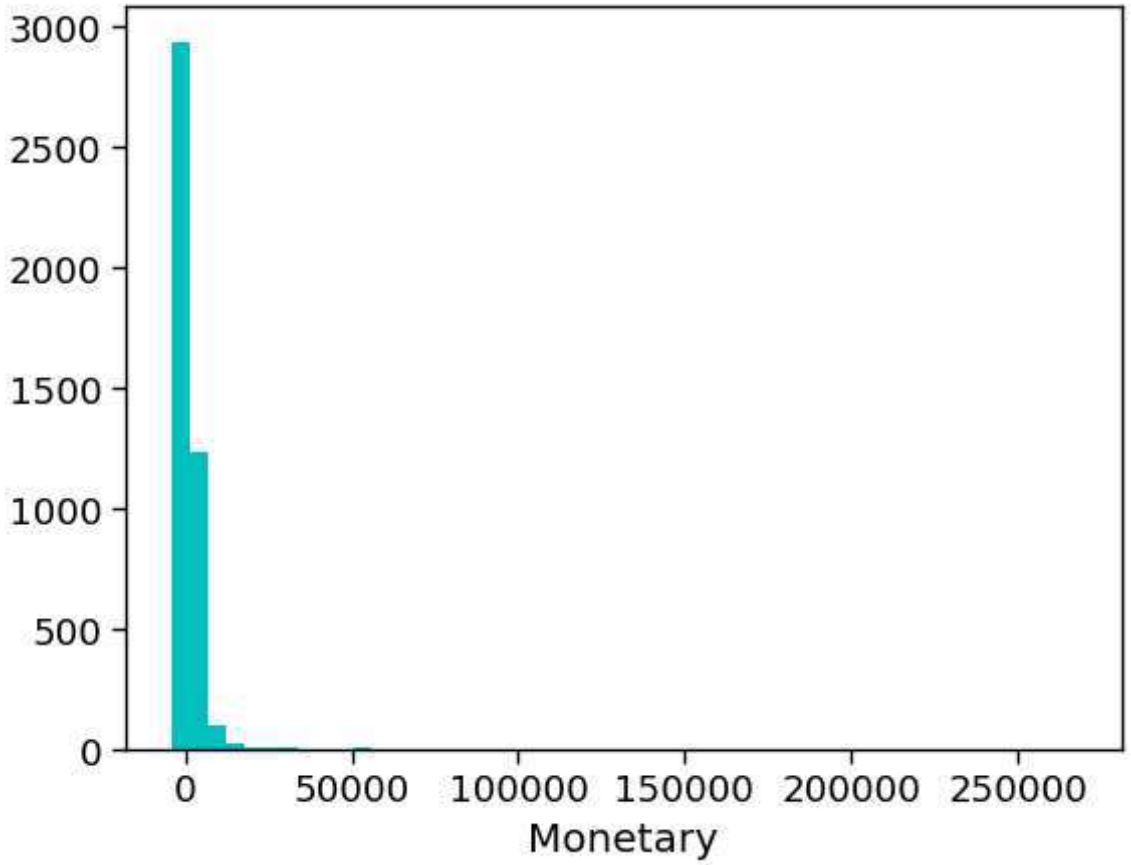
4331 rows × 3 columns

In [142]:

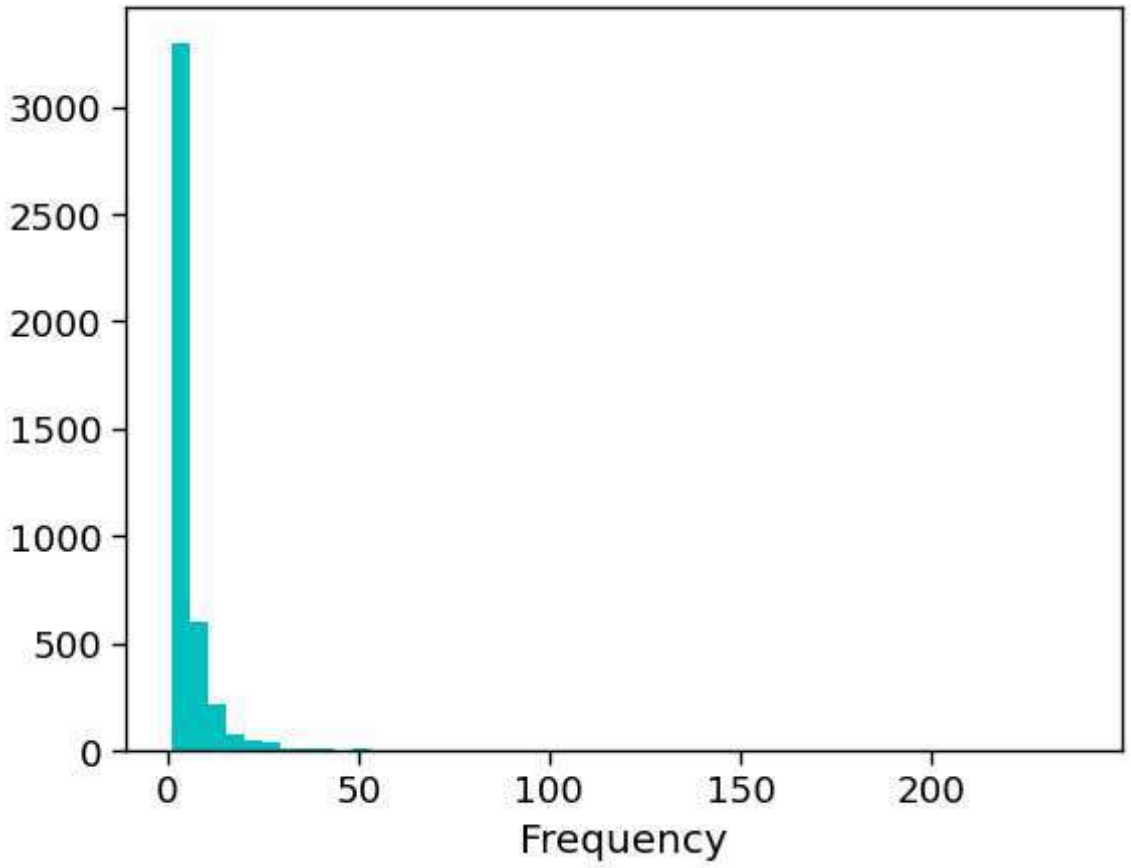
```
#Let us visualize the histogram charts for Recency, Frequency and Monetar
plt.hist(rfm_table.Recency, bins = 50, color='c')
plt.xlabel('Recency')
plt.show()
```



In [143]:
```
1 plt.hist(rfm_table.Monetary, bins = 50, color='c')
2 plt.xlabel('Monetary')
3 plt.show()
```



In [144]:
```
1 plt.hist(rfm_table.Frequency, bins = 50, color='c')
2 plt.xlabel('Frequency')
3 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

In [145]:
```
# Arguments (x = value, p = recency, monetary_value, frequency, d = quant
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 = quant
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=('R
rfm_segment['F_Quartile'] = rfm_segment['Frequency'].apply(FMScore, args=
rfm_segment['M_Quartile'] = rfm_segment['Monetary'].apply(FMScore, args=(
```

In [146]:
```
rfm_segment.head()
rfm_segment[rfm_segment.Monetary == rfm_segment.Monetary.max()]
rfm_segment
```

Out[146]:

CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile
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
...	...	...	...	...	...	...
18280.0	268	1	180.60	1	1	1
18281.0	171	1	80.82	1	1	1
18282.0	113	2	98.76	2	2	1
18283.0	0	15	1837.53	4	4	4
18287.0	33	3	1837.28	3	2	4

4331 rows × 6 columns

For analysis it is critical to combine the scores to create a single score. There are few approaches. One approach is to just concatenate the scores to create a 3 digit number between 111 and 444. Here the drawback is too many categories (4x4x4).

In [147]:
```
rfm_segment['RFMScore'] = rfm_segment.R_Quartile.map(str) \
                            + rfm_segment.F_Quartile.map(str) \
                            + rfm_segment.M_Quartile.map(str)
rfm_segment.head()
```

Out[147]:

CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScore
12346.0	316	2	0.00	1	2	1	121
12347.0	30	6	4085.18	3	4	4	344
12348.0	66	4	1797.24	2	3	4	234
12349.0	9	1	1757.55	4	1	4	414
12350.0	301	1	334.40	1	1	2	112

```
1.Best Recency score = 4 (most recently purchase)
2.Best Frequency score = 4 (most frequently purchase)
3.Best Monetary score = 4 (who spent the most)
```

RFM Segment allocation

Lets define the customers segment best to our knowledge basis RFM score and assign them to each customer respectively

In [148]:
```
# Reset the index to create a customer_ID column
rfm_segment.reset_index(inplace=True)
```

In [149]:

```
import itertools

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

# 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
#These are customers who transacted only once, but very recently and they

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


lowest_spending_active_loyal_customers_comb =  itertools.product([ 3,4],
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_spendi

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_custom




almost_lost = ['244', '234', '243', '233']        #  Low R - Customer's s
print ("Good Customers Almost Lost            : {}".format(almost_lost))

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


lost_cheap_customers = ['122','111' ,'121','112','221','212' ,'211'] # Cu
print ("Lost Cheap Customers                  : {}".format(lost_cheap_cu

```

```
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']
```

In [150]:

```
# 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_loy
    'Recent Customers': recent_customers,
    'Good Customers Almost Lost':almost_lost,
    'Churned Best Customers':   churned_best_customers,
    'Lost Cheap Customers ': lost_cheap_customers,
}
```

In [151]:

```
# 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)

```

Out[151]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
2384	15602.0	13	14	902.79	4	4	3	44
3979	17796.0	31	4	690.00	3	3	3	33
3707	17430.0	23	2	265.76	3	2	1	32
3660	17370.0	63	4	446.18	2	3	2	23
2028	15121.0	41	2	457.72	3	2	2	32
2832	16211.0	72	2	542.77	2	2	2	22
3583	17254.0	6	1	169.06	4	1	1	41
2125	15249.0	22	15	6657.36	3	4	4	34
3777	17524.0	352	1	145.00	1	1	1	11
4265	18200.0	97	1	150.35	2	1	1	21

Let's visualize different customer segments records in general to answers these questions for the retail business. Who are my best customers?
Who are the biggest spenders?
Which customers are at the verge of churning?
Who are lost customers that you don't need to pay much attention to?
Who are your loyal customers?
Which customers you must retain?
Who has the potential to be converted in more profitable customers?
Which group of customers is most likely to respond to your current campaign?

In [152]:
```
# Best Customers who's recency, frequency as well as monetary attribute i
rfm_segment[rfm_segment.RFMScore=='444'].sort_values('Monetary', ascendin
```

Out[152]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
1685	14646.0	7	74	267761.00	4	4	4	44
4193	18102.0	2	59	244952.95	4	4	4	44
3722	17450.0	1	54	185759.77	4	4	4	44
1876	14911.0	0	238	125482.36	4	4	4	44
54	12415.0	15	26	123725.45	4	4	4	44

In [153]:
```
# Biggest spenders
rfm_segment[rfm_segment.RFMScore=='334'].sort_values('Monetary', ascendin
```

Out[153]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
2765	16126.0	20	4	6287.77	3	3	4	33
12	12359.0	48	5	6274.23	3	3	4	33
727	13316.0	28	5	5570.69	3	3	4	33
2894	16303.0	16	4	5305.83	3	3	4	33
2868	16258.0	36	5	5203.51	3	3	4	33

In [154]:
```
# customers that you must retain are those whose monetary and frequency w
rfm_segment[rfm_segment.RFMScore=='244'].sort_values('Monetary', ascendin
```

Out[154]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
457	12939.0	55	8	11581.80	2	4	4	24
49	12409.0	69	7	11056.93	2	4	4	24
2807	16180.0	91	10	10217.48	2	4	4	24
1776	14769.0	68	9	10041.86	2	4	4	24
3215	16745.0	77	18	7157.10	2	4	4	24

In [155]:
```
rfm_segment.to_excel('RFM Segment.xlsx')
```

In [156]:
```
rfm_segment.Segment.value_counts()
rfm_segment.Recency
```

Out[156]:
```
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
```

Project Task: Week 3

Data Modeling :

1. Create clusters using k-means clustering algorithm.

a. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropriate transformation. Standardize the data.

b. Decide the optimum number of clusters to be formed.

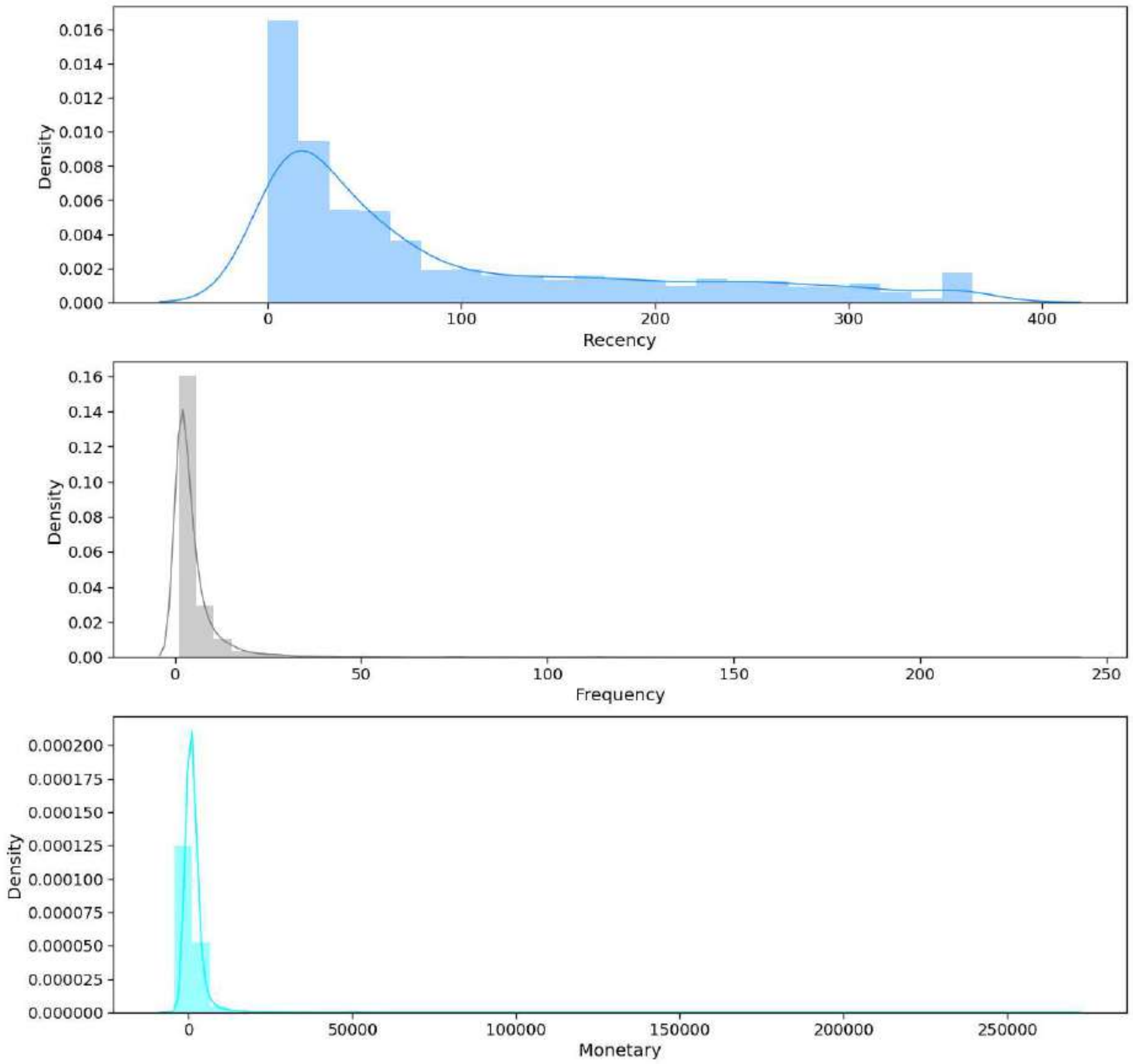
c. Analyze these clusters and comment on the results.

a. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropriate transformation. Standardize the data

In [157]:

```
# Distribution plot
fig, axes = plt.subplots(3, 1, figsize=(15, 15))
sns.distplot(rfm_table.Recency , color="dodgerblue", ax=axes[0], axlabel=
sns.distplot(rfm_table.Frequency , color="grey", ax=axes[1], axlabel='Fre
sns.distplot(rfm_table.Monetary , color="cyan", ax=axes[2], axlabel='Mone
plt.show();
```

```
C:\Users\ANANT KALEKAR\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level func
tion for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\ANANT KALEKAR\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level func
tion for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\ANANT KALEKAR\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level func
tion for histograms).
  warnings.warn(msg, FutureWarning)
```



Here we can observe that the data is highly skewed. So we have to transform and scale the data first because K-Means assumes that the variables should have a symmetric distributions(not skewed) and they should have same average values as well as same variance.

In [158]:
```
# Let's describe the table to see if there are any negative values
rfm_table.describe()
```

Out[158]:

	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

We can observe that Monetary contains negative values. So first we need to make sure that minimum range of value starts from 1 otherwise log transformation may lead to errors in graph plotting as well as K-Means clustering. After that we will utilize log transformation and scaling to make data available for for K-Means clustering

In [159]:
```
# 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 e
rfm_table_scaled.Monetary = rfm_table_scaled.Monetary + abs(rfm_table_sca
rfm_table_scaled.Recency = rfm_table_scaled.Recency + abs(rfm_table_scale

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

Out[159]:

	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

In [160]:
```
# 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=r
```

In [161]:
```
normal_df
```

Out[161]:

CustomerID	Recency	Frequency	Monetary
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

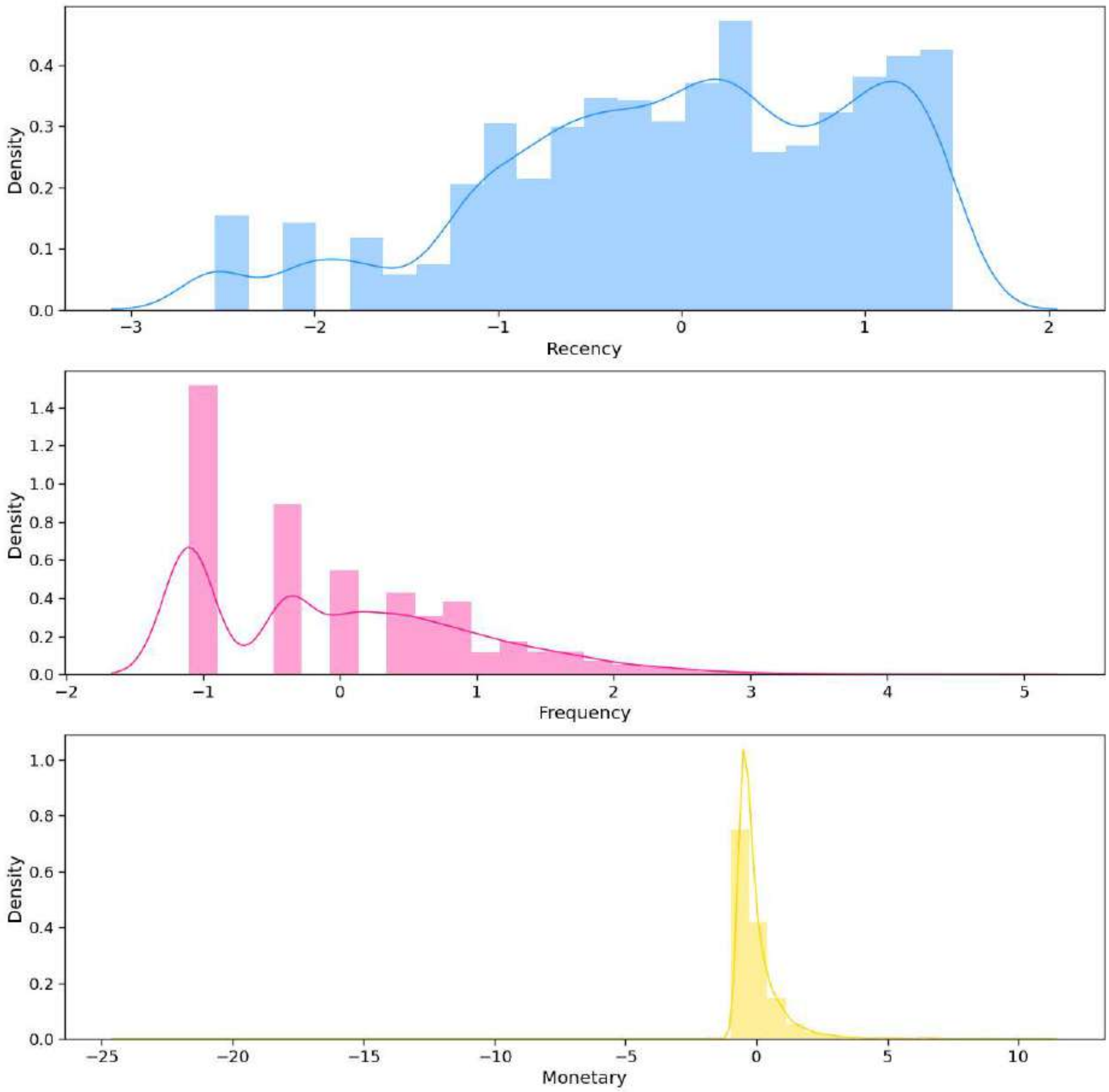
4331 rows × 3 columns

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

In [162]:

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

```
C:\Users\ANANT KALEKAR\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level func
tion for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\ANANT KALEKAR\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level func
tion for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\ANANT KALEKAR\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level func
tion for histograms).
  warnings.warn(msg, FutureWarning)
```

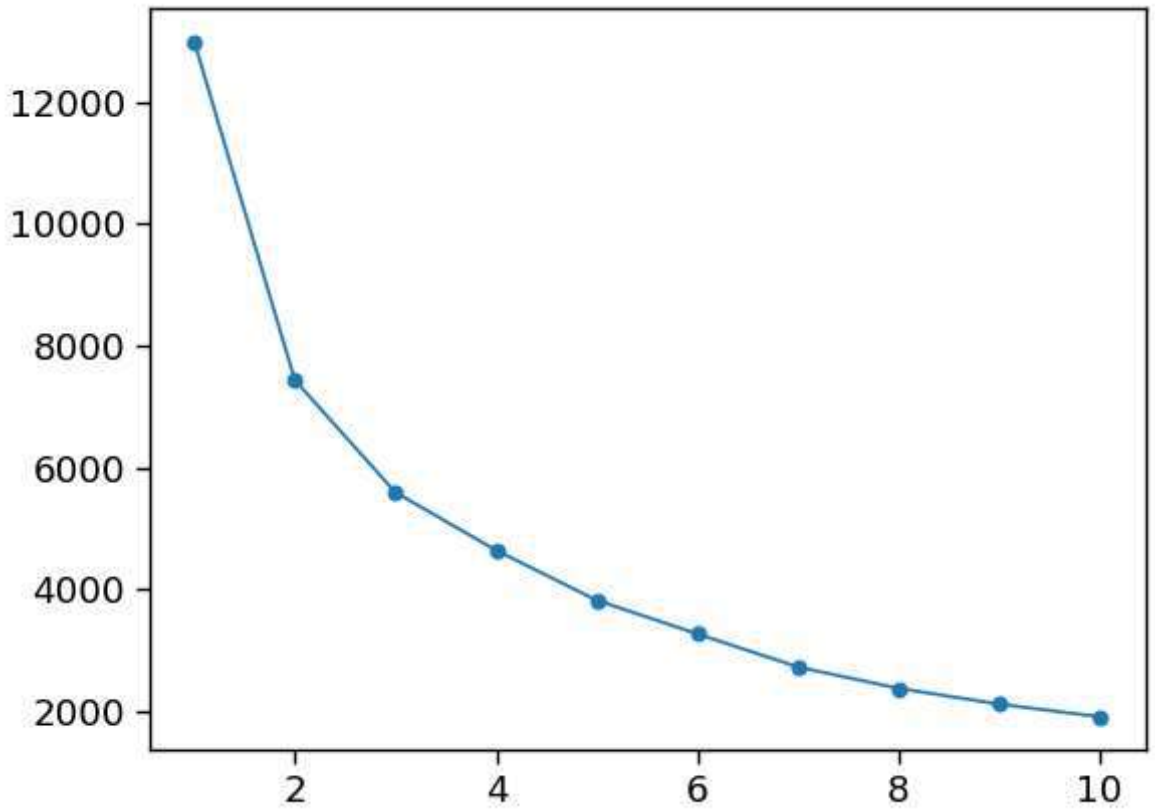


We can observe that the means & averages are approximately uniformed now in each distribution. Now the data is apt for unsupervised algo i.e. K-Means. Lets try to find number of appropriate clusters to divide customers as per there spending pattern with elbow method first

b. Decide the optimum number of clusters to be formed.

b. 1 WCSS - Within Cluster Sum of Squares (WCSS)

In [163]:
```
# 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');

```



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

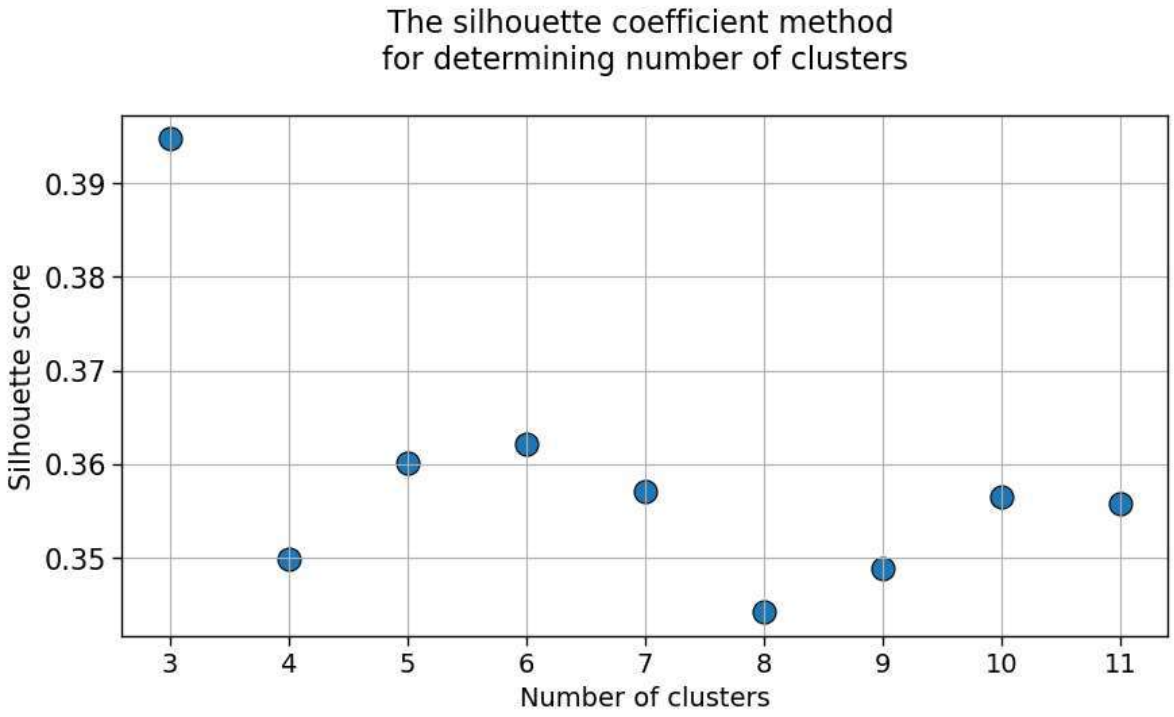
In [164]:
```
ElbowPlot = pd.DataFrame({'Cluster': range(1,11) , 'SSE': wcss})
ElbowPlot.to_excel('Elbow Plot Data.xlsx')
```

b.2 Silhouette Score

In [165]:

```
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
    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,sil

plt.figure(figsize=(10,5))
plt.title("The silhouette coefficient method \nfor determining number of
plt.scatter(x=[i for i in range(3,12)],y=wcss_silhouette,s=150,edgecolor=
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()
```

```
Silhouette score for number of cluster(s) 3: 0.3947454614514734
Silhouette score for number of cluster(s) 4: 0.34984009502580043
Silhouette score for number of cluster(s) 5: 0.3600617831786827
Silhouette score for number of cluster(s) 6: 0.36209423197290214
Silhouette score for number of cluster(s) 7: 0.3570394541175927
Silhouette score for number of cluster(s) 8: 0.3441550819272379
Silhouette score for number of cluster(s) 9: 0.3489031730303187
Silhouette score for number of cluster(s) 10: 0.3565626808393854
Silhouette score for number of cluster(s) 11: 0.3557950595147288
```



In [ ]:

```
1
```

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

In [166]:
```
kmeans = KMeans(n_clusters=4, random_state=1, init='k-means++')
kmeans.fit(normal_df)
cluster_labels = kmeans.labels_
```

In [167]:
```
kmeans
```

Out[167]:
```
KMeans(n_clusters=4, random_state=1)
```

In [168]:
```
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)
```

In [169]:
```
# Assign the clusters as column to each customer
Cluster_table = rfm_segment.assign(Cluster = cluster_labels)
```

In [170]:
```
# Check counts of records assigned to different clusters
Cluster_table.Cluster.value_counts()
```

Out[170]:
```
2    1941
3    1187
0    1013
1     190
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

In [171]:
```
Cluster_table.sample(10)
print ("Platinum customers belong to cluster                    : {} ".
print ("Big Spenders belong to cluster                          : {} ".
print ("High Spend new Customers belong to cluster              : {} ".
print ("Lowest-Spending Active Loyal Customers belong to cluster  : {} ".
print ("Recent Customers belong to cluster                      : {} ".
print ("Good Customers Almost Lost belong to cluster            : {} ".
print ("Churned Best Customers belong to cluster                : {} ".
print ("Lost Cheap customers belong to cluster                  : {} ".

```

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

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

In [172]:
```
1 Cluster_table[Cluster_table.Cluster == 3].sample(5)
```

Out[172]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
3876	17659.0	8	13	2968.86	4	4	4	444
3340	16912.0	14	14	2508.87	4	4	4	444
1183	13953.0	43	6	3104.36	3	4	4	344
787	13397.0	63	6	1539.18	2	4	3	243
1069	13802.0	129	6	3906.27	2	4	4	244

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.

In [173]:
```
1 Cluster_table[Cluster_table.Cluster == 2].sample(5)
```

Out[173]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
4216	18133.0	202	2	715.50	1	2	3	123
3376	16957.0	230	2	416.64	1	2	2	122
3822	17588.0	91	3	1240.20	2	2	3	223
1268	14068.0	311	1	265.85	1	1	1	111
2434	15665.0	159	2	2222.21	1	2	4	124

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

In [174]:
```
1 Cluster_table[Cluster_table.Cluster == 1].sample(5)
```

Out[174]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
3272	16814.0	5	22	6408.20	4	4	4	44
323	12748.0	1	215	27406.56	4	4	4	44
1261	14057.0	14	18	6141.10	4	4	4	44
4025	17865.0	21	27	10158.54	3	4	4	34
1281	14088.0	1	14	50415.49	4	4	4	44

Cluster 1 with very high monetary value along with good frequency and recency values. These are the most valuable customers to the firm. They should be looked after periodically to access there concerns

In [175]:
```
1 Cluster_table[Cluster_table.Cluster == 0].sample(5)
```

Out[175]:

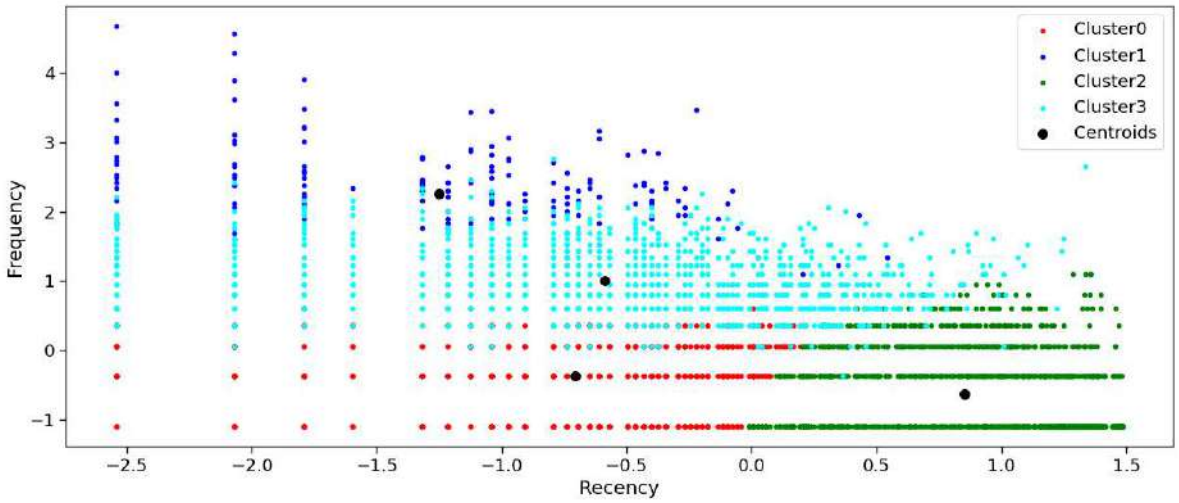
	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScor
645	13203.0	0	4	869.46	4	3	3	43
1632	14567.0	26	3	1524.46	3	2	3	32
356	12795.0	9	1	430.48	4	1	2	41
105	12479.0	2	1	527.20	4	1	2	41
1245	14039.0	24	1	152.20	3	1	1	31

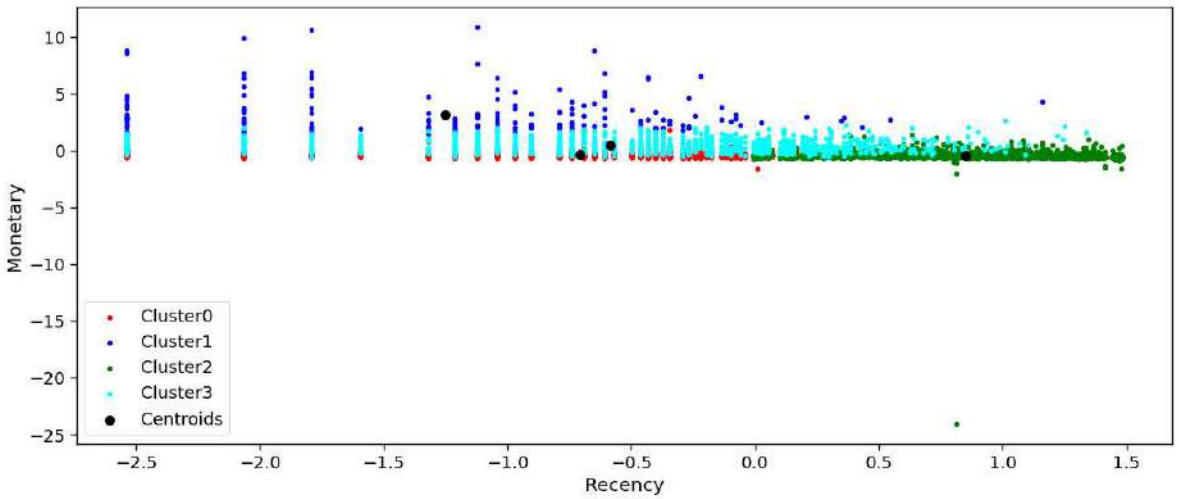
Cluster 0 is somewhat average collectively can respond to the targeted campaigns.

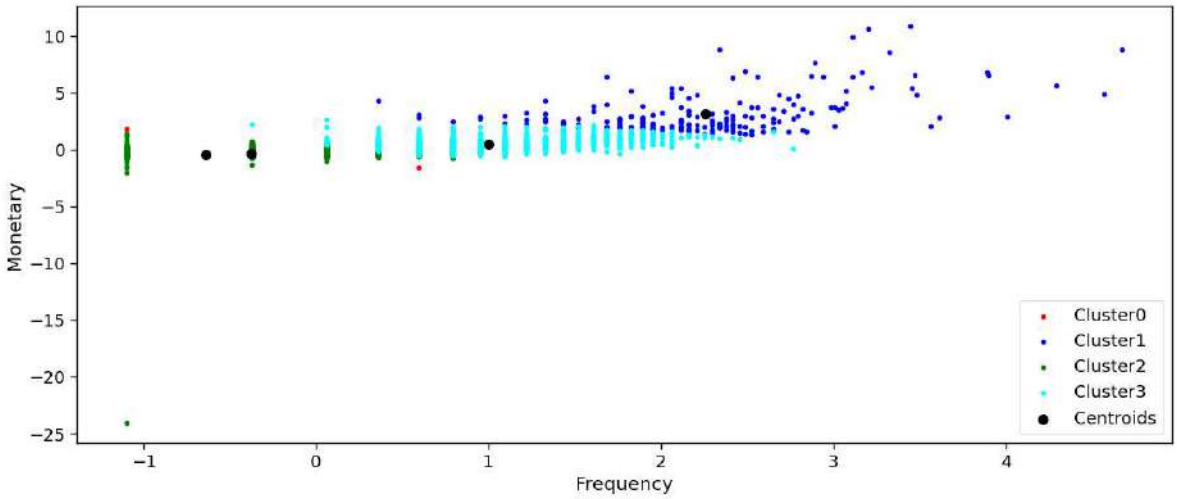
Scatter Plot to visualize the division of customers into different segments based on the RFM atributes.

In [176]:

```
# 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],
        plt.scatter(X[cluster_labels == 1, i], X[cluster_labels == 1, j],
        plt.scatter(X[cluster_labels == 2, i], X[cluster_labels == 2, j],
        plt.scatter(X[cluster_labels == 3, i], X[cluster_labels == 3, j],
        plt.scatter(kmeans.cluster_centers_[:,i], kmeans.cluster_centers_
        plt.xlabel(normal_df.columns[i])
        plt.ylabel(normal_df.columns[j])
        plt.legend()
        plt.show();
```







In [177]:
```
Cluster_table.to_excel('RFMSegment.xlsx')
```

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

Heat Map

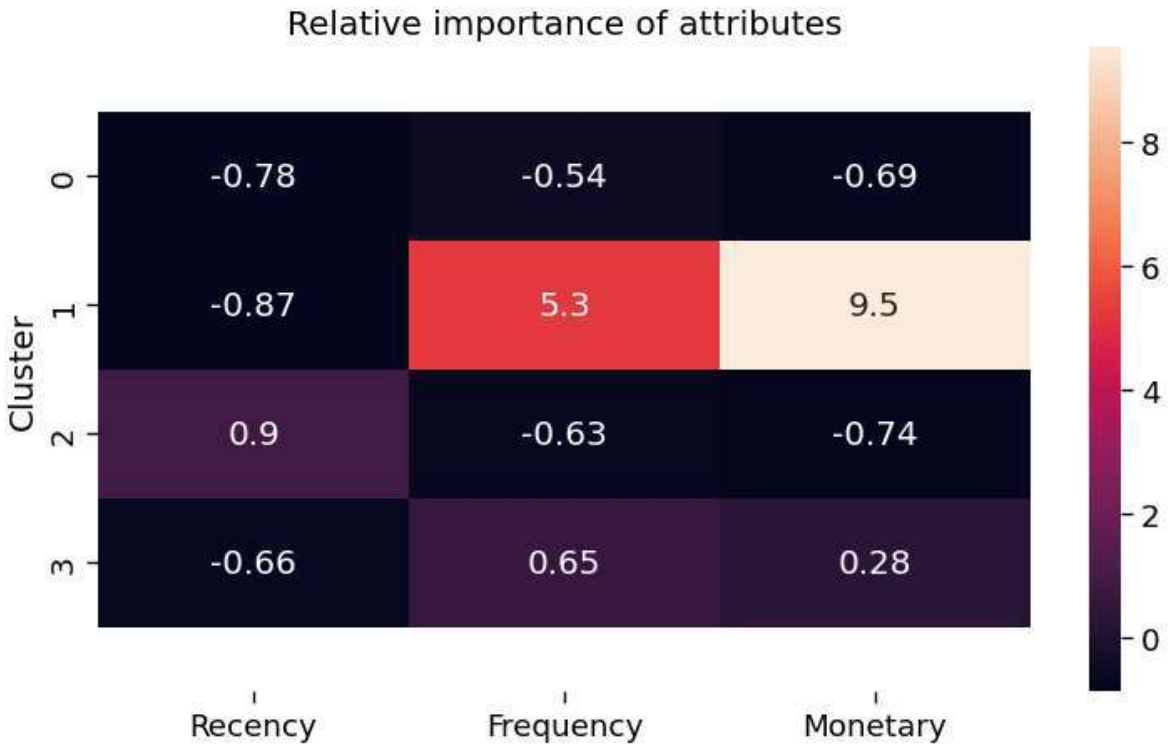
We will utilize heat map to visualize the relative importance of each attributes in all four customer segments i.e. clusters. It calculates importance score by dividing them and subtracting 1 (ensures 0 is returned when cluster average equals population average).

The farther a ratio is from 0, the more important that attribute is for a segment relative to the total population.

In [178]:
```
# 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

```

In [179]:

```
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();
```



TABLEAU

Project Task: Week 4

Data Reporting:

1. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

   a. Country-wise analysis to demonstrate average spend. Use a bar chart to show the monthly figures

   b. Bar graph of top 15 products which are mostly ordered by the users to show the number of products sold

   c. Bar graph to show the count of orders vs. hours throughout the day

   d. Plot the distribution of RFM values using histogram and frequency charts

   e. Plot error (cost) vs. number of clusters selected

   f. Visualize to compare the RFM values of the clusters using heatmap

In [180]:
```
1 data.to_excel('TableauSource.xlsx')
```

User Interactive Online Retail Story Board for UK Retail Store

1. Retail Dashboard

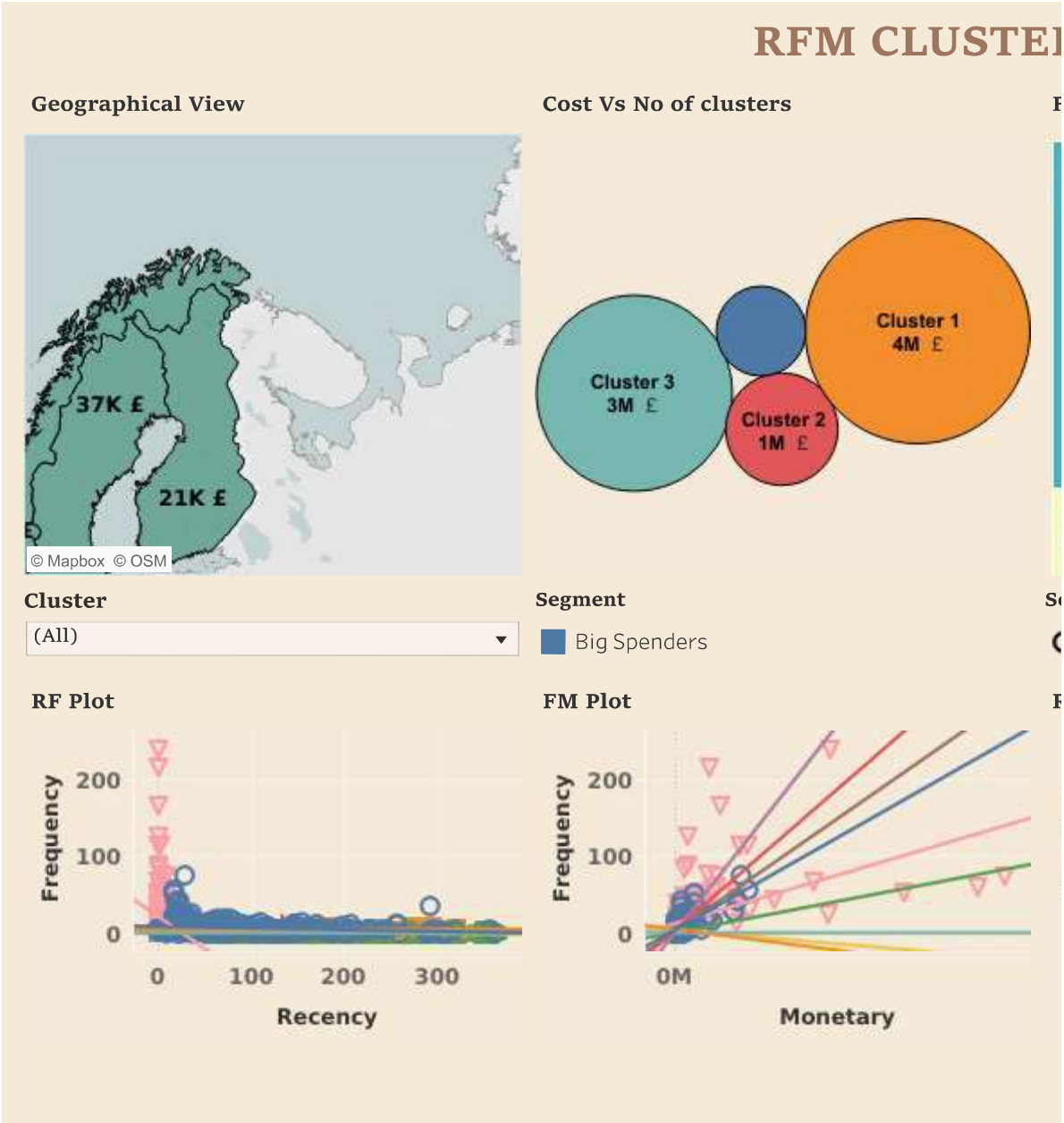
a. Country Wise Analysis
b. Top Products by Sales
c. Top Products by Count
d. Monthly Figures
e.Count of orders Vs Hours Throughout the Day
f. Elbow Plot -Error Cost against the no of clusters
g. Recency Histogram
h. Customer Segments
i.FM Heat Map
j. RM HEat Map

2. RFM Cluster Analysis Dashboard

a. Geaographical Viz
b. Cost Vs No of clusters
c. Frequency Sum Vs Clusters
d. RF Heat Map
e. RF Plot
f. FM Plot
g. RM Plot
h. Cluster View

In [181]:

```
%%HTML
<div class='tableauPlaceholder' id='viz1595687001456' style='position: re
<a href='#'><img alt=' ' src='https:&#47;&#47;public.tableau.com&#47;stat
</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' val
<param name='name' value='OnlineRetailCapstoneVisualization-Reena&#47;RET
<param name='tabs' value='yes' /><param name='toolbar' value='yes' />
<param name='static_image' value='https:&#47;&#47;public.tableau.com&#47;
<param name='animate_transition' value='yes' /><param name='display_stati
<param name='display_spinner' value='yes' /><param name='display_overlay'
<param name='display_count' value='yes' /><param name='language' value='e
<param name='filter' value='publish=yes' /></object></div>
<script type='text/javascript'>
var divElement = document.getElementById('viz1595687001456');
var vizElement = divElement.getElementsByTagName('object')[0];
if ( divElement.offsetWidth > 800 ) { vizElement.style.minWidth='1155px';
else if ( divElement.offsetWidth > 500 ) { vizElement.style.minWidth='115
else { vizElement.style.width='100%';vizElement.style.minHeight='2750px';
var scriptElement = document.createElement('script');
scriptElement.src = 'https://public.tableau.com/javascripts/api/viz_v1.js
vizElement.parentNode.insertBefore(scriptElement, vizElement);
</script>
```



Conclusion:

It i critical requirement for business to understand the value derived from a customer. RFM and cohort analysis is a method used for analyzing customer value. Business optimisation can be achieved with the above RFM customer segmentation with having segregated the customer base into groups of individuals based on well defined characteristics and traits. Visualization is added to implement the user story with relevant charts. Necessary promotion campaigns with agressive price incentives and discounts can help monitor customer attrition.