Capstone Project - Retails

It is a business critical requirement to understand the value derived from a customer. RFM is a method used for analyzing customer value

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). Identifying the most valuable RFM segments can capitalize on chance relationships in the data used for this analysis.

Tushar Bhave
6/11/2022

1. Perform the preliminary Data Inspection and Cleaning

1.1 Import the Necessary library

Let's import the necessary library like pandas, numpy etc.

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    import datetime as dt
    from textwrap import wrap
```

```
[2] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=1
```

Note: Since we are working in Google colab it is important to mount the Google drive where the data is stored

1.2 Import the dataset and analyse it

```
[ ] retail_df = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/Simplilearn Projects/Data Science Capstone project/Retai

retail_df.head()
```

	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

Online_retail dataset have a shape of (541909,8) have 8 features and 541909 records

```
[ ] # Inspect the missing values in the datasets

    print(retail_df.shape)

    (541909, 8)
```

```
[ ] retail_df.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 541909 entries, 0 to 541908
    Data columns (total 8 columns):
     #   Column       Non-Null Count   Dtype
    ---  ------       --------------   -----
     0   InvoiceNo    541909 non-null  object
     1   StockCode    541909 non-null  object
     2   Description  540455 non-null  object
     3   Quantity     541909 non-null  int64
     4   InvoiceDate  541909 non-null  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
```

We have 1 Datetime features (InvoiveDate), 2 Float features (unitPrice, CustomerID), 4 object features (InvoiceNo, StockCode, Description, Country).

Although the customerId should not be float as we are not performing any arithmetic operation on it so we should convert it to object as it is used to identify the customer details

```
[ ] retail_df1['CustomerID'] = retail_df1['CustomerID'].astype(str)
```

1.3 Check for missing values:

Let's check for the missing values using is.na () command. We have 2 features that have missing values i.e.
Description: These features have 1454 missing value (0.27%). For this, we can just drop the features as these missing values are miniscule. We can safely drop the column since we already have Stock code columns that is not null

```
[11] retail_df.isna().sum()

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

1. CustomerID: Customer id have 135080 missing values(25%). But customerId is very important variable and 25% means 1/4 of the data so we need to make sure there is nothing we can do before dropping the data

```
[13] invoice_null_custid = retail_df[retail_df['CustomerID'].isna()]['InvoiceNo'].unique()

[14] retail_df[retail_df['InvoiceNo'].isin(invoice_null_custid) & (~retail_df['CustomerID'].isna())]

InvoiceNo StockCode Description Quantity InvoiceDate UnitPrice CustomerID Country
```

We tried to find the customer id with help of the invoice number to see if we could get any customerId that we can impute but nothing came
perhaps the customer refused to share the customer details
So we have no way to impute the customerId details so we can drop the rows where customerid is null

```
[15] retail_df1 = retail_df.drop('Description', axis=1)

[16] retail_df1 = retail_df1.dropna()
     retail_df1.shape
```

Note: We created a new dataframe as retail_df1 so as to preserve the original

1.4 Removing the Duplicates

For removing the duplicates we see all the records to see if we have any records that are repeated.

```
[17] retail_df1 = retail_df1.drop_duplicates()
     retail_df1.shape

     (401602, 7)
```

So we have 5227 records that was removed

1.5 Descriptive analysis:

1.5.1 Descriptive analysis for Float features: For int and float features we can simple use .describe()

```
[20] retail_df1.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Quantity	401602.0	12.182579	250.283248	-80995.0	2.00	5.00	12.00	80995.0
UnitPrice	401602.0	3.474064	69.764209	0.0	1.25	1.95	3.75	38970.0

- As we can see that total on avg. people buy 12 quantities of goods from store. Also we see that min quantity is in negative so there are some invoices for returning of items

- Unit price: Average unit price of the items is 3 and max is 38970 orders

1.5.2 Descriptive analysis for Object type features: For object type feature just give the parameter include='O'

```
retail_df1.describe(include=['O']).T
```

	count	unique	top	freq
InvoiceNo	401602	22190	576339	542
StockCode	401602	3684	85123A	2065
CustomerID	401602	4372	17841.0	7812
Country	401602	37	United Kingdom	356726

2 Cohort Analyses:

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.

2.1 Create a monthly cohort and analyse active customer in each cohort

- First lets create the function that returns the month of the function using datetime library and passed the InvoiceDate and the Invoice_month information to the dataframe

```
[22] def ret_month(x): return dt.datetime(x.year, x.month, 1)

     # Create a invoice date column based on the invoice month and store it in invoice_month

     retail_df1['invoice_month'] = retail_df1['InvoiceDate'].apply(ret_month)
```

```
[23] retail_df1.head()
```

	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	invoice_month
0	536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01
1	536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01
2	536365	84406B	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01
3	536365	84029G	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01
4	536365	84029E	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01

- Now let's group the records with customerID and select the invoiceMonth and Assign the min values of the invoice month to the dataset

```
[24] # Group the customerId and select the invoice_month

     grouping = retail_df1.groupby('CustomerID')['invoice_month']

     # Assign the min Invoice month value to the dataset

     retail_df1['cohortMonth'] = grouping.transform('min')

     print(retail_df1.head())
```

```
  InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice CustomerID  \
0    536365    85123A         6 2010-12-01 08:26:00       2.55    17850.0
1    536365     71053         6 2010-12-01 08:26:00       3.39    17850.0
2    536365    84406B         8 2010-12-01 08:26:00       2.75    17850.0
3    536365    84029G         6 2010-12-01 08:26:00       3.39    17850.0
4    536365    84029E         6 2010-12-01 08:26:00       3.39    17850.0

          Country invoice_month cohortMonth
0  United Kingdom    2010-12-01  2010-12-01
1  United Kingdom    2010-12-01  2010-12-01
2  United Kingdom    2010-12-01  2010-12-01
3  United Kingdom    2010-12-01  2010-12-01
```

- Calculating the time offset for each transaction allows you to evaluate the metrics for each cohort in a comparable fashion.
- First, we will create 6 variables that capture the integer value of years, months, and days for Transaction and Cohort Date using the get_date_int() function.

```
[25] def get_date_int(df, columns):
       year = df[columns].dt.year
       month = df[columns].dt.month
       day = df[columns].dt.day
       return year, month, day


     invoice_year, invoice_month, _ = get_date_int(retail_df1, 'invoice_month')
     cohort_year, cohort_month, _ = get_date_int(retail_df1, 'cohortMonth')
```

- Now we'll calculate the difference between cohort month and invoice month and same as year to get the cohort offset which will be used to calculate the retention rate

```
[69] # get the difference in year

     year_diff = invoice_year - cohort_year

     # Get the difference in months
     month_diff = invoice_month - cohort_month

     retail_df1['CohortIndex'] = year_diff*12 + month_diff + 1

     print(retail_df1.head())
```

```
  InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice  CustomerID  \
0    536365    85123A         6 2010-12-01 08:26:00       2.55     17850.0
1    536365     71053         6 2010-12-01 08:26:00       3.39     17850.0
2    536365    84406B         8 2010-12-01 08:26:00       2.75     17850.0
3    536365    84029G         6 2010-12-01 08:26:00       3.39     17850.0
4    536365    84029E         6 2010-12-01 08:26:00       3.39     17850.0

          Country invoice_month cohortMonth  CohortIndex  Total_sales
0  United Kingdom    2010-12-01  2010-12-01            1        15.30
1  United Kingdom    2010-12-01  2010-12-01            1        20.34
2  United Kingdom    2010-12-01  2010-12-01            1        22.00
3  United Kingdom    2010-12-01  2010-12-01            1        20.34
4  United Kingdom    2010-12-01  2010-12-01            1        20.34
```

- Now we have CohortIndex and CohortMonth and we will use this information to count the number of active users in each cohort by grouping the records with Cohort month and CohortIndex
- Count the unique customerId falling in this category
- Use the pivot table to get the necessary information

```
[28] # Counting the daily active users in each cohorts

     grouping = retail_df1.groupby(['cohortMonth', 'CohortIndex'])

     # Counting the no of unique customer id falling in each group

     cohort_data = grouping['CustomerID'].apply(pd.Series.nunique)
     cohort_data = cohort_data.reset_index()

     # Assign the column names to the dataframe created above

     cohort_counts = cohort_data.pivot_table(index = 'cohortMonth',
                                             columns = 'CohortIndex',
                                             values = 'CustomerID')
```

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
cohortMonth													
2010-12-01	948.0	362.0	317.0	367.0	341.0	376.0	360.0	336.0	336.0	374.0	354.0	474.0	260.0
2011-01-01	421.0	101.0	119.0	102.0	138.0	126.0	110.0	108.0	131.0	146.0	155.0	63.0	NaN
2011-02-01	380.0	94.0	73.0	106.0	102.0	94.0	97.0	107.0	98.0	119.0	35.0	NaN	NaN
2011-03-01	440.0	84.0	112.0	96.0	102.0	78.0	116.0	105.0	127.0	39.0	NaN	NaN	NaN
2011-04-01	299.0	68.0	66.0	63.0	62.0	71.0	69.0	78.0	25.0	NaN	NaN	NaN	NaN
2011-05-01	279.0	66.0	48.0	48.0	60.0	68.0	74.0	29.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	235.0	49.0	44.0	64.0	58.0	79.0	24.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	191.0	40.0	39.0	44.0	52.0	22.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	167.0	42.0	42.0	42.0	23.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	298.0	89.0	97.0	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	352.0	93.0	46.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	321.0	43.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

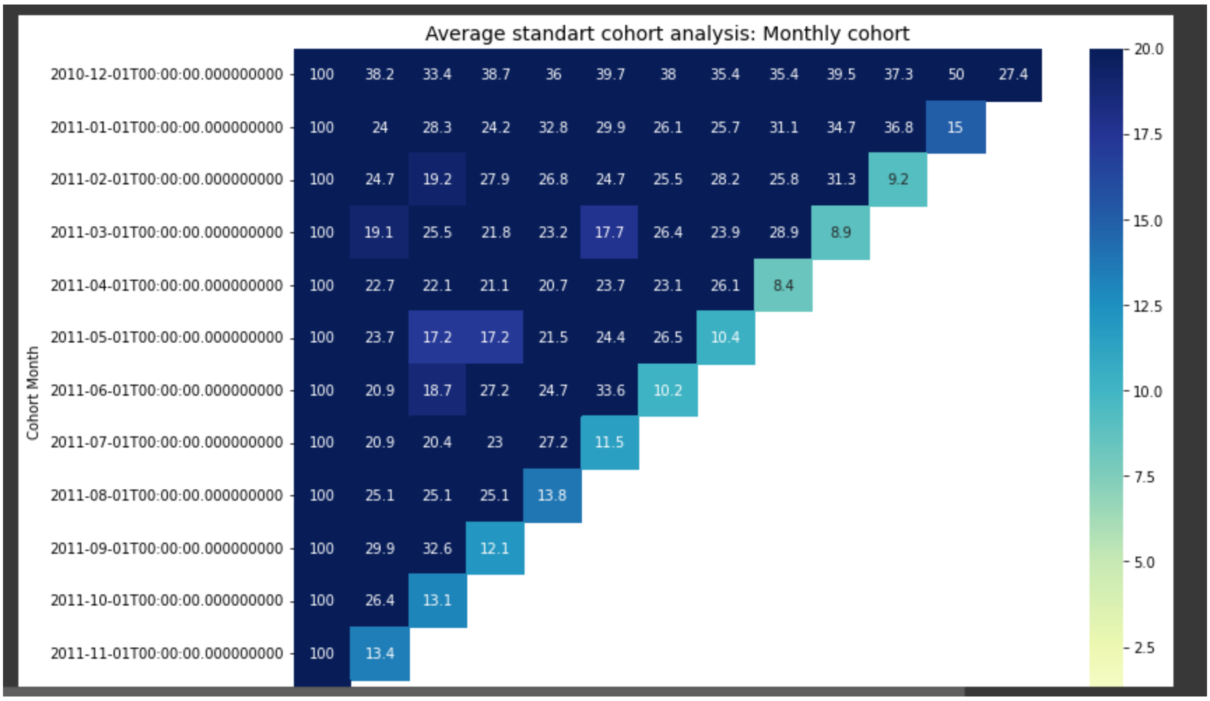
By using this analysis we get the number of active users for each month and we can see the trend for ex. We had 978 active users by Dec-2010 which was reduced to 260 by the 2011.

2.2 Calculating the Retention Rate:

Retention rate is the percentage of active users/customer compared to the total number of users/customer at any specific time period with the range

```
# Calculate the retention rate
# Retention rate: The percentage of active customers compared to the total number of customers after a specific time inter

cohort_size = cohort_counts.iloc[:,0]

retention = cohort_counts.divide(cohort_size, axis =0)
average_standard_cost = retention.round(3)*100
```

```
[31] # average_standard_cost.index = average_standard_cost.index.strftime('%Y-%m')

     # Vizualising the retention rate

     plt.figure(figsize= (12,9))
     plt.title("Average standart cohort analysis: Monthly cohort", fontsize=14)
     sns.heatmap(average_standard_cost, annot = True,vmin = 0.0, vmax =20,cmap="YlGnBu", fmt='g')
     plt.ylabel('Cohort Month')
     plt.xlabel('Cohort Index')
     plt.yticks( rotation='360')
     plt.show()
```



3. Build the RFM Model:

RFM stands for recency, frequency, monetary value. In business analytics, we often use this concept to divide customers into different segments, like high-value customers, medium value customers or low-value customers, and similarly many others.

3.1 Calculating the RFM Metrics:

3.1.1 Recency: How recently the customer made the purchase

```
[33] df_recency = retail_df1.groupby(by='CustomerID', as_index = False)['invoice_month'].max()
     df_recency.columns = ['CustomerID', 'last_purchase']

     recent = df_recency['last_purchase'].max()

     df_recency['Recency'] = df_recency['last_purchase'].apply(
         lambda x: (recent - x).days
     )
```

```
[34] df_recency.head()
```

	CustomerID	last_purchase	Recency
0	12346.0	2011-01-01	334
1	12347.0	2011-12-01	0
2	12348.0	2011-09-01	91
3	12349.0	2011-11-01	30
4	12350.0	2011-02-01	303

3.1.2 Frequency: How frequently the customer buys from the site

```
# Calculating the frequency
# Here we are calculating the frequent transaction of purchase of any items

df_frequency = retail_df1.groupby(by='CustomerID', as_index = False)['invoice_month'].count()
df_frequency.columns = ['CustomerID', 'frequency']
```

```
[36] df_frequency
```

	CustomerID	frequency
0	12346.0	2
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17
...	...	...
4367	18280.0	10
4368	18281.0	7

3.1.2 Monetary: How much the customer spends for the purchase in the time period

```
[37] retail_df1['Total_sales'] = retail_df1['Quantity']* retail_df1['UnitPrice']

     df_monetary = retail_df1.groupby(by='CustomerID', as_index = False)['Total_sales'].sum()
     df_monetary.columns = ['CustomerID', 'Monetary']
```

```
[38] df_monetary
```

	CustomerID	Monetary
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40
...	...	...
4367	18280.0	180.60
4368	18281.0	80.82
4369	18282.0	176.60
4370	18283.0	2045.53
4371	18287.0	1837.28

4372 rows × 2 columns

3.2 Merging into one RFM metrics:

We will merge the entire RFM individual component into one dataset df_rfm based on the customerID

Merging into one RFM metrics

```
[39] df_rf = df_recency.merge(df_frequency ,on='CustomerID')
     df_rfm = df_rf.merge(df_monetary, on='CustomerID')

     df_rfm.head()
```

	CustomerID	last_purchase	Recency	frequency	Monetary
0	12346.0	2011-01-01	334	2	0.00
1	12347.0	2011-12-01	0	182	4310.00
2	12348.0	2011-09-01	91	31	1797.24
3	12349.0	2011-11-01	30	73	1757.55
4	12350.0	2011-02-01	303	17	334.40

Now we have how recently the customer made the purchase, how frequently the customer made the purchase and how much the customer has spent over time

Let's group these into quantile using qcut()

```
[42] # Calculating the R-group

     # Grouping R-values
     r_label = range(4,0,-1)
     r_group = pd.qcut(df_rfm['Recency'], q=4, labels = r_label)

     # Grouping F-values

     f_label = range(1,5)
     f_group = pd.qcut(df_rfm['frequency'], q=4, labels = f_label)

     # Grouping M-values

     m_label = range(1,5)
     m_group = pd.qcut(df_rfm['Monetary'], q=4, labels = m_label)

     # Create the new columns for R,F,M

     df_rfm = df_rfm.assign(R = r_group.values, F = f_group.values, M = m_group.values)

     df_rfm.head()
```

Note: We have given the range of recency as -1 as more recently the customer buys i.e. less the values of recency is good for us

	CustomerID	last_purchase	Recency	frequency	Monetary	R	F	M
0	12346.0	2011-01-01	334	2	0.00	1	1	1
1	12347.0	2011-12-01	0	182	4310.00	4	4	4
2	12348.0	2011-09-01	91	31	1797.24	2	2	4
3	12349.0	2011-11-01	30	73	1757.55	4	3	4
4	12350.0	2011-02-01	303	17	334.40	1	1	2

We created 4 labels for r_label, f_labels, m_labels where 4th is the best quantile and 1 is the least quantile. We then created the R, F, M columns and assign the values of the group

Finally with these R, F, M score we can segment the customer by first adding the individual score together

```
df_rfm['RFM_score']= df_rfm[['R','F','M']].sum(axis=1)
df_rfm.head(15)
```

	CustomerID	last_purchase	Recency	frequency	Monetary	R	F	M	RFM_score
0	12346.0	2011-01-01	334	2	0.00	1	1	1	3
1	12347.0	2011-12-01	0	182	4310.00	4	4	4	12
2	12348.0	2011-09-01	91	31	1797.24	2	2	4	8
3	12349.0	2011-11-01	30	73	1757.55	4	3	4	11
4	12350.0	2011-02-01	303	17	334.40	1	1	2	4
5	12352.0	2011-11-01	30	95	1545.41	4	3	3	10
6	12353.0	2011-05-01	214	4	89.00	1	1	1	3
7	12354.0	2011-04-01	244	58	1079.40	1	3	3	7

```
[45] df_rfm.groupby('RFM_score')['RFM_score'].sum()

RFM_score
3      1101
4      1328
5      2190
6      2892
7      2786
8      3888
9      3348
10     4840
11     4081
12     7704
Name: RFM_score, dtype: int64
```

Now let's define the range which we will use for Customer Segmentation

- RFM_score > 9 : Most Valued customer
- RFM_score >= 8 and < 9 : Champion customer
- RFM_score >= 8 and < 8 : Loyal customer
- RFM_score >= 6 and < 7 : Potential customer
- RFM_score >=5 and < 6 : Promising customer
- RFM_score >=4 and < 5 : Needs Attention
- RFM_score < 3 : Require activation

```
[46] def rfm_level(df):
    if df['RFM_score'] >= 9 :
      return 'cant loose them'
    elif (df['RFM_score']) >= 8 and (df['RFM_score']) < 9 :
      return 'Champion'
    elif (df['RFM_score']) >= 7 and (df['RFM_score']) < 8 :
      return 'Loyal Customer'
    elif (df['RFM_score']) >= 6 and (df['RFM_score']) < 7 :
      return 'potential Customer'
    elif (df['RFM_score']) >= 5 and (df['RFM_score']) < 6 :
      return 'Promising Customer'
    elif (df['RFM_score']) >= 4 and (df['RFM_score']) < 5 :
      return 'Needs Attention'
    else:
      return 'Require Activation'

df_rfm['RFM_levels'] = df_rfm.apply(rfm_level, axis = 1)

df_rfm.head()
```

	CustomerID	last_purchase	Recency	frequency	Monetary	R	F	M	RFM_score	RFM_levels
0	12346.0	2011-01-01	334	2	0.00	1	1	1	3	Require Activation
1	12347.0	2011-12-01	0	182	4310.00	4	4	4	12	cant loose them
2	12348.0	2011-09-01	91	31	1797.24	2	2	4	8	Champion
3	12349.0	2011-11-01	30	73	1757.55	4	3	4	11	cant loose them

Finally we can use aggregate to find the individual values for each level

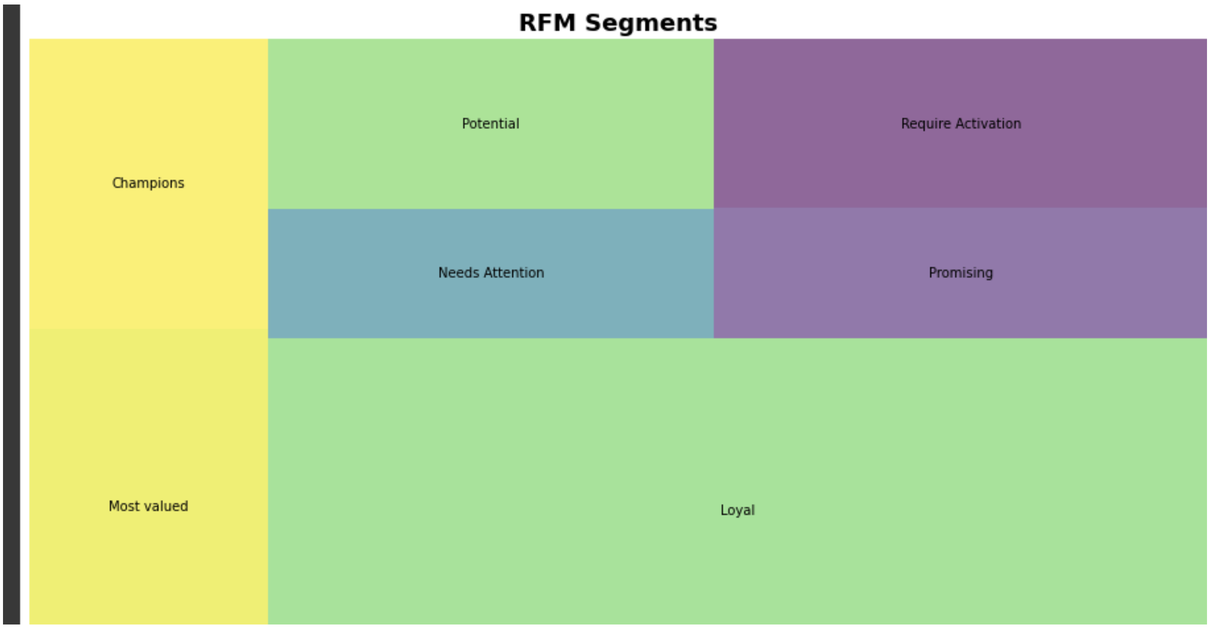
```
rfm_level_agg = df_rfm.groupby('RFM_levels').agg({

    'Recency': 'mean',
    'frequency' : 'mean',
    'Monetary' : ['mean', 'count']

}).round(1)
```

```
[89] print(rfm_level_agg)

                     Recency frequency Monetary
                        mean      mean     mean count
RFM_levels
Champion                68.6      47.5    857.6   486
Loyal Customer          92.8      36.2    643.1   398
Most valued Customer    31.6     179.8   3812.6  1869
Needs Attention        203.2      13.9    211.7   332
Promising Customer     180.5      21.0    350.6   438
Require Activation     277.3       7.8    125.0   367
potential Customer     100.1      23.6    434.9   482
```

So now we have following information about the level

> We have above 60% of the customer either falling into MVP, champion customer or loyal customer i.e. top three tiers. The other 40% we have to devise some strategy to increase their experience with the site

- Potential: These segment have the high potential to enter loyal customer segment. May be offers them some freebies to show them how much we value them
- Promising: Show the promising signs with quantity and values of their purchase but it has been a while since they bought from the sight, so let's target them with the limited time period discount offers
- Needs attention: made some initial purchase but not been seen since. May be due to bad customer experience. Let's spend some resources to build the trust and brand awareness
- Need urgent activation: Poorest performed in the RFM segment. They might have gone to other vendor and require different strategy to win them back



4. Creating Cluster using K-Means Clustering

4.1 Preparing the data for K-Means clustering

On this final part, we will continue to work on the RFM dataframe that we created earlier and apply K-means clustering to segment out customer data. We will continue to work on the feature RFM score that was created in earlier section

K-Means Clustering:

K-means clustering is an unsupervised machine learning algorithms. k-means algorithms identifies k-centroid and allocates the data points to its respective centroid while keeping mean distance as small as possible

Basic assumptions:

We will be using the RFM features engineered in the previous section for segmenting. But before we get into it we must ensure that these features fulfill basic assumption for the K-means and these are -

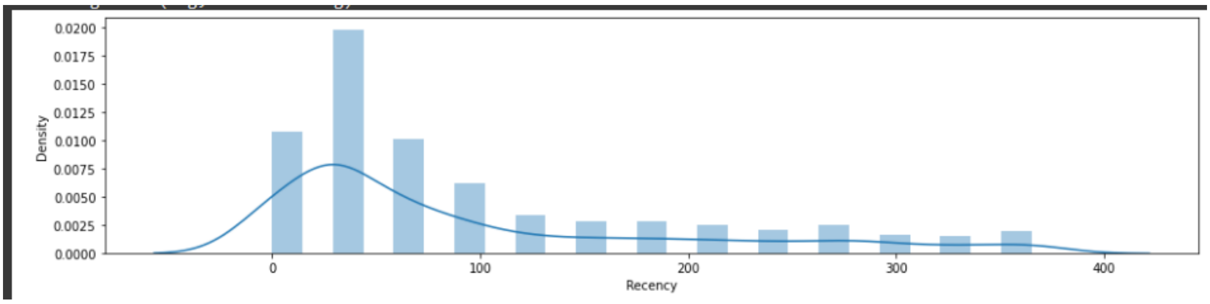
- Distribution of the variable should be normally distributed
- Variable with same average value
- Variable with same variance

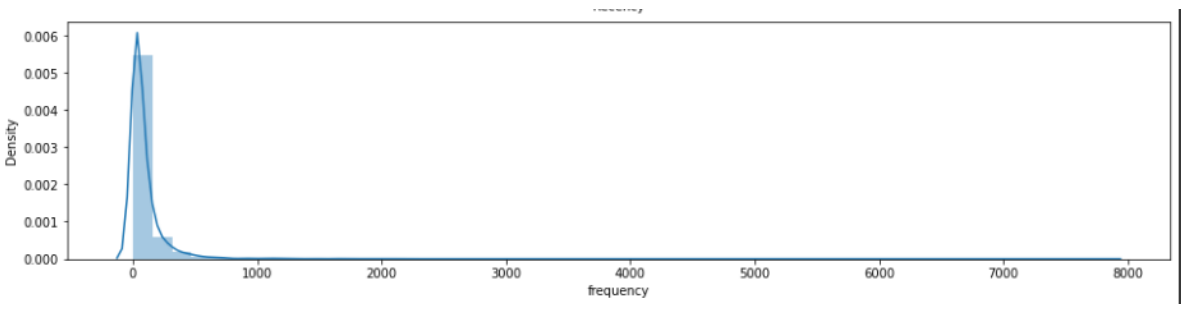
1st can be address by looking at the distribution of the RFM and applying log transformation if needed 2nd & 3rd can be address with scaling the variable to Standardscaler of sklearn library

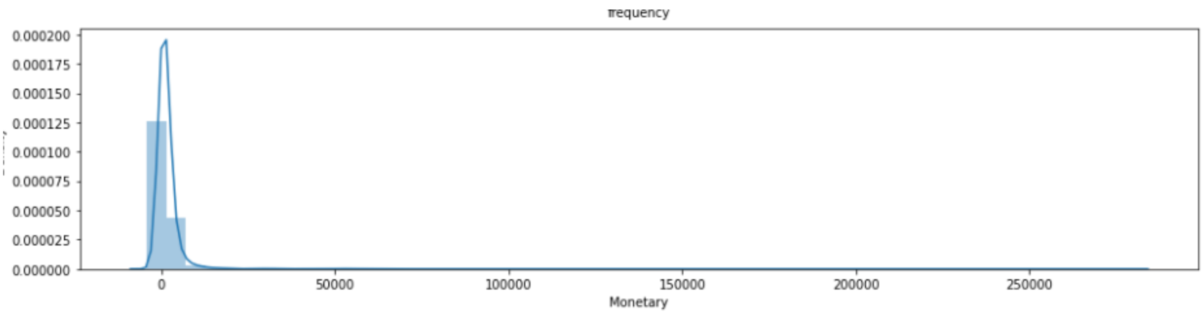
4.1.1 Data Skewness

Let's examine the distribution of the variable RFM

```
plt.figure(figsize=(16,12))

plt.subplot(3,1,1); sns.distplot(df_final['Recency'])
plt.subplot(3,1,2); sns.distplot(df_final['frequency'])
plt.subplot(3,1,3); sns.distplot(df_final['Monetary'])

plt.show()
```



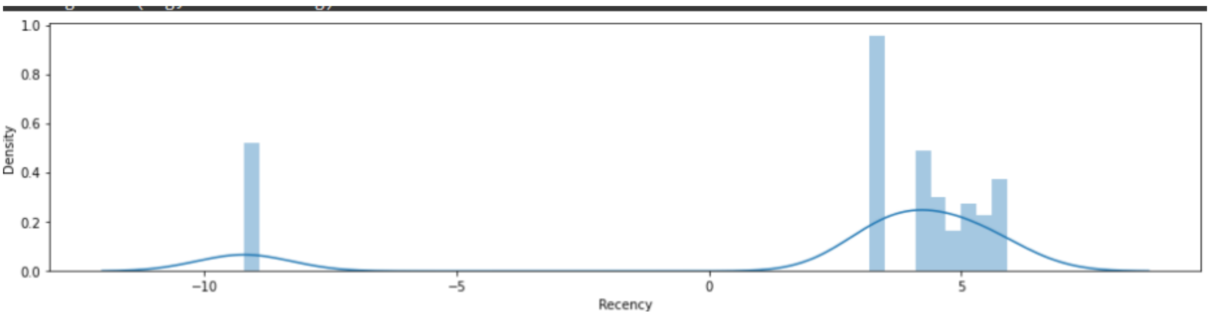


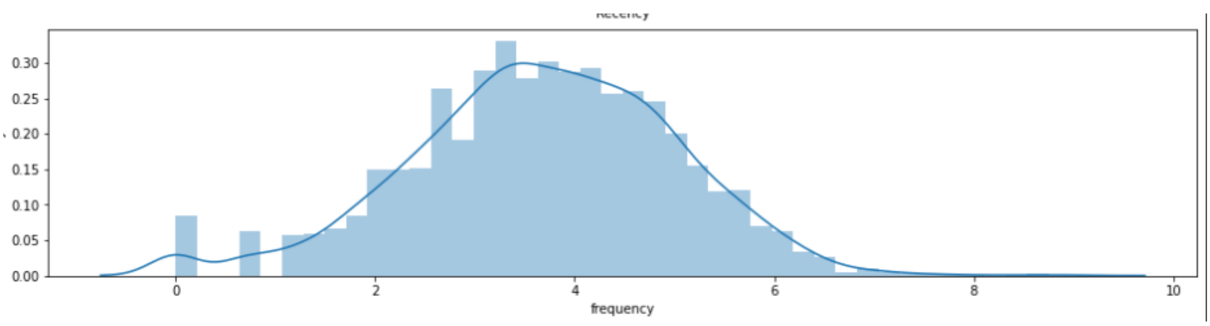


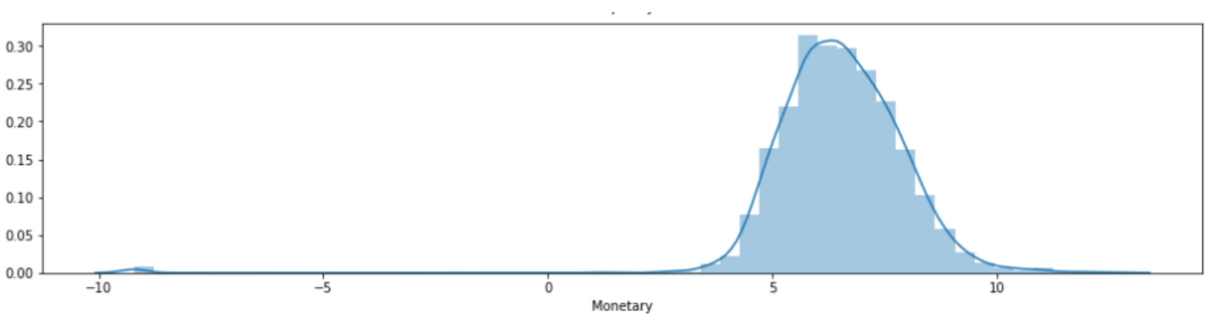
As we can see that there is a general skewness to the right. To address this let's apply logarithmic transformation

```
# we added 0.0001 to recency and monetary variable as these varibale have 0 values and log will not work

df_final['Recency'] = df_final['Recency'] + 0.0001
df_final['Monetary'] = df_final['Monetary'] + 0.0001

recency_log = np.log(df_final['Recency'])
frequency_log = np.log(df_final['frequency'])
Monetary_log = np.log(df_final['Monetary'])
```

Now let's again look at the distribution







Also add the Log RFM values into the dataset

```
[ ]  # df_rfm = df_rfm.assign(R = r_group.values, F = f_group.values, M = m_group.values)

     df_final = df_final.assign(Recency_log = recency_log.values,
                                Frequency_log = frequency_log.values,
                                Monetary_log = Monetary_log.values)

     df_final.head()
```

	Recency	frequency	Monetary	Recency_log	Frequency_log	Monetary_log
0	334.0001	2	0.0001	5.811141	0.693147	-9.210340
1	0.0001	182	4310.0001	-9.210340	5.204007	8.368693
2	91.0001	31	1797.2401	4.510861	3.433987	7.494007
3	30.0001	73	1757.5501	3.401201	4.290459	7.471676
4	303.0001	17	334.4001	5.713733	2.833213	5.812338

4.1.2 Scaling the data:

As we see we have different type of data as recency_log, Frequency_log are based on the number of days, whereas monetary_log is quite large as it is based on monetary sum

So let’s using Sklean Standardscaler library to scale the data

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_final[['Recency_sc', 'Frequency_sc', 'Monetary_sc']] = scaler.fit_transform(df_final[['Recency_log', 'Frequency_log','M
```

```
[ ] df_final.head()
```

	Recency	frequency	Monetary	Recency_log	Frequency_log	Monetary_log	Recency_sc	Frequency_sc	Monetary_sc
0	334.0001	2	0.0001	5.811141	0.693147	-9.210340	0.712347	-2.232793	-10.248292
1	0.0001	182	4310.0001	-9.210340	5.204007	8.368693	-2.288471	1.143915	1.208986
2	91.0001	31	1797.2401	4.510861	3.433987	7.494007	0.452592	-0.181074	0.638903
3	30.0001	73	1757.5501	3.401201	4.290459	7.471676	0.230917	0.460058	0.624348
4	303.0001	17	334.4001	5.713733	2.833213	5.812338	0.692888	-0.630797	-0.457139

4.2 Determine the Number of Clusters for K-means clustering

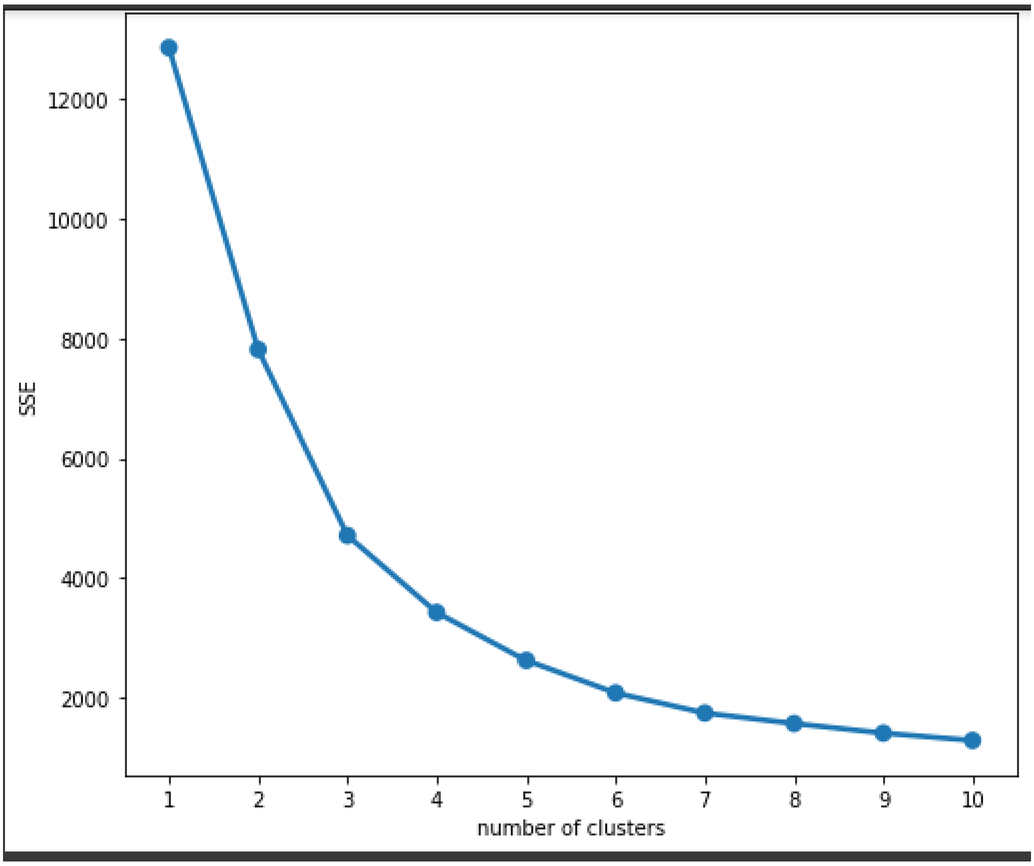
Before we apply the clustering mechanism we need to find the number of cluster which we can do so by applying the elbow method

Elbow Method:

In elbow method, we apply the clustering by varying the number of clusters and measure the SSE from centroid to their data point. We choose the optimal cluster where the reduction in SSE is less

```
[77] df_final_sc = df_final.dropna()
```

```
[86] from sklearn.cluster import KMeans
     sse = {}


     for k in range(1,11):
       kmeans = KMeans(n_clusters=k,random_state = 42)
       kmeans.fit(df_final_sc[['Recency_sc', 'Frequency_sc','Monetary_sc']])
       sse[k] = kmeans.inertia_

     # plt the kmeans for each k

     plt.figure(figsize=(8,7))
     plt.title("Elbow Method SSE ")
     sns.pointplot(x = list(sse.keys()), y=list(sse.values()))
     plt.xlabel("number of clusters")
     plt.ylabel("SSE")
     plt.show()
```



Ideally we would want to choose cluster where SSE stop decreasing drastically. So for our model we will choose K=4.

4.3 K-Means Clustering with K=4

Now let's try the K-Means clustering with K i.e. Number of clusters as 4

```
kmeans = KMeans(n_clusters=4, random_state = 42)
kmeans.fit(df_final_sc[['Recency_sc', 'Frequency_sc','Monetary_sc']])

cluster_labels = kmeans.labels_

df_final_sc = df_final_sc.assign(Clusters = cluster_labels+1)

df_final_sc.head()
```

	CustomerID	Recency	frequency	Monetary	Recency_sc	Frequency_sc	Monetary_sc	Clusters
0	12346.0	334.0001	2	0.0001	0.712349	-2.232793	-10.248249	4
1	12347.0	0.0001	182	4310.0001	-2.288471	1.143915	1.208988	3
2	12348.0	91.0001	31	1797.2401	0.452592	-0.181074	0.638904	1
3	12349.0	30.0001	73	1757.5501	0.230915	0.460058	0.624349	1
4	12350.0	303.0001	17	334.4001	0.692890	-0.630797	-0.457140	2

Note: We add 1 to cluster_labels so that cluster starts with 1

Finally we'll try to analyse the various cluster based on the aggregate values of the clusters

```
[106] df_final_sc.groupby(['Clusters']).agg({
          'Recency': 'mean',
          'frequency': 'mean',
          'Monetary': ['mean', 'count']
      }).round(0)
```

	Recency	frequency	Monetary	
	mean	mean	mean	count
Clusters				
1	74.0	120.0	2169.0	1779
2	151.0	18.0	340.0	1867
3	0.0	229.0	5660.0	671
4	94.0	28.0	0.0	14

As we group the data to find the aggregate information of the R, F and M component of the cluster. We notice that each cluster places different emphasis on different component

Cluster 1: Cluster 1 has good monetary value but has not shopped with us for 2 months. We need to offer them some time-limited offers

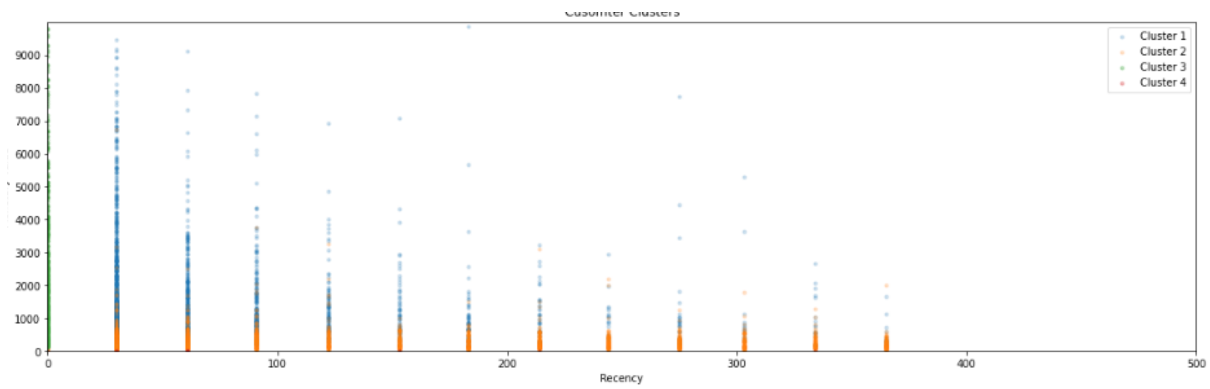
Cluster 2: Cluster 2 performs poorly on the entire component i.e. R, F, and T. This cluster we'll need to focus our attention and design some strategy to activate them

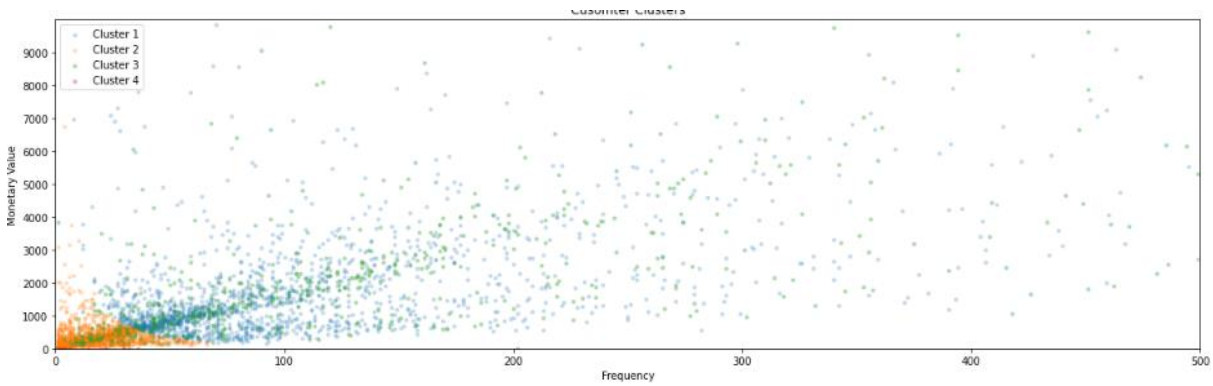
Cluster 3: Cluster 3 which is 15% performs really well in the entire component R, F, and T. So, we need to do everything to retain this cluster

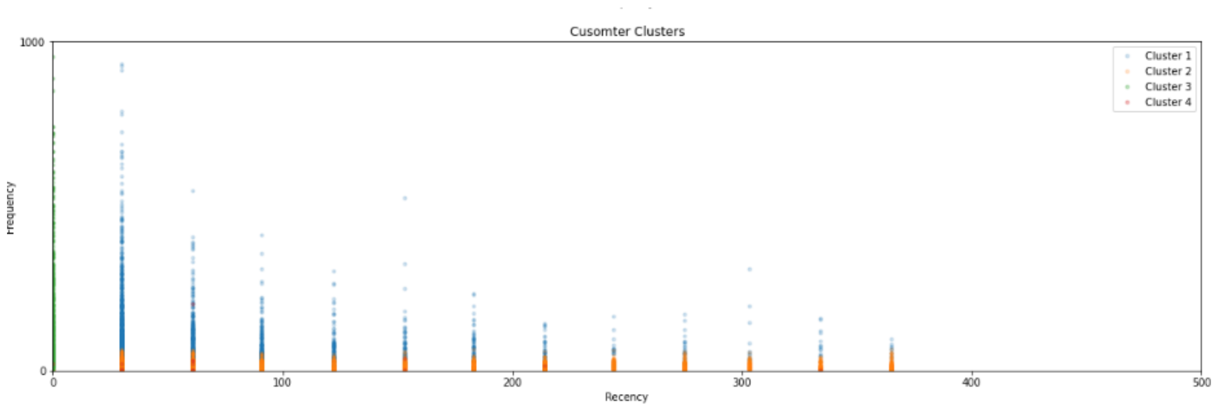
Cluster 4: Cluster 4 is the cluster where we need to put more our efforts and resources. It appears that customers who fall in this cluster have visited the website and may have made some purchase but due to bad customer service, faulty products are not spending or may have returned the product.

4.4 Visualizing the Clusters

Lastly, let's visualize the K-means cluster with the help of scatter plot

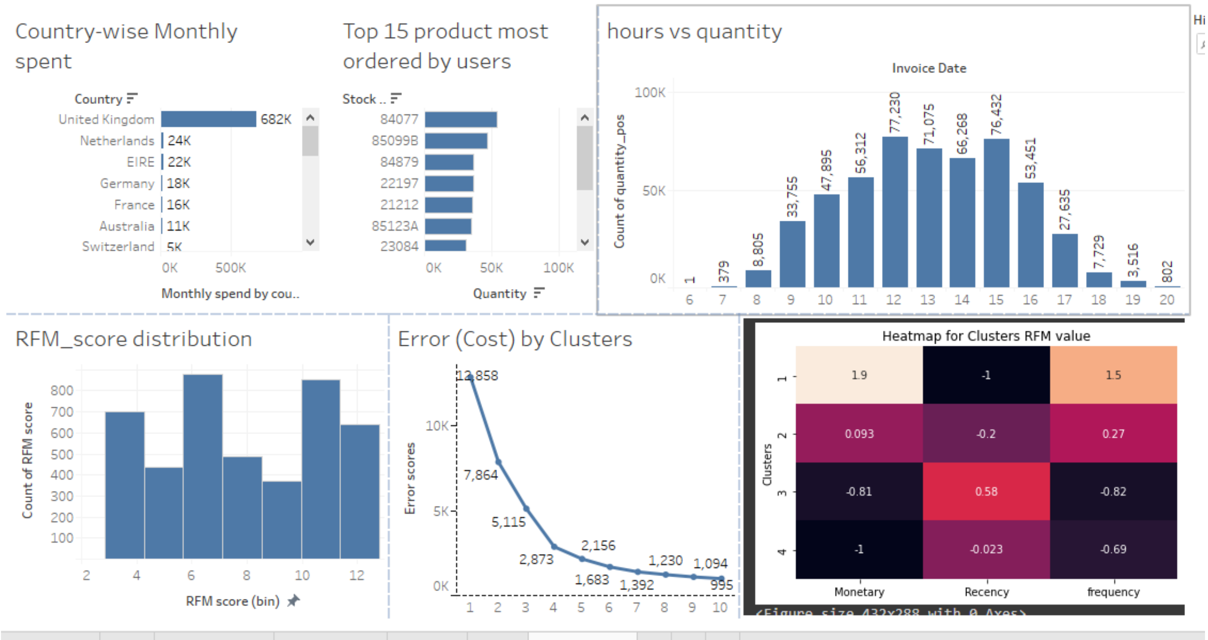






5. Data Reporting

The Dashboard that represent the major visualization are given below



Full features of the dashboards can be found at

https://public.tableau.com/app/profile/tushar.bhave/viz/onlineretail_16550116610710/Dashboard1