Business Case Document

Customer Segmentation using RFM Analysis

and K-Means Clustering

1. Purpose and Use Case

The primary purpose of this project is to perform customer segmentation utilizing the RFM (Recency, Frequency, Monetary) approach. The project aims to analyze customer transaction data and categorize customers into distinct segments based on their purchase behaviors and interactions with the business. The use case for this segmentation is to enable targeted marketing strategies, personalized communication, and tailored service offerings, ultimately enhancing customer engagement, satisfaction, and overall business performance.

2. Target Audiences

The project's outcomes and insights cater to various stakeholders within the organization, including:

Marketing Teams: Utilize customer segments to design targeted marketing campaigns and promotions. Tailoring messages and offers based on RFM segments can lead to improved campaign effectiveness and higher conversion rates.

Sales Teams: Leverage customer segments to identify opportunities for cross-selling and upselling. By understanding each segment's preferences and purchasing patterns, sales teams can optimize their strategies.

Customer Service Teams: Provide better customer support by recognizing the needs and expectations of different customer segments. This allows for more personalized assistance and problem-solving.

Management and Strategy Teams: Gain insights into the customer base's composition and preferences. Use segmented data to inform strategic decisions, allocate resources effectively, and identify areas for growth.

Product Development Teams: Understand customer preferences and demands to develop products and services that align with the needs of each segment.

Technical Design Document

Customer Segmentation using RFM Analysis

and K-Means Clustering

1. Introduction

The primary purpose of this project is to perform customer segmentation utilizing the RFM (Recency, Frequency, Monetary) approach. The project aims to analyze customer transaction data and categorize customers into distinct segments based on their purchase behaviors and interactions with the business.

2. Toolset and Coding Language

- *Python*: For data preprocessing, analysis, and segmentation using pandas and NumPy libraries.
- *Excel*: For storing cleaned, preprocessed, and segmented data.
- *RapidMiner*: For potential K-Means clustering and visualization.

3. Data Models

- *Original Dataset*: Contains customer transaction data with columns like Customer ID, Item Code, Invoice Number, Date of Purchase, Quantity, Price, etc. (Provided by Imarticus Institute & KPMG)
- *RFM Table_Scaled*: Scaled values Recency, Frequency, and Monetary data per customer.
- *Segmented RFM Table*: RFM segments assigned to each customer.

4. Data Volume

- *Original Dataset*: 537979 Rows X 12 Columns
- *RFM Table_scaled*: 395865 Rows X 04 Columns
- *Segmented RFM Table*: 395865 Rows X 07 Columns

5. Technical Workflow

- *Data Cleaning:* Using pandas, remove duplicates, handle missing values, and eliminate negative quantities.
- *RFM Analysis:* Calculate Recency, Frequency, and Monetary metrics per customer using pandas.
- *Data Preprocessing*: Log-transform and scale the RFM data using NumPy and sklearn.
- *RFM Segmentation*: Assign RFM segments and scores using pandas and NumPy based on quartiles.
- *Data Export*: Save the segmented data to Excel files.
- *K-Means Clustering in RapidMiner*:
  - Import the cleaned and scaled data into RapidMiner.
  - Utilize the K-Means operator for clustering based on RFM metrics.
  - Configure input attributes, number of clusters, and distance measure.
  - Analyze and visualize the clustering results within RapidMiner.
  - *Interpretation and Labeling:* Analyze the clusters obtained from K-Means, interpret their characteristics, and assign meaningful labels to each segment.

Data Cleaning

```
[ ] 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=True).
```

```
[ ] #Change Headers to Natural language - Each word in Header staring with Upper Case, Words seperate by underscore
    # Change CustomerId, InvoiceNo type to object
    # Replace Blank Spaces in Customer_Id with nan
    # Drop Duplicates
    # Treat Date of Purchase
    # Remove -ve
    # TRANSP
```

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from scipy import stats
    %matplotlib inline
```

```
[ ] import os
    EcomData = pd.read_excel("/content/E-com_Data.xlsx")
```

```
[ ] print(EcomData.columns)
```

```
Index(['CustomerID', 'Item Code', 'InvoieNo', 'Date of purchase', 'Quantity',
       'Time', 'price per Unit', 'Price', 'Shipping Location',
       'Cancelled_status', 'Reason of return', 'Sold as set'],
      dtype='object')
```

```
[ ] EcomData.rename(columns = {'CustomerID':'Customer Id', 'Item Code':'Item Code',
```

```
[ ] EcomData.rename(columns = {'CustomerID':'Customer_Id', 'Item Code':'Item_Code',
                               'InvoieNo':'Invoice_No', 'Date of purchase':'Date_Of_Purchase',
                               'price per Unit':'Price_Per_Unit','Shipping Location':'Shipping_Location',
                               'Cancelled_status':'Cancelled_Status','Reason of return':'Reason_Of_Return',
                               'Sold as set':'Sold_As_Set'}, inplace = True)
    print(EcomData.columns)

    Index(['Customer_Id', 'Item_Code', 'Invoice_No', 'Date_Of_Purchase',
           'Quantity', 'Time', 'Price_Per_Unit', 'Price', 'Shipping_Location',
           'Cancelled_Status', 'Reason_Of_Return', 'Sold_As_Set'],
          dtype='object')
```

```
[ ] EcomData = EcomData.astype({'Customer_Id':'str', 'Invoice_No':'str'})
    EcomData.replace(r'\s+',np.nan,regex=True).replace('',np.nan)
    x = EcomData[EcomData.Customer_Id=='nan']
    x.shape

    (133790, 12)
```

```
[ ] #Remove Duplicate Rows
    EcomData = EcomData.drop_duplicates(subset=['Customer_Id', 'Item_Code', 'Invoice_No', 'Date_Of_Purchase',
           'Quantity', 'Time', 'Price_Per_Unit', 'Price', 'Shipping_Location',
           'Cancelled_Status', 'Reason_Of_Return', 'Sold_As_Set'])
    EcomData.shape
    # 9 Duplicate rows removed

    (537970, 12)
```

```
    # Dropping TRANSP
    EcomData = EcomData[EcomData.Item_Code!='TRANSP']
    EcomData.shape
    # 144 rows removed

    (537826, 12)
```

```
[ ] # Dropping Blank Customer ID
    EcomData = EcomData[EcomData.Customer_Id!='nan']
    EcomData.shape
```

```
[ ] # Remove Negative Quantity
    EcomData_NR = EcomData[EcomData.Quantity>0]
    EcomData_NR.shape
    # 8182 Rows Removed
    # Total Rows Removed = 142114 (26.4%)
```

```
(395865, 12)
```

```
[ ] # Preparing Data for RFM
    RFM1= EcomData_NR.iloc[:,0:9]
    RFM1=RFM1.drop(['Item_Code','Quantity','Time','Price_Per_Unit','Shipping_Location'], axis = 1)
```

```
print(EcomData.shape)
print(RFM1.shape)
RFM1
```

```
(404047, 12)
(395865, 4)
```

	Customer_Id	Invoice_No	Date_Of_Purchase	Price
0	4355.0	398177	2017-10-29	1926.0
1	4352.0	394422	2017-10-05	1740.0
2	4352.0	394422	2017-10-12	1866.0
3	4352.0	388633	2017-08-22	1869.0
4	4352.0	394422	2017-10-10	1888.0
...	...	...	...	...
537945	37.0	402292	2017-11-28	384.0
537946	37.0	402292	2017-11-27	398.0
537947	21.0	363890	2016-12-21	2464.0
537948	21.0	363890	2016-12-21	4068.0
537949	21.0	363890	2016-12-17	4940.0

```
[ ] RFM_Data = RFM1
```

```
[ ] import datetime as dt
    #Reference Date
    Now = max(RFM_Data['Date_Of_Purchase'])
```

```
[ ] df_recency = RFM_Data.groupby(['Customer_Id'],as_index=False)['Date_Of_Purchase'].max()
    df_recency.columns = ['Customer_Id','Last_Purchase_Date']
    df_recency['Recency'] = (Now-df_recency['Last_Purchase_Date']).dt.days
    df_recency.drop(columns=['Last_Purchase_Date'],inplace=True)
    FM_Table = RFM_Data.groupby('Customer_Id').agg({'Invoice_No'   : lambda x:len(x),
                                                    'Price'   : lambda x:x.sum()})
    FM_Table.rename(columns = {'Invoice_No' :'Frequency',
                               'Price':'Monetary_Value'},inplace= True)
    RFM_Table = df_recency.merge(FM_Table,left_on='Customer_Id',right_on='Customer_Id')
    RFM_Table.head()
```

	Customer_Id	Recency	Frequency	Monetary_Value
0	10.0	24	58	331601.0
1	100.0	187	36	85862.0
2	1000.0	3	37	263771.0
3	1001.0	182	8	10575.0
4	1002.0	63	6	111008.0

```
quantiles = RFM_Table.quantile(q=[0.25,0.50,0.75])
quantiles = quantiles.to_dict()
RFM_Table_seg = RFM_Table.copy()
def RScore(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
```

```python
    else:
        return 4

def FMScore(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
RFM_Table_seg['R_quartile'] = RFM_Table_seg['Recency'].apply(RScore, args=('Recency',quantiles))
RFM_Table_seg['F_quartile'] = RFM_Table_seg['Frequency'].apply(FMScore, args=('Frequency',quantiles))
RFM_Table_seg['M_quartile'] = RFM_Table_seg['Monetary_Value'].apply(FMScore, args=('Monetary_Value',quantiles))
RFM_Table_seg['RFM_Segment'] = RFM_Table_seg.R_quartile.map(str)+RFM_Table_seg.F_quartile.map(str)+RFM_Table_seg.M_quartile.map(str)
RFM_Table_seg['RFM_Score'] = RFM_Table_seg[['R_quartile','F_quartile','M_quartile']].sum(axis=1)
RFM_Table_seg.head()
```

```
<ipython-input-21-f7bf73fb3acf>:1: FutureWarning: The default value of numeric_only in DataFrame.quantile is deprecated. In a future version, it will d
  quantiles = RFM_Table.quantile(q=[0.25,0.50,0.75])
```

	Customer_Id	Recency	Frequency	Monetary_Value	R_quartile	F_quartile	M_quartile	RFM_Segment	RFM_Score
0	10.0	24	58	331601.0	2	2	1	221	5
1	100.0	187	36	85862.0	4	3	3	433	10
2	1000.0	3	37	263771.0	1	3	1	131	5
3	1001.0	182	8	10575.0	4	4	4	444	12
4	1002.0	63	6	111008.0	3	4	2	342	9

```python
print("Best Customers: ",len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='111']))
print('Regular Customers: ',len(RFM_Table_seg[RFM_Table_seg['F_quartile']==1]))
print("Big Spenders: ",len(RFM_Table_seg[RFM_Table_seg['M_quartile']==1]))
print('Almost Lost Customers: ', len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='134']))
print('Lost Customers: ',len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='344']))
print('Lost Worst Customers: ',len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='444']))
```

```
print("Best Customers: ",len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='111']))
print('Regular Customers: ',len(RFM_Table_seg[RFM_Table_seg['F_quartile']==1]))
print("Big Spenders: ",len(RFM_Table_seg[RFM_Table_seg['M_quartile']==1]))
print('Almost Lost Customers: ', len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='134']))
print('Lost Customers: ',len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='344']))
print('Lost Worst Customers: ',len(RFM_Table_seg[RFM_Table_seg['RFM_Segment']=='444']))
```

```
Best Customers:  456
Regular Customers:  1072
Big Spenders:  1081
Almost Lost Customers:  24
Lost Customers:  181
Lost Worst Customers:  407
```

Pre-processing for K- Means Clustering

```
# K-means gives the best result under the following conditions:

# Data's distribution is not skewed.
# Data is standardised.
```

```
RFM_Table.head()
```

	Customer_Id	Recency	Frequency	Monetary_Value
0	10.0	24	58	331601.0
1	100.0	187	36	85862.0
2	1000.0	3	37	263771.0
3	1001.0	182	8	10575.0
4	1002.0	63	6	111008.0

```
[ ] RFM_Table.head()
    RFM_Table.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4324 entries, 0 to 4323
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Customer_Id     4324 non-null   float64
 1   Recency         4324 non-null   int64
 2   Frequency       4324 non-null   int64
 3   Monetary_Value  4324 non-null   float64
dtypes: float64(2), int64(2)
memory usage: 168.9 KB
```

Check Skewness

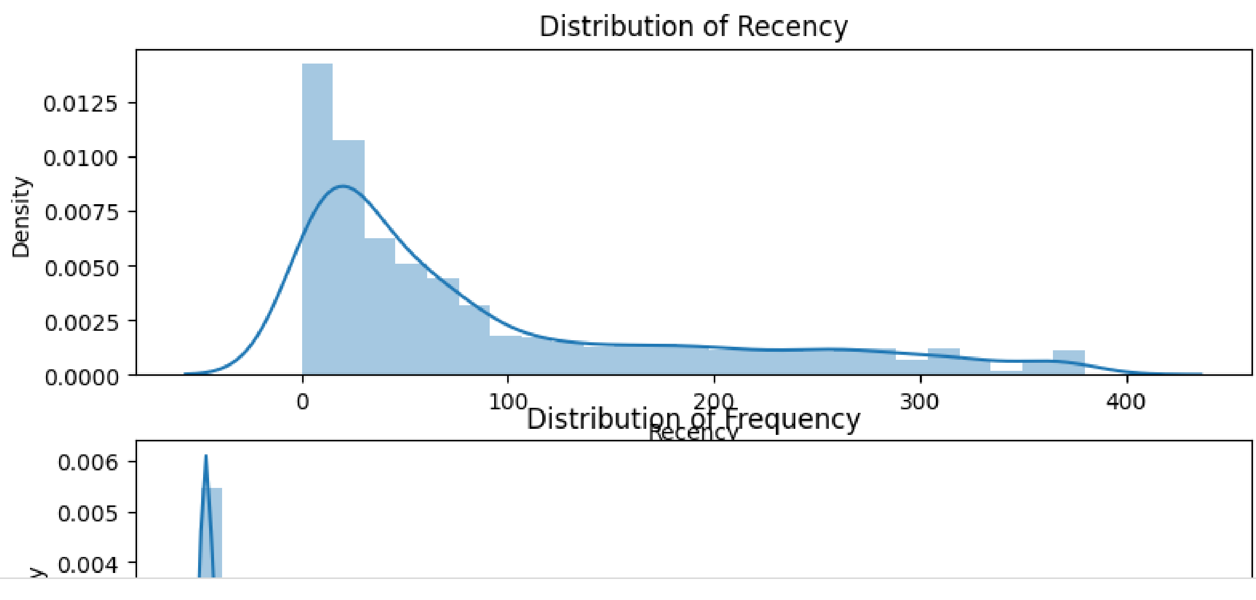
```
[ ] def check_skew(df_skew, column):
        skew = stats.skew(df_skew[column])
        plt.title('Distribution of ' + column)
        sns.distplot(df_skew[column])
        print("{}'s: Skew: {}".format(column, skew))
        return
```

```
# Plot all 3 graphs together for summary findings
plt.figure(figsize=(9, 9))

plt.subplot(3, 1, 1)
check_skew(RFM_Table,'Recency')

plt.subplot(3, 1, 2)
check_skew(RFM_Table,'Frequency')

plt.subplot(3, 1, 3)
check_skew(RFM_Table,'Monetary_Value')
```

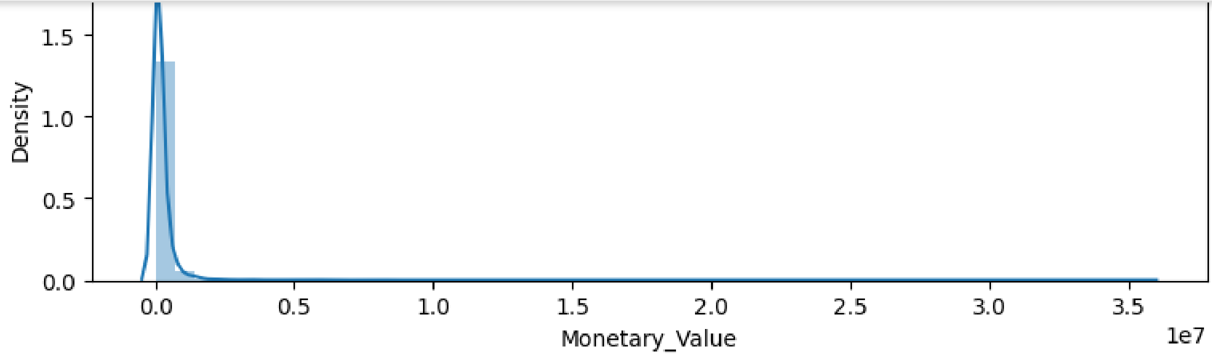
```
<ipython-input-27-2b1f210b83e7>:4: UserWarning:
```

```
plt.subplot(3, 1, 1)
check_skew(RFM_Table,'Recency')

plt.subplot(3, 1, 2)
check_skew(RFM_Table,'Frequency')

plt.subplot(3, 1, 3)
check_skew(RFM_Table,'Monetary_Value')
```

```
<ipython-input-27-2b1f210b83e7>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_skew[column])
Monetary_Value's: Skew: 22.380087010683194
```



[ ]



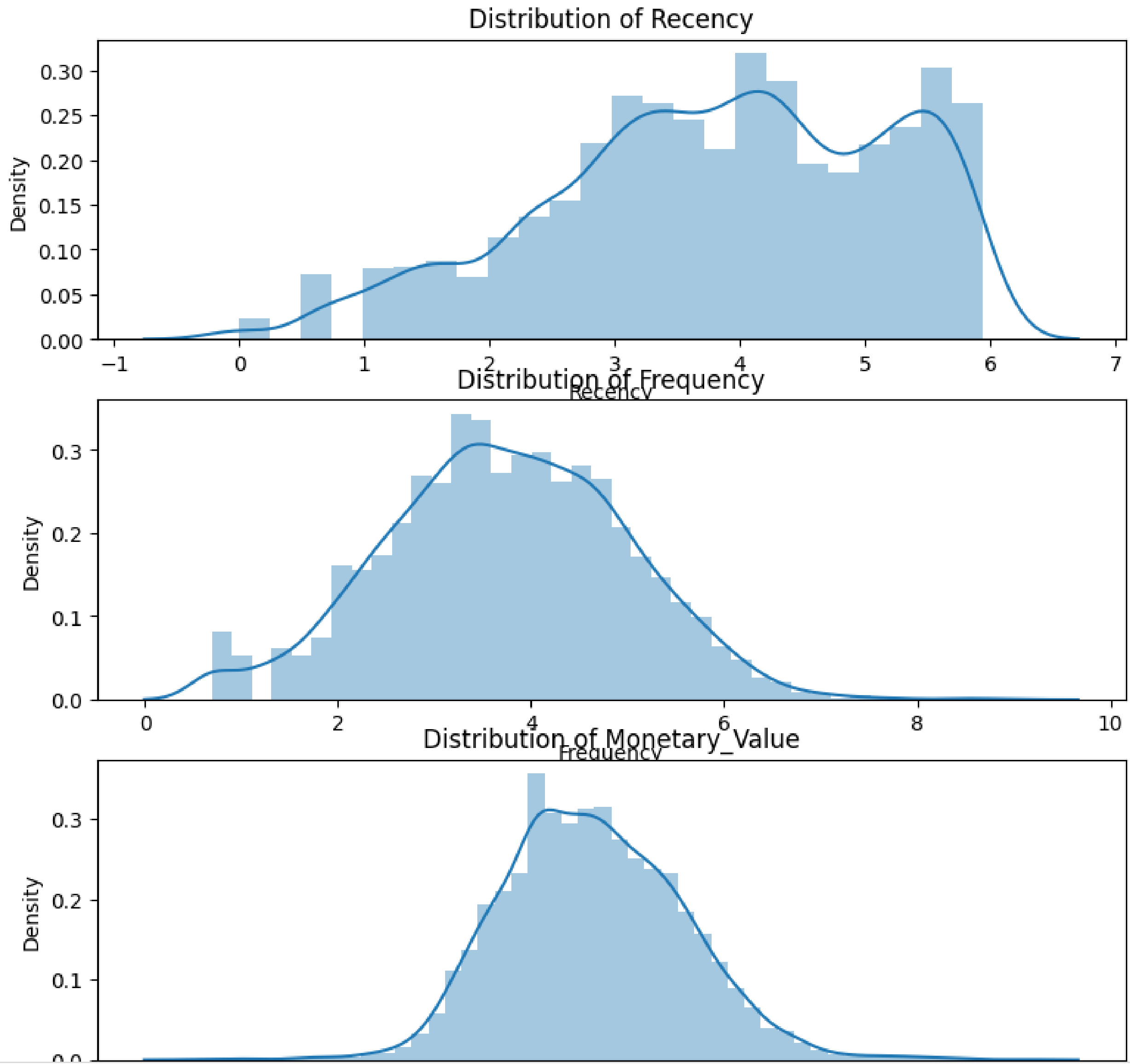

```
[ ] # The data is highly skewed,therefore we will perform log transformations to reduce the skewness of each
    # We add a small constant as log transformation demands all the values to be positive.
```

```
[ ] df_rfm_log = RFM_Table.copy()
```

```
df_rfm_log = np.log(df_rfm_log+1)

plt.figure(figsize=(9, 9))

plt.subplot(3, 1, 1)
check_skew(df_rfm_log,'Recency')

plt.subplot(3, 1, 2)
check_skew(df_rfm_log,'Frequency')

plt.subplot(3, 1, 3)
check_skew(df_rfm_log,'Monetary_Value')
```

```
<ipython-input-27-2b1f210b83e7>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

```
sns.distplot(df_skew[column])
Monetary_Value's: Skew: 0.25603333597876865
```



```
[ ]                          tail='both', # cap left, right or both tails
                             fold=2,
                              variables=['Recency','Frequency','Monetary_Value'])
     winsorizer.fit(df_rfm_log)
```

```
Winsorizer
Winsorizer(fold=2, tail='both',
           variables=['Recency', 'Frequency', 'Monetary_Value'])
```

```
[ ]  df_rfm_log = winsorizer.transform(df_rfm_log)
```

```
[ ]  from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()

     scaler.fit(df_rfm_log)

     RFM_Table_scaled = scaler.transform(df_rfm_log)
```

```
[ ]  RFM_Table_scaled = pd.DataFrame(RFM_Table_scaled, columns=df_rfm_log.columns)
```

```
[ ]  RFM_Table_scaled.head()
```

	Customer_Id	Recency	Frequency	Monetary_Value
0	-5.067664	-0.476419	0.282175	1.102316
1	-2.815381	1.056713	-0.106885	-0.049848
2	-0.485480	-1.868984	-0.084650	0.907174
3	-0.484466	1.036230	-1.285602	-1.835560
4	-0.483452	0.237885	-1.495144	0.169177

```
[ ]  RFM_Table_scaled.to_excel("RFM_Table_scaled.xlsx")
```

Finding the optimal number of clusters

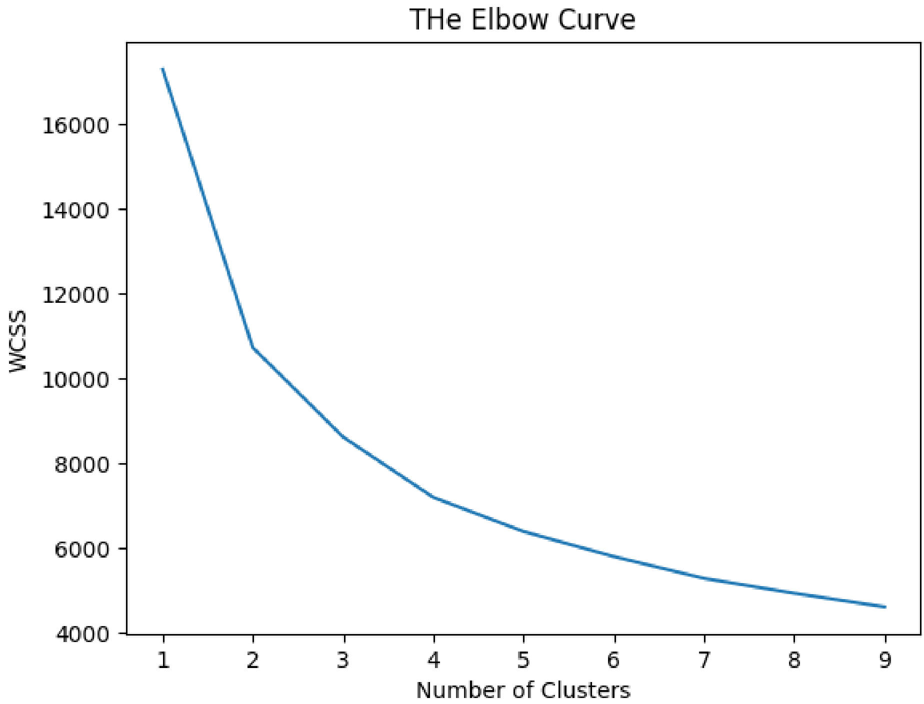
```
[ ] from sklearn.cluster import KMeans
    kmeans = KMeans(n_clusters=4, random_state=0)
    model = kmeans.fit(RFM_Table_scaled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
```

```
# Creating a function with KMeans to plot "The Elbow Curve"

wcss = []
for i in range(1,10):
    kmeans = KMeans(n_clusters=i,init='k-means++' ,max_iter=50,random_state=0)
    kmeans.fit(RFM_Table_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,10),wcss)
plt.title('THe Elbow Curve')
plt.xlabel('Number of Clusters')
plt.ylabel("WCSS") #WCSS stands for total within-cluster sum of sqaure
plt.show()
```

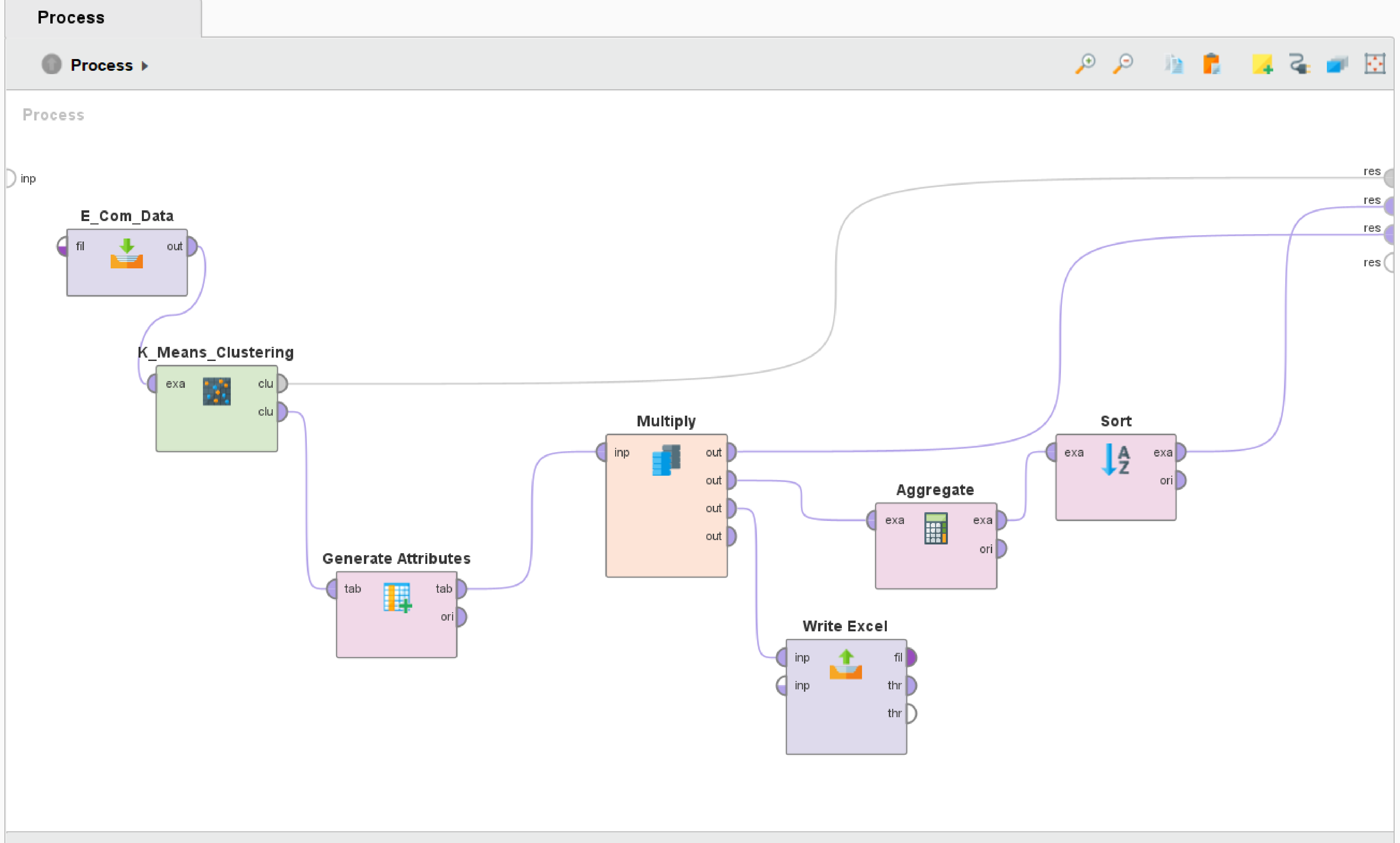
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
```

```
  warnings.warn(
[ ] /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
  warnings.warn(
```



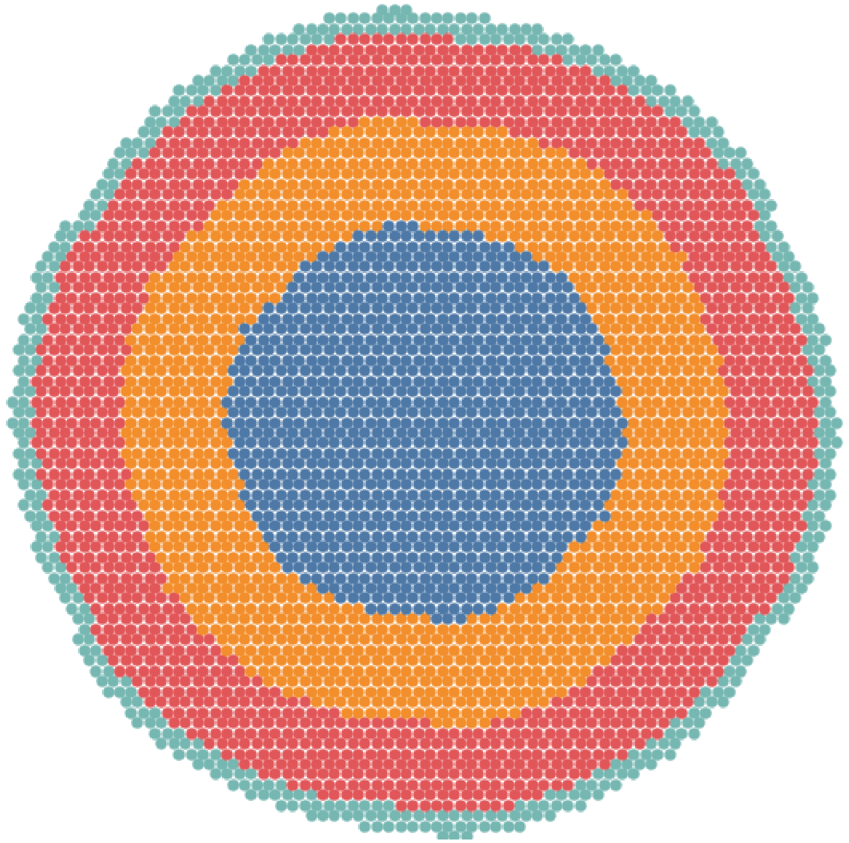

```
[ ] kmeans = KMeans(n_clusters=4)
    kmeans.fit(RFM_Table_scaled)
```

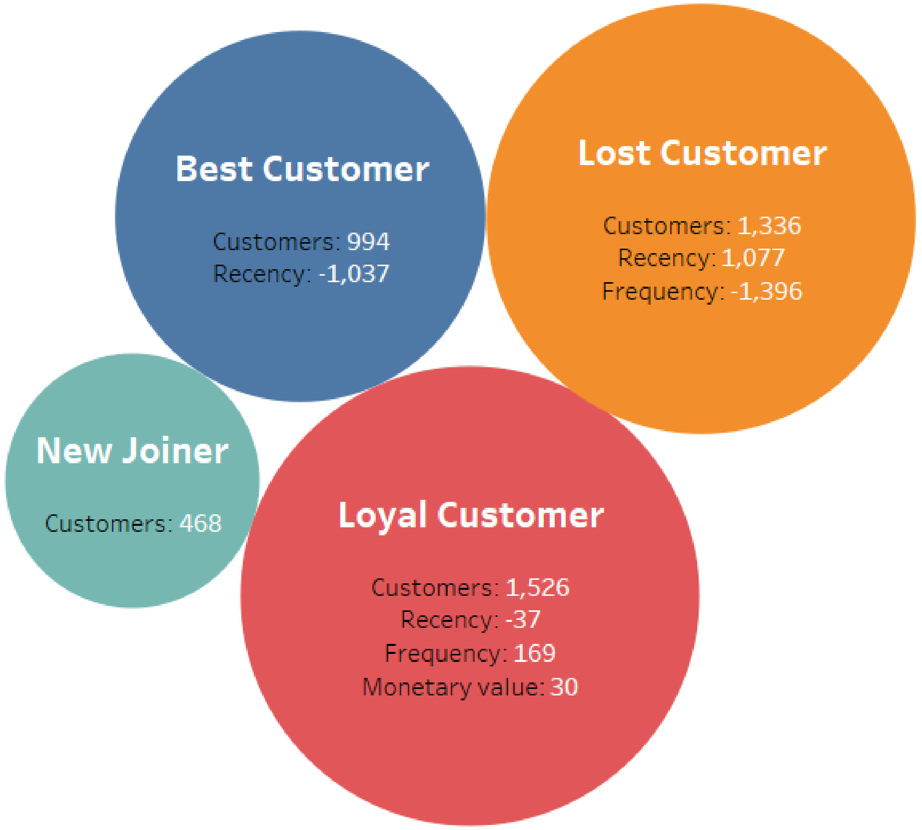
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
  warnings.warn(
▾ KMeans
KMeans(n_clusters=4)
```



Process
Process
Process
inp
E_Com_Data
fil
out
K_Means_Clustering
exa
clu
clu
Generate Attributes
tab
tab
ori
Multiply
inp
out
out
out
out
Aggregate
exa
exa
ori
Sort
exa
exa
ori
Write Excel
inp
inp
fil
thr
thr
res
res
res
res

Customer Segmentation On The Basis of Recency, Frequency and Monetary Value





Customer Segments
(All)

Customer Segments
Best Customer
Lost Customer
Loyal Customer
New Joiner

Customer Segments
(All)

Measure Names
Frequency
Monetary Value
Recency