CENTER FOR DEVELOPMENT OF ADVANCED COMPUTING

Customer Segmentation Using RFM Analysis

PG-DBDA September 2022

Submitted by:

Project Team 5

Tushar Shirsath

Somesh Rewadkar

Ayush Singh

Srujack Gedam

Shubham Mane

1. INTRODUCTION

In today's highly competitive business landscape, understanding and catering to customers' needs and preferences are crucial for a company's success. Customer segmentation is a powerful marketing technique that helps companies gain insights into their customers and create tailored marketing strategies to meet their specific needs. By dividing customers into smaller groups based on shared characteristics such as demographics, psychographics, or behavior, businesses can identify unique patterns and behaviors within their customer base.

Customer segmentation has become increasingly important in recent years, as advances in technology and data analytics have made it easier for companies to collect and analyze customer data. With the rise of e-commerce and social media, businesses can track customers' browsing and purchasing behavior, as well as their likes, interests, and social connections, to gain a better understanding of their preferences and needs.

To implement customer segmentation effectively, businesses need to collect and analyze large volumes of data from various sources. One way to do this is by using big data technologies such as Apache Spark and Amazon EMR (Elastic MapReduce). Amazon EMR is a fully-managed cloud service that enables businesses to process large amounts of data using popular big data frameworks such as Spark, Hive, and Hadoop.

In addition to data processing and segmentation, data visualization is another crucial aspect of customer analytics. By visualizing data, businesses can gain a better understanding of their customers' behavior and preferences and identify trends and patterns that may not be immediately apparent from raw data. Power BI is a powerful data visualization tool that allows businesses to create interactive reports and dashboards that help them make data-driven decisions.

In this project, behavioral segmentation was used to group customers based on their purchasing behavior. This approach is particularly effective because it provides insights into how customers interact with a company's products or services.

RFM (Recency, Frequency, Monetary) analysis is a popular technique for customer segmentation that helps businesses identify their most valuable customers based on their purchasing behavior. It involves analyzing three key metrics: how recently a customer has made a purchase (recency), how frequently they make purchases (frequency), and how much money they spend (monetary). By analyzing these metrics, businesses can identify their high-value customers and tailor their marketing strategies to meet their specific needs.

To perform RFM analysis, the following steps were taken:

Data Collection: Data on customer purchases and interactions with the company's products or services were collected.

Data Preparation: The data was cleaned and normalized to ensure accuracy and completeness.

Analysis: RFM scores were calculated for each customer based on their recency, frequency, and monetary value.

Segmentation: Customers were segmented into distinct groups based on their RFM scores.

2. PROBLEM STATEMENT

A company wants to increase customer loyalty and retention by developing more targeted and effective marketing strategies. To achieve this, the company needs to segment its customer base and identify the most valuable customers using RFM analysis.

The company is facing the challenge of not being able to effectively reach and engage its customers. It has a large customer base, but lacks the necessary understanding of their behaviors, preferences, and needs. As a result, the company's marketing campaigns are not as effective as they could be, resulting in lower sales and revenue.

To overcome this challenge, the company needs to conduct customer segmentation and RFM analysis to gain deeper insights into its customers' behavior and preferences. By segmenting customers based on their purchasing behavior, the company can create more targeted marketing campaigns and promotions that resonate with each customer segment. RFM analysis will help the company identify its most valuable customers and develop strategies to retain them.

The goal of this project is to conduct customer segmentation and RFM analysis to help the company develop more effective marketing strategies and increase customer loyalty and retention. By understanding its customers better, the company can create more personalized and relevant experiences that will drive customer satisfaction and increase revenue.

LITERATURE SURVEY

3.1 Introduction

Customer segmentation is the process of dividing a customer base into groups of individuals who share similar characteristics, needs, and behaviors. This technique is widely used by businesses to better understand their customers and create targeted marketing campaigns, product offerings, and customer experiences. Customer segmentation projects typically involve collecting and analyzing customer data, such as demographic information, purchase history, and behavior patterns, in order to identify distinct customer groups. The insights gained from customer segmentation can help businesses improve customer retention, increase sales, and enhance overall customer satisfaction.

1. Customer Segmentation:

Customer segmentation is the process of dividing customers into distinct groups based on their behaviors, preferences, and needs. According to Yim et al. (2004), customer segmentation helps businesses to better understand their customers and develop more effective marketing strategies.

The study suggests that customer segmentation can improve customer satisfaction, loyalty, and retention.In another study, Verhoef et al. (2010) highlight the importance of customer segmentation for customer relationship management. The study suggests that businesses can use customer segmentation to develop personalized marketing strategies that are more likely to resonate with customers.

2. RFM Analysis:

RFM analysis is a method used to identify a company's most valuable customers based on their purchasing behavior. RFM stands for Recency, Frequency, and Monetary Value. According to Fader and Hardie (2010), RFM analysis is a valuable tool for businesses to identify their most profitable customers and develop strategies to retain them.

In a study by Gupta and Lehmann (2006), the authors suggest that RFM analysis can be used to predict future customer behavior and help businesses to develop targeted marketing campaigns. The study also suggests that RFM analysis can be combined with other data analysis techniques such as predictive modeling to improve the accuracy of customer segmentation.

3. Customer Loyalty and Retention:

Customer loyalty and retention are critical for businesses to maintain a sustainable customer base and increase revenue. According to Reichheld (1996), customer loyalty is essential for businesses to achieve long-term success. The study suggests that businesses can increase customer loyalty by providing high-quality products and services and developing strong relationships with their customers.

In a study by Kim et al. (2010), the authors suggest that businesses can use customer segmentation and RFM analysis to identify customers who are most likely to defect and develop strategies to retain them. The study highlights the importance of customer retention for businesses to maintain a loyal customer base and increase revenue.

4. Amazon EMR and Apache Spark

Amazon Elastic MapReduce (EMR) is a web service that allows businesses to process large amounts of data using a distributed computing framework. One of the most popular distributed computing frameworks supported by Amazon EMR is Apache Spark. Spark is an open-source, in-memory distributed computing framework that provides high performance and scalability for big data processing. Amazon EMR makes it easy to set up and manage Spark clusters on the cloud, allowing businesses to focus on data processing rather than infrastructure management.

Spark is well-suited for data processing tasks such as ETL (extract, transform, load), data mining, machine learning, and graph processing. Spark's ability to process data in-memory provides faster performance compared to traditional disk-based processing frameworks. Spark's API also supports multiple programming languages such as Python, Java, and Scala, making it a versatile choice for data processing tasks.

Amazon EMR provides pre-configured Spark clusters that can be easily customized based on specific business needs. EMR also offers integration with other AWS services such as Amazon S3 (Simple Storage Service), Amazon Redshift (data warehouse service), and Amazon Kinesis (real-time data streaming service), making it easy to ingest and process data from various sources.

In conclusion, Amazon EMR and Spark provide a powerful combination for processing large amounts of data in a scalable and cost-effective manner. Spark's in-memory processing capabilities and multi-language API make it a versatile choice for a wide range of data processing tasks, while Amazon EMR simplifies cluster management and integrates with other AWS services for seamless data processing workflows.

5. KNN Algorithm

The K-nearest neighbors (KNN) algorithm is a type of supervised learning algorithm used for classification and regression tasks. The KNN algorithm works by finding the K closest training examples in the feature space to a given test example, and using the class labels of these neighbors to predict the label of the test example.

The KNN algorithm is a simple but effective algorithm that can be used for both binary and multi-class classification problems. One of the advantages of the KNN algorithm is that it does not require any assumptions about the underlying distribution of the data, making it a non-parametric algorithm. Additionally, KNN can handle non-linear decision boundaries, which makes it useful for a wide range of problems.

However, the main disadvantage of the KNN algorithm is its computational complexity. As the size of the dataset grows, the cost of finding the K-nearest neighbors for each test example can become prohibitively expensive. Additionally, KNN can be sensitive to the choice of K, and choosing the optimal value of K can be a challenging task.

In conclusion, the KNN algorithm is a powerful tool for solving classification and regression problems, particularly when the underlying distribution of the data is unknown or non-linear. However, its high computational cost and sensitivity to the choice of K should be taken into account when deciding whether to use it for a particular task.

3. LIBRARIES USED

1. Pandas

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called *cleaning* the data.

2. Numpy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

3. Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib

4. Sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Important features of scikit-learn:

- Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Accessible to everybody and reusable in various contexts.
- Built on the top of NumPy, SciPy, and matplotlib.
- Open source, commercially usable – BSD license

xvluswdvp

March 5, 2023

[1]:
```
#Import necessary libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
#Import Online Retail Data containing transactions from 01/12/2010 and 09/12/
↪2011
Rtl_data = pd.read_csv('RetailData.csv', encoding = 'unicode_escape')
Rtl_data.head()
```

[2]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

        InvoiceDate  UnitPrice  CustomerID         Country
0  12-01-2010 08:26       2.55     17850.0  United Kingdom
1  12-01-2010 08:26       3.39     17850.0  United Kingdom
2  12-01-2010 08:26       2.75     17850.0  United Kingdom
3  12-01-2010 08:26       3.39     17850.0  United Kingdom
4  12-01-2010 08:26       3.39     17850.0  United Kingdom
```

[3]:
```
#Check the shape (number of columns and rows) in the dataset
Rtl_data.shape
```

[3]: (541909, 8)

[4]:
```
#Customer distribution by country
country_cust_data=Rtl_data[['Country','CustomerID']].drop_duplicates()
country_cust_data.groupby(['Country'])['CustomerID'].aggregate('count').
↪reset_index().sort_values('CustomerID', ascending=False)
```

[4]:
```
                  Country  CustomerID
36         United Kingdom        3950
```

```
14                 Germany           95
13                  France           87
31                   Spain           31
3                  Belgium           25
33             Switzerland           21
27                Portugal           19
19                   Italy           15
12                 Finland           12
1                  Austria           11
25                  Norway           10
24             Netherlands            9
0                Australia            9
6          Channel Islands            9
9                  Denmark            9
7                   Cyprus            8
32                  Sweden            8
20                   Japan            8
26                  Poland            6
34                     USA            4
5                   Canada            4
37             Unspecified            4
18                  Israel            4
15                  Greece            4
10                    EIRE            3
23                   Malta            2
35    United Arab Emirates            2
2                  Bahrain            2
22               Lithuania            1
8           Czech Republic            1
21                 Lebanon            1
28                     RSA            1
29            Saudi Arabia            1
30               Singapore            1
17                 Iceland            1
4                   Brazil            1
11      European Community            1
16               Hong Kong            0
```

[5]:
```
#Keep only United Kingdom data
Rtl_data = Rtl_data.query("Country=='United Kingdom'").reset_index(drop=True)
```

[6]:
```
#Check for missing values in the dataset
Rtl_data.isnull().sum(axis=0)
```

[6]:
```
InvoiceNo         0
StockCode         0
Description    1454
```

```
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     133600
Country             0
dtype: int64
```

[7]:
```
#Remove missing values from CustomerID column, can ignore missing values in␣
↪description column
Rtl_data = Rtl_data[pd.notnull(Rtl_data['CustomerID'])]

#Validate if there are any negative values in Quantity column
Rtl_data.Quantity.min()
```

[7]:
```
-80995
```

[8]:
```
#Validate if there are any negative values in UnitPrice column
Rtl_data.UnitPrice.min()
```

[8]:
```
0.0
```

[9]:
```
#Filter out records with negative values
Rtl_data = Rtl_data[(Rtl_data['Quantity']>0)]
```

[10]:
```
#Convert the string date field to datetime
Rtl_data['InvoiceDate'] = pd.to_datetime(Rtl_data['InvoiceDate'])
```

[11]:
```
#Add new column depicting total amount
Rtl_data['TotalAmount'] = Rtl_data['Quantity'] * Rtl_data['UnitPrice']
```

[12]:
```
#Check the shape (number of columns and rows) in the dataset after data is␣
↪cleaned
Rtl_data.shape
```

[12]:
```
(354345, 9)
```

[13]:
```
Rtl_data.head()
```

[13]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country  TotalAmount
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom        15.30
```

```
1 2010-12-01 08:26:00        3.39     17850.0  United Kingdom         20.34
2 2010-12-01 08:26:00        2.75     17850.0  United Kingdom         22.00
3 2010-12-01 08:26:00        3.39     17850.0  United Kingdom         20.34
4 2010-12-01 08:26:00        3.39     17850.0  United Kingdom         20.34
```

0.1 RFM Modelling

[14]:
```
#Recency = Latest Date - Last Inovice Data, Frequency = count of invoice no. of␣
↪transaction(s), Monetary = Sum of Total
#Amount for each customer
import datetime as dt

#Set Latest date 2011-12-10 as last invoice date was 2011-12-09. This is to␣
↪calculate the number of days from recent purchase
Latest_Date = dt.datetime(2011,12,10)

#Create RFM Modelling scores for each customer
RFMScores = Rtl_data.groupby('CustomerID').agg({'InvoiceDate': lambda x:␣
↪(Latest_Date - x.max()).days, 'InvoiceNo': lambda x: len(x), 'TotalAmount':␣
↪lambda x: x.sum()})

#Convert Invoice Date into type int
RFMScores['InvoiceDate'] = RFMScores['InvoiceDate'].astype(int)

#Rename column names to Recency, Frequency and Monetary
RFMScores.rename(columns={'InvoiceDate': 'Recency',
                         'InvoiceNo': 'Frequency',
                         'TotalAmount': 'Monetary'}, inplace=True)

RFMScores.reset_index().head()
```

[14]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	325	1	77183.60
1	12747.0	2	103	4196.01
2	12748.0	0	4596	33719.73
3	12749.0	3	199	4090.88
4	12820.0	3	59	942.34

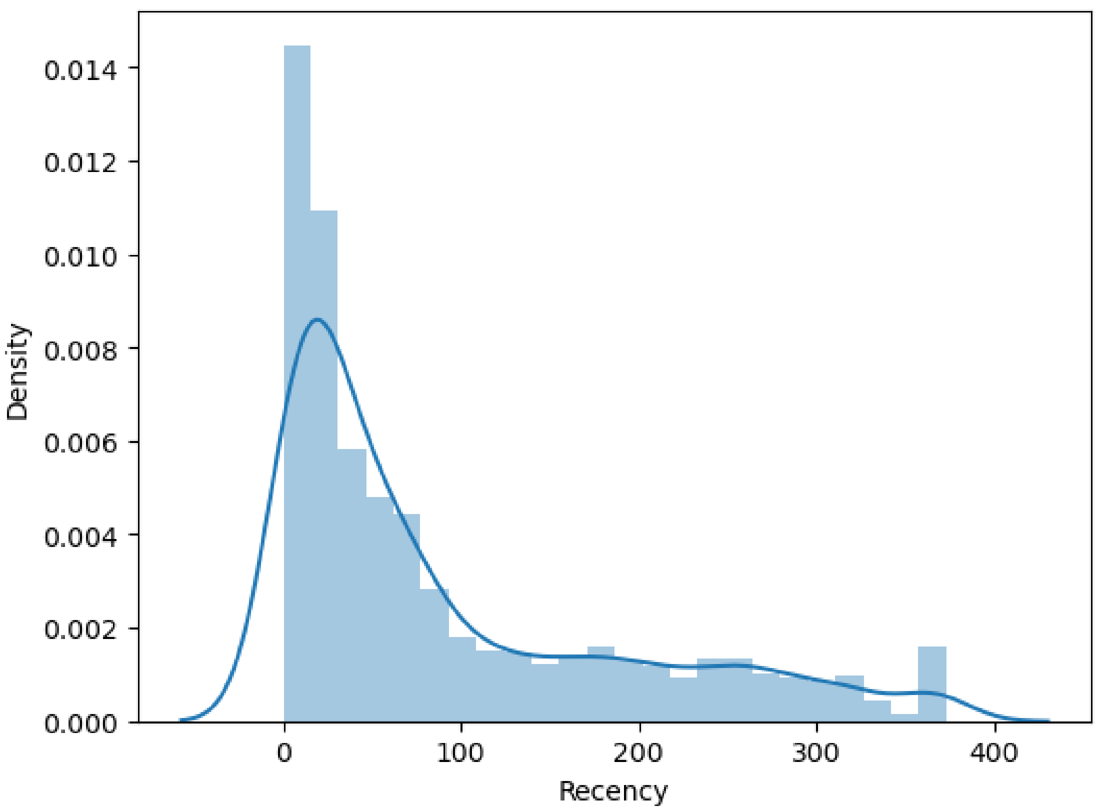
[15]:
```
#Descriptive Statistics (Recency)
RFMScores.Recency.describe()
```

[15]:
```
count    3921.000000
mean       91.722265
std        99.528532
min         0.000000
25%        17.000000
```

```
50%         50.000000
75%        142.000000
max        373.000000
Name: Recency, dtype: float64
```

[16]:
```
#Recency distribution plot
import seaborn as sns
x = RFMScores['Recency']

ax = sns.distplot(x)
```

```
C:\Users\somes\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 figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

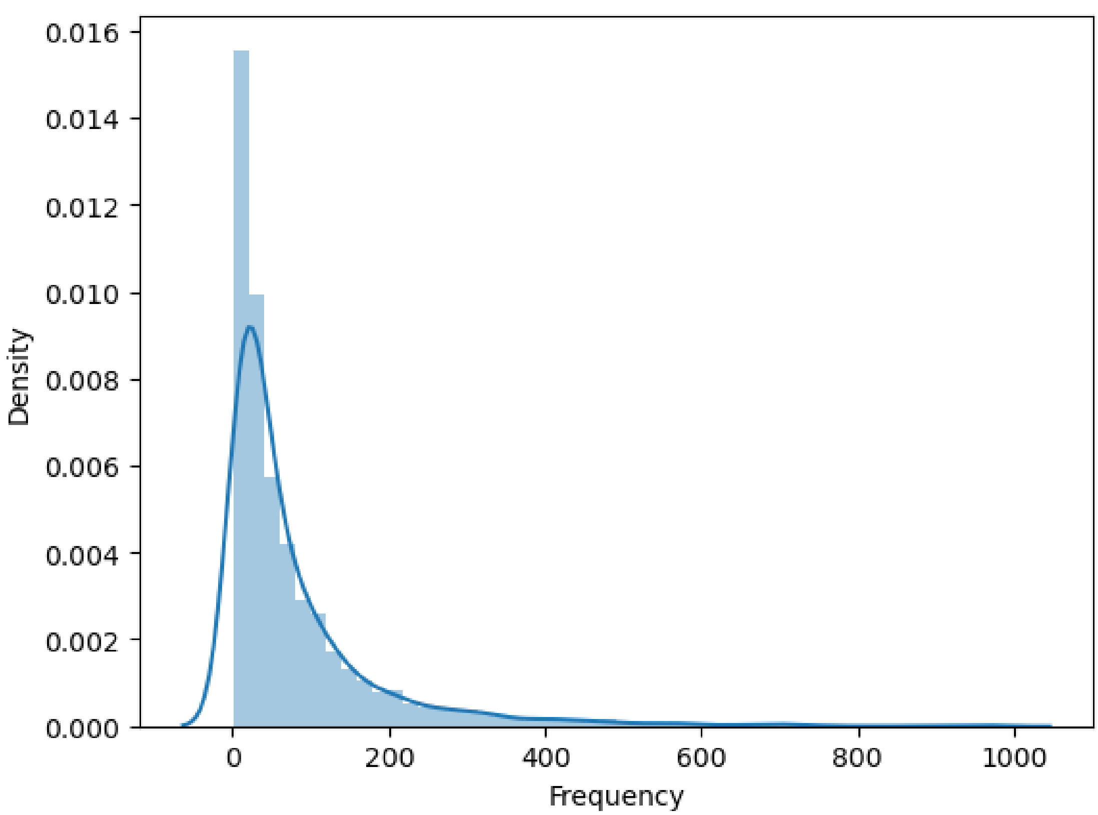


[17]:
```
#Descriptive Statistics (Frequency)
RFMScores.Frequency.describe()
```

```
[17]: count    3921.000000
mean       90.371079
std       217.796155
min         1.000000
25%        17.000000
50%        41.000000
75%        99.000000
max      7847.000000
Name: Frequency, dtype: float64
```

```
[18]: #Frequency distribution plot, taking observations which have frequency less␣
      ↪than 1000
import seaborn as sns
x = RFMScores.query('Frequency < 1000')['Frequency']

ax = sns.distplot(x)
```

```
C:\Users\somes\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 figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

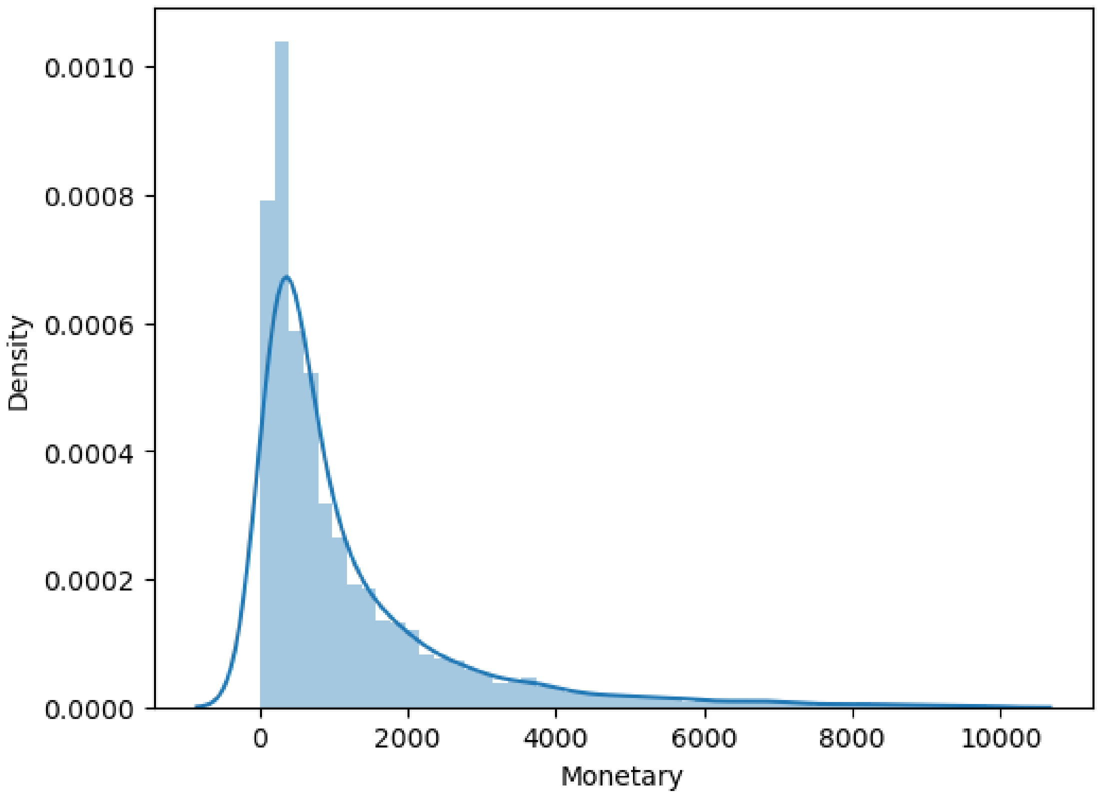


[19]:
```python
#Descriptive Statistics (Monetary)
RFMScores.Monetary.describe()
```

[19]:
```
count      3921.000000
mean       1863.910113
std        7481.922217
min           0.000000
25%         300.040000
50%         651.820000
75%        1575.890000
max      259657.300000
Name: Monetary, dtype: float64
```

[20]:
```python
#Monateray distribution plot, taking observations which have monetary value␣
↪less than 10000
import seaborn as sns
x = RFMScores.query('Monetary < 10000')['Monetary']

ax = sns.distplot(x)
```

```
C:\Users\somes\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 figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



[21]:
```
#Split into four segments using quantiles
quantiles = RFMScores.quantile(q=[0.25,0.5,0.75])
quantiles = quantiles.to_dict()
```

[22]:
```
quantiles
```

[22]:
```
{'Recency': {0.25: 17.0, 0.5: 50.0, 0.75: 142.0},
 'Frequency': {0.25: 17.0, 0.5: 41.0, 0.75: 99.0},
 'Monetary': {0.25: 300.03999999999996, 0.5: 651.8199999999999, 0.75: 1575.89}}
```

[23]:
```
#Functions to create R, F and M segments
def RScoring(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
```

```python
def FnMScoring(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
```

[24]:
```python
#Calculate Add R, F and M segment value columns in the existing dataset to show␣
↪R, F and M segment values
RFMScores['R'] = RFMScores['Recency'].apply(RScoring,␣
↪args=('Recency',quantiles,))
RFMScores['F'] = RFMScores['Frequency'].apply(FnMScoring,␣
↪args=('Frequency',quantiles,))
RFMScores['M'] = RFMScores['Monetary'].apply(FnMScoring,␣
↪args=('Monetary',quantiles,))
RFMScores.head()

# Lower the RFM Score better or loyal is the customer
```

[24]:

CustomerID	Recency	Frequency	Monetary	R	F	M
12346.0	325	1	77183.60	4	4	1
12747.0	2	103	4196.01	1	1	1
12748.0	0	4596	33719.73	1	1	1
12749.0	3	199	4090.88	1	1	1
12820.0	3	59	942.34	1	2	2

[25]:
```python
#Calculate and Add RFMGroup value column showing combined concatenated score of␣
↪RFM
RFMScores['RFMGroup'] = RFMScores.R.map(str) + RFMScores.F.map(str) + RFMScores.
↪M.map(str)

#Calculate and Add RFMScore value column showing total sum of RFMGroup values
RFMScores['RFMScore'] = RFMScores[['R', 'F', 'M']].sum(axis = 1)
RFMScores.head()
```

[25]:

CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore
12346.0	325	1	77183.60	4	4	1	441	9
12747.0	2	103	4196.01	1	1	1	111	3
12748.0	0	4596	33719.73	1	1	1	111	3
12749.0	3	199	4090.88	1	1	1	111	3
12820.0	3	59	942.34	1	2	2	122	5

[26]:
```
#Assign Loyalty Level to each customer
Loyalty_Level = ['Platinum', 'Gold', 'Silver', 'Bronze']
Score_cuts = pd.qcut(RFMScores.RFMScore, q = 4, labels = Loyalty_Level)
RFMScores['RFM_Loyalty_Level'] = Score_cuts.values
RFMScores.reset_index().head()
```

[26]:
```
   CustomerID  Recency  Frequency  Monetary  R  F  M RFMGroup  RFMScore  \
0     12346.0      325          1  77183.60  4  4  1      441         9
1     12747.0        2        103   4196.01  1  1  1      111         3
2     12748.0        0       4596  33719.73  1  1  1      111         3
3     12749.0        3        199   4090.88  1  1  1      111         3
4     12820.0        3         59    942.34  1  2  2      122         5

  RFM_Loyalty_Level
0            Silver
1          Platinum
2          Platinum
3          Platinum
4          Platinum
```

[27]:
```
#Validate the data for RFMGroup = 111
RFMScores[RFMScores['RFMGroup']=='111'].sort_values('Monetary',␣
↪ascending=False).reset_index().head(10)
```

[27]:
```
   CustomerID  Recency  Frequency   Monetary  R  F  M RFMGroup  RFMScore  \
0     18102.0        0        431  259657.30  1  1  1      111         3
1     17450.0        8        337  194550.79  1  1  1      111         3
2     17511.0        2        963   91062.38  1  1  1      111         3
3     16684.0        4        277   66653.56  1  1  1      111         3
4     14096.0        4       5111   65164.79  1  1  1      111         3
5     13694.0        3        568   65039.62  1  1  1      111         3
6     15311.0        0       2379   60767.90  1  1  1      111         3
7     13089.0        2       1818   58825.83  1  1  1      111         3
8     15769.0        7        130   56252.72  1  1  1      111         3
9     15061.0        3        403   54534.14  1  1  1      111         3

  RFM_Loyalty_Level
0          Platinum
1          Platinum
2          Platinum
3          Platinum
4          Platinum
5          Platinum
6          Platinum
7          Platinum
8          Platinum
9          Platinum
```

[28]:
```
pip install chart_studio
```

```
Requirement already satisfied: chart_studio in
c:\users\somes\anaconda3\lib\site-packages (1.1.0)
Requirement already satisfied: plotly in c:\users\somes\anaconda3\lib\site-
packages (from chart_studio) (5.9.0)
Requirement already satisfied: retrying>=1.3.3 in
c:\users\somes\anaconda3\lib\site-packages (from chart_studio) (1.3.4)
Requirement already satisfied: requests in c:\users\somes\anaconda3\lib\site-
packages (from chart_studio) (2.28.1)
Requirement already satisfied: six in c:\users\somes\anaconda3\lib\site-packages
(from chart_studio) (1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\somes\anaconda3\lib\site-packages (from plotly->chart_studio) (8.0.1)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\somes\anaconda3\lib\site-packages (from requests->chart_studio)
(2022.9.14)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
c:\users\somes\anaconda3\lib\site-packages (from requests->chart_studio)
(1.26.11)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\somes\anaconda3\lib\site-packages (from requests->chart_studio) (3.3)
Requirement already satisfied: charset-normalizer<3,>=2 in
c:\users\somes\anaconda3\lib\site-packages (from requests->chart_studio) (2.0.4)
Note: you may need to restart the kernel to use updated packages.
```

[29]:
```
import chart_studio as cs
import plotly.offline as po
import plotly.graph_objs as gobj

#Recency Vs Frequency
graph = RFMScores.query("Monetary < 50000 and Frequency < 2000")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Bronze'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'Bronze'")['Frequency'],
        mode='markers',
        name='Bronze',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Silver'")['Recency'],
```

```
        y=graph.query("RFM_Loyalty_Level == 'Silver'")['Frequency'],
        mode='markers',
        name='Silver',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Gold'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'Gold'")['Frequency'],
        mode='markers',
        name='Gold',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Platinum'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'Platinum'")['Frequency'],
        mode='markers',
        name='Platinum',
        marker= dict(size= 13,
            line= dict(width=1),
            color= 'black',
            opacity= 0.9
           )
    ),
]

plot_layout = gobj.Layout(
        yaxis= {'title': "Frequency"},
        xaxis= {'title': "Recency"},
        title='Segments'
    )
fig = gobj.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)

#Frequency Vs Monetary
graph = RFMScores.query("Monetary < 50000 and Frequency < 2000")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Bronze'")['Frequency'],
```

```
        y=graph.query("RFM_Loyalty_Level == 'Bronze'")['Monetary'],
        mode='markers',
        name='Bronze',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Silver'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'Silver'")['Monetary'],
        mode='markers',
        name='Silver',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Gold'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'Gold'")['Monetary'],
        mode='markers',
        name='Gold',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Platinum'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'Platinum'")['Monetary'],
        mode='markers',
        name='Platinum',
        marker= dict(size= 13,
            line= dict(width=1),
            color= 'black',
            opacity= 0.9
           )
    ),
]

plot_layout = gobj.Layout(
        yaxis= {'title': "Monetary"},
        xaxis= {'title': "Frequency"},
```

```
        title='Segments'
    )
fig = gobj.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)

#Recency Vs Monetary
graph = RFMScores.query("Monetary < 50000 and Frequency < 2000")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Bronze'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'Bronze'")['Monetary'],
        mode='markers',
        name='Bronze',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Silver'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'Silver'")['Monetary'],
        mode='markers',
        name='Silver',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Gold'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'Gold'")['Monetary'],
        mode='markers',
        name='Gold',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Platinum'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'Platinum'")['Monetary'],
        mode='markers',
        name='Platinum',
```

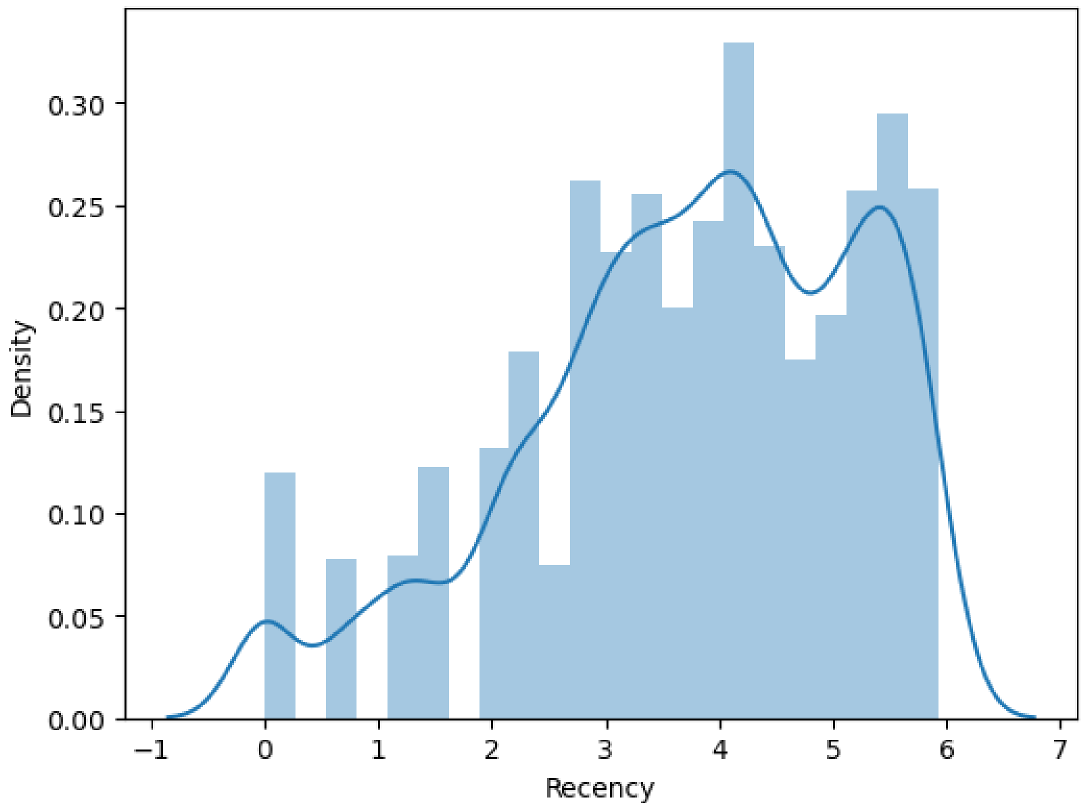
```
        marker= dict(size= 13,
            line= dict(width=1),
            color= 'black',
            opacity= 0.9
           )
    ),
]

plot_layout = gobj.Layout(
        yaxis= {'title': "Monetary"},
        xaxis= {'title': "Recency"},
        title='Segments'
    )
fig = gobj.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```

0.2 K-Means Clustering

[30]:
```
#Handle negative and zero values so as to handle infinite numbers during log␣
 ↪transformation
def handle_neg_n_zero(num):
    if num <= 0:
        return 1
    else:
        return num
#Apply handle_neg_n_zero function to Recency and Monetary columns
RFMScores['Recency'] = [handle_neg_n_zero(x) for x in RFMScores.Recency]
RFMScores['Monetary'] = [handle_neg_n_zero(x) for x in RFMScores.Monetary]

#Perform Log transformation to bring data into normal or near normal␣
 ↪distribution
Log_Tfd_Data = RFMScores[['Recency', 'Frequency', 'Monetary']].apply(np.log,␣
 ↪axis = 1).round(3)
```

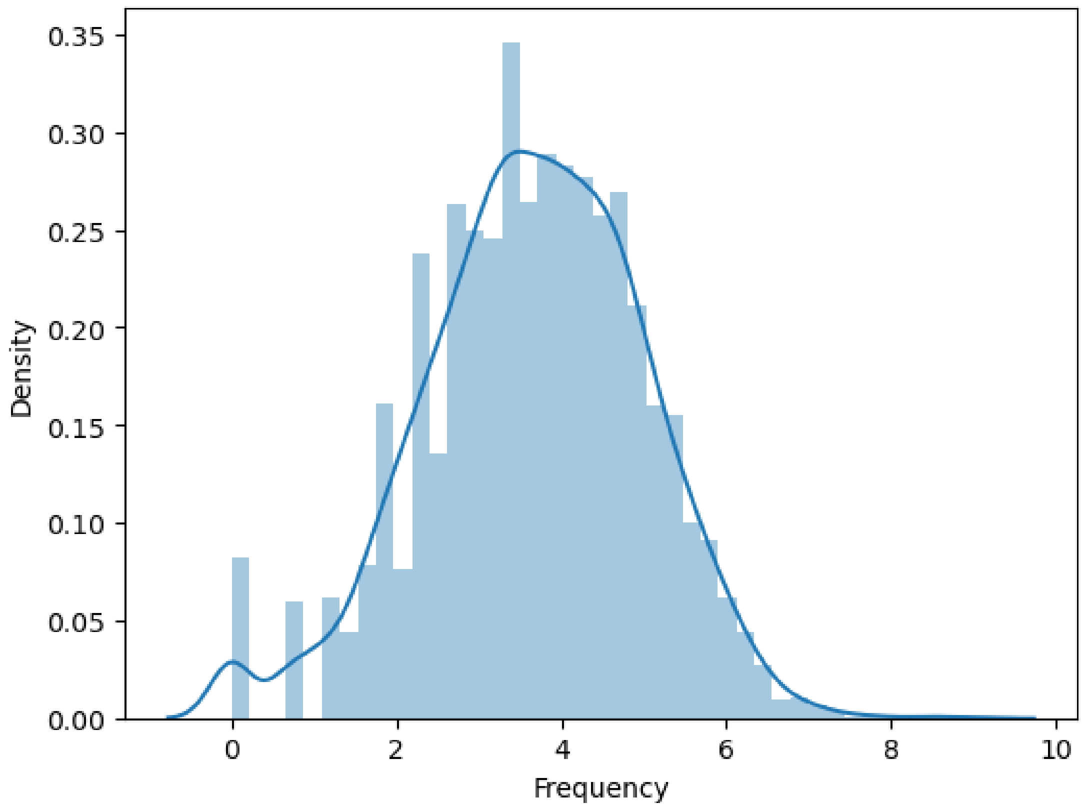
[31]:
```
#Data distribution after data normalization for Recency
Recency_Plot = Log_Tfd_Data['Recency']
ax = sns.distplot(Recency_Plot)
```

```
C:\Users\somes\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 figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



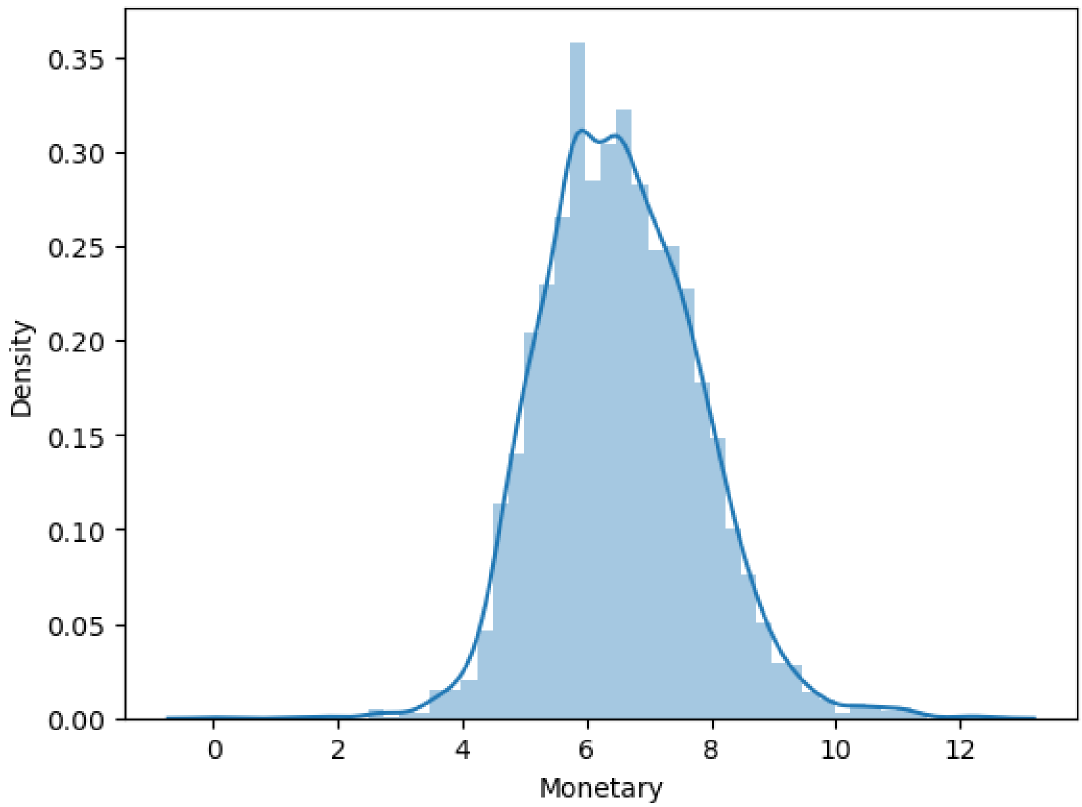
[32]:
```
#Data distribution after data normalization for Frequency
Frequency_Plot = Log_Tfd_Data.query('Frequency < 1000')['Frequency']
ax = sns.distplot(Frequency_Plot)
```

```
C:\Users\somes\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 figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



[33]:
```
#Data distribution after data normalization for Monetary
Monetary_Plot = Log_Tfd_Data.query('Monetary < 10000')['Monetary']
ax = sns.distplot(Monetary_Plot)
```

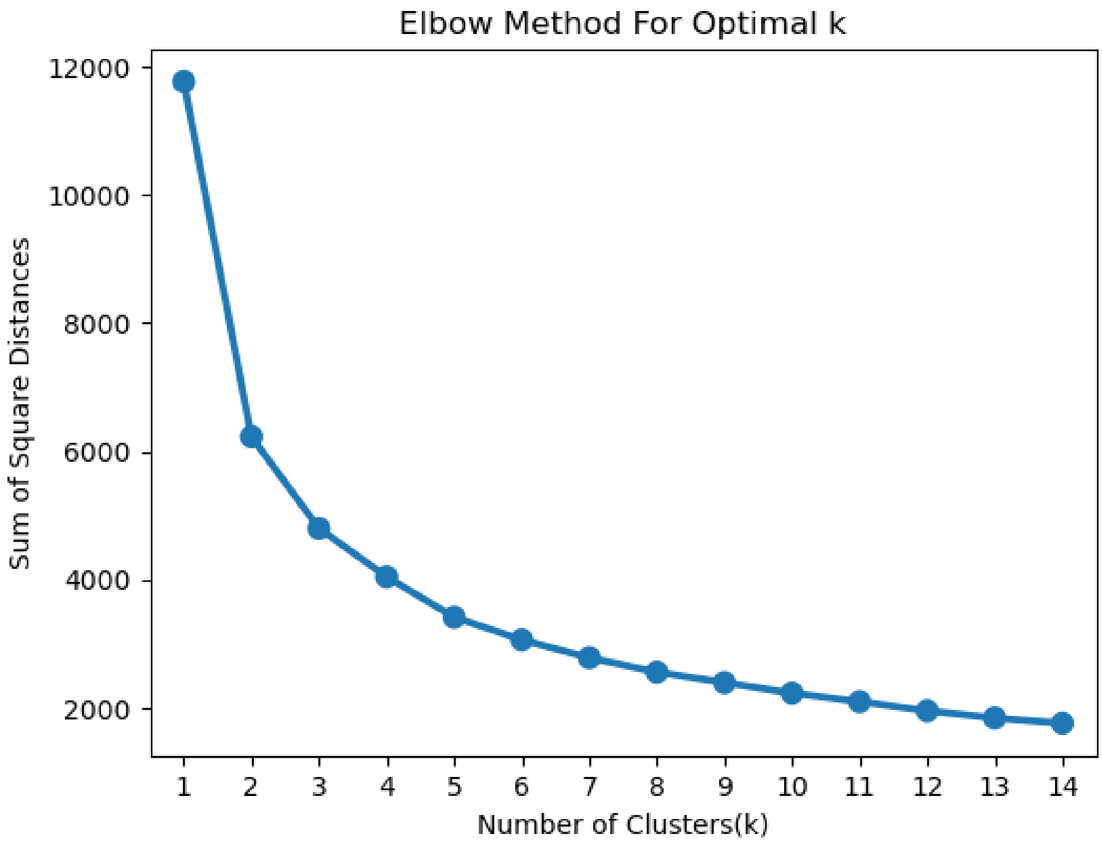
```
C:\Users\somes\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



[34]:
```
from sklearn.preprocessing import StandardScaler

#Bring the data on same scale
scaleobj = StandardScaler()
Scaled_Data = scaleobj.fit_transform(Log_Tfd_Data)

#Transform it back to dataframe
Scaled_Data = pd.DataFrame(Scaled_Data, index = RFMScores.index, columns =␣
↪Log_Tfd_Data.columns)
```

[35]:
```
from sklearn.cluster import KMeans

sum_of_sq_dist = {}
for k in range(1,15):
    km = KMeans(n_clusters= k, init= 'k-means++', max_iter= 1000)
    km = km.fit(Scaled_Data)
    sum_of_sq_dist[k] = km.inertia_

#Plot the graph for the sum of square distance values and Number of Clusters
sns.pointplot(x = list(sum_of_sq_dist.keys()), y = list(sum_of_sq_dist.
↪values()))
```

```
plt.xlabel('Number of Clusters(k)')
plt.ylabel('Sum of Square Distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



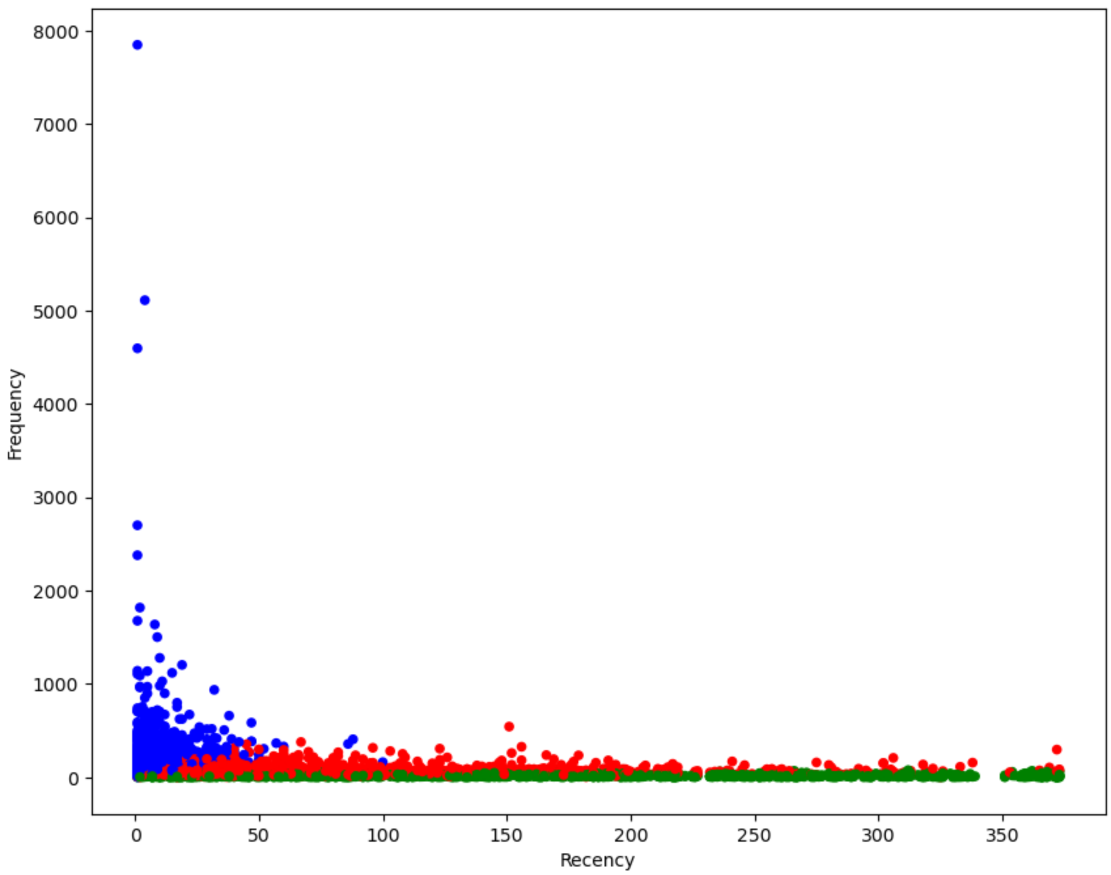
[36]:
```
#Perform K-Mean Clustering or build the K-Means clustering model
KMean_clust = KMeans(n_clusters= 3, init= 'k-means++', max_iter= 1000)
KMean_clust.fit(Scaled_Data)

#Find the clusters for the observation given in the dataset
RFMScores['Cluster'] = KMean_clust.labels_
RFMScores.head()
```

[36]:
```
            Recency  Frequency  Monetary  R  F  M RFMGroup  RFMScore  \
CustomerID
12346.0         325          1  77183.60  4  4  1      441         9
12747.0           2        103   4196.01  1  1  1      111         3
12748.0           1       4596  33719.73  1  1  1      111         3
12749.0           3        199   4090.88  1  1  1      111         3
12820.0           3         59    942.34  1  2  2      122         5
```

```
           RFM_Loyalty_Level  Cluster
CustomerID
12346.0               Silver        0
12747.0             Platinum        2
12748.0             Platinum        2
12749.0             Platinum        2
12820.0             Platinum        2
```

[37]:
```
from matplotlib import pyplot as plt
plt.figure(figsize=(7,7))

##Scatter Plot Frequency Vs Recency
Colors = ["red", "green", "blue"]
RFMScores['Color'] = RFMScores['Cluster'].map(lambda p: Colors[p])
ax = RFMScores.plot(
    kind="scatter",
    x="Recency", y="Frequency",
    figsize=(10,8),
    c = RFMScores['Color']
)
```

```
<Figure size 700x700 with 0 Axes>
```



[38]:
```
RFMScores.head()
```

[38]:
```
            Recency  Frequency  Monetary  R  F  M RFMGroup  RFMScore  \
CustomerID
12346.0         325          1  77183.60  4  4  1      441         9
12747.0           2        103   4196.01  1  1  1      111         3
12748.0           1       4596  33719.73  1  1  1      111         3
12749.0           3        199   4090.88  1  1  1      111         3
12820.0           3         59    942.34  1  2  2      122         5

           RFM_Loyalty_Level  Cluster Color
CustomerID
12346.0               Silver        0   red
12747.0             Platinum        2  blue
12748.0             Platinum        2  blue
12749.0             Platinum        2  blue
12820.0             Platinum        2  blue
```

[39]:
```
# RFMScores.to_csv('RFMScores.csv')
```

[40]:
```
# pip install pyspark
```

[41]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
```

[42]:
```
spark = SparkSession.builder.config('spark.some.config.option','some-value').
  ↪getOrCreate()
spark
```

[42]:
```
<pyspark.sql.session.SparkSession at 0x2e79e68f4c0>
```

[43]:
```
from pyspark.sql.types import StringType, StringType, IntegerType, DoubleType,␣
  ↪DataType, LongType
```

[44]:
```
schema = StructType([
    StructField('CustomerID', StringType(), True),
    StructField('Recency',IntegerType(), True),
    StructField('Frequency',IntegerType(), True),
    StructField('Monetary',DoubleType(), True),
    StructField('R',IntegerType(), True),
    StructField('F',IntegerType(), True),
    StructField('M',IntegerType(), True),
    StructField('RFMGroup',StringType(), True),
    StructField('RFMScore',StringType(), True),
    StructField('RFM_Loyalty_Level',StringType(), True),
    StructField('Cluster',IntegerType(), True),
    StructField('Color',StringType(), True)

])
```

[45]:
```
df = spark.read.format("csv").option("header","True").schema(schema).
  ↪load("RFMScores.csv")
```

[46]:
```
df.head(5)
```

[46]:
```
[Row(CustomerID='12346.0', Recency=325, Frequency=1, Monetary=77183.6, R=4, F=4,
M=1, RFMGroup='441', RFMScore='9', RFM_Loyalty_Level='Silver', Cluster=2,
Color='blue'),
 Row(CustomerID='12747.0', Recency=2, Frequency=103, Monetary=4196.009999999999,
R=1, F=1, M=1, RFMGroup='111', RFMScore='3', RFM_Loyalty_Level='Platinum',
Cluster=0, Color='red'),
 Row(CustomerID='12748.0', Recency=1, Frequency=4596, Monetary=33719.73, R=1,
F=1, M=1, RFMGroup='111', RFMScore='3', RFM_Loyalty_Level='Platinum', Cluster=0,
Color='red'),
 Row(CustomerID='12749.0', Recency=3, Frequency=199, Monetary=4090.88, R=1, F=1,
M=1, RFMGroup='111', RFMScore='3', RFM_Loyalty_Level='Platinum', Cluster=0,
Color='red'),
```

```
 Row(CustomerID='12820.0', Recency=3, Frequency=59, Monetary=942.34, R=1, F=2,
M=2, RFMGroup='122', RFMScore='5', RFM_Loyalty_Level='Platinum', Cluster=0,
Color='red')]
```

[47]:
```
df.registerTempTable("RFM")
```

```
C:\Users\somes\anaconda3\lib\site-packages\pyspark\sql\dataframe.py:229:
FutureWarning:

Deprecated in 2.0, use createOrReplaceTempView instead.
```

[48]:
```
spark.sql("select * from RFM").show(5)
```

```
+----------+-------+---------+-----------------+---+---+---+--------+--------+--
---------------+-------+-----+
|CustomerID|Recency|Frequency|         Monetary|  R|  F|
M|RFMGroup|RFMScore|RFM_Loyalty_Level|Cluster|Color|
+----------+-------+---------+-----------------+---+---+---+--------+--------+--
---------------+-------+-----+
|   12346.0|    325|        1|          77183.6|  4|  4|  1|     441|       9|
Silver|      2| blue|
|   12747.0|      2|      103|4196.009999999999|  1|  1|  1|     111|       3|
Platinum|      0|  red|
|   12748.0|      1|     4596|         33719.73|  1|  1|  1|     111|       3|
Platinum|      0|  red|
|   12749.0|      3|      199|          4090.88|  1|  1|  1|     111|       3|
Platinum|      0|  red|
|   12820.0|      3|       59|           942.34|  1|  2|  2|     122|       5|
Platinum|      0|  red|
+----------+-------+---------+-----------------+---+---+---+--------+--------+--
---------------+-------+-----+
only showing top 5 rows
```

[49]:
```
# How many Distinct customers do we have in our database
spark.sql('SELECT COUNT(DISTINCT CustomerID) AS num_customers FROM RFM').show()
```

```
+-------------+
|num_customers|
+-------------+
|         3921|
+-------------+
```

[50]:
```
# What is the Monetary value generated by each customer?
```

```
spark.sql('SELECT CustomerID, SUM(Monetary) AS Monetary_value FROM RFM GROUP BY␣
↪CustomerID').show()
```

```
+----------+------------------+
|CustomerID|    Monetary_value|
+----------+------------------+
|   12891.0|             331.0|
|   12985.0|1239.3799999999999|
|   13067.0|115.46000000000001|
|   13178.0| 5725.469999999999|
|   13259.0|292.31999999999994|
|   13514.0|152.20000000000002|
|   14349.0|             133.5|
|   14542.0|103.25000000000001|
|   15039.0|          19914.44|
|   15396.0|288.17999999999995|
|   15891.0|            524.52|
|   16553.0|           5719.82|
|   16557.0|281.84999999999997|
|   16917.0|391.52000000000004|
|   16982.0|            384.06|
|   17786.0|            278.74|
|   17955.0|             557.3|
|   17966.0|1098.4299999999998|
|   13499.0|           1159.11|
|   13827.0|            412.05|
+----------+------------------+
only showing top 20 rows
```

[51]:
```
# What are the top 10% of customers in terms of monetary value?

spark.sql('WITH customer_revenue AS (SELECT CustomerID, SUM(Monetary) AS␣
↪total_revenue FROM RFM GROUP BY CustomerID ) SELECT CustomerID,␣
↪total_revenue FROM ( SELECT CustomerID, total_revenue, NTILE(10) OVER (ORDER␣
↪BY total_revenue DESC) AS percentile FROM customer_revenue ) AS␣
↪customer_percentiles WHERE percentile = 1').show()
```

```
+----------+------------------+
|CustomerID|     total_revenue|
+----------+------------------+
|   18102.0|          259657.3|
|   17450.0|194550.78999999998|
|   16446.0|          168472.5|
|   17511.0|          91062.38|
|   16029.0|          81024.84|
|   12346.0|           77183.6|
|   16684.0|          66653.56|
```

```
|    14096.0|          65164.79|
|    13694.0|65039.619999999995|
|    15311.0|60767.899999999994|
|    13089.0|          58825.83|
|    17949.0| 58510.48000000001|
|    15769.0|          56252.72|
|    15061.0|          54534.14|
|    14298.0|51527.299999999996|
|    14088.0|          50491.81|
|    15749.0|           44534.3|
|    12931.0|          42055.96|
|    17841.0|          40991.57|
|    15098.0|           39916.5|
+----------+------------------+
only showing top 20 rows
```

[52]:

```
# What are the Top 10% of customers in terms of RFM Score ?

spark.sql('WITH customer_revenue AS (SELECT CustomerID, SUM(RFMScore) AS␣
↪RFM_Score FROM RFM GROUP BY CustomerID ) SELECT CustomerID, RFM_Score FROM (␣
↪SELECT CustomerID, RFM_Score, NTILE(10) OVER (ORDER BY RFM_Score Asc) AS␣
↪percentile FROM customer_revenue ) AS customer_percentiles WHERE percentile␣
↪= 1').show()
```

```
+----------+---------+
|CustomerID|RFM_Score|
+----------+---------+
|    15039.0|      3.0|
|    14493.0|      3.0|
|    17450.0|      3.0|
|    14092.0|      3.0|
|    15005.0|      3.0|
|    15113.0|      3.0|
|    17811.0|      3.0|
|    17685.0|      3.0|
|    13268.0|      3.0|
|    14178.0|      3.0|
|    14395.0|      3.0|
|    16033.0|      3.0|
|    15856.0|      3.0|
|    17750.0|      3.0|
|    15089.0|      3.0|
|    14099.0|      3.0|
|    18041.0|      3.0|
|    18223.0|      3.0|
|    13694.0|      3.0|
|    13755.0|      3.0|
```

```
+----------+---------+
only showing top 20 rows
```

[53]:
```
# What are the Least 10% of customers in terms of RFM Score ?

spark.sql('WITH customer_revenue AS (SELECT CustomerID, SUM(RFMScore) AS␣
↪RFM_Score FROM RFM GROUP BY CustomerID ) SELECT CustomerID, RFM_Score FROM (␣
↪SELECT CustomerID, RFM_Score, NTILE(10) OVER (ORDER BY RFM_Score DESC) AS␣
↪percentile FROM customer_revenue ) AS customer_percentiles WHERE percentile␣
↪= 1').show()
```

```
+----------+---------+
|CustomerID|RFM_Score|
+----------+---------+
|   14542.0|     12.0|
|   17536.0|     12.0|
|   14727.0|     12.0|
|   15070.0|     12.0|
|   16351.0|     12.0|
|   13672.0|     12.0|
|   15143.0|     12.0|
|   16144.0|     12.0|
|   16050.0|     12.0|
|   13161.0|     12.0|
|   17128.0|     12.0|
|   14241.0|     12.0|
|   15724.0|     12.0|
|   13922.0|     12.0|
|   14368.0|     12.0|
|   15256.0|     12.0|
|   14682.0|     12.0|
|   15083.0|     12.0|
|   16598.0|     12.0|
|   16963.0|     12.0|
+----------+---------+
only showing top 20 rows
```

[57]:
```
# Count the number of customers in each RFM Segment :

spark.sql('SELECT CONCAT(R, F, M) AS rfm_segment, COUNT(*) AS customer_count␣
↪FROM RFM GROUP BY rfm_segment').show()
```

```
+-----------+--------------+
|rfm_segment|customer_count|
+-----------+--------------+
|        124|            13|
```

```
|        334|            42|
|        442|            23|
|        234|            52|
|        232|            49|
|        132|            40|
|        433|           180|
|        422|            58|
|        323|            48|
|        112|            81|
|        424|            34|
|        434|            90|
|        113|            16|
|        432|            35|
|        443|           104|
|        133|            66|
|        343|            79|
|        423|            49|
|        441|             8|
|        223|            63|
+-----------+--------------+
only showing top 20 rows
```

Based on the RFM scores, here are some potential recommendations for customers in each group:

Platinum group (RFM scores of 444):

Offer personalized and exclusive promotions or discounts, such as early access to sales or limited-time offers. Provide premium customer service, such as a dedicated account manager or 24/7 support. Invite them to participate in loyalty programs or VIP events. Ask for their feedback and opinions on new products or services. Consider offering complementary products or services that align with their purchase history. Gold group (RFM scores of 344 or 444):

Offer incentives to encourage repeat purchases, such as discount codes or free shipping on their next order. Provide proactive customer service, such as tracking their orders or sending notifications about restocking products they previously purchased. Invite them to participate in referral programs or leave product reviews. Upsell or cross-sell complementary products or services that align with their purchase history. Silver group (RFM scores of 244, 344, or 444):

Provide personalized recommendations based on their purchase history or browsing behavior. Offer incentives to encourage them to try new products or services. Send targeted email campaigns with exclusive promotions or discounts. Provide customer service that is prompt and helpful. Bronze group (RFM scores of 144, 244, 344, or 444):

Provide incentives to encourage them to make a purchase, such as a discount on their first order or free shipping on orders over a certain amount. Offer a welcome series of emails to introduce them to the brand and its products or services. Provide customer service that is friendly and informative. Use retargeting ads to encourage them to complete their purchase or return to the website.

5. CONCLUSION & FUTURE SCOPE

Conclusion:

In conclusion, customer segmentation is a powerful technique that allows businesses to gain a deeper understanding of their customers and create tailored marketing strategies that cater to each customer group's specific needs. By using big data technologies such as Spark and Amazon EMR, businesses can analyze large volumes of data from various sources and identify their most valuable customers using RFM analysis. Power BI provides a powerful data visualization tool that helps businesses gain insights into customer behavior and preferences and make data-driven decisions.

Future Scope:

1. Integration with AI and Machine Learning: As AI and machine learning technologies continue to advance, businesses will be able to analyze customer data more effectively and make more accurate predictions about customer behavior. Integrating these technologies into the customer segmentation process could lead to even more precise and personalized marketing strategies.

2. Real-time segmentation: Currently, customer segmentation is often done using historical data, which may not reflect current customer behavior. Real-time segmentation would allow businesses to analyze customer behavior as it happens, enabling them to respond to changes in customer preferences and needs more quickly

3. Expansion to other channels: This project focuses on analyzing customer behavior in the context of e-commerce. However, businesses can also benefit from analyzing customer behavior across other channels such as social media, mobile apps, and offline interactions. Future work could explore how to integrate data from these channels into the customer segmentation process.

4. Personalization at scale: Personalized marketing is becoming increasingly important as customers expect more personalized experiences from businesses. However, personalizing marketing at scale can be challenging. Future work could explore how to use customer segmentation to create personalized marketing strategies for a large customer base.

5. Integration with customer feedback: Customer feedback can provide valuable insights into customer preferences and needs. Integrating customer feedback into the customer segmentation process could lead to even more accurate and effective segmentation.

6. REFERENCES

1. Yim, C. K., Tse, D. K., & Chan, K. W. (2004). Strengthening customer loyalty through intimacy and passion: Roles of customer-firm affection and customer-staff relationships in services. Journal of Marketing Research, 41(3), 281-292.

2. Verhoef, P. C., Reinartz, W. J., & Krafft, M. (2010). Customer engagement as a new perspective in customer management. Journal of Service Research, 13(3), 247-252.

3. Fader, P., & Hardie, B. (2010). Customer-base analysis in a discrete-time non-contractual setting. Marketing Science, 29(6), 1086-1108.

4. Gupta, S., & Lehmann, D. R. (2006). Customer metrics and their impact on financial performance. Marketing Science, 25(6), 718-739.

5. Reichheld, F. F. (1996). The loyalty effect: The hidden force behind growth, profits, and lasting value. Harvard Business Press.

6. Kim, Y. J., Lee, J. H., & Kim, W. Y. (2010). The role of customer classification in customer relationship management: An application to the banking industry. Expert Systems with Applications, 37(9), 6143-6150.