In [1]:
```
# Importing the libraries needed:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas import ExcelWriter
%matplotlib inline
```

In [2]:
```
# Importing warnings to ignore warnings to have clean presentation of codes and visualizations
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# Importing the dataset
retail_df = pd.read_excel('Online Retail.xlsx')
```

In [4]:
```
retail_df.head()
```

Out[4]:

	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

retail_df.shape

In [5]:
```
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
```

Project Task: Week 1

Data Cleaning

1. Perform a preliminary data inspection and data cleaning

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

In [6]:
```
# To check for missing values in the data
retail_df.isnull().sum()
```

Out[6]:
```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     135080
Country            0
dtype: int64
```

In [7]:
```
# Check the missing values in percentage of the total data
round((retail_df.isnull().sum()/len(retail_df))*100,2)
```

Out[7]:
```
InvoiceNo       0.00
StockCode       0.00
Description     0.27
Quantity        0.00
InvoiceDate     0.00
UnitPrice       0.00
CustomerID     24.93
Country         0.00
dtype: float64
```

In [8]:
```
# Note:
# 1.Two columns contain nulls values. These columns are categorical(Description) and unique identifier( Customer ID).
# 2.Hence, these null values cannot be replaced  or imputed with mean, median or mode
# 3.Recommended approach is to check if there are corresponding Invoice numbers with the CustomerID nullvalues
```

In [9]:
```
custid_null_treat = retail_df[retail_df['CustomerID'].isnull()]['InvoiceNo']
```

In [10]:
```
(retail_df['CustomerID'].isin(retail_df) & retail_df['InvoiceNo'].isin(custid_null_treat)).any()
```

Out[10]:
```
False
```

In [11]:
```
# None of the elements in the 'CustomerID' column of df 'retail_df' match any customer ID in the same column
# No element in the 'InvoiceNo' column match any value in the df 'custid.null_treat'
# No correponding Invoice number contain the CustomerID null values
```

```
# Removing Column 'Description' as variable is not significant
retail_df.drop('Description', axis=1, inplace=True)
retail_df.dropna(inplace=True)
```

In [12]:
```
retail_df.shape
```

Out[12]:
```
(541909, 8)
```

b. Remove duplicate data records.

In [13]:
```
retail_df[retail_df.duplicated()]
```

Out[13]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	2010-12-01 11:45:00	1.25	17908.0	United Kingdom
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	2010-12-01 11:45:00	2.10	17908.0	United Kingdom
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	2010-12-01 11:45:00	2.95	17908.0	United Kingdom
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	2010-12-01 11:45:00	4.95	17908.0	United Kingdom
555	536412	22327	ROUND SNACK BOXES SET OF 4 SKULLS	1	2010-12-01 11:49:00	2.95	17920.0	United Kingdom
...	...	...	...	...	...	...	...	...
541675	581538	22068	BLACK PIRATE TREASURE CHEST	1	2011-12-09 11:34:00	0.39	14446.0	United Kingdom
541689	581538	23318	BOX OF 6 MINI VINTAGE CRACKERS	1	2011-12-09 11:34:00	2.49	14446.0	United Kingdom
541692	581538	22992	REVOLVER WOODEN RULER	1	2011-12-09 11:34:00	1.95	14446.0	United Kingdom
541699	581538	22694	WICKER STAR	1	2011-12-09 11:34:00	2.10	14446.0	United Kingdom
541701	581538	23343	JUMBO BAG VINTAGE CHRISTMAS	1	2011-12-09 11:34:00	2.08	14446.0	United Kingdom

5268 rows × 8 columns

In [14]:
```
# There are a total of 5227 values which are duplicate values which need to be removed
```

In [15]:
```
retail_df = retail_df.drop_duplicates()
retail_df.shape
```

Out[15]:
```
(536641, 8)
```

c. Perform descriptive analytics on the given data.

In [16]:
```
retail_df.describe(datetime_is_numeric=True)
```

Out[16]:

	Quantity	InvoiceDate	UnitPrice	CustomerID
count	536641.000000	536641	536641.000000	401604.000000
mean	9.620029	2011-07-04 08:57:06.087421952	4.632656	15281.160818
min	-80995.000000	2010-12-01 08:26:00	-11062.060000	12346.000000
25%	1.000000	2011-03-28 10:52:00	1.250000	13939.000000
50%	3.000000	2011-07-19 14:04:00	2.080000	15145.000000
75%	10.000000	2011-10-18 17:05:00	4.130000	16784.000000
max	80995.000000	2011-12-09 12:50:00	38970.000000	18287.000000
std	219.130156	NaN	97.233118	1714.006089

In [17]:
```
retail_df.describe(include = ['O'])
```

Out[17]:

	InvoiceNo	StockCode	Description	Country
count	536641	536641	535187	536641
unique	25900	4070	4223	38
top	573585	85123A	WHITE HANGING HEART T-LIGHT HOLDER	United Kingdom
freq	1114	2301	2357	490300

Data Transformation

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

In [18]:
```
# a. Create month cohorts and analyze active customers for each cohort.
```

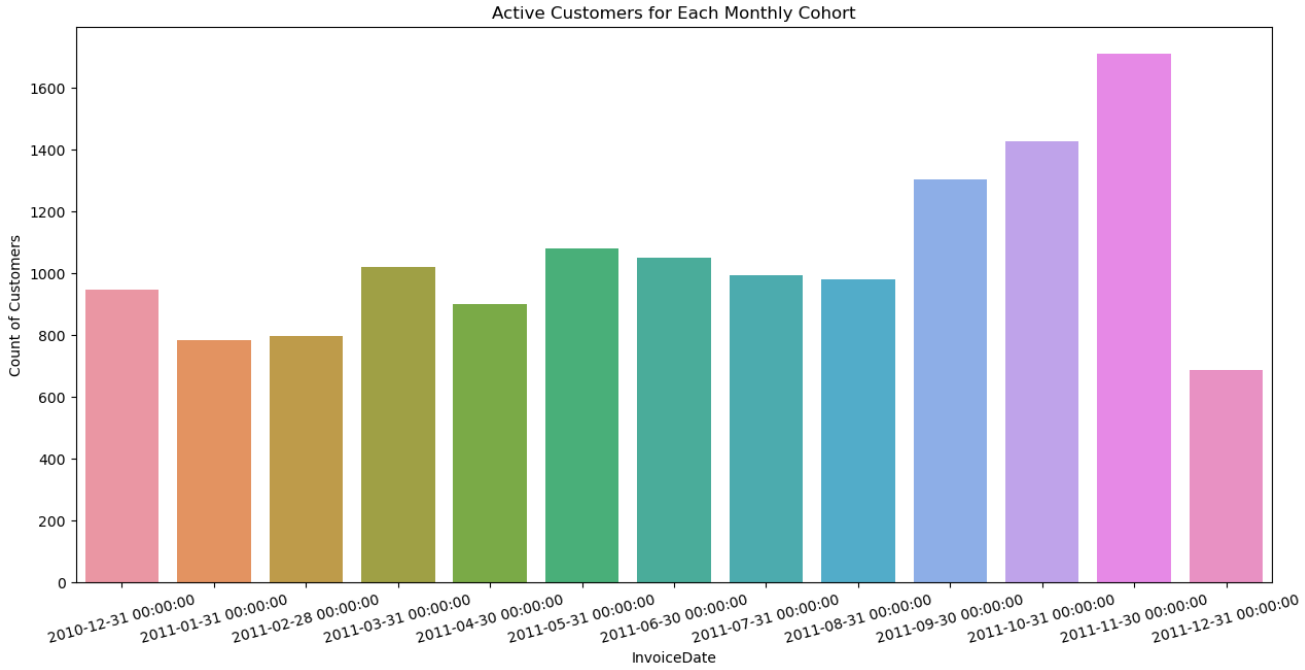
In [19]:
```
df = retail_df.groupby(pd.Grouper(key='InvoiceDate', axis=0, freq='M')).nunique()
```

In [20]:
```
monthwise_cohort = df['CustomerID']
```

In [21]:
```
monthwise_cohort
```

Out[21]:
```
InvoiceDate
2010-12-31     948
2011-01-31     783
2011-02-28     798
2011-03-31    1020
2011-04-30     899
2011-05-31    1079
2011-06-30    1051
2011-07-31     993
2011-08-31     980
2011-09-30    1302
2011-10-31    1425
2011-11-30    1711
2011-12-31     686
Freq: M, Name: CustomerID, dtype: int64
```

In [22]:
```
plt.figure(figsize=(15,7))
sns.barplot(x = monthwise_cohort.index, y = monthwise_cohort.values)
plt.xticks(rotation =15)
plt.title('Active Customers for Each Monthly Cohort')
plt.ylabel('Count of Customers')
plt.show()
```



In [23]:
```
# The Most number of active customers are from 2011.09-2011-11
```

In [24]:
```
active_customers_monthly_cohort = retail_df.groupby(pd.Grouper(key="InvoiceDate", freq="M"))['Description'].nunique()

print("Monthly count of unique products purchased by active customers:")
print(active_customers_monthly_cohort)

```

```
Monthly count of unique products purchased by active customers:
InvoiceDate
2010-12-31    2760
2011-01-31    2566
2011-02-28    2372
2011-03-31    2506
2011-04-30    2456
2011-05-31    2480
2011-06-30    2639
2011-07-31    2693
2011-08-31    2609
2011-09-30    2771
2011-10-31    2929
2011-11-30    2963
2011-12-31    2467
Freq: M, Name: Description, dtype: int64
```

b. Analyze the retention rate of customers

In [25]:
```
monthwise_cohort - monthwise_cohort.shift(1)
```

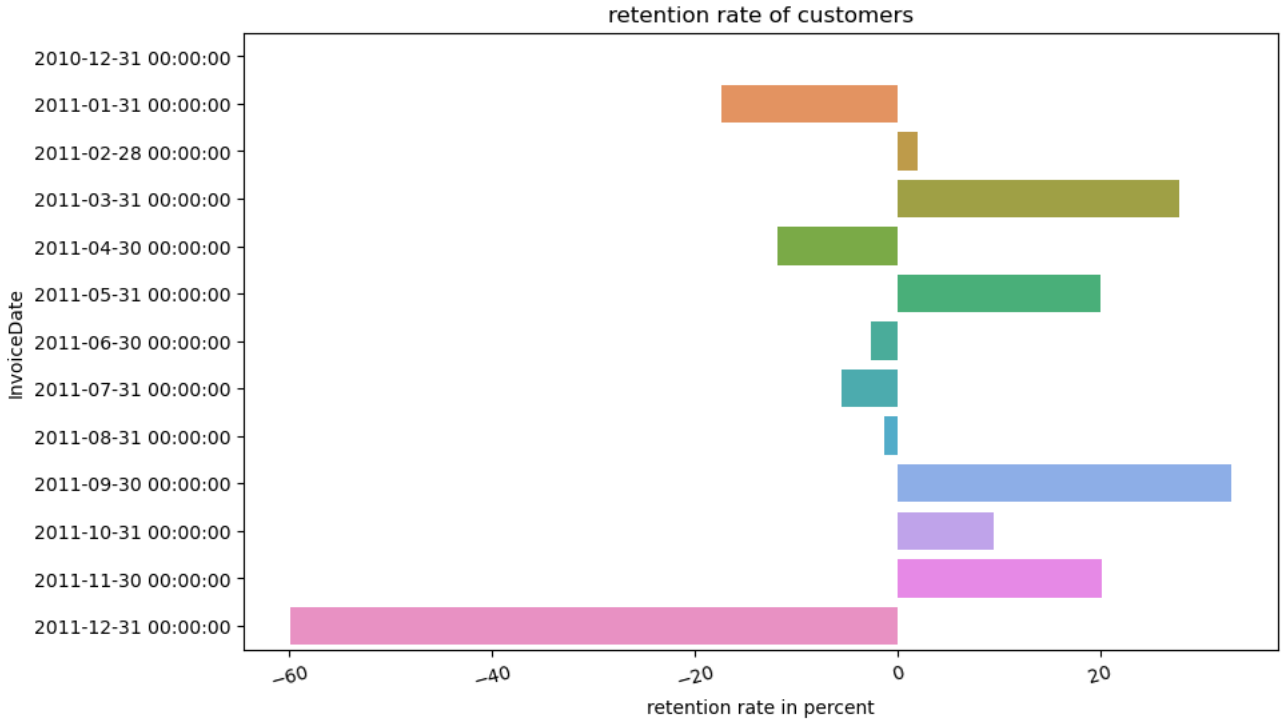
Out[25]:
```
InvoiceDate
2010-12-31       NaN
2011-01-31    -165.0
2011-02-28      15.0
2011-03-31     222.0
2011-04-30    -121.0
2011-05-31     180.0
2011-06-30     -28.0
2011-07-31     -58.0
2011-08-31     -13.0
2011-09-30     322.0
2011-10-31     123.0
2011-11-30     286.0
2011-12-31   -1025.0
Freq: M, Name: CustomerID, dtype: float64
```

In [26]:
```
# Negative results indicate a decrease in the count of unique products purchased by active customers, whilst positive values
```

In [27]:
```
retention_rate = round((monthwise_cohort.pct_change())*100,2)
retention_rate
```

Out[27]:
```
InvoiceDate
2010-12-31      NaN
2011-01-31   -17.41
2011-02-28     1.92
2011-03-31    27.82
2011-04-30   -11.86
2011-05-31    20.02
2011-06-30    -2.59
2011-07-31    -5.52
2011-08-31    -1.31
2011-09-30    32.86
2011-10-31     9.45
2011-11-30    20.07
2011-12-31   -59.91
Freq: M, Name: CustomerID, dtype: float64
```

In [28]:
```
plt.figure(figsize=(10,6))
sns.barplot(x = retention_rate.values, y = retention_rate.index)
plt.xticks(rotation = 15)
plt.title('retention rate of customers')
plt.xlabel('retention rate in percent')
plt.show()
```



Data Modeling

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

In [29]:
```
# RFM Analysis Technique
```

In [30]:
```
# First step: Create a variable TotalPrice, as follows:
retail_df['TotalPrice'] = retail_df['Quantity'] * retail_df['UnitPrice']
```

In [31]:
```
# Second step: Calculate the maximum date from the 'InvoiceDate' column of the Dataframe 'retail_df' and add a day to it usi
from datetime import timedelta
reference_date = (retail_df['InvoiceDate'].max()) + timedelta(days=1)
print('Reference Date: ', reference_date)
```

```
Reference Date:  2011-12-10 12:50:00
```

In [32]:
```
# Create a DataFrame, RFM to perform RFM Analysis using the lambda function for Recency, count for Frequency and sum for Mon
# DF 'retail_df' is grouped by 'CustomerID', the data is grouped based on unique customer IDs, creating separate groups for
RFM = retail_df.groupby(['CustomerID']).agg({'InvoiceDate': lambda x: (reference_date-x.max()),
                                             'InvoiceNo': 'count',
                                             'TotalPrice': 'sum'})
```

Calculate RFM metrics

In [33]:
```
# Rename the variables 'InvoiceDate', 'InvoiceNo', 'TotalPrice' as follows
RFM.rename(columns = {'InvoiceDate':'Recency', 'InvoiceNo':'Frequency', 'TotalPrice': 'MonetaryValue'}, inplace=True)
```

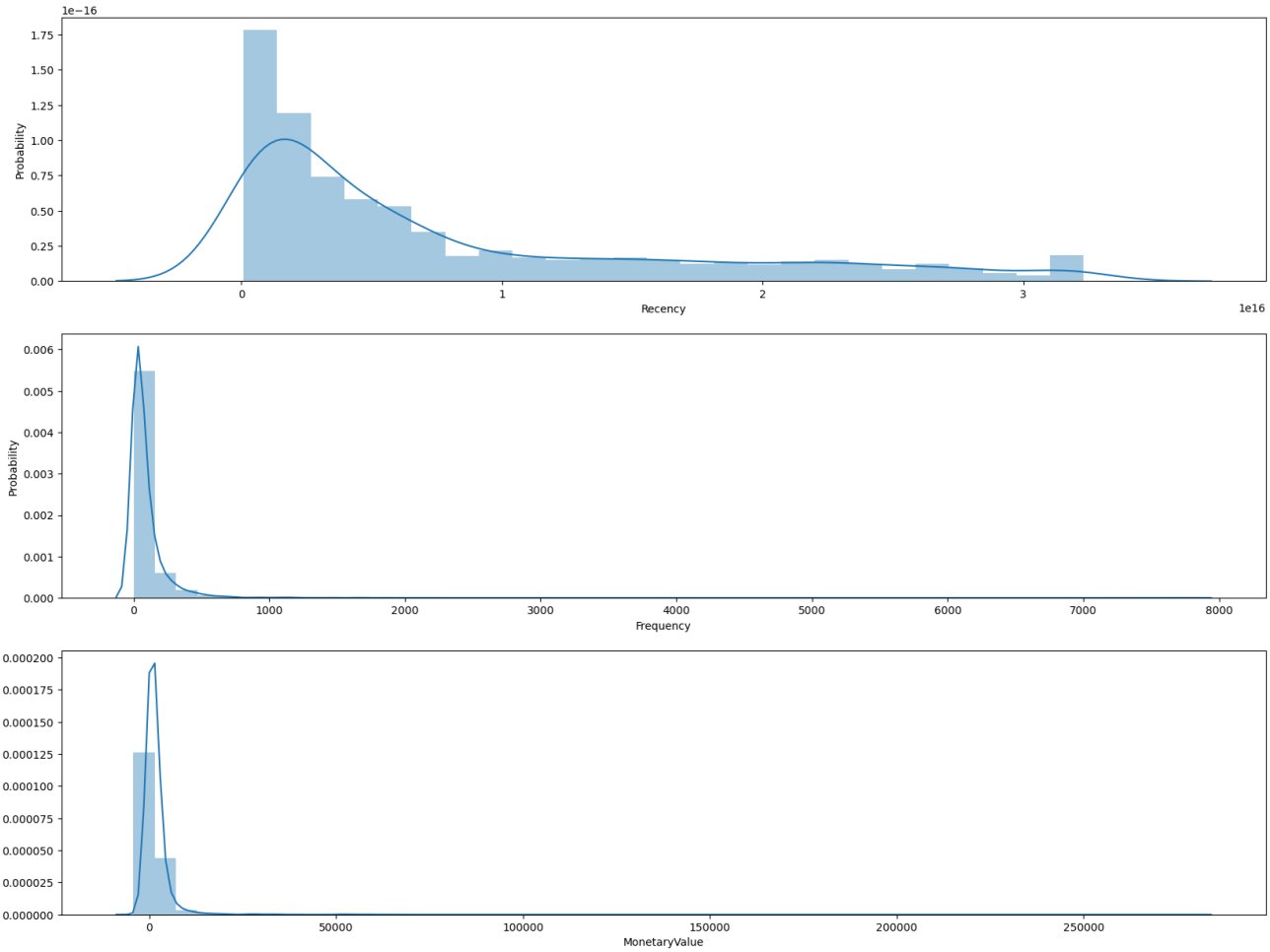
In [34]:
```
RFM
```

Out[34]:

CustomerID	Recency	Frequency	MonetaryValue
12346.0	326 days 02:33:00	2	0.00
12347.0	2 days 20:58:00	182	4310.00
12348.0	75 days 23:37:00	31	1797.24
12349.0	19 days 02:59:00	73	1757.55
12350.0	310 days 20:49:00	17	334.40
...	...	...	...
18280.0	278 days 02:58:00	10	180.60
18281.0	181 days 01:57:00	7	80.82
18282.0	8 days 01:07:00	13	176.60
18283.0	4 days 00:48:00	721	2045.53
18287.0	43 days 03:21:00	70	1837.28

4372 rows × 3 columns

In [35]:
```
# Plot the RFM Distribution
plt.figure(figsize=(20,15))

plt.subplot(3,1,1)
sns.distplot(RFM['Recency'])
plt.xlabel('Recency')
plt.ylabel('Probability')

plt.subplot(3,1,2)
sns.distplot(RFM['Frequency'])
plt.xlabel('Frequency')
plt.ylabel('Probability')

plt.subplot(3,1,3)
sns.distplot(RFM['MonetaryValue'])
plt.xlabel('MonetaryValue')
plt.ylabel('Probability')

plt.show()
```



In [36]:
```
# Distribution for RFM show as positively skewed
```

Build RFM Segments. Give Recency, Frequency, and Monetary Scores individually by dividing them into quartiles.

In [37]:
```
# Combine three ratings to get RFM segment ( as strongs).
```

In [38]:
```
# Creating labels for Recency, Frequency and Monetary by creating labels as " range objects" to categorize the data
rlabel = range(4,0,-1)
flabel = range(1,5)
mlabel = range(1,5)

# Using the qcut function, quartile function to come up with the ratings:
r = pd.qcut(RFM['Recency'], q=4, labels = rlabel)
f = pd.qcut(RFM['Frequency'], q=4, labels = flabel)
m = pd.qcut(RFM['MonetaryValue'], q=4, labels = mlabel)

RFM['R'] = r.values
RFM['F'] = f.values
RFM['M'] = m.values


```

In [39]:
```
RFM.head()
```

Out[39]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M
12346.0	326 days 02:33:00	2	0.00	1	1	1
12347.0	2 days 20:58:00	182	4310.00	4	4	4
12348.0	75 days 23:37:00	31	1797.24	2	2	4
12349.0	19 days 02:59:00	73	1757.55	3	3	4
12350.0	310 days 20:49:00	17	334.40	1	1	2

In [40]:
```
RFM['RFM_concat'] = RFM.apply(lambda x: str(x['R']) + str(x['F']) + str(x['M']), axis=1)
```

In [41]:
```
RFM.head()
```

Out[41]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_concat
12346.0	326 days 02:33:00	2	0.00	1	1	1	111
12347.0	2 days 20:58:00	182	4310.00	4	4	4	444
12348.0	75 days 23:37:00	31	1797.24	2	2	4	224
12349.0	19 days 02:59:00	73	1757.55	3	3	4	334
12350.0	310 days 20:49:00	17	334.40	1	1	2	112

In [42]:
```
RFM['RFM_score'] = RFM.apply(lambda x: x['R'] + x['F'] + x['M'], axis=1)
```

In [43]:
```
RFM.head()
```

Out[43]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_concat	RFM_score
12346.0	326 days 02:33:00	2	0.00	1	1	1	111	3
12347.0	2 days 20:58:00	182	4310.00	4	4	4	444	12
12348.0	75 days 23:37:00	31	1797.24	2	2	4	224	8
12349.0	19 days 02:59:00	73	1757.55	3	3	4	334	10
12350.0	310 days 20:49:00	17	334.40	1	1	2	112	4

In [44]:
```
import pandas as pd
from statistics import mode
```

In [45]:
```
RFM_1 = RFM['RFM_score']
```

In [46]:
```
mean_value = RFM_1.mean()
median_value = RFM_1.median()
print('Mean:', mean_value)
print('Median:', median_value)

try:
    mode_value = mode(RFM_1)
    print('Mode:', mode_value)
except StatisticsError:
    print('No unique mode found. The data has neither no mode or multiple modes.')
```

```
Mean: 7.494510521500457
Median: 7.0
Mode: 5
```

In [47]:
```
# High-Value Active = VIP (>10)
# High-Value Lapsed = Loyal(8-10)
# Low-Value Active = Potential(7)
# Low-Value Lapsed = Need Attention(4-6)
# Require Activation= (0-3)
```

In [48]:
```
# Based on the RFM score, we will create the segments
def segment(df):
    if df['RFM_score']>10:
        return 'VIP'
    elif df['RFM_score']>=8 and df['RFM_score']<=10:
        return 'Loyal'
    elif df['RFM_score']==7 and df['RFM_score']>6:
        return 'Potential'
    elif df['RFM_score']>=4 and df['RFM_score']<7:
        return 'Need Attention'
    else:
        return 'Require Activation'


```

In [49]:
```
RFM['RFM_segment'] = RFM.apply(segment, axis=1)
```

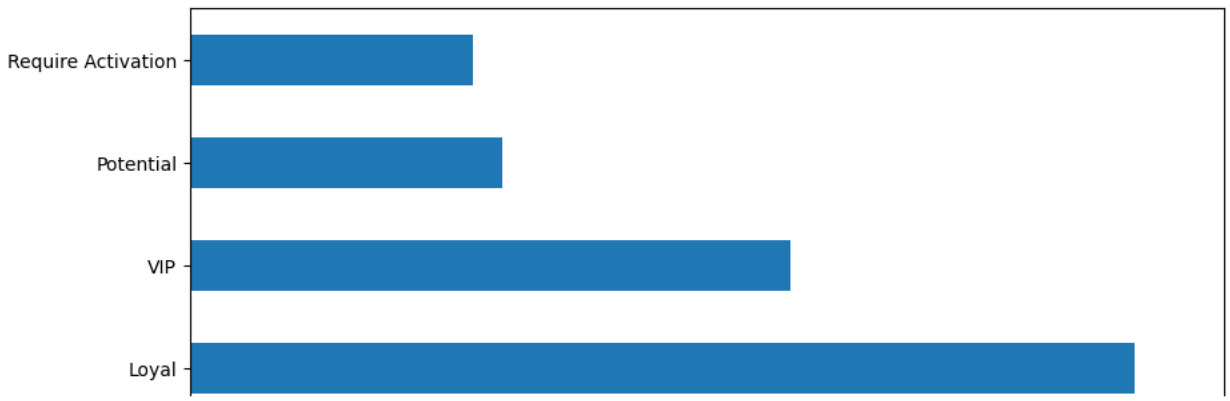
In [50]:
```
RFM.head()
```

Out[50]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_concat	RFM_score	RFM_segment
12346.0	326 days 02:33:00	2	0.00	1	1	1	111	3	Require Activation
12347.0	2 days 20:58:00	182	4310.00	4	4	4	444	12	VIP
12348.0	75 days 23:37:00	31	1797.24	2	2	4	224	8	Loyal
12349.0	19 days 02:59:00	73	1757.55	3	3	4	334	10	Loyal
12350.0	310 days 20:49:00	17	334.40	1	1	2	112	4	Need Attention

In [51]:
```
RFM['RFM_segment'].value_counts().plot(kind = 'barh', figsize=(10,5))
```

Out[51]: <AxesSubplot:>



In [52]:
```
# Investigate further customers falling under 'Need Attention' by looking into
# 1. The number of customers falling under the scores within the Need Attention range 4-6 in terms of RFM scores
# 2. With the said RFM_score range, look into the value counts under Recency, Frwuency and Monetary Value aggregated as mean
```

In [53]:
```
RFM_data = pd.DataFrame(RFM)
```

In [54]:
```
lower_bound = 4
upper_bound = 6
filtered_data = RFM_data[(RFM_data['RFM_score'] >= lower_bound) & (RFM_data['RFM_score'] <= upper_bound)]
counts = filtered_data['RFM_score'].value_counts()
counts

```

Out[54]:
```
5    523
6    464
4    391
Name: RFM_score, dtype: int64
```

In [55]:
```
counts_df = pd.DataFrame(counts).reset_index()
counts_df.columns = ['RFM_score', 'Count']
counts_df['Recency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Recency'].median())
counts_df['Frequency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Frequency'].median())
counts_df['MonetaryValue'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['MonetaryValue'].median())
counts_df
```

Out[55]:

	RFM_score	Count	Recency	Frequency	MonetaryValue
0	5	523	142 days 22:34:00	19.0	319.10
1	6	464	71 days 00:08:00	25.0	392.19
2	4	391	155 days 22:39:00	12.0	216.07

In [56]:
```
counts_df = pd.DataFrame(counts).reset_index()
counts_df.columns = ['RFM_score', 'Count']
counts_df['Recency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Recency'].mean())
counts_df['Frequency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Frequency'].mean())
counts_df['MonetaryValue'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['MonetaryValue'].mean())
counts_df
```

Out[56]:

	RFM_score	Count	Recency	Frequency	MonetaryValue
0	5	523	151 days 09:20:52.657743786	20.954111	342.147402
1	6	464	95 days 16:26:34.913793103	28.321121	496.861812
2	4	391	174 days 09:06:19.641943734	13.772379	227.307238

In [57]:
```
lower_bound = 0
upper_bound = 3
filtered_data = RFM_data[(RFM_data['RFM_score'] >= lower_bound) & (RFM_data['RFM_score'] <= upper_bound)]
counts = filtered_data['RFM_score'].value_counts()
counts

```

Out[57]:
```
3    396
Name: RFM_score, dtype: int64
```

In [58]:
```
counts_df = pd.DataFrame(counts).reset_index()
counts_df.columns = ['RFM_score', 'Count']
counts_df['Recency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Recency'].median())
counts_df['Frequency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Frequency'].median())
counts_df['MonetaryValue'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['MonetaryValue'].median())
counts_df
```

Out[58]:

	RFM_score	Count	Recency	Frequency	MonetaryValue
0	3	396	264 days 12:02:00	8.0	135.915

In [59]:
```
counts_df = pd.DataFrame(counts).reset_index()
counts_df.columns = ['RFM_score', 'Count']
counts_df['Recency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Recency'].mean())
counts_df['Frequency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Frequency'].mean())
counts_df['MonetaryValue'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['MonetaryValue'].mean())
counts_df
```

Out[59]:

	RFM_score	Count	Recency	Frequency	MonetaryValue
0	3	396	264 days 08:22:58.484848484	7.84596	108.518434

In [60]:
```
lower_bound = 8
upper_bound = 10
filtered_data = RFM_data[(RFM_data['RFM_score'] >= lower_bound) & (RFM_data['RFM_score'] <= upper_bound)]
counts = filtered_data['RFM_score'].value_counts()
counts
```

Out[60]:
```
8     468
10    438
9     415
Name: RFM_score, dtype: int64
```

In [61]:
```
counts_df = pd.DataFrame(counts).reset_index()
counts_df.columns = ['RFM_score', 'Count']
counts_df['Recency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Recency'].median())
counts_df['Frequency'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['Frequency'].median())
counts_df['MonetaryValue'] = counts_df['RFM_score'].map(RFM_data.groupby('RFM_score')['MonetaryValue'].median())
counts_df
```

Out[61]:

	RFM_score	Count	Recency	Frequency	MonetaryValue
0	8	468	50 days 22:44:30	49.5	768.720
1	10	438	23 days 02:05:00	97.0	1413.655
2	9	415	29 days 17:01:00	67.0	1039.300

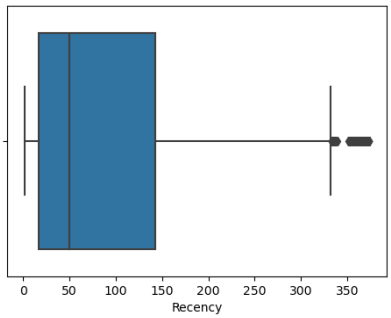
In [62]:
```
# Customers classifed as 'Need Attention' have not ordered or spent much in the past ( non-Frequent buyer), and not within t
```

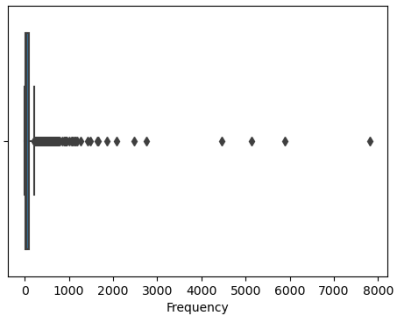
Data Modeling:

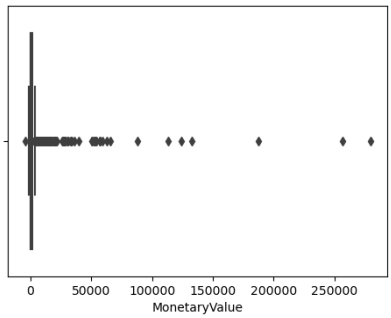
```
1. Create cluster using k-means clustering algorithm
a. Prepare the data for the algorithm. If the data us asmmetrically distributed, manage the skewness with appropriate
transformation.
Standardize the data.
```

In [63]:
```
# Convert Recency from days and time format to integer
RFM['Recency'] = pd.to_timedelta(RFM['Recency']).dt.days
```

In [64]:
```
# Outlier Detection
plt.figure(figsize=(19,4))
plt.subplot(1,3,1)
sns.boxplot(RFM['Recency'])

plt.subplot(1,3,2)
sns.boxplot(RFM['Frequency'])

plt.subplot(1,3,3)
sns.boxplot(RFM['MonetaryValue'])

plt.show()
```







In [65]:
```
Q1_recency = RFM['Recency'].quantile(0.25)
Q1_frequency = RFM['Frequency'].quantile(0.25)
Q1_monetaryvalue = RFM['MonetaryValue'].quantile(0.25)

Q3_recency = RFM['Recency'].quantile(0.75)
Q3_frequency= RFM['Frequency'].quantile(0.75)
Q3_monetaryvalue = RFM['MonetaryValue'].quantile(0.75)

IQR_recency = Q3_recency - Q1_recency
IQR_frequency = Q3_frequency - Q1_frequency
IQR_monetaryvalue = Q3_monetaryvalue - Q1_monetaryvalue

print('Recency Q1:', Q1_recency)
print('Recency Q3:', Q3_recency)
print('Recency IQR:', IQR_recency)

print('Frequency Q1:', Q1_frequency)
print('Frequency Q3:', Q3_frequency)
print('Frequency IQR:', IQR_frequency)

print('MonetaryValue Q1:', Q1_monetaryvalue)
print('MonetaryValue Q3:', Q3_monetaryvalue)
print('MonetaryValue IQR:', IQR_monetaryvalue)
```

```
Recency Q1: 17.0
Recency Q3: 143.0
Recency IQR: 126.0
Frequency Q1: 17.0
Frequency Q3: 99.25
Frequency IQR: 82.25
MonetaryValue Q1: 291.795
MonetaryValue Q3: 1608.335
MonetaryValue IQR: 1316.54
```

In [66]:
```
# Outliers are predominant in Frequency and Monetary Value, outliers will be dropped by creating a new cluster dataframe, se
```

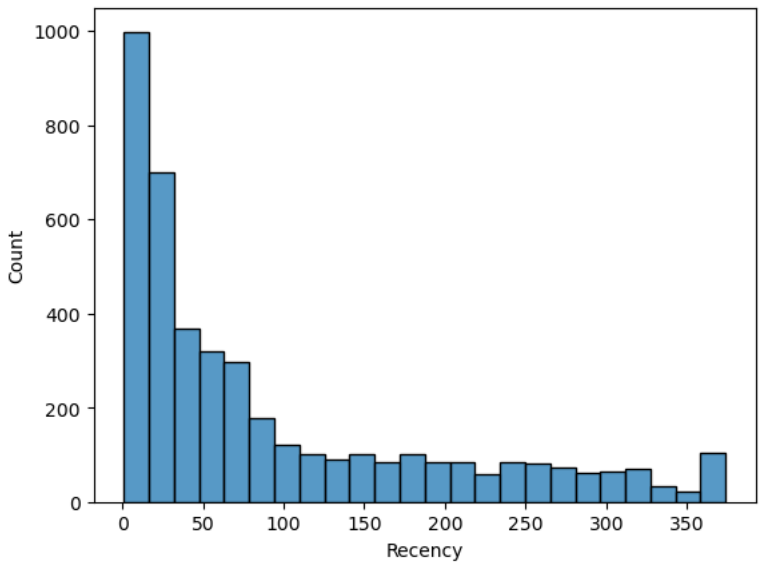
In [67]:
```
RFM_clust = RFM[(RFM['Frequency']<500) & (RFM['MonetaryValue']<50000)]
```

In [68]:
```
RFM_clust.shape
```

Out[68]: (4283, 9)

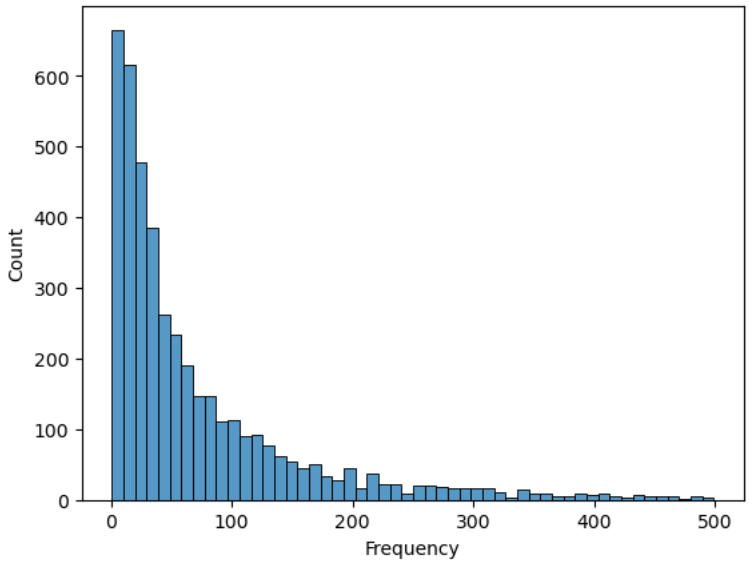
In [69]:
```
# Determine the skewness of the data in each variable to determine whether to apply data transformation
sns.histplot(RFM_clust['Recency'])
```

Out[69]: <AxesSubplot:xlabel='Recency', ylabel='Count'>



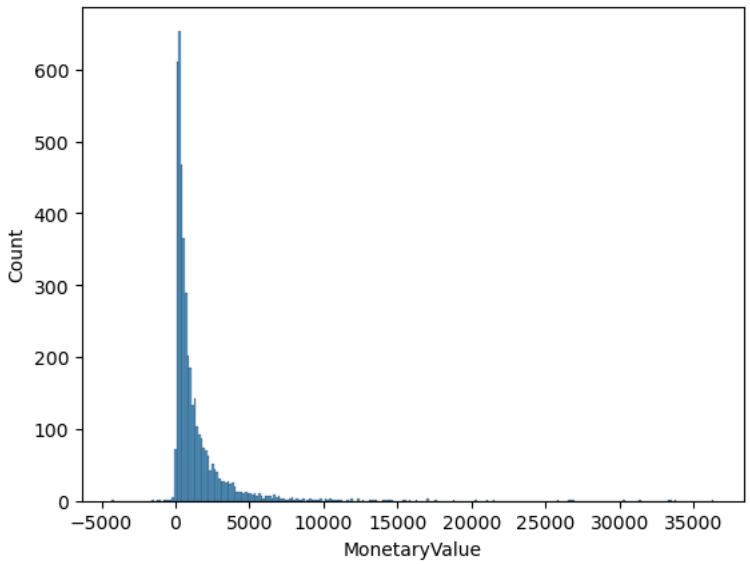
In [70]:
```
1 sns.histplot(RFM_clust['Frequency'])
```

Out[70]: <AxesSubplot:xlabel='Frequency', ylabel='Count'>



In [71]:
```
1 sns.histplot(RFM_clust['MonetaryValue'])
```

Out[71]: <AxesSubplot:xlabel='MonetaryValue', ylabel='Count'>



In [72]:
```
1 constant = 1
2 transform_rfm = pd.DataFrame()
3 transform_rfm['Recency'] = np.log(RFM_clust['Recency'])
4 transform_rfm['Frequency'] = np.log(RFM_clust['Frequency'])
5 transform_rfm['MonetaryValue'] = np.log(RFM_clust['MonetaryValue'] + constant)
```

In [73]:
```
1 # It is important to standardize the features thru Standardscaler prior to performing a distance-based K-Means Algorithms
2 from sklearn.preprocessing import StandardScaler
```

In [74]:
```
1 sc = StandardScaler()
```

In [75]:
```
1 scaled_rfm = sc.fit_transform(transform_rfm)
```

In [76]:
```
scaled_rfm =pd.DataFrame(scaled_rfm, columns = ['Recency', 'Frequency', 'MonetaryValue'])
scaled_rfm.head()
```

Out[76]:

	Recency	Frequency	MonetaryValue
0	1.397441	-2.290336	-5.256490
1	-2.141630	1.246242	1.523169
2	0.376515	-0.141478	0.814849
3	-0.577462	0.530008	0.796769
4	1.362476	-0.612494	-0.545501

b. Decide the optimum number of cluster to be formed

In [77]:
```
from sklearn.cluster import KMeans
```

In [78]:
```
km = KMeans(n_clusters = 4, init = 'k-means++', max_iter = 500, random_state=10)
```

In [79]:
```
has_nan = np.isnan(scaled_rfm).any()
has_inf = np.isinf(scaled_rfm).any()
```

In [80]:
```
has_nan
```

Out[80]:
```
Recency          False
Frequency        False
MonetaryValue     True
dtype: bool
```

In [81]:
```
has_inf
```

Out[81]:
```
Recency          False
Frequency        False
MonetaryValue    False
dtype: bool
```

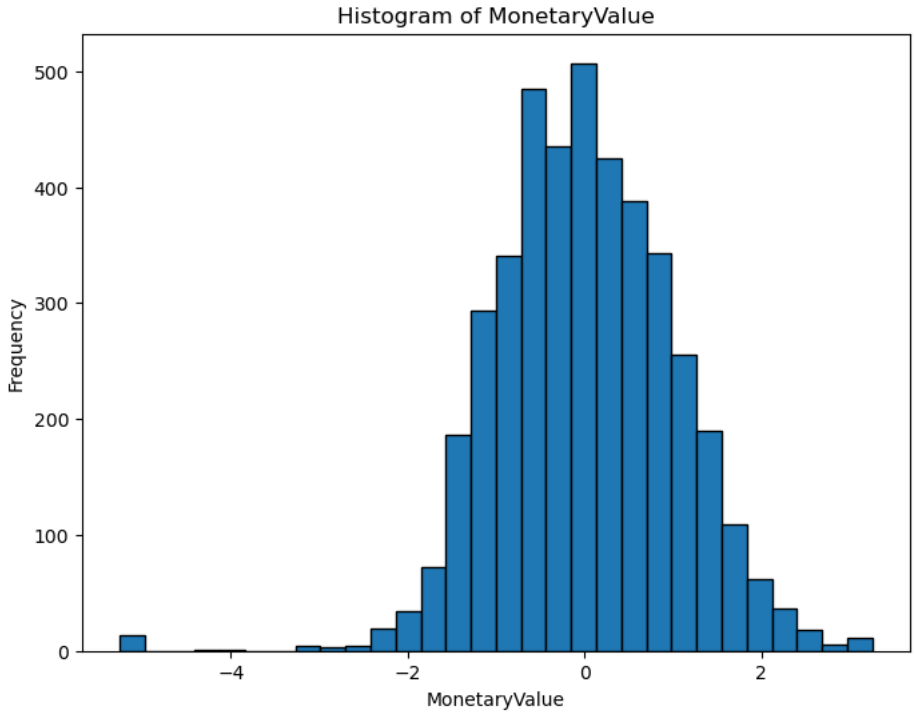
In [82]:
```
monetary_column = 'MonetaryValue'
num_nan_values = scaled_rfm[monetary_column].isna().sum()

print(f"Number of NaN values in '(monetary_column)' : {num_nan_values}")

```

```
Number of NaN values in '(monetary_column)' : 41
```

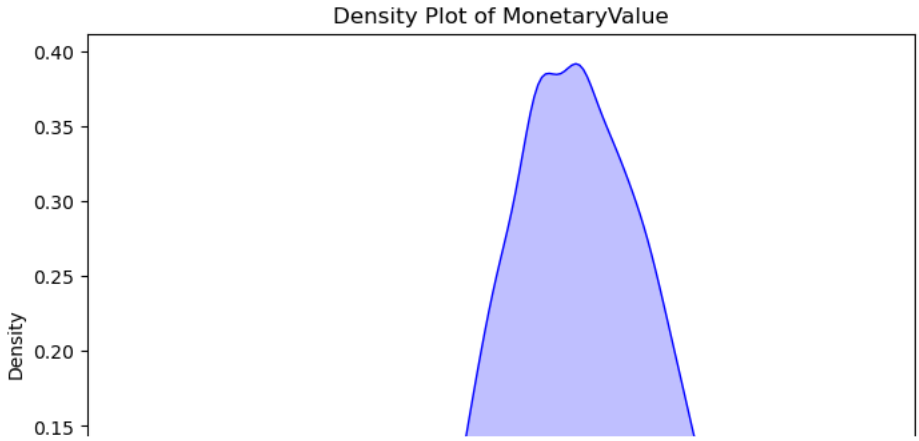
In [83]:
```
scaled_rfm.shape
```

Out[83]:
```
(4283, 3)
```

In [84]:
```
plt.figure(figsize=(8,6))
plt.hist(scaled_rfm[monetary_column], bins=30, edgecolor='black')
plt.xlabel('MonetaryValue')
plt.ylabel('Frequency')
plt.title('Histogram of MonetaryValue')
plt.show()
```



In [85]:
```
plt.figure(figsize=(8,6))
sns.kdeplot(scaled_rfm[monetary_column], fill=True, color='blue')
plt.xlabel('MonetaryValue')
plt.ylabel('Density')
plt.title('Density Plot of MonetaryValue')
plt.show()
```



In [86]:
```
nan_rows = scaled_rfm[scaled_rfm[monetary_column].isna()]
nan_rows
```

Out[86]:

	Recency	Frequency	MonetaryValue
123	1.422557	-2.833773	NaN
125	1.342006	-2.833773	NaN
210	1.475952	-1.972446	NaN
262	1.464436	-2.290336	NaN
409	1.477853	-2.290336	NaN
460	1.342006	-2.833773	NaN
608	0.829742	-2.833773	NaN
976	1.342006	-0.660024	NaN
989	1.395307	-1.746899	NaN
1090	1.464436	-2.833773	NaN
1184	1.489151	-1.571951	NaN

In [87]:
```
# Imputing the values using median
median_monetary_value = scaled_rfm[monetary_column].median()
```

In [88]:
```
scaled_rfm[monetary_column].fillna(median_monetary_value, inplace=True)
```

In [89]:
```
monetary_column = 'MonetaryValue'
num_nan_values = scaled_rfm[monetary_column].isna().sum()

print(f"Number of NaN values in '(monetary_column)' : {num_nan_values}")
```

```
Number of NaN values in '(monetary_column)' : 0
```

In [90]:
```
km = KMeans(n_clusters = 4, init = 'k-means++', max_iter = 500, random_state=10)
```

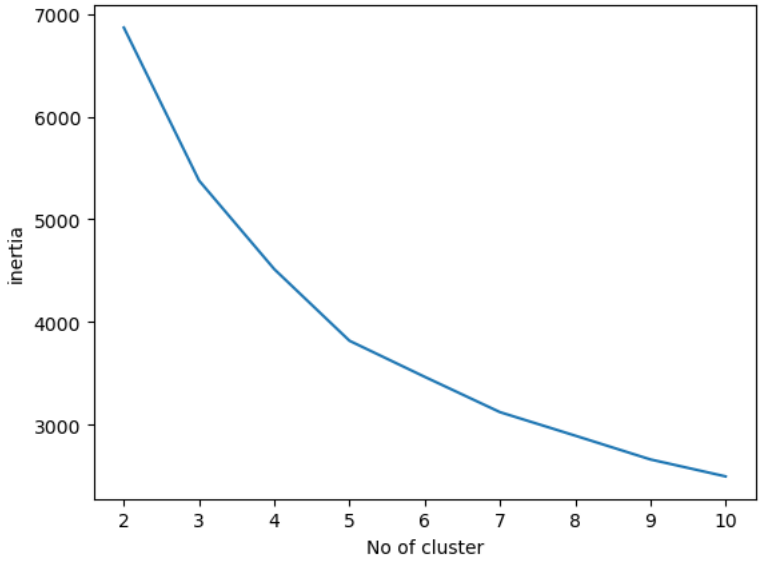
In [91]:
```
clusters = km.fit_predict(scaled_rfm)
```

In [92]:
```
from sklearn.metrics import silhouette_score
silhouette_score(scaled_rfm, clusters)
```

Out[92]:
```
0.29669198583337764
```

In [93]:
```
# Using the 'Elbow Method', we check if we can visually identify the optimal cluster
inertia_list = []
for i in range(2,11):
    km = KMeans(n_clusters = i, random_state =10)
    km.fit(scaled_rfm)
    inertia_list.append(km.inertia_)
```

In [94]:
```
# Plotting the Elbow Curve
plt.plot(range(2,11), inertia_list)
plt.xlabel('No of cluster')
plt.ylabel('inertia')
plt.show()
```



In [95]:
```
from mpl_toolkits.mplot3d import Axes3D
```

In [96]:
```
cluster_centers = km.cluster_centers_
labels = km.labels_
```

In [97]:
```
print('Shape of scaled_rfm:', scaled_rfm.shape)
print('Shape of cluster_centers:', cluster_centers.shape)
```

```
Shape of scaled_rfm: (4283, 3)
Shape of cluster_centers: (10, 3)
```

In [98]:
```
scaled_rfm.head()
```

Out[98]:

	Recency	Frequency	MonetaryValue
0	1.397441	-2.290336	-5.256490
1	-2.141630	1.246242	1.523169
2	0.376515	-0.141478	0.814849
3	-0.577462	0.530008	0.796769
4	1.362476	-0.612494	-0.545501

In [99]:
```
print('Data in cluster_centers:')
print(cluster_centers)
```

```
Data in cluster_centers:
[[-4.58071758e-01  1.14590399e+00  1.14923729e+00]
 [-3.20633710e-01 -8.17791327e-01 -8.94591186e-01]
 [ 9.30202180e-01 -2.43403917e-01 -4.21389199e-01]
 [-1.53695753e+00  2.16153140e-01  1.87921747e-01]
 [-1.86764020e+00  1.33996963e+00  1.50458356e+00]
 [ 1.11721027e+00 -1.07844906e+00 -1.14079210e+00]
 [ 6.00027842e-01 -2.11308836e+00 -2.42835540e+00]
 [-2.42357791e-01  2.56090713e-01  7.86005420e-04]
 [ 5.98195157e-01  4.81642831e-01  5.87472906e-01]
 [ 8.16175035e-01 -2.06710898e+00 -7.44690646e-02]]
```

In [100]:
```
print('Shape of labels:', labels.shape)
print('Number of data points in scaled_rfm:', scaled_rfm.shape[0])
```

```
Shape of labels: (4283,)
Number of data points in scaled_rfm: 4283
```

In [101]:
```
print('Unique labels in labels array:', np.unique(labels))
```

```
Unique labels in labels array: [0 1 2 3 4 5 6 7 8 9]
```

In [102]:
```
print('Data type of labels array:', labels.dtype)
```

```
Data type of labels array: int32
```

In [103]:
```
print(labels)
```

```
[6 4 8 ... 5 1 8]
```

fig = plt.figure() ax = fig.add_subplot(111, projection='3d')

ax.scatter(scaled_rfm[:, 0], scaled_rfm[:, 1], scaled_rfm[:, 2], c=labels, cmap='viridis', s=50, alpha=0.7) ax.scatter(cluster_centers[:, 0], cluster_centers[:, 1], cluster_centers[:, 2], c='red', marker='X', s=200)

ax.set_xlabel('Recency') ax.set_ylabel('Frequency') ax.set_zlabel('MonetaryValue') ax.set_title('KMeans Clustering in 3D') # Corrected line plt.show()

In [104]:
```
n_cluster = list(range(2,11))
inertia = pd.DataFrame(zip(n_cluster, inertia_list), columns = ['cluster', 'inertia'])
inertia
```

Out[104]:

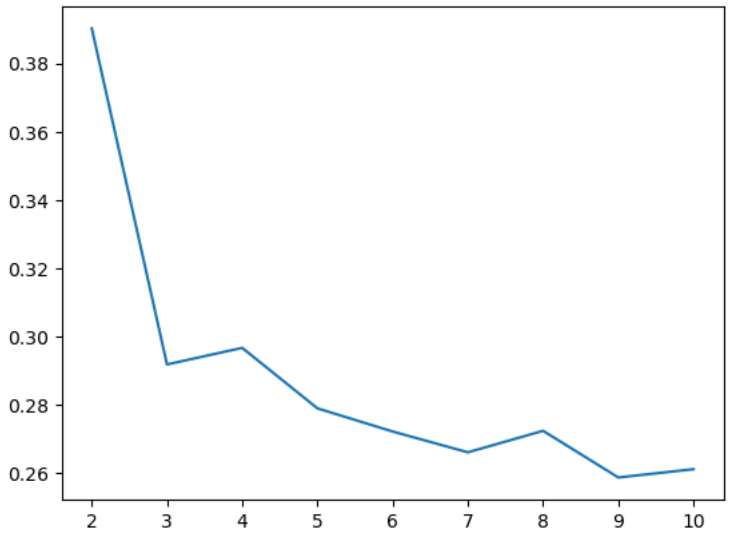
	cluster	inertia
0	2	6866.606144
1	3	5376.962734
2	4	4514.403613
3	5	3817.082765
4	6	3465.942557
5	7	3122.266536
6	8	2893.374997
7	9	2661.677503
8	10	2496.456828

In [105]:
```
# Although the inertia decreases as the number of cluster increases in the Elbow Method, the Elbow bend is not defined or cl
# Suggested to perform silhouette score Method, to better avoid chosing cluster which is not optimal and producing overfitti
```

In [106]:
```
from sklearn.metrics import silhouette_score

silhouette_scores = []  # Initialize an empty list to store silhouette scores

for i in range(2, 11):
    km = KMeans(n_clusters=i, max_iter=500, random_state=10)
    cluster = km.fit_predict(scaled_rfm)
    score = silhouette_score(scaled_rfm, cluster)  # Use the correct variable name "cluster"
    silhouette_scores.append(score)  # Append the score to the list "silhouette_scores"

```

In [107]:
```
plt.plot(range(2,11), silhouette_scores)
```

Out[107]: [<matplotlib.lines.Line2D at 0x1cda3366f70>]



In [108]:
```
# Defining optimal cluster at 2,the cluster where silhouette score is high, we run KMeans again:
km = KMeans(n_clusters = 2, init='k-means++', max_iter=500)
```

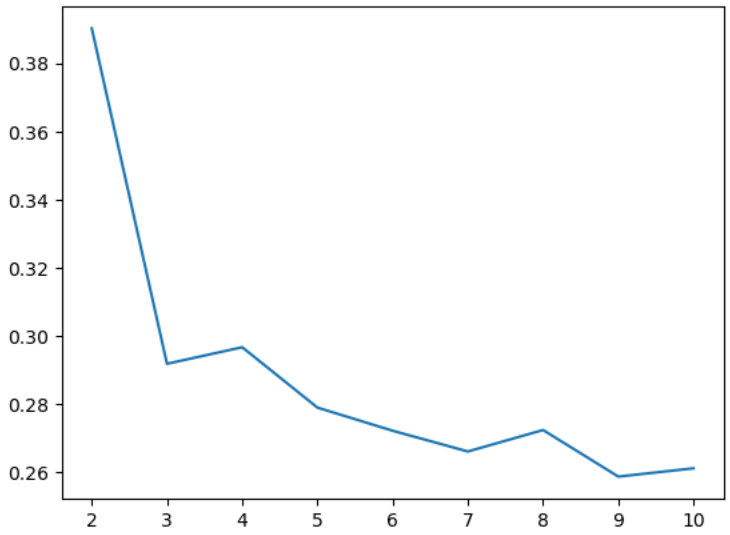
In [109]:
```
cluster = km.fit_predict(scaled_rfm)
```

In [110]:
```
silhouette_score(scaled_rfm, cluster)
```

Out[110]: 0.3902527136527532

In [111]:
```
plt.plot(range(2,11), silhouette_scores)
```

Out[111]:
```
[<matplotlib.lines.Line2D at 0x1cd978dcbe0>]
```



In [112]:
```
# Highest sillouette score in cluster 2, we designte n_clusters =2
```

In [113]:
```
scaled_rfm.head()
```

Out[113]:

	Recency	Frequency	MonetaryValue
0	1.397441	-2.290336	-5.256490
1	-2.141630	1.246242	1.523169
2	0.376515	-0.141478	0.814849
3	-0.577462	0.530008	0.796769
4	1.362476	-0.612494	-0.545501

In [114]:
```
RFM_clust['labeled'] = km.labels_
```

In [115]:
```
RFM_clust
```

Out[115]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_concat	RFM_score	RFM_segment	labeled
12346.0	326	2	0.00	1	1	1	111	3	Require Activation	1
12347.0	2	182	4310.00	4	4	4	444	12	VIP	0
12348.0	75	31	1797.24	2	2	4	224	8	Loyal	0
12349.0	19	73	1757.55	3	3	4	334	10	Loyal	0
12350.0	310	17	334.40	1	1	2	112	4	Need Attention	1
...	...	...	...	...	...	...	...	...	...	...
18278.0	74	9	173.90	2	1	1	211	4	Need Attention	1
18280.0	278	10	180.60	1	1	1	111	3	Require Activation	1
18281.0	181	7	80.82	1	1	1	111	3	Require Activation	1
18282.0	8	13	176.60	4	1	1	411	6	Need Attention	1
18287.0	43	70	1837.28	3	3	4	334	10	Loyal	0

4283 rows × 10 columns

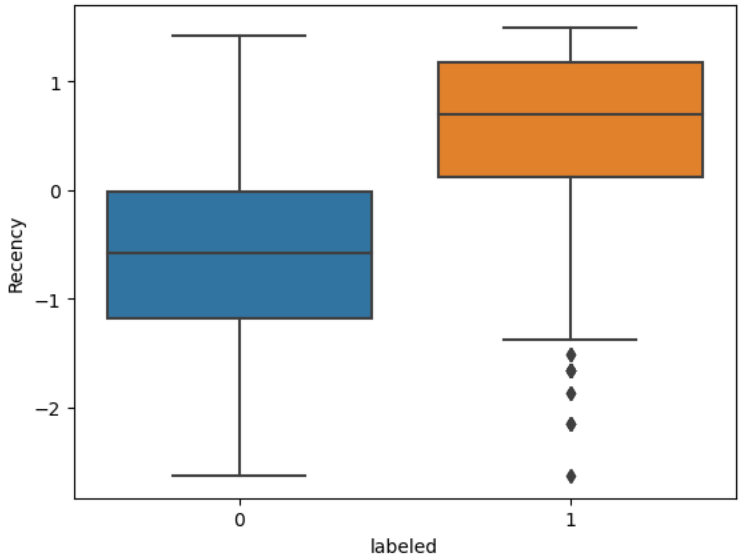
c. Analyze these cluster and comment on the results

In [116]:
```
# Check for outliers in the scaled dataframe after the first Kmeans clustering
```

In [117]:
```
scaled_rfm['labeled'] = km.labels_
```

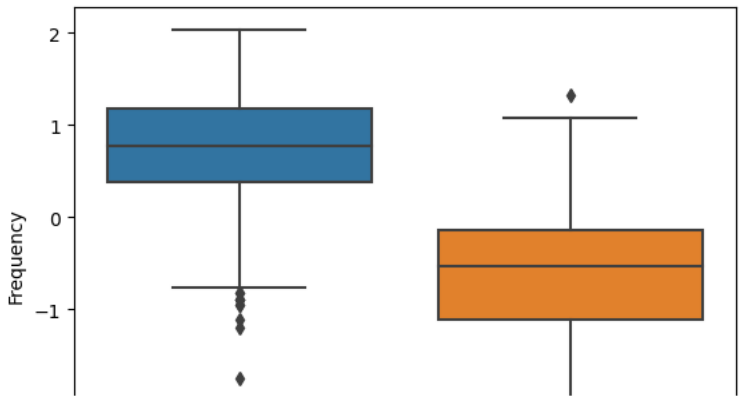
In [118]:
```
1 sns.boxplot(x='labeled', y='Recency', data=scaled_rfm)
```

Out[118]: <AxesSubplot:xlabel='labeled', ylabel='Recency'>

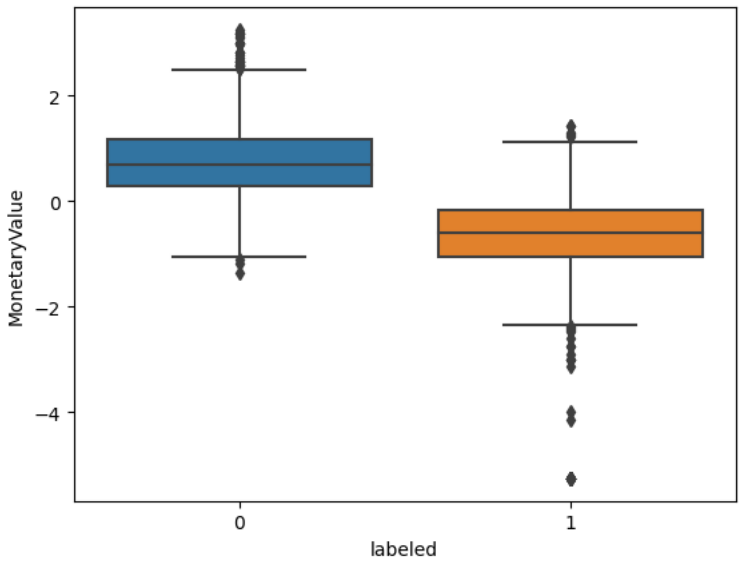


In [119]:
```
1 sns.boxplot(x='labeled', y='Frequency', data=scaled_rfm)
```

Out[119]: <AxesSubplot:xlabel='labeled', ylabel='Frequency'>



In [120]:
```
1 sns.boxplot(x='labeled', y='MonetaryValue', data=scaled_rfm);
```



In [121]:
```
# Outliers are still showing from the boxplot charts above, we will try the Clip or Winsorize Method to cap extreme values t
# Then, we refit the clustered scaled data
```

In [122]:
```
scaled_rfm_copy= scaled_rfm.copy()
```

In [123]:
```
scaled_rfm_copy.head()
```

Out[123]:

	Recency	Frequency	MonetaryValue	labeled
0	1.397441	-2.290336	-5.256490	1
1	-2.141630	1.246242	1.523169	0
2	0.376515	-0.141478	0.814849	0
3	-0.577462	0.530008	0.796769	0
4	1.362476	-0.612494	-0.545501	1

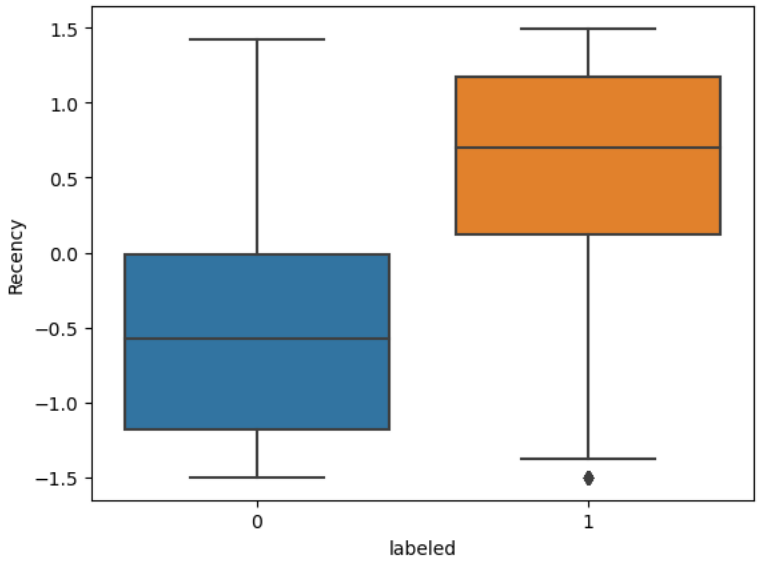
In [124]:
```
scaled_rfm.head()
```

Out[124]:

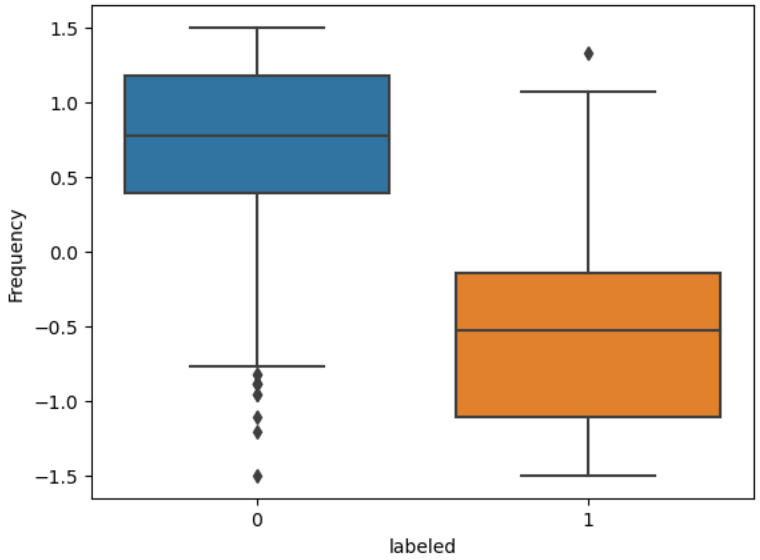
	Recency	Frequency	MonetaryValue	labeled
0	1.397441	-2.290336	-5.256490	1
1	-2.141630	1.246242	1.523169	0
2	0.376515	-0.141478	0.814849	0
3	-0.577462	0.530008	0.796769	0
4	1.362476	-0.612494	-0.545501	1

In [125]:
```
import numpy as np

# Assuming you have a DataFrame named df with the columns to be winsorized (e.g., 'Recency', 'Frequency', 'MonetaryValue')

# Set upper and lower thresholds for winsorization
upper_threshold = 1.5
lower_threshold = -1.5

# Perform winsorization using numpy.clip
scaled_rfm_copy['Recency'] = np.clip(scaled_rfm_copy['Recency'], lower_threshold, upper_threshold)
scaled_rfm_copy['Frequency'] = np.clip(scaled_rfm_copy['Frequency'], lower_threshold, upper_threshold)
scaled_rfm_copy['MonetaryValue'] = np.clip(scaled_rfm_copy['MonetaryValue'], lower_threshold, upper_threshold)
```

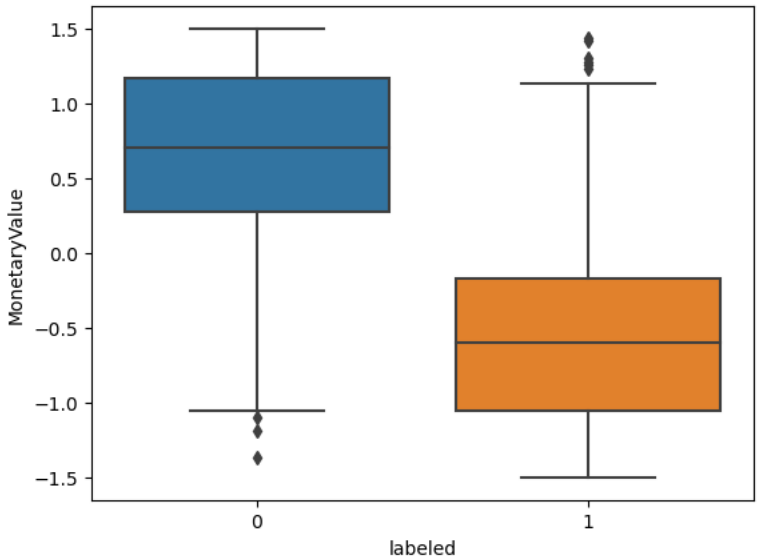
In [126]:
```
sns.boxplot(x='labeled', y='Recency', data=scaled_rfm_copy);
```



In [127]:
```
sns.boxplot(x='labeled', y='Frequency', data=scaled_rfm_copy);
```



In [128]:
```
sns.boxplot(x='labeled', y='MonetaryValue', data=scaled_rfm_copy);
```



In [129]:
```
# After winsorsizing the scaled data, outliers were minimalized for all Features, data points are better distributed
# Next step, is to re-cluster, re-fit and recompute the silhouette score
```

In [130]:
```
km_1 = KMeans(n_clusters = 2, init='k-means++', max_iter=500)
```

In [131]:
```
cluster_1 = km_1.fit_predict(scaled_rfm_copy)
```

In [132]:
```
silhouette_score(scaled_rfm_copy, cluster_1)
```

Out[132]: 0.46979261168445463

In [133]:
```
RFM_clust['labeled'] = km_1.labels_
```

In [134]:
```
RFM_clust
```

Out[134]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_concat	RFM_score	RFM_segment	labeled
12346.0	326	2	0.00	1	1	1	111	3	Require Activation	0
12347.0	2	182	4310.00	4	4	4	444	12	VIP	1
12348.0	75	31	1797.24	2	2	4	224	8	Loyal	1
12349.0	19	73	1757.55	3	3	4	334	10	Loyal	1
12350.0	310	17	334.40	1	1	2	112	4	Need Attention	0
...	...	...	...	...	...	...	...	...	...	...
18278.0	74	9	173.90	2	1	1	211	4	Need Attention	0
18280.0	278	10	180.60	1	1	1	111	3	Require Activation	0
18281.0	181	7	80.82	1	1	1	111	3	Require Activation	0
18282.0	8	13	176.60	4	1	1	411	6	Need Attention	0
18287.0	43	70	1837.28	3	3	4	334	10	Loyal	1

4283 rows × 10 columns

In [135]:
```
tbl = pd.crosstab(columns = RFM_clust.labeled, index = RFM_clust.RFM_segment)
tbl
```

Out[135]:

RFM_segment	0	1
Loyal	134	1186
Need Attention	1376	2
Potential	386	51
Require Activation	396	0
VIP	0	752

In [136]:
```
# High-Value Active = VIP (>10)   : Highly engaged customers who have purchased most recently, most often and generated the h
# High-Value Lapsed = Loyal(8-10) : Customers who have generated high revenue, but may have recently purchased less and less
# Low-Value Active = Potential(7) : Customers who do not generate high revenue, but who purchases often, and frequent
# Low-Value Lapsed = Need Attention(4-6): Customers who do not generate high revenue, but who have not spent much in the pas
# Require Activation= (0-3):  Inactive, customers who generate low revenue, and have not purchased recetnly and for a long t
```

In [147]:
```
RFM_clust['RFM_segment'] = RFM_clust['RFM_segment'].astype('category')
loyal_customers = RFM_clust[(RFM_clust['labeled'] == 0) & (RFM_clust['RFM_segment'] == 'Loyal')]

print('Loyal Customers (Cluster Label 0):')
print(loyal_customers)
```

```
Loyal Customers (Cluster Label 0):
            Recency  Frequency  MonetaryValue  R  F  M RFM_concat  RFM_score  \
CustomerID
12377.0         315         77        1628.12  1  3  4        134          8
12434.0          85         54         765.19  2  3  3        233          8
12452.0          17         23         428.57  4  2  2        422          8
12453.0         134         43         707.09  2  3  3        233          8
12701.0          80         53         797.71  2  3  3        233          8
...             ...        ...            ... .. .. ..        ...        ...
18112.0          12         26         352.69  4  2  2        422          8
18135.0          29         20         681.91  3  2  3        323          8
18228.0          50         32         769.20  3  2  3        323          8
18248.0         114         49         783.02  2  3  3        233          8
18265.0          72         46         801.51  2  3  3        233          8

           RFM_segment  labeled
CustomerID
12377.0          Loyal        0
12434.0          Loyal        0
12452.0          Loyal        0
12453.0          Loyal        0
12701.0          Loyal        0
...                ...      ...
18112.0          Loyal        0
18135.0          Loyal        0
18228.0          Loyal        0
18248.0          Loyal        0
18265.0          Loyal        0

[134 rows x 10 columns]
```

In [138]:
```
ased less

nd better understand your lapsed customers. Did they lapse because of a poor experience, seasonal products, or an isolated event?
```

In [148]:
```
RFM_clust['RFM_segment'] = RFM_clust['RFM_segment'].astype('category')
Potential_customers = RFM_clust[(RFM_clust['labeled'] == 1) & (RFM_clust['RFM_segment'] == 'Potential')]

print('Potential Customers (Cluster Label 1):')
print(Potential_customers)
```

```
13959.0          79         90         594.04  2  3  2         232          7
13989.0          66         41        1266.08  2  2  3         223          7
14020.0          17         39         642.80  3  2  2         322          7
14126.0           8         16         643.63  4  1  2         412          7
14204.0           2         39         150.61  4  2  1         421          7
14267.0         151         66        1279.09  1  3  3         133          7
14578.0           3         24         168.63  4  2  1         421          7
14584.0         169         99        1020.06  1  3  3         133          7
14618.0           9         17         538.59  4  1  2         412          7
14631.0          51         41        1006.98  2  2  3         223          7
14658.0           5         29         247.15  4  2  1         421          7
14740.0         191         93        1423.21  1  3  3         133          7
14758.0         145         71        1484.06  1  3  3         133          7
14970.0          68         34        1592.06  2  2  3         223          7
15097.0           4         25         248.08  4  2  1         421          7
15146.0         165         95        1313.66  1  3  3         133          7
15265.0         151         82        1506.12  1  3  3         133          7
15299.0          67         12        3787.09  2  1  4         214          7
15339.0          66         88         557.57  2  3  2         232          7
```

In [140]:
```
# Based on the list of 51 Potential Customers Clustered 1:
# As defined, they fall under Customers who do not generate high revenue, but who purchases often, and frequent
# There is quite a number from the list who generated high revenue based on high M scores, perhaps they have purchased more
```

In [149]:
```
RFM_clust['RFM_segment'] = RFM_clust['RFM_segment'].astype('category')
Needs_Attention_customers = RFM_clust[(RFM_clust['labeled'] == 1) & (RFM_clust['RFM_segment'] == 'Need Attention')]

print('Needs_Attention_customers (Cluster Label 1):')
print(Needs_Attention_customers)
```

```
Needs_Attention_customers (Cluster Label 1):
            Recency  Frequency  MonetaryValue  R  F  M RFM_concat  RFM_score  \
CustomerID
15749.0         235         15       21535.90  1  1  4        114          6
16313.0           3         16         274.15  4  1  1        411          6

               RFM_segment  labeled
CustomerID
15749.0     Need Attention        1
16313.0     Need Attention        1
```

In [142]:
```
# There are two customers under Need Attention who were clustered at 1
# AS defined, they fall under Customers who do not generate high revenue, but who have not spent much in the past and have n
# One of the customers have either purchased expensive products but have been inactive for 235 days
# The other have low MV, but have purchased the past 3 days.
# These are the factors to consider for the two customers
```

In [143]:
```
# Compute the percentage across each row (axis=1)
tbl_percentage = tbl.apply(lambda row: row / row.sum() * 100, axis=1)

# Print the resulting DataFrame with percentages
print(tbl_percentage)
```

```
labeled                     0          1
RFM_segment
Loyal               10.151515   89.848485
Need Attention      99.854862    0.145138
Potential           88.329519   11.670481
Require Activation 100.000000    0.000000
VIP                  0.000000  100.000000
```

In [144]:
```
# Based on the percentages of clustering on 0 and 1, the company may focus its customer engagement activites more on the thr
# Potential, Need Attention and Require Activation
# For the 10% under Loyal and 11.67% under Potential, whether to neglect or investigate further depends on stakeholder decis

```

Recency Q1: 17.0 Recency Q3: 143.0 Recency IQR: 126.0 Frequency Q1: 17.0 Frequency Q3: 99.25 Frequency IQR: 82.25 MonetaryValue Q1: 291.795 MonetaryValue Q3: 1608.335 MonetaryValue IQR: 1316.54

Project Task: Week 4

```
1. Create a dashboard in tableau by choosing appropriate chart types and metrics
useful for the business. The dashboard must entail the following:
a. Country-wise analysis to demonstrate average spend. Use a bar chart to show the monthly  figures
b. Bar graph of top 15 products which are mostly ordered by the users to show the number of products sold
c. Bar graph to show the count of orders vs. hours throughout the day
d. Plot the distribution of RFM values using histogram and frequency charts
e. Plot error (cost) vs. number of clusters selected
f. Visualize to compare the RFM values of the clusters using heatmap
```

```
# Converting the data frame into the excel to make dashboard in the tableau
retail_df.to_excel('master_data.xlsx', sheet_name='master_data', index=False)
RFM_clust.to_excel('rfm_data.xlsx', sheet_name='rfm_data', index=False)
inertia.to_excel('inertia_data.xlsx', sheet_name='inertia', index=False)
```

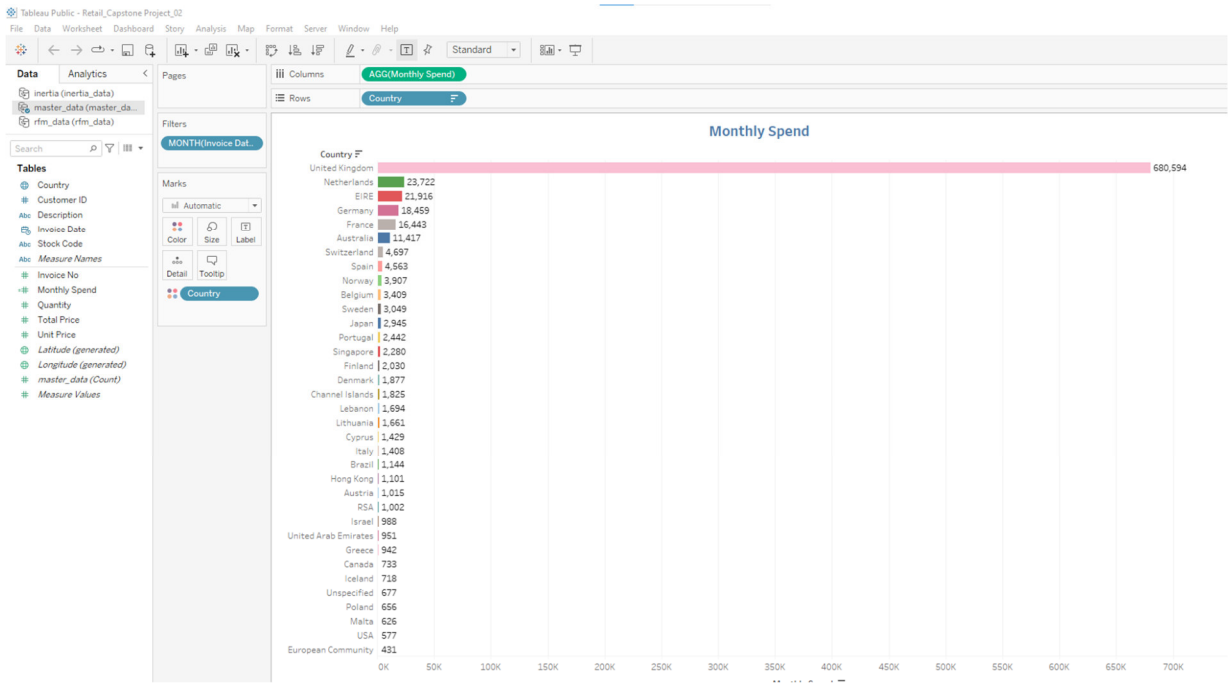
In [145]:
```
RFM_clust.to_excel('rfm_data_2.xlsx', sheet_name='rfm_data_1', index=False)
```

Project Task: Week 4

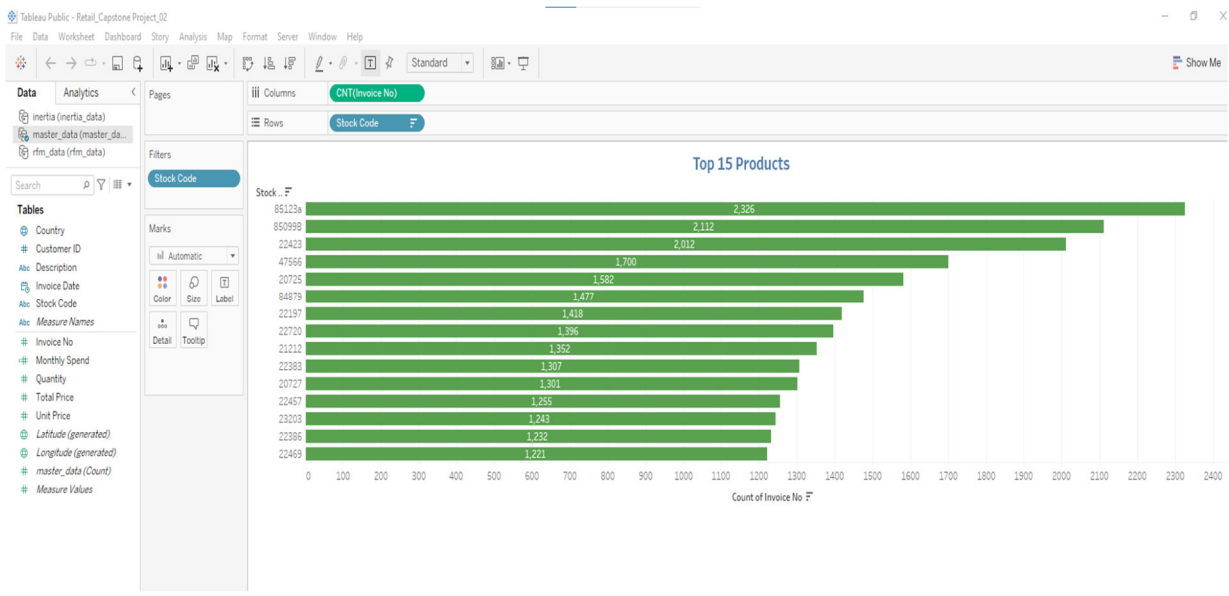
Data Reporting:

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

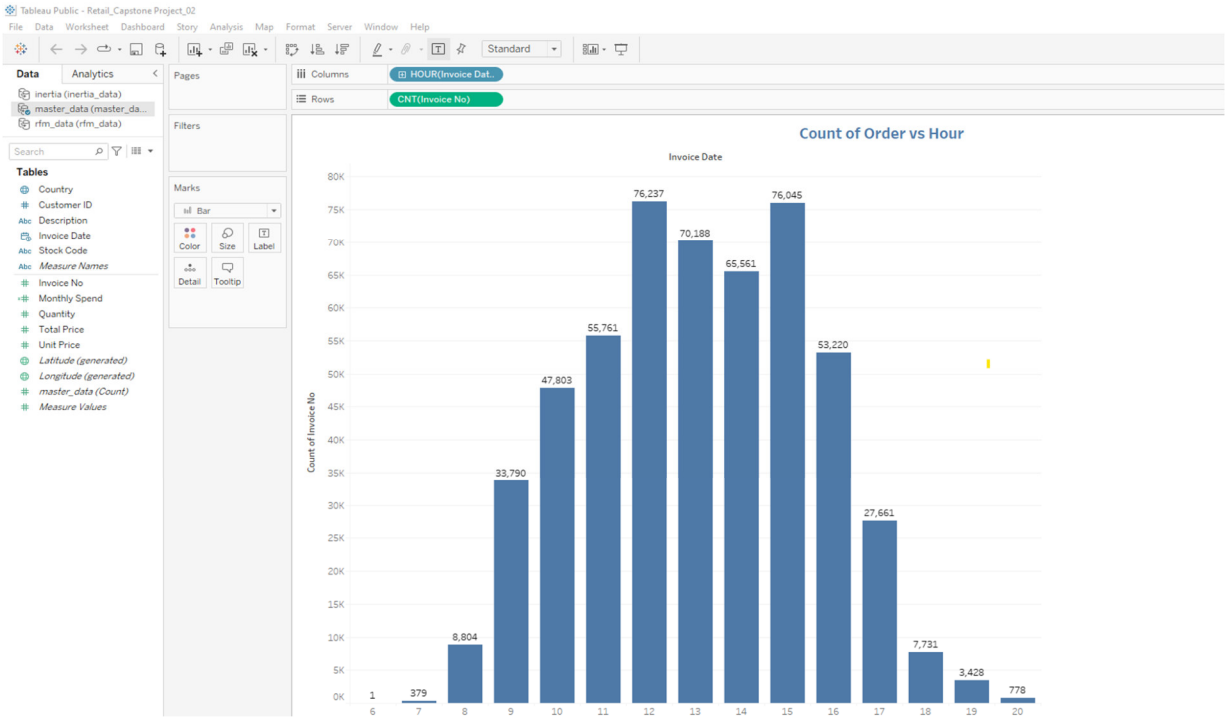
a. Country-wise analysis to demonstrate average spend. Use a bar chart to show the monthly figures



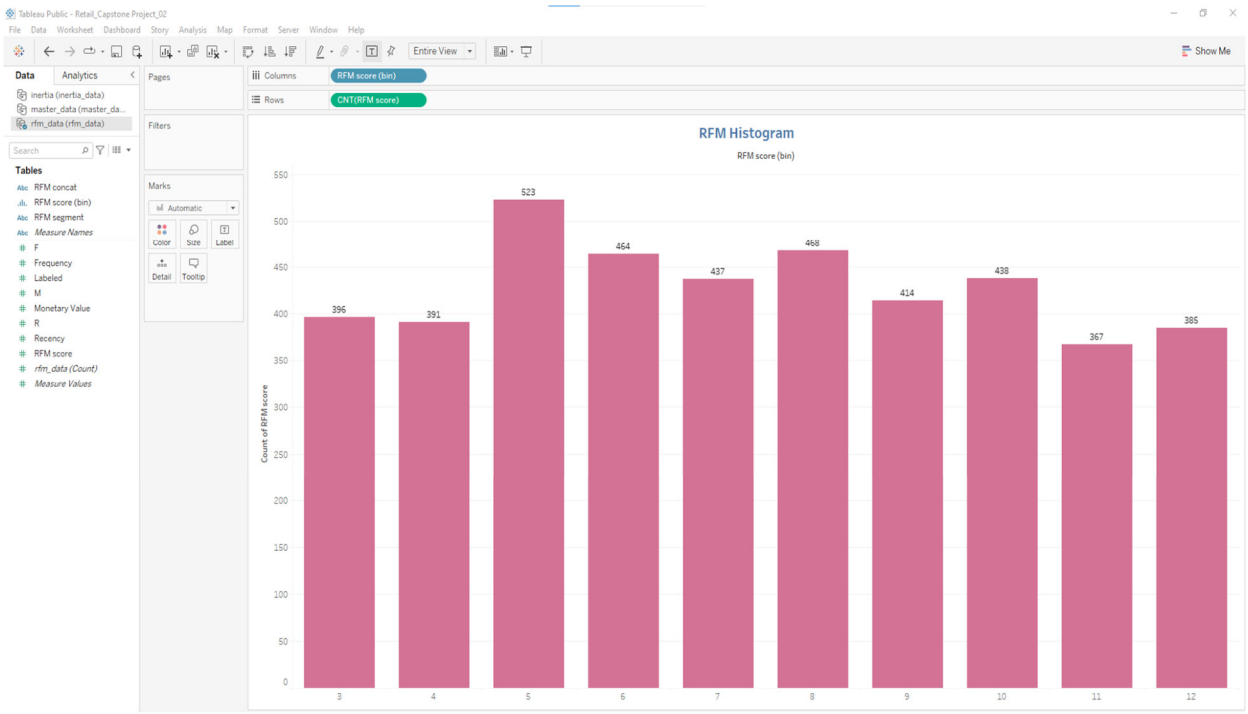
b. Bar graph of top 15 products which are mostly ordered by the users to show the number of products sold



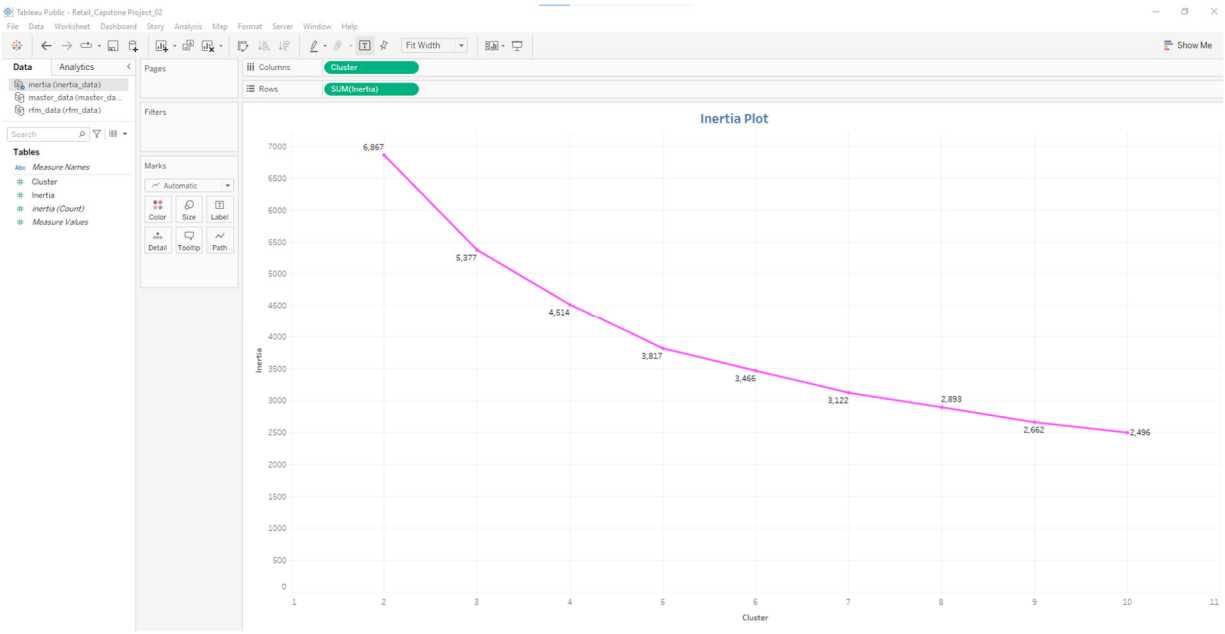
c. Bar graph to show the count of orders vs. hours throughout the day



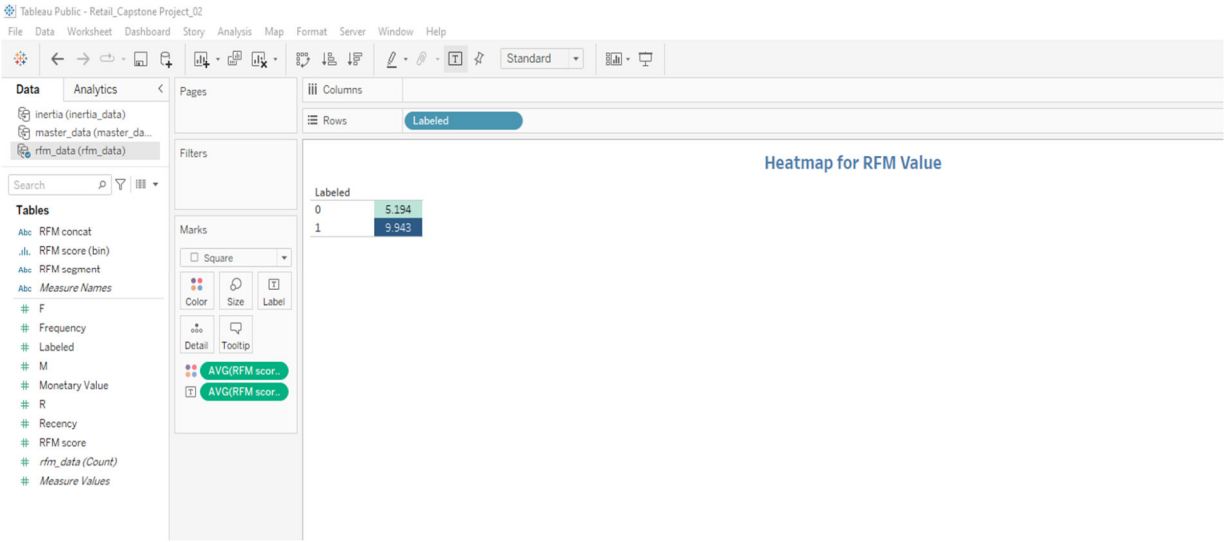
d. Plot the distribution of RFM values using histogram and frequency charts



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



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



FINAL DASHBOARD