Retail Project-03

March 26, 2022

0.1 Name: Sunil Pradhan

0.2 DOMAIN-RETAIL

0.3 OBJECTIVE-

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

• Customer segmentation is the practice of segregating the customer base into groups of individuals based on some common characteristics such as age, gender, interests, and spending habits

• Perform customer segmentation using RFM analysis. The resulting segments can be ordered from most valuable (highest recency, frequency, and value) to least valuable (lowest recency, frequency, and value).

0.4 Project Task: Week 1(Data cleaning and Data transformation)

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
df=pd.read_excel("Online Retail1.xlsx")
df.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 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
```

```
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

[3]:
```
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
```

[4]:
```
df.describe()
```

[4]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

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

[5]:
```
df.isna().sum()
```

[5]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

[6]:
```
round(df.isna().sum()/len(df)*100,2)
```

[6]:
```
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
```

0.5.1 Since, Description column is irrelevent to the model building.So, i deleted the entire column.

[7]:
```
df.drop("Description",axis=1,inplace=True)
df.head()
```

[7]:
```
  InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice  CustomerID  \
0    536365    85123A         6 2010-12-01 08:26:00       2.55     17850.0
1    536365     71053         6 2010-12-01 08:26:00       3.39     17850.0
2    536365    84406B         8 2010-12-01 08:26:00       2.75     17850.0
3    536365    84029G         6 2010-12-01 08:26:00       3.39     17850.0
4    536365    84029E         6 2010-12-01 08:26:00       3.39     17850.0

          Country
0  United Kingdom
1  United Kingdom
2  United Kingdom
3  United Kingdom
4  United Kingdom
```

[8]:
```
df.shape
```

[8]:
```
(541909, 7)
```

[9]:
```
df[df["CustomerID"].isna()]
```

[9]:
```
       InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice  \
622       536414     22139        56 2010-12-01 11:52:00       0.00
1443      536544     21773         1 2010-12-01 14:32:00       2.51
1444      536544     21774         2 2010-12-01 14:32:00       2.51
1445      536544     21786         4 2010-12-01 14:32:00       0.85
1446      536544     21787         2 2010-12-01 14:32:00       1.66
…          …        …       …                  …        …
541536    581498    85099B         5 2011-12-09 10:26:00       4.13
541537    581498    85099C         4 2011-12-09 10:26:00       4.13
```

```
541538    581498     85150         1 2011-12-09 10:26:00       4.96
541539    581498     85174         1 2011-12-09 10:26:00      10.79
541540    581498       DOT         1 2011-12-09 10:26:00    1714.17

        CustomerID         Country
622            NaN  United Kingdom
1443           NaN  United Kingdom
1444           NaN  United Kingdom
1445           NaN  United Kingdom
1446           NaN  United Kingdom
…             …              …
541536         NaN  United Kingdom
541537         NaN  United Kingdom
541538         NaN  United Kingdom
541539         NaN  United Kingdom
541540         NaN  United Kingdom

[135080 rows x 7 columns]
```

CustomerID is important feature of our analysis,since our analysis is centered around Customers only so we can not impute null values with mean/ median/ mode in this case. We will check possibility to fill null values in CustomerID column by looking up for InvoiceNo of the row having null CustomerID. If there are still any null values in CustomerID after this process then we will drop complete row having missing CustomerID.

[10]:
```
null_customerID=set(df[df["CustomerID"].isna()]["InvoiceNo"])
null_customerID
```

[10]:
```
{540673,
 540674,
 540675,
 540676,
 540677,
 540678,
 540679,
 'A563186',
 540681,
 540683,
 540684,
 540685,
 540693,
 540694,
 540695,
 540696,
 548886,
 'C539756',
```

540699,
548887,
548888,
548889,
548890,
548893,
548894,
548895,
548897,
565275,
548901,
565286,
565288,
573482,
573487,
565296,
565297,
573488,
573489,
557108,
573490,
565302,
565303,
557112,
565304,
565306,
565307,
565308,
573493,
565310,
573494,
573495,
573497,
565314,
573498,
573499,
573503,
573505,
565319,
548945,
557137,
557138,
557139,
557140,
557151,
565346,
573540,

```
540778,
573547,
548977,
573553,
'C572456',
548980,
565368,
565369,
548986,
565370,
565371,
548989,
565376,
548994,
565378,
548996,
548997,
548998,
548999,
549000,
549001,
549002,
549003,
549004,
549005,
549006,
565385,
549008,
549009,
549010,
549011,
565389,
540821,
565396,
573585,
565400,
573589,
573590,
573591,
540828,
573592,
573593,
549023,
540832,
540833,
573594,
573595,
```

540836,
549028,
549029,
549030,
549031,
549032,
549033,
549034,
549035,
549037,
549036,
549038,
540848,
549040,
549042,
549043,
557229,
557231,
573633,
557253,
557254,
565455,
573664,
573669,
573670,
573672,
578066,
549100,
549103,
'C566579',
578067,
549122,
557323,
549132,
549133,
549134,
549135,
549136,
549137,
549138,
549139,
549140,
549141,
549142,
549143,
549144,
549145,

549146,
549147,
549149,
549150,
549151,
'C537652',
573726,
549155,
549156,
549157,
549158,
549159,
549160,
549161,
565543,
549163,
549164,
549165,
549166,
540973,
549167,
540977,
549170,
549171,
540978,
549168,
540982,
549172,
549173,
549174,
549175,
540987,
549176,
549177,
549178,
549179,
549180,
549181,
557377,
540995,
541000,
557390,
565060,
'C547913',
557412,
557415,
557417,

```
565617,
549236,
573815,
573816,
549244,
572477,
549264,
549266,
549267,
549268,
549270,
549271,
557480,
557481,
557482,
557486,
549295,
541104,
557491,
557493,
557494,
557495,
557496,
557497,
573880,
573881,
'C541653',
557500,
541118,
557501,
557502,
573882,
549314,
573883,
573884,
573885,
541127,
541128,
541129,
541130,
541131,
541132,
541133,
549326,
549327,
549328,
549329,
```

```
549330,
549331,
549332,
579740,
549340,
573916,
549342,
549343,
549344,
549345,
549346,
549347,
549348,
549349,
549350,
549351,
573919,
573920,
573926,
549355,
'C544054',
549371,
565755,
549373,
549374,
549375,
549376,
565764,
565767,
'C574288',
573978,
573979,
557596,
557597,
'C563537',
557599,
557600,
573980,
573981,
541219,
573983,
541221,
565797,
565798,
573984,
573985,
557610,
```

```
573986,
573987,
557613,
573989,
573990,
573991,
549426,
549427,
557640,
557641,
557642,
557643,
557644,
549453,
549454,
549455,
549456,
549457,
549458,
549459,
549460,
549461,
549462,
549463,
549464,
549465,
549466,
'C553378',
565847,
565852,
549470,
573254,
573255,
573256,
557675,
573258,
549489,
573259,
'C547905',
549492,
549494,
549497,
549498,
574073,
549500,
574074,
549502,
```

```
574076,
549506,
549510,
'C564183',
549514,
574094,
549524,
549525,
574101,
549527,
549528,
549529,
549531,
'C541492',
565917,
549534,
565919,
549536,
549537,
'C572467',
565927,
565928,
574123,
'C558036',
565935,
565938,
574133,
557756,
557768,
'C561966',
557777,
549586,
557778,
557780,
557784,
574171,
574173,
574176,
541412,
541421,
541422,
541423,
541424,
553745,
557806,
553746,
'C570454',
```

```
574198,
'C572346',
557847,
557851,
581435,
549668,
541482,
566060,
541487,
566064,
566065,
566066,
566067,
549684,
541493,
541494,
541495,
541496,
541497,
566068,
566072,
581439,
574270,
549695,
541506,
541507,
541508,
'C564494',
541516,
541519,
549712,
549714,
574298,
'C537572',
574309,
566121,
549738,
'C557663',
566122,
557936,
557937,
557938,
'C572422',
'C572445',
541584,
566164,
574356,
```

```
541592,
541600,
541601,
566179,
541607,
'C579907',
'C546557',
'C537610',
'C544584',
566221,
558032,
558033,
558034,
566224,
541652,
549846,
549848,
549849,
574428,
558046,
558047,
574432,
574438,
558055,
558064,
558066,
558068,
541685,
541687,
541692,
576594,
541695,
541696,
541697,
541702,
549896,
'C544581',
558116,
558117,
558118,
558119,
558120,
'C570025',
566316,
549935,
566323,
566324,
```

```
566325,
574516,
566327,
574518,
574519,
574520,
549948,
549950,
549952,
574556,
574561,
541802,
'C554716',
574574,
541809,
541810,
541811,
574577,
574578,
574579,
574580,
574581,
574582,
574583,
550011,
550012,
541827,
541830,
'C544049',
558236,
558238,
558240,
558242,
558243,
558244,
558245,
558246,
541869,
541870,
541871,
566446,
566449,
'C579757',
541878,
541879,
541882,
558268,
```

```
'C574897',
566476,
566478,
566493,
558305,
558306,
558307,
558308,
558309,
550123,
558316,
558318,
558319,
550128,
558321,
'C567713',
550133,
558337,
'C542540',
558340,
558342,
558343,
558344,
550153,
558345,
558346,
566534,
541967,
541969,
541971,
574742,
541975,
541982,
558369,
558371,
579909,
574757,
541993,
558377,
558378,
558379,
566573,
566574,
541999,
566577,
558388,
'C570463',
```

```
550203,
550204,
550205,
550206,
550207,
550208,
550209,
550210,
550211,
550212,
550213,
550214,
550215,
566593,
566602,
566603,
566615,
574816,
574822,
550249,
566635,
558445,
558446,
558448,
558449,
558450,
558451,
558453,
558457,
558464,
550278,
574856,
558475,
558476,
558477,
558485,
576675,
542109,
542115,
'C546975',
542127,
566710,
542135,
542136,
542137,
558520,
566711,
```

```
566712,
574904,
574905,
574906,
574918,
542156,
560406,
560407,
574941,
566757,
558566,
574950,
558569,
566761,
566762,
566763,
566764,
566765,
566766,
560411,
566767,
566768,
574967,
566787,
'C558364',
578327,
542225,
558610,
558614,
542252,
542253,
542255,
'C556874',
550458,
'C570464',
550460,
'C544671',
550470,
550471,
558663,
575048,
550474,
575057,
558680,
558681,
558682,
558683,
```

```
566874,
550496,
550499,
550500,
550501,
550502,
558697,
575084,
573463,
'C566899',
558715,
558720,
'C553372',
558723,
558724,
'C540155',
558725,
542344,
558728,
558730,
558732,
550541,
550542,
566924,
558741,
558742,
558743,
542361,
542362,
542363,
542364,
542365,
575130,
542367,
542368,
575138,
566949,
'C573550',
566952,
542378,
542379,
542380,
542387,
558774,
542391,
542392,
558777,
```

542394,
575162,
542396,
566977,
566982,
566983,
575175,
566985,
575176,
575177,
542414,
542415,
542417,
542418,
558808,
558809,
558810,
558811,
550636,
550638,
550640,
550641,
567024,
558835,
558837,
573491,
558864,
558865,
558866,
567056,
567057,
558869,
567058,
567059,
567060,
567061,
558874,
567063,
567064,
542504,
542505,
558889,
542507,
542508,
542509,
542510,
558890,

542512,
542513,
542514,
542515,
542516,
558896,
542518,
575287,
558904,
542529,
558913,
542531,
542532,
558914,
558915,
542536,
567113,
542541,
542544,
542545,
542546,
542547,
542548,
550740,
542550,
542551,
542552,
542553,
542554,
542555,
542556,
542557,
542558,
542559,
542560,
542561,
542562,
542563,
542564,
542565,
542566,
542567,
550753,
542569,
542570,
542571,
542572,

```
542573,
542574,
542575,
542576,
542577,
550761,
542579,
542580,
542581,
542582,
567150,
567151,
567152,
'C576233',
542594,
550800,
575386,
542622,
550815,
542624,
542625,
567201,
567207,
567208,
542633,
'C549931',
'C553370',
559037,
559038,
559039,
559040,
559041,
559042,
559043,
'C546943',
542664,
559051,
559052,
'C553531',
559055,
550898,
575477,
550908,
'C537644',
567301,
571915,
550922,
```

```
550926,
550927,
550928,
550929,
550930,
550931,
575503,
575505,
575506,
575513,
575518,
550943,
550945,
550947,
550948,
550950,
550951,
550952,
550953,
550954,
559147,
567335,
567337,
550963,
550757,
559161,
550758,
550970,
559162,
559163,
559166,
542783,
542784,
550759,
550975,
550976,
550760,
550977,
550979,
'C547908',
550980,
550981,
542794,
550982,
550983,
550984,
550762,
```

```
550985,
550986,
550987,
567371,
550763,
554020,
551004,
551005,
551006,
551007,
551008,
551009,
551010,
551012,
551019,
551020,
575601,
575613,
575614,
575615,
542849,
575617,
567427,
575625,
542861,
542867,
'C537251',
542871,
542879,
542880,
575649,
542882,
542883,
542884,
542885,
567461,
'C545887',
542906,
575677,
559296,
573580,
559304,
575696,
559314,
567507,
551126,
542936,
```

```
575704,
575714,
575715,
575716,
575717,
559335,
575719,
559337,
559338,
575720,
575721,
575722,
567539,
542965,
575739,
551168,
575748,
551173,
551174,
559365,
551176,
551177,
559373,
559374,
559375,
559376,
559377,
559378,
559379,
573596,
542997,
542998,
542999,
543000,
573597,
573598,
559390,
551201,
543013,
543014,
559398,
543018,
…}
```

[11]:
```
df[df["InvoiceNo"].isin(null_customerID) & (~df['CustomerID'].isnull())]
```

[11]: Empty DataFrame
Columns: [InvoiceNo, StockCode, Quantity, InvoiceDate, UnitPrice, CustomerID, Country]
Index: []

0.5.2 We could not find any value to impute null values in CustomerID column,since all entries for a particular InvoiceNo have missing CustomerID if that particular InvoiceNo has null CustomerID in even one entry. So we will drop all rows having null values in CustomerID.

[12]:
```
df.dropna(inplace=True)
df.isna().sum()
print("\n")
print("The shape of dataset is",df.shape)
```

```
The shape of dataset is (406829, 7)
```

0.6 Remove duplicate data records

Since our data is transactional data and it has duplicate entries for InvoiceNo and CustomerID, we will drop only those rows which are completely duplicated.

[13]:
```
df.shape
```

[13]: (406829, 7)

[14]:
```
df.drop_duplicates()
df.shape
```

[14]: (406829, 7)

0.7 Perform descriptive analytics on the given data

[15]:
```
df["CustomerID"]=df["CustomerID"].astype(str)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406829 entries, 0 to 541908
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    406829 non-null  object
 1   StockCode    406829 non-null  object
 2   Quantity     406829 non-null  int64
 3   InvoiceDate  406829 non-null  datetime64[ns]
```

```
 4   UnitPrice    406829 non-null  float64
 5   CustomerID   406829 non-null  object
 6   Country      406829 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(1), object(4)
memory usage: 24.8+ MB
```

[16]:
```python
df.describe()
```

[16]:

	Quantity	UnitPrice
count	406829.000000	406829.000000
mean	12.061303	3.460471
std	248.693370	69.315162
min	-80995.000000	0.000000
25%	2.000000	1.250000
50%	5.000000	1.950000
75%	12.000000	3.750000
max	80995.000000	38970.000000

[17]:
```python
df.describe(datetime_is_numeric=True)
```

[17]:

	Quantity	InvoiceDate	UnitPrice
count	406829.000000	406829	406829.000000
mean	12.061303	2011-07-10 16:30:57.879207424	3.460471
min	-80995.000000	2010-12-01 08:26:00	0.000000
25%	2.000000	2011-04-06 15:02:00	1.250000
50%	5.000000	2011-07-31 11:48:00	1.950000
75%	12.000000	2011-10-20 13:06:00	3.750000
max	80995.000000	2011-12-09 12:50:00	38970.000000
std	248.693370	NaN	69.315162

1.Quantity: Average quantity of each product in transaction is 12.06. Also note that minimum value in Quantity column is negative. This implies that some customers had returned the product during our period of analysis.

2.InvoiceDate: Our data has transaction between 01-12-2010 to 09-12-2011

3.UnitPrice: Average price of each product in transactions is 3.46

[18]:
```python
df.describe(include="object")
```

[18]:

	InvoiceNo	StockCode	CustomerID	Country
count	406829	406829	406829	406829
unique	22190	3684	4372	37
top	576339	85123A	17841.0	United Kingdom
freq	542	2077	7983	361878

1.InvoiceNo: Total entries in preprocessed data are 4,06,829 but transactions are 22,190. Most number of entries (count of unique products) are in Invoice No. '576339' and is 542 nos.

2.StockCode: There are total 3684 unique products in our data and product with stock code '85123A' appears most frequently (2077 times) in our data.

3.CustomerID: There are 4372 unique customers in our final preprocessed data. Customer with ID '17841' appears most frequently in data (7983 times)

4.Country: Company has customers across 37 countries. Most entries are from United Kingdom in our dataset (361878)

0.7.1 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.

Create month cohorts and analyze active customers for each cohort.

[19]:
```
df.head()
```

[19]:
```
  InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice CustomerID  \
0    536365    85123A         6 2010-12-01 08:26:00       2.55    17850.0
1    536365     71053         6 2010-12-01 08:26:00       3.39    17850.0
2    536365    84406B         8 2010-12-01 08:26:00       2.75    17850.0
3    536365    84029G         6 2010-12-01 08:26:00       3.39    17850.0
4    536365    84029E         6 2010-12-01 08:26:00       3.39    17850.0

          Country
0  United Kingdom
1  United Kingdom
2  United Kingdom
3  United Kingdom
4  United Kingdom
```

[20]:
```
df["Year_month"]=df["InvoiceDate"].dt.to_period("M")
df.head()
```

[20]:
```
  InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice CustomerID  \
0    536365    85123A         6 2010-12-01 08:26:00       2.55    17850.0
1    536365     71053         6 2010-12-01 08:26:00       3.39    17850.0
2    536365    84406B         8 2010-12-01 08:26:00       2.75    17850.0
3    536365    84029G         6 2010-12-01 08:26:00       3.39    17850.0
4    536365    84029E         6 2010-12-01 08:26:00       3.39    17850.0

          Country Year_month
0  United Kingdom    2010-12
1  United Kingdom    2010-12
2  United Kingdom    2010-12
3  United Kingdom    2010-12
4  United Kingdom    2010-12
```

[21]:
```
df.groupby("Year_month")["CustomerID"].unique()
```

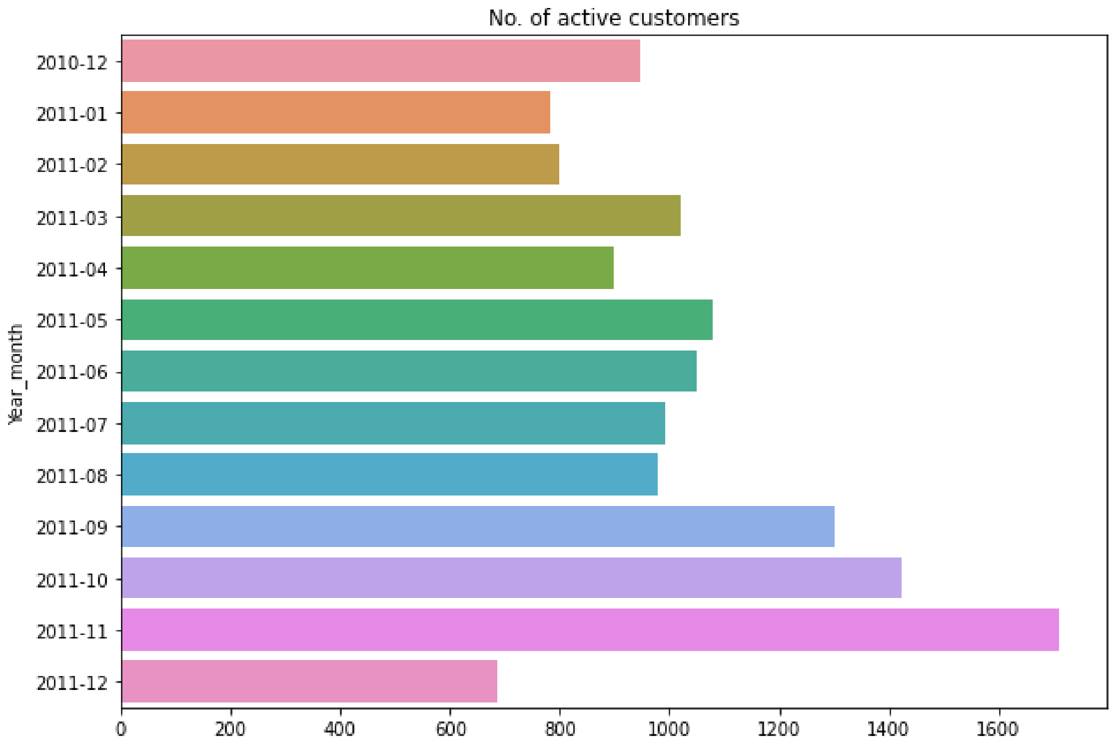
[21]:
```
Year_month
2010-12    [17850.0, 13047.0, 12583.0, 13748.0, 15100.0, …
2011-01    [13313.0, 18097.0, 16656.0, 16875.0, 13094.0, …
2011-02    [15240.0, 14911.0, 14496.0, 17147.0, 17675.0, …
2011-03    [14620.0, 14740.0, 13880.0, 16462.0, 17068.0, …
2011-04    [18161.0, 14886.0, 17613.0, 12523.0, 13694.0, …
2011-05    [15606.0, 14800.0, 16931.0, 15708.0, 14304.0, …
2011-06    [15643.0, 14842.0, 15124.0, 14646.0, 12423.0, …
2011-07    [16317.0, 13492.0, 14911.0, 17865.0, 17667.0, …
2011-08    [17941.0, 14947.0, 12921.0, 14060.0, 14239.0, …
2011-09    [13509.0, 13305.0, 16187.0, 17306.0, 12474.0, …
2011-10    [16353.0, 16591.0, 16923.0, 15038.0, 17811.0, …
2011-11    [17733.0, 17419.0, 13461.0, 13697.0, 14911.0, …
2011-12    [13853.0, 15197.0, 13644.0, 13310.0, 13468.0, …
Freq: M, Name: CustomerID, dtype: object
```

[22]:
```
Month_cohort=df.groupby("Year_month")["CustomerID"].nunique()
Month_cohort
```

[22]:
```
Year_month
2010-12     948
2011-01     783
2011-02     798
2011-03    1020
2011-04     899
2011-05    1079
2011-06    1051
2011-07     993
2011-08     980
2011-09    1302
2011-10    1425
2011-11    1711
2011-12     686
Freq: M, Name: CustomerID, dtype: int64
```

[23]:
```
plt.figure(figsize=(10,7))
sns.barplot(x=Month_cohort.values,y=Month_cohort.index)
plt.title("No. of active customers")
```

[23]:
```
Text(0.5, 1.0, 'No. of active customers')
```



Analyze the retention rate of customers

[24]:
```
Month_cohort
```

[24]:
```
Year_month
2010-12     948
2011-01     783
2011-02     798
2011-03    1020
2011-04     899
2011-05    1079
2011-06    1051
2011-07     993
2011-08     980
2011-09    1302
2011-10    1425
2011-11    1711
2011-12     686
Freq: M, Name: CustomerID, dtype: int64
```

[25]:
```
Month_cohort.shift
```

[25]:
```
<bound method Series.shift of Year_month
2010-12     948
2011-01     783
2011-02     798
2011-03    1020
2011-04     899
2011-05    1079
2011-06    1051
2011-07     993
2011-08     980
2011-09    1302
2011-10    1425
2011-11    1711
2011-12     686
Freq: M, Name: CustomerID, dtype: int64>
```

[26]:
```
Month_cohort.shift(1)
```

[26]:
```
Year_month
2010-12       NaN
2011-01     948.0
2011-02     783.0
2011-03     798.0
2011-04    1020.0
2011-05     899.0
2011-06    1079.0
2011-07    1051.0
2011-08     993.0
2011-09     980.0
2011-10    1302.0
2011-11    1425.0
2011-12    1711.0
Freq: M, Name: CustomerID, dtype: float64
```

[27]:
```
Month_cohort-Month_cohort.shift(1)
```

[27]:
```
Year_month
2010-12       NaN
2011-01    -165.0
2011-02      15.0
2011-03     222.0
2011-04    -121.0
2011-05     180.0
2011-06     -28.0
2011-07     -58.0
2011-08     -13.0
2011-09     322.0
```

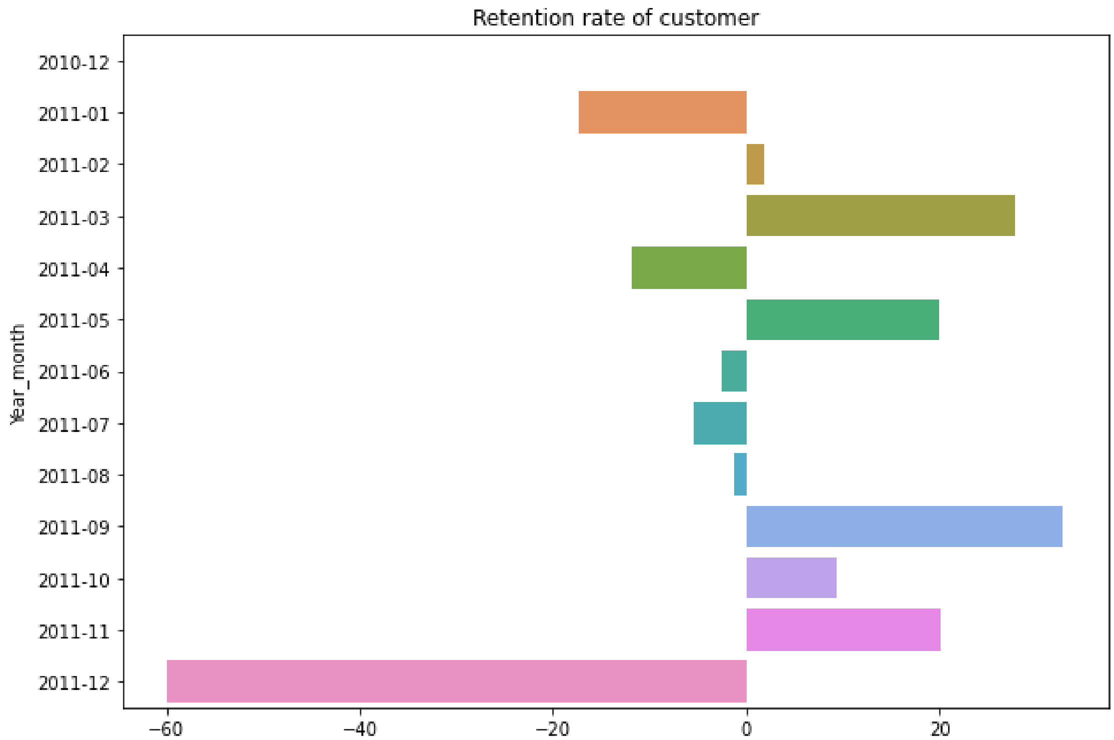
```
2011-10     123.0
2011-11     286.0
2011-12   -1025.0
Freq: M, Name: CustomerID, dtype: float64
```

[28]:
```
retention_rate = round(Month_cohort.pct_change(periods=1)*100,2)
retention_rate
```

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

[29]:
```
plt.figure(figsize=(10,7))
sns.barplot(x=retention_rate.values,y=retention_rate.index)
plt.title("Retention rate of customer")
```

[29]:
```
Text(0.5, 1.0, 'Retention rate of customer')
```



0.8 Data modeling

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

0.8.2 Recency model

[30]:
```
df["InvoiceDate"]
```

[30]:
```
0         2010-12-01 08:26:00
1         2010-12-01 08:26:00
2         2010-12-01 08:26:00
3         2010-12-01 08:26:00
4         2010-12-01 08:26:00
                  …
541904    2011-12-09 12:50:00
541905    2011-12-09 12:50:00
541906    2011-12-09 12:50:00
541907    2011-12-09 12:50:00
541908    2011-12-09 12:50:00
```

```
Name: InvoiceDate, Length: 406829, dtype: datetime64[ns]
```

[31]:
```
# We will fix reference date for calculating recency as last transaction day in␣
↪data + 1 day
from  datetime import timedelta
reference_day=max(df["InvoiceDate"]) + timedelta(days=1)
reference_day
```

[31]:
```
Timestamp('2011-12-10 12:50:00')
```

[32]:
```
df["days_to_last_order"]=(reference_day - df["InvoiceDate"]).dt.days
df.head()
```

[32]:
```
  InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice CustomerID  \
0    536365    85123A         6 2010-12-01 08:26:00       2.55    17850.0
1    536365     71053         6 2010-12-01 08:26:00       3.39    17850.0
2    536365    84406B         8 2010-12-01 08:26:00       2.75    17850.0
3    536365    84029G         6 2010-12-01 08:26:00       3.39    17850.0
4    536365    84029E         6 2010-12-01 08:26:00       3.39    17850.0

          Country Year_month  days_to_last_order
0  United Kingdom    2010-12                 374
1  United Kingdom    2010-12                 374
2  United Kingdom    2010-12                 374
3  United Kingdom    2010-12                 374
4  United Kingdom    2010-12                 374
```

[33]:
```
df_recency=df.groupby("CustomerID")["days_to_last_order"].min().reset_index()
df_recency
```

[33]:
```
      CustomerID  days_to_last_order
0        12346.0                 326
1        12347.0                   2
2        12348.0                  75
3        12349.0                  19
4        12350.0                 310
…            …                 …
4367     18280.0                 278
4368     18281.0                 181
4369     18282.0                   8
4370     18283.0                   4
4371     18287.0                  43

[4372 rows x 2 columns]
```

0.9 Frequency model

[34]:
```
df.groupby("CustomerID").nunique()
```

[34]:
```
            InvoiceNo  StockCode  Quantity  InvoiceDate  UnitPrice  Country  \
CustomerID
12346.0             2          1         2            2          1        1
12347.0             7        103        15            7         32        1
12348.0             4         22        10            4          7        1
12349.0             1         73        12            1         32        1
12350.0             1         17         5            1          7        1
…                 …          …        …            …          …        …
18280.0             1         10         5            1          7        1
18281.0             1          7         4            1          4        1
18282.0             3         12         9            3          9        1
18283.0            16        263        10           16         31        1
18287.0             3         59        10            3         17        1

            Year_month  days_to_last_order
CustomerID
12346.0              1                   1
12347.0              7                   7
12348.0              4                   4
12349.0              1                   1
12350.0              1                   1
…                  …                   …
18280.0              1                   1
18281.0              1                   1
18282.0              2                   3
18283.0             10                  14
18287.0              2                   3

[4372 rows x 8 columns]
```

[35]:
```
df_frequency=df.groupby("CustomerID").nunique()["InvoiceNo"].reset_index()
df_frequency
```

[35]:
```
      CustomerID  InvoiceNo
0        12346.0          2
1        12347.0          7
2        12348.0          4
3        12349.0          1
4        12350.0          1
…            …        …
4367     18280.0          1
4368     18281.0          1
4369     18282.0          3
```

```
4370     18283.0          16
4371     18287.0           3

[4372 rows x 2 columns]
```

0.10 Monetary model

[36]:
```
df["Amount"]=df["Quantity"]*df["UnitPrice"]
df.head()
```

[36]:
```
   InvoiceNo StockCode  Quantity         InvoiceDate  UnitPrice CustomerID  \
0     536365    85123A         6 2010-12-01 08:26:00       2.55    17850.0
1     536365     71053         6 2010-12-01 08:26:00       3.39    17850.0
2     536365    84406B         8 2010-12-01 08:26:00       2.75    17850.0
3     536365    84029G         6 2010-12-01 08:26:00       3.39    17850.0
4     536365    84029E         6 2010-12-01 08:26:00       3.39    17850.0

          Country Year_month  days_to_last_order  Amount
0  United Kingdom    2010-12                 374   15.30
1  United Kingdom    2010-12                 374   20.34
2  United Kingdom    2010-12                 374   22.00
3  United Kingdom    2010-12                 374   20.34
4  United Kingdom    2010-12                 374   20.34
```

[37]:
```
df_monetary=df.groupby("CustomerID").sum()["Amount"].reset_index()
df_monetary
```

[37]:
```
      CustomerID   Amount
0        12346.0     0.00
1        12347.0  4310.00
2        12348.0  1797.24
3        12349.0  1757.55
4        12350.0   334.40
…            …        …
4367     18280.0   180.60
4368     18281.0    80.82
4369     18282.0   176.60
4370     18283.0  2094.88
4371     18287.0  1837.28

[4372 rows x 2 columns]
```

0.11 RMF metrics

[38]:
```
df_recency
```

[38]:
```
      CustomerID  days_to_last_order
0        12346.0                 326
1        12347.0                   2
2        12348.0                  75
3        12349.0                  19
4        12350.0                 310
…            …                 …
4367     18280.0                 278
4368     18281.0                 181
4369     18282.0                   8
4370     18283.0                   4
4371     18287.0                  43

[4372 rows x 2 columns]
```

[39]:
```
df_frequency
```

[39]:
```
      CustomerID  InvoiceNo
0        12346.0          2
1        12347.0          7
2        12348.0          4
3        12349.0          1
4        12350.0          1
…            …        …
4367     18280.0          1
4368     18281.0          1
4369     18282.0          3
4370     18283.0         16
4371     18287.0          3

[4372 rows x 2 columns]
```

[40]:
```
df_monetary
```

[40]:
```
      CustomerID   Amount
0        12346.0     0.00
1        12347.0  4310.00
2        12348.0  1797.24
3        12349.0  1757.55
4        12350.0   334.40
…            …   …
4367     18280.0   180.60
4368     18281.0    80.82
4369     18282.0   176.60
4370     18283.0  2094.88
4371     18287.0  1837.28
```

```
[4372 rows x 2 columns]
```

[41]:
```
rf_model=pd.merge(df_recency,df_frequency,on="CustomerID",how="inner")
rfm_model=pd.merge(rf_model,df_monetary,on="CustomerID",how="inner")
rfm_model.columns=["CustomerID","Recency","Frequency","Monetary"]
rfm_model
```

[41]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	326	2	0.00
1	12347.0	2	7	4310.00
2	12348.0	75	4	1797.24
3	12349.0	19	1	1757.55
4	12350.0	310	1	334.40
…	…	…	…	…
4367	18280.0	278	1	180.60
4368	18281.0	181	1	80.82
4369	18282.0	8	3	176.60
4370	18283.0	4	16	2094.88
4371	18287.0	43	3	1837.28

```
[4372 rows x 4 columns]
```

0.12 Build RFM Segments.Give recency,frequency and monetary scores individually by dividing them into quartiles.Combine three ratings to get a RFM segment (as strings)

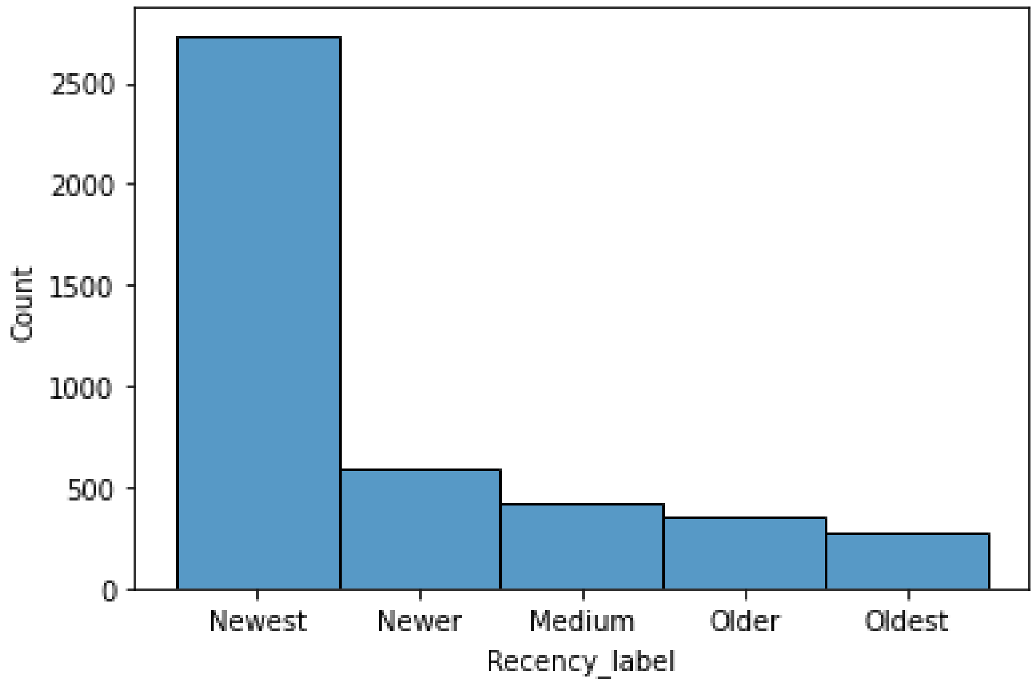
Note:

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

2.Rate "frequency" and "monetary" higher, because the company wants the customer to visit more often and spend more money.

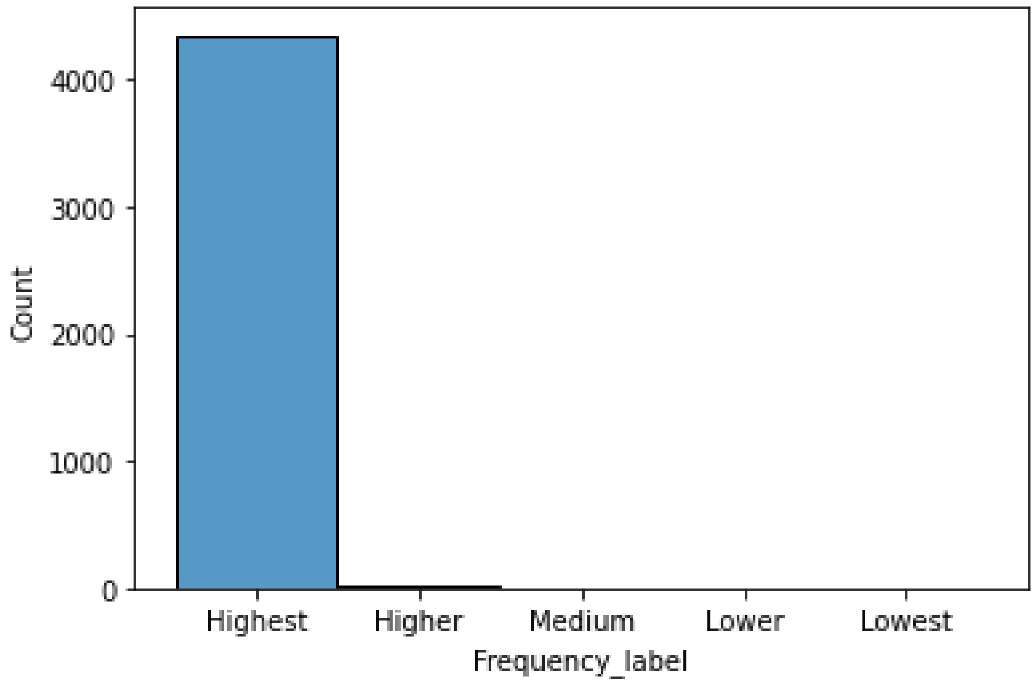
[42]:
```
rfm_model["Recency_label"]=pd.
↪cut(rfm_model["Recency"],bins=5,labels=["Newest","Newer","Medium","Older","Oldest"])
sns.histplot(rfm_model["Recency_label"])
rfm_model["Recency_label"].value_counts()
```

[42]:
```
Newest    2734
Newer      588
Medium     416
Older      353
Oldest     281
Name: Recency_label, dtype: int64
```



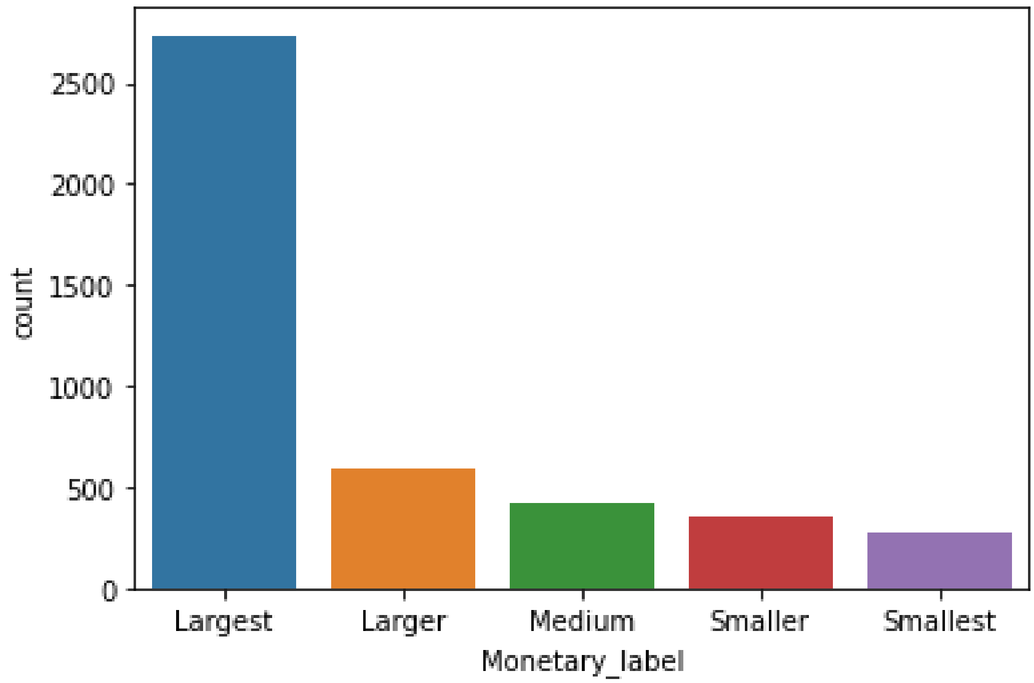
[43]:
```
rfm_model["Frequency_label"]=pd.
 ↪cut(rfm_model["Frequency"],bins=5,labels=["Highest","Higher","Medium","Lower","Lowest"])
sns.histplot(rfm_model["Frequency_label"])
rfm_model["Frequency_label"].value_counts()
```

[43]:
```
Highest    4348
Higher       18
Medium        3
Lowest        2
Lower         1
Name: Frequency_label, dtype: int64
```



[44]:
```
rfm_model["Monetary_label"]=pd.
 ↪cut(rfm_model["Recency"],bins=5,labels=["Largest","Larger","Medium","Smaller","Smallest"])
sns.countplot(rfm_model["Monetary_label"])
rfm_model["Monetary_label"].value_counts()
```

[44]:
```
Largest     2734
Larger       588
Medium       416
Smaller      353
Smallest     281
Name: Monetary_label, dtype: int64
```



[45]:
```
rfm_model["RFM_label"]=rfm_model[["Recency_label","Frequency_label","Monetary_label"]].
  ↪agg("-".join,axis=1)
rfm_model.head()
```

[45]:
```
   CustomerID  Recency  Frequency  Monetary Recency_label Frequency_label  \
0     12346.0      326          2      0.00        Oldest         Highest
1     12347.0        2          7   4310.00        Newest         Highest
2     12348.0       75          4   1797.24        Newest         Highest
3     12349.0       19          1   1757.55        Newest         Highest
4     12350.0      310          1    334.40        Oldest         Highest

  Monetary_label                 RFM_label
0       Smallest  Oldest-Highest-Smallest
1        Largest   Newest-Highest-Largest
2        Largest   Newest-Highest-Largest
3        Largest   Newest-Highest-Largest
4       Smallest  Oldest-Highest-Smallest
```

0.13 Get the RFM score by adding up the three ratings

[46]:
```
recency_dict={"Newest":5,"Newer":4,"Medium":3,"Older":2,"Oldest":1}
frequency_dict={"Lowest":1,"Lower":2,"Medium":3,"Higher":4,"Highest":5}
monetary_dict={"Smallest":1,"Smaller":2,"Medium":3,"Larger":4,"Largest":5}
```

```
rfm_model["RFM_score"]=rfm_model["Recency_label"].map(recency_dict).astype(int)␣
↪+ rfm_model["Frequency_label"].map(frequency_dict).astype(int) +␣
↪rfm_model["Monetary_label"].map(monetary_dict).astype(int)
rfm_model.head()
```

[46]:

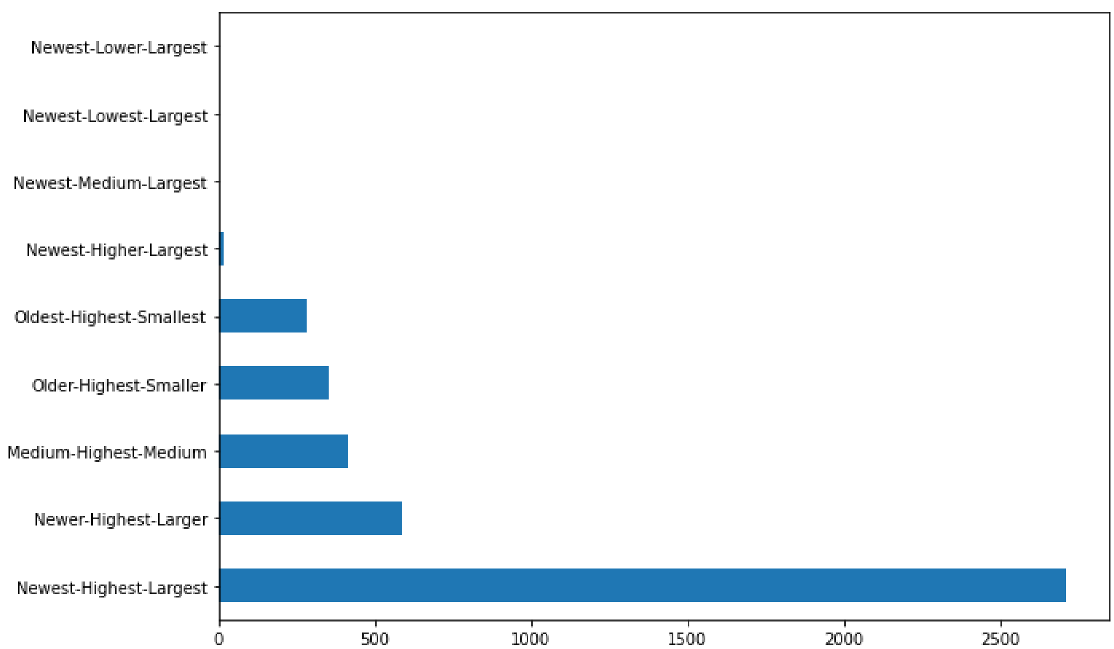
```
   CustomerID  Recency  Frequency  Monetary Recency_label Frequency_label  \
0     12346.0      326          2      0.00        Oldest         Highest
1     12347.0        2          7   4310.00        Newest         Highest
2     12348.0       75          4   1797.24        Newest         Highest
3     12349.0       19          1   1757.55        Newest         Highest
4     12350.0      310          1    334.40        Oldest         Highest

  Monetary_label                RFM_label  RFM_score
0       Smallest  Oldest-Highest-Smallest          7
1        Largest   Newest-Highest-Largest         15
2        Largest   Newest-Highest-Largest         15
3        Largest   Newest-Highest-Largest         15
4       Smallest  Oldest-Highest-Smallest          7
```

0.14 Analyze the RFM segments by summarizing them and comment on the findings

[47]:

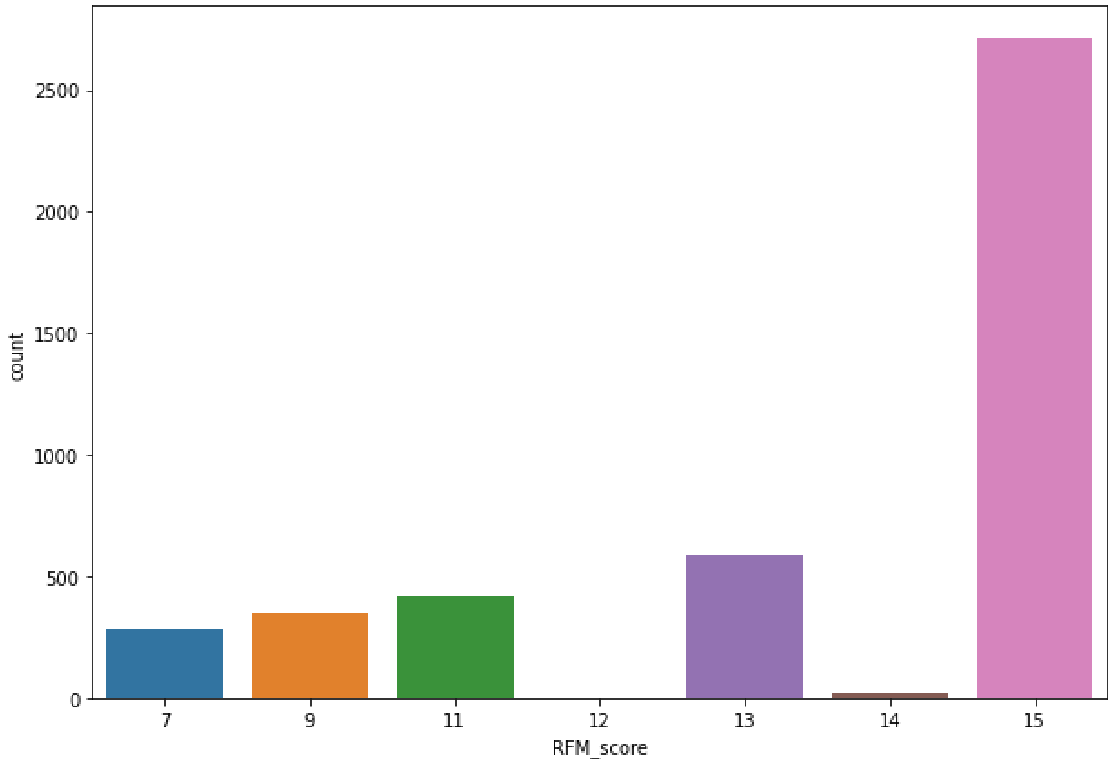
```
plt.figure(figsize=(10,7))
rfm_model["RFM_label"].value_counts().plot(kind="barh")
```

[47]: <AxesSubplot:>



[48]:
```
plt.figure(figsize=(10,7))
sns.countplot(rfm_model["RFM_score"])
```

[48]: <AxesSubplot:xlabel='RFM_score', ylabel='count'>



0.15 Project Task: Week 2(Create clusters using k-means clustering algorithm)

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

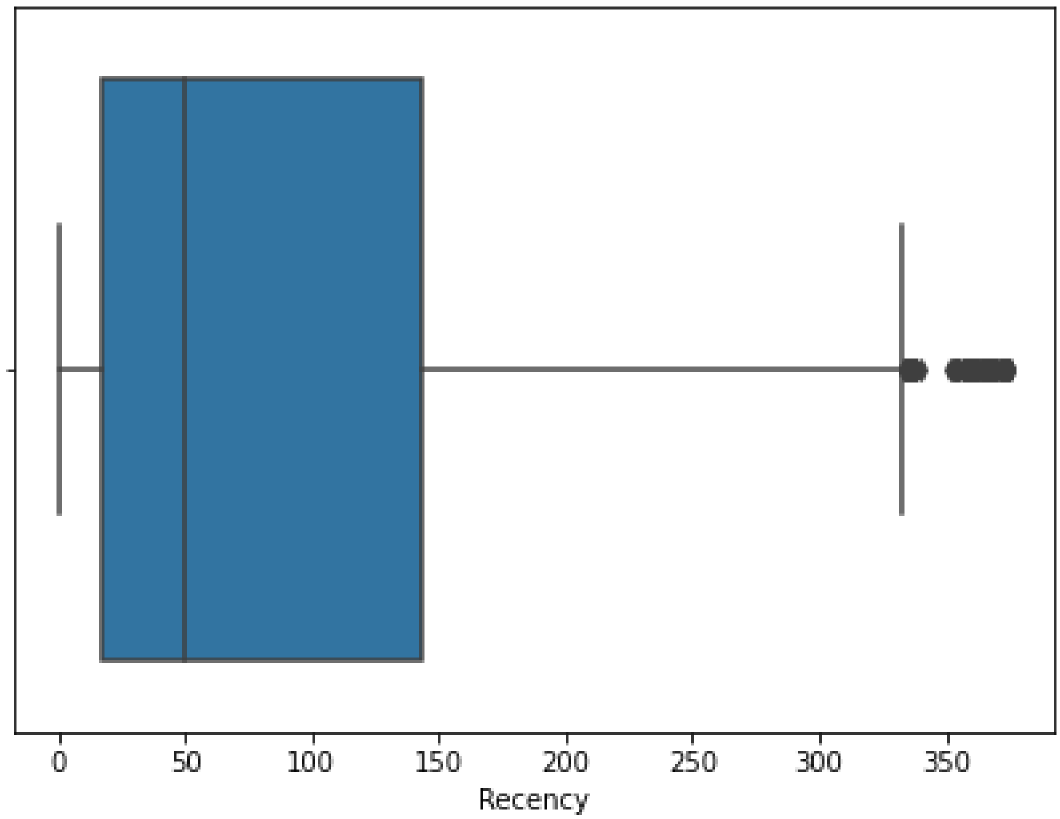
[49]:
```
rfm_model.head()
```

[49]:
```
   CustomerID  Recency  Frequency  Monetary Recency_label Frequency_label  \
0     12346.0      326          2      0.00        Oldest         Highest
1     12347.0        2          7   4310.00        Newest         Highest
2     12348.0       75          4   1797.24        Newest         Highest
3     12349.0       19          1   1757.55        Newest         Highest
4     12350.0      310          1    334.40        Oldest         Highest

  Monetary_label                RFM_label  RFM_score
0       Smallest  Oldest-Highest-Smallest          7
```

```
1         Largest   Newest-Highest-Largest        15
2         Largest   Newest-Highest-Largest        15
3         Largest   Newest-Highest-Largest        15
4        Smallest  Oldest-Highest-Smallest         7
```

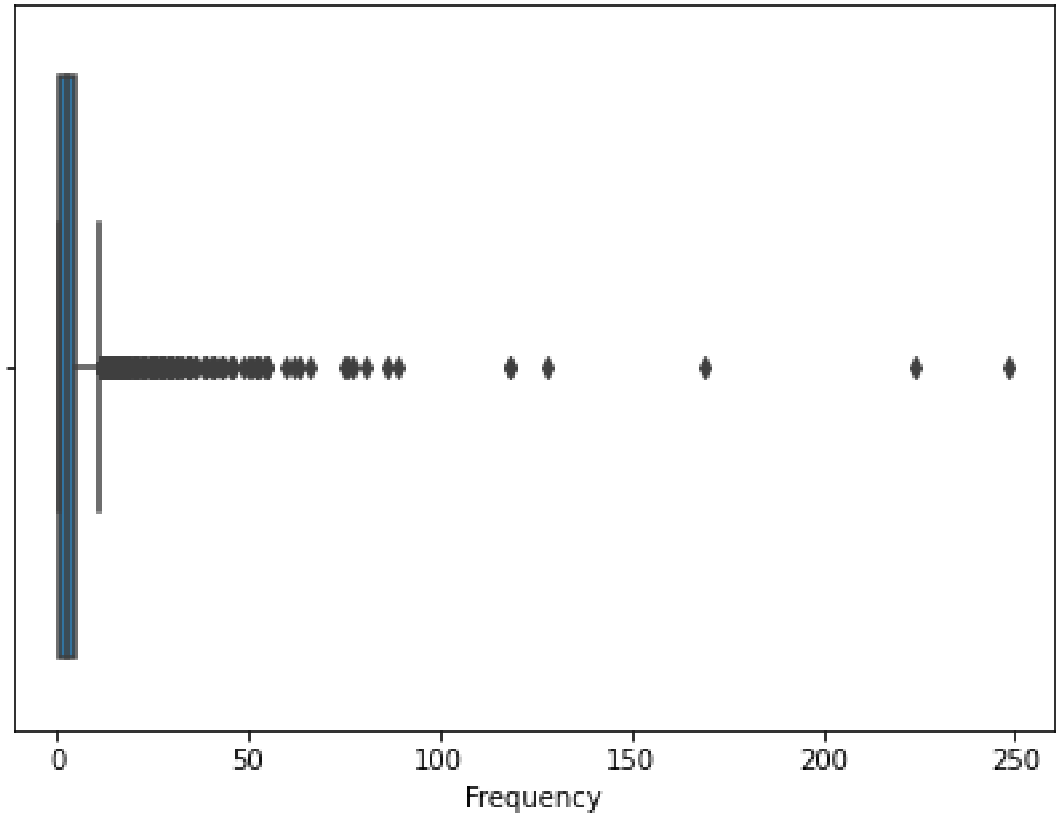
[50]:
```
plt.figure(figsize=(7,5))
sns.boxplot(rfm_model["Recency"])
```

[50]: <AxesSubplot:xlabel='Recency'>



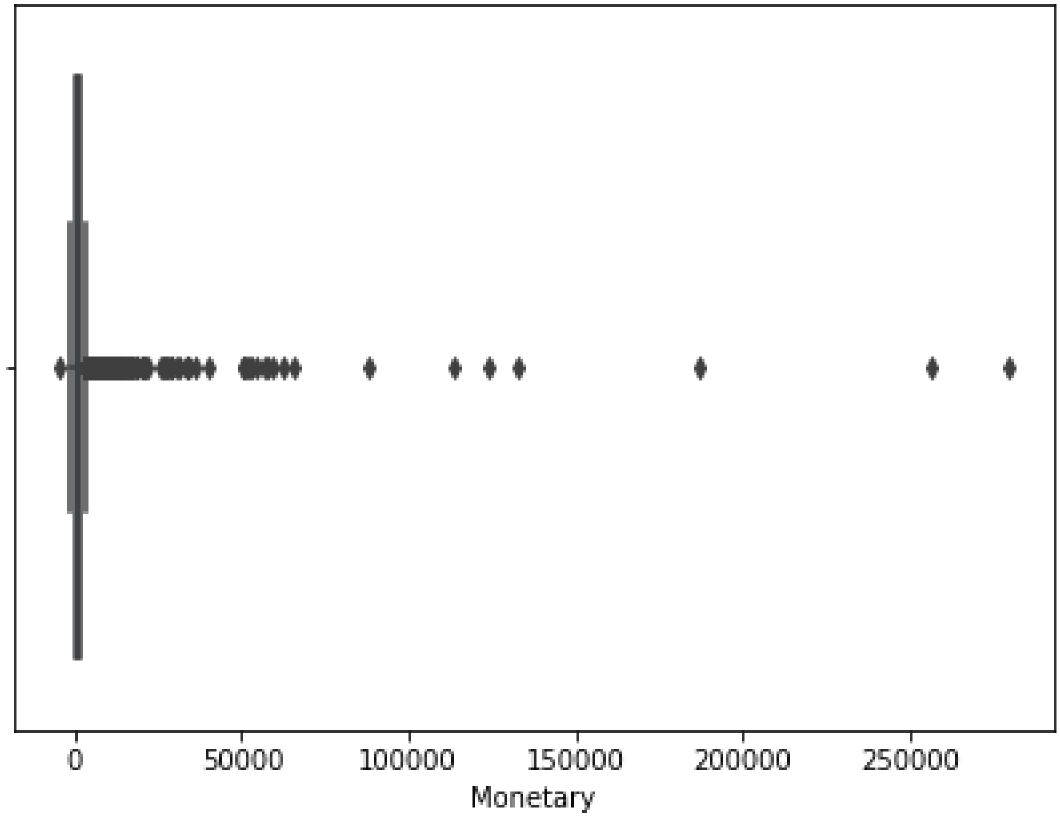
[51]:
```
plt.figure(figsize=(7,5))
sns.boxplot(rfm_model["Frequency"])
```

[51]: <AxesSubplot:xlabel='Frequency'>

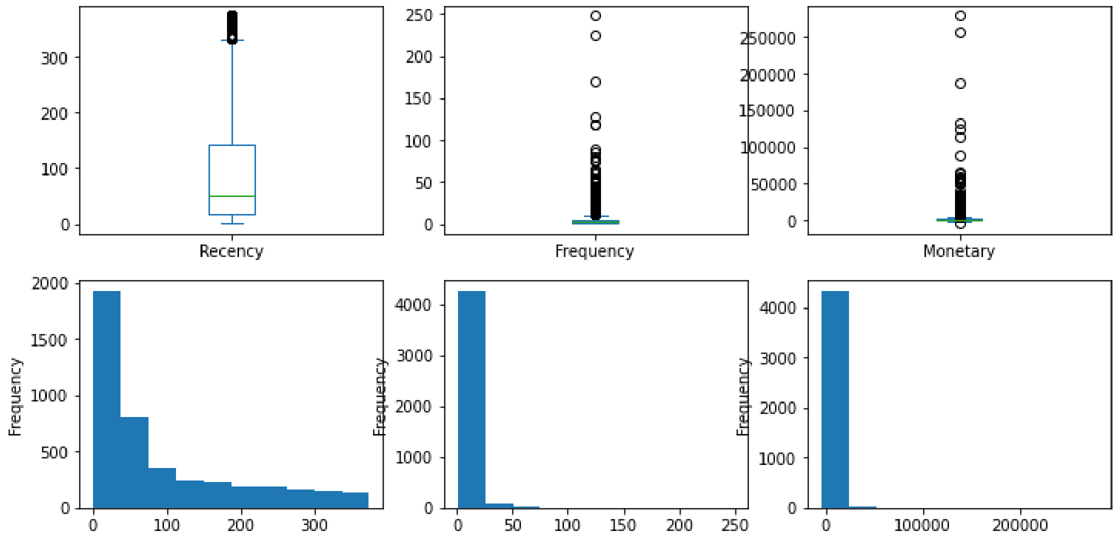



```
[52]: plt.figure(figsize=(7,5))
      sns.boxplot(rfm_model["Monetary"])
```

```
[52]: <AxesSubplot:xlabel='Monetary'>
```



[53]:
```
plt.figure(figsize=(12,6))

for i, feature in enumerate(['Recency', 'Frequency', 'Monetary']):
    plt.subplot(2,3,i+1)
    rfm_model[feature].plot(kind='box')
    plt.subplot(2,3,i+1+3)
    rfm_model[feature].plot(kind='hist')
```

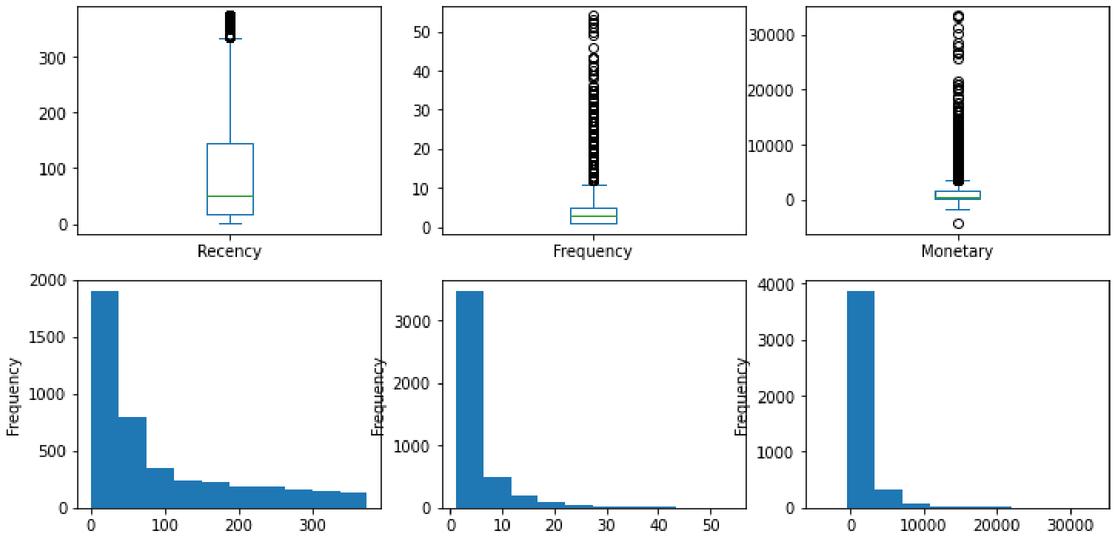


0.15.2 Outliers: Frequency and Monetary features in above data seem to have lot of outliers. Lets drop them.

[54]:
```
rfm_model=rfm_model[(rfm_model["Frequency"]<60) & (rfm_model["Monetary"]<40000)]
rfm_model.shape
```

[54]: (4346, 9)

[55]:
```
plt.figure(figsize=(12,6))

for i, feature in enumerate(['Recency', 'Frequency', 'Monetary']):
    plt.subplot(2,3,i+1)
    rfm_model[feature].plot(kind='box')
    plt.subplot(2,3,i+1+3)
    rfm_model[feature].plot(kind='hist')
```



0.15.3 Log Transformation: Now since all three features have right skewed data therefore we will use log transformation of these features in our model.

[56]:
```
rfm_model_log=pd.DataFrame()
rfm_model_log["Recency"]=np.log(rfm_model["Recency"])
rfm_model_log["Frequency"]=np.log(rfm_model["Frequency"])
rfm_model_log["Monetary"]=np.log(rfm_model["Monetary"]-rfm_model["Monetary"].
    ↪min()+1)
rfm_model_log.head()
```

[56]:

	Recency	Frequency	Monetary
0	5.786897	0.693147	8.363723
1	0.693147	1.945910	9.059358
2	4.317488	1.386294	8.713725
3	2.944439	0.000000	8.707182
4	5.736572	0.000000	8.438806

Standard Scalar Transformation: It is extremely important to rescale the features so that they have a comparable scale.

[57]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
rfm_model_scaled=scaler.
    ↪fit_transform(rfm_model_log[["Recency","Frequency","Monetary"]])

rfm_model_scaled=pd.DataFrame(rfm_model_scaled)
rfm_model_scaled.columns=["Recency","Frequency","Monetary"]
rfm_model_scaled.head()
```

[57]:

```
     Recency  Frequency  Monetary
0   1.402988  -0.388507 -0.772738
1  -2.100874   0.967301  1.481096
2   0.392218   0.361655  0.361257
3  -0.552268  -1.138669  0.340058
4   1.368370  -1.138669 -0.529472
```

0.16 Decide the optimum number of clusters to be formed

[58]:

```python
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=3,max_iter=50)
kmeans.fit(rfm_model_scaled)
```

[58]:

```
KMeans(max_iter=50, n_clusters=3)
```

[59]:

```python
kmeans.labels_
```

[59]:

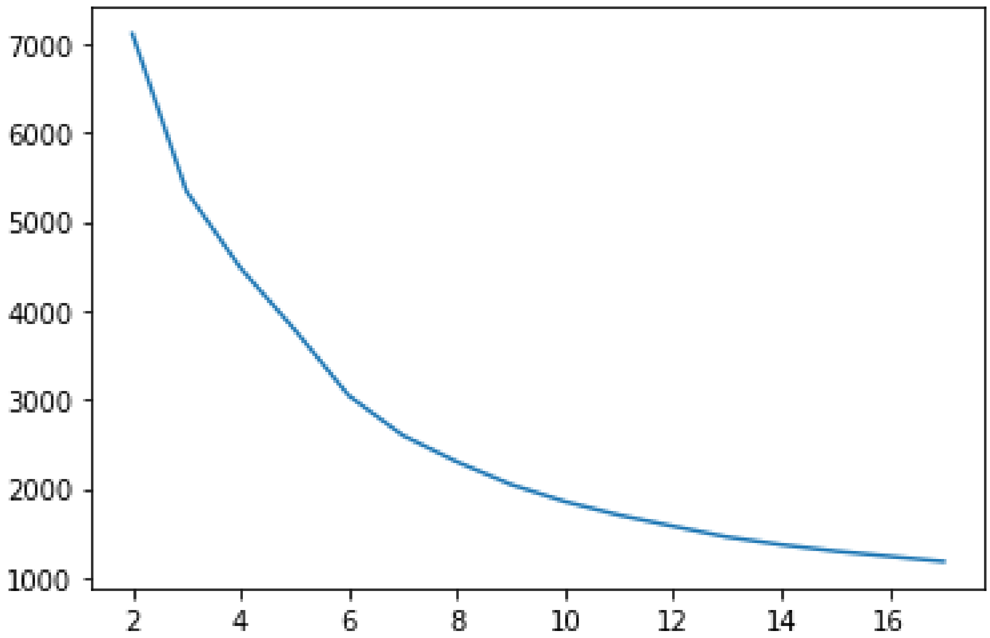
```
array([1, 2, 0, …, 0, 2, 0])
```

[60]:

```python
#Finding the Optimal Number of Clusters with the help of Elbow Curve/ SSD.
ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,13,14,15,16,17]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=100)
    kmeans.fit(rfm_model_scaled)

    ssd.append(kmeans.inertia_)
plt.plot(range_n_clusters,ssd)
```

[60]:

```
[<matplotlib.lines.Line2D at 0x26561abc8e0>]
```



[61]:
```
#Creating dataframe for exporting to create visualization in tableau later
df_inertia=pd.DataFrame(list(zip(range_n_clusters, ssd)),columns=['clusters',␣
↪'intertia'])
df_inertia
```

[61]:

	clusters	intertia
0	2	7109.487111
1	3	5340.527958
2	4	4478.752264
3	5	3786.319617
4	6	3043.888575
5	7	2598.312427
6	8	2300.954046
7	9	2045.352385
8	10	1852.890952
9	11	1700.738793
10	12	1575.673375
11	13	1454.142269
12	14	1367.616241
13	15	1295.630322
14	16	1236.615492
15	17	1179.595720

[62]:
```
# Finding the Optimal Number of Clusters with the help of Silhouette Analysis
from sklearn.metrics import silhouette_score
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_model_scaled)

    cluster_labels = kmeans.labels_

    silhouette_avg = silhouette_score(rfm_model_scaled,cluster_labels)
    print("For n_clusters={0}, the silhouette score is {1}".
 ↪format(num_clusters, silhouette_avg))
```

```
For n_clusters=2, the silhouette score is 0.4413791847967848
For n_clusters=3, the silhouette score is 0.3812337793128446
For n_clusters=4, the silhouette score is 0.36216517600280823
For n_clusters=5, the silhouette score is 0.3648252955662078
For n_clusters=6, the silhouette score is 0.3442909657149203
For n_clusters=7, the silhouette score is 0.34272780946054593
For n_clusters=8, the silhouette score is 0.33536391955837613
For n_clusters=9, the silhouette score is 0.3463493827137049
For n_clusters=10, the silhouette score is 0.35609789640541023
```

0.16.1 We can select optimum number of clusters as 3 in our final model

[63]:
```
# Final model with k=3
kmeans=KMeans(n_clusters=3,max_iter=50)
kmeans.fit(rfm_model_scaled)
```

[63]:
```
KMeans(max_iter=50, n_clusters=3)
```

0.17 Analyze these clusters and comment on the results

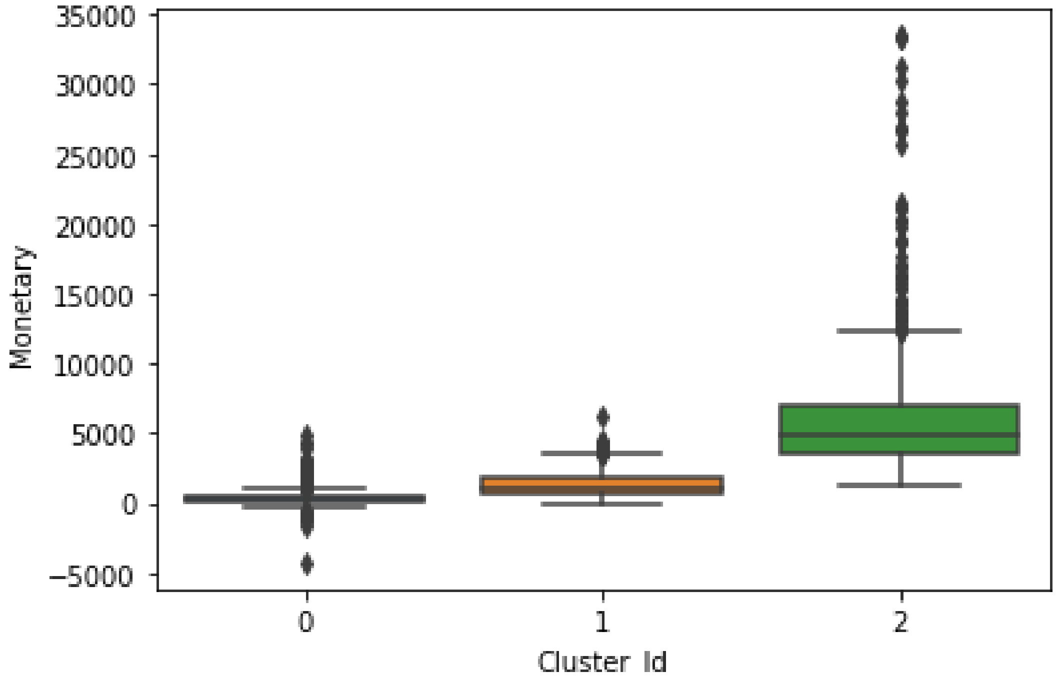
[64]:
```
# assign the label
rfm_model['Cluster_Id']=kmeans.labels_
rfm_model.head()
```

[64]:
```
   CustomerID  Recency  Frequency  Monetary Recency_label Frequency_label  \
0     12346.0      326          2      0.00        Oldest         Highest
1     12347.0        2          7   4310.00        Newest         Highest
2     12348.0       75          4   1797.24        Newest         Highest
3     12349.0       19          1   1757.55        Newest         Highest
4     12350.0      310          1    334.40        Oldest         Highest

  Monetary_label                RFM_label  RFM_score  Cluster_Id
0       Smallest  Oldest-Highest-Smallest          7           0
1        Largest   Newest-Highest-Largest         15           2
2        Largest   Newest-Highest-Largest         15           1
3        Largest   Newest-Highest-Largest         15           0
```

```
4       Smallest   Oldest-Highest-Smallest          7           0
```

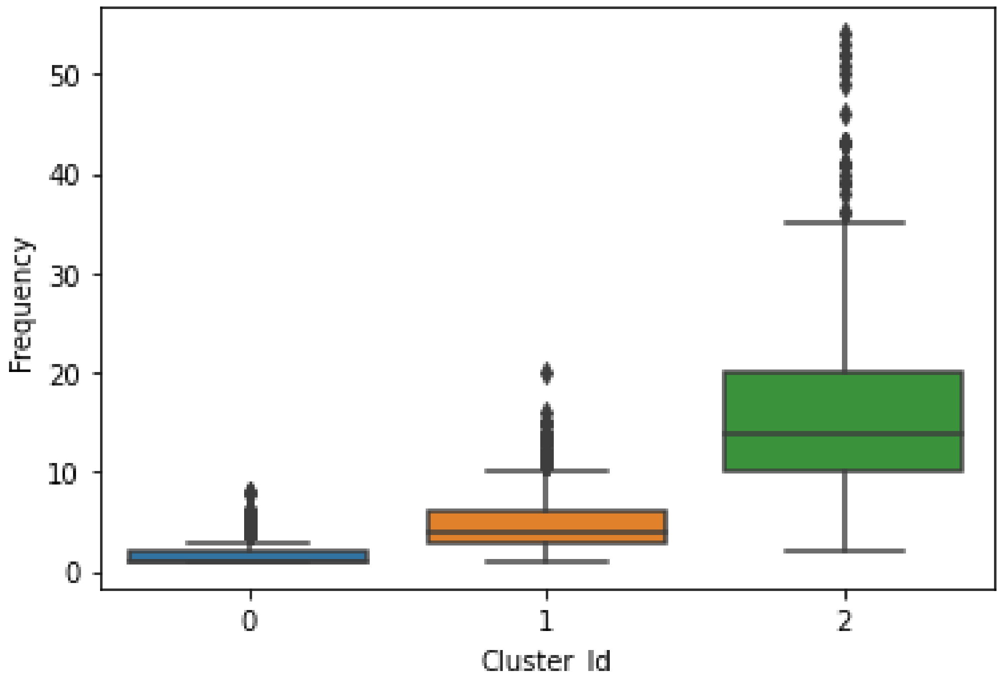
[65]:
```python
# Box plot to visualize Cluster Id vs Monetary
sns.boxplot(x="Cluster_Id",y="Monetary",data=rfm_model)
```

[65]: <AxesSubplot:xlabel='Cluster_Id', ylabel='Monetary'>



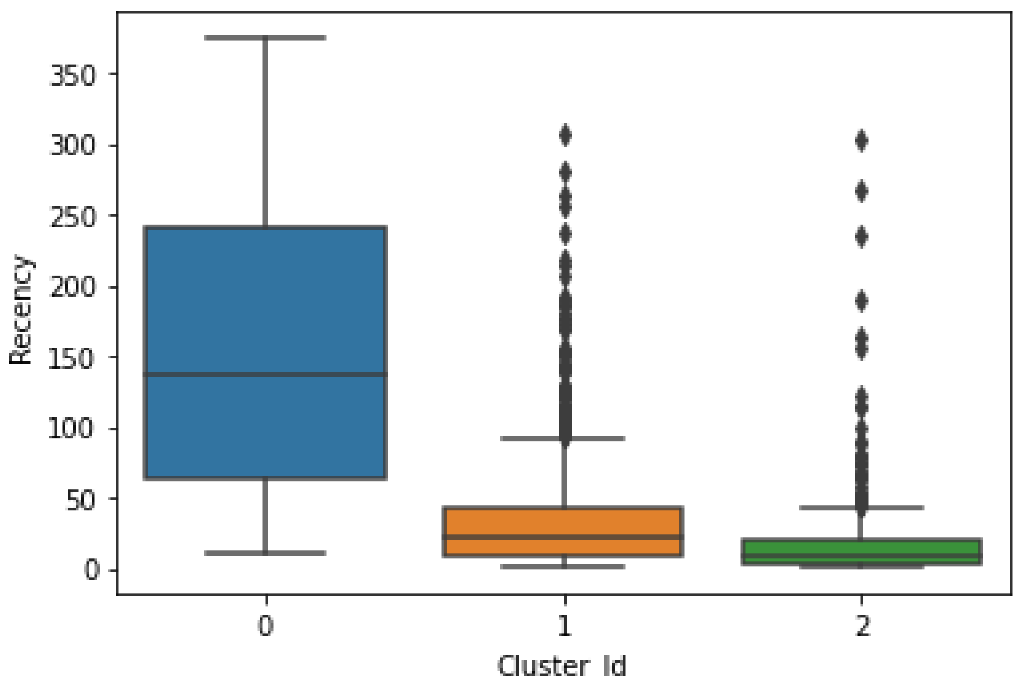
[66]:
```python
# Box plot to visualize Cluster Id vs Frequency
sns.boxplot(x="Cluster_Id",y="Frequency",data=rfm_model)
```

[66]: <AxesSubplot:xlabel='Cluster_Id', ylabel='Frequency'>



[67]:
```
# Box plot to visualize Cluster Id vs Recency
sns.boxplot(x="Cluster_Id",y="Recency",data=rfm_model)
```

[67]: <AxesSubplot:xlabel='Cluster_Id', ylabel='Recency'>



Inference: As we can observe from above boxplots that our model has nicely created 3 segements of customer with the interpretation as below:-

. Customers with Cluster Id 0 are less frequent buyers with low monetary expenditure and also they have not purchased anything in recent time and hence least important for business.

. Customers with Cluster Id 1 are the customers having Recency, Frequency and Monetary score in the medium range.

. Customers with Cluster Id 2 are the most frequent buyers, spending high amount and recently placing orders so they are the most important customers from business point of view.

[68]:
```
#Writing dataframe to excel file for creating visualization in tableau
from pandas import ExcelWriter
writer = pd.ExcelWriter("Online Retail.xlsx",engine="xlsxwriter")

df.to_excel(writer,sheet_name="master_data",index=False)
rfm_model.to_excel(writer,sheet_name="rfm_model",index=False)
df_inertia.to_excel(writer,sheet_name="inertia",index=False)
writer.save()
```

[70]:
```
product_desc = pd.read_excel("Online Retail1.xlsx")
product_desc=product_desc[["StockCode","Description"]]
product_desc=product_desc.drop_duplicates()
product_desc.to_csv("product_desc.csv",index=False)
```

0.17.1 THANK YOU…!!!