23012064_dsm2

November 5, 2024

Name: Abdallah Saber

ID: 23012064

```
[ ]: # Importing Libraries
     import numpy as np
     import pandas as pd


     import time, warnings
     import datetime as dt

     #visualizations
     import matplotlib.pyplot as plt
     from pandas.plotting import scatter_matrix
     import seaborn as sns
```

1 Get the Data

```
[2]: #load the dataset
     retail_df = pd.read_csv('./data.csv',encoding="ISO-8859-1",dtype={'CustomerID':␣
      ↪str,'InvoiceID': str})
     retail_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  12/1/2010 8:26       2.55      17850  United Kingdom
     1  12/1/2010 8:26       3.39      17850  United Kingdom
     2  12/1/2010 8:26       2.75      17850  United Kingdom
     3  12/1/2010 8:26       3.39      17850  United Kingdom
     4  12/1/2010 8:26       3.39      17850  United Kingdom
```

2 Prepare the Data

restrict the data to only United Kingdom customers

```
[3]: retail_uk = retail_df[retail_df['Country']=='United Kingdom']
     #check the shape
     retail_uk.shape
```

```
[3]: (495478, 8)
```

```
[4]: #remove canceled orders
     retail_uk = retail_uk[retail_uk['Quantity']>0]
     retail_uk.shape
```

```
[4]: (486286, 8)
```

```
[5]: #remove rows where customerID are NA
     retail_uk.dropna(subset=['CustomerID'],how='all',inplace=True)
     retail_uk.shape
```

```
[5]: (354345, 8)
```

```
[6]: #restrict the data to one full year because it's better to use a metric per␣
       ↪Months or Years in RFM
     retail_uk = retail_uk[retail_uk['InvoiceDate']>= "2010-12-09"]
     retail_uk.shape
```

```
[6]: (176137, 8)
```

```
[7]: print("Summary..")
     #exploring the unique values of each attribute
     print("Number of transactions: ", retail_uk['InvoiceNo'].nunique())
     print("Number of products bought: ",retail_uk['StockCode'].nunique())
     print("Number of customers:", retail_uk['CustomerID'].nunique() )
     print("Percentage of customers NA: ", round(retail_uk['CustomerID'].isnull().
       ↪sum() * 100 / len(retail_df),2),"%" )
```

```
Summary..
Number of transactions:  8789
Number of products bought:  3294
Number of customers: 2864
Percentage of customers NA:  0.0 %
```

3 RFM Analysis

3.1 Recency

[8]:
```
#last date available in our dataset
retail_uk['InvoiceDate'].max()
```

[8]:
```
'9/9/2011 9:52'
```

The last date we have is 2011-12-09 so we will use it as reference.

[9]:
```
now = dt.date(2011,12,9)
print(now)
```

```
2011-12-09
```

[10]:
```
#create a new column called date which contains the date of invoice only
retail_uk['date'] = pd.DatetimeIndex(retail_uk['InvoiceDate']).date
```

[11]:
```
retail_uk.head()
```

[11]:
```
       InvoiceNo StockCode                          Description  Quantity  \
105335    545220     21955    DOORMAT UNION JACK GUNS AND ROSES         2
105336    545220     48194                       DOORMAT HEARTS         2
105337    545220     22556       PLASTERS IN TIN CIRCUS PARADE         12
105338    545220     22139     RETROSPOT TEA SET CERAMIC 11 PC          3
105339    545220    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         4

          InvoiceDate  UnitPrice CustomerID         Country        date
105335  3/1/2011 8:30       7.95      14620  United Kingdom  2011-03-01
105336  3/1/2011 8:30       7.95      14620  United Kingdom  2011-03-01
105337  3/1/2011 8:30       1.65      14620  United Kingdom  2011-03-01
105338  3/1/2011 8:30       4.95      14620  United Kingdom  2011-03-01
105339  3/1/2011 8:30       3.75      14620  United Kingdom  2011-03-01
```

[12]:
```
#group by customers and check last date of purshace
recency_df = retail_uk.groupby(by='CustomerID', as_index=False)['date'].max()
recency_df.columns = ['CustomerID','LastPurshaceDate']
recency_df.head()
```

[12]:
```
  CustomerID LastPurshaceDate
0      12747       2011-08-22
1      12748       2011-09-30
2      12749       2011-08-01
3      12820       2011-09-26
4      12821       2011-05-09
```

[13]:
```
#calculate recency
```

```
recency_df['Recency'] = recency_df['LastPurshaceDate'].apply(lambda x: (now -␣
 ↪x).days)
```

[14]:
```
recency_df.head()
```

[14]:
```
   CustomerID LastPurshaceDate  Recency
0       12747       2011-08-22      109
1       12748       2011-09-30       70
2       12749       2011-08-01      130
3       12820       2011-09-26       74
4       12821       2011-05-09      214
```

[15]:
```
#drop LastPurchaseDate as we don't need it anymore
recency_df.drop('LastPurshaceDate',axis=1,inplace=True)
```

3.2 Frequency

[16]:
```
# drop duplicates
retail_uk_copy = retail_uk
retail_uk_copy.drop_duplicates(subset=['InvoiceNo', 'CustomerID'],␣
 ↪keep="first", inplace=True)
#calculate frequency of purchases
frequency_df = retail_uk_copy.groupby(by=['CustomerID'],␣
 ↪as_index=False)['InvoiceNo'].count()
frequency_df.columns = ['CustomerID','Frequency']
frequency_df.head()
```

[16]:
```
   CustomerID  Frequency
0       12747          5
1       12748         96
2       12749          3
3       12820          1
4       12821          1
```

3.3 Monetary

[17]:
```
#create column total cost
retail_uk['TotalCost'] = retail_uk['Quantity'] * retail_uk['UnitPrice']
```

[18]:
```
monetary_df = retail_uk.groupby(by='CustomerID',as_index=False).
 ↪agg({'TotalCost': 'sum'})
monetary_df.columns = ['CustomerID','Monetary']
monetary_df.head()
```

[18]:
```
   CustomerID  Monetary
0       12747    191.85
1       12748   1054.43
```

```
2      12749     67.00
3      12820     15.00
4      12821     19.92
```

3.4 Create RFM Table

[19]:

```
#merge recency dataframe with frequency dataframe
temp_df = recency_df.merge(frequency_df,on='CustomerID')
temp_df.head()
```

[19]:

```
   CustomerID  Recency  Frequency
0       12747      109          5
1       12748       70         96
2       12749      130          3
3       12820       74          1
4       12821      214          1
```

[20]:

```
#merge with monetary dataframe to get a table with the 3 columns
rfm_df = temp_df.merge(monetary_df,on='CustomerID')
#use CustomerID as index
rfm_df.set_index('CustomerID',inplace=True)
#check the head
rfm_df.head()
```

[20]:

```
            Recency  Frequency  Monetary
CustomerID
12747           109          5    191.85
12748            70         96   1054.43
12749           130          3     67.00
12820            74          1     15.00
12821           214          1     19.92
```

3.5 RFM Table Correctness verification

[21]:

```
retail_uk[retail_uk['CustomerID']=='12820']
```

[21]:

```
       InvoiceNo StockCode                        Description  Quantity  \
360567    568236     23328  SET 6 SCHOOL MILK BOTTLES IN CRATE         4

            InvoiceDate  UnitPrice CustomerID         Country        date  \
360567  9/26/2011 11:49       3.75      12820  United Kingdom  2011-09-26

        TotalCost
360567       15.0
```

[22]:

```
(now - dt.date(2011,9,26)).days == 74
```

[22]:

```
True
```

3.5.1 RFM Quartiles

[23]:
```
quantiles = rfm_df.quantile(q=[0.25,0.5,0.75])
quantiles
```

[23]:
```
      Recency  Frequency  Monetary
0.25     85.0        1.0     16.35
0.50    119.0        2.0     35.40
0.75    183.0        3.0     92.42
```

[24]:
```
quantiles.to_dict()
```

[24]:
```
{'Recency': {0.25: 85.0, 0.5: 119.0, 0.75: 183.0},
 'Frequency': {0.25: 1.0, 0.5: 2.0, 0.75: 3.0},
 'Monetary': {0.25: 16.35, 0.5: 35.400000000000006, 0.75: 92.42}}
```

3.5.2 Creation of RFM Segments

We will create two segmentation classes since, high recency is bad, while high frequency and monetary value is good.

[ ]:
```
# Arguments (x = value, p = recency, monetary_value, frequency, d = quartiles␣
 ↪dict)
def RScore(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1


# Arguments (x = value, p = recency, monetary_value, frequency, k = quartiles␣
 ↪dict)
def FMScore(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
```

[26]:
```
#create rfm segmentation table
rfm_segmentation = rfm_df
```

```
rfm_segmentation['R_Quartile'] = rfm_segmentation['Recency'].apply(RScore,␣
 ↪args=('Recency',quantiles,))
rfm_segmentation['F_Quartile'] = rfm_segmentation['Frequency'].apply(FMScore,␣
 ↪args=('Frequency',quantiles,))
rfm_segmentation['M_Quartile'] = rfm_segmentation['Monetary'].apply(FMScore,␣
 ↪args=('Monetary',quantiles,))
```

[27]:

```
rfm_segmentation.head()
```

[27]:

```
            Recency  Frequency  Monetary  R_Quartile  F_Quartile  M_Quartile
CustomerID
12747           109          5    191.85           3           4           4
12748            70         96   1054.43           4           4           4
12749           130          3     67.00           2           3           3
12820            74          1     15.00           4           1           1
12821           214          1     19.92           1           1           2
```

Get the RFM Segment value

[28]:

```
rfm_segmentation['RFMScore'] = rfm_segmentation.R_Quartile.map(str) \
                            + rfm_segmentation.F_Quartile.map(str) \
                            + rfm_segmentation.M_Quartile.map(str)
rfm_segmentation.head()
```

[28]:

```
            Recency  Frequency  Monetary  R_Quartile  F_Quartile  M_Quartile  \
CustomerID
12747           109          5    191.85           3           4           4
12748            70         96   1054.43           4           4           4
12749           130          3     67.00           2           3           3
12820            74          1     15.00           4           1           1
12821           214          1     19.92           1           1           2

           RFMScore
CustomerID
12747           344
12748           444
12749           233
12820           411
12821           112
```

(best customers)

[29]:

```
rfm_segmentation[rfm_segmentation['RFMScore']=='444'].sort_values('Monetary',␣
 ↪ascending=False).head(10)
```

[29]:

```
            Recency  Frequency  Monetary  R_Quartile  F_Quartile  M_Quartile  \
CustomerID
18102            72         34  26632.62           4           4           4
```

```
17949               70          32  22504.73           4           4           4
17450               70          28  18009.06           4           4           4
16029               80          39  15119.49           4           4           4
16013               70          24  10402.34           4           4           4
12901               81          20   5915.66           4           4           4
13798               72          34   4648.80           4           4           4
17857               72          12   4644.68           4           4           4
13694               71          32   4472.68           4           4           4
15061               73          23   3417.70           4           4           4

           RFMScore
CustomerID
18102           444
17949           444
17450           444
16029           444
16013           444
12901           444
13798           444
17857           444
13694           444
15061           444
```

How many customers do we have in each segment?

```
[31]: print("Best Customers:␣
        ↪",len(rfm_segmentation[rfm_segmentation['RFMScore']=='444']))
      print('Loyal Customers:␣
        ↪',len(rfm_segmentation[rfm_segmentation['F_Quartile']==4]))
      print("Big Spenders: ",len(rfm_segmentation[rfm_segmentation['M_Quartile']==4]))
```

```
Best Customers:  218
Loyal Customers:  687
Big Spenders:  716
```