RFM Analysis

May 19, 2024

1 Description

This project is a recency, frequency, monetary value (RFM) analysis of a pharmacy chain. Here customers are divided into 27 segments based on the aforementioned parameters of their purchases. Segmenting the client base could be used to conduct personalized marketing campaigns after the analysis.

The used dataset is customer loyalty's bonus system where each participant has a personal card number that starts with 2000. Sometimes the terminal goes offline and card numbers are recorded as a unique unidentifiable code. Since the latter cannot be helpful for the analysis, it will have to be got rid of.

[1]:
```
# Importing necessary libraries
import pandas as pd
import seaborn as sns
import numpy as np
import plotly.express as px
```

[2]:
```
# Reading the data
df = pd.read_csv('apteka.csv',sep=';')
```

[3]:
```
# Looking at the dataframe
df.head()
```

[3]:

	datetime	shop	card	bonus_earned	bonus_spent	summ	summ_with_disc	doc_id
0	2021-07-13 12:56:09.000	2	2000200195023	51	0	3400	3400	15#2002741#65938#2_29
1	2021-07-30 10:42:00.000	2	2000200193494	57	0	747	747	15#2002972#65955#2_5
2	2021-10-11 12:55:23.000	2	2000200199106	92	253	3077	3077	15#2004060#66028#2_29
3	2021-10-14 14:48:56.000	2	2000200168768	1	0	54	54	15#2004107#66031#2_57
4	2021-10-20 11:09:39.000	2	2000200226314	101	0	1733	1733	15#2004192#66037#2_16

Here "datetime" is the time of a transaction, "shop" is the shop where the purchase was made,

"card" is the card number, "bonus_earned" is the bonus for the purchase, "bonus_spent" is how many bonuses were used to make the purchase, "summ" is the sales sum, "sum_with_disc" is the sales sum after a discount if any, and "doc_id" is the receipt info. The only columns that will be used in this analysis are the "datetime", "card", and "summ_with_disc" columns.

[4]:
```
# Looking at datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38486 entries, 0 to 38485
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   datetime        38486 non-null  object
 1   shop            38486 non-null  object
 2   card            38486 non-null  object
 3   bonus_earned    38486 non-null  int64
 4   bonus_spent     38486 non-null  int64
 5   summ            38486 non-null  int64
 6   summ_with_disc  38486 non-null  int64
 7   doc_id          38486 non-null  object
dtypes: int64(4), object(4)
memory usage: 2.3+ MB
```

Since we need the "datetime" column and it is currently in the text format, it is best to change its format to datetime.

[5]:
```
# Changing the format
df['datetime'] = pd.to_datetime(df['datetime'])
```

2 Getting rid of unidentifiable card numbers

[6]:
```
# Looking at the proper card numbers
df[df.card.str.startswith('2000')].head()
```

[6]:

	datetime	shop	card	bonus_earned	bonus_spent	summ	summ_with_disc	doc_id
0	2021-07-13 12:56:09	2	2000200195023	51	0	3400	3400	15#2002741#65938#2_29
1	2021-07-30 10:42:00	2	2000200193494	57	0	747	747	15#2002972#65955#2_5
2	2021-10-11 12:55:23	2	2000200199106	92	253	3077	3077	15#2004060#66028#2_29
3	2021-10-14 14:48:56	2	2000200168768	1	0	54	54	15#2004107#66031#2_57
4	2021-10-20 11:09:39	2	2000200226314	101	0	1733	1733	15#2004192#66037#2_16

[7]:
```
# Checking that the filter has worked
df[~df.card.str.startswith('2000')].head()
```

[7]:

	datetime	shop	card	\
21	2021-07-12 08:09:44	1	de445db4-2fe3-4870-ac71-25f11b5174b9	
22	2021-07-12 08:26:45	1	70eccdfc-79dc-400c-8eed-1b864fe7170d	
26	2021-07-12 09:44:35	7	8323fcfb-6425-49e6-a86e-906c0609ce08	
30	2021-07-12 10:23:06	7	1629a910-ca86-45be-98c7-cb9822f838a2	
31	2021-07-12 10:28:42	4	cffa9b33-cd47-42e6-942a-83a561d8b74f	

	bonus_earned	bonus_spent	summ	summ_with_disc	doc_id
21	14	79	523	523	15#13002262#65937#13_1
22	14	0	567	567	15#13002262#65937#13_5
26	8	0	859	859	15#11006888#65937#11_11
30	5	17	190	190	15#11006888#65937#11_24
31	14	0	775	775	15#7001776#65937#7_22

[8]:
```
# Updating the dataframe
df = df[df.card.str.startswith('2000')]
```

[9]:
```
# Sorting the values for convenience
df = df.sort_values(['card','datetime'])
df.head()
```

[9]:

	datetime	shop	card	bonus_earned	bonus_spent	\
641	2021-07-19 09:37:20	2	2000200150015	0	1	
16455	2021-12-07 20:25:21	2	2000200150022	30	0	
8751	2021-10-05 16:31:25	2	2000200150053	15	0	
28718	2022-03-17 20:50:23	7	2000200150053	1	0	
834	2021-07-21 11:10:25	2	2000200150091	22	0	

	summ	summ_with_disc	doc_id
641	21	21	15#2002822#65944#2_6
16455	1351	1351	15#2004825#66085#2_140
8751	649	649	15#2003981#66022#2_65
28718	64	64	15#11002624#66185#11_177
834	746	746	15#2002857#65946#2_4

3 Data Wrangling

Frequency and monetary value are simpler to obtain - just counting each row for a given card number and just the sum of the sales to each card number.

[10]:
```
# Checking the last entry to use for recency
max(df.datetime)
```

[10]:
```
Timestamp('2022-06-09 21:49:45')
```

The last entry was made a long time ago, so it is better not to calculate how much time has passed since the last purhase of a client to compare recency. This is why the above date will be used as "today", that is to say that we assume the RFM analysis is being conducted on Sept 9th, 2022.

[11]:
```
df2 = df.groupby('card').agg(
    freq = ('card','count'),
    mone = ('summ_with_disc','sum'),
    rece = ('datetime','last')
).reset_index()
```

[12]:
```
df2.head()
```

[12]:

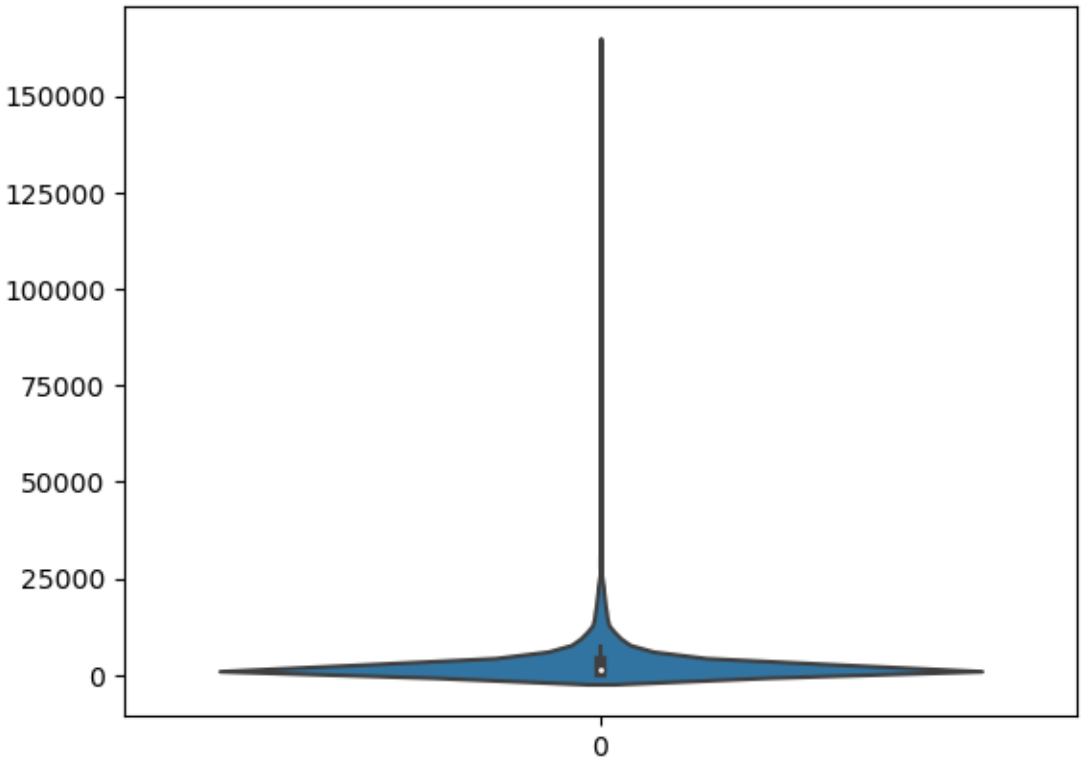
	card	freq	mone	rece
0	2000200150015	1	21	2021-07-19 09:37:20
1	2000200150022	1	1351	2021-12-07 20:25:21
2	2000200150053	2	713	2022-03-17 20:50:23
3	2000200150091	5	3549	2022-06-05 11:49:47
4	2000200150107	3	1735	2022-06-02 17:54:19

[13]:
```
# Calculating days since last purchase
df2['days'] = (max(df.datetime)-df2.rece).dt.days
df2.head()
```

[13]:

	card	freq	mone	rece	days
0	2000200150015	1	21	2021-07-19 09:37:20	325
1	2000200150022	1	1351	2021-12-07 20:25:21	184
2	2000200150053	2	713	2022-03-17 20:50:23	84
3	2000200150091	5	3549	2022-06-05 11:49:47	4
4	2000200150107	3	1735	2022-06-02 17:54:19	7

[14]:
```
# Looking at data distribution
sns.violinplot(df2['mone']);
```



There are outliers but since these are the clients that belong to one of the segments it should be best to keep them.

[15]:
```
# Creating percentiles to look at data distribution
percentiles = np.arange(0.1,1.1,0.1)
df2['mone'].quantile(percentiles)
```

[15]:
```
0.1       405.5
0.2       620.0
0.3       844.0
0.4      1103.0
0.5      1470.5
0.6      2008.0
0.7      2731.0
0.8      4078.0
0.9      6906.5
1.0    162687.0
Name: mone, dtype: float64
```

[16]:
```
df2['freq'].quantile(percentiles)
```

[16]:
```
0.1      1.0
0.2      1.0
0.3      1.0
0.4      1.0
0.5      2.0
0.6      2.0
0.7      3.0
0.8      5.0
0.9      8.0
1.0    217.0
Name: freq, dtype: float64
```

[17]:
```
df2['days'].quantile(percentiles)
```

[17]:
```
0.1      8.0
0.2     21.0
0.3     41.0
0.4     61.0
0.5     87.0
0.6    122.0
0.7    162.0
0.8    208.0
0.9    265.0
1.0    332.0
Name: days, dtype: float64
```

There may be various ways to divide the above into three groups. In this project a simple 33/66 divide will be used even though only recency satisfies this division and there might be less than 27 groups later on.

[18]:
```
new_percent = [0.33,0.66]
```

The values will be divided into three namely 1, 2 and 3. Where 1 is the best performance among all analyzed fields.

[19]:
```
# Defining a function that will assign scores
# Arguments - value of the field (val), field (var), 33rd and 66th percentiles␣
 ↪(perc_33,perc_66)
def set_score(val,var,perc_33,perc_66):
    if val < perc_33:
        return 3 if var != 'R' else 1 # R for recency
    elif val < perc_66:
        return 2
    else:
        return 1 if var != 'R' else 3
```

[20]:
```
# Assigning percentiles
rece_perc = df2['days'].quantile(new_percent)
```

```
freq_perc = df2['freq'].quantile(new_percent)
mone_perc = df2['mone'].quantile(new_percent)
```

[21]:
```
# Applying the aformentioned function for segmentation
df2['R'] = df2['days'].apply(set_score,args = ('R',rece_perc.iloc[0],rece_perc.
 ↪iloc[1]))
df2['F'] = df2['freq'].apply(set_score,args = ('F',freq_perc.iloc[0],freq_perc.
 ↪iloc[1]))
df2['M'] = df2['mone'].apply(set_score,args = ('M',mone_perc.iloc[0],mone_perc.
 ↪iloc[1]))
```

[22]:
```
# Having a look at the new dataframe
df2.head()
```

[22]:

	card	freq	mone	rece	days	R	F	M
0	2000200150015	1	21	2021-07-19 09:37:20	325	3	2	3
1	2000200150022	1	1351	2021-12-07 20:25:21	184	3	2	2
2	2000200150053	2	713	2022-03-17 20:50:23	84	2	2	3
3	2000200150091	5	3549	2022-06-05 11:49:47	4	1	1	1
4	2000200150107	3	1735	2022-06-02 17:54:19	7	1	1	2

[23]:
```
# Combining the three new fields into a new one
df2['RFM'] = df2.apply(lambda row: f"{row['R']}{row['F']}{row['M']}",axis=1)
df2.head()
```

[23]:

	card	freq	mone	rece	days	R	F	M	RFM
0	2000200150015	1	21	2021-07-19 09:37:20	325	3	2	3	323
1	2000200150022	1	1351	2021-12-07 20:25:21	184	3	2	2	322
2	2000200150053	2	713	2022-03-17 20:50:23	84	2	2	3	223
3	2000200150091	5	3549	2022-06-05 11:49:47	4	1	1	1	111
4	2000200150107	3	1735	2022-06-02 17:54:19	7	1	1	2	112

[24]:
```
# Counting clients in each group
df3 = df2.groupby('RFM')['RFM'].agg({'count'}).reset_index()
```

[25]:
```
# Looking at the final dataframe
df3
```

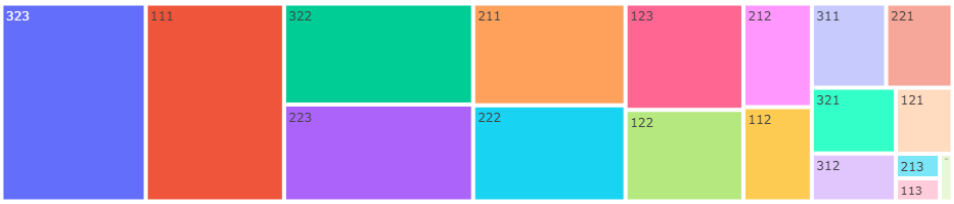
[25]:

	RFM	count
0	111	862
1	112	203
2	113	32
3	121	118
4	122	340
5	123	395
6	211	488
7	212	223

8	213	34
9	221	176
10	222	461
11	223	577
12	311	197
13	312	127
14	313	19
15	321	174
16	322	601
17	323	899

As can be seen there are less than 27 groups due to a simple division into groups using 33/66 percentiles as mentioned earlier. At the end, there are only 2 groups for frequency which is to be expected considering frequency's distribution (no values less than the 33rd percentile).

[26]:
```
# Visualizing the groups
px.treemap(
    df3,
    path=['RFM'],
    values='count'
)
```



4 Conclusions

As can be seen, the most numerous group is 323 (the poorest perfomers across the board) and the least numerous group is 313 (the poorest performers except for frequency).

The end goal of this kind of analysis might be to step the groups up for each lower category and also do not lose the best group. If using this as the goal, we might figure out ways like platinum cards to retain the 111 group since they are the best clients. Finding the right measure for this group might even trigger some word-of-mouth marketing.

A group that also deserves extra attention is 311. This group might include our lost fans and just one more sale to this group will make them the 111 group clients. We might offer the products they used to buy for free in order to reactivate the customers.

Groups like 222 might be made 112 fairly easily using the recurring bonuses with a deadline. Groups like 221 have a high monetary value - we might offer them a new exclusive product.

Overall for any group it might be best to work on frequency first so that our clients get used to buying our products and this might be done by suggesting the clients complementary products.