Section 4: RFM Analysis

In this lesson, the RFM analysis is applied to the dataset.

We'll cover the following

RFM analysis

RFM is a categorizing technique that uses the previous purchasing behavior of the customers to divide customers into groups so that an optimal marketing strategy can be developed for each individual. RFM stands for recency, frequency, and monetary, respectively.

- **Recency**: How many days have passed since a customer has bought an item
- **Frequency**: How many orders a customer has placed
- **Monetary**: How much money a customer has spent

Need for RFM analysis

- This technique efficiently categorizes the customers into specific rank-based groups taking into account their past online behaviors.
- This can help marketers and advertisers target each group of consumers separately, enabling them to cater to the needs of groups instead of each individual.
- This technique also informs us of the most and least profit yielding customers so relevant resources can be deployed to each group according to their needs.
- If the results of this technique are correctly used, then even customers who don’t engage in much activity(view, cart, buy) can be influenced to be high potential customers.

RFM technique and steps to perform

In this process, the customers are separated into four groups under each of the RFM metrics, i.e., *recency*, *frequency*, and *monetary*. This means we'll have a maximum of (4 x 4 x 4) **sixty-four** groups to deal with, which is not very large considering that the total number of customers can be in the thousands. **Quantiles** will be used to divide the customers into groups.

The following steps will be performed to get the final list of segmented customers.

Step 1: Get the purchase data of all customers.

```
import pandas as pd

df = pd.read_csv("2019-Oct.csv") # Reading the data from file

#Convert the type of event_time column to datetime
df['event_time'] = pd.to_datetime(df.event_time)

# Get rows with event_type equals purchase
purchased = df[df['event_type'] == 'purchase']

# Filter relevant data from Data Frame
purchased = purchased[['user_id', 'user_session', 'event_time' ,'price']]

print(purchased)
```

	user_id	user_session	event_time	price
162	543272936	8187d148-3c41-46d4-b0c0-9c08cd9dc564	2019-10-01 00:02:14+00:00	130.76
308	551377651	3c80f0d6-e9ec-4181-8c5c-837a30be2d68	2019-10-01 00:04:37+00:00	642.69
379	514591159	0e5dfc4b-2a55-43e6-8c05-97e1f07fbb56	2019-10-01 00:06:02+00:00	29.51
442	555332717	1dea3ee2-2ded-42e8-8e7a-4e2ad6ae942f	2019-10-01 00:07:07+00:00	54.42
574	524601178	2af9b570-0942-4dcd-8f25-4d84fba82553	2019-10-01 00:09:26+00:00	189.91
...	...	...	...	...
42448148	516604399	a98bd433-57ac-42d0-ba81-e18b135a7a16	2019-10-31 23:55:13+00:00	102.45
42448173	514622109	5724116e-365b-4ac1-9d03-b8d66e1ccc7c	2019-10-31 23:55:21+00:00	20.59
42448271	512717356	f35ac37c-9573-4e30-b3d9-c09bb0b95a2b	2019-10-31 23:56:03+00:00	577.89
42448362	533892594	3a5a3b01-2ab1-4a1d-a202-30d336e0057b	2019-10-31 23:56:53+00:00	1054.60
42448657	544501248	e330d051-37ad-4dc3-b1ee-ff16a28b7998	2019-10-31 23:59:16+00:00	160.57

742849 rows × 4 columns

On **line 9**, the rows from the `DataFrame` where `event_type` is assigned `purchase` value are selected.

value are selected.

On **line 12**, a subset of columns is selected to create a new `DataFrame`. This subset includes `user_id`, `user_session`, `event_time`, and `price`, as can be seen in the above output. These columns are selected based on their importance in computing the *R*, *F*, and *M* values.

Now, we can tell which customer placed how many orders of what price at what time. As we have filtered out all `purchase` event types from the data set, the number of `user_session` against a single `user_id` gives the number of orders an individual user placed. A `user_session` against a particular `user_id` could be repeated as a user might have made multiple purchases during a single visit, and a `user_session` could be different as well.

Step 2: Compute the *RFM* metrics for each customer.

```
# Compute the R, F, and M values for each user
rfm = purchased.groupby('user_id').agg({'event_time': lambda date: ((purchased['event_time'].max()
                                    'user_session': lambda num: num.count(),
                                    'price': lambda price: price.sum()})
print(rfm)


rfm['event_time'] = rfm['event_time'].apply(lambda days: int(str(days).split(' days')[0]) + 1)


rfm.columns=['recency','frequency','monetary']

print(rfm)
```

user_id	event_time	user_session	price
264649825	25 days 03:23:05	2	1240.04
303160429	18 days 10:33:47	1	340.59
340041246	17 days 09:13:37	4	915.52
371877401	22 days 08:08:08	1	29.89
384989212	13 days 12:39:17	1	41.44

user_id	recency	frequency	monetary
264649825	26	2	1240.04
303160429	19	1	340.59
340041246	18	4	915.52
371877401	23	1	29.89
384989212	14	1	41.44

On **lines 2-4**, a new DataFrame `rfm` is assigned the calculated values of *R*, *F*, and *M*.

- First, the `DataFrame` is grouped by the `user_id`. All values associated with a single `user_id` are combined in groups, and operations can be performed on each user's data individually. You can refresh the `groupby()` function from here.
- The `agg()` function allows performing different operations on each individual group. How the `agg()` function works can be seen here.
- `event_time` is used to calculate the *R* (recency) value. Using the `lambda()` function, the maximum `date` from each individual user group can be

selected. This `date` is then subtracted from the highest `date` in the `purchased`

DataFrame, which has all the product data. This way, we can get the number of days it's been since the last purchase of the user. Keep in mind that the highest date here is the one obtained from the dataset and not the current date.

- `user_session` is used to calculate the *F* (frequency) value, as mentioned above. The number of user sessions against a single `user_id`, whether repeated or different is the number of orders of that user. So, the `lambda()` function calculates the total count of those user sessions against a single `user_id` using the `count()` function.

- `price` is used to calculate the *M* (monetary) value. It is the total money spent by a single user on all the orders. Using the `lambda` function, the total money spent is calculated by summing all the `price` values against a single `user_id` using the `sum()` function.

The first output shows the result of this big operation. In the `event_time` column, there is some extra information that is not needed right now.

On **line 8**, again, a `lambda()` function is applied, which removes everything except the number of days and adds **1** to it to avoid **0** days.

On **line 11**, the column names of the `rfm` DataFrame are changed to `recency`, `frequency`, and `monetary.`

The result of these two operations can be viewed in the second output above. The above outputs are just a subset picture of the whole resultant `DataFrame`.

Step 3: Compute ranks for each *RFM* metric using quantiles.

Now that we have the correct *R*, *F*, and *M* values, it's time to rank them. It should be noted that in the case of recency, the lower the value the better, but for frequency and monetary, the higher the value the better. So, recency is inverse of frequency and monetary.

As mentioned above, each of the *RFM* values needs to be divided into four groups, so quantiles are used to categorize the *R*, *F*, and *M* values into correct groups. You can refresh the quantiles function here.

The **1st**, **2nd**, and **3rd** quantiles of the `recency`, `frequency`, and `monetary` columns are calculated from the `rfm` DataFrame and then converted to `dictionary` objects for easy access.

```
quantiles = rfm.quantile(q=[0.25,0.50,0.75])
quantiles = quantiles.to_dict()
print(quantiles)
```

```
{'recency': {0.25: 7.0, 0.5: 14.0, 0.75: 21.0},
 'frequency': {0.25: 1.0, 0.5: 1.0, 0.75: 2.0},
 'monetary': {0.25: 107.59, 0.5: 246.52, 0.75: 595.015}}
```

Now, for each *R*, *F*, and *M* metric in the `rfm` DataFrame, their values will be compared with their quantile values and will be assigned a rank between **1 and 4** based on the comparison. Here, **1** indicates the highest rank and **4** indicates the lowest rank.

The following functions will compute ranks for the *R*, *F*, and *M* values in the `rfm` DataFrame.

```
# Compute Ranks for Recency metric
def Compute_R(val,metric,quantile):
    if val <= quantile[metric][0.25]:
        return 1
    elif val <= quantile[metric][0.50]:
        return 2
    elif val <= quantile[metric][0.75]:
        return 3
    else:
        return 4

# Compute Ranks for Frequency & Monetary metrics
def Compute_FM(val,metric,quantile):
    if val <= quantile[metric][0.25]:
        return 4
    elif val <= quantile[metric][0.50]:
        return 3
    elif val <= quantile[metric][0.75]:
        return 2
    else:
        return 1
```

Both functions in the above code snippet have the same parameters and return values and are only different in their comparisons. As low values are better for `recency` and high values are better for `frequency` and `monetary`, two functions are created.

- The `val` parameter is the value of the *R*, *F*, or *M* metric from the `rf`

- The `val` parameter is the value of the *R*, *F*, or *M* metric from the `rfm` DataFrame that is compared by their respective quantile values.
- The `metric` parameter can be `recency`, `frequency` or `monetary` and is used to access the correct quantile value from the quantiles dictionary.
- The `quantile` parameter is the calculated quantiles dictionary which contains the **1st**, **2nd**, and **3rd** quantiles of the *R*, *F*, and *M* metrics from the `rfm` DataFrame.

The above functions compare the input values with the quantile values of the respective metrics and return ranks based on the mentioned conditions. For `recency`, the lower the value the higher the rank. For `frequency` and `monetary`, the higher the value the higher the rank.

```
# Compute new column with recency rank of that row
rfm['R_rank'] = rfm_new['recency'].apply(Compute_R, args=('recency',quartiles))

# Compute new column with frequency rank of that row
rfm['F_rank'] = rfm_new['frequency'].apply(Compute_FM, args=('frequency',quartiles))

# Compute new column with monetary rank of that row
rfm['M_rank'] = rfm_new['monetary'].apply(Compute_FM, args=('monetary',quartiles))

print(rfm)
```

user_id	recency	frequency	monetary	R_rank	F_rank	M_rank
264649825	26	2	1240.04	4	2	1
303160429	19	1	340.59	3	4	2
340041246	18	4	915.52	3	1	1
371877401	23	1	29.89	4	4	4
384989212	14	1	41.44	2	4	4
...	...	...	...	...	...	...
566270177	1	1	75.94	1	4	4
566272569	1	2	254.84	1	2	2
566274637	1	1	2011.63	1	4	1
566276996	1	1	74.39	1	4	4
566278294	1	1	1661.09	1	4	1

347118 rows × 7 columns

Three new rank columns are created for the *R*, *F*, and *M* metrics. The `apply()` function is used on each of the respective columns of the `rfm` DataFrame. The `Compute_R()` function is used for `recency`, and the `Compute_FM()` function is used for `frequency` and `monetary`. The second and third parameters of the functions are placed in the `args` parameter of the `apply()` function.

The above output displays the new resultant `DataFrame` with ranks of each of the *RFM* metrics.

Step 4: Combine the *RFM* values to obtain a combined RFM Score.

Now, the individual *R*, *F*, and *M* metrics are combined to generate the *RFM score* which is then added as a column in our `rfm` dataframe.

```
# Convert RFM values to type string
R = rfm.R_rank.astype(str)
F = rfm.F_rank.astype(str)
M = rfm.M_rank.astype(str)

# Compute new colum with combined RFM values
rfm['RFM_Score'] = R + F + M

print(rfm)
```

user_id	recency	frequency	monetary	R_rank	F_rank	M_rank	RFM_Score
264649825	26	2	1240.04	4	2	1	421
303160429	19	1	340.59	3	4	2	342
340041246	18	4	915.52	3	1	1	311
371877401	23	1	29.89	4	4	4	444
384989212	14	1	41.44	2	4	4	244
...	...	...	...	...	...	...	...
566270177	1	1	75.94	1	4	4	144
566272569	1	2	254.84	1	2	2	122
566274637	1	1	2011.63	1	4	1	141
566276996	1	1	74.39	1	4	4	144
566278294	1	1	1661.09	1	4	1	141

347118 rows × 7 columns

On **lines 2-4**, the *R*, *F*, and *M* values are converted to *string* using the `str` with `astype()` function.

On **line** 7, the new *RFM* ranks are concatenated to form a final `RFM_Score` column.

The above output displays the newly computed column.

Step 5: Sort the final `DataFrame` in ascending order.

Now, the final `DataFrame` is sorted in ascending order according to the **RFM Score** to get customer groups from best to worst.**

```
# Sort the DataFrame by RFM_Socre values
rfm = rfm.sort_values('RFM_Score')

print(rfm)
```

user_id	recency	frequency	monetary	R_rank	F_rank	M_rank	RFM_Score
521401518	4	10	7646.33	1	1	1	111
561251665	1	6	2029.17	1	1	1	111
524164850	5	5	3849.73	1	1	1	111
514028365	1	82	2971.36	1	1	1	111
558999650	6	22	4412.54	1	1	1	111
...	...	...	...	...	...	...	...
515637415	27	1	62.94	4	4	4	444
533542158	28	1	56.89	4	4	4	444
548352338	29	1	27.80	4	4	4	444
552263423	28	1	71.82	4	4	4	444
518935307	24	1	4.09	4	4	4	444

347118 rows × 7 columns

On **line 2**, the `rfm` DataFrame is now sorted in ascending order based on the `RFM_Score` column. In the above output, the final `DataFrame` is obtained with the customers sorted from having the highest *RFM* value to having the lowest *RFM*

value.

In the table below, the RFM score is mentioned with the corresponding customer group, what it means to have that RFM score, and what type of marketing strategy can be developed to deal with that group of customers. You should keep in mind that these groups are not a standard and can be adjusted to one's requirement or problem.

Customer Type	RFM Score	Explanation	Marketing Strategy
Best Customers	111	Bought most recently and frequently, and spends the most money	Introduce new products
Current Customer	1XX	Bought most recently	Upsell products related to current purchase
Loyal Customers	X1X	Bought most frequently	Use R and M metrics to further segment
Big Spenders	XX1	Spends the most money	Suggest costly products
Absent Customers	411	Purchased frequently and spent the most	Suggest products with discounts
Absent Common Customers	444	Purchased long ago, purchased few and spent little	Pay least amount of attention

		little	

> The `X` in the **RFM** score indicates that any value can occur, and it does not affect the result as long as the `1` stays in its position.

We just reduced the marketing load from managing thousands of customers to managing only *sixt-four* groups. Resources of the website can be allocated in the best possible way if the above information is correctly calculated and used.

This marks the end of our **second** and final data analysis project.