# Business Example: RFM Analysis in Python

This lesson will focus on how to do RFM analysis in Python with Pandas as an example of the usability of pandas.

WE'LL COVER THE FOLLOWING


- RFM Analysis
  - Recency
  - Frequency
  - Monetary
  - Putting it all together
  - Using quartiles to form classes
  - Combining classes into a number
  - Sort using `rfm` values
- Benefits of RFM analysis


In this chapter, we have seen how we can extract useful information from raw data very easily using *pandas* in Python. But we have only scratched the surface. A lot more can be done to obtain useful insights from the data. Professionals can use their domain expertise to perform different kinds of analysis on the data. In this lesson, we will explore a dataset from the perspective of a business or a marketing professional. We will be doing an *RFM analysis* of the Sample Sales Data that we have been using.

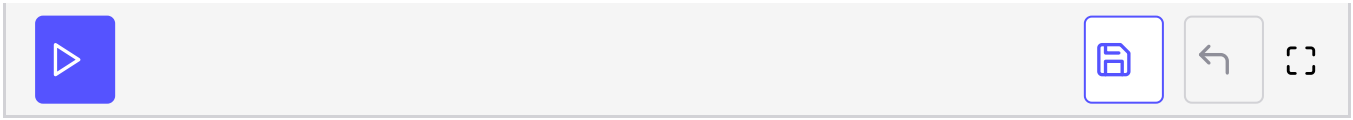
## RFM Analysis #

**RFM** (**R**ecency, **F**requency, **M**onetary) **analysis** is a marketing technique used to determine quantitatively which customers are the best ones by examining how recently a customer has purchased (recency), how often they purchase (frequency), and how much the customer spends (monetary). Using RFM analysis, customers are assigned a ranking number of $1, 2, 3, 4$ (with $4$ being highest) for each RFM parameter. The three scores together are referred to as

an RFM “cell”. The data is sorted to determine which customers were the **best customers** in the past, with a cell ranking of 444 being ideal.

So, let’s start coding.

```
import pandas as pd

# Read the data
df = pd.read_csv('sales_data.csv')

# Filter the data
cols_needed = ['CUSTOMERNAME','ORDERNUMBER','ORDERDATE','SALES']
df = df[cols_needed]
```



We read the data in **line 4**. Since we are doing RFM analysis, we will only need four columns. `CUSTOMERNAME` to group customers, `ORDERDATE` to calculate recency, `ORDERNUMBER` to calculate frequency, and `SALES` to calculate monetary. Therefore, we write these column names in a list in **line** 7 and filter the dataset in **line 8**.

## Recency #

**Recency** is a measure of how recently a customer has purchased a product. We find recency for every customer by finding out the date of their last order. Then we subtract this date from the date of the most recent order in the dataset to find the number of days between these two dates. This difference gives us the recency values. For a recent customer, this value will be smaller. For a customer who has not purchased a product for a long time, this value will be higher.

```
# Convert Date to Datetime type
df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])

# Calculate last order date for every customer
recent_dates = df.groupby('CUSTOMERNAME')['ORDERDATE'].max()

# Calculate the most recent order date in the dataset
most_recent = df['ORDERDATE'].max()

# Function to find difference between most recent date and last order date of every customer
def subtract_date(dt):
```

```
    difference = (most_recent  - dt).days
    return difference

# apply function to get the days since last order for every customer
recency = recent_dates.apply(subtract_date)
```

First of all, we change the data type of `ORDERDATE` to the `datetime` datatype of pandas in **line 2** so that we can easily compare dates. Then we find the date of the last order of every customer in **line 5**. We group the data by `CUSTOMERNAME`. Then we only retrieve the `ORDERDATE` column from the grouping. Then we use the `max` function which gives us the latest `ORDERDATE` for every customer. To find the most recent `ORDERDATE`, we get the `max` of the `ORDERDATE` column in **line 8**. We name it `most_recent`.

Our next step is to subtract the last `ORDERDATE` of every customer from the most recent date. Therefore, we define a function `subtract_date` in **line 11** that takes a date `dt`, find the difference in days between the `most_recent` and `dt` in **line 12** and returns the difference in **line 13**.

Afterward, we subtract `recent_dates` from `most_recent` in **line 16** by using the function that we defined. We use `apply` to apply `subtract_date` to `recent_dates`. The function `subtract_date` is called for every value in `recent_dates`. Each value is passed as an argument to the function one by one. The results are stored in `recency` in **line 16**.

## Frequency #

**Frequency** is a measure of how often a customer purchases a product. We will calculate the total number of times a customer has made an order.

Note that in the dataset, a single order denoted by an `ORDERNUMBER` can have orders on more than one product. Therefore, a single value of `ORDERNUMBER` can appear more than once for the same customer if that order had different kinds of products purchased. We want to calculate the number of times they have placed an order. So, if a single order has 3 different products, we will count it as a single order.

```
# Group by with both CUSTOMERNAME and ORDERNUMBeR
frequency = df.groupby(['CUSTOMERNAME','ORDERNUMBER']).size()
# Group by with CUSTOMERNAME
frequency = frequency.groupby('CUSTOMERNAME').size()
```

We calculate frequency in 2 steps:

1. We group by using `CUSTOMERNAME` and `ORDERNUMBER` and then use `size` as aggregation function in **line 2**. This gives us the number of times each `ORDERNUMBER` appeared for every customer. If we print head of `frequency` after **line 2**, we will get the following output:

```
CUSTOMERNAME    ORDERNUMBER
AV Stores, Co.  10110       16
                10306       17
                10332       18
Alpha Cognac    10136        3
                10178       12
dtype: int64
```

2. In the second step, we group the above table by `CUSTOMERNAME` and use `size` as the aggregation function in **line 4**. This gives us the number of times each customer appears in the above table, which is the number of times a customer has placed an order. If we print the head of `frequency` after **line 4**, we will get the following output:

```
CUSTOMERNAME
AV Stores, Co.          3
Alpha Cognac            3
Amica Models & Co.      2
Anna's Decorations, Ltd 4
Atelier graphique       3
dtype: int64
```

The values in this table will form our frequency values.

## Monetary #

## Monetary #

It is a measure of how much the customer has spent. It is the sum of all the sales made to the customer. It can be calculated by grouping the data by `CUSTOMERNAME` and summing the `SALES`.

```
monetary = df.groupby('CUSTOMERNAME')['SALES'].sum()
```

We use `groupby` to group by `CUSTOMERNAME`, then we select the `SALES` column, and use `sum` as the aggregation function. This gives us the sum of all sales made to the customer.

## Putting it all together #

Now we will combine all three `recency`, `frequency`, and `monetary` into a single table.

```
import pandas as pd

# Read the data
df = pd.read_csv('sales_data.csv')

# Filter the data
cols_needed = ['CUSTOMERNAME','ORDERNUMBER','ORDERDATE','SALES']
df = df[cols_needed]

# --------- RECENCY --------

df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
recent_dates = df.groupby('CUSTOMERNAME')['ORDERDATE'].max()
most_recent = df['ORDERDATE'].max()

def subtract_date(dt):
    date = (most_recent  - dt).days
    return date

recency = recent_dates.apply(subtract_date)


# --------- FREQUENCY -------

frequency = df.groupby(['CUSTOMERNAME','ORDERNUMBER']).size()
frequency = frequency.groupby('CUSTOMERNAME').size()

# --------- MONETARY --------

monetary = df.groupby('CUSTOMERNAME')['SALES'].sum()

# Combine all three into a table
rfm_table = pd.DataFrame()
rfm_table['recency'] = recency
```

```
rfm_table['frequency'] = frequency
rfm_table['monetary'] = monetary
print(rfm_table.head())
```

## Using quartiles to form classes #

Now that we have raw values for recency, frequency, and monetary, we can convert these values into classes. We will convert each value into a class based on which quartile it falls into.

First, we will obtain quartile values using the `quantile` function. Then we will assign the class values according to the quantile values. For instance, if the quartiles of `recency` are $(80, 125, 230)$, then a customer with a recency value of $70$ can be given a class value of `4` while a value of $120$ can be assigned a class value of `3` and so on. Let's implement this below.

```
quantile_df = rfm_table.quantile(q=[0.25,0.5,0.75])
```

This line will give us the quantiles of all three columns in a dataframe that we call `quantile_df`. The `quantile_df` will look like:

| | recency | frequency | monetary |
|---|---|---|---|
| 0.25 | 80.25 | 2.0 | 70129.4325 |
| 0.50 | 185.00 | 3.0 | 86522.6100 |
| 0.75 | 229.25 | 3.0 | 120575.8750 |

We can use this dataframe to assign classes to our variables. To accomplish that we define a function `quantile_classes` that we will apply to all three columns.

Recall that when we apply functions using `apply`, the function is called on every value of the column one by one. The applied functions expect only one argument, which is the value in the column. But the function we define below expects three arguments.

- The first argument, `x`, will be the value taken from the column.

- The second argument, `quantile_values`, will be the quantile values we calculated above.
- The third argument, `attribute`, will be the name of the column that we are applying the function on.

So, let's look at `quantile_classes` below.

```
def quantile_classes(x,quantile_values,attribute):
    # recency
    if attribute == 'recency':
        if x <= quantile_values.loc[0,attribute]:
            return '4'
        elif x >= quantile_values.loc[0,attribute] and x <= quantile_values.loc[1,attribute]:
            return '3'
        elif x >= quantile_values.loc[1,attribute] and x <= quantile_values.loc[2,attribute]:
            return '2'
        else:
            return '1'
    else:
        # Frequency and Monetary
        if x <= quantile_values.loc[0,attribute]:
            return '1'
        elif x >= quantile_values.loc[0,attribute] and x <= quantile_values.loc[1,attribute]:
            return '2'
        elif x >= quantile_values.loc[1,attribute] and x <= quantile_values.loc[2,attribute]:
            return '3'
        else:
            return '4'
```

The function checks in **line 3** if `attribute` is `recency` or not. If it is `recency` then the execution jumps to **line 4**. The function checks in what quantile the value `x` falls. It can access the quantile values using the `loc` keyword. `loc` can be provided the row name and the column name to access the quantile value. For the column name, it uses `attribute` since it contains the column name on which the function was applied.

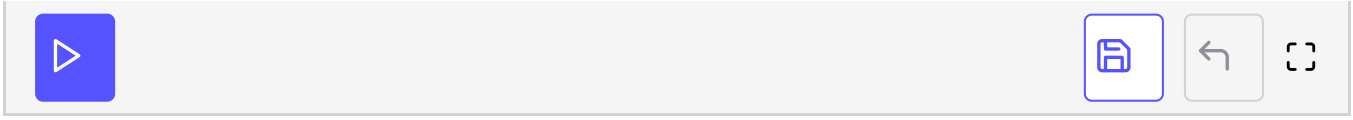
If the value is less than the 25th percentile, it is assigned the class value `4` in **line 5**. If it is between the 25th and 50th percentile, then it is assigned the class value `3` in **line 7**. If it falls between the 50th and 75th percentile, then it is assigned the class value `2` in **line 9**, and if it falls above the 75th percentile then it is assigned the class value `1` in **line 11**.

Now if the attribute is not `recency`, then the class value assigned to `x` for being less than the 25th percentile is `1` in **line 15**. In the same way, we assign class values `2`, `3`, and `4` in **lines 17,19, and 21** respectively.

This change in the order of assigning class values was made because low `recency` values mark the more recent customers and we want to assign them higher class values. But for `frequency` and `monetary`, higher values indicate top customers.

```
import pandas as pd

# Read the data
df = pd.read_csv('sales_data.csv')

# Filter the data
cols_needed = ['CUSTOMERNAME','ORDERNUMBER','ORDERDATE','SALES']
df = df[cols_needed]

# --------- RECENCY --------

df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
recent_dates = df.groupby('CUSTOMERNAME')['ORDERDATE'].max()
most_recent = df['ORDERDATE'].max()

def subtract_date(dt):
    date = (most_recent  - dt).days
    return date

recency = recent_dates.apply(subtract_date)


# --------- FREQUENCY -------

frequency = df.groupby(['CUSTOMERNAME','ORDERNUMBER']).size()
frequency = frequency.groupby('CUSTOMERNAME').size()

# --------- MONETARY --------

monetary = df.groupby('CUSTOMERNAME')['SALES'].sum()

# Combine all three into a table
rfm_table = pd.DataFrame()
rfm_table['recency'] = recency
rfm_table['frequency'] = frequency
rfm_table['monetary'] = monetary

# Function to Convert into classes using quartiles
def quantile_classes(x,quantile_values,attribute):
    # recency
    if attribute == 'recency':
        if x <= quantile_values.loc[0.25,attribute]:
            return '4'
        elif x >= quantile_values.loc[0.25,attribute] and x <= quantile_values.loc[0.5,attrib
            return '3'
        elif x >= quantile_values.loc[0.5,attribute] and x <= quantile_values.loc[0.75,attrib
            return '2'
        else:
            return '1'
    else:
```

```
        # Frequency and Monetary
        if x <= quantile_values.loc[0.25,attribute]:
            return '1'
        elif x >= quantile_values.loc[0.25,attribute] and x <= quantile_values.loc[0.5,attrib
            return '2'
        elif x >= quantile_values.loc[0.5,attribute] and x <= quantile_values.loc[0.75,attrib
            return '3'
        else:
            return '4'

# Caculate quantiles
quantile_df = rfm_table.quantile(q=[0.25,0.5,0.75])

# COnvert to classes using quantiles
rfm_table['r_class'] = rfm_table['recency'].apply(quantile_classes,args = (quantile_df,'recen
rfm_table['f_class'] = rfm_table['frequency'].apply(quantile_classes,args = (quantile_df,'fre
rfm_table['m_class'] = rfm_table['monetary'].apply(quantile_classes,args = (quantile_df,'mone
print(rfm_table.head())
```



We write our function in **lines 39-59**. Then we calculate the quantiles in **line 62**. To convert `recency` into classes, we use the `apply` function on the `recency` column in `rfm_table` in **line 65**. The `apply` function is first given the name of the function. Then it is given the extra arguments that the `quantile_classes` expects in parenthesis as `args`. For `recency`, the arguments are `quantile_df` and `recency`. The output is stored as a new column named `r_class`.

We do the same for `frequency` and `monetary` in **lines 66 and 67**, respectively. By looking at the output of **line 68**, we can see the classes assigned to each customer.

## Combining classes into a number #

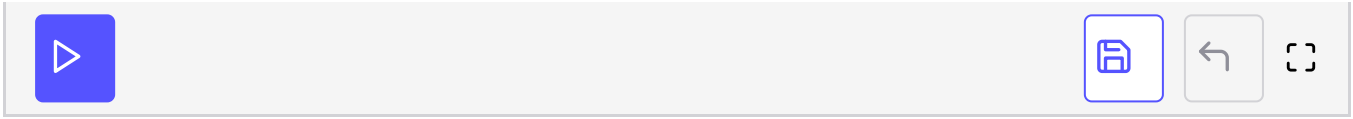
Now that we have assigned separate classes to all customers for `recency`, `frequency`, and `monetary`, we can combine these three separate classes into a single class. For instance, if a customer has the values `2`, `3`, `4` for `recency`, `frequency`, and `monetary`, respectively, the combined value will be `234`.

Since the columns `r_class`, `f_class`, and `m_class` are of the string data type, we can join the values by using the `+` operator. For strings, `+` joins them together to form a single string. For instance,

```
import pandas as pd
```

```
# Read the data
df = pd.read_csv('sales_data.csv')


# Filter the data
cols_needed = ['CUSTOMERNAME','ORDERNUMBER','ORDERDATE','SALES']
df = df[cols_needed]

# --------- RECENCY --------

df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
recent_dates = df.groupby('CUSTOMERNAME')['ORDERDATE'].max()
most_recent = df['ORDERDATE'].max()

def subtract_date(dt):
    date = (most_recent  - dt).days
    return date

recency = recent_dates.apply(subtract_date)


# --------- FREQUENCY -------

frequency = df.groupby(['CUSTOMERNAME','ORDERNUMBER']).size()
frequency = frequency.groupby('CUSTOMERNAME').size()

# --------- MONETARY --------

monetary = df.groupby('CUSTOMERNAME')['SALES'].sum()

# Combine all three into a table
rfm_table = pd.DataFrame()
rfm_table['recency'] = recency
rfm_table['frequency'] = frequency
rfm_table['monetary'] = monetary

# Function to Convert into classes using quartiles
def quantile_classes(x,quantile_values,attribute):
    # recency
    if attribute == 'recency':
        if x <= quantile_values.loc[0.25,attribute]:
            return '4'
        elif x >= quantile_values.loc[0.25,attribute] and x <= quantile_values.loc[0.5,attrib
            return '3'
        elif x >= quantile_values.loc[0.5,attribute] and x <= quantile_values.loc[0.75,attrib
            return '2'
        else:
            return '1'
    else:
        # Frequency and Monetary
        if x <= quantile_values.loc[0.25,attribute]:
            return '1'
        elif x >= quantile_values.loc[0.25,attribute] and x <= quantile_values.loc[0.5,attrib
            return '2'
        elif x >= quantile_values.loc[0.5,attribute] and x <= quantile_values.loc[0.75,attrib
            return '3'
        else:
            return '4'

# Caculate quantiles
quantile_df = rfm_table.quantile(q=[0.25,0.5,0.75])
```

```
# COnvert to classes using quantiles
rfm_table['r_class'] = rfm_table['recency'].apply(quantile_classes,args = (quantile_df,'recen
rfm_table['f_class'] = rfm_table['frequency'].apply(quantile_classes,args = (quantile_df,'fre
rfm_table['m_class'] = rfm_table['monetary'].apply(quantile_classes,args = (quantile_df,'mone

# Combine classes
rfm_table['rfm'] = rfm_table['r_class'] + rfm_table['f_class'] + rfm_table['m_class']
print(rfm_table['rfm'].head())
```



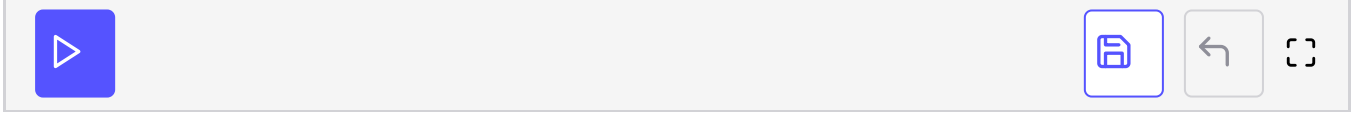
In **line 70**, we add a new column `rfm` to `rfm_table` that will store the combined classes. We combine the three columns using the `+` operator. We can see the output of **line 71**, to verify.

## Sort using `rfm` values #

To find the top customers we can sort the dataframe using `rfm` values. But there is an issue with sorting. The `rfm` column is of string type and we can only sort numerical values. Therefore, we need to convert them to integer values. To do this, we can use the function `to_numeric` available in pandas.

```
import pandas as pd

# Read the data
df = pd.read_csv('sales_data.csv')

# Filter the data
cols_needed = ['CUSTOMERNAME','ORDERNUMBER','ORDERDATE','SALES']
df = df[cols_needed]

# --------- RECENCY --------

df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
recent_dates = df.groupby('CUSTOMERNAME')['ORDERDATE'].max()
most_recent = df['ORDERDATE'].max()

def subtract_date(dt):
    date = (most_recent  - dt).days
    return date

recency = recent_dates.apply(subtract_date)


# --------- FREQUENCY -------

frequency = df.groupby(['CUSTOMERNAME','ORDERNUMBER']).size()
frequency = frequency.groupby('CUSTOMERNAME').size()

# --------- MONETARY --------
```

```
monetary = df.groupby('CUSTOMERNAME')['SALES'].sum()


# Combine all three into a table
rfm_table = pd.DataFrame()
rfm_table['recency'] = recency
rfm_table['frequency'] = frequency
rfm_table['monetary'] = monetary

# Function to Convert into classes using quartiles
def quantile_classes(x,quantile_values,attribute):
    # recency
    if attribute == 'recency':
        if x <= quantile_values.loc[0.25,attribute]:
            return '4'
        elif x >= quantile_values.loc[0.25,attribute] and x <= quantile_values.loc[0.5,attrib
            return '3'
        elif x >= quantile_values.loc[0.5,attribute] and x <= quantile_values.loc[0.75,attrib
            return '2'
        else:
            return '1'
    else:
        # Frequency and Monetary
        if x <= quantile_values.loc[0.25,attribute]:
            return '1'
        elif x >= quantile_values.loc[0.25,attribute] and x <= quantile_values.loc[0.5,attrib
            return '2'
        elif x >= quantile_values.loc[0.5,attribute] and x <= quantile_values.loc[0.75,attrib
            return '3'
        else:
            return '4'

# Caculate quantiles
quantile_df = rfm_table.quantile(q=[0.25,0.5,0.75])

# COnvert to classes using quantiles
rfm_table['r_class'] = rfm_table['recency'].apply(quantile_classes,args = (quantile_df,'recen
rfm_table['f_class'] = rfm_table['frequency'].apply(quantile_classes,args = (quantile_df,'fre
rfm_table['m_class'] = rfm_table['monetary'].apply(quantile_classes,args = (quantile_df,'mone

# Combine classes
rfm_table['rfm'] = rfm_table['r_class'] + rfm_table['f_class'] + rfm_table['m_class']

# Convert to Integers
rfm_table['rfm'] = pd.to_numeric(rfm_table['rfm'])
rfm_table = rfm_table.sort_values(by = 'rfm',ascending=False)
print(rfm_table['rfm'].head(20))
```



We use `pd.to_numeric` in **line 73**, to convert the `rfm` values into numerical values. Then we sort the dataframe using the function `sort_values` in **line 74**. We tell the function to sort the dataframe using the column `rfm` by giving it as the argument `by`. We give `ascending=False` to sort in descending order. We

can see the top 20 customers from the output of **line 75**.

## Benefits of RFM analysis #

Now that we have arranged our customers according to the `rfm` values, we can answer a lot of questions such as:

- Who are the top 10 customers?
- Which customers are close to churning?
- Who are the most loyal customers?
- Who are the customers that buy the most?
- Which group of customers do we need to retain?
- On which customers do we need to focus more?

All of these and many more questions can be answered by exploring the dataset using the `recency`, `frequency`, and `monetary` values we calculated. We can filter the dataset using different conditions to answer these questions.

This was an example of using Python for performing a business technique easily. People from other domains can apply their expertise to perform any kind of task on the data to gain more insights.

This marks the end of this chapter. In this chapter, we looked at how to find certain trends and patterns in the data. In the next chapter, we will look at how we can verify our findings.