INTERNATIONAL UNIVERSITY – VNU HCM

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

INTRODUCTION TO DATA MINING PROJECT

PROJECT REPORT

Group 07

Lecturer:

PhD Nguyen Thi Thanh Sang

Members:

Kieu Chi Huy – ITDSIU19004

Phan Hung Thinh – ITDSIU19055

Nguyen Dinh Thong – ITDSIU19018

Ly Minh Trung – ITDSIU19023

Abstract

Data mining is a powerful new technique that can help businesses focus on the most critical information in the data they have gathered about their customers and potential customers' behavior. It uncovers information in the data that queries, and reports cannot expose. There are many applications of data mining techniques in daily life but one of the vital is in market basket analysis. This is a data mining technique used by retailers to increase sales by better understanding customer purchasing patterns. It involves analyzing large data sets, such as purchase history, to reveal product groupings, as well as products that are likely to be bought together. Using the k-means and FP-growth approaches, our group will create a data mining framework to handle an e-commercial data set to cluster the customer and discover the frequent itemset in this project.

Table of Contents

I. **Overview** 4

1) **Summary of our work:** 4

2) **Objective:** 4

II. **Algorithm** 5

1. **Clustering concept** 5

2. **K-means clustering algorithm** 5

3. **FP-growth association rule algorithm** 5

III. **Pre-processing** 6

1) **Missing Value Imputation** 6

2) **RFM Analysis** 9

IV. **Implement a clustering/classification algorithm** 12

V. **Implement a sequence mining algorithm (FPGrowth)** 17

VI. **Model Evaluation** 20

VII. **Conclusion** 22

VIII. **References** 23

References 23

I. Overview

1) Summary of our work:

We will create a data mining framework from the ground up in this project. One clustering/classification approach and one sequence mining method are included in this framework. The following resources make up the dataset:

https://www.kaggle.com/datasets/carrie1/ecommerce-data

Our system be able to construct a dataset of event sequences, develop a clustering/classification process, and a sequence mining process as a prediction engine that will be used to forecast next occurrences in the application domain, given a certain training dataset in a specific data format.

To finish this project, we have completed the following four phases:

- Step 1: Create training and testing datasets by extracting sequential datasets of events and associated characteristics based on timestamps.
- Step 2: Apply a clustering/classification method to events in the sequential datasets, such as K-Means.
- Step 3: Create a prediction model based on the sequences in the (clustered/classified) training datasets using a sequence mining method (or association rules), such as FPGrowth.
- Step 4: Compare and contrast the prediction model with and without clustering/classification. Assess its efficiency and remark on the drawbacks of our work.

2) Objective:

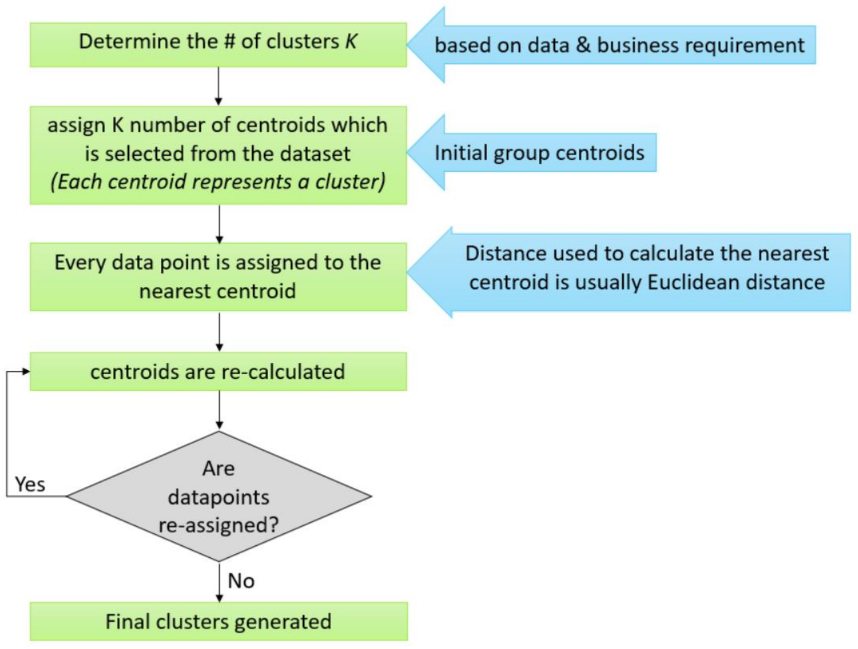
- 1. Gaining an understanding of the data mining method
- 2. Improve our machine learning algorithm implementation skills.
- 3. Improve our data processing and transformation abilities.
- 4. Improve our ability to evaluate data mining approaches.

II. Algorithm

1. Clustering concept

Clustering is an "unsupervised learning" strategy in general. That suggests there is no target variable. We are simply allowing the patterns in the data to form. The goal of clustering is not to forecast the target class as in classification, but to group similar objects by considering the most fulfilled condition: all items in the same group should be similar, and no two items from separate groups should not be similar. Different similarity metrics might be used to group comparable objects in clustering.

2. K-means clustering algorithm


Determine the # of clusters *K*
based on data & business requirement
assign K number of centroids which is selected from the dataset *(Each centroid represents a cluster)*
Initial group centroids
Every data point is assigned to the nearest centroid
Distance used to calculate the nearest centroid is usually Euclidean distance
centroids are re-calculated
Are datapoints re-assigned?
Yes
No
Final clusters generated

One of the most basic and often used unsupervised machine learning techniques is K-means clustering. The k-means clustering technique can be simply defined as follows:

- Initialize K random centroids: You'll specify a target number, k, for the number of centroids required in the dataset.
- Examine which centroids are closest to each data point: by using some sort of measurement like Euclidean distance
- Assign the data point to the centroid that is closest to it.
- For every centroid, move the centroid to the average of the points assigned to that centroid
- Repeat the process until the centroid assignment no longer changes

3. FP-growth association rule algorithm

The purpose of Frequent Itemset Mining is to use a fast and efficient method to detect often occurring product pairings. Different algorithms exist for this. The Apriori algorithm is one of the core algorithms. The FP Growth algorithm is a contemporary version of Apriori, as it is faster and more efficient while achieving the same goal.

The FP Growth algorithm's goal is to locate the most frequent itemsets in a dataset quicker than the Apriori technique. The Apriori method loops back and forth across the data set looking for product co-occurrences, but the FP approach reorganized the data into a tree rather than sets. The algorithm saves time because of its tree data structure, which allows for quicker scanning. We can briefly summarize this algorithm in these steps:

- Step 1: the algorithm counts the occurrences of items (attribute-value pairs) in the dataset of transactions and stores these counts in a 'header table'.
- Step 2: it builds the FP-tree structure by inserting transactions into a tree. Before being entered, items in each transaction must be sorted in decreasing order of their frequency in the dataset so that the tree may be processed fast. Items that do not fulfill the minimum support criterion in each transaction are eliminated. The FP-tree provides strong compression around the tree root when several transactions share the most frequent elements.
- Step 3: A new conditional tree is created which is the original FP-tree projected onto I. The supports of all nodes in the projected tree are re-counted with each node getting the sum of its children counts. Nodes (and hence subtrees) that do not meet the minimum support are pruned. Recursive growth ends when no individual items are conditional on I meet the minimum support threshold. The resulting paths from the root to I will be frequent itemsets. After this step, processing continues with the next least-supported header item of the original FP-tree. Once the recursive process has completed, all frequent itemsets will have been found, and association rule creation begins

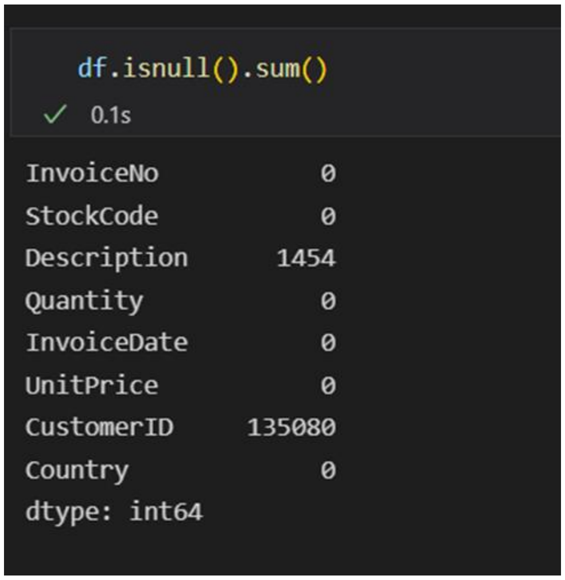
III. Pre-processing

Data preparation, also known as data pre-processing, is primarily concerned with two issues: first, the data must be organized in a suitable format for data mining algorithms, and second, the data sets used must provide the best performance and quality for the models generated by data mining operations.

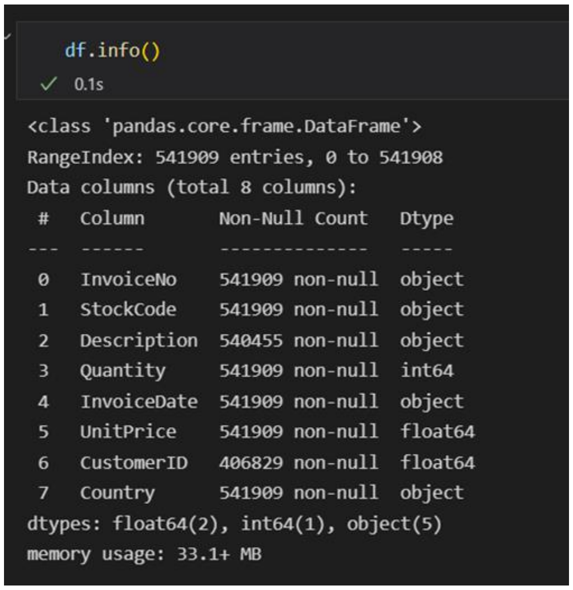
Data in the real world is frequently incomplete, noisy, and inconsistent. This can result in poor data quality and, as a result, poor model quality based on that data.

1) Missing Value Imputation

After loading data, we check information of data to find the answers for question: Is any data missing? How much data is missing?



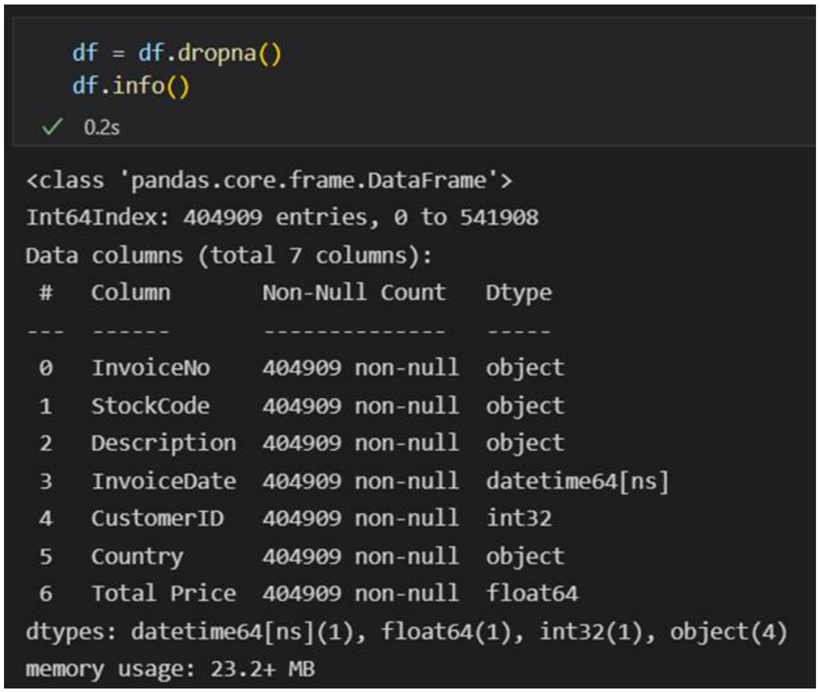
```
df.isnull().sum()
✓ 0.1s

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



```
df.info()
✓ 0.1s

<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  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

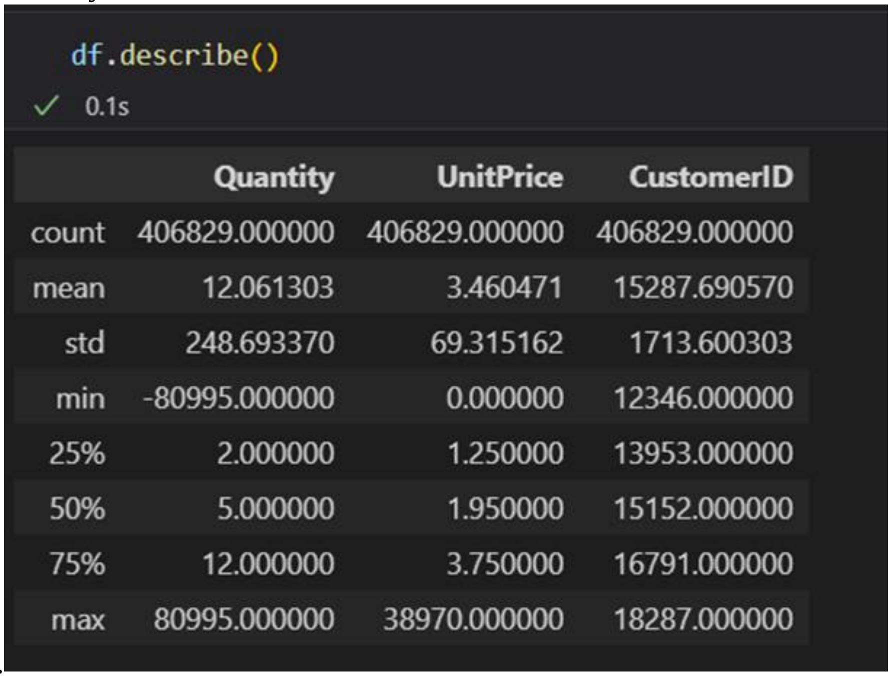
To address these issues, when creating operational data, there are two common approaches to dealing with missing values. The first option is to simply remove data samples with missing values because most data mining algorithms are unable to handle missing values. Only when the fraction of missing information is insignificant can this strategy be used. The second option is to use missing value imputation algorithms to fill in blanks with inferred values.

In this case, Customer ID and Description is a peculiarity data, cannot use missing value imputation algorithms to fill in blanks with inferred values. We must use the first option to remove data with missing values.



```
df = df.dropna()
df.info()
✓ 0.2s

<class 'pandas.core.frame.DataFrame'>
Int64Index: 404909 entries, 0 to 541908
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   InvoiceNo    404909 non-null  object        
 1   StockCode    404909 non-null  object        
 2   Description  404909 non-null  object        
 3   InvoiceDate  404909 non-null  datetime64[ns]
 4   CustomerID   404909 non-null  int32         
 5   Country      404909 non-null  object        
 6   Total Price  404909 non-null  float64       
dtypes: datetime64[ns](1), float64(1), int32(1), object(4)
memory usage: 23.2+ MB
```

Then, we use describe() function to show details of data, make sure that there is nothing out of the ordinary in this data set



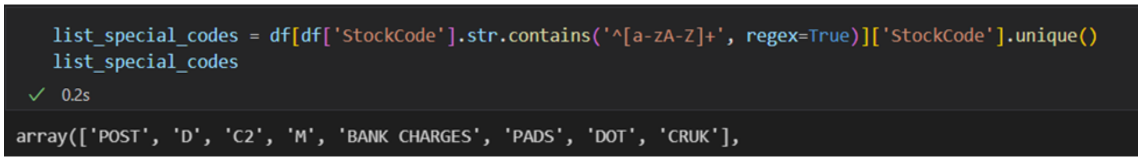
```
df.describe()
✓ 0.1s
```

```
            Quantity       UnitPrice     CustomerID
count  406829.000000   406829.000000  406829.000000
mean       12.061303        3.460471   15287.690570
std       248.693370       69.315162    1713.600303
min    -80995.000000        0.000000   12346.000000
25%         2.000000        1.250000   13953.000000
50%         5.000000        1.950000   15152.000000
75%        12.000000        3.750000   16791.000000
max     80995.000000    38970.000000   18287.000000
```

.

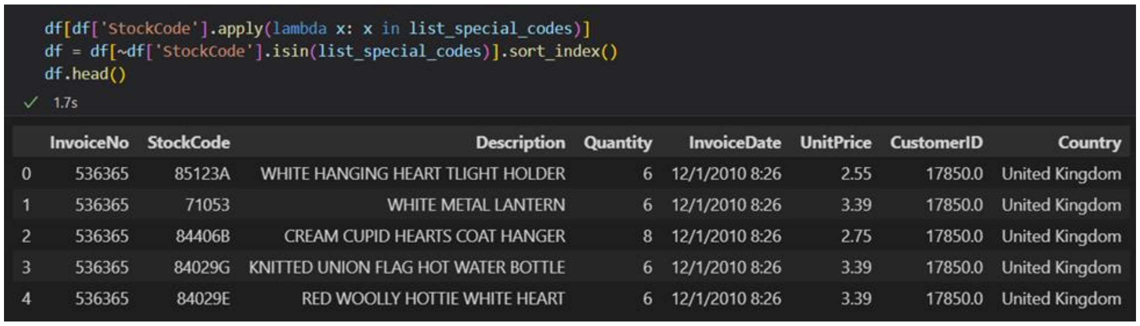
At Quantity and Unit Price, the minimum of these 2 columns has something wrong. It is about negative Quantity, and the Unit Price is equal to zero. We have 2 hypotheses. The first hypothesis is that the customer may have canceled the order during the purchase process, resulting in the statistics recording a negative value in the Quantity column. Unit Price is equal to 0 maybe because customers buy products with promotions at that time and are included. The second hypothesis is that these data are wrong and unreasonable.

After examining the data, we realized that some characters have been added to note special items in the Stock Code column. The first hypothesis was right, so we screened and removed these data to ensure the integrity of the data in terms of revenue.



```
list_special_codes = df[df['StockCode'].str.contains('^[a-zA-Z]+', regex=True)]['StockCode'].unique()
list_special_codes
✓ 0.2s
array(['POST', 'D', 'C2', 'M', 'BANK CHARGES', 'PADS', 'DOT', 'CRUK'],
```

POST	->	POSTAGE
D	->	DISCOUNT
C2	->	CARRIAGE
M	->	MANUAL
BANK CHARGES	->	BANK CHARGES
PADS	->	PADS TO MATCH ALL CUSHIONS
DOT	->	DOTCOM POSTAGE

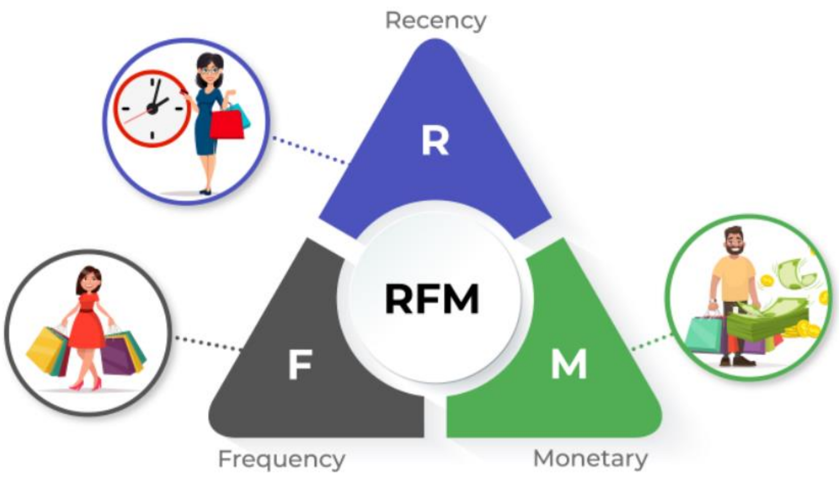


```
df[df['StockCode'].apply(lambda x: x in list_special_codes)]
df = df[~df['StockCode'].isin(list_special_codes)].sort_index()
df.head()
```

✓ 1.7s

```
   InvoiceNo StockCode                        Description  Quantity     InvoiceDate  UnitPrice  CustomerID         Country
0     536365    85123A   WHITE HANGING HEART TLIGHT HOLDER         6  12/1/2010 8:26       2.55     17850.0  United Kingdom
1     536365     71053                 WHITE METAL LANTERN         6  12/1/2010 8:26       3.39     17850.0  United Kingdom
2     536365    84406B      CREAM CUPID HEARTS COAT HANGER         8  12/1/2010 8:26       2.75     17850.0  United Kingdom
3     536365    84029G KNITTED UNION FLAG HOT WATER BOTTLE         6  12/1/2010 8:26       3.39     17850.0  United Kingdom
4     536365    84029E      RED WOOLLY HOTTIE WHITE HEART          6  12/1/2010 8:26       3.39     17850.0  United Kingdom
```

2) RFM Analysis

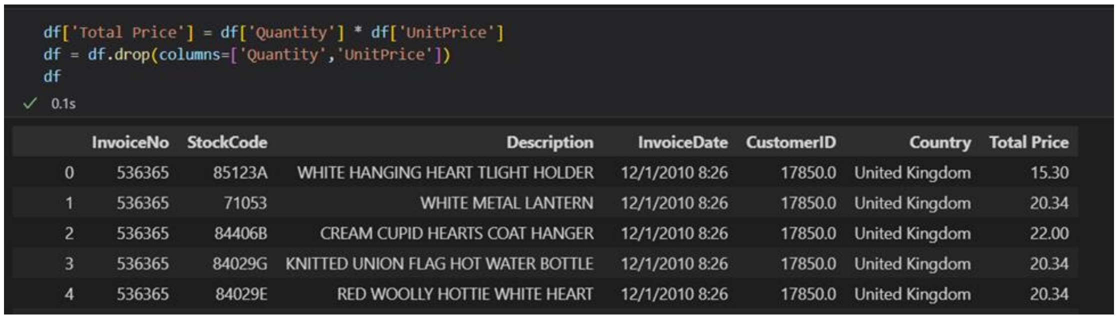


RFM analysis is a marketing approach that ranks, and groups clients statistically based on the recency, frequency, and monetary amount of their recent transactions to find the best customers and perform marketing campaigns.

The following factors are used in RFM analysis to rate each customer:

- Recency: what was the customer's latest recent purchase? Customers who have just made a purchase will remember the product and are more inclined to purchase or use it again.
- Frequency: when was the last time this consumer bought something? Customers who have previously purchased are more likely to do so again.
- Monetary: how much did the consumer spend in a specific time? Customers that spend a lot of money are more likely to spend money again in the future and are valuable to a company.

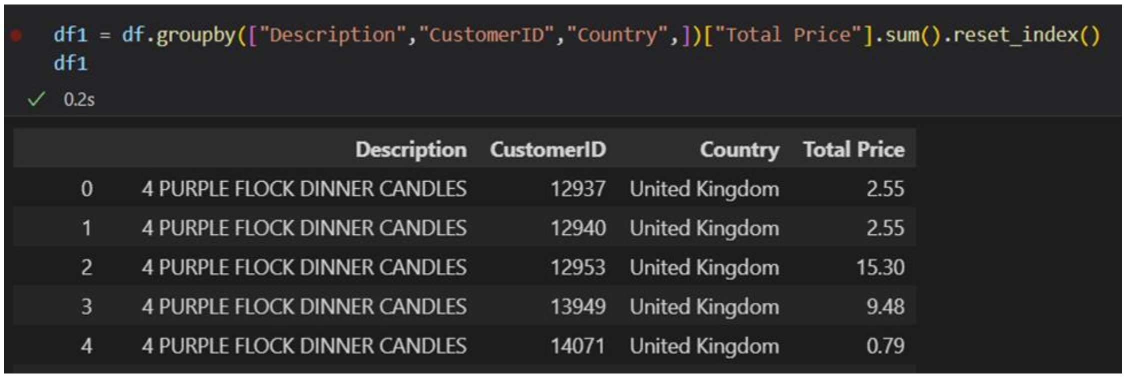
To apply RFM Analysis, we need to calculate total price products. Then group them based on Customer ID and Description.



```python
df['Total Price'] = df['Quantity'] * df['UnitPrice']
df = df.drop(columns=['Quantity','UnitPrice'])
df
```

✓ 0.1s

	InvoiceNo	StockCode	Description	InvoiceDate	CustomerID	Country	Total Price
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	12/1/2010 8:26	17850.0	United Kingdom	15.30
1	536365	71053	WHITE METAL LANTERN	12/1/2010 8:26	17850.0	United Kingdom	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	12/1/2010 8:26	17850.0	United Kingdom	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	12/1/2010 8:26	17850.0	United Kingdom	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART	12/1/2010 8:26	17850.0	United Kingdom	20.34

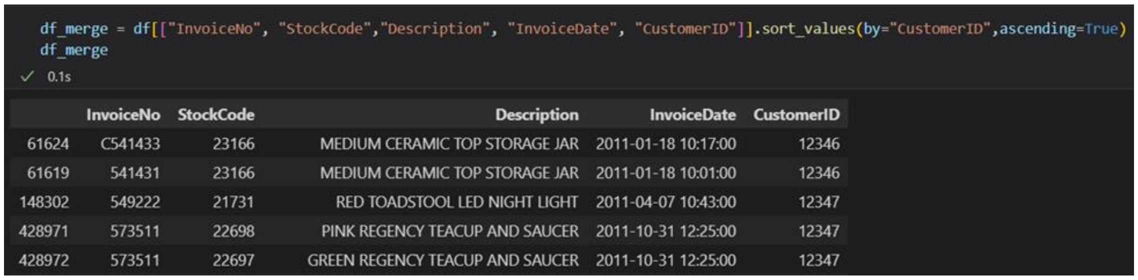


```python
df1 = df.groupby(["Description","CustomerID","Country",])["Total Price"].sum().reset_index()
df1
```

✓ 0.2s

	Description	CustomerID	Country	Total Price
0	4 PURPLE FLOCK DINNER CANDLES	12937	United Kingdom	2.55
1	4 PURPLE FLOCK DINNER CANDLES	12940	United Kingdom	2.55
2	4 PURPLE FLOCK DINNER CANDLES	12953	United Kingdom	15.30
3	4 PURPLE FLOCK DINNER CANDLES	13949	United Kingdom	9.48
4	4 PURPLE FLOCK DINNER CANDLES	14071	United Kingdom	0.79

When calculating the total price, we can only be based on Customer ID and Description, not Invoice No. Because of a purchase for each product, there will be a corresponding Invoice No. That is why we must create another data frame to save the Invoice No and Invoice Date so we can merge them with the original data for RFM analysis.

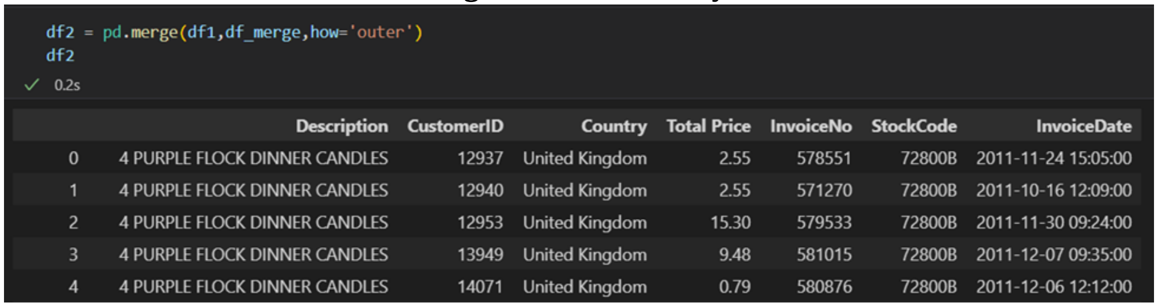


```python
df_merge = df[["InvoiceNo", "StockCode","Description", "InvoiceDate", "CustomerID"]].sort_values(by="CustomerID",ascending=True)
df_merge
```

✓ 0.1s

	InvoiceNo	StockCode	Description	InvoiceDate	CustomerID
61624	C541433	23166	MEDIUM CERAMIC TOP STORAGE JAR	2011-01-18 10:17:00	12346
61619	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	2011-01-18 10:01:00	12346
148302	549222	21731	RED TOADSTOOL LED NIGHT LIGHT	2011-04-07 10:43:00	12347
428971	573511	22698	PINK REGENCY TEACUP AND SAUCER	2011-10-31 12:25:00	12347
428972	573511	22697	GREEN REGENCY TEACUP AND SAUCER	2011-10-31 12:25:00	12347

Data after merger for RFM Analysis as below:

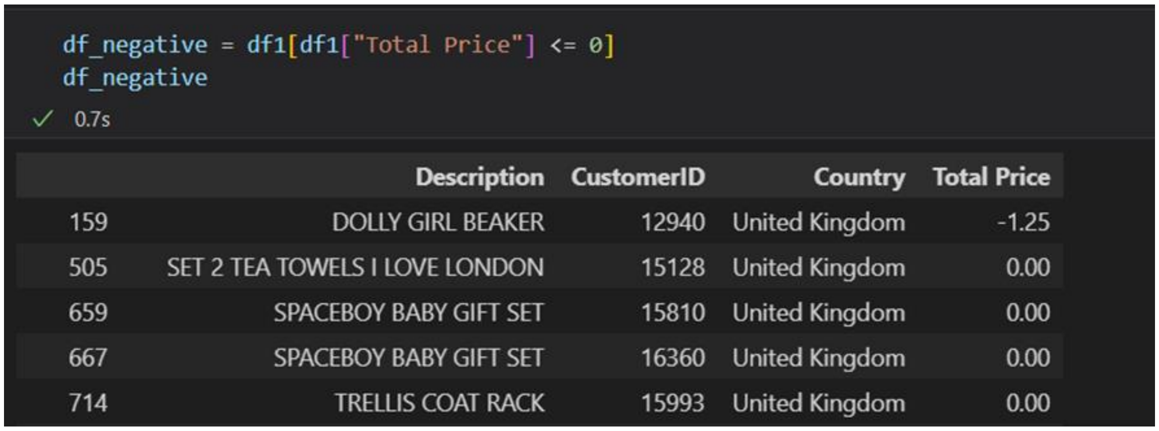


```python
df2 = pd.merge(df1,df_merge,how='outer')
df2
```

✓ 0.2s

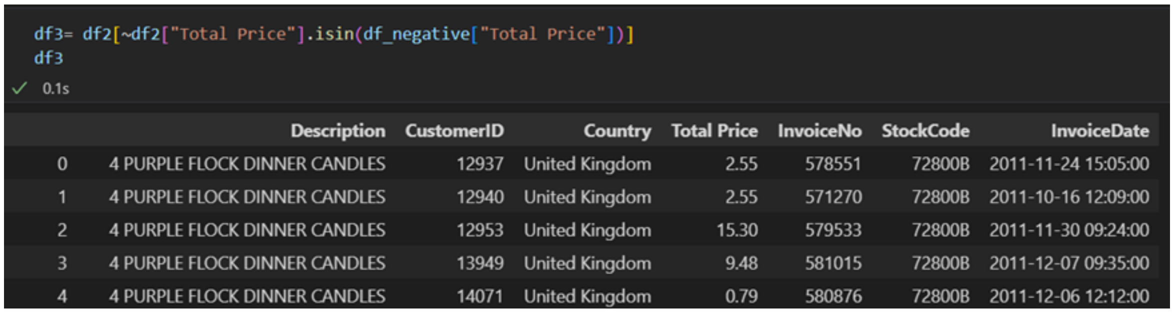
	Description	CustomerID	Country	Total Price	InvoiceNo	StockCode	InvoiceDate
0	4 PURPLE FLOCK DINNER CANDLES	12937	United Kingdom	2.55	578551	72800B	2011-11-24 15:05:00
1	4 PURPLE FLOCK DINNER CANDLES	12940	United Kingdom	2.55	571270	72800B	2011-10-16 12:09:00
2	4 PURPLE FLOCK DINNER CANDLES	12953	United Kingdom	15.30	579533	72800B	2011-11-30 09:24:00
3	4 PURPLE FLOCK DINNER CANDLES	13949	United Kingdom	9.48	581015	72800B	2011-12-07 09:35:00
4	4 PURPLE FLOCK DINNER CANDLES	14071	United Kingdom	0.79	580876	72800B	2011-12-06 12:12:00

Back to our first hypothesis, the negative Quantity (customer cancels the order) problem is still unresolved. It means that the transactions when the customer orders must also be processed or dropped according to the canceled transactions. This is to ensure that revenue statistics will not be affected by those transactions. Then, we created a new data frame named df_negative to save all data with a Total Price is less than 0. Based on that, we drop all data from df_negative.



```
df_negative = df1[df1["Total Price"] <= 0]
df_negative
✓ 0.7s
```

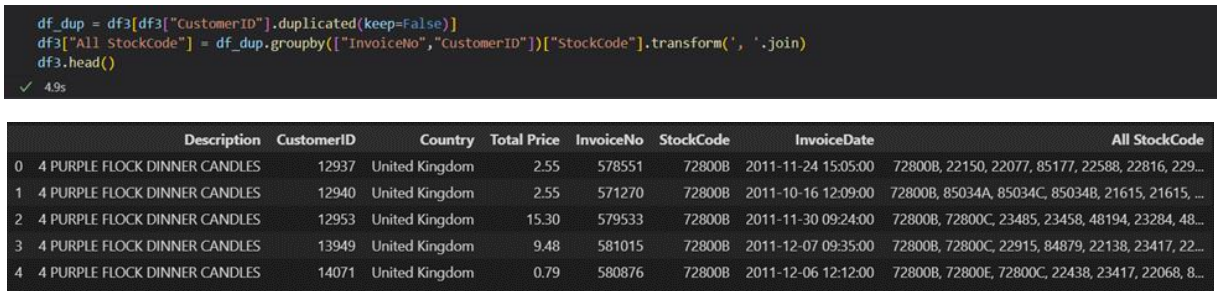
```
                     Description  CustomerID         Country  Total Price
159            DOLLY GIRL BEAKER       12940  United Kingdom        -1.25
505  SET 2 TEA TOWELS I LOVE LONDON    15128  United Kingdom         0.00
659         SPACEBOY BABY GIFT SET     15810  United Kingdom         0.00
667         SPACEBOY BABY GIFT SET     16360  United Kingdom         0.00
714              TRELLIS COAT RACK     15993  United Kingdom         0.00
```



```
df3= df2[~df2["Total Price"].isin(df_negative["Total Price"])]
df3
✓ 0.1s
```

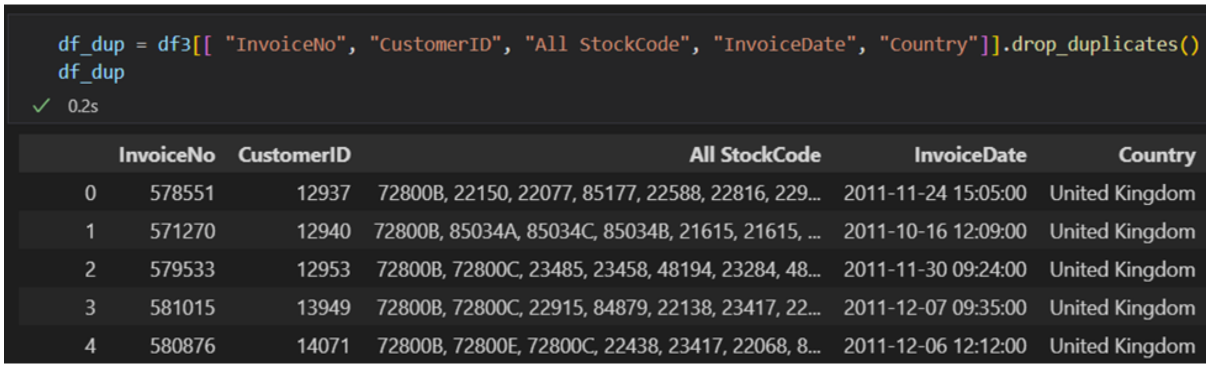
```
                  Description  CustomerID         Country  Total Price  InvoiceNo  StockCode          InvoiceDate
0  4 PURPLE FLOCK DINNER CANDLES       12937  United Kingdom         2.55     578551     72800B  2011-11-24 15:05:00
1  4 PURPLE FLOCK DINNER CANDLES       12940  United Kingdom         2.55     571270     72800B  2011-10-16 12:09:00
2  4 PURPLE FLOCK DINNER CANDLES       12953  United Kingdom        15.30     579533     72800B  2011-11-30 09:24:00
3  4 PURPLE FLOCK DINNER CANDLES       13949  United Kingdom         9.48     581015     72800B  2011-12-07 09:35:00
4  4 PURPLE FLOCK DINNER CANDLES       14071  United Kingdom         0.79     580876     72800B  2011-12-06 12:12:00
```

Next, we group all duplicate descriptions based on Customer ID, then, we join the new column is All Description and finally, drop all redundant rows with duplicates.



```
df_dup = df3[df3["CustomerID"].duplicated(keep=False)]
df3["All StockCode"] = df_dup.groupby(["InvoiceNo","CustomerID"])["StockCode"].transform(', '.join)
df3.head()
✓ 4.9s
```

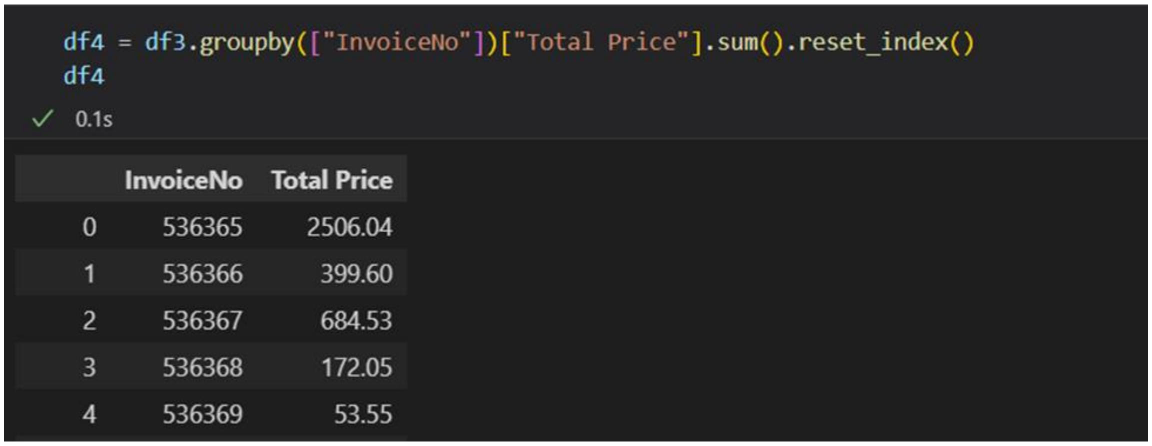
```
                  Description  CustomerID         Country  Total Price  InvoiceNo  StockCode          InvoiceDate                                    All StockCode
0  4 PURPLE FLOCK DINNER CANDLES       12937  United Kingdom         2.55     578551     72800B  2011-11-24 15:05:00   72800B, 22150, 22077, 85177, 22588, 22816, 229...
1  4 PURPLE FLOCK DINNER CANDLES       12940  United Kingdom         2.55     571270     72800B  2011-10-16 12:09:00  72800B, 85034A, 85034C, 85034B, 21615, 21615, ...
2  4 PURPLE FLOCK DINNER CANDLES       12953  United Kingdom        15.30     579533     72800B  2011-11-30 09:24:00   72800B, 72800C, 23485, 23458, 48194, 23284, 48...
3  4 PURPLE FLOCK DINNER CANDLES       13949  United Kingdom         9.48     581015     72800B  2011-12-07 09:35:00   72800B, 72800C, 22915, 84879, 22138, 23417, 22...
4  4 PURPLE FLOCK DINNER CANDLES       14071  United Kingdom         0.79     580876     72800B  2011-12-06 12:12:00   72800B, 72800E, 72800C, 22438, 23417, 22068, 8...
```



```
df_dup = df3[[ "InvoiceNo", "CustomerID", "All StockCode", "InvoiceDate", "Country"]].drop_duplicates()
df_dup
✓ 0.2s
```

```
   InvoiceNo  CustomerID                                     All StockCode          InvoiceDate         Country
0     578551       12937   72800B, 22150, 22077, 85177, 22588, 22816, 229...  2011-11-24 15:05:00  United Kingdom
1     571270       12940  72800B, 85034A, 85034C, 85034B, 21615, 21615, ...  2011-10-16 12:09:00  United Kingdom
2     579533       12953   72800B, 72800C, 23485, 23458, 48194, 23284, 48...  2011-11-30 09:24:00  United Kingdom
3     581015       13949   72800B, 72800C, 22915, 84879, 22138, 23417, 22...  2011-12-07 09:35:00  United Kingdom
4     580876       14071   72800B, 72800E, 72800C, 22438, 23417, 22068, 8...  2011-12-06 12:12:00  United Kingdom
```

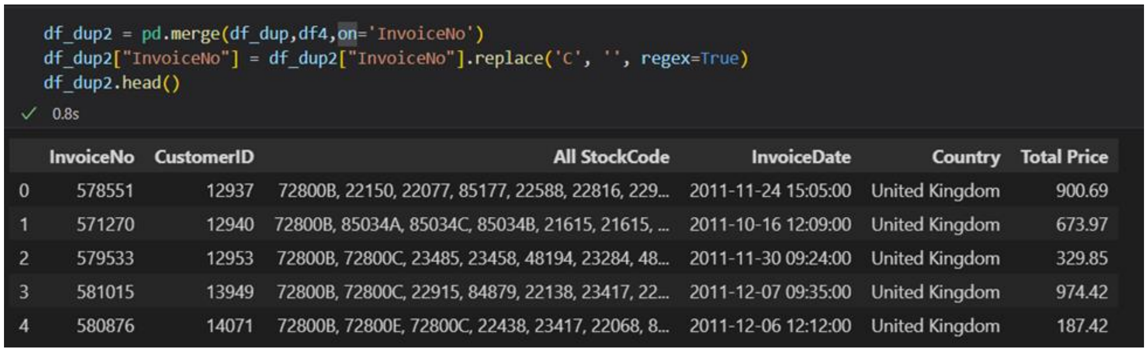
Because we cannot group based on both Customer ID and Total Price, we must merge 2 data frames one more time based on Invoice No.



```
df4 = df3.groupby(["InvoiceNo"])["Total Price"].sum().reset_index()
df4
```

✓ 0.1s

	InvoiceNo	Total Price
0	536365	2506.04
1	536366	399.60
2	536367	684.53
3	536368	172.05
4	536369	53.55



```
df_dup2 = pd.merge(df_dup,df4,on='InvoiceNo')
df_dup2["InvoiceNo"] = df_dup2["InvoiceNo"].replace('C', '', regex=True)
df_dup2.head()
```

✓ 0.8s

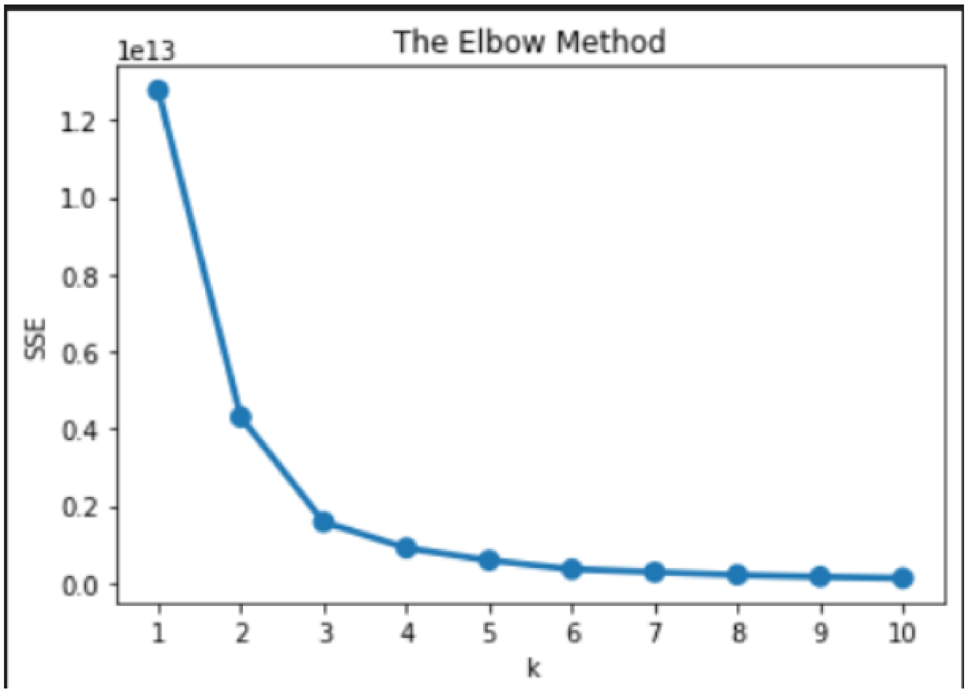
	InvoiceNo	CustomerID	All StockCode	InvoiceDate	Country	Total Price
0	578551	12937	72800B, 22150, 22077, 85177, 22588, 22816, 229...	2011-11-24 15:05:00	United Kingdom	900.69
1	571270	12940	72800B, 85034A, 85034C, 85034B, 21615, 21615, ...	2011-10-16 12:09:00	United Kingdom	673.97
2	579533	12953	72800B, 72800C, 23485, 23458, 48194, 23284, 48...	2011-11-30 09:24:00	United Kingdom	329.85
3	581015	13949	72800B, 72800C, 22915, 84879, 22138, 23417, 22...	2011-12-07 09:35:00	United Kingdom	974.42
4	580876	14071	72800B, 72800E, 72800C, 22438, 23417, 22068, 8...	2011-12-06 12:12:00	United Kingdom	187.42

Finally, data (df_dup2) ready to apply RFM Analysis and clustering.

IV. Implement a clustering/classification algorithm

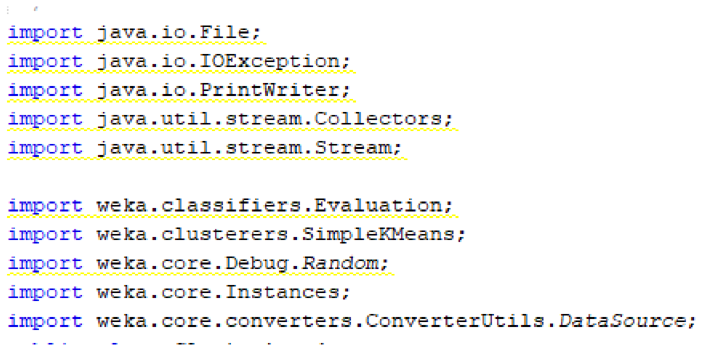
To categorize the type of consumer, our group used the k-means approach to do the clustering process based on the RFM study results. However, before we begin clustering, our team must first scale the dataset. When using K-means, attribute scaling is critical. The usual Euclidean distance (as a distance function of K-means) is most used, under the assumption that the attributes are normalized. To put it another way, you want to be sure that your calculations aren't skewed toward the extremes of the scale. In other words, you want to make sure that all your data is at the same level. All information associated with a specific customer is used to calculate the placement of each data point on the graph. K-means may not build meaningful clusters for you if any of the data is not on the same distance scale.

Another issue we must handle is determining the optimal number of Clusters. A varied number of clusters might result in very different outcomes. To cope with this, we apply the Elbow technique, which determines the best cluster number by employing the sum of squared distance (SSE) to select an ideal value of k depending on the distance between data points and their assigned clusters. We'd pick a k number where the SSE starts to flatten out and an inflection point appears.



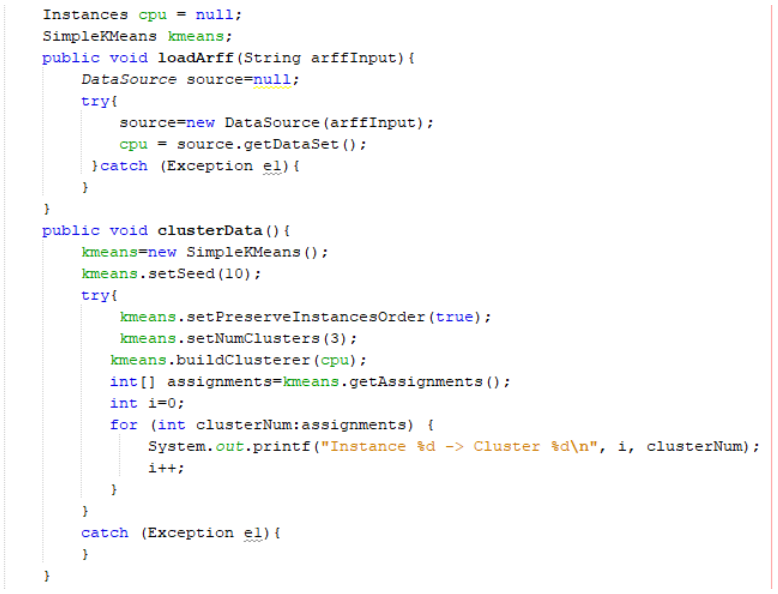
The graph above shows that k=3 is probably a good choice for the number of clusters.

* Clustering using Weka Lib Java:



```
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import weka.classifiers.Evaluation;
import weka.clusterers.SimpleKMeans;
import weka.core.Debug.Random;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
```

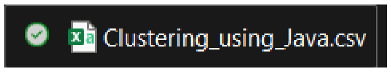
All libraries that we used.



```
Instances cpu = null;
SimpleKMeans kmeans;
public void loadArff(String arffInput){
    DataSource source=null;
    try{
        source=new DataSource(arffInput);
        cpu = source.getDataSet();
     }catch (Exception e1){
    }
}
public void clusterData(){
    kmeans=new SimpleKMeans();
    kmeans.setSeed(10);
    try{
        kmeans.setPreserveInstancesOrder(true);
        kmeans.setNumClusters(3);
       kmeans.buildClusterer(cpu);
       int[] assignments=kmeans.getAssignments();
       int i=0;
       for (int clusterNum:assignments) {
           System.out.printf("Instance %d -> Cluster %d\n", i, clusterNum);
           i++;
       }
    }
    catch (Exception e1){
    }
}
```

Then, we apply K-Means Algorithm to print cluster.

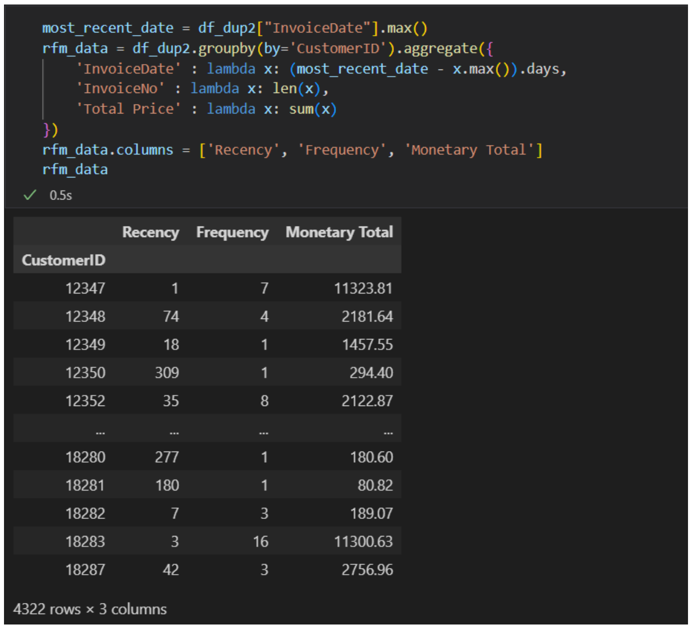
After we exported the csv file containing the clusters 0 1 2 for each Instance, we realized the inconsistency because the values were too extreme. So, we decided to use Python to try to see the results later. You can see that in



Clustering_using_Java.csv

* **Clustering using Python**:

In this step, we use RFM analysis result to calculate Recency, Frequency, Monetary Total as below:



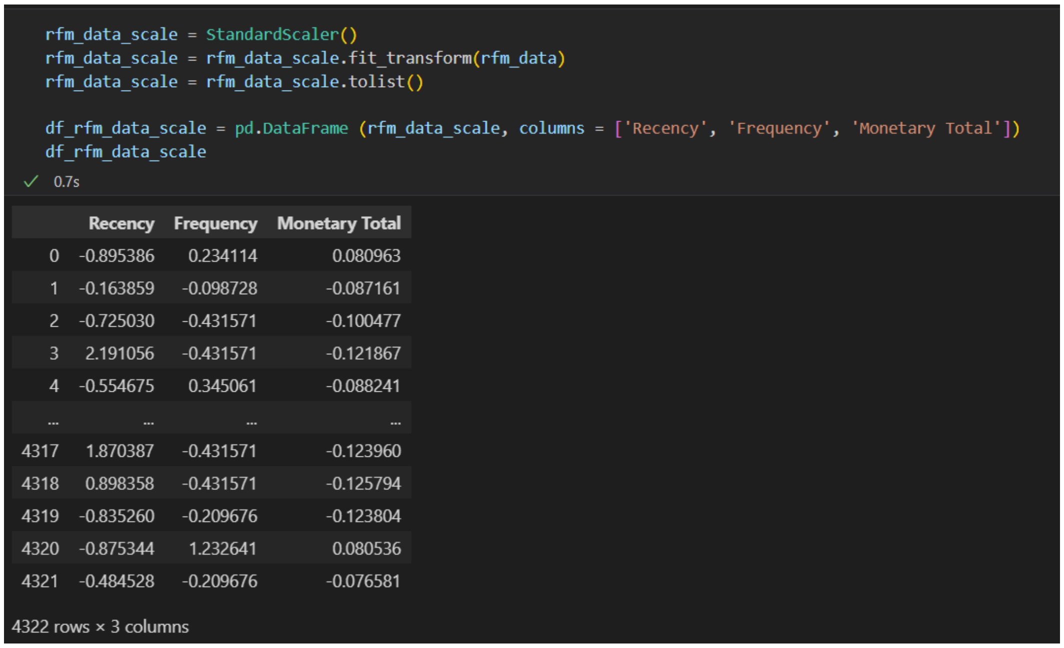
```
most_recent_date = df_dup2["InvoiceDate"].max()
rfm_data = df_dup2.groupby(by='CustomerID').aggregate({
    'InvoiceDate' : lambda x: (most_recent_date - x.max()).days,
    'InvoiceNo' : lambda x: len(x),
    'Total Price' : lambda x: sum(x)
})
rfm_data.columns = ['Recency', 'Frequency', 'Monetary Total']
rfm_data
```

✓ 0.5s

	Recency	Frequency	Monetary Total
CustomerID			
12347	1	7	11323.81
12348	74	4	2181.64
12349	18	1	1457.55
12350	309	1	294.40
12352	35	8	2122.87
...	...	...	...
18280	277	1	180.60
18281	180	1	80.82
18282	7	3	189.07
18283	3	16	11300.63
18287	42	3	2756.96

4322 rows × 3 columns

*** Scaling data:**



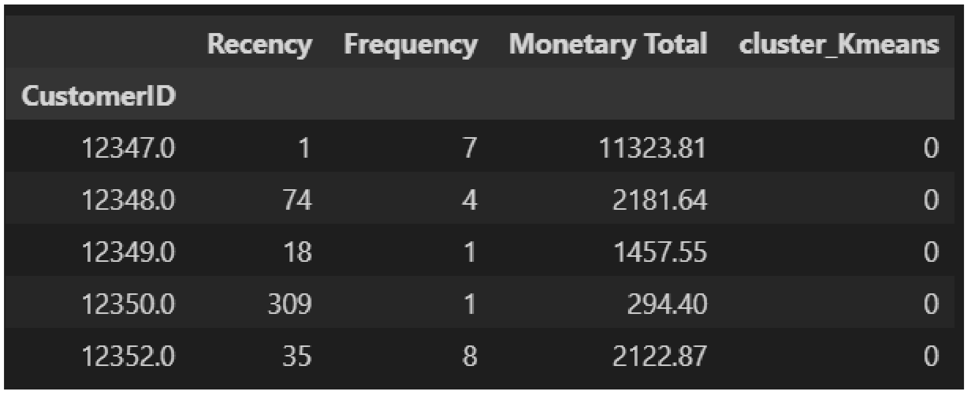
```
rfm_data_scale = StandardScaler()
rfm_data_scale = rfm_data_scale.fit_transform(rfm_data)
rfm_data_scale = rfm_data_scale.tolist()

df_rfm_data_scale = pd.DataFrame (rfm_data_scale, columns = ['Recency', 'Frequency', 'Monetary Total'])
df_rfm_data_scale
```

✓ 0.7s

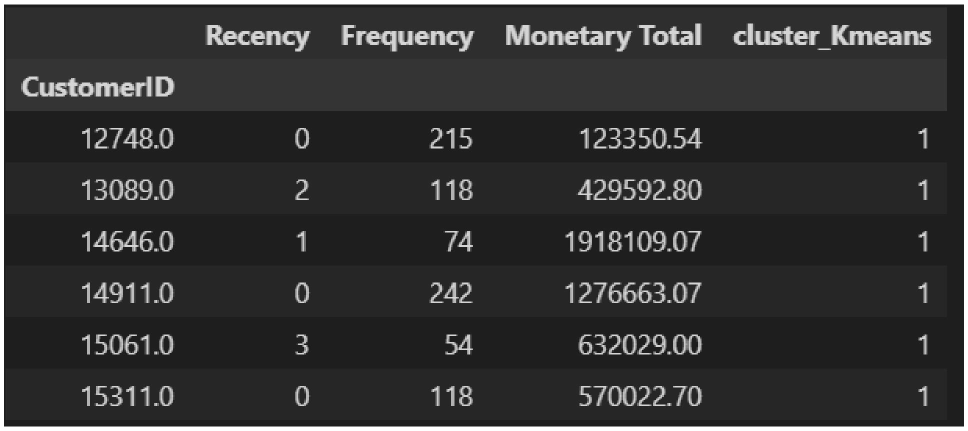
	Recency	Frequency	Monetary Total
0	-0.895386	0.234114	0.080963
1	-0.163859	-0.098728	-0.087161
2	-0.725030	-0.431571	-0.100477
3	2.191056	-0.431571	-0.121867
4	-0.554675	0.345061	-0.088241
...	...	...	...
4317	1.870387	-0.431571	-0.123960
4318	0.898358	-0.431571	-0.125794
4319	-0.835260	-0.209676	-0.123804
4320	-0.875344	1.232641	0.080536
4321	-0.484528	-0.209676	-0.076581

4322 rows × 3 columns

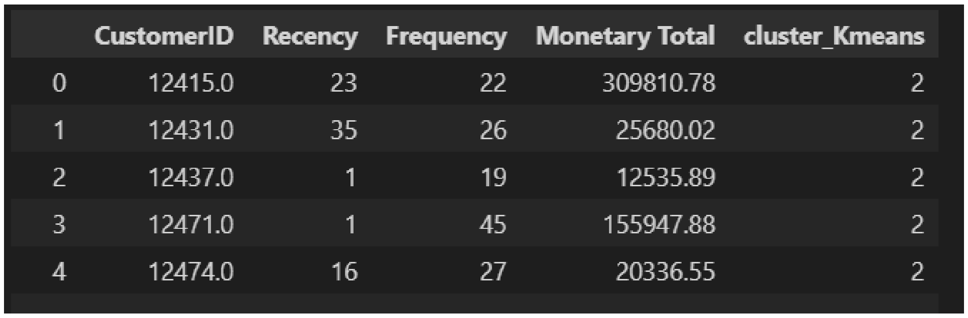
*** Result after running K-means:**



	Recency	Frequency	Monetary Total	cluster_Kmeans
CustomerID				
12347.0	1	7	11323.81	0
12348.0	74	4	2181.64	0
12349.0	18	1	1457.55	0
12350.0	309	1	294.40	0
12352.0	35	8	2122.87	0



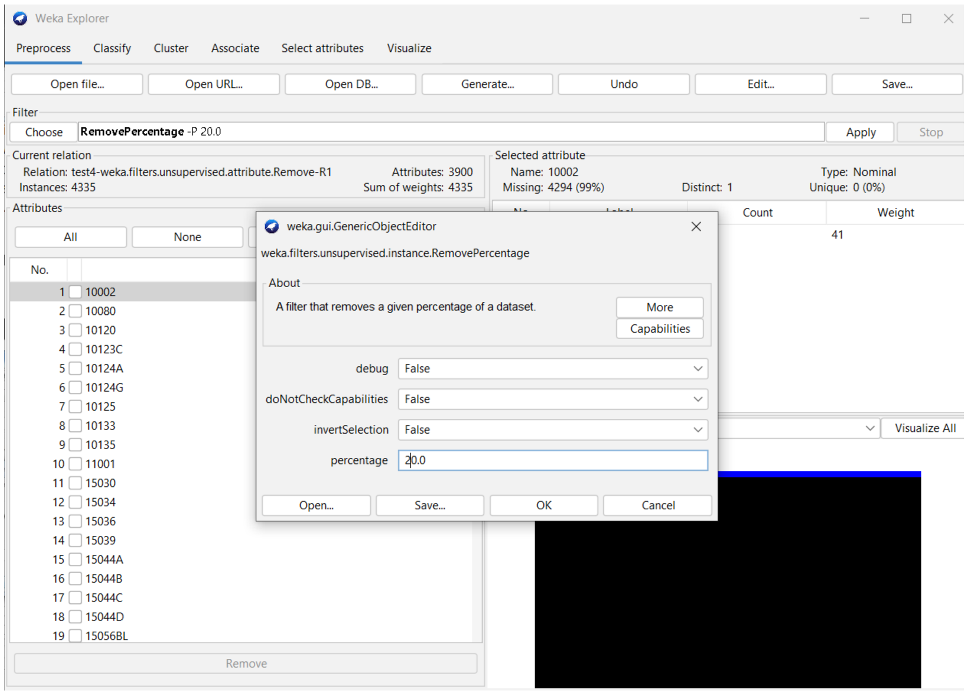
	Recency	Frequency	Monetary Total	cluster_Kmeans
CustomerID				
12748.0	0	215	123350.54	1
13089.0	2	118	429592.80	1
14646.0	1	74	1918109.07	1
14911.0	0	242	1276663.07	1
15061.0	3	54	632029.00	1
15311.0	0	118	570022.70	1



	CustomerID	Recency	Frequency	Monetary Total	cluster_Kmeans
0	12415.0	23	22	309810.78	2
1	12431.0	35	26	25680.02	2
2	12437.0	1	19	12535.89	2
3	12471.0	1	45	155947.88	2
4	12474.0	16	27	20336.55	2

V. Implement a sequence mining algorithm (FPGrowth)

We used Weka to split the dataset into train and test sets before using the fp-growth approach to identify the association rule. We divide the dataset into two parts: 20% testing and 80% training. To do this, we must first Randomize the dataset (Unsupervised > Instance) to generate a random permutation. Then we use RemovePercentage (Unsupervised > Instance) with a percentage of 20 and save the dataset as training:



RemovePercentage -P 20.0

Current relation
Relation: test4-weka.filters.unsupervised.attribute.Remove-R1 Attributes: 3900
Instances: 4335 Sum of weights: 4335

Selected attribute
Name: 10002 Type: Nominal
Missing: 4294 (99%) Distinct: 1 Unique: 0 (0%)

Count	Weight
	41

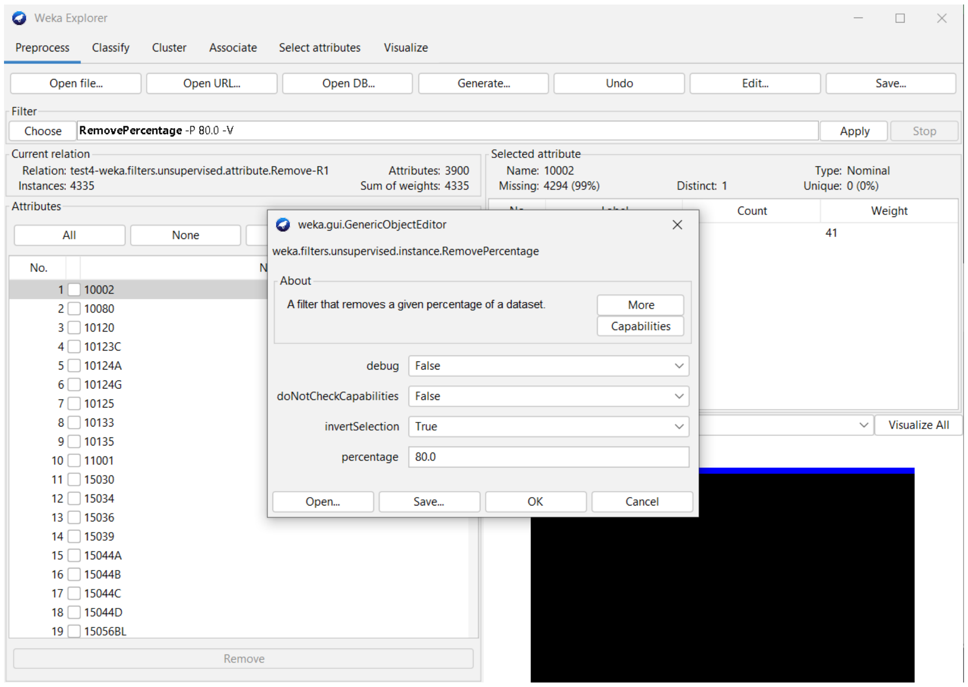
Attributes

No.	
1	10002
2	10080
3	10120
4	10123C
5	10124A
6	10124G
7	10125
8	10133
9	10135
10	11001
11	15030
12	15034
13	15036
14	15039
15	15044A
16	15044B
17	15044C
18	15044D
19	15056BL

weka.gui.GenericObjectEditor
weka.filters.unsupervised.instance.RemovePercentage
About
A filter that removes a given percentage of a dataset.

debug	False
doNotCheckCapabilities	False
invertSelection	False
percentage	20.0

After that, we undo and apply the same filter again, this time using InvertSelection. This will choose the remaining data (20%), which we will store for testing.



Filter
Choose **RemovePercentage** -P 80.0 -V

Current relation
Relation: test4-weka.filters.unsupervised.attribute.Remove-R1 Attributes: 3900
Instances: 4335 Sum of weights: 4335

Selected attribute
Name: 10002 Type: Nominal
Missing: 4294 (99%) Distinct: 1 Unique: 0 (0%)

No.	Label	Count	Weight
		41	

Attributes

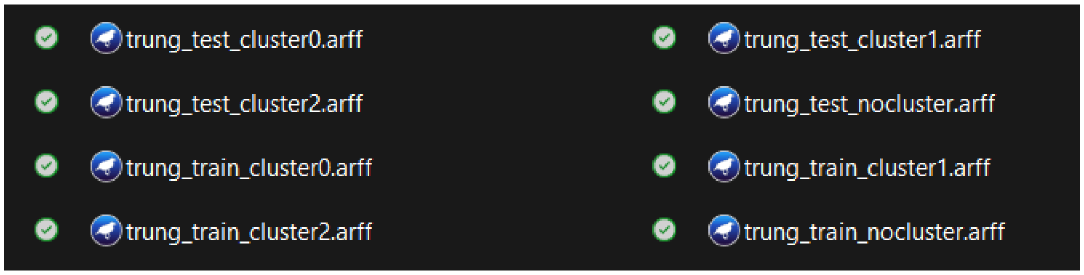
No.	Name
1	10002
2	10080
3	10120
4	10123C
5	10124A
6	10124G
7	10125
8	10133
9	10135
10	11001
11	15030
12	15034
13	15036
14	15039
15	15044A
16	15044B
17	15044C
18	15044D
19	15056BL

weka.gui.GenericObjectEditor
weka.filters.unsupervised.instance.RemovePercentage
About
A filter that removes a given percentage of a dataset.
More
Capabilities

debug	False
doNotCheckCapabilities	False
invertSelection	True
percentage	80.0

Open... Save... OK Cancel

Result of the train/test dataset can be seen in these files:



trung_test_cluster0.arff
trung_test_cluster1.arff
trung_test_cluster2.arff
trung_test_nocluster.arff
trung_train_cluster0.arff
trung_train_cluster1.arff
trung_train_cluster2.arff
trung_train_nocluster.arff

After splitting the whole dataset and clustered dataset to the test/train set, we run FP-growth algorithm on these files to get the frequent item sets

In this step, we use the ARFF file that was clustered in the previous step. We drop the CustomerID column for the Weka library in Java to understand.

We use the following libraries:

```java
import java.io.BufferedReader;
import java.io.FileReader;
import weka.core.converters.ConverterUtils.DataSource;
import java.util.Random;
import weka.associations.FPGrowth;
import weka.core.Instances;
```

Next, we read the clustered ARFF file with the following command:

Cluster 0:

```java
Instances dataset = new Instances(new BufferedReader(new FileReader("C:\\Users\\Asus\\OneDrive - VietNam National University - HCM INTERNATIONAL UNIVERSITY"
        + "\\Study Document - INTERNATIONAL UNIVERSITY\\Năm 3rd\\Sem 2\\Intro to Data Mining\\Project\\trung_train_cluster0.arff")));
```

Cluster 1:

```
Instances dataset = new Instances(new BufferedReader(new FileReader("C:\\Users\\Asus\\OneDrive - VietNam National University - HCM INTERNATIONAL UNIVERSITY"
        + "\\Study Document - INTERNATIONAL UNIVERSITY\\Năm 3rd\\Sem 2\\Intro to Data Mining\\Project\\trung_train_cluster1.arff")));
```

Cluster 2:

```
Instances dataset = new Instances(new BufferedReader(new FileReader("C:\\Users\\Asus\\OneDrive - VietNam National University - HCM INTERNATIONAL UNIVERSITY"
        + "\\Study Document - INTERNATIONAL UNIVERSITY\\Năm 3rd\\Sem 2\\Intro to Data Mining\\Project\\trung_train_cluster2.arff")));
```

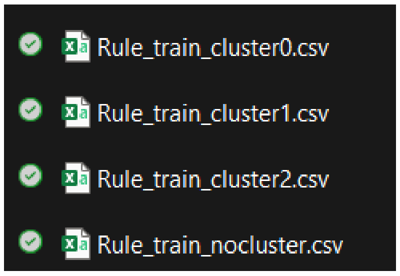
Declare the object FPGrowth:

```
FPGrowth fpg = new FPGrowth();
```

Then, Set value of MinSupport, MinMetric, LowerBoundMinSupport, NumRulesToFind. And then we call buildAssociation() function.

```
fpg.setMinMetric(0.3);
fpg.setLowerBoundMinSupport(0.05);
fpg.setNumRulesToFind(100);
fpg.buildAssociations(dataset);
```

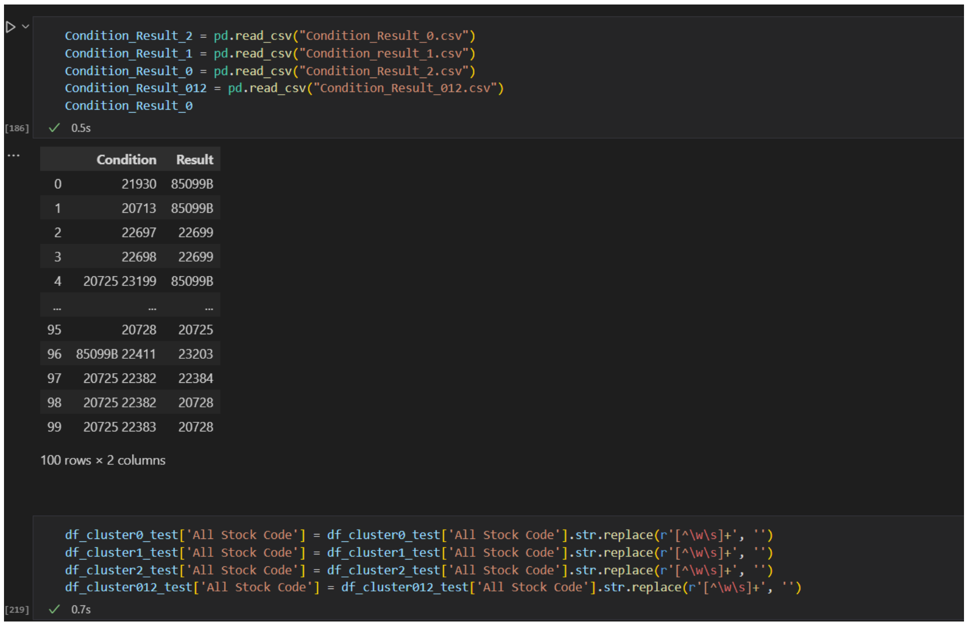
Finally, we print the Rules and export them into csv files:



Rule_train_cluster0.csv
Rule_train_cluster1.csv
Rule_train_cluster2.csv
Rule_train_nocluster.csv

VI. Model Evaluation

1. Initially we import association rules result after being trained on test dataset.



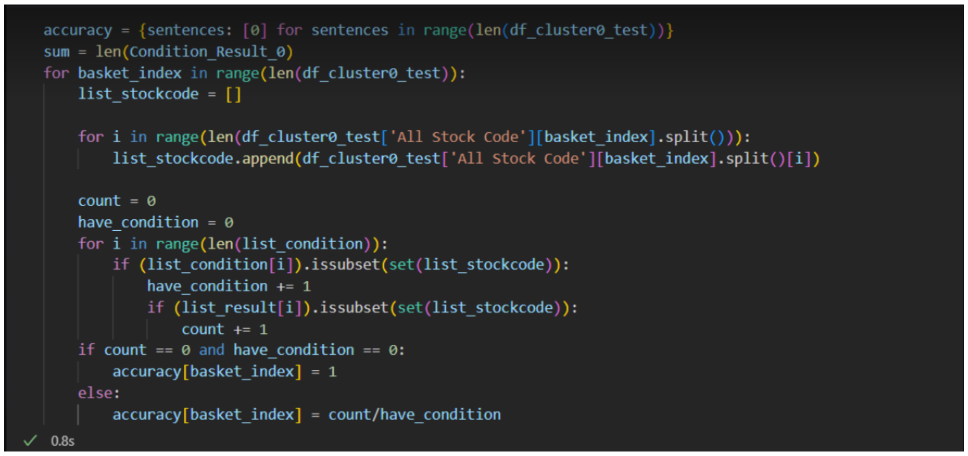
```
Condition_Result_2 = pd.read_csv("Condition_Result_0.csv")
Condition_Result_1 = pd.read_csv("Condition_result_1.csv")
Condition_Result_0 = pd.read_csv("Condition_Result_2.csv")
Condition_Result_012 = pd.read_csv("Condition_Result_012.csv")
Condition_Result_0
```

```
[186]  ✓ 0.5s

       Condition   Result
0          21930  85099B
1          20713  85099B
2          22697   22699
3          22698   22699
4    20725 23199  85099B
...          ...     ...
95         20728   20725
96  85099B 22411   23203
97   20725 22382   22384
98   20725 22382   20728
99   20725 22383   20728

100 rows × 2 columns
```

```
df_cluster0_test['All Stock Code'] = df_cluster0_test['All Stock Code'].str.replace(r'[^\w\s]+', '')
df_cluster1_test['All Stock Code'] = df_cluster1_test['All Stock Code'].str.replace(r'[^\w\s]+', '')
df_cluster2_test['All Stock Code'] = df_cluster2_test['All Stock Code'].str.replace(r'[^\w\s]+', '')
df_cluster012_test['All Stock Code'] = df_cluster012_test['All Stock Code'].str.replace(r'[^\w\s]+', '')
```

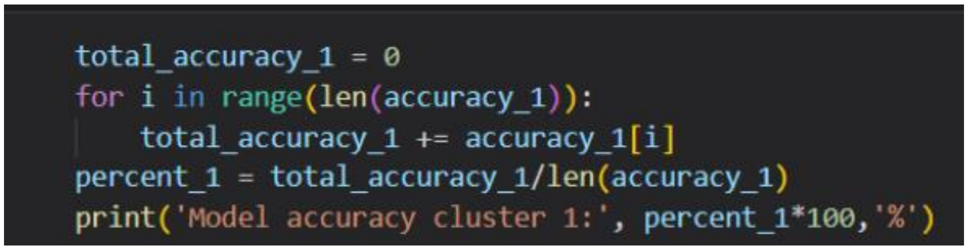
```
[219]  ✓ 0.7s
```

2. In the next step, we compare it with the test dataset and count how many times we correctly guessed what the next product to buy is and divide it by the total number of transactions to calculate the accuracy of the rules.



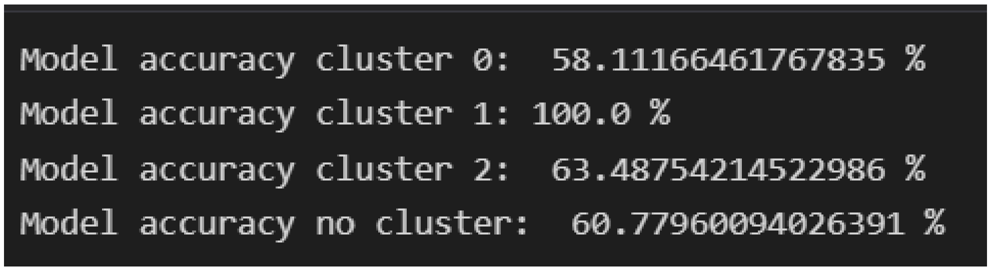
```
accuracy = {sentences: [0] for sentences in range(len(df_cluster0_test))}
sum = len(Condition_Result_0)
for basket_index in range(len(df_cluster0_test)):
    list_stockcode = []

    for i in range(len(df_cluster0_test['All Stock Code'][basket_index].split())):
        list_stockcode.append(df_cluster0_test['All Stock Code'][basket_index].split()[i])

    count = 0
    have_condition = 0
    for i in range(len(list_condition)):
        if (list_condition[i]).issubset(set(list_stockcode)):
            have_condition += 1
            if (list_result[i]).issubset(set(list_stockcode)):
                count += 1
    if count == 0 and have_condition == 0:
        accuracy[basket_index] = 1
    else:
        accuracy[basket_index] = count/have_condition
```

```
✓ 0.8s
```



```
total_accuracy_1 = 0
for i in range(len(accuracy_1)):
    total_accuracy_1 += accuracy_1[i]
percent_1 = total_accuracy_1/len(accuracy_1)
print('Model accuracy cluster 1:', percent_1*100,'%')
```

The result:



```
Model accuracy cluster 0:  58.11166461767835 %
Model accuracy cluster 1: 100.0 %
Model accuracy cluster 2:  63.48754214522986 %
Model accuracy no cluster:  60.77960094026391 %
```

Remark: We may conclude from the results that clustering the dataset can improve the accuracy of the association rule compared to a large dataset without clustering.

VII. Conclusion

Through this project, we realize that data mining is extremely necessary. Because when we receive an original dataset, it is not necessarily a beautiful and clear dataset, so we must apply data mining algorithms to process them into a clean dataset. That is, the dataset must have no missing values, no Null Values, ... Then we proceed to apply Weka's algorithms for clustering and use the Association Rule to make our dataset more accurate during the evaluation process.

Also from this project, we found that the Weka library on Java has quite a few limitations, such as when applying K-Means to clustering, the results between clusters are quite close, this does not seem appropriate, very unreasonable compared to reality. When running in Python, it is better. In addition, we also gain more knowledge about tools in data processing, which will be especially useful to us later. For example, after using the Weka library in Java code, we have a better understanding of how the Weka Tool color performs the output process.

Another aspect we find very interesting in this project is that we must be cautious when clustering datasets, especially when scaling data before clustering, because K-means clustering results are potentially sensitive to the order of objects in the data set and leaving variances unequal is equivalent to giving more weight to variables with smaller variance, causing clusters to be separated along variables with greater variance. The accuracy of the clustering method can be affected by changes in the dataset after scaling, thus we must be cautious when doing the scaling step. Another consideration is that selecting the ideal K number is critical, since selecting the incorrect K number might result in poor clustering results.

VIII. References

References

[1] J. Korstanje, "TowardDataScience," 2021. [Online]. Available: https://towardsdatascience.com/the-fp-growth-algorithm-1ffa20e839b8.

[2] Paresh Tanna, Dr. Yogesh Ghodasara, "Using Apriori with WEKA for Frequent Pattern Mining," 2014. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1406/1406.7371.pdf.

[3] WekaWiki, "Weka Wiki," Weka and Github, [Online]. Available: https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/.

[4] N. Sadawi, "Weka API," Youtube, [Online]. Available: https://www.youtube.com/playlist?list=PLea0WJq13cnBVfsPVNyRAus2NK-KhCuzJ.

[5] Weka, "Weka Source Forge," Weka, [Online]. Available: https://weka.sourceforge.io/doc.dev/weka/associations/FPGrowth.html.

[6] StackExchange, "StackExchange," [Online]. Available: https://stats.stackexchange.com/questions/21222/are-mean-normalization-and-feature-scaling-needed-for-k-means-clustering?fbclid=IwAR0Yx0Evek4uEKoOFpP8TEHtUQTOAxOo0VzhxjuUthP3Q4WDYr4C4VVWUY8.