VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme	**:**	**B. Tech CSE**
Course Title	**:**	**Social and Information Networks**
Course Code	**:**	**CSE3021**
Slot	**:**	**C2+TC2**
Title	**:**	**Item Basket Analysis**
Team Members	**:**	**Kotha Eshwar Diwakar / 20BCE1963**
		Nagaraj S / 20BCE1187
		Mahesh Pavan Varma / 20BCE1131

Faculty: Dr Punitha K **Sign:**

Date:

A project report on

TITLE OF THE PROJECT REPORT

‘

Submitted in partial fulfilment for the course

Social and Information Networks

by

KOTHA ESHWAR DIWAKAR / 20BCE1963

NAGARAJ S / 20BCE1187

MAHESH PAVAN VARMA / 20BCE1131

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2023

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

DECLARATION

We here by declare that the thesis entitled “ITEM BASKET ANALYSIS ” submitted by Eshwar ,Nagaraj and Pavan , for the completion of the course, Social and Information Network(CSE3021) is a record of bonafide work carried out by we under the supervision of Dr Punitha K, Our course instructor, we further declare that the work reported in this document has not been submitted and will not be submitted, either in part or in full, for any other courses in this institute or any other institute or university.

Place: Chennai

Date:

Kotha Eshwar Diwakar
Nagaraj S
Mahesh Pavan Varma

Signature of the Candidates

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **"ITEM BASKET ANALYSIS"** is prepared and submitted by **KOTHA ESHWAR DIWAKAR** (**20BCE1963**), **NAGARAJS** (20BCE1187) and **MAHESH PAVAN VARMA**(20BCE1131) to Vellore Institute of Technology, Chennai, in partial fulfilment of the requirement for the course, **Social and Information Networks (CSE3021),** is a Bonafede record carried out under my guidance. The project fulfils the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the any other course and the same is certified.

Name: Dr. Punitha K

Signature of the Faculty

Date: 13.04.23

ABSTRACT

Many contemporary trade or retail organizations today struggle to obtain vital customer data from huge databases of item attributes and data collecting. The majority of data mining research has focused on developing algorithms to identify statistically significant patterns in enormous datasets. The usefulness of such patterns is widely disputed, and it takes more skill than science to use the resulting patterns in decision support. In this study, we formalize a data mining strategy for proposing the best products to customers in a store based on the items in their market baskets.

Market basket databases maintain track of prior consumer selections, where each customer chose a market basket—a collection of goods from a wide but constrained range. Using this data, new things may be dynamically suggested to customers who are thinking about buying something.

As the user adds things to the virtual shopping cart or market basket, many commercial websites on the Internet provide dynamic product possibilities. A constantly changing collection of links to relevant websites is also shown on websites, depending on the browsing activity during a surfing session. By highlighting purchasing trends, market basket analysis (MBA), also known as item basket analysis, assists managers in the retail and restaurant industries in better understanding and ultimately satisfying their consumers. The most frequent product combinations seen in orders are listed by MBA. These relationships may be taken advantage of to increase revenue through cross selling, suggestions, promotions, or even the positioning of items on a menu or in a shop.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

In successfully completing this project, many people have helped me. I would like tothank all those who are related to this project. I would like to thank my teacher (Dr. Punitha) who gave me this opportunity to work on this project and also for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and implementation of the project. I got to learn a lot from this project about how to search for the best datasets for the projects, how to make your project more effective, which will be very helpful in solving some real-world problems and many other things. I would also like to thank our school HOD. At last, I would like to extend my heartfelt thanks to my parents because without their help this project would not have been successful. Finally, I would like to thank my dear friends and my fellow students for many helpful discussions and good ideas along the way who have been with me all the time.

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

In order to increase market sales, revenues, and consumer purchasing habits, market owners can be found most effectively through market basket analysis. In order to maximize revenue through maximum sales, our team's purpose for this project is to understand how large businesses and general stores target their customers by selling them additional goods that are comparable to the stuff they've already purchased. The products they sell are price-driven and, in many circumstances, less valuable than the object the customer bought, but they are just as significant to the customer. To increase their profits, a store might, for instance, sell eggs or butter at a low margin discount if a consumer buys bread.

1. We frequently use the Apriori algorithm since it makes it simpler to find these patterns inside goods in a shorter length of time and because the majority of data transactions in a real-world retail market or an online e-commerce site are large.
2. We want to be able to develop a system of pattern recommendations that differentretail firms can employ.
3. By highlighting the purchasing trend,this will aid in better understanding people'spurchasing habits.
4. Since gathering real-world data is challenging, the trained models are fed information from two real-world datasets gathered from Kaggle.
5. Through cross-selling, suggestions, advertising, or even the placement of times on a menu or in a store, this data may be leveraged to ultimately increase profitability.

1.2 OVERALL DESCRIPTION :

We are using datasets from many nations for this investigation. The datasets are set up as a list, with the transaction id as the first element and a list of item ids for items bought in tandem as the second element. List of names for the items is called Items. The no of items column contains the total number of items in the dataset. These records are made up of tuples, where the first element is an itemset, and the second element is a group of transaction ids with an associated itemset. After using the APRIORI, FP, SETM, K MEANS algorithm, ECLAT algorithm, Association rule mining, we can quickly obtain the output dataset of the goods that are typically purchased alongside a given item in stores for a particular nation. As a result, it can tell the difference between items that are absolutely required and those that are the most annoying. Market tactics, consumer convenience, and the quantity of goods sold all improve as a result.

DATASET:

Datasets which are used in the project have been collected from kaggle datasets. Columns present in these datasets are Invoice number, Stock code, Description, Quantity, Invoice date, Unit Price, Customer ID, Country.

1.3 PROPOSED SYSTEM

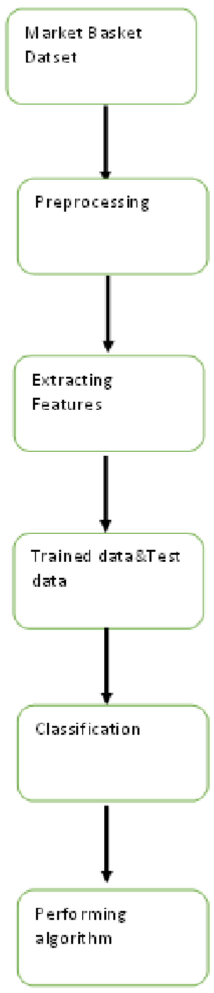


Fig1. Proposed model Flow chart

Pre-processing

In Machine Learning, data pre-processing refers to the process of preparing raw data to make it appropriate for the construction and training of Machine Learning models. Here the job prediction datasets are pre processed so as to improve the working of the system, as in this process the raw data is processed into proper data.

Extracting features

Feature Extraction is a technique for reducing the amount of features in a dataset by generating new ones from existing ones. The original set of features should then be able to summarise the majority of the information in the new reduced set of features.

Trained data & test data

A training set is a subset of a larger data set that is used to fit a model for predicting or classifying values that are known in the training set but unknown in the rest of the data. The training set is used in conjunction with the validation and/or test sets to assess various models.

A test set is a subset of a data set that is used in data mining to evaluate the likely future performance of a single prediction or classification model that has been chosen from a pool of competing models based on its performance with the validation set.

Classification

A data mining function called classification allocates objects in a collection to desired groups or classes. Classification's purpose is to correctly anticipate the target class for each case in the data. For example, a categorization model could be used to categorise loan applicants as having low, medium, or high credit risks.

Performing Algorithm

Here we will be performing various classification algorithms for finding the accuracy and finding the error. Here we will also find specificity and sensitivity for comparison of algorithms.

Packages

Packages used in job prediction are as follows –

1) Pandas

2) Numpy

3) Matplotlib

4) Sklearn

Pandas

Pandas are widely used open-source Python library for data science, data analysis, and machine learning activities.

Numpy

NumPy (numerical Python) is a library that consists of multidimensional array objects and a collection of functions for manipulating them. NumPy allows you to conduct mathematical and logical operations on arrays.

Matplotlib

Matplotlib is a Python package that allows you to create static, animated, and interactive visualisations.

Sklearn

Sklearn is a Python-based machine learning package that is available for free. Support-vector machines, random forests, gradient boosting, and k-means are among the classification, regression, and clustering algorithms included.

In this study, we have implemented some of the classifications algorithm for market basket analysis. Here we have implemented classification algorithms like the following :

- APPRIORI ALGORITHM
- FP ALGORITHM
- ASSOCIATION RULE MINING
- SETM ALGORITHM
- K – MEANS CLUSTERING
- ECLAT ALGORITHM

CHAPTER 2

IMPLEMENTATION

2.1 IMPLEMENTATION

To assess the effectiveness of the Apriori algorithm, we conducted a series of experiments on two different sets of data: a Market Basket Optimization available from the basket optimization repository and real e-commerce data with over 50,000 records. These benchmark data sets were chosen because their structure is typical of the most common application scenarios and they are widely used standard benchmark data sets. In practise, we may utilise this strategy in any situation where we have previous transactions and a current client transaction.

Finding out how things are related to one another is done via association rules analysis. There are three typical methods for calculating association.

Measure 1: Support.

Based on how frequently an itemset appears in overall transactions, this shows how popular it is. We refer to item frequency in various words.

Measure 2: Confidence.

This indicates the likelihood of purchasing item Y when item X is purchased, given as {X -> Y}. This is determined by the percentage of transactions in which item X appears with item Y.

Measure 3: Lift.

It's the proportion of expected to observed confidence. It is defined as the ratio of Y's confidence when item X was already known(x/y) to Y's confidence when item X is unknown. In other words, Y's confidence in relation to x and Y's confidence in the absenceof X. (means both are independent to each other).

Support = occurrence of item / total no of transaction.

$$supp(X \Rightarrow Y) = \frac{|X \cup Y|}{n}$$

Fig2. Support equation

Confidence = support (X Union Y) / support(X)

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$

Fig2.1. Confidence equation

Lift = support (X Union Y) / support(X) * support(Y)

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) supp(Y)}$$

Fig2.2 Lift equation

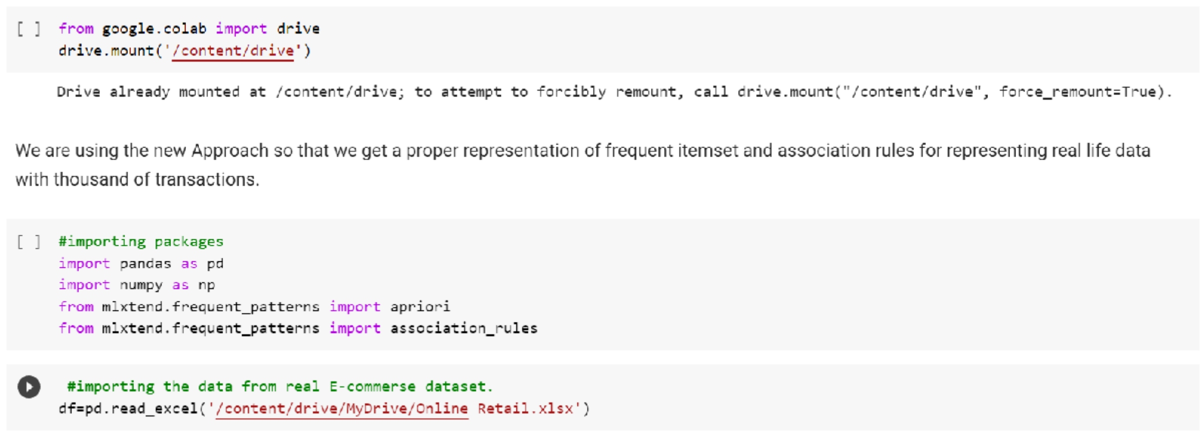
Association rule :

Let I = {I1, I2,…, Im} be an itemset. These itemsets are called antecedents. Let D, the data, be a set of database transactions where each transaction T is a nonempty itemset such that **T ⊆ I**. Each transaction is associated with an identifier called a TID(or Tid). Let A be a set of items(itemset). T is the Transaction that is said to contain A if **A ⊆ T**. An **Association Rule** is an implication of form **A ⇒ B**, where **A ⊂ I, B ⊂ I**, and **A ∩B = φ**.

For the Dataset we are using mlxtend.frequent_patterns package, and there are afair number of advantages for using this approach.

1. The necessity to convert data into a list format was a limitation of the first strategy. A store has thousands of transactions in real life; hence it is computationally expensive.
2. The Apriori package is no longer supported.
3. Because the results are in a less user-friendly format, pre-processing is required.
4. Let's instead utilize the mlxtend package. It creates frequent itemsets as wellasassociation rules.

PRE-PROCESSING:



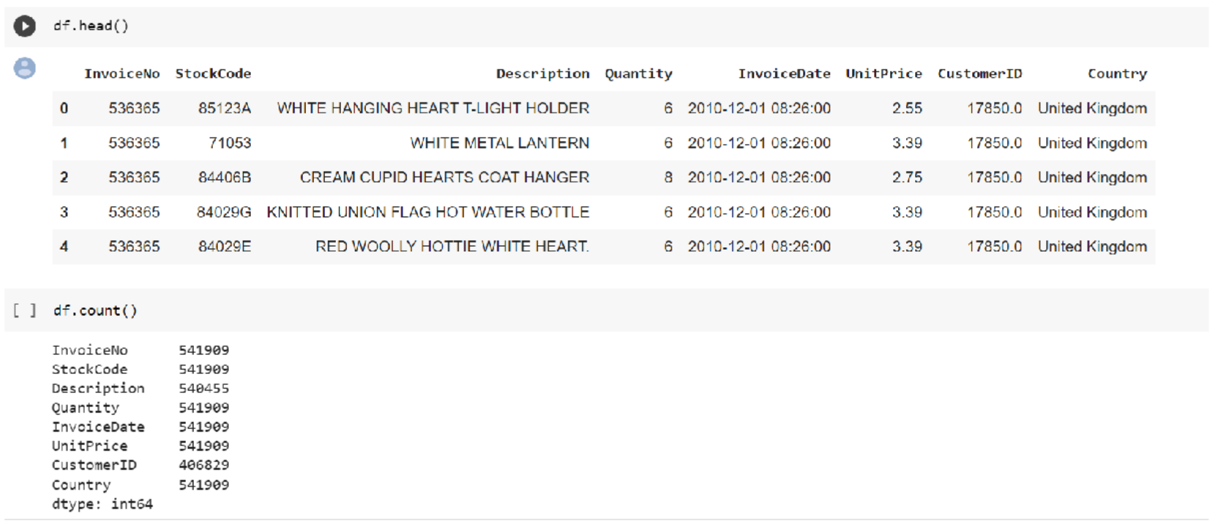
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

We are using the new Approach so that we get a proper representation of frequent itemset and association rules for representing real life data with thousand of transactions.

```
[ ] #importing packages
    import pandas as pd
    import numpy as np
    from mlxtend.frequent_patterns import apriori
    from mlxtend.frequent_patterns import association_rules
```

```
     #importing the data from real E-commerse dataset.
    df=pd.read_excel('/content/drive/MyDrive/Online Retail.xlsx')
```

Fig3.Importing the packages and dataset

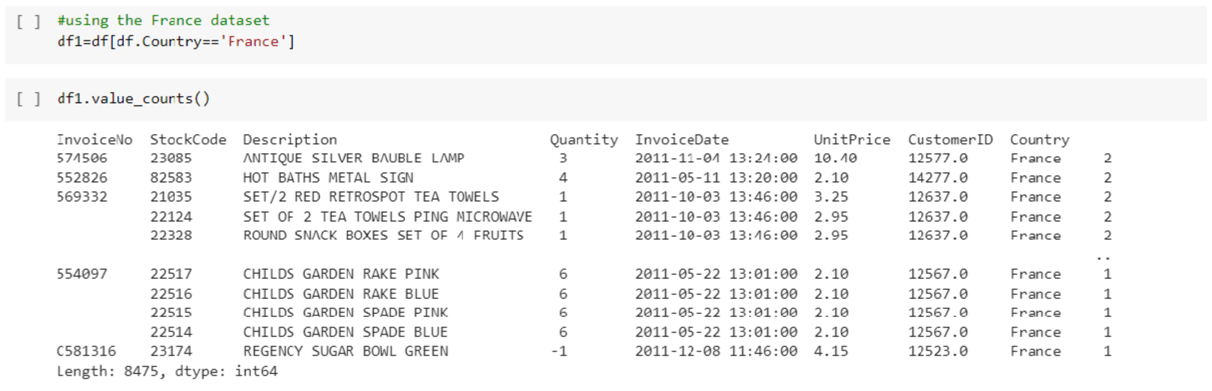


```
df.head()

     InvoiceNo  StockCode                          Description  Quantity          InvoiceDate  UnitPrice  CustomerID         Country
0       536365     85123A   WHITE HANGING HEART T-LIGHT HOLDER         6  2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1       536365      71053                  WHITE METAL LANTERN         6  2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2       536365     84406B       CREAM CUPID HEARTS COAT HANGER         8  2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3       536365     84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6  2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4       536365     84029E       RED WOOLLY HOTTIE WHITE HEART.         6  2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

```
[ ] df.count()

InvoiceNo      541909
StockCode      541909
Description    540455
Quantity       541909
InvoiceDate    541909
UnitPrice      541909
CustomerID     406829
Country        541909
dtype: int64
```

Fig3.1.Total number of data presented, and the serial count for the head

Since we have a very large dataset, we are limiting the dataset by using the data presented by France, by extracting them from the full dataset.

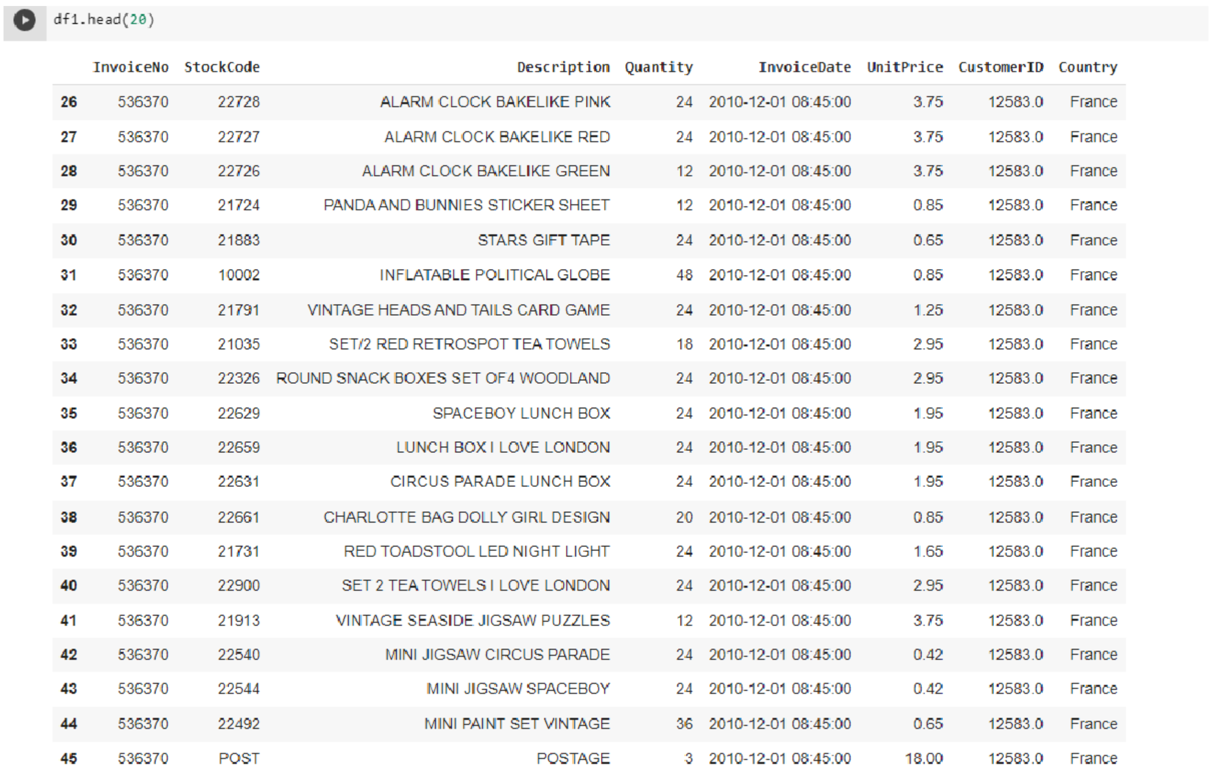


```
[ ] #using the France dataset
    df1=df[df.Courtry=='France']
```

```
[ ] df1.value_courts()
```

```
InvoiceNo  StockCode  Description                          Quantity  InvoiceDate          UnitPrice  CustomerID  Country
574506     23085      ANTIQUE SILVER BAUBLE LAMP            3        2011-11-04 13:24:00  10.40      12577.0     France     2
552826     82583      HOT BATHS METAL SIGN                  4        2011-05-11 13:20:00  2.10       14277.0     France     2
569332     21035      SET/2 RED RETROSPOT TEA TOWELS        1        2011-10-03 13:46:00  3.25       12637.0     France     2
           22124      SET OF 2 TEA TOWELS PING MICROWAVE    1        2011-10-03 13:46:00  2.95       12637.0     France     2
           22328      ROUND SNACK BOXES SET OF 4 FRUITS     1        2011-10-03 13:46:00  2.95       12637.0     France     2
                                                                                                                          ..
554097     22517      CHILDS GARDEN RAKE PINK               6        2011-05-22 13:01:00  2.10       12567.0     France     1
           22516      CHILDS GARDEN RAKE BLUE               6        2011-05-22 13:01:00  2.10       12567.0     France     1
           22515      CHILDS GARDEN SPADE PINK              6        2011-05-22 13:01:00  2.10       12567.0     France     1
           22514      CHILDS GARDEN SPADE BLUE              6        2011-05-22 13:01:00  2.10       12567.0     France     1
C581316    23174      REGENCY SUGAR BOWL GREEN             -1        2011-12-08 11:46:00  4.15       12523.0     France     1
Length: 8475, dtype: int64
```

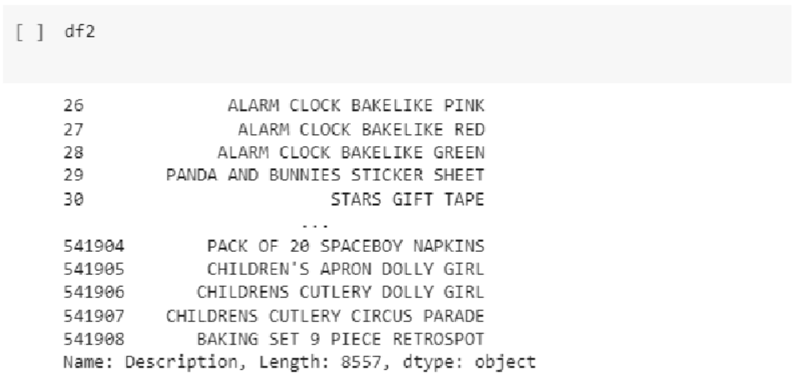
Fig3.2.Extracting



```
df1.head(20)
```

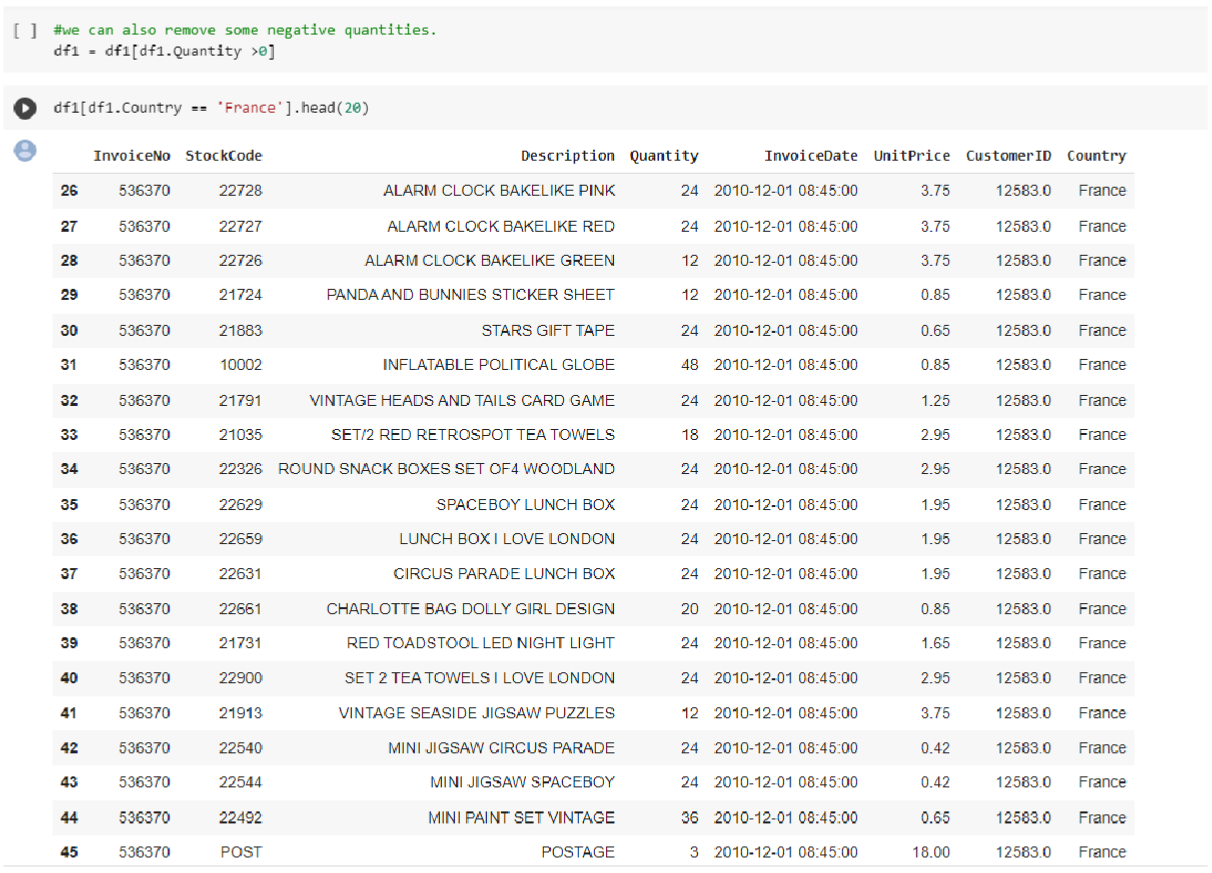
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
26	536370	22728	ALARM CLOCK BAKELIKE PINK	24	2010-12-01 08:45:00	3.75	12583.0	France
27	536370	22727	ALARM CLOCK BAKELIKE RED	24	2010-12-01 08:45:00	3.75	12583.0	France
28	536370	22726	ALARM CLOCK BAKELIKE GREEN	12	2010-12-01 08:45:00	3.75	12583.0	France
29	536370	21724	PANDA AND BUNNIES STICKER SHEET	12	2010-12-01 08:45:00	0.85	12583.0	France
30	536370	21883	STARS GIFT TAPE	24	2010-12-01 08:45:00	0.65	12583.0	France
31	536370	10002	INFLATABLE POLITICAL GLOBE	48	2010-12-01 08:45:00	0.85	12583.0	France
32	536370	21791	VINTAGE HEADS AND TAILS CARD GAME	24	2010-12-01 08:45:00	1.25	12583.0	France
33	536370	21035	SET/2 RED RETROSPOT TEA TOWELS	18	2010-12-01 08:45:00	2.95	12583.0	France
34	536370	22326	ROUND SNACK BOXES SET OF4 WOODLAND	24	2010-12-01 08:45:00	2.95	12583.0	France
35	536370	22629	SPACEBOY LUNCH BOX	24	2010-12-01 08:45:00	1.95	12583.0	France
36	536370	22659	LUNCH BOX I LOVE LONDON	24	2010-12-01 08:45:00	1.95	12583.0	France
37	536370	22631	CIRCUS PARADE LUNCH BOX	24	2010-12-01 08:45:00	1.95	12583.0	France
38	536370	22661	CHARLOTTE BAG DOLLY GIRL DESIGN	20	2010-12-01 08:45:00	0.85	12583.0	France
39	536370	21731	RED TOADSTOOL LED NIGHT LIGHT	24	2010-12-01 08:45:00	1.65	12583.0	France
40	536370	22900	SET 2 TEA TOWELS I LOVE LONDON	24	2010-12-01 08:45:00	2.95	12583.0	France
41	536370	21913	VINTAGE SEASIDE JIGSAW PUZZLES	12	2010-12-01 08:45:00	3.75	12583.0	France
42	536370	22540	MINI JIGSAW CIRCUS PARADE	24	2010-12-01 08:45:00	0.42	12583.0	France
43	536370	22544	MINI JIGSAW SPACEBOY	24	2010-12-01 08:45:00	0.42	12583.0	France
44	536370	22492	MINI PAINT SET VINTAGE	36	2010-12-01 08:45:00	0.65	12583.0	France
45	536370	POST	POSTAGE	3	2010-12-01 08:45:00	18.00	12583.0	France

Fig3.3.Extracting



```
[ ]  df2

     26                ALARM CLOCK BAKELIKE PINK
     27                 ALARM CLOCK BAKELIKE RED
     28               ALARM CLOCK BAKELIKE GREEN
     29          PANDA AND BUNNIES STICKER SHEET
     30                          STARS GIFT TAPE
                                    ...
     541904        PACK OF 20 SPACEBOY NAPKINS
     541905        CHILDREN'S APRON DOLLY GIRL
     541906       CHILDRENS CUTLERY DOLLY GIRL
     541907    CHILDRENS CUTLERY CIRCUS PARADE
     541908      BAKING SET 9 PIECE RETROSPOT
     Name: Description, Length: 8557, dtype: object
```

Fig.3.4.Extracting

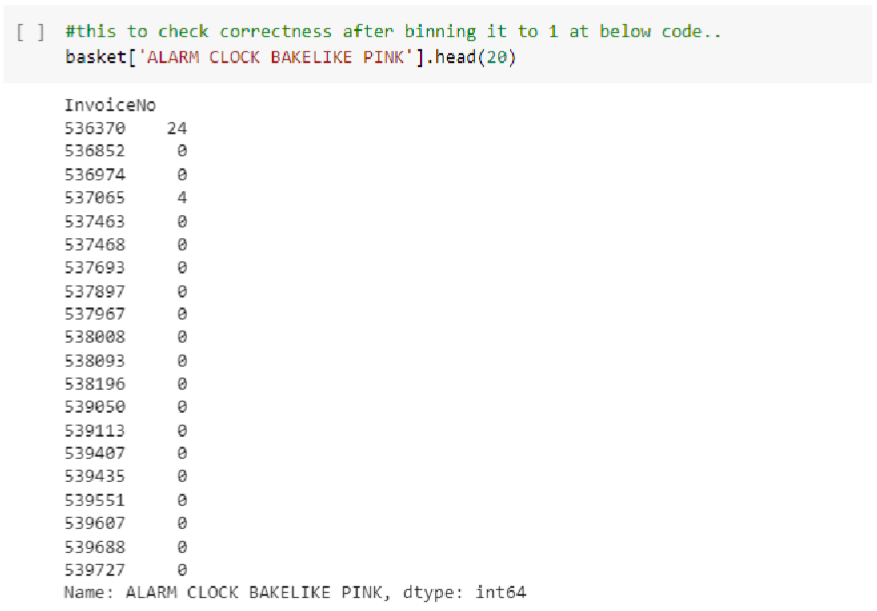


```
[ ]  #we can also remove some negative quantities.
     df1 = df1[df1.Quantity >0]
```

```
df1[df1.Country == 'France'].head(20)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
26	536370	22728	ALARM CLOCK BAKELIKE PINK	24	2010-12-01 08:45:00	3.75	12583.0	France
27	536370	22727	ALARM CLOCK BAKELIKE RED	24	2010-12-01 08:45:00	3.75	12583.0	France
28	536370	22726	ALARM CLOCK BAKELIKE GREEN	12	2010-12-01 08:45:00	3.75	12583.0	France
29	536370	21724	PANDA AND BUNNIES STICKER SHEET	12	2010-12-01 08:45:00	0.85	12583.0	France
30	536370	21883	STARS GIFT TAPE	24	2010-12-01 08:45:00	0.65	12583.0	France
31	536370	10002	INFLATABLE POLITICAL GLOBE	48	2010-12-01 08:45:00	0.85	12583.0	France
32	536370	21791	VINTAGE HEADS AND TAILS CARD GAME	24	2010-12-01 08:45:00	1.25	12583.0	France
33	536370	21035	SET/2 RED RETROSPOT TEA TOWELS	18	2010-12-01 08:45:00	2.95	12583.0	France
34	536370	22326	ROUND SNACK BOXES SET OF4 WOODLAND	24	2010-12-01 08:45:00	2.95	12583.0	France
35	536370	22629	SPACEBOY LUNCH BOX	24	2010-12-01 08:45:00	1.95	12583.0	France
36	536370	22659	LUNCH BOX I LOVE LONDON	24	2010-12-01 08:45:00	1.95	12583.0	France
37	536370	22631	CIRCUS PARADE LUNCH BOX	24	2010-12-01 08:45:00	1.95	12583.0	France
38	536370	22661	CHARLOTTE BAG DOLLY GIRL DESIGN	20	2010-12-01 08:45:00	0.85	12583.0	France
39	536370	21731	RED TOADSTOOL LED NIGHT LIGHT	24	2010-12-01 08:45:00	1.65	12583.0	France
40	536370	22900	SET 2 TEA TOWELS I LOVE LONDON	24	2010-12-01 08:45:00	2.95	12583.0	France
41	536370	21913	VINTAGE SEASIDE JIGSAW PUZZLES	12	2010-12-01 08:45:00	3.75	12583.0	France
42	536370	22540	MINI JIGSAW CIRCUS PARADE	24	2010-12-01 08:45:00	0.42	12583.0	France
43	536370	22544	MINI JIGSAW SPACEBOY	24	2010-12-01 08:45:00	0.42	12583.0	France
44	536370	22492	MINI PAINT SET VINTAGE	36	2010-12-01 08:45:00	0.65	12583.0	France
45	536370	POST	POSTAGE	3	2010-12-01 08:45:00	18.00	12583.0	France

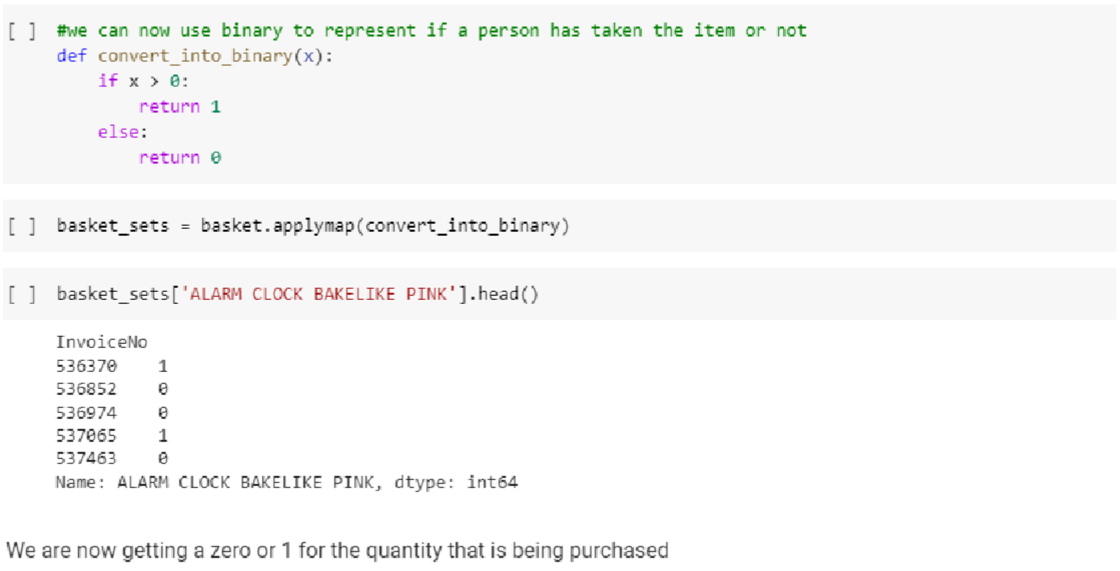
Fig.3.5.Removing some negative quantities.



```
[ ] #this to check correctness after binning it to 1 at below code..
    basket['ALARM CLOCK BAKELIKE PINK'].head(20)

    InvoiceNo
    536370    24
    536852     0
    536974     0
    537065     4
    537463     0
    537468     0
    537693     0
    537897     0
    537967     0
    538008     0
    538093     0
    538196     0
    539050     0
    539113     0
    539407     0
    539435     0
    539551     0
    539607     0
    539688     0
    539727     0
    Name: ALARM CLOCK BAKELIKE PINK, dtype: int64
```

Fig.3.6.Preprocessing

We are now getting a zero or 1 for the quantity that is being purchased



```
[ ] #we can now use binary to represent if a person has taken the item or not
    def convert_into_binary(x):
        if x > 0:
            return 1
        else:
            return 0

[ ] basket_sets = basket.applymap(convert_into_binary)

[ ] basket_sets['ALARM CLOCK BAKELIKE PINK'].head()

    InvoiceNo
    536370    1
    536852    0
    536974    0
    537065    1
    537463    0
    Name: ALARM CLOCK BAKELIKE PINK, dtype: int64
```

We are now getting a zero or 1 for the quantity that is being purchased

Fig3.7.Getting 0 or 1

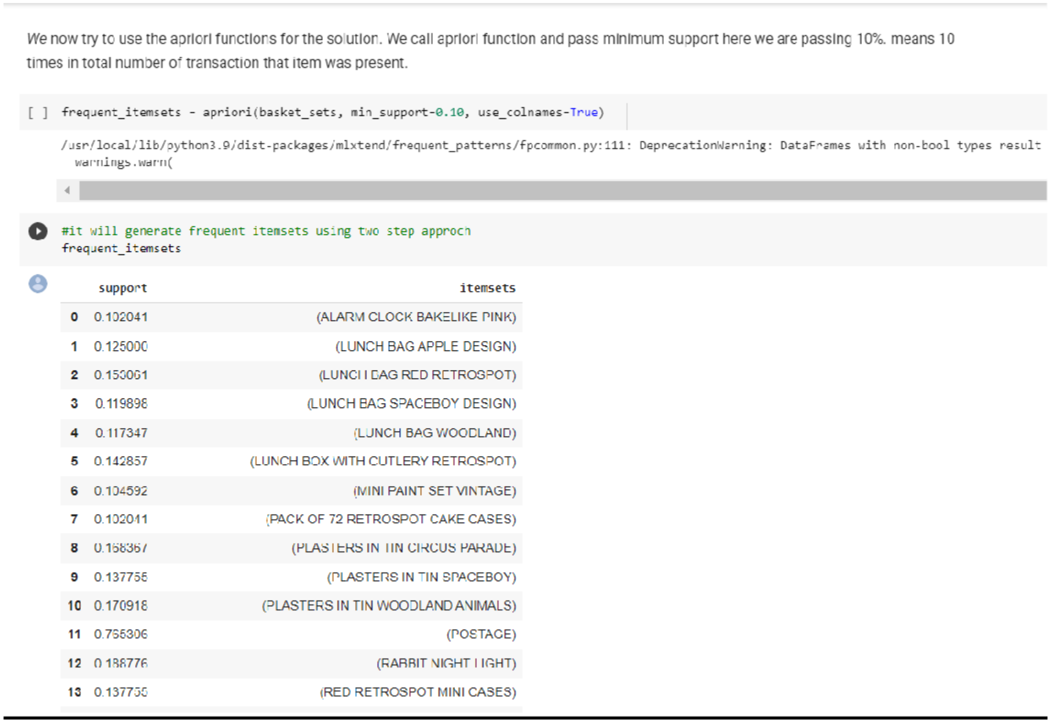
APRIORI ALGORITHM :

Apriori Algorithm is a widely-used and well-known Association Rule algorithm and is a popular algorithm used in market basket analysis. It is also considered accurate and overtop AIS and SETM algorithms. It helps to find frequent itemsets in transactions and identifies association rules between these items. The limitation of the Apriori Algorithm is *frequent itemset generation*. It needs to scan the database many times, leading to increased time and reduced performance as a computationally costly step because of a large dataset. It uses the concepts of Confidence and Support.

In the implementation, first we have installed the packages like 'numpy', 'pandas' imported the packages for running the system. After importing, we have written code for reading the dataset from excel (.csv) file for the predictions.

For showing the results in a better manner, we have plotted the graph so that we can visualize the data graphically and can understand the results, in a better way. For implementing the graph we need to install and import package 'matplotlib'.

We now try to use the Apriori functions for the solution. We call apriori function and pass minimum support here we are passing 10% meaning 10 times in total number of transaction that item was present.



We now try to use the apriori functions for the solution. We call apriori function and pass minimum support here we are passing 10%. means 10 times in total number of transaction that item was present.

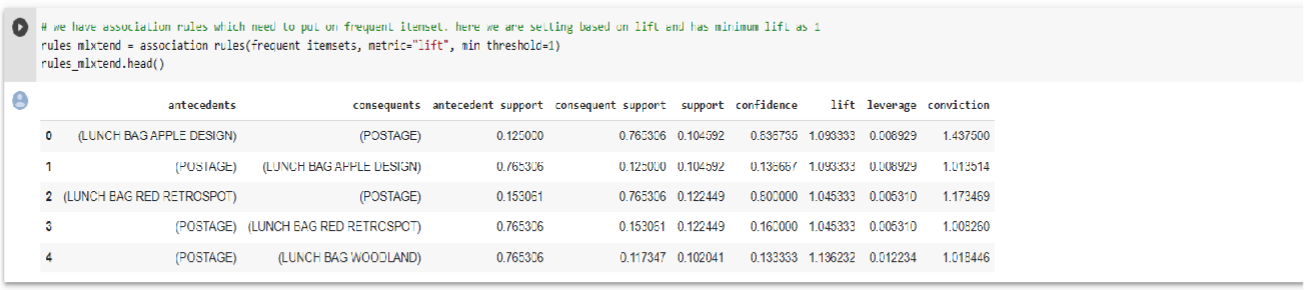
```
[ ] frequent_itemsets = apriori(basket_sets, min_support=0.10, use_colnames=True)

    /usr/local/lib/python3.9/dist-packages/mlxtend/frequent_patterns/fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result
      warnings.warn(
```

```
#it will generate frequent itemsets using two step approch
frequent_itemsets
```

	support	itemsets
0	0.102041	(ALARM CLOCK BAKELIKE PINK)
1	0.125000	(LUNCH BAG APPLE DESIGN)
2	0.153061	(LUNCH BAG RED RETROSPOT)
3	0.119898	(LUNCH BAG SPACEBOY DESIGN)
4	0.117347	(LUNCH BAG WOODLAND)
5	0.142857	(LUNCH BOX WITH CUTLERY RETROSPOT)
6	0.104592	(MINI PAINT SET VINTAGE)
7	0.102041	(PACK OF 72 RETROSPOT CAKE CASES)
8	0.168367	(PLASTERS IN TIN CIRCUS PARADE)
9	0.137755	(PLASTERS IN TIN SPACEBOY)
10	0.170918	(PLASTERS IN TIN WOODLAND ANIMALS)
11	0.765306	(POSTAGE)
12	0.188776	(RABBIT NIGHT LIGHT)
13	0.137755	(RED RETROSPOT MINI CASES)

Fig.4.Implementing appriori algorithm

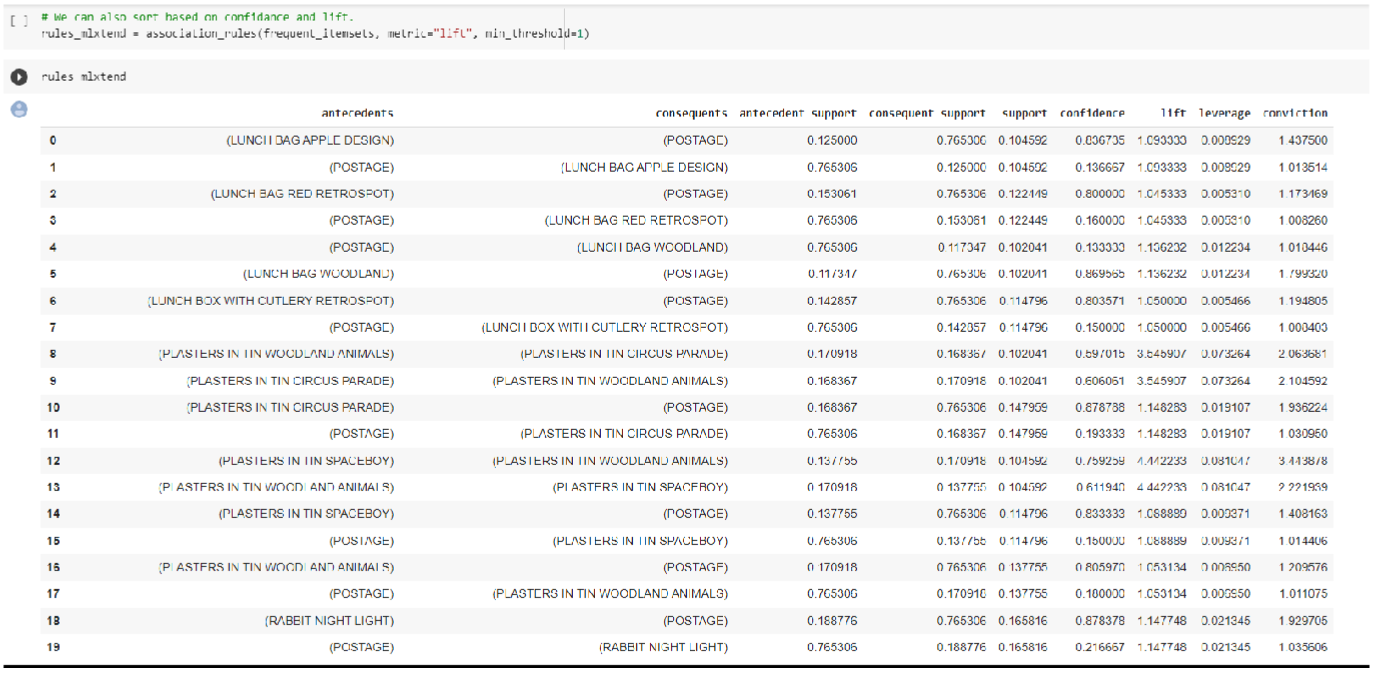
We have association rules which need to put on frequent itemset. here we are setting based on lift and has minimum lift as 1.



```
# we have association rules which need to put on frequent itemset. here we are setting based on lift and has minimum lift as 1
rules mlxtend = association rules(frequent itemsets, metric="lift", min threshold=1)
rules_mlxtend.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(LUNCH BAG APPLE DESIGN)	(POSTAGE)	0.125000	0.765306	0.104592	0.836735	1.093333	0.008929	1.437500
1	(POSTAGE)	(LUNCH BAG APPLE DESIGN)	0.765306	0.125000	0.104592	0.136667	1.093333	0.008929	1.013514
2	(LUNCH BAG RED RETROSPOT)	(POSTAGE)	0.153061	0.765306	0.122449	0.800000	1.045333	0.005310	1.173469
3	(POSTAGE)	(LUNCH BAG RED RETROSPOT)	0.765306	0.153061	0.122449	0.160000	1.045333	0.005310	1.008260
4	(POSTAGE)	(LUNCH BAG WOODLAND)	0.765306	0.117347	0.102041	0.133333	1.136232	0.012234	1.018446

Fig.4.1.Implementing appriori algorithm



```
[ ] # We can also sort based on confidance and lift.
    rules_mlxtend = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
```

```
rules mlxtend
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(LUNCH BAG APPLE DESIGN)	(POSTAGE)	0.125000	0.765306	0.104592	0.836735	1.093333	0.008929	1.437500
1	(POSTAGE)	(LUNCH BAG APPLE DESIGN)	0.765306	0.125000	0.104592	0.136667	1.093333	0.008929	1.013514
2	(LUNCH BAG RED RETROSPOT)	(POSTAGE)	0.153061	0.765306	0.122449	0.800000	1.045333	0.005310	1.173469
3	(POSTAGE)	(LUNCH BAG RED RETROSPOT)	0.765306	0.153061	0.122449	0.160000	1.045333	0.005310	1.008260
4	(POSTAGE)	(LUNCH BAG WOODLAND)	0.765306	0.117347	0.102041	0.133333	1.136232	0.012234	1.010446
5	(LUNCH BAG WOODLAND)	(POSTAGE)	0.117347	0.765306	0.102041	0.869565	1.136232	0.012234	1.799320
6	(LUNCH BOX WITH CUTLERY RETROSPOT)	(POSTAGE)	0.142857	0.765306	0.114796	0.803571	1.050000	0.005466	1.194805
7	(POSTAGE)	(LUNCH BOX WITH CUTLERY RETROSPOT)	0.765306	0.142857	0.114796	0.150000	1.050000	0.005466	1.008403
8	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN CIRCUS PARADE)	0.170918	0.168367	0.102041	0.597015	3.545907	0.073264	2.063681
9	(PLASTERS IN TIN CIRCUS PARADE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.168367	0.170918	0.102041	0.606061	3.545907	0.073264	2.104592
10	(PLASTERS IN TIN CIRCUS PARADE)	(POSTAGE)	0.168367	0.765306	0.147959	0.878788	1.148283	0.019107	1.936224
11	(POSTAGE)	(PLASTERS IN TIN CIRCUS PARADE)	0.765306	0.168367	0.147959	0.193333	1.148283	0.019107	1.030950
12	(PLASTERS IN TIN SPACEBOY)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.137755	0.170918	0.104592	0.759259	4.442233	0.081047	3.443878
13	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN SPACEBOY)	0.170918	0.137755	0.104592	0.611940	4.442233	0.081047	2.221939
14	(PLASTERS IN TIN SPACEBOY)	(POSTAGE)	0.137755	0.765306	0.114796	0.833333	1.088889	0.009371	1.408163
15	(POSTAGE)	(PLASTERS IN TIN SPACEBOY)	0.765306	0.137755	0.114796	0.150000	1.088889	0.009371	1.014406
16	(PLASTERS IN TIN WOODLAND ANIMALS)	(POSTAGE)	0.170918	0.765306	0.137755	0.805970	1.053134	0.006950	1.209576
17	(POSTAGE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.765306	0.170918	0.137755	0.180000	1.053134	0.006950	1.011075
18	(RABBIT NIGHT LIGHT)	(POSTAGE)	0.188776	0.765306	0.165816	0.878378	1.147748	0.021345	1.929705
19	(POSTAGE)	(RABBIT NIGHT LIGHT)	0.765306	0.188776	0.165816	0.216667	1.147748	0.021345	1.035606

Fig.4.2.sorting based on lift&confidence

There will always be incoming and outgoing edges in this graph. The antecedents willbe represented by the incoming edge(s), and the stub (arrow) will be next to the node.

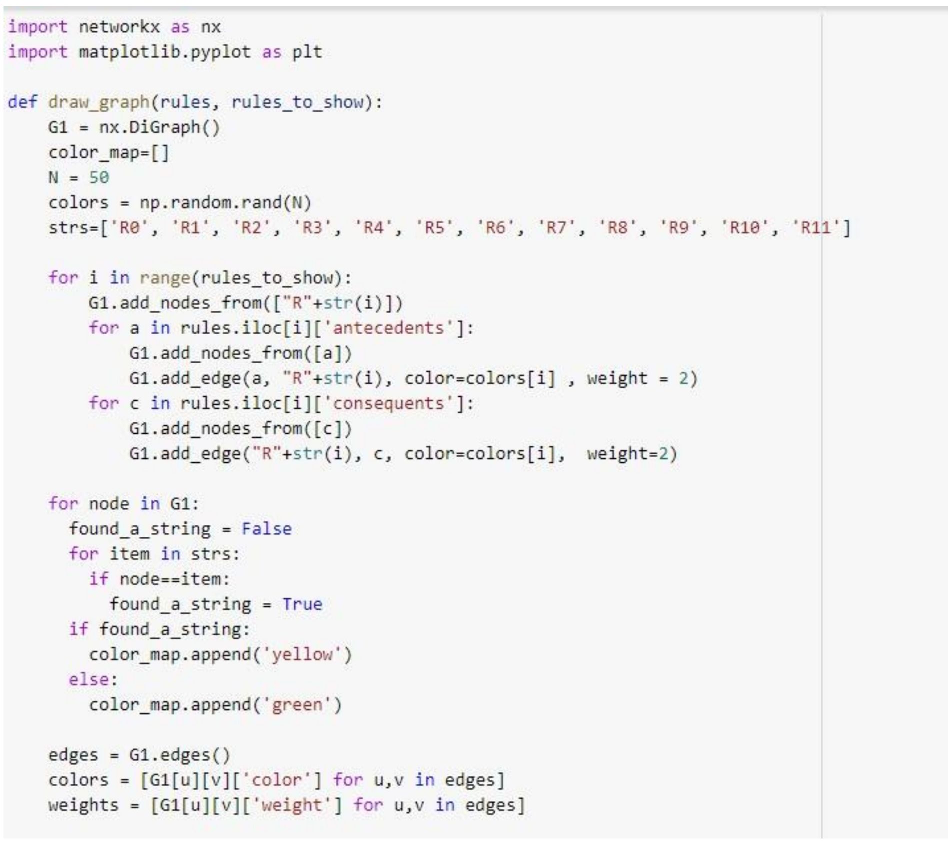
Visualization

For the same values we try to present a seaborn visualization for support and confidence

code:

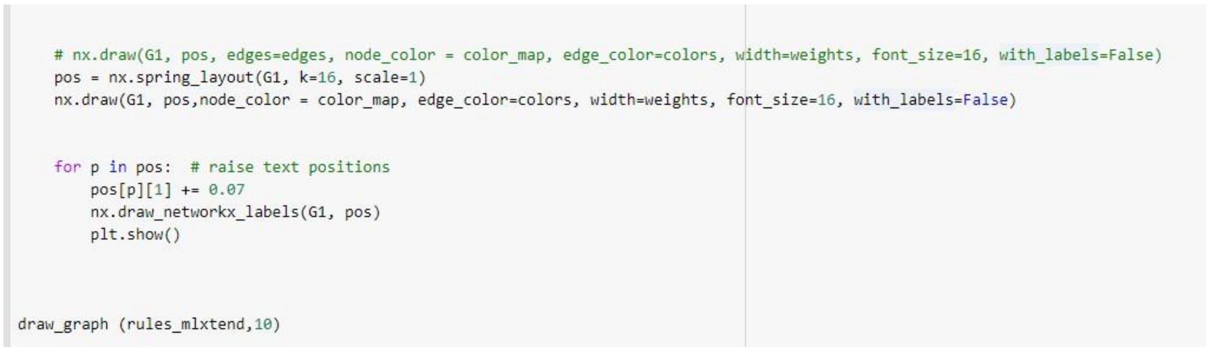
```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(rules, rules_to_show):
    G1 = nx.DiGraph()
    color_map=[]
    N = 50
    colors = np.random.rand(N)
    strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']

    for i in range(rules_to_show):
        G1.add_nodes_from(["R"+str(i)])
        for a in rules.iloc[i]['antecedents']:
            G1.add_nodes_from([a])
            G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)
        for c in rules.iloc[i]['consequents']:
            G1.add_nodes_from([c])
            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

    for node in G1:
      found_a_string = False
      for item in strs:
        if node==item:
          found_a_string = True
      if found_a_string:
      color_map.append('yellow')
      else:
      color_map.append('green')

    edges = G1.edges()
    colors = [G1[u][v]['color'] for u,v in edges]
```

```
  weights = [G1[u][v]['weight'] for u,v in edges]
 # nx.draw(G1, pos, edges=edges, node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)
   pos = nx.spring_layout(G1, k=16, scale=1)
   nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)


   for p in pos: # raise text positions
      pos[p][1] += 0.07
      nx.draw_networkx_labels(G1, pos)
      plt.show()



draw_graph (rules_mlxtend,10)
```



```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(rules, rules_to_show):
    G1 = nx.DiGraph()
    color_map=[]
    N = 50
    colors = np.random.rand(N)
    strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']

    for i in range(rules_to_show):
        G1.add_nodes_from(["R"+str(i)])
        for a in rules.iloc[i]['antecedents']:
            G1.add_nodes_from([a])
            G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)
        for c in rules.iloc[i]['consequents']:
            G1.add_nodes_from([c])
            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

    for node in G1:
      found_a_string = False
      for item in strs:
        if node==item:
          found_a_string = True
      if found_a_string:
        color_map.append('yellow')
      else:
        color_map.append('green')

    edges = G1.edges()
    colors = [G1[u][v]['color'] for u,v in edges]
    weights = [G1[u][v]['weight'] for u,v in edges]
```

Fig.4.3.visualization code



```
    # nx.draw(G1, pos, edges=edges, node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)
    pos = nx.spring_layout(G1, k=16, scale=1)
    nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)


    for p in pos:  # raise text positions
        pos[p][1] += 0.07
        nx.draw_networkx_labels(G1, pos)
        plt.show()


draw_graph (rules_mlxtend,10)
```

Fig.4.4.visualization code

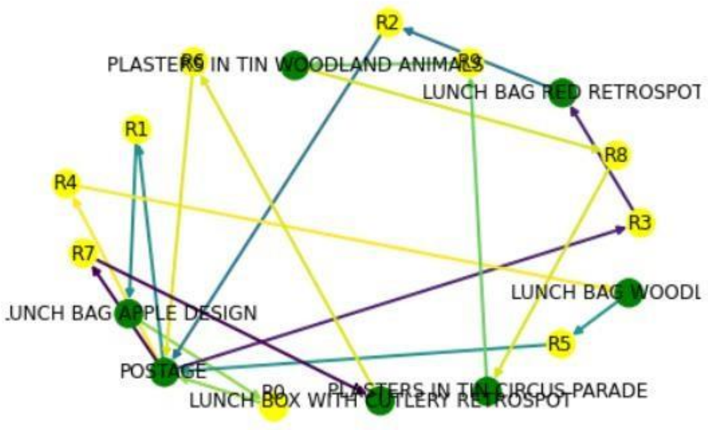

R2
PLASTERS IN TIN WOODLAND ANIMALS
R6
R9
LUNCH BAG RED RETROSPOT
R1
R8
R4
R3
R7
LUNCH BAG WOODL
LUNCH BAG APPLE DESIGN
R5
POSTAGE
R0
LUNCH BOX WITH CUTLERY RETROSPOT
PLASTERS IN TIN CIRCUS PARADE

Fig.4.5.visualization

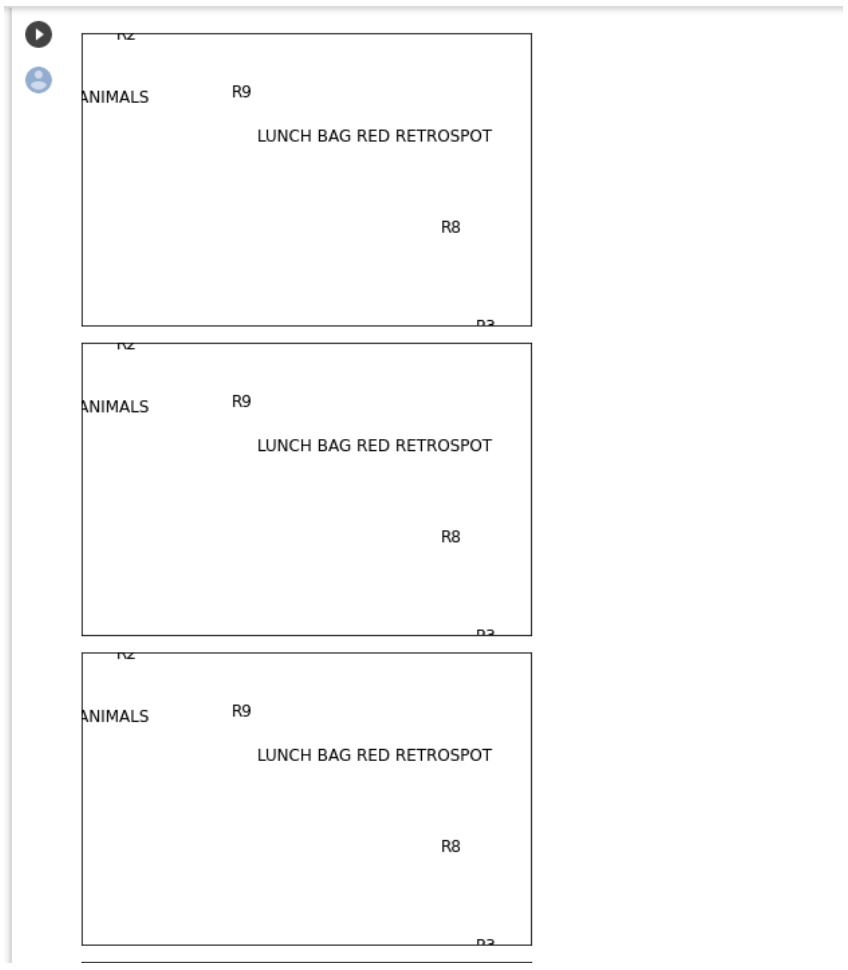


Fig.4.6.Visualized in NetworkX using Matplotlib

ASSOCIATION RULE MINING :

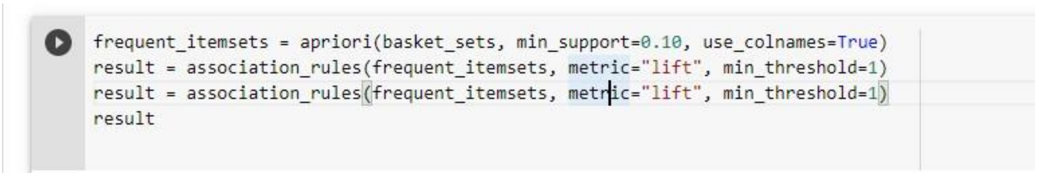
Association rule mining is a data mining technique that is used to identify frequent patterns, relationships, and associations among a set of items in a dataset. It is used to extract useful information from large datasets and identify the correlations between different items.

The process of association rule mining involves two steps: frequent itemset mining and rule generation.

Frequent itemset mining involves identifying sets of items that occur together frequently in the dataset. This is done by setting a minimum support threshold, which specifies the minimum frequency required for an itemset to be considered "frequent". The Apriori algorithm and FP-Growth algorithm are two popular methods used for frequent itemset mining.

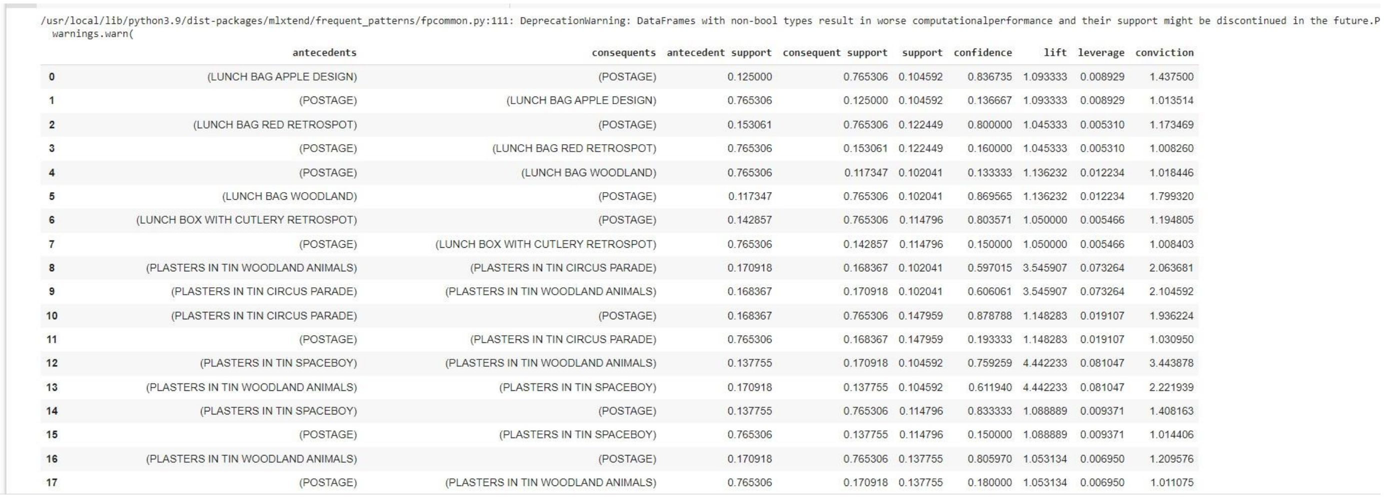
Once the frequent itemsets are identified, association rules can be generated. Association rules are "if-then" statements that express the relationships between different items. For example, an association rule might state "If a customer buys bread and milk, then they are likely to also buy eggs." The strength of an association rule is typically measured by metrics such as support, confidence, and lift.

Association rule mining has numerous applications in a wide range of fields such as marketing, customer behavior analysis, recommendation systems, and web usage analysis. By identifying patterns and relationships among items, association rule mining can help businesses to better understand their customers' behavior and preferences, and make data-driven decisions to improve their operations and sales strategies.



```
frequent_itemsets = apriori(basket_sets, min_support=0.10, use_colnames=True)
result = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
result = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
result
```

Fig.5.Implementing association rule mining



```
/usr/local/lib/python3.9/dist-packages/mlxtend/frequent_patterns/fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.P
  warnings.warn(
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(LUNCH BAG APPLE DESIGN)	(POSTAGE)	0.125000	0.765306	0.104592	0.836735	1.093333	0.008929	1.437500
1	(POSTAGE)	(LUNCH BAG APPLE DESIGN)	0.765306	0.125000	0.104592	0.136667	1.093333	0.008929	1.013514
2	(LUNCH BAG RED RETROSPOT)	(POSTAGE)	0.153061	0.765306	0.122449	0.800000	1.045333	0.005310	1.173469
3	(POSTAGE)	(LUNCH BAG RED RETROSPOT)	0.765306	0.153061	0.122449	0.160000	1.045333	0.005310	1.008260
4	(POSTAGE)	(LUNCH BAG WOODLAND)	0.765306	0.117347	0.102041	0.133333	1.136232	0.012234	1.018446
5	(LUNCH BAG WOODLAND)	(POSTAGE)	0.117347	0.765306	0.102041	0.869565	1.136232	0.012234	1.799320
6	(LUNCH BOX WITH CUTLERY RETROSPOT)	(POSTAGE)	0.142857	0.765306	0.114796	0.803571	1.050000	0.005466	1.194805
7	(POSTAGE)	(LUNCH BOX WITH CUTLERY RETROSPOT)	0.765306	0.142857	0.114796	0.150000	1.050000	0.005466	1.008403
8	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN CIRCUS PARADE)	0.170918	0.168367	0.102041	0.597015	3.545907	0.073264	2.063681
9	(PLASTERS IN TIN CIRCUS PARADE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.168367	0.170918	0.102041	0.606061	3.545907	0.073264	2.104592
10	(PLASTERS IN TIN CIRCUS PARADE)	(POSTAGE)	0.168367	0.765306	0.147959	0.878788	1.148283	0.019107	1.936224
11	(POSTAGE)	(PLASTERS IN TIN CIRCUS PARADE)	0.765306	0.168367	0.147959	0.193333	1.148283	0.019107	1.030950
12	(PLASTERS IN TIN SPACEBOY)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.137755	0.170918	0.104592	0.759259	4.442233	0.081047	3.443878
13	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN SPACEBOY)	0.170918	0.137755	0.104592	0.611940	4.442233	0.081047	2.221939
14	(PLASTERS IN TIN SPACEBOY)	(POSTAGE)	0.137755	0.765306	0.114796	0.833333	1.088889	0.009371	1.408163
15	(POSTAGE)	(PLASTERS IN TIN SPACEBOY)	0.765306	0.137755	0.114796	0.150000	1.088889	0.009371	1.014406
16	(PLASTERS IN TIN WOODLAND ANIMALS)	(POSTAGE)	0.170918	0.765306	0.137755	0.805970	1.053134	0.006950	1.209576
17	(POSTAGE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.765306	0.170918	0.137755	0.180000	1.053134	0.006950	1.011075

Fig.5.1.Implementing association rule mining

Visualization

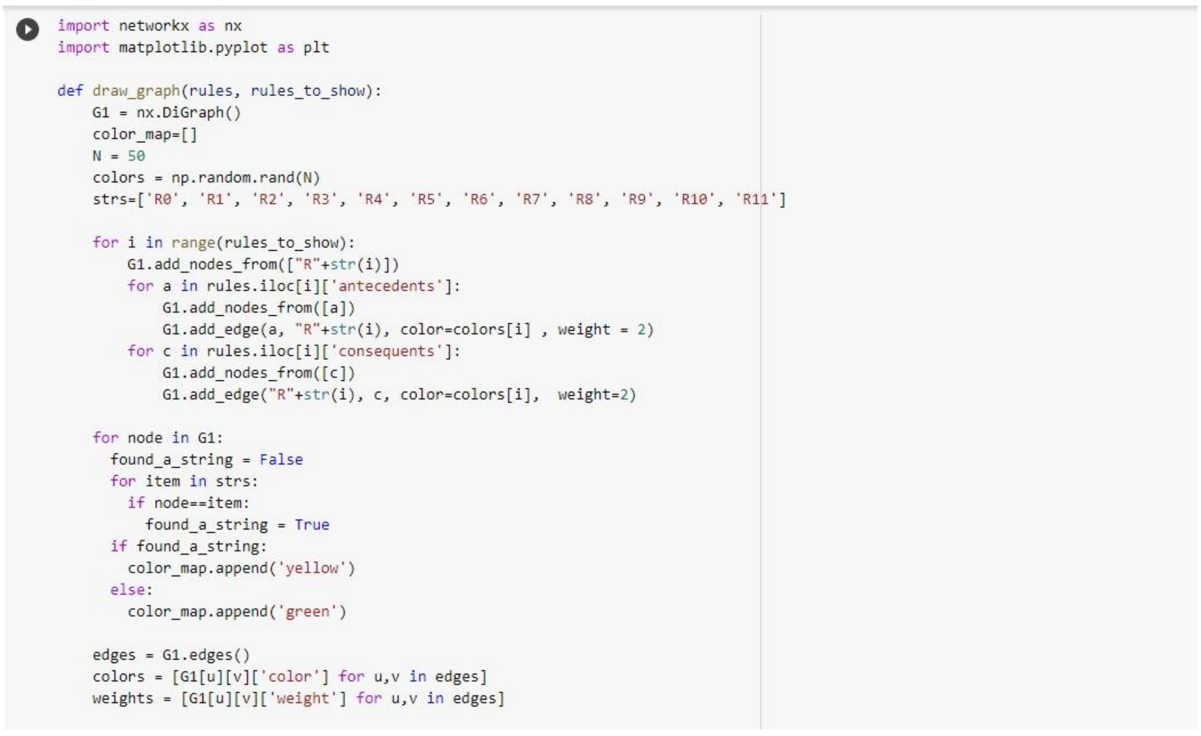
For the same values we try to present a seaborn visualization for support and confidence

code:

```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(rules, rules_to_show):
    G1 = nx.DiGraph()
    color_map=[]
    N = 50
    colors = np.random.rand(N)
    strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']

    for i in range(rules_to_show):
        G1.add_nodes_from(["R"+str(i)])
        for a in rules.iloc[i]['antecedents']:
            G1.add_nodes_from([a])
            G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)
        for c in rules.iloc[i]['consequents']:
            G1.add_nodes_from([c])
            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

    for node in G1:
      found_a_string = False
      for item in strs:
        if node==item:
          found_a_string = True
      if found_a_string:
        color_map.append('yellow')
      else:
        color_map.append('green')

    edges = G1.edges()
    colors = [G1[u][v]['color'] for u,v in edges]
    weights = [G1[u][v]['weight'] for u,v in edges]


    # nx.draw(G1, pos, edges=edges, node_color = color_map, edge_color=colors, width=wei
ghts, font_size=16, with_labels=False)
    pos = nx.spring_layout(G1, k=16, scale=1)
    nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=1
6, with_labels=False)
      for p in pos:  # raise text positions
```

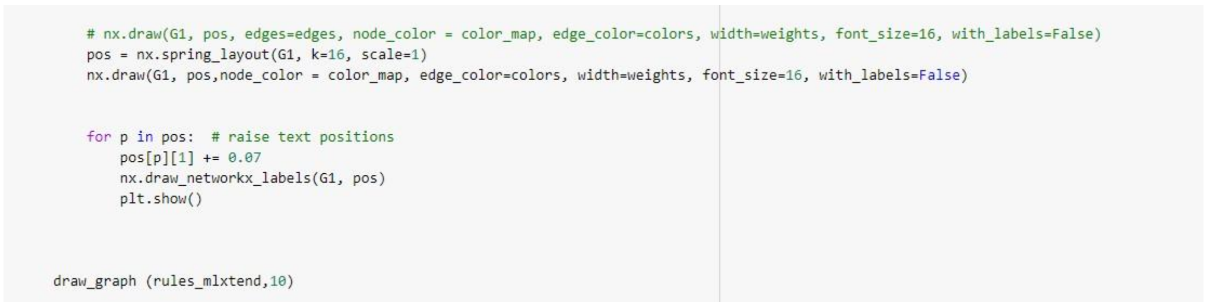
```
pos[p][1] += 0.07
        nx.draw_networkx_labels(G1, pos)
        plt.show()
```

```
draw_graph (rules_mlxtend,10)
```



```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(rules, rules_to_show):
    G1 = nx.DiGraph()
    color_map=[]
    N = 50
    colors = np.random.rand(N)
    strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']

    for i in range(rules_to_show):
        G1.add_nodes_from(["R"+str(i)])
        for a in rules.iloc[i]['antecedents']:
            G1.add_nodes_from([a])
            G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)
        for c in rules.iloc[i]['consequents']:
            G1.add_nodes_from([c])
            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

    for node in G1:
      found_a_string = False
      for item in strs:
        if node==item:
          found_a_string = True
      if found_a_string:
        color_map.append('yellow')
      else:
        color_map.append('green')

    edges = G1.edges()
    colors = [G1[u][v]['color'] for u,v in edges]
    weights = [G1[u][v]['weight'] for u,v in edges]
```

Fig.5.2.visualization code



```
    # nx.draw(G1, pos, edges=edges, node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)
    pos = nx.spring_layout(G1, k=16, scale=1)
    nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)


    for p in pos:  # raise text positions
        pos[p][1] += 0.07
        nx.draw_networkx_labels(G1, pos)
        plt.show()



draw_graph (rules_mlxtend,10)
```

Fig.5.3.visualization code

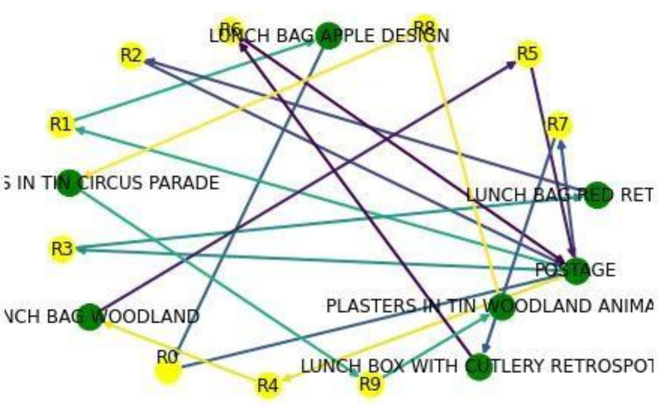


Fig.5.4.visualization

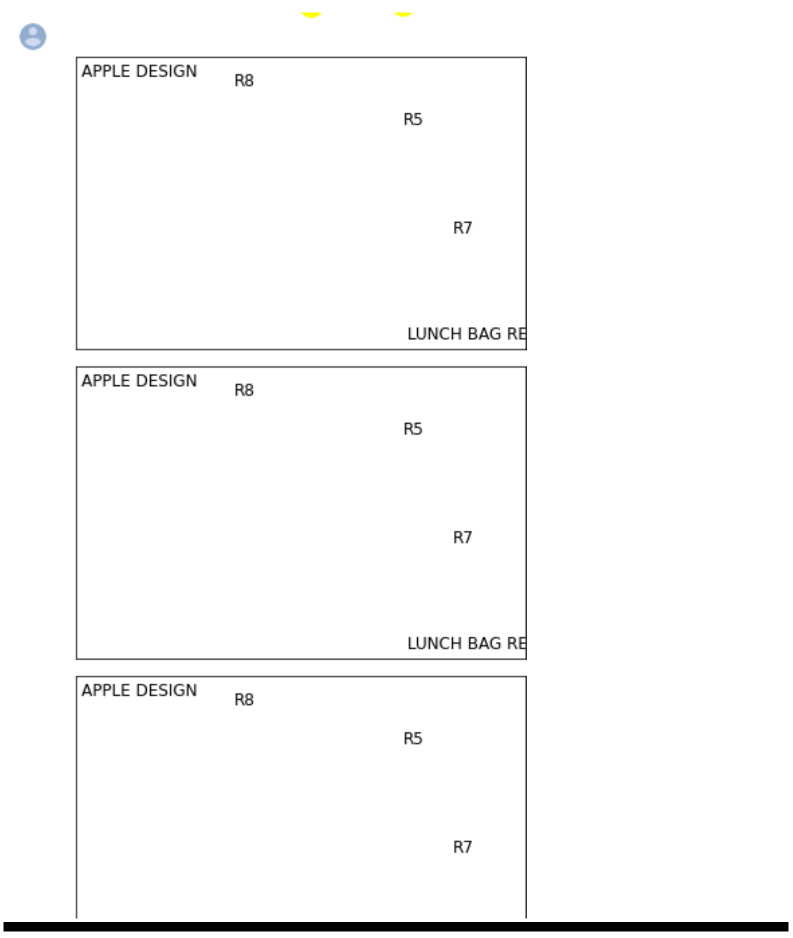


Fig.5.5.Visualized in NetworkX using Matplotlib

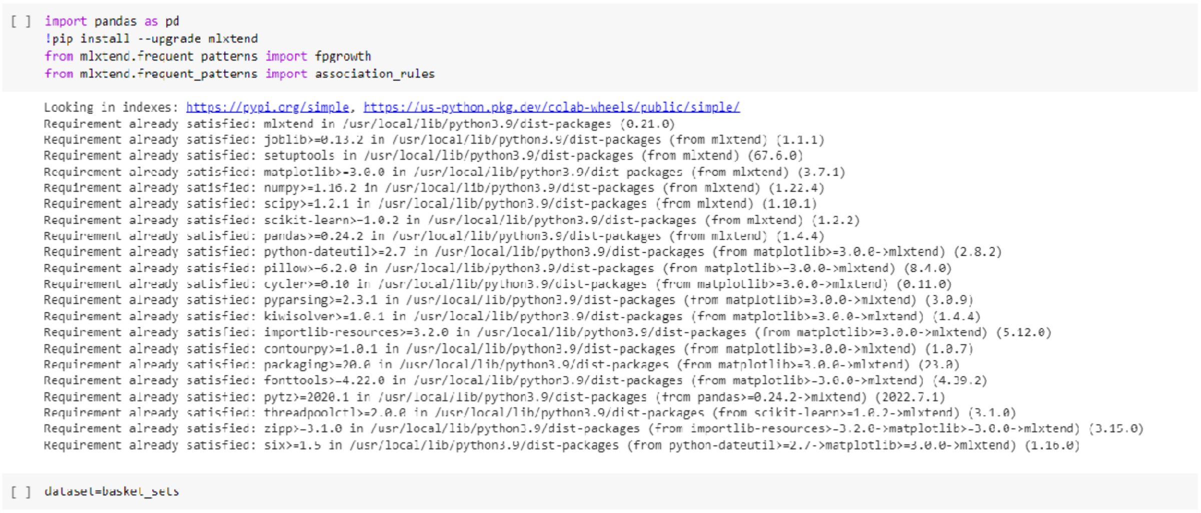
FP ALGORITHM

FP algorithm refers to frequent pattern mining algorithms that are used to find the most frequent patterns or itemsets in a given dataset. These algorithms are widely used in data mining, machine learning, and business intelligence applications to identify relationships and patterns in large datasets.

FP algorithm is an efficient method to discover frequent patterns in transactional databases. The idea behind the algorithm is to first identify the frequent items, i.e., the items that occur frequently in the transactions. Then, the algorithm generates the candidate itemsets of size k by joining the frequent itemsets of size k-1. Finally, the algorithm counts the support of each candidate itemset, i.e., the number of transactions that contain the itemset, and prunes the candidate itemsets with support less than a predefined threshold.

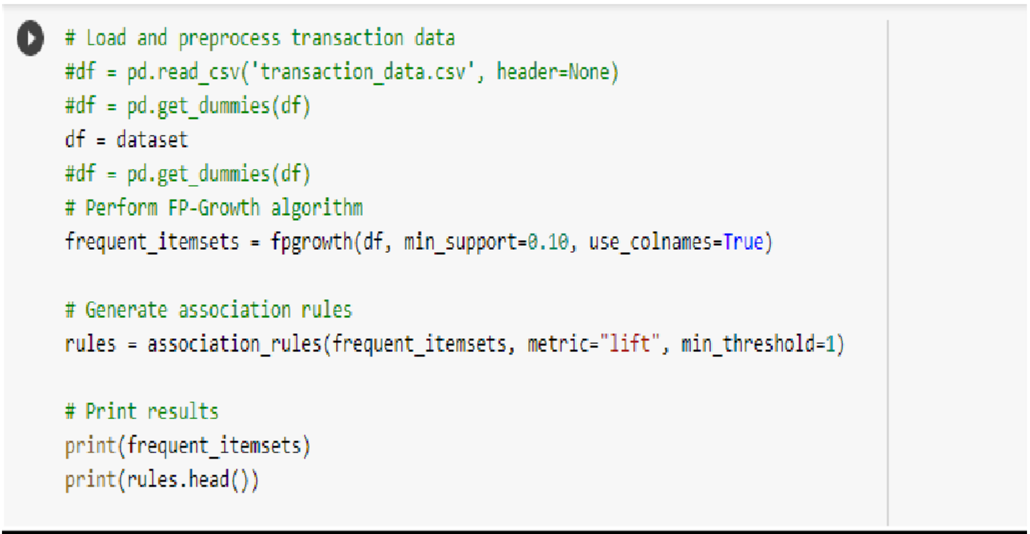
popular FP algorithm is the FP-Growth algorithm. This algorithm builds a frequent pattern tree (FP-tree) to represent the dataset and uses this tree to efficiently mine frequent patterns. The FP-tree is built by first finding the frequent items in the dataset and then sorting the transactions based on their frequent items. The frequent items are then removed from each transaction and the remaining items are added to the tree. The algorithm then recursively constructs conditional FP-trees for each frequent item in the tree and mines frequent itemsets from these trees.

Overall, FP algorithms are widely used in various applications to discover frequent patterns in large datasets efficiently. These algorithms help in identifying relationships and patterns that can be used for various purposes, such as market basket analysis, product recommendations, and fraud detection.



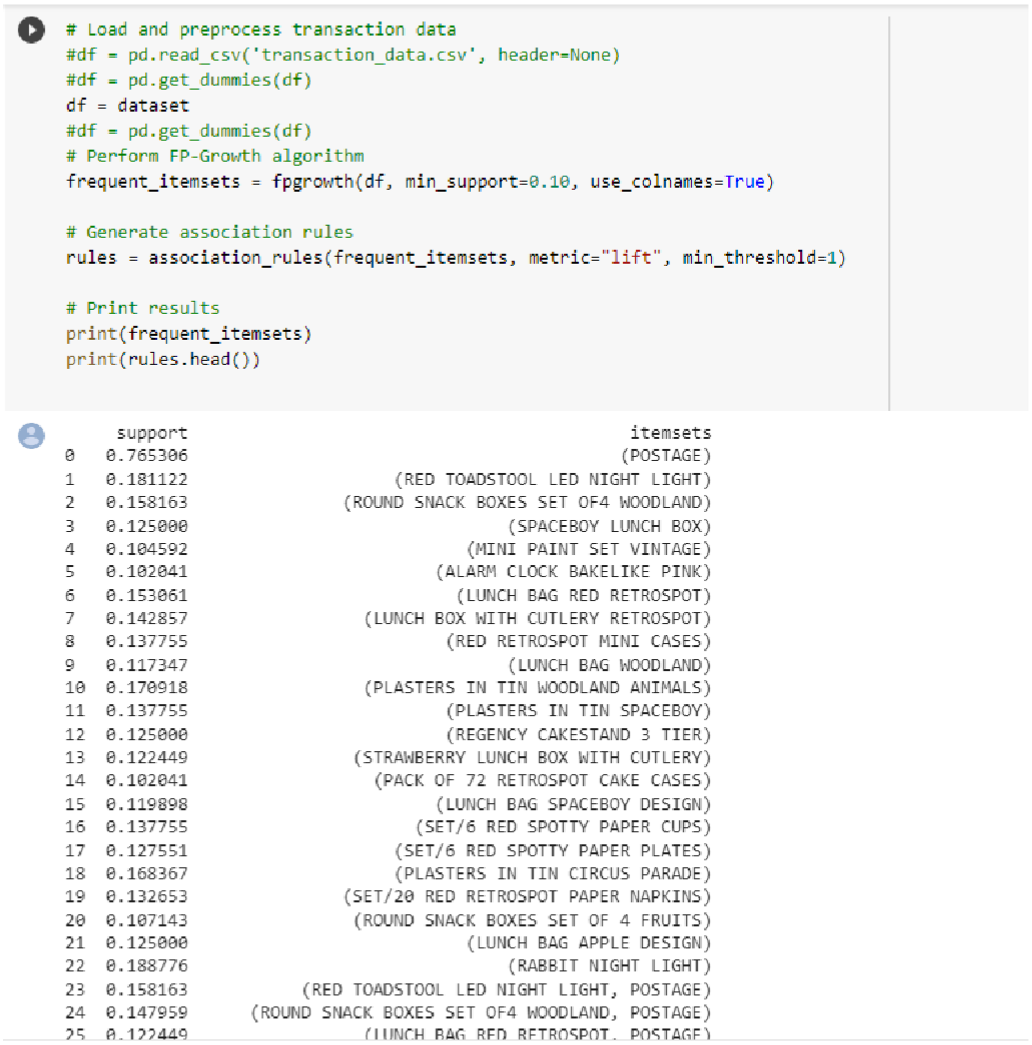
```
[ ] import pandas as pd
    !pip install --upgrade mlxtend
    from mlxtend.frequent patterns import fpgrowth
    from mlxtend.frequent_patterns import association_rules

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: mlxtend in /usr/local/lib/python3.9/dist-packages (0.21.0)
    Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.1.1)
    Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from mlxtend) (67.6.0)
    Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.9/dist packages (from mlxtend) (3.7.1)
    Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.22.4)
    Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.10.1)
    Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.2.2)
    Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.4.4)
    Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
    Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (8.4.0)
    Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
    Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
    Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
    Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (5.12.0)
    Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.0.7)
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (23.0)
    Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (4.39.2)
    Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.2->mlxtend) (2022.7.1)
    Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn>=1.0.2->mlxtend) (3.1.0)
    Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib>=3.0.0->mlxtend) (3.15.0)
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)

[ ] dataset=basket_sets
```

Fig.6.Importing packages



```python
# Load and preprocess transaction data
#df = pd.read_csv('transaction_data.csv', header=None)
#df = pd.get_dummies(df)
df = dataset
#df = pd.get_dummies(df)
# Perform FP-Growth algorithm
frequent_itemsets = fpgrowth(df, min_support=0.10, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Print results
print(frequent_itemsets)
print(rules.head())
```

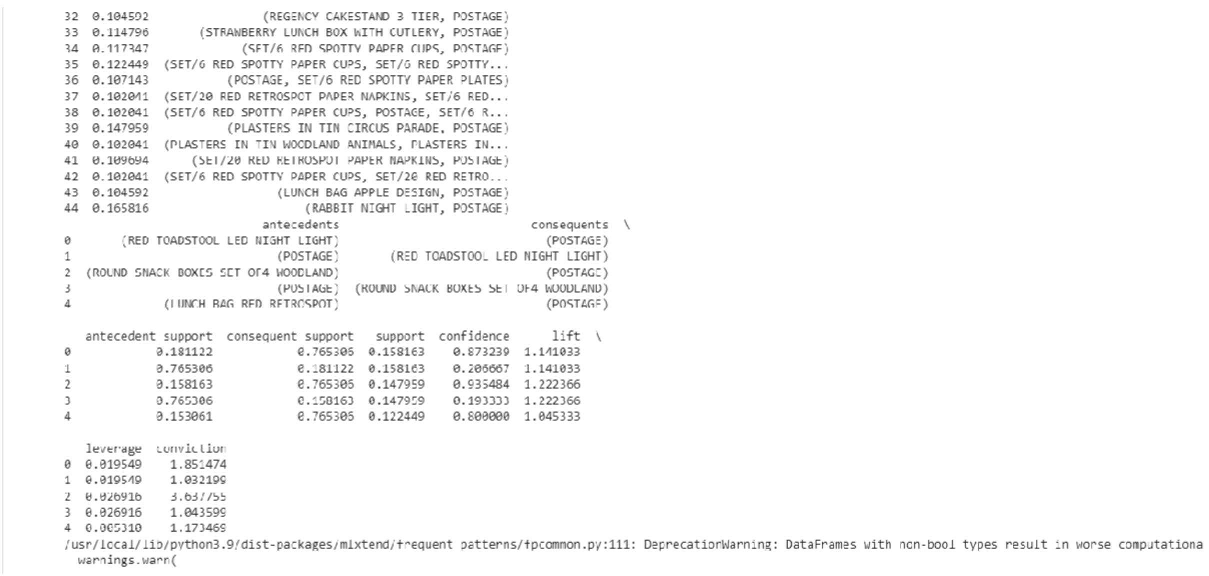
Fig.6.1.Implementing FP algorithm



```
# Load and preprocess transaction data
#df = pd.read_csv('transaction_data.csv', header=None)
#df = pd.get_dummies(df)
df = dataset
#df = pd.get_dummies(df)
# Perform FP-Growth algorithm
frequent_itemsets = fpgrowth(df, min_support=0.10, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Print results
print(frequent_itemsets)
print(rules.head())
```

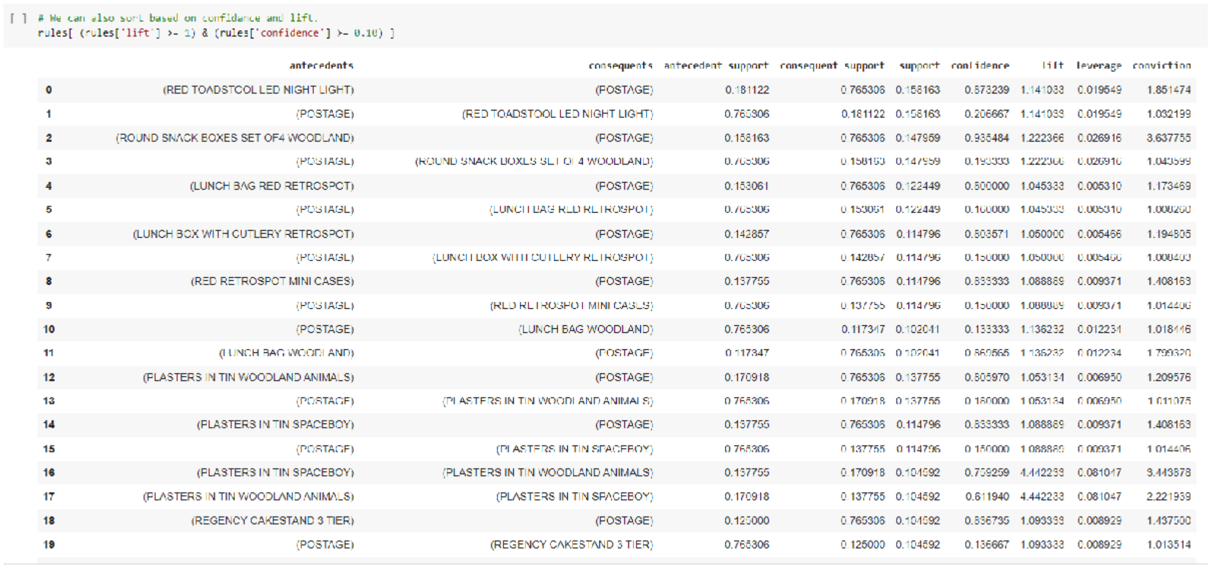
```
     support                                     itemsets
0   0.765306                                    (POSTAGE)
1   0.181122                    (RED TOADSTOOL LED NIGHT LIGHT)
2   0.158163               (ROUND SNACK BOXES SET OF4 WOODLAND)
3   0.125000                          (SPACEBOY LUNCH BOX)
4   0.104592                        (MINI PAINT SET VINTAGE)
5   0.102041                    (ALARM CLOCK BAKELIKE PINK)
6   0.153061                      (LUNCH BAG RED RETROSPOT)
7   0.142857              (LUNCH BOX WITH CUTLERY RETROSPOT)
8   0.137755                     (RED RETROSPOT MINI CASES)
9   0.117347                          (LUNCH BAG WOODLAND)
10  0.170918              (PLASTERS IN TIN WOODLAND ANIMALS)
11  0.137755                     (PLASTERS IN TIN SPACEBOY)
12  0.125000                     (REGENCY CAKESTAND 3 TIER)
13  0.122449              (STRAWBERRY LUNCH BOX WITH CUTLERY)
14  0.102041               (PACK OF 72 RETROSPOT CAKE CASES)
15  0.119898                   (LUNCH BAG SPACEBOY DESIGN)
16  0.137755                   (SET/6 RED SPOTTY PAPER CUPS)
17  0.127551                 (SET/6 RED SPOTTY PAPER PLATES)
18  0.168367                 (PLASTERS IN TIN CIRCUS PARADE)
19  0.132653              (SET/20 RED RETROSPOT PAPER NAPKINS)
20  0.107143              (ROUND SNACK BOXES SET OF 4 FRUITS)
21  0.125000                      (LUNCH BAG APPLE DESIGN)
22  0.188776                         (RABBIT NIGHT LIGHT)
23  0.158163           (RED TOADSTOOL LED NIGHT LIGHT, POSTAGE)
24  0.147959      (ROUND SNACK BOXES SET OF4 WOODLAND, POSTAGE)
25  0.122449                (LUNCH BAG RED RETROSPOT, POSTAGE)
```

Fig.6.2.Implementing FP algorithm



```
32  0.104592              (REGENCY CAKESTAND 3 TIER, POSTAGE)
33  0.114796        (STRAWBERRY LUNCH BOX WITH CUTLERY, POSTAGE)
34  0.117347             (SET/6 RED SPOTTY PAPER CUPS, POSTAGE)
35  0.122449  (SET/6 RED SPOTTY PAPER CUPS, SET/6 RED SPOTTY...
36  0.107143           (POSTAGE, SET/6 RED SPOTTY PAPER PLATES)
37  0.102041  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...
38  0.102041  (SET/6 RED SPOTTY PAPER CUPS, POSTAGE, SET/6 R...
39  0.147959           (PLASTERS IN TIN CIRCUS PARADE, POSTAGE)
40  0.102041  (PLASTERS IN TIN WOODLAND ANIMALS, PLASTERS IN...
41  0.109694      (SET/20 RED RETROSPOT PAPER NAPKINS, POSTAGE)
42  0.102041  (SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...
43  0.104592                  (LUNCH BAG APPLE DESIGN, POSTAGE)
44  0.165816                      (RABBIT NIGHT LIGHT, POSTAGE)
                       antecedents                           consequents  \
0      (RED TOADSTOOL LED NIGHT LIGHT)                             (POSTAGE)
1                            (POSTAGE)       (RED TOADSTOOL LED NIGHT LIGHT)
2  (ROUND SNACK BOXES SET OF4 WOODLAND)                            (POSTAGE)
3                            (POSTAGE)  (ROUND SNACK BOXES SET OF4 WOODLAND)
4           (LUNCH BAG RED RETROSPOT)                              (POSTAGE)

   antecedent support  consequent support   support  confidence      lift  \
0            0.181122            0.765306  0.158163    0.873239  1.141033
1            0.765306            0.181122  0.158163    0.206667  1.141033
2            0.158163            0.765306  0.147959    0.935484  1.222366
3            0.765306            0.158163  0.147959    0.193333  1.222366
4            0.153061            0.765306  0.122449    0.800000  1.045333

   leverage  conviction
0  0.019549    1.851474
1  0.019549    1.032199
2  0.026916    3.637755
3  0.026916    1.043599
4  0.005310    1.173469
/usr/local/lib/python3.9/dist-packages/mlxtend/frequent_patterns/fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result in worse computationa
  warnings.warn(
```

Fig.6.3.Implementing FP algorithm



```
[ ] # We can also sort based on confidence and lift.
    rules[ (rules['lift'] >= 1) & (rules['confidence'] >= 0.10) ]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(RED TOADSTOOL LED NIGHT LIGHT)	(POSTAGE)	0.181122	0.765306	0.158163	0.873239	1.141033	0.019549	1.851474
1	(POSTAGE)	(RED TOADSTOOL LED NIGHT LIGHT)	0.765306	0.181122	0.158163	0.206667	1.141033	0.019549	1.032199
2	(ROUND SNACK BOXES SET OF4 WOODLAND)	(POSTAGE)	0.158163	0.765306	0.147959	0.935484	1.222366	0.026916	3.637755
3	(POSTAGE)	(ROUND SNACK BOXES SET OF 4 WOODLAND)	0.765306	0.158163	0.147959	0.193333	1.222366	0.026916	1.043599
4	(LUNCH BAG RED RETROSPOT)	(POSTAGE)	0.153061	0.765306	0.122449	0.800000	1.045333	0.005310	1.173469
5	(POSTAGE)	(LUNCH BAG RED RETROSPOT)	0.765306	0.153061	0.122449	0.160000	1.045333	0.005310	1.008260
6	(LUNCH BOX WITH CUTLERY RETROSPOT)	(POSTAGE)	0.142857	0.765306	0.114796	0.803571	1.050000	0.005466	1.194805
7	(POSTAGE)	(LUNCH BOX WITH CUTLERY RETROSPOT)	0.765306	0.142857	0.114796	0.150000	1.050000	0.005466	1.008403
8	(RED RETROSPOT MINI CASES)	(POSTAGE)	0.137755	0.765306	0.114796	0.833333	1.088889	0.009371	1.408163
9	(POSTAGE)	(RED RETROSPOT MINI CASES)	0.765306	0.137755	0.114796	0.150000	1.088889	0.009371	1.014406
10	(POSTAGE)	(LUNCH BAG WOODLAND)	0.765306	0.117347	0.102041	0.133333	1.136232	0.012234	1.018446
11	(LUNCH BAG WOODLAND)	(POSTAGE)	0.117347	0.765306	0.102041	0.869565	1.136232	0.012234	1.799320
12	(PLASTERS IN TIN WOODLAND ANIMALS)	(POSTAGE)	0.170918	0.765306	0.137755	0.805970	1.053134	0.006950	1.209576
13	(POSTAGE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.765306	0.170918	0.137755	0.180000	1.053134	0.006950	1.011075
14	(PLASTERS IN TIN SPACEBOY)	(POSTAGE)	0.137755	0.765306	0.114796	0.833333	1.088889	0.009371	1.408163
15	(POSTAGE)	(PLASTERS IN TIN SPACEBOY)	0.765306	0.137755	0.114796	0.150000	1.088889	0.009371	1.014406
16	(PLASTERS IN TIN SPACEBOY)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.137755	0.170918	0.104592	0.759259	4.442233	0.081047	3.443878
17	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN SPACEBOY)	0.170918	0.137755	0.104592	0.611940	4.442233	0.081047	2.221939
18	(REGENCY CAKESTAND 3 TIER)	(POSTAGE)	0.125000	0.765306	0.104592	0.836735	1.093333	0.008929	1.437500
19	(POSTAGE)	(REGENCY CAKESTAND 3 TIER)	0.765306	0.125000	0.104592	0.136667	1.093333	0.008929	1.013514

Fig.6.4.sorting based on confidence and lift

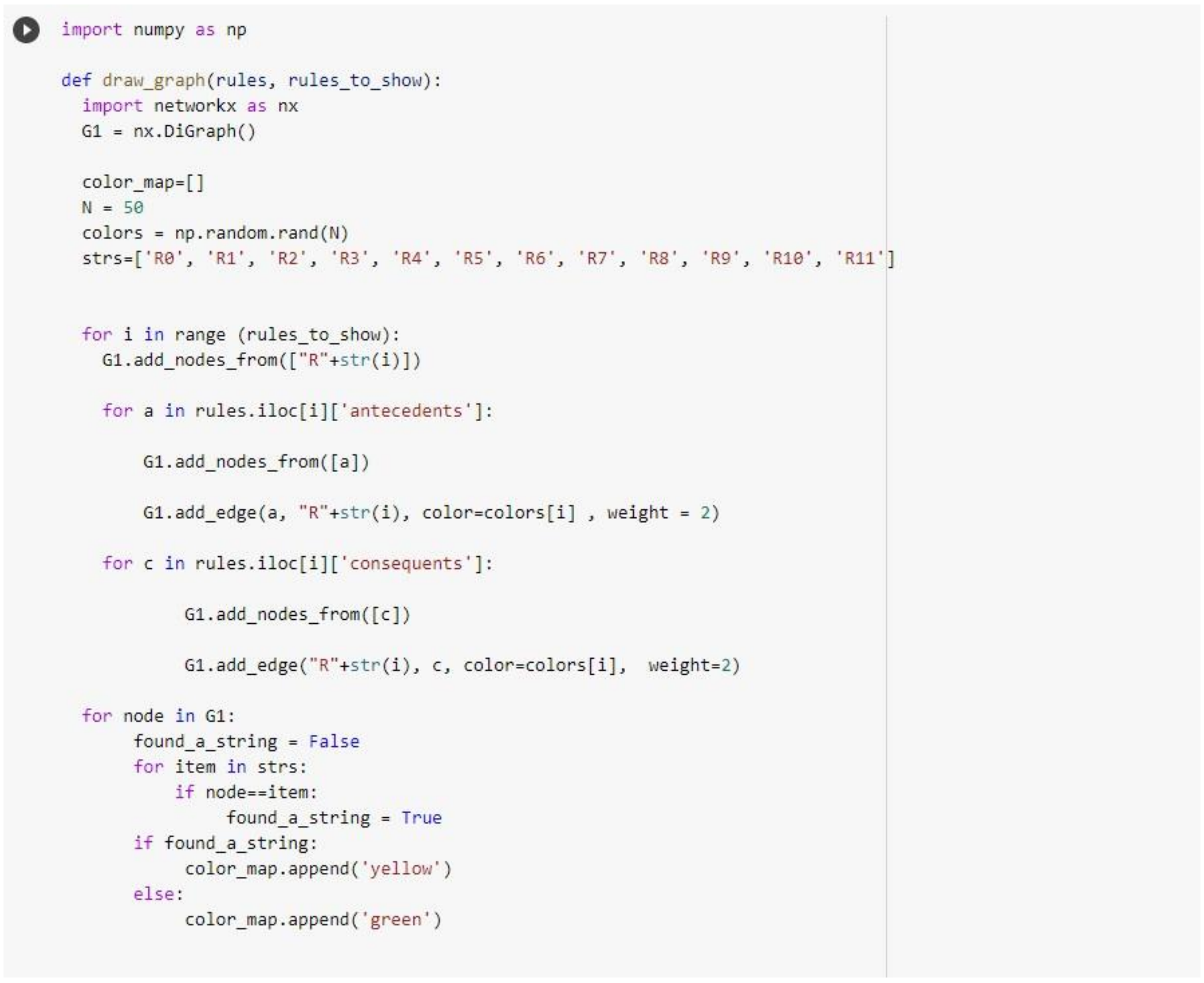
Visualization

For the same values we try to present a seaborn visualization for support and confidence

code:

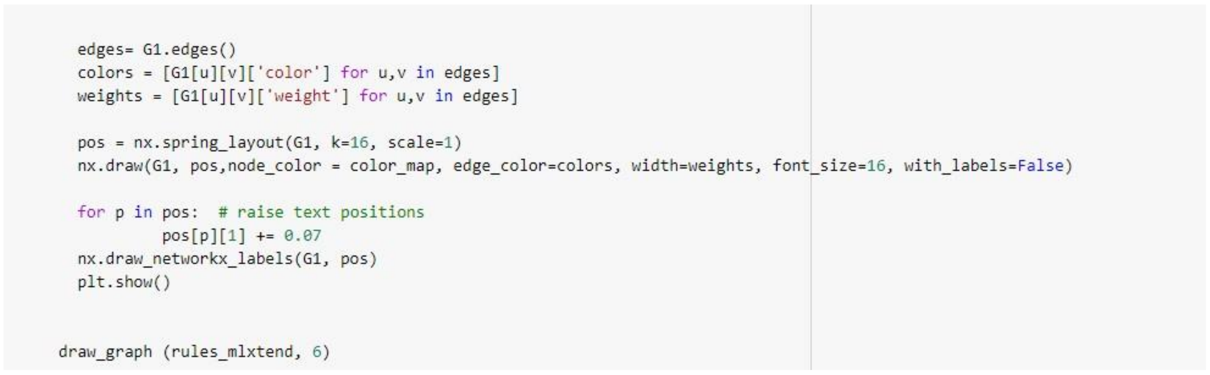
```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(rules, rules_to_show):
    G1 = nx.DiGraph()
    color_map=[]
    N = 50
    colors = np.random.rand(N)
    strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']

    for i in range(rules_to_show):
        G1.add_nodes_from(["R"+str(i)])
        for a in rules.iloc[i]['antecedents']:
            G1.add_nodes_from([a])
            G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)
        for c in rules.iloc[i]['consequents']:
            G1.add_nodes_from([c])
            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

    for node in G1:
      found_a_string = False
      for item in strs:
        if node==item:
          found_a_string = True
      if found_a_string:
        color_map.append('yellow')
      else:
        color_map.append('green')

    edges = G1.edges()
    colors = [G1[u][v]['color'] for u,v in edges]
    weights = [G1[u][v]['weight'] for u,v in edges]
    # nx.draw(G1, pos, edges=edges, node_color = color_map, edge_color=colors, width=wei
ghts, font_size=16, with_labels=False)
    pos = nx.spring_layout(G1, k=16, scale=1)
    nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=1
6, with_labels=False)
```

```
   for p in pos: # raise text positions
      pos[p][1] += 0.07
      nx.draw_networkx_labels(G1, pos)
      plt.show()



draw_graph (rules_mlxtend,6)
```



```
import numpy as np

def draw_graph(rules, rules_to_show):
  import networkx as nx
  G1 = nx.DiGraph()

  color_map=[]
  N = 50
  colors = np.random.rand(N)
  strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']


  for i in range (rules_to_show):
    G1.add_nodes_from(["R"+str(i)])

    for a in rules.iloc[i]['antecedents']:

        G1.add_nodes_from([a])

        G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)

    for c in rules.iloc[i]['consequents']:

            G1.add_nodes_from([c])

            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

  for node in G1:
       found_a_string = False
       for item in strs:
           if node==item:
                found_a_string = True
       if found_a_string:
            color_map.append('yellow')
       else:
            color_map.append('green')
```

Fig.6.5.visualization code



```
    edges= G1.edges()
    colors = [G1[u][v]['color'] for u,v in edges]
    weights = [G1[u][v]['weight'] for u,v in edges]

    pos = nx.spring_layout(G1, k=16, scale=1)
    nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)

    for p in pos:  # raise text positions
            pos[p][1] += 0.07
    nx.draw_networkx_labels(G1, pos)
    plt.show()


draw_graph (rules_mlxtend, 6)
```

Fig.6.6.visualization code

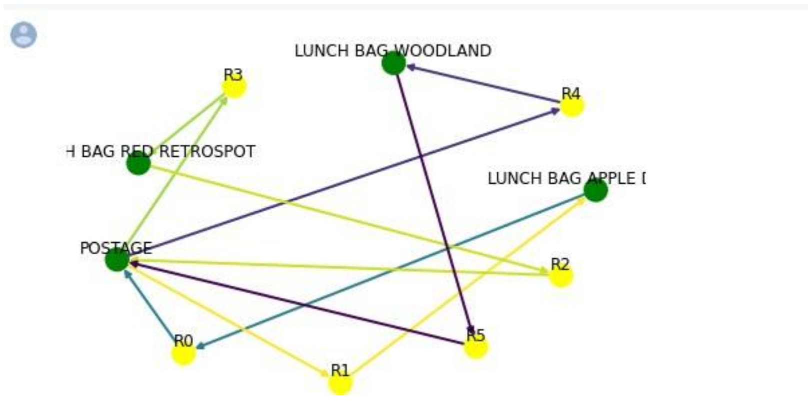


Fig.6.7.visualization

K MEANS CLUSTERING

K-means clustering can be used for market basket analysis, which is the process of analyzing customer transactions to identify patterns and associations among purchased items. In this context, K-means clustering can be used to group similar items together based on their co-occurrence in customer transactions.

To use K-means clustering for market basket analysis, we first need to create a transaction dataset where each row represents a customer transaction, and each column represents an item that was purchased. The values in the cells are binary, indicating whether the item was present or not in the transaction.

We then run K-means clustering on the transaction dataset, where the clusters represent groups of similar items that tend to co-occur in transactions. The number of clusters (k) can be chosen based on domain knowledge or by using an elbow plot to determine the optimal number of clusters.

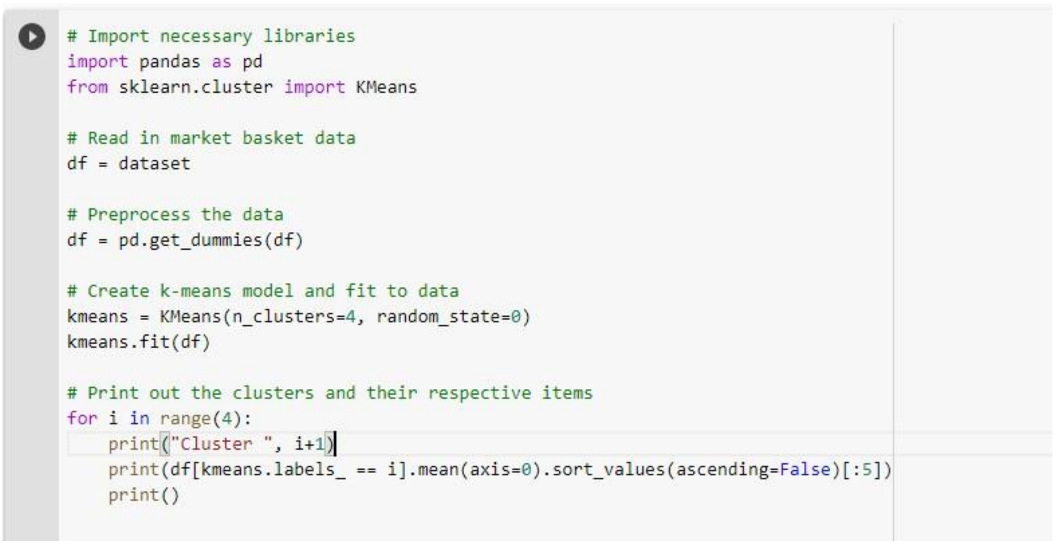
Once we have the clusters, we can analyze the contents of each cluster to identify the items that tend to co-occur. We can also calculate the support, confidence, and lift of item pairs within each cluster to identify strong associations between items.

For example, if we have a cluster containing bread, milk, and eggs, we can infer that these items are frequently purchased together. We can also calculate the support, confidence, and lift of item pairs within this cluster, such as bread and milk, to determine the strength of the association between these items.

Overall, K-means clustering can be a useful tool for market basket analysis, as it can help us identify patterns and associations among purchased items, which can inform product placement, promotional strategies, and inventory management.

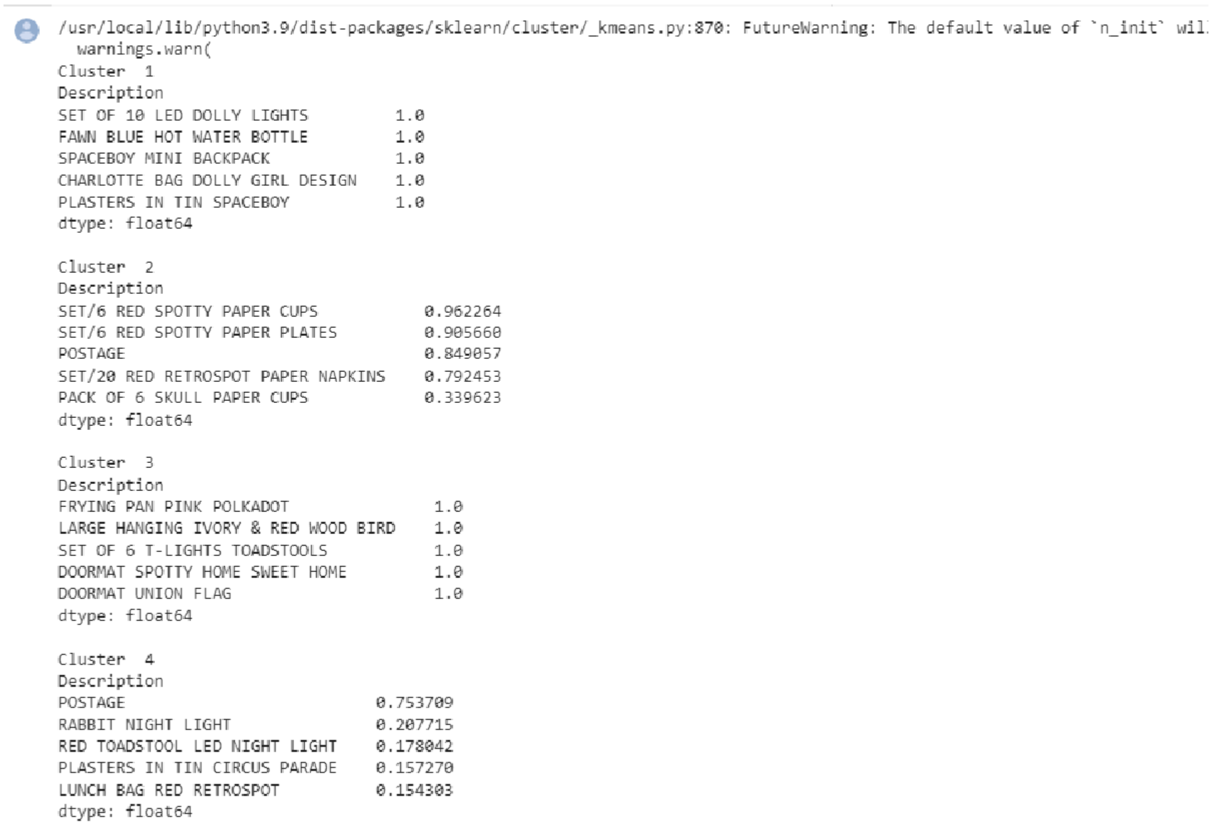
code :

```
# Import necessary libraries
import pandas as pd
from sklearn.cluster import KMeans

# Read in market basket data
df = dataset

# Preprocess the data
df = pd.get_dummies(df)

# Create k-means model and fit to data
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(df)

# Print out the clusters and their respective items
for i in range(4):
    print("Cluster ", i+1)
    print(df[kmeans.labels_ == i].mean(axis=0).sort_values(ascending=False)[:5])
    print()
```



```
# Import necessary libraries
import pandas as pd
from sklearn.cluster import KMeans

# Read in market basket data
df = dataset

# Preprocess the data
df = pd.get_dummies(df)

# Create k-means model and fit to data
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(df)

# Print out the clusters and their respective items
for i in range(4):
    print("Cluster ", i+1)
    print(df[kmeans.labels_ == i].mean(axis=0).sort_values(ascending=False)[:5])
    print()
```

Fig.7.Implementing K-means clustering



```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` wil
  warnings.warn(
Cluster  1
Description
SET OF 10 LED DOLLY LIGHTS         1.0
FAWN BLUE HOT WATER BOTTLE         1.0
SPACEBOY MINI BACKPACK             1.0
CHARLOTTE BAG DOLLY GIRL DESIGN    1.0
PLASTERS IN TIN SPACEBOY           1.0
dtype: float64

Cluster  2
Description
SET/6 RED SPOTTY PAPER CUPS           0.962264
SET/6 RED SPOTTY PAPER PLATES         0.905660
POSTAGE                               0.849057
SET/20 RED RETROSPOT PAPER NAPKINS    0.792453
PACK OF 6 SKULL PAPER CUPS            0.339623
dtype: float64

Cluster  3
Description
FRYING PAN PINK POLKADOT               1.0
LARGE HANGING IVORY & RED WOOD BIRD    1.0
SET OF 6 T-LIGHTS TOADSTOOLS           1.0
DOORMAT SPOTTY HOME SWEET HOME         1.0
DOORMAT UNION FLAG                     1.0
dtype: float64

Cluster  4
Description
POSTAGE                         0.753709
RABBIT NIGHT LIGHT              0.207715
RED TOADSTOOL LED NIGHT LIGHT   0.178042
PLASTERS IN TIN CIRCUS PARADE   0.157270
LUNCH BAG RED RETROSPOT         0.154303
dtype: float64
```

Fig.7.1.Implementing K-means clustering

ECLAT ALGORITHM :

ECLAT (Equivalence Class Clustering and Bottom-Up Lattice Traversal) is a popular algorithm used for market basket analysis.

The ECLAT algorithm is a depth-first search algorithm that is used to find frequent itemsets in transactional datasets. It works by generating a list of candidate itemsets of size k+1 from frequent itemsets of size k, and then pruning any candidate itemsets that contain infrequent subsets. The algorithm repeats this process until no new frequent itemsets can be found. The minimum support threshold is used to specify the minimum number of transactions that an itemset must appear in to be considered frequent. ECLAT is known for its efficiency and scalability, making it a popular choice for large datasets.

Define a minimum support threshold (e.g., 2% or 3%) that specifies the minimum number of transactions that an itemset must appear in to be considered frequent.

Create a list of frequent 1-itemsets by scanning the entire dataset and counting the frequency of each item. Discard any item that does not meet the minimum support threshold.

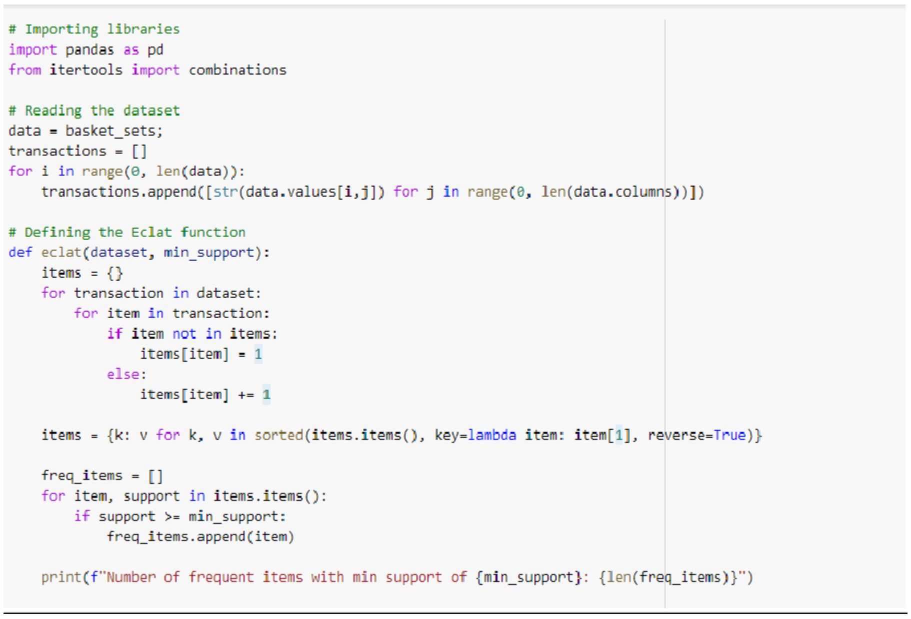
For each frequent itemset of size k, generate a list of candidate itemsets of size k+1 by joining each frequent itemset with the next item in the list, pruning any candidate itemsets that contain infrequent subsets.

Repeat steps 2-3 until no new frequent itemsets can be found.

CODE :

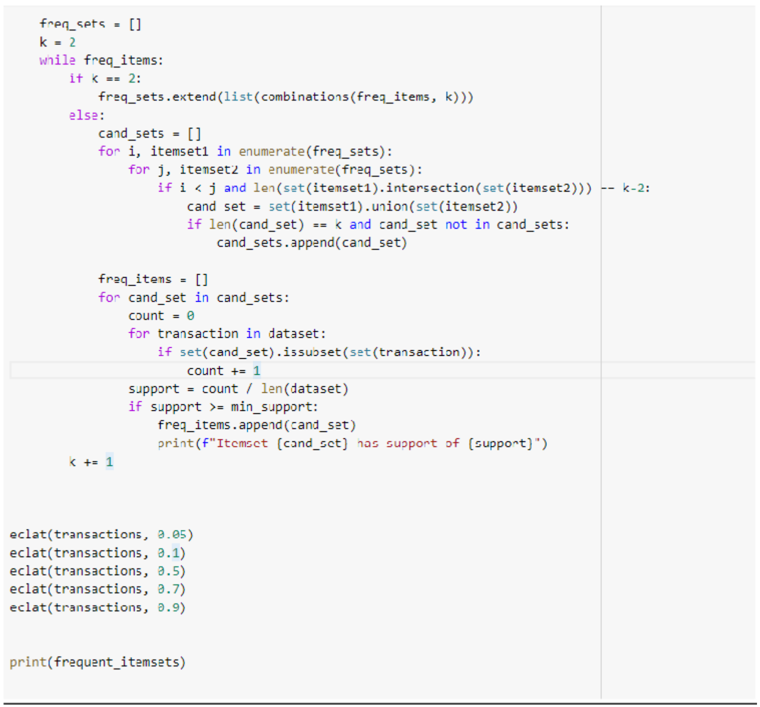
```
# Importing libraries
import pandas as pd
from itertools import combinations
# Reading the dataset
data = basket_sets;
transactions = []
for i in range(0, len(data)):
    transactions.append([str(data.values[i,j]) for j in range(0, len(data.columns))])
# Defining the Eclat function
def eclat(dataset, min_support):
    items = {}
    for transaction in dataset:
        for item in transaction:
            if item not in items:
                items[item] = 1
            else:
                items[item] += 1
    items = {k: v for k, v in sorted(items.items(), key=lambda item: item[1], reverse=True)}
    freq_items = []
    for item, support in items.items():
        if support >= min_support:
            freq_items.append(item)

    print(f"Number of frequent items with min support of {min_support}: {len(freq_items)}")
    freq_sets = []
    k = 2
```

```
 while freq_items:
        if k == 2:
            freq_sets.extend(list(combinations(freq_items, k)))
        else:
            cand_sets = []
            for i, itemset1 in enumerate(freq_sets):
                for j, itemset2 in enumerate(freq_sets):
                    if i < j and len(set(itemset1).intersection(set(itemset2))) == k-2:
                        cand_set = set(itemset1).union(set(itemset2))
                        if len(cand_set) == k and cand_set not in cand_sets:
                            cand_sets.append(cand_set)
            freq_items = []
            for cand_set in cand_sets:
                count = 0
                for transaction in dataset:
                    if set(cand_set).issubset(set(transaction)):
                        count += 1
                support = count / len(dataset)
                if support >= min_support:
                    freq_items.append(cand_set)
                    print(f"Itemset {cand_set} has support of {support}")
        k += 1
eclat(transactions, 0.05)
eclat(transactions, 0.1)
eclat(transactions, 0.5)
eclat(transactions, 0.7)
eclat(transactions, 0.9)
print(frequent_itemsets)
```



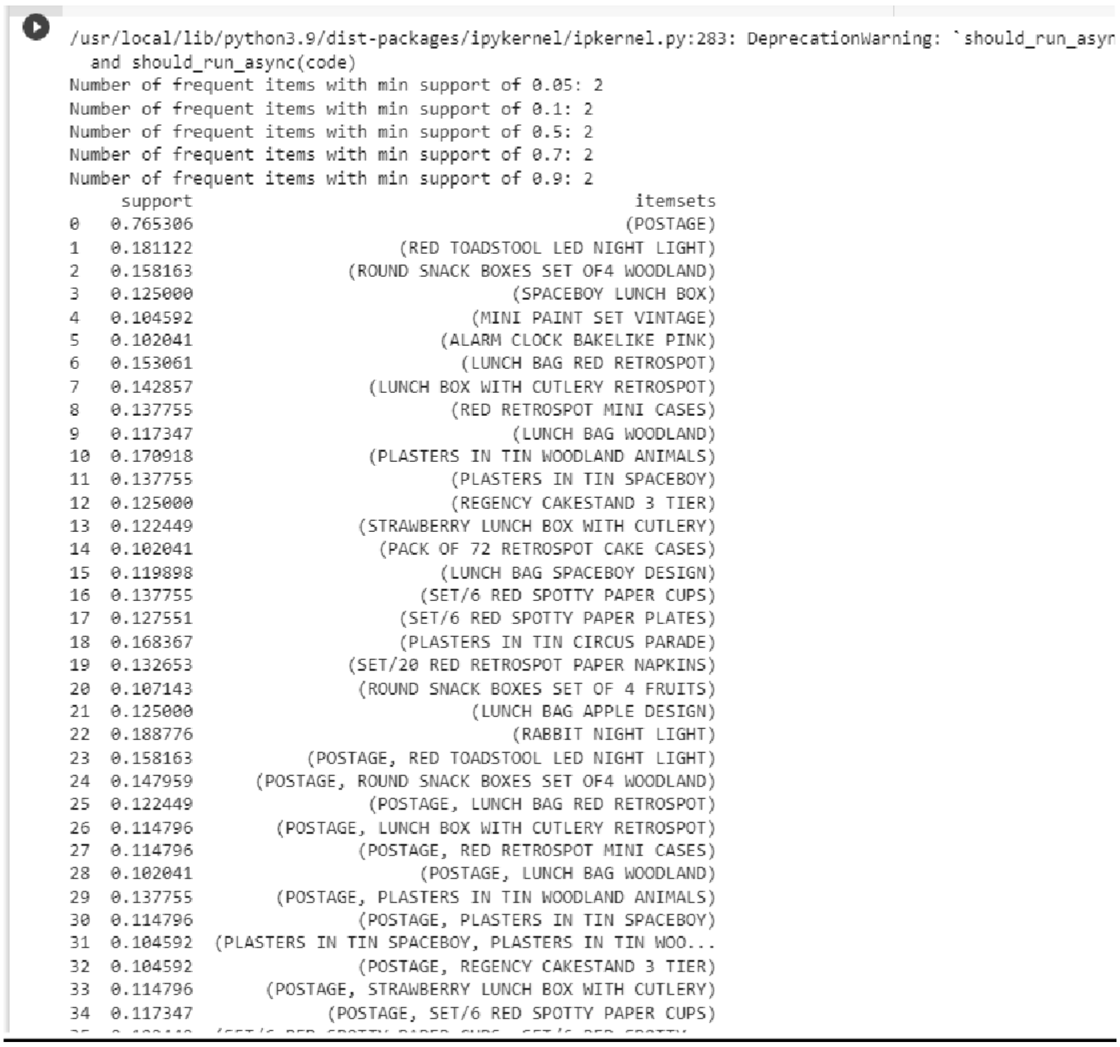
```python
# Importing libraries
import pandas as pd
from itertools import combinations

# Reading the dataset
data = basket_sets;
transactions = []
for i in range(0, len(data)):
    transactions.append([str(data.values[i,j]) for j in range(0, len(data.columns))])

# Defining the Eclat function
def eclat(dataset, min_support):
    items = {}
    for transaction in dataset:
        for item in transaction:
            if item not in items:
                items[item] = 1
            else:
                items[item] += 1

    items = {k: v for k, v in sorted(items.items(), key=lambda item: item[1], reverse=True)}

    freq_items = []
    for item, support in items.items():
        if support >= min_support:
            freq_items.append(item)

    print(f"Number of frequent items with min support of {min_support}: {len(freq_items)}")
```

Fig.8.Implementing ECLAT



```python
    freq_sets = []
    k = 2
    while freq_items:
        if k == 2:
            freq_sets.extend(list(combinations(freq_items, k)))
        else:
            cand_sets = []
            for i, itemset1 in enumerate(freq_sets):
                for j, itemset2 in enumerate(freq_sets):
                    if i < j and len(set(itemset1).intersection(set(itemset2))) -- k-2:
                        cand set = set(itemset1).union(set(itemset2))
                        if len(cand_set) == k and cand_set not in cand_sets:
                            cand_sets.append(cand_set)

            freq_items = []
            for cand_set in cand_sets:
                count = 0
                for transaction in dataset:
                    if set(cand_set).issubset(set(transaction)):
                        count += 1
                support = count / len(dataset)
                if support >= min_support:
                    freq_items.append(cand_set)
                    print(f"Itemset {cand_set} has support of {support}")
        k += 1


eclat(transactions, 0.05)
eclat(transactions, 0.1)
eclat(transactions, 0.5)
eclat(transactions, 0.7)
eclat(transactions, 0.9)


print(frequent_itemsets)
```

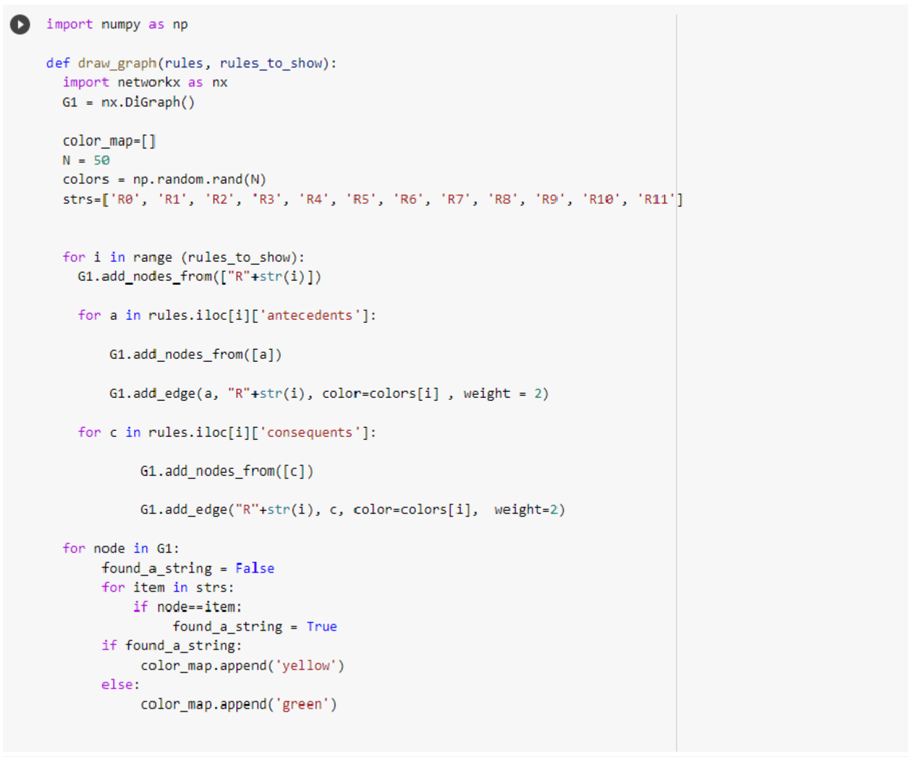
Fig.8.1.Implementing ECLAT



```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_asyn
  and should_run_async(code)
Number of frequent items with min support of 0.05: 2
Number of frequent items with min support of 0.1: 2
Number of frequent items with min support of 0.5: 2
Number of frequent items with min support of 0.7: 2
Number of frequent items with min support of 0.9: 2
     support                                        itemsets
0   0.765306                                       (POSTAGE)
1   0.181122                   (RED TOADSTOOL LED NIGHT LIGHT)
2   0.158163              (ROUND SNACK BOXES SET OF4 WOODLAND)
3   0.125000                             (SPACEBOY LUNCH BOX)
4   0.104592                            (MINI PAINT SET VINTAGE)
5   0.102041                        (ALARM CLOCK BAKELIKE PINK)
6   0.153061                          (LUNCH BAG RED RETROSPOT)
7   0.142857                 (LUNCH BOX WITH CUTLERY RETROSPOT)
8   0.137755                         (RED RETROSPOT MINI CASES)
9   0.117347                               (LUNCH BAG WOODLAND)
10  0.170918                 (PLASTERS IN TIN WOODLAND ANIMALS)
11  0.137755                         (PLASTERS IN TIN SPACEBOY)
12  0.125000                         (REGENCY CAKESTAND 3 TIER)
13  0.122449                (STRAWBERRY LUNCH BOX WITH CUTLERY)
14  0.102041                   (PACK OF 72 RETROSPOT CAKE CASES)
15  0.119898                        (LUNCH BAG SPACEBOY DESIGN)
16  0.137755                      (SET/6 RED SPOTTY PAPER CUPS)
17  0.127551                    (SET/6 RED SPOTTY PAPER PLATES)
18  0.168367                    (PLASTERS IN TIN CIRCUS PARADE)
19  0.132653              (SET/20 RED RETROSPOT PAPER NAPKINS)
20  0.107143                (ROUND SNACK BOXES SET OF 4 FRUITS)
21  0.125000                           (LUNCH BAG APPLE DESIGN)
22  0.188776                               (RABBIT NIGHT LIGHT)
23  0.158163          (POSTAGE, RED TOADSTOOL LED NIGHT LIGHT)
24  0.147959     (POSTAGE, ROUND SNACK BOXES SET OF4 WOODLAND)
25  0.122449                 (POSTAGE, LUNCH BAG RED RETROSPOT)
26  0.114796        (POSTAGE, LUNCH BOX WITH CUTLERY RETROSPOT)
27  0.114796                (POSTAGE, RED RETROSPOT MINI CASES)
28  0.102041                      (POSTAGE, LUNCH BAG WOODLAND)
29  0.137755        (POSTAGE, PLASTERS IN TIN WOODLAND ANIMALS)
30  0.114796                (POSTAGE, PLASTERS IN TIN SPACEBOY)
31  0.104592  (PLASTERS IN TIN SPACEBOY, PLASTERS IN TIN WOO...
32  0.104592                (POSTAGE, REGENCY CAKESTAND 3 TIER)
33  0.114796       (POSTAGE, STRAWBERRY LUNCH BOX WITH CUTLERY)
34  0.117347             (POSTAGE, SET/6 RED SPOTTY PAPER CUPS)
```

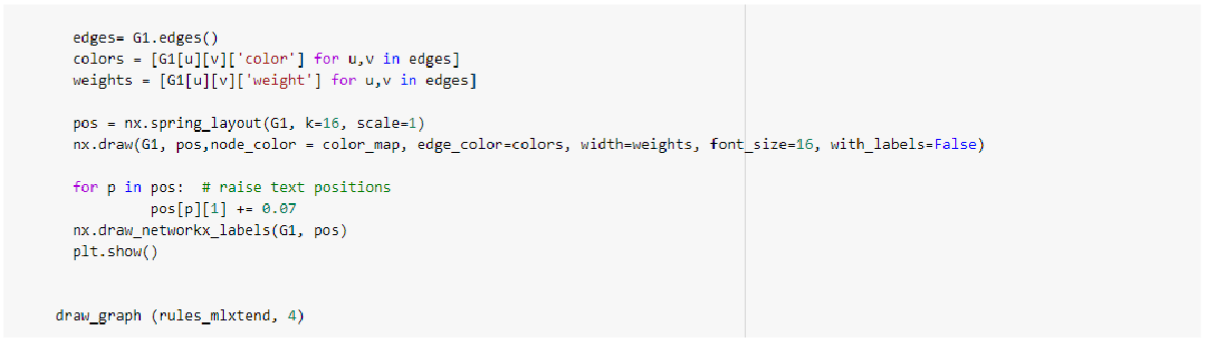
Fig.8.2.Implementing ECLAT

Visualization



```
import numpy as np

def draw_graph(rules, rules_to_show):
  import networkx as nx
  G1 = nx.DiGraph()

  color_map=[]
  N = 50
  colors = np.random.rand(N)
  strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']


  for i in range (rules_to_show):
    G1.add_nodes_from(["R"+str(i)])

    for a in rules.iloc[i]['antecedents']:

        G1.add_nodes_from([a])

        G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)

    for c in rules.iloc[i]['consequents']:

            G1.add_nodes_from([c])

            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

  for node in G1:
       found_a_string = False
       for item in strs:
           if node==item:
                found_a_string = True
       if found_a_string:
            color_map.append('yellow')
       else:
            color_map.append('green')
```

Fig.8.3.Implementing ECLAT



```
edges= G1.edges()
colors = [G1[u][v]['color'] for u,v in edges]
weights = [G1[u][v]['weight'] for u,v in edges]

pos = nx.spring_layout(G1, k=16, scale=1)
nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)

for p in pos:  # raise text positions
          pos[p][1] += 0.07
nx.draw_networkx_labels(G1, pos)
plt.show()


draw_graph (rules_mlxtend, 4)
```

Fig.8.4.Implementing ECLAT

```
and should_run_async(code)
```

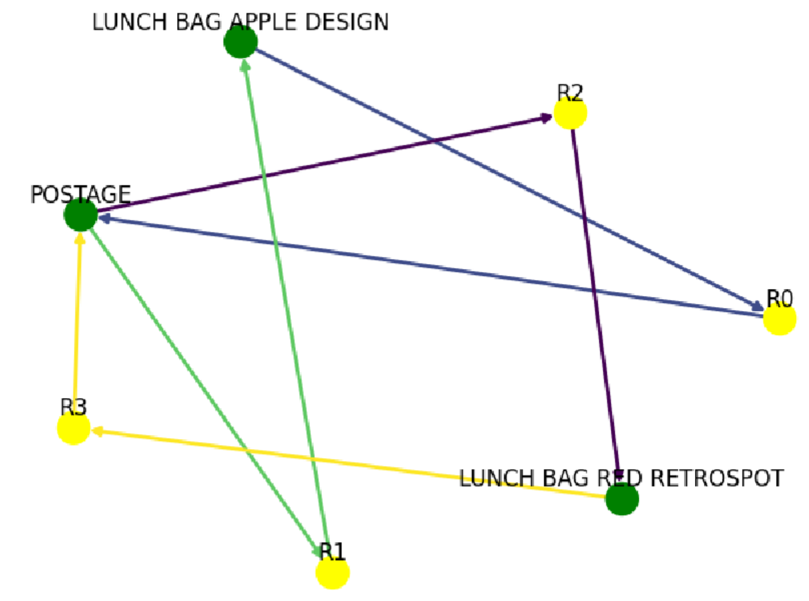


Fig.8.5.Implementing ECLAT

SETM ALGORITHM

The SETM (Subgroup Extraction by Tree Mining) algorithm is a popular algorithm used for market basket analysis. It is a pattern-based approach that works by constructing a decision tree to identify subgroups of customers with similar purchasing patterns. SETM is known for its scalability and ability to handle large datasets.

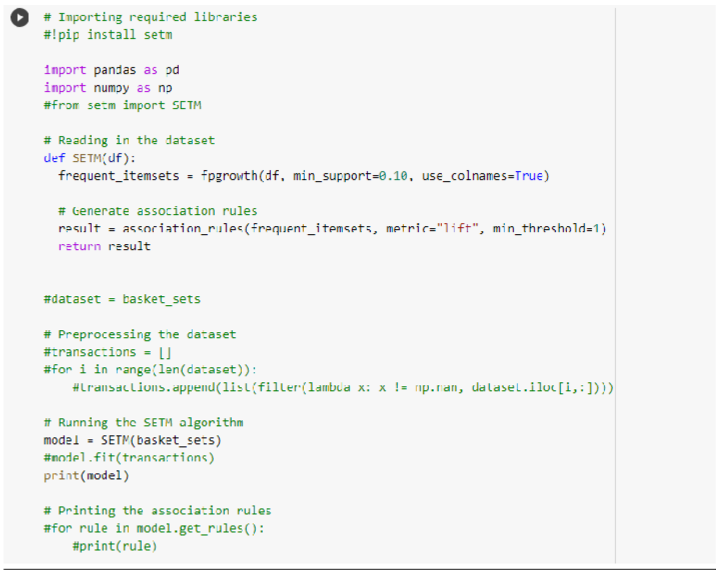
It works by constructing a decision tree and extracting frequent itemsets from the tree nodes. The algorithm is known for its scalability and ability to handle large datasets.

One of the main advantages of the SETM algorithm is that it is able to handle large datasets and is highly scalable. This is because it uses a tree-based approach, which allows it to efficiently partition the data and extract relevant patterns.

The SETM algorithm has been used in a variety of applications beyond market basket analysis, including social network analysis, customer profiling, and web usage mining. It is also used in combination with other techniques, such as clustering and association rule mining, to further improve the accuracy and relevance of the extracted patterns.

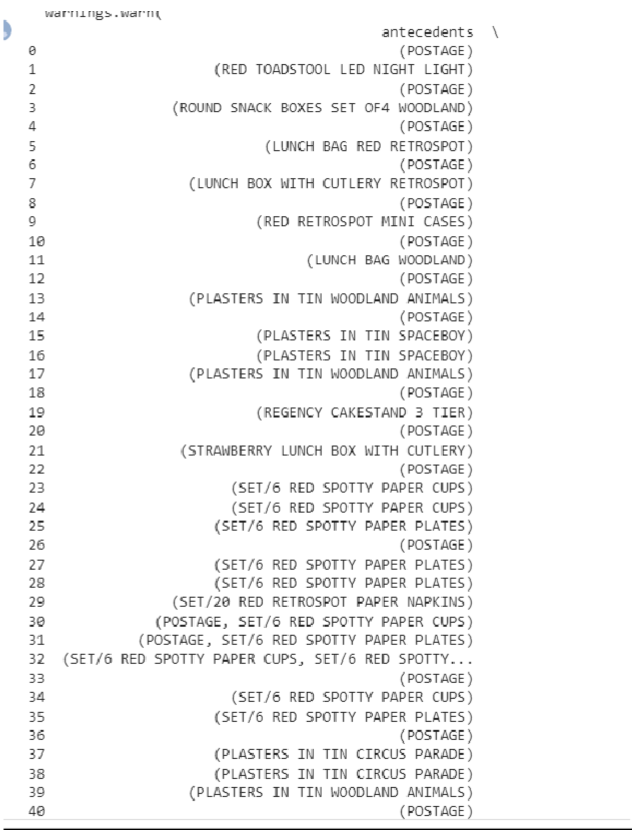
Overall, the SETM algorithm is a powerful and flexible technique for identifying subgroups of customers with similar purchasing patterns, and extracting relevant insights from large and complex datasets.

CODE:



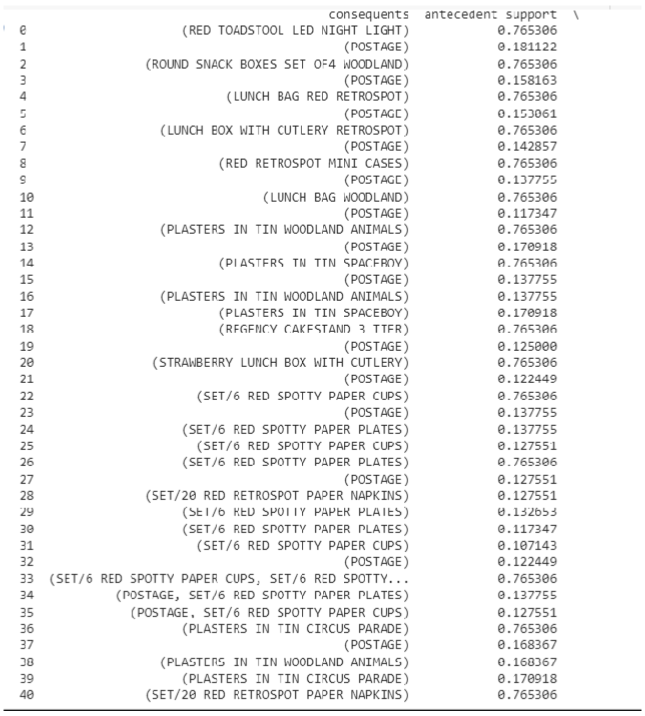
```python
# Importing required libraries
#!pip install setm

import pandas as pd
import numpy as np
#from setm import SETM

# Reading in the dataset
def SETM(df):
  frequent_itemsets = fpgrowth(df, min_support=0.10, use_colnames=True)

  # Generate association rules
  result = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
  return result


#dataset = basket_sets

# Preprocessing the dataset
#transactions = []
#for i in range(len(dataset)):
    #transactions.append(list(filter(lambda x: x != np.nan, dataset.iloc[i,:])))

# Running the SETM algorithm
model = SETM(basket_sets)
#model.fit(transactions)
print(model)

# Printing the association rules
#for rule in model.get_rules():
    #print(rule)
```

Fig.9.Implementing SETM



```
  warnings.warn(
                                          antecedents  \
0                                           (POSTAGE)
1                           (RED TOADSTOOL LED NIGHT LIGHT)
2                                           (POSTAGE)
3                    (ROUND SNACK BOXES SET OF4 WOODLAND)
4                                           (POSTAGE)
5                                (LUNCH BAG RED RETROSPOT)
6                                           (POSTAGE)
7                      (LUNCH BOX WITH CUTLERY RETROSPOT)
8                                           (POSTAGE)
9                               (RED RETROSPOT MINI CASES)
10                                          (POSTAGE)
11                                   (LUNCH BAG WOODLAND)
12                                          (POSTAGE)
13                     (PLASTERS IN TIN WOODLAND ANIMALS)
14                                          (POSTAGE)
15                              (PLASTERS IN TIN SPACEBOY)
16                              (PLASTERS IN TIN SPACEBOY)
17                     (PLASTERS IN TIN WOODLAND ANIMALS)
18                                          (POSTAGE)
19                              (REGENCY CAKESTAND 3 TIER)
20                                          (POSTAGE)
21                     (STRAWBERRY LUNCH BOX WITH CUTLERY)
22                                          (POSTAGE)
23                           (SET/6 RED SPOTTY PAPER CUPS)
24                           (SET/6 RED SPOTTY PAPER CUPS)
25                         (SET/6 RED SPOTTY PAPER PLATES)
26                                          (POSTAGE)
27                         (SET/6 RED SPOTTY PAPER PLATES)
28                         (SET/6 RED SPOTTY PAPER PLATES)
29                    (SET/20 RED RETROSPOT PAPER NAPKINS)
30                  (POSTAGE, SET/6 RED SPOTTY PAPER CUPS)
31                (POSTAGE, SET/6 RED SPOTTY PAPER PLATES)
32   (SET/6 RED SPOTTY PAPER CUPS, SET/6 RED SPOTTY...
33                                          (POSTAGE)
34                           (SET/6 RED SPOTTY PAPER CUPS)
35                         (SET/6 RED SPOTTY PAPER PLATES)
36                                          (POSTAGE)
37                         (PLASTERS IN TIN CIRCUS PARADE)
38                         (PLASTERS IN TIN CIRCUS PARADE)
39                     (PLASTERS IN TIN WOODLAND ANIMALS)
40                                          (POSTAGE)
```

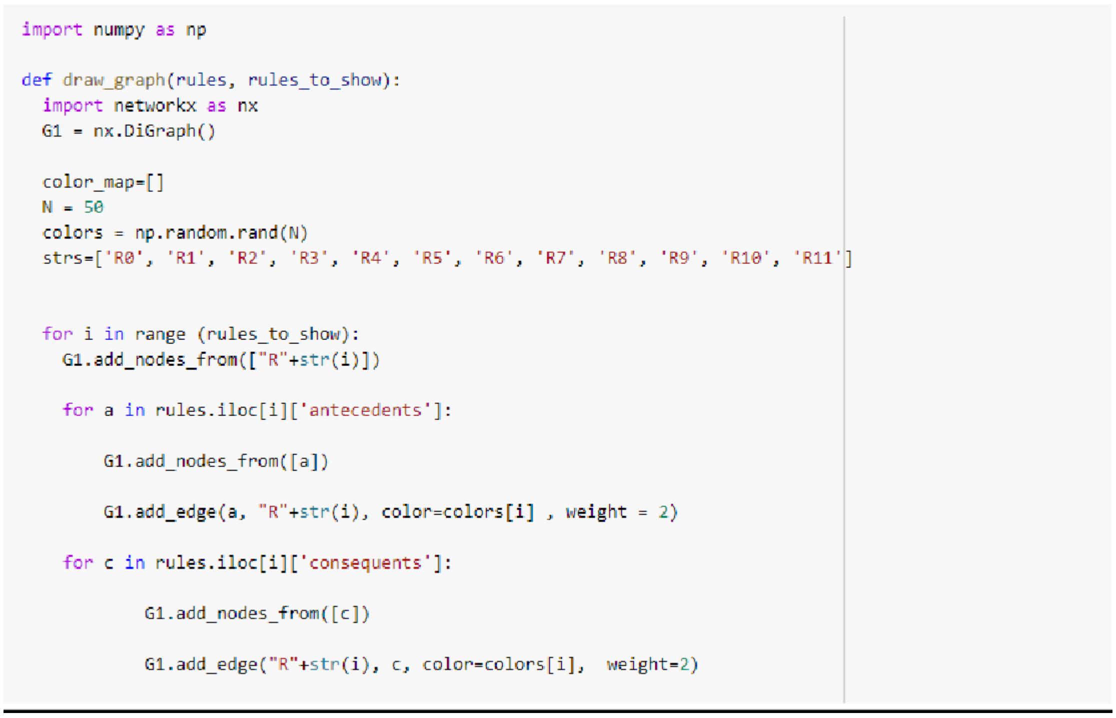
Fig.9.1.Implementing SETM



```
                                          consequents  antecedent support  \
0             (RED TOADSTOOL LED NIGHT LIGHT)            0.765306
1                                   (POSTAGE)            0.181122
2        (ROUND SNACK BOXES SET OF4 WOODLAND)            0.765306
3                                   (POSTAGE)            0.158163
4                   (LUNCH BAG RED RETROSPOT)            0.765306
5                                   (POSTAGE)            0.153061
6           (LUNCH BOX WITH CUTLERY RETROSPOT)            0.765306
7                                   (POSTAGE)            0.142857
8                  (RED RETROSPOT MINI CASES)            0.765306
9                                   (POSTAGE)            0.137755
10                       (LUNCH BAG WOODLAND)            0.765306
11                                  (POSTAGE)            0.117347
12         (PLASTERS IN TIN WOODLAND ANIMALS)            0.765306
13                                  (POSTAGE)            0.170918
14                 (PLASTERS IN TIN SPACEBOY)            0.765306
15                                  (POSTAGE)            0.137755
16         (PLASTERS IN TIN WOODLAND ANIMALS)            0.137755
17                 (PLASTERS IN TIN SPACEBOY)            0.170918
18                 (REGENCY CAKESTAND 3 TIER)            0.765306
19                                  (POSTAGE)            0.125000
20        (STRAWBERRY LUNCH BOX WITH CUTLERY)            0.765306
21                                  (POSTAGE)            0.122449
22              (SET/6 RED SPOTTY PAPER CUPS)            0.765306
23                                  (POSTAGE)            0.137755
24            (SET/6 RED SPOTTY PAPER PLATES)            0.137755
25              (SET/6 RED SPOTTY PAPER CUPS)            0.127551
26            (SET/6 RED SPOTTY PAPER PLATES)            0.765306
27                                  (POSTAGE)            0.127551
28       (SET/20 RED RETROSPOT PAPER NAPKINS)            0.127551
29            (SET/6 RED SPOTTY PAPER PLATES)            0.132653
30            (SET/6 RED SPOTTY PAPER PLATES)            0.117347
31              (SET/6 RED SPOTTY PAPER CUPS)            0.107143
32                                  (POSTAGE)            0.122449
33  (SET/6 RED SPOTTY PAPER CUPS, SET/6 RED SPOTTY...            0.765306
34   (POSTAGE, SET/6 RED SPOTTY PAPER PLATES)            0.137755
35     (POSTAGE, SET/6 RED SPOTTY PAPER CUPS)            0.127551
36            (PLASTERS IN TIN CIRCUS PARADE)            0.765306
37                                  (POSTAGE)            0.168367
38         (PLASTERS IN TIN WOODLAND ANIMALS)            0.168367
39            (PLASTERS IN TIN CIRCUS PARADE)            0.170918
40       (SET/20 RED RETROSPOT PAPER NAPKINS)            0.765306
```

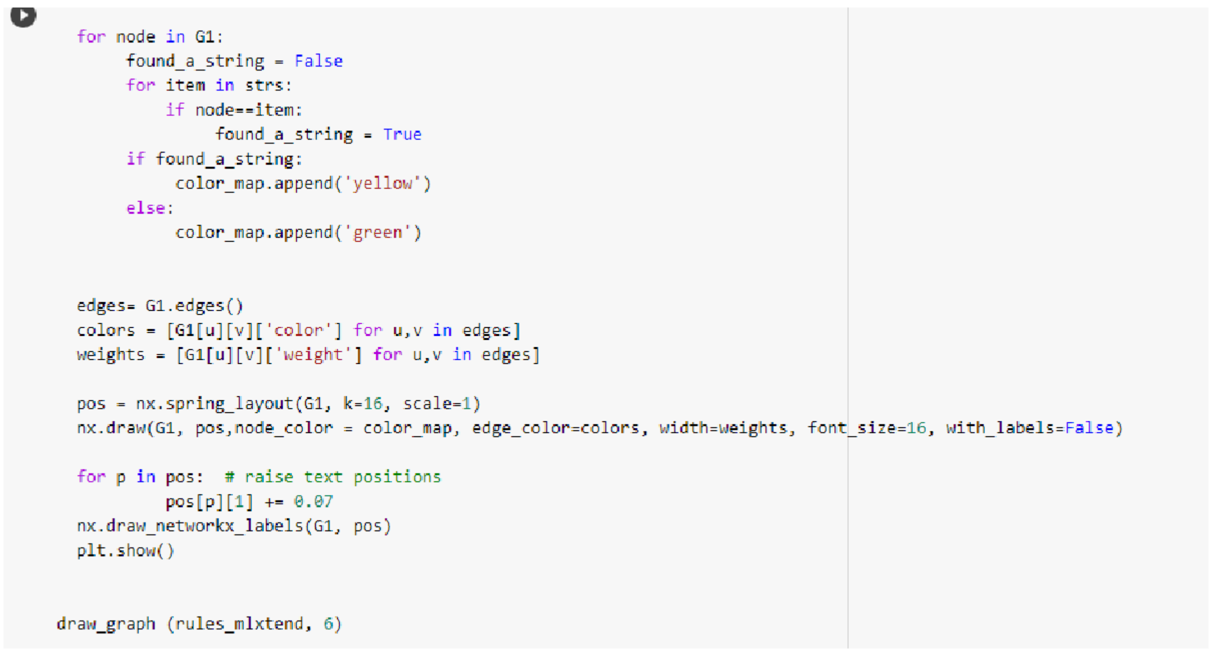
Fig.9.2.Implementing SETM

Visualization



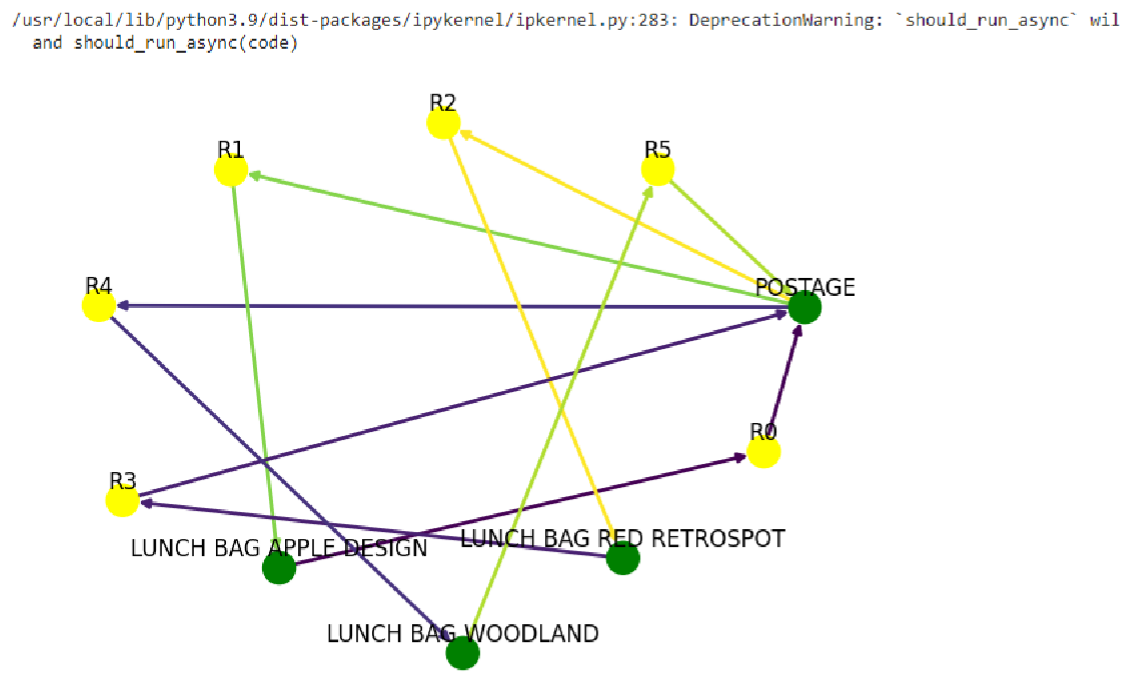
```python
import numpy as np

def draw_graph(rules, rules_to_show):
  import networkx as nx
  G1 = nx.DiGraph()

  color_map=[]
  N = 50
  colors = np.random.rand(N)
  strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']


  for i in range (rules_to_show):
    G1.add_nodes_from(["R"+str(i)])

    for a in rules.iloc[i]['antecedents']:

        G1.add_nodes_from([a])

        G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)

    for c in rules.iloc[i]['consequents']:

            G1.add_nodes_from([c])

            G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)
```

Fig.9.3.Implementing SETM



```
  for node in G1:
       found_a_string = False
       for item in strs:
           if node==item:
                found_a_string = True
       if found_a_string:
            color_map.append('yellow')
       else:
            color_map.append('green')


  edges= G1.edges()
  colors = [G1[u][v]['color'] for u,v in edges]
  weights = [G1[u][v]['weight'] for u,v in edges]

  pos = nx.spring_layout(G1, k=16, scale=1)
  nx.draw(G1, pos,node_color = color_map, edge_color=colors, width=weights, font_size=16, with_labels=False)

  for p in pos:  # raise text positions
           pos[p][1] += 0.07
  nx.draw_networkx_labels(G1, pos)
  plt.show()


draw_graph (rules_mlxtend, 6)
```

Fig.9.4.Implementing SETM



```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` wil
  and should_run_async(code)
```

Fig.9.5.Implementing SETM

2.2 COMPARISION BETWEEN ALGORITHMS :

FP, Apriori, SETM, k-means, ECLAT are all data mining techniques used in market basket analysis, but they have different approaches and applications. Here's a brief comparison:

Frequent pattern (FP) mining:

FP mining is a technique for discovering frequent itemsets and association rules in transactional datasets. FP mining algorithms use a more memory-efficient data structure (the FP-tree) to mine frequent itemsets and can handle sparse datasets, which can be challenging for other methods. FP mining is best suited for discovering patterns that occur frequently in the data and can be used for recommending related products in online shopping carts.

Apriori algorithm:

Apriori is a classical algorithm for frequent itemset mining and association rule generation. It works by generating a list of candidate itemsets and removing those that do not meet the minimum support threshold. Apriori is easy to implement and understand but can be computationally expensive and memory-intensive for large datasets. It is suitable for finding frequent itemsets and association rules in transactional data, making it useful for market basket analysis.

SETM:

SetM is a data mining algorithm used to mine high-utility itemsets from transactional databases. It focuses on finding itemsets with high utility rather than just frequent ones. The utility of an itemset is determined by the profit it generates, taking into account the quantity and price of the items in the set. SetM is well suited for retail businesses, where identifying high-utility itemsets can help optimize pricing and inventory.

k-means clustering:

k-means clustering is a technique for grouping similar items or customers into clusters based on their purchasing behavior. It is an unsupervised learning algorithm that partitions the data into k clusters based on their similarity. k-means clustering can be used to segment customers into groups with similar purchasing behavior, allowing businesses to tailor their marketing strategies to each group.

ECLAT :

The Eclat algorithm is a depth-first search algorithm that uses a vertical data format to find frequent itemsets.

It is fast and memory-efficient, making it a good choice for large datasets with a high number of items.

Eclat is able to find frequent itemsets without generating candidate itemsets, which can be computationally expensive.

However, it may not be as effective as Apriori or SETM in identifying subgroups of customers with similar purchasing patterns.

In summary, FP mining and Apriori are useful for discovering frequent itemsets and association rules, while SetM is useful for identifying high-utility itemsets . k-means clustering, on the other hand, is useful for segmenting customers into groups with similar purchasing behavior. The choice of algorithm depends on the specific requirements of the analysis and the characteristics of the data being analyzed.

comparison between appriori and FP algorithm,

Working Principle:

The Apriori algorithm works by generating candidate itemsets of increasing size and pruning the ones that do not satisfy the minimum support threshold. This process is repeated until no more frequent itemsets can be generated. On the other hand, the FP-Growth algorithm works by constructing a tree-like data structure called the FP-tree, which is used to generate frequent itemsets without generating candidate itemsets explicitly.

Efficiency:

The Apriori algorithm is known to be less efficient as it requires multiple scans of the database to generate candidate itemsets. This can be time-consuming for large datasets. In contrast, the FP-Growth algorithm is more efficient as it requires only two passes over the database to construct the FP-tree and generate frequent itemsets.

Memory Usage:

The Apriori algorithm uses a lot of memory as it generates a large number of candidate itemsets. This can be a problem for datasets with a large number of items. The FP-Growth algorithm, on the other hand, uses less memory as it constructs the FP-tree which is a compact data structure.

Scalability:

The Apriori algorithm does not scale well with large datasets as it requires multiple scans of the database. As a result, it can become computationally expensive for datasets with a large number of transactions. The FP-Growth algorithm, on the other hand, scales better with large datasets as it requires only two passes over the database.

In conclusion, the Apriori algorithm is easier to understand and implement, but it is less efficient and more memory-intensive than the FP-Growth algorithm. The FP-Growth algorithm, on the other hand, is more efficient and scalable, making it suitable for large datasets.

CHAPTER 3

CONCLUSION

3.1 CONCLUSION

We finally choose the Apriori algorithm since it is simpler and more efficient way to find such patterns inside goods quickly because most data transactions in a physical retail market or an online e- commerce site are large. In order to increase profitability through cross-selling, recommendations, promotion, or even the placement of times on a menu or in a store, we hope to be able to develop a pattern recommendation system that can be used by various retail businesses. This system will highlight the purchasing pattern of customers.

In this project, we tried to formulate a suggestion that might be used as a method for producing the common itemset and aiding us in selecting the most important itemset. The Apriori Algorithm will therefore help us identify the most inconvenient and necessary itemset, which will lead to enhanced market strategies, customer convenience, and increased product sales, all of which will result in a successful business.

The aforementioned study project offers outcomes for promotional products based on uncommon items that customers are less inclined to buy and maximizes profit by selling the uncommon items. The proposed methodology for this study involves linear programming to start the profit maximizing process, operational research theories that solely handle linear equations, and producing promotional products utilizing the Apriori algorithm to extract common things.

CHAPTER 4

REFERENCES

4.1 REFERENCES :

1. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-market-basket- analysis/
2. Data Mining: Market Basket Analysis with Apriori Algorithm | by Yenwee Lim | Towards DataScience
3. A. Hagberg, D. Schult, P. Swart. NetworkX Reference Release 2.7.1. (2022). Retrieved fromhttps://networkx.org/
4. D.H. Goh, R.P. Ang. An introduction to association rule mining: An application in counselingand helpseeking behavior of adolescents. (2007). Behavior Research Methods 39, 259–266
5. H. Dedhia. Groceries Dataset licensed under GPL 2. (2020). Retrieved from: https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset
6. Online Retail For Market Basket Analysis Dataset: https://www.kaggle.com/datasets/yekahaaagayeham/online-retail-for-market-basket-analysis
7. **https://towardsdatascience.com/the-fp-growth-algorithm-1ffa20e839b8**
8. **https://ieeexplore.ieee.org/abstract/document/9038197**
9. **https://www.dotactiv.com/blog/why-k-means-clustering-is-good-for-business#:~:text=You%20can%20use%20k-means%20clustering%20to%20analyse%20different,everyday-low-price%20strategy%2C%20or%20products%20on%20sale%20before%20ex - pir y.**

10. **http://9.https//www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-%20market-basket%20analysis/#:~:text=The%20SETM%20algorithm%20creates%20collective%20pa%20sses%20over%20the,by%20enlarging%20large%20itemsets%20of%20the%20p%20revious%20pass.**