LAB EXPERIMENT NO. 06

Aim: Implementation of Association rule mining Using
1. Apriori Algorithm
2. FPTree

Perform the experiment in Python.

Read any market basket analysis dataset from UCI dataset repository

Part A:

Read min_support and confidence from the user

Program Apriori algorithm using inbuilt functions.

Print the association rules

THEORY:

Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers but at a Big Bazar.
Apriori algorithm helps the customers to buy their products with ease and increases the sales performance of the particular store.

Components of Apriori algorithm

The given three components comprise the apriori algorithm.

- Support
- Confidence
- Lift

Suppose you have 4000 customers transactions in a Big Bazar. You have to calculate the Support, Confidence, and Lift for two products, and you may say Biscuits and Chocolate. This is because customers frequently buy these two items together.

Out of 4000 transactions, 400 contain Biscuits, whereas 600 contain Chocolate, and these 600 transactions include a 200 that includes Biscuits and chocolates. Using this data, we will find out the support, confidence, and lift.

Support

Support refers to the default popularity of any product. You find the support as a quotient of the division of the number of transactions comprising that product by the total number of transactions. Hence, we get

Support (Biscuits) = (Transactions relating biscuits) / (Total transactions)

= 400/4000 = 10 percent.

Confidence

Confidence refers to the possibility that the customers bought both biscuits and chocolates together. So, you need to divide the number of transactions that comprise both biscuits and chocolates by the total number of transactions to get the confidence.

Hence,

Confidence = (Transactions relating both biscuits and Chocolate) / (Total transactions involving Biscuits)

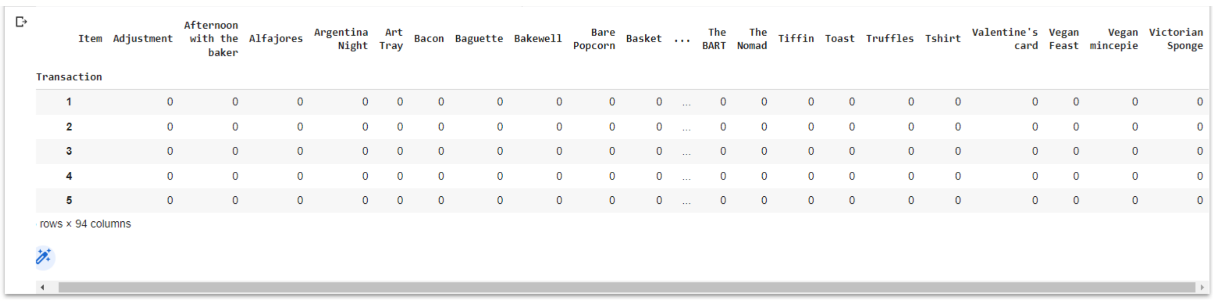
= 200/400

= 50 percent.

It means that 50 percent of customers who bought biscuits bought chocolates also.

CODE:

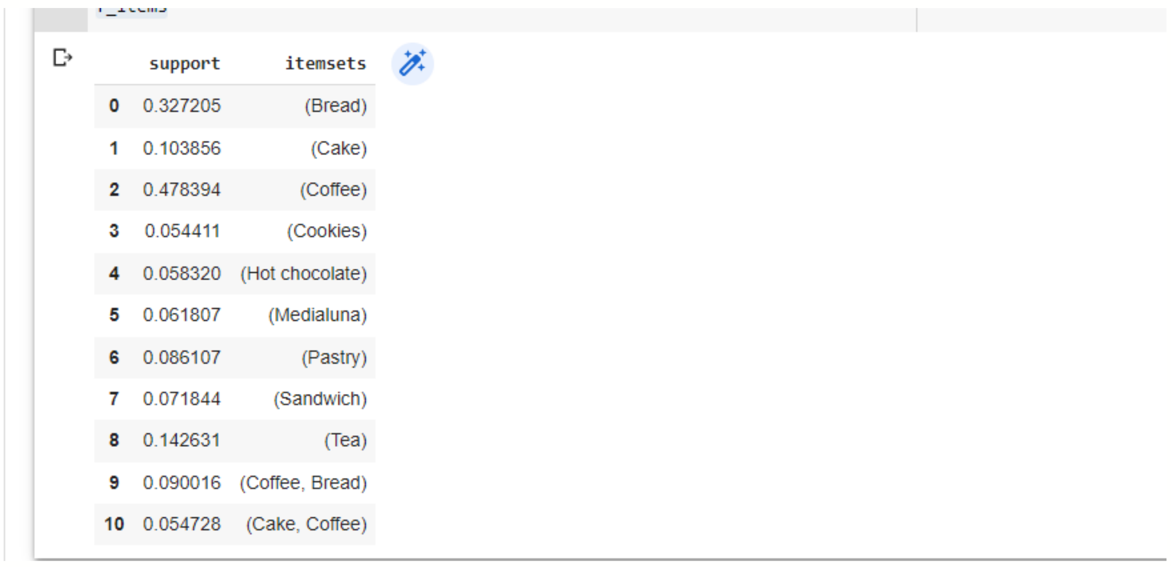
```
import pandas as pd
import numpy as np
import networkx as nx
import plotly.express as px
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
from PyARMViz import PyARMViz

warnings.filterwarnings('ignore')

plt.style.use('seaborn')

data = pd.read_csv('bread basket.csv')
from google.colab import drive
drive.mount('/content/drive')

from mlxtend.frequent_patterns import association_rules, aprio
ri

def encoder(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

apriori_data = data.groupby(['Transaction','Item'])['Item'].co
unt().reset_index(name ='Count')
apriori_basket = apriori_data.pivot_table(index = 'Transaction
', columns = 'Item', values = 'Count', aggfunc = 'sum').fillna
(0)
apriori_basket_set = apriori_basket.applymap(encoder)
apriori_basket_set.head()
```



Item	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Bacon	Baguette	Bakewell	Bare Popcorn	Basket	...	The BART	The Nomad	Tiffin	Toast	Truffles	Tshirt	Valentine's card	Vegan Feast	Vegan mincepie	Victorian Sponge
Transaction																					
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

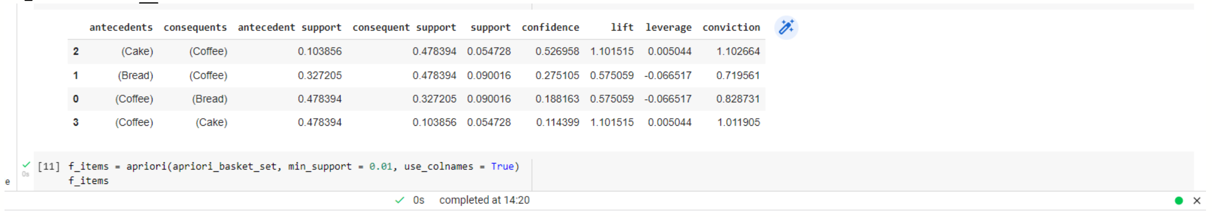
rows × 94 columns

```
f_items = apriori(apriori_basket_set, min_support = 0.05, use_
colnames = True)
f_items
```



	support	itemsets
0	0.327205	(Bread)
1	0.103856	(Cake)
2	0.478394	(Coffee)
3	0.054411	(Cookies)
4	0.058320	(Hot chocolate)
5	0.061807	(Medialuna)
6	0.086107	(Pastry)
7	0.071844	(Sandwich)
8	0.142631	(Tea)
9	0.090016	(Coffee, Bread)
10	0.054728	(Cake, Coffee)

```
apriori_rules = association_rules(f_items, metric = 'lift', mi
n_threshold = 0.05)
apriori_rules.sort_values('confidence', ascending = False, inp
lace = True)
apriori_rules
```

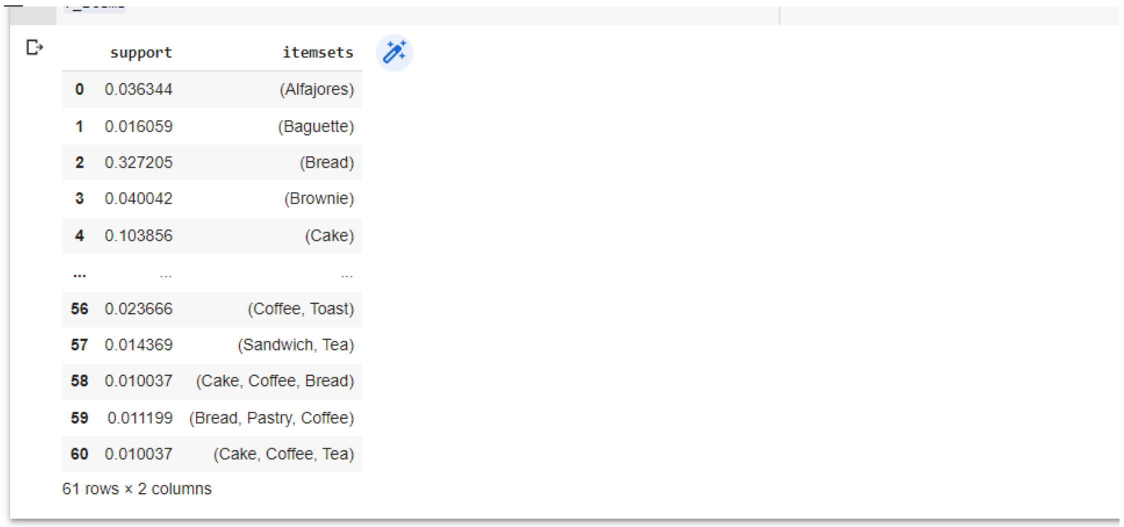


	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(Cake)	(Coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.005044	1.102664
1	(Bread)	(Coffee)	0.327205	0.478394	0.090016	0.275105	0.575059	-0.066517	0.719561
0	(Coffee)	(Bread)	0.478394	0.327205	0.090016	0.188163	0.575059	-0.066517	0.828731
3	(Coffee)	(Cake)	0.478394	0.103856	0.054728	0.114399	1.101515	0.005044	1.011905

```
[11] f_items = apriori(apriori_basket_set, min_support = 0.01, use_colnames = True)
     f_items
```

✓ 0s completed at 14:20

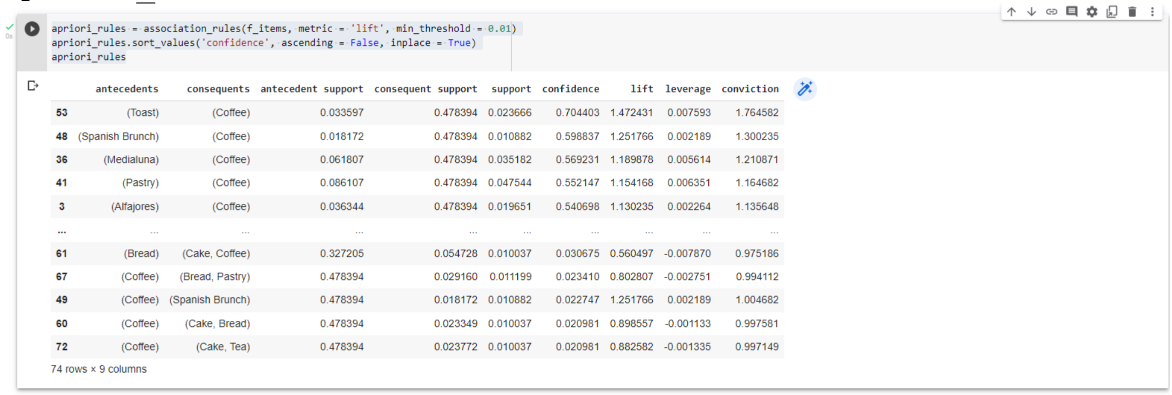
```
     f_items = apriori(apriori_basket_set, min_support = 0.01,
 use_colnames = True)
f_items
```



	support	itemsets
0	0.036344	(Alfajores)
1	0.016059	(Baguette)
2	0.327205	(Bread)
3	0.040042	(Brownie)
4	0.103856	(Cake)
...	...	...
56	0.023666	(Coffee, Toast)
57	0.014369	(Sandwich, Tea)
58	0.010037	(Cake, Coffee, Bread)
59	0.011199	(Bread, Pastry, Coffee)
60	0.010037	(Cake, Coffee, Tea)

61 rows × 2 columns

```
apriori_rules = association_rules(f_items, metric = 'lift', mi
n_threshold = 0.01)
apriori_rules.sort_values('confidence', ascending = False, inp
lace = True)
apriori_rules
```

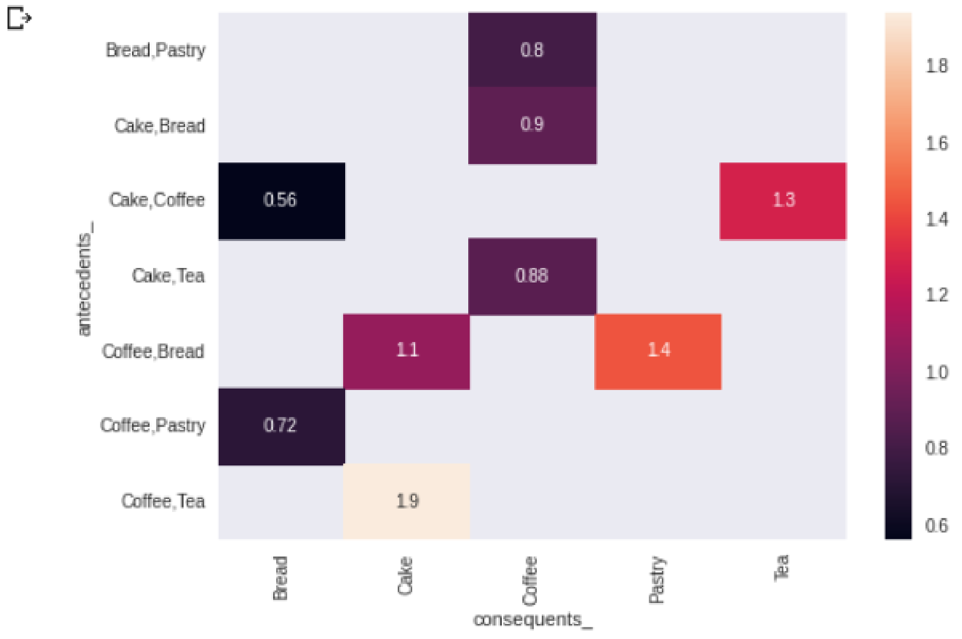


```
apriori_rules = association_rules(f_items, metric = 'lift', min_threshold = 0.01)
apriori_rules.sort_values('confidence', ascending = False, inplace = True)
apriori_rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
53	(Toast)	(Coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.007593	1.764582
48	(Spanish Brunch)	(Coffee)	0.018172	0.478394	0.010882	0.598837	1.251766	0.002189	1.300235
36	(Medialuna)	(Coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.005614	1.210871
41	(Pastry)	(Coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.006351	1.164682
3	(Alfajores)	(Coffee)	0.036344	0.478394	0.019651	0.540698	1.130235	0.002264	1.135648
...	...	...	...	...	...	...	...	...	...
61	(Bread)	(Cake, Coffee)	0.327205	0.054728	0.010037	0.030675	0.560497	-0.007870	0.975186
67	(Coffee)	(Bread, Pastry)	0.478394	0.029160	0.011199	0.023410	0.802807	-0.002751	0.994112
49	(Coffee)	(Spanish Brunch)	0.478394	0.018172	0.010882	0.022747	1.251766	0.002189	1.004682
60	(Coffee)	(Cake, Bread)	0.478394	0.023349	0.010037	0.020981	0.898557	-0.001133	0.997581
72	(Coffee)	(Cake, Tea)	0.478394	0.023772	0.010037	0.020981	0.882582	-0.001335	0.997149

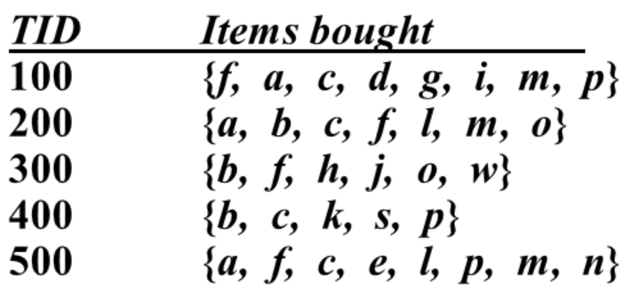
74 rows × 9 columns

```
apriori_rules['lhs_items'] = apriori_rules['antecedents'].appl
y(lambda x:len(x) )
apriori_rules[apriori_rules['lhs_items']>1].sort_values('lift'
, ascending=False).head()
apriori_rules['antecedents_'] = apriori_rules['antecedents'].a
pply(lambda a: ','.join(list(a)))
apriori_rules['consequents_'] = apriori_rules['consequents'].a
pply(lambda a: ','.join(list(a)))
pivot = apriori_rules[apriori_rules['lhs_items']>1].pivot(inde
x = 'antecedents_', columns = 'consequents_', values= 'lift')
sns.heatmap(pivot, annot = True)
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```



Part B:

Program FP tree using inbuilt functions for the following dataset



TID	Items bought
100	{*f, a, c, d, g, i, m, p*}
200	{*a, b, c, f, l, m, o*}
300	{*b, f, h, j, o, w*}
400	{*b, c, k, s, p*}
500	{*a, f, c, e, l, p, m, n*}

Print the frequent patterns generated.

THEORY:

FP Tree
Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset.

The root node represents null while the lower nodes represent the itemsets. The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree.

Frequent Pattern Algorithm Steps
The frequent pattern growth method lets us find the frequent pattern without candidate generation.

Let us see the steps followed to mine the frequent pattern using frequent pattern growth algorithm:
1) The first step is to scan the database to find the occurrences of the itemsets in the database. This step is the same as the first step of Apriori. The count of 1-itemsets in the database is called support count or frequency of 1-itemset.
2) The second step is to construct the FP tree. For this, create the root of the tree. The root is represented by null.
3) The next step is to scan the database again and examine the transactions. Examine the first transaction and find out the itemset in it. The itemset with the max count is taken at the top, the next itemset with lower count and so on. It means that the branch of the tree is constructed with transaction itemsets in descending order of count.
4) The next transaction in the database is examined. The itemsets are ordered in descending order of count. If any itemset of this transaction is already present in another branch (for example in the 1st transaction), then this transaction branch would share a common prefix to the root.
This means that the common itemset is linked to the new node of another itemset in this transaction.

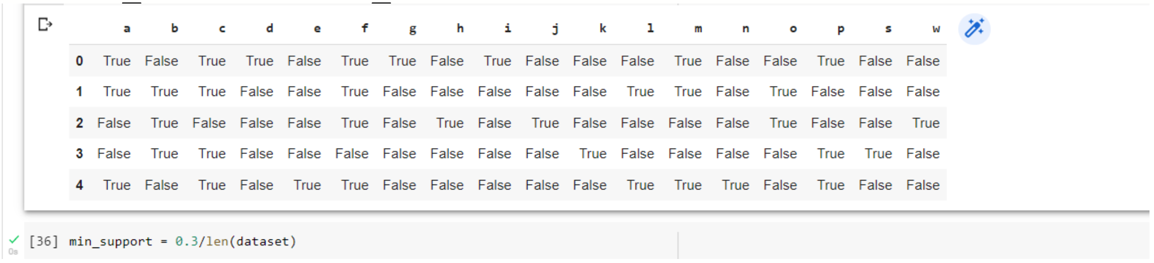
5) Also, the count of the itemset is incremented as it occurs in the transactions. Both the common node and new node count is increased by 1 as they are created and linked according to transactions.
6) The next step is to mine the created FP Tree. For this, the lowest node is examined first along with the links of the lowest nodes. The lowest node represents the frequency pattern

length 1. From this, traverse the path in the FP Tree. This path or paths are called a conditional pattern base.
Conditional pattern base is a sub-database consisting of prefix paths in the FP tree occurring with the lowest node (suffix).

7) Construct a Conditional FP Tree, which is formed by a count of itemsets in the path. The itemsets meeting the threshold support are considered in the Conditional FP Tree.
8) Frequent Patterns are generated from the Conditional FP Tree.

CODE:

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
dataset = [['f', 'a', 'c', 'd', 'g', 'i','m','p'],
           ['a', 'b', 'c', 'f', 'l', 'm','o'],
           ['b', 'f', 'h', 'j','o','w'],
           ['b', 'c', 'k', 's', 'p'],
           ['a', 'f', 'c', 'e', 'l', 'p','m','n']]
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

# instantiate a transaction encoder
my_transactionencoder = TransactionEncoder()

# fit the transaction encoder using the list of transaction tu
ples
my_transactionencoder.fit(dataset)

# transform the list of transaction tuples into an array of en
coded transactions
encoded_transactions = my_transactionencoder.transform(dataset
)

# convert the array of encoded transactions into a dataframe
encoded_transactions_df = pd.DataFrame(encoded_transactions, c
olumns=my_transactionencoder.columns_)
encoded_transactions_df
```

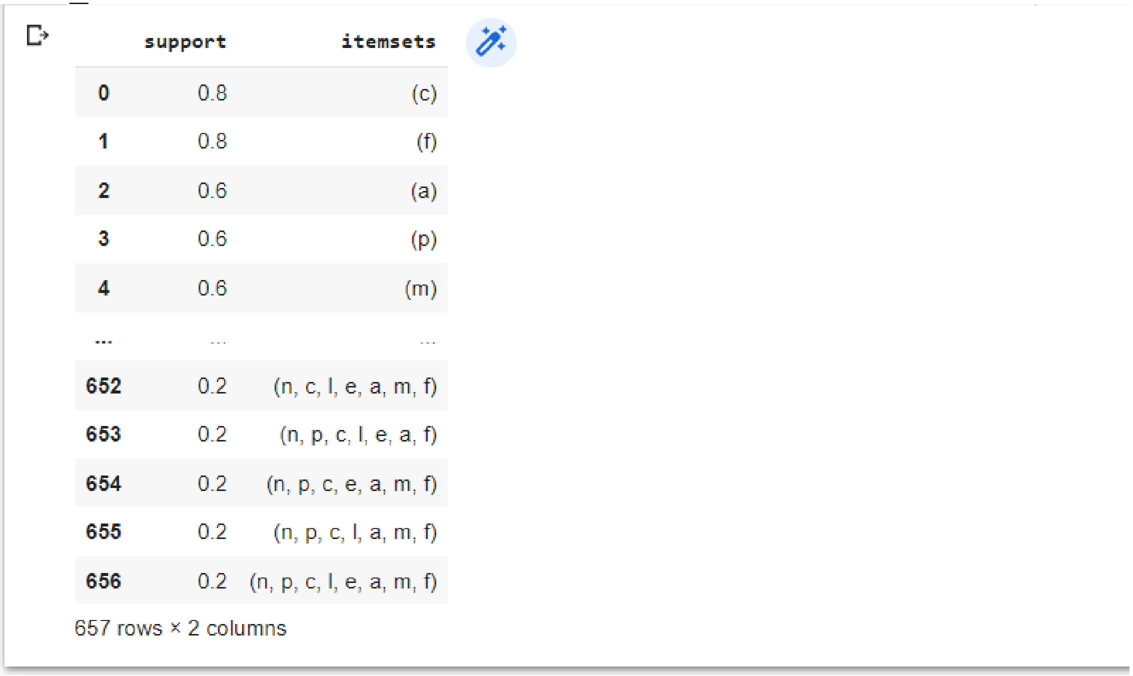


	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	s	w
0	True	False	True	True	False	True	True	False	True	False	False	False	True	False	False	True	False	False
1	True	True	True	False	False	True	False	False	False	False	False	True	True	False	True	False	False	False
2	False	True	False	False	False	True	False	True	False	True	False	False	False	False	True	False	False	True
3	False	True	True	False	False	False	False	False	False	False	True	False	False	False	False	True	True	False
4	True	False	True	False	True	True	False	False	False	False	False	True	True	True	False	True	False	False

```
[36] min_support = 0.3/len(dataset)
```

```
min_support = 0.3/len(dataset)

# compute the frequent itemsets using fpgriowth from mlxtend
```

```
from mlxtend.frequent_patterns.fpgrowth import fpgrowth
frequent_itemsets = fpgrowth(encoded_transactions_df, min_supp
ort=min_support, use_colnames = True)

# print the frequent itemsets
frequent_itemsets
```



```
     support               itemsets
0        0.8                    (c)
1        0.8                    (f)
2        0.6                    (a)
3        0.6                    (p)
4        0.6                    (m)
...      ...                    ...
652      0.2  (n, c, l, e, a, m, f)
653      0.2  (n, p, c, l, e, a, f)
654      0.2  (n, p, c, e, a, m, f)
655      0.2  (n, p, c, l, a, m, f)
656      0.2  (n, p, c, l, e, a, m, f)

657 rows × 2 columns
```

CONCLUSION:

The Apriori algorithm is used for mining association rules. It works on the principle, “the non-empty subsets of frequent itemsets must also be frequent”. It forms k-itemset candidates from (k-1) itemsets and scans the database to find the frequent itemsets.

Frequent Pattern Growth Algorithm is the method of finding frequent patterns without candidate generation. It constructs an FP Tree rather than using the generate and test strategy of Apriori. The focus of the FP Growth algorithm is on fragmenting the paths of the items and mining frequent patterns.