LAB EXPERIMENT NO. 06

NAME:Parth Parekh **SAP ID:60004200006**

BATCH:A1 **BRANCH:COMPUTER ENGG.**

Aim:

Implementation of Association Rule Mining algorithm

1. Apriori algorithm
2. FP Tree algorithm

Perform the experiment in Python.

Read any market basket analysis dataset from UCI dataset repository

Part A:

  Program Apriori from scratch.

  Read min_support and confidence from the user

  Print the association rules

Part B:

  Program FP tree using inbuilt functions .

  Print the frequent patterns generated.

Theory:

Apriori Algorithm

The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a **breadth-first search** and **Hash Tree** to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

Apriori algorithm is a sequence of steps to be followed to find the most frequent itemset in the given database. This data mining technique follows the join and the prune steps iteratively until the most frequent itemset is achieved. A minimum support threshold is given in the problem or it is assumed by the user.

Steps:

1) In the first iteration of the algorithm, each item is taken as a 1-itemsets candidate. The algorithm will count the occurrences of each item.

2) Let there be some minimum support, min_sup (eg 2). The set of 1 – itemsets whose occurrence is satisfying the min sup are determined. Only those candidates which count more than or equal to min_sup, are taken ahead for the next iteration and the others are pruned.

3) Next, 2-itemset frequent items with min_sup are discovered. For this in the join step, the 2-itemset is generated by forming a group of 2 by combining items with itself.

4) The 2-itemset candidates are pruned using min-sup threshold value. Now the table will have 2 – itemsets with min-sup only.

5) The next iteration will form 3 –itemsets using join and prune step. This iteration will follow antimonotone property where the subsets of 3-itemsets, that is the 2 –itemset subsets of each group fall in min_sup. If all 2-itemset subsets are frequent then the superset will be frequent otherwise it is pruned.

6) Next step will follow making 4-itemset by joining 3-itemset with itself and pruning if its subset does not meet the min_sup criteria. The algorithm is stopped when the most frequent itemset is achieved.

Shortcomings Of Apriori Algorithm

1) Using Apriori needs a generation of candidate itemsets. These itemsets may be large in number if the itemset in the database is huge.

2) Apriori needs multiple scans of the database to check the support of each itemset generated and this leads to high costs.

These shortcomings can be overcome using the FP growth algorithm.

FP Tree Algorithm

This algorithm is an improvement to the Apriori method. A frequent pattern is generated without the need for candidate generation. FP growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree.

This tree structure will maintain the association between the itemsets. The database is fragmented using one frequent item. This fragmented part is called "pattern fragment". The itemsets of these fragmented patterns are analyzed. Thus, with this method, the search for frequent itemsets is reduced comparatively.

Steps:

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.

Frequent Patterns are generated from the Conditional FP Tree.

Implementation:

Part A

```
# Apriori Algorithm from scratch

data = [
        ['T100',['I1','I2','I5']],
        ['T200',['I2','I4']],
        ['T300',['I2','I3']],
        ['T400',['I1','I2','I4']],
        ['T500',['I1','I3']],
        ['T600',['I2','I3']],
        ['T700',['I1','I3']],
        ['T800',['I1','I2','I3','I5']],
        ['T900',['I1','I2','I3']]
        ]

init = []
for i in data:
    for q in i[1]:
        if(q not in init):
            init.append(q)
init = sorted(init)
print(init)

sp = 0.4
s = int(sp*len(init))
s

from collections import Counter

c = Counter()
for i in init:
    for d in data:
        if(i in d[1]):
            c[i]+=1
print("C1:")
for i in c:
    print(str([i])+": "+str(c[i]))
print()
l = Counter()
for i in c:
    if(c[i] >= s):
        l[frozenset([i])]+=c[i]
print("L1:")
for i in l:
    print(str(list(i))+": "+str(l[i]))
print()
pl = l
pos = 1
for count in range (2,1000):
    nc = set()
    temp = list(l)
    for i in range(0,len(temp)):
        for j in range(i+1,len(temp)):
            t = temp[i].union(temp[j])
            if(len(t) == count):
                nc.add(temp[i].union(temp[j]))
    nc = list(nc)
    c = Counter()
    for i in nc:
        c[i] = 0
        for q in data:
            temp = set(q[1])
            if(i.issubset(temp)):
                c[i]+=1
    print("C"+str(count)+":")
    for i in c:
        print(str(list(i))+": "+str(c[i]))
    print()
    l = Counter()
    for i in c:
        if(c[i] >= s):
            l[i]+=c[i]
    print("L"+str(count)+":")
    for i in l:
        print(str(list(i))+": "+str(l[i]))
    print()
    if(len(l) == 0):
        break
    pl = l
    pos = count
print("Result: ")
print("L"+str(pos)+":")
for i in pl:
    print(str(list(i))+": "+str(pl[i]))
print()

from itertools import combinations
for l in pl:
    c = [frozenset(q) for q in combinations(l,len(l)-1)]
    mmax = 0
    for a in c:
        b = l-a
        ab = l
        sab = 0
        sa = 0
        sb = 0
        for q in data:
            temp = set(q[1])
            if(a.issubset(temp)):
                sa+=1
            if(b.issubset(temp)):
                sb+=1
            if(ab.issubset(temp)):
                sab+=1
        temp = sab/sa*100
        if(temp > mmax):
            mmax = temp
        temp = sab/sb*100
        if(temp > mmax):
            mmax = temp
        print(str(list(a))+" -> "+str(list(b))+" = "+str(sab/sa*100)+"%")
        print(str(list(b))+" -> "+str(list(a))+" = "+str(sab/sb*100)+"%")
    curr = 1
    print("choosing:", end=' ')
    for a in c:
        b = l-a
        ab = l
        sab = 0
        sa = 0
        sb = 0
        for q in data:
            temp = set(q[1])
            if(a.issubset(temp)):
                sa+=1
            if(b.issubset(temp)):
                sb+=1
            if(ab.issubset(temp)):
                sab+=1
        temp = sab/sa*100
        if(temp == mmax):
            print(curr, end = ' ')
        curr += 1
        temp = sab/sb*100
        if(temp == mmax):
            print(curr, end = ' ')
        curr += 1


# Apriori Algorithm using inbuilt functions
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# !pip install apyori
from apyori import apriori

dataset=pd.read_csv('Market_Basket_Optimisation.csv', header=None)
transactions=[]

for i in range (0,7501):
 transactions.append([str(dataset.values[i,j]) for j in range(0,20)])

rules=apriori(transactions=transactions, min_support=0.003, min_confidence
=0.2, min_lift=3, min_length=2, max_length=2)

results=list(rules)

for i in results:
  print(i, end="\n\n")

print()

def inspect(results):
 lhs = [tuple(result[2][0][0])[0] for result in results]
 rhs = [tuple(result[2][0][1])[0] for result in results]
 supports = [result[1] for result in results]
 confidences = [result[2][0][2] for result in results]
 lifts = [result[2][0][3] for result in results]
 return list(zip(lhs, rhs, supports, confidences, lifts))

resultsinDataFrame = pd.DataFrame(inspect(results), columns = ['Left Hand
Side', 'Right Hand Side', 'Support', 'Confidence', 'Lift'])
print(resultsinDataFrame.nlargest(n = 10, columns = 'Lift'))
```

Part B

```
# FP Tree using inbuilt functions

import pandas as pd
import numpy as np

# %pip install mlxtend --upgrade
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules
```

```
# Load the dataset from excel sheet
df = pd.read_excel('https://archive.ics.uci.edu/ml/machine-learning-
databases/00352/Online%20Retail.xlsx')

# Data cleaning Process
# Remove extra spaces in the description part of the data
df['Description'] = df['Description'].str.strip()

# Drop the rows that are without Invoice Number
df.dropna(axis=0, subset=['InvoiceNo'], inplace=True)
df['InvoiceNo'] = df['InvoiceNo'].astype('str')

# Remove the credit transactions (Credit transactions contain 'C')
df = df[~df['InvoiceNo'].str.contains('C')]

# Consolidate the items into 1 transaction per row with each product 1 hot
 encoded.
'''
Hot encoding, the basic strategy is to convert each category value into a
new column and assigns a 1 or 0 (T
rue/False) value to the column
'''
# We will look for the sales of France
basket = (df[df['Country'] =="France"]
 .groupby(['InvoiceNo', 'Description'])['Quantity']
 .sum().unstack().reset_index().fillna(0)
 .set_index('InvoiceNo'))

'''
The lots of zeros in the dataset. We need to make sure that all positive n
umbers are converted to 1 and oth
ers are 0
we create a function for that named encode_units which returns 1 for all p
ositive numbers and 0 otherwis
e
'''

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

# To make the data structured.
```

```
basket_sets = basket.applymap(encode_units)

# This step will complete the one hot encoding of the data and remove the
postage column (since that charge is not one we wish to explore).
basket_sets.drop('POSTAGE', inplace=True, axis=1)

# Get the frequent item sets using fp tree with minimum support 0.07 and c
olumn names as items
frequent_itemsets = fpgrowth.fpgrowth(basket_sets, min_support=0.07, use_c
olnames=True)
print(frequent_itemsets)
print()

# Get the association rules based on the fp tree generated frequent item s
ets
rules = association_rules(frequent_itemsets, metric="confidence", min_thre
shold=0.7)
rules[ (rules['lift'] >= 6) &
 (rules['confidence'] >= 0.8) ]
```

Output

Part A (From scratch & using inbuilt functions)

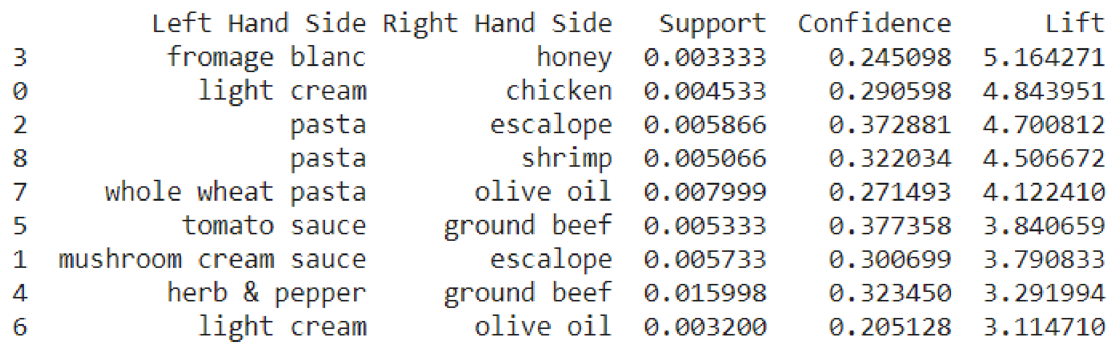
From Scratch

```
['I1', 'I2', 'I3', 'I4', 'I5']
C1:
['I1']: 6
['I2']: 7
['I3']: 6
['I4']: 2
['I5']: 2

L1:
['I1']: 6
['I2']: 7
['I3']: 6
['I4']: 2
['I5']: 2

C2:
['I3', 'I4']: 0
['I3', 'I2']: 4
['I1', 'I5']: 2
['I3', 'I1']: 4
['I4', 'I5']: 0
['I1', 'I4']: 1
```

```
['I3', 'I5']: 1
['I1', 'I2']: 4
['I2', 'I5']: 2
['I4', 'I2']: 2

L2:
['I3', 'I2']: 4
['I1', 'I5']: 2
['I3', 'I1']: 4
['I1', 'I2']: 4
['I2', 'I5']: 2
['I4', 'I2']: 2

C3:
['I4', 'I2', 'I5']: 0
['I3', 'I1', 'I2']: 2
['I3', 'I2', 'I5']: 1
['I1', 'I2', 'I5']: 2
['I3', 'I4', 'I2']: 0
['I1', 'I2', 'I4']: 1
['I3', 'I1', 'I5']: 1

L3:
['I3', 'I1', 'I2']: 2
['I1', 'I2', 'I5']: 2

C4:
['I1', 'I5', 'I3', 'I2']: 1

L4:

Result:
L3:
['I3', 'I1', 'I2']: 2
['I1', 'I2', 'I5']: 2

['I3', 'I1'] -> ['I2'] = 50.0%
['I2'] -> ['I3', 'I1'] = 28.57142857142857%
['I3', 'I2'] -> ['I1'] = 50.0%
['I1'] -> ['I3', 'I2'] = 33.33333333333333%
['I1', 'I2'] -> ['I3'] = 50.0%
['I3'] -> ['I1', 'I2'] = 33.33333333333333%
choosing: 1 3 5 ['I1', 'I2'] -> ['I5'] = 50.0%
['I5'] -> ['I1', 'I2'] = 100.0%
['I1', 'I5'] -> ['I2'] = 100.0%
['I2'] -> ['I1', 'I5'] = 28.57142857142857%
['I2', 'I5'] -> ['I1'] = 100.0%
['I1'] -> ['I2', 'I5'] = 33.33333333333333%
choosing: 2 3 5
```

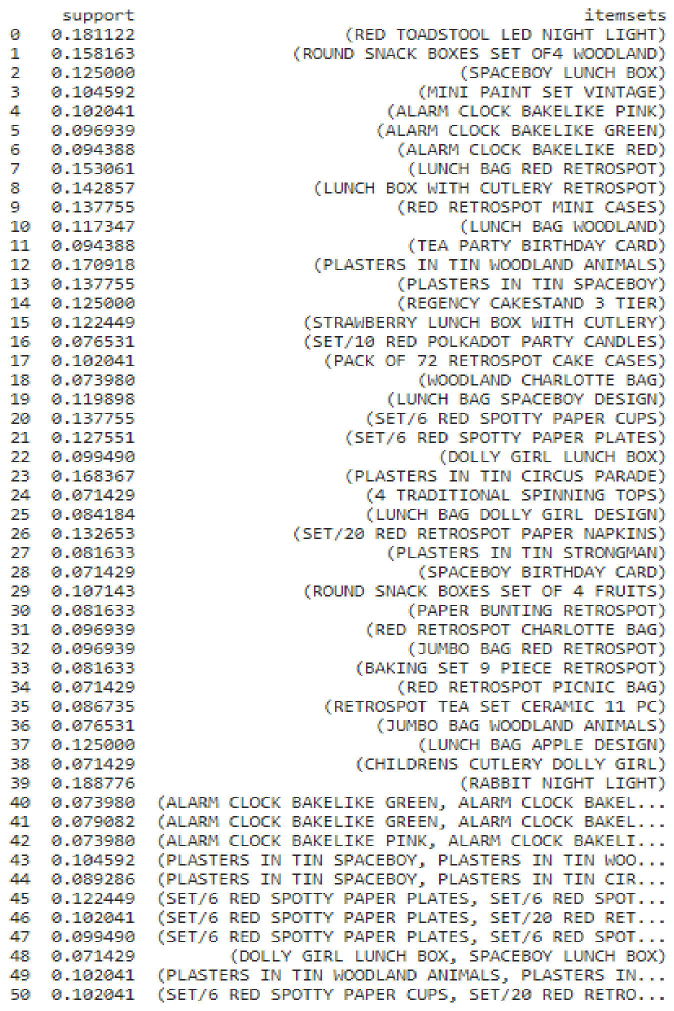
Using inbuilt functions

```
RelationRecord(items=frozenset({'chicken', 'light cream'}),
support=0.004532728969470737,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'light
cream'}), items_add=frozenset({'chicken'}),
confidence=0.29059829059829057, lift=4.84395061728395)])

RelationRecord(items=frozenset({'escalope', 'mushroom cream sauce'}),
support=0.005732568990801226,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'mushroom cream
sauce'}), items_add=frozenset({'escalope'}),
confidence=0.3006993006993007, lift=3.790832696715049)])

RelationRecord(items=frozenset({'escalope', 'pasta'}),
support=0.005865884548726837,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'pasta'}),
items_add=frozenset({'escalope'}), confidence=0.3728813559322034,
lift=4.700811850163794)])

RelationRecord(items=frozenset({'fromage blanc', 'honey'}),
support=0.003332888948140248,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'fromage
blanc'}), items_add=frozenset({'honey'}), confidence=0.2450980392156863,
lift=5.164270764485569)])

RelationRecord(items=frozenset({'herb & pepper', 'ground beef'}),
support=0.015997866951073192,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'herb &
pepper'}), items_add=frozenset({'ground beef'}),
confidence=0.3234501347708895, lift=3.2919938411349285)])

RelationRecord(items=frozenset({'tomato sauce', 'ground beef'}),
support=0.005332622317024397,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'tomato
sauce'}), items_add=frozenset({'ground beef'}),
confidence=0.3773584905660377, lift=3.840659481324083)])

RelationRecord(items=frozenset({'light cream', 'olive oil'}),
support=0.003199573390214638,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'light
cream'}), items_add=frozenset({'olive oil'}),
confidence=0.20512820512820515, lift=3.1147098515519573)])

RelationRecord(items=frozenset({'olive oil', 'whole wheat pasta'}),
support=0.007998933475536596,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'whole wheat
pasta'}), items_add=frozenset({'olive oil'}),
confidence=0.2714932126696833, lift=4.122410097642296)])

RelationRecord(items=frozenset({'shrimp', 'pasta'}),
support=0.005065991201173177,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'pasta'}),
items_add=frozenset({'shrimp'}), confidence=0.3220338983050847,
lift=4.506672147735896)])
```

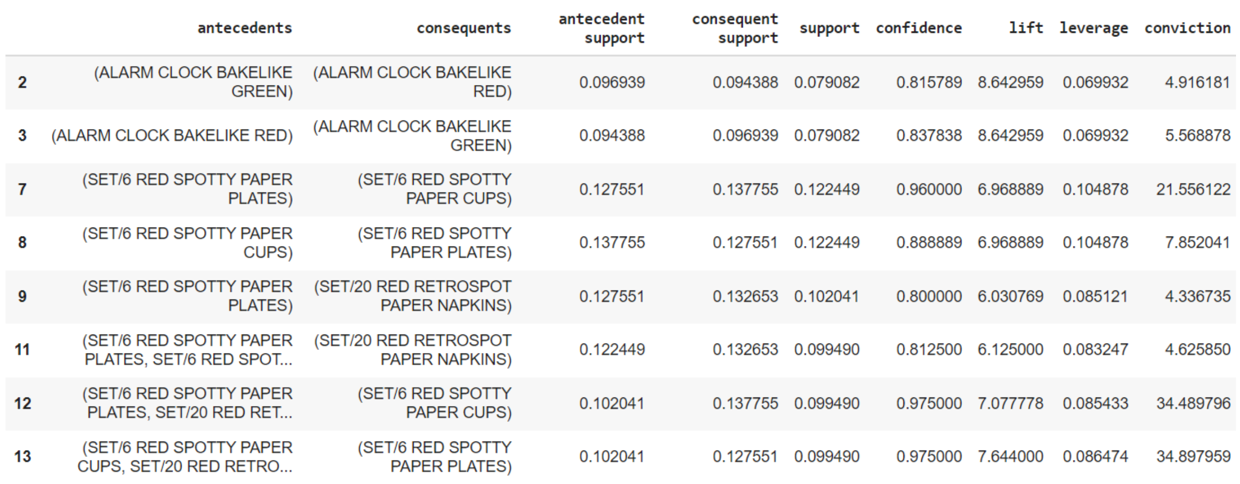


```
       Left Hand Side Right Hand Side   Support  Confidence      Lift
3       fromage blanc           honey  0.003333    0.245098  5.164271
0         light cream         chicken  0.004533    0.290598  4.843951
2               pasta        escalope  0.005866    0.372881  4.700812
8               pasta          shrimp  0.005066    0.322034  4.506672
7   whole wheat pasta       olive oil  0.007999    0.271493  4.122410
5        tomato sauce     ground beef  0.005333    0.377358  3.840659
1 mushroom cream sauce       escalope  0.005733    0.300699  3.790833
4        herb & pepper    ground beef  0.015998    0.323450  3.291994
6         light cream       olive oil  0.003200    0.205128  3.114710
```

Part B



```
     support                                         itemsets
0   0.181122                    (RED TOADSTOOL LED NIGHT LIGHT)
1   0.158163                (ROUND SNACK BOXES SET OF4 WOODLAND)
2   0.125000                              (SPACEBOY LUNCH BOX)
3   0.104592                           (MINI PAINT SET VINTAGE)
4   0.102041                          (ALARM CLOCK BAKELIKE PINK)
5   0.096939                         (ALARM CLOCK BAKELIKE GREEN)
6   0.094388                           (ALARM CLOCK BAKELIKE RED)
7   0.153061                            (LUNCH BAG RED RETROSPOT)
8   0.142857                  (LUNCH BOX WITH CUTLERY RETROSPOT)
9   0.137755                          (RED RETROSPOT MINI CASES)
10  0.117347                               (LUNCH BAG WOODLAND)
11  0.094388                            (TEA PARTY BIRTHDAY CARD)
12  0.170918                  (PLASTERS IN TIN WOODLAND ANIMALS)
13  0.137755                          (PLASTERS IN TIN SPACEBOY)
14  0.125000                          (REGENCY CAKESTAND 3 TIER)
15  0.122449                   (STRAWBERRY LUNCH BOX WITH CUTLERY)
16  0.076531                   (SET/10 RED POLKADOT PARTY CANDLES)
17  0.102041                    (PACK OF 72 RETROSPOT CAKE CASES)
18  0.073980                           (WOODLAND CHARLOTTE BAG)
19  0.119898                          (LUNCH BAG SPACEBOY DESIGN)
20  0.137755                        (SET/6 RED SPOTTY PAPER CUPS)
21  0.127551                      (SET/6 RED SPOTTY PAPER PLATES)
22  0.099490                              (DOLLY GIRL LUNCH BOX)
23  0.168367                      (PLASTERS IN TIN CIRCUS PARADE)
24  0.071429                        (4 TRADITIONAL SPINNING TOPS)
25  0.084184                        (LUNCH BAG DOLLY GIRL DESIGN)
26  0.132653                   (SET/20 RED RETROSPOT PAPER NAPKINS)
27  0.081633                         (PLASTERS IN TIN STRONGMAN)
28  0.071429                            (SPACEBOY BIRTHDAY CARD)
29  0.107143                   (ROUND SNACK BOXES SET OF 4 FRUITS)
30  0.081633                           (PAPER BUNTING RETROSPOT)
31  0.096939                        (RED RETROSPOT CHARLOTTE BAG)
32  0.096939                           (JUMBO BAG RED RETROSPOT)
33  0.081633                        (BAKING SET 9 PIECE RETROSPOT)
34  0.071429                          (RED RETROSPOT PICNIC BAG)
35  0.086735                       (RETROSPOT TEA SET CERAMIC 11 PC)
36  0.076531                        (JUMBO BAG WOODLAND ANIMALS)
37  0.125000                           (LUNCH BAG APPLE DESIGN)
38  0.071429                        (CHILDRENS CUTLERY DOLLY GIRL)
39  0.188776                                (RABBIT NIGHT LIGHT)
40  0.073980  (ALARM CLOCK BAKELIKE GREEN, ALARM CLOCK BAKEL...
41  0.079082  (ALARM CLOCK BAKELIKE GREEN, ALARM CLOCK BAKEL...
42  0.073980  (ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELI...
43  0.104592  (PLASTERS IN TIN SPACEBOY, PLASTERS IN TIN WOO...
44  0.089286  (PLASTERS IN TIN SPACEBOY, PLASTERS IN TIN CIR...
45  0.122449  (SET/6 RED SPOTTY PAPER PLATES, SET/6 RED SPOT...
46  0.102041  (SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...
47  0.099490  (SET/6 RED SPOTTY PAPER PLATES, SET/6 RED SPOT...
48  0.071429              (DOLLY GIRL LUNCH BOX, SPACEBOY LUNCH BOX)
49  0.102041  (PLASTERS IN TIN WOODLAND ANIMALS, PLASTERS IN...
50  0.102041  (SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...
```



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.096939	0.094388	0.079082	0.815789	8.642959	0.069932	4.916181
3	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.094388	0.096939	0.079082	0.837838	8.642959	0.069932	5.568878
7	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.127551	0.137755	0.122449	0.960000	6.968889	0.104878	21.556122
8	(SET/6 RED SPOTTY PAPER CUPS)	(SET/6 RED SPOTTY PAPER PLATES)	0.137755	0.127551	0.122449	0.888889	6.968889	0.104878	7.852041
9	(SET/6 RED SPOTTY PAPER PLATES)	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.127551	0.132653	0.102041	0.800000	6.030769	0.085121	4.336735
11	(SET/6 RED SPOTTY PAPER PLATES, SET/6 RED SPOT...	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.122449	0.132653	0.099490	0.812500	6.125000	0.083247	4.625850
12	(SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...	(SET/6 RED SPOTTY PAPER CUPS)	0.102041	0.137755	0.099490	0.975000	7.077778	0.085433	34.489796
13	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	(SET/6 RED SPOTTY PAPER PLATES)	0.102041	0.127551	0.099490	0.975000	7.644000	0.086474	34.897959

Conclusion:

Thus, we have successfully implemented successfully implemented Association Rule Mining in Python.

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.