Experiment No: 09	TE AI&DS
Date of Performance: 17/10/20233	Roll No: 9696
Aim: To Implement Apriory algorithm to find frequent itemsets and strong association rules.	
Related CO4: Implement Classification, Clustering and Association mining techniques to extract knowledge	
Related LO 5. Perform exploratory analysis of the data to be used for mining. **LO6**: Implement the appropriate data mining methods like classification, clustering or Frequent Pattern mining on large data sets. **Objective:** To learn and implement Association Mining techniques.	

Rubrics for assessment of Experiment:

Sr. No	Parameters	Exceed Expectations(EE)	Meet Expectations (ME)	Below Expectations (BE)
1	Timeline (2)	Early or on time (2)	One session late (1)	More than one session late (0)
2	Preparedness (2)	Knows the basic theory related to the experiment very well. (2)	Managed to explain the theory related to the experiment. (1)	Not aware of the theory to the point. (1)
3	Effort (3)	Done expt on their own. (3)	Done expt with help from other. (2)	Just managed. (1)
4	Documentation(2)	Lab experiment is documented in proper format and maintained neatly. (2)	Documented in proper format but some formatting guidelines are missed. (1)	Experiments not written in proper format (0.5)
5	Result (1)	Specific conclusion.(1)	Partially specific conclusion. (0.5)	Not specific at all. (0)

Assessment Marks:

Timeline(2)	Preparedness(2)	Effort(3)	Documentation(2)	Result(1)	Total(10)

Theory:

Theory : Frequent patterns are patterns (such as itemsets, subsequences or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a frequent sequential pattern. A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructures occurs frequently, it is called a frequent structured pattern. Finding such frequent patterns plays an essential role in mining associations, correlations and many other interesting relationships among data. A typical example of frequent itemset mining is **market basket analysis**. This process analyzes customer buying habits by finding association between different items that customers place in their shopping basket. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. In general, association rule mining can be viewed as a two-step process:

1. Find all frequent itemsets : By definition, each of these itemsets will occur atleast as frquently as a predetermined minimum support count.

2. Generate strong association rules fro the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

An itemset X is closed in a data set S if there exists no proper super-itemset Y such that Y has the same support count as X in S. An itemset X is a closed frequent itemset in set S if X is both closed and frequent in S. An itemset X is a maximal frequent itemset in set S if X is frequent, and there exists no super-itemset Y such that X is subset of Y and Y is frequent in S.

Apriori is a seminal algorithm proposed by R. Agrawal and R.Srikant in 1994 for mining frequent itemsets for boolean association rule. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties.

In data mining, **Apriori** is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation).
Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing).
Overview
The whole point of the algorithm (and data mining, in general) is to extract useful information from large amounts of data. For example, the information that a customer who purchases a keyboard also tends to buy a mouse at the same time is acquired from the association rule below:
Support: The percentage of task-relevant data transactions for which the pattern is true.

Support (Keyboard -> Mouse)

$$= \frac{\text{No. of transacti ons containing both Keyboard and Mouse}}{\text{No. of total transacti ons}}$$

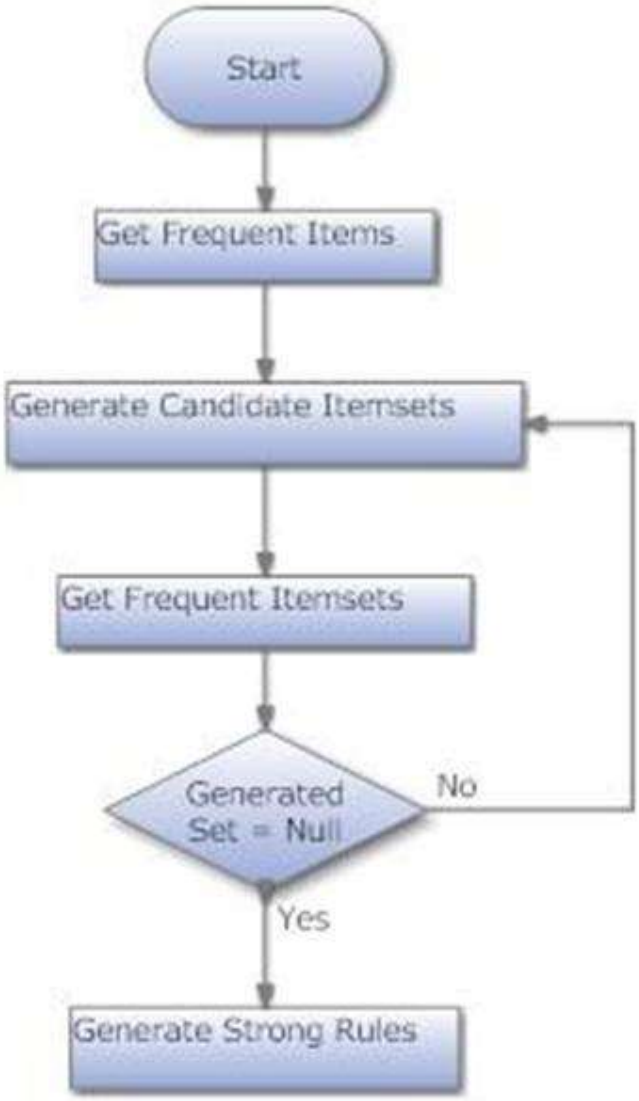
Confidence: The measure of certainty or trustworthiness associated with each discovered pattern.
Confidence (Keyboard -> Mouse) =
The algorithm aims to find the rules which satisfy both a minimum support threshold and a minimum confidence threshold (Strong Rules).

- Item: article in the basket.
    - Itemset: a group of items purchased together in a single transaction.

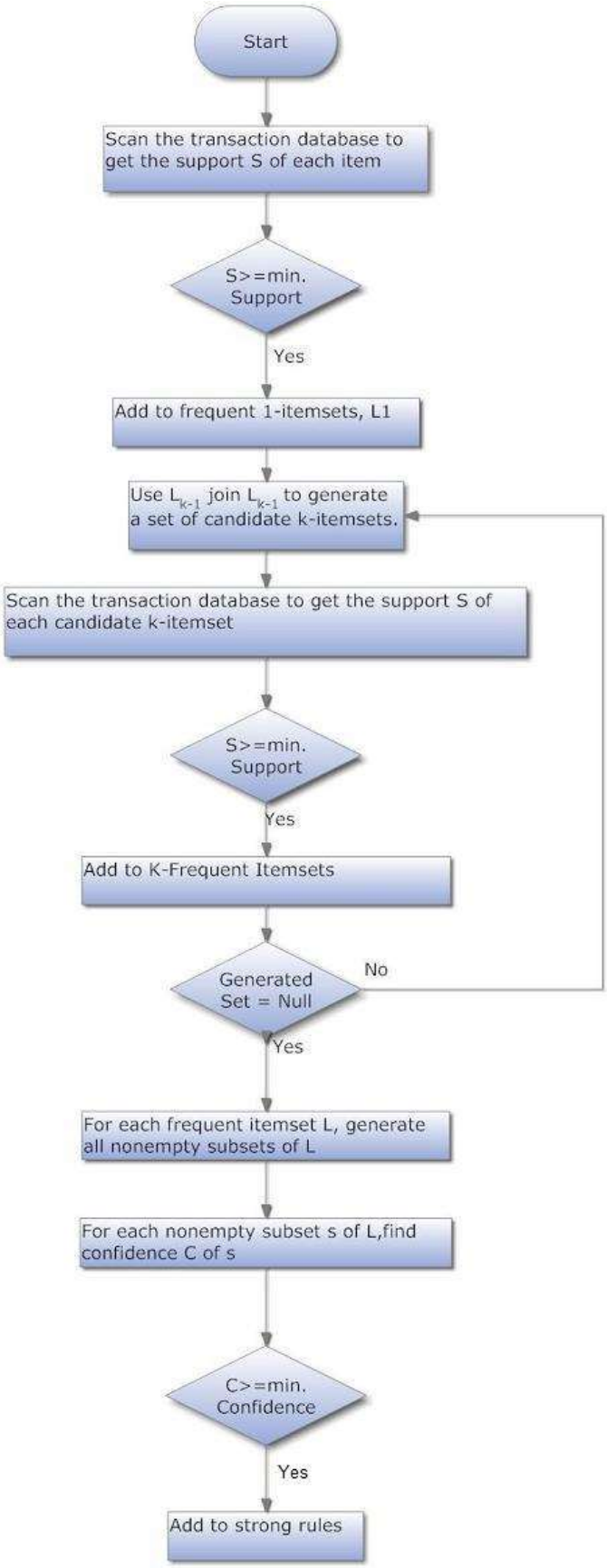
How Apriori Works

1. Find all frequent itemsets:
    - Get frequent items:
        - Items whose occurrence in database is greater than or equal to the min.support threshold.
    - Get frequent itemsets:
        - Generate candidates from frequent items.
        - Prune the results to find the frequent itemsets.
2. Generate strong association rules from frequent itemsets
    - Rules which satisfy the min.support and min.confidence threshold.

High Level Design



Low Level Design

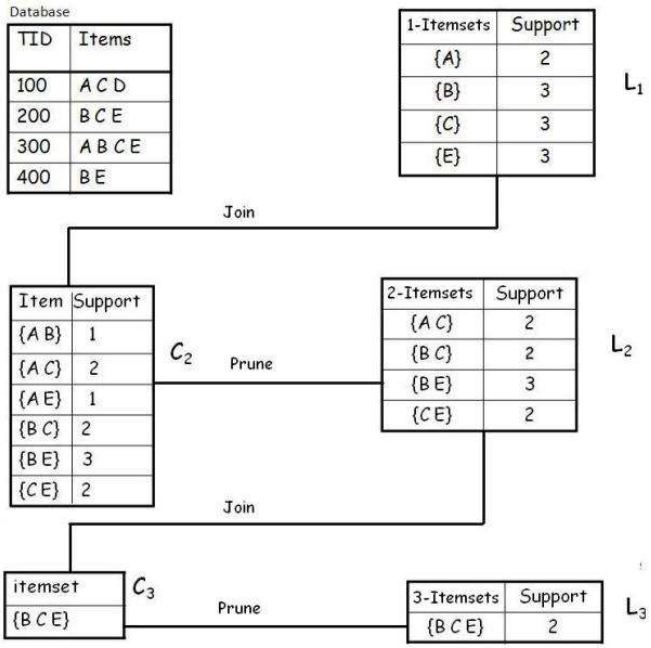

Start
Scan the transaction database to get the support S of each item
S>=min. Support
Yes
Add to frequent 1-itemsets, L1
Use L_{k-1} join L_{k-1} to generate a set of candidate k-itemsets.
Scan the transaction database to get the support S of each candidate k-itemset
S>=min. Support
Yes
Add to K-Frequent Itemsets
Generated Set = Null
No
Yes
For each frequent itemset L, generate all nonempty subsets of L
For each nonempty subset s of L,find confidence C of s
C>=min. Confidence
Yes
Add to strong rules

Example
A database has five transactions. Let the min sup = 50% and min con f = 80%.

Database

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Solution
Step 1: Find all Frequent Itemsets



Frequent Itemsets
Hide Copy Code
{A} {B} {C} {E} {A C} {B C} {B E} {C E} {B C E}
Step 2: Generate strong association rules from the frequent itemsets

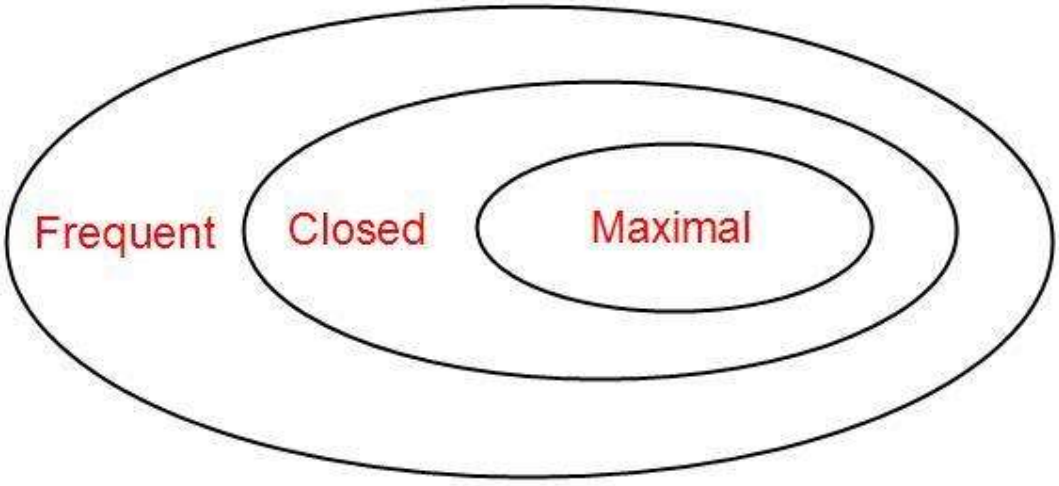
Rules	Support (X Y)	Support(X)	Confidence
{A} -> {C}	2	2	100
{B} -> {C}	2	3	66.66666667
{B} -> {E}	3	3	100
{C} -> {E}	2	3	66.66666667
{B} -> {C E}	2	3	66.66666667
{C} -> {B E}	2	3	66.66666667
{E} -> {B C}	2	3	66.66666667
{C} -> {A}	2	3	66.66666667
{C} -> {B}	2	3	66.66666667
{E} -> {B}	3	3	100
{E} -> {C}	2	3	66.66666667
{C E} -> {B}	2	2	100
{B E} -> {C}	2	3	66.66666667
{B C} -> {E}	2	2	100

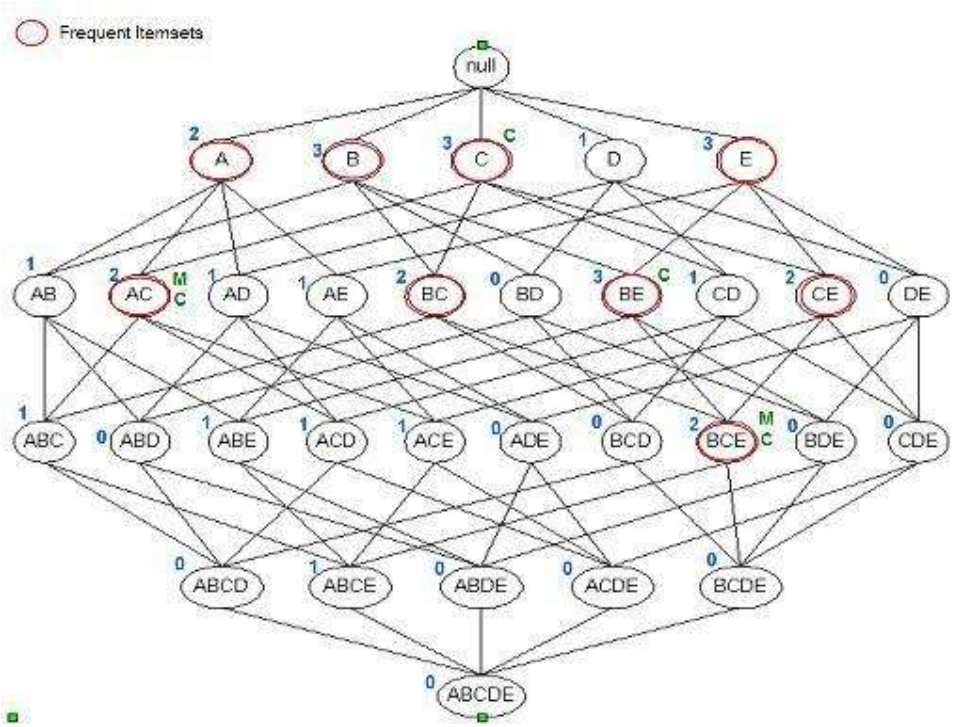
Lattice

Closed Itemset: support of all parents are not equal to the support of the itemset.

Maximal Itemset: all parents of that itemset must be infrequent.

Keep in mind:





Itemset {c} is closed as support of parents (supersets) {A C}:2, {B C}:2, {C D}:1, {C E}:2 not equal support of {c}:3.
And the same for {A C}, {B E} & {B C E}.
Itemset {A C} is maximal as all parents (supersets) {A B C}, {A C D}, {A C E} are infrequent.
And the same for {B C E

Implementation:

Part1 : WAP in Python/java to implement Apriori –Attach code and output
Apriori Algorithm (Python 3.0) - A Data Analyst
Apriori Algorithm from Scratch - Python (vucreations.com)
Apriori: Association Rule Mining In-depth Explanation and Python Implementation | by Chonyy | Towards Data Science)

Part2: using Std libraries- Attach code and output
Link to download dataset(Grocery Store Data Set | Kaggle)
Implementing Apriori algorithm in Python - GeeksforGeeks
Give summary table Comparison of Pattern Evaluation Measures Using Contingency Tables for a Variety of Data Sets(refer text book table 6.9)(CHI-SQAURE, *lift, all conf, max con,. Kulc, cosine)*

Other resources:
Beginner's Guide To Apriori Algorithm With Implementation In Python (analyticsindiamag.com)
Introduction to Market Basket Analysis in Python - Practical Business Python (pbpython.com)
How to Create Data Visualization for Association Rules in Data Mining - Machine Learning Applications (intelligentonlinetools.com)
Apriori - mlxtend (rasbt.github.io)

Code with output:

Part 1:

Code:

```
from itertools import combinations
from collections import Counter

data = [
        ['T100',['A', 'C', 'D']],
        ['T200',['B', 'C', 'E']],
        ['T300',['A', 'B', 'C', 'E']],
        ['T400',['B', 'E']],
        ]

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

# min support
sp = 0.4
s = int(sp*len(init))
s

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()

# generate association rules
for l in pl:
    c = [frozenset(q) for q in combinations(l,len(l)-1)]
    mmax = 0

    # calculate confidence and print association rules
    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
    print()
    print()
```

Output:

```
['A', 'B', 'C', 'D', 'E']
C1:
['A']: 2
['B']: 3
['C']: 3
['D']: 1
['E']: 3
```

L1:
['A']: 2
['B']: 3
['C']: 3
['E']: 3

C2:
['E', 'C']: 2
['E', 'A']: 1
['A', 'C']: 2
['E', 'B']: 3
['B', 'A']: 1
['B', 'C']: 2

L2:
['E', 'C']: 2
['A', 'C']: 2
['E', 'B']: 3
['B', 'C']: 2

C3:
['E', 'B', 'C']: 2
['E', 'A', 'C']: 1
['B', 'A', 'C']: 1

L3:
['E', 'B', 'C']: 2

C4:

L4:

Result:
L3:
['E', 'B', 'C']: 2

['E', 'B'] -> ['C'] = 66.66666666666666%
['C'] -> ['E', 'B'] = 66.66666666666666%
['E', 'C'] -> ['B'] = 100.0%
['B'] -> ['E', 'C'] = 66.66666666666666%
['B', 'C'] -> ['E'] = 100.0%
['E'] -> ['B', 'C'] = 66.66666666666666%
choosing: 3 5

Part 2:

Code:

```
from apyori import apriori

# Define a list of transactions
transactions = [
    ['MILK', 'BREAD', 'BISCUIT'],
    ['MILK', 'BISCUIT'],
    ['MILK', 'CEREAL'],
    ['BREAD', 'BISCUIT'],
    ['BREAD', 'CEREAL'],
    ['BISCUIT', 'CEREAL'],
]

# Adjusted parameters
min_support = 0.1  # Adjusted for demonstration
min_confidence = 0.1  # Adjusted for demonstration
min_lift = 1
min_length = 2

# Apply Apriori algorithm
rules = apriori(transactions, min_support=min_support, min_confidence=min_confidence)

# Print the rules and details
for rule in rules:
    print("Rule:", rule.items)
    print("Support:", rule.support)
    print("Confidence:", rule.ordered_statistics[0].confidence)
    print("Lift:", rule.ordered_statistics[0].lift)
    print()
```

Output:

```
Rule: frozenset({'BISCUIT'})
Support: 0.6666666666666666
Confidence: 0.6666666666666666
Lift: 1.0

Rule: frozenset({'BREAD'})
Support: 0.5
Confidence: 0.5
Lift: 1.0

Rule: frozenset({'CEREAL'})
Support: 0.5
Confidence: 0.5
Lift: 1.0
```

Rule: frozenset({'MILK'})
Support: 0.5
Confidence: 0.5
Lift: 1.0

Rule: frozenset({'BISCUIT', 'BREAD'})
Support: 0.3333333333333333
Confidence: 0.3333333333333333
Lift: 1.0

Rule: frozenset({'CEREAL', 'BISCUIT'})
Support: 0.16666666666666666
Confidence: 0.16666666666666666
Lift: 1.0

Rule: frozenset({'BISCUIT', 'MILK'})
Support: 0.3333333333333333
Confidence: 0.3333333333333333
Lift: 1.0

Rule: frozenset({'CEREAL', 'BREAD'})
Support: 0.16666666666666666
Confidence: 0.16666666666666666
Lift: 1.0

Rule: frozenset({'BREAD', 'MILK'})
Support: 0.16666666666666666
Confidence: 0.16666666666666666
Lift: 1.0

Rule: frozenset({'CEREAL', 'MILK'})
Support: 0.16666666666666666
Confidence: 0.16666666666666666
Lift: 1.0

Rule: frozenset({'BISCUIT', 'BREAD', 'MILK'})
Support: 0.16666666666666666
Confidence: 0.16666666666666666
Lift: 1.0

Conclusion: I understand apriori algorithm and able to implement the code