Experiment 5

**AIM:** Implementation of Association rule mining Using

1) Apriori Algorithm

2) FPTree

Part A:

1. Program Apriori from scratch.
2. Read min\_support and confidence from the user
3. Print the association rules

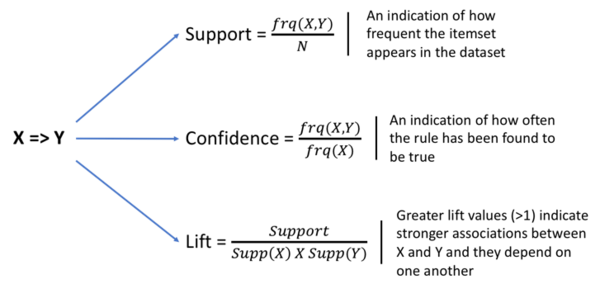
Part B:

1. Program FP tree using inbuilt functions.
2. Print the frequent patterns generated.

**THEORY:**

**Association Rule Algorithm:**

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness. The rule shows how frequently an itemset occurs in a transaction.



**Apriori Algorithm:**

Apriori algorithm is used for finding frequent itemsets in a dataset for boolean association rule. The name Apriori, is because it uses prior knowledge of frequent itemset properties. It is designed to work on the databases that contain transactions.

Apriori uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length k-1. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

**Limitations:** Time and Space complexity of the algorithm are very high: O(2| D | ), where |D| is the horizontal width (the total number of items) present in the database.

To improve the efficiency of level-wise generation of frequent itemsets, an important property is used called Apriori property which helps by reducing the search space.

**FP Tree Algorithm:**

A FP-tree(or Frequent Pattern Tree) is a compact data structure that represents the data set in tree form. Each transaction is read and then mapped onto a path in the FP-tree which is used to maintain associations between the itemsets. This is done until all transactions have been read. Different transactions that have common subsets allow the tree to remain compact because their paths overlap.

A picture containing table

Description automatically generated

**CODE:**

**Part A:**

**1) Program Apriori from scratch.**

**2) Read min\_support and confidence from the user**

**3) Print the association rules**

import pandas as pd

import numpy as np

import math

transaction\_df = pd.read\_csv('GroceryStoreDataSet.csv') print(transaction\_df)

transaction\_df.index.rename('TID', inplace=True)

transaction\_df.rename(columns={'MILK,BREAD,BISCUIT' : 'item\_list'}, inplace=True)

trans\_df = transaction\_df.item\_list.str.split(',') def prune(data,supp):

df = data[data.supp\_count >= supp] return df

def count\_itemset(transaction\_df, itemsets): count\_item = {}

for item\_set in itemsets: set\_A = set(item\_set) for row in trans\_df: set\_B = set(row)

if set\_B.intersection(set\_A) == set\_A: if item\_set in count\_item.keys(): count\_item[item\_set] += 1

else:

count\_item[item\_set] = 1 data = pd.DataFrame() data['item\_sets'] = count\_item.keys()

data['supp\_count'] = count\_item.values() return data

def count\_item(trans\_items): count\_ind\_item={}

for row in trans\_items: for i in range(len(row)):

if row[i] in count\_ind\_item.keys(): count\_ind\_item[row[i]] += 1

else:

count\_ind\_item[row[i]] = 1 data = pd.DataFrame()

data['item\_sets'] = count\_ind\_item.keys() data['supp\_count'] = count\_ind\_item.values() data = data.sort\_values('item\_sets')

return data

def join(list\_of\_items): itemsets = []

i = 1

for entry in list\_of\_items: proceding\_items = list\_of\_items[i:] for item in proceding\_items:

if(type(item) is str): if entry != item:

tuples = (entry, item) itemsets.append(tuples)

else:

if entry[0:-1] == item[0:-1]: tuples = entry+item[1:] itemsets.append(tuples)

i = i+1

if(len(itemsets) == 0): return None

return itemsets

def apriori(trans\_data,supp=3, con=0.5): freq = pd.DataFrame()

df = count\_item(trans\_data) while(len(df) != 0):

df = prune(df, supp)

if len(df) > 1 or (len(df) == 1 and int(df.supp\_count >= supp)): freq = df

itemsets = join(df.item\_sets) if(itemsets is None):

return freq

df = count\_itemset(trans\_data, itemsets) return df req\_item\_sets = apriori(trans\_df, 4) freq\_item\_sets

def calculate\_conf(value1, value2):

return round(int(value1)/int(value2) \* 100, 2) def strong\_rules(freq\_item\_sets, threshold):

confidences = {}

for row in freq\_item\_sets.item\_sets:

for i in range(len(row)): for j in range(len(row)):

if i != j:

tuples = (row[i], row[j])

conf = calculate\_conf(freq\_item\_sets[freq\_item\_sets.item\_sets == row].supp\_count, count\_item(trans\_df)[count\_item(trans\_df).item\_sets == row[i]].supp\_count)

confidences[tuples] = conf conf\_df = pd.DataFrame() conf\_df['item\_set'] = confidences.keys()

conf\_df['confidence'] = confidences.values() return conf\_df[conf\_df.confidence >= threshold]

strong\_rules(freq\_item\_sets, 50.0)

from functools import reduce

import operator

def interesting\_rules(freq\_item\_sets): lifts = {}

prob\_of\_items = []

for row in freq\_item\_sets.item\_sets: num\_of\_items = len(row)

prob\_all = freq\_item\_sets[freq\_item\_sets.item\_sets == row].supp\_count / 18 for i in range(num\_of\_items):

prob\_of\_items.append(count\_item(trans\_df)[count\_item(trans\_df).item\_sets

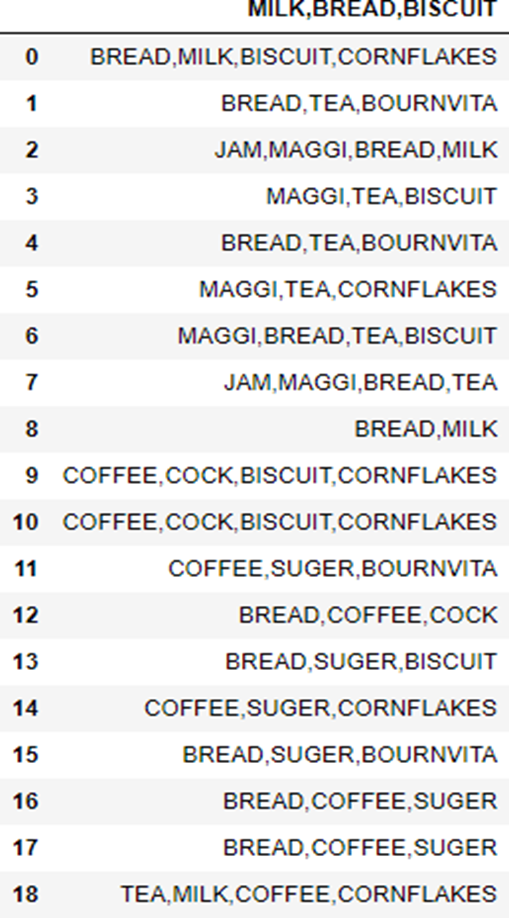
== row[i]].supp\_count / 18)

lifts[row] = round(float(prob\_all / reduce(operator.mul, (np.array(prob\_of\_items)), 1)), 2)

prob\_of\_items = [] lifts\_df = pd.DataFrame() lifts\_df['Rules'] = lifts.keys() lifts\_df['lift'] = lifts.values() return lifts\_df

int\_rules = interesting\_rules(freq\_item\_sets) int\_rules

**OUTPUT:**

****

Text

Description automatically generated

Table

Description automatically generated with low confidence

Text

Description automatically generated

**Part B:**

**1. Program FP tree using inbuilt functions.**

**2. Print the frequent patterns generated. Code:**

import pandas as pd

import numpy as np

import pyfpgrowth transactions = [

['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']

]

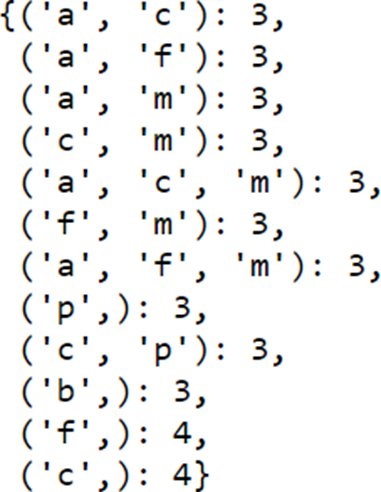
FP = pyfpgrowth.find\_frequent\_patterns( transactions = transactions , support\_threshold=3)

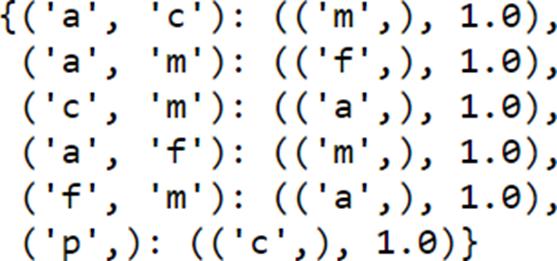
print(FP)

FP\_Rules = pyfpgrowth.generate\_association\_rules( patterns = FrequentPatterns, confidence\_threshold=0.8)

print(FP\_Rules)

**OUTPUT:**

****

****

**CONCLUSION:**

In this experiment, I implemented Association Rule Mining using Apriori Algorithm and FPTree. Apriori Algorithm is used for finding frequent pattern in a dataset for boolean association rule using Breadth-First search and a Hash tree structure. It reduces the size of the itemsets in the database significantly, providing a good performance. On the other hand, FP Tree needs to scan the database only twice as compared to Apriori which scans the transactions for each iteration. Also, the pairing of items is not done which makes it faster than FP Tree. Finally, the FP Tree is an improvisation of Apriori Algorithm used for association rule mining.