**AIM: Implementation of Associate Rule Mining algorithm (Apriori) using Python**

# THEORY:

Apriori Algorithm is a basic method used in data analysis to find groups of items that often appear together in large sets of data. It helps to discover useful patterns or rules about how items are related which is particularly valuable in market basket analysis.

### Steps For Apriori:

1. Set minimum support and confidence values.
2. Find frequent 1-itemsets (L1) – items meeting minimum support.
3. Generate candidate k-itemsets (Ck) from frequent (k−1)-itemsets.
4. Prune candidates that have infrequent subsets.
5. Count support of remaining candidates.
6. Keep frequent itemsets (Lk) with support ≥ minimum support.
7. Repeat steps 3–6 until no more itemsets can be formed.
8. Generate association rules from frequent itemsets using confidence and lift.

**Example:**

|  |  |
| --- | --- |
| **TID** | **Item Bought** |
| **T1** | **Milk, Bread, Butter** |
| **T2** | **Milk, Bread** |
| **T3** | **Bread, Butter** |
| **T4** | **Milk, Butter** |

## Step 1: Minimum Support = 50%

Total transactions = 4 → So, support ≥ 2 transactions.

## Step 2: Frequent 1-itemsets

* + Milk = 3
  + Bread = 3
  + Butter = 3

All ≥ 2 → keep all.

## Step 3: Generate 2-itemsets

{Milk, Bread}, {Milk, Butter}, {Bread, Butter} Support counts:

* + {Milk, Bread} = 2
  + {Milk, Butter} = 2
  + {Bread, Butter} = 2 All frequent.

## Step 4: Generate 3-itemset

{Milk, Bread, Butter} = 1 (less than 2 → remove)

## Step 5: Generate Rules (Confidence)

From {Milk, Bread} →

* + Milk ➺ Bread = 2/3 = 66.7%
  + Bread ➺ Milk = 2/3 = 66.7%

Both meet confidence (if min conf ≤ 60%).

### Final Rule Example:

***If Milk, then Bread***

**(Support = 50%, Confidence = 66.7%)**

## Key Terms in Apriori Algorithm

### Itemset

→ A collection (set) of one or more items.

*Example:* {Milk, Bread}, {Soap, Shampoo}

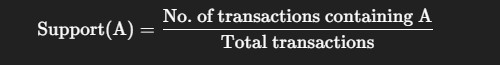
### Frequent Itemset

→ An itemset that appears frequently in the dataset — i.e., its support ≥ minimum support threshold.

*Example:* {Milk, Bread} appears in 40% of transactions (if min support = 30%, it’s frequent).

### Support

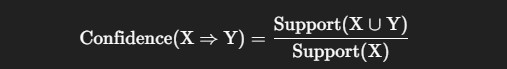
→ The proportion (or percentage) of transactions that contain an itemset.



*Example:* If {Milk, Bread} occurs in 20 out of 100 transactions → Support = 0.2 (20%).

### Confidence

→ Measures how often items in Y appear in transactions that contain X (for rule X

➺ Y).

*Example:* If {Milk, Bread} ➺ {Butter} has confidence 80%, it means 80% of people who bought Milk & Bread also bought Butter.

### Lift

→ Measures how much more likely Y is to occur with X than alone.



# CODE:

from itertools import combinations, chain

import pandas as pd

min\_support = 0.1

min\_confidence = 0.9

df = pd.read\_csv("model\_train\_expanded.csv")

categorical\_cols = ["neighbourhood\_group", "room\_type", "availability", "price\_category"]

df = df[categorical\_cols]

transactions = [set(row.astype(str)) for \_, row in df.iterrows()]

def get\_frequent\_itemsets(transactions, min\_support):

n = len(transactions)

items = sorted(set(chain.from\_iterable(transactions)))

freq\_itemsets = []

support\_data = {}

c1 = [{item} for item in items]

l1 = []

for candidate in c1:

support = sum(1 for t in transactions if candidate.issubset(t)) / n

if support >= min\_support:

l1.append(candidate)

support\_data[frozenset(candidate)] = support

freq\_itemsets.append(l1)

k = 2

while freq\_itemsets[k - 2]:

candidates = apriori\_gen(freq\_itemsets[k - 2], k)

lk = []

for candidate in candidates:

support = sum(1 for t in transactions if candidate.issubset(t)) / n

if support >= min\_support:

lk.append(candidate)

support\_data[frozenset(candidate)] = support

freq\_itemsets.append(lk)

k += 1

all\_freq\_itemsets = [item for sublist in freq\_itemsets for item in sublist]

return all\_freq\_itemsets, support\_data

def apriori\_gen(prev\_freq\_itemsets, k):

candidates = []

len\_prev = len(prev\_freq\_itemsets)

for i in range(len\_prev):

for j in range(i + 1, len\_prev):

l1 = sorted(prev\_freq\_itemsets[i])

l2 = sorted(prev\_freq\_itemsets[j])

if l1[:k-2] == l2[:k-2]:

candidate = prev\_freq\_itemsets[i] | prev\_freq\_itemsets[j]

if len(candidate) == k and candidate not in candidates:

candidates.append(candidate)

return candidates

def generate\_association\_rules(freq\_itemsets, support\_data, min\_confidence):

rules = []

for itemset in freq\_itemsets:

if len(itemset) < 2:

continue

for i in range(1, len(itemset)):

for antecedent in combinations(itemset, i):

antecedent = set(antecedent)

consequent = set(itemset) - antecedent

confidence = support\_data[frozenset(itemset)] / support\_data[frozenset(antecedent)]

if confidence >= min\_confidence:

lift = confidence / support\_data[frozenset(consequent)]

rules.append({

'antecedent': antecedent,

'consequent': consequent,

'support': support\_data[frozenset(itemset)],

'confidence': confidence,

'lift': lift

})

return rules

freq\_itemsets, support\_data = get\_frequent\_itemsets(transactions, min\_support)

print("Frequent Itemsets:")

for itemset in freq\_itemsets:

print(f"{set(itemset)} - Support: {support\_data[frozenset(itemset)]:.2f}")

rules = generate\_association\_rules(freq\_itemsets, support\_data, min\_confidence)

print("\nAssociation Rules:")

for idx, rule in enumerate(rules, 1):

print(f"Rule {idx}: {rule['antecedent']} => {rule['consequent']} "

f"(Support: {rule['support']:.2f}, Confidence: {rule['confidence']:.2f}, Lift: {rule['lift']:.2f})")

**OUTPUT**

