Assignment No. 6

Problem Statement: Design and implement the **Apriori Algorithm** to find frequent itemsets and generate association rules from a given transactional dataset.

Objective:

1. To understand the concept of frequent itemset mining and association rule learning.
2. To implement the Apriori algorithm using an appropriate data mining library.
3. To analyze the output of the algorithm and interpret association rules (support, confidence, lift).
4. To identify the most relevant patterns within a dataset based on minimum support and confidence thresholds.

Prerequisite :

1. **Basic Programming Concepts** – Python, loops, data structures.
2. **Set Theory & Probability** – Understanding of sets, unions, intersections, probability
3. **Data Mining Concepts** – Understanding of support, confidence, lift, itemsets.
4. **Libraries** – Basic understanding of pandas, mlxtend, or scikit-learn.

Theory :

The Apriori algorithm is a fundamental and widely used technique in the field of data mining, particularly for the task of **association rule learning**. It is designed to extract meaningful patterns from large transactional datasets by identifying frequent combinations of items (known as **frequent itemsets**) and using these combinations to infer **association rules**. This algorithm is most commonly associated with **market basket analysis**, where the objective is to understand customer purchasing behavior by analyzing the items bought together.

Apriori is based on the **anti-monotonicity property** of support, also called the **Apriori Principle**, which states:

> *"If an itemset is frequent, then all of its subsets must also be frequent."*

This property allows Apriori to efficiently prune the search space and avoid evaluating itemsets that cannot possibly be frequent, significantly reducing computational overhead in large databases.

Key Terminologies

1. **Itemset**
   An itemset refers to a collection of one or more items. These items are typically part of transactions. For example, in a supermarket, a transaction might consist of items like milk, bread, and butter. A 1-itemset contains one item (e.g., {milk}), a 2-itemset contains two items (e.g., {milk, bread}), and so on.

2. **Support**
   Support is a measure of how frequently an itemset appears in the dataset. It reflects the proportion of transactions that include a particular itemset. Mathematically, it is defined as:

$$\text{Support}(X) = \frac{|\{T \in D : X \subseteq T\}|}{|D|}$$

Where:

1. XXX is an itemset,
2. TTT is a transaction,
3. DDD is the database of all transactions.

A high support indicates that the itemset is common and may be useful in discovering stable patterns.

3. **Confidence**
   Confidence is a measure of the reliability of an association rule. It indicates the probability that a transaction containing itemset X also contains itemset Y. In rule notation X→Y confidence is given by:

$$\text{Confidence}(X \to Y) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}$$

4. **Lift**
   Lift measures the strength of a rule compared to the random co-occurrence of the antecedent and consequent. It quantifies how much more likely items in Y are purchased when X is purchased, compared to when they are purchased independently. It is defined as:

$$\text{Lift}(X \to Y) = \frac{\text{Confidence}(X \to Y)}{\text{Support}(Y)}$$

- **If lift = 1: X and Y are independent.**
- **If lift > 1: X and Y are positively correlated.**
- **If lift < 1: X and Y are negatively correlated.**

5. **Frequent Itemset**
   A frequent itemset is any group of items that appears together in at least a specified minimum number of transactions (as determined by the support threshold).

6. **Association Rule**
   An association rule is an implication of the form X→Y, where X and Y are disjoint itemsets. The rule suggests that if X occurs in a transaction, then Y is likely to also occur.

Working of the Apriori Algorithm

1. Finding Frequent Itemsets

This step aims to discover all item combinations that occur in the dataset with a frequency greater than or equal to the user-defined minimum support threshold.

1. **Initialization (k = 1):**
   Identify all frequent 1-itemsets by counting their individual occurrences in the dataset.
2. **Iteration (k ≥ 2):**
   For each level kkk, generate candidate kkk-itemsets by joining frequent (k−1) -itemsets with each other.
3. **Pruning:**
   Remove those candidate itemsets where any (k−1) -subset is not frequent. This is based on the Apriori property.
4. **Support Counting:**
   Count the support of each candidate k-itemset and remove those below the threshold.
5. **Repeat**
   Continue the process until no new frequent itemsets are found.

2. Generating Association Rules

Once frequent itemsets are identified, the next step is to generate rules from them.

1. For each frequent itemset L, generate all non-empty subsets.
2. For each subset S of L, construct the rule S→(L−S)
3. Compute confidence and lift for each rule.
4. Retain only those rules that meet the minimum confidence and lift thresholds.

For example, from frequent itemset {A,B,C} we can generate:

1. A→B,C
2. B→A,C

3. C→A,B
4. and so on.

Optimizations

Several optimizations have been proposed to improve the efficiency of Apriori:

1. **Hash-Based Candidate Pruning:** Uses hash tables to reduce the number of candidate itemsets generated in the early stages.
2. **Transaction Reduction:** Removes transactions that do not contain any frequent itemsets, reducing the size of the dataset.
3. **Partitioning:** Divides the database into partitions and mines frequent itemsets locally before combining them globally.
4. **Sampling:** Selects a representative sample of the dataset to approximate frequent itemsets.

Applications

Apriori and its variants are used across a wide range of applications:

1. **Retail and Market Basket Analysis** – Identifying products that are frequently bought together.
2. **Healthcare** – Detecting co-occurrence of symptoms and treatments.
3. **E-commerce Recommendation Engines** – Suggesting products based on association rules.
4. **Fraud Detection** – Finding unusual combinations of transaction patterns.
5. **Web Usage Mining** – Analyzing user navigation patterns on websites.

Advantages

1. Conceptually simple and easy to implement.
2. Effectively uses the Apriori property to reduce the number of candidate itemsets.
3. Generates interpretable rules useful for strategic decisions.

Limitations

1. Generates a vast number of candidate itemsets when the dataset has a large number of items, leading to high memory and computational costs.
2. Requires multiple passes over the entire dataset, which is time-consuming for large databases.
3. Poor performance with low support thresholds.
4. Does not handle numeric or continuous data natively; requires discretization.

Code & Output

[1]:
```
import numpy as np
import pandas as pd
```

[15]:
```
df= pd.read_csv("C:/Users/dnyan/ML Assignments/Dataset/store_data.csv")
```

[16]:
```
print(df.head())
```

```
            shrimp    almonds     avocado    vegetables mix green grapes  \
0          burgers  meatballs        eggs               NaN          NaN
1          chutney        NaN         NaN               NaN          NaN
2           turkey    avocado         NaN               NaN          NaN
3    mineral water       milk  energy bar  whole wheat rice    green tea
4   low fat yogurt        NaN         NaN               NaN          NaN

  whole weat flour yams cottage cheese energy drink tomato juice  \
0              NaN  NaN            NaN          NaN          NaN
1              NaN  NaN            NaN          NaN          NaN
2              NaN  NaN            NaN          NaN          NaN
3              NaN  NaN            NaN          NaN          NaN
4              NaN  NaN            NaN          NaN          NaN

  low fat yogurt green tea honey salad mineral water salmon antioxydant juice  \
0            NaN       NaN   NaN   NaN           NaN    NaN               NaN
1            NaN       NaN   NaN   NaN           NaN    NaN               NaN
2            NaN       NaN   NaN   NaN           NaN    NaN               NaN
3            NaN       NaN   NaN   NaN           NaN    NaN               NaN
4            NaN       NaN   NaN   NaN           NaN    NaN               NaN
```

[17]:
```
# shape of the data
df.shape
```

[17]:
```
(7500, 20)
```

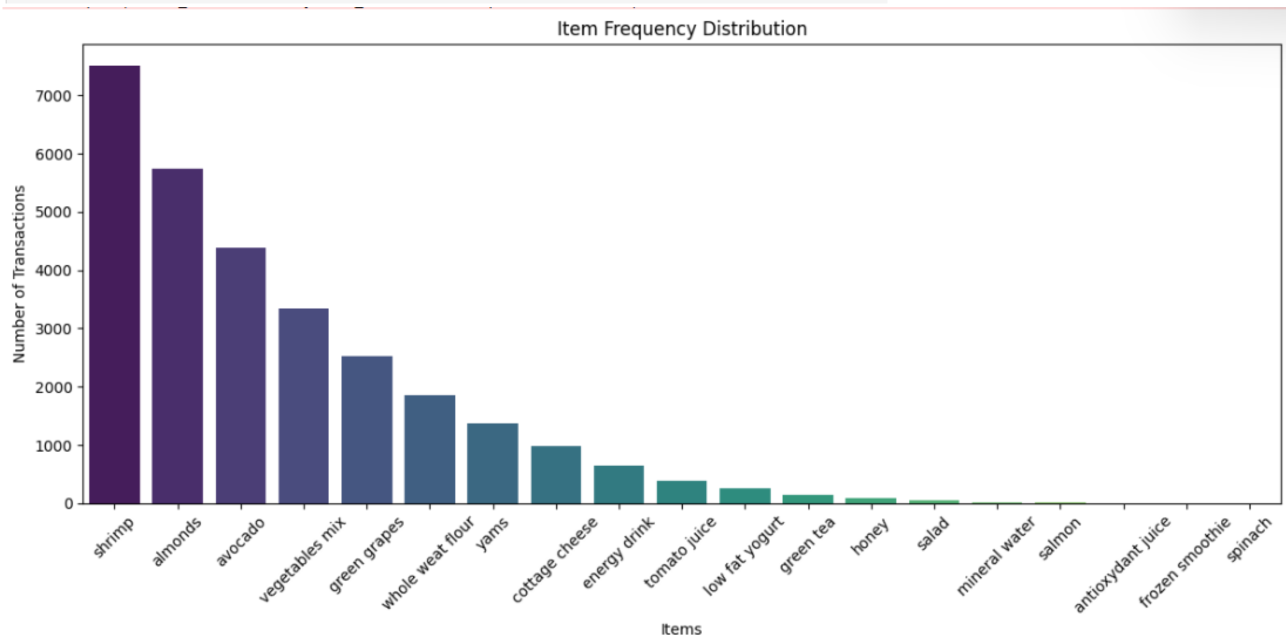
[18]:
```
#data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7500 entries, 0 to 7499
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   shrimp             7500 non-null   object
 1   almonds            5746 non-null   object
 2   avocado            4388 non-null   object
 3   vegetables mix     3344 non-null   object
 4   green grapes       2528 non-null   object
 5   whole weat flour   1863 non-null   object
 6   yams               1368 non-null   object
 7   cottage cheese     980 non-null    object
 8   energy drink       653 non-null    object
 9   tomato juice       394 non-null    object
 10  low fat yogurt     255 non-null    object
 11  green tea          153 non-null    object
 12  honey              86 non-null     object
 13  salad              46 non-null     object
 14  mineral water      24 non-null     object
 15  salmon             7 non-null      object
 16  antioxydant juice  3 non-null      object
 17  frozen smoothie    3 non-null      object
 18  spinach            2 non-null      object
 19  olive oil          0 non-null      float64
dtypes: float64(1), object(19)
memory usage: 1.1+ MB
```

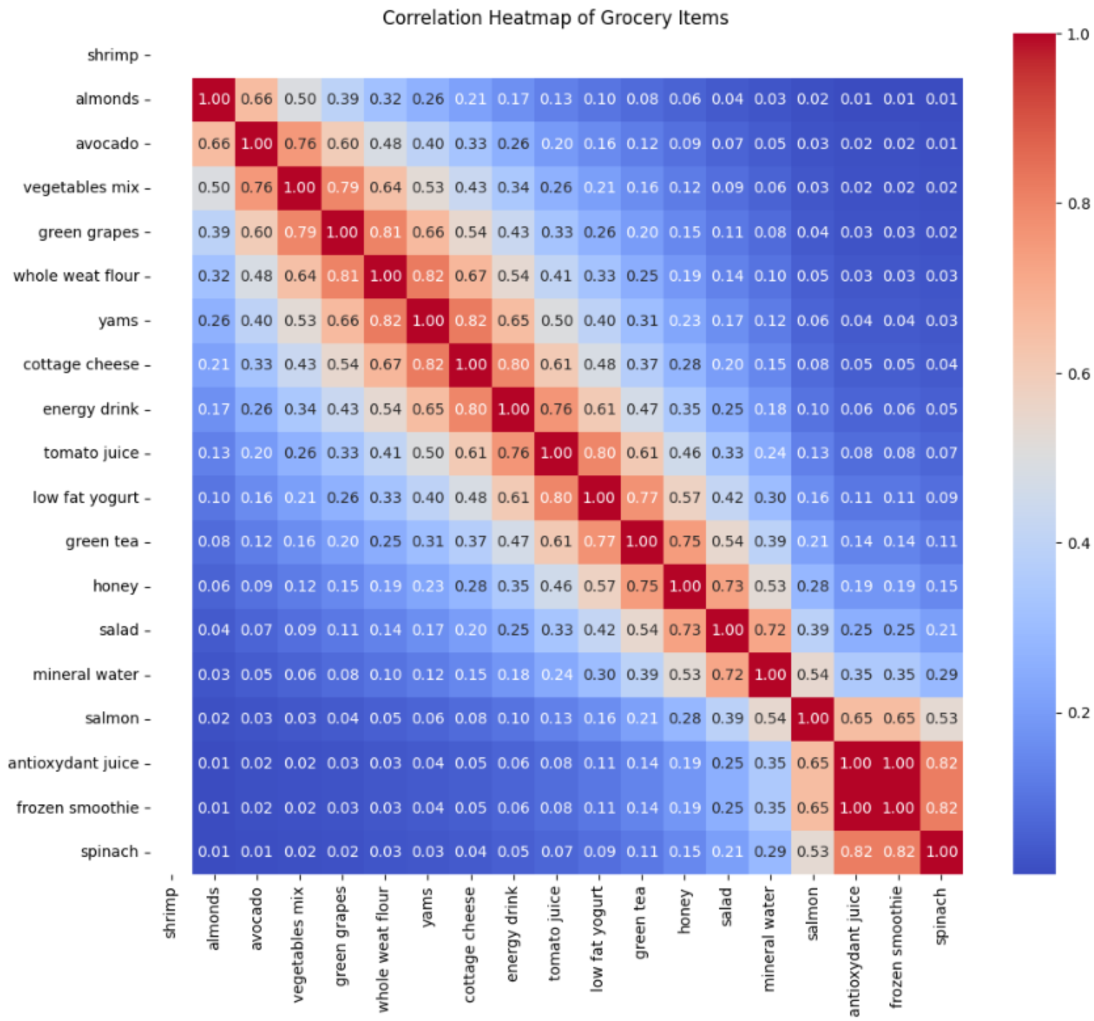
```python
df = df.drop(columns=['olive oil'])

# Convert each column into binary: 1 if item exists (non-null), 0 otherwise.
# We use .notnull() which returns True/False, then convert Boolean to integer (1/0)
df_binary = df.notnull().astype(int)

# Quick check of the preprocessed binary data
print("\nBinary Data Sample:")
print(df_binary.head())
```

```
Binary Data Sample:
   shrimp  almonds  avocado  vegetables mix  green grapes  whole weat flour  \
0       1        1        1               0             0                 0
1       1        0        0               0             0                 0
2       1        1        0               0             0                 0
3       1        1        1               1             1                 0
4       1        0        0               0             0                 0

   yams  cottage cheese  energy drink  tomato juice  low fat yogurt  \
0     0               0             0             0               0
1     0               0             0             0               0
2     0               0             0             0               0
3     0               0             0             0               0
4     0               0             0             0               0

   green tea  honey  salad  mineral water  salmon  antioxydant juice  \
0          0      0      0              0       0                  0
1          0      0      0              0       0                  0
2          0      0      0              0       0                  0
3          0      0      0              0       0                  0
4          0      0      0              0       0                  0
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Sum each column to count the number of transactions that include the item
item_counts = df_binary.sum().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(x=item_counts.index, y=item_counts.values, palette='viridis')
plt.title("Item Frequency Distribution")
plt.xlabel("Items")
plt.ylabel("Number of Transactions")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```python
plt.figure(figsize=(12, 10))
sns.heatmap(df_binary.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Grocery Items")
plt.show()
```

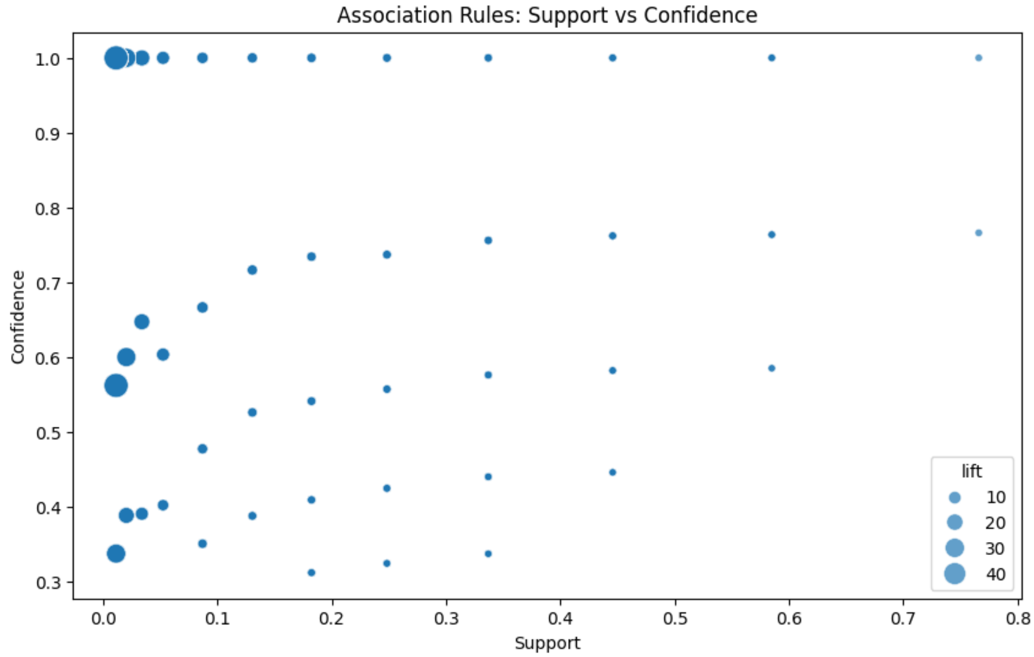



```python
from mlxtend.frequent_patterns import apriori

# Calculate frequent itemsets with a minimum support threshold.
frequent_itemsets = apriori(df_binary, min_support=0.01, use_colnames=True)

print("Frequent Itemsets:")
print(frequent_itemsets.sort_values(by="support", ascending=False).head())
```

```
C:\Users\dnyan\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxte
non-bool types result in worse computationalperformance and their support migh
  warnings.warn(
Frequent Itemsets:
     support            itemsets
0   1.000000            (shrimp)
13  0.766133   (shrimp, almonds)
1   0.766133           (almonds)
14  0.585067   (shrimp, avocado)
25  0.585067  (almonds, avocado)
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x="support", y="confidence", size="lift",
                data=rules, sizes=(20, 200), alpha=0.7)
plt.title("Association Rules: Support vs Confidence")
plt.xlabel("Support")
plt.ylabel("Confidence")
plt.show()
```

```
C:\Users\dnyan\AppData\Local\Programs\Python\Python312\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Creatir
n be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



Github : https://github.com/dnyaneshwardhere/ML

Conclusion:

The Apriori algorithm is a basic but powerful method for finding frequent itemsets and generating association rules in transactional data. It uses the principle that all subsets of a frequent itemset must also be frequent, which helps reduce unnecessary computations. While simple and easy to understand, it can be slow for large datasets. Despite this, it remains a valuable tool for learning and discovering useful patterns in fields like retail, healthcare, and web analytics.