NAME: Yash Chaudhary

NJIT UCID: ygc2

Email Address: ygc2@njit.edu

Date: October 11, 2024

Professor: Yasser Abdullah

Course: FA24-CS6364101 / Data Mining

MIDTERM PROJECT REPORT

IMPLEMENTATION OF BRUTE FORCE FOR ASSOCIATION RULE MINING AND COMPARISON WITH APRIORI AND FP-GROWTH ALGORITHMS.

TABLE OF CONTENT

ABSTRACT

This report reviews and analyses three methods of association rule mining: Brute Force, Apriori, and FP-Growth. In the course of this specified work, the aim was to discover the frequent itemsets and generate association rules, taking into account certain values of support and confidence levels defined by the user. Apart from the dynamic implementation discusses, it should provide for user introduction of custom datasets or just the use of preexisting supermarket datasets, for which the rules are generated by the three algorithms.

Meeting the challenge of rigorous comparisons of results produced by different algorithms have been significant. In the first place, a certain disparity in rule comparisons emerged due to the unsorted antecedents and consequents. Accordingly, a sorting of antecedents and consequents must be resorted to for fair comparison to enable outputs to be the same. The analysis concludes by comparing each algorithm on execution duration and manifests the fastest with FP-Growth running efficiently for large datasets and Brute Force comparatively slower.

The implementation provides a flexible, efficient, and user-friendly association rule miner. The results and insights drawn out of the study can be applied in market basket analysis and recommendation systems.

INTRODUCTION

Association rule mining is a crucial technique in data mining, used to discover relationships and patterns between items in large datasets. This process involves identifying frequently co-occurring items and generating rules that describe their dependencies. The insights derived from these rules are valuable for businesses, enabling them to understand customer behavior, improve product placement, and optimize marketing strategies.

This report focuses on implementing and comparing three popular association rule mining algorithms: Brute Force, Apriori, and FP-Growth. The Brute Force algorithm explores all possible itemsets to generate association rules, but it is computationally expensive for larger datasets. Apriori and FP-Growth, on the other hand, are optimized algorithms designed to handle large datasets efficiently by pruning the search space and reducing the number of itemsets considered.

The aim of this project is to provide a detailed comparison of the execution time, rule generation, and performance of these three algorithms across multiple datasets representing supermarket transactions. Additionally, we explore the flexibility of the implemented program, allowing users to input custom datasets, define minimum support and confidence thresholds, and analyze the generated rules. This report outlines the methodology, implementation details, and performance analysis of each algorithm.

CORE CONCEPTS AND PRINCIPLES

1. Association Rule Mining

Association rule mining is a technique used to identify interesting relationships between variables in large datasets. It's often applied in market basket analysis to find items frequently purchased together. An association rule is typically written as:

If X, then Y

Where X and Y represent sets of items. The goal is to find rules that satisfy specific thresholds for **support** and **confidence**, which indicate how often the rule occurs in the dataset and how reliable the rule is, respectively.

---

2. Support

Support is the fraction of transactions in the dataset that contain a particular itemset. It reflects how frequent an itemset is within the dataset. Support for an itemset **X** is calculated as:

Support(X) = (Number of transactions containing X) / (Total number of transactions)

A high support value indicates that an itemset appears frequently, while a low value suggests rarity.

---

3. Confidence

Confidence measures the reliability of an association rule. It is the ratio of the number of transactions that contain both **X** and **Y** to the number of transactions that contain **X**. Confidence is calculated as:

$$\textbf{Confidence}\ (X \rightarrow Y) = \textbf{Support}\ (X \cup Y) / \textbf{Support}(X)$$

A high confidence value suggests that the rule is likely to hold in the dataset.

4. Lift

Lift is a metric that measures how much more likely **X** and **Y** are to appear together than if they were independent. Lift is calculated as:

$$\textbf{Lift}\ (X \rightarrow Y) = \textbf{Support}\ (X \cup Y) / (\textbf{Support}(X) * \textbf{Support}(Y))$$

A lift value greater than 1 indicates a strong positive correlation between **X** and **Y**, while a value less than 1 indicates a negative correlation.

---

5. Algorithms in Association Rule Mining

Several algorithms are used in association rule mining to efficiently generate rules, particularly in large datasets:

a) **Brute Force Algorithm:** The Brute Force approach involves generating all possible itemsets and testing each one against the dataset. This method is inefficient for large datasets because the number of itemsets grows exponentially.

b) **Apriori Algorithm:** The Apriori algorithm improves on Brute Force by using the principle that if an itemset is infrequent, all its supersets will also be infrequent. This reduces the number of itemsets that need to be tested, making the algorithm more efficient.

c) **FP-Growth Algorithm:** The FP-Growth algorithm avoids generating candidate itemsets by using a data structure called an FP-tree (Frequent Pattern Tree). This structure compresses the dataset, enabling faster rule generation compared to Apriori, particularly for large datasets.

---

6. Performance Metrics

Key metrics to evaluate the performance of association rule mining algorithms include:

- **Execution Time:** The time taken to generate association rules.
- **Memory Usage:** The amount of memory used by the algorithm.
- **Rule Generation Efficiency:** The number of rules generated that meet the support and confidence thresholds.

---

7. Applications of Association Rule Mining

Association rule mining has diverse applications:

- **Market Basket Analysis:** Finding items often purchased together to optimize store layouts and marketing strategies.
- **Web Usage Mining:** Analyzing browsing behavior to improve user experience and recommend products.
- **Fraud Detection:** Identifying unusual transaction patterns in financial data to detect fraud.

PROJECT FLOW

1. Problem Definition and Objective

- **Objective**: To generate association rules from transaction datasets using three different algorithms: **Brute Force**, **Apriori**, and **FP-Growth**.

- The program allows users to choose one of five pre-defined datasets or a custom CSV file to analyze, but the custom dataset should have an attribute of **"Transaction Details"** containing the items of a transaction. The goal is to discover hidden patterns (rules) in the data that indicate relationships between items in customer transactions.

---

2. Dataset Selection

- The program includes five supermarket datasets: **Amazon, Walmart, Target, Aldi, and Costco.**

- The user also has the option to provide a custom dataset in CSV format for analysis.

- Each dataset contains transactional data where items bought together in one transaction are recorded.

---

3. Preprocessing the Data

- **Loading the Dataset**: The dataset is read from a CSV file using **pandas**. Each transaction is extracted and processed into a list of items for further analysis.

- **Transaction Encoding**: The **TransactionEncoder** from **mlxtend** package is used to convert the list of transactions into a one-hot encoded **DataFrame** format, where each column represents an item and each row represents a transaction.

4. User Inputs for Analysis

- **Dataset Choice:** The user is prompted to input for choosing a dataset or provide a custom dataset for analysis.

- **Support and Confidence Thresholds**: The user is prompted to input minimum support and confidence values (in percentages). These values guide the algorithm to generate only the most relevant association rules.

- **Validation**: Input values are validated to ensure they fall within the accepted range (1 – 7) for datasets and range (1-100%) for thresholds.

---

5. Association Rule Mining Algorithms

- **Brute Force Algorithm**: The brute-force method generates all possible itemsets and evaluates their support and confidence to generate rules. It uses for loops and generates all possible combinations to exhaustively search for rules.

- **Apriori Algorithm**: The Apriori algorithm is implemented using built-in package **mlxtend.frequent_patterns.apriori.** It generates frequent itemsets based on the user-defined minimum support, and then uses confidence threshold to produce association rules.

- **FP-Growth Algorithm**: The FP-Growth algorithm, implemented using built-in package **mlxtend.frequent_patterns.fpgrowth**, it also generates frequent itemsets and rules based on user-defined support and confidence levels. It is generally more efficient than Apriori for larger datasets.

---

6. Measuring Execution Time

- For each algorithm, the execution time is measured using Python's **time.perf_counter()** function. This provides insight into the efficiency of each method with more accuracy.

- A comparison of the time taken by each algorithm is presented to help evaluate performance.

---

7. Displaying Itemsets and Rules

- **Frequent Itemsets**: The program first displays all the frequent itemsets (sets of items that occur together in transactions) along with their support values.

- **Generated Rules**: The program then displays the association rules generated by each algorithm, including the rule’s support, confidence, and the antecedent and consequent itemsets.

- For Brute Force, Apriori, and FP-Growth, the rules are printed in a user-friendly format, showing the relationships between items.

---

8. Comparison of Algorithms

- The results and rules generated by the Brute Force algorithm are compared with those generated by Apriori and FP-Growth.

- The execution times of the algorithms are compared to analyze which method is more efficient under different dataset conditions.

RESULTS AND EVALUATIONS

1. Results Overview

In this section, we present the outcomes of running the association rule mining algorithms (brute-force, Apriori, and FP-Growth) on the selected datasets. Each algorithm's results include the generated rules, their corresponding support and confidence values, and execution time.

---

2. Brute-Force Algorithm Results

- **Generated Rules**: The brute-force algorithm produces a comprehensive list of all possible association rules derived from the transaction data.

- **Performance Metrics**: Rules, Support and confidence values are calculated accurately. But its least efficient as it generates and checks every possible combination of itemsets.

- **Error Handling**: Any issues encountered during execution are clearly indicated with a preceding asterisk (*).

---

3. Apriori Algorithm Results

- **Generated Rules**: The Apriori algorithm effectively identifies frequent itemsets, resulting in a manageable set of association rules based on the minimum support and confidence thresholds set by the user.

- **Performance Metrics**: Rules, Support and confidence values are calculated accurately. Comparatively its better than Brute Force and also efficient for small or medium size datasets but for large datasets, it is also inefficient by FP – Growth.

- **Error Handling**: Similar to the brute-force implementation, errors are marked with a preceding asterisk (*).

---

4. FP-Growth Algorithm Results

- **Generated Rules**: The FP-Growth algorithm provides a compact set of frequent patterns and association rules, leveraging its tree structure to minimize memory usage and enhance processing speed.

- **Performance Metrics**: Efficiency is great compared to both the brute-force and Apriori methods, demonstrating significant improvements in speed and resource utilization.

- **Error Handling**: Errors encountered are also indicated with a preceding asterisk (*).

---

5. Comparative Analysis

- **Execution Time Comparison**: FP – Growth is the quickest in all cases followed by the Apriori which is good in speed for small or medium datasets but Brute force is very time consuming as it checks for every possible combination of the itemsets.

- **Rule Generation Comparison**: All algorithms are mostly generating same rules, indicating that the implementation is correctly done.

---

6. Evaluation of Effectiveness

- The results indicate that the brute-force algorithm is exhaustive, it is less efficient in terms of execution time compared to the Apriori and FP-Growth algorithms.

- The Apriori algorithm, while faster than brute-force, still has limitations with large datasets due to its iterative nature.

- The FP-Growth algorithm consistently outperforms both in terms of execution speed and memory efficiency, making it the preferred choice for large-scale datasets.

DATASETS FOR ASSOCIATION RULE MINING

This section gives a summary of transaction datasets collected from various supermarkets such as Amazon, Walmart, Target, Costco, and Aldi. These datasets would be put into the implementation of association rule mining algorithms. A major focus of this project is studying customer-buying habits and finding out the set of commonly associated items in these stores.

• Selected Supermarkets

The following supermarkets have been selected for analysis:

- **Amazon**
- **Target**
- **Walmart**
- **Aldi**
- **Costco**

These stores were chosen due to their popularity and the diversity of items they offer, making them suitable for studying consumer purchasing patterns.

• Transaction Data Structure

Data consists of several tables; each represents different purchases against each transaction for every store. Each store's dataset contains:

1. **Items Table:** Here, list the items this store usually sells to this customer.

2. **Transaction Table**: This represents details of different unique transaction ids (TID), with items forming part of each transaction.

3. **Binary Table:** This table represents transactions in its column using the binary digits. **1** represents that an item forms part of a transaction and **0** does the opposite. It’s basically for the “Tidy Representation” of the datasets.

- ## Data Collection Methodology

The datasets for this report were created manually in order to ease the implementation of algorithms in mining association rules.

The process included:

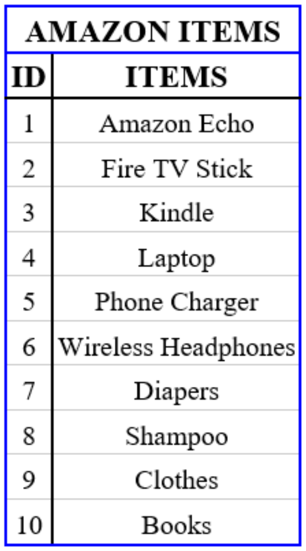
- **Store Selection:** Five supermarkets are selected: Amazon, Walmart, Target, Costco, and Aldi.
- **Product Identification:** In each store, ten commodities were identified that are most in demand and readily available.
- **Transaction generation:** For each store, twenty realistic sets of transactions were generated to best model how any normal customer could shop. They were based on my purchases with the stores and insight from my friends to make the data sets as realistic as possible.
- **Data Formatting:** Further, tabulated data were put into tables, while all the binary and detailed information was kept in comma-separated values format.
- The documentation was important in terms of methodological transparency. Hence, the represented data sets were reproducible. Help of an AI tool is also taken to make the datasets more deterministic and accurate.

- ## Use of Dataset in Code

  - The **item table** itself is not directly used in the code; instead, it's utilized to construct the **transaction table**. This transaction table, which includes **“TID”** and **“Transaction Details”** serves as the input for the **Brute Force Algorithm** to generate association rules.
  - The binary table, which is a **one-hot** encoded representation of the transaction data, performs a similar function to what **TransactionEncoder** does for built-in algorithms like **Apriori** and **FP-Growth**. This transformation from a regular dataset to a **one-hot** format enhances accuracy and improves the overall efficiency of the algorithms.

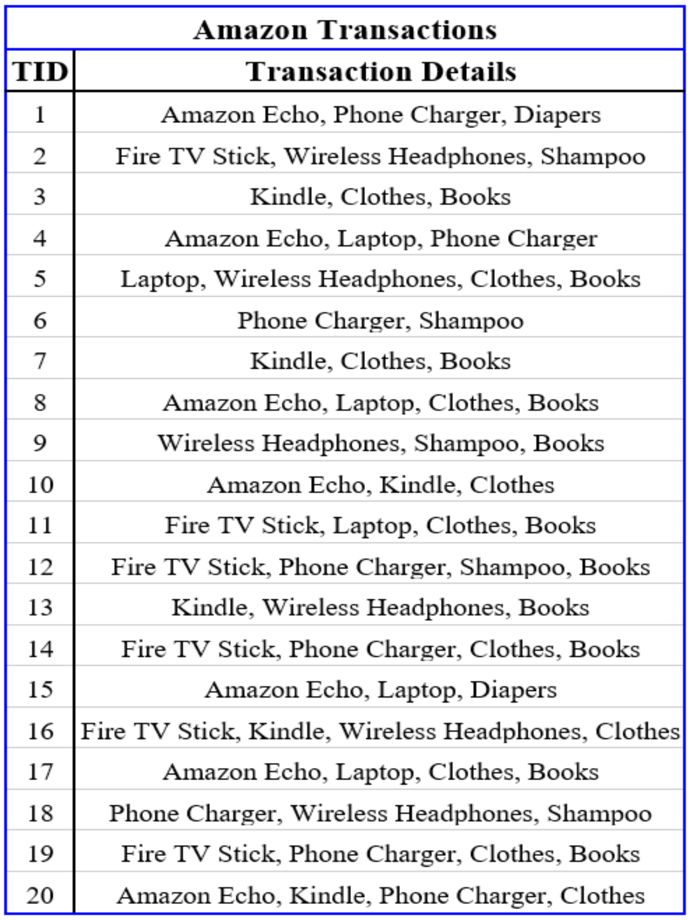
---

1) AMAZON Dataset



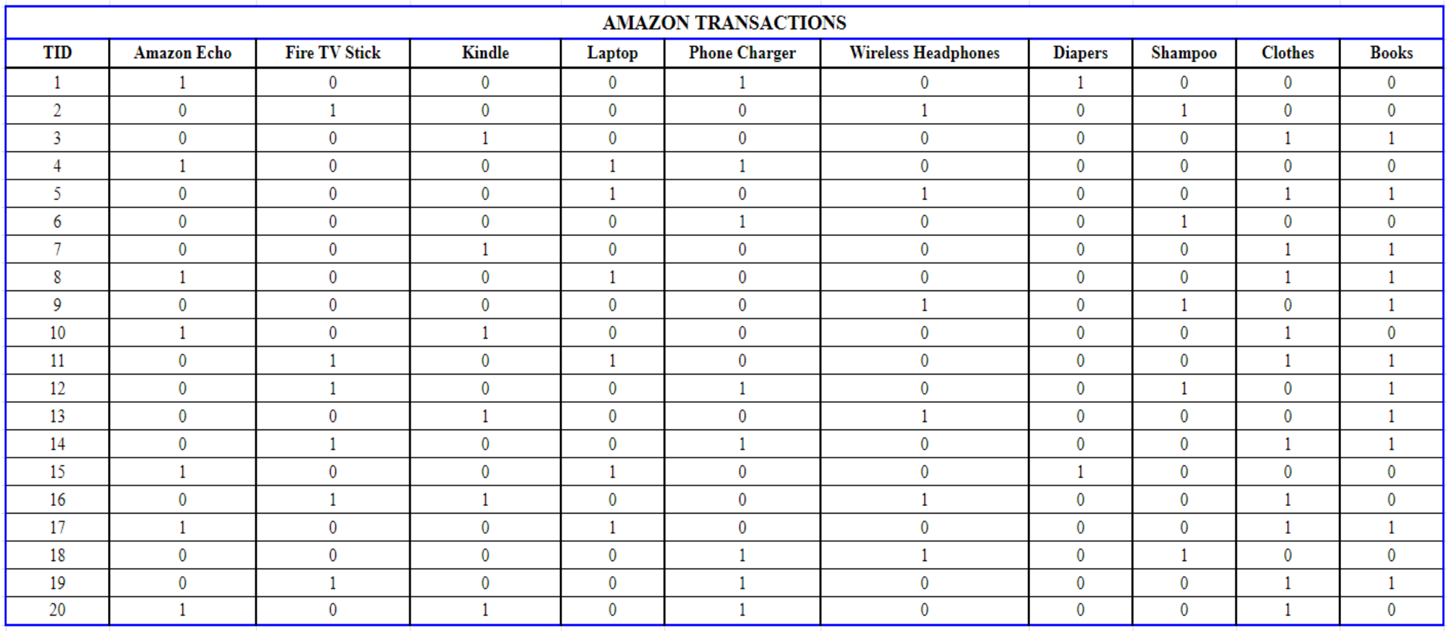
AMAZON ITEMS	
ID	**ITEMS**
1	Amazon Echo
2	Fire TV Stick
3	Kindle
4	Laptop
5	Phone Charger
6	Wireless Headphones
7	Diapers
8	Shampoo
9	Clothes
10	Books

Table: 1 (Amazon Item Table)



Amazon Transactions	
TID	**Transaction Details**
1	Amazon Echo, Phone Charger, Diapers
2	Fire TV Stick, Wireless Headphones, Shampoo
3	Kindle, Clothes, Books
4	Amazon Echo, Laptop, Phone Charger
5	Laptop, Wireless Headphones, Clothes, Books
6	Phone Charger, Shampoo
7	Kindle, Clothes, Books
8	Amazon Echo, Laptop, Clothes, Books
9	Wireless Headphones, Shampoo, Books
10	Amazon Echo, Kindle, Clothes
11	Fire TV Stick, Laptop, Clothes, Books
12	Fire TV Stick, Phone Charger, Shampoo, Books
13	Kindle, Wireless Headphones, Books
14	Fire TV Stick, Phone Charger, Clothes, Books
15	Amazon Echo, Laptop, Diapers
16	Fire TV Stick, Kindle, Wireless Headphones, Clothes
17	Amazon Echo, Laptop, Clothes, Books
18	Phone Charger, Wireless Headphones, Shampoo
19	Fire TV Stick, Phone Charger, Clothes, Books
20	Amazon Echo, Kindle, Phone Charger, Clothes

Table: 2 (Amazon Transaction Table)



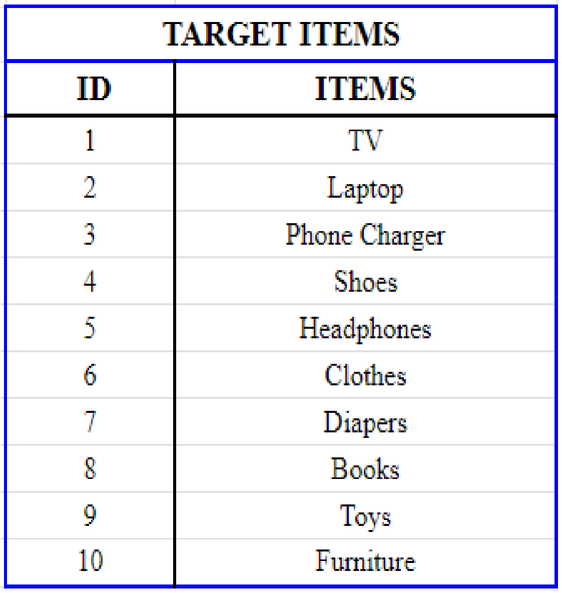
AMAZON TRANSACTIONS										
TID	**Amazon Echo**	**Fire TV Stick**	**Kindle**	**Laptop**	**Phone Charger**	**Wireless Headphones**	**Diapers**	**Shampoo**	**Clothes**	**Books**
1	1	0	0	0	1	0	1	0	0	0
2	0	1	0	0	0	1	0	1	0	0
3	0	0	1	0	0	0	0	0	1	1
4	1	0	0	1	1	0	0	0	0	0
5	0	0	0	1	0	1	0	0	1	1
6	0	0	0	0	1	0	0	1	0	0
7	0	0	1	0	0	0	0	0	1	1
8	1	0	0	1	0	0	0	0	1	1
9	0	0	0	0	0	1	0	1	0	1
10	1	0	1	0	0	0	0	0	1	0
11	0	1	0	1	0	0	0	0	1	1
12	0	1	0	0	1	0	0	1	0	1
13	0	0	1	0	0	1	0	0	0	1
14	0	1	0	0	1	0	0	0	1	1
15	1	0	0	1	0	0	1	0	0	0
16	0	1	1	0	0	1	0	0	1	0
17	1	0	0	1	0	0	0	0	1	1
18	0	0	0	0	1	1	0	1	0	0
19	0	1	0	0	1	0	0	0	1	1
20	1	0	1	0	1	0	0	0	1	0

Table: 3 (Amazon Binary Table)

Note: 0 indicates that the particular item is not present in the transaction and **1** indicates that the item is present. This is used for the "**Tidy Representation**" of the dataset.

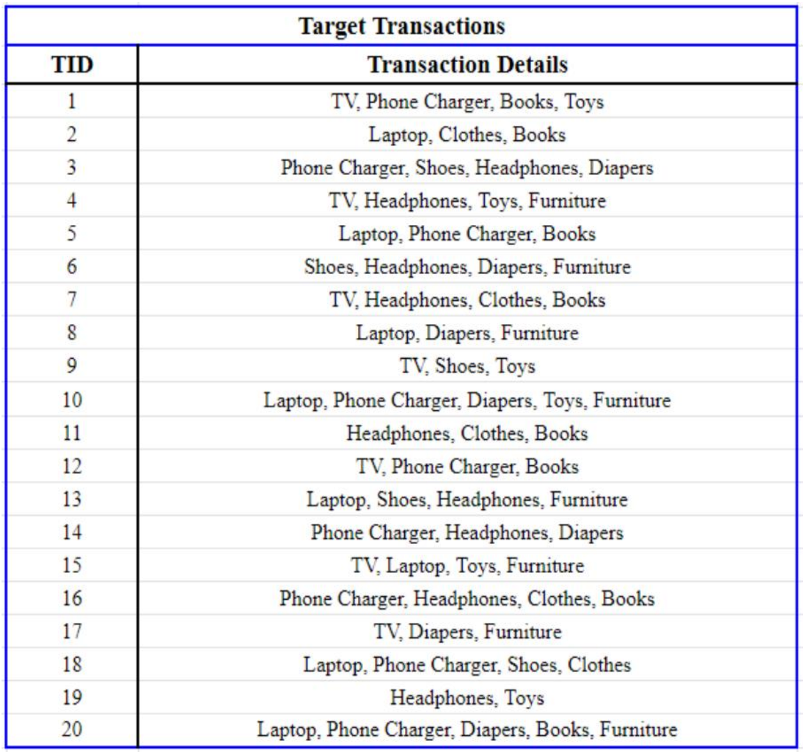
---

2) TARGET Dataset



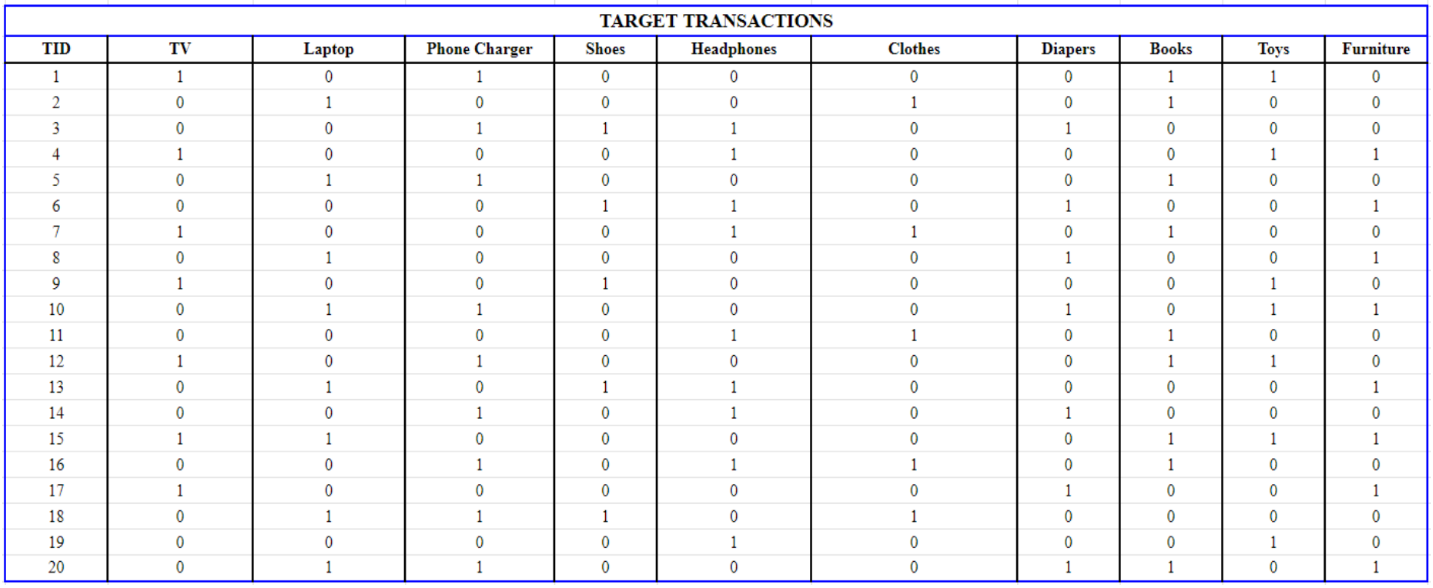
TARGET ITEMS	
ID	**ITEMS**
1	TV
2	Laptop
3	Phone Charger
4	Shoes
5	Headphones
6	Clothes
7	Diapers
8	Books
9	Toys
10	Furniture

Table: 4 (Target Item Table)



Target Transactions	
TID	**Transaction Details**
1	TV, Phone Charger, Books, Toys
2	Laptop, Clothes, Books
3	Phone Charger, Shoes, Headphones, Diapers
4	TV, Headphones, Toys, Furniture
5	Laptop, Phone Charger, Books
6	Shoes, Headphones, Diapers, Furniture
7	TV, Headphones, Clothes, Books
8	Laptop, Diapers, Furniture
9	TV, Shoes, Toys
10	Laptop, Phone Charger, Diapers, Toys, Furniture
11	Headphones, Clothes, Books
12	TV, Phone Charger, Books
13	Laptop, Shoes, Headphones, Furniture
14	Phone Charger, Headphones, Diapers
15	TV, Laptop, Toys, Furniture
16	Phone Charger, Headphones, Clothes, Books
17	TV, Diapers, Furniture
18	Laptop, Phone Charger, Shoes, Clothes
19	Headphones, Toys
20	Laptop, Phone Charger, Diapers, Books, Furniture

Table: 5 (Target Transaction Table)

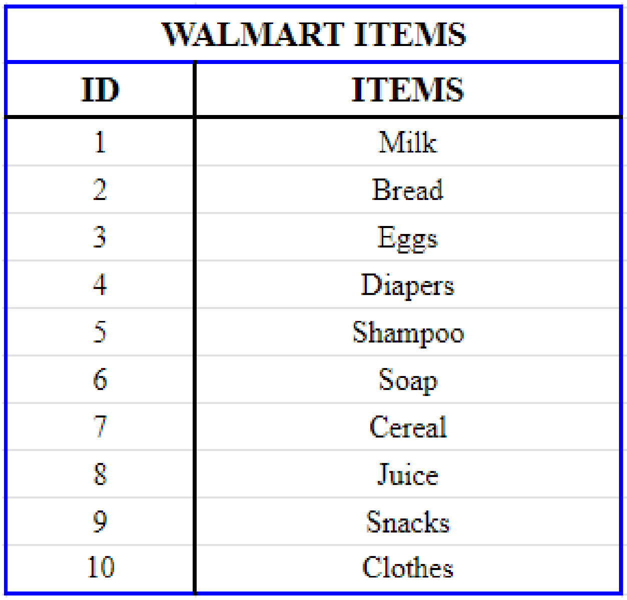


TARGET TRANSACTIONS										
TID	**TV**	**Laptop**	**Phone Charger**	**Shoes**	**Headphones**	**Clothes**	**Diapers**	**Books**	**Toys**	**Furniture**
1	1	0	1	0	0	0	0	1	1	0
2	0	1	0	0	0	1	0	1	0	0
3	0	0	1	1	1	0	1	0	0	0
4	1	0	0	0	1	0	0	0	1	1
5	0	1	1	0	0	0	0	1	0	0
6	0	0	0	1	1	0	1	0	0	1
7	1	0	0	0	1	1	0	1	0	0
8	0	1	0	0	0	0	1	0	0	1
9	1	0	0	1	0	0	0	0	1	0
10	0	1	1	0	0	0	1	0	1	1
11	0	0	0	0	1	1	0	1	0	0
12	1	0	1	0	0	0	0	1	1	0
13	0	1	0	1	1	0	0	0	0	1
14	0	0	1	0	1	0	1	0	0	0
15	1	1	0	0	0	0	0	1	1	1
16	0	0	1	0	1	1	0	1	0	0
17	1	0	0	0	0	0	1	0	0	1
18	0	1	1	1	0	1	0	0	0	0
19	0	0	0	0	1	0	0	0	1	0
20	0	1	1	0	0	0	1	1	0	1

Table: 6 (Target Binary Table)

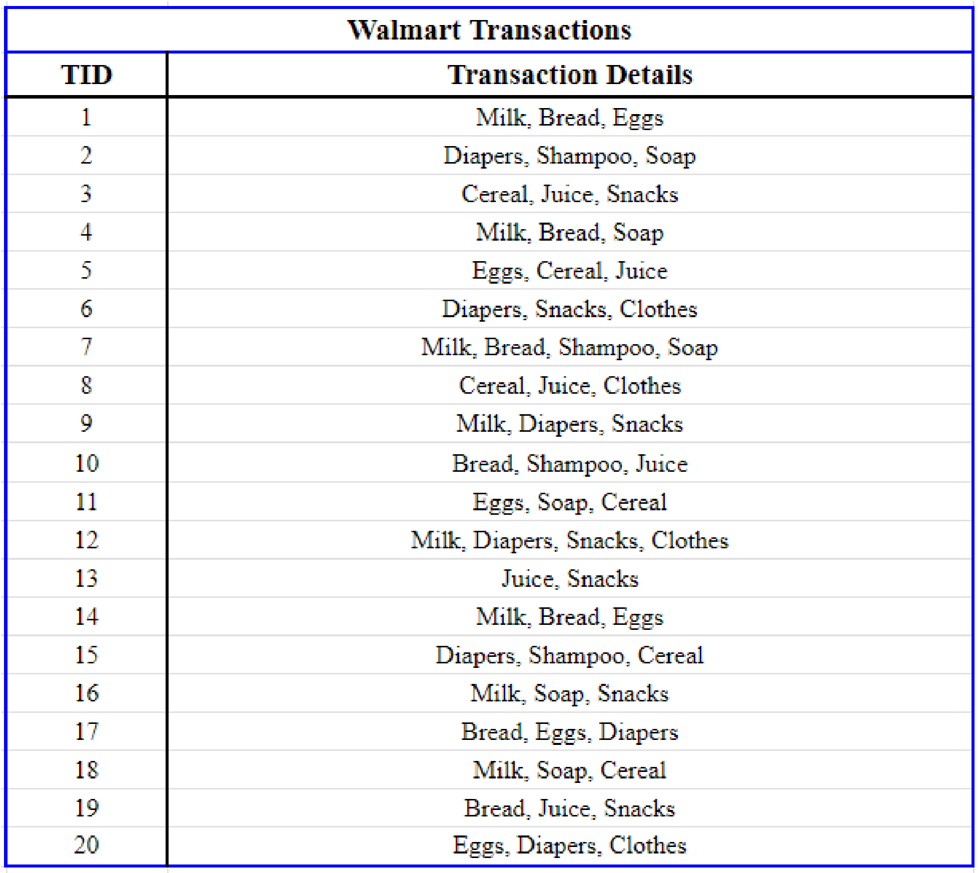
Note: 0 indicates that the particular item is not present in the transaction and **1** indicates that the item is present. This is used for the "**Tidy Representation**" of the dataset.

3) WALMART Dataset



WALMART ITEMS	
ID	ITEMS
1	Milk
2	Bread
3	Eggs
4	Diapers
5	Shampoo
6	Soap
7	Cereal
8	Juice
9	Snacks
10	Clothes

Table: 7 (Walmart Item Table)



Walmart Transactions	
TID	Transaction Details
1	Milk, Bread, Eggs
2	Diapers, Shampoo, Soap
3	Cereal, Juice, Snacks
4	Milk, Bread, Soap
5	Eggs, Cereal, Juice
6	Diapers, Snacks, Clothes
7	Milk, Bread, Shampoo, Soap
8	Cereal, Juice, Clothes
9	Milk, Diapers, Snacks
10	Bread, Shampoo, Juice
11	Eggs, Soap, Cereal
12	Milk, Diapers, Snacks, Clothes
13	Juice, Snacks
14	Milk, Bread, Eggs
15	Diapers, Shampoo, Cereal
16	Milk, Soap, Snacks
17	Bread, Eggs, Diapers
18	Milk, Soap, Cereal
19	Bread, Juice, Snacks
20	Eggs, Diapers, Clothes

Table: 8 (Walmart Transaction Table)

WALMART TRANSACTIONS										
TID	Milk	Bread	Eggs	Diapers	Shampoo	Soap	Cereal	Juice	Snacks	Clothes
1	1	1	1	0	0	0	0	0	0	0
2	0	0	0	1	1	1	0	0	0	0
3	0	0	0	0	0	0	1	1	1	0
4	1	1	0	0	0	1	0	0	0	0
5	0	0	1	0	0	0	1	1	0	0
6	0	0	0	1	0	0	0	0	1	1
7	1	1	0	0	1	1	0	0	0	0
8	0	0	0	0	0	0	1	1	0	1
9	1	0	0	1	0	0	0	0	1	0
10	0	1	0	0	1	0	0	1	0	0
11	0	0	1	0	0	1	1	0	0	0
12	1	0	0	1	0	0	0	0	1	1
13	0	0	0	0	0	0	0	1	1	0
14	1	1	1	0	0	0	0	0	0	0
15	0	0	0	1	1	0	1	0	0	0
16	1	0	0	0	0	1	0	0	1	0
17	0	1	1	1	0	0	0	0	0	0
18	1	0	0	0	0	1	1	0	0	0
19	0	1	0	0	0	0	0	1	1	0
20	0	0	1	1	0	0	0	0	0	1

Table: 9 (Walmart Binary Table)

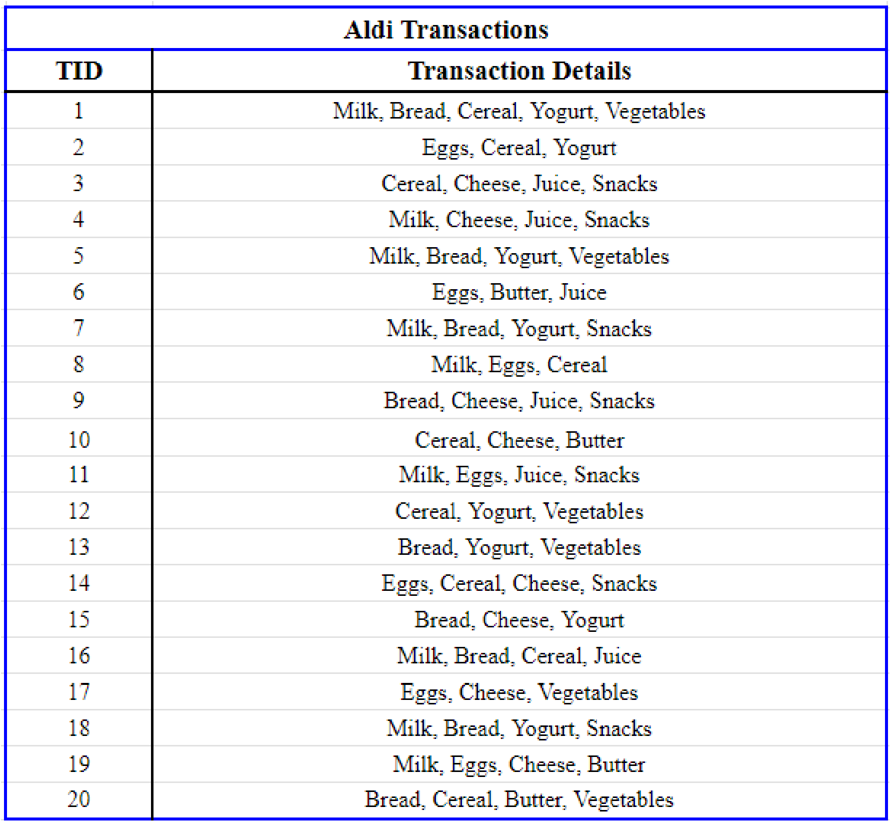
Note: 0 indicates that the particular item is not present in the transaction and **1** indicates that the item is present. This is used for the "**Tidy Representation**" of the dataset.

---

4) ALDI Dataset

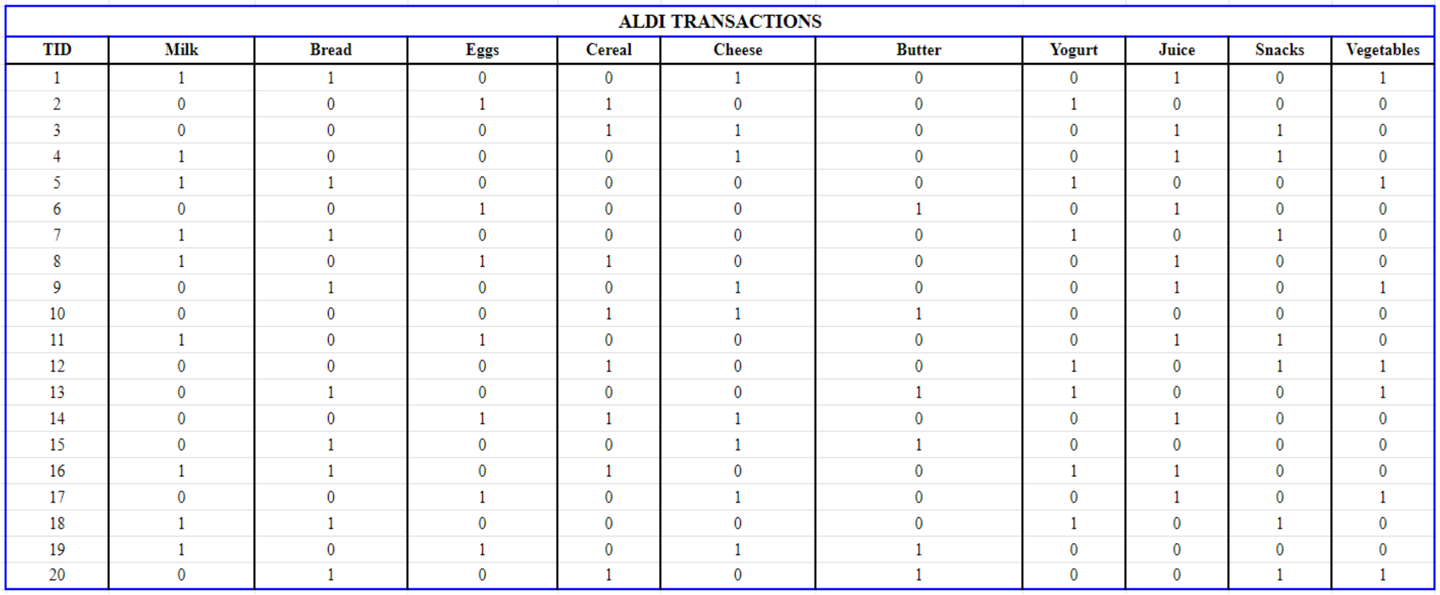
ALDI ITEMS	
ID	Items
1	Milk
2	Bread
3	Eggs
4	Cereal
5	Cheese
6	Butter
7	Yogurt
8	Juice
9	Snacks
10	Vegetables

Table: 10 (Aldi Item Table)



Aldi Transactions	
TID	**Transaction Details**
1	Milk, Bread, Cereal, Yogurt, Vegetables
2	Eggs, Cereal, Yogurt
3	Cereal, Cheese, Juice, Snacks
4	Milk, Cheese, Juice, Snacks
5	Milk, Bread, Yogurt, Vegetables
6	Eggs, Butter, Juice
7	Milk, Bread, Yogurt, Snacks
8	Milk, Eggs, Cereal
9	Bread, Cheese, Juice, Snacks
10	Cereal, Cheese, Butter
11	Milk, Eggs, Juice, Snacks
12	Cereal, Yogurt, Vegetables
13	Bread, Yogurt, Vegetables
14	Eggs, Cereal, Cheese, Snacks
15	Bread, Cheese, Yogurt
16	Milk, Bread, Cereal, Juice
17	Eggs, Cheese, Vegetables
18	Milk, Bread, Yogurt, Snacks
19	Milk, Eggs, Cheese, Butter
20	Bread, Cereal, Butter, Vegetables

Table: 11 (Aldi Transaction Table)

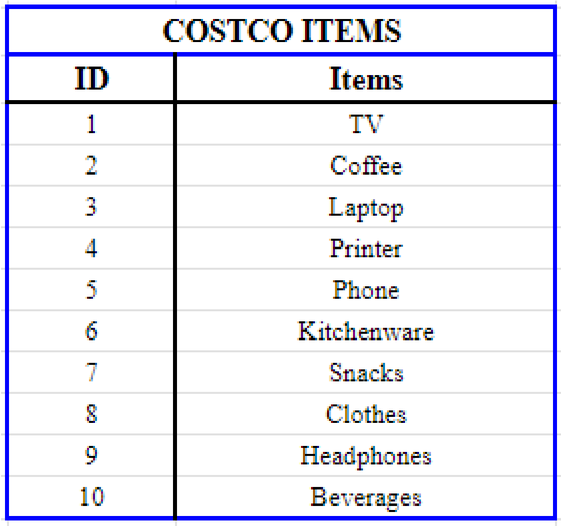


ALDI TRANSACTIONS										
TID	**Milk**	**Bread**	**Eggs**	**Cereal**	**Cheese**	**Butter**	**Yogurt**	**Juice**	**Snacks**	**Vegetables**
1	1	1	0	0	1	0	0	1	0	1
2	0	0	1	1	0	0	1	0	0	0
3	0	0	0	1	1	0	0	1	1	0
4	1	0	0	0	1	0	0	1	1	0
5	1	1	0	0	0	0	1	0	0	1
6	0	0	1	0	0	1	0	1	0	0
7	1	1	0	0	0	0	1	0	1	0
8	1	0	1	1	0	0	0	1	0	0
9	0	1	0	0	1	0	0	1	0	1
10	0	0	0	1	1	1	0	0	0	0
11	1	0	1	0	0	0	0	1	1	0
12	0	0	0	1	0	0	1	0	1	1
13	0	1	0	0	0	1	1	0	0	1
14	0	0	1	1	1	0	0	1	0	0
15	0	1	0	0	1	1	0	0	0	0
16	1	1	0	1	0	0	1	1	0	0
17	0	0	1	0	1	0	0	1	0	1
18	1	1	0	0	0	0	1	0	1	0
19	1	0	1	0	1	1	0	0	0	0
20	0	1	0	1	0	1	0	0	1	1

Table: 12 (Aldi Binary Table)

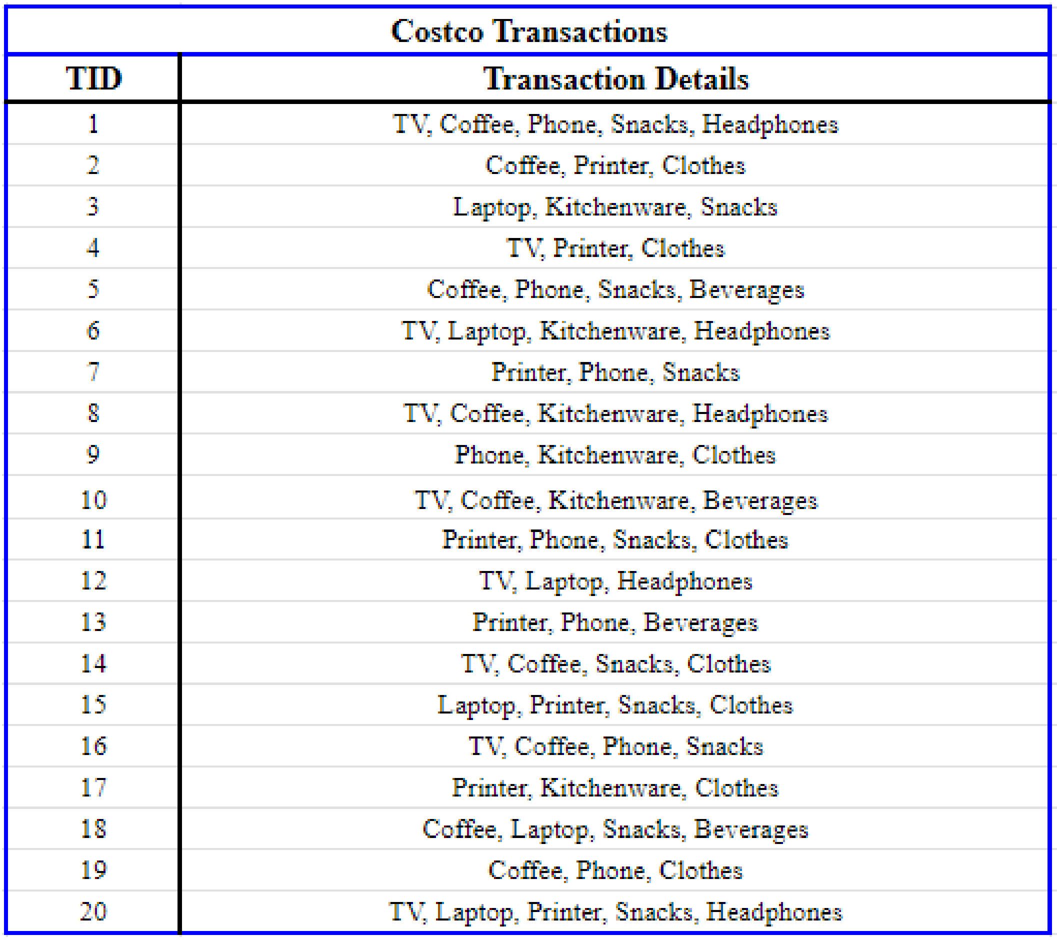
Note: 0 indicates that the particular item is not present in the transaction and **1** indicates that the item is present. This is used for the "**Tidy Representation**" of the dataset.

5) COSTCO Dataset



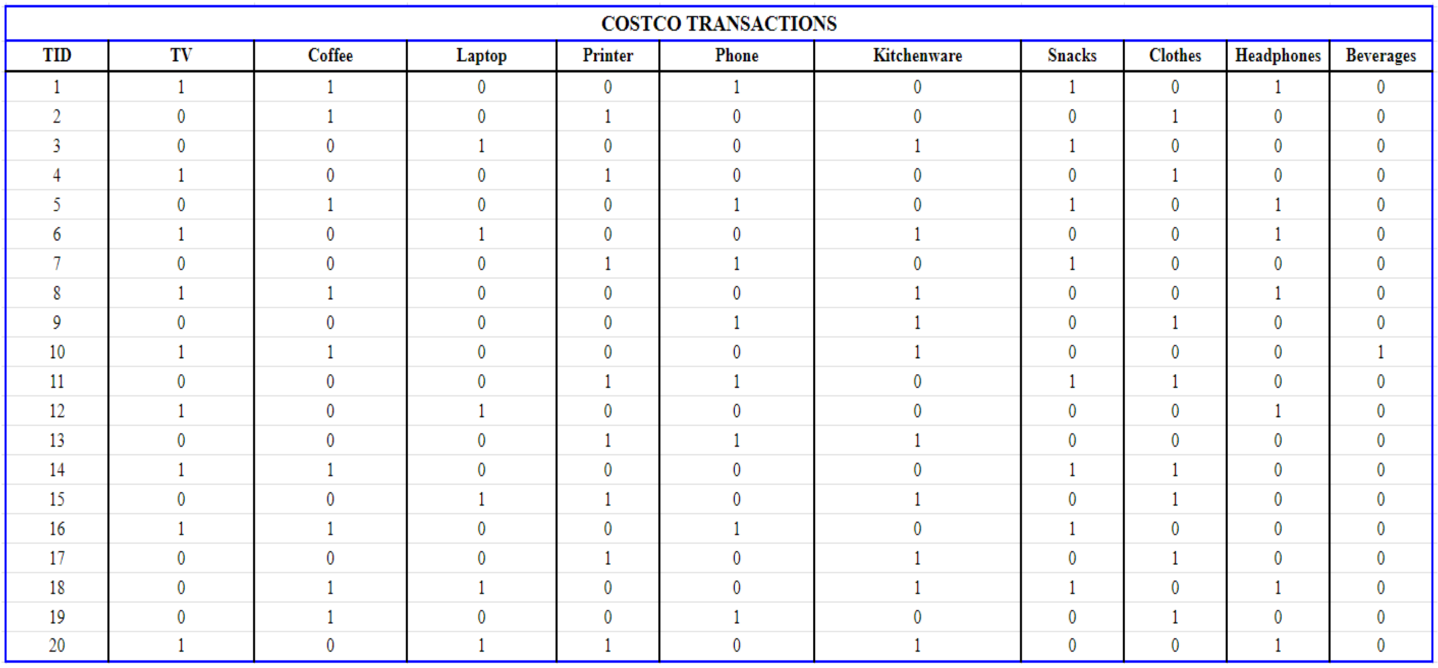
COSTCO ITEMS	
ID	**Items**
1	TV
2	Coffee
3	Laptop
4	Printer
5	Phone
6	Kitchenware
7	Snacks
8	Clothes
9	Headphones
10	Beverages

Table: 13 (Costco Item Table)



Costco Transactions	
TID	**Transaction Details**
1	TV, Coffee, Phone, Snacks, Headphones
2	Coffee, Printer, Clothes
3	Laptop, Kitchenware, Snacks
4	TV, Printer, Clothes
5	Coffee, Phone, Snacks, Beverages
6	TV, Laptop, Kitchenware, Headphones
7	Printer, Phone, Snacks
8	TV, Coffee, Kitchenware, Headphones
9	Phone, Kitchenware, Clothes
10	TV, Coffee, Kitchenware, Beverages
11	Printer, Phone, Snacks, Clothes
12	TV, Laptop, Headphones
13	Printer, Phone, Beverages
14	TV, Coffee, Snacks, Clothes
15	Laptop, Printer, Snacks, Clothes
16	TV, Coffee, Phone, Snacks
17	Printer, Kitchenware, Clothes
18	Coffee, Laptop, Snacks, Beverages
19	Coffee, Phone, Clothes
20	TV, Laptop, Printer, Snacks, Headphones

Table: 14 (Costco Transaction Table)



COSTCO TRANSACTIONS										
TID	**TV**	**Coffee**	**Laptop**	**Printer**	**Phone**	**Kitchenware**	**Snacks**	**Clothes**	**Headphones**	**Beverages**
1	1	1	0	0	1	0	1	0	1	0
2	0	1	0	1	0	0	0	1	0	0
3	0	0	1	0	0	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	1	0	0	1	0	1	0	1	0
6	1	0	1	0	0	1	0	0	1	0
7	0	0	0	1	1	0	1	0	0	0
8	1	1	0	0	0	1	0	0	1	0
9	0	0	0	0	1	1	0	1	0	0
10	1	1	0	0	0	1	0	0	0	1
11	0	0	0	1	1	0	1	1	0	0
12	1	0	1	0	0	0	0	0	1	0
13	0	0	0	1	1	1	0	0	0	0
14	1	1	0	0	0	0	1	1	0	0
15	0	0	1	1	0	1	0	1	0	0
16	1	1	0	0	1	0	1	0	0	0
17	0	0	0	1	0	1	0	1	0	0
18	0	1	1	0	0	1	1	0	1	0
19	0	1	0	0	1	0	0	1	0	0
20	1	0	1	1	0	1	0	0	1	0

Table: 15 (Costco Binary Table)

Note: 0 indicates that the particular item is not present in the transaction and **1** indicates that the item is present. This is used for the "**Tidy Representation**" of the dataset.

HOW TO RUN ASSOCIATION RULE MINING CODE?

- **Prerequisites**

Before running the code, ensure that you have the following installed on your system:

1. **Python 3.6 or higher**: The code is compatible with Python versions 3.6 and above. You can download Python from python.org.
2. **Required Libraries**:
   - **pandas**: Used for data manipulation and analysis.
   - **warnings**: Used for fixing the warnings.
   - **time**: Used for various time related functions.
   - **itertools**: It provides functions for efficient looping and creating iterators, like combinations and permutations.

The installation of the required can be done by using executing **requreiments.txt** file or by **manually** installing by user:

- To install the necessary libraries for this project, a **requirements.txt** file is included. This file lists all the external libraries needed to run the code. Built-in Python libraries like **time**, **warnings**, and **itertools** do not need installation.
  - Make sure the **requirements.txt** file is in the same folder as your project file then you can run the following command in your terminal or command prompt after navigating to the folder with the project and **requirements.txt**, now run this command:

  "pip install -r requirements.txt"

  This will install all necessary libraries automatically.

- Install the necessary libraries using "**pip**". You can run the following command in your terminal or command prompt:

  "**pip install pandas mlxtend"**

  This command installs **pandas** for data manipulation and **mlxtend** for the association rule mining algorithms.

- **Setting Up the Environment**

1. **Download the Code**: Clone or Download the repository from GitHub Link provided at the last page of the report and ensure you have the complete code saved in a **(.py)** file or a Jupyter Notebook **(.ipynb)** file.

2. **Dataset**: For custom datasets, prepare your dataset in a **CSV format**. Ensure that the CSV file has a column named **"Transaction Details"** containing the transactions as comma-separated values. Place the CSV file in the same directory as the source code or provide the correct path when prompted. The default five datasets are located in the **"Datasets_CSV"** directory.

- **Running the Code**

1. **Open Terminal/Command Prompt**:
   - Navigate to the **directory** where your **Python file** or **Jupyter Notebook** is saved.
2. **Execute the Code**:
   - For a Python script, run the following command:

     "python source_code.py" or "python3 source_code.py"

   - For a Jupyter Notebook, you can open it using Jupyter Notebook or JupyterLab. If you don't have Jupyter installed, you can do so by running:

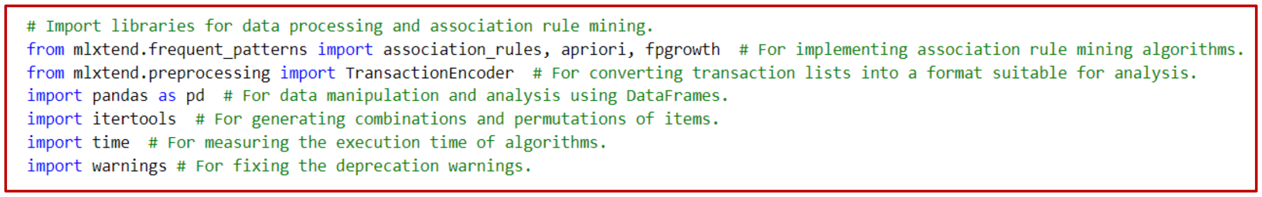
     "pip install notebook"

   - Then start with Jupyter Notebook:

     "jupyter notebook"
3. **Follow On-Screen Prompts**:
   - The program will guide you through selecting a dataset and entering the required parameters for analysis.
   - If you choose to use a custom CSV dataset, make sure to enter the correct file path when prompted.

- **Troubleshooting Common Issues**
  - **Import Errors**: If you encounter errors related to missing libraries, ensure that you have installed all required libraries as mentioned above. The time and warnings packages are included with Python and do not require separate installation.
  - **File Not Found**: If you get an error saying the CSV file is not found, double-check the file path and ensure the file is in the specified location.
  - **Invalid Input**: The program validates user inputs. If you enter an invalid choice or value, you will receive a prompt to try again.

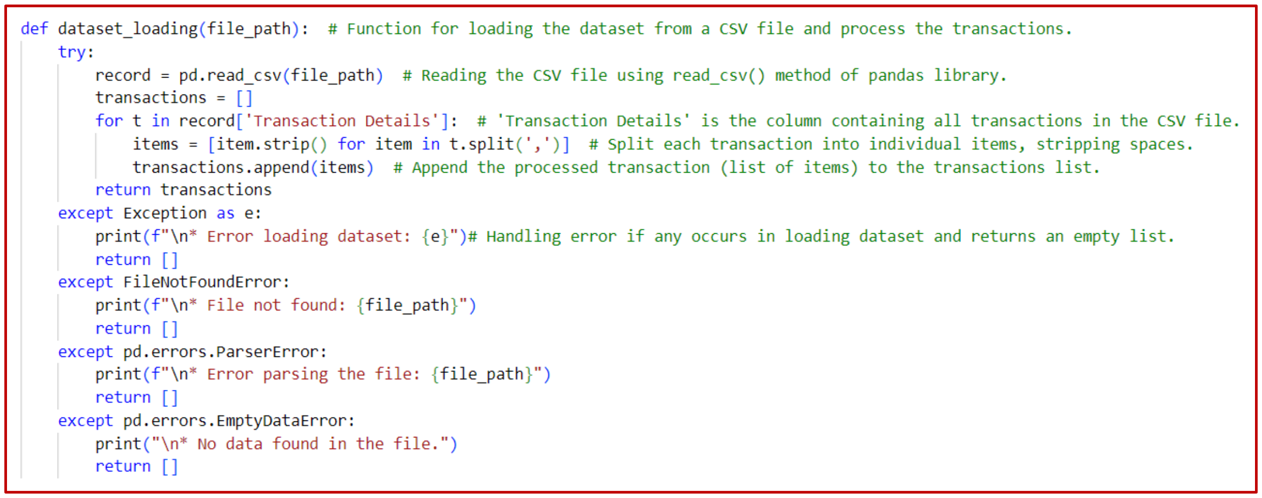
SCREENSHOTS



```python
# Import libraries for data processing and association rule mining.
from mlxtend.frequent_patterns import association_rules, apriori, fpgrowth  # For implementing association rule mining algorithms.
from mlxtend.preprocessing import TransactionEncoder  # For converting transaction lists into a format suitable for analysis.
import pandas as pd  # For data manipulation and analysis using DataFrames.
import itertools  # For generating combinations and permutations of items.
import time  # For measuring the execution time of algorithms.
import warnings # For fixing the deprecation warnings.
```

Figure: 1

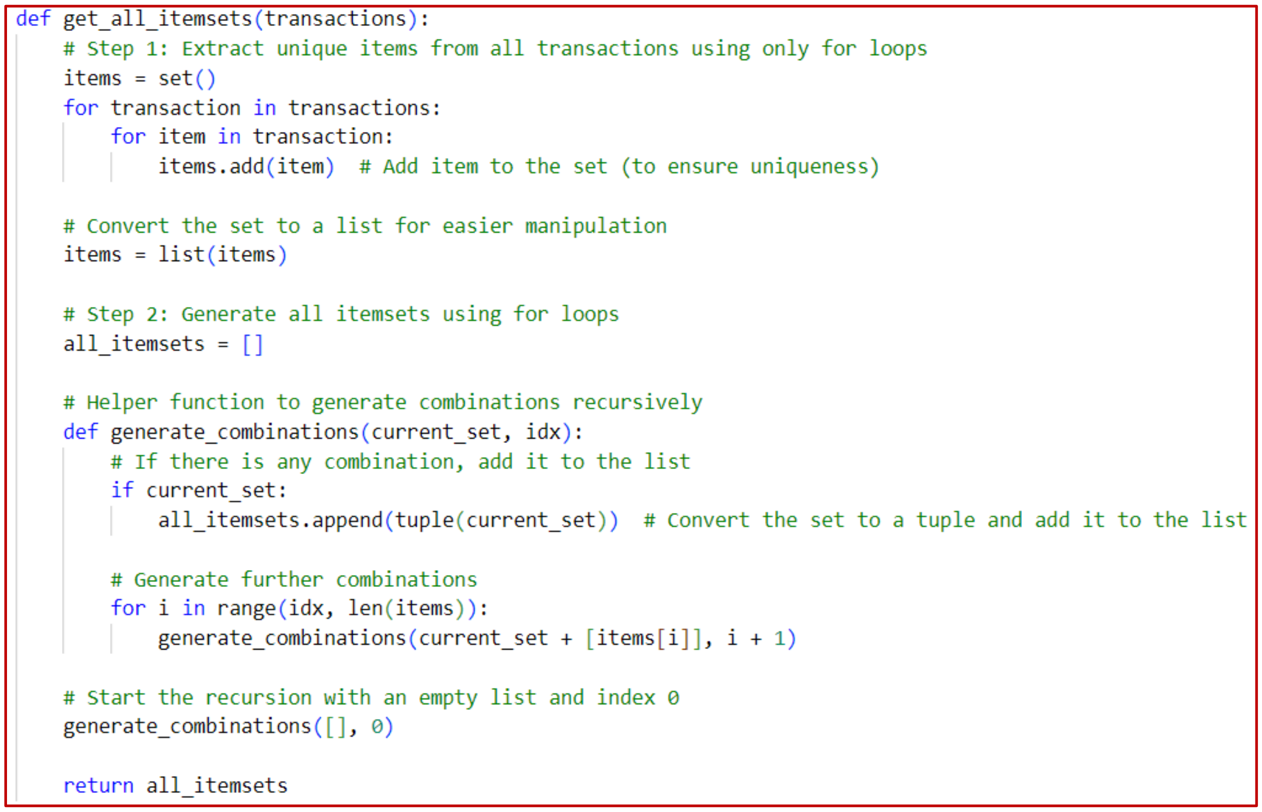
This section imports the necessary libraries for data processing, implementing association rule mining algorithms, and handling potential warnings.



```python
def dataset_loading(file_path):  # Function for loading the dataset from a CSV file and process the transactions.
    try:
        record = pd.read_csv(file_path)  # Reading the CSV file using read_csv() method of pandas library.
        transactions = []
        for t in record['Transaction Details']:  # 'Transaction Details' is the column containing all transactions in the CSV file.
            items = [item.strip() for item in t.split(',')]  # Split each transaction into individual items, stripping spaces.
            transactions.append(items)  # Append the processed transaction (list of items) to the transactions list.
        return transactions
    except Exception as e:
        print(f"\n* Error loading dataset: {e}")# Handling error if any occurs in loading dataset and returns an empty list.
        return []
    except FileNotFoundError:
        print(f"\n* File not found: {file_path}")
        return []
    except pd.errors.ParserError:
        print(f"\n* Error parsing the file: {file_path}")
        return []
    except pd.errors.EmptyDataError:
        print("\n* No data found in the file.")
        return []
```

Figure: 2

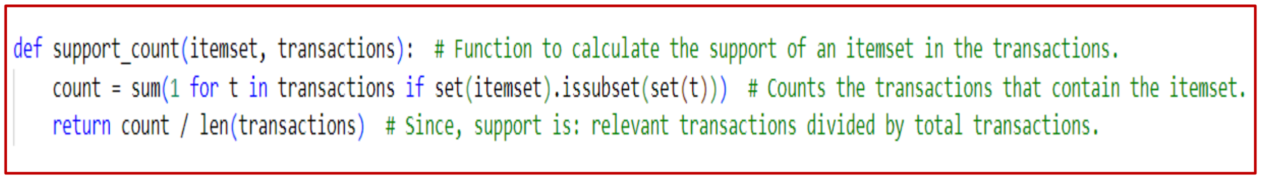
This function loads the dataset from a specified CSV file, processes the transactions, and handles any potential loading errors.



```python
def get_all_itemsets(transactions):
    # Step 1: Extract unique items from all transactions using only for loops
    items = set()
    for transaction in transactions:
        for item in transaction:
            items.add(item)  # Add item to the set (to ensure uniqueness)

    # Convert the set to a list for easier manipulation
    items = list(items)

    # Step 2: Generate all itemsets using for loops
    all_itemsets = []

    # Helper function to generate combinations recursively
    def generate_combinations(current_set, idx):
        # If there is any combination, add it to the list
        if current_set:
            all_itemsets.append(tuple(current_set))  # Convert the set to a tuple and add it to the list

        # Generate further combinations
        for i in range(idx, len(items)):
            generate_combinations(current_set + [items[i]], i + 1)

    # Start the recursion with an empty list and index 0
    generate_combinations([], 0)

    return all_itemsets
```

Figure: 3

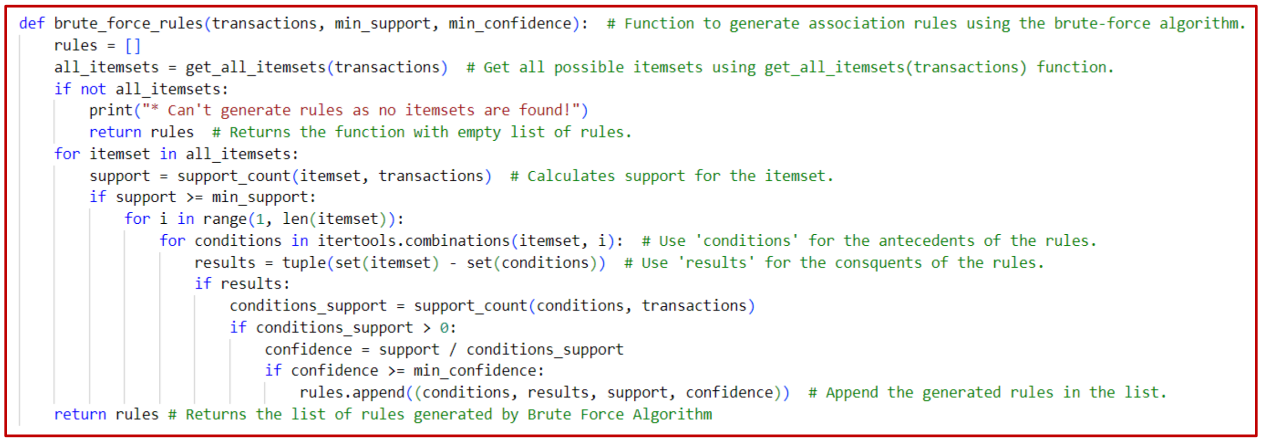
This section defines a function to extract unique items from transactions and generate all possible itemsets through recursive combinations.



```python
def support_count(itemset, transactions):  # Function to calculate the support of an itemset in the transactions.
    count = sum(1 for t in transactions if set(itemset).issubset(set(t)))  # Counts the transactions that contain the itemset.
    return count / len(transactions)  # Since, support is: relevant transactions divided by total transactions.
```

Figure: 4

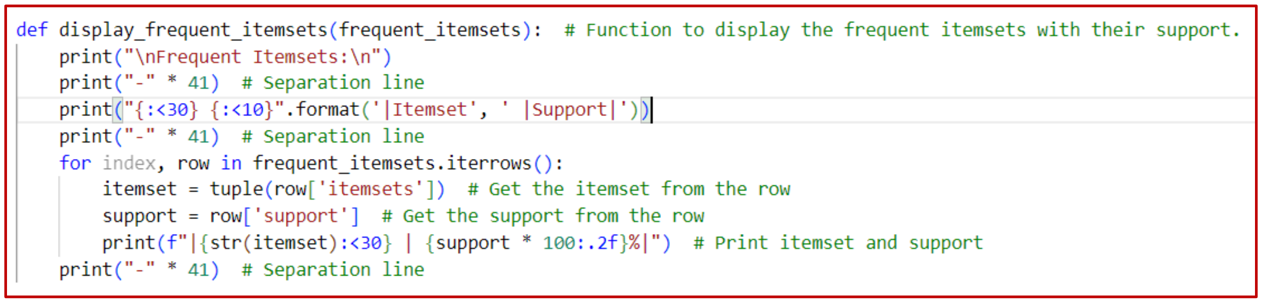
This function calculates the support of a given itemset based on the transactions.



```python
def brute_force_rules(transactions, min_support, min_confidence):  # Function to generate association rules using the brute-force algorithm.
    rules = []
    all_itemsets = get_all_itemsets(transactions)  # Get all possible itemsets using get_all_itemsets(transactions) function.
    if not all_itemsets:
        print("* Can't generate rules as no itemsets are found!")
        return rules  # Returns the function with empty list of rules.
    for itemset in all_itemsets:
        support = support_count(itemset, transactions)  # Calculates support for the itemset.
        if support >= min_support:
            for i in range(1, len(itemset)):
                for conditions in itertools.combinations(itemset, i):  # Use 'conditions' for the antecedents of the rules.
                    results = tuple(set(itemset) - set(conditions))  # Use 'results' for the consquents of the rules.
                    if results:
                        conditions_support = support_count(conditions, transactions)
                        if conditions_support > 0:
                            confidence = support / conditions_support
                            if confidence >= min_confidence:
                                rules.append((conditions, results, support, confidence))  # Append the generated rules in the list.
    return rules # Returns the list of rules generated by Brute Force Algorithm
```

Figure: 5

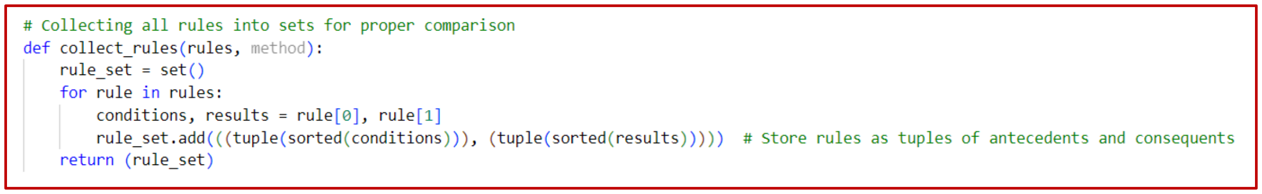
This function implements the brute-force algorithm to generate association rules based on the calculated support and confidence.



```python
def display_frequent_itemsets(frequent_itemsets):  # Function to display the frequent itemsets with their support.
    print("\nFrequent Itemsets:\n")
    print("-" * 41)  # Separation line
    print("{:<30} {:<10}".format('|Itemset', ' |Support|'))
    print("-" * 41)  # Separation line
    for index, row in frequent_itemsets.iterrows():
        itemset = tuple(row['itemsets'])  # Get the itemset from the row
        support = row['support']  # Get the support from the row
        print(f"|{str(itemset):<30} | {support * 100:.2f}%|")  # Print itemset and support
    print("-" * 41)  # Separation line
```

Figure: 6

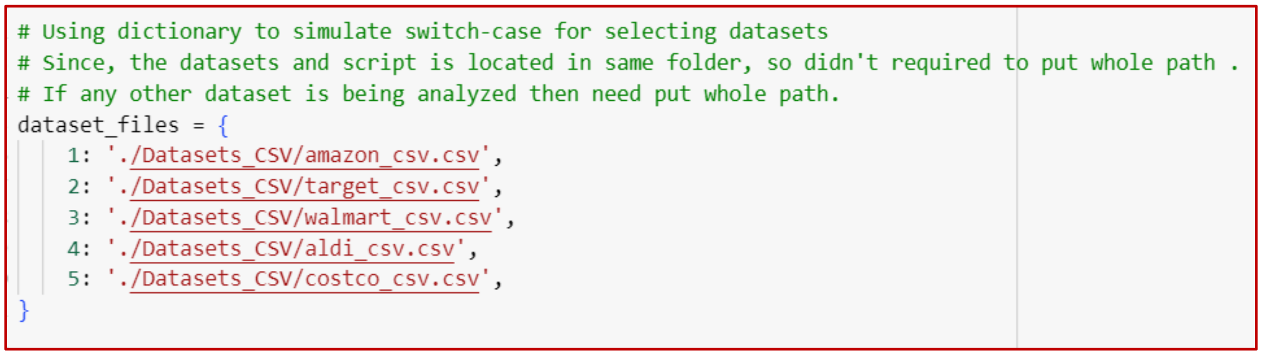
This section defines a function to display the frequent itemsets along with their support values in a formatted manner.



```python
# Collecting all rules into sets for proper comparison
def collect_rules(rules, method):
    rule_set = set()
    for rule in rules:
        conditions, results = rule[0], rule[1]
        rule_set.add(((tuple(sorted(conditions))), (tuple(sorted(results)))))  # Store rules as tuples of antecedents and consequents
    return (rule_set)
```

Figure: 7

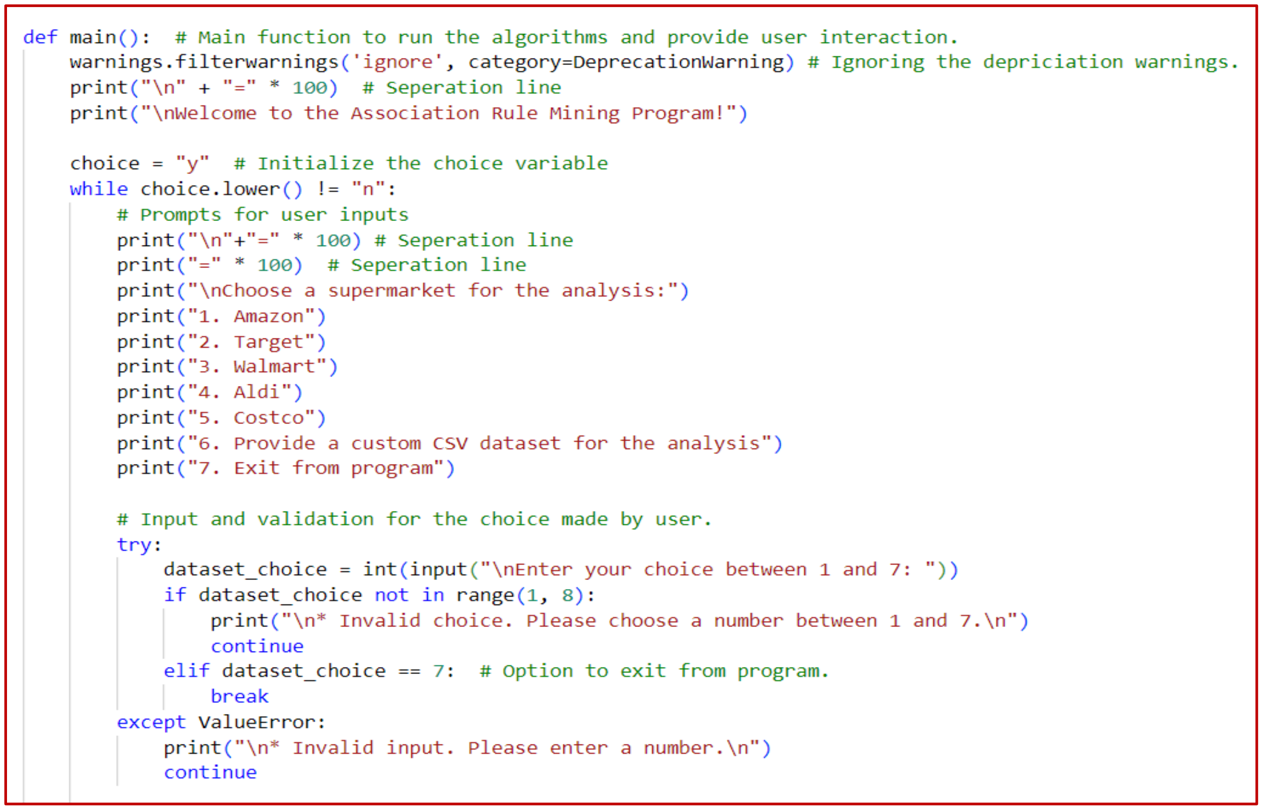
This function collects generated rules into a set for easy comparison of rules across different methods.



```python
# Using dictionary to simulate switch-case for selecting datasets
# Since, the datasets and script is located in same folder, so didn't required to put whole path .
# If any other dataset is being analyzed then need put whole path.
dataset_files = {
    1: './Datasets_CSV/amazon_csv.csv',
    2: './Datasets_CSV/target_csv.csv',
    3: './Datasets_CSV/walmart_csv.csv',
    4: './Datasets_CSV/aldi_csv.csv',
    5: './Datasets_CSV/costco_csv.csv',
}
```

Figure: 8

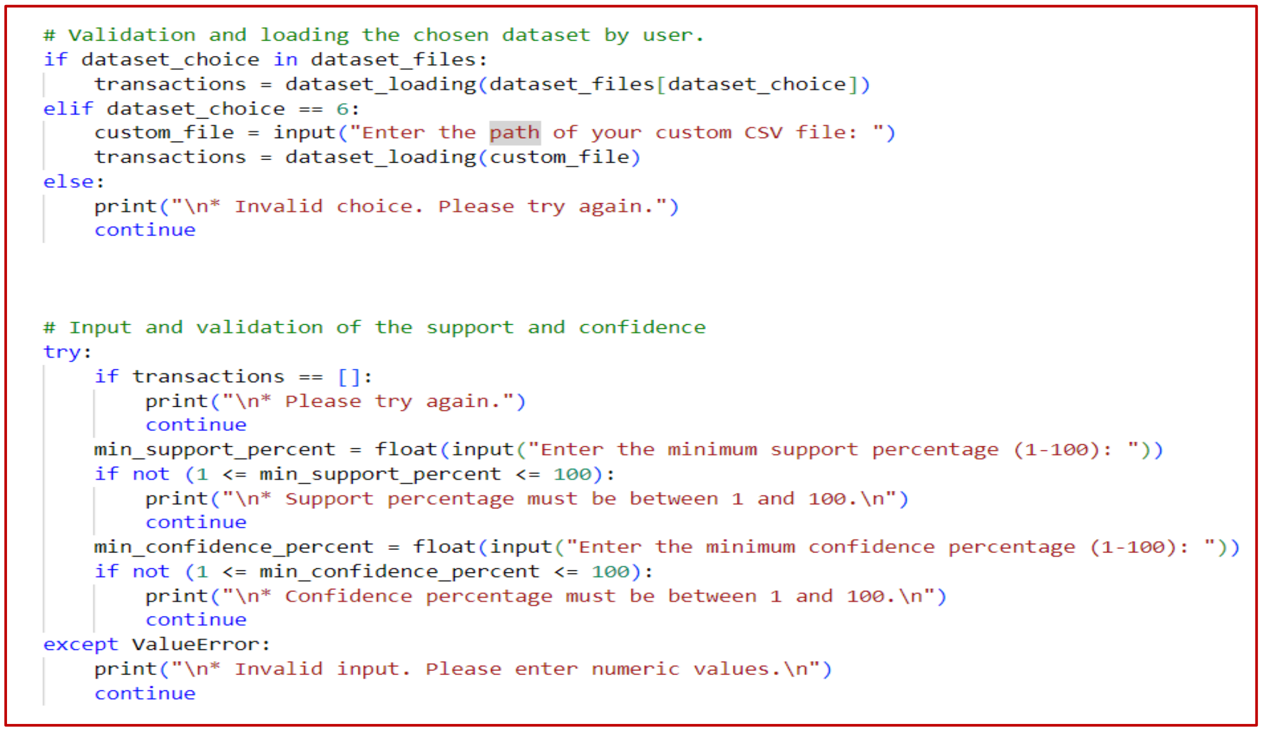
This dictionary simulates a switch-case structure to facilitate the selection of datasets for analysis.



```python
def main():  # Main function to run the algorithms and provide user interaction.
    warnings.filterwarnings('ignore', category=DeprecationWarning) # Ignoring the depriciation warnings.
    print("\n" + "=" * 100)  # Seperation line
    print("\nWelcome to the Association Rule Mining Program!")

    choice = "y"  # Initialize the choice variable
    while choice.lower() != "n":
        # Prompts for user inputs
        print("\n"+"=" * 100) # Seperation line
        print("=" * 100)  # Seperation line
        print("\nChoose a supermarket for the analysis:")
        print("1. Amazon")
        print("2. Target")
        print("3. Walmart")
        print("4. Aldi")
        print("5. Costco")
        print("6. Provide a custom CSV dataset for the analysis")
        print("7. Exit from program")

        # Input and validation for the choice made by user.
        try:
            dataset_choice = int(input("\nEnter your choice between 1 and 7: "))
            if dataset_choice not in range(1, 8):
                print("\n* Invalid choice. Please choose a number between 1 and 7.\n")
                continue
            elif dataset_choice == 7:  # Option to exit from program.
                break
        except ValueError:
            print("\n* Invalid input. Please enter a number.\n")
            continue
```

Figure: 9

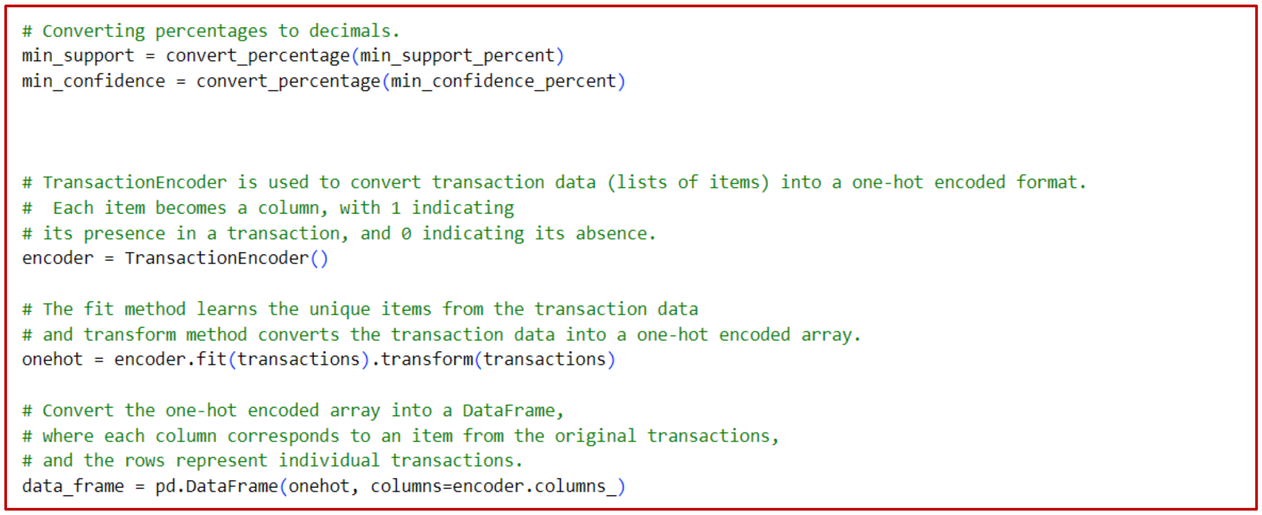
This section handles user input for selecting the dataset and parameters, ensuring proper validation and error handling.



```python
# Validation and loading the chosen dataset by user.
if dataset_choice in dataset_files:
    transactions = dataset_loading(dataset_files[dataset_choice])
elif dataset_choice == 6:
    custom_file = input("Enter the path of your custom CSV file: ")
    transactions = dataset_loading(custom_file)
else:
    print("\n* Invalid choice. Please try again.")
    continue



# Input and validation of the support and confidence
try:
    if transactions == []:
        print("\n* Please try again.")
        continue
    min_support_percent = float(input("Enter the minimum support percentage (1-100): "))
    if not (1 <= min_support_percent <= 100):
        print("\n* Support percentage must be between 1 and 100.\n")
        continue
    min_confidence_percent = float(input("Enter the minimum confidence percentage (1-100): "))
    if not (1 <= min_confidence_percent <= 100):
        print("\n* Confidence percentage must be between 1 and 100.\n")
        continue
except ValueError:
    print("\n* Invalid input. Please enter numeric values.\n")
    continue
```

Figure: 10

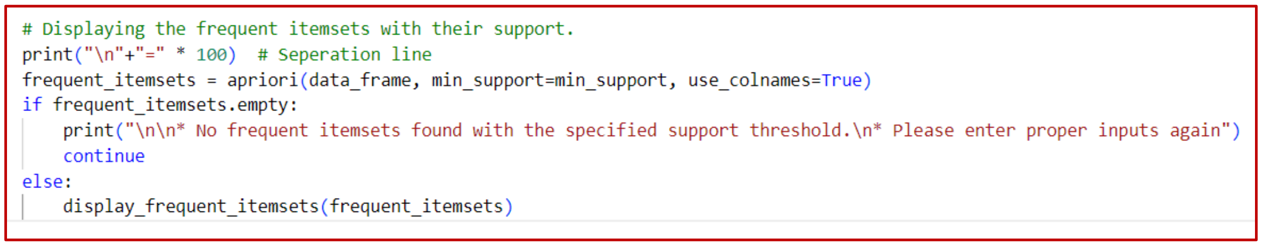
This section processes transactions based on user input, validating the entered support and confidence percentages.



```python
# Converting percentages to decimals.
min_support = convert_percentage(min_support_percent)
min_confidence = convert_percentage(min_confidence_percent)



# TransactionEncoder is used to convert transaction data (lists of items) into a one-hot encoded format.
#  Each item becomes a column, with 1 indicating
# its presence in a transaction, and 0 indicating its absence.
encoder = TransactionEncoder()

# The fit method learns the unique items from the transaction data
# and transform method converts the transaction data into a one-hot encoded array.
onehot = encoder.fit(transactions).transform(transactions)

# Convert the one-hot encoded array into a DataFrame,
# where each column corresponds to an item from the original transactions,
# and the rows represent individual transactions.
data_frame = pd.DataFrame(onehot, columns=encoder.columns_)
```

Figure: 11

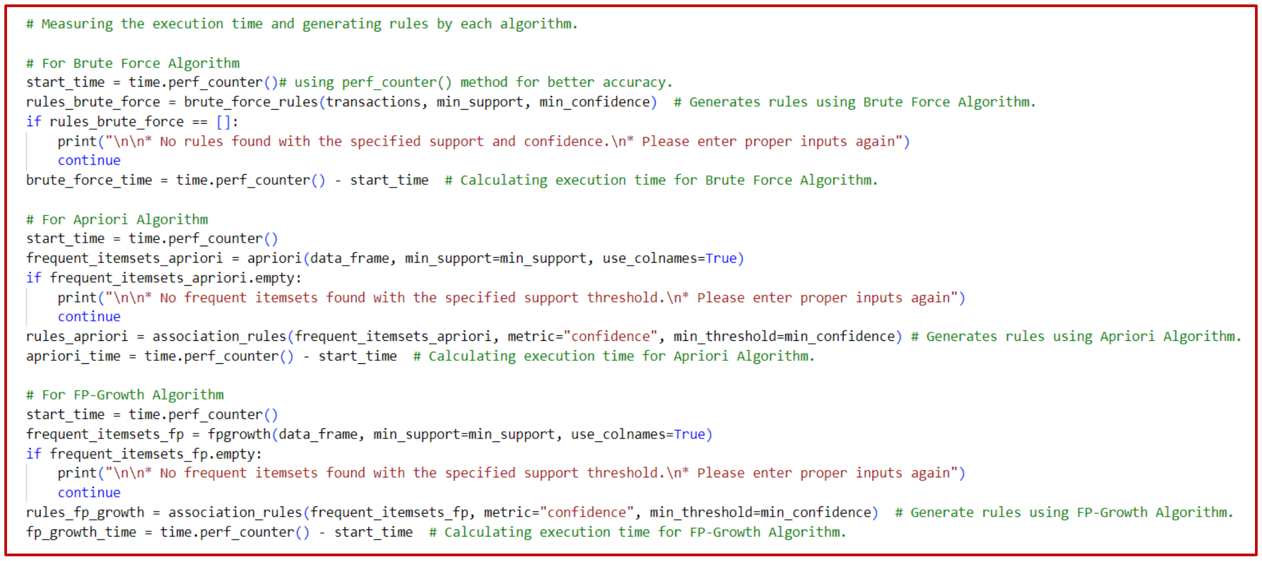
This section converts user input of percentage to decimals and implements the TransactionEncoder to convert dataset into onehot format for execution of in-built Apriori and FP-Growth algorithms.



```python
# Displaying the frequent itemsets with their support.
print("\n"+"=" * 100)  # Seperation line
frequent_itemsets = apriori(data_frame, min_support=min_support, use_colnames=True)
if frequent_itemsets.empty:
    print("\n\n* No frequent itemsets found with the specified support threshold.\n* Please enter proper inputs again")
    continue
else:
    display_frequent_itemsets(frequent_itemsets)
```

Figure: 12

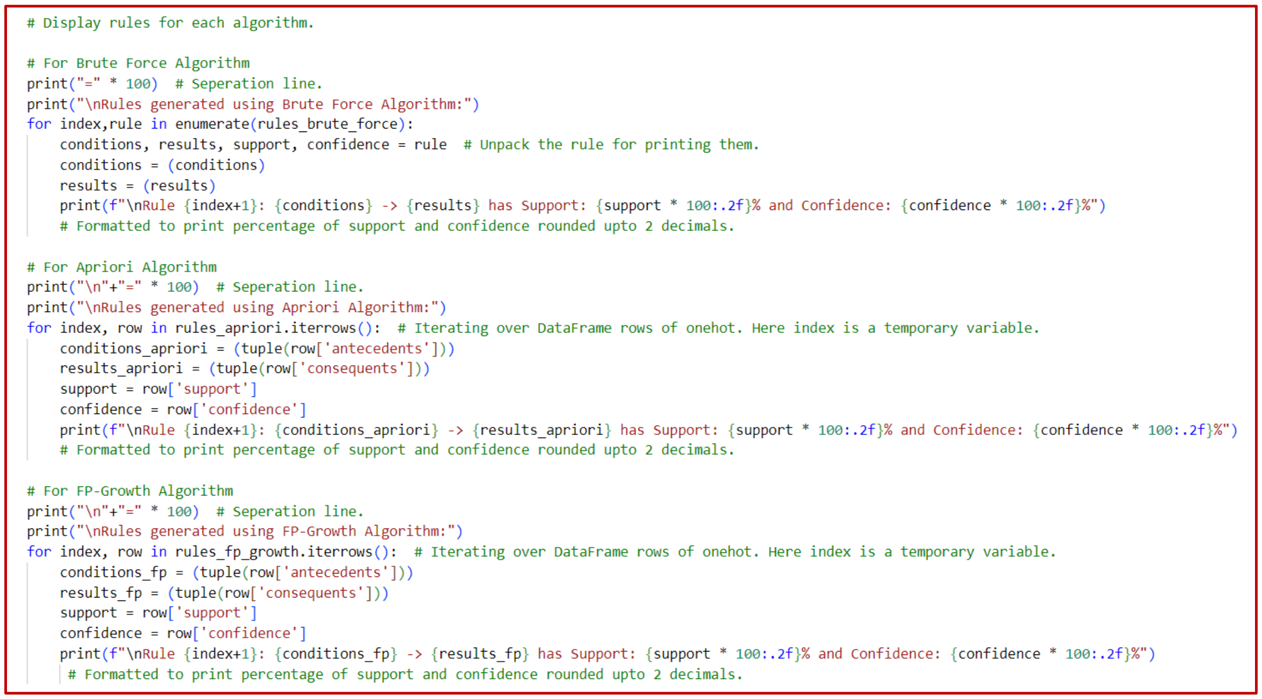
This section displays the frequent items of the dataset with their support.



```python
# Measuring the execution time and generating rules by each algorithm.

# For Brute Force Algorithm
start_time = time.perf_counter()# using perf_counter() method for better accuracy.
rules_brute_force = brute_force_rules(transactions, min_support, min_confidence)  # Generates rules using Brute Force Algorithm.
if rules_brute_force == []:
    print("\n\n* No rules found with the specified support and confidence.\n* Please enter proper inputs again")
    continue
brute_force_time = time.perf_counter() - start_time  # Calculating execution time for Brute Force Algorithm.

# For Apriori Algorithm
start_time = time.perf_counter()
frequent_itemsets_apriori = apriori(data_frame, min_support=min_support, use_colnames=True)
if frequent_itemsets_apriori.empty:
    print("\n\n* No frequent itemsets found with the specified support threshold.\n* Please enter proper inputs again")
    continue
rules_apriori = association_rules(frequent_itemsets_apriori, metric="confidence", min_threshold=min_confidence) # Generates rules using Apriori Algorithm.
apriori_time = time.perf_counter() - start_time  # Calculating execution time for Apriori Algorithm.

# For FP-Growth Algorithm
start_time = time.perf_counter()
frequent_itemsets_fp = fpgrowth(data_frame, min_support=min_support, use_colnames=True)
if frequent_itemsets_fp.empty:
    print("\n\n* No frequent itemsets found with the specified support threshold.\n* Please enter proper inputs again")
    continue
rules_fp_growth = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=min_confidence)  # Generate rules using FP-Growth Algorithm.
fp_growth_time = time.perf_counter() - start_time  # Calculating execution time for FP-Growth Algorithm.
```

Figure: 13

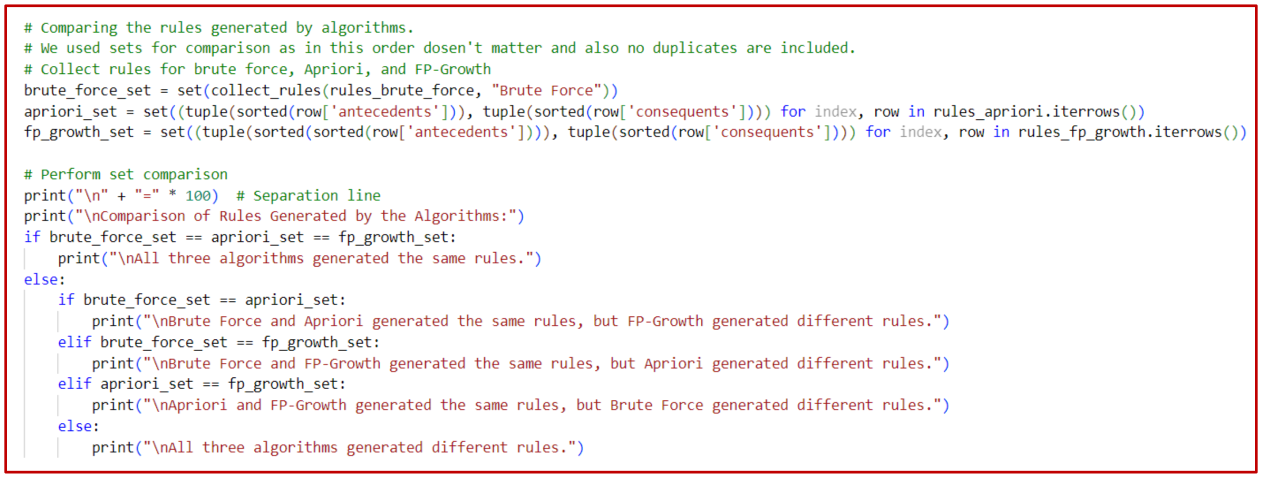
This section generates the rules by each algorithm and also extracts the execution time by each algorithm to generate that rule.



```python
# Display rules for each algorithm.

# For Brute Force Algorithm
print("=" * 100)  # Seperation line.
print("\nRules generated using Brute Force Algorithm:")
for index,rule in enumerate(rules_brute_force):
    conditions, results, support, confidence = rule  # Unpack the rule for printing them.
    conditions = (conditions)
    results = (results)
    print(f"\nRule {index+1}: {conditions} -> {results} has Support: {support * 100:.2f}% and Confidence: {confidence * 100:.2f}%")
    # Formatted to print percentage of support and confidence rounded upto 2 decimals.

# For Apriori Algorithm
print("\n"+"=" * 100)  # Seperation line.
print("\nRules generated using Apriori Algorithm:")
for index, row in rules_apriori.iterrows():  # Iterating over DataFrame rows of onehot. Here index is a temporary variable.
    conditions_apriori = (tuple(row['antecedents']))
    results_apriori = (tuple(row['consequents']))
    support = row['support']
    confidence = row['confidence']
    print(f"\nRule {index+1}: {conditions_apriori} -> {results_apriori} has Support: {support * 100:.2f}% and Confidence: {confidence * 100:.2f}%")
    # Formatted to print percentage of support and confidence rounded upto 2 decimals.

# For FP-Growth Algorithm
print("\n"+"=" * 100)  # Seperation line.
print("\nRules generated using FP-Growth Algorithm:")
for index, row in rules_fp_growth.iterrows():  # Iterating over DataFrame rows of onehot. Here index is a temporary variable.
    conditions_fp = (tuple(row['antecedents']))
    results_fp = (tuple(row['consequents']))
    support = row['support']
    confidence = row['confidence']
    print(f"\nRule {index+1}: {conditions_fp} -> {results_fp} has Support: {support * 100:.2f}% and Confidence: {confidence * 100:.2f}%")
     # Formatted to print percentage of support and confidence rounded upto 2 decimals.
```

Figure: 14

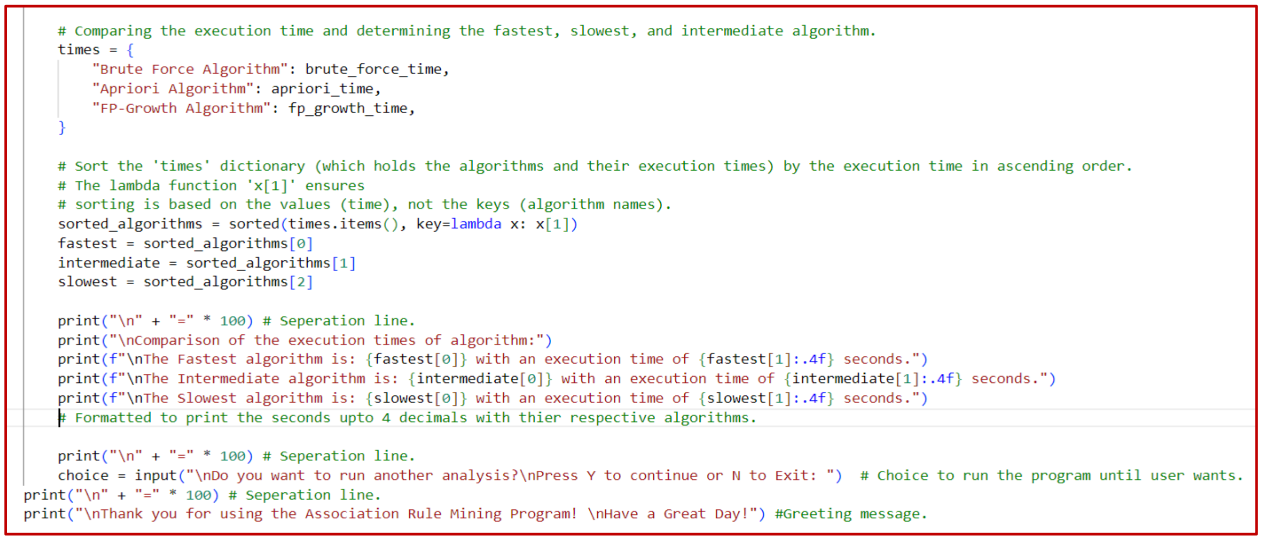
This section displays the rules generated by each algorithm.



```python
# Comparing the rules generated by algorithms.
# We used sets for comparison as in this order dosen't matter and also no duplicates are included.
# Collect rules for brute force, Apriori, and FP-Growth
brute_force_set = set(collect_rules(rules_brute_force, "Brute Force"))
apriori_set = set((tuple(sorted(row['antecedents'])), tuple(sorted(row['consequents']))) for index, row in rules_apriori.iterrows())
fp_growth_set = set((tuple(sorted(sorted(row['antecedents']))), tuple(sorted(row['consequents'])) for index, row in rules_fp_growth.iterrows())

# Perform set comparison
print("\n" + "=" * 100)  # Separation line
print("\nComparison of Rules Generated by the Algorithms:")
if brute_force_set == apriori_set == fp_growth_set:
    print("\nAll three algorithms generated the same rules.")
else:
    if brute_force_set == apriori_set:
        print("\nBrute Force and Apriori generated the same rules, but FP-Growth generated different rules.")
    elif brute_force_set == fp_growth_set:
        print("\nBrute Force and FP-Growth generated the same rules, but Apriori generated different rules.")
    elif apriori_set == fp_growth_set:
        print("\nApriori and FP-Growth generated the same rules, but Brute Force generated different rules.")
    else:
        print("\nAll three algorithms generated different rules.")
```

Figure: 15

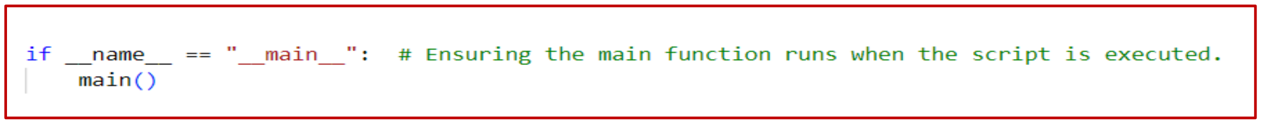
This section compares the rules generated by algorithms are same or not and gives a statement in output accordingly.



```python
    # Comparing the execution time and determining the fastest, slowest, and intermediate algorithm.
    times = {
        "Brute Force Algorithm": brute_force_time,
        "Apriori Algorithm": apriori_time,
        "FP-Growth Algorithm": fp_growth_time,
    }

    # Sort the 'times' dictionary (which holds the algorithms and their execution times) by the execution time in ascending order.
    # The lambda function 'x[1]' ensures
    # sorting is based on the values (time), not the keys (algorithm names).
    sorted_algorithms = sorted(times.items(), key=lambda x: x[1])
    fastest = sorted_algorithms[0]
    intermediate = sorted_algorithms[1]
    slowest = sorted_algorithms[2]

    print("\n" + "=" * 100) # Seperation line.
    print("\nComparison of the execution times of algorithm:")
    print(f"\nThe Fastest algorithm is: {fastest[0]} with an execution time of {fastest[1]:.4f} seconds.")
    print(f"\nThe Intermediate algorithm is: {intermediate[0]} with an execution time of {intermediate[1]:.4f} seconds.")
    print(f"\nThe Slowest algorithm is: {slowest[0]} with an execution time of {slowest[1]:.4f} seconds.")
    # Formatted to print the seconds upto 4 decimals with thier respective algorithms.

    print("\n" + "=" * 100) # Seperation line.
    choice = input("\nDo you want to run another analysis?\nPress Y to continue or N to Exit: ")  # Choice to run the program until user wants.
print("\n" + "=" * 100) # Seperation line.
print("\nThank you for using the Association Rule Mining Program! \nHave a Great Day!") #Greeting message.
```

Figure: 16

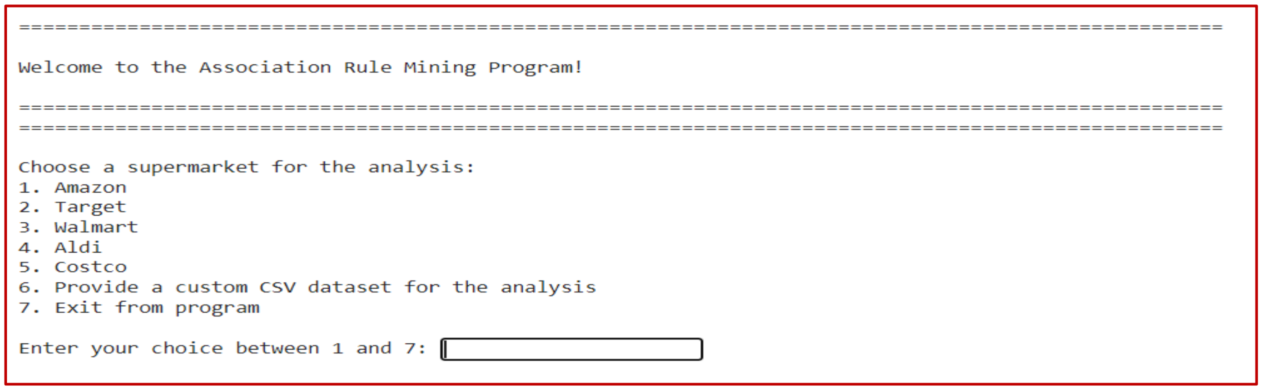
This section compares the execution time of each algorithm to generate rules and gives a statement in output about which algorithm is efficient.



```python
if __name__ == "__main__":  # Ensuring the main function runs when the script is executed.
    main()
```

Figure: 17

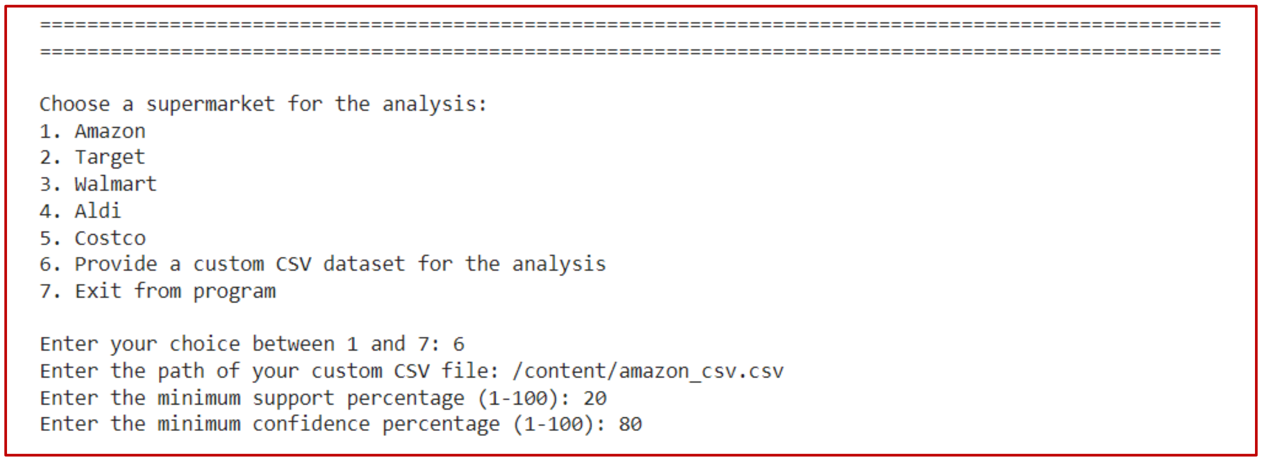
This is to ensure that main function executes when the script runs.



```
====================================================================================================

Welcome to the Association Rule Mining Program!

====================================================================================================
====================================================================================================

Choose a supermarket for the analysis:
1. Amazon
2. Target
3. Walmart
4. Aldi
5. Costco
6. Provide a custom CSV dataset for the analysis
7. Exit from program

Enter your choice between 1 and 7:
```

Figure: 18

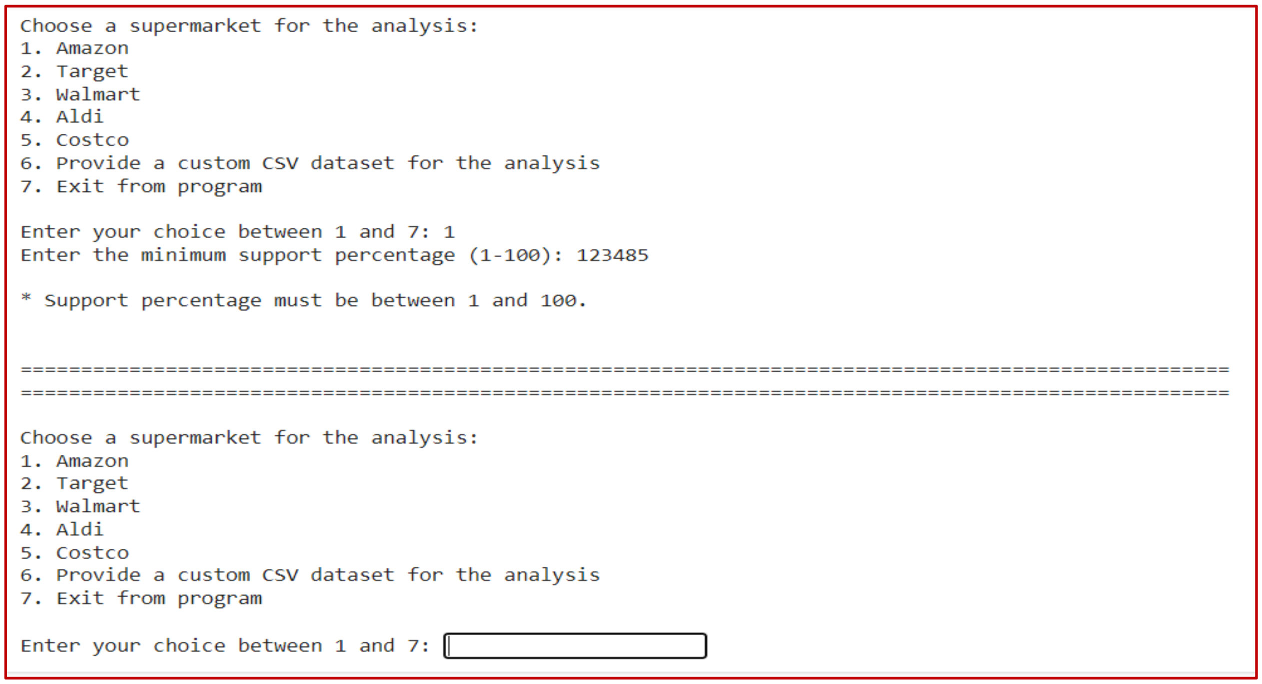
This is the output after running the script, here user is prompted with the choices of selecting dataset for the analysis and generating rules. User can choose one of the provided datasets or can provide a custom dataset from their system.



```
====================================================================================================
====================================================================================================

Choose a supermarket for the analysis:
1. Amazon
2. Target
3. Walmart
4. Aldi
5. Costco
6. Provide a custom CSV dataset for the analysis
7. Exit from program

Enter your choice between 1 and 7: 6
Enter the path of your custom CSV file: /content/amazon_csv.csv
Enter the minimum support percentage (1-100): 20
Enter the minimum confidence percentage (1-100): 80
```

Figure: 19

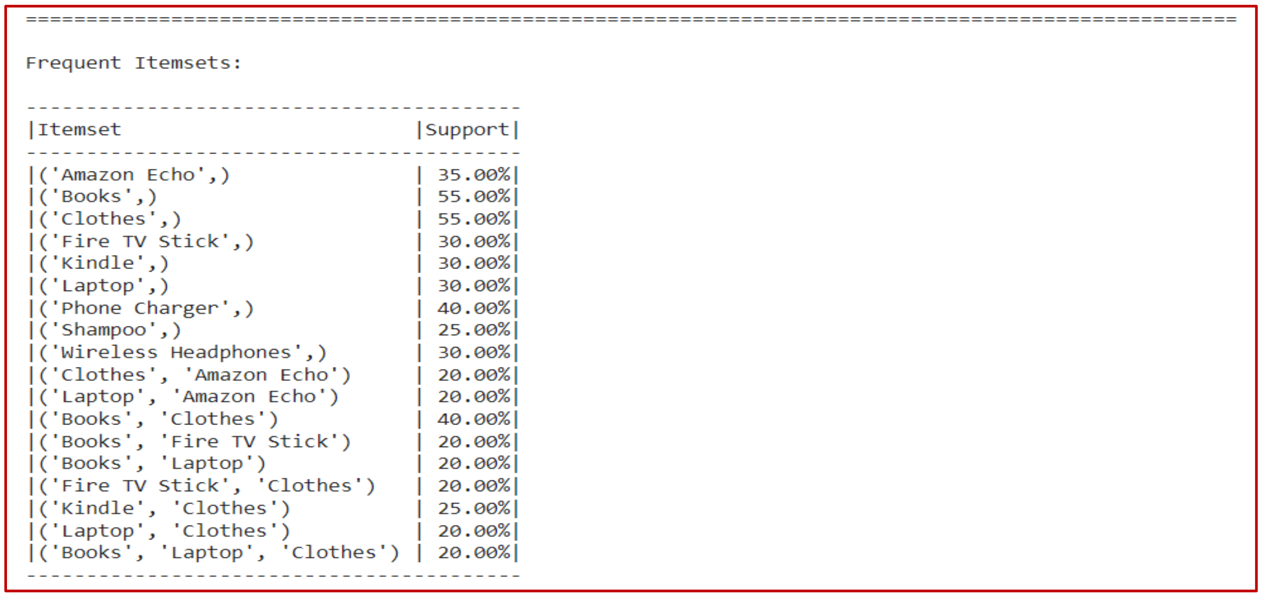
If user selects any provided dataset (1-5) after that minimum support and confidence has to be provided by the user in percentage (1-100), and if user selects to give their own dataset, path of the CSV file should be given as an input after choice is made as shown in above screenshot and later provide the thresholds for the analysis.



```
Choose a supermarket for the analysis:
1. Amazon
2. Target
3. Walmart
4. Aldi
5. Costco
6. Provide a custom CSV dataset for the analysis
7. Exit from program

Enter your choice between 1 and 7: 1
Enter the minimum support percentage (1-100): 123485

* Support percentage must be between 1 and 100.


====================================================================================================
====================================================================================================

Choose a supermarket for the analysis:
1. Amazon
2. Target
3. Walmart
4. Aldi
5. Costco
6. Provide a custom CSV dataset for the analysis
7. Exit from program

Enter your choice between 1 and 7:
```

Figure: 20

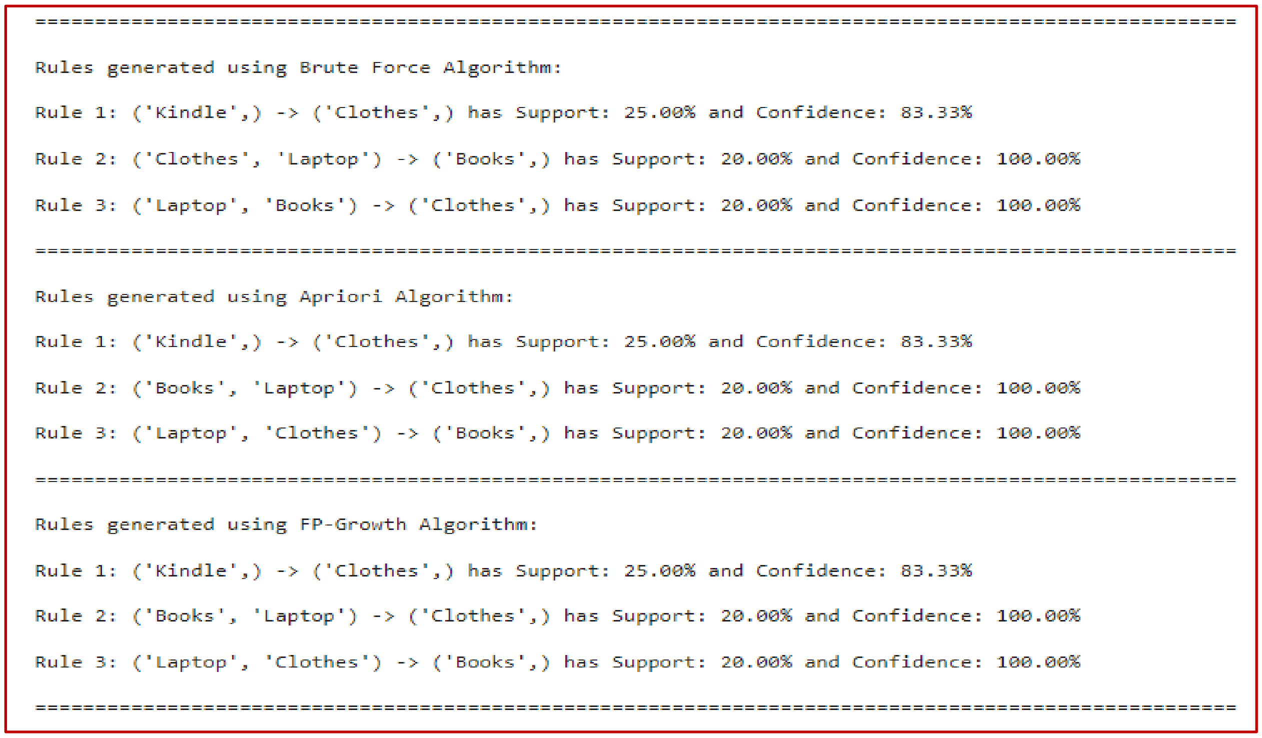
If inputs are inappropriate, the code will handle such situations and will ask user again for providing proper inputs for the analysis. The errors are shown with a preceding (*).



```
====================================================================================================

Frequent Itemsets:

-----------------------------------------
|Itemset                          |Support|
-----------------------------------------
|('Amazon Echo',)                 | 35.00%|
|('Books',)                       | 55.00%|
|('Clothes',)                     | 55.00%|
|('Fire TV Stick',)               | 30.00%|
|('Kindle',)                      | 30.00%|
|('Laptop',)                      | 30.00%|
|('Phone Charger',)               | 40.00%|
|('Shampoo',)                     | 25.00%|
|('Wireless Headphones',)         | 30.00%|
|('Clothes', 'Amazon Echo')       | 20.00%|
|('Laptop', 'Amazon Echo')        | 20.00%|
|('Books', 'Clothes')             | 40.00%|
|('Books', 'Fire TV Stick')       | 20.00%|
|('Books', 'Laptop')              | 20.00%|
|('Fire TV Stick', 'Clothes')     | 20.00%|
|('Kindle', 'Clothes')            | 25.00%|
|('Laptop', 'Clothes')            | 20.00%|
|('Books', 'Laptop', 'Clothes')   | 20.00%|
-----------------------------------------
```

Figure: 21

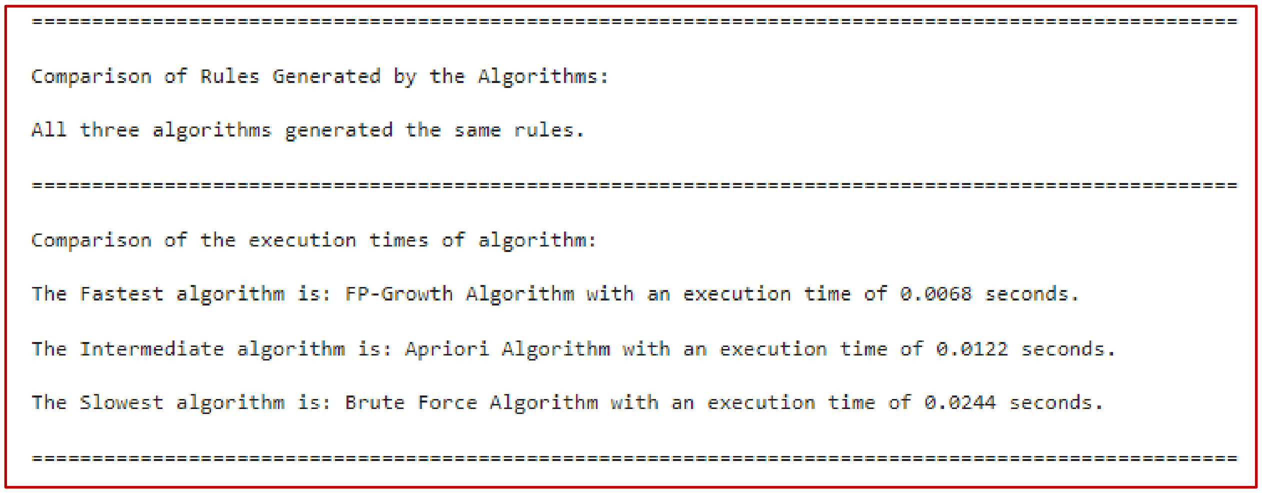
This is the frequent itemset table with itemsets and support for the itemsets.



```
====================================================================================================

Rules generated using Brute Force Algorithm:

Rule 1: ('Kindle',) -> ('Clothes',) has Support: 25.00% and Confidence: 83.33%

Rule 2: ('Clothes', 'Laptop') -> ('Books',) has Support: 20.00% and Confidence: 100.00%

Rule 3: ('Laptop', 'Books') -> ('Clothes',) has Support: 20.00% and Confidence: 100.00%

====================================================================================================

Rules generated using Apriori Algorithm:

Rule 1: ('Kindle',) -> ('Clothes',) has Support: 25.00% and Confidence: 83.33%

Rule 2: ('Books', 'Laptop') -> ('Clothes',) has Support: 20.00% and Confidence: 100.00%

Rule 3: ('Laptop', 'Clothes') -> ('Books',) has Support: 20.00% and Confidence: 100.00%

====================================================================================================

Rules generated using FP-Growth Algorithm:

Rule 1: ('Kindle',) -> ('Clothes',) has Support: 25.00% and Confidence: 83.33%

Rule 2: ('Books', 'Laptop') -> ('Clothes',) has Support: 20.00% and Confidence: 100.00%

Rule 3: ('Laptop', 'Clothes') -> ('Books',) has Support: 20.00% and Confidence: 100.00%

====================================================================================================
```

Figure: 22

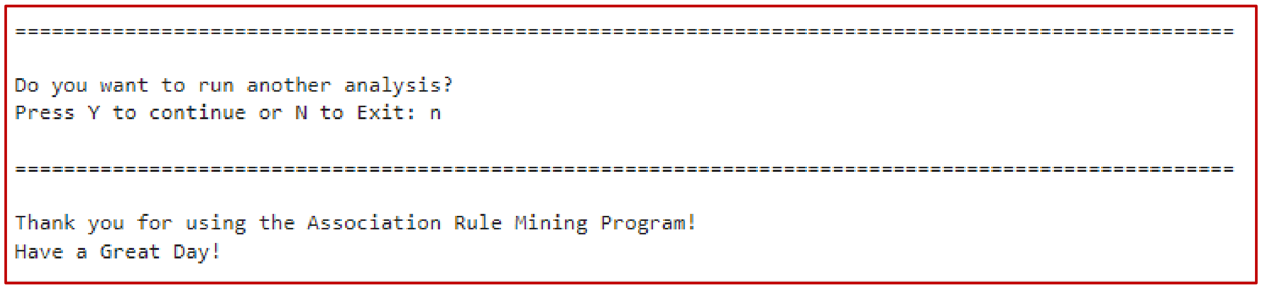
This is the rules generated by each algorithm with support and confidence of that rule.



```
==================================================================================================

Comparison of Rules Generated by the Algorithms:

All three algorithms generated the same rules.

==================================================================================================

Comparison of the execution times of algorithm:

The Fastest algorithm is: FP-Growth Algorithm with an execution time of 0.0068 seconds.

The Intermediate algorithm is: Apriori Algorithm with an execution time of 0.0122 seconds.

The Slowest algorithm is: Brute Force Algorithm with an execution time of 0.0244 seconds.

==================================================================================================
```

Figure: 23

This is the comparison of algorithms for the rules in terms of whether they all are generated same or not by them and also in terms of execution time for generating rules which is more efficient.



```
==================================================================================================

Do you want to run another analysis?
Press Y to continue or N to Exit: n

==================================================================================================

Thank you for using the Association Rule Mining Program!
Have a Great Day!
```

Figure: 24

After getting all the analysis, user is asked to do another analysis or just exit the program.

CONCLUSION

This project aimed to explore and implement various association rule mining algorithms: brute-force, Apriori, and FP-Growth, to analyze transaction datasets effectively. Each algorithm was designed to extract meaningful patterns from the data, with a focus on measuring support and confidence to derive actionable insights.

The findings highlighted the strengths and limitations of each algorithm. While the brute-force method provided a comprehensive set of association rules, its high computational cost made it less suitable for large datasets. In contrast, the Apriori algorithm improved efficiency by eliminating infrequent itemsets; however, it still faced challenges with scalability. The FP-Growth algorithm emerged as the most effective solution, leveraging its compact tree structure to significantly reduce execution time while maintaining accuracy in rule generation.

Throughout the project, rigorous error handling was implemented, ensuring that any issues encountered during execution were clearly indicated with a preceding asterisk (*), promoting transparency in the analysis.

The comparative evaluation of execution times and the number of rules generated underscored the importance of selecting the right algorithm based on the specific context and dataset size. This project not only reinforces the principles of association rule mining but also provides a foundation for future work in optimizing algorithms for enhanced performance and usability.

In conclusion, the insights gained from this analysis contribute to a better understanding of how different algorithms operate and their applicability in real-world scenarios. As data continues to grow in volume and complexity, further research and optimization of these algorithms will be essential for effective data mining and analysis.

SOURCE CODE AND REPOSITORY

- The source code (**.py** file) and data sets (**.csv** files) will be attached to the zip file.

- **Link for GitHub Repository:** https://github.com/Yash3561/Midterm_Project_DM