NAME: Kaviya Sree Ravikumar Meenakshi
NJIT UCID: kr549
Email Address: kr549@njit.edu
Date: 10.13.2024
Professor: Yasser Abduallah
Course: FA24 - CS634101 (Data Mining)

New Jersey Institute of Technology

MIDTERM PROJECT REPORT

Analysis of Association Rule Mining: A Comparative Study of Brute-Force Apriori, Built-in Apriori and FP-Tree

TABLE OF CONTENTS

S No.	Content	Page No.
1	Abstract	03
2	Introduction	04
3	Core Concepts and Principles	05
4	Project Workflow	07
5	Dataset Description	09
6	How to Execute the Code	16
7	Code	17
8	Results and Evaluation	22
9	Conclusion	25
10	Source Code and Repository	25

ABSTRACT

In this project, I explore association rule mining using three different strategies: a brute force Apriori algorithm, Apriori with a built-in Python function and the FP-tree algorithm. Here, five different retail transactional datasets are analyzed to extract meaningful patterns and relationships between items. Identifying association rules is crucial for businesses, as it helps uncover hidden connections between products, enabling more effective inventory management, targeted marketing and improved decision-making. The implementation of this project was quite challenging, particularly building the brute force part from scratch. The effectiveness of the three methods is demonstrated by the fact that all of them produced the same set of frequent itemsets. By implementing three strategies, I was also able to understand the computational strengths and challenges of each method. The analysis of computational time allows for a critical assessment of their performance, particularly when dealing with larger datasets. This project can be applied in market basket analysis to understand customer purchasing behavior by identifying item combinations that frequently occur together. It can also be used in recommendation systems to suggest products based on associated buying patterns.

INTRODUCTION

Data mining is the process of discovering patterns and extracting valuable insights from large sets of data. It helps companies to make informed decisions by identifying trends and relationships that may not be immediately evident. One popular technique in data mining is the Apriori algorithm, which is particularly effective in association rule learning. This algorithm aims to identify frequent itemsets within transactional data and subsequently derive rules that can guide decision-making.

The Apriori algorithm operates by systematically expanding the size of itemsets. It begins with individual items and gradually combines them into larger sets, all while filtering out those that fail to meet a specified minimum support threshold. This iterative approach ensures that only the most significant itemsets are considered, resulting in a more efficient mining process. When implementing Apriori using a brute force approach, the algorithm evaluates all possible combinations of items to determine their frequency in the dataset. Although this method is straightforward, it can be computationally intensive, especially with larger datasets, as it does not leverage any optimizations to reduce the number of combinations being checked.

In this project, I have used the mlxtend package functions for both the Apriori and FP-Growth algorithms, which enhance the efficiency of the overall mining process. The built-in functions simplify the implementation and allow for a more straightforward analysis of the datasets. The users will be prompted to choose between five datasets: Amazon, Best Buy, KMart, Nike, and Lidl. Each dataset presents unique transaction, enabling diverse analyses and insights. The results of this exploration will help to uncover valuable patterns that can inform strategic decision-making for these retail outlets.

CORE CONCEPTS AND PRINCIPLES

(1) Support

Support measures the frequency of an itemset appearing in a dataset. It helps identify the relevance of itemsets by quantifying how often they occur together. A higher support value indicates a more significant itemset within the dataset.

(2) Confidence

Confidence represents the likelihood that an item B is purchased when item A is purchased. It is expressed as a percentage and helps assess the strength of an association rule. A higher confidence value indicates a stronger relationship between the items.

(3) Frequent Itemsets

Frequent itemsets are combinations of items that appear together in transactions with a support above a defined threshold (minimum threshold). Identifying these sets is crucial for discovering patterns in consumer behavior.

(4) Association Rules

Association rules are implications of the form $A \rightarrow B$, indicating that if item A is purchased, item B is likely to be purchased as well. These rules help businesses understand relationships between products and can inform marketing strategies.

(5) Execution time

Execution time refers to the total duration it takes for a program or algorithm to complete its task on a computer system. This metric is crucial in evaluating the efficiency and performance of algorithms.

(6) Strategies/Algorithms used in the Project

As mentioned earlier, this project is basically a comparative analysis of three algorithms:

- Brute Force algorithm – This approach evaluates all possible combinations of items to identify frequent itemsets. While straightforward, it is computationally expensive and inefficient for large datasets, leading to long execution times.
- Apriori Algorithm – The Apriori algorithm builds frequent itemsets by incrementally adding items and uses a minimum support threshold to filter out less frequent sets. It is more efficient than brute force but can still struggle with large datasets due to its multiple database scans.
- FP-Growth – The FP-Growth algorithm uses a tree structure to store itemsets and allows for the discovery of frequent itemsets without generating candidate sets explicitly. This approach is generally faster than Apriori for larger datasets, as it requires only two passes through the data.

PROJECT WORKFLOW

The project workflow follows a systematic process in order to generate association rules using three methods: brute-force Apriori, Apriori using a built-in Python function and FP-growth. The steps are outlined below:

(1) Data Loading and Preprocessing

The first step involves loading the transactional datasets: Amazon, BestBuy, KMart, Nike and Lidl based on user input. Each dataset contains customer transaction records where each transaction is a list of items bought together. The dataset is preprocessed by converting these transactions into a structured format suitable (lists) for analysis. Unique items from all transactions are identified and the data is encoded into a binary format (one hot encoding; for built-in function), where each row represents a transaction and each column corresponds to an item (present or absent).

(2) Input of Minimum Support and Confidence

The user is prompted to input minimum support and confidence values. These thresholds are crucial for determining which itemsets are significant and which association rules will be generated. The minimum support defines the frequency of occurrence required for an itemset to be considered frequent, while the minimum confidence determines the strength of the relationship between items in an association rule.

(3) Brute Force Apriori (Frequent Itemset Generation)

The brute-force Apriori method generates all possible itemsets by starting with 1-itemsets and iteratively increasing the size of itemsets. For each itemset, the algorithm calculates how many transactions contain that itemset (support). Itemsets that meet the minimum support threshold are retained, while non-frequent ones are discarded. This process continues until no larger frequent itemsets can be found.

(4) Built-in Apriori and FP-growth

Built-in Python functions for Apriori (mlxtend.frequent_patterns.apriori) and FP-growth (mlxtend.frequent_patterns.fpgrowth) are used with the help of mlxtend package. These functions offer an optimized way of finding frequent itemsets without the need for manually generating and counting itemsets. These built-in methods are much more efficient, particularly for larger datasets.

(5) Association Rule Generation

Once the frequent itemsets are identified, association rules are generated. The rules are evaluated based on their confidence, with only the rules that meet the minimum confidence threshold being reported. This provides insight into which items are frequently purchased together.

(6) Time Analysis and Evaluation

The computational time for all three methods—brute force Apriori, Apriori using the built-in function and FP-growth – is recorded and compared to assess the efficiency of each approach. More on this will be discussed on the result section of this report. The efficiency of the project can be inferred by comparing the results from all three methods.

DATASET DESCRIPTION

For this project, a total of 5 datasets were used. A brief description on how each dataset was obtained is given below:

(1) Amazon – from sample dataset pdf

(2) BestBuy – from sample dataset pdf

(3) Kmart – from sample dataset pdf

(4) Nike – from sample dataset pdf

(5) Lidl – generated using https://www.lidl.com/ and ChatGPT

The above datasets were chosen because they feature popular products and stores and have well-organized transactions. Analyzing and deriving insights from this data will enable us to connect the actual outcomes with the generated rules thereby making the understanding of association rules easier.

Let us look at each dataset in detail.

(1) Amazon

Amazon is a leading global e-commerce platform known for its vast selection of products, customer-centric services and rapid delivery options. In our dataset, a list of books purchased from amazon are selected.

Characteristics of dataset:

Number of Items = 10

Number of transactions = 20

Dataset snapshots:

Table 1: Amazon Items

Item #	Item Name
1	A Beginner's Guide
2	Java: The Complete Reference
3	Java For Dummies
4	Android Programming: The Big Nerd Ranch
5	Head First Java 2nd Edition
6	Beginning Programming with Java
7	Java 8 Pocket Guide
8	C++ Programming in Easy Steps
9	Effective Java (2nd Edition)
10	HTML and CSS: Design and Build Websites

Table 2: Amazon Transactions

Transaction ID	Transaction
Trans1	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans2	A Beginner's Guide, Java: The Complete Reference, Java For Dummies
Trans3	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans4	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition, Beginning Programming with Java
Trans5	Android Programming: The Big Nerd Ranch, Beginning Programming with Java, Java 8 Pocket Guide
Trans6	A Beginner's Guide, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans7	A Beginner's Guide, Head First Java 2nd Edition, Beginning Programming with Java
Trans8	Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans9	Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition, Beginning Programming with Java
Trans10	Beginning Programming with Java, Java 8 Pocket Guide, C++ Programming in Easy Steps
Trans11	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans12	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, HTML and CSS: Design and Build Websites
Trans13	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Java 8 Pocket Guide, HTML and CSS: Design and Build Websites
Trans14	Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans15	Java For Dummies, Android Programming: The Big Nerd Ranch
Trans16	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans17	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans18	Head First Java 2nd Edition, Beginning Programming with Java, Java 8 Pocket Guide
Trans19	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans20	A Beginner's Guide, Java: The Complete Reference, Java For Dummies

(2) BestBuy

Best Buy is a global retailer of consumer electronics and technology products and services. In our dataset, a bunch of electronics and software services form the items.

Characteristics of dataset:

Number of Items = 10

Number of transactions = 20

Dataset snapshots:

Table 3: BestBuy Items

Item #	Item Name
1	Digital Camera
2	Lab Top
3	Desk Top
4	Printer
5	Flash Drive
6	Microsoft Office
7	Speakers
8	Lab Top Case
9	Anti-Virus
10	External Hard-Drive

Table 4: BestBuy Transactions

Transaction ID	Items
Trans1	Desk Top, Printer, Flash Drive, Microsoft Office, Speakers, Anti-Virus
Trans2	Lab Top, Flash Drive, Microsoft Office, Lab Top Case, Anti-Virus
Trans3	Lab Top, Printer, Flash Drive, Microsoft Office, Anti-Virus, Lab Top Case, External Hard-Drive
Trans4	Lab Top, Printer, Flash Drive, Anti-Virus, External Hard-Drive, Lab Top Case
Trans5	Lab Top, Flash Drive, Lab Top Case, Anti-Virus
Trans6	Lab Top, Printer, Flash Drive, Microsoft Office
Trans7	Desk Top, Printer, Flash Drive, Microsoft Office
Trans8	Lab Top, External Hard-Drive, Anti-Virus
Trans9	Desk Top, Printer, Flash Drive, Microsoft Office, Lab Top Case, Anti-Virus, Speakers, External Hard-Drive
Trans10	Digital Camera , Lab Top, Desk Top, Printer, Flash Drive, Microsoft Office, Lab Top Case, Anti-Virus, External Hard-Drive, Speakers
Trans11	Lab Top, Desk Top, Lab Top Case, External Hard-Drive, Speakers, Anti-Virus
Trans12	Digital Camera , Lab Top, Lab Top Case, External Hard-Drive, Anti-Virus, Speakers
Trans13	Digital Camera , Speakers
Trans14	Digital Camera , Desk Top, Printer, Flash Drive, Microsoft Office
Trans15	Printer, Flash Drive, Microsoft Office, Anti-Virus, Lab Top Case, Speakers, External Hard-Drive
Trans16	Digital Camera, Flash Drive, Microsoft Office, Anti-Virus, Lab Top Case, External Hard-Drive, Speakers
Trans17	Digital Camera , Lab Top, Lab Top Case
Trans18	Digital Camera , Lab Top Case, Speakers
Trans19	Digital Camera , Lab Top, Printer, Flash Drive, Microsoft Office, Speakers, Lab Top Case, Anti-Virus
Trans20	Digital Camera , Lab Top, Speakers, Anti-Virus, Lab Top Case

(3) Kmart

Kmart is an American retail chain known for offering a wide range of affordable household goods, clothing, and groceries, though its presence has significantly diminished in recent years. Our dataset contains home decor, bedroom and bathroom essentials products.

Characteristics of dataset:

Number of Items = 10

Number of transactions = 20

Dataset snapshots:

Table 5: Kmart Items

Item #	Item Name
1	Quilts
2	Bedspreads
3	Decorative Pillows
4	Bed Skirts
5	Sheets
6	Shams
7	Bedding Collections
8	Kids Bedding
9	Embroidered Bedspread
10	Towels

Table 6: Kmart Transactions

Transaction ID	Transactions
Trans1	Decorative Pillows, Quilts, Embroidered Bedspread
Trans2	Embroidered Bedspread, Shams, Kids Bedding, Bedding Collections, Bed Skirts, Bedspreads, Sheets
Trans3	Decorative Pillows, Quilts, Embroidered Bedspread, Shams, Kids Bedding, Bedding Collections
Trans4	Kids Bedding, Bedding Collections, Sheets, Bedspreads, Bed Skirts
Trans5	Decorative Pillows, Kids Bedding, Bedding Collections, Sheets, Bed Skirts, Bedspreads
Trans6	Bedding Collections, Bedspreads, Bed Skirts, Sheets, Shams, Kids Bedding
Trans7	Decorative Pillows, Quilts
Trans8	Decorative Pillows, Quilts, Embroidered Bedspread
Trans9	Bedspreads, Bed Skirts, Shams, Kids Bedding, Sheets
Trans10	Quilts, Embroidered Bedspread, Bedding Collections
Trans11	Bedding Collections, Bedspreads, Bed Skirts, Kids Bedding, Shams, Sheets
Trans12	Decorative Pillows, Quilts
Trans13	Embroidered Bedspread, Shams
Trans14	Sheets, Shams, Bed Skirts, Kids Bedding
Trans15	Decorative Pillows, Quilts
Trans16	Decorative Pillows, Kids Bedding, Bed Skirts, Shams
Trans17	Decorative Pillows, Shams, Bed Skirts
Trans18	Quilts, Sheets, Kids Bedding
Trans19	Shams, Bed Skirts, Kids Bedding, Sheets
Trans20	Decorative Pillows, Bedspreads, Shams, Sheets, Bed Skirts, Kids Bedding

(4) Nike

Nike is a global leader in athletic footwear, apparel and equipment, renowned for its innovative designs. Catering to its specialty, our dataset contains products from the sports, fitness, and activewear category.

Characteristics of dataset:

Number of Items = 10

Number of transactions = 20

Dataset snapshots:

Table 7: Nike Items

Item #	Item Name
1	Running Shoe
2	Soccer Shoe
3	Socks
4	Swimming Shirt
5	Dry Fit V-Nick
6	Rash Guard
7	Sweatshirts
8	Hoodies
9	Tech Pants
10	Modern Pants

Table 8: Nike Transactions

Transaction ID	Items
Trans1	Running Shoe, Socks, Sweatshirts, Modern Pants
Trans2	Running Shoe, Socks, Sweatshirts
Trans3	Running Shoe, Socks, Sweatshirts, Modern Pants
Trans4	Running Shoe, Sweatshirts, Modern Pants
Trans5	Running Shoe, Socks, Sweatshirts, Modern Pants, Soccer Shoe
Trans6	Running Shoe, Socks, Sweatshirts
Trans7	Running Shoe, Socks, Sweatshirts, Modern Pants, Tech Pants, Rash Guard, Hoodies
Trans8	Swimming Shirt, Socks, Sweatshirts
Trans9	Swimming Shirt, Rash Guard, Dry Fit V-Nick, Hoodies, Tech Pants
Trans10	Swimming Shirt, Rash Guard, Dry
Trans11	Swimming Shirt, Rash Guard, Dry Fit V-Nick
Trans12	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Hoodies, Tech Pants, Dry Fit V-Nick
Trans13	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Tech Pants, Dry Fit V-Nick, Hoodies
Trans14	Running Shoe, Swimming Shirt, Rash Guard, Tech Pants, Hoodies, Dry Fit V-Nick
Trans15	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Dry Fit V-Nick, Rash Guard, Tech Pants
Trans16	Swimming Shirt, Soccer Shoe, Hoodies, Dry Fit V-Nick, Tech Pants, Rash Guard
Trans17	Running Shoe, Socks
Trans18	Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Rash Guard, Tech Pants, Dry Fit V-Nick
Trans19	Running Shoe, Swimming Shirt, Rash Guard
Trans20	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Tech Pants, Rash Guard, Dry Fit V-Nick

(5) Lidl

Lidl is a global discount German supermarket chain known for offering a wide variety of groceries and household items at competitive prices. Our dataset contains basic food, beverages and groceries. This is a comparatively large dataset chosen for observing variation in computational time.

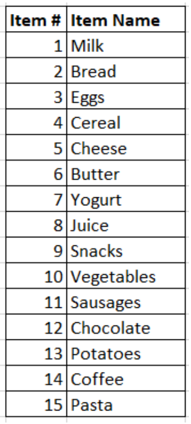
<u>Characteristics of dataset:</u>

Number of Items = 15

Number of transactions = 40

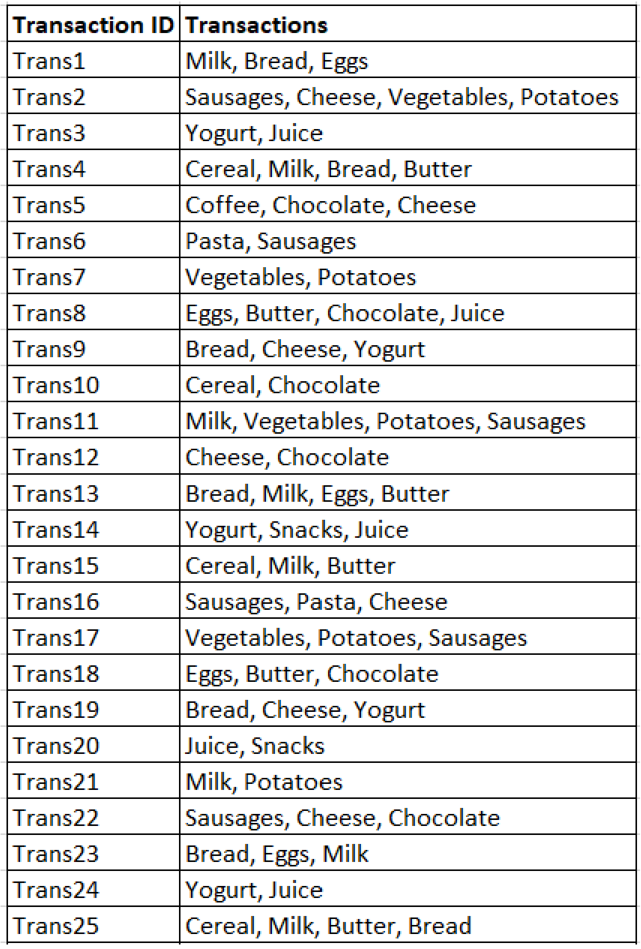
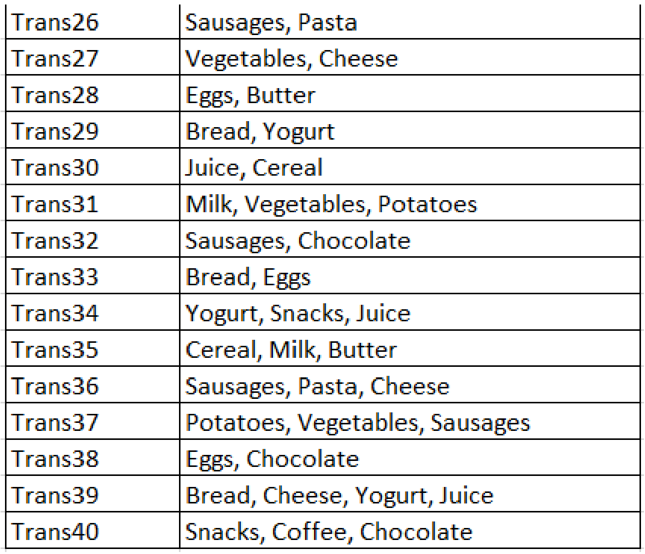
<u>Dataset snapshots:</u>

<u>Table 9: Lidl Items</u>



Item #	Item Name
1	Milk
2	Bread
3	Eggs
4	Cereal
5	Cheese
6	Butter
7	Yogurt
8	Juice
9	Snacks
10	Vegetables
11	Sausages
12	Chocolate
13	Potatoes
14	Coffee
15	Pasta

Table 10: Lidl Transactions

Transaction ID	**Transactions**
Trans1	Milk, Bread, Eggs
Trans2	Sausages, Cheese, Vegetables, Potatoes
Trans3	Yogurt, Juice
Trans4	Cereal, Milk, Bread, Butter
Trans5	Coffee, Chocolate, Cheese
Trans6	Pasta, Sausages
Trans7	Vegetables, Potatoes
Trans8	Eggs, Butter, Chocolate, Juice
Trans9	Bread, Cheese, Yogurt
Trans10	Cereal, Chocolate
Trans11	Milk, Vegetables, Potatoes, Sausages
Trans12	Cheese, Chocolate
Trans13	Bread, Milk, Eggs, Butter
Trans14	Yogurt, Snacks, Juice
Trans15	Cereal, Milk, Butter
Trans16	Sausages, Pasta, Cheese
Trans17	Vegetables, Potatoes, Sausages
Trans18	Eggs, Butter, Chocolate
Trans19	Bread, Cheese, Yogurt
Trans20	Juice, Snacks
Trans21	Milk, Potatoes
Trans22	Sausages, Cheese, Chocolate
Trans23	Bread, Eggs, Milk
Trans24	Yogurt, Juice
Trans25	Cereal, Milk, Butter, Bread
Trans26	Sausages, Pasta
Trans27	Vegetables, Cheese
Trans28	Eggs, Butter
Trans29	Bread, Yogurt
Trans30	Juice, Cereal
Trans31	Milk, Vegetables, Potatoes
Trans32	Sausages, Chocolate
Trans33	Bread, Eggs
Trans34	Yogurt, Snacks, Juice
Trans35	Cereal, Milk, Butter
Trans36	Sausages, Pasta, Cheese
Trans37	Potatoes, Vegetables, Sausages
Trans38	Eggs, Chocolate
Trans39	Bread, Cheese, Yogurt, Juice
Trans40	Snacks, Coffee, Chocolate

HOW TO EXECUTE THE CODE

(1) Perquisites

Although this program was coded using `Python 3.9.12`, this program can be run on systems with `Python 3.9` or higher. The program also requires the following basic libraries to be installed.

Table 11: Prerequisites

Library	Description	Command to install
pandas	A library used for data manipulation and analysis, especially for structured data.	`pip install pandas`
time	A library used for working with time-related functions, such as time delays or current time retrieval.	No installation needed, it's built-in
warnings	A library used for issuing warning messages to alert users of potential issues in the code.	No installation needed, it's built-in
mlxtend	A library that provides additional machine learning tools (Apriori and FP-Growth) and extensions, such as model stacking and plotting	`pip install mlxtend`

(2) Set up the environment

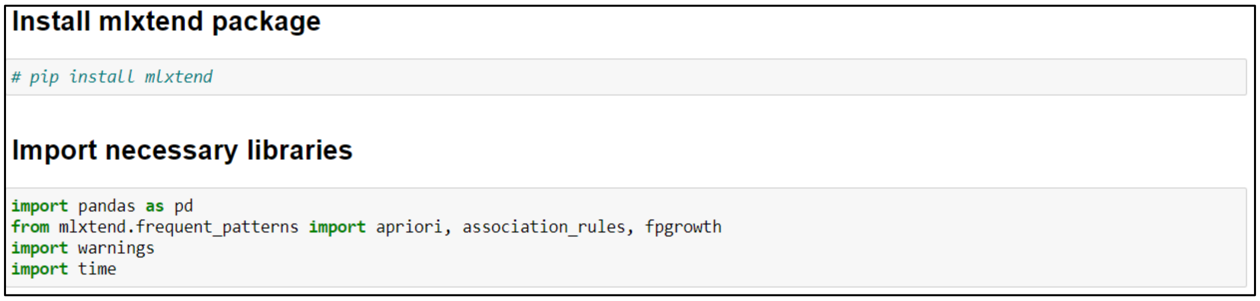
Download either the .ipynb file or .py file from GitHub and the five datasets provided in folders as csv files. Run the source code by having the datasets in the same directory. If you want to run .ipynb file use google colab or jupyter notebook. In case you prefer the .py file, it can be run using the command prompt.

(3) Provide user inputs

Upon execution, you will be asked to choose a dataset and provide minimum support and confidence. When user provides these values, the program will return the association rules from the three different strategies and their execution time.

CODE

Installing the mlxtend and Importing the necessary libraries



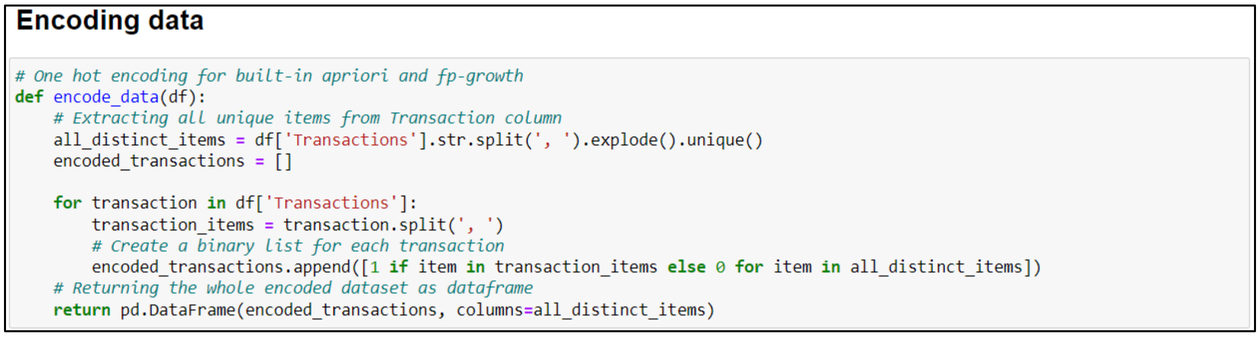
Install mlxtend package

```
# pip install mlxtend
```

Import necessary libraries

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
import warnings
import time
```

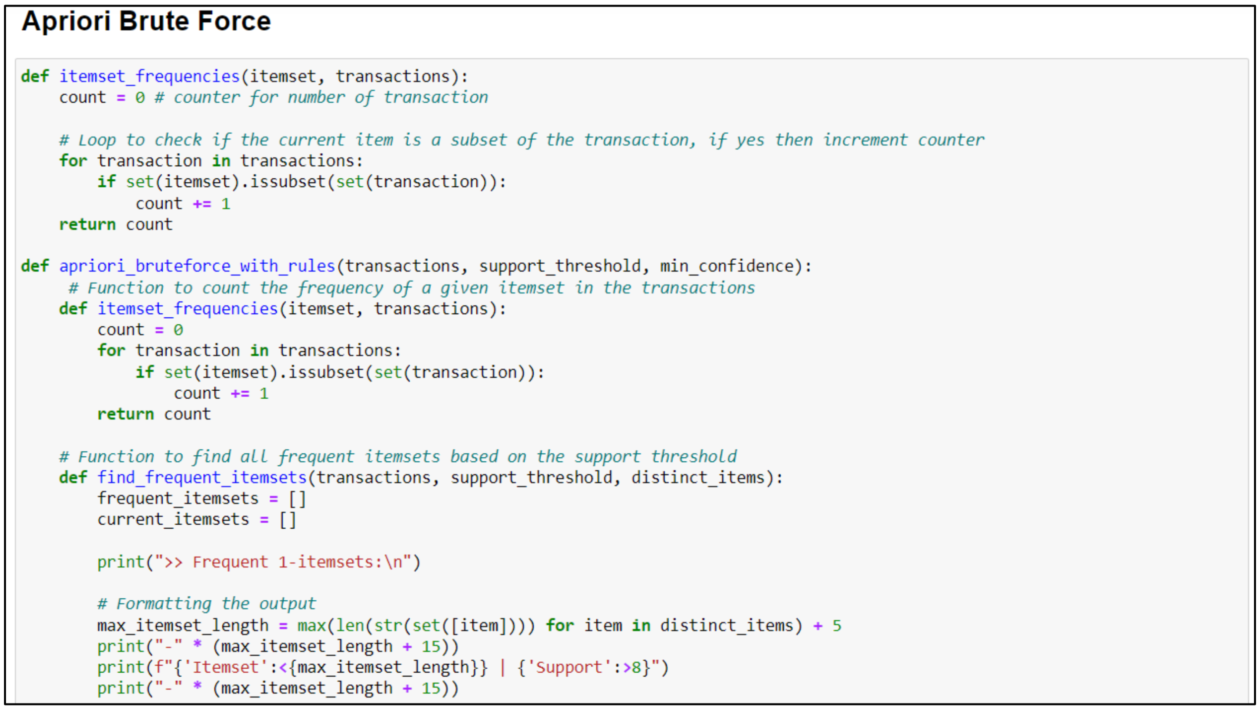
Preprocessing data (one-hot encoding for applying built-in functions)



Encoding data

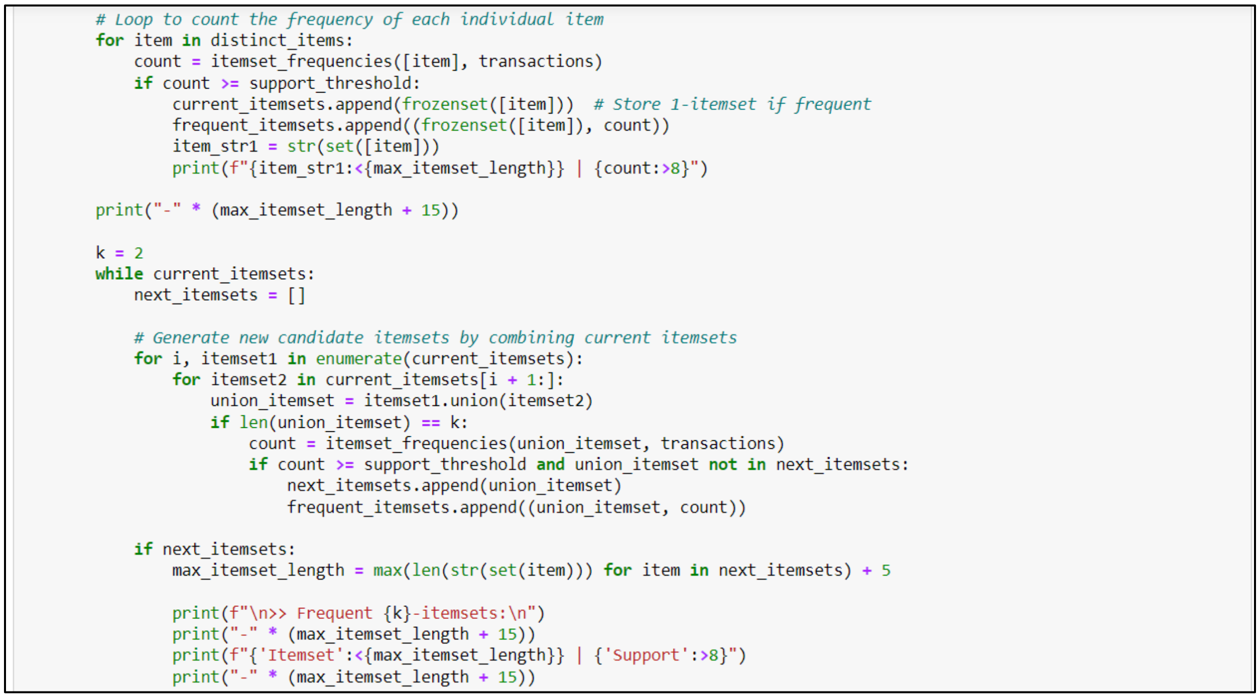
```python
# One hot encoding for built-in apriori and fp-growth
def encode_data(df):
    # Extracting all unique items from Transaction column
    all_distinct_items = df['Transactions'].str.split(', ').explode().unique()
    encoded_transactions = []

    for transaction in df['Transactions']:
        transaction_items = transaction.split(', ')
        # Create a binary list for each transaction
        encoded_transactions.append([1 if item in transaction_items else 0 for item in all_distinct_items])
    # Returning the whole encoded dataset as dataframe
    return pd.DataFrame(encoded_transactions, columns=all_distinct_items)
```

Function for implementing Apriori using a Brute force approach

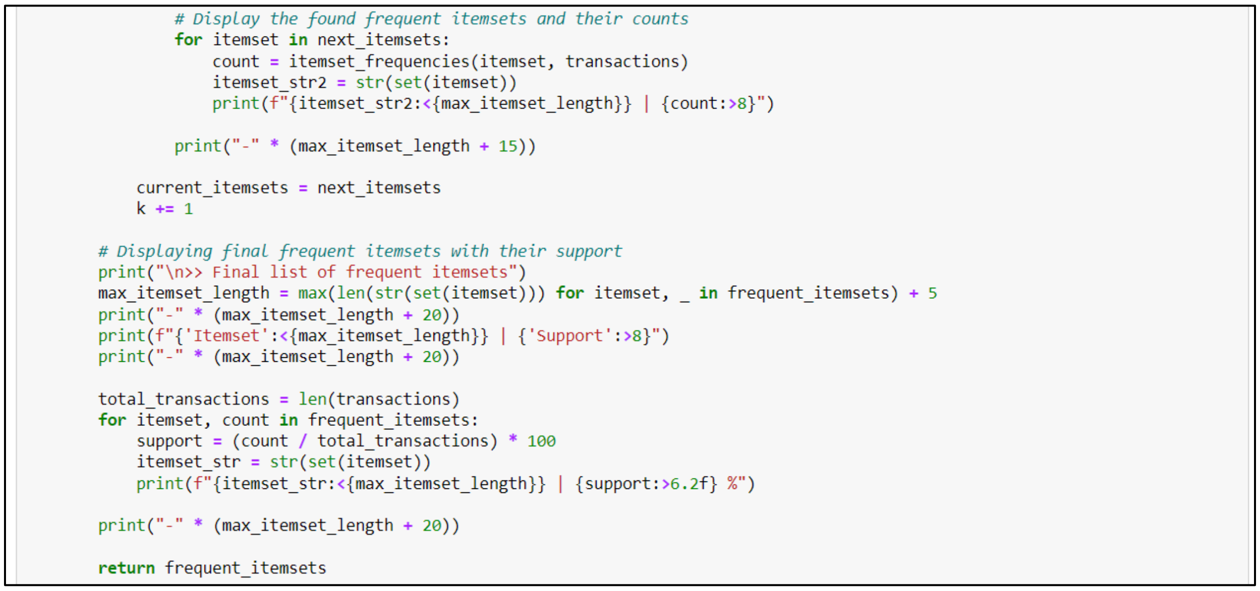


Apriori Brute Force

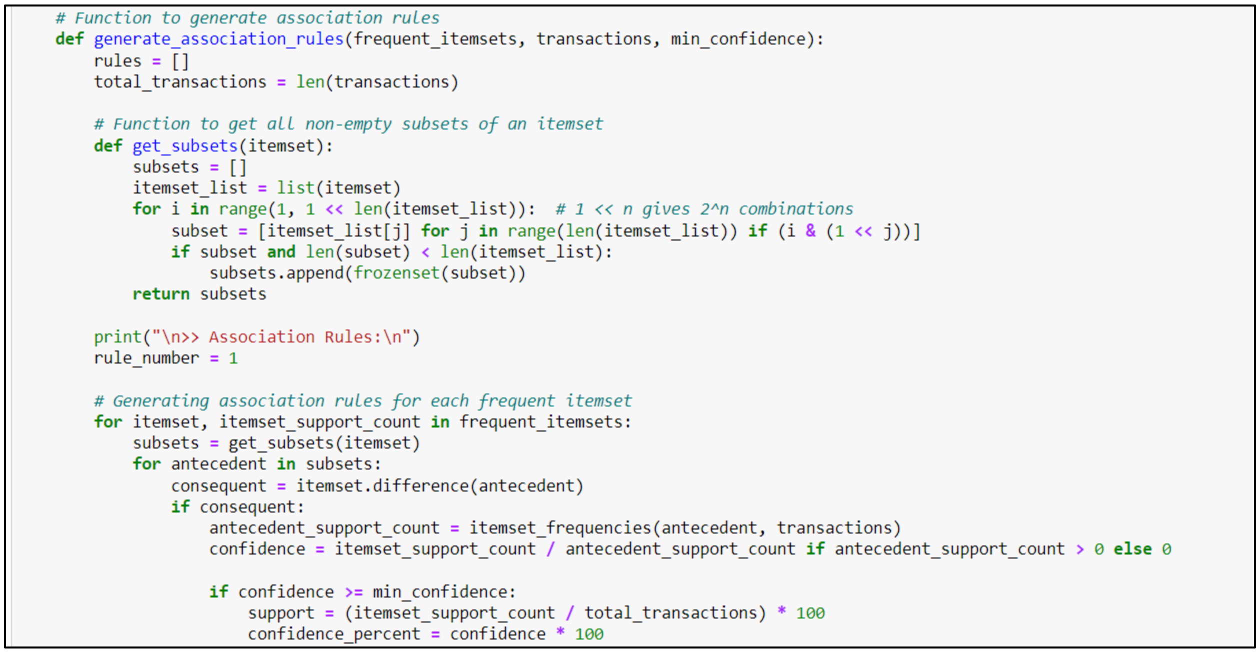
```python
def itemset_frequencies(itemset, transactions):
    count = 0 # counter for number of transaction

    # Loop to check if the current item is a subset of the transaction, if yes then increment counter
    for transaction in transactions:
        if set(itemset).issubset(set(transaction)):
            count += 1
    return count

def apriori_bruteforce_with_rules(transactions, support_threshold, min_confidence):
     # Function to count the frequency of a given itemset in the transactions
    def itemset_frequencies(itemset, transactions):
        count = 0
        for transaction in transactions:
            if set(itemset).issubset(set(transaction)):
                count += 1
        return count

    # Function to find all frequent itemsets based on the support threshold
    def find_frequent_itemsets(transactions, support_threshold, distinct_items):
        frequent_itemsets = []
        current_itemsets = []

        print(">> Frequent 1-itemsets:\n")

        # Formatting the output
        max_itemset_length = max(len(str(set([item]))) for item in distinct_items) + 5
        print("-" * (max_itemset_length + 15))
        print(f"{'Itemset':<{max_itemset_length}} | {'Support':>8}")
        print("-" * (max_itemset_length + 15))
```



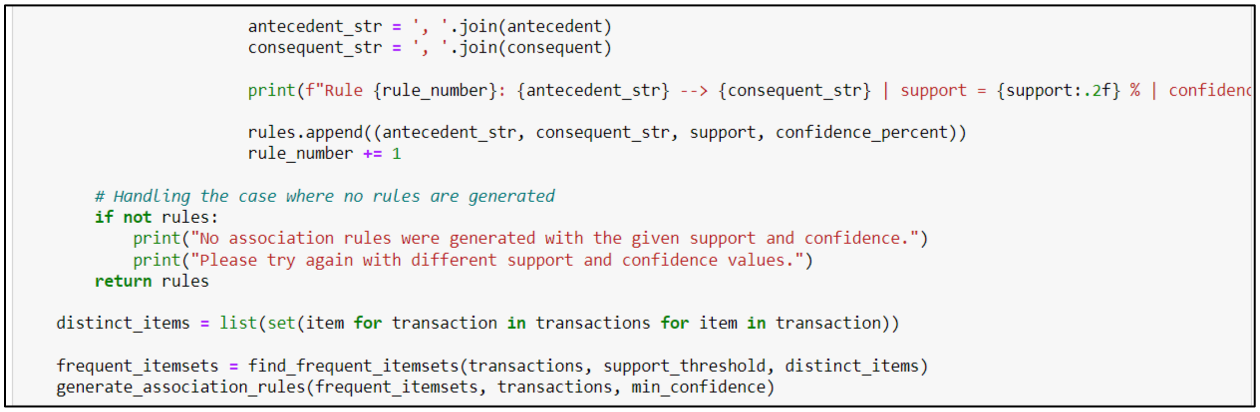
```python
    # Loop to count the frequency of each individual item
    for item in distinct_items:
        count = itemset_frequencies([item], transactions)
        if count >= support_threshold:
            current_itemsets.append(frozenset([item]))  # Store 1-itemset if frequent
            frequent_itemsets.append((frozenset([item]), count))
            item_str1 = str(set([item]))
            print(f"{item_str1:<{max_itemset_length}} | {count:>8}")

    print("-" * (max_itemset_length + 15))

    k = 2
    while current_itemsets:
        next_itemsets = []

        # Generate new candidate itemsets by combining current itemsets
        for i, itemset1 in enumerate(current_itemsets):
            for itemset2 in current_itemsets[i + 1:]:
                union_itemset = itemset1.union(itemset2)
                if len(union_itemset) == k:
                    count = itemset_frequencies(union_itemset, transactions)
                    if count >= support_threshold and union_itemset not in next_itemsets:
                        next_itemsets.append(union_itemset)
                        frequent_itemsets.append((union_itemset, count))

        if next_itemsets:
            max_itemset_length = max(len(str(set(item))) for item in next_itemsets) + 5

            print(f"\n>> Frequent {k}-itemsets:\n")
            print("-" * (max_itemset_length + 15))
            print(f"{'Itemset':<{max_itemset_length}} | {'Support':>8}")
            print("-" * (max_itemset_length + 15))
```



```python
            # Display the found frequent itemsets and their counts
            for itemset in next_itemsets:
                count = itemset_frequencies(itemset, transactions)
                itemset_str2 = str(set(itemset))
                print(f"{itemset_str2:<{max_itemset_length}} | {count:>8}")

            print("-" * (max_itemset_length + 15))

        current_itemsets = next_itemsets
        k += 1

    # Displaying final frequent itemsets with their support
    print("\n>> Final list of frequent itemsets")
    max_itemset_length = max(len(str(set(itemset))) for itemset, _ in frequent_itemsets) + 5
    print("-" * (max_itemset_length + 20))
    print(f"{'Itemset':<{max_itemset_length}} | {'Support':>8}")
    print("-" * (max_itemset_length + 20))

    total_transactions = len(transactions)
    for itemset, count in frequent_itemsets:
        support = (count / total_transactions) * 100
        itemset_str = str(set(itemset))
        print(f"{itemset_str:<{max_itemset_length}} | {support:>6.2f} %")

    print("-" * (max_itemset_length + 20))

    return frequent_itemsets
```

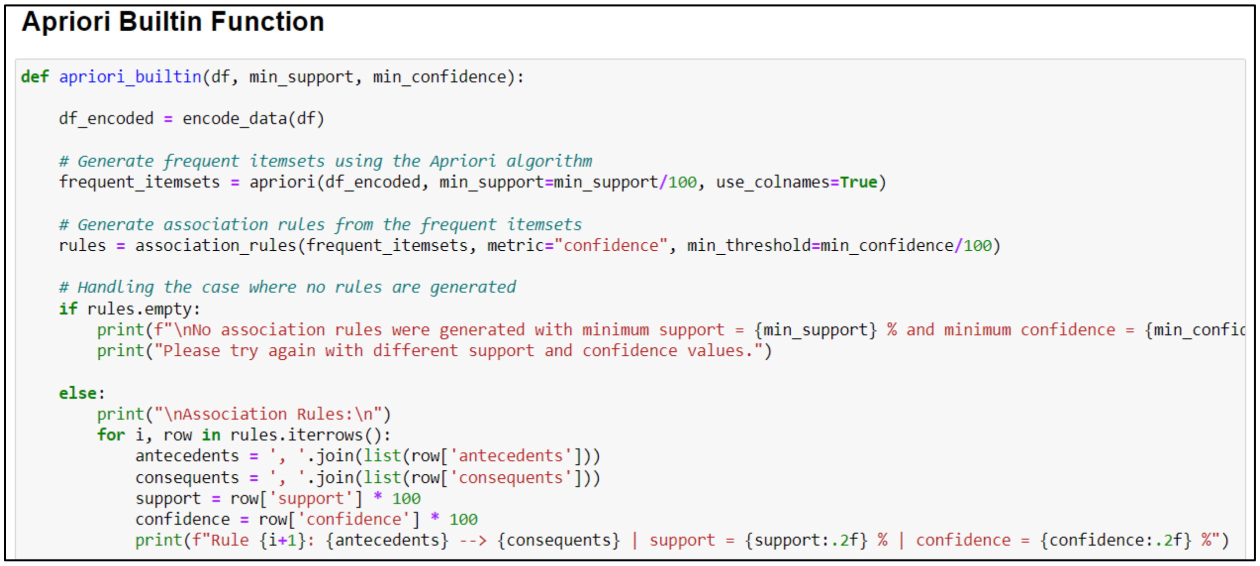


```python
# Function to generate association rules
def generate_association_rules(frequent_itemsets, transactions, min_confidence):
    rules = []
    total_transactions = len(transactions)

    # Function to get all non-empty subsets of an itemset
    def get_subsets(itemset):
        subsets = []
        itemset_list = list(itemset)
        for i in range(1, 1 << len(itemset_list)):  # 1 << n gives 2^n combinations
            subset = [itemset_list[j] for j in range(len(itemset_list)) if (i & (1 << j))]
            if subset and len(subset) < len(itemset_list):
                subsets.append(frozenset(subset))
        return subsets

    print("\n>> Association Rules:\n")
    rule_number = 1

    # Generating association rules for each frequent itemset
    for itemset, itemset_support_count in frequent_itemsets:
        subsets = get_subsets(itemset)
        for antecedent in subsets:
            consequent = itemset.difference(antecedent)
            if consequent:
                antecedent_support_count = itemset_frequencies(antecedent, transactions)
                confidence = itemset_support_count / antecedent_support_count if antecedent_support_count > 0 else 0

                if confidence >= min_confidence:
                    support = (itemset_support_count / total_transactions) * 100
                    confidence_percent = confidence * 100
```



```python
                    antecedent_str = ', '.join(antecedent)
                    consequent_str = ', '.join(consequent)

                    print(f"Rule {rule_number}: {antecedent_str} --> {consequent_str} | support = {support:.2f} % | confidenc

                    rules.append((antecedent_str, consequent_str, support, confidence_percent))
                    rule_number += 1

    # Handling the case where no rules are generated
    if not rules:
        print("No association rules were generated with the given support and confidence.")
        print("Please try again with different support and confidence values.")
    return rules

distinct_items = list(set(item for transaction in transactions for item in transaction))

frequent_itemsets = find_frequent_itemsets(transactions, support_threshold, distinct_items)
generate_association_rules(frequent_itemsets, transactions, min_confidence)
```

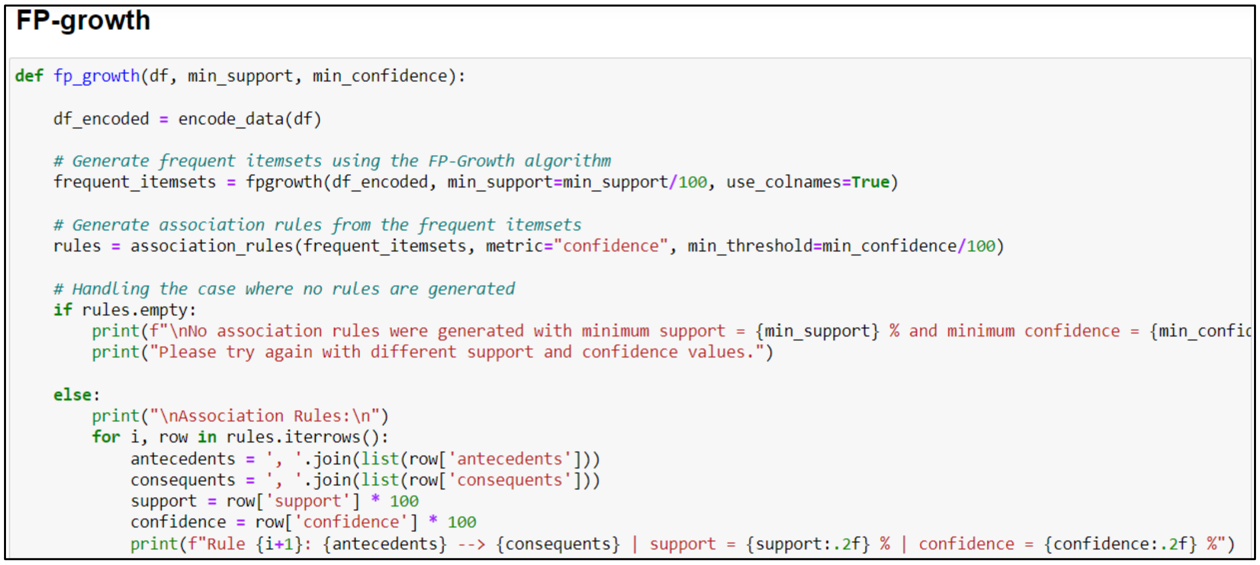
Function for implementing Apriori using built-in method (from mlxtend library)



Apriori Builtin Function

```python
def apriori_builtin(df, min_support, min_confidence):

    df_encoded = encode_data(df)

    # Generate frequent itemsets using the Apriori algorithm
    frequent_itemsets = apriori(df_encoded, min_support=min_support/100, use_colnames=True)

    # Generate association rules from the frequent itemsets
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence/100)

    # Handling the case where no rules are generated
    if rules.empty:
        print(f"\nNo association rules were generated with minimum support = {min_support} % and minimum confidence = {min_confic
        print("Please try again with different support and confidence values.")

    else:
        print("\nAssociation Rules:\n")
        for i, row in rules.iterrows():
            antecedents = ', '.join(list(row['antecedents']))
            consequents = ', '.join(list(row['consequents']))
            support = row['support'] * 100
            confidence = row['confidence'] * 100
            print(f"Rule {i+1}: {antecedents} --> {consequents} | support = {support:.2f} % | confidence = {confidence:.2f} %")
```

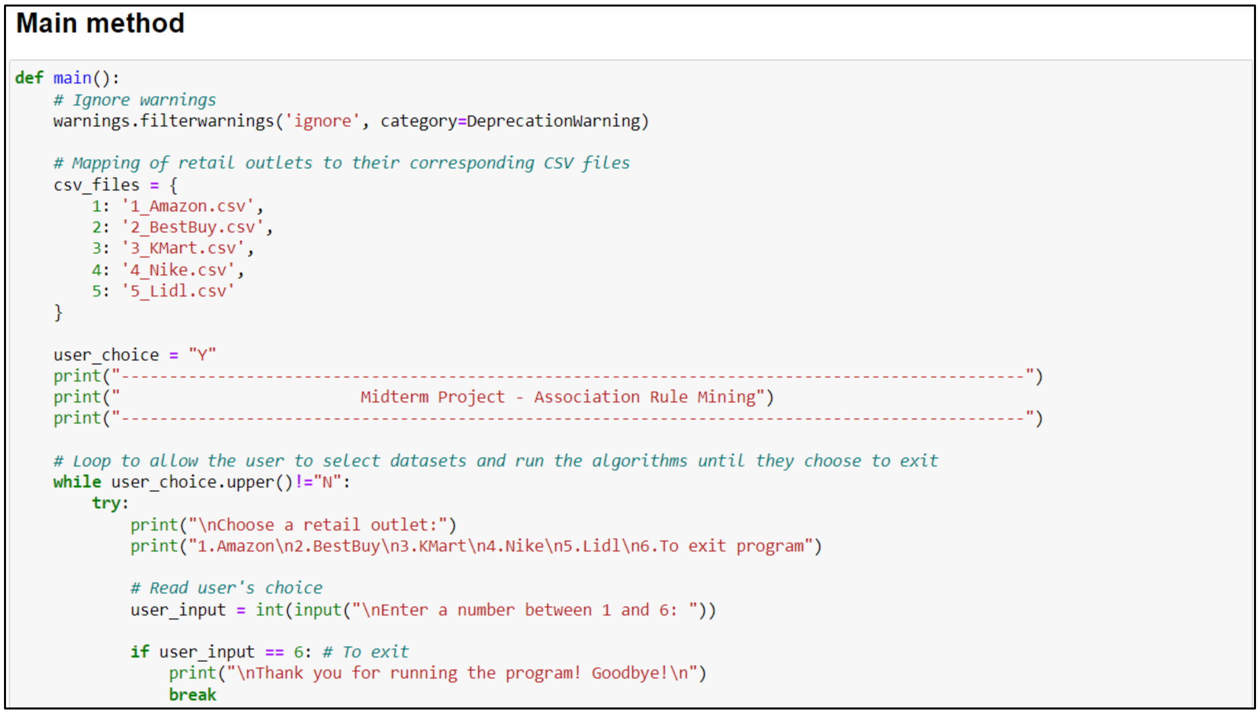
Function for implementing FP-Growth algorithm using built-in method (from mlxtend library)



FP-growth

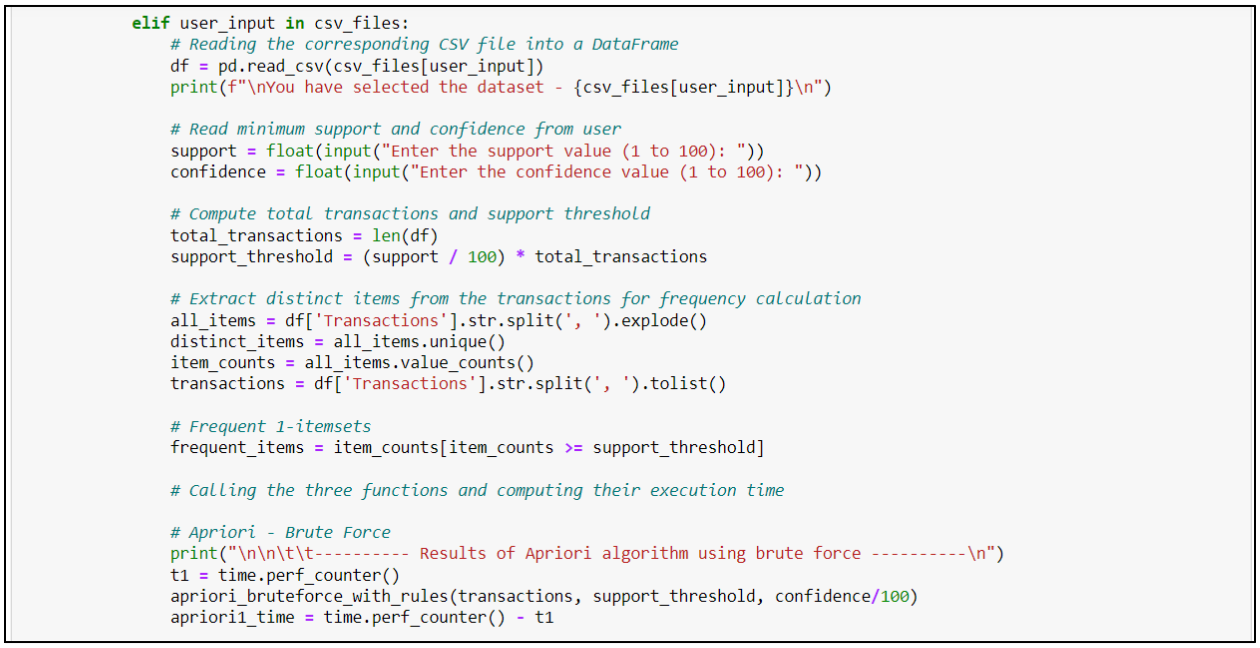
```python
def fp_growth(df, min_support, min_confidence):

    df_encoded = encode_data(df)

    # Generate frequent itemsets using the FP-Growth algorithm
    frequent_itemsets = fpgrowth(df_encoded, min_support=min_support/100, use_colnames=True)

    # Generate association rules from the frequent itemsets
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence/100)

    # Handling the case where no rules are generated
    if rules.empty:
        print(f"\nNo association rules were generated with minimum support = {min_support} % and minimum confidence = {min_confic
        print("Please try again with different support and confidence values.")

    else:
        print("\nAssociation Rules:\n")
        for i, row in rules.iterrows():
            antecedents = ', '.join(list(row['antecedents']))
            consequents = ', '.join(list(row['consequents']))
            support = row['support'] * 100
            confidence = row['confidence'] * 100
            print(f"Rule {i+1}: {antecedents} --> {consequents} | support = {support:.2f} % | confidence = {confidence:.2f} %")
```

Main method to call the functions from above to obtain association rules and compute the execution time.

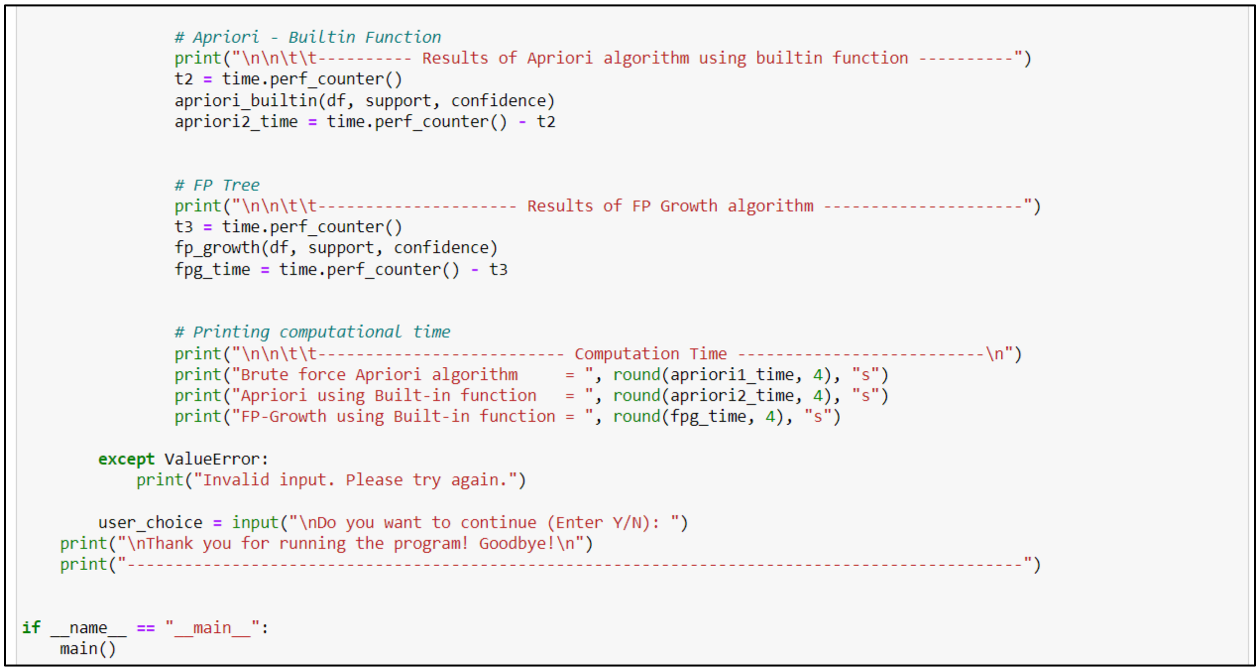


Main method

```python
def main():
    # Ignore warnings
    warnings.filterwarnings('ignore', category=DeprecationWarning)

    # Mapping of retail outlets to their corresponding CSV files
    csv_files = {
        1: '1_Amazon.csv',
        2: '2_BestBuy.csv',
        3: '3_KMart.csv',
        4: '4_Nike.csv',
        5: '5_Lidl.csv'
    }

    user_choice = "Y"
    print("----------------------------------------------------------------------------------------------")
    print("                         Midterm Project - Association Rule Mining")
    print("----------------------------------------------------------------------------------------------")

    # Loop to allow the user to select datasets and run the algorithms until they choose to exit
    while user_choice.upper()!="N":
        try:
            print("\nChoose a retail outlet:")
            print("1.Amazon\n2.BestBuy\n3.KMart\n4.Nike\n5.Lidl\n6.To exit program")

            # Read user's choice
            user_input = int(input("\nEnter a number between 1 and 6: "))

            if user_input == 6: # To exit
                print("\nThank you for running the program! Goodbye!\n")
                break
```



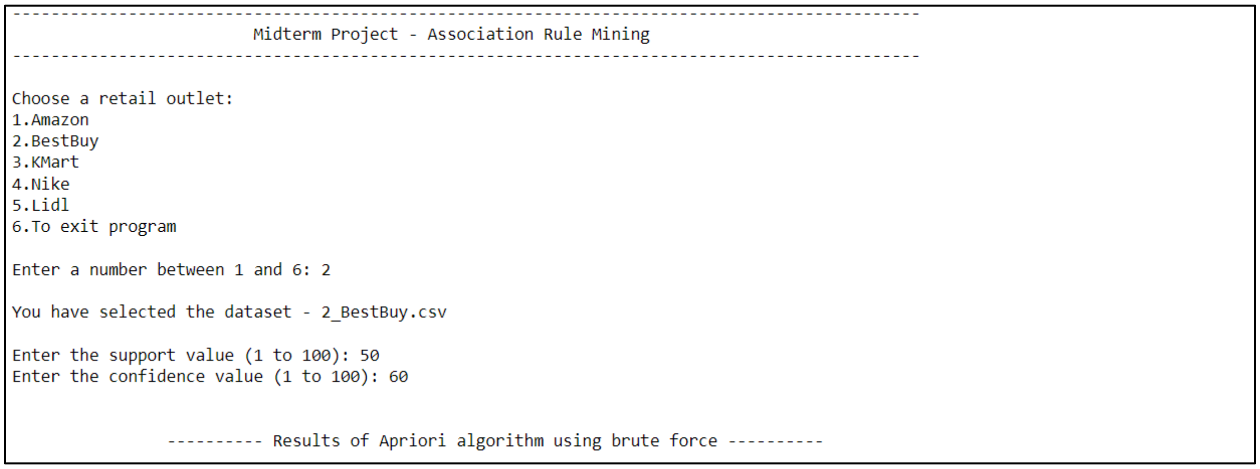
```
            elif user_input in csv_files:
                # Reading the corresponding CSV file into a DataFrame
                df = pd.read_csv(csv_files[user_input])
                print(f"\nYou have selected the dataset - {csv_files[user_input]}\n")

                # Read minimum support and confidence from user
                support = float(input("Enter the support value (1 to 100): "))
                confidence = float(input("Enter the confidence value (1 to 100): "))

                # Compute total transactions and support threshold
                total_transactions = len(df)
                support_threshold = (support / 100) * total_transactions

                # Extract distinct items from the transactions for frequency calculation
                all_items = df['Transactions'].str.split(', ').explode()
                distinct_items = all_items.unique()
                item_counts = all_items.value_counts()
                transactions = df['Transactions'].str.split(', ').tolist()

                # Frequent 1-itemsets
                frequent_items = item_counts[item_counts >= support_threshold]

                # Calling the three functions and computing their execution time

                # Apriori - Brute Force
                print("\n\n\t\t---------- Results of Apriori algorithm using brute force ----------\n")
                t1 = time.perf_counter()
                apriori_bruteforce_with_rules(transactions, support_threshold, confidence/100)
                apriori1_time = time.perf_counter() - t1
```



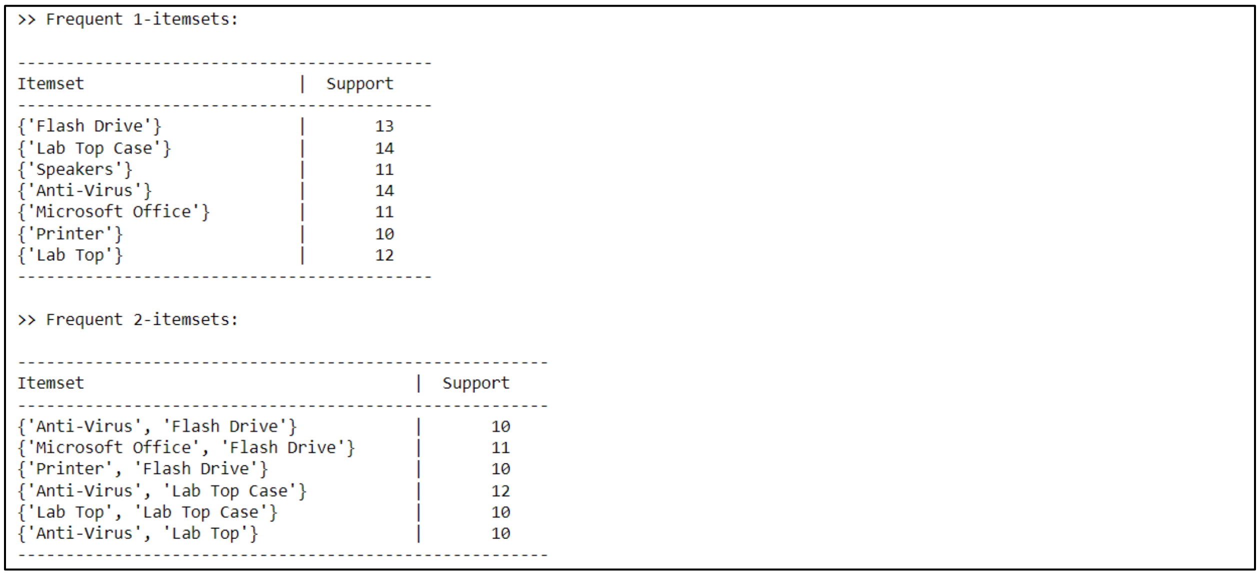
```
                # Apriori - Builtin Function
                print("\n\n\t\t---------- Results of Apriori algorithm using builtin function ----------")
                t2 = time.perf_counter()
                apriori_builtin(df, support, confidence)
                apriori2_time = time.perf_counter() - t2


                # FP Tree
                print("\n\n\t\t-------------------- Results of FP Growth algorithm --------------------")
                t3 = time.perf_counter()
                fp_growth(df, support, confidence)
                fpg_time = time.perf_counter() - t3


                # Printing computational time
                print("\n\n\t\t------------------------- Computation Time -------------------------\n")
                print("Brute force Apriori algorithm     = ", round(apriori1_time, 4), "s")
                print("Apriori using Built-in function   = ", round(apriori2_time, 4), "s")
                print("FP-Growth using Built-in function = ", round(fpg_time, 4), "s")

        except ValueError:
            print("Invalid input. Please try again.")

        user_choice = input("\nDo you want to continue (Enter Y/N): ")
    print("\nThank you for running the program! Goodbye!\n")
    print("---------------------------------------------------------------------------------------------")


if __name__ == "__main__":
    main()
```

RESULTS AND EVALUATION

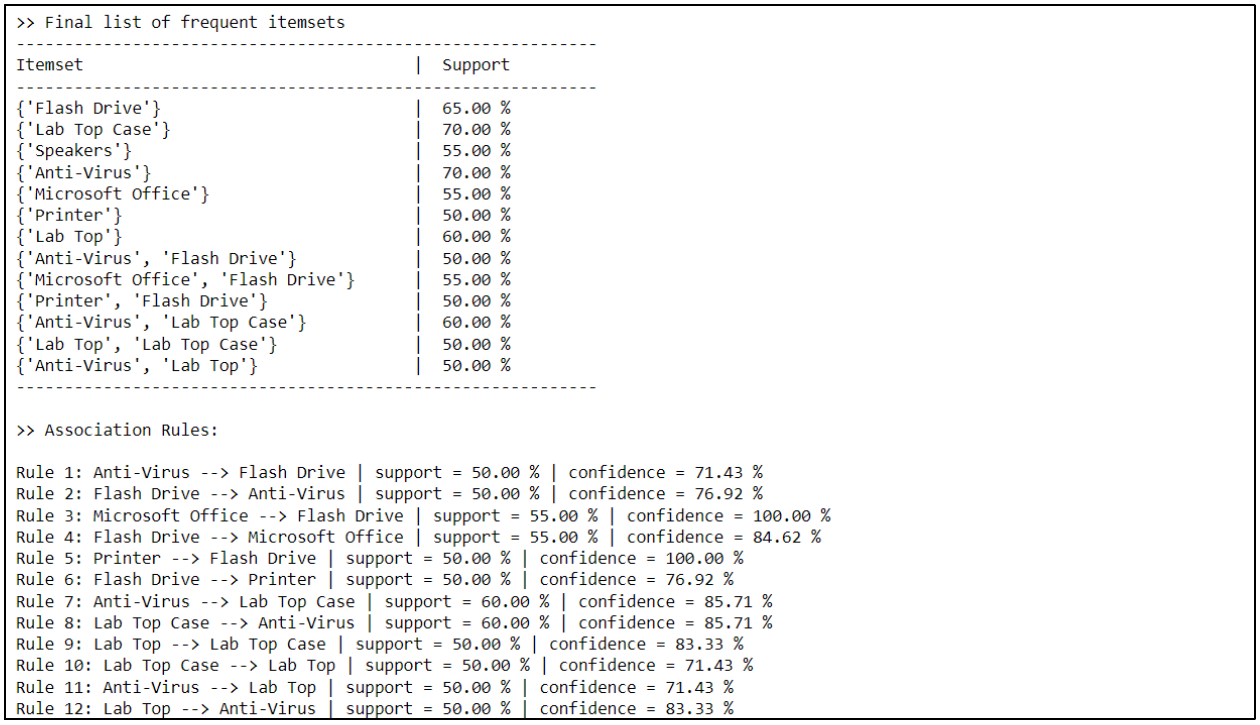
(1) Output screenshots



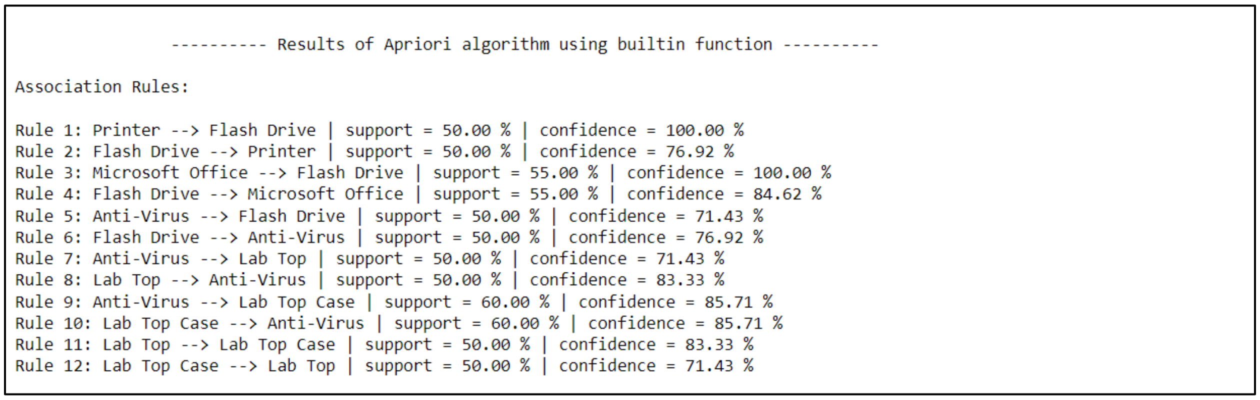
```
---------------------------------------------------------------------------------------------
                         Midterm Project - Association Rule Mining
---------------------------------------------------------------------------------------------

Choose a retail outlet:
1.Amazon
2.BestBuy
3.KMart
4.Nike
5.Lidl
6.To exit program

Enter a number between 1 and 6: 2

You have selected the dataset - 2_BestBuy.csv

Enter the support value (1 to 100): 50
Enter the confidence value (1 to 100): 60


                ---------- Results of Apriori algorithm using brute force ----------
```



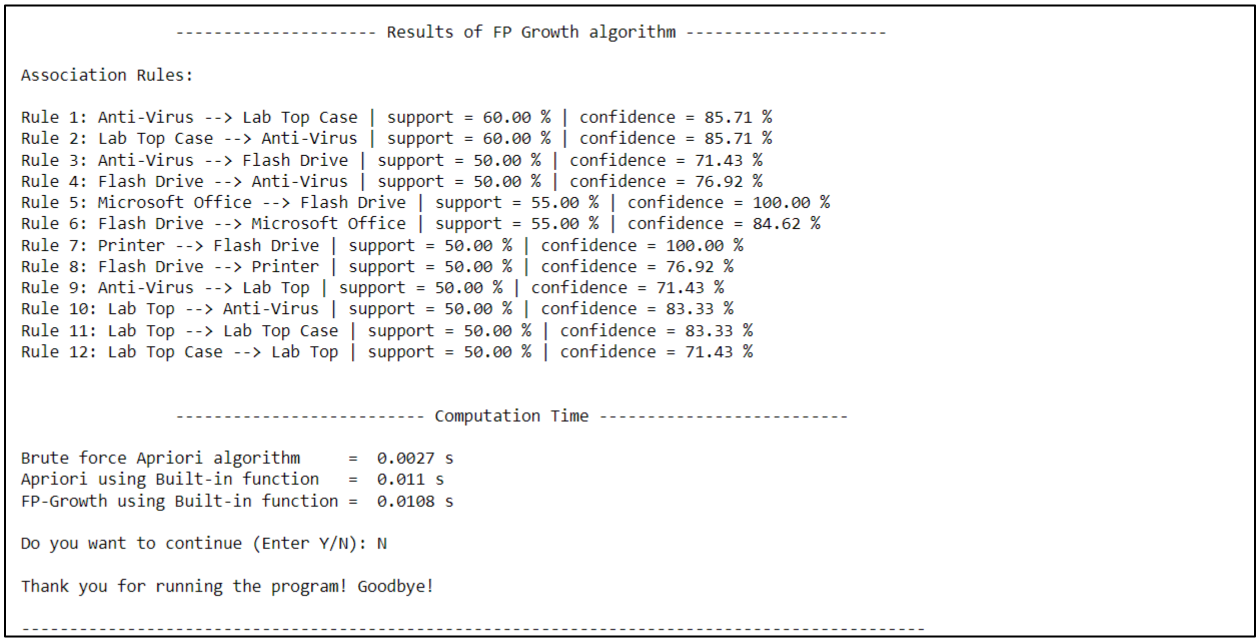
```
>> Frequent 1-itemsets:

-------------------------------------------
Itemset                       |  Support
-------------------------------------------
{'Flash Drive'}               |       13
{'Lab Top Case'}              |       14
{'Speakers'}                  |       11
{'Anti-Virus'}                |       14
{'Microsoft Office'}          |       11
{'Printer'}                   |       10
{'Lab Top'}                   |       12
-------------------------------------------

>> Frequent 2-itemsets:

-------------------------------------------------------
Itemset                                   |  Support
-------------------------------------------------------
{'Anti-Virus', 'Flash Drive'}             |       10
{'Microsoft Office', 'Flash Drive'}       |       11
{'Printer', 'Flash Drive'}                |       10
{'Anti-Virus', 'Lab Top Case'}            |       12
{'Lab Top', 'Lab Top Case'}               |       10
{'Anti-Virus', 'Lab Top'}                 |       10
-------------------------------------------------------
```



```
>> Final list of frequent itemsets
------------------------------------------------------------
Itemset                                  |  Support
------------------------------------------------------------
{'Flash Drive'}                          |  65.00 %
{'Lab Top Case'}                         |  70.00 %
{'Speakers'}                             |  55.00 %
{'Anti-Virus'}                           |  70.00 %
{'Microsoft Office'}                     |  55.00 %
{'Printer'}                              |  50.00 %
{'Lab Top'}                              |  60.00 %
{'Anti-Virus', 'Flash Drive'}            |  50.00 %
{'Microsoft Office', 'Flash Drive'}      |  55.00 %
{'Printer', 'Flash Drive'}               |  50.00 %
{'Anti-Virus', 'Lab Top Case'}           |  60.00 %
{'Lab Top', 'Lab Top Case'}              |  50.00 %
{'Anti-Virus', 'Lab Top'}                |  50.00 %
------------------------------------------------------------

>> Association Rules:

Rule 1: Anti-Virus --> Flash Drive | support = 50.00 % | confidence = 71.43 %
Rule 2: Flash Drive --> Anti-Virus | support = 50.00 % | confidence = 76.92 %
Rule 3: Microsoft Office --> Flash Drive | support = 55.00 % | confidence = 100.00 %
Rule 4: Flash Drive --> Microsoft Office | support = 55.00 % | confidence = 84.62 %
Rule 5: Printer --> Flash Drive | support = 50.00 % | confidence = 100.00 %
Rule 6: Flash Drive --> Printer | support = 50.00 % | confidence = 76.92 %
Rule 7: Anti-Virus --> Lab Top Case | support = 60.00 % | confidence = 85.71 %
Rule 8: Lab Top Case --> Anti-Virus | support = 60.00 % | confidence = 85.71 %
Rule 9: Lab Top --> Lab Top Case | support = 50.00 % | confidence = 83.33 %
Rule 10: Lab Top Case --> Lab Top | support = 50.00 % | confidence = 71.43 %
Rule 11: Anti-Virus --> Lab Top | support = 50.00 % | confidence = 71.43 %
Rule 12: Lab Top --> Anti-Virus | support = 50.00 % | confidence = 83.33 %
```



```
                ---------- Results of Apriori algorithm using builtin function ----------

Association Rules:

Rule 1: Printer --> Flash Drive | support = 50.00 % | confidence = 100.00 %
Rule 2: Flash Drive --> Printer | support = 50.00 % | confidence = 76.92 %
Rule 3: Microsoft Office --> Flash Drive | support = 55.00 % | confidence = 100.00 %
Rule 4: Flash Drive --> Microsoft Office | support = 55.00 % | confidence = 84.62 %
Rule 5: Anti-Virus --> Flash Drive | support = 50.00 % | confidence = 71.43 %
Rule 6: Flash Drive --> Anti-Virus | support = 50.00 % | confidence = 76.92 %
Rule 7: Anti-Virus --> Lab Top | support = 50.00 % | confidence = 71.43 %
Rule 8: Lab Top --> Anti-Virus | support = 50.00 % | confidence = 83.33 %
Rule 9: Anti-Virus --> Lab Top Case | support = 60.00 % | confidence = 85.71 %
Rule 10: Lab Top Case --> Anti-Virus | support = 60.00 % | confidence = 85.71 %
Rule 11: Lab Top --> Lab Top Case | support = 50.00 % | confidence = 83.33 %
Rule 12: Lab Top Case --> Lab Top | support = 50.00 % | confidence = 71.43 %
```



```
                -------------------- Results of FP Growth algorithm --------------------

Association Rules:

Rule 1: Anti-Virus --> Lab Top Case | support = 60.00 % | confidence = 85.71 %
Rule 2: Lab Top Case --> Anti-Virus | support = 60.00 % | confidence = 85.71 %
Rule 3: Anti-Virus --> Flash Drive | support = 50.00 % | confidence = 71.43 %
Rule 4: Flash Drive --> Anti-Virus | support = 50.00 % | confidence = 76.92 %
Rule 5: Microsoft Office --> Flash Drive | support = 55.00 % | confidence = 100.00 %
Rule 6: Flash Drive --> Microsoft Office | support = 55.00 % | confidence = 84.62 %
Rule 7: Printer --> Flash Drive | support = 50.00 % | confidence = 100.00 %
Rule 8: Flash Drive --> Printer | support = 50.00 % | confidence = 76.92 %
Rule 9: Anti-Virus --> Lab Top | support = 50.00 % | confidence = 71.43 %
Rule 10: Lab Top --> Anti-Virus | support = 50.00 % | confidence = 83.33 %
Rule 11: Lab Top --> Lab Top Case | support = 50.00 % | confidence = 83.33 %
Rule 12: Lab Top Case --> Lab Top | support = 50.00 % | confidence = 71.43 %


                ------------------------- Computation Time -------------------------

Brute force Apriori algorithm     =  0.0027 s
Apriori using Built-in function   =  0.011 s
FP-Growth using Built-in function =  0.0108 s

Do you want to continue (Enter Y/N): N

Thank you for running the program! Goodbye!

--------------------------------------------------------------------------------------------
```

(2) Inference

The association rules generated by all three methods were consistent, producing identical frequent itemsets, which is a positive outcome. However, one notable concern in my implementation is related to execution time. Typically, brute force methods are expected to take the longest, with FP-Growth being the most efficient. Surprisingly, in my tests, the brute force method yielded the fastest results, followed by FP-Growth and then the Apriori algorithm using the built-in function. This pattern was consistent across various datasets with different support and confidence levels. While this anomaly can sometimes occur with smaller datasets, I attempted to address it by using a larger dataset (Lidl – 40 transactions), but the results remained the same. It is possible that with even larger datasets, FP-Growth might demonstrate its expected superiority in terms of execution time.

CONCLUSION

In conclusion, this project highlights the efficiency of various association rule mining techniques, including the brute-force Apriori algorithm, Apriori algorithm using built-in functions and FP-Growth. By analyzing diverse datasets from retail outlets such as Amazon, Best Buy, and others, this work demonstrated how these methods can identify frequent itemsets and generate meaningful association rules. The three strategies provided the same frequent items and association rules thereby verifying their correctness. Although, the outcome is not what I intended to obtain, the brute force method provides the fastest execution time followed by FP-Growth and Built-in Apriori respectively. This is something that can be investigated as a part of further improvements to this project. The analysis of transactional data has diverse applications and offers profitability to most businesses in multiple industries and thereby making this project relevant.

SOURCE CODE AND REPOSITORY

The source code .py file, Jupyter Notebook .ipynb file and datasets .csv files are attached to the zip file.

Link to GitHub repository: https://github.com/kaviyasree0103/Association-Rules---Midterm-Project.git