NAME: KAUSHIK KUMAR KOLAR RAVINDRA KUMAR

NJIT UCID: kk795

Email Address: kk795@njit.edu

13th October, 2024.

Professor: Yasser Abduallah CS 634 101 Data Mining

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm Implementation in Retail Data Mining

Abstract:

This project implements and compares three algorithms—Brute Force, Apriori, and FP-Growth—for generating frequent itemsets and association rules from transactional datasets. Using five databases of transactions from major companies, the Brute Force method exhaustively generates and checks all possible itemsets, starting from 1-itemsets and continuing until no further frequent sets are found. The Apriori and FP-Growth algorithms, implemented using Python libraries, are employed to verify and optimize the results. User-defined support and confidence thresholds are applied to generate multiple sets of association rules, and the performance of all three algorithms is evaluated in terms of accuracy and execution time. This comparison highlights the efficiency gains of Apriori and FP-Growth over the brute force approach.

Introduction:

Frequent itemset mining and association rule generation are fundamental tasks in data mining, widely applied in various industries, including retail, banking, and finance, to uncover patterns and relationships in transactional data. This project focuses on implementing and comparing three key algorithms—Brute Force, Apriori, and FP-Growth (also known as FP-Tree)—to generate frequent itemsets and derive association rules from transactional databases. The datasets used in this project contain transactions from five major companies: Amazon, Costco, Nike, Lewis, and Walmart.

The brute force method, while exhaustive, provides a baseline by enumerating all possible itemsets, starting with 1-itemsets and proceeding until no more frequent itemsets can be found. Although accurate, this approach is computationally expensive for larger datasets. To address this inefficiency, optimized algorithms like Apriori and FP-Growth are employed. Apriori reduces the number of candidate itemsets by leveraging the anti-monotonicity property, while FP-Growth compresses data into a prefix tree to efficiently mine frequent patterns.

This project not only implements these algorithms but also compares their performance in terms of result accuracy, runtime, and scalability. By using user-defined support and confidence levels, the project demonstrates how different thresholds affect the association rules generated for each dataset. Ultimately, this comparison highlights the trade-offs between brute force and more optimized approaches in real-world data mining applications.

Key steps in the implementation include:

1. **Data Preparation :** Creating transactional datasets for five companies (Amazon, Costco, Nike, Lewis, and Walmart), ensuring each dataset contains at least 20 transactions of frequently purchased items.

2. **Brute Force Implementation :** Developing an algorithm to enumerate all possible itemsets, starting from 1-itemsets, followed by 2-itemsets, and so on. For each (k)-itemset, the algorithm checks its frequency against a user-specified support threshold until no more frequent (k)-itemsets are found.

3. **Apriori Algorithm :** Utilizing an existing Python implementation of the Apriori algorithm to generate frequent itemsets and association rules. The Apriori method prunes infrequent itemsets early, making it more efficient than the brute force approach.

4. **FP-Growth Algorithm :** Implementing the FP-Growth algorithm using a Python package to compress the transactional data into a prefix tree structure, allowing for faster frequent itemset generation.

5. **Association Rule Generation :** Generating association rules from the frequent itemsets for all three algorithms, using user-specified support and confidence values.

6. **Performance Comparison :** Measuring and comparing the execution time and output consistency of the Brute Force, Apriori, and FP-Growth algorithms. The timing performance is recorded for different support and confidence levels across all datasets.

7. **Result Analysis :** Evaluating whether the three algorithms produce identical results in terms of frequent itemsets and association rules, and identifying which algorithm performs fastest for each dataset.

Core Concepts and Principles:

1. **Frequent Itemset Discovery :**

- The project involves identifying frequent itemsets from the transactional data of five companies using three algorithms—Brute Force, Apriori, and FP-Growth.
- Frequent itemsets represent combinations of items that appear frequently in the transactions, providing insights into patterns of item co-occurrence.

2. **Support and Confidence :**

- Support is used to filter out infrequent itemsets, ensuring only frequently occurring ones are considered. The project allows user-specified support thresholds for each dataset.
- Confidence is applied to evaluate association rules, determining the likelihood of items being purchased together. The project uses varying confidence levels to generate multiple association rules and analyze their significance.

3. **Association Rule Generation :**

- Association rules are generated from the frequent itemsets discovered by all three algorithms. These rules show relationships between items frequently purchased together, helping understand customer behavior.
- Strong association rules derived from the dataset are analyzed to offer potential recommendations for optimizing business strategies, such as product bundling or targeted marketing.

4. **Algorithm Comparison :**

- Brute Force method exhaustively generates all possible itemsets but is computationally expensive.
- Apriori and FP-Growth algorithms reduce computational overhead by leveraging optimizations, such as pruning and tree-based structures, allowing more efficient discovery of frequent itemsets and association rules.

5. **Performance Evaluation :**

- The project compares the algorithms in terms of runtime and accuracy, demonstrating how different approaches yield consistent results but with varying levels of efficiency.

Project Workflow:

1. **Data Loading and Preprocessing :**

- Transactional datasets from five companies (Amazon, Costco, Nike, Lewis, Walmart) are loaded into the system.
- The data is preprocessed by filtering unique items in each transaction and sorting them based on a predefined order. This ensures consistency and accuracy across all datasets before frequent itemset mining begins.

2. **Determination of Minimum Support and Confidence :**

- User input is collected for setting the minimum support and confidence thresholds. These values dictate which frequent itemsets and association rules are retained.
- By allowing flexible support and confidence levels, the user can explore varying insights from the same dataset.

3. **Iteration Through Candidate Itemsets :**

- For each algorithm (Brute Force, Apriori, and FP-Growth), itemset generation begins with single items (K = 1) and iteratively increases to K = 2, K = 3, and so on.
- In the Brute Force approach, all possible itemset combinations are generated and checked exhaustively. For Apriori, candidate sets are pruned based on previously generated frequent itemsets, while FP-Growth uses a compressed prefix-tree to reduce redundant calculations.

4. **Support Count Calculation :**

- Each candidate itemset's support is calculated by counting its occurrences across all transactions.
- Itemsets that meet the user-defined minimum support threshold are retained, while those that do not are discarded.

5. **Confidence Calculation :**

- After determining frequent itemsets, the confidence for each association rule is computed by comparing the support of the combined itemset with the support of the antecedent item.
- Only rules that meet the minimum confidence threshold are retained for further analysis.

6. **Association Rule Generation :**

- Based on the frequent itemsets, association rules are generated for each of the three algorithms. These rules are filtered by the user-specified support and confidence values.
- The generated rules reveal which items are frequently purchased together, providing actionable insights into customer purchasing behavior.

7. **Performance Evaluation :**

- The timing and efficiency of Brute Force, Apriori, and FP-Growth algorithms are compared for each dataset.
- The results, including execution time and generated association rules, are analyzed to determine the most efficient approach for different support and confidence thresholds.

Results and Evaluation :

The project successfully implemented and compared three algorithms—Brute Force, Apriori, and FP-Growth—for frequent itemset mining and association rule generation using transactional datasets from five major companies. Each algorithm generated consistent association rules, confirming their effectiveness in identifying item relationships. The Brute Force method, while exhaustive, provided a benchmark for accuracy, while the Apriori and FP-Growth algorithms demonstrated significantly improved performance in terms of execution time, making them more suitable for larger datasets. The results highlight the practical applications of these algorithms in enhancing sales strategies and customer insights, showcasing the project's success in delivering valuable findings through efficient data mining techniques.

Conclusion:

In conclusion, this project effectively demonstrated the application of Brute Force, Apriori, and FP-Growth algorithms in mining frequent itemsets and generating association rules from transactional datasets of five major companies. The findings revealed that while the Brute Force method provided accurate results, its computational inefficiency highlighted the advantages of using optimized algorithms like Apriori and FP-Growth. By comparing execution times and the quality of generated rules, the project showcased the importance of selecting appropriate data mining techniques for practical applications in retail and other industries. Ultimately, this work contributes valuable insights into customer purchasing behavior, emphasizing the role of data mining in driving informed business decisions and enhancing sales strategies.

Screenshots

The following are the data from the CSV files that are considered for the project :

1) Amazon

```
Displaying Items and Transactions for: Amazon

Loading data from data/Amazon.csv
Data loaded successfully from data/Amazon.csv

--- List of Items in All Transactions ---
```

	Items
1	Coffee Maker
2	Fitness Tracker
3	Headphones
4	Kitchen Mixer
5	Laptop
6	Phone Charger
7	Portable Charger
8	Smart Speaker
9	Wireless Mouse
10	Yoga Mat

```
--- List of Transactions ---
```

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Laptop	Headphones	Smart Speaker	NaN
2	Yoga Mat	Kitchen Mixer	Wireless Mouse	NaN
3	Fitness Tracker	Portable Charger	Headphones	Laptop
4	Smart Speaker	Phone Charger	Yoga Mat	Kitchen Mixer
5	Coffee Maker	Wireless Mouse	Fitness Tracker	NaN
6	Headphones	Laptop	Smart Speaker	Wireless Mouse
7	Yoga Mat	Kitchen Mixer	Coffee Maker	Fitness Tracker
8	Laptop	Wireless Mouse	Portable Charger	NaN
9	Phone Charger	Headphones	Yoga Mat	NaN
10	Coffee Maker	Fitness Tracker	Wireless Mouse	Laptop
11	Headphones	Smart Speaker	Yoga Mat	Wireless Mouse
12	Fitness Tracker	Portable Charger	NaN	NaN
13	Kitchen Mixer	Phone Charger	Smart Speaker	Wireless Mouse
14	Laptop	Coffee Maker	Yoga Mat	NaN
15	Wireless Mouse	Headphones	Smart Speaker	NaN
16	Yoga Mat	Kitchen Mixer	Laptop	NaN
17	Portable Charger	Coffee Maker	Fitness Tracker	NaN
18	Smart Speaker	Phone Charger	Wireless Mouse	NaN
19	Laptop	Yoga Mat	Fitness Tracker	NaN
20	Headphones	Coffee Maker	Smart Speaker	Wireless Mouse

2) Costco

```
Displaying Items and Transactions for: Costco

Loading data from data/Costco.csv
Data loaded successfully from data/Costco.csv

--- List of Items in All Transactions ---
```

	Items
1	Bulk Chicken
2	Cereal
3	Cheese
4	Eggs
5	Fresh Vegetables
6	Frozen Berries
7	Laundry Detergent
8	Olive Oil
9	Snacks
10	Toilet Paper

--- List of Transactions ---

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Bulk Chicken	Toilet Paper	Cereal	NaN
2	Snacks	Cheese	Eggs	Fresh Vegetables
3	Frozen Berries	Olive Oil	NaN	NaN
4	Cereal	Laundry Detergent	Snacks	NaN
5	Fresh Vegetables	Cheese	Bulk Chicken	NaN
6	Eggs	Frozen Berries	Snacks	NaN
7	Toilet Paper	Olive Oil	Fresh Vegetables	NaN
8	Cereal	Bulk Chicken	Laundry Detergent	NaN
9	Snacks	Cheese	Eggs	Frozen Berries
10	Laundry Detergent	Bulk Chicken	Toilet Paper	NaN
11	Fresh Vegetables	Cereal	Snacks	Cheese
12	Eggs	Frozen Berries	Toilet Paper	NaN
13	Cheese	Laundry Detergent	Snacks	NaN
14	Bulk Chicken	Eggs	Fresh Vegetables	NaN
15	Snacks	Cereal	Cheese	NaN
16	Frozen Berries	Olive Oil	Laundry Detergent	NaN
17	Eggs	Fresh Vegetables	Cereal	NaN
18	Cheese	Toilet Paper	Bulk Chicken	NaN
19	Snacks	Eggs	Frozen Berries	NaN
20	Laundry Detergent	Fresh Vegetables	Cereal	NaN

3) Levis

```
Displaying Items and Transactions for: Levis

Loading data from data/Levis.csv
Data loaded successfully from data/Levis.csv

--- List of Items in All Transactions ---
```

	Items
1	Beanie
2	Dress
3	Flip Flops
4	Handbag
5	Hoodie
6	Jacket
7	Jeans
8	Sandals
9	Scarf
10	Shorts
11	Sneakers
12	Sweatpants
13	T-Shirt
14	Tank Top

--- List of Transactions ---

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	T-Shirt	Jeans	Jacket	NaN
2	Dress	Handbag	Scarf	Sandals
3	Shorts	Flip Flops	NaN	NaN
4	Hoodie	Sweatpants	Sneakers	NaN
5	Jacket	T-Shirt	Dress	Handbag
6	Jeans	Scarf	NaN	NaN
7	Sneakers	T-Shirt	Shorts	NaN
8	Sweatpants	Jacket	Beanie	NaN
9	Handbag	Dress	Sandals	Scarf
10	Hoodie	T-Shirt	Jeans	Sneakers
11	Flip Flops	Tank Top	NaN	NaN
12	Sweatpants	Hoodie	Jacket	NaN
13	Dress	Beanie	Handbag	Jeans
14	T-Shirt	Sandals	Scarf	NaN
15	Hoodie	Sweatpants	Flip Flops	NaN
16	Shorts	Jacket	Sneakers	T-Shirt
17	Dress	Handbag	Beanie	NaN
18	Scarf	Tank Top	Shorts	Sneakers
19	T-Shirt	Hoodie	Sweatpants	NaN
20	Jeans	Flip Flops	Dress	NaN

4) Nike

```
Displaying Items and Transactions for: Nike

Loading data from data/Nike.csv
Data loaded successfully from data/Nike.csv

--- List of Items in All Transactions ---
```

	Items
1	Backpack
2	Fitness Leggings
3	Headband
4	Jacket
5	Running Shoes
6	Soccer Ball
7	Socks
8	Sports Bra
9	T-Shirt
10	Water Bottle

```
--- List of Transactions ---
```

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Running Shoes	Sports Bra	NaN	NaN
2	Jacket	Backpack	Water Bottle	Soccer Ball
3	Headband	Socks	Running Shoes	Fitness Leggings
4	T-Shirt	Sports Bra	Backpack	NaN
5	Fitness Leggings	Jacket	Soccer Ball	Headband
6	Running Shoes	Water Bottle	T-Shirt	NaN
7	Sports Bra	Headband	Backpack	NaN
8	Jacket	T-Shirt	Water Bottle	Soccer Ball
9	Fitness Leggings	Socks	Sports Bra	NaN
10	Running Shoes	Backpack	Water Bottle	T-Shirt
11	Soccer Ball	Fitness Leggings	Jacket	NaN
12	T-Shirt	Sports Bra	Headband	NaN
13	Backpack	Water Bottle	Soccer Ball	NaN
14	Headband	Jacket	Running Shoes	T-Shirt
15	Socks	Sports Bra	Water Bottle	Backpack
16	Fitness Leggings	T-Shirt	Soccer Ball	NaN
17	Running Shoes	Backpack	Jacket	NaN
18	Water Bottle	T-Shirt	Fitness Leggings	NaN
19	Headband	Running Shoes	Jacket	Soccer Ball
20	Socks	T-Shirt	Backpack	NaN

5) Walmart

```
Displaying Items and Transactions for: Walmart

Loading data from data/Walmart.csv
Data loaded successfully from data/Walmart.csv

--- List of Items in All Transactions ---
```

	Items
1	Bread
2	Cereal
3	Chicken
4	Dish Soap
5	Eggs
6	Milk
7	Pasta
8	Rice
9	Shampoo
10	Toilet Paper

```
--- List of Transactions ---
```

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Milk	Bread	Eggs	NaN
2	Cereal	Toilet Paper	Shampoo	NaN
3	Rice	Pasta	Eggs	NaN
4	Bread	Chicken	Cereal	Toilet Paper
5	Shampoo	Dish Soap	Rice	NaN
6	Eggs	Milk	Chicken	Bread
7	Toilet Paper	Shampoo	Cereal	NaN
8	Chicken	Bread	Milk	Pasta
9	Eggs	Rice	Toilet Paper	Dish Soap
10	Shampoo	Bread	Chicken	NaN
11	Milk	Eggs	Toilet Paper	Rice
12	Pasta	Shampoo	Chicken	NaN
13	Cereal	Milk	Toilet Paper	NaN
14	Chicken	Shampoo	Rice	Pasta
15	Eggs	Dish Soap	Cereal	Milk
16	Bread	Toilet Paper	Chicken	Shampoo
17	Rice	Milk	Eggs	NaN
18	Chicken	Dish Soap	Bread	NaN
19	Shampoo	Pasta	Milk	Eggs
20	Rice	Chicken	Cereal	NaN

Following are the implementation procedures :

- Install packages

```
%pip install pandas
%pip install mlxtend
```

- Importing Libraries

```
import pandas as pd
import itertools
from collections import defaultdict
import time
from mlxtend.frequent_patterns import apriori, fpgrowth, association_rules
from pathlib import Path
from IPython.display import display, HTML
```

- Initialising path for datasets

```
# Use relative paths for datasets
datasets = {
    1: 'data/Amazon.csv',    # Amazon
    2: 'data/Costco.csv',    # Costco
    3: 'data/Levis.csv',     # Levis
    4: 'data/Nike.csv',      # Nike
    5: 'data/Walmart.csv'    # Walmart
}
```

- Loading the dataset and performing preprocessing operations

```
# Load dataset from CSV file
def load_data(file_path):
    df = pd.read_csv(file_path)
    transactions = df.apply(lambda x: set(x.dropna().astype(str)), axis=1).tolist()  # Ensure all items are strings
    return transactions
```

■ Displaying data from datasets i.e., item and transactions list

```
# Function to load data using pandas
def load_data_display(file_path):
    try:
        # Load the CSV file using pandas
        data = pd.read_csv(file_path)
        print(f"Data loaded successfully from {file_path}")
        return data
    except FileNotFoundError:
        print(f"File not found: {file_path}")
        return None
    except Exception as e:
        print(f"An error occurred while loading the file: {e}")
        return None

# Function to display items and transactions in table format with borders
def display_items_and_transactions(data):
    # Combine all item columns into one list for unique items
    items = pd.concat([data['Item 1'], data['Item 2'], data['Item 3'], data['Item 4']]).dropna().unique()

    # Create a DataFrame for items
    items_df = pd.DataFrame(sorted(items), columns=["Items"])
    items_df.index = range(1, len(items_df) + 1)  # Adjust the index to start from 1
    print("\n--- List of Items in All Transactions ---")

    # Display items with borders
    display(HTML(items_df.to_html(border=1)))

    # Create a DataFrame for transactions
    transactions_df = data[['Transaction ID', 'Item 1', 'Item 2', 'Item 3', 'Item 4']].copy()
    transactions_df = transactions_df.set_index('Transaction ID')  # Keep the default index for transactions
    print("\n--- List of Transactions ---")

    # Display transactions with borders
    display(HTML(transactions_df.to_html(border=1)))

list = ['Amazon', 'Costco', 'Levis', 'Nike', 'Walmart']

# Loop until valid input or exit
while True:
    print("Please choose a dataset:")
    for key, name in zip(datasets.keys(), list):
        print(f"{key}. {name}")
    print("6. Exit")  # Option to exit

    try:
        choice = int(input("Enter the number corresponding to the dataset: "))
        if choice == 6:
            print("Exiting program.")
            break  # Exit the program
        elif choice in datasets:
            print(f"\nDisplaying Items and Transactions for: {list[choice-1]}\n")
            file_path = datasets[choice]
            print(f"Loading data from {file_path}")
            transactions = load_data_display(file_path)
            if transactions is not None:
                display_items_and_transactions(transactions)  # Display items and transactions in table format
                break  # Valid dataset chosen and loaded, exit the loop
        else:
            print("Invalid choice. Please select a valid dataset.")
    except ValueError:
        print("Invalid input. Please enter a number corresponding to the dataset.")
```

■ Brute Force Approach

➡ Generate 1-itemsets

```
def generate_1_itemsets(transactions):
    itemset = defaultdict(int)
    for transaction in transactions:
        for item in transaction:
            itemset[item] += 1
    return {frozenset([item]): count for item, count in itemset.items()}
```

➡ Generate k-itemsets from (k-1)-itemsets

```
def generate_k_itemsets(prev_itemsets, k):
    new_itemsets = set()
    prev_itemsets = list(prev_itemsets.keys())
    for i in range(len(prev_itemsets)):
        for j in range(i + 1, len(prev_itemsets)):
            candidate = prev_itemsets[i] | prev_itemsets[j]
            if len(candidate) == k:
                new_itemsets.add(candidate)

    return new_itemsets
```

➡ Count support for itemsets

```
def count_support(transactions, itemsets):
    itemset_count = defaultdict(int)

    for transaction in transactions:
        for itemset in itemsets:
            if itemset.issubset(transaction):
                itemset_count[itemset] += 1

    return itemset_count
```

➡ Generate frequent itemsets for Brute Force

```
def generate_frequent_itemsets(transactions, min_support):
    frequent_itemsets = {}
    total_transactions = len(transactions)
    k = 1

    # Generate 1-itemsets
    current_itemsets = generate_1_itemsets(transactions)

    while current_itemsets:
        # Filter itemsets based on min_support
        frequent_itemsets_k = {itemset: count for itemset, count in current_itemsets.items() if (count / total_transactions) * 100 >= min_support}
        if not frequent_itemsets_k:
            break

        # Print the current k-itemsets and their counts
        print(f"\n{k}-itemsets:")
        for itemset, count in frequent_itemsets_k.items():
            support = (count / total_transactions) * 100
            print(f"Itemset: {set(itemset)}, Count: {count}, Support: {support:.2f}%")

        frequent_itemsets.update(frequent_itemsets_k)

        # Generate next k-itemsets
        k += 1
        current_candidates = generate_k_itemsets(frequent_itemsets_k, k)
        current_itemsets = count_support(transactions, current_candidates)

    return frequent_itemsets
```

➡ Generate association rules from frequent itemsets

```
def generate_association_rules(frequent_itemsets, min_confidence, total_transactions):
    rules = []

    for itemset, support_count in frequent_itemsets.items():
        for i in range(1, len(itemset)):
            for antecedent in itertools.combinations(itemset, i):
                antecedent = frozenset(antecedent)
                consequent = itemset - antecedent

                if consequent:
                    confidence = (support_count / frequent_itemsets[antecedent]) * 100
                    support = (support_count / total_transactions) * 100
                    if confidence >= min_confidence:
                        rules.append((antecedent, consequent, support, confidence))

    return rules
```

➡ Calling Brute Force function and calculating time

```
start_time = time.time()
frequent_itemsets_bf = generate_frequent_itemsets(transactions, min_support)
total_transactions = len(transactions)
association_rules_bf = generate_association_rules(frequent_itemsets_bf, min_confidence, total_transactions)
brute_force_time = time.time() - start_time
```

■ Apriori Approach and calculating time

```
start_time = time.time()
encoded_df = encode_transactions(transactions)
frequent_itemsets_apriori = apriori(encoded_df, min_support=min_support / 100, use_colnames=True)
rules_apriori = association_rules(frequent_itemsets_apriori, metric="confidence", min_threshold=min_confidence / 100)
apriori_time = time.time() - start_time
```

■ FP-Growth Approach and calculating time

```
start_time = time.time()
frequent_itemsets_fp = fpgrowth(encoded_df, min_support=min_support / 100, use_colnames=True)
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=min_confidence / 100)
fp_growth_time = time.time() - start_time
```

■ Formatting and comparing results (Between Brute Force, Apriori and FP-Growth)

```
# Function to transform rules into a consistent format for comparison (antecedents, consequents, support, confidence)
def format_rules(rule_list, brute_force=False):
    formatted_rules = set()
    if brute_force:
        # Brute force rules already come as tuples (antecedent, consequent, support, confidence)
        for antecedent, consequent, support, confidence in rule_list:
            formatted_rules.add((frozenset(antecedent), frozenset(consequent), round(support, 2), round(confidence, 2)))
    else:
        # For Apriori and FP-Growth, rules are in DataFrame format with 'antecedents', 'consequents', 'support', 'confidence'
        for _, row in rule_list.iterrows():
            formatted_rules.add((frozenset(row['antecedents']), frozenset(row['consequents']), round(row['support'] * 100, 2), round(row['confidence'] * 100, 2)))
    return formatted_rules

# Function to compare the association rules across algorithms
def compare_algorithm_rules(brute_force_rules, apriori_rules, fp_growth_rules):
    # Format the rules from each approach
    brute_force_set = format_rules(brute_force_rules, brute_force=True)
    apriori_set = format_rules(apriori_rules)
    fp_growth_set = format_rules(fp_growth_rules)

    # Check if all sets of rules are identical
    identical_rules = brute_force_set == apriori_set == fp_growth_set

    if identical_rules:
        print("\nThe algorithms produced exactly the same association rules (matching antecedents, consequents, support, and confidence).")
    else:
        print("\nThe algorithms generated differing association rules.")

        # Display specific differences for further analysis
        print("\nRules unique to Brute Force:")
        print(brute_force_set - apriori_set - fp_growth_set)

        print("\nRules unique to Apriori:")
        print(apriori_set - brute_force_set - fp_growth_set)

        print("\nRules unique to FP-Growth:")
        print(fp_growth_set - brute_force_set - apriori_set)
```

■ Comparing results and Time performance (Between Brute Force, Apriori and FP-Growth)

```
# Display results
print("\nBrute Force Results:")
for antecedent, consequent, support, confidence in association_rules_bf:
    print(f"Rule: {set(antecedent)} -> {set(consequent)}, Support: {support:.2f}%, Confidence: {confidence:.2f}%")

print("\nApriori Results:")
print(rules_apriori[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

print("\nFP-Growth Results:")
print(rules_fp[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

# Comparison of results (calling the comparison function)
compare_algorithm_rules(association_rules_bf, rules_apriori, rules_fp)


# Timing performance
print(f"\nTiming Performance:")
print(f"Brute Force Time: {brute_force_time:.4f} seconds")
print(f"Apriori Time: {apriori_time:.4f} seconds")
print(f"FP-Growth Time: {fp_growth_time:.4f} seconds")

# Determine which algorithm is fastest
fastest = min(("Brute Force", brute_force_time), ("Apriori", apriori_time), ("FP-Growth", fp_growth_time), key=lambda x: x[1])
print(f"\nFastest Algorithm: {fastest[0]} with a time of {fastest[1]:.4f} seconds")
```

Following are the results from the python file run on terminal :

User Input :

```
(base) krkaushikkumar@Ks-MacBook-Air Midterm % python KolarRavindraKumar_KaushikKumar_Midtermproj.py
Please choose a dataset:
1. Amazon
2. Costco
3. Levis
4. Nike
5. Walmart
6. Exit
Enter the number corresponding to the dataset: 4
Loading data
Enter minimum support threshold (0-100): 20
Support =  20.0 %
Enter minimum confidence threshold (0-100): 30
Confidence =  30.0 %
```

Result:

```
Brute Force Results:
Rule: {'Soccer Ball'} -> {'Jacket'}, Support: 25.00%, Confidence: 71.43%
Rule: {'Jacket'} -> {'Soccer Ball'}, Support: 25.00%, Confidence: 71.43%
Rule: {'Water Bottle'} -> {'Backpack'}, Support: 20.00%, Confidence: 57.14%
Rule: {'Backpack'} -> {'Water Bottle'}, Support: 20.00%, Confidence: 50.00%
Rule: {'T-Shirt'} -> {'Water Bottle'}, Support: 20.00%, Confidence: 44.44%
Rule: {'Water Bottle'} -> {'T-Shirt'}, Support: 20.00%, Confidence: 57.14%

Apriori Results:
      antecedents     consequents  support  confidence      lift
0  (Water Bottle)      (Backpack)     0.20    0.571429  1.428571
1      (Backpack)  (Water Bottle)     0.20    0.500000  1.428571
2   (Soccer Ball)        (Jacket)     0.25    0.714286  2.040816
3        (Jacket)   (Soccer Ball)     0.25    0.714286  2.040816
4       (T-Shirt)  (Water Bottle)     0.20    0.444444  1.269841
5  (Water Bottle)       (T-Shirt)     0.20    0.571429  1.269841

FP-Growth Results:
      antecedents     consequents  support  confidence      lift
0  (Water Bottle)      (Backpack)     0.20    0.571429  1.428571
1      (Backpack)  (Water Bottle)     0.20    0.500000  1.428571
2       (T-Shirt)  (Water Bottle)     0.20    0.444444  1.269841
3  (Water Bottle)       (T-Shirt)     0.20    0.571429  1.269841
4   (Soccer Ball)        (Jacket)     0.25    0.714286  2.040816
5        (Jacket)   (Soccer Ball)     0.25    0.714286  2.040816

The algorithms produced exactly the same association rules (matching antecedents, consequents, support, and confidence).

Timing Performance:
Brute Force Time: 0.0006 seconds
Apriori Time: 0.0128 seconds
FP-Growth Time: 0.0038 seconds

Fastest Algorithm: Brute Force with a time of 0.0006 seconds
```

Other

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

Link to Git Repository

https://github.com/kk795-NJIT/KolarRavindraKumar_KaushikKumar_Midtermproj.git