NAME: JYOTHSNA RAVI PRAKASH

NJIT UCID: jr987

Email Address: jr987@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 involves the implementation and comparison of three algorithms—Brute Force, Apriori, and FP-Growth—to extract frequent itemsets and generate association rules from transactional datasets. Utilizing transaction data from five major companies, the Brute Force method thoroughly generates and evaluates all possible itemsets, starting from single-itemsets and progressing until no further frequent sets can be identified. To improve efficiency and verify the results, the Apriori and FP-Growth algorithms are implemented using Python libraries. User-specified support and confidence thresholds are applied to generate various sets of association rules. The performance of the three algorithms is assessed based on their accuracy and execution time, emphasizing the efficiency improvements offered by Apriori and FP-Growth compared to the exhaustive Brute Force approach.

Introduction:

Frequent itemset mining and association rule generation are essential tasks in data mining, commonly applied across industries such as retail, banking, and finance to discover patterns and relationships within transactional data. This project aims to implement and compare three fundamental algorithms—Brute Force, Apriori, and FP-Growth (also referred to as FP-Tree)—for generating frequent itemsets and deriving association rules from transactional datasets. The datasets used in this analysis consist of transactions from five major companies: Amazon, Best Buy, IKEA, Puma, and Target.

The Brute Force method serves as a baseline by exhaustively enumerating all possible itemsets, starting from 1-itemsets and continuing until no further frequent itemsets are identified. While accurate, this method is computationally intensive, especially for larger datasets. To improve efficiency, optimized algorithms such as Apriori and FP-Growth are employed. Apriori reduces the number of candidate itemsets by utilizing the anti-

monotonicity principle, whereas FP-Growth compresses the dataset into a prefix tree structure, enabling more efficient mining of frequent patterns.

In addition to implementing these algorithms, this project evaluates their performance in terms of accuracy, execution time, and scalability. By applying user-defined support and confidence thresholds, the project illustrates how varying parameters influence the association rules generated for each dataset. Ultimately, the comparison highlights the trade-offs between the exhaustive Brute Force method and the more efficient Apriori and FP-Growth algorithms in practical data mining scenarios.

Key steps in the implementation include:

1. **Data Preparation:** Creating transactional datasets for five companies—Amazon, Best Buy, IKEA, Puma, and Target—ensuring that each dataset contains a minimum of 20 transactions representing frequently purchased items.

2. **Brute Force Implementation:** Developing an algorithm that generates all possible itemsets, beginning with 1-itemsets, followed by 2-itemsets, and continuing progressively. For each k-itemset, the algorithm evaluates its frequency against a user-defined support threshold, halting when no more frequent k-itemsets are identified.

3. **Apriori Algorithm:** Employing a pre-existing Python implementation of the Apriori algorithm to extract frequent itemsets and derive association rules. The Apriori method enhances efficiency by pruning infrequent itemsets early in the process, reducing computational effort compared to the brute force method.

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

5. **Association Rule Generation:** Deriving association rules from the frequent itemsets identified by all three algorithms, applying user-defined support and confidence thresholds.

6. **Performance Comparison:** Recording and comparing the execution time and consistency of results between the Brute Force, Apriori, and FP-Growth algorithms. Execution time is measured at varying support and confidence levels across all datasets.

7. **Result Analysis:** Assessing whether the three algorithms yield identical frequent itemsets and association rules, while determining which algorithm demonstrates the fastest performance for each dataset.

Core Concepts and Principles:

1. **Frequent Itemset Discovery:**

- This project focuses on identifying frequent itemsets from transactional data of five companies using three different algorithms—Brute Force, Apriori, and FP-Growth.
- Frequent itemsets refer to groups of items that frequently occur together in transactions, offering insights into item co-occurrence patterns.

2. **Support and Confidence:**

- Support is used to filter out infrequent itemsets, ensuring only those that occur frequently are considered. The project allows the user to define support thresholds for each dataset.
- Confidence is applied to evaluate association rules, assessing the likelihood of items being purchased together. The project explores various confidence levels to generate multiple association rules and analyze their significance.

3. **Association Rule Generation:**

- Association rules are derived from the frequent itemsets generated by all three algorithms. These rules highlight relationships between items that are frequently purchased together, providing insights into customer purchasing behavior.
- Strong association rules are further analyzed to suggest potential strategies for business optimization, such as product bundling or targeted marketing.

4. **Algorithm Comparison:**

- The Brute Force method exhaustively generates all possible itemsets but is computationally intensive.
- The Apriori and FP-Growth algorithms reduce computational complexity through optimizations such as pruning and tree-based structures, allowing for more efficient identification of frequent itemsets and association rules.

5. **Performance Evaluation:**

- The project evaluates and compares the algorithms based on their runtime and accuracy, demonstrating that while the results are consistent, the approaches vary in terms of computational efficiency.

Project Workflow:

1. **Data Loading and Preprocessing:**
    - Transactional datasets from five companies—Amazon, Best Buy, IKEA, Puma, and Target—are loaded into the system.
    - The data is preprocessed by filtering unique items in each transaction and arranging them in a predefined order, ensuring consistency and accuracy across datasets before initiating the frequent itemset mining process.

2. **Setting Minimum Support and Confidence:**
    - Users are prompted to define minimum support and confidence thresholds. These values determine which frequent itemsets and association rules are retained for analysis.
    - The flexibility of adjustable support and confidence levels enables users to explore varying insights from the same dataset.

3. **Iterative Candidate Itemset Generation:**
    - For each algorithm—Brute Force, Apriori, and FP-Growth—itemset generation starts with single-item sets (K = 1) and progressively increases to K = 2, K = 3, and beyond.
    - The Brute Force method exhaustively generates and evaluates all possible itemset combinations, while the Apriori algorithm prunes candidates based on previously identified frequent itemsets. FP-Growth uses a compressed prefix tree to minimize redundant calculations and improve efficiency.

4. **Support Count Calculation:**
    - The support for each candidate itemset is calculated by counting its occurrences across all transactions.
    - Itemsets that meet or exceed the user-specified minimum support threshold are retained, while those that do not are excluded from further consideration.

5. **Confidence Calculation:**
    - Once frequent itemsets are determined, confidence for each association rule is calculated by comparing the support of the combined itemset with the support of the antecedent.
    - Only rules that satisfy the minimum confidence threshold are retained for detailed analysis.

6. **Association Rule Generation:**

- Based on the identified frequent itemsets, association rules are generated using all three algorithms. These rules are filtered according to the user-defined support and confidence thresholds.
- The resulting rules provide insights into items frequently purchased together, offering valuable information about customer purchasing patterns.

7. **Performance Evaluation:**

- The project compares the performance of the Brute Force, Apriori, and FP-Growth algorithms in terms of execution time and computational efficiency for each dataset.
- The results, including execution time and the generated association rules, are analyzed to identify the most efficient algorithm under varying support and confidence conditions.

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 data from five major companies. All three algorithms produced consistent association rules, validating their effectiveness in uncovering relationships between items. Although the Brute Force method served as a benchmark for accuracy due to its exhaustive nature, the Apriori and FP-Growth algorithms significantly outperformed it in terms of execution time, making them more suitable for larger datasets. The findings emphasize the practical utility of these algorithms in optimizing sales strategies and gaining customer insights, demonstrating the project's success in generating valuable outcomes through efficient data mining techniques.

Conclusion:

In conclusion, this project successfully demonstrated the use of Brute Force, Apriori, and FP-Growth algorithms for mining frequent itemsets and generating association rules from transactional data across five major companies. While the Brute Force method provided accurate results, its computational inefficiency underscored the benefits of employing more optimized approaches like Apriori and FP-Growth. By comparing execution times and the quality of the generated rules, the project highlighted the importance of selecting suitable data mining techniques for practical applications in retail and other industries. Ultimately, this work offers valuable insights into customer purchasing behavior, showcasing the critical role of data mining in supporting data-driven business decisions and optimizing sales strategies.

Screenshots

Here are what the csv files (This program takes in two separate csv files: Item Names & Transactions).

Figure 1 : Amazon Item Names CSV file

Figure 2 : Amazon Transactions CSV file

```
Displaying Items and Transactions for: Amazon

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

Figure 3 : Best Buy Item Names CSV file

Figure 4 : Best Buy Transactions CSV file

```
Displaying Items and Transactions for: Best Buy

Data loaded successfully from Data/Best_Buy.csv

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

	Items
1	Bluetooth Speaker
2	Camera
3	Headphones
4	Keyboard
5	Laptop
6	Lens
7	Monitor
8	Mouse
9	Smartwatch
10	Tablet
11	Tripod

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

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Laptop	Monitor	Keyboard	Mouse
2	Tablet	Smartwatch	Bluetooth Speaker	NaN
3	Headphones	Laptop	Smartwatch	NaN
4	Camera	Tripod	Lens	NaN
5	Smartwatch	Tablet	Laptop	Bluetooth Speaker
6	Monitor	Keyboard	Mouse	NaN
7	Bluetooth Speaker	Headphones	Laptop	NaN
8	Keyboard	Tablet	Smartwatch	Camera
9	Monitor	Laptop	Headphones	NaN
10	Mouse	Keyboard	Laptop	Monitor
11	Smartwatch	Tablet	Bluetooth Speaker	NaN
12	Laptop	Camera	Tripod	NaN
13	Headphones	Monitor	Keyboard	NaN
14	Bluetooth Speaker	Smartwatch	Laptop	Camera
15	Tablet	Keyboard	Monitor	NaN
16	Camera	Mouse	Smartwatch	NaN
17	Headphones	Tablet	Monitor	NaN
18	Smartwatch	Laptop	Bluetooth Speaker	NaN
19	Monitor	Mouse	Keyboard	NaN
20	Camera	Smartwatch	Headphones	NaN

Figure 5 : IKEA Item Names CSV file

Figure 6 : IKEA Transactions CSV file

```
Displaying Items and Transactions for: IKEA

Data loaded successfully from Data/IKEA.csv

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

	Items
1	Bed
2	Bedding
3	Chairs
4	Coffee Table
5	Dining Table
6	Hangers
7	Lamp
8	Mattress
9	Shelf
10	Sofa
11	Tableware
12	Wardrobe

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

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Dining Table	Chairs	Tableware	NaN
2	Sofa	Coffee Table	Lamp	NaN
3	Bed	Mattress	Bedding	NaN
4	Wardrobe	Hangers	Shelf	NaN
5	Dining Table	Coffee Table	Chairs	Tableware
6	Bed	Wardrobe	Hangers	NaN
7	Sofa	Dining Table	Tableware	NaN
8	Coffee Table	Chairs	Lamp	NaN
9	Bed	Mattress	Hangers	NaN
10	Dining Table	Shelf	Chairs	NaN
11	Sofa	Mattress	Coffee Table	Hangers
12	Wardrobe	Tableware	Shelf	NaN
13	Chairs	Coffee Table	Bed	Lamp
14	Dining Table	Sofa	Hangers	NaN
15	Mattress	Wardrobe	Shelf	NaN
16	Coffee Table	Chairs	Tableware	NaN
17	Bed	Hangers	Sofa	NaN
18	Dining Table	Mattress	Wardrobe	NaN
19	Chairs	Coffee Table	Lamp	Shelf
20	Sofa	Dining Table	Hangers	Tableware

Figure 7 : Puma Item Names CSV file

Figure 8 : Puma Transactions CSV file

```
Displaying Items and Transactions for: Puma

Data loaded successfully from Data/Puma.csv

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

	Items
1	Backpack
2	Cap
3	Flip Flops
4	Hoodie
5	Running Shoes
6	Sneakers
7	Socks
8	Sports Bra
9	Sweatpants
10	T-Shirt

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

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Running Shoes	Sports Bra	T-Shirt	NaN
2	Hoodie	Sweatpants	Sneakers	NaN
3	Flip Flops	Cap	Backpack	NaN
4	Socks	Running Shoes	T-Shirt	NaN
5	Hoodie	Flip Flops	Backpack	NaN
6	Sports Bra	Sneakers	Sweatpants	NaN
7	T-Shirt	Socks	Cap	NaN
8	Running Shoes	Hoodie	Flip Flops	NaN
9	Sports Bra	Sweatpants	Sneakers	T-Shirt
10	Backpack	Socks	Cap	NaN
11	Hoodie	Running Shoes	T-Shirt	NaN
12	Flip Flops	Sports Bra	Backpack	NaN
13	Sweatpants	Sneakers	Socks	NaN
14	T-Shirt	Hoodie	Cap	NaN
15	Running Shoes	Flip Flops	Backpack	NaN
16	Sports Bra	Sweatpants	Sneakers	T-Shirt
17	Socks	Hoodie	Cap	NaN
18	T-Shirt	Running Shoes	Backpack	NaN
19	Sweatpants	Sports Bra	Socks	NaN
20	Flip Flops	Sneakers	Hoodie	Cap

Figure 9 : Target Item Names CSV file

Figure 10 : Target Transactions CSV file

```
Displaying Items and Transactions for: Targe

Data loaded successfully from Data/Target.cs

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

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

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

Transaction ID	Item 1	Item 2	Item 3	Item 4
1	Milk	Bread	Eggs	Cheese
2	Cereal	Toilet Paper	Shampoo	NaN
3	Rice	Pasta	Bread	NaN
4	Chicken	Cereal	Eggs	Milk
5	Shampoo	Toilet Paper	Rice	NaN
6	Eggs	Milk	Chicken	Bread
7	Toilet Paper	Shampoo	Cereal	NaN
8	Chicken	Bread	Milk	Pasta
9	Eggs	Rice	Toilet Paper	Shampoo
10	Cereal	Chicken	Milk	NaN
11	Milk	Eggs	Toilet Paper	Rice
12	Pasta	Chicken	Shampoo	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

Below are screenshots of the code from python file:

Installing required Libraries:

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

Importing Packages:

```
import pandas as pd
import time
from mlxtend.frequent_patterns import apriori, fpgrowth, association_rules
from IPython.display import display , HTML
```

Loading the dataset:

```
# 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
```

Relative paths for datasets:

```
datasets = {
    1: 'Data/Amazon.csv',  # Amazon
    2: 'Data/Best_Buy.csv',  # Costco
    3: 'Data/IKEA.csv',  # Levis
    4: 'Data/Puma.csv',  # Nike
    5: 'Data/Target.csv'   # Walmart
}
```

Displaying items and transactions for the choosen dataset:

```
# 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):
    # Gather all unique items from the item columns
    items = pd.concat([data['Item 1'], data['Item 2'], data['Item 3'], data['Item 4']]).dropna().unique()

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

    # Create a DataFrame for transactions, with Transaction ID as index
    transactions_df = data[['Transaction ID', 'Item 1', 'Item 2', 'Item 3', 'Item 4']].set_index('Transaction ID')
    print("\n--- List of Transactions ---")
    display(HTML(transactions_df.to_html(border=1)))

# Dataset names for display
dataset_names = ['Amazon', 'Best Buy', 'IKEA', 'Puma', 'Target']

# Loop to prompt user for dataset selection
while True:
    print("Please choose a dataset:")
    for key, name in zip(datasets.keys(), dataset_names):
        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 the program.")
            break  # Exit the loop
        elif choice in datasets:
            print(f"\nDisplaying Items and Transactions for: {dataset_names[choice - 1]}\n")
            file_path = datasets[choice]
            transactions = load_data_display(file_path)
            if transactions is not None:
                display_items_and_transactions(transactions)  # Display items and transactions
            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:

1. Generating 1-itemsets for brute force:

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

2. Generating k-itemsets from (k-1)-itemsets:

```
def generate_k_itemsets(prev_itemsets, k):
    new_itemsets = []
    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 and candidate not in new_itemsets:
                new_itemsets.append(candidate)

    return new_itemsets
```

3. Count support for itemsets:

```
def count_support(transactions, itemsets):
    itemset_count = {itemset: 0 for itemset in itemsets}

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

    return itemset_count
```

4. Generating 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
```

5. Generating 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)):
            antecedent_list = [frozenset(antecedent) for antecedent in generate_combinations(list(itemset), i)]
            for antecedent in antecedent_list:
                consequent = itemset - antecedent

                if consequent:
                    confidence = (support_count / frequent_itemsets[antecedent]) * 100 if antecedent in frequent_itemsets else 0
                    support = (support_count / total_transactions) * 100
                    if confidence >= min_confidence:
                        rules.append((antecedent, consequent, support, confidence))

    return rules
```

6. Helper function to generate combinations:

```
def generate_combinations(items, n):
    if n == 0:
        return [[]]
    elif len(items) < n:
        return []
    else:
        with_first = generate_combinations(items[1:], n - 1)
        without_first = generate_combinations(items[1:], n)

        return [[items[0]] + comb for comb in with_first] + without_first
```

7. Encoding transactions to one-hot format:

```
def encode_transactions(transactions):
    items = sorted(set(item for transaction in transactions for item in transaction))
    encoded_df = pd.DataFrame(0, index=range(len(transactions)), columns=items)

    for idx, transaction in enumerate(transactions):
        for item in transaction:
            encoded_df.at[idx, item] = 1

    return encoded df
```

Comparison of results:

```
# Helper function to convert rules into comparable sets (antecedents, consequents, support, confidence)
def extract_rules(rule_list, is_bf=False):
    extracted_rules = set()
    if is_bf:
        # Brute force rules already structured as (antecedent, consequent, support, confidence)
        for antecedent, consequent, support, confidence in rule_list:
            extracted_rules.add((frozenset(antecedent), frozenset(consequent), round(support, 2), round(confidence, 2)))
    else:
        # Apriori and FP-Growth: DataFrame format with 'antecedents', 'consequents', 'support', 'confidence'
        for _, row in rule_list.iterrows():
            extracted_rules.add((frozenset(row['antecedents']), frozenset(row['consequents']), round(row['support']*100, 2), round(row['confidence']*100, 2)))
    return extracted_rules

# Function to compare rules from different algorithms
def compare_rules(association_rules_bf, rules_apriori, rules_fp):
    # Extract rules from the three approaches
    rules_bf_set = extract_rules(association_rules_bf, is_bf=True)
    rules_apriori_set = extract_rules(rules_apriori)
    rules_fp_set = extract_rules(rules_fp)

    # Compare the exact content of the rules
    same_rules = rules_bf_set == rules_apriori_set == rules_fp_set

    if same_rules:
        print("\nThe algorithms produced identical association rules (including antecedents, consequents, support, and confidence).")
    else:
        print("\nThe algorithms produced different association rules.")

        # Print differences for debugging/comparison
        print("\nBrute Force Rules not in Apriori or FP-Growth:")
        print(rules_bf_set - rules_apriori_set - rules_fp_set)

        print("\nApriori Rules not in Brute Force or FP-Growth:")
        print(rules_apriori_set - rules_bf_set - rules_fp_set)

        print("\nFP-Growth Rules not in Brute Force or Apriori:")
        print(rules_fp_set - rules_bf_set - rules_apriori_set)
```

Main Function:

1. Importing Datasets:

```
def main():
    # Loop until valid input or exit
    while True:
        print("Please choose a dataset:")
        for key, name in zip(datasets.keys(), ['Amazon', 'Best Buy', 'IKEA', 'Puma', 'Target']):
            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.")
                return  # Exit the program
            elif choice in datasets:
                file_path = datasets[choice]
                print(f"Loading data")
                transactions = load_data(file_path)
                break  # Valid dataset chosen, 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.")

            # Input support and confidence thresholds
    min_support = float(input("Enter minimum support threshold (0-100): "))
    min_confidence = float(input("Enter minimum confidence threshold (0-100): "))
```

2. Running Brute-Force Approach:

```
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
```

3. Apriori Approach:

```
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
```

4. FP- Growth Approach:

```
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
```

5. Displaying the results of Brute-Force, Apriori and FP- Growth Approach:

```
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']])
```

6. Comparison of the results, Timing performance and determining which algorithm is faster:

```
    # Comparison of results (call the comparison function)
    compare_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")


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

Below are screenshots to show that the program runs in the Terminal:

```
Please choose a dataset:
1. Amazon
2. Best Buy
3. IKEA
4. Puma
5. Target
6. Exit
Enter the number corresponding to the dataset: 4
Loading data
Enter minimum support threshold (0-100): 20
Enter minimum confidence threshold (0-100): 35
```

Final Output:

```
Brute Force Results:
Rule: {'Running Shoes'} -> {'T-Shirt'}, Support: 20.00%, Confidence: 66.67%
Rule: {'T-Shirt'} -> {'Running Shoes'}, Support: 20.00%, Confidence: 50.00%
Rule: {'Sweatpants'} -> {'Sports Bra'}, Support: 20.00%, Confidence: 66.67%
Rule: {'Sports Bra'} -> {'Sweatpants'}, Support: 20.00%, Confidence: 66.67%
Rule: {'Sweatpants'} -> {'Sneakers'}, Support: 25.00%, Confidence: 83.33%
Rule: {'Sneakers'} -> {'Sweatpants'}, Support: 25.00%, Confidence: 83.33%
Rule: {'Backpack'} -> {'Flip Flops'}, Support: 20.00%, Confidence: 66.67%
Rule: {'Flip Flops'} -> {'Backpack'}, Support: 20.00%, Confidence: 66.67%

Apriori Results:
       antecedents      consequents  support  confidence      lift
0       (Backpack)     (Flip Flops)     0.20    0.666667  2.222222
1     (Flip Flops)       (Backpack)     0.20    0.666667  2.222222
2  (Running Shoes)        (T-Shirt)     0.20    0.666667  1.666667
3        (T-Shirt)  (Running Shoes)     0.20    0.500000  1.666667
4     (Sweatpants)       (Sneakers)     0.25    0.833333  2.777778
5       (Sneakers)     (Sweatpants)     0.25    0.833333  2.777778
6     (Sweatpants)     (Sports Bra)     0.20    0.666667  2.222222
7     (Sports Bra)     (Sweatpants)     0.20    0.666667  2.222222

FP-Growth Results:
       antecedents      consequents  support  confidence      lift
0     (Sweatpants)     (Sports Bra)     0.20    0.666667  2.222222
1     (Sports Bra)     (Sweatpants)     0.20    0.666667  2.222222
2  (Running Shoes)        (T-Shirt)     0.20    0.666667  1.666667
3        (T-Shirt)  (Running Shoes)     0.20    0.500000  1.666667
4     (Sweatpants)       (Sneakers)     0.25    0.833333  2.777778
5       (Sneakers)     (Sweatpants)     0.25    0.833333  2.777778
6       (Backpack)     (Flip Flops)     0.20    0.666667  2.222222
7     (Flip Flops)       (Backpack)     0.20    0.666667  2.222222

The algorithms produced identical association rules (including antecedents, consequents, support, and confidence).

Timing Performance:
Brute Force Time: 0.0010 seconds
Apriori Time: 0.0081 seconds
FP-Growth Time: 0.0029 seconds
```

Other

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

Link to Git Repository: https://github.com/jr987-NJIT/RaviPrakash_Jyothsna_MidtermProj.git