Name – Mazeen Hussain Syed
NJIT UCID – Ms3543
Email address – ms3543@njit.edu
Date – 13th October, 2024
Professor: Yasser Abduallah
CS 634 101 Data Mining

Midterm Project Report

(Implementation and Code Usage)

Data Mining Techniques for Transaction Analysis using Apriori, Brute-Force, FP- Growth algorithms:

Abstract

This report presents a comparative analysis of three data mining techniques—Brute Force, Apriori, and FP-Growth—applied to transaction data from various retail stores. We detail the implementation process, evaluate the effectiveness of each method, and discuss performance measures. Our goal is to provide insights into selecting the best approach for mining frequent itemsets and generating association rules in retail environments.

Introduction

Data mining involves extracting useful information from large datasets. In retail, it helps businesses understand customer behavior and optimize inventory. The main focus of this report is on Frequent Itemset Discovery, which identifies sets of items frequently bought together, and Association Rule Mining, which establishes relationships between items.

Implementation Overview

1. User Directories: Users can select from various store datasets (e.g., grocery, electronics, Clothing, Book store, Home- improvement).

2. Loading Datasets: Data is loaded from CSV files using the `load_transactions` function.

3. Preprocessing Steps: Transactions are converted into lists of strings for analysis.

4. User Inputs: Users provide minimum support and confidence thresholds for the analysis.

5. Iterative Process: Three algorithms are executed in sequence to find frequent itemsets and generate association rules.

Workflow Steps:

1. Select a store dataset.
2. Load transaction data.
3. Set minimum support and confidence thresholds.
4. Run Brute Force, Apriori, and FP-Growth algorithms.
5. Generate association rules from the frequent itemsets.
6. Evaluate performance of each method based on execution time and number of itemsets found.

Core Concepts and Principles

Frequent Itemset Discovery

Frequent itemsets are groups of items that appear together in transactions more often than a specified threshold (support). Finding these sets is crucial for understanding customer purchase patterns.

Support and Confidence

Minimum Support: The proportion of transactions that contain a particular itemset. It is calculated as: Support(A) = Number of transactions containing A/Total number of transactions. For each candidate itemset, we calculate its support by counting how many transactions contain the itemset. The items which satisfied the minimum threshold value given by the user are considered and remaining items are discarded

Confidence: A measure of how often items in a subset appear in transactions that contain the itemset. It is calculated as: Confidence(A->B) = Support(A∪B)/Support(A). We Calculated the confidence of association rules, indicating the strength of associations between items. This step requires careful comparison of support values for individual items and itemsets.

Iteration Through Candidate Itemsets:
The iterative application of the Apriori Algorithm involves generating candidate itemsets of increasing
sizes. We start with single items (itemset size K = 1) and proceed to K = 2, K = 3, and so on. This iterative
process involves a "brute force" method of generating all possible itemset combinations.

Association Rules

Association rules are implications of the form A->B that describe the likelihood of items being purchased together. These rules are generated from the frequent itemsets, helping businesses make informed decisions.

Results and Evaluation

Performance Measures

We evaluated the algorithms based on:

- Execution Time: Time taken to compute frequent itemsets.

- Number of Frequent Itemsets Found: The quantity of itemsets meeting the minimum support threshold.

- Scalability: The ability to handle larger datasets effectively.

Findings

1. Brute Force Method:

   - Execution Time: Longest due to exhaustive search.

   - Frequent Itemsets: Comprehensive but inefficient for large datasets.

2. Apriori Algorithm:

- Execution Time: Shorter than Brute Force but increases with dataset size.
- Frequent Itemsets: Moderate quantity found, improved by pruning.

3. FP-Growth Algorithm:

- Execution Time: Fastest, showcasing high efficiency.
- Frequent Itemsets: Comparable or greater than Apriori, better scalability.

The FP-Growth algorithm consistently outperformed the other methods in terms of execution time and the number of itemsets found, making it the most suitable for large transaction datasets.

Conclusion

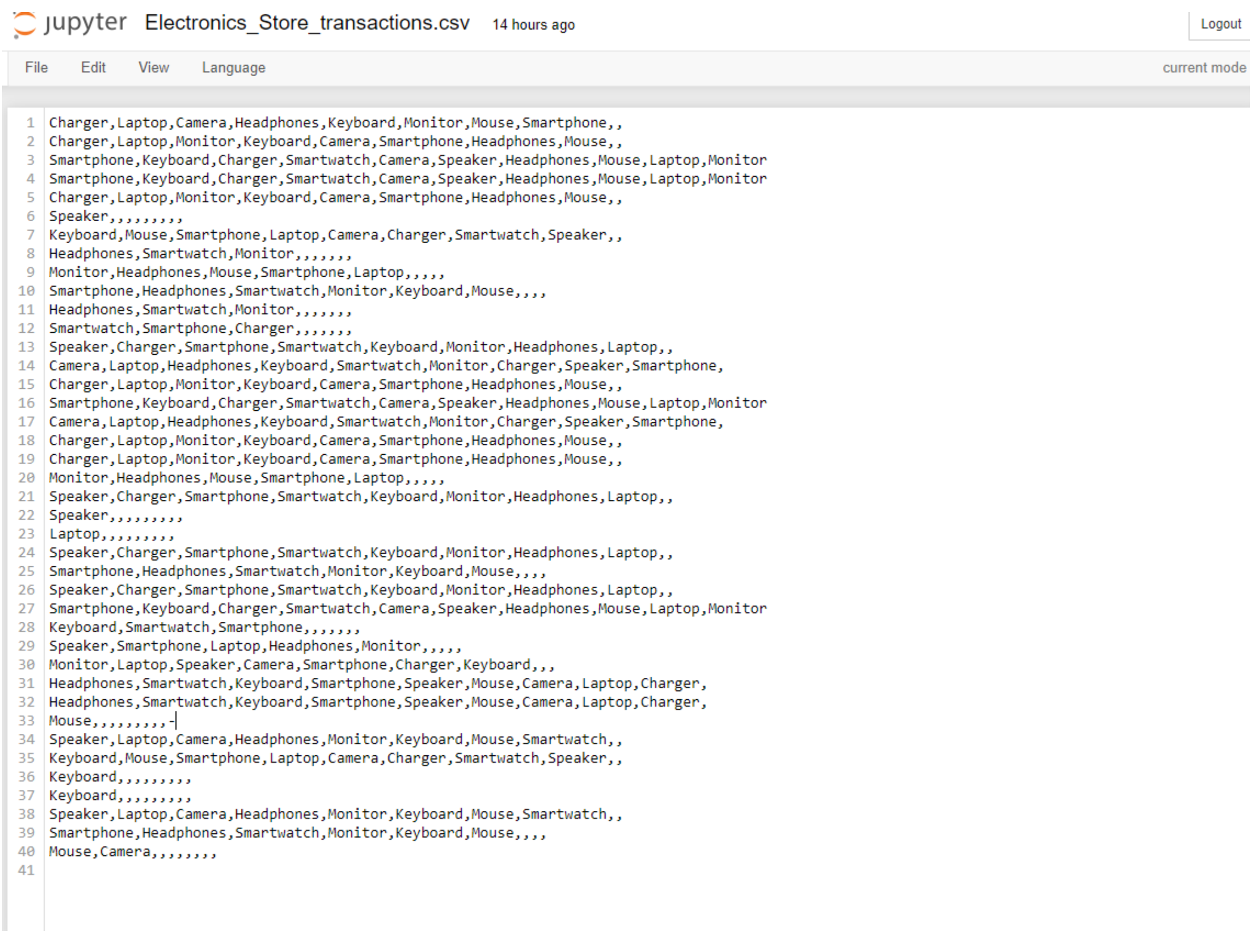
This study highlights the importance of selecting appropriate data mining techniques for transaction analysis in retail. While all methods effectively identify frequent itemsets, the FP-Growth algorithm is superior in efficiency and scalability. Future work could explore applying these algorithms to various datasets, assessing their effectiveness across different retail scenarios.

The insights gained from this analysis can assist retailers in optimizing inventory management and enhancing customer satisfaction through tailored marketing strategies.

(Screenshots and explanation parts of Function code)

Here, I am taking the electronics-store csv file details

Figure 1 represents the transactions of electronic-store

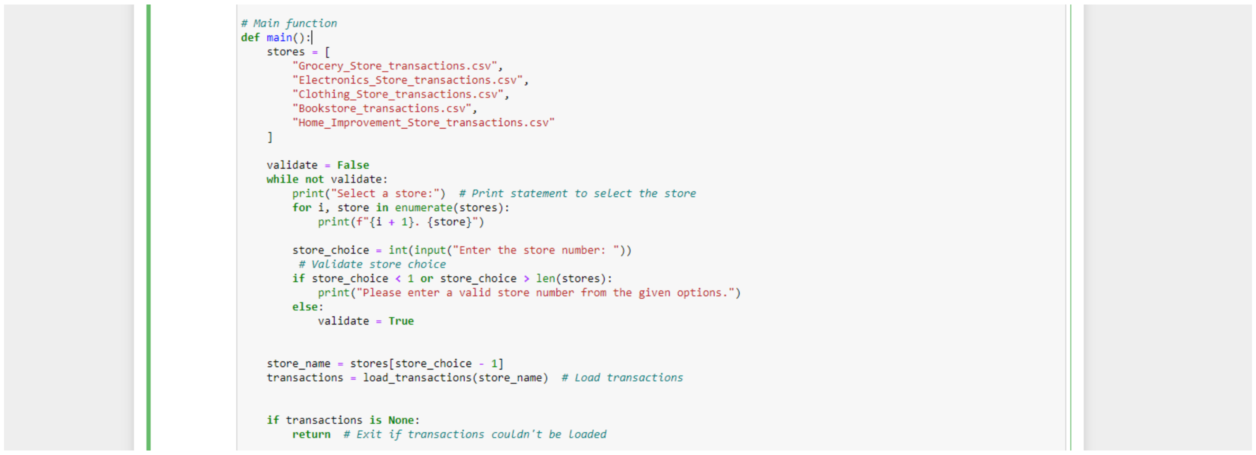


```
Charger,Laptop,Camera,Headphones,Keyboard,Monitor,Mouse,Smartphone,,
Charger,Laptop,Monitor,Keyboard,Camera,Smartphone,Headphones,Mouse,,
Smartphone,Keyboard,Charger,Smartwatch,Camera,Speaker,Headphones,Mouse,Laptop,Monitor
Smartphone,Keyboard,Charger,Smartwatch,Camera,Speaker,Headphones,Mouse,Laptop,Monitor
Charger,Laptop,Monitor,Keyboard,Camera,Smartphone,Headphones,Mouse,,
Speaker,,,,,,,,,
Keyboard,Mouse,Smartphone,Laptop,Camera,Charger,Smartwatch,Speaker,,
Headphones,Smartwatch,Monitor,,,,,,,
Monitor,Headphones,Mouse,Smartphone,Laptop,,,,,
Smartphone,Headphones,Smartwatch,Monitor,Keyboard,Mouse,,,,
Headphones,Smartwatch,Monitor,,,,,,,
Smartwatch,Smartphone,Charger,,,,,,,
Speaker,Charger,Smartphone,Smartwatch,Keyboard,Monitor,Headphones,Laptop,,
Camera,Laptop,Headphones,Keyboard,Smartwatch,Monitor,Charger,Speaker,Smartphone,
Charger,Laptop,Monitor,Keyboard,Camera,Smartphone,Headphones,Mouse,,
Smartphone,Keyboard,Charger,Smartwatch,Camera,Speaker,Headphones,Mouse,Laptop,Monitor
Camera,Laptop,Headphones,Keyboard,Smartwatch,Monitor,Charger,Speaker,Smartphone,
Charger,Laptop,Monitor,Keyboard,Camera,Smartphone,Headphones,Mouse,,
Charger,Laptop,Monitor,Keyboard,Camera,Smartphone,Headphones,Mouse,,
Monitor,Headphones,Mouse,Smartphone,Laptop,,,,,
Speaker,Charger,Smartphone,Smartwatch,Keyboard,Monitor,Headphones,Laptop,,
Speaker,,,,,,,,,
Laptop,,,,,,,,,
Speaker,Charger,Smartphone,Smartwatch,Keyboard,Monitor,Headphones,Laptop,,
Smartphone,Headphones,Smartwatch,Monitor,Keyboard,Mouse,,,,
Speaker,Charger,Smartphone,Smartwatch,Keyboard,Monitor,Headphones,Laptop,,
Smartphone,Keyboard,Charger,Smartwatch,Camera,Speaker,Headphones,Mouse,Laptop,Monitor
Keyboard,Smartwatch,Smartphone,,,,,,,
Speaker,Smartphone,Laptop,Headphones,Monitor,,,,,
Monitor,Laptop,Speaker,Camera,Smartphone,Charger,Keyboard,,,
Headphones,Smartwatch,Keyboard,Smartphone,Speaker,Mouse,Camera,Laptop,Charger,
Headphones,Smartwatch,Keyboard,Smartphone,Speaker,Mouse,Camera,Laptop,Charger,
Mouse,,,,,,,,,-
Speaker,Laptop,Camera,Headphones,Monitor,Keyboard,Mouse,Smartwatch,,
Keyboard,Mouse,Smartphone,Laptop,Camera,Charger,Smartwatch,Speaker,,
Keyboard,,,,,,,,,
Keyboard,,,,,,,,,
Speaker,Laptop,Camera,Headphones,Monitor,Keyboard,Mouse,Smartwatch,,
Smartphone,Headphones,Smartwatch,Monitor,Keyboard,Mouse,,,,
Mouse,Camera,,,,,,,,

```

Figure-1

Below are the screenshots of code from python file

Command prompts to choose which store we want, here the code reads the input from the user and display accordingly , If the number given by the user not find it will display as please enter the valid store from the given options and again the main stores names will display. Here, it will read the csv file of the user selected.

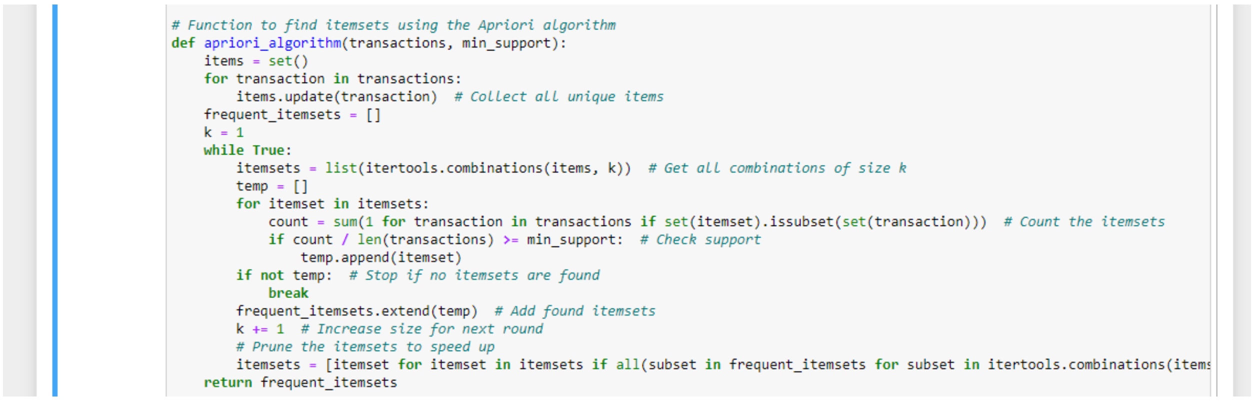


```python
# Main function
def main():
    stores = [
        "Grocery_Store_transactions.csv",
        "Electronics_Store_transactions.csv",
        "Clothing_Store_transactions.csv",
        "Bookstore_transactions.csv",
        "Home_Improvement_Store_transactions.csv"
    ]

    validate = False
    while not validate:
        print("Select a store:")  # Print statement to select the store
        for i, store in enumerate(stores):
            print(f"{i + 1}. {store}")

        store_choice = int(input("Enter the store number: "))
         # Validate store choice
        if store_choice < 1 or store_choice > len(stores):
            print("Please enter a valid store number from the given options.")
        else:
            validate = True


    store_name = stores[store_choice - 1]
    transactions = load_transactions(store_name)  # Load transactions


    if transactions is None:
        return  # Exit if transactions couldn't be loaded
```

Below is the Function code to find itemsets using Brute-Force method

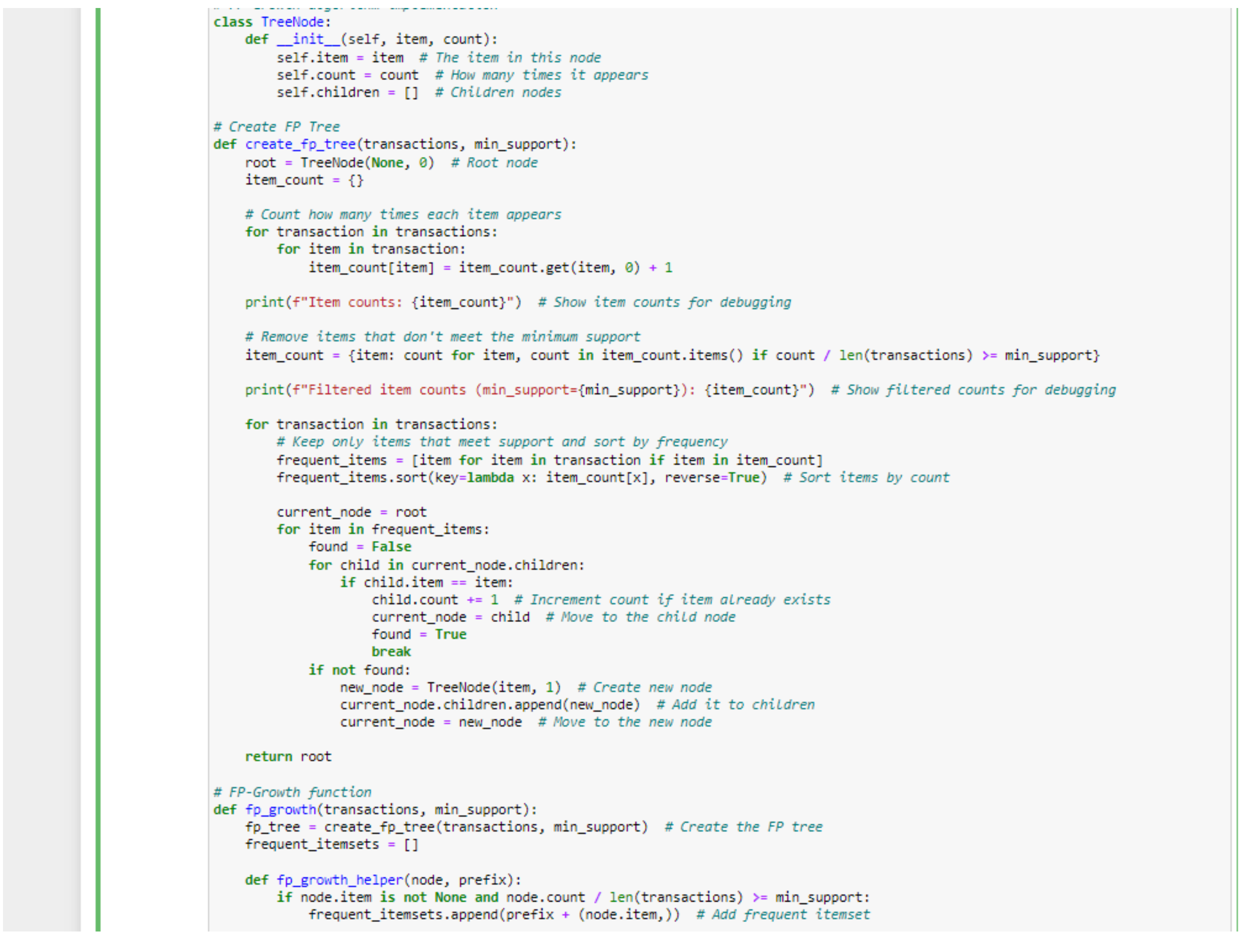
```python
def generate_itemsets_brute_force(transactions, min_support):
    items = set()
    for transaction in transactions:
        items.update(transaction)  # Collect all unique items
    frequent_itemsets = []
    k = 1
    while True:
        itemsets = list(itertools.combinations(items, k))  # Get all combinations of items of size k
        temp = []
        for itemset in itemsets:
            count = sum(1 for transaction in transactions if set(itemset).issubset(set(transaction)))
            if count / len(transactions) >= min_support:  # Check if it meets the minimum support
                temp.append(itemset)
        if not temp:  # If no more itemsets found, stop
            break
        frequent_itemsets.extend(temp)  # Add found itemsets to the list
        k += 1  # Increase size for the next round
    return frequent_itemsets
```

Below is the function code to find itemsets using Apriori algorithm



```python
# Function to find itemsets using the Apriori algorithm
def apriori_algorithm(transactions, min_support):
    items = set()
    for transaction in transactions:
        items.update(transaction)  # Collect all unique items
    frequent_itemsets = []
    k = 1
    while True:
        itemsets = list(itertools.combinations(items, k))  # Get all combinations of size k
        temp = []
        for itemset in itemsets:
            count = sum(1 for transaction in transactions if set(itemset).issubset(set(transaction)))  # Count the itemsets
            if count / len(transactions) >= min_support:  # Check support
                temp.append(itemset)
        if not temp:  # Stop if no itemsets are found
            break
        frequent_itemsets.extend(temp)  # Add found itemsets
        k += 1  # Increase size for next round
        # Prune the itemsets to speed up
        itemsets = [itemset for itemset in itemsets if all(subset in frequent_itemsets for subset in itertools.combinations(items
    return frequent_itemsets
```

Below is the function code to implement FP-Growth algorithm, Here we created the FP-tree and counted how many times each items appeared, removed the items which do not meet the minimum support.



```python
class TreeNode:
    def __init__(self, item, count):
        self.item = item  # The item in this node
        self.count = count  # How many times it appears
        self.children = []  # Children nodes

# Create FP Tree
def create_fp_tree(transactions, min_support):
    root = TreeNode(None, 0)  # Root node
    item_count = {}

    # Count how many times each item appears
    for transaction in transactions:
        for item in transaction:
            item_count[item] = item_count.get(item, 0) + 1

    print(f"Item counts: {item_count}")  # Show item counts for debugging

    # Remove items that don't meet the minimum support
    item_count = {item: count for item, count in item_count.items() if count / len(transactions) >= min_support}

    print(f"Filtered item counts (min_support={min_support}): {item_count}")  # Show filtered counts for debugging

    for transaction in transactions:
        # Keep only items that meet support and sort by frequency
        frequent_items = [item for item in transaction if item in item_count]
        frequent_items.sort(key=lambda x: item_count[x], reverse=True)  # Sort items by count

        current_node = root
        for item in frequent_items:
            found = False
            for child in current_node.children:
                if child.item == item:
                    child.count += 1  # Increment count if item already exists
                    current_node = child  # Move to the child node
                    found = True
                    break
            if not found:
                new_node = TreeNode(item, 1)  # Create new node
                current_node.children.append(new_node)  # Add it to children
                current_node = new_node  # Move to the new node

    return root

# FP-Growth function
def fp_growth(transactions, min_support):
    fp_tree = create_fp_tree(transactions, min_support)  # Create the FP tree
    frequent_itemsets = []

    def fp_growth_helper(node, prefix):
        if node.item is not None and node.count / len(transactions) >= min_support:
            frequent_itemsets.append(prefix + (node.item,))  # Add frequent itemset
```

```python
# FP-Growth function
def fp_growth(transactions, min_support):
    fp_tree = create_fp_tree(transactions, min_support)  # Create the FP tree
    frequent_itemsets = []

    def fp_growth_helper(node, prefix):
        if node.item is not None and node.count / len(transactions) >= min_support:
            frequent_itemsets.append(prefix + (node.item,))  # Add frequent itemset

        for child in node.children:
            fp_growth_helper(child, prefix + (child.item,))  # Recur for children

    fp_growth_helper(fp_tree, ())  # Start recursive function
    print(f"Frequent itemsets found by FP-Growth: {frequent_itemsets}")
    return frequent_itemsets
```

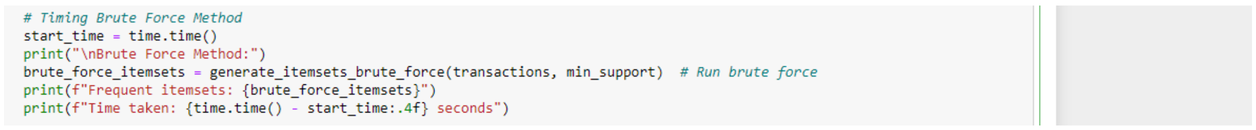
Below is the function code to generate association rules

```python
# Function to generate association rules
def generate_association_rules(frequent_itemsets, transactions, min_confidence):
    association_rules = []
    for itemset in frequent_itemsets:
        for i in range(1, len(itemset)):
            subsets = list(itertools.combinations(itemset, i))  # Get all subsets of the itemset
            for subset in subsets:
                count = sum(1 for transaction in transactions if set(subset).issubset(set(transaction)))  # Count occurrences
                confidence = count / len(transactions)  # Calculate confidence
                if confidence >= min_confidence:  # Check if it meets the confidence threshold
                    association_rules.append((subset, tuple(set(itemset) - set(subset)), confidence))
    return association_rules
```

Below is the main function for the above python code to run the program here it reads the data from csv files and get the input from the user which he/she wants to explore the itemsets from the given stores list and it will calculate the min support and confidence and all the three algorithms Brute force, Apriori algorithm, and FP-Growth for the given min support and confidence

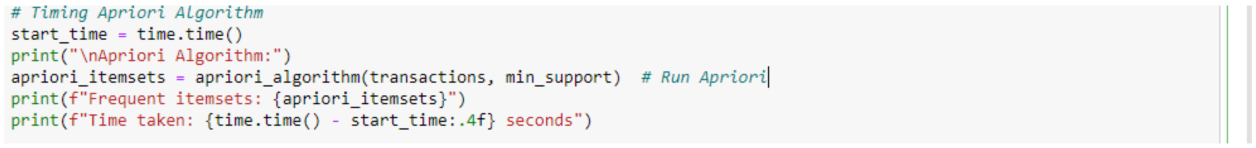
```python
# Main function
def main():
    stores = [
        "Grocery_Store_transactions.csv",
        "Electronics_Store_transactions.csv",
        "Clothing_Store_transactions.csv",
        "Bookstore_transactions.csv",
        "Home_Improvement_Store_transactions.csv"
    ]

    print("Select a store:")  # Print statement to select the store
    for i, store in enumerate(stores):
        print(f"{i + 1}. {store}")

    store_choice = int(input("Enter the store number: "))
    store_name = stores[store_choice - 1]
    transactions = load_transactions(store_name)  # Load transactions

    if transactions is None:
        return  # Exit if transactions couldn't be loaded

    min_support = int(input("Enter the minimum support threshold (1 to 100): ")) / 100.0  # Get support threshold
    min_confidence = int(input("Enter the minimum confidence threshold (1 to 100): ")) / 100.0  # Get confidence threshold
```

Below is the function code of time complexity for Brute force method



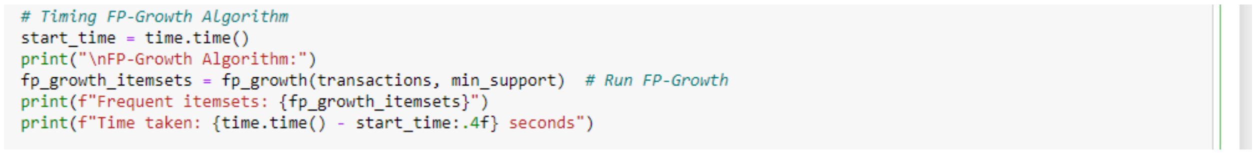
```python
# Timing Brute Force Method
start_time = time.time()
print("\nBrute Force Method:")
brute_force_itemsets = generate_itemsets_brute_force(transactions, min_support)  # Run brute force
print(f"Frequent itemsets: {brute_force_itemsets}")
print(f"Time taken: {time.time() - start_time:.4f} seconds")
```

Below is the function code of time complexity for Apriori algorithm



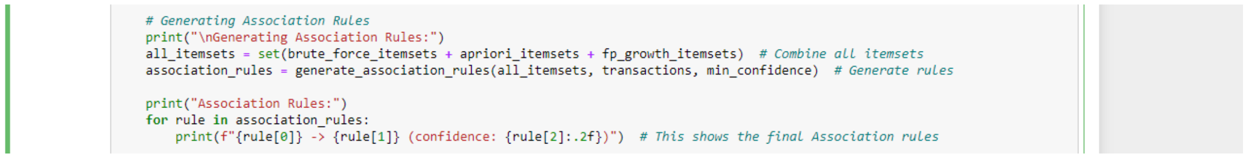
```python
# Timing Apriori Algorithm
start_time = time.time()
print("\nApriori Algorithm:")
apriori_itemsets = apriori_algorithm(transactions, min_support)  # Run Apriori
print(f"Frequent itemsets: {apriori_itemsets}")
print(f"Time taken: {time.time() - start_time:.4f} seconds")
```

Below is the function code for time complexity for FP-Growth algorithm



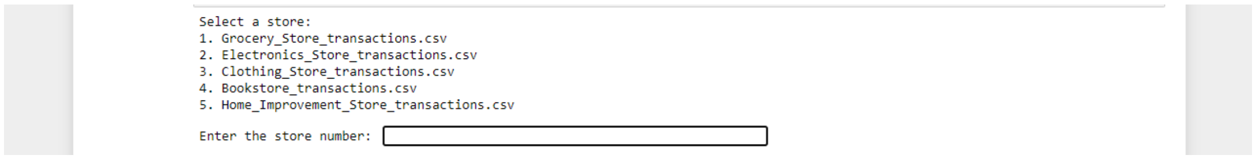
```python
# Timing FP-Growth Algorithm
start_time = time.time()
print("\nFP-Growth Algorithm:")
fp_growth_itemsets = fp_growth(transactions, min_support)  # Run FP-Growth
print(f"Frequent itemsets: {fp_growth_itemsets}")
print(f"Time taken: {time.time() - start_time:.4f} seconds")
```

Below is the function code for generating the final association rules for the items with given minimum support and confidence



```python
# Generating Association Rules
print("\nGenerating Association Rules:")
all_itemsets = set(brute_force_itemsets + apriori_itemsets + fp_growth_itemsets)  # Combine all itemsets
association_rules = generate_association_rules(all_itemsets, transactions, min_confidence)  # Generate rules

print("Association Rules:")
for rule in association_rules:
    print(f"{rule[0]} -> {rule[1]} (confidence: {rule[2]:.2f})")  # This shows the final Association rules
```

Below are the screenshots of the output to show that the function code is running properly without any errors

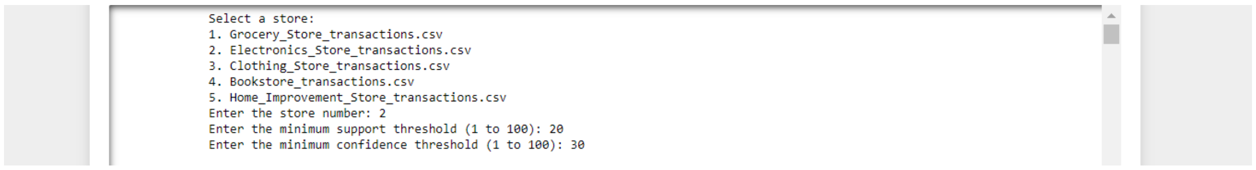


```
Select a store:
1. Grocery_Store_transactions.csv
2. Electronics_Store_transactions.csv
3. Clothing_Store_transactions.csv
4. Bookstore_transactions.csv
5. Home_Improvement_Store_transactions.csv

Enter the store number:
```

Step-1: Here I am going to select the store 2

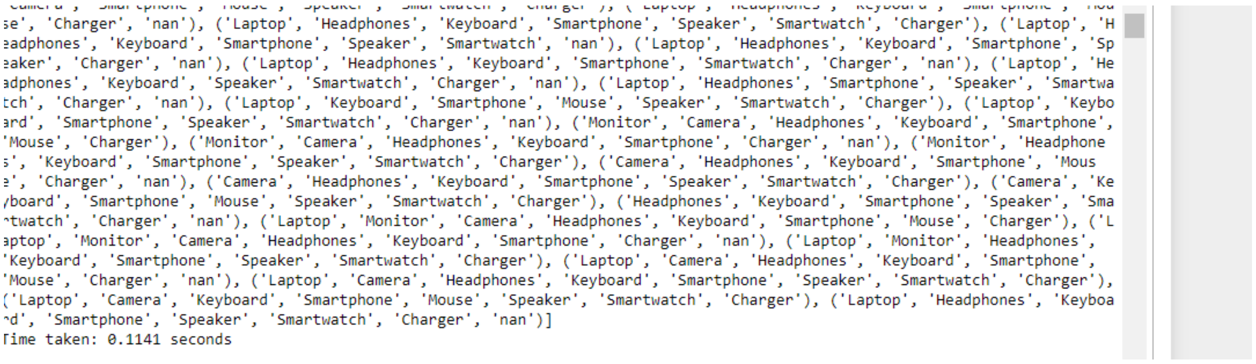
Step-2: Now I am going to give the minimum support as 20 and confidence as 30

Step-3: The program generates the frequent itemsets with the help of brute force, Apriori algorithm and FP-Growth method and give the time complexity for every algorithm how much time it takes to run then at the end it will give the association rules



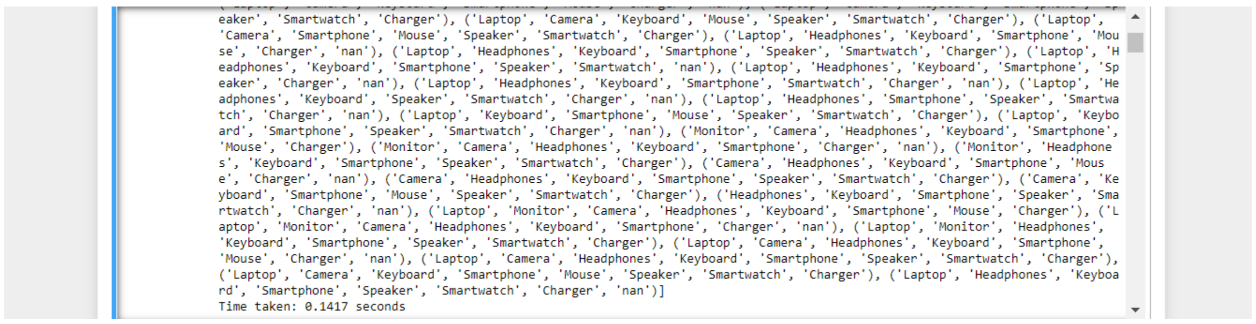
```
Select a store:
1. Grocery_Store_transactions.csv
2. Electronics_Store_transactions.csv
3. Clothing_Store_transactions.csv
4. Bookstore_transactions.csv
5. Home_Improvement_Store_transactions.csv
Enter the store number: 2
Enter the minimum support threshold (1 to 100): 20
Enter the minimum confidence threshold (1 to 100): 30
```

Step-4: The below are the screenshots of final output of brute force method items and time complexity



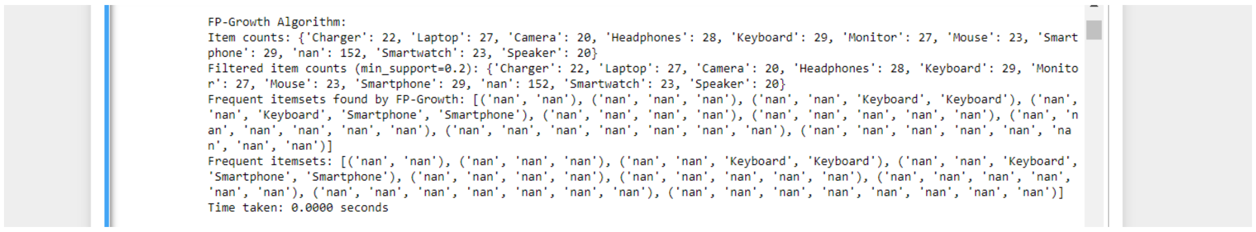
```
se', 'Charger', 'nan'), ('Laptop', 'Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'H
eadphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'nan'), ('Laptop', 'Headphones', 'Keyboard', 'Smartphone', 'Sp
eaker', 'Charger', 'nan'), ('Laptop', 'Headphones', 'Keyboard', 'Smartphone', 'Smartwatch', 'Charger', 'nan'), ('Laptop', 'He
adphones', 'Keyboard', 'Speaker', 'Smartwatch', 'Charger', 'nan'), ('Laptop', 'Headphones', 'Smartphone', 'Speaker', 'Smartwa
tch', 'Charger', 'nan'), ('Laptop', 'Keyboard', 'Smartphone', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'Keybo
ard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger', 'nan'), ('Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone',
'Mouse', 'Charger'), ('Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Charger', 'nan'), ('Monitor', 'Headphone
s', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Mous
e', 'Charger', 'nan'), ('Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Camera', 'Ke
yboard', 'Smartphone', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Sma
rtwatch', 'Charger', 'nan'), ('Laptop', 'Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Mouse', 'Charger'), ('L
aptop', 'Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Charger', 'nan'), ('Laptop', 'Monitor', 'Headphones',
'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'Camera', 'Headphones', 'Keyboard', 'Smartphone',
'Mouse', 'Charger', 'nan'), ('Laptop', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'),
('Laptop', 'Camera', 'Keyboard', 'Smartphone', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'Headphones', 'Keyboa
rd', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger', 'nan')]
Time taken: 0.1141 seconds
```

Step-5 The below is the screenshot of final output of Apriori algorithm and time complexity



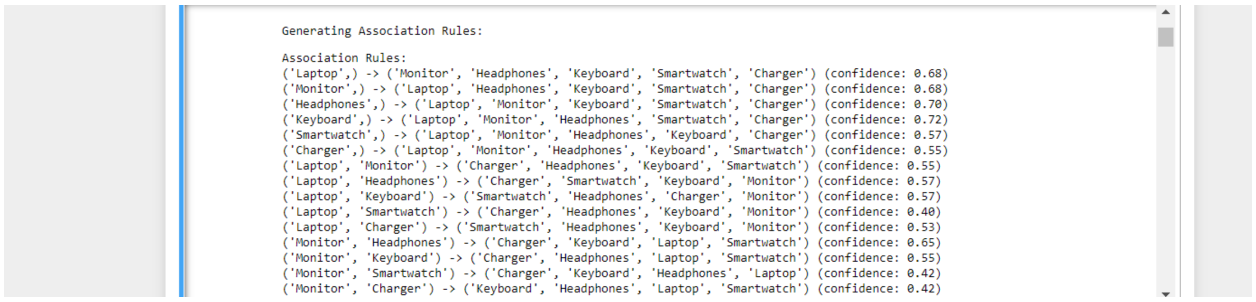
```
eaker', 'Smartwatch', 'Charger'), ('Laptop', 'Camera', 'Keyboard', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop',
'Camera', 'Smartphone', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'Headphones', 'Keyboard', 'Smartphone', 'Mou
se', 'Charger', 'nan'), ('Laptop', 'Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'H
eadphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'nan'), ('Laptop', 'Headphones', 'Keyboard', 'Smartphone', 'Sp
eaker', 'Charger', 'nan'), ('Laptop', 'Headphones', 'Keyboard', 'Smartphone', 'Smartwatch', 'Charger', 'nan'), ('Laptop', 'He
adphones', 'Keyboard', 'Speaker', 'Smartwatch', 'Charger', 'nan'), ('Laptop', 'Headphones', 'Smartphone', 'Speaker', 'Smartwa
tch', 'Charger', 'nan'), ('Laptop', 'Keyboard', 'Smartphone', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'Keybo
ard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger', 'nan'), ('Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone',
'Mouse', 'Charger'), ('Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Charger', 'nan'), ('Monitor', 'Headphone
s', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Mous
e', 'Charger', 'nan'), ('Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Camera', 'Ke
yboard', 'Smartphone', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Sma
rtwatch', 'Charger', 'nan'), ('Laptop', 'Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Mouse', 'Charger'), ('L
aptop', 'Monitor', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Charger', 'nan'), ('Laptop', 'Monitor', 'Headphones',
'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'Camera', 'Headphones', 'Keyboard', 'Smartphone',
'Mouse', 'Charger', 'nan'), ('Laptop', 'Camera', 'Headphones', 'Keyboard', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger'),
('Laptop', 'Camera', 'Keyboard', 'Smartphone', 'Mouse', 'Speaker', 'Smartwatch', 'Charger'), ('Laptop', 'Headphones', 'Keyboa
rd', 'Smartphone', 'Speaker', 'Smartwatch', 'Charger', 'nan')]
Time taken: 0.1417 seconds
```

Step-6 The below is the screenshot of the final output of FP-Growth and time complexity



```
FP-Growth Algorithm:
Item counts: {'Charger': 22, 'Laptop': 27, 'Camera': 20, 'Headphones': 28, 'Keyboard': 29, 'Monitor': 27, 'Mouse': 23, 'Smart
phone': 29, 'nan': 152, 'Smartwatch': 23, 'Speaker': 20}
Filtered item counts (min_support=0.2): {'Charger': 22, 'Laptop': 27, 'Camera': 20, 'Headphones': 28, 'Keyboard': 29, 'Monito
r': 27, 'Mouse': 23, 'Smartphone': 29, 'nan': 152, 'Smartwatch': 23, 'Speaker': 20}
Frequent itemsets found by FP-Growth: [('nan', 'nan'), ('nan', 'nan', 'nan'), ('nan', 'nan', 'Keyboard', 'Keyboard'), ('nan',
'nan', 'Keyboard', 'Smartphone', 'Smartphone'), ('nan', 'nan', 'nan', 'nan'), ('nan', 'nan', 'nan', 'nan', 'nan'), ('nan', 'n
an', 'nan', 'nan', 'nan', 'nan'), ('nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'), ('nan', 'nan', 'nan', 'nan', 'nan', 'na
n', 'nan', 'nan')]
Frequent itemsets: [('nan', 'nan'), ('nan', 'nan', 'nan'), ('nan', 'nan', 'Keyboard', 'Keyboard'), ('nan', 'nan', 'Keyboard',
'Smartphone', 'Smartphone'), ('nan', 'nan', 'nan', 'nan'), ('nan', 'nan', 'nan', 'nan', 'nan'), ('nan', 'nan', 'nan', 'nan',
'nan', 'nan'), ('nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'), ('nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan')]
Time taken: 0.0000 seconds
```

Step-7 The below is the output for association rules



```
Generating Association Rules:

Association Rules:
('Laptop',) -> ('Monitor', 'Headphones', 'Keyboard', 'Smartwatch', 'Charger') (confidence: 0.68)
('Monitor',) -> ('Laptop', 'Headphones', 'Keyboard', 'Smartwatch', 'Charger') (confidence: 0.68)
('Headphones',) -> ('Laptop', 'Monitor', 'Keyboard', 'Smartwatch', 'Charger') (confidence: 0.70)
('Keyboard',) -> ('Laptop', 'Monitor', 'Headphones', 'Smartwatch', 'Charger') (confidence: 0.72)
('Smartwatch',) -> ('Laptop', 'Monitor', 'Headphones', 'Keyboard', 'Charger') (confidence: 0.57)
('Charger',) -> ('Laptop', 'Monitor', 'Headphones', 'Keyboard', 'Smartwatch') (confidence: 0.55)
('Laptop', 'Monitor') -> ('Charger', 'Headphones', 'Keyboard', 'Smartwatch') (confidence: 0.55)
('Laptop', 'Headphones') -> ('Charger', 'Smartwatch', 'Keyboard', 'Monitor') (confidence: 0.57)
('Laptop', 'Keyboard') -> ('Smartwatch', 'Headphones', 'Charger', 'Monitor') (confidence: 0.57)
('Laptop', 'Smartwatch') -> ('Charger', 'Headphones', 'Keyboard', 'Monitor') (confidence: 0.40)
('Laptop', 'Charger') -> ('Smartwatch', 'Headphones', 'Keyboard', 'Monitor') (confidence: 0.53)
('Monitor', 'Headphones') -> ('Charger', 'Keyboard', 'Laptop', 'Smartwatch') (confidence: 0.65)
('Monitor', 'Keyboard') -> ('Charger', 'Headphones', 'Laptop', 'Smartwatch') (confidence: 0.55)
('Monitor', 'Smartwatch') -> ('Charger', 'Keyboard', 'Headphones', 'Laptop') (confidence: 0.42)
('Monitor', 'Charger') -> ('Keyboard', 'Headphones', 'Laptop', 'Smartwatch') (confidence: 0.42)
```

Steps to install the packages and run the program

The packages required to install are:

Import pandas

Import itertools

Import time

The run steps as follows:

Step-1: Install jupyter notebook.

Step-2: Open jupyter notebook, install the following above packages with command “pip install” For all the above packages.

Step-3: Now after running the code please enter the store number from the given stores 1 to 5, Then please enter the minimum support and confidence for getting the frequent items datsets using all three algorithms.

Step-4: you will get the final output.

Others

The python file(.pynb) and data sets .csv files attached to the folder file

Link to Git Repository

https://github.com/mazeenhussiansyed/Apriorialgorithm