NAME: Saketh Lakshmanan Sathiskumar
NJIT UCID: sl778
Email Address: sl778@njit.edu
03-10-2024
Professor: Yasser Abduallah
CS 634-104 Data Mining

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm Implementation in Retail Data Mining

Abstract:
In this project, I delve into the Brute Force Algorithm, Apriori Algorithm, and FPGrowth Algorithm which are fundamental tools in the realm of data mining, aiming to unveil associations within retail transactions. My exploration involves the practical implementation of these algorithms and the application of diverse data mining concepts, principles, and methodologies. By thoroughly assessing its effectiveness and efficiency, I seek to understand how well the above algorithms perform in identifying patterns within the given retail transaction data.

Introduction:
Data mining proves to be a robust methodology for revealing concealed patterns and associations within vast datasets. My project centers around the Brute Force Algorithm, Apriori Algorithm, and FP Growth Algorithm which are classic techniques specifically designed for association rule mining, with a particular emphasis on its practical application in a retail context.

My primary focus lies in the generation and presentation of association rules, a crucial aspect of the three algorithms above. In this implementation, I applied the three algorithms to five datasets associated with a retail store, allowing me to find frequent itemsets and association rules. The key steps in this process included:

- ➔ Loading the datasets and collecting user input for minimum support and confidence
- ➔ Initializing sets for candidate and frequent items.
- ➔ Cleaning up the dataset to ensure item order and uniqueness.
- ➔ Iteratively generating candidate itemsets and updating frequent itemsets using the Brute Force algorithm
- ➔ Generating the association rules for the Brute Force attempt
- ➔ Verifying the Brute Force attempts with Apriori and FP Growth python packages

Core Concepts and Principles:

Frequent Itemset Discovery:
At its core, the Apriori Algorithm is centered on the identification of frequent itemsets – groups of items that recurrently appear together in transactions. These itemsets serve as valuable indicators, offering insights into customer purchasing patterns and preferences.

Support and Confidence:
In data mining, two pivotal metrics come into play: support and confidence. Support gauges the frequency of occurrence for a specific item or itemset, while confidence evaluates the probability of items being bought together. These metrics serve as important guides in shaping and directing my analysis.

Association Rules:
Through the identification of robust association rules, items that are frequently bought together become apparent. These rules play an important role in refining sales strategies, particularly in the context of recommendations.

Project Workflow:
My project adheres to a methodical workflow, encompassing distinct stages and incorporating the utilization of the Brute Force Algorithm, Apriori Algorithm, and FP Growth Algorithm

Data Loading and Preprocessing:
The user is prompted to enter a dataset choice among Amazon, Best Buy, Kmart, Nike, or General. Based on the user's input, the code loads two CSV files for each specific dataset: one for the itemset list (itemset_list) and one for the transactions (the_transactions). The datasets are loaded using pd.read_csv() from the pandas library. The code initializes an empty list called dataset to store preprocessed transactions. The resulting dataset is a list of transactions, where each transaction is a sorted and unique list of items based on the specified order. This preprocessed dataset is then used for further analysis.

Determination of Minimum Support and Confidence:
User input was a main priority in this project. I collect the user's preferences for minimum support and confidence levels to get rid of less significant patterns.

Iteration Through Candidate Itemsets:
The code generates candidate itemsets by iterating through pairs of frequent itemsets, joining them, and adding the resulting sets to the list of candidates. The order of items is used in the process of joining itemsets. The function encapsulates the logic of candidate generation in the Brute Force algorithm. We start with itemset size K = 1 and go on to K = 2, K = 3, etc. This iterative process involves a "brute force" method of generating all possible itemset combinations.

Support Count Calculation:
For every candidate itemset, we determine its support by tallying the number of transactions that include the itemset. Itemsets exceeding the minimum support threshold are kept, whereas those falling below are discarded.

Confidence Calculation:
The code assesses the confidence of association rules, measuring the robustness of associations between items. This is done by comparing the support values for each item and itemsets.

Association Rule Generation:
In the code, the association rules are generated by iterating through different itemset sizes and their subsets. For each itemset, all possible non-empty subsets are considered. The confidence of each association rule is calculated by comparing the support values of the whole itemset and its subsets. If the confidence and support meet user-defined thresholds, the association rule is formatted and added to the final result. This process continues for all itemsets and subsets, resulting in a list of association rules.

Results and Evaluation:
I assess the project's effectiveness and efficiency using performance metrics like support, confidence, and the derived association rules. Additionally, the code gauges the reliability of my custom Brute Force Algorithm by comparing it with implementations from the Apriori and FP Growth libraries. **(PLEASE SEE END OF PAPER FOR COMPARISONS AND EVALUATION)**

Conclusion:
In summary, my project exemplifies the practical application of data mining concepts, principles, and methods. I achieved successful implementation of the Brute Force Algorithm, extracting meaningful association rules from retail transaction data. The brute force approach, strict adherence to user-defined parameters, and verification from the python libraries collectively showcase the efficacy of data mining in uncovering valuable patterns for informed decision-making in the retail industry.

Screenshots

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

Figure 1 : Amazon Item Names CSV file	Figure 2 : Amazon Transactions CSV file
Figure 3 : Best Buy Item Names CSV file	Figure 4 : Best Buy Transactions CSV file
Figure 5 : K-Mart Item Names CSV file	Figure 6 : K-Mart Transactions CSV file
Figure 7 : Nike Item Names CSV file	Figure 8 : Nike Transactions CSV file
Figure 9 : General Item Names CSV file	Figure 10 : General Transactions CSV file

Figure 1:

```
ItemNo,ItemName
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
```

Figure 2:

```
TransactionID,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"
```

Figure 3:

```
ItemNo,ItemName
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
```

Figure 4:

```
TransactionID,Transaction
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"
```

Figure 5:

```
ItemNo,ItemName
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
```

Figure 6:

```
TransactionID,Transaction
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"
```

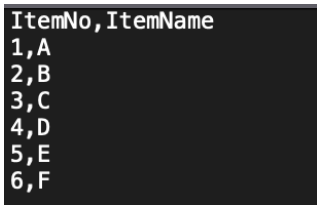
Figure 7:

```
ItemNo,ItemName
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
```

Figure 8:

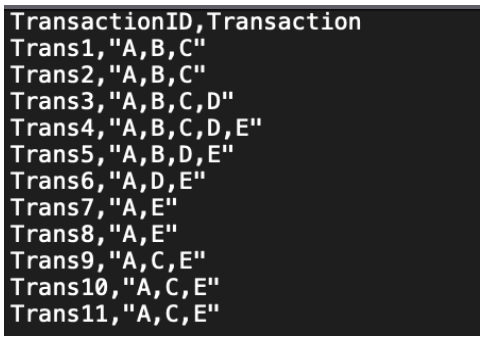
```
TransactionID,Transaction
Trans1,"Running Shoe,Socks,Sweatshirts,Modern Pants"
Trans2,"Running Shoe,Socks,Sweatshirts"
Trans3,"Running Shoe,Socks,Sweatshirts,Modern Pants"
Trans4,"Running Shoe,Sweatshirts,Modern Pants"
Trans5,"Running Shoe,Socks,Sweatshirts,Modern Pants,Soccer Shoe"
Trans6,"Running 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 Fit V-Nick"
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,Rash Guard,Tech Pants,Hoodies,Dry Fit V-Nick"
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"
```

Figure 9:



```
ItemNo,ItemName
1,A
2,B
3,C
4,D
5,E
6,F
```

Figure 10:



```
TransactionID,Transaction
Trans1,"A,B,C"
Trans2,"A,B,C"
Trans3,"A,B,C,D"
Trans4,"A,B,C,D,E"
Trans5,"A,B,D,E"
Trans6,"A,D,E"
Trans7,"A,E"
Trans8,"A,E"
Trans9,"A,C,E"
Trans10,"A,C,E"
Trans11,"A,C,E"
```

These datasets were taken from the example datasets provided by the instructor Dr. Yasser Abdullah.

Below are screenshots of the code from python file:

First we write code for the BRUTE FORCE ALGORITHM

Load and collect all data

```
import csv
import sys
import numpy as np
from itertools import chain, combinations
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

def load_csv(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        return list(reader)

def powerset(s):
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+1))
```

```
list_input = input("Enter a dataset to analyze: Amazon, Best Buy, Kmart, Nike, or General: ")
if list_input.lower() == 'amazon':
    itemset_list = pd.read_csv('amazonlist.csv')
    the_transactions = pd.read_csv('amazontransactions.csv')
elif list_input.lower() == 'best buy':
    itemset_list = pd.read_csv('bestbuylist.csv')
    the_transactions = pd.read_csv('bestbuytransactions.csv')
elif list_input.lower() in ['kmart', 'k-mart']:
    itemset_list = pd.read_csv('kmartlist.csv')
    the_transactions = pd.read_csv('kmarttransactions.csv')
elif list_input.lower() == 'nike':
    itemset_list = pd.read_csv('nikelist.csv')
    the_transactions = pd.read_csv('niketransactions.csv')
elif list_input.lower() == 'general':
    itemset_list = pd.read_csv('generallist.csv')
    the_transactions = pd.read_csv('generaltransactions.csv')
else:
    print("Enter Valid Info (correct spelling)")
    sys.exit()

order = sorted(itemset_list['ItemName'])

dataset = []
for lines in the_transactions['Transaction']:
    trans = list(lines.strip().split(','))
    trans1 = list(np.unique(trans))
    trans1.sort(key=lambda x: order.index(x.strip()))
    dataset.append(sorted(trans1))

transaction_num = len(dataset)

min_support = int(input("Enter the minimum support count (integer, percent between 1 and 100): "))
min_confidence = float(input("Enter the minimum confidence (decimal, percent between 0 and 100): "))
```

Output:

```
Enter a dataset to analyze: Amazon, Best Buy, Kmart, Nike, or General:  Nike
Enter the minimum support count (integer, percent between 1 and 100):  50
Enter the minimum confidence (decimal, percent between 0 and 100):  30
```

Initialize the dictionaries

```
candidate_itemsets = {}
freq_itemsets = {}
support_count_L = {}
itemset_size = 1     # set the initial itemset size
non_frequent = {itemset_size: []}

# populate candidate itemsets with singleton itemsets
candidate_itemsets[itemset_size] = [[f] for f in order]

# print the initialized candidate dictionary for verification
print("Candidate Itemsets:", candidate_itemsets)
```

Output:

```
Candidate Itemsets: {1: [['Dry Fit V-Nick'], ['Hoodies'], ['Modern Pants'], ['Rash Guard'], ['Running Shoe'], ['Soccer Shoe'], ['Socks'], ['Sw
eatshirts'], ['Swimming Shirt'], ['Tech Pants']]}
```

Define a function for finding frequent itemsets

```
def frequent_itemsets(candidate_itemsets, transactions, minimum_support, non_frequent):
    # Initialize lists to store frequent itemsets, their support counts, and new non-frequent itemsets
    frequent_itemsets_list = []
    support_count_list = []
    new_non_frequent = []

    # Get the total number of transactions in the dataset
    transaction_num = len(transactions)
    # Get the current itemset size
    K = len(non_frequent.keys())

    # Iterate through each candidate itemset
    for i in range(0, len(candidate_itemsets)):
        # Flag to check if the candidate itemset is a subset of any non-frequent itemset
        is_candidate_set = 0

        # Check against non-frequent itemsets of the previous size (if any)
        if K > 0:
            for j in non_frequent[K]:
                if set(j).issubset(set(candidate_itemsets[i])):
                    is_candidate_set = 1
                    break

        # If the candidate itemset is not a subset of any non-frequent itemset, proceed
        if is_candidate_set == 0:
            # Count the occurrences of the candidate itemset in the transactions
            frequent_count = count_items(candidate_itemsets[i], transactions)

            # Check if the frequent count meets the minimum support threshold
            if frequent_count >= (minimum_support / 100) * transaction_num:
                # Append the frequent itemset and its support count to the lists
                frequent_itemsets_list.append(candidate_itemsets[i])
                support_count_list.append(frequent_count)
            else:
                # If not frequent, add the itemset to the list of new non-frequent itemsets
                new_non_frequent.append(candidate_itemsets[i])

    # Return the lists containing frequent itemsets, their support counts, and new non-frequent itemsets
    return frequent_itemsets_list, support_count_list, new_non_frequent
```

Define a function for counting items and a table

```
def count_items(candidate_itemset, transactions):
    count = 0
    for i in range (0, len(transactions)):
        if set(candidate_itemset).issubset(set(transactions[i])):
            count += 1
    return count
```

```
def print_table (table, support_count):
    print("Itemset | Count")
    for i in range (0, len(table)):
        print("{} : {}".format(table[i], support_count[i]))
    print("\n\n")
```

Define a function to link together itemsets

```
def join_itemsets(itemset1, itemset2, order):
    itemset1.sort(key=lambda x: order.index(x))
    itemset2.sort(key=lambda x: order.index(x))

    for i in range(len(itemset1) - 1):
        if itemset1[i] != itemset2[i]:
            return []

    if order.index(itemset1[-1]) < order.index(itemset2[-1]):
        return itemset1 + [itemset2[-1]]

    return []
```

Define a function to find the candidate itemset

```
def get_candidate_set(frequent_itemsets, order):
    candidate_set = []
    for i in range(len(frequent_itemsets)):
        for j in range(i + 1, len(frequent_itemsets)):
            joined_itemset = join_itemsets(frequent_itemsets[i], frequent_itemsets[j], order)

            if len(joined_itemset) > 0:
                candidate_set.append(joined_itemset)

    return candidate_set
```

Define a function to find all possible subsets of a set

```
from itertools import combinations, chain

def possible_subsets(s):
    a = list(s)
    subsets = list(chain.from_iterable(combinations(a, r) for r in range(1, len(a) + 1)))
    return subsets
```

Rule formatting to look nicer

```
def write_association_rules(lhs_itemset, rhs_itemset, confidence, support, transaction_num, rule_number):
    association_rules_str = (
        f"Rule {rule_number}: {list(lhs_itemset)} -> {list(rhs_itemset)}\n"
        f"Confidence: {confidence * 100:.2f}%\n"
        f"Support: {(support / transaction_num) * 100:.2f}%\n\n"
    )
    return association_rules_str
```

Find the frequent itemsets and the corresponding support then update

```
frequent_set, support, new_non_frequent = frequent_itemsets(candidate_itemsets[itemset_size], dataset, min_support, non_frequent)

freq_itemsets[itemset_size] = frequent_set
non_frequent[itemset_size] = new_non_frequent
support_count_L[itemset_size] = support
```

Define and print a table for itemsets

```
def print_itemset_table(itemset_type, itemsets, support_counts):
    print(f"\nTable {itemset_type}:\n")
    for i, itemset in enumerate(itemsets):
        itemset_str = ','.join(itemset)
        count = support_counts[i]
        print(f"{itemset_str} : {count}")
```

```
print_itemset_table("Candidate", candidate_itemsets[1], [count_items(item, dataset) for item in candidate_itemsets[1]])
print_itemset_table("Frequent", freq_itemsets[1], support_count_L[1])
```

Output:

```
Table Candidate:

Dry Fit V-Nick : 10
Hoodies : 9
Modern Pants : 9
Rash Guard : 12
Running Shoe : 14
Soccer Shoe : 6
Socks : 12
Sweatshirts : 11
Swimming Shirt : 11
Tech Pants : 9
```

```
Table Frequent:

Dry Fit V-Nick : 10
Rash Guard : 12
Running Shoe : 14
Socks : 12
Sweatshirts : 11
Swimming Shirt : 11
```

Generate candidate itemsets iteratively using the BRUTE FORCE ALGORITHM

```
start_time = time.time() ## Starting the time calc here
# Initialize variables for the brute-force approach
K = itemset_size + 1  # Increment the itemset size
candidate_set = 0  # Flag to check if there are frequent itemsets
candidate_itemsets = []  # List to store candidate itemsets at each iteration

# Continue generating candidate itemsets until none are frequent for the current itemset size
while candidate_set == 0:
    # Generate candidate itemsets using the brute-force method
    candid_itemsets = get_candidate_set(freq_itemsets[K-1], order)
    # Append the generated candidate itemsets to the list
    candidate_itemsets.append(candid_itemsets)
    print("Table Candidate Itemsets {}: \n".format(K))
    # Print the table of candidate itemsets along with their support counts
    print_table(candid_itemsets, [count_items(item, dataset) for item in candid_itemsets])

    # Calculate frequent itemsets and update dictionaries
    frequent_set, support, new_non_frequent = frequent_itemsets(candidate_itemsets[-1], dataset, min_support, non_frequent)
    # Update dictionaries with the frequent itemsets, non-frequent itemsets, and support counts
    freq_itemsets.update({K: frequent_set})
    non_frequent.update({K: new_non_frequent})
    support_count_L.update({K: support})

    # Check if there are no frequent itemsets for the current itemset size
    if len(freq_itemsets[K]) == 0:
        candidate_set = 1  # Set the flag to terminate the loop
    else:
        print(f"Table Frequent Itemsets {K}:\n")
        # Print the table of frequent itemsets along with their support counts
        print_table(freq_itemsets[K], support_count_L[K])

    K += 1  # Increment the itemset size for the next iteration
```

Output:

```
Table Candidate Itemsets 2:

Itemset | Count
['Dry Fit V-Nick', 'Rash Guard'] : 10
['Dry Fit V-Nick', 'Running Shoe'] : 5
['Dry Fit V-Nick', 'Socks'] : 4
['Dry Fit V-Nick', 'Sweatshirts'] : 3
['Dry Fit V-Nick', 'Swimming Shirt'] : 9
['Rash Guard', 'Running Shoe'] : 7
['Rash Guard', 'Socks'] : 5
['Rash Guard', 'Sweatshirts'] : 4
['Rash Guard', 'Swimming Shirt'] : 10
['Running Shoe', 'Socks'] : 10
['Running Shoe', 'Sweatshirts'] : 10
['Running Shoe', 'Swimming Shirt'] : 6
['Socks', 'Sweatshirts'] : 10
['Socks', 'Swimming Shirt'] : 4
['Sweatshirts', 'Swimming Shirt'] : 4
```

```
Table Frequent Itemsets 2:

Itemset | Count
['Dry Fit V-Nick', 'Rash Guard'] : 10
['Rash Guard', 'Swimming Shirt'] : 10
['Running Shoe', 'Socks'] : 10
['Running Shoe', 'Sweatshirts'] : 10
['Socks', 'Sweatshirts'] : 10



Table Candidate Itemsets 3:

Itemset | Count
['Running Shoe', 'Socks', 'Sweatshirts'] : 9
```

Generate the final association rules for BRUTE FORCE ALGORITHM

```
# Initialize variables for final association rules
final_association_rules = ""
rule_number = 1

# Loop over each itemset size (starting from 1) in the frequent itemsets
for itemset_size in range(1, len(freq_itemsets)):
    # Loop over each itemset in the current itemset size
    for itemset in freq_itemsets[itemset_size]:
        # Generate all possible non-empty subsets of the current itemset
        subsets = list(possible_subsets(set(itemset)))
        subsets.pop()  # Remove the empty set

        # Loop over each subset of the current itemset
        for subset in subsets:
            # Create item1 as a set representing the current subset
            item1 = set(subset)
            # Convert the current itemset to a set
            itemset_set = set(itemset)
            # Generate item2 as the complement of item1 in the itemset
            item2 = itemset_set - item1

            # Calculate support counts for the itemsets and subsets
            support_frequent_set = count_items(itemset_set, dataset)
            support_item1 = count_items(item1, dataset)
            support_item2 = count_items(item2, dataset)

            # Calculate confidence of the association rule
            confidence = support_frequent_set / support_item1

            # Check if the confidence and support meet the user-defined thresholds
            if confidence >= (min_confidence / 100) and support_frequent_set >= (min_support / 100) * transaction_num:
                # Generate and append the association rule to the final result
                final_association_rules += write_association_rules(item2, item1, confidence, support_frequent_set, transaction_num, rule_numbe
                # Increment the rule number for the next association rule
                rule_number += 1

end_time = time.time() ## Ending the time calc here
elapsed_time = end_time - start_time

# Print the final association rules
print("Final Association Rules: \n")
print(final_association_rules)
print(f"\nProgram execution time: {elapsed_time:.4f} seconds")
```

Output:

```
Final Association Rules:

Rule 1: ['Rash Guard'] -> ['Dry Fit V-Nick']
Confidence: 100.00%
Support: 50.00%

Rule 2: ['Dry Fit V-Nick'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 3: ['Rash Guard'] -> ['Swimming Shirt']
Confidence: 90.91%
Support: 50.00%

Rule 4: ['Swimming Shirt'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 5: ['Socks'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 6: ['Running Shoe'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%

Rule 7: ['Sweatshirts'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 8: ['Running Shoe'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%

Rule 9: ['Sweatshirts'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%

Rule 10: ['Socks'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%


Program execution time: 0.9253 seconds
```

Verify the results with APRIORI built-in package

```
from mlxtend.frequent_patterns import apriori as mlxtend_apriori
from mlxtend.frequent_patterns import association_rules as mlxtend_association_rules
from mlxtend.preprocessing import TransactionEncoder

start_time = time.time() ## Starting the time calc here

# Use Apriori implementation from mlxtend
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
mlxtend_freq_itemsets = mlxtend_apriori(df, min_support=(min_support / 100), use_colnames=True)

# Extract association rules from frequent itemsets using mlxtend
mlxtend_rules = mlxtend_association_rules(mlxtend_freq_itemsets, metric="confidence", min_threshold=min_confidence / 100)

# Initialize variables for final association rules (mlxtend)
final_association_rules_mlxtend = ""
rule_number_mlxtend = 1

# Loop over each rule generated by mlxtend
for idx, row in mlxtend_rules.iterrows():
    # Extract items and details from mlxtend result
    item1_mlxtend = set(row['antecedents'])
    item2_mlxtend = set(row['consequents'])
    confidence_mlxtend = row['confidence']
    support_mlxtend = row['support'] * transaction_num  # Convert support to count

    # Check if the confidence and support meet the user-defined thresholds
    if confidence_mlxtend >= (min_confidence / 100) and support_mlxtend >= (min_support / 100) * transaction_num:
        # Generate and append the association rule to the final result (mlxtend)
        final_association_rules_mlxtend += write_association_rules(item2_mlxtend, item1_mlxtend, confidence_mlxtend, support_mlxtend, transact
        # Increment the rule number for the next association rule
        rule_number_mlxtend += 1

end_time = time.time() ## Ending the time calc here
elapsed_time = end_time - start_time

# Print the final association rules from mlxtend
print("Final Association Rules (Apriori): \n")
print(final_association_rules_mlxtend)
print(f"\nProgram execution time: {elapsed_time:.4f} seconds")
```

Output:

```
Final Association Rules (mlxtend):

Rule 1: ['Rash Guard'] -> ['Dry Fit V-Nick']
Confidence: 100.00%
Support: 50.00%

Rule 2: ['Dry Fit V-Nick'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 3: ['Rash Guard'] -> ['Swimming Shirt']
Confidence: 90.91%
Support: 50.00%

Rule 4: ['Swimming Shirt'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 5: ['Socks'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 6: ['Running Shoe'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%

Rule 7: ['Sweatshirts'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 8: ['Running Shoe'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%

Rule 9: ['Sweatshirts'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%

Rule 10: ['Socks'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%
```

Verify Results using FP GROWTH built-in package

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules

start_time = time.time() ## Starting the time calc here

# Use TransactionEncoder for one-hot encoding
te = TransactionEncoder()
one_hot_df = te.fit_transform(dataset)
one_hot_df = pd.DataFrame(one_hot_df, columns=te.columns_)

# Find frequent itemsets using FPGrowth
frequent_itemsets_mlxtend = fpgrowth(one_hot_df, min_support=min_support / 100, use_colnames=True)
print(f"\nFrequent Itemsets using FPGrowth:")
print(frequent_itemsets_mlxtend)

# Generate association rules
rules_mlxtend = association_rules(frequent_itemsets_mlxtend, metric="confidence", min_threshold=min_confidence / 100)

end_time = time.time() ## Ending the time calc here
elapsed_time = end_time - start_time

print("\nAssociation Rules using FPGrowth:")
print(rules_mlxtend)
print(f"\nProgram execution time: {elapsed_time:.4f} seconds")
```

Output:

```
Frequent Itemsets using FPGrowth:
    support                      itemsets
0      0.70                (Running Shoe)
1      0.60                       (Socks)
2      0.55                  (Sweatshirts)
3      0.60                   (Rash Guard)
4      0.55               (Swimming Shirt)
5      0.50               (Dry Fit V-Nick)
6      0.50          (Running Shoe, Socks)
7      0.50           (Socks, Sweatshirts)
8      0.50    (Running Shoe, Sweatshirts)
9      0.50   (Swimming Shirt, Rash Guard)
10     0.50   (Dry Fit V-Nick, Rash Guard)

Association Rules using FPGrowth:
        antecedents       consequents  antecedent support  consequent support  \
0    (Running Shoe)           (Socks)                0.70                0.60
1           (Socks)    (Running Shoe)                0.60                0.70
2           (Socks)     (Sweatshirts)                0.60                0.55
3     (Sweatshirts)           (Socks)                0.55                0.60
4    (Running Shoe)     (Sweatshirts)                0.70                0.55
5     (Sweatshirts)    (Running Shoe)                0.55                0.70
6  (Swimming Shirt)      (Rash Guard)                0.55                0.60
7      (Rash Guard)  (Swimming Shirt)                0.60                0.55
8  (Dry Fit V-Nick)      (Rash Guard)                0.50                0.60
9      (Rash Guard)  (Dry Fit V-Nick)                0.60                0.50

   support  confidence      lift  leverage  conviction  zhangs_metric
0      0.5    0.714286  1.190476     0.080       1.400       0.533333
1      0.5    0.833333  1.190476     0.080       1.800       0.400000
2      0.5    0.833333  1.515152     0.170       2.700       0.850000
3      0.5    0.909091  1.515152     0.170       4.400       0.755556
4      0.5    0.714286  1.298701     0.115       1.575       0.766667
5      0.5    0.909091  1.298701     0.115       3.300       0.511111
6      0.5    0.909091  1.515152     0.170       4.400       0.755556
7      0.5    0.833333  1.515152     0.170       2.700       0.850000
8      0.5    1.000000  1.666667     0.200         inf       0.800000
9      0.5    0.833333  1.666667     0.200       3.000       1.000000
```

Now that the verification is done, I will now display the outputs for each dataset using different minimum support and confidence values:

Figure 1 : Amazon(Min Sup=40, Conf=30) | Figure 2 : Nike(Min Sup=50, Conf=30)
Figure 3 : Best Buy(Min Sup=60, Conf=40) | Figure 4 : General(Min Sup=40, Conf=20)
Figure 5 : K-Mart(Min Sup=40, Conf=45)

Figure 1:

Brute Force (Execution time: **3.54 seconds**)

```
Final Association Rules:

Rule 1: ['Java For Dummies'] -> ['A Beginner's Guide']
Confidence: 81.82%
Support: 45.00%

Rule 2: ['A Beginner's Guide'] -> ['Java For Dummies']
Confidence: 69.23%
Support: 45.00%

Rule 3: ['Java: The Complete Reference'] -> ['A Beginner's Guide']
Confidence: 81.82%
Support: 45.00%

Rule 4: ['A Beginner's Guide'] -> ['Java: The Complete Reference']
Confidence: 90.00%
Support: 45.00%

Rule 5: ['Java For Dummies'] -> ['Android Programming: The Big Nerd Ranch']
Confidence: 69.23%
Support: 45.00%

Rule 6: ['Android Programming: The Big Nerd Ranch'] -> ['Java For Dummies']
Confidence: 69.23%
Support: 45.00%

Rule 7: ['Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 100.00%
Support: 50.00%

Rule 8: ['Java: The Complete Reference'] -> ['Java For Dummies']
Confidence: 76.92%
Support: 50.00%

Rule 9: ['Java: The Complete Reference', 'Java For Dummies'] -> ['A Beginner's Guide']
Confidence: 81.82%
Support: 45.00%

Rule 10: ['A Beginner's Guide', 'Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 90.00%
Support: 45.00%

Rule 11: ['A Beginner's Guide', 'Java: The Complete Reference'] -> ['Java For Dummies']
Confidence: 69.23%
Support: 45.00%

Rule 12: ['Java For Dummies'] -> ['A Beginner's Guide', 'Java: The Complete Reference']
Confidence: 100.00%
Support: 45.00%

Rule 13: ['Java: The Complete Reference'] -> ['A Beginner's Guide', 'Java For Dummies']
Confidence: 100.00%
Support: 45.00%

Rule 14: ['A Beginner's Guide'] -> ['Java: The Complete Reference', 'Java For Dummies']
Confidence: 90.00%
Support: 45.00%



Program execution time: 3.5385 seconds
```

Apriori(Execution time: **0.0211 seconds**)

```
Final Association Rules (Apriori):

Rule 1: ['Java For Dummies'] -> ['A Beginner's Guide']
Confidence: 81.82%
Support: 45.00%

Rule 2: ['A Beginner's Guide'] -> ['Java For Dummies']
Confidence: 69.23%
Support: 45.00%

Rule 3: ['Java: The Complete Reference'] -> ['A Beginner's Guide']
Confidence: 81.82%
Support: 45.00%

Rule 4: ['A Beginner's Guide'] -> ['Java: The Complete Reference']
Confidence: 90.00%
Support: 45.00%

Rule 5: ['Java For Dummies'] -> ['Android Programming: The Big Nerd Ranch']
Confidence: 69.23%
Support: 45.00%

Rule 6: ['Android Programming: The Big Nerd Ranch'] -> ['Java For Dummies']
Confidence: 69.23%
Support: 45.00%

Rule 7: ['Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 100.00%
Support: 50.00%

Rule 8: ['Java: The Complete Reference'] -> ['Java For Dummies']
Confidence: 76.92%
Support: 50.00%

Rule 9: ['Java For Dummies'] -> ['A Beginner's Guide', 'Java: The Complete Reference']
Confidence: 100.00%
Support: 45.00%

Rule 10: ['Java: The Complete Reference'] -> ['A Beginner's Guide', 'Java For Dummies']
Confidence: 100.00%
Support: 45.00%

Rule 11: ['A Beginner's Guide'] -> ['Java: The Complete Reference', 'Java For Dummies']
Confidence: 90.00%
Support: 45.00%

Rule 12: ['Java: The Complete Reference', 'Java For Dummies'] -> ['A Beginner's Guide']
Confidence: 81.82%
Support: 45.00%

Rule 13: ['A Beginner's Guide', 'Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 90.00%
Support: 45.00%

Rule 14: ['A Beginner's Guide', 'Java: The Complete Reference'] -> ['Java For Dummies']
Confidence: 69.23%
Support: 45.00%



Program execution time: 0.0211 seconds
```

FP Growth

```
Frequent Itemsets using FPGrowth:
   support                                        itemsets
0     0.65                                (Java For Dummies)
1     0.65          (Android Programming: The Big Nerd Ranch)
2     0.55                               (A Beginner’s Guide)
3     0.50                     (Java: The Complete Reference)
4     0.40                      (Head First Java 2nd Edition)
5     0.45  (Android Programming: The Big Nerd Ranch, Java...
6     0.45             (A Beginner’s Guide, Java For Dummies)
7     0.50   (Java: The Complete Reference, Java For Dummies)
8     0.45  (Java: The Complete Reference, A Beginner’s Gu...
9     0.45  (Java: The Complete Reference, A Beginner’s Gu...

Association Rules using FPGrowth:
                                        antecedents  \
0           (Android Programming: The Big Nerd Ranch)
1                                 (Java For Dummies)
2                               (A Beginner’s Guide)
3                                 (Java For Dummies)
4                     (Java: The Complete Reference)
5                                 (Java For Dummies)
6                     (Java: The Complete Reference)
7                               (A Beginner’s Guide)
8   (Java: The Complete Reference, A Beginner’s Gu...
9    (Java: The Complete Reference, Java For Dummies)
10            (A Beginner’s Guide, Java For Dummies)
11                    (Java: The Complete Reference)
12                              (A Beginner’s Guide)
13                                (Java For Dummies)

                                        consequents  antecedent support  \
0                                 (Java For Dummies)                0.65
1           (Android Programming: The Big Nerd Ranch)                0.65
2                                 (Java For Dummies)                0.55
3                               (A Beginner’s Guide)                0.65
4                                 (Java For Dummies)                0.50
5                     (Java: The Complete Reference)                0.65
6                               (A Beginner’s Guide)                0.50
7                     (Java: The Complete Reference)                0.55
8                                 (Java For Dummies)                0.45
9                               (A Beginner’s Guide)                0.50
10                    (Java: The Complete Reference)                0.45
11            (A Beginner’s Guide, Java For Dummies)                0.50
12   (Java: The Complete Reference, Java For Dummies)                0.55
13  (Java: The Complete Reference, A Beginner’s Gu...                0.65

    consequent support  support  confidence      lift  leverage  conviction  \
0                 0.65     0.45    0.692308  1.065089    0.0275    1.137500
1                 0.65     0.45    0.692308  1.065089    0.0275    1.137500
2                 0.65     0.45    0.818182  1.258741    0.0925    1.925000
3                 0.55     0.45    0.692308  1.258741    0.0925    1.462500
4                 0.65     0.50    1.000000  1.538462    0.1750         inf
5                 0.50     0.50    0.769231  1.538462    0.1750    2.166667
6                 0.55     0.45    0.900000  1.636364    0.1750    4.500000
7                 0.50     0.45    0.818182  1.636364    0.1750    2.750000
8                 0.65     0.45    1.000000  1.538462    0.1575         inf
9                 0.55     0.45    0.900000  1.636364    0.1750    4.500000
10                0.50     0.45    1.000000  2.000000    0.2250         inf
11                0.45     0.45    0.900000  2.000000    0.2250    5.500000
12                0.50     0.45    0.818182  1.636364    0.1750    2.750000
13                0.45     0.45    0.692308  1.538462    0.1575    1.787500
```

```
    zhangs_metric
0        0.174603
1        0.174603
2        0.456790
3        0.587302
4        0.700000
5        1.000000
6        0.777778
7        0.864198
8        0.636364
9        0.777778
10       0.909091
11       1.000000
12       0.864198
13       1.000000

Program execution time: 0.0119 seconds
```

Figure 2:

Brute Force(Execution time: **1.609 seconds**)

```
Final Association Rules:

Rule 1: ['Rash Guard'] -> ['Dry Fit V-Nick']
Confidence: 100.00%
Support: 50.00%

Rule 2: ['Dry Fit V-Nick'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 3: ['Rash Guard'] -> ['Swimming Shirt']
Confidence: 90.91%
Support: 50.00%

Rule 4: ['Swimming Shirt'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 5: ['Socks'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 6: ['Running Shoe'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%

Rule 7: ['Running Shoe'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%

Rule 8: ['Sweatshirts'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 9: ['Socks'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%

Rule 10: ['Sweatshirts'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%



Program execution time: 1.6088 seconds
```

Apriori(Execution time: **0.029 seconds**)

```
Final Association Rules (Apriori):

Rule 1: ['Rash Guard'] -> ['Dry Fit V-Nick']
Confidence: 100.00%
Support: 50.00%

Rule 2: ['Dry Fit V-Nick'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 3: ['Rash Guard'] -> ['Swimming Shirt']
Confidence: 90.91%
Support: 50.00%

Rule 4: ['Swimming Shirt'] -> ['Rash Guard']
Confidence: 83.33%
Support: 50.00%

Rule 5: ['Socks'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 6: ['Running Shoe'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%

Rule 7: ['Running Shoe'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%

Rule 8: ['Sweatshirts'] -> ['Running Shoe']
Confidence: 71.43%
Support: 50.00%

Rule 9: ['Socks'] -> ['Sweatshirts']
Confidence: 90.91%
Support: 50.00%

Rule 10: ['Sweatshirts'] -> ['Socks']
Confidence: 83.33%
Support: 50.00%


Program execution time: 0.0293 seconds
```

FP Growth(Execution time: **0.014 seconds**)

```
Frequent Itemsets using FPGrowth:
    support                    itemsets
0      0.70                (Running Shoe)
1      0.60                       (Socks)
2      0.55                  (Sweatshirts)
3      0.60                   (Rash Guard)
4      0.55               (Swimming Shirt)
5      0.50               (Dry Fit V-Nick)
6      0.50         (Running Shoe, Socks)
7      0.50          (Sweatshirts, Socks)
8      0.50   (Sweatshirts, Running Shoe)
9      0.50  (Swimming Shirt, Rash Guard)
10     0.50  (Dry Fit V-Nick, Rash Guard)

Association Rules using FPGrowth:
        antecedents       consequents  antecedent support  consequent support  \
0    (Running Shoe)            (Socks)                0.70                0.60
1           (Socks)     (Running Shoe)                0.60                0.70
2     (Sweatshirts)            (Socks)                0.55                0.60
3           (Socks)      (Sweatshirts)                0.60                0.55
4     (Sweatshirts)     (Running Shoe)                0.55                0.70
5    (Running Shoe)      (Sweatshirts)                0.70                0.55
6  (Swimming Shirt)       (Rash Guard)                0.55                0.60
7      (Rash Guard)  (Swimming Shirt)                0.60                0.55
8  (Dry Fit V-Nick)       (Rash Guard)                0.50                0.60
9      (Rash Guard)  (Dry Fit V-Nick)                0.60                0.50

   support  confidence      lift  leverage  conviction  zhangs_metric
0      0.5    0.714286  1.190476     0.080       1.400       0.533333
1      0.5    0.833333  1.190476     0.080       1.800       0.400000
2      0.5    0.909091  1.515152     0.170       4.400       0.755556
3      0.5    0.833333  1.515152     0.170       2.700       0.850000
4      0.5    0.909091  1.298701     0.115       3.300       0.511111
5      0.5    0.714286  1.298701     0.115       1.575       0.766667
6      0.5    0.909091  1.515152     0.170       4.400       0.755556
7      0.5    0.833333  1.515152     0.170       2.700       0.850000
8      0.5    1.000000  1.666667     0.200         inf       0.800000
9      0.5    0.833333  1.666667     0.200       3.000       1.000000

Program execution time: 0.0144 seconds
```

Figure 3:

Brute Force(Execution time: **0.036 seconds**)

```
Final Association Rules:

Rule 1: ['Anti-Virus'] -> ['Lab Top Case']
Confidence: 85.71%
Support: 60.00%

Rule 2: ['Lab Top Case'] -> ['Anti-Virus']
Confidence: 85.71%
Support: 60.00%



Program execution time: 0.8361 seconds
```

Apriori(Execution time: **0.033 seconds**)

```
Final Association Rules (Apriori):

Rule 1: ['Anti-Virus'] -> ['Lab Top Case']
Confidence: 85.71%
Support: 60.00%

Rule 2: ['Lab Top Case'] -> ['Anti-Virus']
Confidence: 85.71%
Support: 60.00%



Program execution time: 0.0332 seconds
```

FP Growth(Execution time: **0.012 seconds**)

```
Frequent Itemsets using FPGrowth:
   support                  itemsets
0     0.70               (Anti-Virus)
1     0.65              (Flash Drive)
2     0.70             (Lab Top Case)
3     0.60                  (Lab Top)
4     0.60  (Lab Top Case, Anti-Virus)

Association Rules using FPGrowth:
      antecedents     consequents  antecedent support  consequent support  \
0  (Lab Top Case)    (Anti-Virus)                 0.7                 0.7
1    (Anti-Virus)  (Lab Top Case)                 0.7                 0.7

   support  confidence     lift  leverage  conviction  zhangs_metric
0      0.6    0.857143  1.22449      0.11         2.1       0.611111
1      0.6    0.857143  1.22449      0.11         2.1       0.611111

Program execution time: 0.0118 seconds
```

Figure 4:

Brute Force(Execution time: **0.854 seconds**)

```
Final Association Rules:

Rule 1: ['B'] -> ['A']
Confidence: 45.45%
Support: 45.45%

Rule 2: ['A'] -> ['B']
Confidence: 100.00%
Support: 45.45%

Rule 3: ['C'] -> ['A']
Confidence: 63.64%
Support: 63.64%

Rule 4: ['A'] -> ['C']
Confidence: 100.00%
Support: 63.64%

Rule 5: ['E'] -> ['A']
Confidence: 72.73%
Support: 72.73%

Rule 6: ['A'] -> ['E']
Confidence: 100.00%
Support: 72.73%



Program execution time: 0.8536 seconds
```

Apriori(Execution time: **0.049 seconds**)

```
Final Association Rules (Apriori):

Rule 1: ['B'] -> ['A']
Confidence: 45.45%
Support: 45.45%

Rule 2: ['A'] -> ['B']
Confidence: 100.00%
Support: 45.45%

Rule 3: ['C'] -> ['A']
Confidence: 63.64%
Support: 63.64%

Rule 4: ['A'] -> ['C']
Confidence: 100.00%
Support: 63.64%

Rule 5: ['E'] -> ['A']
Confidence: 72.73%
Support: 72.73%

Rule 6: ['A'] -> ['E']
Confidence: 100.00%
Support: 72.73%



Program execution time: 0.0490 seconds
```

FP Growth(Execution time:)

```
Frequent Itemsets using FPGrowth:
    support itemsets
0  1.000000      (A)
1  0.636364      (C)
2  0.454545      (B)
3  0.727273      (E)
4  0.636364   (A, C)
5  0.454545   (A, B)
6  0.727273   (A, E)

Association Rules using FPGrowth:
  antecedents consequents  antecedent support  consequent support   support  \
0         (A)         (C)             1.000000             0.636364  0.636364
1         (C)         (A)             0.636364             1.000000  0.636364
2         (A)         (B)             1.000000             0.454545  0.454545
3         (B)         (A)             0.454545             1.000000  0.454545
4         (A)         (E)             1.000000             0.727273  0.727273
5         (E)         (A)             0.727273             1.000000  0.727273

   confidence  lift  leverage  conviction  zhangs_metric
0    0.636364   1.0       0.0         1.0            0.0
1    1.000000   1.0       0.0         inf            0.0
2    0.454545   1.0       0.0         1.0            0.0
3    1.000000   1.0       0.0         inf            0.0
4    0.727273   1.0       0.0         1.0            0.0
5    1.000000   1.0       0.0         inf            0.0

Program execution time: 0.0123 seconds
```

Figure 5:

Brute Force(Execution time: **0.925 seconds**)

```
Final Association Rules:

Rule 1: ['Bed Skirts'] -> ['Kids Bedding']
Confidence: 83.33%
Support: 50.00%

Rule 2: ['Kids Bedding'] -> ['Bed Skirts']
Confidence: 90.91%
Support: 50.00%

Rule 3: ['Shams'] -> ['Bed Skirts']
Confidence: 72.73%
Support: 40.00%

Rule 4: ['Bed Skirts'] -> ['Shams']
Confidence: 80.00%
Support: 40.00%

Rule 5: ['Sheets'] -> ['Bed Skirts']
Confidence: 81.82%
Support: 45.00%

Rule 6: ['Bed Skirts'] -> ['Sheets']
Confidence: 90.00%
Support: 45.00%

Rule 7: ['Shams'] -> ['Kids Bedding']
Confidence: 66.67%
Support: 40.00%

Rule 8: ['Kids Bedding'] -> ['Shams']
Confidence: 80.00%
Support: 40.00%

Rule 9: ['Sheets'] -> ['Kids Bedding']
Confidence: 83.33%
Support: 50.00%

Rule 10: ['Kids Bedding'] -> ['Sheets']
Confidence: 100.00%
Support: 50.00%

Rule 11: ['Bed Skirts', 'Sheets'] -> ['Kids Bedding']
Confidence: 75.00%
Support: 45.00%

Rule 12: ['Kids Bedding', 'Sheets'] -> ['Bed Skirts']
Confidence: 81.82%
Support: 45.00%

Rule 13: ['Kids Bedding', 'Bed Skirts'] -> ['Sheets']
Confidence: 90.00%
Support: 45.00%

Rule 14: ['Sheets'] -> ['Kids Bedding', 'Bed Skirts']
Confidence: 90.00%
Support: 45.00%

Rule 15: ['Bed Skirts'] -> ['Kids Bedding', 'Sheets']
Confidence: 90.00%
Support: 45.00%

Rule 16: ['Kids Bedding'] -> ['Bed Skirts', 'Sheets']
Confidence: 100.00%
Support: 45.00%



Program execution time: 0.9253 seconds
```

Apriori(Execution time: **0.019 seconds**)

```
Final Association Rules (Apriori):

Rule 1: ['Bed Skirts'] -> ['Kids Bedding']
Confidence: 83.33%
Support: 50.00%

Rule 2: ['Kids Bedding'] -> ['Bed Skirts']
Confidence: 90.91%
Support: 50.00%

Rule 3: ['Shams'] -> ['Bed Skirts']
Confidence: 72.73%
Support: 40.00%

Rule 4: ['Bed Skirts'] -> ['Shams']
Confidence: 80.00%
Support: 40.00%

Rule 5: ['Sheets'] -> ['Bed Skirts']
Confidence: 81.82%
Support: 45.00%

Rule 6: ['Bed Skirts'] -> ['Sheets']
Confidence: 90.00%
Support: 45.00%

Rule 7: ['Shams'] -> ['Kids Bedding']
Confidence: 66.67%
Support: 40.00%

Rule 8: ['Kids Bedding'] -> ['Shams']
Confidence: 80.00%
Support: 40.00%

Rule 9: ['Sheets'] -> ['Kids Bedding']
Confidence: 83.33%
Support: 50.00%

Rule 10: ['Kids Bedding'] -> ['Sheets']
Confidence: 100.00%
Support: 50.00%

Rule 11: ['Sheets'] -> ['Kids Bedding', 'Bed Skirts']
Confidence: 90.00%
Support: 45.00%

Rule 12: ['Bed Skirts'] -> ['Kids Bedding', 'Sheets']
Confidence: 90.00%
Support: 45.00%

Rule 13: ['Kids Bedding'] -> ['Bed Skirts', 'Sheets']
Confidence: 100.00%
Support: 45.00%

Rule 14: ['Bed Skirts', 'Sheets'] -> ['Kids Bedding']
Confidence: 75.00%
Support: 45.00%

Rule 15: ['Kids Bedding', 'Sheets'] -> ['Bed Skirts']
Confidence: 81.82%
Support: 45.00%

Rule 16: ['Kids Bedding', 'Bed Skirts'] -> ['Sheets']
Confidence: 90.00%
Support: 45.00%



Program execution time: 0.0188 seconds
```

FP Growth(Execution time: **0.013 seconds**)

```
Frequent Itemsets using FPGrowth:
    support                            itemsets
0      0.50                (Decorative Pillows)
1      0.40                             (Quilts)
2      0.60                       (Kids Bedding)
3      0.55                         (Bed Skirts)
4      0.50                             (Sheets)
5      0.50                              (Shams)
6      0.50           (Kids Bedding, Bed Skirts)
7      0.50              (Kids Bedding, Sheets)
8      0.45                (Bed Skirts, Sheets)
9      0.45  (Kids Bedding, Bed Skirts, Sheets)
10     0.40                 (Bed Skirts, Shams)
11     0.40               (Kids Bedding, Shams)

Association Rules using FPGrowth:
                     antecedents                consequents  \
0                (Kids Bedding)                (Bed Skirts)
1                  (Bed Skirts)               (Kids Bedding)
2                (Kids Bedding)                     (Sheets)
3                       (Sheets)               (Kids Bedding)
4                  (Bed Skirts)                     (Sheets)
5                       (Sheets)                (Bed Skirts)
6   (Kids Bedding, Bed Skirts)                     (Sheets)
7       (Kids Bedding, Sheets)                (Bed Skirts)
8         (Bed Skirts, Sheets)               (Kids Bedding)
9                (Kids Bedding)        (Bed Skirts, Sheets)
10                 (Bed Skirts)      (Kids Bedding, Sheets)
11                      (Sheets)  (Kids Bedding, Bed Skirts)
12                 (Bed Skirts)                      (Shams)
13                      (Shams)                (Bed Skirts)
14               (Kids Bedding)                      (Shams)
15                      (Shams)               (Kids Bedding)

    antecedent support  consequent support  support  confidence      lift  \
0                 0.60                0.55     0.50    0.833333  1.515152
1                 0.55                0.60     0.50    0.909091  1.515152
2                 0.60                0.50     0.50    0.833333  1.666667
3                 0.50                0.60     0.50    1.000000  1.666667
4                 0.55                0.50     0.45    0.818182  1.636364
5                 0.50                0.55     0.45    0.900000  1.636364
6                 0.50                0.50     0.45    0.900000  1.800000
7                 0.50                0.55     0.45    0.900000  1.636364
8                 0.45                0.60     0.45    1.000000  1.666667
9                 0.60                0.45     0.45    0.750000  1.666667
10                0.55                0.50     0.45    0.818182  1.636364
11                0.50                0.50     0.45    0.900000  1.800000
12                0.55                0.50     0.40    0.727273  1.454545
13                0.50                0.55     0.40    0.800000  1.454545
14                0.60                0.50     0.40    0.666667  1.333333
15                0.50                0.60     0.40    0.800000  1.333333

    leverage  conviction  zhangs_metric
0      0.170    2.700000       0.850000
1      0.170    4.400000       0.755556
2      0.200    3.000000       1.000000
3      0.200         inf       0.800000
4      0.175    2.750000       0.864198
5      0.175    4.500000       0.777778
6      0.200    5.000000       0.888889
7      0.175    4.500000       0.777778
8      0.180         inf       0.727273
9      0.180    2.200000       1.000000
10     0.175    2.750000       0.864198
11     0.200    5.000000       0.888889
12     0.125    1.833333       0.694444
13     0.125    2.250000       0.625000
14     0.100    1.500000       0.625000
15     0.100    2.000000       0.500000

Program execution time: 0.0126 seconds
```

Comparison and Evaluation of Results:

The results of each algorithm seem to match up with each other very well even if some of the physical orders of item rules don't match up one to one exactly. As expected, the Brute Force method took the longest to execute, typically averaging from tenths of a second to over a second. The Apriori and FP Growth algorithms were consistently a good step faster than Brute Force staying in the low hundredths of a second. All times from each run are shown above with screenshots.

Other

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

Link to Git Repository:
https://github.com/sakethl20/Retail-Data-Mining