Association Rule Mining Project Report

Name: Sushil Saindane
NJIT UCID: sbs8
Email Address: sbs8@njit.edu
Date: 10/10/2024
Professor: Yasser Abduallah
Course: CS 634101 Data Mining

Abstract

- This project implements and compares three association rule mining algorithms: Brute Force, Apriori, and FP-Growth. By applying these algorithms to retail transaction data from five different stores (Amazon, Best Buy, Nike, Walmart, and Target), we explore their effectiveness in uncovering purchase patterns and generating association rules. The project demonstrates the practical application of data mining techniques in a retail context, providing insights into algorithm performance, customer behavior, and the trade-offs between different approaches to association rule mining.

Introduction

- Association rule mining is a fundamental technique in data analytics, particularly valuable in retail and e-commerce sectors. This project focuses on implementing and comparing three key algorithms:
    1. **Brute Force**: A custom implementation that exhaustively checks all possible itemsets.
    2. **Apriori**: An efficient algorithm for frequent itemset generation, implemented using the mlxtend library.
    3. **FP-Growth**: A tree-based approach for mining frequent patterns, also implemented using mlxtend.
- These algorithms are applied to transaction data from five major retailers: Amazon, Best Buy, Nike, Walmart, and Target. By comparing their performance and results, we aim to gain insights into the strengths and weaknesses of each approach in the context of retail data analysis.

Core Concepts and Principles

- **Frequent Itemset Discovery:**
    - The core of association rule mining is identifying sets of items that frequently appear together in transactions. This project implements and compares different approaches to this task, from the exhaustive brute force method to more optimized algorithms like Apriori and FP-Growth.
- **Support and Confidence Metrics:**

  Two key metrics guide our analysis:

  Support: Measures how frequently an itemset appears in the dataset.

Confidence: Assesses the likelihood of an item being purchased given the purchase of another item.

These metrics are crucial for filtering and ranking association rules.

- **Association Rules:**

The ultimate goal of the project is to generate meaningful association rules that reveal patterns in customer purchasing behavior. These rules can be used for various business applications, such as product recommendations or store layout optimization.

- **Algorithm Efficiency:**

A key aspect of this project is comparing the computational efficiency of different algorithms. We measure and compare execution times to understand the trade-offs between exhaustive searches and more optimized approaches.

- **Data Preprocessing:**

The project includes steps for loading and preprocessing transaction data, demonstrating the importance of data preparation in data mining tasks.

Project Workflow

1. **Data Preparation and Loading:**
   - CSV File Creation: Transaction data provided by the professor was expanded and organized into Excel sheets for each store (Amazon, Best Buy, Nike, Walmart, and Target).
   - CSV Loading: A custom function (load_transactions_from_csv) was implemented to read these CSV files and convert them into a format suitable for analysis.
2. **User Interaction:**
   - The program implements a user-friendly interface allowing selection of the store to analyze and input of support and confidence thresholds.
   - Input validation functions ensure that user inputs are within acceptable ranges.
3. **Algorithm Implementation:**
   a. Brute Force Algorithm:
   - Custom implementation using Python's itertools for combination generation.
   - Exhaustively checks all possible itemsets against the minimum support threshold.
   - Generates association rules based on frequent itemsets and minimum confidence.

   b. Apriori Algorithm:

   - Utilizes the mlxtend library's implementation of Apriori.
   - Efficiently generates frequent itemsets using the apriori principle.

   c. FP-Growth Algorithm:

   - Also uses mlxtend library's implementation.
   - Employs a tree-based approach for mining frequent patterns.
4. **Performance Measurement:**

    - Execution time is measured for each algorithm using Python's time module.
    - Allows for direct comparison of algorithmic efficiency.
5. **Result Comparison and Analysis:**
    - Compares the number of frequent itemsets and rules generated by each algorithm.
    - Checks for consistency in results across all three methods.
    - Identifies the fastest algorithm for each analysis.
6. **Rule Display and Interpretation:**
    - Implements functions to display association rules in a readable format.
    - Provides insights into the strength of associations through support and confidence metrics.
7. **Item Analysis:**
    - Calculates and displays item counts and support values.
    - Indicates whether individual items meet the specified support threshold.

Implementation Details

Data Handling

- Utilized pandas for data manipulation
- Implemented custom functions for CSV file operations

Brute Force Method

- Iteratively generated candidate itemsets
- Calculated support and confidence for each potential rule

Library Methods

- Used mlxtend's implementations of Apriori and FP-Growth
- Leveraged pandas DataFrames for efficient data processing

Performance Measurement

- Used Python's time module to measure execution time for each algorithm

Results and Analysis

When analyzing the Nike store transactions with a minimum support of 55% and minimum confidence of 70%, we obtained the following results:

1. **Brute Force Method:**
    - Generated 6 association rules
    - Execution time: 0.0010 seconds
2. **Apriori Algorithm:**
    - Generated 6 association rules
    - Execution time: 0.0080 seconds

3. **FP-Growth Algorithm:**
    - Generated 6 association rules
    - Execution time: 0.0020 seconds

Sample rules generated:

Rule 1: {'Socks'} -> {'Sweatshirts'} (Confidence: 0.92, Support: 0.60)

Rule 2: {'Sweatshirts'} -> {'Socks'} (Confidence: 0.92, Support: 0.60)

Rule 3: {'Sweatshirts'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)

Rule 4: {'Running Shoe'} -> {'Sweatshirts'} (Confidence: 0.79, Support: 0.55)

Comparison of Algorithms

- Our analysis showed that while all three methods produced the same number of rules for the given parameters, their execution times varied:

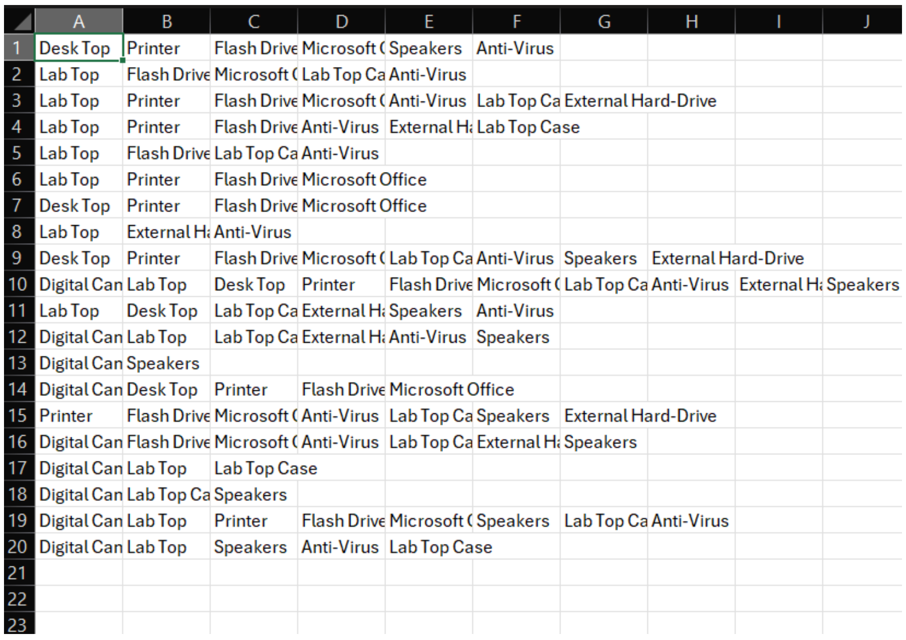
    1. Brute Force Method: Was the fastest, provides a baseline for comparison
    2. Apriori Algorithm: Was the slowest
    3. FP-Growth: Was faster than Apriori but slower than Brute Force.

Conclusion

- This project successfully implemented and compared three different approaches to association rule mining. We found that while the custom brute force method provides a good understanding of the underlying process. The association rules generated from our retail transaction data provide valuable insights into customer purchasing behavior, which can be used for product placement, marketing strategies, and inventory management.

Screenshots

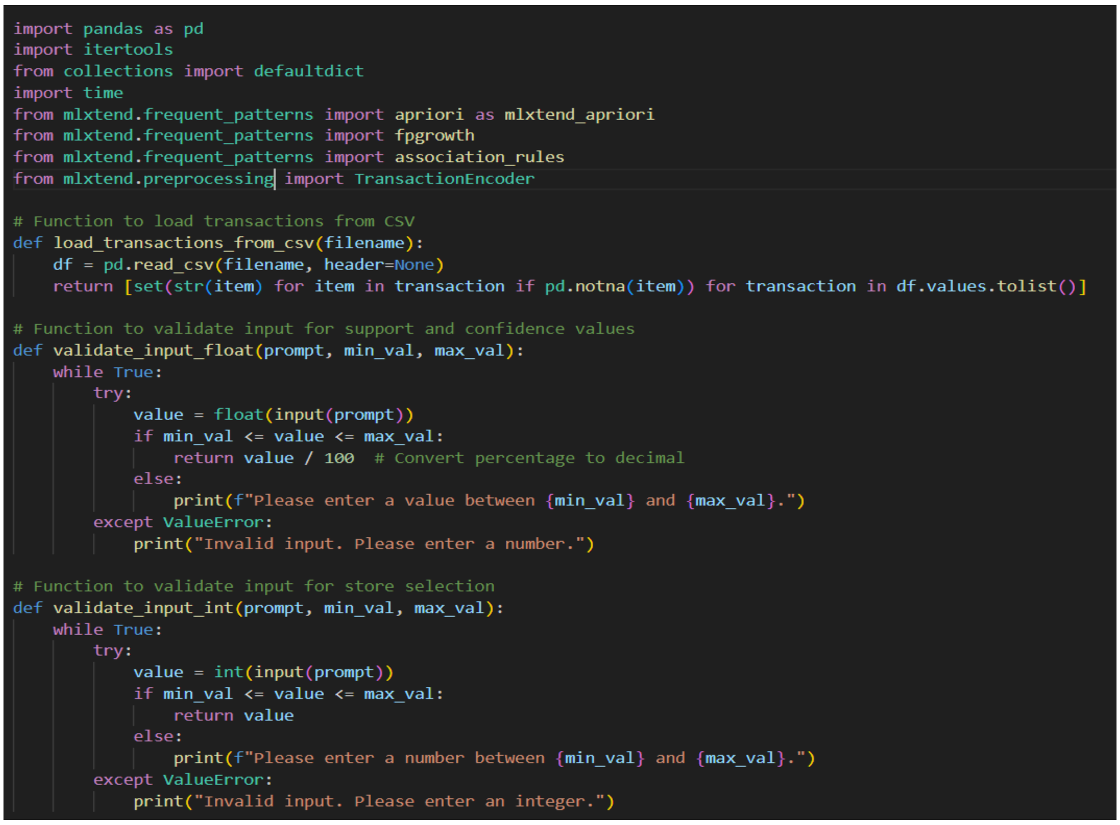
My CSV files look like this:



	A	B	C	D	E	F	G	H	I	J
1	Desk Top	Printer	Flash Drive	Microsoft (	Speakers	Anti-Virus				
2	Lab Top	Flash Drive	Microsoft (	Lab Top Ca	Anti-Virus					
3	Lab Top	Printer	Flash Drive	Microsoft (	Anti-Virus	Lab Top Ca	External Hard-Drive			
4	Lab Top	Printer	Flash Drive	Anti-Virus	External H	Lab Top Case				
5	Lab Top	Flash Drive	Lab Top Ca	Anti-Virus						
6	Lab Top	Printer	Flash Drive	Microsoft Office						
7	Desk Top	Printer	Flash Drive	Microsoft Office						
8	Lab Top	External H	Anti-Virus							
9	Desk Top	Printer	Flash Drive	Microsoft (	Lab Top Ca	Anti-Virus	Speakers	External Hard-Drive		
10	Digital Can	Lab Top	Desk Top	Printer	Flash Drive	Microsoft (	Lab Top Ca	Anti-Virus	External H	Speakers
11	Lab Top	Desk Top	Lab Top Ca	External H	Speakers	Anti-Virus				
12	Digital Can	Lab Top	Lab Top Ca	External H	Anti-Virus	Speakers				
13	Digital Can	Speakers								
14	Digital Can	Desk Top	Printer	Flash Drive	Microsoft Office					
15	Printer	Flash Drive	Microsoft (	Anti-Virus	Lab Top Ca	Speakers	External Hard-Drive			
16	Digital Can	Flash Drive	Microsoft (	Anti-Virus	Lab Top Ca	External H	Speakers			
17	Digital Can	Lab Top	Lab Top Case							
18	Digital Can	Lab Top Ca	Speakers							
19	Digital Can	Lab Top	Printer	Flash Drive	Microsoft (	Speakers	Lab Top Ca	Anti-Virus		
20	Digital Can	Lab Top	Speakers	Anti-Virus	Lab Top Case					
21										
22										
23										

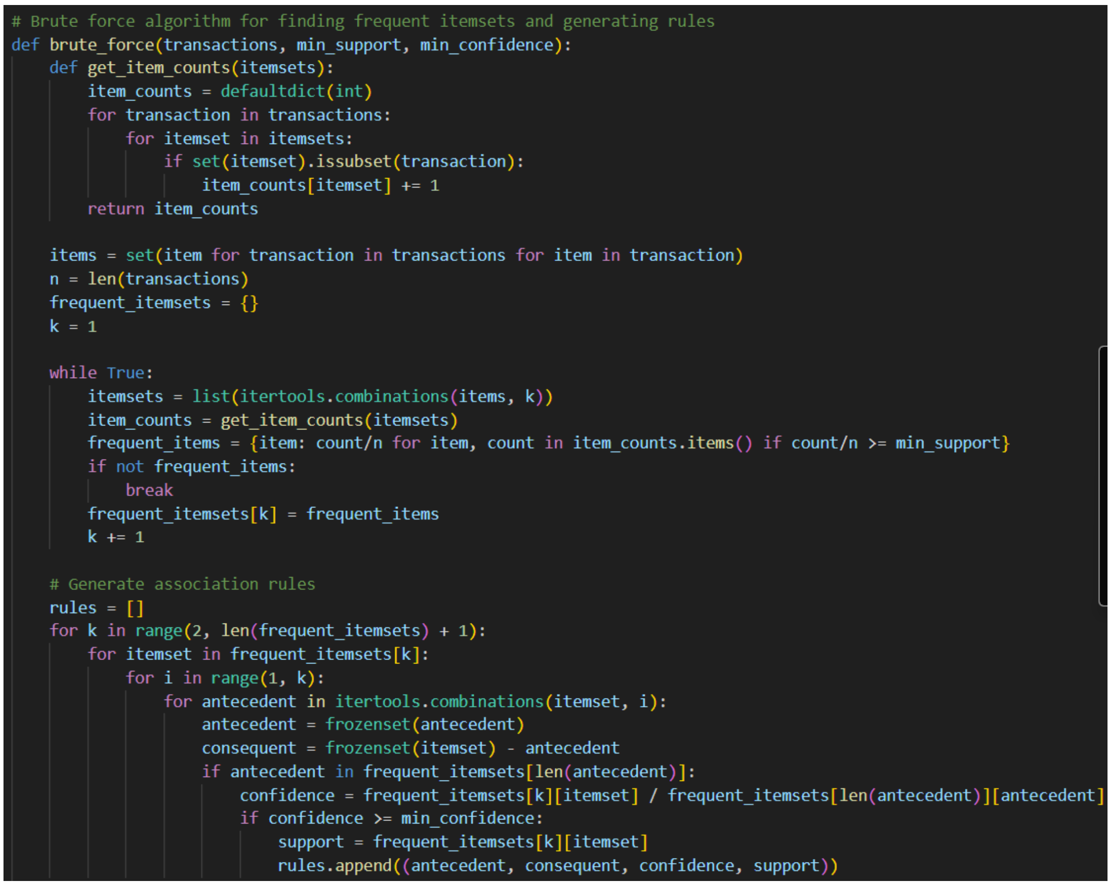
Figure 1: Best Buy Transactions

Following screenshots are snippets from my code:



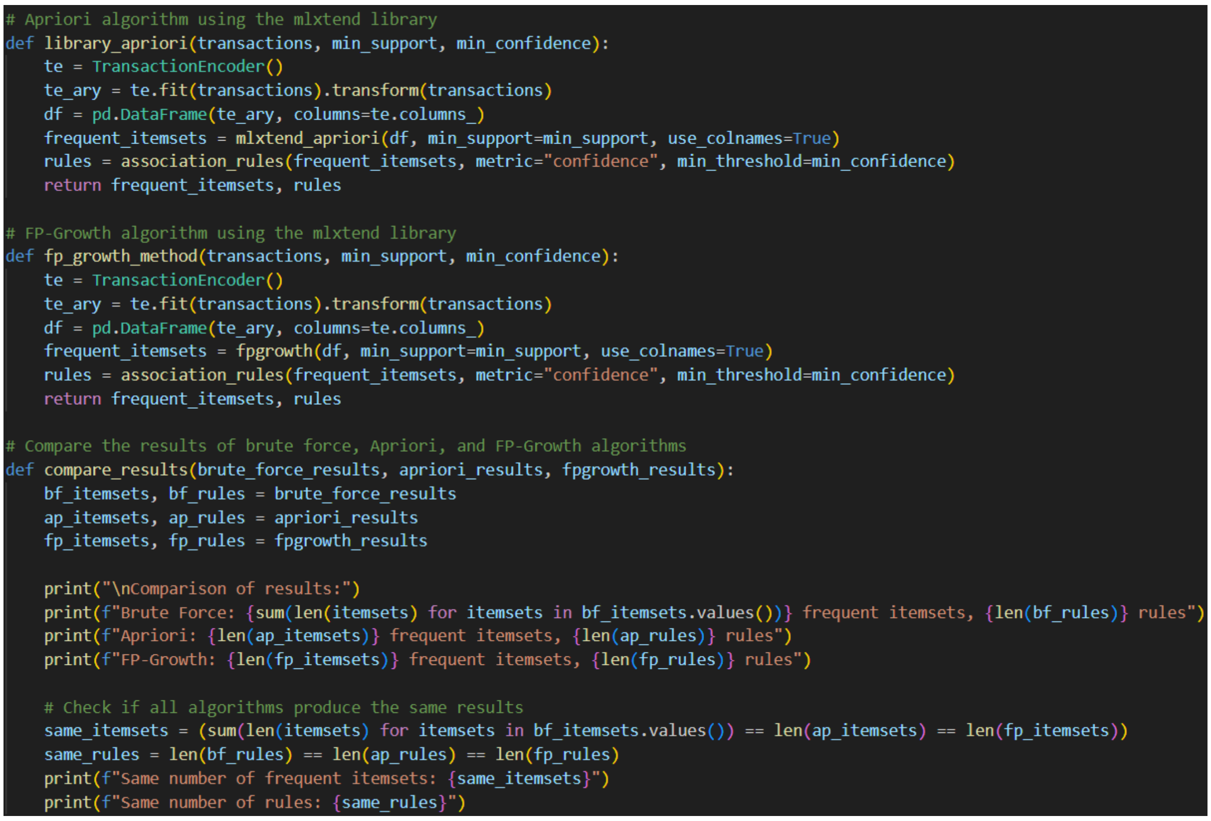
```python
import pandas as pd
import itertools
from collections import defaultdict
import time
from mlxtend.frequent_patterns import apriori as mlxtend_apriori
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder

# Function to load transactions from CSV
def load_transactions_from_csv(filename):
    df = pd.read_csv(filename, header=None)
    return [set(str(item) for item in transaction if pd.notna(item)) for transaction in df.values.tolist()]

# Function to validate input for support and confidence values
def validate_input_float(prompt, min_val, max_val):
    while True:
        try:
            value = float(input(prompt))
            if min_val <= value <= max_val:
                return value / 100  # Convert percentage to decimal
            else:
                print(f"Please enter a value between {min_val} and {max_val}.")
        except ValueError:
            print("Invalid input. Please enter a number.")

# Function to validate input for store selection
def validate_input_int(prompt, min_val, max_val):
    while True:
        try:
            value = int(input(prompt))
            if min_val <= value <= max_val:
                return value
            else:
                print(f"Please enter a number between {min_val} and {max_val}.")
        except ValueError:
            print("Invalid input. Please enter an integer.")
```

Figure 2: Loading transactions from CSV & function validation



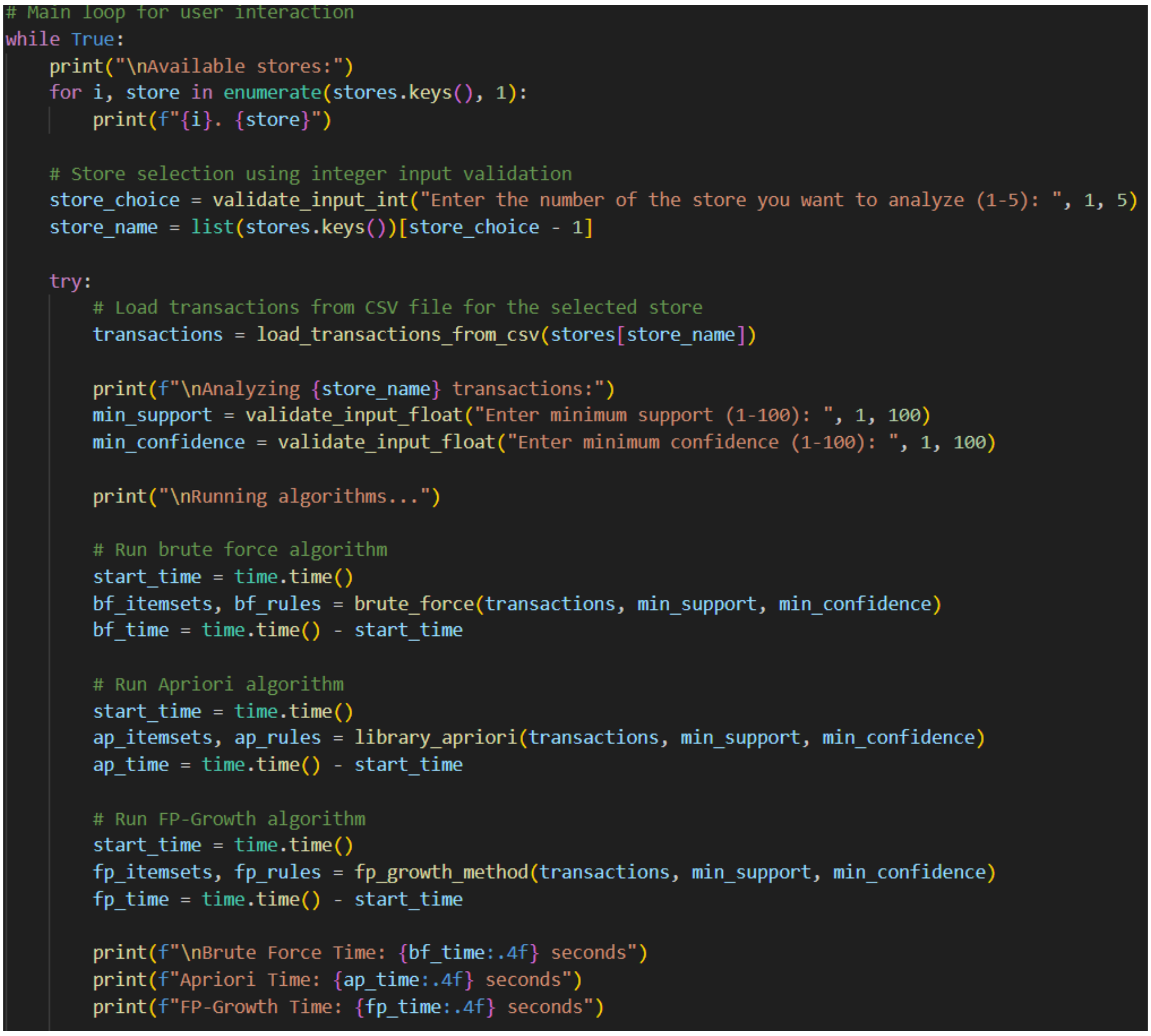
```python
# Brute force algorithm for finding frequent itemsets and generating rules
def brute_force(transactions, min_support, min_confidence):
    def get_item_counts(itemsets):
        item_counts = defaultdict(int)
        for transaction in transactions:
            for itemset in itemsets:
                if set(itemset).issubset(transaction):
                    item_counts[itemset] += 1
        return item_counts

    items = set(item for transaction in transactions for item in transaction)
    n = len(transactions)
    frequent_itemsets = {}
    k = 1

    while True:
        itemsets = list(itertools.combinations(items, k))
        item_counts = get_item_counts(itemsets)
        frequent_items = {item: count/n for item, count in item_counts.items() if count/n >= min_support}
        if not frequent_items:
            break
        frequent_itemsets[k] = frequent_items
        k += 1

    # Generate association rules
    rules = []
    for k in range(2, len(frequent_itemsets) + 1):
        for itemset in frequent_itemsets[k]:
            for i in range(1, k):
                for antecedent in itertools.combinations(itemset, i):
                    antecedent = frozenset(antecedent)
                    consequent = frozenset(itemset) - antecedent
                    if antecedent in frequent_itemsets[len(antecedent)]:
                        confidence = frequent_itemsets[k][itemset] / frequent_itemsets[len(antecedent)][antecedent]
                        if confidence >= min_confidence:
                            support = frequent_itemsets[k][itemset]
                            rules.append((antecedent, consequent, confidence, support))
```

Figure 3: Brute Force & Association Rule generation



```python
# Apriori algorithm using the mlxtend library
def library_apriori(transactions, min_support, min_confidence):
    te = TransactionEncoder()
    te_ary = te.fit(transactions).transform(transactions)
    df = pd.DataFrame(te_ary, columns=te.columns_)
    frequent_itemsets = mlxtend_apriori(df, min_support=min_support, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
    return frequent_itemsets, rules

# FP-Growth algorithm using the mlxtend library
def fp_growth_method(transactions, min_support, min_confidence):
    te = TransactionEncoder()
    te_ary = te.fit(transactions).transform(transactions)
    df = pd.DataFrame(te_ary, columns=te.columns_)
    frequent_itemsets = fpgrowth(df, min_support=min_support, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
    return frequent_itemsets, rules

# Compare the results of brute force, Apriori, and FP-Growth algorithms
def compare_results(brute_force_results, apriori_results, fpgrowth_results):
    bf_itemsets, bf_rules = brute_force_results
    ap_itemsets, ap_rules = apriori_results
    fp_itemsets, fp_rules = fpgrowth_results

    print("\nComparison of results:")
    print(f"Brute Force: {sum(len(itemsets) for itemsets in bf_itemsets.values())} frequent itemsets, {len(bf_rules)} rules")
    print(f"Apriori: {len(ap_itemsets)} frequent itemsets, {len(ap_rules)} rules")
    print(f"FP-Growth: {len(fp_itemsets)} frequent itemsets, {len(fp_rules)} rules")

    # Check if all algorithms produce the same results
    same_itemsets = (sum(len(itemsets) for itemsets in bf_itemsets.values()) == len(ap_itemsets) == len(fp_itemsets))
    same_rules = len(bf_rules) == len(ap_rules) == len(fp_rules)
    print(f"Same number of frequent itemsets: {same_itemsets}")
    print(f"Same number of rules: {same_rules}")
```

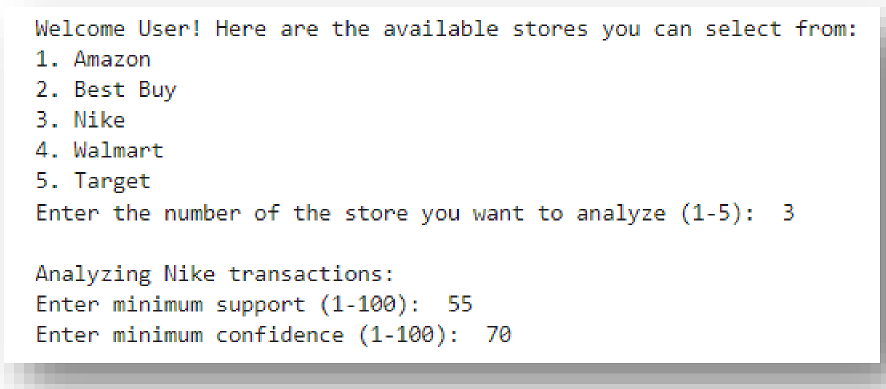
Figure 4: Apriori, FP-Growth Algorithm & Comparison of all Algorithms



```python
# Main loop for user interaction
while True:
    print("\nAvailable stores:")
    for i, store in enumerate(stores.keys(), 1):
        print(f"{i}. {store}")

    # Store selection using integer input validation
    store_choice = validate_input_int("Enter the number of the store you want to analyze (1-5): ", 1, 5)
    store_name = list(stores.keys())[store_choice - 1]

    try:
        # Load transactions from CSV file for the selected store
        transactions = load_transactions_from_csv(stores[store_name])

        print(f"\nAnalyzing {store_name} transactions:")
        min_support = validate_input_float("Enter minimum support (1-100): ", 1, 100)
        min_confidence = validate_input_float("Enter minimum confidence (1-100): ", 1, 100)

        print("\nRunning algorithms...")

        # Run brute force algorithm
        start_time = time.time()
        bf_itemsets, bf_rules = brute_force(transactions, min_support, min_confidence)
        bf_time = time.time() - start_time

        # Run Apriori algorithm
        start_time = time.time()
        ap_itemsets, ap_rules = library_apriori(transactions, min_support, min_confidence)
        ap_time = time.time() - start_time

        # Run FP-Growth algorithm
        start_time = time.time()
        fp_itemsets, fp_rules = fp_growth_method(transactions, min_support, min_confidence)
        fp_time = time.time() - start_time

        print(f"\nBrute Force Time: {bf_time:.4f} seconds")
        print(f"Apriori Time: {ap_time:.4f} seconds")
        print(f"FP-Growth Time: {fp_time:.4f} seconds")
```

Figure 5: Main function for User Interaction

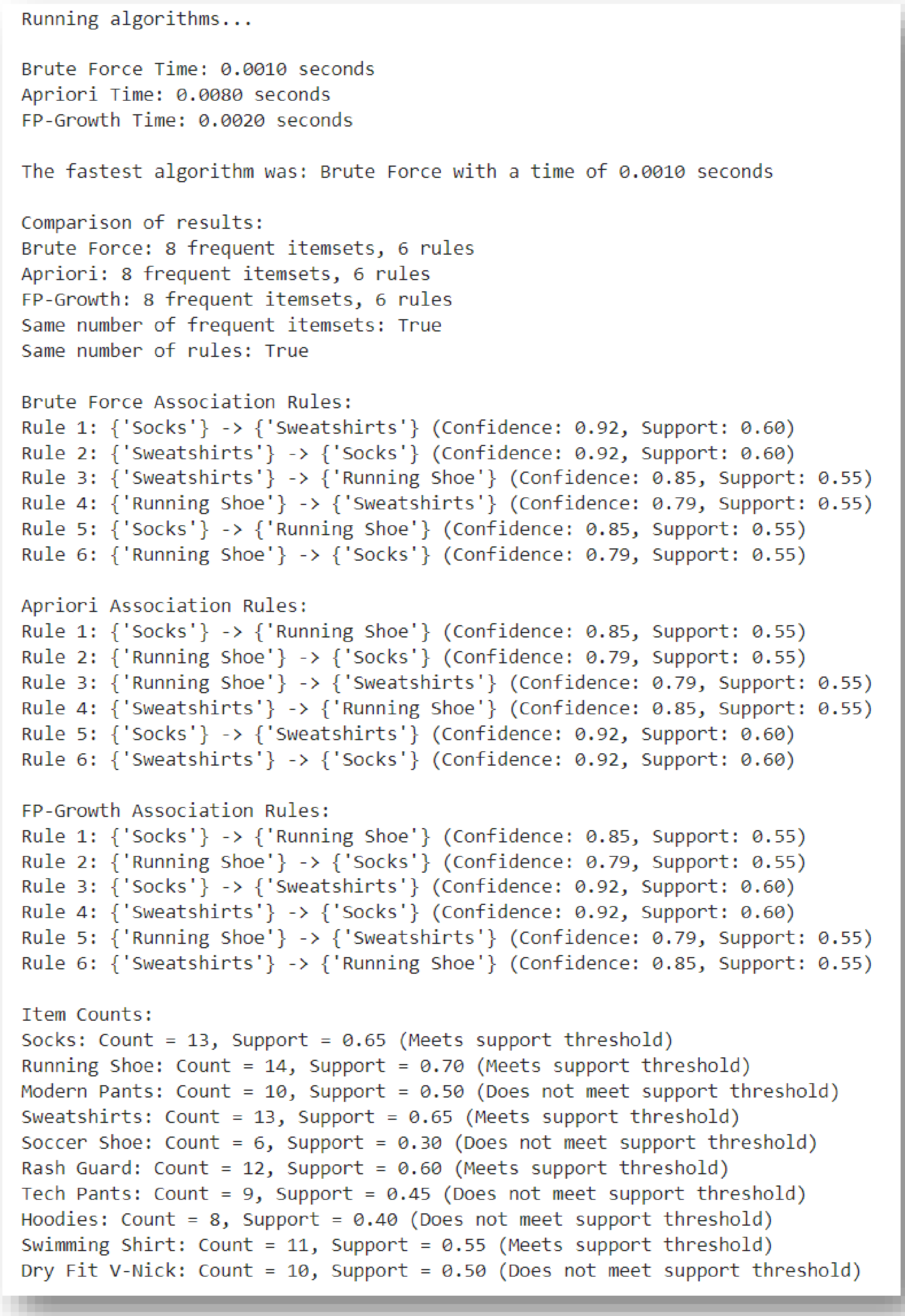
Below screenshots show the output and user interaction:



```
Welcome User! Here are the available stores you can select from:
1. Amazon
2. Best Buy
3. Nike
4. Walmart
5. Target
Enter the number of the store you want to analyze (1-5):  3

Analyzing Nike transactions:
Enter minimum support (1-100):  55
Enter minimum confidence (1-100):  70
```

Figure 6: User selects a store and enters minimum support and confidence for that store.

After entering minimum support and confidence values, we get the following:



```
Running algorithms...

Brute Force Time: 0.0010 seconds
Apriori Time: 0.0080 seconds
FP-Growth Time: 0.0020 seconds

The fastest algorithm was: Brute Force with a time of 0.0010 seconds

Comparison of results:
Brute Force: 8 frequent itemsets, 6 rules
Apriori: 8 frequent itemsets, 6 rules
FP-Growth: 8 frequent itemsets, 6 rules
Same number of frequent itemsets: True
Same number of rules: True

Brute Force Association Rules:
Rule 1: {'Socks'} -> {'Sweatshirts'} (Confidence: 0.92, Support: 0.60)
Rule 2: {'Sweatshirts'} -> {'Socks'} (Confidence: 0.92, Support: 0.60)
Rule 3: {'Sweatshirts'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 4: {'Running Shoe'} -> {'Sweatshirts'} (Confidence: 0.79, Support: 0.55)
Rule 5: {'Socks'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 6: {'Running Shoe'} -> {'Socks'} (Confidence: 0.79, Support: 0.55)

Apriori Association Rules:
Rule 1: {'Socks'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 2: {'Running Shoe'} -> {'Socks'} (Confidence: 0.79, Support: 0.55)
Rule 3: {'Running Shoe'} -> {'Sweatshirts'} (Confidence: 0.79, Support: 0.55)
Rule 4: {'Sweatshirts'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 5: {'Socks'} -> {'Sweatshirts'} (Confidence: 0.92, Support: 0.60)
Rule 6: {'Sweatshirts'} -> {'Socks'} (Confidence: 0.92, Support: 0.60)

FP-Growth Association Rules:
Rule 1: {'Socks'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 2: {'Running Shoe'} -> {'Socks'} (Confidence: 0.79, Support: 0.55)
Rule 3: {'Socks'} -> {'Sweatshirts'} (Confidence: 0.92, Support: 0.60)
Rule 4: {'Sweatshirts'} -> {'Socks'} (Confidence: 0.92, Support: 0.60)
Rule 5: {'Running Shoe'} -> {'Sweatshirts'} (Confidence: 0.79, Support: 0.55)
Rule 6: {'Sweatshirts'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)

Item Counts:
Socks: Count = 13, Support = 0.65 (Meets support threshold)
Running Shoe: Count = 14, Support = 0.70 (Meets support threshold)
Modern Pants: Count = 10, Support = 0.50 (Does not meet support threshold)
Sweatshirts: Count = 13, Support = 0.65 (Meets support threshold)
Soccer Shoe: Count = 6, Support = 0.30 (Does not meet support threshold)
Rash Guard: Count = 12, Support = 0.60 (Meets support threshold)
Tech Pants: Count = 9, Support = 0.45 (Does not meet support threshold)
Hoodies: Count = 8, Support = 0.40 (Does not meet support threshold)
Swimming Shirt: Count = 11, Support = 0.55 (Meets support threshold)
Dry Fit V-Nick: Count = 10, Support = 0.50 (Does not meet support threshold)
```

Figure 7: We get each algorithm time, fastest algorithm, association rules & item count

GitHub Repository

https://github.com/sushilsaindane/saindane_sushil_midtermproj

References

1. Mlxtend library documentation
2. Brute Force
3. Association Rule Mining
4. Github
5. Perplexity