**CS634 DATA MINING - MIDTERM PROJECT REPORT**

**Student Name:** Yesha Dobariya  
**Email:** yd326@njit.edu  
**Course:** CS634 - Data Mining  
**Instructor:** Dr. Yasser Abduallah

1. **INTRODUCTION:**

The objective of this project is to apply frequent itemset mining and association rule learning to multiple manually created transactional databases. Using these datasets, I explore patterns and relationships among items frequently purchased together across different types of stores such as Amazon, BestBuy, K-Mart, Nike, and a Generic example.

This project demonstrates how real-world market basket data can be modeled and analyzed using data mining techniques. The analysis will employ the Apriori and FP-Growth algorithms implemented in Python to extract meaningful association rules.

1. **DATASET CREATION:**

* **Overview**

For this project, I created five transactional databases, each representing a different store. The stores and corresponding CSV files are:

* amazon.csv
* bestbuy.csv
* kmart.csv
* nike.csv
* generic.csv

Each dataset contains at least five unique items and 20 deterministic transactions.

* + **Item Selection and Transaction Design**

All transactions were designed deterministically to simulate realistic purchasing patterns. Each CSV file contains two columns:

* **TransactionID:** A unique ID for each transaction (e.g., Trans1, Trans2, …)
* **Items:** A comma-separated list of items purchased together

The datasets were created manually in Excel and exported as .csv files. Each dataset represents a different type of store with logically grouped products.

* **Dataset Summaries**

1. **Amazon Dataset**

* Focus: Books and learning materials related to programming
* Example Items: *A Beginner’s Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition*
* Number of Transactions: 20

1. **BestBuy Dataset**

* Focus: Electronics and computer accessories
* Example Items: *Lab Top, Printer, Flash Drive, Microsoft Office, Anti-Virus, External Hard Drive*
* Number of Transactions: 20

1. **K-Mart Dataset**

* Focus: Home and bedding products
* Example Items: *Quilts, Bedspreads, Decorative Pillows, Sheets, Shams, Bed Skirts*
* Number of Transactions: 20

1. **Nike Dataset**

* Focus: Sportswear and athletic apparel
* Example Items: *Running Shoe, Socks, Sweatshirts, Tech Pants, Rash Guard, Hoodies*
* Number of Transactions: 20

1. **Generic Dataset**

* Focus: Abstract item labels used to test algorithm scalability
* Example Items: *A, B, C, D, E, F*
* Number of Transactions: 20
* **Dataset Notes**
* All datasets are small (few KB) to allow fast algorithm execution.
* Each dataset was saved in CSV format with deterministic transactions to ensure reproducibility.
* These datasets form the foundation for implementing Apriori and FP-Growth algorithms in the next phase.

1. **BRUTE FORCE ALGORITHM**

* **Method**

The Brute Force algorithm checks every possible combination of items to find which ones appear together often enough to be considered frequent.  
Once the frequent groups are found, it creates “if–then” rules that describe how the presence of some items predicts others.

The steps are:

**Step-1: Generate all item combinations**

The algorithm starts with single items (1-itemsets) and then creates all possible pairs, triples, and so on, using combinations.

**Step-2: Count how often each combination appears**

For every group of items, it scans all transactions and counts how many contain that group. This number is called the support count.

**Step-3: Filter by minimum support**

Only the itemsets whose support (frequency) is greater than or equal to the chosen minimum support threshold are kept. All others are discarded.

**Step-4: Repeat for larger itemsets**

The process continues (k = 1, 2, 3, …) until no new frequent itemsets are found.

**Step-5: Generate association rules**

For each frequent itemset, the algorithm forms rules such as X → Y. It calculates confidence as: Confidence(X→Y) = Support(X) / Support(X∪Y)

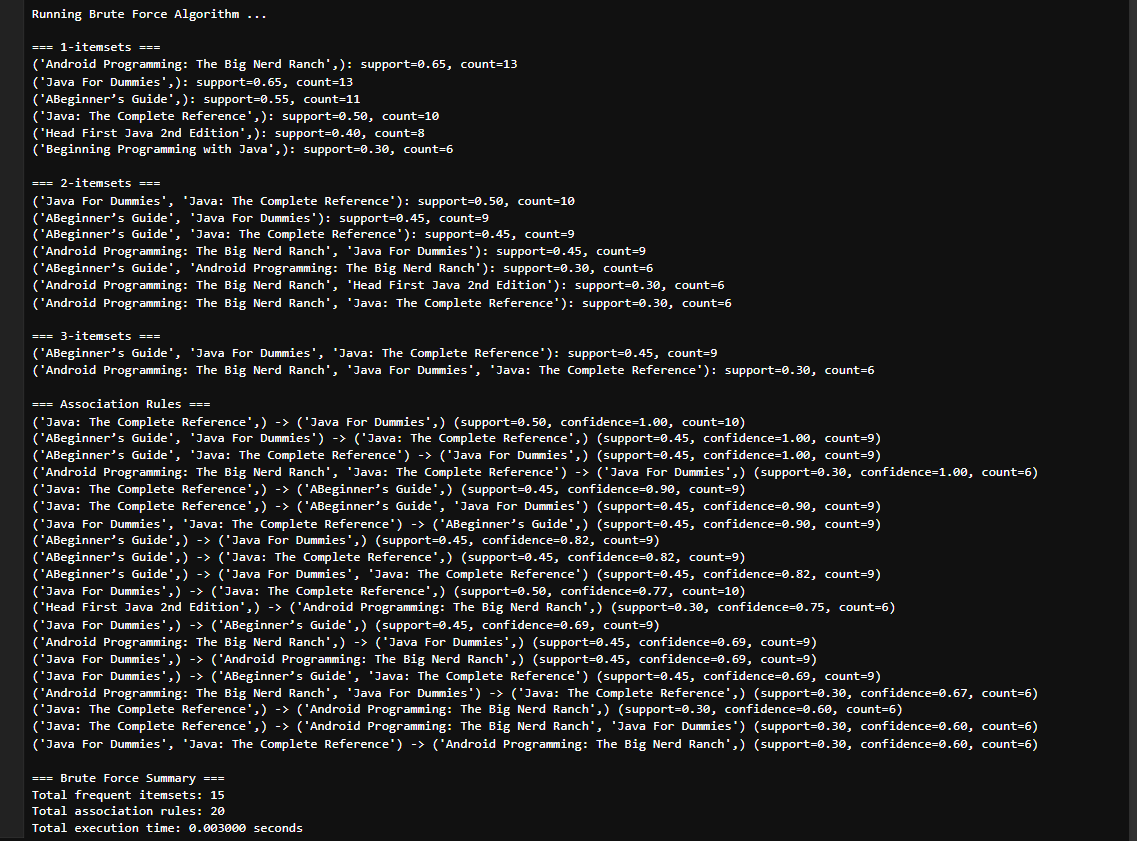
**Step-6: Select final rules**

Only the rules that meet the minimum confidence threshold are included in the final output.

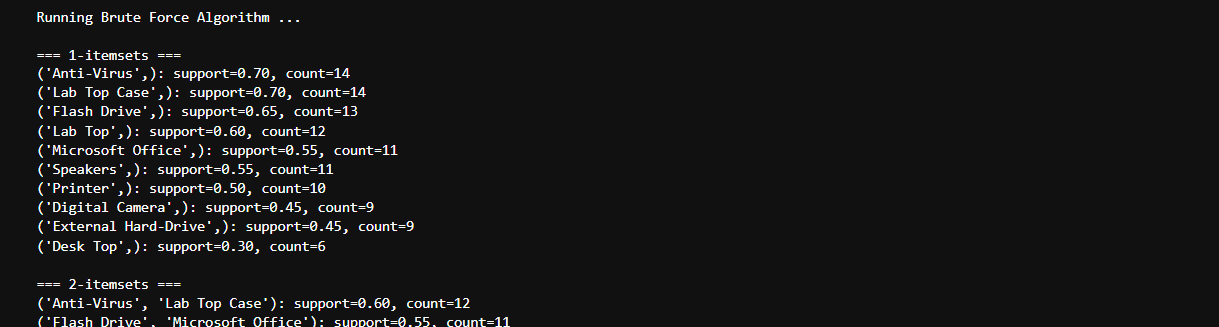
* **Example Run**

Parameters: Minimum Support = 0.3, Minimum Confidence = 0.6

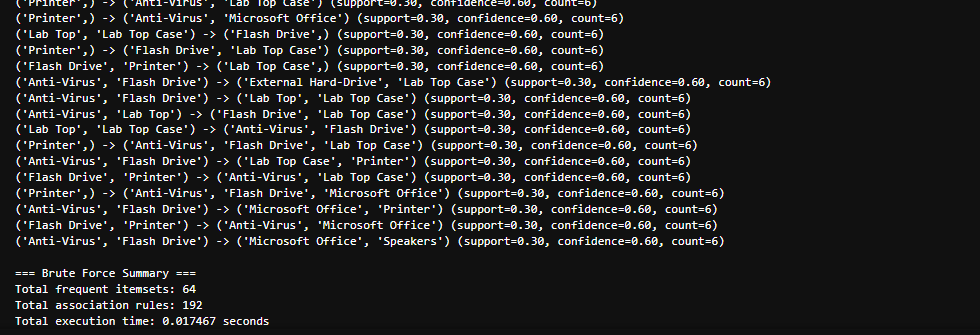
**Dataset:** amazon.csv

****

**Dataset:** bestbuy.csv

****

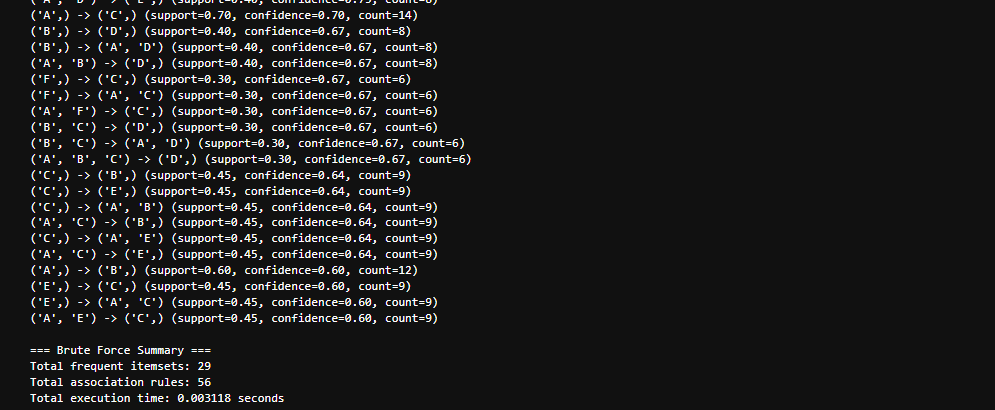
**…..**

****

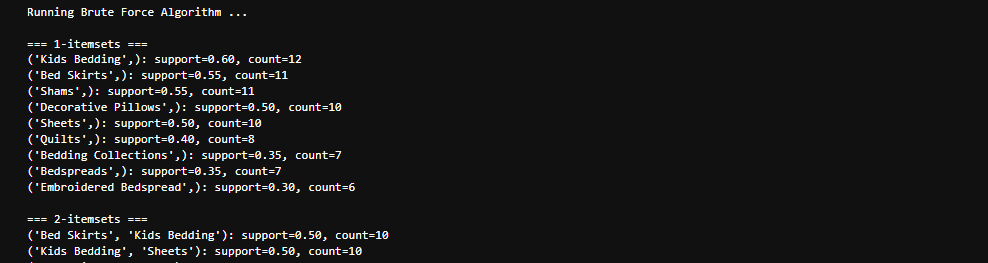
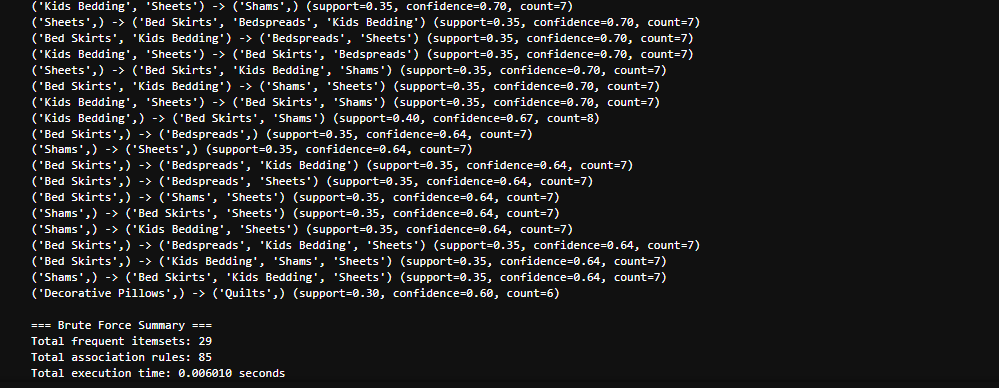
**Dataset:** generic.csv

****

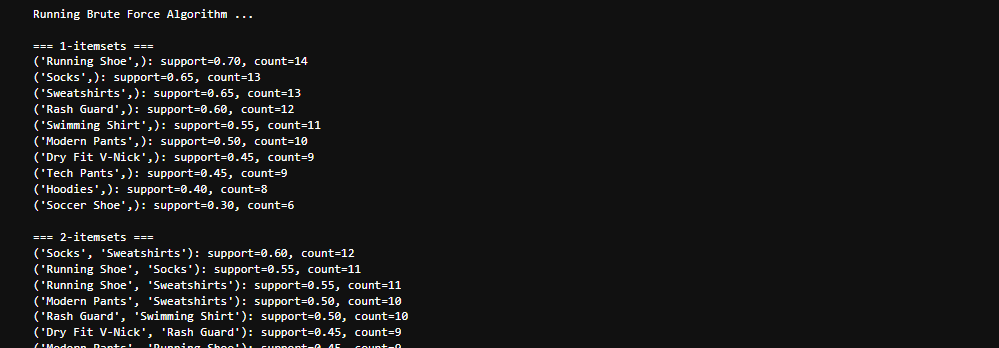
**…..**

****

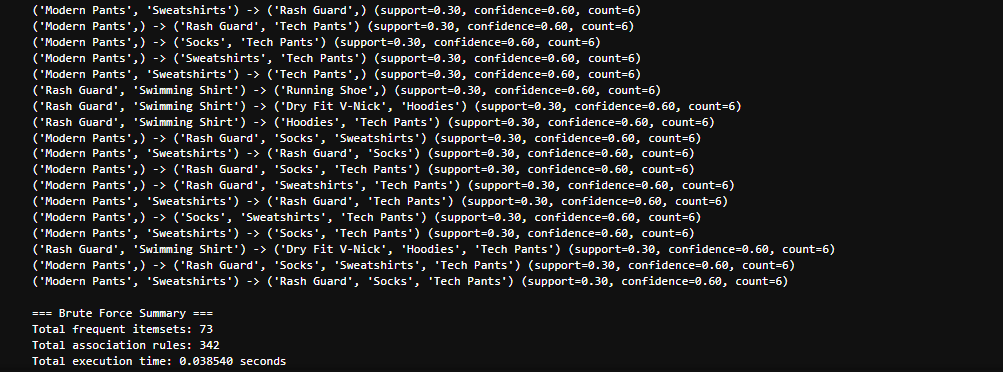
**Dataset:** kmart.csv

   
**…..**  


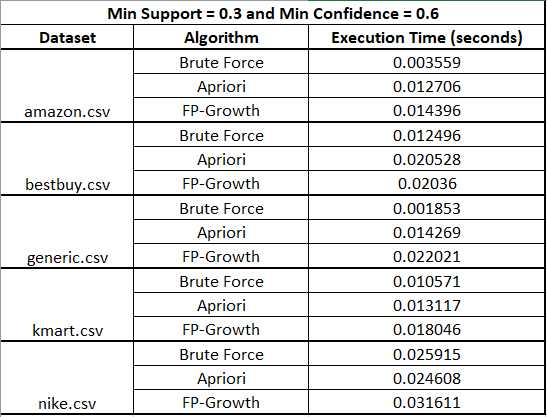
**Dataset:** nike.csv



**…..**



* **Timing Comparison**

****

1. **APRORI ALGORITHM**

* **Method**

The Apriori algorithm improves efficiency over the Brute Force approach by reducing the number of combinations that need to be checked. It uses the Apriori property, which states: “**If an itemset is frequent, then all of its subsets must also be frequent.**” This allows the algorithm to skip checking combinations that include infrequent subsets, saving a lot of computation time.

**The steps are:**

**Step-1: Generate candidate 1-itemsets**

Start by scanning all transactions to find the frequency of individual items. Keep only those that meet the minimum support threshold.

**Step-2: Generate candidate k-itemsets**

Use the frequent (k−1)-itemsets to generate new candidate k-itemsets by joining them with each other.

**Step-3: Prune infrequent subsets**

Before counting supports, remove any candidate itemset that contains an infrequent subset (as it cannot be frequent according to the Apriori property).

**Step-4: Count support for remaining candidates**

Scan the dataset again to count how many transactions contain each candidate itemset.

**Step-5: Select frequent itemsets**

Keep only those itemsets whose support is above or equal to the minimum support threshold.

**Step-6: Generate association rules**

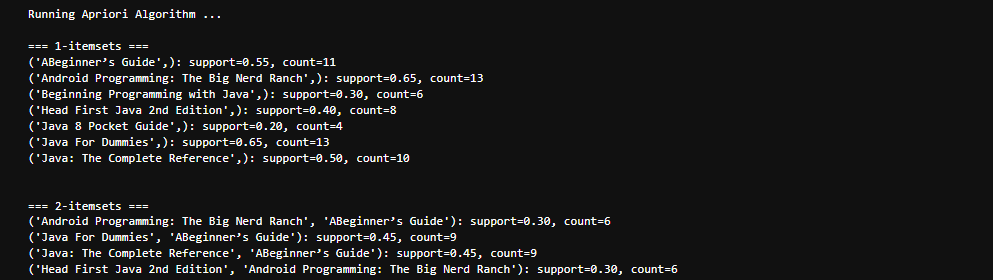
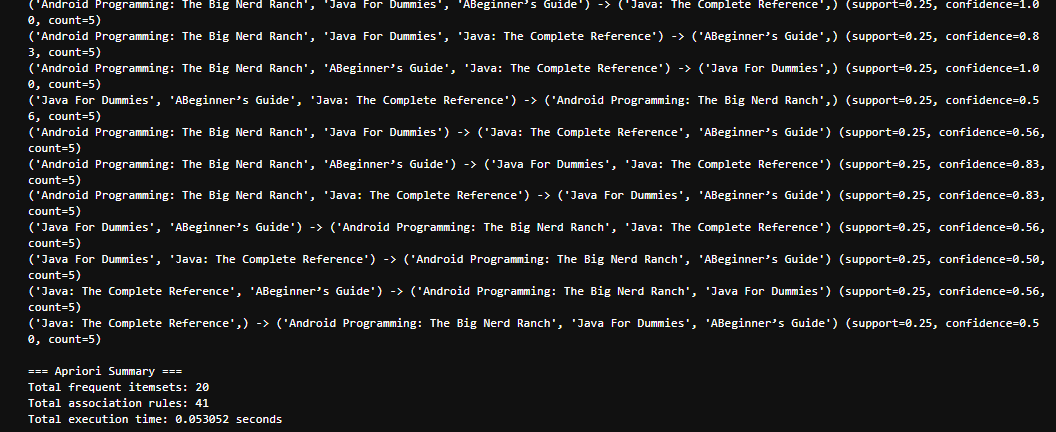
From each frequent itemset, generate rules of the form X → Y and compute the confidence value: Confidence(X→Y) = Support(X∪Y) / Support(X)

**Step-7: Keep the strong rules**  
Only the rules that meet or exceed the minimum confidence are included in the final output.

* **Example Run**

Parameters: Minimum Support = 0.2, Minimum Confidence = 0.5

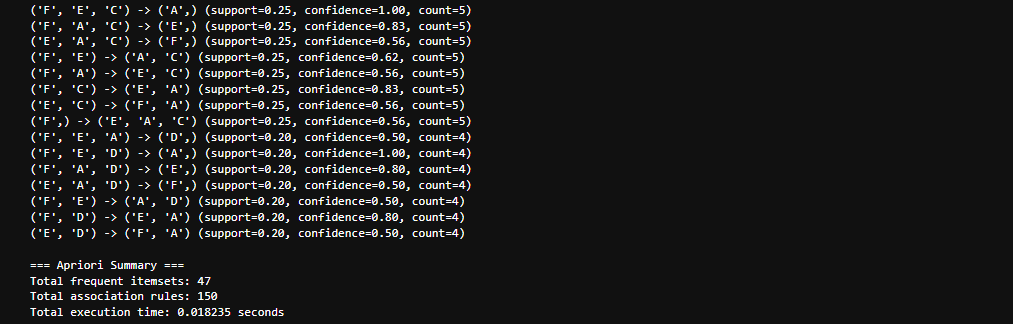
**Dataset:** amazon.csv

 **…..**

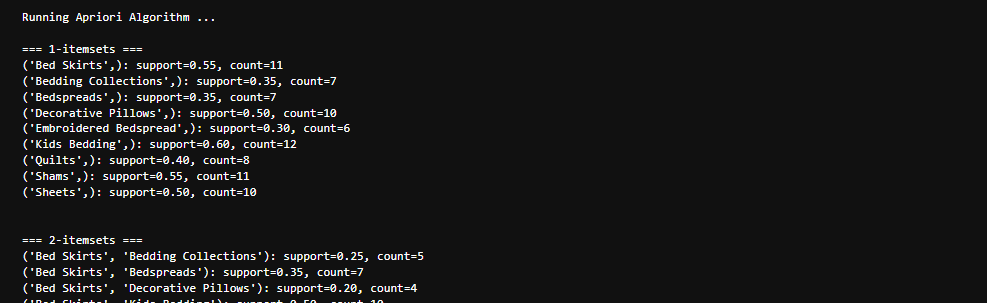
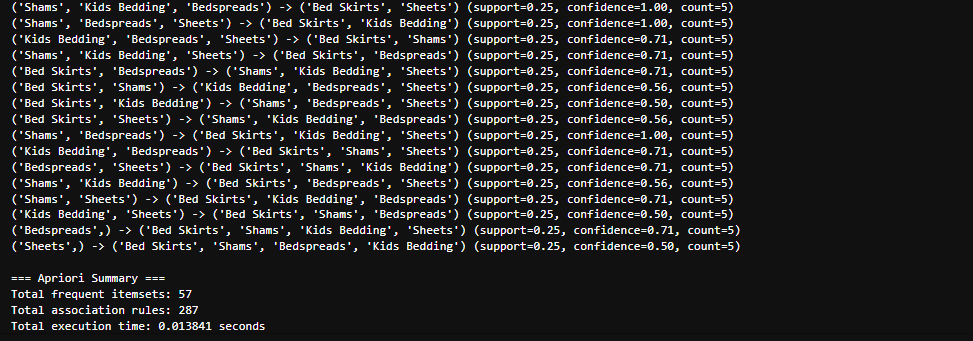
**Dataset:** bestbuy.csv

**……**

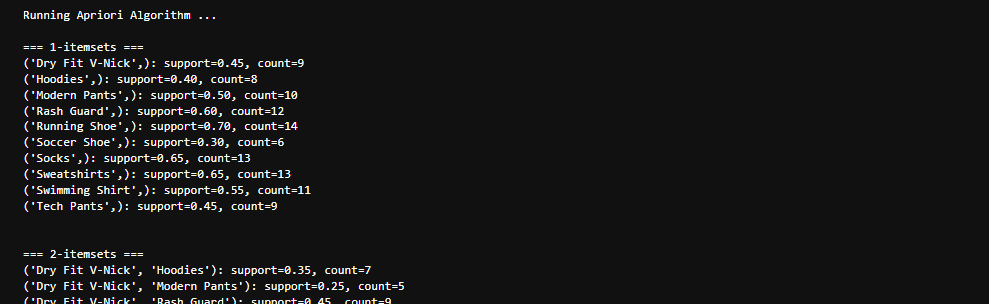
**Dataset:** generic.csv

 **…..  
**

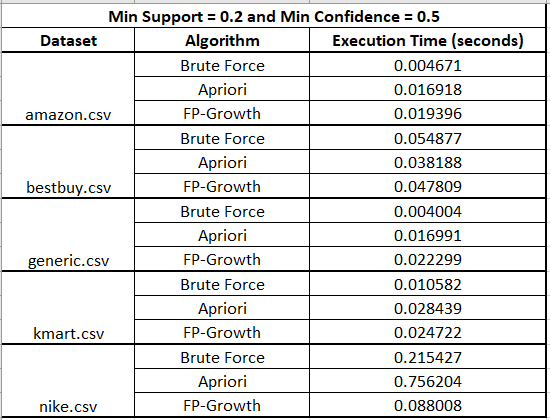
**Dataset:** kmart.csv

 **…..**

**Dataset:** nike.csv

 **…..**

* **Timing Comparison:**

****

1. **FP-GROWTH ALGORITHM**

* **Method**

The FP-Growth (Frequent Pattern Growth) algorithm is an advanced method for frequent itemset mining that eliminates the need to generate and test candidate itemsets (unlike Apriori). It uses a compact data structure called an FP-Tree (Frequent Pattern Tree) to store transactions efficiently and directly extract frequent patterns. This makes FP-Growth significantly faster, especially for large datasets.

The steps are:

**Step-1: Construct the FP-Tree**

* Scan the dataset once to count the frequency of each item.
* Remove infrequent items (those below minimum support).
* Sort remaining items in descending order of frequency.
* Build the FP-Tree by inserting transactions one by one following the sorted order.

**Step-2: Mine frequent patterns from the FP-Tree**

* Start from the bottom of the tree and extract conditional pattern bases (subsets of transactions related to an item).
* Construct conditional FP-Trees for each item recursively.
* Combine results to find all frequent itemsets.

**Step-3: Generate association rules**

From the discovered frequent itemsets, generate rules of the form X → Y.

Calculate confidence for each rule using: Confidence(X→Y) = Support(X∪Y) / Support(X).

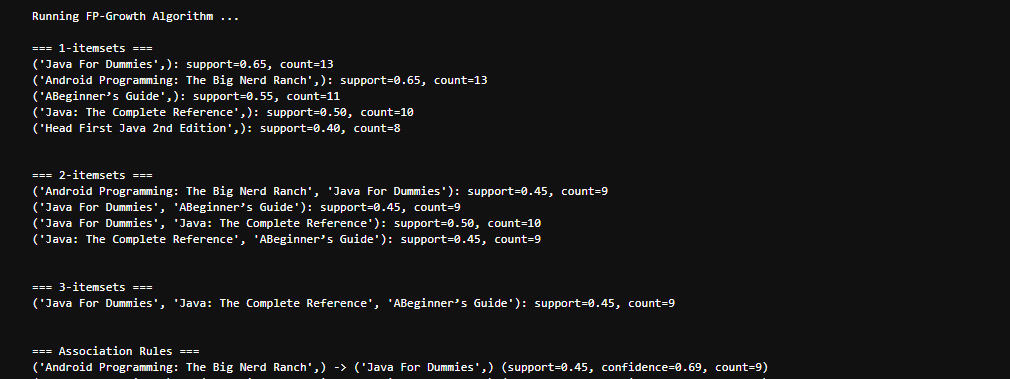
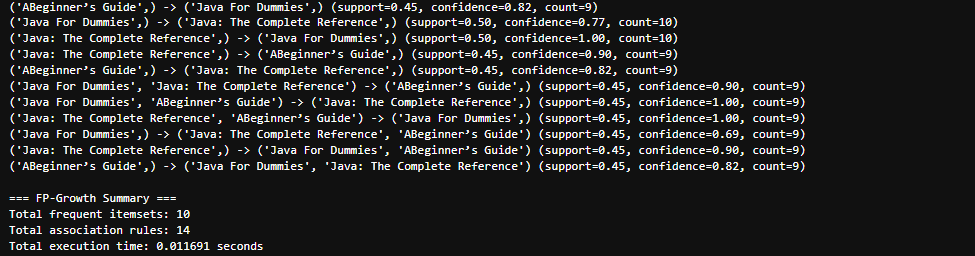
**Step-4: Select strong rules**

Only rules that satisfy both minimum support and minimum confidence thresholds are included in the final result.

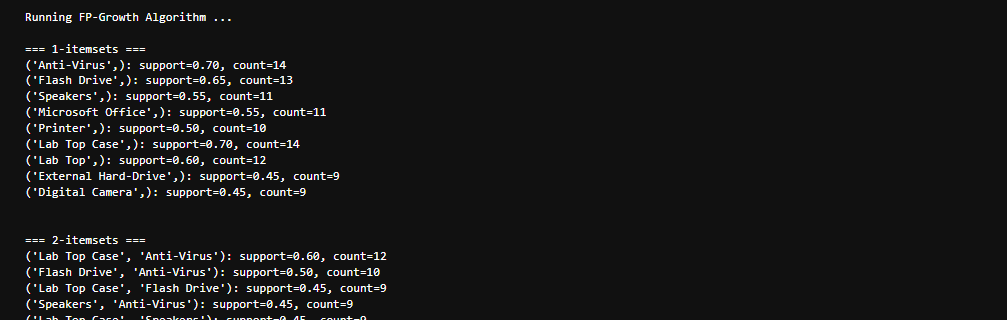
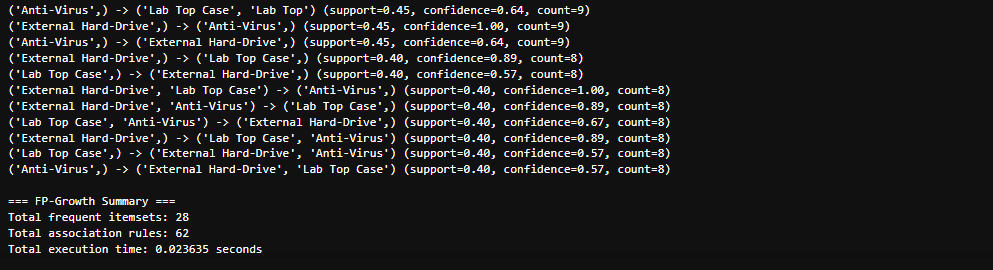
* **Example Run**

Parameters: Minimum Support = 0.4, Minimum Confidence = 0.5

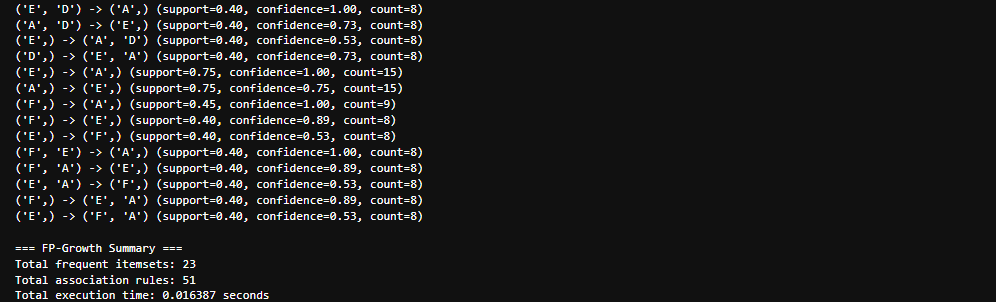
**Dataset:** amazon.csv

**  
…..**   


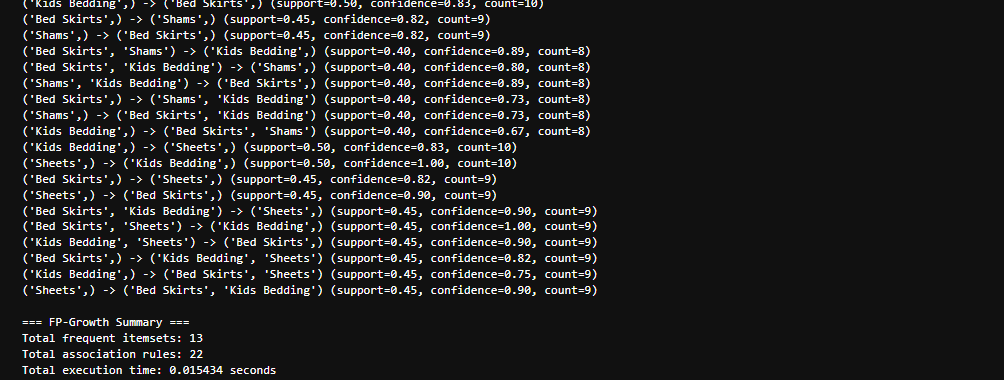
**Dataset:** bestbuy.csv

 **…..**  


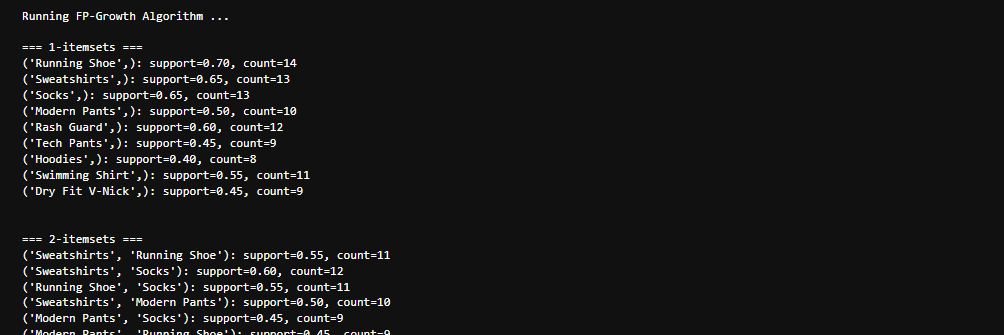
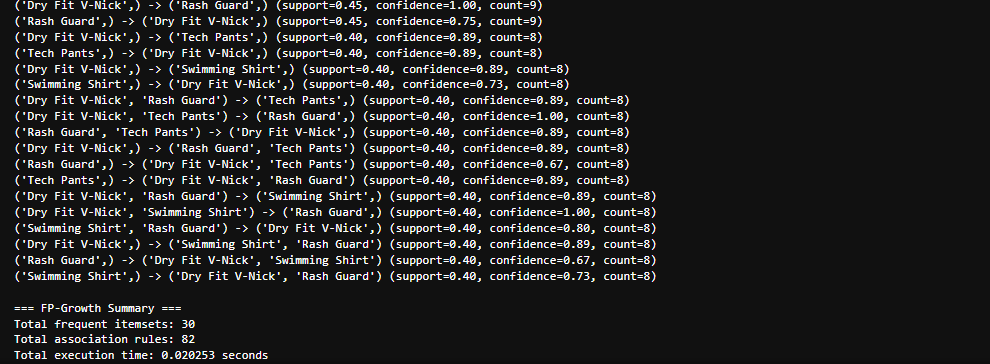
**Dataset:** generic.csv

 **…..**  


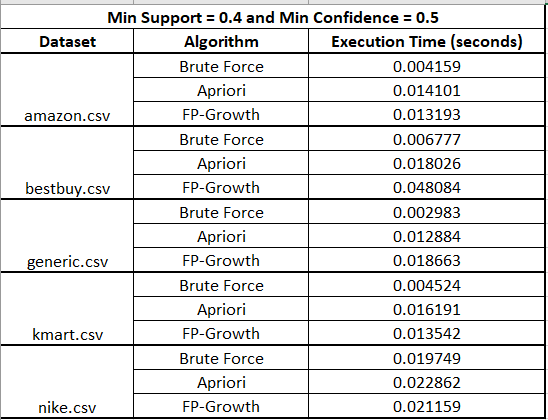
**Dataset:** kmart.csv

 **…..**  ****

**Dataset:** nike.csv

****   
**…..  
**

* **Timing Comparison**

****

1. **HOW TO RUN THE CODE**

* **Install Requirements**
* Before running the project, make sure Python 3.8 or higher is installed.
* Then, open a terminal or command prompt and install the required libraries by running: **pip install pandas mlxtend**
* pandas - used to read and handle the transactional CSV datasets.
* mlxtend - provides built-in functions for Apriori and FP-Growth algorithms.
* **Run Options**

1. **Run through Python Script**

* Run Brute Force algorithm: python src/brute\_force.py
* Run Apriori algorithm**:** python src/apriori\_runner.py
* Run FP-Growth algorithm: python src/fpgrowth\_runner.py

1. **Run through Jupyter Notebook**

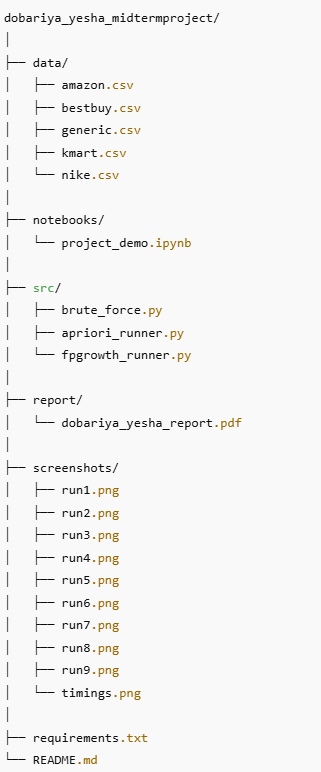
* jupyter notebook: notebooks/project\_demo.ipynb

1. **CONCLUSION**

* When **support** and **confidence thresholds** increase, the number of frequent itemsets and rules decreases across all algorithms, confirming correct behaviour.
* **Brute Force** remains the slowest approach but consistent for smaller datasets.
* **Apriori** scales better, but due to repeated database scans, its execution time grows moderately
* **FP-Growth** maintains efficiency and performs best for larger datasets or lower support thresholds where candidate sets explode.
* Overall, **FP-Growth shows superior scalability**, followed by **Apriori**, while **Brute Force** serves as a useful baseline reference.

1. **GITHUB REPOSITORY STRUCTURE AND LINK:**

* **LINK:** [**GitHub Link**](https://github.com/yesha46/cs634_yeshadobariya_midtermproject) (yesha46)
* **Note on Email Account:** Used personal email **dobariyayeshaa19@gmail.com** for this project.

****