CS634 – Data Mining Midterm Project Report

Student Name: Shivaji Burle

Email: sb2857@njit.edu

Course: CS634 – Data Mining

Instrucator: Dr. Yasser Abduallah

1.Introduction

This report documents the process of implementing frequent itemset mining and association rule learning using three methods:

Brute Force Method: This method checks all possible item combinations one by one. It is simple but takes a lot of time when there are many items.

Apriori Algorithm: Apriori finds patterns by removing item combinations that do not appear often. It is faster than brute force.

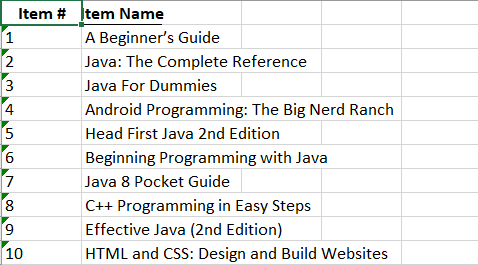
FP-Growth Algorithm: FP-Growth builds a special tree to find frequent patterns quickly. It works faster on large datasets.

The report explains how to prepare data, find itemsets, and create rules clearly.

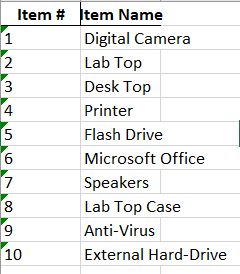
2.Dataset Creation

2.1. Data Items:

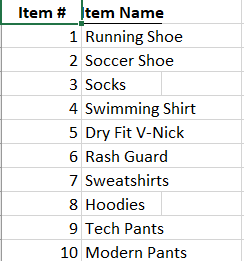
Amazon Data:



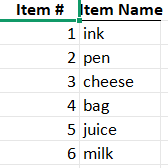
Best Buy Data:



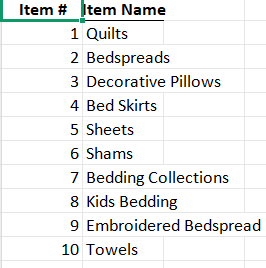
Nike Data:



Custom Data:

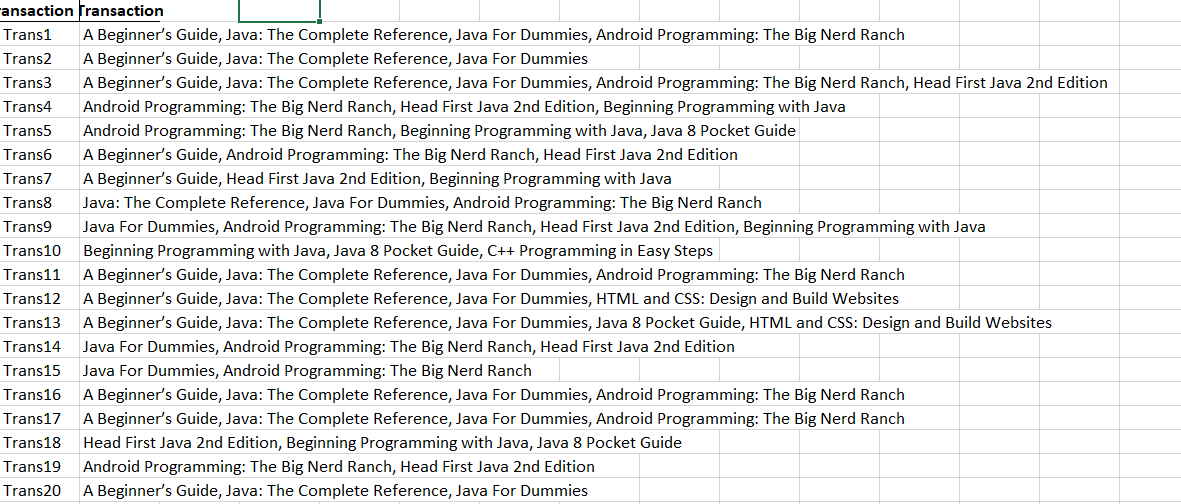


K-mart Data:



2.2 Transactions

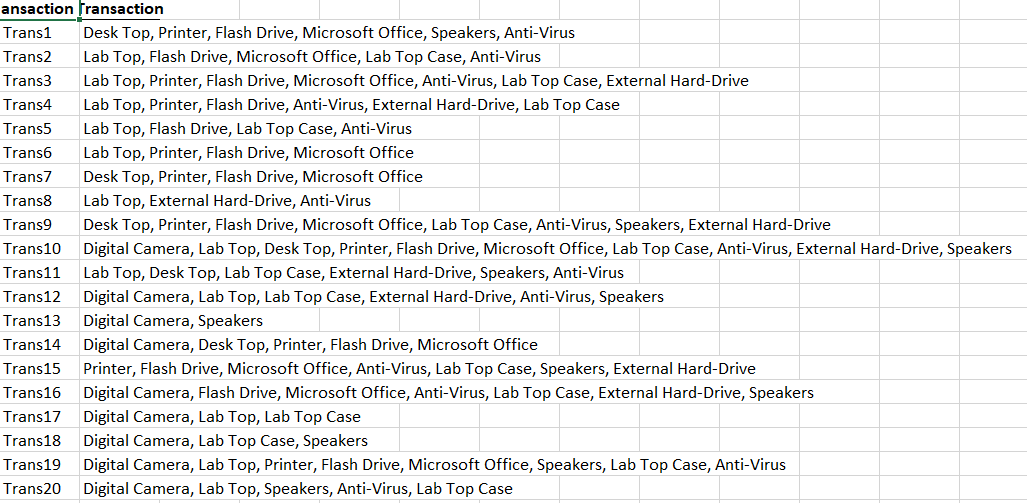
Amazon:



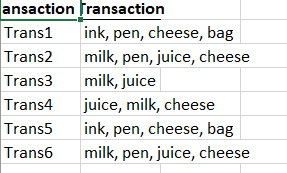
K-Mart:



BestBuy:



Custom:



Nike:



2.3 Datsets Note:

• The datasets were provided by the faculty for this project.

• I have converted and saved them in CSV file format.

• No random or artificial data was used in this analysis.

3. Brute Force Algorithm

3.1 Method

1. The code first converts all transactions into sets.
2. It counts how often each single item appears.
3. It then generates larger itemsets (pairs, triplets) and checks their support one by one.
4. If the itemset meets the minimum support, it’s kept as frequent.
5. Finally, it forms rules (A → B) and prints their confidence and support.

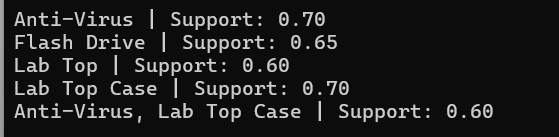
3.2 Example run

Dataset: Bestbuy.csv

Parameters: Support = 0.6, Confidence = 0.6

Output:

Frequent Itemsets:



Association rules:



4. Apriori and FP-Growth

4.1 Apriori:

1. The code encodes all transactions using **TransactionEncoder** to create a one-hot DataFrame.
2. It runs the **apriori()** function from mlxtend to find frequent itemsets.
3. It then applies **association\_rules()** to get rules based on confidence.
4. It displays itemsets with their support and rules with confidence values.
5. Execution time is recorded and printed for comparison.

Results same as Brute Force.

4.2 FP-Growth:

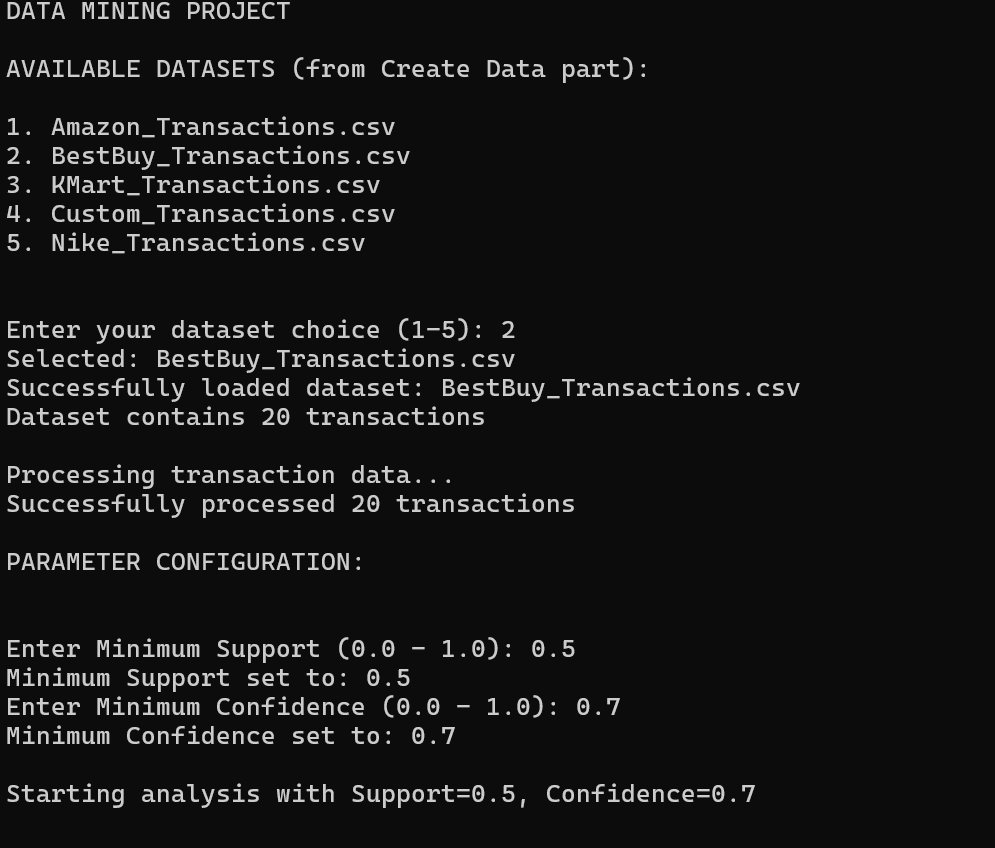
1. Similar to Apriori, it encodes transactions using TransactionEncoder.
2. It runs the fpgrowth() function from mlxtend instead of apriori.
3. FP-Growth builds an FP-Tree internally to find frequent patterns efficiently.
4. The association\_rules() function is used again to form rules.
5. It displays results and runtime alongside the other methods for comparison.

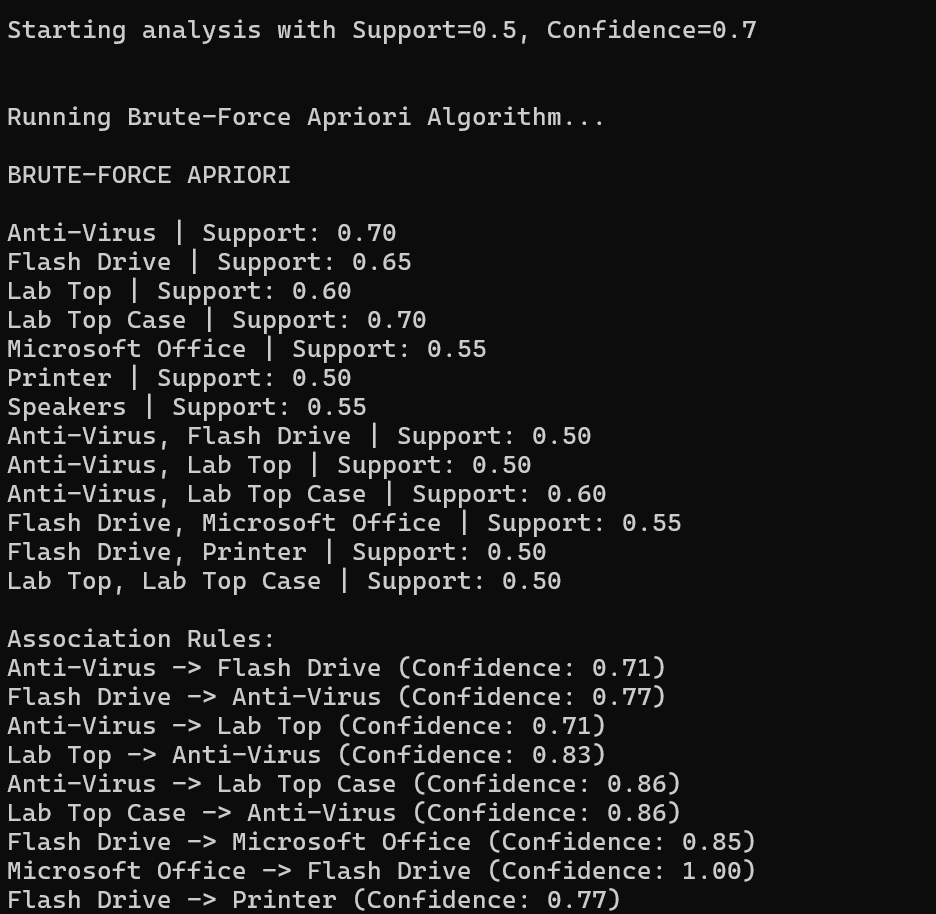
Results same as Brute Force

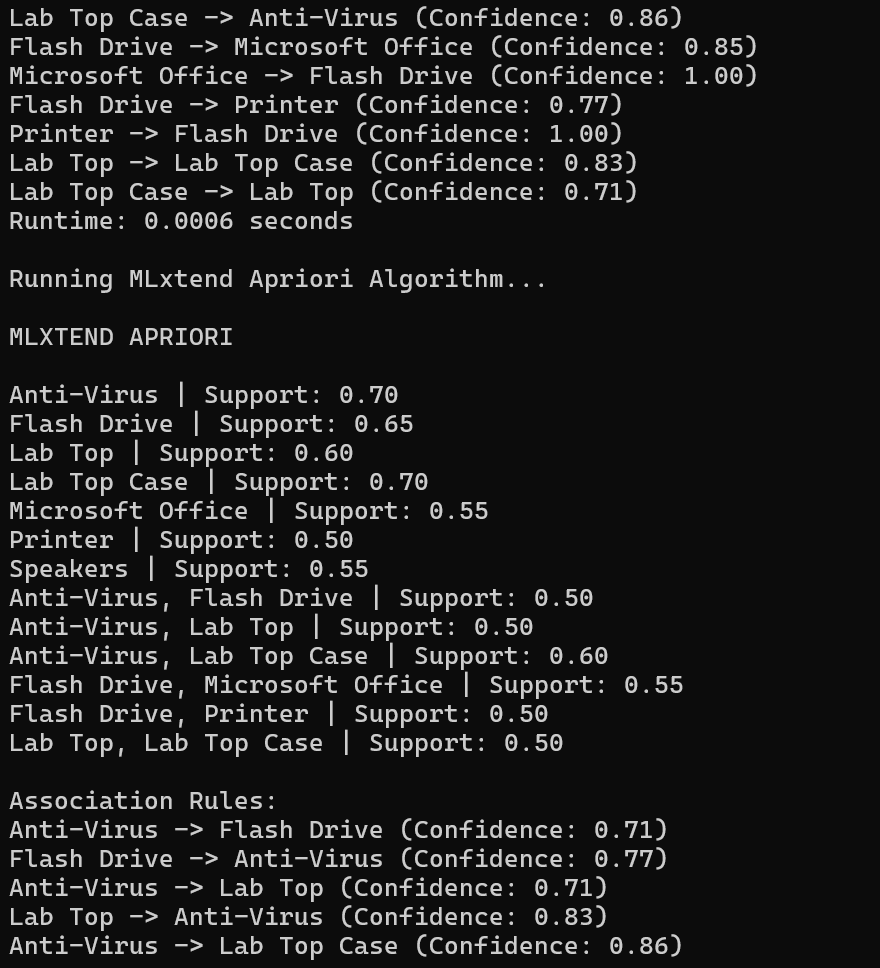
5.Multiple Parameters:

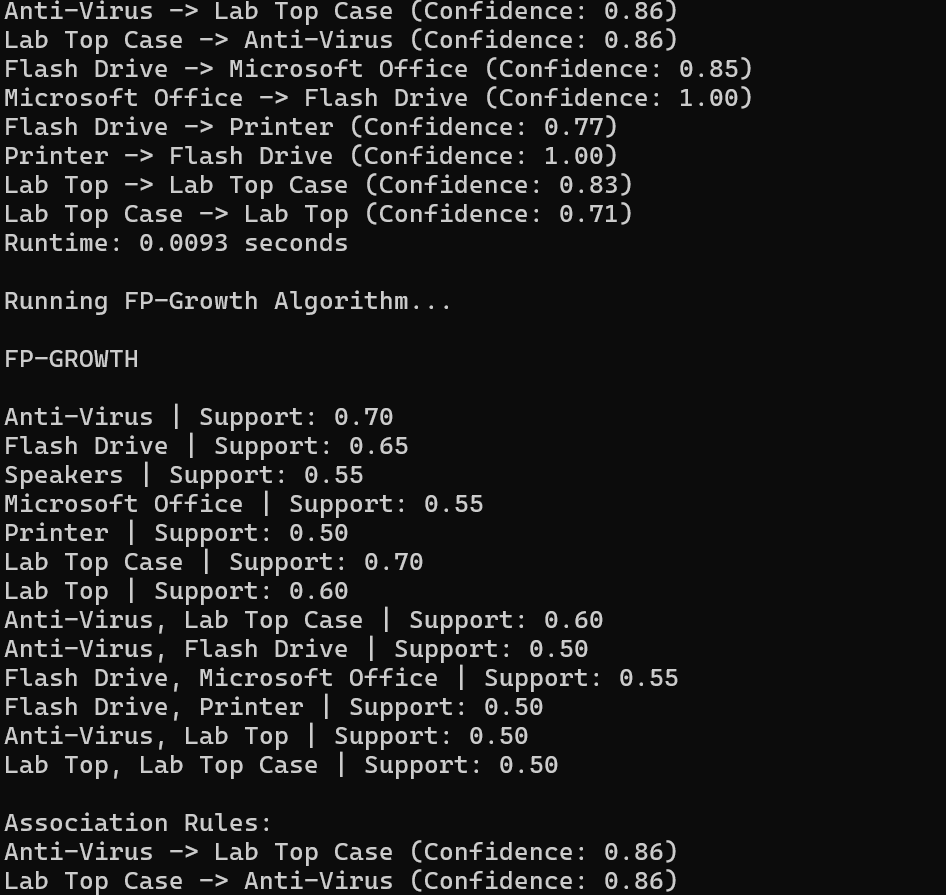
Tested here 3 settings for a BestBuy dataset and attached the output

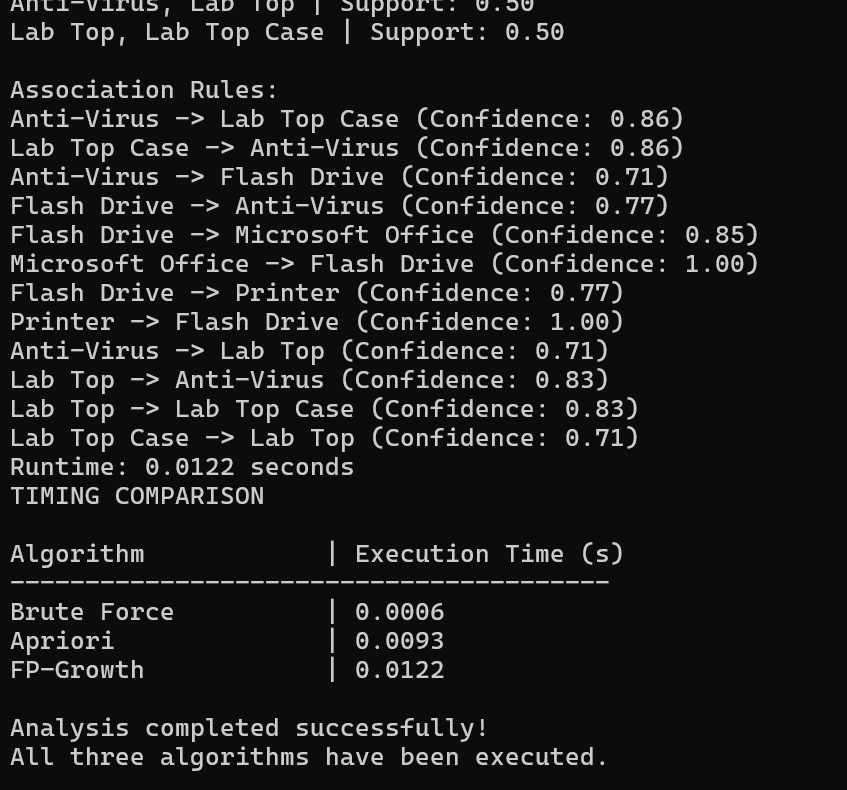
Support: 0.5, confidence: 0.7



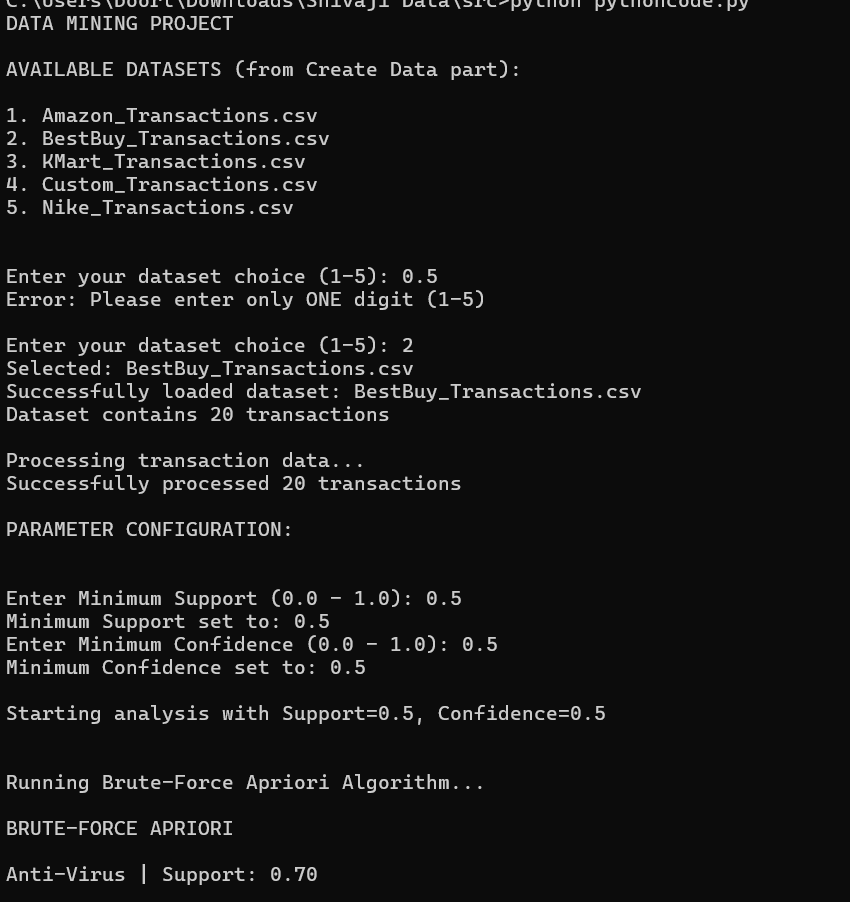




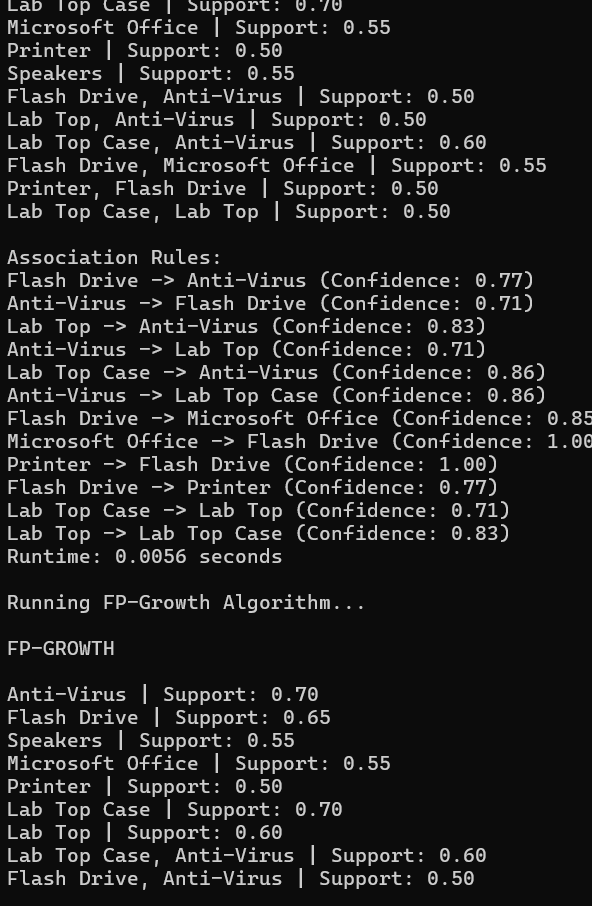


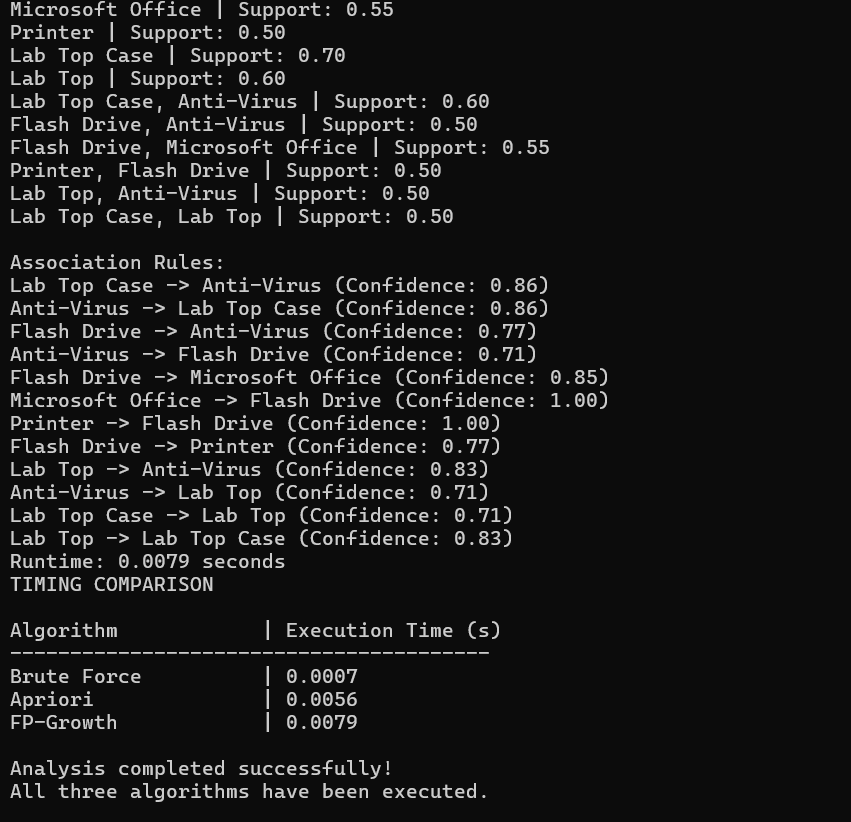


Support: 0.5, confidence: 0.5

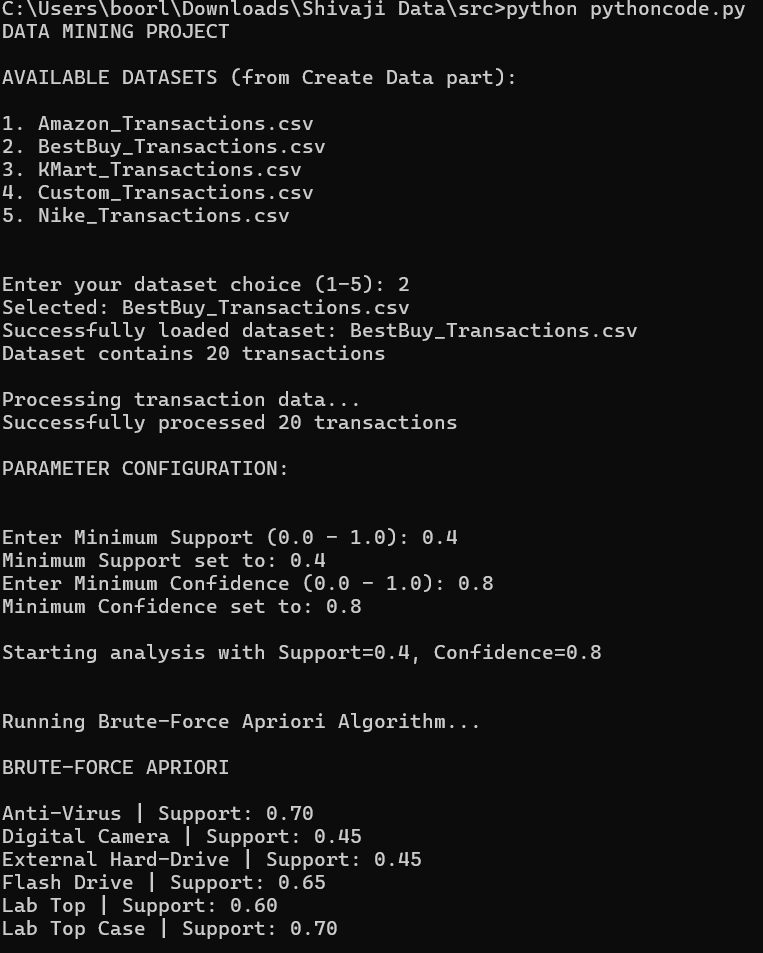


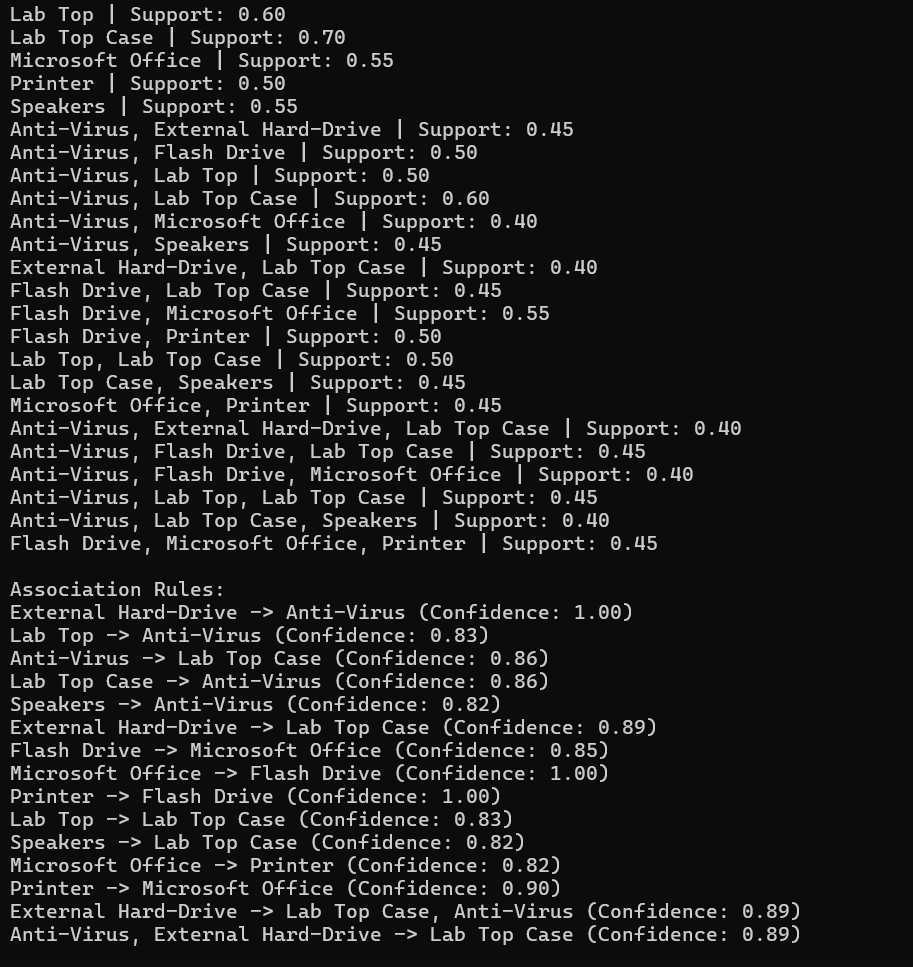


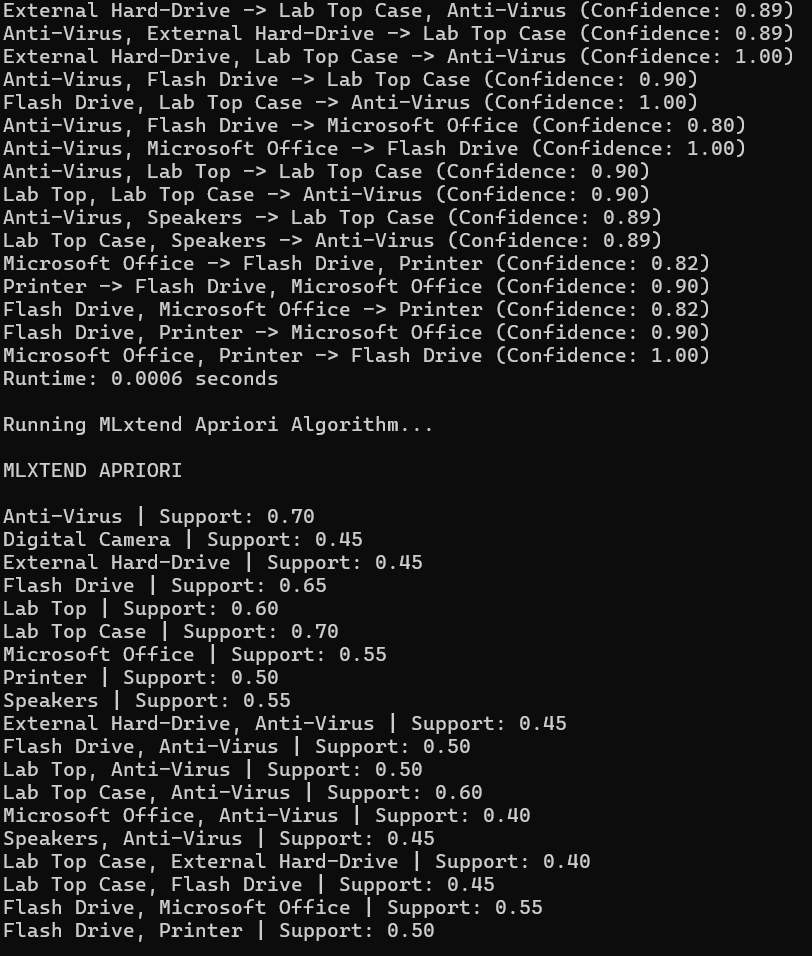


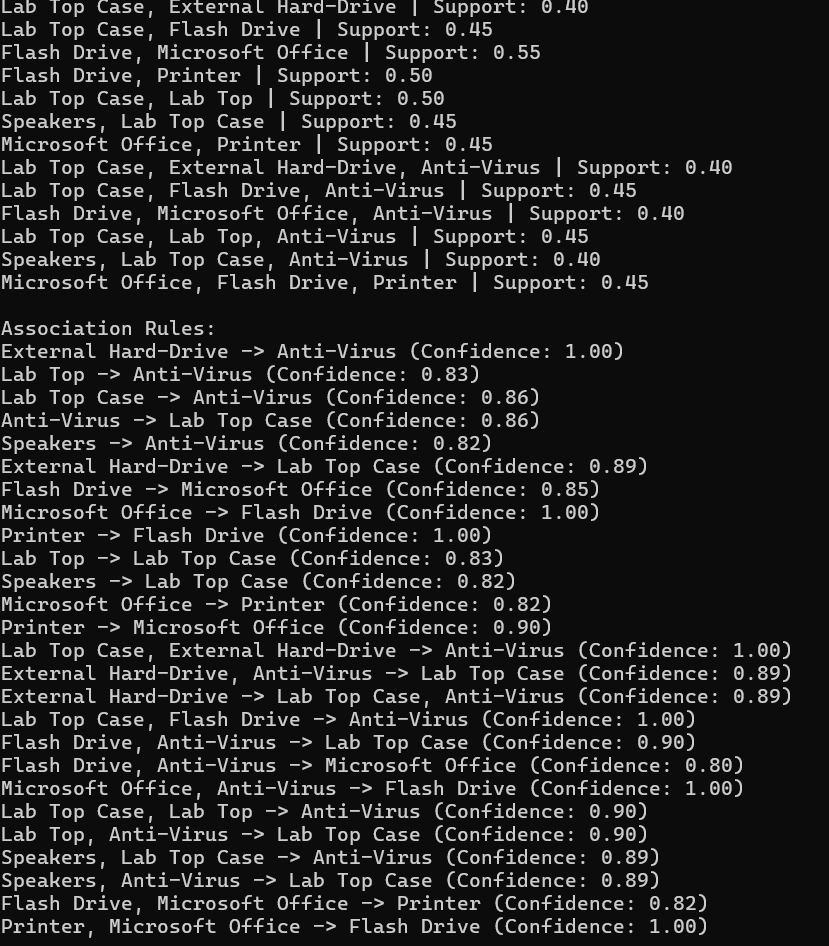


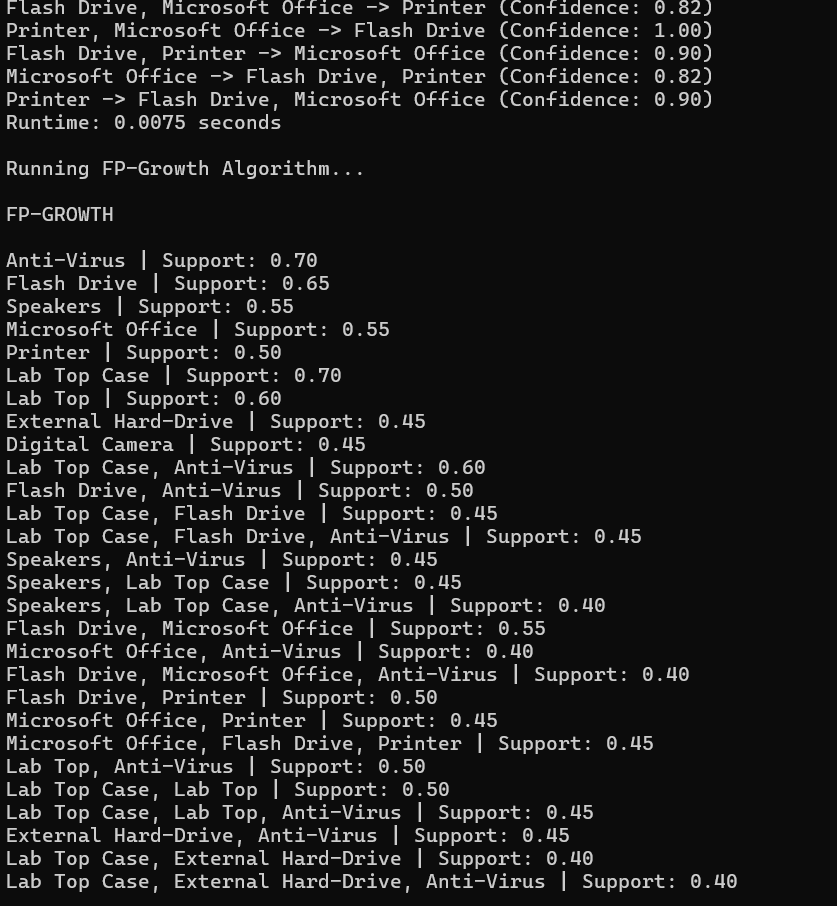
Support: 0.4, confidence: 0.8

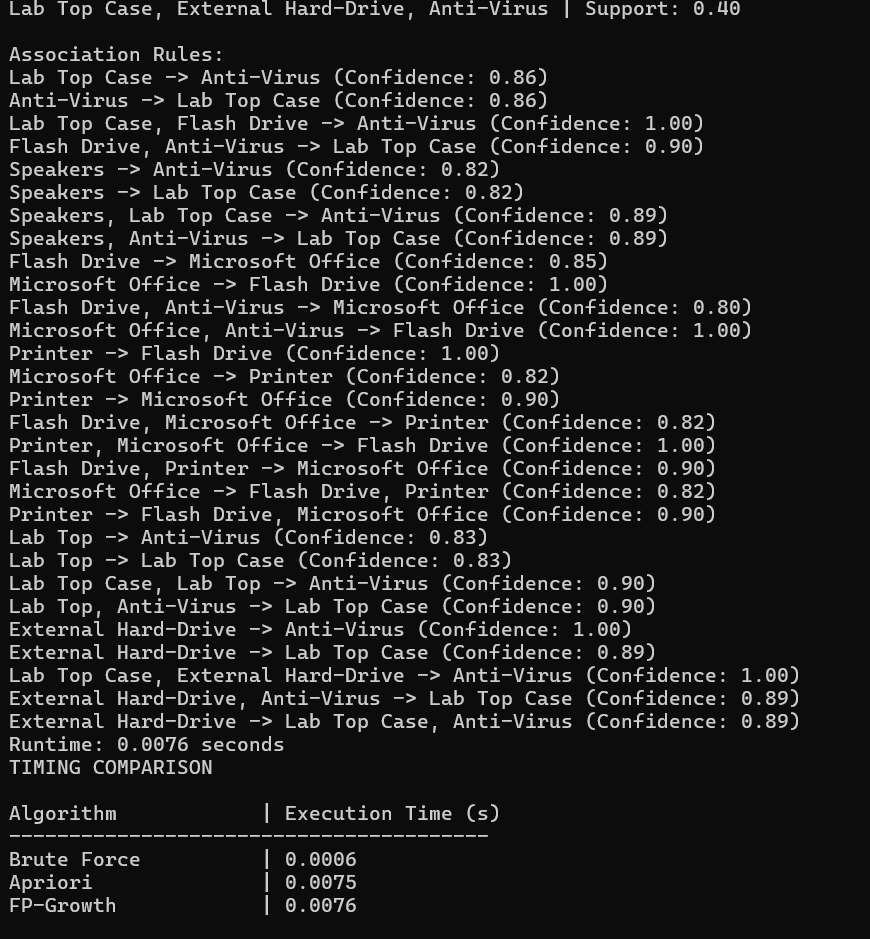


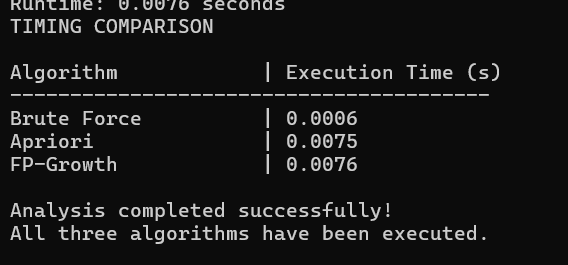












6.Github Repository

Code and datasets uploaded here: [burleshivajishivajiburle-ops/burle\_shivaji\_midtermproject](https://github.com/burleshivajishivajiburle-ops/burle_shivaji_midtermproject)  
Collaborator access given to [ya54@njit.edu](mailto:ya54@njit.edu)

7.How to run the code

7.1 Installation Requirements

Environment Setup

Required Dependencies

- pandas

- mlxtend

Install the core packages:

pip install pandas

pip install mlxtend

OPTIONAL:

If you encounter dependency/version issues, a frozen requirements file is provided. From the project root burle\_shivaji\_midtermproject and run pip install -r requirements.txt

To avoid system-wide installs and keep dependencies isolated to this project:

python -m venv venv

venv\Scripts\activate

(This creates and activates a virtual environment named 'venv' on Windows.)

# For macOS/Linux: source venv/bin/activate

After activation, install the required packages using the commands above or via requirements.txt.

7.2 Run options

PYTHON FILE

1. Extract the archive

Unzip burle\_shivaji\_midtermproject.zip to a folder on your computer.

2. Open a terminal

- Windows: Command Prompt

- macOS/Linux: Terminal

3. Go to the source directory

cd into the src folder.

4. Run the program

- Windows: python pythoncode.py

- macOS/Linux: python3 pythoncode.py

5. Home menu

When prompted, choose your dataset.

6. After selecting a dataset

You’ll be asked for min support and min confidence.

The program computes frequent itemsets and association rules using:

- Brute Force Apriori

- Apriori (mlxtend)

- FP-Growth (mlxtend)

It also prints timing comparisons for all three methods.

IPYNB FILE

1. Open your notebook environment

Jupyter Notebook, JupyterLab, or VS Code.

2. Open the notebook

Navigate to the notebook folder or open the notebook file from there.

3. Run cells top-to-bottom

Execute each cell in order.

4. Same behavior as the CLI

You’ll be prompted for min support and min confidence.

The notebook displays frequent itemsets, association rules, and timing comparisons.

RUN FROM GITHUB

1. Clone the repository

git clone "[burleshivajishivajiburle-ops/burle\_shivaji\_midtermproject](https://github.com/burleshivajishivajiburle-ops/burle_shivaji_midtermproject)"

2. Enter the project directory

cd into the cloned folder.

3. Run as above

- Python script: cd src and run python pythoncode.py (or python3 on macOS/Linux).

- Notebook: cd notebook and run the notebook cell-by-cell.

8.Conclusion

This project implemented and compared three approaches to frequent itemset mining and association rule learning—Brute Force, Apriori, and FP-Growth—on multiple retail-style datasets (Amazon, Best Buy, K-Mart, Nike, and a custom set) that were curated and converted to CSV for analysis. Using support and confidence as the controlling parameters, we generated frequent itemsets and confidence-based rules across methods. Conceptually, Brute Force served as a correctness baseline through exhaustive candidate evaluation, Apriori improved efficiency via level-wise pruning; and FP-Growth offered the strongest scalability by leveraging a compact FP-Tree with fewer passes over the data. In practice, Apriori and FP-Growth reproduced the same patterns and rules as Brute Force while completing in substantially less time—especially as itemset size and dataset volume increased. Parameter experiments with different support–confidence settings further illustrated the trade-off between broader pattern coverage at lower thresholds and higher rule reliability at stricter thresholds.

While Brute Force becomes impractical on larger datasets, Apriori and FP-Growth proved both accurate and efficient for the tasks considered here. Future work can extend this study to larger, more diverse transaction logs, add rule visualizations and post-filtering for interpretability, and explore scaling, parallelization for high-volume scenarios. Collectively, the implementation, timed comparisons, and parameter sweeps provide a solid, replicable foundation for applying association-rule mining to real retail analytics and decision-support use cases.