CS634 Midterm Project Report

Course: CS634 Data Mining
Student: Tanushri Vijayakumar
Instructor: Dr. Yasser Abdullah
Email: tv233@njit.edu

INTRODUCTION

This project explores association rule mining which is used to find frequent patterns, associations or correlations among items in large datasets. In this project we compare three different approaches for finding frequent patterns and mining association rules:

1. *Brute Force Approach*
2. *Apriori Algorithm*
3. *FP- Growth Algorithm*

The goal is to implement, run, and compare these approaches in terms of frequent itemsets discovered, rules generated, and execution time.
So, we begin by creating and organizing transaction datasets in CSV format. Each dataset simulates customer purchases in different stores like Amazon, BestBuy, Walmart, etc. When the program runs, the user is first prompted to select the dataset of their choice from the available options. After that, the program asks the user to provide two key thresholds: the minimum support and the minimum confidence. These thresholds determine which itemsets are considered as frequent and which association rules are strong enough to be kept. Once the inputs are provided, the program proceeds to generate candidate itemsets, and finally output the frequent itemsets and the association rules.

DATASET CREATION

I created transaction-style CSV datasets representing purchases from different stores where all the transactions are deterministic. Each dataset was saved as a CSV file in the dataset/ folder for consistency.

Files available:

- amazon_transactions.csv
- bestbuy_transactions.csv
- walmart_transactions.csv
- nike_transactions.csv
- wholefoods_transactions.csv

Example dataset - amazon_transactions.csv

Item No.	Item Name	TID	Transaction
1	A Beginner's Guide	T1	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
2	Java: The Complete Reference	T2	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Effective Java (2nd Edition)
3	Java For Dummies	T3	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition, Effective Java (2nd Edition)
4	Android Programming: The Big Nerd Ranch	T4	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition, Beginning Programming with Java
5	Head First Java 2nd Edition	T5	Android Programming: The Big Nerd Ranch, Beginning Programming with Java, Java 8 Pocket Guide
6	Beginning Programming with Java	T6	A Beginner's Guide, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
7	Java 8 Pocket Guide	T7	A Beginner's Guide, Head First Java 2nd Edition, Beginning Programming with Java
8	C++ Programming in Easy Steps	T8	Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
9	Effective Java (2nd Edition)	T9	Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition, Beginning Programming with Java, Effective Java (2nd Edition)
10	HTML and CSS: Design and Build Websites	T10	Beginning Programming with Java, Java 8 Pocket Guide, C++ Programming in Easy Steps
		T11	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
		T12	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, HTML and CSS: Design and Build Websites
		T13	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Java 8 Pocket Guide, HTML and CSS: Design and Build Websites
		T14	Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
		T15	Java For Dummies, Android Programming: The Big Nerd Ranch
		T16	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
		T17	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
		T18	Head First Java 2nd Edition, Beginning Programming with Java, Java 8 Pocket Guide
		T19	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
		T20	A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Effective Java (2nd Edition)

Each dataset is a CSV file and contains the following 4 columns:

- Item No. - identifies each item in the dataset
- Item Name - the name of the product
- TID - a unique ID for each transaction (e.g., T1, T2, T3).
- Transaction - a comma-separated list of items purchased together in one transaction.

Each row corresponds to a single transaction made by a customer, and the Transaction field lists all the books/items bought together. This structure was followed across all store datasets (Amazon, Walmart, etc.), which allows users to select any dataset at runtime and run the mining algorithms on it.

PACKAGES AND INSTALLATION STEPS

The following python libraries were used:

- pip install pandas
- pip install mlxtend
- pip install jupyter

pandas: for handling CSVs and one-hot encoding transactions
mlxtend: provides built-in implementations of apriori, fpgrowth, and association_rules
jupyter: to run and document the project interactively

Additional built-in libraries used:

- ***os***: file path handling
- ***time:*** measuring execution time

BRUTE FORCE APPROACH

The brute force method was implemented from scratch in Python. This implementation helps illustrate the internal mechanics of Apriori and serves as a baseline for comparison.

Steps:

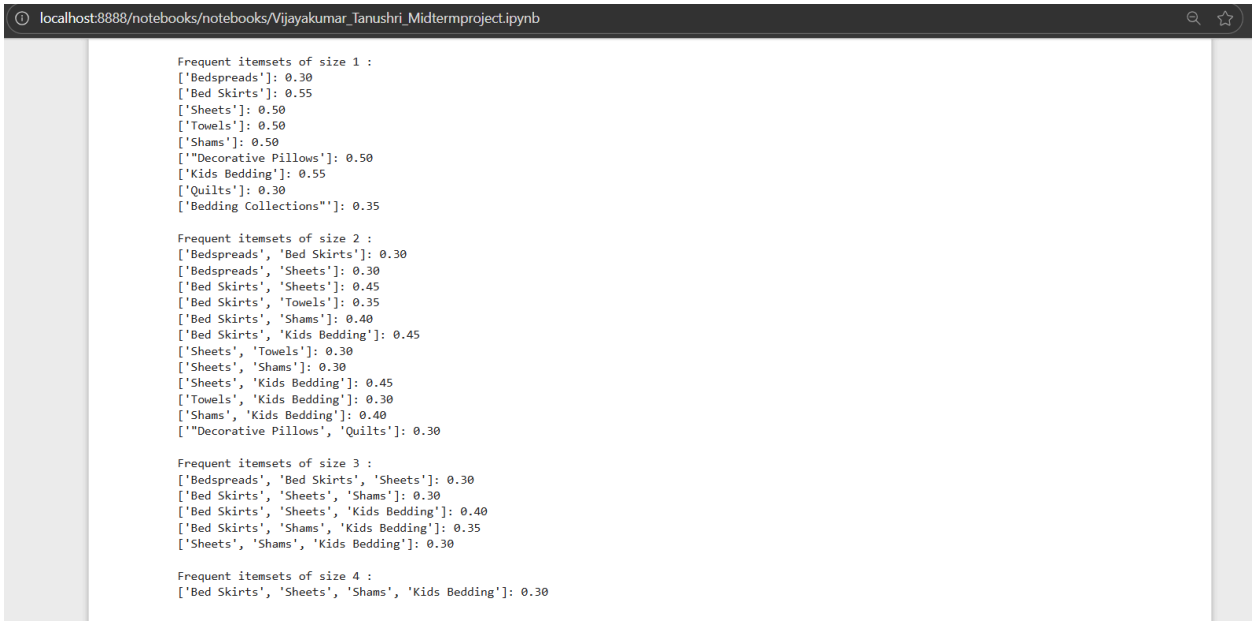
1. Generate all possible candidate items of size k
2. Calculate their support values across transactions
3. Select those meeting the minimum support threshold which is the frequent itemsets
4. Repeat until no more frequent itemsets can be found
5. Generate association rules by splitting each frequent itemset into antecedent-consequent pairs and calculating confidence

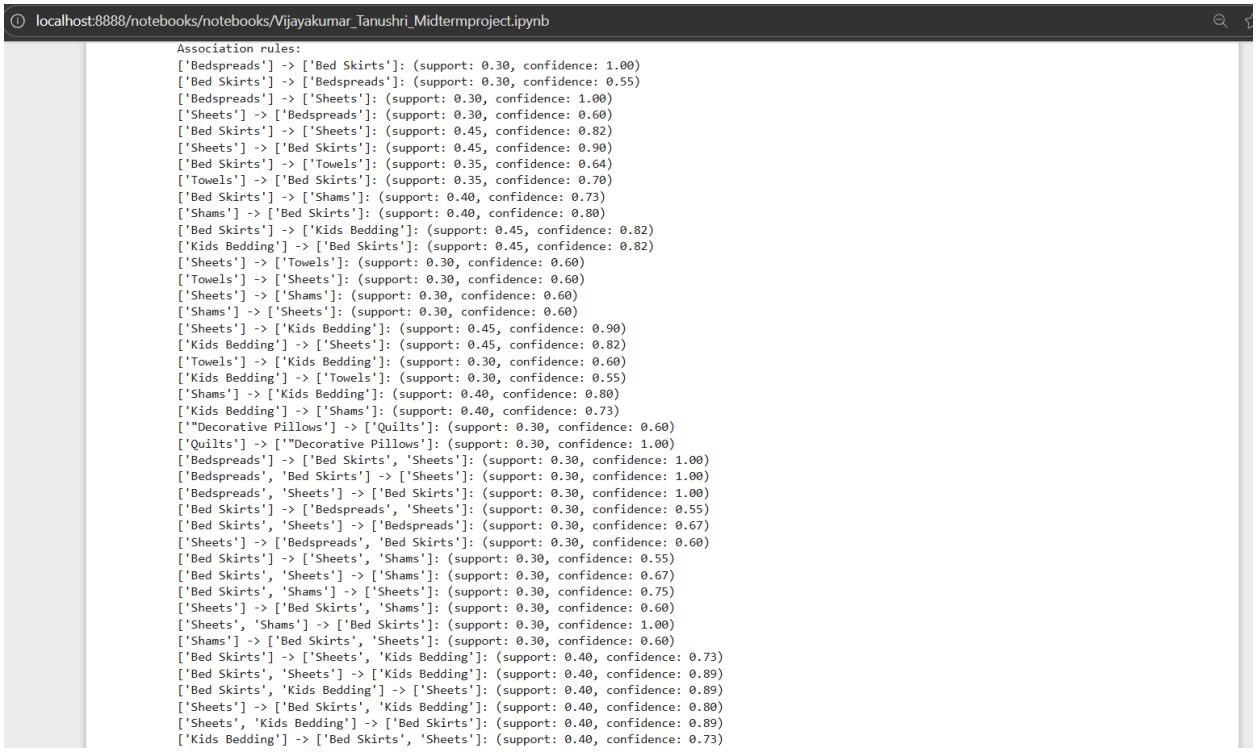
Sample Input:

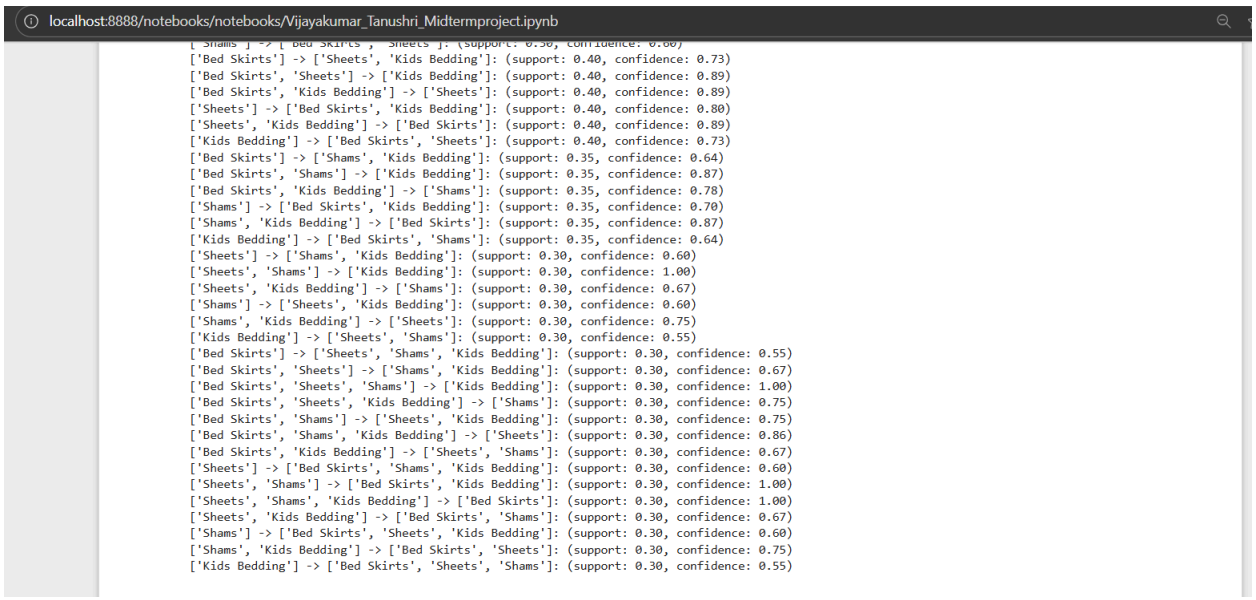
```
AVAILABLE DATASETS:
1. amazon_transactions.csv
2. bestbuy_transactions.csv
3. walmart_transactions.csv
4. nike_transactions.csv
5. wholefoods_transactions.csv


Enter the dataset number to choose which dataset to use:  3
Enter the minimum support threshold (0-1):  0.3
Enter the minimum confidence threshold (0-1):  0.5
```

Sample output:







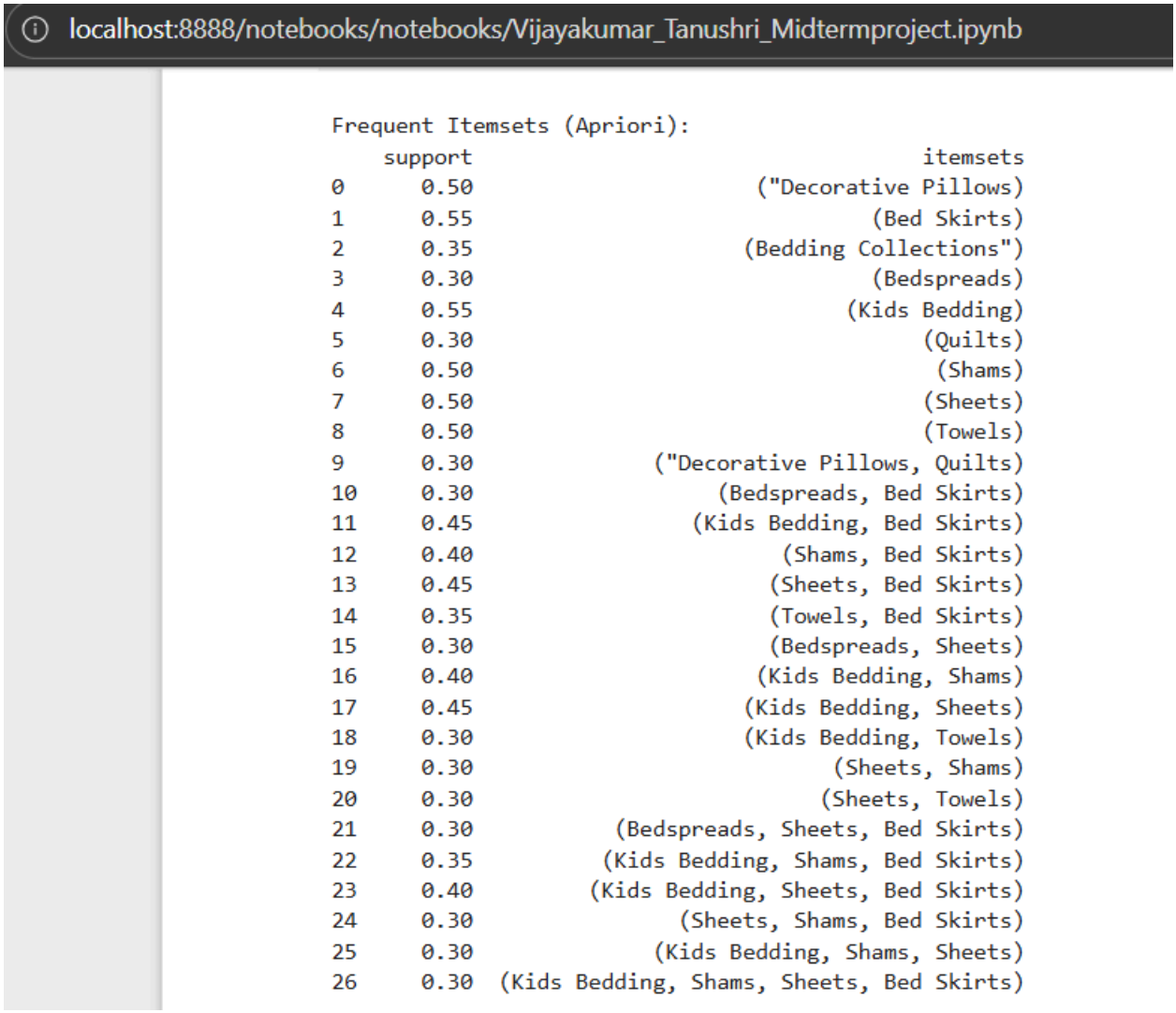
APRIORI ALGORITHM

The Apriori algorithm is an efficient improvement over brute force. It uses the Apriori property: all non-empty subsets of a frequent itemset must also be frequent. This reduces the number of candidate sets drastically compared to brute force. In this project, we used Python's built-in libraries for Apriori Implementation.

Steps:

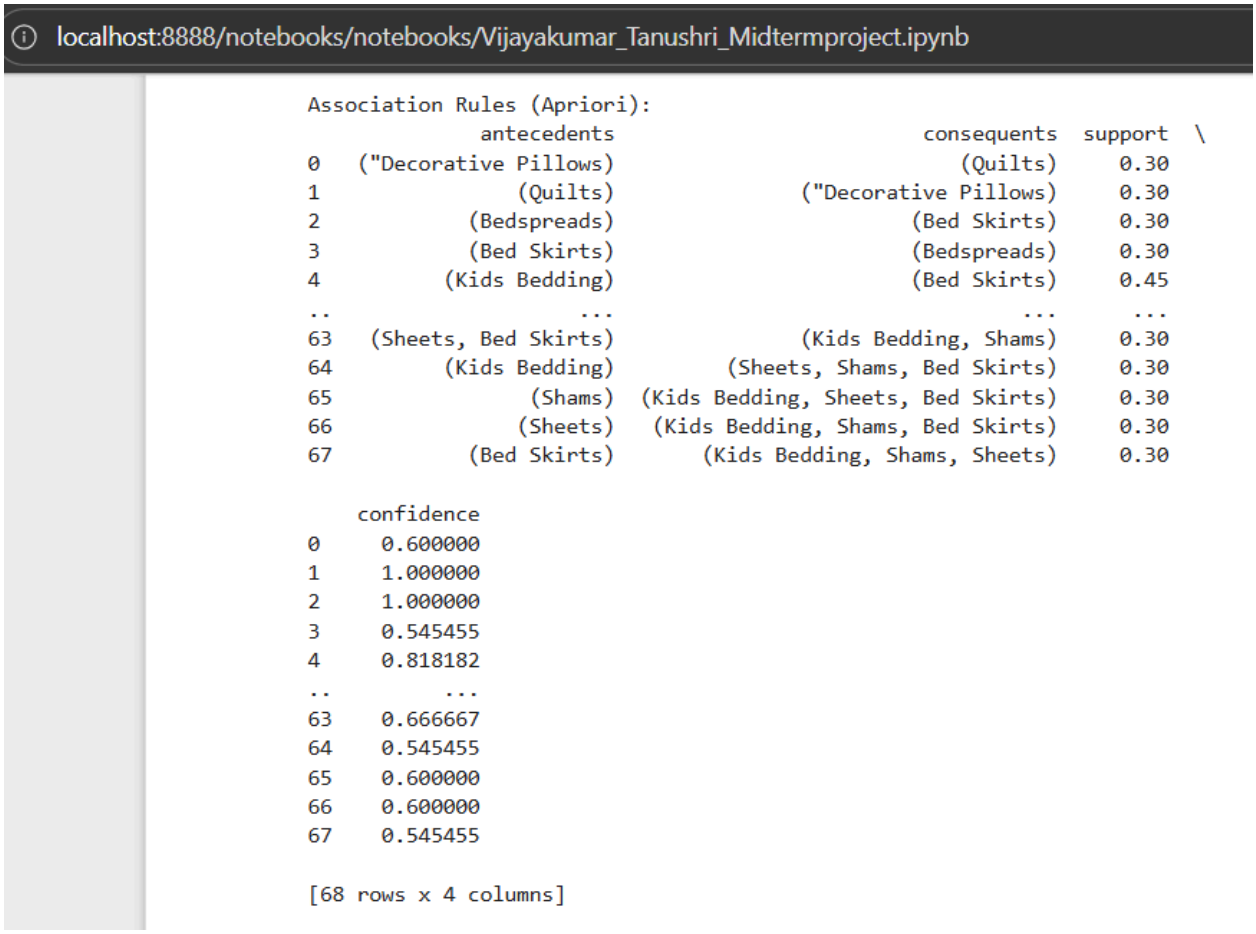
1. Convert transactions into a one-hot encoded DataFrame
2. Run mlxtend.frequent_patterns.apriori() to find frequent itemsets
3. Use association_rules() to generate rules based on minimum confidence

Sample output:



Frequent Itemsets (Apriori):

	support	itemsets
0	0.50	("Decorative Pillows)
1	0.55	(Bed Skirts)
2	0.35	(Bedding Collections")
3	0.30	(Bedspreads)
4	0.55	(Kids Bedding)
5	0.30	(Quilts)
6	0.50	(Shams)
7	0.50	(Sheets)
8	0.50	(Towels)
9	0.30	("Decorative Pillows, Quilts)
10	0.30	(Bedspreads, Bed Skirts)
11	0.45	(Kids Bedding, Bed Skirts)
12	0.40	(Shams, Bed Skirts)
13	0.45	(Sheets, Bed Skirts)
14	0.35	(Towels, Bed Skirts)
15	0.30	(Bedspreads, Sheets)
16	0.40	(Kids Bedding, Shams)
17	0.45	(Kids Bedding, Sheets)
18	0.30	(Kids Bedding, Towels)
19	0.30	(Sheets, Shams)
20	0.30	(Sheets, Towels)
21	0.30	(Bedspreads, Sheets, Bed Skirts)
22	0.35	(Kids Bedding, Shams, Bed Skirts)
23	0.40	(Kids Bedding, Sheets, Bed Skirts)
24	0.30	(Sheets, Shams, Bed Skirts)
25	0.30	(Kids Bedding, Shams, Sheets)
26	0.30	(Kids Bedding, Shams, Sheets, Bed Skirts)




```
Association Rules (Apriori):
              antecedents                        consequents  support  \
0   ("Decorative Pillows)                           (Quilts)     0.30
1                 (Quilts)              ("Decorative Pillows)     0.30
2            (Bedspreads)                        (Bed Skirts)     0.30
3            (Bed Skirts)                        (Bedspreads)     0.30
4          (Kids Bedding)                        (Bed Skirts)     0.45
..                    ...                                 ...      ...
63   (Sheets, Bed Skirts)              (Kids Bedding, Shams)     0.30
64         (Kids Bedding)         (Sheets, Shams, Bed Skirts)     0.30
65                (Shams)  (Kids Bedding, Sheets, Bed Skirts)     0.30
66               (Sheets)   (Kids Bedding, Shams, Bed Skirts)     0.30
67           (Bed Skirts)       (Kids Bedding, Shams, Sheets)     0.30

    confidence
0     0.600000
1     1.000000
2     1.000000
3     0.545455
4     0.818182
..         ...
63    0.666667
64    0.545455
65    0.600000
66    0.600000
67    0.545455

[68 rows x 4 columns]
```

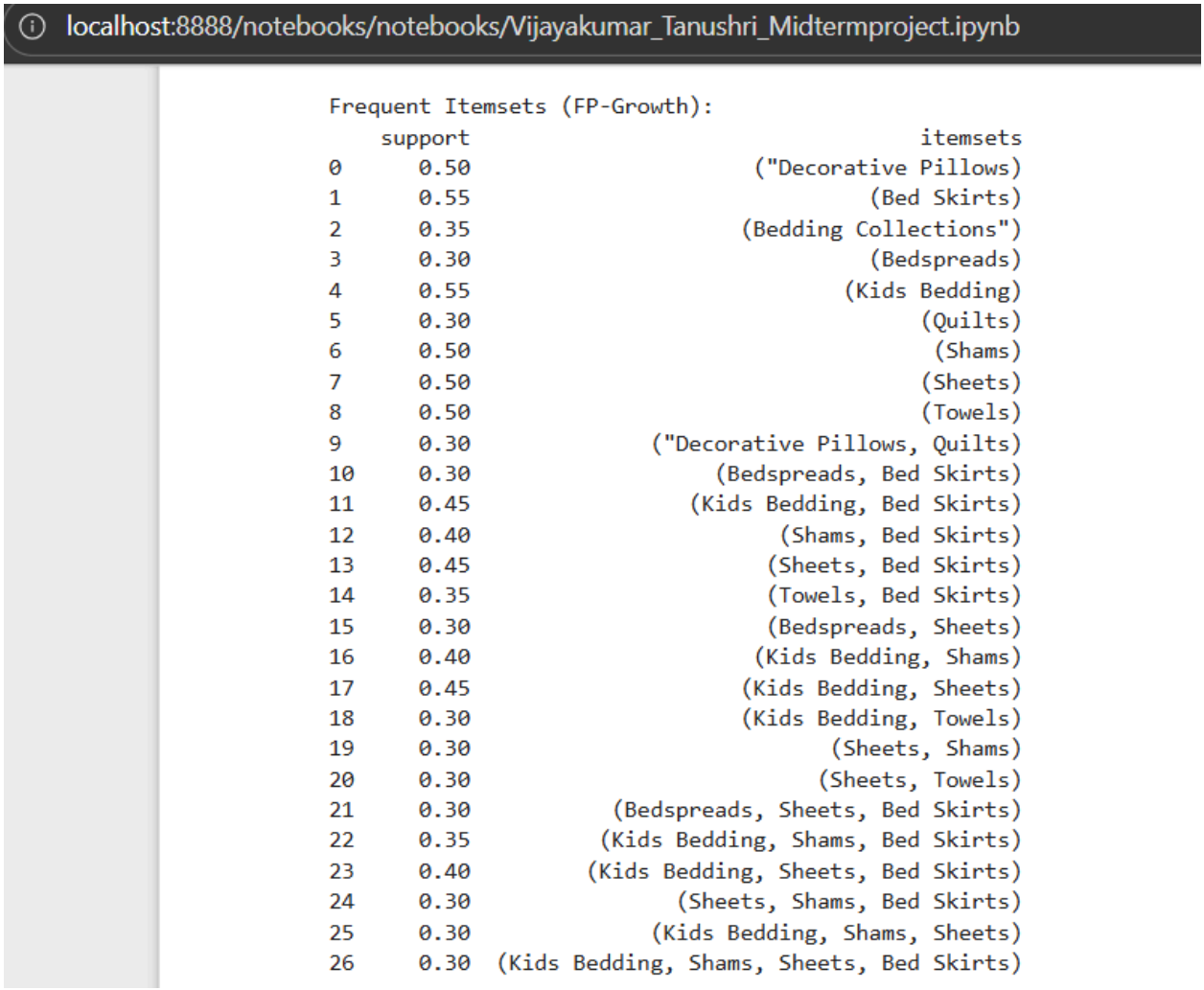
FP-GROWTH ALGORITHM

The FP-Growth algorithm is a further optimization over Apriori. Instead of generating candidate sets, it compresses the dataset into a Frequent Pattern Tree. It recursively mines frequent itemsets directly from the FP-tree.

Steps:

1. Convert transactions into a one-hot encoded DataFrame
2. Run mlxtend.frequent_patterns.fpgrowth() to find frequent itemsets
3. Use association_rules() for rule generation

Sample Output:



Frequent Itemsets (FP-Growth):

	support	itemsets
0	0.50	("Decorative Pillows)
1	0.55	(Bed Skirts)
2	0.35	(Bedding Collections")
3	0.30	(Bedspreads)
4	0.55	(Kids Bedding)
5	0.30	(Quilts)
6	0.50	(Shams)
7	0.50	(Sheets)
8	0.50	(Towels)
9	0.30	("Decorative Pillows, Quilts)
10	0.30	(Bedspreads, Bed Skirts)
11	0.45	(Kids Bedding, Bed Skirts)
12	0.40	(Shams, Bed Skirts)
13	0.45	(Sheets, Bed Skirts)
14	0.35	(Towels, Bed Skirts)
15	0.30	(Bedspreads, Sheets)
16	0.40	(Kids Bedding, Shams)
17	0.45	(Kids Bedding, Sheets)
18	0.30	(Kids Bedding, Towels)
19	0.30	(Sheets, Shams)
20	0.30	(Sheets, Towels)
21	0.30	(Bedspreads, Sheets, Bed Skirts)
22	0.35	(Kids Bedding, Shams, Bed Skirts)
23	0.40	(Kids Bedding, Sheets, Bed Skirts)
24	0.30	(Sheets, Shams, Bed Skirts)
25	0.30	(Kids Bedding, Shams, Sheets)
26	0.30	(Kids Bedding, Shams, Sheets, Bed Skirts)

localhost:8888/notebooks/notebooks/Vijayakumar_Tanushri_Midtermproject.ipynb

```
Association Rules (FP-Growth):
                 antecedents                       consequents  support  \
0   ("Decorative Pillows)                           (Quilts)     0.30
1                 (Quilts)                ("Decorative Pillows)     0.30
2            (Bedspreads)                          (Bed Skirts)     0.30
3            (Bed Skirts)                          (Bedspreads)     0.30
4          (Kids Bedding)                          (Bed Skirts)     0.45
..                    ...                                   ...      ...
63   (Sheets, Bed Skirts)               (Kids Bedding, Shams)     0.30
64         (Kids Bedding)          (Sheets, Shams, Bed Skirts)     0.30
65                (Shams)  (Kids Bedding, Sheets, Bed Skirts)     0.30
66               (Sheets)   (Kids Bedding, Shams, Bed Skirts)     0.30
67           (Bed Skirts)       (Kids Bedding, Shams, Sheets)     0.30

    confidence
0     0.600000
1     1.000000
2     1.000000
3     0.545455
4     0.818182
..         ...
63    0.666667
64    0.545455
65    0.600000
66    0.600000
67    0.545455

[68 rows x 4 columns]
```

EXECUTION COMPARISON

For the above mentioned sample input and output, I got the following:

To evaluate the efficiency of the three algorithms, we measured the execution time for each: Brute Force, Apriori, FP-Growth

```
13]: df_times = pd.DataFrame(results, columns=["Algorithm", "Execution Time (seconds)"])
     print(df_times)
```

```
     Algorithm  Execution Time (seconds)
0  Brute Force                  0.291917
1      Apriori                  0.027268
2    FP-Growth                  0.011993
```

I used Python's built-in timing tools (%time in Jupyter and time module in scripts) to compute the runtime for generating frequent itemsets and association rules.

I observed that:

Brute Force is the slowest since it checks every possible itemset. Apriori is faster because it prunes unnecessary candidates early. FP-Growth is the fastest, using a tree structure to avoid candidate generation.

Surprisingly, for small datasets, brute force appeared faster, since the dataset size was not large enough to showcase its inefficiency.

STEPS TO RUN THE CODE

Clone the repository from GitHub. Activate the Python virtual environment (venv)
Navigate to the source_code/ folder and run desired script:
python brute_force.py
python apriori.py
python fp_growth.py

Alternatively, open the Jupyter notebook:
‘ jupyter notebook’ and run midterm_association_rule_mining_comparisons.ipynb

MY REPO STRUCTURE

main/

- dataset/
  - amazon_transactions.csv
  - bestbuy_transactions.csv
  - walmart_transactions.csv
  - nike_transactions.csv
  - wholefoods_transactions.csv
- source_code/
  - brute_force.py
  - apriori.py
  - fp_growth.py
- notebooks/
  - Vijayakumar_Tanushri_Midtermproject.ipynb
- report/
  - Vijayakumar_Tanushri_mideterm_report.pdf
- requirements.txt
- readme.md

GITHUB REPOSITORY

https://github.com/TanushriVijay12/Vijayakumar_Tanushri_midtermproject_datamining

CONCLUSION

The project successfully implemented and compared three association rule mining techniques: Brute Force, Apriori, and FP-Growth.

- Brute force is educational but computationally expensive for larger datasets.
- Apriori improves efficiency by pruning candidate sets.
- FP-Growth is the most scalable and efficient method for large transaction datasets.