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.

2. 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**

a) 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

b) 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

c) K-Mart Dataset

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

d) Nike Dataset

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

e) 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.

3. 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 $X \rightarrow Y$. It calculates confidence as: $\text{Confidence}(X \rightarrow Y) = \text{Support}(X) / \text{Support}(X \cup Y)$

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

```
Running Brute Force Algorithm ...

=== 1-itemsets ===
('Android Programming: The Big Nerd Ranch',): support=0.65, count=13
('Java For Dummies',): support=0.65, count=13
('ABeginner's Guide',): support=0.55, count=11
('Java: The Complete Reference',): support=0.50, count=10
('Head First Java 2nd Edition',): support=0.40, count=8
('Beginning Programming with Java',): support=0.30, count=6

=== 2-itemsets ===
('Java For Dummies', 'Java: The Complete Reference'): support=0.50, count=10
('ABeginner's Guide', 'Java For Dummies'): support=0.45, count=9
('ABeginner's Guide', 'Java: The Complete Reference'): support=0.45, count=9
('Android Programming: The Big Nerd Ranch', 'Java For Dummies'): support=0.45, count=9
('ABeginner's Guide', 'Android Programming: The Big Nerd Ranch'): support=0.30, count=6
('Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edition'): support=0.30, count=6
('Android Programming: The Big Nerd Ranch', 'Java: The Complete Reference'): support=0.30, count=6

=== 3-itemsets ===
('ABeginner's Guide', 'Java For Dummies', 'Java: The Complete Reference'): support=0.45, count=9
('Android Programming: The Big Nerd Ranch', 'Java For Dummies', 'Java: The Complete Reference'): support=0.30, count=6

=== Association Rules ===
('Java: The Complete Reference',) -> ('Java For Dummies',) (support=0.50, confidence=1.00, count=10)
('ABeginner's Guide', 'Java For Dummies') -> ('Java: The Complete Reference',) (support=0.45, confidence=1.00, count=9)
('ABeginner's Guide', 'Java: The Complete Reference') -> ('Java For Dummies',) (support=0.45, confidence=1.00, count=9)
('Android Programming: The Big Nerd Ranch', 'Java: The Complete Reference') -> ('Java For Dummies',) (support=0.30, confidence=1.00, count=6)
('Java: The Complete Reference',) -> ('ABeginner's Guide',) (support=0.45, confidence=0.90, count=9)
('Java: The Complete Reference',) -> ('ABeginner's Guide', 'Java For Dummies') (support=0.45, confidence=0.90, count=9)
('Java For Dummies', 'Java: The Complete Reference') -> ('ABeginner's Guide',) (support=0.45, confidence=0.90, count=9)
('ABeginner's Guide',) -> ('Java For Dummies',) (support=0.45, confidence=0.82, count=9)
('ABeginner's Guide',) -> ('Java: The Complete Reference',) (support=0.45, confidence=0.82, count=9)
('ABeginner's Guide',) -> ('Java For Dummies', 'Java: The Complete Reference') (support=0.45, confidence=0.82, count=9)
('Java For Dummies',) -> ('Java: The Complete Reference',) (support=0.50, confidence=0.77, count=10)
('Head First Java 2nd Edition',) -> ('Android Programming: The Big Nerd Ranch',) (support=0.30, confidence=0.75, count=6)
('Java For Dummies',) -> ('ABeginner's Guide',) (support=0.45, confidence=0.69, count=9)
('Android Programming: The Big Nerd Ranch',) -> ('Java For Dummies',) (support=0.45, confidence=0.69, count=9)
('Java For Dummies',) -> ('Android Programming: The Big Nerd Ranch',) (support=0.45, confidence=0.69, count=9)
('Java For Dummies',) -> ('ABeginner's Guide', 'Java: The Complete Reference') (support=0.45, confidence=0.69, count=9)
('Android Programming: The Big Nerd Ranch', 'Java For Dummies') -> ('Java: The Complete Reference',) (support=0.30, confidence=0.67, count=6)
('Java: The Complete Reference',) -> ('Android Programming: The Big Nerd Ranch',) (support=0.30, confidence=0.60, count=6)
('Java: The Complete Reference',) -> ('Android Programming: The Big Nerd Ranch', 'Java For Dummies') (support=0.30, confidence=0.60, count=6)
('Java For Dummies', 'Java: The Complete Reference') -> ('Android Programming: The Big Nerd Ranch',) (support=0.30, confidence=0.60, count=6)

=== Brute Force Summary ===
Total frequent itemsets: 15
Total association rules: 20
Total execution time: 0.003000 seconds
```

Dataset: bestbuy.csv

```
Running Brute Force Algorithm ...

=== 1-itemsets ===
('Anti-Virus',): support=0.70, count=14
('Lab Top Case',): support=0.70, count=14
('Flash Drive',): support=0.65, count=13
('Lab Top',): support=0.60, count=12
('Microsoft Office',): support=0.55, count=11
('Speakers',): support=0.55, count=11
('Printer',): support=0.50, count=10
('Digital Camera',): support=0.45, count=9
('External Hard-Drive',): support=0.45, count=9
('Desk Top',): support=0.30, count=6

=== 2-itemsets ===
('Anti-Virus', 'Lab Top Case'): support=0.60, count=12
('Flash Drive', 'Microsoft Office'): support=0.55, count=11
```

.....

```
('Printer',) -> ('Anti-Virus', 'Lab Top Case') (support=0.30, confidence=0.60, count=6)
('Printer',) -> ('Anti-Virus', 'Microsoft Office') (support=0.30, confidence=0.60, count=6)
('Lab Top', 'Lab Top Case') -> ('Flash Drive',) (support=0.30, confidence=0.60, count=6)
('Printer',) -> ('Flash Drive', 'Lab Top Case') (support=0.30, confidence=0.60, count=6)
('Flash Drive', 'Printer') -> ('Lab Top Case',) (support=0.30, confidence=0.60, count=6)
('Anti-Virus', 'Flash Drive') -> ('External Hard-Drive', 'Lab Top Case') (support=0.30, confidence=0.60, count=6)
('Anti-Virus', 'Flash Drive') -> ('Lab Top', 'Lab Top Case') (support=0.30, confidence=0.60, count=6)
('Anti-Virus', 'Lab Top') -> ('Flash Drive', 'Lab Top Case') (support=0.30, confidence=0.60, count=6)
('Lab Top', 'Lab Top Case') -> ('Anti-Virus', 'Flash Drive') (support=0.30, confidence=0.60, count=6)
('Printer',) -> ('Anti-Virus', 'Flash Drive', 'Lab Top Case') (support=0.30, confidence=0.60, count=6)
('Anti-Virus', 'Flash Drive') -> ('Lab Top Case', 'Printer') (support=0.30, confidence=0.60, count=6)
('Flash Drive', 'Printer') -> ('Anti-Virus', 'Lab Top Case') (support=0.30, confidence=0.60, count=6)
('Printer',) -> ('Anti-Virus', 'Flash Drive', 'Microsoft Office') (support=0.30, confidence=0.60, count=6)
('Anti-Virus', 'Flash Drive') -> ('Microsoft Office', 'Printer') (support=0.30, confidence=0.60, count=6)
('Flash Drive', 'Printer') -> ('Anti-Virus', 'Microsoft Office') (support=0.30, confidence=0.60, count=6)
('Anti-Virus', 'Flash Drive') -> ('Microsoft Office', 'Speakers') (support=0.30, confidence=0.60, count=6)

=== Brute Force Summary ===
Total frequent itemsets: 64
Total association rules: 192
Total execution time: 0.017467 seconds
```

Dataset: generic.csv

```
Running Brute Force Algorithm ...

=== 1-itemsets ===
('A',): support=1.00, count=20
('E',): support=0.75, count=15
('C',): support=0.70, count=14
('B',): support=0.60, count=12
('D',): support=0.55, count=11
('F',): support=0.45, count=9

=== 2-itemsets ===
('A', 'E'): support=0.75, count=15
('A', 'C'): support=0.70, count=14
('A', 'B'): support=0.60, count=12
```

.....

```
('A',) -> ('C',) (support=0.70, confidence=0.70, count=14)
('B',) -> ('D',) (support=0.40, confidence=0.67, count=8)
('B',) -> ('A', 'D') (support=0.40, confidence=0.67, count=8)
('A', 'B') -> ('D',) (support=0.40, confidence=0.67, count=8)
('F',) -> ('C',) (support=0.30, confidence=0.67, count=6)
('F',) -> ('A', 'C') (support=0.30, confidence=0.67, count=6)
('A', 'F') -> ('C',) (support=0.30, confidence=0.67, count=6)
('B', 'C') -> ('D',) (support=0.30, confidence=0.67, count=6)
('B', 'C') -> ('A', 'D') (support=0.30, confidence=0.67, count=6)
('A', 'B', 'C') -> ('D',) (support=0.30, confidence=0.67, count=6)
('C',) -> ('B',) (support=0.45, confidence=0.64, count=9)
('C',) -> ('E',) (support=0.45, confidence=0.64, count=9)
('C',) -> ('A', 'B') (support=0.45, confidence=0.64, count=9)
('A', 'C') -> ('B',) (support=0.45, confidence=0.64, count=9)
('C',) -> ('A', 'E') (support=0.45, confidence=0.64, count=9)
('A', 'C') -> ('E',) (support=0.45, confidence=0.64, count=9)
('A',) -> ('B',) (support=0.60, confidence=0.60, count=12)
('E',) -> ('C',) (support=0.45, confidence=0.60, count=9)
('E',) -> ('A', 'C') (support=0.45, confidence=0.60, count=9)
('A', 'E') -> ('C',) (support=0.45, confidence=0.60, count=9)

=== Brute Force Summary ===
Total frequent itemsets: 29
Total association rules: 56
Total execution time: 0.003118 seconds
```

Dataset: kmart.csv

```
Running Brute Force Algorithm ...

=== 1-itemsets ===
('Kids Bedding',): support=0.60, count=12
('Bed Skirts',): support=0.55, count=11
('Shams',): support=0.55, count=11
('Decorative Pillows',): support=0.50, count=10
('Sheets',): support=0.50, count=10
('Quilts',): support=0.40, count=8
('Bedding Collections',): support=0.35, count=7
('Bedspreads',): support=0.35, count=7
('Embroidered Bedspread',): support=0.30, count=6

=== 2-itemsets ===
('Bed Skirts', 'Kids Bedding'): support=0.50, count=10
('Kids Bedding', 'Sheets'): support=0.50, count=10
```

.....

```
('Kids Bedding', 'Sheets') -> ('Shams',) (support=0.35, confidence=0.70, count=7)
('Sheets',) -> ('Bed Skirts', 'Bedspreads', 'Kids Bedding') (support=0.35, confidence=0.70, count=7)
('Bed Skirts', 'Kids Bedding') -> ('Bedspreads', 'Sheets') (support=0.35, confidence=0.70, count=7)
('Kids Bedding', 'Sheets') -> ('Bed Skirts', 'Bedspreads') (support=0.35, confidence=0.70, count=7)
('Sheets',) -> ('Bed Skirts', 'Kids Bedding', 'Shams') (support=0.35, confidence=0.70, count=7)
('Bed Skirts', 'Kids Bedding') -> ('Shams', 'Sheets') (support=0.35, confidence=0.70, count=7)
('Kids Bedding', 'Sheets') -> ('Bed Skirts', 'Shams') (support=0.35, confidence=0.70, count=7)
('Kids Bedding',) -> ('Bed Skirts', 'Shams') (support=0.40, confidence=0.67, count=8)
('Bed Skirts',) -> ('Bedspreads',) (support=0.35, confidence=0.64, count=7)
('Shams',) -> ('Sheets',) (support=0.35, confidence=0.64, count=7)
('Bed Skirts',) -> ('Bedspreads', 'Kids Bedding') (support=0.35, confidence=0.64, count=7)
('Bed Skirts',) -> ('Bedspreads', 'Sheets') (support=0.35, confidence=0.64, count=7)
('Bed Skirts',) -> ('Shams', 'Sheets') (support=0.35, confidence=0.64, count=7)
('Shams',) -> ('Bed Skirts', 'Sheets') (support=0.35, confidence=0.64, count=7)
('Shams',) -> ('Kids Bedding', 'Sheets') (support=0.35, confidence=0.64, count=7)
('Bed Skirts',) -> ('Bedspreads', 'Kids Bedding', 'Sheets') (support=0.35, confidence=0.64, count=7)
('Bed Skirts',) -> ('Kids Bedding', 'Shams', 'Sheets') (support=0.35, confidence=0.64, count=7)
('Shams',) -> ('Bed Skirts', 'Kids Bedding', 'Sheets') (support=0.35, confidence=0.64, count=7)
('Decorative Pillows',) -> ('Quilts',) (support=0.30, confidence=0.60, count=6)

=== Brute Force Summary ===
Total frequent itemsets: 29
Total association rules: 85
Total execution time: 0.006010 seconds
```

Dataset: nike.csv

```
Running Brute Force Algorithm ...

=== 1-itemsets ===
('Running Shoe',): support=0.70, count=14
('Socks',): support=0.65, count=13
('Sweatshirts',): support=0.65, count=13
('Rash Guard',): support=0.60, count=12
('Swimming Shirt',): support=0.55, count=11
('Modern Pants',): support=0.50, count=10
('Dry Fit V-Nick',): support=0.45, count=9
('Tech Pants',): support=0.45, count=9
('Hoodies',): support=0.40, count=8
('Soccer Shoe',): support=0.30, count=6

=== 2-itemsets ===
('Socks', 'Sweatshirts'): support=0.60, count=12
('Running Shoe', 'Socks'): support=0.55, count=11
('Running Shoe', 'Sweatshirts'): support=0.55, count=11
('Modern Pants', 'Sweatshirts'): support=0.50, count=10
('Rash Guard', 'Swimming Shirt'): support=0.50, count=10
('Dry Fit V-Nick', 'Rash Guard'): support=0.45, count=9
```

.....

```
('Modern Pants', 'Sweatshirts') -> ('Rash Guard',) (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Rash Guard', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Socks', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Sweatshirts', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants', 'Sweatshirts') -> ('Tech Pants',) (support=0.30, confidence=0.60, count=6)
('Rash Guard', 'Swimming Shirt') -> ('Running Shoe',) (support=0.30, confidence=0.60, count=6)
('Rash Guard', 'Swimming Shirt') -> ('Dry Fit V-Nick', 'Hoodies') (support=0.30, confidence=0.60, count=6)
('Rash Guard', 'Swimming Shirt') -> ('Hoodies', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Rash Guard', 'Socks', 'Sweatshirts') (support=0.30, confidence=0.60, count=6)
('Modern Pants', 'Sweatshirts') -> ('Rash Guard', 'Socks') (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Rash Guard', 'Socks', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Rash Guard', 'Sweatshirts', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants', 'Sweatshirts') -> ('Rash Guard', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Socks', 'Sweatshirts', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants', 'Sweatshirts') -> ('Socks', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Rash Guard', 'Swimming Shirt') -> ('Dry Fit V-Nick', 'Hoodies', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants',) -> ('Rash Guard', 'Socks', 'Sweatshirts', 'Tech Pants') (support=0.30, confidence=0.60, count=6)
('Modern Pants', 'Sweatshirts') -> ('Rash Guard', 'Socks', 'Tech Pants') (support=0.30, confidence=0.60, count=6)

=== Brute Force Summary ===
Total frequent itemsets: 73
Total association rules: 342
Total execution time: 0.038540 seconds
```

- **Timing Comparison**

Min Support = 0.3 and Min Confidence = 0.6		
Dataset	**Algorithm**	**Execution Time (seconds)**
amazon.csv	Brute Force	0.003559
	Apriori	0.012706
	FP-Growth	0.014396
bestbuy.csv	Brute Force	0.012496
	Apriori	0.020528
	FP-Growth	0.02036
generic.csv	Brute Force	0.001853
	Apriori	0.014269
	FP-Growth	0.022021
kmart.csv	Brute Force	0.010571
	Apriori	0.013117
	FP-Growth	0.018046
nike.csv	Brute Force	0.025915
	Apriori	0.024608
	FP-Growth	0.031611

4. APRIORI 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 $X \rightarrow Y$ and compute the confidence value: $\text{Confidence}(X \rightarrow Y) = \text{Support}(X \cup Y) / \text{Support}(X)$

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

```
Running Apriori Algorithm ...

=== 1-itemsets ===
('ABeginner's Guide',): support=0.55, count=11
('Android Programming: The Big Nerd Ranch',): support=0.65, count=13
('Beginning Programming with Java',): support=0.30, count=6
('Head First Java 2nd Edition',): support=0.40, count=8
('Java 8 Pocket Guide',): support=0.20, count=4
('Java For Dummies',): support=0.65, count=13
('Java: The Complete Reference',): support=0.50, count=10


=== 2-itemsets ===
('Android Programming: The Big Nerd Ranch', 'ABeginner's Guide'): support=0.30, count=6
('Java For Dummies', 'ABeginner's Guide'): support=0.45, count=9
('Java: The Complete Reference', 'ABeginner's Guide'): support=0.45, count=9
('Head First Java 2nd Edition', 'Android Programming: The Big Nerd Ranch'): support=0.30, count=6
```

.....

```
('Android Programming: The Big Nerd Ranch', 'Java For Dummies', 'ABeginner's Guide') -> ('Java: The Complete Reference',) (support=0.25, confidence=1.0
0, count=5)
('Android Programming: The Big Nerd Ranch', 'Java For Dummies', 'Java: The Complete Reference') -> ('ABeginner's Guide',) (support=0.25, confidence=0.8
3, count=5)
('Android Programming: The Big Nerd Ranch', 'ABeginner's Guide', 'Java: The Complete Reference') -> ('Java For Dummies',) (support=0.25, confidence=1.0
0, count=5)
('Java For Dummies', 'ABeginner's Guide', 'Java: The Complete Reference') -> ('Android Programming: The Big Nerd Ranch',) (support=0.25, confidence=0.5
6, count=5)
('Android Programming: The Big Nerd Ranch', 'Java For Dummies') -> ('Java: The Complete Reference', 'ABeginner's Guide') (support=0.25, confidence=0.56,
count=5)
('Android Programming: The Big Nerd Ranch', 'ABeginner's Guide') -> ('Java For Dummies', 'Java: The Complete Reference') (support=0.25, confidence=0.83,
count=5)
('Android Programming: The Big Nerd Ranch', 'Java: The Complete Reference') -> ('Java For Dummies', 'ABeginner's Guide') (support=0.25, confidence=0.83,
count=5)
('Java For Dummies', 'ABeginner's Guide') -> ('Android Programming: The Big Nerd Ranch', 'Java: The Complete Reference') (support=0.25, confidence=0.56,
count=5)
('Java For Dummies', 'Java: The Complete Reference') -> ('Android Programming: The Big Nerd Ranch', 'ABeginner's Guide') (support=0.25, confidence=0.50,
count=5)
('Java: The Complete Reference', 'ABeginner's Guide') -> ('Android Programming: The Big Nerd Ranch', 'Java For Dummies') (support=0.25, confidence=0.56,
count=5)
('Java: The Complete Reference',) -> ('Android Programming: The Big Nerd Ranch', 'Java For Dummies', 'ABeginner's Guide') (support=0.25, confidence=0.5
0, count=5)

=== Apriori Summary ===
Total frequent itemsets: 20
Total association rules: 41
Total execution time: 0.053052 seconds
```

Dataset: bestbuy.csv

```
Running Apriori Algorithm ...

=== 1-itemsets ===
('Anti-Virus',): support=0.70, count=14
('Desk Top',): support=0.30, count=6
('Digital Camera',): support=0.45, count=9
('External Hard-Drive',): support=0.45, count=9
('Flash Drive',): support=0.65, count=13
('Lab Top',): support=0.60, count=12
('Lab Top Case',): support=0.70, count=14
('Microsoft Office',): support=0.55, count=11
('Printer',): support=0.50, count=10
('Speakers',): support=0.55, count=11
```

.....

```
('Speakers', 'Flash Drive', 'Anti-Virus') -> ('Lab Top Case', 'Printer', 'Microsoft Office') (support=0.20, confidence=0.67, count=4
('Lab Top Case', 'Printer', 'Microsoft Office') -> ('Speakers', 'Flash Drive', 'Anti-Virus') (support=0.20, confidence=0.80, count=4
('Lab Top Case', 'Speakers', 'Printer') -> ('Flash Drive', 'Microsoft Office', 'Anti-Virus') (support=0.20, confidence=1.00, count=4
('Lab Top Case', 'Printer', 'Anti-Virus') -> ('Speakers', 'Flash Drive', 'Microsoft Office') (support=0.20, confidence=0.67, count=4
('Lab Top Case', 'Speakers', 'Microsoft Office') -> ('Printer', 'Flash Drive', 'Anti-Virus') (support=0.20, confidence=0.80, count=4
('Lab Top Case', 'Microsoft Office', 'Anti-Virus') -> ('Speakers', 'Printer', 'Flash Drive') (support=0.20, confidence=0.57, count=4
('Lab Top Case', 'Speakers', 'Anti-Virus') -> ('Printer', 'Flash Drive', 'Microsoft Office') (support=0.20, confidence=0.50, count=4
('Speakers', 'Printer', 'Microsoft Office') -> ('Lab Top Case', 'Flash Drive', 'Anti-Virus') (support=0.20, confidence=0.80, count=4
('Printer', 'Microsoft Office', 'Anti-Virus') -> ('Lab Top Case', 'Speakers', 'Flash Drive') (support=0.20, confidence=0.67, count=4
('Speakers', 'Printer', 'Anti-Virus') -> ('Lab Top Case', 'Flash Drive', 'Microsoft Office') (support=0.20, confidence=0.80, count=4
('Speakers', 'Microsoft Office', 'Anti-Virus') -> ('Lab Top Case', 'Printer', 'Flash Drive') (support=0.20, confidence=0.67, count=4
('Speakers', 'Flash Drive') -> ('Lab Top Case', 'Printer', 'Microsoft Office', 'Anti-Virus') (support=0.20, confidence=0.67, count=4
('Lab Top Case', 'Printer') -> ('Speakers', 'Flash Drive', 'Microsoft Office', 'Anti-Virus') (support=0.20, confidence=0.67, count=4
('Lab Top Case', 'Microsoft Office') -> ('Speakers', 'Printer', 'Flash Drive', 'Anti-Virus') (support=0.20, confidence=0.57, count=4
('Speakers', 'Printer') -> ('Lab Top Case', 'Flash Drive', 'Microsoft Office', 'Anti-Virus') (support=0.20, confidence=0.80, count=4
('Printer', 'Anti-Virus') -> ('Lab Top Case', 'Speakers', 'Flash Drive', 'Microsoft Office') (support=0.20, confidence=0.57, count=4
('Speakers', 'Microsoft Office') -> ('Lab Top Case', 'Printer', 'Flash Drive', 'Anti-Virus') (support=0.20, confidence=0.67, count=4
('Microsoft Office', 'Anti-Virus') -> ('Lab Top Case', 'Speakers', 'Printer', 'Flash Drive') (support=0.20, confidence=0.50, count=4

=== Apriori Summary ===
Total frequent itemsets: 175
Total association rules: 1231
Total execution time: 0.032813 seconds
```

Dataset: generic.csv

```
Running Apriori Algorithm ...

=== 1-itemsets ===
('A',): support=1.00, count=20
('B',): support=0.60, count=12
('C',): support=0.70, count=14
('D',): support=0.55, count=11
('E',): support=0.75, count=15
('F',): support=0.45, count=9


=== 2-itemsets ===
('A', 'B'): support=0.60, count=12
('A', 'C'): support=0.70, count=14
('A', 'D'): support=0.55, count=11
('E', 'A'): support=0.75, count=15
('F', 'A'): support=0.45, count=9
('C', 'B'): support=0.45, count=9
```

.....

```
('F', 'E', 'C') -> ('A',) (support=0.25, confidence=1.00, count=5)
('F', 'A', 'C') -> ('E',) (support=0.25, confidence=0.83, count=5)
('E', 'A', 'C') -> ('F',) (support=0.25, confidence=0.56, count=5)
('F', 'E') -> ('A', 'C') (support=0.25, confidence=0.62, count=5)
('F', 'A') -> ('E', 'C') (support=0.25, confidence=0.56, count=5)
('F', 'C') -> ('E', 'A') (support=0.25, confidence=0.83, count=5)
('E', 'C') -> ('F', 'A') (support=0.25, confidence=0.56, count=5)
('F',) -> ('E', 'A', 'C') (support=0.25, confidence=0.56, count=5)
('F', 'E', 'A') -> ('D',) (support=0.20, confidence=0.50, count=4)
('F', 'E', 'D') -> ('A',) (support=0.20, confidence=1.00, count=4)
('F', 'A', 'D') -> ('E',) (support=0.20, confidence=0.80, count=4)
('E', 'A', 'D') -> ('F',) (support=0.20, confidence=0.50, count=4)
('F', 'E') -> ('A', 'D') (support=0.20, confidence=0.50, count=4)
('F', 'D') -> ('E', 'A') (support=0.20, confidence=0.80, count=4)
('E', 'D') -> ('F', 'A') (support=0.20, confidence=0.50, count=4)

=== Apriori Summary ===
Total frequent itemsets: 47
Total association rules: 150
Total execution time: 0.018235 seconds
```

Dataset: kmart.csv

```
Running Apriori Algorithm ...

=== 1-itemsets ===
('Bed Skirts',): support=0.55, count=11
('Bedding Collections',): support=0.35, count=7
('Bedspreads',): support=0.35, count=7
('Decorative Pillows',): support=0.50, count=10
('Embroidered Bedspread',): support=0.30, count=6
('Kids Bedding',): support=0.60, count=12
('Quilts',): support=0.40, count=8
('Shams',): support=0.55, count=11
('Sheets',): support=0.50, count=10


=== 2-itemsets ===
('Bed Skirts', 'Bedding Collections'): support=0.25, count=5
('Bed Skirts', 'Bedspreads'): support=0.35, count=7
('Bed Skirts', 'Decorative Pillows'): support=0.20, count=4
```

.....

```
('Shams', 'Kids Bedding', 'Bedspreads') -> ('Bed Skirts', 'Sheets') (support=0.25, confidence=1.00, count=5)
('Shams', 'Bedspreads', 'Sheets') -> ('Bed Skirts', 'Kids Bedding') (support=0.25, confidence=1.00, count=5)
('Kids Bedding', 'Bedspreads', 'Sheets') -> ('Bed Skirts', 'Shams') (support=0.25, confidence=0.71, count=5)
('Shams', 'Kids Bedding', 'Sheets') -> ('Bed Skirts', 'Bedspreads') (support=0.25, confidence=0.71, count=5)
('Bed Skirts', 'Bedspreads') -> ('Shams', 'Kids Bedding', 'Sheets') (support=0.25, confidence=0.71, count=5)
('Bed Skirts', 'Shams') -> ('Kids Bedding', 'Bedspreads', 'Sheets') (support=0.25, confidence=0.56, count=5)
('Bed Skirts', 'Kids Bedding') -> ('Shams', 'Bedspreads', 'Sheets') (support=0.25, confidence=0.50, count=5)
('Bed Skirts', 'Sheets') -> ('Shams', 'Kids Bedding', 'Bedspreads') (support=0.25, confidence=0.56, count=5)
('Shams', 'Bedspreads') -> ('Bed Skirts', 'Kids Bedding', 'Sheets') (support=0.25, confidence=1.00, count=5)
('Kids Bedding', 'Bedspreads') -> ('Bed Skirts', 'Shams', 'Sheets') (support=0.25, confidence=0.71, count=5)
('Bedspreads', 'Sheets') -> ('Bed Skirts', 'Shams', 'Kids Bedding') (support=0.25, confidence=0.71, count=5)
('Shams', 'Kids Bedding') -> ('Bed Skirts', 'Bedspreads', 'Sheets') (support=0.25, confidence=0.56, count=5)
('Shams', 'Sheets') -> ('Bed Skirts', 'Kids Bedding', 'Bedspreads') (support=0.25, confidence=0.71, count=5)
('Kids Bedding', 'Sheets') -> ('Bed Skirts', 'Shams', 'Bedspreads') (support=0.25, confidence=0.50, count=5)
('Bedspreads',) -> ('Bed Skirts', 'Shams', 'Kids Bedding', 'Sheets') (support=0.25, confidence=0.71, count=5)
('Sheets',) -> ('Bed Skirts', 'Shams', 'Bedspreads', 'Kids Bedding') (support=0.25, confidence=0.50, count=5)

=== Apriori Summary ===
Total frequent itemsets: 57
Total association rules: 287
Total execution time: 0.013841 seconds
```

Dataset: nike.csv

```
Running Apriori Algorithm ...

=== 1-itemsets ===
('Dry Fit V-Nick',): support=0.45, count=9
('Hoodies',): support=0.40, count=8
('Modern Pants',): support=0.50, count=10
('Rash Guard',): support=0.60, count=12
('Running Shoe',): support=0.70, count=14
('Soccer Shoe',): support=0.30, count=6
('Socks',): support=0.65, count=13
('Sweatshirts',): support=0.65, count=13
('Swimming Shirt',): support=0.55, count=11
('Tech Pants',): support=0.45, count=9


=== 2-itemsets ===
('Dry Fit V-Nick', 'Hoodies'): support=0.35, count=7
('Dry Fit V-Nick', 'Modern Pants'): support=0.25, count=5
('Dry Fit V-Nick', 'Rash Guard'): support=0.45, count=9
```

.....

```
('Swimming Shirt', 'Modern Pants') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Rash Guard', 'Running Shoe', 'Tech Pants', 'Socks') (support=0.20, confiden
e=1.00, count=4)
('Running Shoe', 'Rash Guard') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Modern Pants', 'Tech Pants', 'Swimming Shirt', 'Socks') (support=0.20, confiden
e=0.57, count=4)
('Socks', 'Rash Guard') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Modern Pants', 'Running Shoe', 'Tech Pants', 'Swimming Shirt') (support=0.20, confiden
e=0.67, count=4)
('Running Shoe', 'Tech Pants') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Modern Pants', 'Rash Guard', 'Swimming Shirt', 'Socks') (support=0.20, confiden
e=0.67, count=4)
('Swimming Shirt', 'Running Shoe') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Modern Pants', 'Rash Guard', 'Tech Pants', 'Socks') (support=0.20, confiden
e=0.67, count=4)
('Swimming Shirt', 'Tech Pants') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Modern Pants', 'Rash Guard', 'Running Shoe', 'Socks') (support=0.20, confiden
e=0.57, count=4)
('Socks', 'Tech Pants') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Modern Pants', 'Rash Guard', 'Running Shoe', 'Swimming Shirt') (support=0.20, confiden
e=0.67, count=4)
('Swimming Shirt', 'Socks') -> ('Dry Fit V-Nick', 'Sweatshirts', 'Modern Pants', 'Rash Guard', 'Running Shoe', 'Tech Pants') (support=0.20, confiden
e=0.80, count=4)

=== Apriori Summary ===
Total frequent itemsets: 523
Total association rules: 10736
Total execution time: 0.700776 seconds
```

- **Timing Comparison:**

Min Support = 0.2 and Min Confidence = 0.5		
Dataset	**Algorithm**	**Execution Time (seconds)**
amazon.csv	Brute Force	0.004671
	Apriori	0.016918
	FP-Growth	0.019396
bestbuy.csv	Brute Force	0.054877
	Apriori	0.038188
	FP-Growth	0.047809
generic.csv	Brute Force	0.004004
	Apriori	0.016991
	FP-Growth	0.022299
kmart.csv	Brute Force	0.010582
	Apriori	0.028439
	FP-Growth	0.024722
nike.csv	Brute Force	0.215427
	Apriori	0.756204
	FP-Growth	0.088008

5. 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 $X \rightarrow Y$.
Calculate confidence for each rule using: $Confidence(X \rightarrow Y) = Support(X \cup Y) / Support(X)$.

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

```
Running FP-Growth Algorithm ...

=== 1-itemsets ===
('Java For Dummies',): support=0.65, count=13
('Android Programming: The Big Nerd Ranch',): support=0.65, count=13
('ABeginner's Guide',): support=0.55, count=11
('Java: The Complete Reference',): support=0.50, count=10
('Head First Java 2nd Edition',): support=0.40, count=8


=== 2-itemsets ===
('Android Programming: The Big Nerd Ranch', 'Java For Dummies'): support=0.45, count=9
('Java For Dummies', 'ABeginner's Guide'): support=0.45, count=9
('Java For Dummies', 'Java: The Complete Reference'): support=0.50, count=10
('Java: The Complete Reference', 'ABeginner's Guide'): support=0.45, count=9


=== 3-itemsets ===
('Java For Dummies', 'Java: The Complete Reference', 'ABeginner's Guide'): support=0.45, count=9


=== Association Rules ===
('Android Programming: The Big Nerd Ranch',) -> ('Java For Dummies',) (support=0.45, confidence=0.69, count=9)
```

.....

```
('ABeginner's Guide',) -> ('Java For Dummies',) (support=0.45, confidence=0.82, count=9)
('Java For Dummies',) -> ('Java: The Complete Reference',) (support=0.50, confidence=0.77, count=10)
('Java: The Complete Reference',) -> ('Java For Dummies',) (support=0.50, confidence=1.00, count=10)
('Java: The Complete Reference',) -> ('ABeginner's Guide',) (support=0.45, confidence=0.90, count=9)
('ABeginner's Guide',) -> ('Java: The Complete Reference',) (support=0.45, confidence=0.82, count=9)
('Java For Dummies', 'Java: The Complete Reference') -> ('ABeginner's Guide',) (support=0.45, confidence=0.90, count=9)
('Java For Dummies', 'ABeginner's Guide') -> ('Java: The Complete Reference',) (support=0.45, confidence=1.00, count=9)
('Java: The Complete Reference', 'ABeginner's Guide') -> ('Java For Dummies',) (support=0.45, confidence=1.00, count=9)
('Java For Dummies',) -> ('Java: The Complete Reference', 'ABeginner's Guide') (support=0.45, confidence=0.69, count=9)
('Java: The Complete Reference',) -> ('Java For Dummies', 'ABeginner's Guide') (support=0.45, confidence=0.90, count=9)
('ABeginner's Guide',) -> ('Java For Dummies', 'Java: The Complete Reference') (support=0.45, confidence=0.82, count=9)

=== FP-Growth Summary ===
Total frequent itemsets: 10
Total association rules: 14
Total execution time: 0.011691 seconds
```

Dataset: bestbuy.csv

```
Running FP-Growth Algorithm ...

=== 1-itemsets ===
('Anti-Virus',): support=0.70, count=14
('Flash Drive',): support=0.65, count=13
('Speakers',): support=0.55, count=11
('Microsoft Office',): support=0.55, count=11
('Printer',): support=0.50, count=10
('Lab Top Case',): support=0.70, count=14
('Lab Top',): support=0.60, count=12
('External Hard-Drive',): support=0.45, count=9
('Digital Camera',): support=0.45, count=9


=== 2-itemsets ===
('Lab Top Case', 'Anti-Virus'): support=0.60, count=12
('Flash Drive', 'Anti-Virus'): support=0.50, count=10
('Lab Top Case', 'Flash Drive'): support=0.45, count=9
('Speakers', 'Anti-Virus'): support=0.45, count=9
```

.....

```
('Anti-Virus',) -> ('Lab Top Case', 'Lab Top') (support=0.45, confidence=0.64, count=9)
('External Hard-Drive',) -> ('Anti-Virus',) (support=0.45, confidence=1.00, count=9)
('Anti-Virus',) -> ('External Hard-Drive',) (support=0.45, confidence=0.64, count=9)
('External Hard-Drive',) -> ('Lab Top Case',) (support=0.40, confidence=0.89, count=8)
('Lab Top Case',) -> ('External Hard-Drive',) (support=0.40, confidence=0.57, count=8)
('External Hard-Drive', 'Lab Top Case') -> ('Anti-Virus',) (support=0.40, confidence=1.00, count=8)
('External Hard-Drive', 'Anti-Virus') -> ('Lab Top Case',) (support=0.40, confidence=0.89, count=8)
('Lab Top Case', 'Anti-Virus') -> ('External Hard-Drive',) (support=0.40, confidence=0.67, count=8)
('External Hard-Drive',) -> ('Lab Top Case', 'Anti-Virus') (support=0.40, confidence=0.89, count=8)
('Lab Top Case',) -> ('External Hard-Drive', 'Anti-Virus') (support=0.40, confidence=0.57, count=8)
('Anti-Virus',) -> ('External Hard-Drive', 'Lab Top Case') (support=0.40, confidence=0.57, count=8)

=== FP-Growth Summary ===
Total frequent itemsets: 28
Total association rules: 62
Total execution time: 0.023635 seconds
```

Dataset: generic.csv

```
Running FP-Growth Algorithm ...

=== 1-itemsets ===
('A',): support=1.00, count=20
('C',): support=0.70, count=14
('B',): support=0.60, count=12
('D',): support=0.55, count=11
('E',): support=0.75, count=15
('F',): support=0.45, count=9


=== 2-itemsets ===
('A', 'C'): support=0.70, count=14
('E', 'C'): support=0.45, count=9
('A', 'B'): support=0.60, count=12
```

.....

```
('E', 'D') -> ('A',) (support=0.40, confidence=1.00, count=8)
('A', 'D') -> ('E',) (support=0.40, confidence=0.73, count=8)
('E',) -> ('A', 'D') (support=0.40, confidence=0.53, count=8)
('D',) -> ('E', 'A') (support=0.40, confidence=0.73, count=8)
('E',) -> ('A',) (support=0.75, confidence=1.00, count=15)
('A',) -> ('E',) (support=0.75, confidence=0.75, count=15)
('F',) -> ('A',) (support=0.45, confidence=1.00, count=9)
('F',) -> ('E',) (support=0.40, confidence=0.89, count=8)
('E',) -> ('F',) (support=0.40, confidence=0.53, count=8)
('F', 'E') -> ('A',) (support=0.40, confidence=1.00, count=8)
('F', 'A') -> ('E',) (support=0.40, confidence=0.89, count=8)
('E', 'A') -> ('F',) (support=0.40, confidence=0.53, count=8)
('F',) -> ('E', 'A') (support=0.40, confidence=0.89, count=8)
('E',) -> ('F', 'A') (support=0.40, confidence=0.53, count=8)

=== FP-Growth Summary ===
Total frequent itemsets: 23
Total association rules: 51
Total execution time: 0.016387 seconds
```

Dataset: kmart.csv

```
Running FP-Growth Algorithm ...

=== 1-itemsets ===
('Decorative Pillows',): support=0.50, count=10
('Quilts',): support=0.40, count=8
('Kids Bedding',): support=0.60, count=12
('Shams',): support=0.55, count=11
('Bed Skirts',): support=0.55, count=11
('Sheets',): support=0.50, count=10


=== 2-itemsets ===
('Shams', 'Kids Bedding'): support=0.45, count=9
('Bed Skirts', 'Kids Bedding'): support=0.50, count=10
('Bed Skirts', 'Shams'): support=0.45, count=9
```

.....

```
('Kids Bedding',) -> ('Bed Skirts',) (support=0.50, confidence=0.83, count=10)
('Bed Skirts',) -> ('Shams',) (support=0.45, confidence=0.82, count=9)
('Shams',) -> ('Bed Skirts',) (support=0.45, confidence=0.82, count=9)
('Bed Skirts', 'Shams') -> ('Kids Bedding',) (support=0.40, confidence=0.89, count=8)
('Bed Skirts', 'Kids Bedding') -> ('Shams',) (support=0.40, confidence=0.80, count=8)
('Shams', 'Kids Bedding') -> ('Bed Skirts',) (support=0.40, confidence=0.89, count=8)
('Bed Skirts',) -> ('Shams', 'Kids Bedding') (support=0.40, confidence=0.73, count=8)
('Shams',) -> ('Bed Skirts', 'Kids Bedding') (support=0.40, confidence=0.73, count=8)
('Kids Bedding',) -> ('Bed Skirts', 'Shams') (support=0.40, confidence=0.67, count=8)
('Kids Bedding',) -> ('Sheets',) (support=0.50, confidence=0.83, count=10)
('Sheets',) -> ('Kids Bedding',) (support=0.50, confidence=1.00, count=10)
('Bed Skirts',) -> ('Sheets',) (support=0.45, confidence=0.82, count=9)
('Sheets',) -> ('Bed Skirts',) (support=0.45, confidence=0.90, count=9)
('Bed Skirts', 'Kids Bedding') -> ('Sheets',) (support=0.45, confidence=0.90, count=9)
('Bed Skirts', 'Sheets') -> ('Kids Bedding',) (support=0.45, confidence=1.00, count=9)
('Kids Bedding', 'Sheets') -> ('Bed Skirts',) (support=0.45, confidence=0.90, count=9)
('Bed Skirts',) -> ('Kids Bedding', 'Sheets') (support=0.45, confidence=0.82, count=9)
('Kids Bedding',) -> ('Bed Skirts', 'Sheets') (support=0.45, confidence=0.75, count=9)
('Sheets',) -> ('Bed Skirts', 'Kids Bedding') (support=0.45, confidence=0.90, count=9)

=== FP-Growth Summary ===
Total frequent itemsets: 13
Total association rules: 22
Total execution time: 0.015434 seconds
```

Dataset: nike.csv

```
Running FP-Growth Algorithm ...

=== 1-itemsets ===
('Running Shoe',): support=0.70, count=14
('Sweatshirts',): support=0.65, count=13
('Socks',): support=0.65, count=13
('Modern Pants',): support=0.50, count=10
('Rash Guard',): support=0.60, count=12
('Tech Pants',): support=0.45, count=9
('Hoodies',): support=0.40, count=8
('Swimming Shirt',): support=0.55, count=11
('Dry Fit V-Nick',): support=0.45, count=9


=== 2-itemsets ===
('Sweatshirts', 'Running Shoe'): support=0.55, count=11
('Sweatshirts', 'Socks'): support=0.60, count=12
('Running Shoe', 'Socks'): support=0.55, count=11
('Sweatshirts', 'Modern Pants'): support=0.50, count=10
('Modern Pants', 'Socks'): support=0.45, count=9
('Modern Pants', 'Running Shoe'): support=0.45, count=9
```

.....

```
('Dry Fit V-Nick',) -> ('Rash Guard',) (support=0.45, confidence=1.00, count=9)
('Rash Guard',) -> ('Dry Fit V-Nick',) (support=0.45, confidence=0.75, count=9)
('Dry Fit V-Nick',) -> ('Tech Pants',) (support=0.40, confidence=0.89, count=8)
('Tech Pants',) -> ('Dry Fit V-Nick',) (support=0.40, confidence=0.89, count=8)
('Dry Fit V-Nick',) -> ('Swimming Shirt',) (support=0.40, confidence=0.89, count=8)
('Swimming Shirt',) -> ('Dry Fit V-Nick',) (support=0.40, confidence=0.73, count=8)
('Dry Fit V-Nick', 'Rash Guard') -> ('Tech Pants',) (support=0.40, confidence=0.89, count=8)
('Dry Fit V-Nick', 'Tech Pants') -> ('Rash Guard',) (support=0.40, confidence=1.00, count=8)
('Rash Guard', 'Tech Pants') -> ('Dry Fit V-Nick',) (support=0.40, confidence=0.89, count=8)
('Dry Fit V-Nick',) -> ('Rash Guard', 'Tech Pants') (support=0.40, confidence=0.89, count=8)
('Rash Guard',) -> ('Dry Fit V-Nick', 'Tech Pants') (support=0.40, confidence=0.67, count=8)
('Tech Pants',) -> ('Dry Fit V-Nick', 'Rash Guard') (support=0.40, confidence=0.89, count=8)
('Dry Fit V-Nick', 'Rash Guard') -> ('Swimming Shirt',) (support=0.40, confidence=0.89, count=8)
('Dry Fit V-Nick', 'Swimming Shirt') -> ('Rash Guard',) (support=0.40, confidence=1.00, count=8)
('Swimming Shirt', 'Rash Guard') -> ('Dry Fit V-Nick',) (support=0.40, confidence=0.80, count=8)
('Dry Fit V-Nick',) -> ('Swimming Shirt', 'Rash Guard') (support=0.40, confidence=0.89, count=8)
('Rash Guard',) -> ('Dry Fit V-Nick', 'Swimming Shirt') (support=0.40, confidence=0.67, count=8)
('Swimming Shirt',) -> ('Dry Fit V-Nick', 'Rash Guard') (support=0.40, confidence=0.73, count=8)

=== FP-Growth Summary ===
Total frequent itemsets: 30
Total association rules: 82
Total execution time: 0.020253 seconds
```

- **Timing Comparison**

Min Support = 0.4 and Min Confidence = 0.5		
Dataset	**Algorithm**	**Execution Time (seconds)**
amazon.csv	Brute Force	0.004159
	Apriori	0.014101
	FP-Growth	0.013193
bestbuy.csv	Brute Force	0.006777
	Apriori	0.018026
	FP-Growth	0.048084
generic.csv	Brute Force	0.002983
	Apriori	0.012884
	FP-Growth	0.018663
kmart.csv	Brute Force	0.004524
	Apriori	0.016191
	FP-Growth	0.013542
nike.csv	Brute Force	0.019749
	Apriori	0.022862
	FP-Growth	0.021159

6. ENVIRONMENT AND INSTALLATION:

➢ Recommended Versions & Prerequisites

- **Operating System:** Windows / macOS / Linux
- **Python:** Python 3.8 or Higher
- **Shell:** PowerShell / bash / cmd
- **Tools Required**: Jupyter Notebook

➢ Create a Virtual Environment

- It's best practice to create a virtual environment to isolate dependencies for this project.
- Windows: Run **py -3 -m venv .venv** and then **.\.venv\Scripts\activate**
- macOS/Linux: Run **python3 -m venv .venv** and then **source .venv/bin/activate**

➢ Install Requirements

- Open a terminal or command prompt and install the library by running: **pip install pandas mlxtend**
- You can install all the necessary dependencies using the provided requirements.txt file using the command: pip install -r requirements.txt.

7. HOW TO RUN THE CODE:

a) 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

b) Run through Jupyter Notebook

- jupyter notebook: notebooks/project_demo.ipynb

8. 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.

9. GITHUB REPOSITORY STRUCTURE AND LINK:

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

```
dobariya_yesha_midtermproject/
│
├── data/
│   ├── amazon.csv
│   ├── bestbuy.csv
│   ├── generic.csv
│   ├── kmart.csv
│   └── nike.csv
│
├── notebooks/
│   └── project_demo.ipynb
│
├── src/
│   ├── brute_force.py
│   ├── apriori_runner.py
│   └── fpgrowth_runner.py
│
├── report/
│   └── dobariya_yesha_report.pdf
│
├── screenshots/
│   ├── run1.png
│   ├── run2.png
│   ├── run3.png
│   ├── run4.png
│   ├── run5.png
│   ├── run6.png
│   ├── run7.png
│   ├── run8.png
│   ├── run9.png
│   └── timings.png
│
├── requirements.txt
└── README.md
```