Data Mining Midterm Project Report

Name – Aniket Khalate

UCID – ak3274

Email – ak3274@njit.edu

Course - CS634 – Data Mining

Profressor - Dr. Yasser Abduallah

Topic - A Comparative Analysis of Brute Force, Apriori, and FP-Growth Algorithms

Introduction:

This project outlines the implementation of three algorithms: Brute Force, Apriori, and FP-Growth for extracting frequent itemsets and generating association rules from transactional datasets. The objective is to evaluate the performance and efficiency of these algorithms in identifying frequent itemsets and rules, while analyzing their execution times across different support and confidence thresholds.

Key Concepts:

- ***Association Rules***: By analyzing the groups, "association rules" are created to determine which products are commonly purchased at the same time. These rules are key for improving sales tactics, such as making product recommendations.

- ***Frequent Items***: The main purpose of the algorithms is to discover "frequent itemsets," which are simply groups of items that are often bought together. This helps in understanding customer buying habits.

- ***Support***: This is a measure of how frequently an itemset (a collection of one or more items) appears in the dataset. It's calculated as:
  Support(Itemset) = (Number of transactions containing the itemset) / (Total number of

transactions)
A high support value means the item combination is common.

- ***Confidence***: This measure indicates the likelihood that an item is purchased, given that another item has been purchased. For a rule "If A, then B" (A -> B), it's calculated as:
Confidence(A -> B) = (Number of transactions containing both A and B) / (Number of transactions containing A)
A high confidence value suggests a strong relationship.

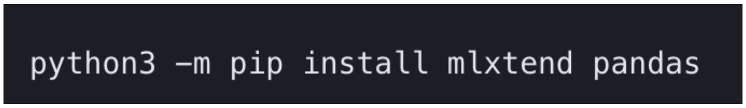
Dataset:

The dataset used consists of five separate datasets containing transaction records from a variety of stores, such as Amazon, Nike, and Best Buy. These datasets, which are saved in CSV files, list items typically sold in supermarkets. The data has been carefully organized to ensure it produces consistent and predictable results.
Note: This is the same dataset that professor has provided in the canvas.

Step by step guide to run and understand the .py file:

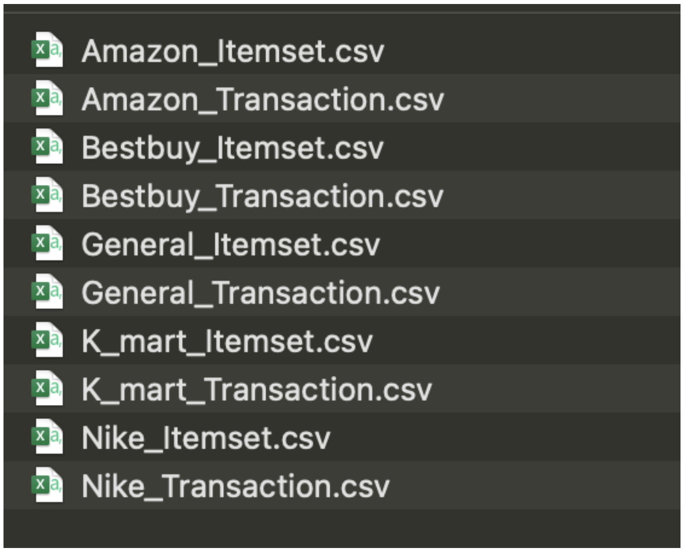
1. **Prerequisite**
Install Python - If you don't already have Python, get the newest version from python.org. It is important to select the "Add Python to PATH" option during the installation process.

2. **Install Required Libraries**
Run the below line of code to install and setup required packages. You will need *mlxtend* to run Apriori and FP-Growth and *pandas* for dataframes.



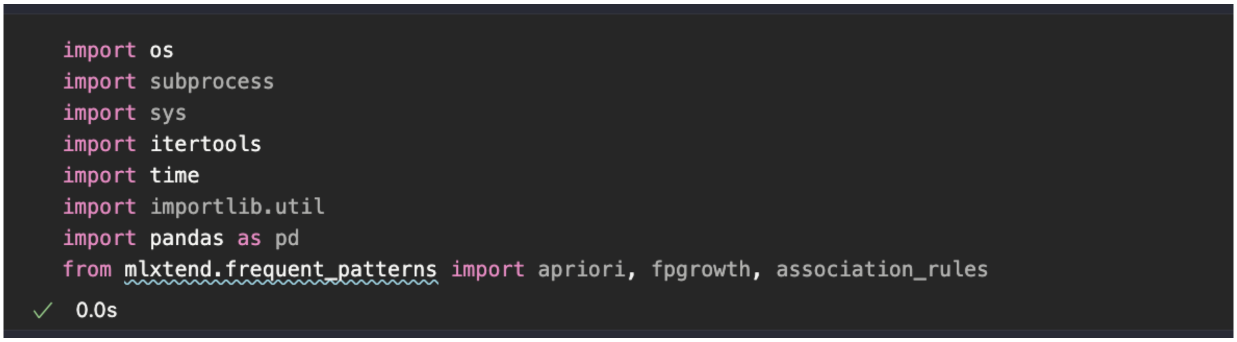
```
python3 -m pip install mlxtend pandas
```

3. **Setup Project Folder**
Make sure you have the following CSV files for each store in a single directory and with same naming convention as the given below:



```
Amazon_Itemset.csv
Amazon_Transaction.csv
Bestbuy_Itemset.csv
Bestbuy_Transaction.csv
General_Itemset.csv
General_Transaction.csv
K_mart_Itemset.csv
K_mart_Transaction.csv
Nike_Itemset.csv
Nike_Transaction.csv
```

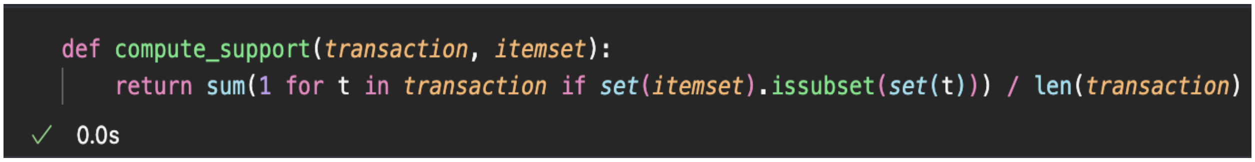
4. **Importing necessary libraries**
   Once you have installed the ‘pandas’ and ‘mlxtend’ packages then import them along with other libraries which are in-built within python.



```python
import os
import subprocess
import sys
import itertools
import time
import importlib.util
import pandas as pd
from mlxtend.frequent_patterns import apriori, fpgrowth, association_rules
```

✓ 0.0s

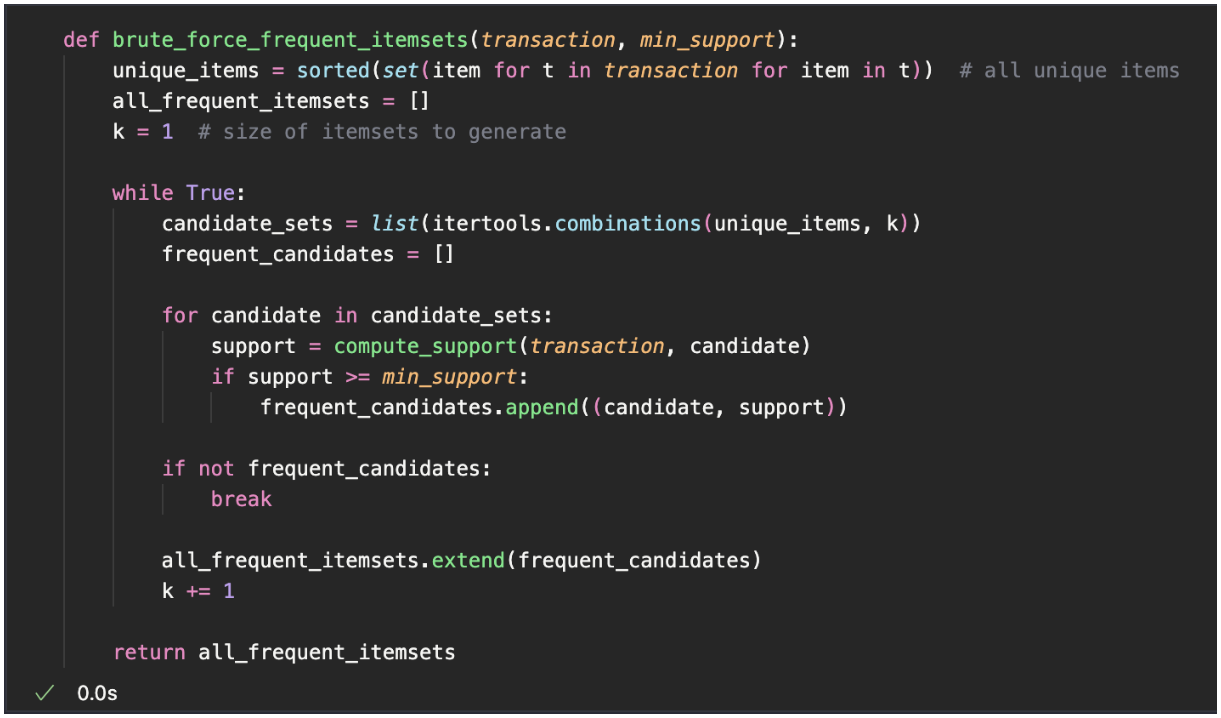
5. **Different Functions and their purpose:**
   - **compute_support**: This function calculates the support of a given itemset which is nothing but the proportion of transactions that contain all the items in the itemset.



```python
def compute_support(transaction, itemset):
    return sum(1 for t in transaction if set(itemset).issubset(set(t))) / len(transaction)
```

✓ 0.0s

- **brute_force_frequent_itemsets**:

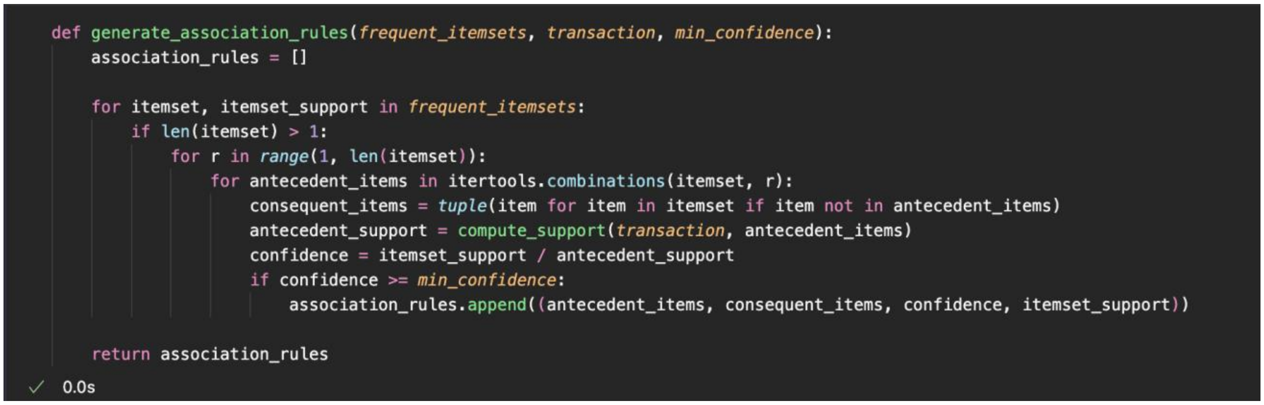


```python
def brute_force_frequent_itemsets(transaction, min_support):
    unique_items = sorted(set(item for t in transaction for item in t))  # all unique items
    all_frequent_itemsets = []
    k = 1  # size of itemsets to generate

    while True:
        candidate_sets = list(itertools.combinations(unique_items, k))
        frequent_candidates = []

        for candidate in candidate_sets:
            support = compute_support(transaction, candidate)
            if support >= min_support:
                frequent_candidates.append((candidate, support))

        if not frequent_candidates:
            break

        all_frequent_itemsets.extend(frequent_candidates)
        k += 1

    return all_frequent_itemsets
```

This function searches for frequent itemsets. It operates by:

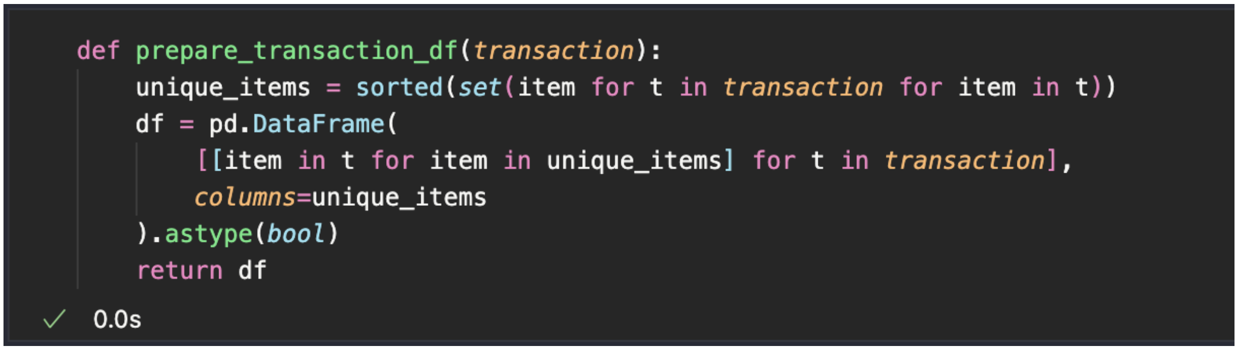
  - Identifying every unique item.
  - Creating every possible item combination, starting from single items.
  - Calculating the support for each combination.
  - Saving any combination that meets the minimum support requirement.
  - Repeating the process for larger combinations until no more frequent sets are found.
  - This method is easy to grasp but becomes very inefficient with larger datasets.

- **generate_association_rules**: This function derives association rules from the frequent itemsets. For each itemset, it splits items into possible antecedent consequent pairs, calculates the confidence of each rule, and keeps the ones that meet the minimum confidence threshold.



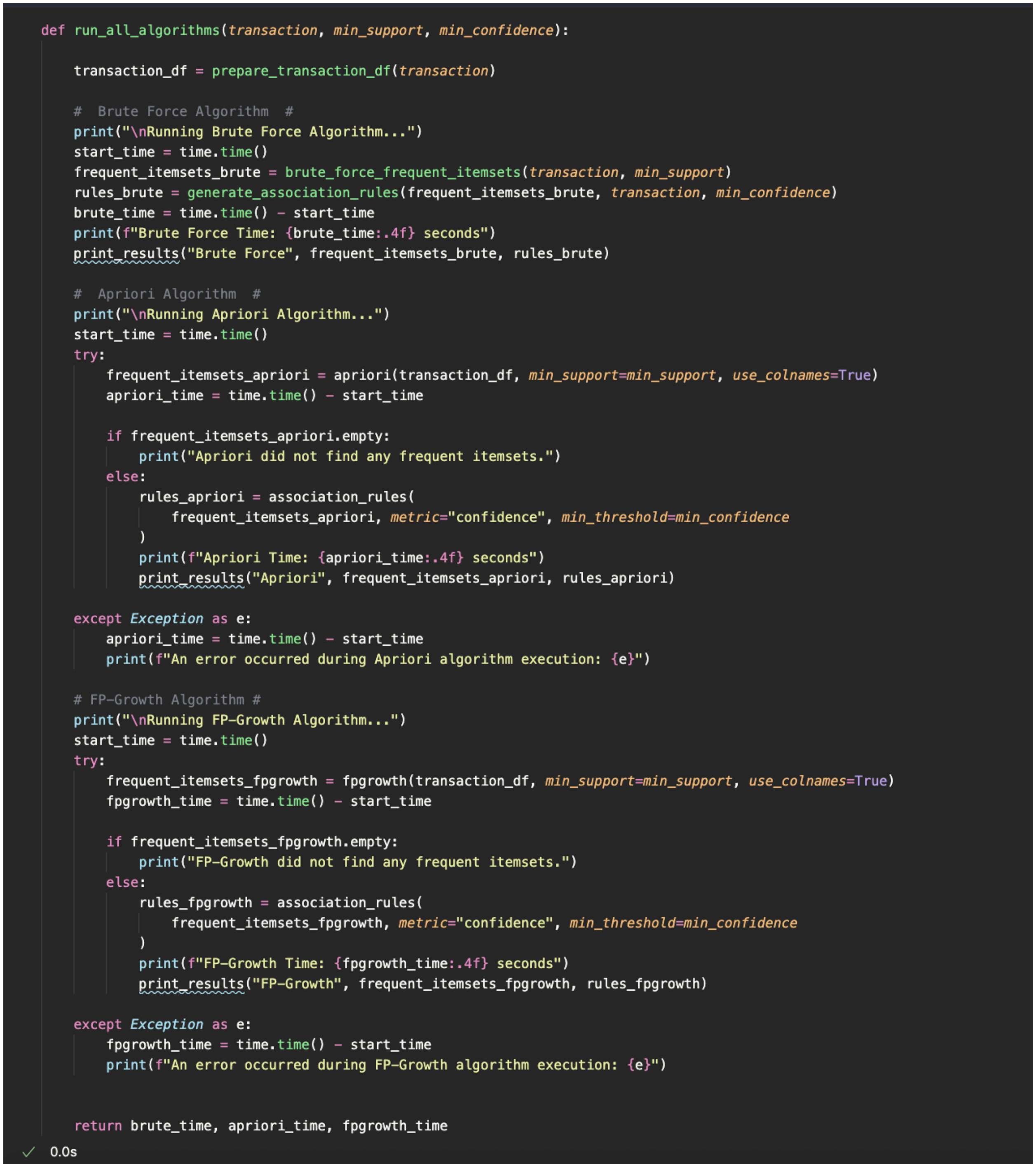
```python
def generate_association_rules(frequent_itemsets, transaction, min_confidence):
    association_rules = []

    for itemset, itemset_support in frequent_itemsets:
        if len(itemset) > 1:
            for r in range(1, len(itemset)):
                for antecedent_items in itertools.combinations(itemset, r):
                    consequent_items = tuple(item for item in itemset if item not in antecedent_items)
                    antecedent_support = compute_support(transaction, antecedent_items)
                    confidence = itemset_support / antecedent_support
                    if confidence >= min_confidence:
                        association_rules.append((antecedent_items, consequent_items, confidence, itemset_support))

    return association_rules
```

- **prepare_transaction_df**: This function converts the list of transactions into a boolean DataFrame. A cell is True if the item is present in that transaction, and False otherwise.



```python
def prepare_transaction_df(transaction):
    unique_items = sorted(set(item for t in transaction for item in t))
    df = pd.DataFrame(
        [[item in t for item in unique_items] for t in transaction],
        columns=unique_items
    ).astype(bool)
    return df
```

- **run_all_algorithms**:



```
def run_all_algorithms(transaction, min_support, min_confidence):

    transaction_df = prepare_transaction_df(transaction)

    #  Brute Force Algorithm  #
    print("\nRunning Brute Force Algorithm...")
    start_time = time.time()
    frequent_itemsets_brute = brute_force_frequent_itemsets(transaction, min_support)
    rules_brute = generate_association_rules(frequent_itemsets_brute, transaction, min_confidence)
    brute_time = time.time() - start_time
    print(f"Brute Force Time: {brute_time:.4f} seconds")
    print_results("Brute Force", frequent_itemsets_brute, rules_brute)

    #  Apriori Algorithm  #
    print("\nRunning Apriori Algorithm...")
    start_time = time.time()
    try:
        frequent_itemsets_apriori = apriori(transaction_df, min_support=min_support, use_colnames=True)
        apriori_time = time.time() - start_time

        if frequent_itemsets_apriori.empty:
            print("Apriori did not find any frequent itemsets.")
        else:
            rules_apriori = association_rules(
                frequent_itemsets_apriori, metric="confidence", min_threshold=min_confidence
            )
            print(f"Apriori Time: {apriori_time:.4f} seconds")
            print_results("Apriori", frequent_itemsets_apriori, rules_apriori)

    except Exception as e:
        apriori_time = time.time() - start_time
        print(f"An error occurred during Apriori algorithm execution: {e}")

    # FP-Growth Algorithm #
    print("\nRunning FP-Growth Algorithm...")
    start_time = time.time()
    try:
        frequent_itemsets_fpgrowth = fpgrowth(transaction_df, min_support=min_support, use_colnames=True)
        fpgrowth_time = time.time() - start_time

        if frequent_itemsets_fpgrowth.empty:
            print("FP-Growth did not find any frequent itemsets.")
        else:
            rules_fpgrowth = association_rules(
                frequent_itemsets_fpgrowth, metric="confidence", min_threshold=min_confidence
            )
            print(f"FP-Growth Time: {fpgrowth_time:.4f} seconds")
            print_results("FP-Growth", frequent_itemsets_fpgrowth, rules_fpgrowth)

    except Exception as e:
        fpgrowth_time = time.time() - start_time
        print(f"An error occurred during FP-Growth algorithm execution: {e}")


    return brute_time, apriori_time, fpgrowth_time
```

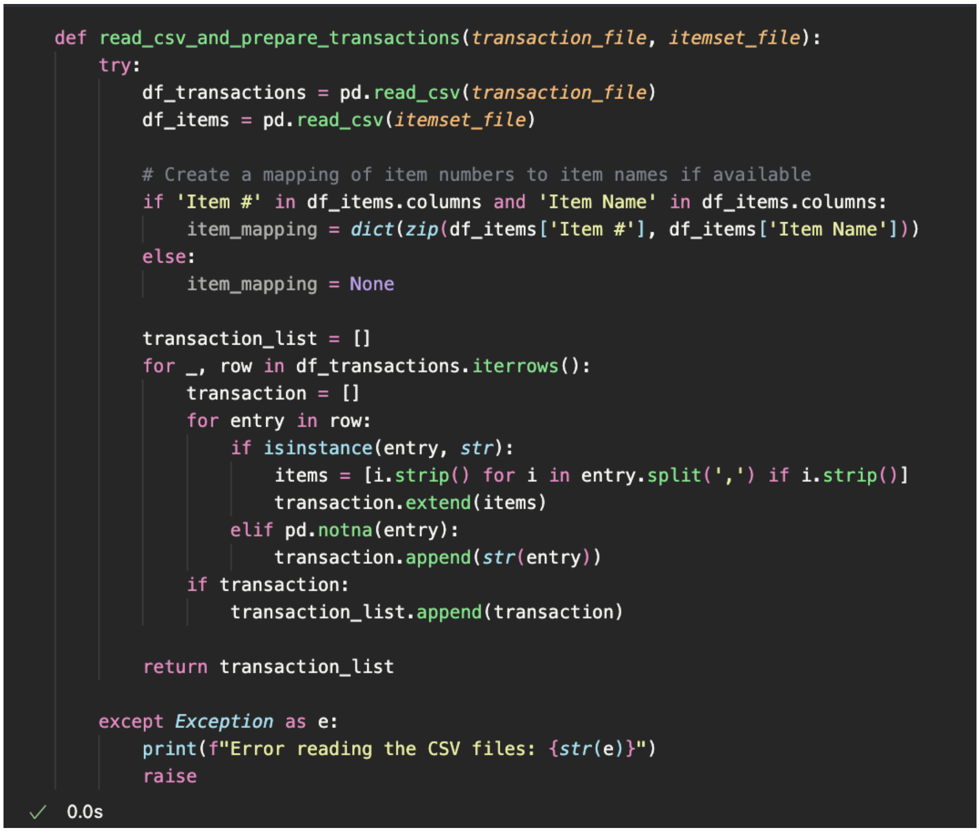
✓ 0.0s

This is the main analysis engine. It takes the dataset and user-defined thresholds for support and confidence, then performs the following steps for all three algorithms:

- Records the start time.
- Executes the algorithm to identify frequent itemsets.
- Creates association rules from these itemsets.
- Calculates the total execution time.

  - Displays the results in a clear format.
  - It leverages the optimized Apriori and FP-Growth algorithms provided by the mlxtend library.

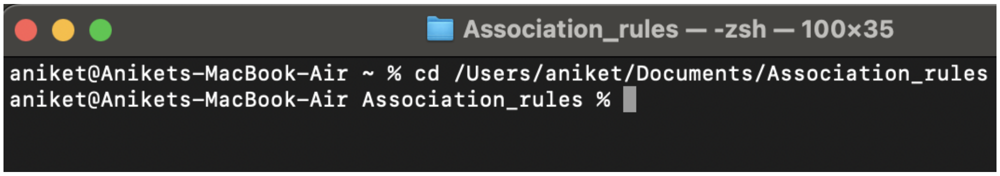
- **read_csv_and_prepare_transactions**: This function reads transaction and item data from CSV files, optionally maps item numbers to item names, and processes each row into a clean list of transactions that can be used for frequent itemset mining.



```python
def read_csv_and_prepare_transactions(transaction_file, itemset_file):
    try:
        df_transactions = pd.read_csv(transaction_file)
        df_items = pd.read_csv(itemset_file)

        # Create a mapping of item numbers to item names if available
        if 'Item #' in df_items.columns and 'Item Name' in df_items.columns:
            item_mapping = dict(zip(df_items['Item #'], df_items['Item Name']))
        else:
            item_mapping = None

        transaction_list = []
        for _, row in df_transactions.iterrows():
            transaction = []
            for entry in row:
                if isinstance(entry, str):
                    items = [i.strip() for i in entry.split(',') if i.strip()]
                    transaction.extend(items)
                elif pd.notna(entry):
                    transaction.append(str(entry))
            if transaction:
                transaction_list.append(transaction)

        return transaction_list

    except Exception as e:
        print(f"Error reading the CSV files: {str(e)}")
        raise
```

✓ 0.0s

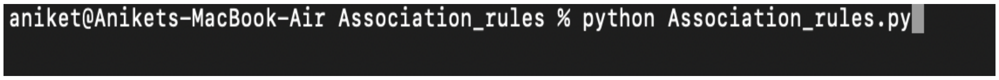
6. **Running the Program**:
   Once your setup is complete, you can start the analysis.
   a. Make sure that the python script is named as – “association_rules.py”
   b. Save the script and all the CSV files within a single folder named ‘Association_rules’
   c. Launch a terminal or command prompt.
   d. Change your directory to the folder where the python script and all the CSV files are stored using below command:



```
aniket@Anikets-MacBook-Air ~ % cd /Users/aniket/Documents/Association_rules
aniket@Anikets-MacBook-Air Association_rules % 
```

Note: Make sure that the python script and the CSV files are all in the same folder.

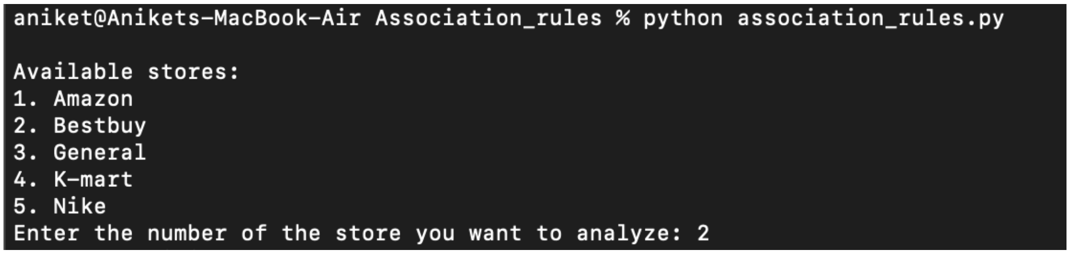
e. Run the script using the command:



```
aniket@Anikets-MacBook-Air Association_rules % python Association_rules.py
```

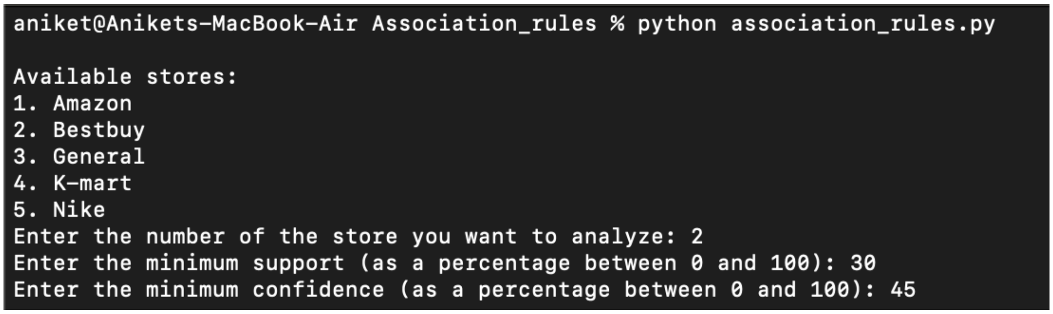
f. Program Interaction & Execution:

When the script is running you will get a screen like this, then type in a number through 1-5 and press *Enter* to select a store:



```
aniket@Anikets-MacBook-Air Association_rules % python association_rules.py

Available stores:
1. Amazon
2. Bestbuy
3. General
4. K-mart
5. Nike
Enter the number of the store you want to analyze: 2
```

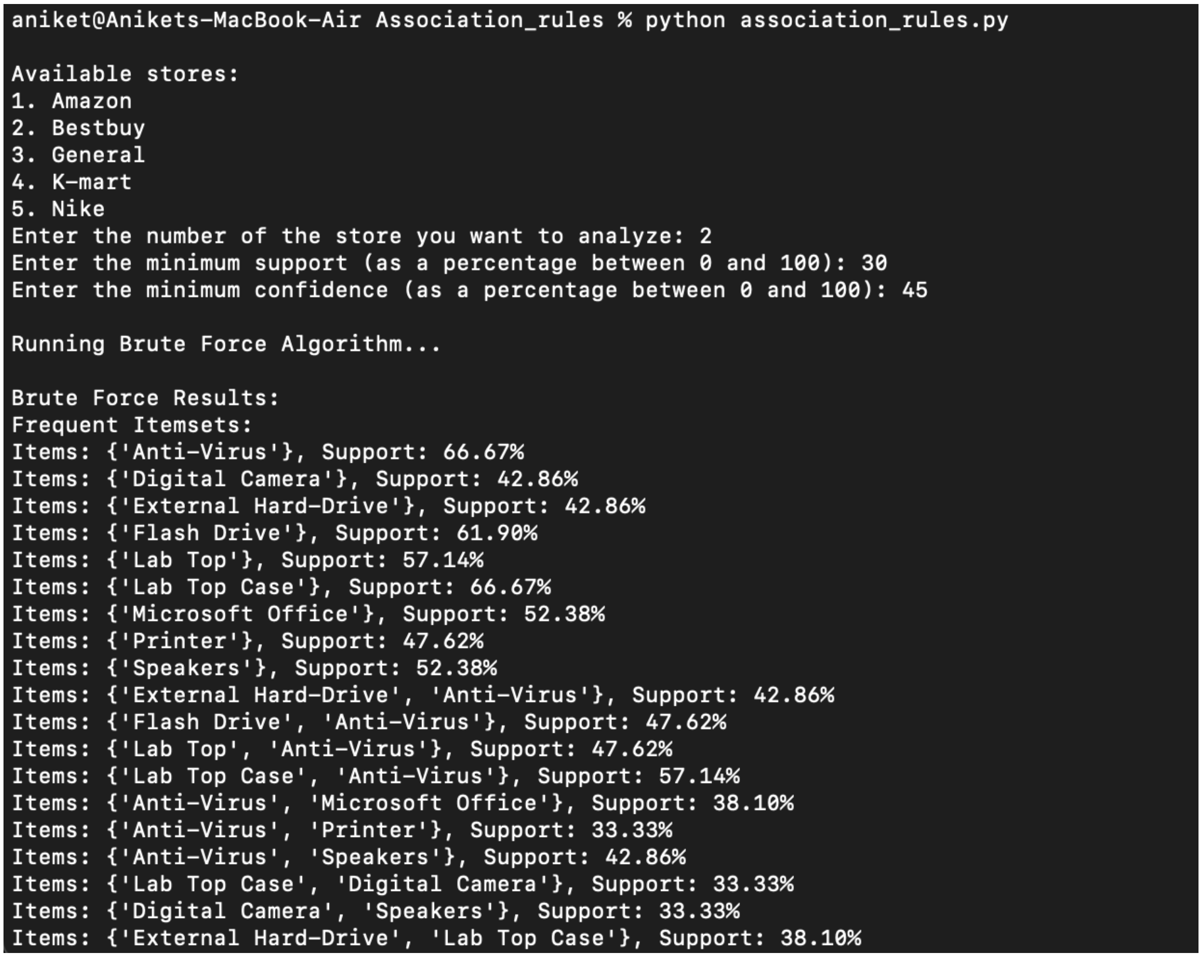
Now type the minimum support, it should be a number from 0-100 as it represents a percentage and press *Enter*, after that type the minimum confidence as well from 0-100 and press *Enter*:



```
aniket@Anikets-MacBook-Air Association_rules % python association_rules.py

Available stores:
1. Amazon
2. Bestbuy
3. General
4. K-mart
5. Nike
Enter the number of the store you want to analyze: 2
Enter the minimum support (as a percentage between 0 and 100): 30
Enter the minimum confidence (as a percentage between 0 and 100): 45
```

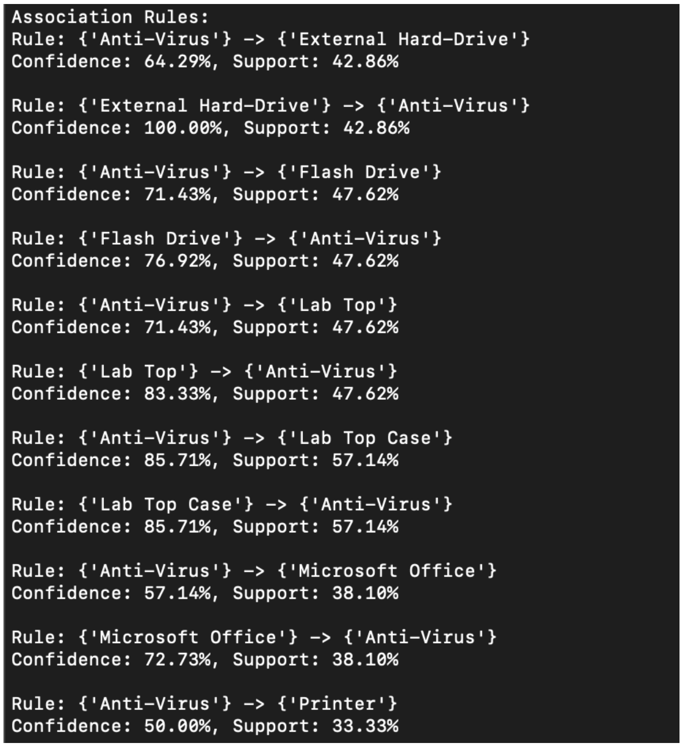
The program will now run all 3 algorithms starting with Brute Force & display the itemsets from selected store along with the support percentage:

<u>Brute Force Implementation</u>:



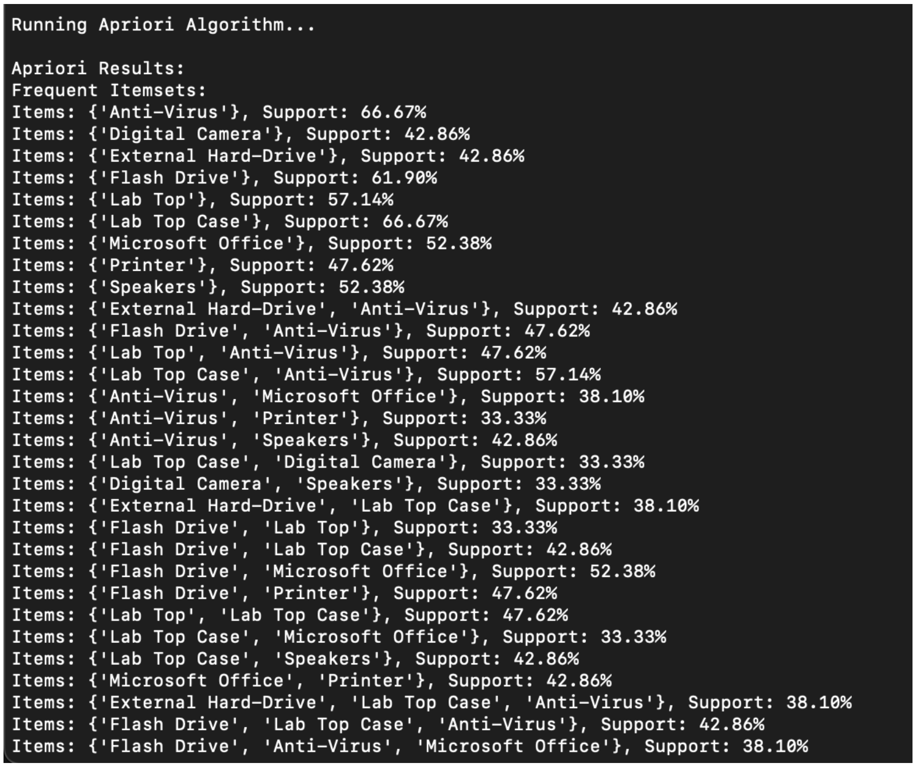
```
aniket@Anikets-MacBook-Air Association_rules % python association_rules.py

Available stores:
1. Amazon
2. Bestbuy
3. General
4. K-mart
5. Nike
Enter the number of the store you want to analyze: 2
Enter the minimum support (as a percentage between 0 and 100): 30
Enter the minimum confidence (as a percentage between 0 and 100): 45

Running Brute Force Algorithm...

Brute Force Results:
Frequent Itemsets:
Items: {'Anti-Virus'}, Support: 66.67%
Items: {'Digital Camera'}, Support: 42.86%
Items: {'External Hard-Drive'}, Support: 42.86%
Items: {'Flash Drive'}, Support: 61.90%
Items: {'Lab Top'}, Support: 57.14%
Items: {'Lab Top Case'}, Support: 66.67%
Items: {'Microsoft Office'}, Support: 52.38%
Items: {'Printer'}, Support: 47.62%
Items: {'Speakers'}, Support: 52.38%
Items: {'External Hard-Drive', 'Anti-Virus'}, Support: 42.86%
Items: {'Flash Drive', 'Anti-Virus'}, Support: 47.62%
Items: {'Lab Top', 'Anti-Virus'}, Support: 47.62%
Items: {'Lab Top Case', 'Anti-Virus'}, Support: 57.14%
Items: {'Anti-Virus', 'Microsoft Office'}, Support: 38.10%
Items: {'Anti-Virus', 'Printer'}, Support: 33.33%
Items: {'Anti-Virus', 'Speakers'}, Support: 42.86%
Items: {'Lab Top Case', 'Digital Camera'}, Support: 33.33%
Items: {'Digital Camera', 'Speakers'}, Support: 33.33%
Items: {'External Hard-Drive', 'Lab Top Case'}, Support: 38.10%
```

It will also display the association rules along with their confidence and support percentage:

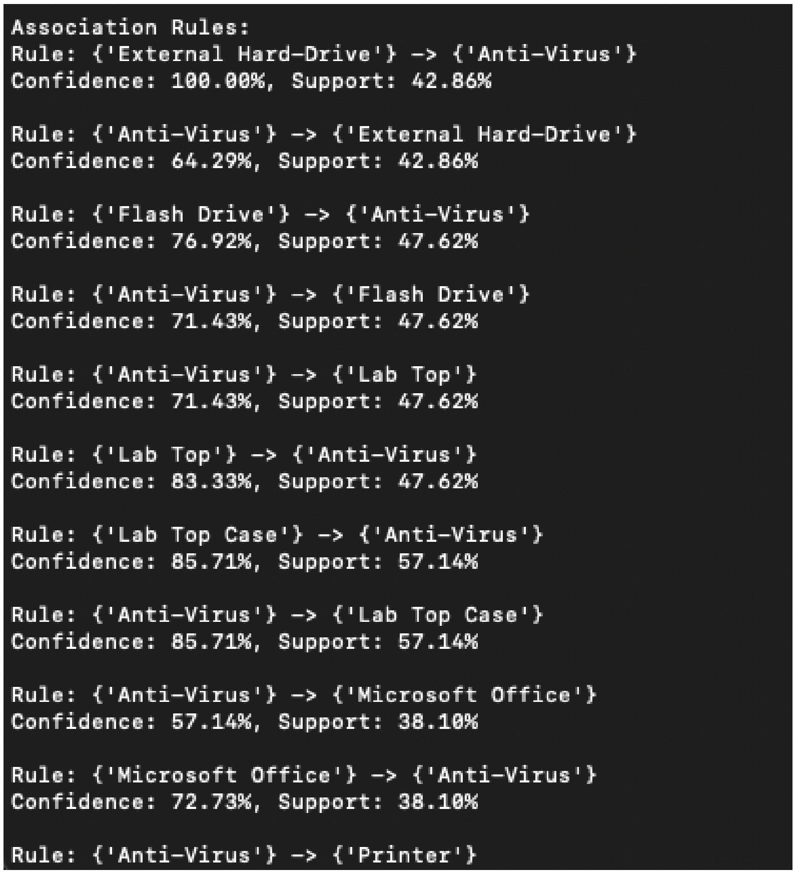


```
Association Rules:
Rule: {'Anti-Virus'} -> {'External Hard-Drive'}
Confidence: 64.29%, Support: 42.86%

Rule: {'External Hard-Drive'} -> {'Anti-Virus'}
Confidence: 100.00%, Support: 42.86%

Rule: {'Anti-Virus'} -> {'Flash Drive'}
Confidence: 71.43%, Support: 47.62%

Rule: {'Flash Drive'} -> {'Anti-Virus'}
Confidence: 76.92%, Support: 47.62%

Rule: {'Anti-Virus'} -> {'Lab Top'}
Confidence: 71.43%, Support: 47.62%

Rule: {'Lab Top'} -> {'Anti-Virus'}
Confidence: 83.33%, Support: 47.62%

Rule: {'Anti-Virus'} -> {'Lab Top Case'}
Confidence: 85.71%, Support: 57.14%

Rule: {'Lab Top Case'} -> {'Anti-Virus'}
Confidence: 85.71%, Support: 57.14%

Rule: {'Anti-Virus'} -> {'Microsoft Office'}
Confidence: 57.14%, Support: 38.10%

Rule: {'Microsoft Office'} -> {'Anti-Virus'}
Confidence: 72.73%, Support: 38.10%

Rule: {'Anti-Virus'} -> {'Printer'}
Confidence: 50.00%, Support: 33.33%
```

Apriori Algorithm Implementation:

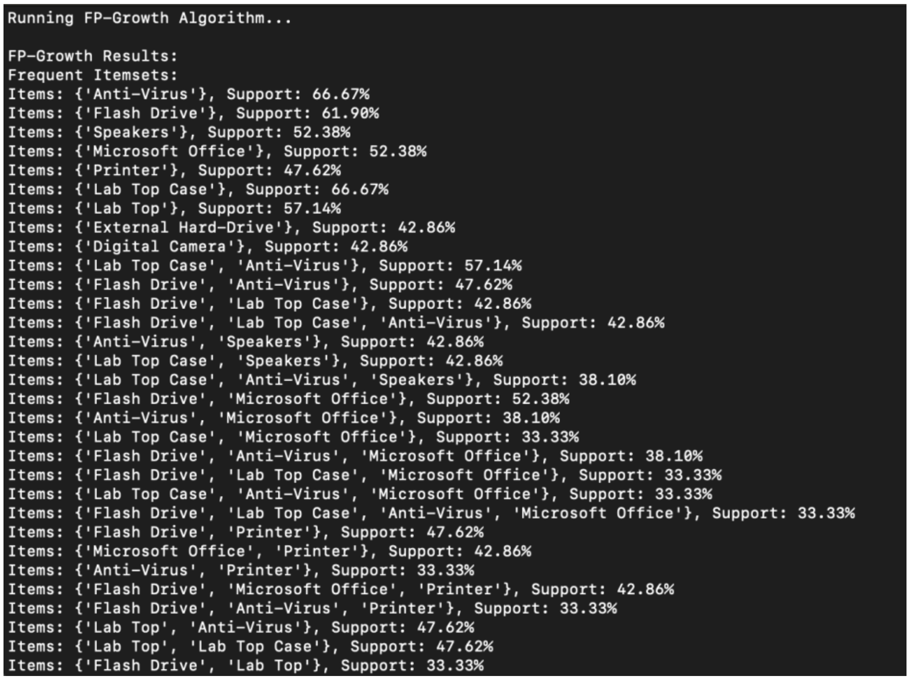


```
Running Apriori Algorithm...

Apriori Results:
Frequent Itemsets:
Items: {'Anti-Virus'}, Support: 66.67%
Items: {'Digital Camera'}, Support: 42.86%
Items: {'External Hard-Drive'}, Support: 42.86%
Items: {'Flash Drive'}, Support: 61.90%
Items: {'Lab Top'}, Support: 57.14%
Items: {'Lab Top Case'}, Support: 66.67%
Items: {'Microsoft Office'}, Support: 52.38%
Items: {'Printer'}, Support: 47.62%
Items: {'Speakers'}, Support: 52.38%
Items: {'External Hard-Drive', 'Anti-Virus'}, Support: 42.86%
Items: {'Flash Drive', 'Anti-Virus'}, Support: 47.62%
Items: {'Lab Top', 'Anti-Virus'}, Support: 47.62%
Items: {'Lab Top Case', 'Anti-Virus'}, Support: 57.14%
Items: {'Anti-Virus', 'Microsoft Office'}, Support: 38.10%
Items: {'Anti-Virus', 'Printer'}, Support: 33.33%
Items: {'Anti-Virus', 'Speakers'}, Support: 42.86%
Items: {'Lab Top Case', 'Digital Camera'}, Support: 33.33%
Items: {'Digital Camera', 'Speakers'}, Support: 33.33%
Items: {'External Hard-Drive', 'Lab Top Case'}, Support: 38.10%
Items: {'Flash Drive', 'Lab Top'}, Support: 33.33%
Items: {'Flash Drive', 'Lab Top Case'}, Support: 42.86%
Items: {'Flash Drive', 'Microsoft Office'}, Support: 52.38%
Items: {'Flash Drive', 'Printer'}, Support: 47.62%
Items: {'Lab Top', 'Lab Top Case'}, Support: 47.62%
Items: {'Lab Top Case', 'Microsoft Office'}, Support: 33.33%
Items: {'Lab Top Case', 'Speakers'}, Support: 42.86%
Items: {'Microsoft Office', 'Printer'}, Support: 42.86%
Items: {'External Hard-Drive', 'Lab Top Case', 'Anti-Virus'}, Support: 38.10%
Items: {'Flash Drive', 'Lab Top Case', 'Anti-Virus'}, Support: 42.86%
Items: {'Flash Drive', 'Anti-Virus', 'Microsoft Office'}, Support: 38.10%
```

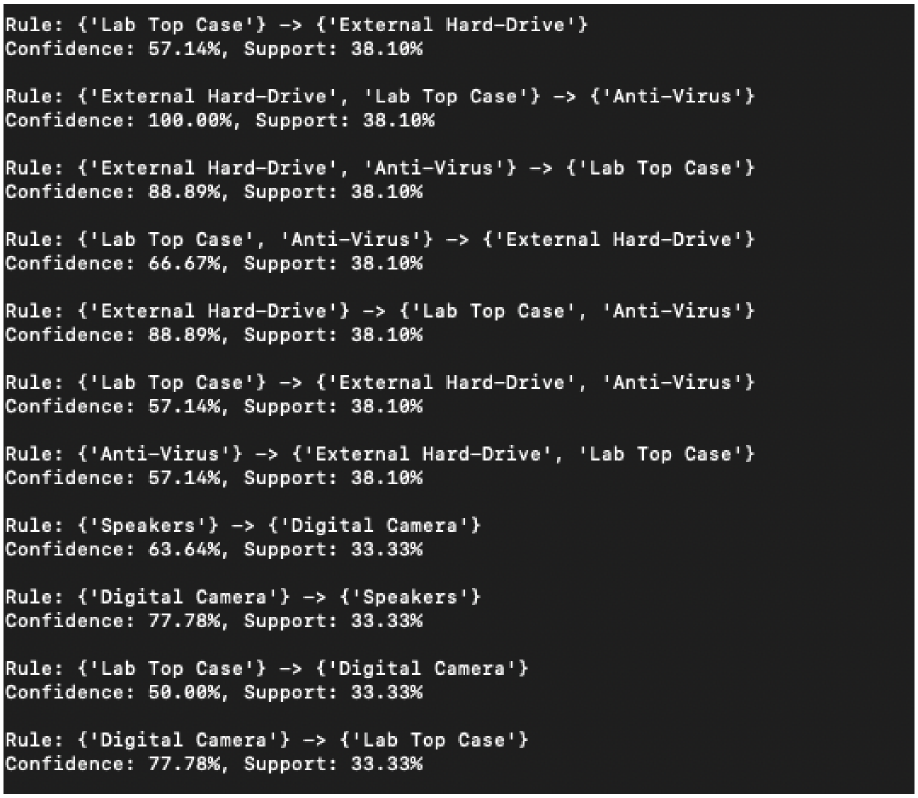


```
Association Rules:
Rule: {'External Hard-Drive'} -> {'Anti-Virus'}
Confidence: 100.00%, Support: 42.86%

Rule: {'Anti-Virus'} -> {'External Hard-Drive'}
Confidence: 64.29%, Support: 42.86%

Rule: {'Flash Drive'} -> {'Anti-Virus'}
Confidence: 76.92%, Support: 47.62%

Rule: {'Anti-Virus'} -> {'Flash Drive'}
Confidence: 71.43%, Support: 47.62%

Rule: {'Anti-Virus'} -> {'Lab Top'}
Confidence: 71.43%, Support: 47.62%

Rule: {'Lab Top'} -> {'Anti-Virus'}
Confidence: 83.33%, Support: 47.62%

Rule: {'Lab Top Case'} -> {'Anti-Virus'}
Confidence: 85.71%, Support: 57.14%

Rule: {'Anti-Virus'} -> {'Lab Top Case'}
Confidence: 85.71%, Support: 57.14%

Rule: {'Anti-Virus'} -> {'Microsoft Office'}
Confidence: 57.14%, Support: 38.10%

Rule: {'Microsoft Office'} -> {'Anti-Virus'}
Confidence: 72.73%, Support: 38.10%

Rule: {'Anti-Virus'} -> {'Printer'}
```

<u>FP-Growth Algorithm Implementation</u>:



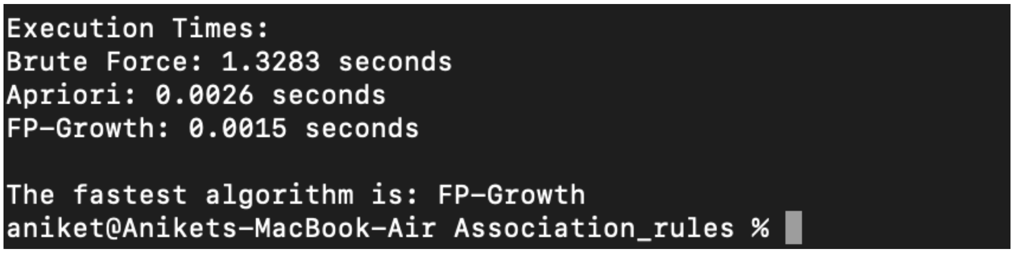
```
Running FP-Growth Algorithm...

FP-Growth Results:
Frequent Itemsets:
Items: {'Anti-Virus'}, Support: 66.67%
Items: {'Flash Drive'}, Support: 61.90%
Items: {'Speakers'}, Support: 52.38%
Items: {'Microsoft Office'}, Support: 52.38%
Items: {'Printer'}, Support: 47.62%
Items: {'Lab Top Case'}, Support: 66.67%
Items: {'Lab Top'}, Support: 57.14%
Items: {'External Hard-Drive'}, Support: 42.86%
Items: {'Digital Camera'}, Support: 42.86%
Items: {'Lab Top Case', 'Anti-Virus'}, Support: 57.14%
Items: {'Flash Drive', 'Anti-Virus'}, Support: 47.62%
Items: {'Flash Drive', 'Lab Top Case'}, Support: 42.86%
Items: {'Flash Drive', 'Lab Top Case', 'Anti-Virus'}, Support: 42.86%
Items: {'Anti-Virus', 'Speakers'}, Support: 42.86%
Items: {'Lab Top Case', 'Speakers'}, Support: 42.86%
Items: {'Lab Top Case', 'Anti-Virus', 'Speakers'}, Support: 38.10%
Items: {'Flash Drive', 'Microsoft Office'}, Support: 52.38%
Items: {'Anti-Virus', 'Microsoft Office'}, Support: 38.10%
Items: {'Lab Top Case', 'Microsoft Office'}, Support: 33.33%
Items: {'Flash Drive', 'Anti-Virus', 'Microsoft Office'}, Support: 38.10%
Items: {'Flash Drive', 'Lab Top Case', 'Microsoft Office'}, Support: 33.33%
Items: {'Lab Top Case', 'Anti-Virus', 'Microsoft Office'}, Support: 33.33%
Items: {'Flash Drive', 'Lab Top Case', 'Anti-Virus', 'Microsoft Office'}, Support: 33.33%
Items: {'Flash Drive', 'Printer'}, Support: 47.62%
Items: {'Microsoft Office', 'Printer'}, Support: 42.86%
Items: {'Anti-Virus', 'Printer'}, Support: 33.33%
Items: {'Flash Drive', 'Microsoft Office', 'Printer'}, Support: 42.86%
Items: {'Flash Drive', 'Anti-Virus', 'Printer'}, Support: 33.33%
Items: {'Lab Top', 'Anti-Virus'}, Support: 47.62%
Items: {'Lab Top', 'Lab Top Case'}, Support: 47.62%
Items: {'Flash Drive', 'Lab Top'}, Support: 33.33%
```



```
Rule: {'Lab Top Case'} -> {'External Hard-Drive'}
Confidence: 57.14%, Support: 38.10%

Rule: {'External Hard-Drive', 'Lab Top Case'} -> {'Anti-Virus'}
Confidence: 100.00%, Support: 38.10%

Rule: {'External Hard-Drive', 'Anti-Virus'} -> {'Lab Top Case'}
Confidence: 88.89%, Support: 38.10%

Rule: {'Lab Top Case', 'Anti-Virus'} -> {'External Hard-Drive'}
Confidence: 66.67%, Support: 38.10%

Rule: {'External Hard-Drive'} -> {'Lab Top Case', 'Anti-Virus'}
Confidence: 88.89%, Support: 38.10%

Rule: {'Lab Top Case'} -> {'External Hard-Drive', 'Anti-Virus'}
Confidence: 57.14%, Support: 38.10%

Rule: {'Anti-Virus'} -> {'External Hard-Drive', 'Lab Top Case'}
Confidence: 57.14%, Support: 38.10%

Rule: {'Speakers'} -> {'Digital Camera'}
Confidence: 63.64%, Support: 33.33%

Rule: {'Digital Camera'} -> {'Speakers'}
Confidence: 77.78%, Support: 33.33%

Rule: {'Lab Top Case'} -> {'Digital Camera'}
Confidence: 50.00%, Support: 33.33%

Rule: {'Digital Camera'} -> {'Lab Top Case'}
Confidence: 77.78%, Support: 33.33%
```

you can notice that for every algorithm it creates and displays association rules with their respective support and confidence percentages.

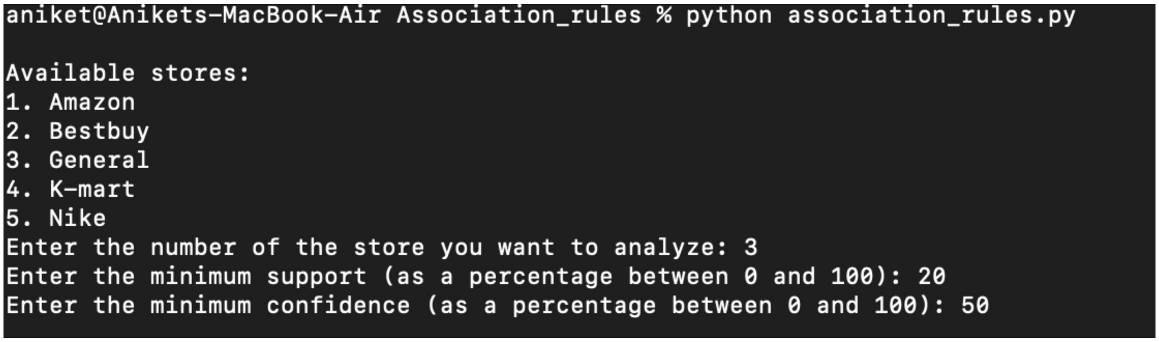
7. **Final Results of the program**:
   Once all the algorithms are finished and executed successfully, it will display the execution times for all 3 algorithms and based on it will tell which performed the fastest:



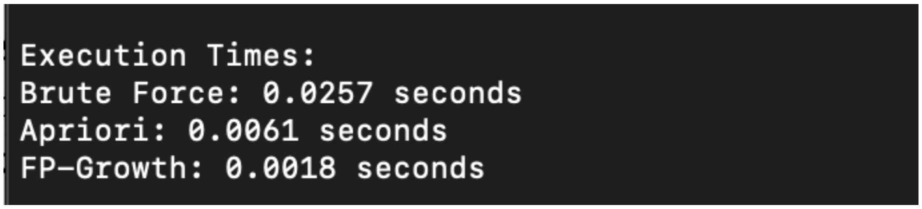
```
Execution Times:
Brute Force: 1.3283 seconds
Apriori: 0.0026 seconds
FP-Growth: 0.0015 seconds

The fastest algorithm is: FP-Growth
aniket@Anikets-MacBook-Air Association_rules % 
```

8. **Trying out different parameters**
   Below is the execution times for different algorithms with different parameters for ‘General’, but this can be tried with other stores as well.

For support = 20% and confidence = 50% the execution times are:

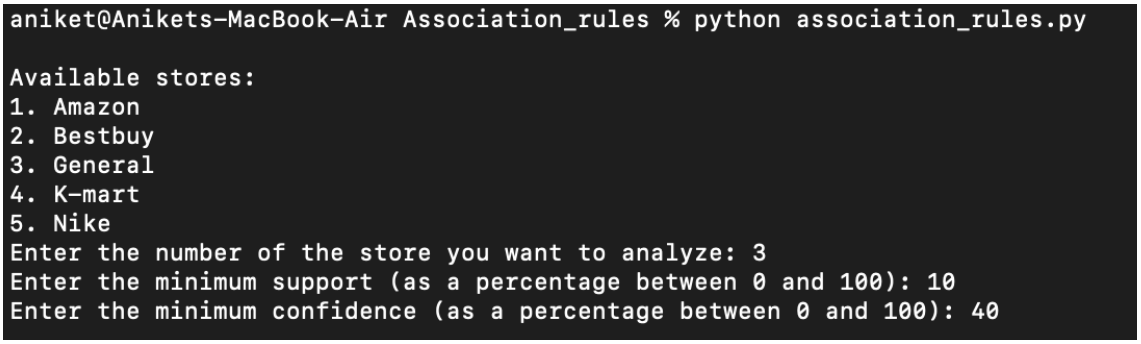


```
aniket@Anikets-MacBook-Air Association_rules % python association_rules.py

Available stores:
1. Amazon
2. Bestbuy
3. General
4. K-mart
5. Nike
Enter the number of the store you want to analyze: 3
Enter the minimum support (as a percentage between 0 and 100): 20
Enter the minimum confidence (as a percentage between 0 and 100): 50
```

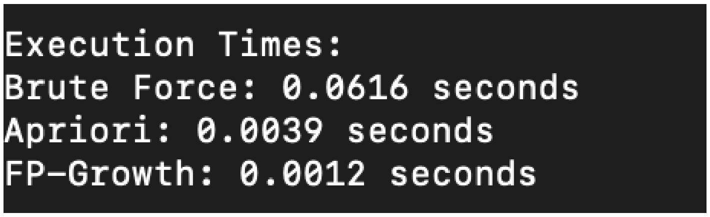


```
Execution Times:
Brute Force: 0.0257 seconds
Apriori: 0.0061 seconds
FP-Growth: 0.0018 seconds
```

For support = 10% and confidence = 40% the execution times are:



```
aniket@Anikets-MacBook-Air Association_rules % python association_rules.py

Available stores:
1. Amazon
2. Bestbuy
3. General
4. K-mart
5. Nike
Enter the number of the store you want to analyze: 3
Enter the minimum support (as a percentage between 0 and 100): 10
Enter the minimum confidence (as a percentage between 0 and 100): 40
```



```
Execution Times:
Brute Force: 0.0616 seconds
Apriori: 0.0039 seconds
FP-Growth: 0.0012 seconds
```

Result & Analysis:

The three algorithms; Brute Force, Apriori, and FP-Growth were tested on the General Store dataset using a minimum support threshold of 20% and a minimum confidence threshold of 50%. The algorithms produced consistent frequent itemsets and association rules, validating their correctness.

In terms of performance, the Brute Force algorithm took the longest (0.0257 seconds) because it exhaustively evaluated all possible itemset combinations without pruning, which becomes increasingly expensive as the dataset size grows. The Apriori algorithm was significantly faster (0.0061 seconds) by pruning infrequent candidate itemsets early in the process, thereby reducing the number of combinations to evaluate. The FP-Growth algorithm achieved the fastest execution time (0.0018 seconds) due to its efficient FP-tree structure, which compresses transactions and avoids candidate generation altogether.

Overall, the runtime results align with the expected computational characteristics of the three algorithms: brute-force being the slowest, Apriori providing a clear efficiency boost through pruning, and FP-Growth offering superior scalability and speed.

Conclusion:

This project highlights the trade-offs between the three frequent itemset mining algorithms. While Brute Force is conceptually simple and guarantees accurate results, it scales poorly with larger datasets due to its exponential search space. Apriori improves efficiency through systematic pruning but still relies on iterative candidate generation. FP-Growth emerges as the most efficient approach, leveraging its tree-based structure to achieve the fastest performance while producing the same frequent itemsets and association rules as the other methods.

In conclusion, FP-Growth is the preferred choice for larger datasets where computational efficiency is crucial, while Apriori offers a good balance of simplicity and performance. Brute Force remains useful primarily for small datasets or for educational demonstration purposes.

GitHub Link:

https://github.com/Aniket-NJIT/khalate-aniket-midtermproject.git