NAME: Abhiram Krishnam
NJIT UCID: ak3236
Email Address: ak3236@njit.edu
Date: 4th October, 2024
Professor: Yasser Abduallah
CS 634(101) - Data Mining

Midterm Project Report

Abstract:

Frequent pattern mining and association rule generation are essential techniques for uncovering meaningful insights from large datasets. In this project, we implement three core algorithms for these tasks: Apriori, FP-Growth, and Brute Force. Each method employs a distinct strategy for discovering patterns and generating association rules in transactional data.

The Apriori algorithm uses a level-wise search, applying the Apriori principle to reduce the search space by pruning unlikely candidates. FP-Growth improves performance with a compact FP-tree structure, eliminating the need for candidate generation. Additionally, we examine the brute force method as a reference point, discussing its simplicity and computational challenges.

This project provides in-depth explanations and implementation details for each algorithm, offering practical guidance for users. By performing empirical evaluations on standard datasets, we evaluate the scalability and efficiency of each method, highlighting the superior performance of FP-Growth on large datasets.

Ultimately, this project delivers accessible implementations and valuable insights to further research and experimentation in the field of frequent pattern mining and association rule generation.

Introduction:

Discovering hidden patterns and relationships within large datasets is a fundamental aspect of modern data analysis. In this project, we explore three key algorithms—Apriori, FP-Growth, and Brute Force—each crucial for association rule mining, with an emphasis on their practical use in the retail sector. We provide a detailed overview of the core data mining concepts and techniques that form the foundation of our approach.

Association Rule Generation is a key component of data mining, aiming to uncover meaningful relationships between items in transactional datasets. The process begins by identifying the most frequent items across a set of transactions. Using a user-defined support threshold, the support value for each item is calculated, which helps filter out items that do not meet this threshold. The Apriori algorithm, a well-known method in this field, iteratively builds larger itemsets by combining smaller frequent ones and removes those that don't satisfy the minimum support criteria.

Once the frequent itemsets are identified, the next step is to generate association rules. These rules are formed by splitting each frequent itemset into antecedents and consequents, calculating the confidence for each rule to ensure they meet the user-specified confidence threshold. By traversing all possible combinations of itemsets, association rules can be created, revealing patterns and relationships that provide valuable insights into how items in a dataset are connected. This process not only helps in identifying frequent itemsets but also in deriving actionable insights that guide decision-making, especially in fields like retail and market basket analysis.

Implementation:

Our implementation of the algorithms is applied to a customized dataset related to a retail setting. The primary steps include:

1) Initializing DataFrames to store candidate and frequent itemsets.
2) Loading and preprocessing transactional data from CSV files, ensuring items are ordered and unique.
3) Gathering user input to define minimum support and confidence thresholds.
4) Iteratively generating candidate itemsets and updating frequent itemsets using the chosen algorithms.

Core Principles:

- **Frequent Itemset Identification**:
  At the core of the Apriori Algorithm is the process of discovering frequent itemsets—groups of items that consistently occur together in transactions. These itemsets provide valuable insights into customer purchasing behaviors and trends.
- **Support and Confidence Metrics**:
  Support and confidence are critical measures in data mining. Support reflects how often an item or itemset appears in the dataset, while confidence evaluates the probability of items being purchased together. These metrics are key in identifying meaningful patterns.

- **Association Rule Generation**:
  Association rules highlight which items are frequently bought together. These rules are essential

for improving business strategies, such as creating personalized recommendations and enhancing sales approaches.

Workflow:

This project follows a systematic workflow, integrating multiple stages to implement the Apriori, FP-Growth, and Brute Force algorithms:

1) **Data Loading and Preprocessing**:
We begin by loading transactional data from a retail dataset, followed by filtering for unique items and organizing them based on predefined criteria.

2) **Setting Minimum Support and Confidence**:
User input is utilized to establish minimum support and confidence thresholds, which are critical for refining and identifying significant patterns.

3) **Iterating Through Candidate Itemsets**:
Using a brute-force strategy, we systematically evaluate all possible itemset combinations, gradually increasing their size to uncover frequent patterns.

4) **Support and Confidence Evaluation**:
For each candidate itemset, we compute the support count, and subsequently, the confidence levels are calculated to identify strong association rules.

5) **Generating Association Rules**:
Association rules that meet the defined support and confidence criteria are generated, offering valuable insights into items frequently purchased together.

File Details:

Here's a brief overview of the various Python files in the project:

1. **apriori.py**: This file implements the Apriori algorithm for discovering frequent itemsets and generating association rules.
2. **fp_growth.py**: This file implements the FP-Growth algorithm to find frequent itemsets and extract association rules.
3. **brute_force.py**: This file provides the implementation of the Brute Force algorithm for generating association rules.
4. **input.py**: This file includes functions for selecting and reading transaction data from a chosen database file.

5. **select_file.py**: This file contains a function that prompts the user to choose the appropriate database file.
6. **main.py**: This is the main script that integrates all components. It reads the transaction data from the selected database, collects the user-defined support and confidence thresholds, and applies all three algorithms to extract association rules.

Prerequisites:

Before running the project, ensure you have the following prerequisites installed:

1. Python 3.x: You need to have Python 3.x installed on your system. You can download it

from the official Python website: https://www.python.org/downloads/

2. Required Python Libraries: The project requires the following Python libraries:

- Pandas
- mlxtend

You can install these libraries using pip, the Python package installer. Open your terminal or

command prompt and run the following commands:

pip install pandas

pip install mlxtend

Running the Project:

1) **Download Project Files**: Download the necessary Python scripts (such as apriori.py, fp_growth.py, brute_force.py, input.py, select_file.py, and main.py) along with the dataset files (e.g., Amazon.csv) to a folder on your local computer.

2) **Navigate to Project Directory**: Open your terminal or command prompt and use the cd command to move to the folder where the project files are saved.

3) **Run the Main Script**: Execute the main script by running the following command in your terminal or command prompt:

```
python main.py
```

4) **Select a Database**: When prompted, choose the dataset by entering the corresponding number as shown in the on-screen prompt.

5) **Input Support and Confidence Values**: After selecting the dataset, the script will ask for the minimum support and confidence values (in percentage form) which will be used by the algorithms to generate association rules.

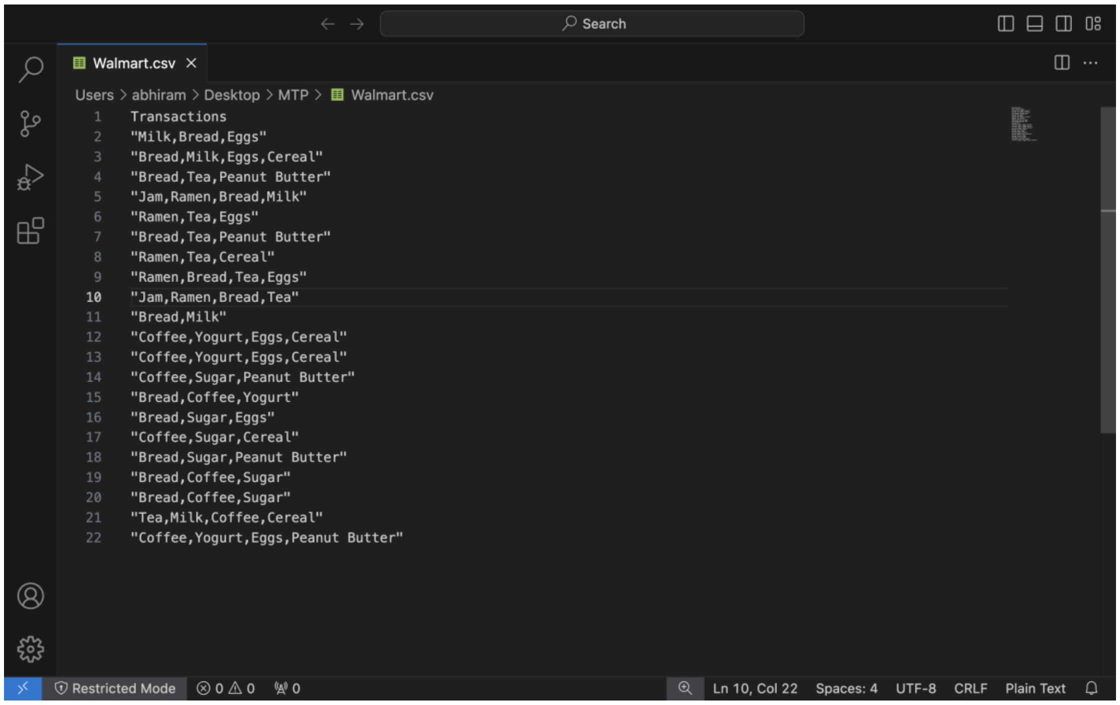
6) **View Results**: The script will run all three algorithms (Apriori, FP-Growth, and Brute Force) and display the identified association rules along with their corresponding support and confidence values. It will also show the execution time for each algorithm.

Summary:

This project illustrates the successful application of the Apriori, FP-Growth, and Brute Force algorithms for mining frequent patterns and generating association rules. By following a well-defined workflow, we effectively load and preprocess transactional data, establish user-specified support and confidence thresholds, and examine various itemsets to reveal significant patterns. The calculations of support and confidence are essential in identifying strong association rules that yield practical insights into item relationships and consumer purchasing habits. This approach demonstrates how these algorithms can enhance decision-making and strategy formulation in the retail sector, emphasizing the significance of frequent pattern mining in practical contexts.

Screenshots

I have taken 4 databases that are assigned for the project and one of my own.
Here is my sample Walmart CSV file with transactions that I have created for the project:

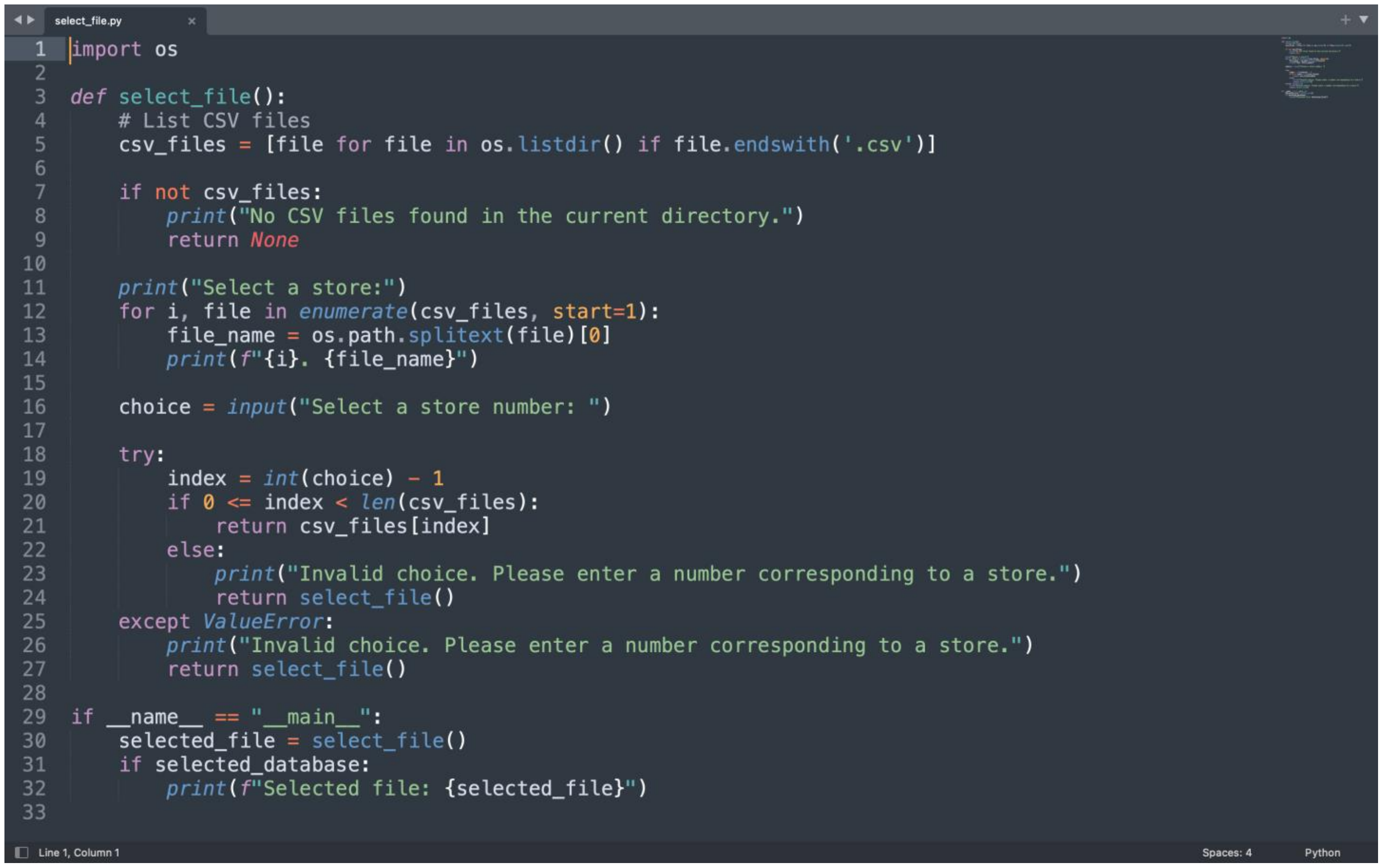

Search
Walmart.csv
Users > abhiram > Desktop > MTP > Walmart.csv
Restricted Mode
Ln 10, Col 22 Spaces: 4 UTF-8 CRLF Plain Text

```
Transactions
"Milk,Bread,Eggs"
"Bread,Milk,Eggs,Cereal"
"Bread,Tea,Peanut Butter"
"Jam,Ramen,Bread,Milk"
"Ramen,Tea,Eggs"
"Bread,Tea,Peanut Butter"
"Ramen,Tea,Cereal"
"Ramen,Bread,Tea,Eggs"
"Jam,Ramen,Bread,Tea"
"Bread,Milk"
"Coffee,Yogurt,Eggs,Cereal"
"Coffee,Yogurt,Eggs,Cereal"
"Coffee,Sugar,Peanut Butter"
"Bread,Coffee,Yogurt"
"Bread,Sugar,Eggs"
"Coffee,Sugar,Cereal"
"Bread,Sugar,Peanut Butter"
"Bread,Coffee,Sugar"
"Bread,Coffee,Sugar"
"Tea,Milk,Coffee,Cereal"
"Coffee,Yogurt,Eggs,Peanut Butter"
```

For the implementation, I developed a main.py file that encompasses all three algorithms.

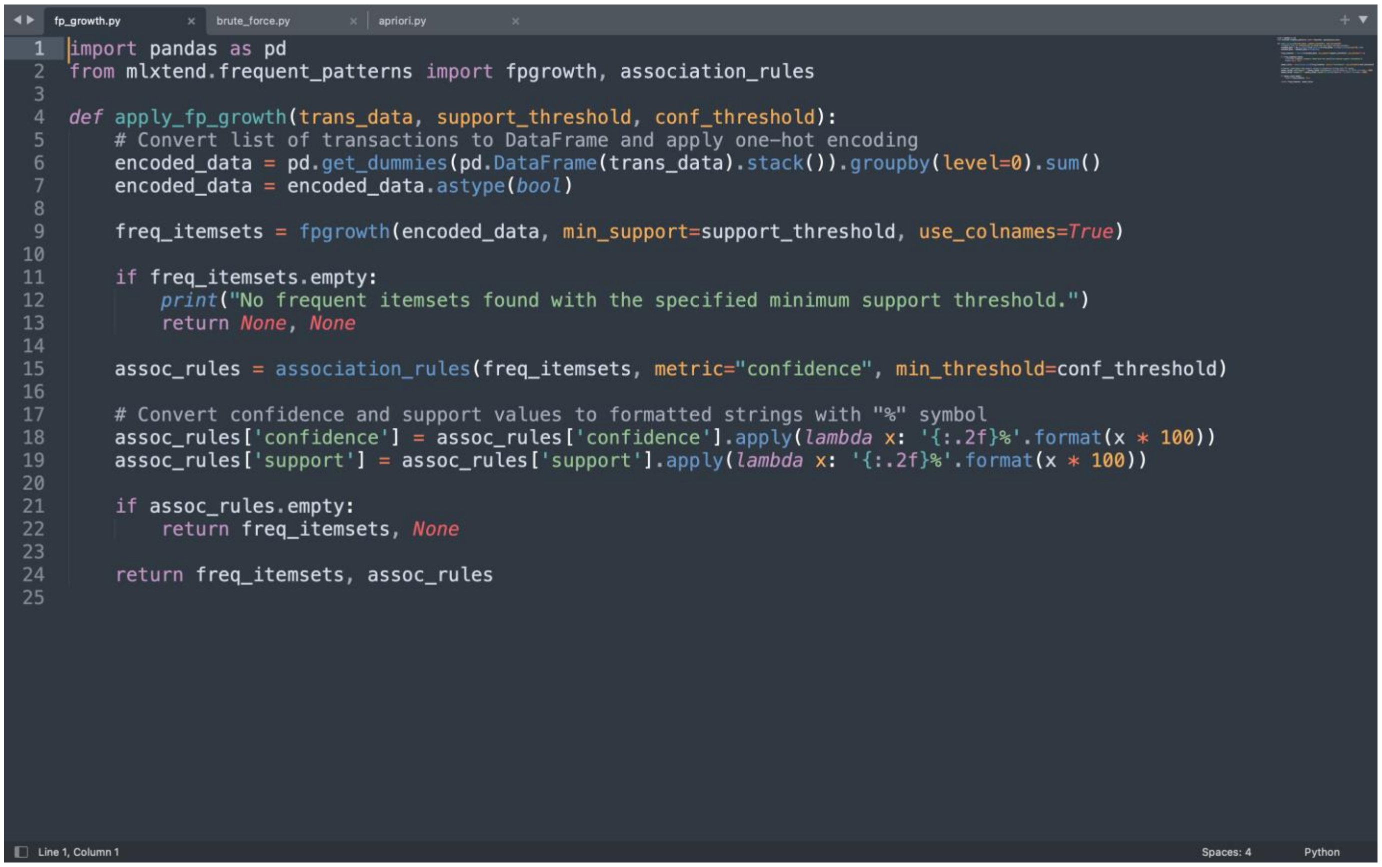
The code includes prompts allowing users to select their preferred store. Initially, we read the CSV files and ensure that the user inputs are validated properly.

select_file.py: To select what dataset/store we want to find the association rules for.



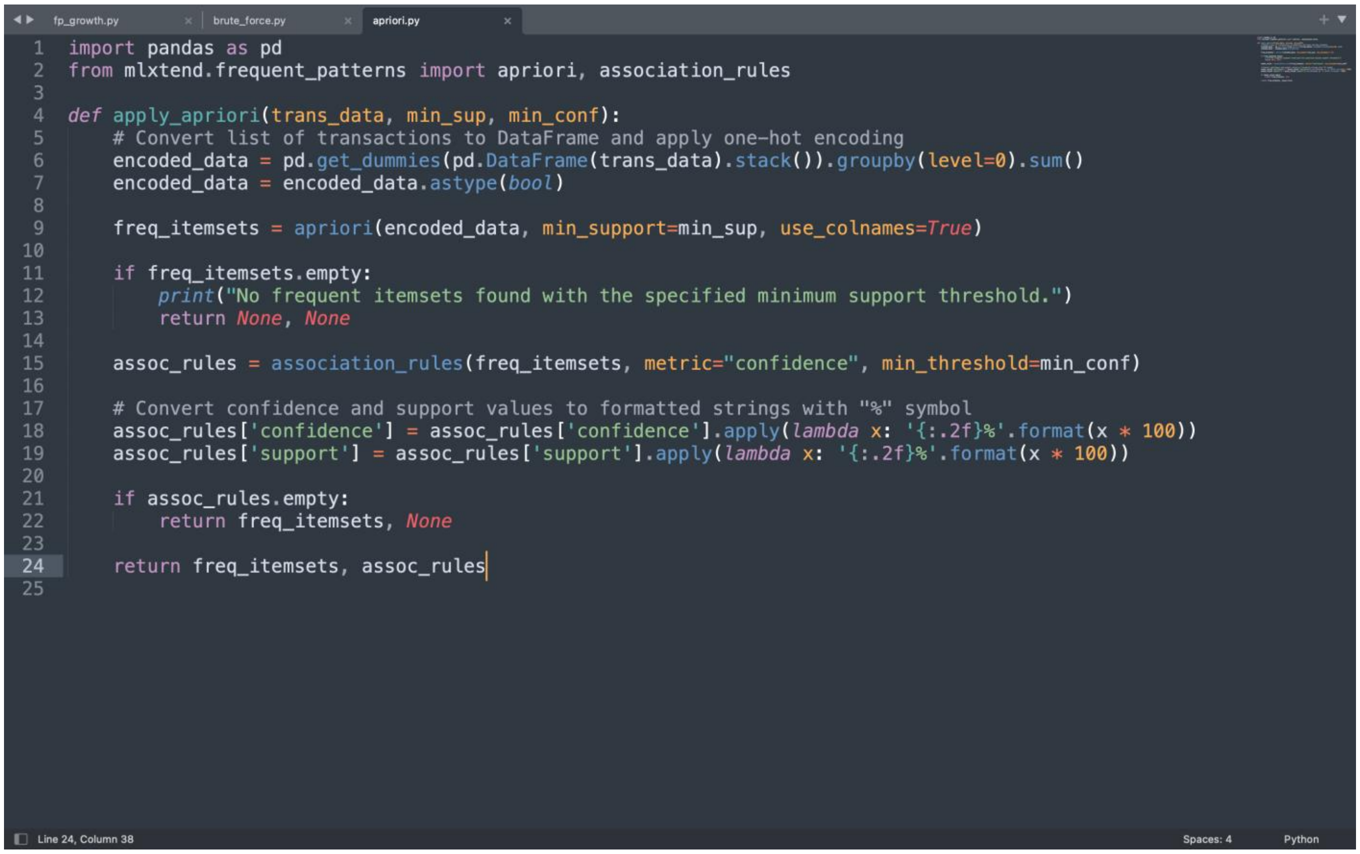
```
import os

def select_file():
    # List CSV files
    csv_files = [file for file in os.listdir() if file.endswith('.csv')]

    if not csv_files:
        print("No CSV files found in the current directory.")
        return None

    print("Select a store:")
    for i, file in enumerate(csv_files, start=1):
        file_name = os.path.splitext(file)[0]
        print(f"{i}. {file_name}")

    choice = input("Select a store number: ")

    try:
        index = int(choice) - 1
        if 0 <= index < len(csv_files):
            return csv_files[index]
        else:
            print("Invalid choice. Please enter a number corresponding to a store.")
            return select_file()
    except ValueError:
        print("Invalid choice. Please enter a number corresponding to a store.")
        return select_file()

if __name__ == "__main__":
    selected_file = select_file()
    if selected_database:
        print(f"Selected file: {selected_file}")
```

fp_growth.py: Implementation of FP-tree algorithm.



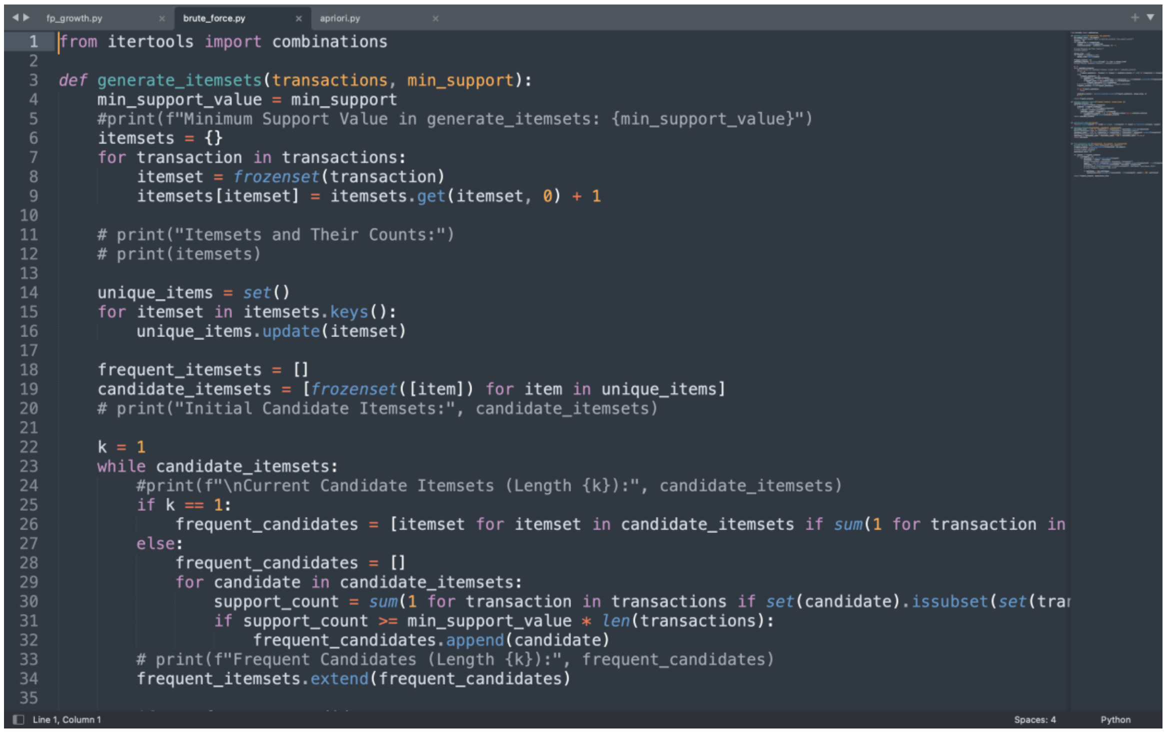
```python
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules

def apply_fp_growth(trans_data, support_threshold, conf_threshold):
    # Convert list of transactions to DataFrame and apply one-hot encoding
    encoded_data = pd.get_dummies(pd.DataFrame(trans_data).stack()).groupby(level=0).sum()
    encoded_data = encoded_data.astype(bool)

    freq_itemsets = fpgrowth(encoded_data, min_support=support_threshold, use_colnames=True)

    if freq_itemsets.empty:
        print("No frequent itemsets found with the specified minimum support threshold.")
        return None, None

    assoc_rules = association_rules(freq_itemsets, metric="confidence", min_threshold=conf_threshold)

    # Convert confidence and support values to formatted strings with "%" symbol
    assoc_rules['confidence'] = assoc_rules['confidence'].apply(lambda x: '{:.2f}%'.format(x * 100))
    assoc_rules['support'] = assoc_rules['support'].apply(lambda x: '{:.2f}%'.format(x * 100))

    if assoc_rules.empty:
        return freq_itemsets, None

    return freq_itemsets, assoc_rules
```

apriori.py: Implementation of Apriori algorithm.

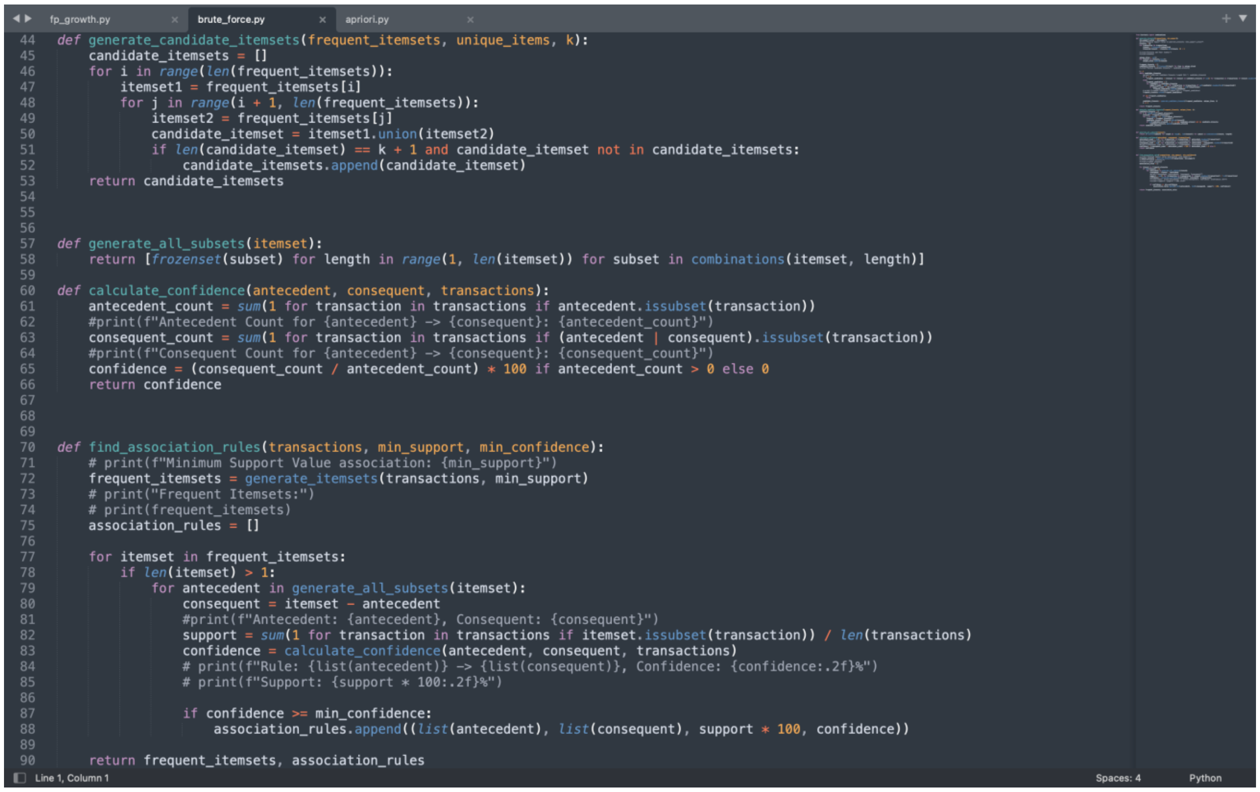


```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

def apply_apriori(trans_data, min_sup, min_conf):
    # Convert list of transactions to DataFrame and apply one-hot encoding
    encoded_data = pd.get_dummies(pd.DataFrame(trans_data).stack()).groupby(level=0).sum()
    encoded_data = encoded_data.astype(bool)

    freq_itemsets = apriori(encoded_data, min_support=min_sup, use_colnames=True)

    if freq_itemsets.empty:
        print("No frequent itemsets found with the specified minimum support threshold.")
        return None, None

    assoc_rules = association_rules(freq_itemsets, metric="confidence", min_threshold=min_conf)

    # Convert confidence and support values to formatted strings with "%" symbol
    assoc_rules['confidence'] = assoc_rules['confidence'].apply(lambda x: '{:.2f}%'.format(x * 100))
    assoc_rules['support'] = assoc_rules['support'].apply(lambda x: '{:.2f}%'.format(x * 100))

    if assoc_rules.empty:
        return freq_itemsets, None

    return freq_itemsets, assoc_rules

```

brute_force.py: Implementation of Brute Force algorithm.

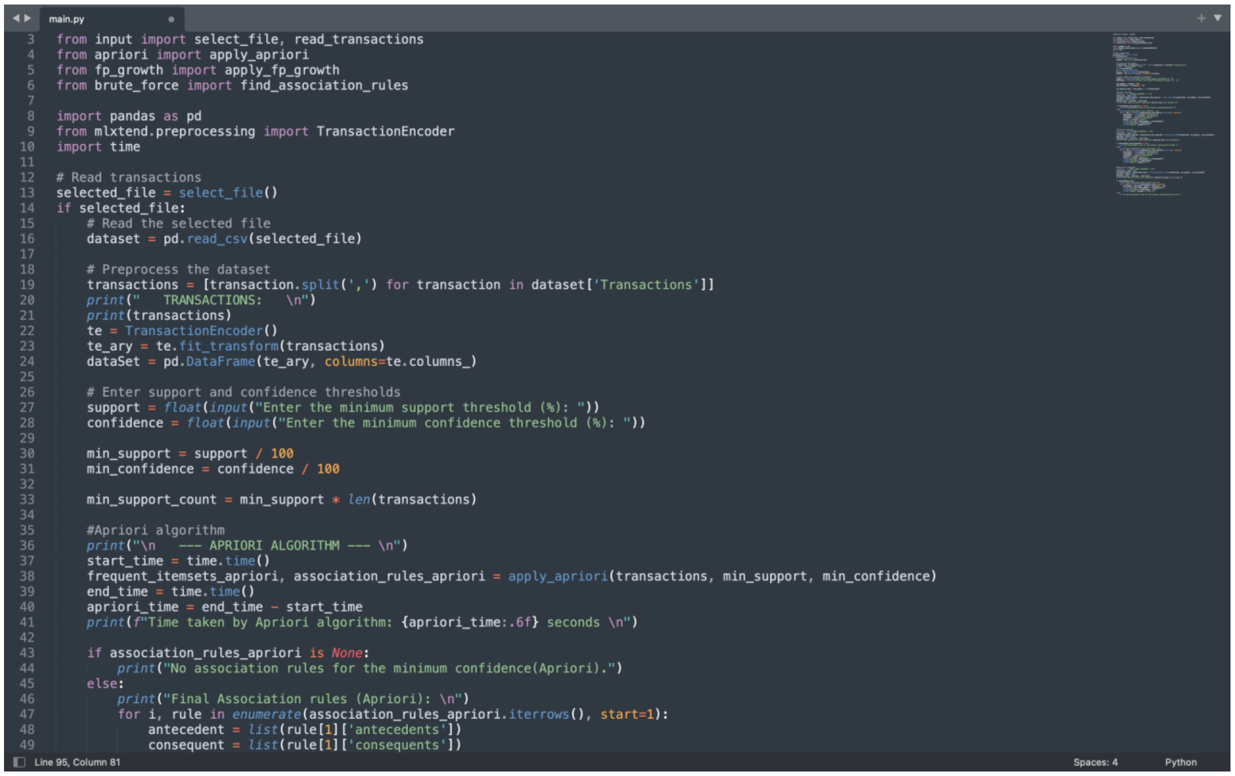


```python
from itertools import combinations

def generate_itemsets(transactions, min_support):
    min_support_value = min_support
    #print(f"Minimum Support Value in generate_itemsets: {min_support_value}")
    itemsets = {}
    for transaction in transactions:
        itemset = frozenset(transaction)
        itemsets[itemset] = itemsets.get(itemset, 0) + 1

    # print("Itemsets and Their Counts:")
    # print(itemsets)

    unique_items = set()
    for itemset in itemsets.keys():
        unique_items.update(itemset)

    frequent_itemsets = []
    candidate_itemsets = [frozenset([item]) for item in unique_items]
    # print("Initial Candidate Itemsets:", candidate_itemsets)

    k = 1
    while candidate_itemsets:
        #print(f"\nCurrent Candidate Itemsets (Length {k}):", candidate_itemsets)
        if k == 1:
            frequent_candidates = [itemset for itemset in candidate_itemsets if sum(1 for transaction in
        else:
            frequent_candidates = []
            for candidate in candidate_itemsets:
                support_count = sum(1 for transaction in transactions if set(candidate).issubset(set(tra
                if support_count >= min_support_value * len(transactions):
                    frequent_candidates.append(candidate)
        # print(f"Frequent Candidates (Length {k}):", frequent_candidates)
        frequent_itemsets.extend(frequent_candidates)
```

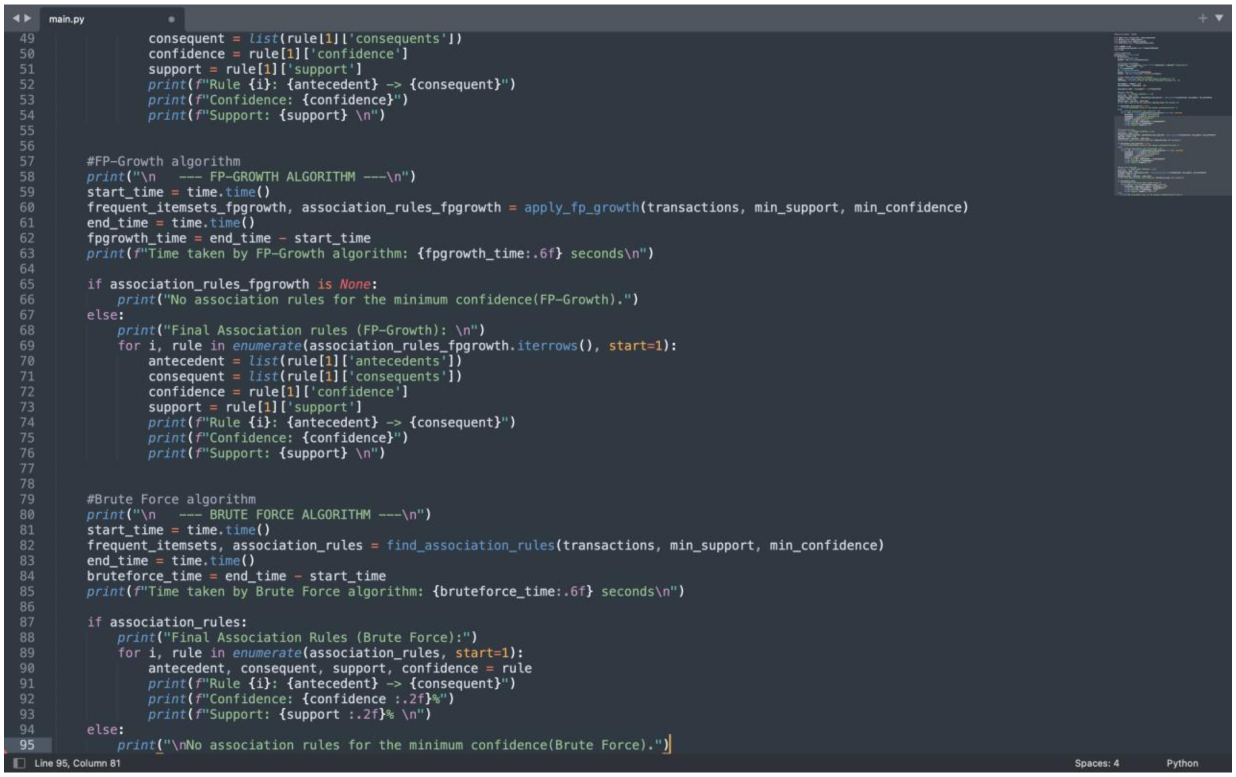


```python
def generate_candidate_itemsets(frequent_itemsets, unique_items, k):
    candidate_itemsets = []
    for i in range(len(frequent_itemsets)):
        itemset1 = frequent_itemsets[i]
        for j in range(i + 1, len(frequent_itemsets)):
            itemset2 = frequent_itemsets[j]
            candidate_itemset = itemset1.union(itemset2)
            if len(candidate_itemset) == k + 1 and candidate_itemset not in candidate_itemsets:
                candidate_itemsets.append(candidate_itemset)
    return candidate_itemsets



def generate_all_subsets(itemset):
    return [frozenset(subset) for length in range(1, len(itemset)) for subset in combinations(itemset, length)]

def calculate_confidence(antecedent, consequent, transactions):
    antecedent_count = sum(1 for transaction in transactions if antecedent.issubset(transaction))
    #print(f"Antecedent Count for {antecedent} -> {consequent}: {antecedent_count}")
    consequent_count = sum(1 for transaction in transactions if (antecedent | consequent).issubset(transaction))
    #print(f"Consequent Count for {antecedent} -> {consequent}: {consequent_count}")
    confidence = (consequent_count / antecedent_count) * 100 if antecedent_count > 0 else 0
    return confidence



def find_association_rules(transactions, min_support, min_confidence):
    # print(f"Minimum Support Value association: {min_support}")
    frequent_itemsets = generate_itemsets(transactions, min_support)
    # print("Frequent Itemsets:")
    # print(frequent_itemsets)
    association_rules = []

    for itemset in frequent_itemsets:
        if len(itemset) > 1:
            for antecedent in generate_all_subsets(itemset):
                consequent = itemset - antecedent
                #print(f"Antecedent: {antecedent}, Consequent: {consequent}")
                support = sum(1 for transaction in transactions if itemset.issubset(transaction)) / len(transactions)
                confidence = calculate_confidence(antecedent, consequent, transactions)
                # print(f"Rule: {list(antecedent)} -> {list(consequent)}, Confidence: {confidence:.2f}%")
                # print(f"Support: {support * 100:.2f}%")

                if confidence >= min_confidence:
                    association_rules.append((list(antecedent), list(consequent), support * 100, confidence))

    return frequent_itemsets, association_rules
```

main.py: The main python file to be executed to prompt the user for the dataset, enter the support and confidence percentage to call and run the 3 algorithms.

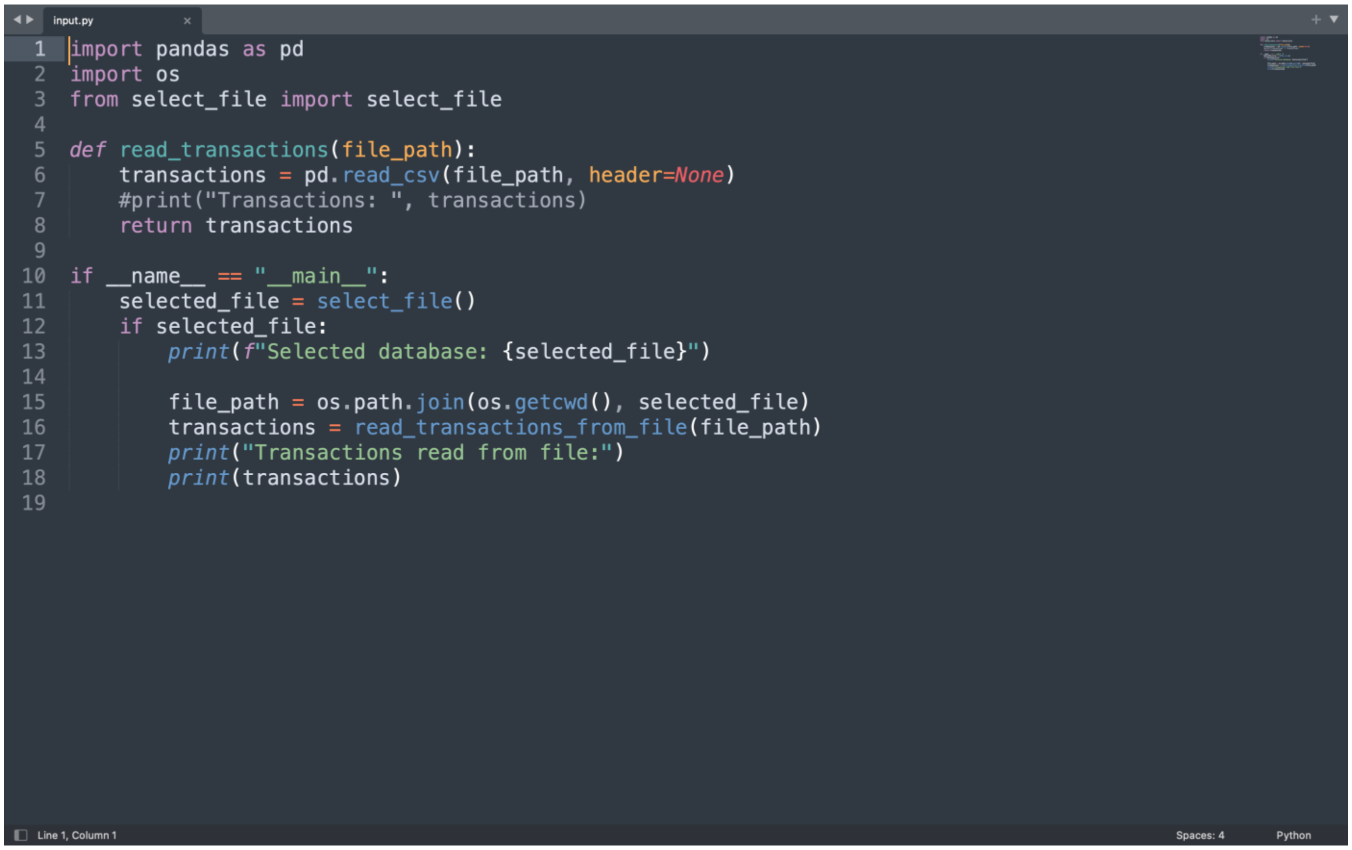


```python
from input import select_file, read_transactions
from apriori import apply_apriori
from fp_growth import apply_fp_growth
from brute_force import find_association_rules

import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
import time

# Read transactions
selected_file = select_file()
if selected_file:
    # Read the selected file
    dataset = pd.read_csv(selected_file)

    # Preprocess the dataset
    transactions = [transaction.split(',') for transaction in dataset['Transactions']]
    print("   TRANSACTIONS:   \n")
    print(transactions)
    te = TransactionEncoder()
    te_ary = te.fit_transform(transactions)
    dataSet = pd.DataFrame(te_ary, columns=te.columns_)

    # Enter support and confidence thresholds
    support = float(input("Enter the minimum support threshold (%): "))
    confidence = float(input("Enter the minimum confidence threshold (%): "))

    min_support = support / 100
    min_confidence = confidence / 100

    min_support_count = min_support * len(transactions)

    #Apriori algorithm
    print("\n   --- APRIORI ALGORITHM --- \n")
    start_time = time.time()
    frequent_itemsets_apriori, association_rules_apriori = apply_apriori(transactions, min_support, min_confidence)
    end_time = time.time()
    apriori_time = end_time - start_time
    print(f"Time taken by Apriori algorithm: {apriori_time:.6f} seconds \n")

    if association_rules_apriori is None:
        print("No association rules for the minimum confidence(Apriori).")
    else:
        print("Final Association rules (Apriori): \n")
        for i, rule in enumerate(association_rules_apriori.iterrows(), start=1):
            antecedent = list(rule[1]['antecedents'])
            consequent = list(rule[1]['consequents'])
```



```python
            consequent = list(rule[1]['consequents'])
            confidence = rule[1]['confidence']
            support = rule[1]['support']
            print(f"Rule {i}: {antecedent} -> {consequent}")
            print(f"Confidence: {confidence}")
            print(f"Support: {support} \n")


    #FP-Growth algorithm
    print("\n   --- FP-GROWTH ALGORITHM ---\n")
    start_time = time.time()
    frequent_itemsets_fpgrowth, association_rules_fpgrowth = apply_fp_growth(transactions, min_support, min_confidence)
    end_time = time.time()
    fpgrowth_time = end_time - start_time
    print(f"Time taken by FP-Growth algorithm: {fpgrowth_time:.6f} seconds\n")

    if association_rules_fpgrowth is None:
        print("No association rules for the minimum confidence(FP-Growth).")
    else:
        print("Final Association rules (FP-Growth): \n")
        for i, rule in enumerate(association_rules_fpgrowth.iterrows(), start=1):
            antecedent = list(rule[1]['antecedents'])
            consequent = list(rule[1]['consequents'])
            confidence = rule[1]['confidence']
            support = rule[1]['support']
            print(f"Rule {i}: {antecedent} -> {consequent}")
            print(f"Confidence: {confidence}")
            print(f"Support: {support} \n")


    #Brute Force algorithm
    print("\n   --- BRUTE FORCE ALGORITHM ---\n")
    start_time = time.time()
    frequent_itemsets, association_rules = find_association_rules(transactions, min_support, min_confidence)
    end_time = time.time()
    bruteforce_time = end_time - start_time
    print(f"Time taken by Brute Force algorithm: {bruteforce_time:.6f} seconds\n")

    if association_rules:
        print("Final Association Rules (Brute Force):")
        for i, rule in enumerate(association_rules, start=1):
            antecedent, consequent, support, confidence = rule
            print(f"Rule {i}: {antecedent} -> {consequent}")
            print(f"Confidence: {confidence :.2f}%")
            print(f"Support: {support :.2f}% \n")
    else:
        print("\nNo association rules for the minimum confidence(Brute Force).")
```

input.py: Reading and displaying the transactions from a dataset.



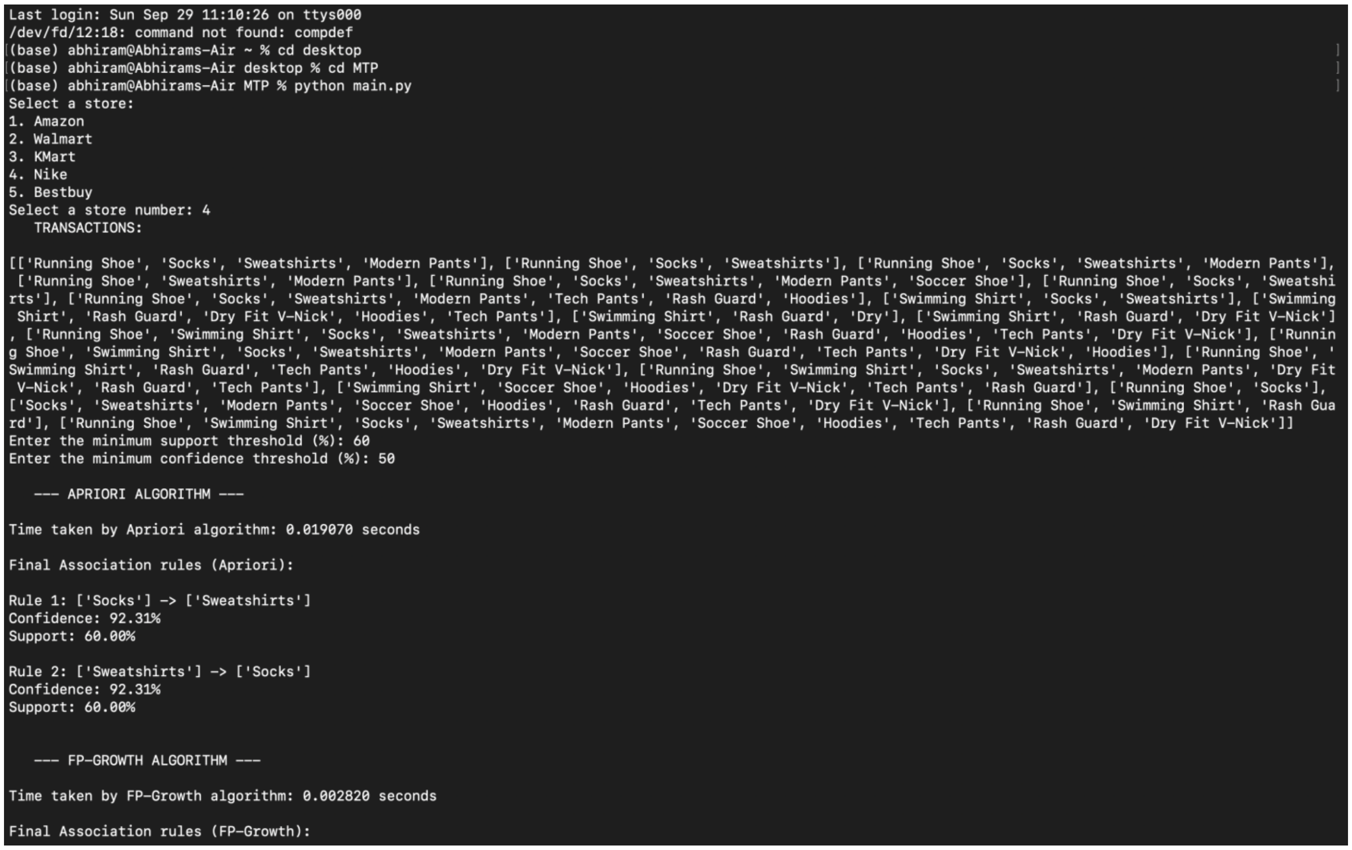
```python
import pandas as pd
import os
from select_file import select_file

def read_transactions(file_path):
    transactions = pd.read_csv(file_path, header=None)
    #print("Transactions: ", transactions)
    return transactions

if __name__ == "__main__":
    selected_file = select_file()
    if selected_file:
        print(f"Selected database: {selected_file}")

        file_path = os.path.join(os.getcwd(), selected_file)
        transactions = read_transactions_from_file(file_path)
        print("Transactions read from file:")
        print(transactions)
```

Results

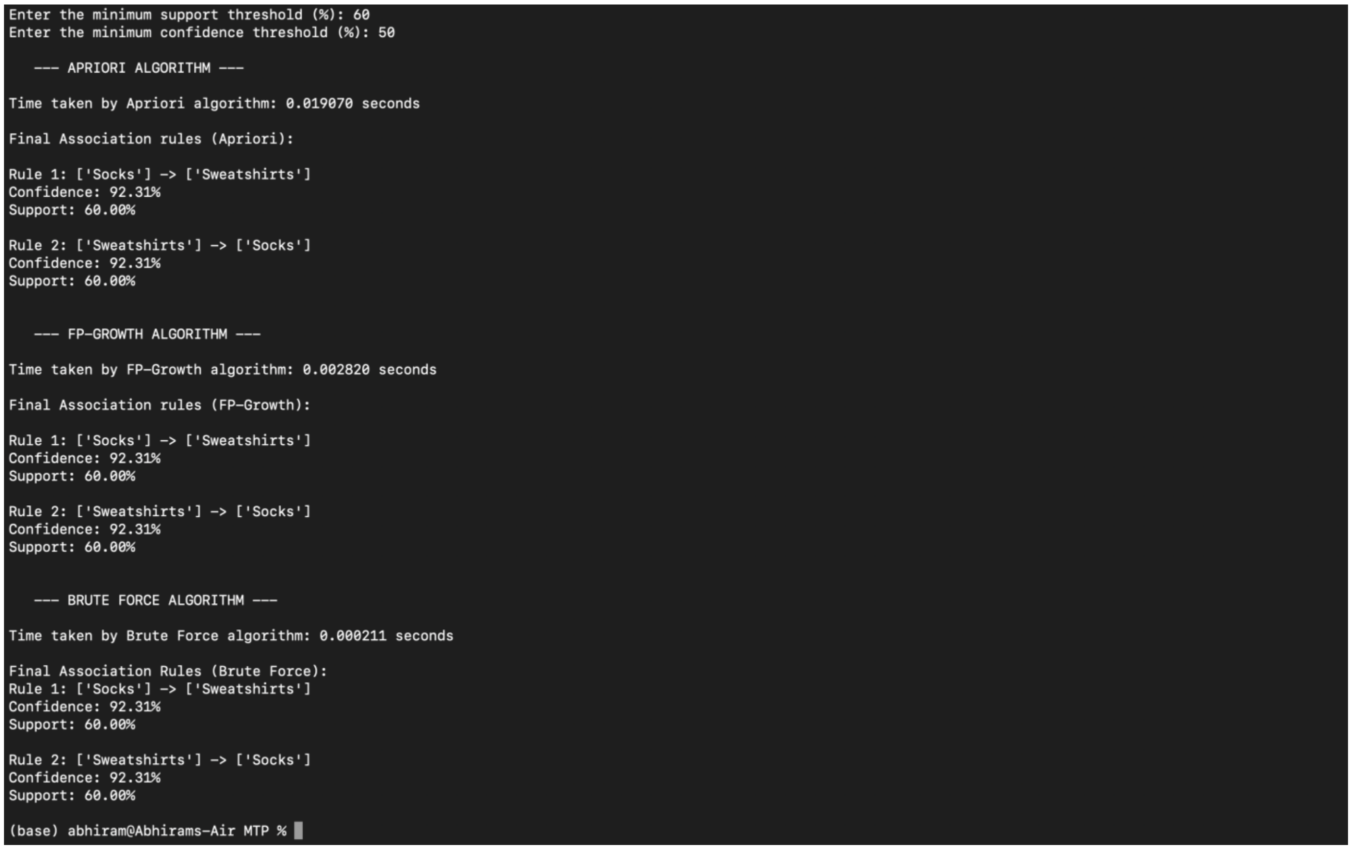
Test Case 1

Support: 60%
Confidence: 50%

Dataset: Nike



```
Last login: Sun Sep 29 11:10:26 on ttys000
/dev/fd/12:18: command not found: compdef
(base) abhiram@Abhirams-Air ~ % cd desktop
(base) abhiram@Abhirams-Air desktop % cd MTP
(base) abhiram@Abhirams-Air MTP % python main.py
Select a store:
1. Amazon
2. Walmart
3. KMart
4. Nike
5. Bestbuy
Select a store number: 4
   TRANSACTIONS:

[['Running Shoe', 'Socks', 'Sweatshirts', 'Modern Pants'], ['Running Shoe', 'Socks', 'Sweatshirts'], ['Running Shoe', 'Socks', 'Sweatshirts', 'Modern Pants'],
 ['Running Shoe', 'Sweatshirts', 'Modern Pants'], ['Running Shoe', 'Socks', 'Sweatshirts', 'Modern Pants', 'Soccer Shoe'], ['Running Shoe', 'Socks', 'Sweatshi
rts'], ['Running Shoe', 'Socks', 'Sweatshirts', 'Modern Pants', 'Tech Pants', 'Rash Guard', 'Hoodies'], ['Swimming Shirt', 'Socks', 'Sweatshirts'], ['Swimming
 Shirt', 'Rash Guard', 'Dry Fit V-Nick', 'Hoodies', 'Tech Pants'], ['Swimming Shirt', 'Rash Guard', 'Dry'], ['Swimming Shirt', 'Rash Guard', 'Dry Fit V-Nick']
, ['Running Shoe', 'Swimming Shirt', 'Socks', 'Sweatshirts', 'Modern Pants', 'Soccer Shoe', 'Rash Guard', 'Hoodies', 'Tech Pants', 'Dry Fit V-Nick'], ['Runnin
g Shoe', 'Swimming Shirt', 'Socks', 'Sweatshirts', 'Modern Pants', 'Soccer Shoe', 'Rash Guard', 'Tech Pants', 'Dry Fit V-Nick', 'Hoodies'], ['Running Shoe', '
Swimming Shirt', 'Rash Guard', 'Tech Pants', 'Hoodies', 'Dry Fit V-Nick'], ['Running Shoe', 'Swimming Shirt', 'Socks', 'Sweatshirts', 'Modern Pants', 'Dry Fit
 V-Nick', 'Rash Guard', 'Tech Pants'], ['Swimming Shirt', 'Soccer Shoe', 'Hoodies', 'Dry Fit V-Nick', 'Tech Pants', 'Rash Guard'], ['Running Shoe', 'Socks'],
['Socks', 'Sweatshirts', 'Modern Pants', 'Soccer Shoe', 'Hoodies', 'Rash Guard', 'Tech Pants', 'Dry Fit V-Nick'], ['Running Shoe', 'Swimming Shirt', 'Rash Gua
rd'], ['Running Shoe', 'Swimming Shirt', 'Socks', 'Sweatshirts', 'Modern Pants', 'Soccer Shoe', 'Hoodies', 'Tech Pants', 'Rash Guard', 'Dry Fit V-Nick']]
Enter the minimum support threshold (%): 60
Enter the minimum confidence threshold (%): 50

   --- APRIORI ALGORITHM ---

Time taken by Apriori algorithm: 0.019070 seconds

Final Association rules (Apriori):

Rule 1: ['Socks'] -> ['Sweatshirts']
Confidence: 92.31%
Support: 60.00%

Rule 2: ['Sweatshirts'] -> ['Socks']
Confidence: 92.31%
Support: 60.00%


   --- FP-GROWTH ALGORITHM ---

Time taken by FP-Growth algorithm: 0.002820 seconds

Final Association rules (FP-Growth):
```

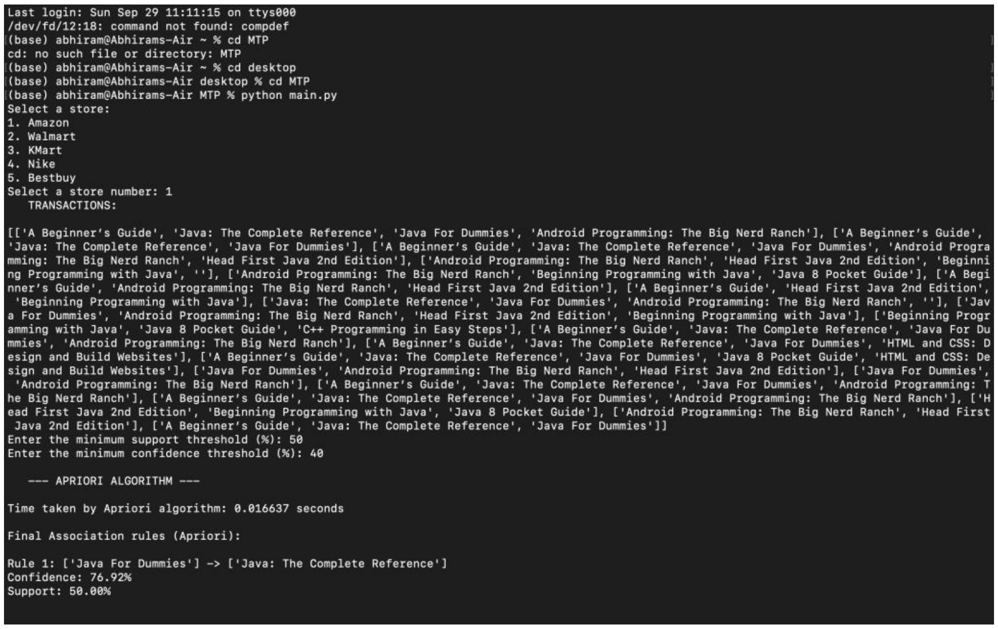


```
Enter the minimum support threshold (%): 60
Enter the minimum confidence threshold (%): 50

   --- APRIORI ALGORITHM ---

Time taken by Apriori algorithm: 0.019070 seconds

Final Association rules (Apriori):

Rule 1: ['Socks'] -> ['Sweatshirts']
Confidence: 92.31%
Support: 60.00%

Rule 2: ['Sweatshirts'] -> ['Socks']
Confidence: 92.31%
Support: 60.00%


   --- FP-GROWTH ALGORITHM ---

Time taken by FP-Growth algorithm: 0.002820 seconds

Final Association rules (FP-Growth):

Rule 1: ['Socks'] -> ['Sweatshirts']
Confidence: 92.31%
Support: 60.00%

Rule 2: ['Sweatshirts'] -> ['Socks']
Confidence: 92.31%
Support: 60.00%


   --- BRUTE FORCE ALGORITHM ---

Time taken by Brute Force algorithm: 0.000211 seconds

Final Association Rules (Brute Force):
Rule 1: ['Socks'] -> ['Sweatshirts']
Confidence: 92.31%
Support: 60.00%

Rule 2: ['Sweatshirts'] -> ['Socks']
Confidence: 92.31%
Support: 60.00%

(base) abhiram@Abhirams-Air MTP %
```

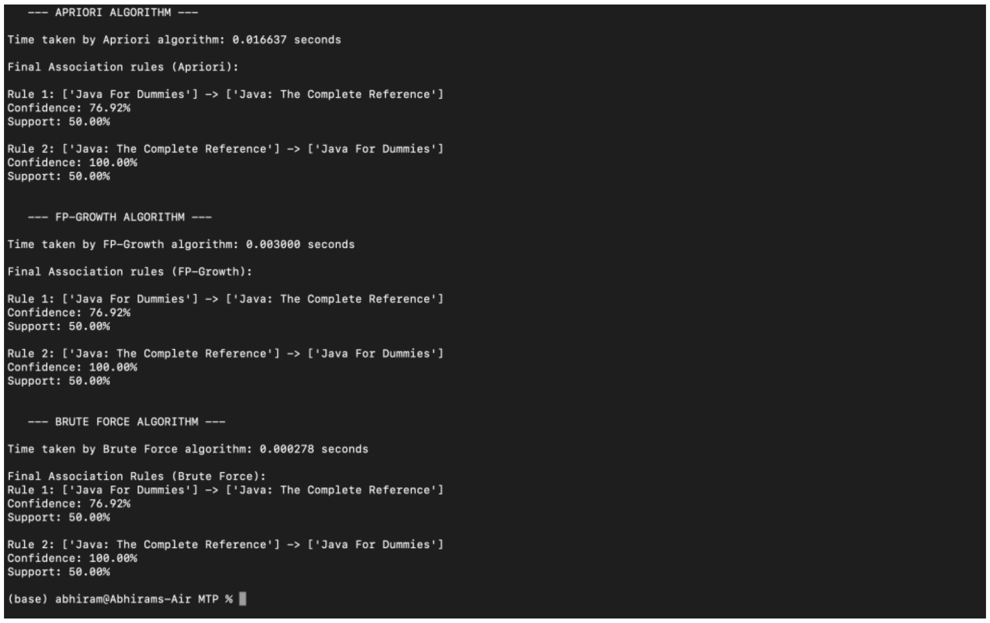
<u>**Test Case 2**</u>

Support: 50%
Confidence: 40%

Dataset: Amazon



```
Last login: Sun Sep 29 11:11:15 on ttys000
/dev/fd/12:18: command not found: compdef
(base) abhiram@Abhirams-Air ~ % cd MTP
cd: no such file or directory: MTP
(base) abhiram@Abhirams-Air ~ % cd desktop
(base) abhiram@Abhirams-Air desktop % cd MTP
(base) abhiram@Abhirams-Air MTP % python main.py
Select a store:
1. Amazon
2. Walmart
3. KMart
4. Nike
5. Bestbuy
Select a store number: 1
   TRANSACTIONS:

[['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Dummies', 'Android Programming: The Big Nerd Ranch'], ['A Beginner's Guide',
'Java: The Complete Reference', 'Java For Dummies'], ['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Dummies', 'Android Progra
mming: The Big Nerd Ranch', 'Head First Java 2nd Edition'], ['Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edition', 'Beginni
ng Programming with Java', ''], ['Android Programming: The Big Nerd Ranch', 'Beginning Programming with Java', 'Java 8 Pocket Guide'], ['A Begi
nner's Guide', 'Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edition'], ['A Beginner's Guide', 'Head First Java 2nd Edition',
 'Beginning Programming with Java'], ['Java: The Complete Reference', 'Java For Dummies', 'Android Programming: The Big Nerd Ranch', ''], ['Jav
a For Dummies', 'Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edition', 'Beginning Programming with Java'], ['Beginning Progr
amming with Java', 'Java 8 Pocket Guide', 'C++ Programming in Easy Steps'], ['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Du
mmies', 'Android Programming: The Big Nerd Ranch'], ['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Dummies', 'HTML and CSS: D
esign and Build Websites'], ['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Dummies', 'Java 8 Pocket Guide', 'HTML and CSS: De
sign and Build Websites'], ['Java For Dummies', 'Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edition'], ['Java For Dummies',
 'Android Programming: The Big Nerd Ranch'], ['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Dummies', 'Android Programming: T
he Big Nerd Ranch'], ['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Dummies', 'Android Programming: The Big Nerd Ranch'], ['H
ead First Java 2nd Edition', 'Beginning Programming with Java', 'Java 8 Pocket Guide'], ['Android Programming: The Big Nerd Ranch', 'Head First
 Java 2nd Edition'], ['A Beginner's Guide', 'Java: The Complete Reference', 'Java For Dummies']]
Enter the minimum support threshold (%): 50
Enter the minimum confidence threshold (%): 40

   --- APRIORI ALGORITHM ---

Time taken by Apriori algorithm: 0.016637 seconds

Final Association rules (Apriori):

Rule 1: ['Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 76.92%
Support: 50.00%
```

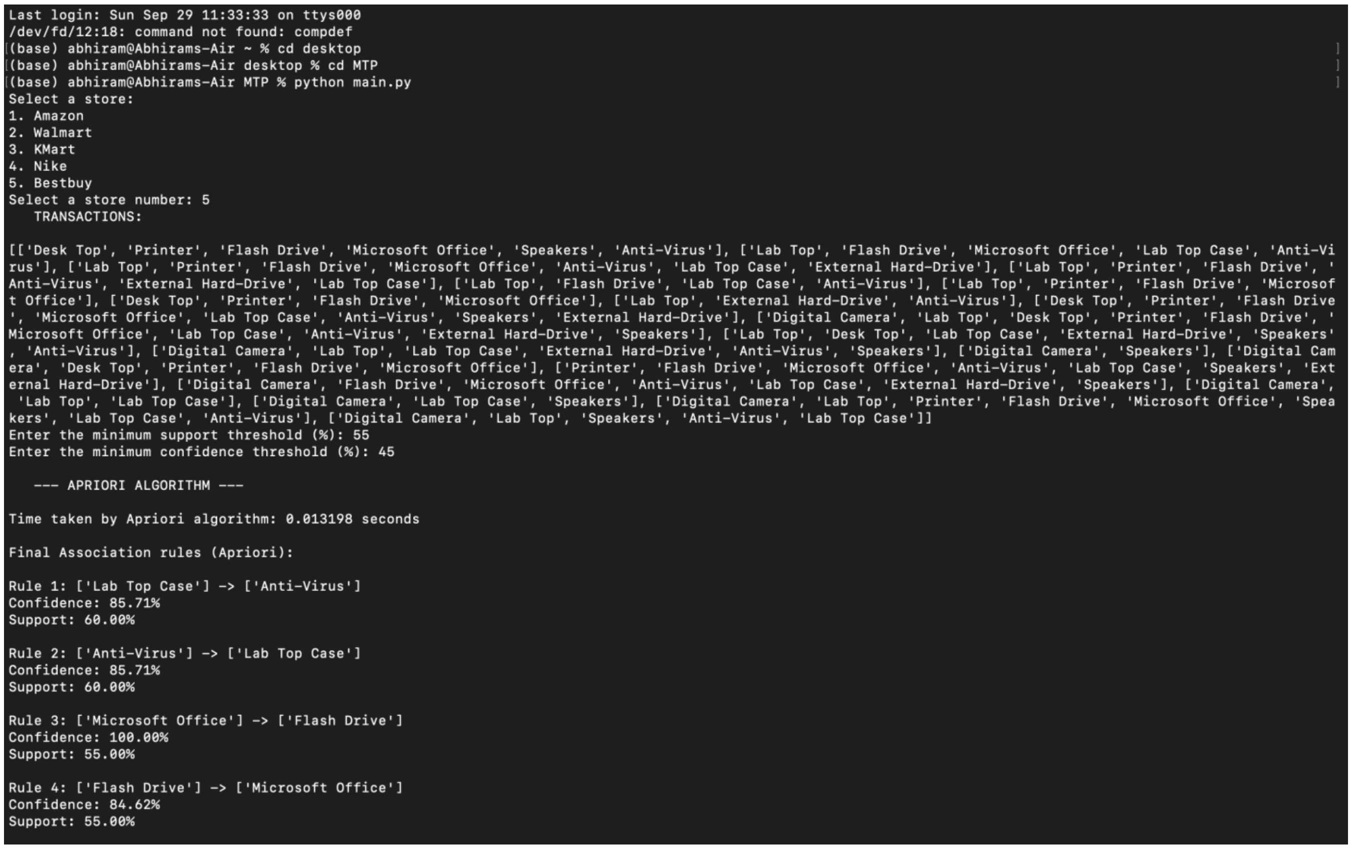


```
   --- APRIORI ALGORITHM ---

Time taken by Apriori algorithm: 0.016637 seconds

Final Association rules (Apriori):

Rule 1: ['Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 76.92%
Support: 50.00%

Rule 2: ['Java: The Complete Reference'] -> ['Java For Dummies']
Confidence: 100.00%
Support: 50.00%


   --- FP-GROWTH ALGORITHM ---

Time taken by FP-Growth algorithm: 0.003000 seconds

Final Association rules (FP-Growth):

Rule 1: ['Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 76.92%
Support: 50.00%

Rule 2: ['Java: The Complete Reference'] -> ['Java For Dummies']
Confidence: 100.00%
Support: 50.00%


   --- BRUTE FORCE ALGORITHM ---

Time taken by Brute Force algorithm: 0.000278 seconds

Final Association Rules (Brute Force):
Rule 1: ['Java For Dummies'] -> ['Java: The Complete Reference']
Confidence: 76.92%
Support: 50.00%

Rule 2: ['Java: The Complete Reference'] -> ['Java For Dummies']
Confidence: 100.00%
Support: 50.00%

(base) abhiram@Abhirams-Air MTP %
```

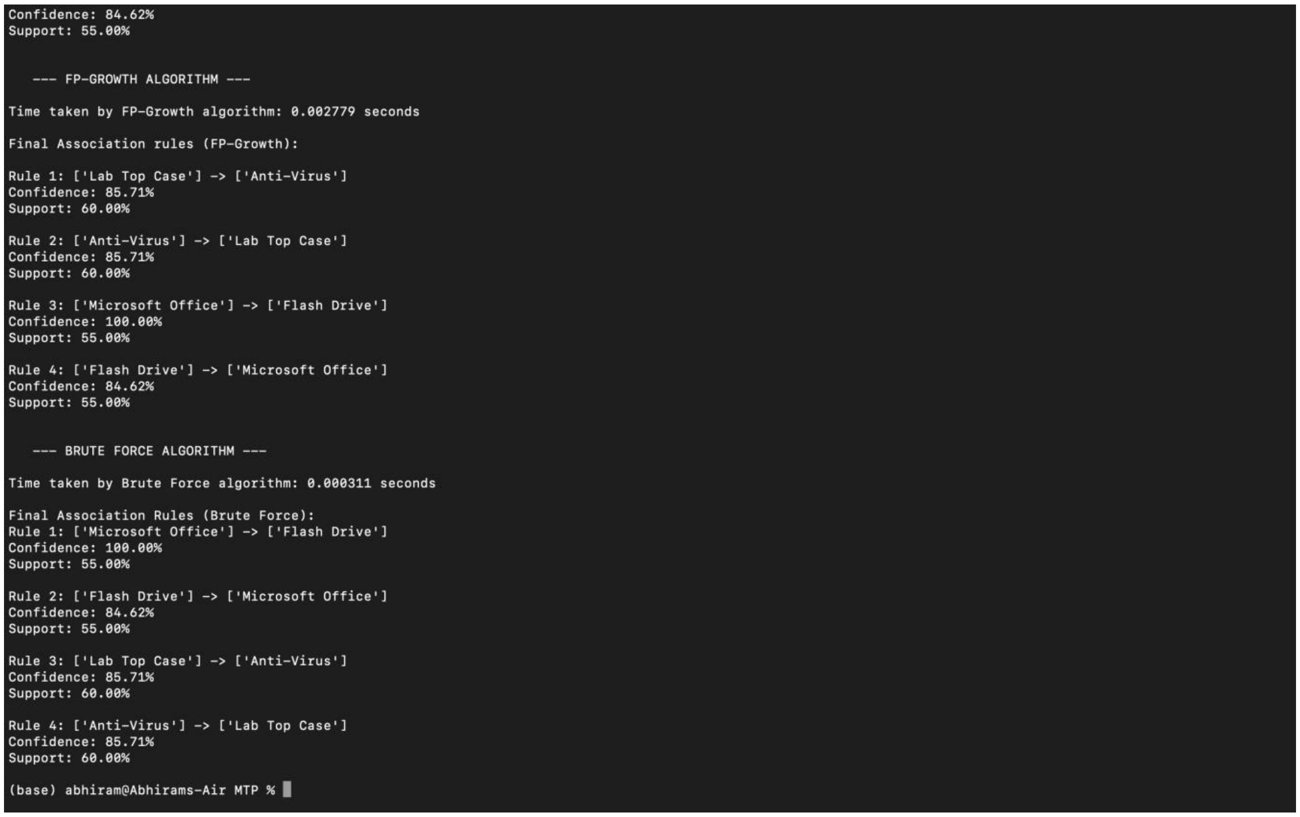
<u>Test Case 3</u>

Support: 55%
Confidence: 45%

Dataset: BestBuy



```
Last login: Sun Sep 29 11:33:33 on ttys000
/dev/fd/12:18: command not found: compdef
(base) abhiram@Abhirams-Air ~ % cd desktop
(base) abhiram@Abhirams-Air desktop % cd MTP
(base) abhiram@Abhirams-Air MTP % python main.py
Select a store:
1. Amazon
2. Walmart
3. KMart
4. Nike
5. Bestbuy
Select a store number: 5
   TRANSACTIONS:

[['Desk Top', 'Printer', 'Flash Drive', 'Microsoft Office', 'Speakers', 'Anti-Virus'], ['Lab Top', 'Flash Drive', 'Microsoft Office', 'Lab Top Case', 'Anti-Vi
rus'], ['Lab Top', 'Printer', 'Flash Drive', 'Microsoft Office', 'Anti-Virus', 'Lab Top Case', 'External Hard-Drive'], ['Lab Top', 'Printer', 'Flash Drive', '
Anti-Virus', 'External Hard-Drive', 'Lab Top Case'], ['Lab Top', 'Flash Drive', 'Lab Top Case', 'Anti-Virus'], ['Lab Top', 'Printer', 'Flash Drive', 'Microsof
t Office'], ['Desk Top', 'Printer', 'Flash Drive', 'Microsoft Office'], ['Lab Top', 'External Hard-Drive', 'Anti-Virus'], ['Desk Top', 'Printer', 'Flash Drive
', 'Microsoft Office', 'Lab Top Case', 'Anti-Virus', 'Speakers', 'External Hard-Drive'], ['Digital Camera', 'Lab Top', 'Desk Top', 'Printer', 'Flash Drive', '
Microsoft Office', 'Lab Top Case', 'Anti-Virus', 'External Hard-Drive', 'Speakers'], ['Lab Top', 'Desk Top', 'Lab Top Case', 'External Hard-Drive', 'Speakers'
, 'Anti-Virus'], ['Digital Camera', 'Lab Top', 'Lab Top Case', 'External Hard-Drive', 'Anti-Virus', 'Speakers'], ['Digital Camera', 'Speakers'], ['Digital Cam
era', 'Desk Top', 'Printer', 'Flash Drive', 'Microsoft Office'], ['Printer', 'Flash Drive', 'Microsoft Office', 'Anti-Virus', 'Lab Top Case', 'Speakers', 'Ext
ernal Hard-Drive'], ['Digital Camera', 'Flash Drive', 'Microsoft Office', 'Anti-Virus', 'Lab Top Case', 'External Hard-Drive', 'Speakers'], ['Digital Camera',
 'Lab Top', 'Lab Top Case'], ['Digital Camera', 'Lab Top Case', 'Speakers'], ['Digital Camera', 'Lab Top', 'Printer', 'Flash Drive', 'Microsoft Office', 'Spea
kers', 'Lab Top Case', 'Anti-Virus'], ['Digital Camera', 'Lab Top', 'Speakers', 'Anti-Virus', 'Lab Top Case']]
Enter the minimum support threshold (%): 55
Enter the minimum confidence threshold (%): 45

   --- APRIORI ALGORITHM ---

Time taken by Apriori algorithm: 0.013198 seconds

Final Association rules (Apriori):

Rule 1: ['Lab Top Case'] -> ['Anti-Virus']
Confidence: 85.71%
Support: 60.00%

Rule 2: ['Anti-Virus'] -> ['Lab Top Case']
Confidence: 85.71%
Support: 60.00%

Rule 3: ['Microsoft Office'] -> ['Flash Drive']
Confidence: 100.00%
Support: 55.00%

Rule 4: ['Flash Drive'] -> ['Microsoft Office']
Confidence: 84.62%
Support: 55.00%
```



```
Confidence: 84.62%
Support: 55.00%


   --- FP-GROWTH ALGORITHM ---

Time taken by FP-Growth algorithm: 0.002779 seconds

Final Association rules (FP-Growth):

Rule 1: ['Lab Top Case'] -> ['Anti-Virus']
Confidence: 85.71%
Support: 60.00%

Rule 2: ['Anti-Virus'] -> ['Lab Top Case']
Confidence: 85.71%
Support: 60.00%

Rule 3: ['Microsoft Office'] -> ['Flash Drive']
Confidence: 100.00%
Support: 55.00%

Rule 4: ['Flash Drive'] -> ['Microsoft Office']
Confidence: 84.62%
Support: 55.00%


   --- BRUTE FORCE ALGORITHM ---

Time taken by Brute Force algorithm: 0.000311 seconds

Final Association Rules (Brute Force):
Rule 1: ['Microsoft Office'] -> ['Flash Drive']
Confidence: 100.00%
Support: 55.00%

Rule 2: ['Flash Drive'] -> ['Microsoft Office']
Confidence: 84.62%
Support: 55.00%

Rule 3: ['Lab Top Case'] -> ['Anti-Virus']
Confidence: 85.71%
Support: 60.00%

Rule 4: ['Anti-Virus'] -> ['Lab Top Case']
Confidence: 85.71%
Support: 60.00%

(base) abhiram@Abhirams-Air MTP % 
```

Conclusion

Based on the test case results, all three algorithms; Brute Force, Apriori, and FP-Growth produce identical outcomes.

The Brute Force algorithm seems to be the fastest for higher support and confidence levels. However, for lower support and confidence levels FP-Tree Algorithm seems to be the fastest.