NAME: Arpit Shrotriya
NJIT UCID: as4893
Email Address: arpit.shrotriya5945@gmail.com
Professor: Yasser Abduallah
CS 634 101 Data Mining

Mid Term Project Report

Implementation and Code Usage

Comparison of Data Mining algorithms (Brute Force, Apriori, FP-Growth) on Retail Transactions Dataset

Abstract: In This Project, I implemented and compared three different Data Mining Algorithms: Brute Force, Apriori, and FP-Growth on Retail Transactions Dataset. The Purpose was to understand and evaluate the efficiency and effectiveness of each algorithm in identifying Frequent Itemsets and Association Rules within Transactional Datasets. By applying each method to the same dataset, I was able to analyze their accuracy, differences in results, and overall efficiency in terms of speed. This comparison helps identify the most suitable algorithm depending on the nature of the data and the performance requirements, providing insights for real-world applications like market basket analysis.

Introduction:

Data mining is an effective technique for revealing hidden patterns and meaningful relationships within extensive datasets. In this project, I focused on comparing three different association rule mining algorithms: Brute Force, Apriori, and FP-Growth, and applied them to a retail dataset. These algorithms are commonly used for association rule mining, which helps identify relationships between items that frequently occur together in transactions, similar to products purchased together in a store.

The Apriori and FP-Growth algorithms are well-known for their efficiency in finding frequent itemsets and generating association rules, but their performance can vary depending on the nature of the dataset. To understand these differences, I implemented a brute force approach as a baseline and compared it against the Apriori and FP-Growth algorithms.

Key steps in this project included:

- Loading transaction data from a CSV file representing retail purchases.
- Collecting user input for minimum support and confidence thresholds to filter frequent itemsets.
- Preprocessing the dataset to ensure that items are consistent and unique across all transactions.

- Initializing the frequent itemsets using a brute force method and calculating support values for all possible combinations of items.
- Implementing the Apriori and FP-Growth algorithms using Python's `mlxtend` library and comparing their results with the brute force method.
- Evaluating the runtime performance and accuracy of each method by comparing frequent itemsets and analyzing computational efficiency.

Core Concepts and Principles:

1. **Frequent Itemset Mining**:
    - The goal is to identify combinations of items that frequently co-occur in transactions. This helps in understanding relationships between items, which is useful in applications like market basket analysis.
2. **User-Specified Thresholds**:
    - The algorithm accepts user input for minimum support and minimum confidence. These thresholds help in filtering out non-frequent itemsets and generating significant rules, allowing customization based on the application context.
3. **Brute Force Method:**
    - A naive approach where all possible combinations of items are generated and evaluated. This method is computationally intensive but helps as a baseline to understand the efficiency of more sophisticated algorithms.
4. **Apriori Algorithm**:
    - Apriori is an iterative algorithm used for frequent itemset mining and association rule learning. It generates larger itemsets from smaller frequent ones and uses a support threshold to filter out non-frequent itemsets. It operates by progressively growing itemsets and is computationally expensive because of candidate generation.
5. **FP-Growth Algorithm**:
    - FP-Growth is a more efficient frequent itemset mining method compared to Apriori. It uses a tree structure called an FP-tree to compress the data, which reduces the need for candidate generation. It finds frequent itemsets by recursively exploring conditional patterns.
6. **Association Rules**:
    - Once frequent itemsets are identified, association rules are generated to show relationships between items. Rules consist of antecedents and consequent, with a confidence level that indicates the likelihood of finding the consequent given the antecedent.

Project Workflow:

1. **Data Preparation**:
    - Load transaction data from a CSV file.
    - Preprocess data to extract transactions as lists of items.
2. **User Input**:

- Collect user-defined thresholds for minimum support and confidence levels.

3. **Brute Force Implementation**:
    - Use a brute force approach to generate and evaluate all possible itemsets.
    - Calculate support values for each itemset.
    - Identify frequent itemsets based on the support threshold.
    - Generate Association Rules based on Frequent Itemsets.
4. **Apriori Algorithm**:
    - Encode transactions for compatibility with Apriori.
    - Apply the Apriori algorithm to find frequent itemsets and association rules.
    - Extract and print results.
5. **FP-Growth Algorithm**:
    - Encode transactions for compatibility with FP-Growth.
    - Use the FP-Growth algorithm to identify frequent itemsets and association rules.
    - Extract and print results.
6. **Comparison of Results**:
    - Compare frequent itemsets obtained by Brute Force, Apriori, and FP-Growth.
    - Evaluate and present differences in identified frequent itemsets.
7. **Performance Evaluation**:
    - Record and compare the execution times of all three methods to evaluate computational efficiency.

Screenshots

Here are the csv files

Amazon Transactions csv file

	A	B
1	Transaction_ID	Transaction
2	Trans1	A Beginner's Guide, Head First Java 2nd Edition, Effective Java (2nd Edition), Android Programming: The Big Nerd Ranch, Java: The Complete Reference, HTML and CSS: Design and Build Websites
3	Trans2	Java For Dummies, Java 8 Pocket Guide, C++ Programming in Easy Steps, Beginning Programming with Java, Java: The Complete Reference
4	Trans3	Android Programming: The Big Nerd Ranch, A Beginner's Guide, HTML and CSS: Design and Build Websites, Effective Java (2nd Edition)
5	Trans4	Java For Dummies, Head First Java 2nd Edition, Beginning Programming with Java, Java 8 Pocket Guide, A Beginner's Guide, Java: The Complete Reference
6	Trans5	HTML and CSS: Design and Build Websites, Beginning Programming with Java, C++ Programming in Easy Steps
7	Trans6	Beginning Programming with Java, Java 8 Pocket Guide, Java: The Complete Reference, C++ Programming in Easy Steps
8	Trans7	A Beginner's Guide, C++ Programming in Easy Steps, Head First Java 2nd Edition, Java: The Complete Reference
9	Trans8	Beginning Programming with Java, HTML and CSS: Design and Build Websites, Effective Java (2nd Edition), Android Programming: The Big Nerd Ranch, C++ Programming in Easy Steps
10	Trans9	Head First Java 2nd Edition, C++ Programming in Easy Steps
11	Trans10	Java: The Complete Reference, Effective Java (2nd Edition)
12	Trans11	Java 8 Pocket Guide, C++ Programming in Easy Steps, Java: The Complete Reference, Effective Java (2nd Edition)
13	Trans12	Java: The Complete Reference, C++ Programming in Easy Steps, Head First Java 2nd Edition, A Beginner's Guide
14	Trans13	C++ Programming in Easy Steps, Beginning Programming with Java
15	Trans14	Java 8 Pocket Guide, Effective Java (2nd Edition)
16	Trans15	Java: The Complete Reference, Java 8 Pocket Guide
17	Trans16	Effective Java (2nd Edition), C++ Programming in Easy Steps, Java For Dummies, A Beginner's Guide, Java 8 Pocket Guide
18	Trans17	Head First Java 2nd Edition, Android Programming: The Big Nerd Ranch, A Beginner's Guide, Beginning Programming with Java, Java: The Complete Reference
19	Trans18	Java: The Complete Reference, Java For Dummies, Effective Java (2nd Edition), C++ Programming in Easy Steps, Java 8 Pocket Guide
20	Trans19	Effective Java (2nd Edition), A Beginner's Guide, HTML and CSS: Design and Build Websites, Java 8 Pocket Guide
21	Trans20	Effective Java (2nd Edition), Beginning Programming with Java, Java: The Complete Reference, A Beginner's Guide, Android Programming: The Big Nerd Ranch, Java For Dummies
22	Trans21	Effective Java (2nd Edition), Android Programming: The Big Nerd Ranch
23	Trans22	Beginning Programming with Java, Head First Java 2nd Edition
24	Trans23	Head First Java 2nd Edition, Effective Java (2nd Edition)
25	Trans24	Android Programming: The Big Nerd Ranch, Java 8 Pocket Guide, A Beginner's Guide, Head First Java 2nd Edition
26	Trans25	HTML and CSS: Design and Build Websites, Android Programming: The Big Nerd Ranch, Java 8 Pocket Guide, Java For Dummies, A Beginner's Guide
27		

<u>K-Mart Transactions csv file</u>

	A	B	C
1	Transaction_ID	Transaction	
2	Trans1	Embroidered Bedspread, Bedspreads, Kids Bedding, Bedding Collections, Decorative Pillows	
3	Trans2	Bed Skirts, Kids Bedding, Quilts, Sheets	
4	Trans3	Quilts, Embroidered Bedspread, Towels, Kids Bedding, Decorative Pillows, Bedspreads	
5	Trans4	Embroidered Bedspread, Kids Bedding	
6	Trans5	Bed Skirts, Towels, Bedspreads, Shams, Kids Bedding	
7	Trans6	Bed Skirts, Bedding Collections, Kids Bedding	
8	Trans7	Bedding Collections, Decorative Pillows, Embroidered Bedspread, Bedspreads, Sheets, Quilts	
9	Trans8	Bedspreads, Bedding Collections, Shams, Towels, Embroidered Bedspread	
10	Trans9	Sheets, Shams, Bedding Collections, Quilts, Towels	
11	Trans10	Embroidered Bedspread, Decorative Pillows, Shams, Kids Bedding	
12	Trans11	Kids Bedding, Decorative Pillows, Towels, Bedspreads	
13	Trans12	Bed Skirts, Bedding Collections	
14	Trans13	Quilts, Bedding Collections, Shams, Bed Skirts, Bedspreads	
15	Trans14	Towels, Bed Skirts, Embroidered Bedspread, Decorative Pillows	
16	Trans15	Sheets, Bedspreads, Towels, Bedding Collections, Shams	
17	Trans16	Shams, Quilts, Kids Bedding, Decorative Pillows	
18	Trans17	Kids Bedding, Quilts, Towels, Bed Skirts, Sheets	
19	Trans18	Sheets, Decorative Pillows	
20	Trans19	Sheets, Quilts, Shams, Embroidered Bedspread, Bedding Collections	
21	Trans20	Towels, Embroidered Bedspread, Bed Skirts, Shams, Sheets, Quilts	
22	Trans21	Towels, Kids Bedding	
23	Trans22	Bedding Collections, Sheets, Bedspreads, Kids Bedding	
24	Trans23	Towels, Embroidered Bedspread, Shams, Bed Skirts, Bedding Collections	
25	Trans24	Bedding Collections, Sheets, Shams, Bedspreads, Decorative Pillows, Bed Skirts	
26	Trans25	Shams, Quilts, Bedding Collections	
27			

<u>Shoprite Transaction csv file</u>

	A	B	C
1	Transaction_ID	Transaction	
2	Trans1	Soccer Shoe, Hoodies, Socks, Running Shoe	
3	Trans2	Tech Pants, Running Shoe, Dry Fit V-Nick	
4	Trans3	Soccer Shoe, Tech Pants, Dry Fit V-Nick, Modern Pants, Running Shoe, Swimming Shirt	
5	Trans4	Soccer Shoe, Sweatshirts, Modern Pants	
6	Trans5	Swimming Shirt, Dry Fit V-Nick, Running Shoe, Soccer Shoe	
7	Trans6	Soccer Shoe, Sweatshirts	
8	Trans7	Hoodies, Socks	
9	Trans8	Sweatshirts, Dry Fit V-Nick, Hoodies, Soccer Shoe	
10	Trans9	Tech Pants, Socks, Soccer Shoe, Hoodies, Running Shoe	
11	Trans10	Swimming Shirt, Tech Pants	
12	Trans11	Soccer Shoe, Socks, Tech Pants, Sweatshirts, Modern Pants, Running Shoe	
13	Trans12	Tech Pants, Swimming Shirt, Soccer Shoe, Rash Guard, Running Shoe, Dry Fit V-Nick	
14	Trans13	Running Shoe, Socks, Rash Guard, Sweatshirts, Modern Pants, Soccer Shoe	
15	Trans14	Hoodies, Tech Pants, Modern Pants, Running Shoe, Swimming Shirt, Soccer Shoe	
16	Trans15	Hoodies, Running Shoe, Modern Pants, Rash Guard	
17	Trans16	Modern Pants, Sweatshirts, Tech Pants, Soccer Shoe, Swimming Shirt, Rash Guard	
18	Trans17	Dry Fit V-Nick, Soccer Shoe, Socks	
19	Trans18	Running Shoe, Dry Fit V-Nick, Hoodies, Modern Pants, Socks, Swimming Shirt	
20	Trans19	Hoodies, Running Shoe, Dry Fit V-Nick, Swimming Shirt	
21	Trans20	Tech Pants, Swimming Shirt	
22	Trans21	Soccer Shoe, Tech Pants, Swimming Shirt	
23	Trans22	Socks, Dry Fit V-Nick, Modern Pants, Swimming Shirt, Tech Pants	
24	Trans23	Running Shoe, Sweatshirts	
25	Trans24	Sweatshirts, Modern Pants, Tech Pants	
26	Trans25	Swimming Shirt, Soccer Shoe, Running Shoe	
27			

Target Transaction csv file

	A	B	C
1	Transaction_ID	Transaction	
2	Trans1	Shampoo, Mouthwash, Conditioner, Body Wash	
3	Trans2	Conditioner, Shampoo, Soap, Mouthwash, Lotion	
4	Trans3	Mouthwash, Body Wash, Soap, Conditioner, Deodorant	
5	Trans4	Body Wash, Shampoo, Face Wash, Mouthwash	
6	Trans5	Body Wash, Toothpaste, Mouthwash, Shampoo, Soap	
7	Trans6	Mouthwash, Soap, Shampoo, Face Wash, Lotion, Conditioner	
8	Trans7	Hand Sanitizer, Body Wash, Conditioner, Toothpaste, Shampoo	
9	Trans8	Face Wash, Body Wash, Conditioner, Shampoo, Lotion, Mouthwash	
10	Trans9	Soap, Toothpaste, Conditioner	
11	Trans10	Hand Sanitizer, Shampoo, Lotion	
12	Trans11	Deodorant, Conditioner, Shampoo	
13	Trans12	Toothpaste, Face Wash, Mouthwash, Body Wash, Deodorant, Hand Sanitizer	
14	Trans13	Face Wash, Soap, Lotion, Mouthwash, Body Wash	
15	Trans14	Toothpaste, Shampoo, Lotion, Body Wash, Deodorant	
16	Trans15	Soap, Hand Sanitizer	
17	Trans16	Face Wash, Body Wash, Lotion, Conditioner	
18	Trans17	Soap, Mouthwash, Deodorant, Hand Sanitizer, Toothpaste, Shampoo	
19	Trans18	Shampoo, Toothpaste, Body Wash	
20	Trans19	Shampoo, Soap, Lotion	
21	Trans20	Lotion, Conditioner	
22	Trans21	Hand Sanitizer, Soap, Conditioner, Face Wash	
23	Trans22	Mouthwash, Hand Sanitizer, Toothpaste	
24	Trans23	Toothpaste, Shampoo	
25	Trans24	Toothpaste, Conditioner, Soap	
26	Trans25	Lotion, Face Wash, Shampoo, Body Wash	
27			

Walmart Transaction csv file

	A	B	C
1	Transaction_ID	Transaction	
2	Trans1	Microsoft Office, Speakers, Desk Top, Anti-Virus, Flash Drive, Lab Top	
3	Trans2	Speakers, Lab Top Case, Digital Camera, Anti-Virus, Lab Top, Printer	
4	Trans3	Lab Top, Speakers, External Hard-Drive, Desk Top	
5	Trans4	Printer, Digital Camera, Microsoft Office, Lab Top	
6	Trans5	Lab Top, Lab Top Case	
7	Trans6	Desk Top, External Hard-Drive, Flash Drive, Lab Top, Digital Camera	
8	Trans7	Digital Camera, Speakers, Microsoft Office, Lab Top	
9	Trans8	Digital Camera, Desk Top	
10	Trans9	Flash Drive, Microsoft Office, Lab Top Case	
11	Trans10	Flash Drive, Speakers	
12	Trans11	Digital Camera, Lab Top, Lab Top Case	
13	Trans12	Printer, Digital Camera	
14	Trans13	Microsoft Office, Anti-Virus, Lab Top, Flash Drive, Digital Camera, External Hard-Drive	
15	Trans14	Lab Top, Printer, Microsoft Office, Flash Drive	
16	Trans15	Flash Drive, Lab Top Case, Lab Top, Speakers, Microsoft Office, Printer	
17	Trans16	Lab Top, Flash Drive, External Hard-Drive, Lab Top Case, Speakers, Microsoft Office	
18	Trans17	Desk Top, Printer	
19	Trans18	Lab Top Case, Anti-Virus, Flash Drive, Desk Top	
20	Trans19	Lab Top Case, Speakers, Digital Camera	
21	Trans20	Anti-Virus, Lab Top Case, External Hard-Drive, Flash Drive, Microsoft Office	
22	Trans21	Lab Top, Printer, Desk Top	
23	Trans22	Lab Top, Printer	
24	Trans23	Lab Top, Anti-Virus	
25	Trans24	Lab Top Case, Desk Top, External Hard-Drive, Speakers, Printer	
26	Trans25	Anti-Virus, Lab Top, Desk Top	
27			

Here is the python code

Prompts to choose which store dataset you want, and specify minimum support and minimum confidence to generate the association rules.

```
[179] import time
      import itertools
      import pandas as pd
      from collections import defaultdict
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically
  and should_run_async(code)
```

```
n = int(input("Select the Supermarket: \n1.Amazon \n2.K-Mart \n3.Shoprite \n4.Target \n5.Walmart \n[1-5]"))
min_confidence = float(input("Enter Minimum Confidence (Enter Float Value Between 0-1)"))
min_support = float(input("Enter Minimum Support (Enter Float Value Between 0-1)"))
if (n==1):
  df = pd.read_csv("Amazon_transactions_final.csv")
elif (n==2):
  df = pd.read_csv("K-Mart_transactions_final.csv")
elif (n==3):
  df = pd.read_csv("Shoprite_transactions_final.csv")
elif (n==4):
  df = pd.read_csv("Target_transactions_final.csv")
elif (n==5):
  df = pd.read_csv("Walmart_transactions_final.csv")
```

```
transactions = df['Transaction'].apply(lambda x: x.split(', ')).tolist()
all_items = sorted(set(item for transaction in transactions for item in transaction))
#print(transactions)
#print(len(transactions))
#print(all_items)
```

Created User defined Function to calculate support and find Frequent Itemsets to implement Brute Force Approach

```
[182] # Brute Force Method

      def calculate_support(itemset, transactions):
        count = 0
        for transaction in transactions:
          if set(itemset).issubset(set(transaction)):
            count += 1
        return count / len(transactions)
```

```
def find_frequent_itemsets(transactions, all_items, min_support):
  k = 1
  frequent_itemsets = []
  candidate_itemsets = all_items

  while candidate_itemsets:
    print(f'Generating {k}-itemsets')
    current_frequent_itemsets = []
    if k==1:
      candidate_itemsets = [[item] for item in candidate_itemsets]
    else:
      candidate_itemsets = [list(itemset) for itemset in itertools.combinations(all_items, k)]

    for itemset in candidate_itemsets:
      support = calculate_support(itemset, transactions)
      if support >= min_support:
        current_frequent_itemsets.append((itemset, support))

    if not current_frequent_itemsets:
      break

    frequent_itemsets.extend(current_frequent_itemsets)
    k += 1

  return frequent_itemsets
```

Using Resultant Frequent Itemsets to generate Association Rules with minimum confidence

```
[184] def generate_association_rules(frequent_itemsets, transactions, min_confidence):
        rules = []
        for itemset, support in frequent_itemsets:
          if len(itemset) < 2:
            continue
          for i in range(1, len(itemset)):
            for antecedent in itertools.combinations(itemset, i):
              consequent = list(set(itemset) - set(antecedent))
              antecedent_support = calculate_support(antecedent, transactions)
              if antecedent_support > 0:
                confidence = support / antecedent_support
                if confidence >= min_confidence:
                  rules.append((antecedent, consequent, confidence))

        return rules
```

Initiating Brute Force Approach and Noting Time Of Execution

```
[209] # Brute Force Method

      start_time = time.time()
      frequent_itemsets = find_frequent_itemsets(transactions, all_items, min_support)
      brute_force_time = time.time() - start_time

      print(f"Brute Force Time : {brute_force_time}")
      frequent_itemsets_dict = {tuple(itemset): support for itemset, support in frequent_itemsets}
      #print(frequent_itemsets)
      for itemset, support in frequent_itemsets:
        print(f"Itemset: {itemset}, Support: {support}")
```

Generating Association Rules for brute force

```
[210] # Generating Association Rules for Brute Force
      association_rules_brute_force = generate_association_rules(frequent_itemsets, transactions, min_confidence)
      association_rules_brute_force_set = set((frozenset(antecedent), frozenset(consequent), confidence) for antecedent, consequent, confidence in associat
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically
  and should_run_async(code)
```

```
[211] print("\nAssociation Rules (Brute Force): ")
      for antecedent, consequent, confidence in association_rules_brute_force_set:
        print(f"Rule: {antecedent} -> {consequent}, confidence: {confidence:.2f}")
```

```
Association Rules (Brute Force):
Rule: frozenset({'Bedspreads'}) -> frozenset({'Bedding Collections'}), confidence: 0.70
Rule: frozenset({'Sheets'}) -> frozenset({'Bedding Collections'}), confidence: 0.60
Rule: frozenset({'Quilts'}) -> frozenset({'Shams'}), confidence: 0.60
Rule: frozenset({'Quilts'}) -> frozenset({'Sheets'}), confidence: 0.60
Rule: frozenset({'Sheets'}) -> frozenset({'Quilts'}), confidence: 0.60
Rule: frozenset({'Bedding Collections'}) -> frozenset({'Shams'}), confidence: 0.62
Rule: frozenset({'Shams'}) -> frozenset({'Bedding Collections'}), confidence: 0.67
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically
  and should_run_async(code)
```

Using *mlxtend* library to import predefined functions for Apriori Algorithm and calculating frequent itemsets and generating association rules for the same

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

# Assuming transactions and min_support, min_confidence are defined earlier in the code
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Run Apriori Algorithm
start_time = time.time()
frequent_itemsets_apriori = apriori(df_encoded, min_support=min_support, use_colnames=True)
apriori_time = time.time() - start_time

print(f"Apriori Time : {apriori_time}")

# Check if any frequent itemsets were found
if frequent_itemsets_apriori.empty:
    print("No frequent itemsets were found. Exiting.")
else:
    print(frequent_itemsets_apriori)

    # Generate association rules if frequent itemsets exist
    association_rules_apriori = association_rules(frequent_itemsets_apriori, metric="confidence", min_threshold=min_confidence)
    association_rules_apriori_set = set(
        (frozenset(antecedent), frozenset(consequent), confidence)
        for antecedent, consequent, confidence in zip(
            association_rules_apriori['antecedents'],
            association_rules_apriori['consequents'],
            association_rules_apriori['confidence']
        )
    )
    print("\nAssociation Rules (Apriori):")
    print(association_rules_apriori[['antecedents', 'consequents', 'confidence']])
```

Using *mlxtend* library to import predefined functions for FP Growth Algorithm and calculating frequent itemsets and generating association rules for the same

```python
[192] from mlxtend.frequent_patterns import fpgrowth, association_rules
import pandas as pd
import time

# Assuming df_encoded, min_support, and min_confidence are defined earlier in the code

# Run FP-Growth Algorithm
start_time = time.time()
frequent_itemsets_fp = fpgrowth(df_encoded, min_support=min_support, use_colnames=True)
fp_growth_time = time.time() - start_time

print(f"\nFP-Growth Time : {fp_growth_time}")

# Check if any frequent itemsets were found
if frequent_itemsets_fp.empty:
    print("No frequent itemsets were found. Exiting.")
else:
    print("\nFrequent Itemsets (FP-Growth):")
    print(frequent_itemsets_fp)

    # Create dictionary of frequent itemsets
    frequent_itemsets_fp_dict = {tuple(row['itemsets']): row['support'] for _, row in frequent_itemsets_fp.iterrows()}

    # Generate association rules if frequent itemsets exist
    association_rules_fp_growth = association_rules(frequent_itemsets_fp, metric='confidence', min_threshold=min_confidence)
    association_rules_fp_growth_set = set(
        (frozenset(antecedents), frozenset(consequents), confidence)
        for antecedents, consequents, confidence in zip(
            association_rules_fp_growth['antecedents'],
            association_rules_fp_growth['consequents'],
            association_rules_fp_growth['confidence']
        )
    )
    print("\nAssociation Rules (FP-Growth):")
    print(association_rules_fp_growth[['antecedents', 'consequents', 'confidence']])
```

Confirming Results of all the algorithms by taking intersection of Algorithm Result Sets and checking the generated Rules

```
[195] # Compare association rules between brute force, apriori, and fp-growth
      print('\nComparison of Association Rules (Brute Force vs Apriori):')
      common_rules_bf_apriori = association_rules_brute_force_set.intersection(association_rules_apriori_set)
      for rule in common_rules_bf_apriori:
          print(f'Rule: {set(rule[0])} -> {set(rule[1])}, Confidence: {rule[2]:.2f}')

      print('\nComparison of Association Rules (Brute Force vs FP-Growth):')
      common_rules_bf_fp = association_rules_brute_force_set.intersection(association_rules_fp_growth_set)
      for rule in common_rules_bf_fp:
          print(f'Rule: {set(rule[0])} -> {set(rule[1])}, Confidence: {rule[2]:.2f}')

      print('\nComparison of Association Rules (Apriori vs FP-Growth):')
      common_rules_apriori_fp = association_rules_apriori_set.intersection(association_rules_fp_growth_set)
      for rule in common_rules_apriori_fp:
          print(f'Rule: {set(rule[0])} -> {set(rule[1])}, Confidence: {rule[2]:.2f}')
```

Comparing Performance of All the Algorithms

```
[18] print('\nTiming Performance:')
     print(f'Brute Force Time: {brute_force_time:.4f} seconds')
     print(f'Apriori Time: {apriori_time:.4f} seconds')
     print(f'FP-Growth Time: {fp_growth_time:.4f} seconds')
```

Below are screenshots to show that the program runs in the terminal

```
#############################################################################################################
Welcome!!!!!!!!!!
-------------------------------------------------------------------------------------------------------------

Select the Supermarket:
1.Amazon
2.K-Mart
3.Shoprite
4.Target
5.Walmart
[1-5]3

Enter Minimum Support (Enter Float Value Between 0-1)0.3

Enter Minimum Confidence (Enter Float Value Between 0-1)0.6
Generating 1-itemsets
Generating 2-itemsets
Generating 3-itemsets

Frequent Itemsets (Brute Force):
Itemset: ['Dry Fit V-Nick'], Support: 0.36
Itemset: ['Hoodies'], Support: 0.32
Itemset: ['Modern Pants'], Support: 0.4
Itemset: ['Running Shoe'], Support: 0.56
Itemset: ['Soccer Shoe'], Support: 0.6
Itemset: ['Socks'], Support: 0.32
Itemset: ['Sweatshirts'], Support: 0.32
Itemset: ['Swimming Shirt'], Support: 0.48
Itemset: ['Tech Pants'], Support: 0.48
Itemset: ['Running Shoe', 'Soccer Shoe'], Support: 0.36
Itemset: ['Swimming Shirt', 'Tech Pants'], Support: 0.32

Association Rules (Brute Force):
Rule: frozenset({'Swimming Shirt'}) -> frozenset({'Tech Pants'}), confidence: 0.67
Rule: frozenset({'Running Shoe'}) -> frozenset({'Soccer Shoe'}), confidence: 0.64
Rule: frozenset({'Tech Pants'}) -> frozenset({'Swimming Shirt'}), confidence: 0.67
Rule: frozenset({'Soccer Shoe'}) -> frozenset({'Running Shoe'}), confidence: 0.60
```

```
Frequent Itemsets (Apriori):
    support                     itemsets
0      0.36              (Dry Fit V-Nick)
1      0.32                     (Hoodies)
2      0.40                (Modern Pants)
3      0.56                (Running Shoe)
4      0.60                 (Soccer Shoe)
5      0.32                       (Socks)
6      0.32                 (Sweatshirts)
7      0.48              (Swimming Shirt)
8      0.48                  (Tech Pants)
9      0.36   (Soccer Shoe, Running Shoe)
10     0.32  (Tech Pants, Swimming Shirt)

Association Rules (Apriori):
        antecedents       consequents  confidence
0     (Soccer Shoe)    (Running Shoe)    0.600000
1    (Running Shoe)     (Soccer Shoe)    0.642857
2      (Tech Pants)  (Swimming Shirt)    0.666667
3  (Swimming Shirt)      (Tech Pants)    0.666667
```

```
Frequent Itemsets (FP-Growth):
    support                     itemsets
0      0.60                 (Soccer Shoe)
1      0.56                (Running Shoe)
2      0.32                       (Socks)
3      0.32                     (Hoodies)
4      0.48                  (Tech Pants)
5      0.36              (Dry Fit V-Nick)
6      0.48              (Swimming Shirt)
7      0.40                (Modern Pants)
8      0.32                 (Sweatshirts)
9      0.36   (Soccer Shoe, Running Shoe)
10     0.32  (Tech Pants, Swimming Shirt)

Association Rules (FP-Growth):
        antecedents       consequents  confidence
0     (Soccer Shoe)    (Running Shoe)    0.600000
1    (Running Shoe)     (Soccer Shoe)    0.642857
2      (Tech Pants)  (Swimming Shirt)    0.666667
3  (Swimming Shirt)      (Tech Pants)    0.666667
```

```
Intersection of Association Rules (Brute Force vs Apriori):
Rule: {'Swimming Shirt'} -> {'Tech Pants'}, Confidence: 0.67
Rule: {'Running Shoe'} -> {'Soccer Shoe'}, Confidence: 0.64
Rule: {'Tech Pants'} -> {'Swimming Shirt'}, Confidence: 0.67
Rule: {'Soccer Shoe'} -> {'Running Shoe'}, Confidence: 0.60

Intersection of Association Rules (Brute Force vs FP-Growth):
Rule: {'Swimming Shirt'} -> {'Tech Pants'}, Confidence: 0.67
Rule: {'Running Shoe'} -> {'Soccer Shoe'}, Confidence: 0.64
Rule: {'Tech Pants'} -> {'Swimming Shirt'}, Confidence: 0.67
Rule: {'Soccer Shoe'} -> {'Running Shoe'}, Confidence: 0.60

Intersection of Association Rules (Apriori vs FP-Growth):
Rule: {'Swimming Shirt'} -> {'Tech Pants'}, Confidence: 0.67
Rule: {'Running Shoe'} -> {'Soccer Shoe'}, Confidence: 0.64
Rule: {'Tech Pants'} -> {'Swimming Shirt'}, Confidence: 0.67
Rule: {'Soccer Shoe'} -> {'Running Shoe'}, Confidence: 0.60
```

Results and Evaluation:

The Brute Force, Apriori, and FP-Growth algorithms were all successful in identifying frequent itemsets from the dataset, but their efficiency varied significantly. Brute Force was the slowest due to its exhaustive nature. Apriori improved efficiency by pruning non-frequent itemsets, but still required multiple candidate generations. FP-Growth was the fastest, leveraging an efficient tree structure to avoid exhaustive candidate generation. The results highlight FP-Growth's superior performance, especially for larger datasets.

```
Timing Performance:
Brute Force Time: 0.0066 seconds
Apriori Time: 0.0091 seconds
FP-Growth Time: 0.0043 seconds
```

Conclusion:

The comparison of Brute Force, Apriori, and FP-Growth algorithms for frequent itemset mining demonstrated significant differences in efficiency and performance. The Brute Force method, while simple, was the least efficient due to its exhaustive search process. Apriori improved on this by pruning non-frequent itemsets, but still required multiple passes through the data. FP-Growth proved to be the most efficient, using a compact tree structure to eliminate the need for candidate generation. Overall, FP-Growth is the preferred choice for larger datasets, offering both speed and scalability.