Midterm Project: Market Basket Analysis Using Apriori Algorithm

Christopher Chan
DS634 – Data Mining
February 25^{th}, 2024
Professor Yassar Abduallah

Table of Contents

1. Overview

The Apriori algorithm is a data mining tool proposed by Agrawal, Imilinski, and Swani that is used in a plethora of industries to discover frequently occurring items in a large data set. Retail and e-commerce would implement the algorithm to determine product that frequently bought together. This helps with optimizing product placements, recommending products, and developing discounts to enhance customer experience and increase revenues. Additionally, the algorithm could be employed in the healthcare industry to determine treatment plans by employing drugs commonly used together and discovering new drugs based on structural and functional group similarity to preexisting drugs. This project involves the evaluation of three python scripts: brute force, Apriori algorithm, and library-based Apriori algorithm for conducting market-basket analysis across multiple transaction databases with different numbers of unique items and transactions.

2. Script Description

A high-level of the scripts will be described in this section. The instructions to execute the script provided in this report can be found in Appendix II.

Brute Force – Generate every itemset combination and filter out itemset based on min_support.

1. Generate a 1-itemset of unique items in the transaction list.
2. Generate k-itemset using the 1-itemset, where the stop condition is when k > max itemset size.
3. Prune all itemsets where support is lower than minimum support.
4. Return frequent itemsets and associated support.

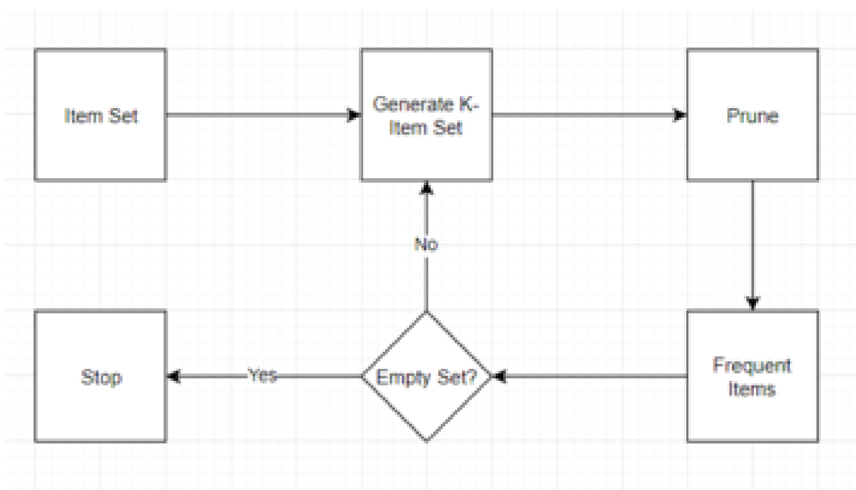

Figure 1. Diagram of Apriori Algorithm[1]

Apriori Algorithm – Iteratively generates itemsets based on previous frequent itemsets, which is filtered by the min_support.

1. Generate a 1-itemset of unique items in the transaction list.
2. Find frequent 1-itemset by comparing the support of 1-itemset against the minimum support.
    - Support is how frequent X occurs in the transaction list.
    - Support(X) = number of transaction X appears / total number of transactions.
3. Use frequent (k-1)-itemsets to generate k-itemsets.
4. Prune all k-itemsets that are not frequent in the (k-1)-itemsets.
    - This is a principle where if {X} is not above min support that all supersets of {X} will not have supports above the minimum.
5. Find frequent k-itemset by comparing the support of k-itemset against the minimum support.
6. Repeat 3-5 until there are no more frequent k-itemsets that can be generated or when the max transaction size is reached.
7. Return frequent itemsets and associated support.

Association Rule – Generate association rules from the frequent itemset that fulfill the minimum confidence threshold.

1. Generate X -> Y rules for all combinations of the frequent itemset where items in X are not in Y.
    - {X, Y} -> {Y, Z} is not a valid rule.
    - {X, Y} -> {X} is a valid rule.
2. Calculate the confidence of each valid association rule and eliminate all rules where the confidence is lower than the minimum confidence.
    - Confidence is how frequent Y occurs in a transaction that contains X.
    - Confidence of {X} -> {Y} = Support of {X, Y} / Support of {X}.
3. Return association rules that are above the confidence.

Apriori Library – incorporates mlxtend python library to generate frequent list and associations rules based on minimum support and minimum confidence.

3. Database Information

The performance of the scripts were evaluated using seven databases with varying number of items and number of transactions. Here is an overview of the databases:

1. amazon_book.db – a transaction list of programming books sold by Amazon.
   - 10 unique items
   - 20 transactions
   - 5 items max per transaction
2. bestbuy.db – a transaction list of electronics sold by BestBuy.
   - 10 unique items
   - 20 transactions
   - 10 items max per transaction
3. kmart.dt – a transaction list of sleeping products sold by Kmart
   - 10 unique items
   - 20 transactions
   - 7 items max per transaction
4. nike.db - a transaction list of athletic gear sold by Nike.
   - 10 unique items
   - 20 transactions
   - 10 items max per transaction
5. generic.db - a transaction list of pseudorandom letters
   - 6 unique items
   - 11 transactions
   - 4 items max per transaction
6. class_example.db – a transaction list of letters used in the lecture notes.
   - 5 unique items
   - 7 transactions
   - 4 items max per transaction
7. amazon_food – a randomly generated transaction list of top food products sold by Amazon.
   - 30 unique items
   - 20 transactions
   - 8 items max per transaction

The datasets were provided by Professor Yassar Abduallah and databases (.db files) were generated using sqlite3.[2,3] A script demonstrating how the database is generated is provided in the appendix II. The database will be converted into a pandas dataframe and fed into each script for performance evaluation.

4. Performance Evaluation

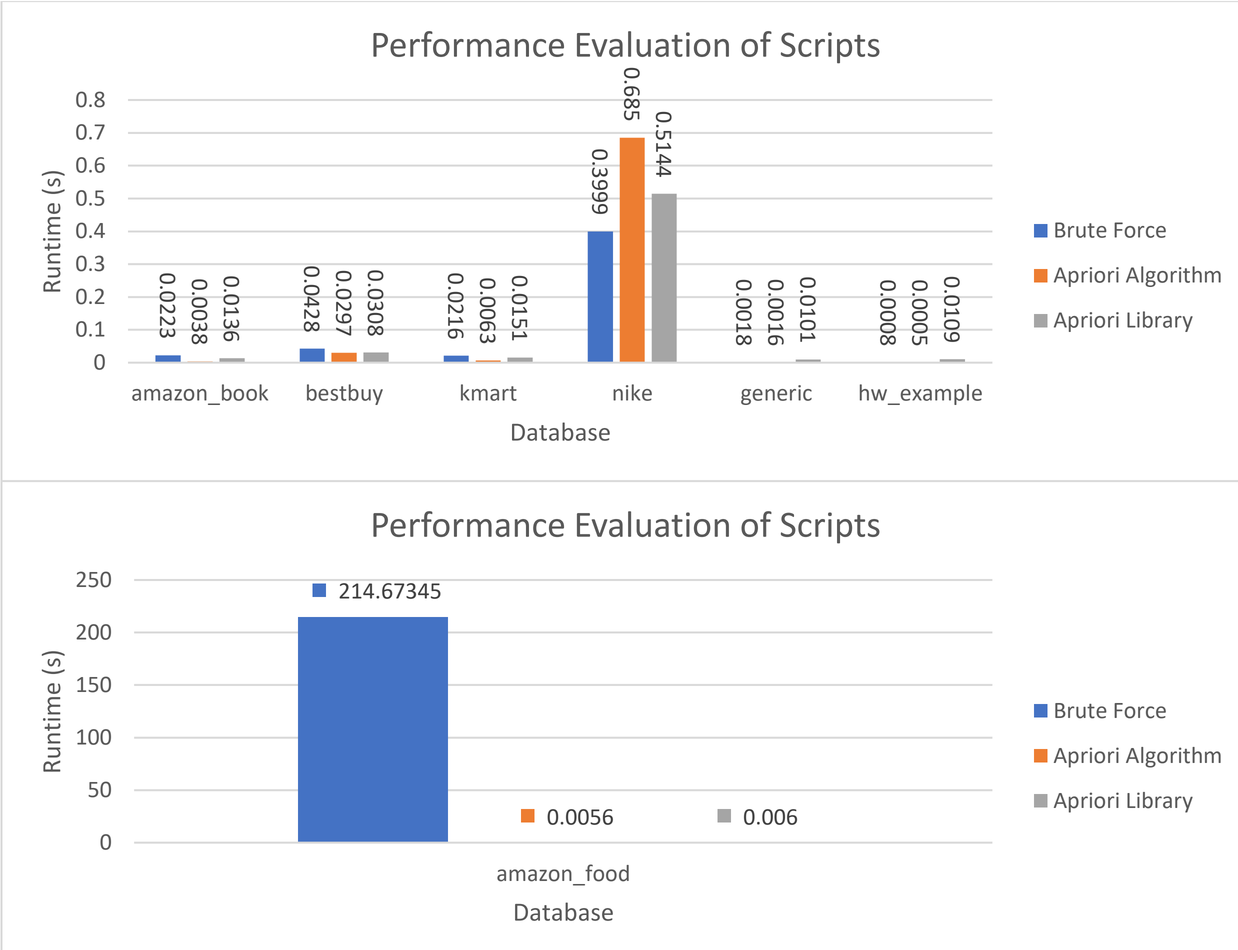

Figure 2. Runtime evaluation of dataset with min_support = 0.2 & min_confidence = 0.2

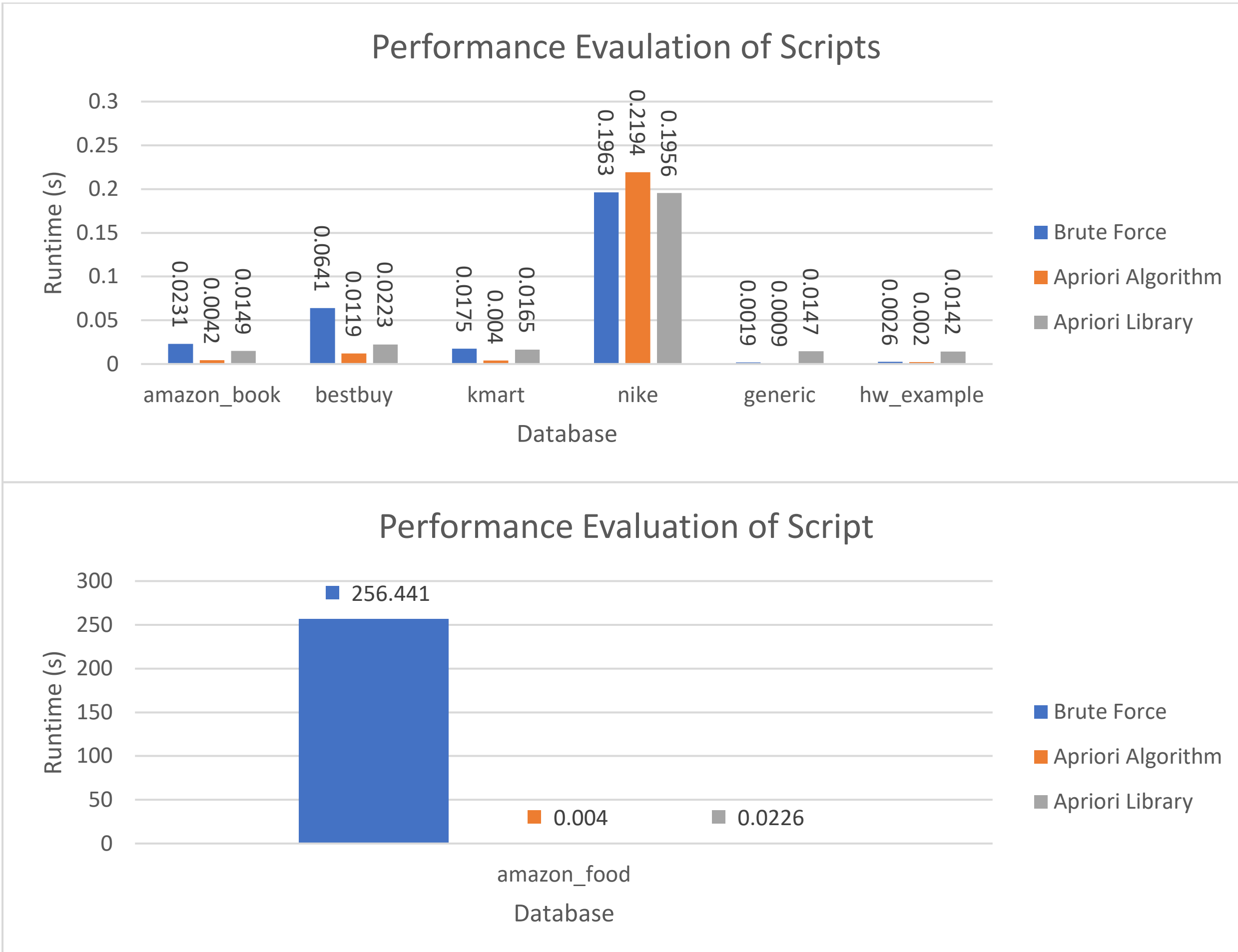

Figure 3. Runtime evaluation of dataset with min_support = 0.3 & min_confidence = 0.6

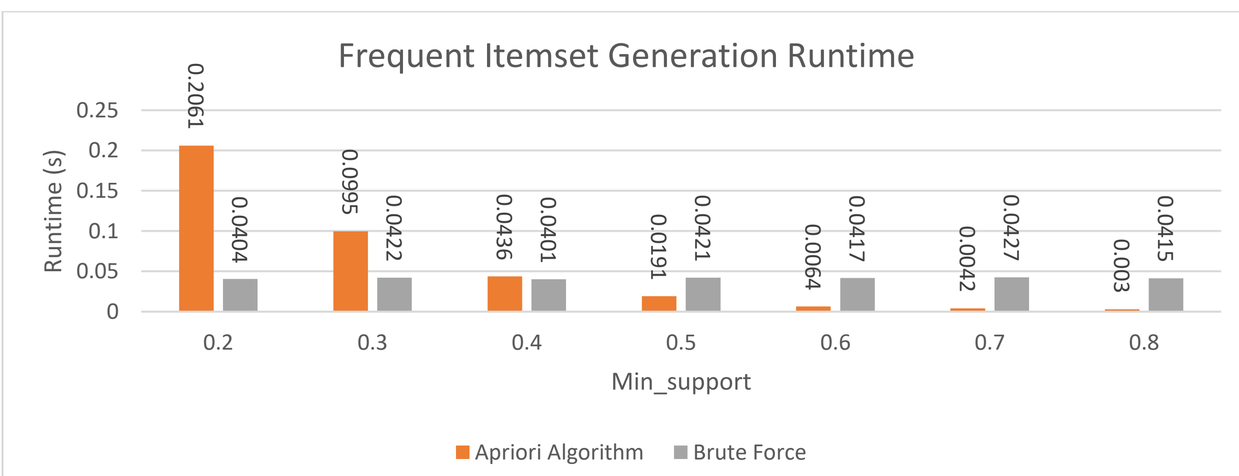

Figure 4. Runtime of frequent itemset generation for nike.db with min_confidence = 0.6

Each script was executed on each database 10 times and the average runtime was calculated; raw values are provided in the appendix. In general, the Apriori algorithm implementation generated the frequent itemsets and association rules much faster than the brute force and the Apriori library script (Figures 2 & 3). An exception to this trend would be the Nike database where the brute force performed exceptionally well. A possible explanation for the deviation could be due to low minimum support which may result in overhead when generating and pruning itemsets. The explanation was supported in Figure 4, where there was a significant decrease in runtime for generating the itemsets in the Nike database when the minimum support increased from 0.2 and 0.8 while holding the minimum confidence constant at 0.6.

The amazon_food.db demonstrated the advantage of the Apriori algorithm over the brute force method as it reduces computational time by efficiently pruning unnecessary itemsets, thereby reducing the number of supersets. This was consistent across different minimum support and minimum confidence (Figures 2 & 3).

Although the Apriori algorithm is efficient, there are some limitations to the algorithm especially when dealing with large datasets and low minimum support. Possible improvements may include partitioning the dataset and mining the frequent itemsets independently, sampling a portion of the dataset that may have the same insights, and parallel computing with multiple computer clusters.

5. Conclusion

The Apriori algorithm is a simple and efficient algorithm that can be used to find relationships between different items in large datasets by leveraging the Apriori property to significantly reduce the search space. Despite its simplicity, the algorithm is used by a plethora of industries for market basket analysis, inventory management, recommender systems, and discovering drug interactions.

6. Reference

1. https://en.wikipedia.org/wiki/Association_rule_learning
2. DS634 Midterm datasets
3. DS634 Lecture 2A

7. Appendix

I. How to run the script

Python Package Requirements

- Sqllite3
- Pandas
- Mlxtend
- Time
- Combinations

Prior to executing the script, please ensure that all .db files are in the same working location as where the midterm.ipynb is located.

```python
# Parameters

print("Pick Database: ")
print("1 - Amazon Books")
print("2 - Best Buy")
print("3 - K-mart")
print("4 - Nike")
print("5 - Generic")
print("6 - Homework Example")
print("7 - Amazon Food")

db = str(input("Please input the desired database"))
min_support = float(input("Please input the desired minimum support"))
min_confidence = float(input("Please input the desired minimum confidence"))
```

Figure 5. Input script

When executing the script, there will be three input popups that prompt users to input desired database, minimum support and minimum confidence (Figure 5). Unfortunately, there are no checks in place to ensure that the inputs are valid.

```
Transaction_ID                                    Transactions
             1  Wonderful Pistachios No Shells, PASTA RONI Qua...
             2                                 Frito-Lay Party Mix
             3  Quaker Instant Oatmeal Express Cups, Jack Link...
             4  GoGo squeeZ Fruit on the Go, Nature Valley Cru...
             5  Bumble Bee Chunk Light Tuna In Water, HORMEL C...
             6  Mott Fruit Flavored Snacks, Nature Valley Crun...
             7  Kelloggs Cold Breakfast Cereal, Chef Boyardee ...
             8  TWIZZLERS Twists Strawberry Flavored Licorice ...
             9  GoGo squeeZ Fruit on the Go, Maruchan Ramen Ch...
            10  Kelloggs Cold Breakfast Cereal, TWIZZLERS Twis...
            11  Quaker Instant Oatmeal, Quaker Instant Oatmeal...
            12  SpaghettiOs Canned Pasta with Meatballs, Jack ...
            13  Chef Boyardee Beef Ravioli, Premier Liquid Pro...
            14  Frito-Lay Party Mix, Glico Pocky, Kelloggs Col...
            15  Pop-Tarts Toaster Pastries, Glico Pocky, Slim ...
            16  Nissin Chow Mein Teriyaki, Glico Pocky, Premie...
            17  Bumble Bee Chunk Light Tuna In Water, Kraft Ea...
            18  GoGo squeeZ Fruit on the Go, Glico Pocky, Bumb...
            19  Wonderful Pistachios No Shells, BUMBLE BEE Sna...
            20  Nissin Chow Mein Teriyaki, Quaker Instant Oatm...
```

Figure 6. Example input

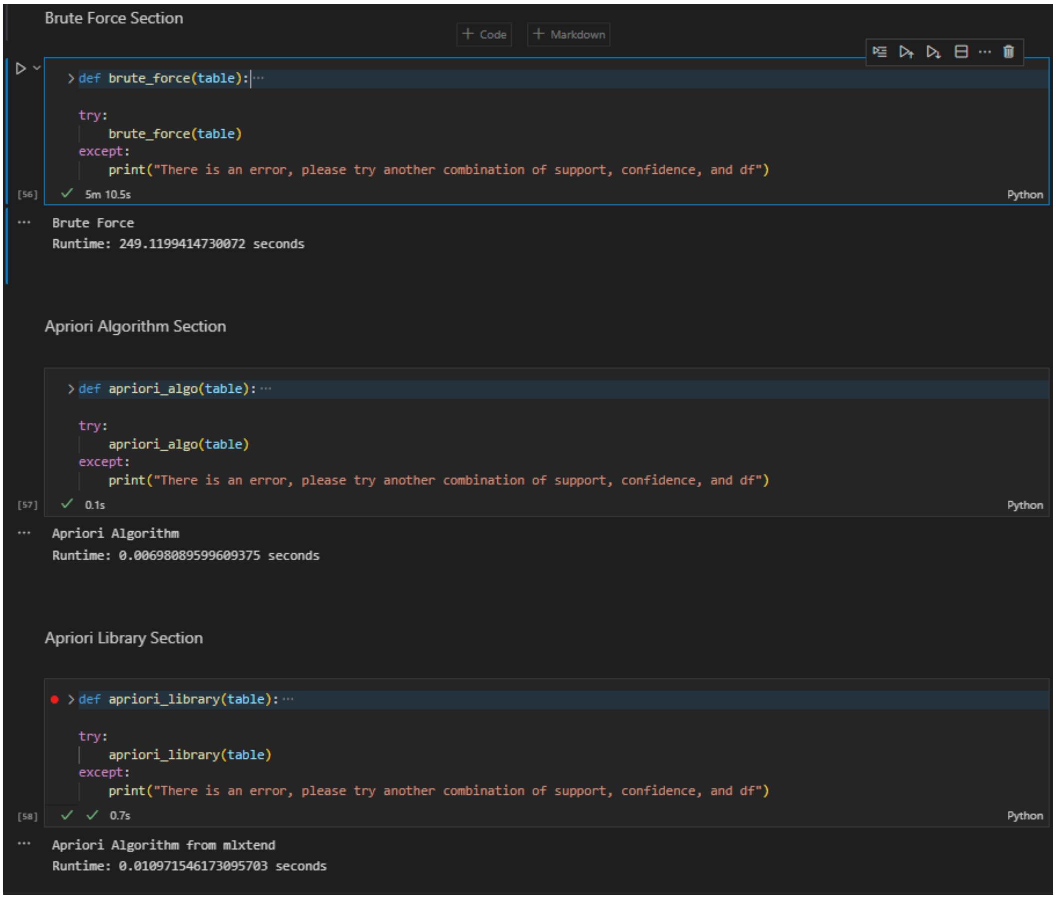

Figure 7. Executing the scripts

After inputting desired parameters, the user will be printed their selection. Figure 6 highlights the expected input dataframe for the scripts, which is pulled from the db using sql query and saved into a pandas dataframe. Our column of interest will be the transactions column.

The user can freely run three scripts (brute force, Apriori algorithm, and Apriori library), which will run the script (Code snippet provided in Appendix III) with user parameters and output the runtime and save the outputs as text files in the working folder. The text files are named as follows:

{db}_{script_type}_{result}_output.txt

Where,
db = database selected
script_type = either brute force, Apriori algorithm, or Apriori library
result = either the frequent itemsets or the association rules (Brute force will also output a unfiltered list of all the itemset to demonstrate brute force all combinations)

The users can open the text files and compare the results (Sample output provided in Appendix IV).

II. Database Generation

The script below generates a .db for the transactions provided as “data”. Due to repetitiveness, the creation of other databases has been omitted.

```python
import pandas as pd
import sqlite3

data = [
    "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 Programming: The Big Nerd Ranch, Head First Java 2nd Edition",
    "Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition, Beginning Programming with Java",
    "Android Programming: The Big Nerd Ranch, Beginning Programming with Java, Java 8 Pocket Guide",
    "A Beginner'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",
    "Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition, Beginning Programming with Java",
    "Beginning Programming with Java, Java 8 Pocket Guide, C++ Programming in Easy Steps",
    "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, HTML and CSS: Design and Build Websites",
    "A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Java 8 Pocket Guide, HTML and CSS: Design 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: The Big Nerd Ranch",
    "A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch",
    "Head First Java 2nd Edition, Beginning Programming with Java, Java & Pocket Guide",
    "Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition",
    "A Beginner's Guide, Java: The Complete Reference, Java For Dummies"
]

# Create a dataframe with a column named "Transactions"
df = pd.DataFrame(data, columns=["Transactions"])

# Save the df as a csv
df.to_csv("amazon_book.csv", index=False)
```

```
data_df = pd.read_csv("amazon_book.csv")

# Connect to SQLite database
conn = sqlite3.connect('amazon_book.db')
cursor = conn.cursor()

# Create database table
cursor.execute('''CREATE TABLE IF NOT EXISTS amazon_book (Transaction_ID INTEGER
PRIMARY KEY, Transactions TEXT)''')

# Insert rows in the csv into the database
for row in data_df.itertuples():
    cursor.execute('''INSERT INTO amazon_book (Transactions) VALUES (?)''',
(row.Transactions,))

conn.commit()
conn.close()
```

III. Frequent Itemset and Association Rule Scripts

A. Brute Force

```
def brute_force(table):
    # Connect to SQLite database
    conn = sqlite3.connect(f'{table}.db')

    # Execute SELECT query to retrieve data from the table
    df = pd.read_sql_query(f"SELECT * FROM {table}", conn)

    # Close the connection
    conn.close()


    df['Transactions'] = df['Transactions'].str.split(', ').apply(set)

    def generate_combinations(item, k):
        return set(combinations(item, k))

    def calculate_min_support(data, itemset):
        count = 0
        for row in data:
            if set(item for tpl in itemset for item in tpl).issubset(row):
                count += 1
        return count / df.shape[0]

    # Generate all frequent 1-itemset
    def first_pass(df, support):
        item = set()
        max_transaction_length = 1
        # Find all unique items in the transaction list and max transaction length
        for index, row in df.iterrows():
            for i in row['Transactions']:
                item.add(tuple([i]))
                max_transaction_length = max(len(row['Transactions']),
max_transaction_length)

        itemsets = []

        # Add all 1-itemset and support to the itemsets
        for i in item:
            support = calculate_min_support(df['Transactions'].values, tuple([i]))
            itemsets.append([[i], support])

        return item, itemsets, max_transaction_length

    # Generate all k-itemset, where k is between 2 and the max transaction length
    def second_pass(df, item, itemsets, support, max_transaction_length):
        k = 2

        # The stop condition for brute force will be the k-itemset where k is the max
transaction length
        while k <= max_transaction_length:
```

```
            # Generate and add all k-itemset and support to the itemsets
            all_combinations = generate_combinations(item, k)
            for i in all_combinations:
                support = calculate_min_support(df['Transactions'].values, i)
                itemsets.append([list(i), support])
            k += 1

        return itemsets

    def filter_itemsets(itemsets, min_support):
        res = []
        for i in itemsets:
            subset, support = i
            if support >= min_support:
                res.append(i)
        return res

    # Begin timer
    start = time.time()

    item, itemsets, max_transaction_length = first_pass(df, min_support)
    if max_transaction_length >= 2:
        itemsets = second_pass(df, item, itemsets, min_support,
max_transaction_length)
    frequent_itemsets = filter_itemsets(itemsets, min_support)

    # Create all subset for an item in the frequent itemset
    # If item = A, B and item length is 2
    # The subsets would be A, B, AB
    def find_subset(item, item_length):
        subsets = []
        for i in range(1, item_length + 1):
            subsets.extend(combinations(item, i))
        return subsets

    # Generate all association rules from the frequent itemset
    def association_rules(frequent_itemsets, min_confidence):
        rules = list()
        dic = {}

        # Convert the frequent itemset into a dictionary
        for item in frequent_itemsets:
            key = frozenset(item[0])
            value = item[1]
            dic[key] = value

        # Generate all possible {X} -> {Y} and checks if {XY} exist to compute
confidence
        for item, support in dic.items():
            item_length = len(item)
            if item_length > 1:
                subsets = find_subset(item, item_length)
                for X in subsets:
                    Y = set(item).difference(X)
```

```
                    if Y:
                        X = frozenset(X)
                        XY = X | frozenset(Y)

                        # Checks if XY and X exists in the dictionary of frequent
items
                        if XY in dic and X in dic:
                            confidence = dic[XY] / dic[X]

                            # Add {X} -> {Y} that have confidence above the
min_confidence
                            if confidence >= min_confidence:
                                rules.append((X, Y, confidence))
        return rules

    # Generate all association rules above the minimum confidence
    rules = association_rules(frequent_itemsets, min_confidence)

    # Stop timer
    end = time.time()

    # Write all itemsets and support to textfile, bruteforce will add all possible
itemset combinations
    with open(f"{table}_bruteforce_unfilteredfreqlist_output.txt", "w") as f:
        f.write(f"min_support: {min_support}, min_confidence: {min_confidence}\n")

        for i in itemsets:
            item, supp = i
            f.write(f"{item}: {supp}\n")

    # Write all frequent itemsets and support to textfile
    with open(f"{table}_bruteforce_freqlist_output.txt", "w") as f:
        f.write(f"min_support: {min_support}, min_confidence: {min_confidence}\n")

        for i in frequent_itemsets:
            item, supp = i
            f.write(f"{item}: {supp}\n")

    # Write all association rules and confidence to textfile
    with open(f"{table}_bruteforce_association_output.txt", "w") as f:
        f.write(f"min_support: {min_support}, min_confidence: {min_confidence}\n")

        for i in rules:
            X, Y, conf = i
            f.write(f"{X} -> {Y} : {conf}\n")

    print("Runtime:", end - start, "seconds")
    return
```

B. Apriori Algorithm

```
def apriori_algo(table):
    # Intialize db as a dataframe
    conn = sqlite3.connect(f'{table}.db')

    df = pd.read_sql_query(f"SELECT * FROM {table}", conn)

    conn.close()

    df['Transactions'] = df['Transactions'].str.split(', ').apply(set)

    start = time.time()

    # Generate all k-itemsets using the combinations from [k-1]-itemsets starting
from k=3
    def generate_new_combinations(prev_frequent_item, k):
        print(k)
        prev_frequent_item_list = list(prev_frequent_item)  # Convert set to list
        new_combinations = set()
        # Iterate over unique pairs of combinations
        for i, item_1 in enumerate(prev_frequent_item_list):
            for item_2 in prev_frequent_item_list[i + 1:]:
                # Prunes k-itemset that are not frequent in the previous level
                if all(x == y for x, y in zip(item_1[:-1], item_2[:-1])):

                    # Create the k-itemset from the frequent
                    union_set = tuple(sorted(set(item_1 + item_2)))
                    print(k, union_set)
                    if len(union_set) == k:
                        new_combinations.add(union_set)
        return new_combinations

    # Generates all 2-itemsets from the frequent 1-itemset
    def generate_combinations(item, k):
        return set(combinations(item, k))

    # Calcualtes the support for a given itemset by dividing the frequency of the
itemset in the transaction list by the total number of transactions
    def calculate_support(data, itemset):
        count = 0
        for row in data:
            if set(item for tpl in itemset for item in tpl).issubset(row):
                count += 1
        return count / df.shape[0]

    # Generate all frequent 1-itemset
    def first_pass(df, min_support):
        item = set()
        max_transaction_length = 1

        # Find all unique items in the transaction list
        for index, row in df.iterrows():
            for i in row['Transactions']:
                item.add(tuple([i]))
```

```
                max_transaction_length = max(len(row['Transactions']),
max_transaction_length)

        frequent_itemsets = []
        remove = []

        # Checks if the unique items are frequent
        for i in item:
            support = calculate_support(df['Transactions'].values, tuple([i]))
            if support < min_support:
                remove.append(i)
            else:
                frequent_itemsets.append([[i], support])

        # Remove infrequent items
        item.difference_update(remove)

        return item, frequent_itemsets, max_transaction_length

    # Generate all frequent k-itemset, where k >= 2
    def second_pass(df, item, frequent_itemsets, min_support,
max_transaction_length):
        k = 2
        while True:
            # Generate k-itemsets
            if k == 2:
                all_combinations = generate_combinations(item, k)
            else:
                all_combinations = generate_new_combinations(next_combinations, k)
            next_combinations = set()
            # For every k-itemset, check if they are frequent
            for i in all_combinations:
                support = calculate_support(df['Transactions'].values, i)
                if support >= min_support:
                    frequent_itemsets.append([list(i), support])
                    # Adds only the frequent
                    next_combinations.add(i)

            # If there are no more frequent itemsets, we know to stop looking at the
next level
            if len(next_combinations) == 0:
                break

            k += 1

        return frequent_itemsets

    # Generate frequent 1-itemsets
    item, frequent_itemsets, max_transaction_length = first_pass(df, min_support)
    print(max_transaction_length)
    # Generate frequent k-itemsets
    frequent_itemsets = second_pass(df, item, frequent_itemsets, min_support,
max_transaction_length)

    # Create all subset for an item in the frequent itemset
```

```
    # If item = A, B and item length is 2
    # The subsets would be A, B, AB
    def find_subset(item, item_length):
        subsets = []
        for i in range(1, item_length + 1):
            subsets.extend(combinations(item, i))
        return subsets

    # Generate all association rules from the frequent itemset
    def association_rules(frequent_itemsets, min_confidence):
        rules = list()
        dic = {}

        # Convert the frequent itemset into a dictionary
        for item in frequent_itemsets:
            key = frozenset(item[0])
            value = item[1]
            dic[key] = value

        # Generate all possible {X} -> {Y} and checks if {XY} exist to compute
confidence
        for item, support in dic.items():
            item_length = len(item)
            if item_length > 1:
                subsets = find_subset(item, item_length)
                for X in subsets:
                    Y = set(item).difference(X)
                    if Y:
                        X = frozenset(X)
                        XY = X | frozenset(Y)

                        # Checks if XY and X exists in the dictionary of frequent
items
                        if XY in dic and X in dic:
                            confidence = dic[XY] / dic[X]

                            # Add {X} -> {Y} that have confidence above the
min_confidence
                            if confidence >= min_confidence:
                                rules.append((X, Y, confidence))
        return rules

    # Generate all association rules above the minimum confidence
    rules = association_rules(frequent_itemsets, min_confidence)

    # Stop timer
    end = time.time()

    # Write all frequent itemsets and support to textfile
    with open(f"{table}_apriorialgo_freqlist_output.txt", "w") as f:
        f.write(f"min_support: {min_support}, min_confidence: {min_confidence}\n")

        for i in frequent_itemsets:
            item, supp = i
            f.write(f"{item}: {supp}\n")
```

```
    # Write all association rules and confidence to textfile
    with open(f"{table}_apriorialgo_association_output.txt", "w") as f:
        f.write(f"min_support: {min_support}, min_confidence: {min_confidence}\n")

        for i in rules:
            X, Y, conf = i
            f.write(f"{X} -> {Y} : {conf}\n")

    print("Runtime:", end - start, "seconds")
    return
```

C. Apriori Library

```
def apriori_library(table):

    # Intialize db as a dataframe
    conn = sqlite3.connect(f'{table}.db')

    df = pd.read_sql_query(f"SELECT * FROM {table}", conn)

    conn.close()

    # Start timer
    start = time.time()

    # One hot encode the unique items
    new_df = df['Transactions'].str.get_dummies(', ')

    # Generate frequent itemsets using mlxtend
    frequent_itemsets = apriori(new_df, min_support, use_colnames=True)

    # Generate association rules using mlxtend
    rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=min_confidence)

    # Stop timer
    end = time.time()

    # Write all frequent itemsets and support to textfile
    with open(f"{table}_apriorilib_freqlist_output.txt", "w") as f:
        f.write(f"min_support: {min_support}, min_confidence: {min_confidence}\n")

        for index, row in frequent_itemsets.iterrows():
            f.write(f"{row['itemsets']}: {row['support']}\n")

    # Write all association rules and confidence to textfile
    with open(f"{table}_apriorilib_association_output.txt", "w") as f:
        f.write(f"min_support: {min_support}, min_confidence: {min_confidence}\n")

        for index, rule in rules.iterrows():
            f.write(f"{rule['antecedents']} -> {rule['consequents']}:
{rule['confidence']}\n")

    print("Runtime:", end - start, "seconds")
    return
```

IV . Output

Here is the example output for nike.db with minimum support = 0.8 and minimum confidence = 0.6.

nike_bruteforce_freqlist_output - Notepad

File Edit Format View Help

```
min_support: 0.8, min_confidence: 0.6
[('Dry Fit V-Nick',)]: 0.9
[('Tech Pants',)]: 0.8
[('Swimming Shirt',)]: 0.85
[('Rash Guard',)]: 0.95
[('Dry Fit V-Nick',), ('Tech Pants',)]: 0.8
[('Tech Pants',), ('Rash Guard',)]: 0.8
[('Swimming Shirt',), ('Rash Guard',)]: 0.85
[('Dry Fit V-Nick',), ('Swimming Shirt',)]: 0.8
[('Dry Fit V-Nick',), ('Rash Guard',)]: 0.9
[('Dry Fit V-Nick',), ('Tech Pants',), ('Rash Guard',)]: 0.8
[('Dry Fit V-Nick',), ('Swimming Shirt',), ('Rash Guard',)]: 0.8
```

nike_apriorialgo_freqlist_output - Notepad

File Edit Format View Help

```
min_support: 0.8, min_confidence: 0.6
[('Dry Fit V-Nick',)]: 0.9
[('Tech Pants',)]: 0.8
[('Swimming Shirt',)]: 0.85
[('Rash Guard',)]: 0.95
[('Dry Fit V-Nick',), ('Tech Pants',)]: 0.8
[('Dry Fit V-Nick',), ('Swimming Shirt',)]: 0.8
[('Tech Pants',), ('Rash Guard',)]: 0.8
[('Swimming Shirt',), ('Rash Guard',)]: 0.85
[('Dry Fit V-Nick',), ('Rash Guard',)]: 0.9
[('Dry Fit V-Nick',), ('Rash Guard',), ('Swimming Shirt',)]: 0.8
[('Dry Fit V-Nick',), ('Rash Guard',), ('Tech Pants',)]: 0.8
```

nike_apriorilib_freqlist_output - Notepad

File Edit Format View Help

```
min_support: 0.8, min_confidence: 0.6
frozenset({'Dry Fit V-Nick'}): 0.9
frozenset({'Rash Guard'}): 0.95
frozenset({'Swimming Shirt'}): 0.85
frozenset({'Tech Pants'}): 0.8
frozenset({'Rash Guard', 'Dry Fit V-Nick'}): 0.9
frozenset({'Swimming Shirt', 'Dry Fit V-Nick'}): 0.8
frozenset({'Tech Pants', 'Dry Fit V-Nick'}): 0.8
frozenset({'Rash Guard', 'Swimming Shirt'}): 0.85
frozenset({'Rash Guard', 'Tech Pants'}): 0.8
frozenset({'Rash Guard', 'Swimming Shirt', 'Dry Fit V-Nick'}): 0.8
frozenset({'Rash Guard', 'Tech Pants', 'Dry Fit V-Nick'}): 0.8
```

nike_bruteforce_association_output - Notepad

File Edit Format View Help

```
min_support: 0.8, min_confidence: 0.6
frozenset({('Dry Fit V-Nick',)}) -> {('Tech Pants',)} : 0.888888888888889
frozenset({('Tech Pants',)}) -> {('Dry Fit V-Nick',)} : 1.0
frozenset({('Tech Pants',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Rash Guard',)}) -> {('Tech Pants',)} : 0.8421052631578948
frozenset({('Rash Guard',)}) -> {('Swimming Shirt',)} : 0.8947368421052632
frozenset({('Swimming Shirt',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Dry Fit V-Nick',)}) -> {('Swimming Shirt',)} : 0.888888888888889
frozenset({('Swimming Shirt',)}) -> {('Dry Fit V-Nick',)} : 0.9411764705882354
frozenset({('Dry Fit V-Nick',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Rash Guard',)}) -> {('Dry Fit V-Nick',)} : 0.9473684210526316
frozenset({('Dry Fit V-Nick',)}) -> {('Tech Pants',), ('Rash Guard',)} : 0.888888888888889
frozenset({('Tech Pants',)}) -> {('Dry Fit V-Nick',), ('Rash Guard',)} : 1.0
frozenset({('Rash Guard',)}) -> {('Dry Fit V-Nick',), ('Tech Pants',)} : 0.8421052631578948
frozenset({('Dry Fit V-Nick',), ('Tech Pants',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Dry Fit V-Nick',), ('Rash Guard',)}) -> {('Tech Pants',)} : 0.888888888888889
frozenset({('Tech Pants',), ('Rash Guard',)}) -> {('Dry Fit V-Nick',)} : 1.0
frozenset({('Rash Guard',)}) -> {('Dry Fit V-Nick',), ('Swimming Shirt',)} : 0.8421052631578948
frozenset({('Dry Fit V-Nick',)}) -> {('Rash Guard',), ('Swimming Shirt',)} : 0.888888888888889
frozenset({('Swimming Shirt',)}) -> {('Rash Guard',), ('Dry Fit V-Nick',)} : 0.9411764705882354
frozenset({('Dry Fit V-Nick',), ('Rash Guard',)}) -> {('Swimming Shirt',)} : 0.888888888888889
frozenset({('Swimming Shirt',), ('Rash Guard',)}) -> {('Dry Fit V-Nick',)} : 0.9411764705882354
frozenset({('Dry Fit V-Nick',), ('Swimming Shirt',)}) -> {('Rash Guard',)} : 1.0
```

nike_apriorilib_association_output - Notepad

File Edit Format View Help

```
min_support: 0.8, min_confidence: 0.6
frozenset({'Rash Guard'}) -> frozenset({'Dry Fit V-Nick'}): 0.9473684210526316
frozenset({'Dry Fit V-Nick'}) -> frozenset({'Rash Guard'}): 1.0
frozenset({'Swimming Shirt'}) -> frozenset({'Dry Fit V-Nick'}): 0.9411764705882354
frozenset({'Dry Fit V-Nick'}) -> frozenset({'Swimming Shirt'}): 0.888888888888889
frozenset({'Tech Pants'}) -> frozenset({'Dry Fit V-Nick'}): 1.0
frozenset({'Dry Fit V-Nick'}) -> frozenset({'Tech Pants'}): 0.888888888888889
frozenset({'Rash Guard'}) -> frozenset({'Swimming Shirt'}): 0.8947368421052632
frozenset({'Swimming Shirt'}) -> frozenset({'Rash Guard'}): 1.0
frozenset({'Rash Guard'}) -> frozenset({'Tech Pants'}): 0.8421052631578948
frozenset({'Tech Pants'}) -> frozenset({'Rash Guard'}): 1.0
frozenset({'Rash Guard', 'Swimming Shirt'}) -> frozenset({'Dry Fit V-Nick'}): 0.9411764705882354
frozenset({'Rash Guard', 'Dry Fit V-Nick'}) -> frozenset({'Swimming Shirt'}): 0.888888888888889
frozenset({'Swimming Shirt', 'Dry Fit V-Nick'}) -> frozenset({'Rash Guard'}): 1.0
frozenset({'Rash Guard'}) -> frozenset({'Swimming Shirt', 'Dry Fit V-Nick'}): 0.8421052631578948
frozenset({'Swimming Shirt'}) -> frozenset({'Rash Guard', 'Dry Fit V-Nick'}): 0.9411764705882354
frozenset({'Dry Fit V-Nick'}) -> frozenset({'Rash Guard', 'Swimming Shirt'}): 0.888888888888889
frozenset({'Rash Guard', 'Tech Pants'}) -> frozenset({'Dry Fit V-Nick'}): 1.0
frozenset({'Rash Guard', 'Dry Fit V-Nick'}) -> frozenset({'Tech Pants'}): 0.888888888888889
frozenset({'Tech Pants', 'Dry Fit V-Nick'}) -> frozenset({'Rash Guard'}): 1.0
frozenset({'Rash Guard'}) -> frozenset({'Tech Pants', 'Dry Fit V-Nick'}): 0.8421052631578948
frozenset({'Tech Pants'}) -> frozenset({'Rash Guard', 'Dry Fit V-Nick'}): 1.0
frozenset({'Dry Fit V-Nick'}) -> frozenset({'Rash Guard', 'Tech Pants'}): 0.888888888888889
```

nike_apriorialgo_association_output - Notepad

File Edit Format View Help

```
min_support: 0.8, min_confidence: 0.6
frozenset({('Dry Fit V-Nick',)}) -> {('Tech Pants',)} : 0.888888888888889
frozenset({('Tech Pants',)}) -> {('Dry Fit V-Nick',)} : 1.0
frozenset({('Dry Fit V-Nick',)}) -> {('Swimming Shirt',)} : 0.888888888888889
frozenset({('Swimming Shirt',)}) -> {('Dry Fit V-Nick',)} : 0.9411764705882354
frozenset({('Tech Pants',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Rash Guard',)}) -> {('Tech Pants',)} : 0.8421052631578948
frozenset({('Rash Guard',)}) -> {('Swimming Shirt',)} : 0.8947368421052632
frozenset({('Swimming Shirt',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Dry Fit V-Nick',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Rash Guard',)}) -> {('Dry Fit V-Nick',)} : 0.9473684210526316
frozenset({('Dry Fit V-Nick',)}) -> {('Swimming Shirt',), ('Rash Guard',)} : 0.888888888888889
frozenset({('Swimming Shirt',)}) -> {('Dry Fit V-Nick',), ('Rash Guard',)} : 0.9411764705882354
frozenset({('Rash Guard',)}) -> {('Dry Fit V-Nick',), ('Swimming Shirt',)} : 0.8421052631578948
frozenset({('Dry Fit V-Nick',), ('Swimming Shirt',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Dry Fit V-Nick',), ('Rash Guard',)}) -> {('Swimming Shirt',)} : 0.888888888888889
frozenset({('Rash Guard',), ('Swimming Shirt',)}) -> {('Dry Fit V-Nick',)} : 0.9411764705882354
frozenset({('Dry Fit V-Nick',)}) -> {('Tech Pants',), ('Rash Guard',)} : 0.888888888888889
frozenset({('Tech Pants',)}) -> {('Dry Fit V-Nick',), ('Rash Guard',)} : 1.0
frozenset({('Rash Guard',)}) -> {('Dry Fit V-Nick',), ('Tech Pants',)} : 0.8421052631578948
frozenset({('Dry Fit V-Nick',), ('Tech Pants',)}) -> {('Rash Guard',)} : 1.0
frozenset({('Dry Fit V-Nick',), ('Rash Guard',)}) -> {('Tech Pants',)} : 0.888888888888889
frozenset({('Tech Pants',), ('Rash Guard',)}) -> {('Dry Fit V-Nick',)} : 1.0
```

nike_bruteforce_unfilteredfreqlist_output - Notepad

File Edit Format View Help

```
min_support: 0.8, min_confidence: 0.6
[('Dry Fit V-Nick',)]: 0.9
[('Running Shoe',)]: 0.7
[('Soccer Shoe',)]: 0.4
[('Tech Pants',)]: 0.8
[('Swimming Shirt',)]: 0.85
[('Sweatshirts',)]: 0.65
[('Hoodies',)]: 0.65
[('Socks',)]: 0.65
[('Rash Guard',)]: 0.95
[('Modern Pants',)]: 0.6
[('Dry Fit V-Nick',), ('Tech Pants',)]: 0.8
[('Hoodies',), ('Modern Pants',)]: 0.45
[('Swimming Shirt',), ('Hoodies',)]: 0.55
[('Sweatshirts',), ('Hoodies',)]: 0.5
[('Dry Fit V-Nick',), ('Modern Pants',)]: 0.6
[('Rash Guard',), ('Modern Pants',)]: 0.6
[('Soccer Shoe',), ('Hoodies',)]: 0.4
[('Running Shoe',), ('Socks',)]: 0.55
[('Dry Fit V-Nick',), ('Sweatshirts',)]: 0.65
[('Tech Pants',), ('Socks',)]: 0.6
[('Swimming Shirt',), ('Socks',)]: 0.5
[('Sweatshirts',), ('Socks',)]: 0.6
[('Soccer Shoe',), ('Socks',)]: 0.3
[('Socks',), ('Rash Guard',)]: 0.6
[('Running Shoe',), ('Swimming Shirt',)]: 0.6
[('Dry Fit V-Nick',), ('Hoodies',)]: 0.65
[('Dry Fit V-Nick',), ('Soccer Shoe',)]: 0.4
[('Tech Pants',), ('Swimming Shirt',)]: 0.7
[('Socks',), ('Modern Pants',)]: 0.55
[('Hoodies',), ('Socks',)]: 0.45
[('Soccer Shoe',), ('Swimming Shirt',)]: 0.3
[('Running Shoe',), ('Rash Guard',)]: 0.65
[('Dry Fit V-Nick',), ('Socks',)]: 0.6
[('Running Shoe',), ('Tech Pants',)]: 0.6
[('Tech Pants',), ('Hoodies',)]: 0.65
```

V. Performance Data Tables

Method	amazon_book runtime (s)	bestbuy runtime (s)	kmart runtime (s)	nike runtime (s)	generic runtime (s)	hw_example runtime(s)	amazon_food runtime (s)
Brute Force	0.0223	0.0428	0.0216	0.3999	0.0018	0.0008	214.67345
Apriori Algorithm	0.0038	0.0297	0.0063	0.6850	0.0016	0.0005	0.0056
Apriori Library	0.0136	0.0308	0.0151	0.5144	0.0101	0.0109	0.0060

Table 1. Runtime evaluation of the datasets with Min_support = 0.2 & Min_confidence = 0.2

Method	amazon_book runtime (s)	bestbuy runtime (s)	kmart runtime (s)	nike runtime (s)	generic runtime (s)	hw_example runtime (s)	amazon_food runtime (s)
Brute Force	0.0231	0.0641	0.0175	0.1963	0.0019	0.0026	256.441
Apriori Algorithm	0.0042	0.0119	0.004	0.2194	0.0009	0.002	0.004
Apriori Library	0.0149	0.0223	0.0165	0.1956	0.0147	0.0142	0.0226

Table 2. Runtime evaluation of datasets with Min_support = 0.3 & Min_confidence = 0.6

Minimum Support	Apriori Algorithm	Brute Force
0.2	0.2061	0.0404
0.3	0.0995	0.0422
0.4	0.0436	0.0401
0.5	0.0191	0.0421
0.6	0.0064	0.0417
0.7	0.0042	0.0427
0.8	0.003	0.0415

Table 3. Performance evaluation of frequent itemset generation with min_confidence = 0.6