DATE: 10/13/2024 **NAME**: ANJALIGUPA RAGHAVENDRA

PROFESSOR: Dr. YASSER ABDUALLAH **UCID**: AR2729

CS 634 101 DATA MINING **EMAIL ID:** ar2729@njit.edu

MIDTERM PROJECT REPORT

ABSTRACT

This project implements and compares three algorithms: Brute Force, Apriori, and FP-Growth, for finding frequent itemsets and generating association rules from transactional datasets. The program loads transaction data from CSV files that show purchases made from various retailers and then preprocesses the data to format the dataset for regular pattern mining. The Apriori and FP-Growth methods use more effective strategies to detect frequent patterns, while the Brute Force method assesses every conceivable itemset in search of those that fulfill a given support level. The performance of each algorithm is measured, and execution times are compared to determine the fastest algorithm for a given dataset. Results include both frequent itemsets and association rules, which provide valuable insights for market basket analysis and decision-making in retail and e-commerce.

INTRODUCTION

In data mining, frequent itemset mining and association rule development are vital activities, especially in market basket research where the objective is to find patterns of often purchased items together. These patterns can be used to optimize marketing strategies, improve product placements, and enhance customer experience.

The purpose of this project is to explore and compare the three frequently used pattern mining algorithms, Brute Force, Apriori, and FP-Growth.

i. The Brute Force algorithm systematically checks all possible itemsets, which, while simple, becomes computationally expensive as the number of items grows.
ii. The Apriori algorithm increases efficiency by making use of the requirement that every subset of a frequent itemset be frequent in and of itself. As a result, there is less room for candidate itemset searches.
iii. A more direct technique is used by the FP-Growth algorithm, which builds the FP-Tree, a compact representation of the dataset, and then mines the tree to identify frequently occurring itemsets.

CORE CONCEPTS AND PRINCIPLES

Project Workflow

This project focuses on mining transaction data with three different algorithms to discover frequent itemsets and derive association rules according to given support and confidence levels. It starts with data preparation by loading and encoding the transactions, followed by executing the algorithms to find itemsets that meet the support criteria and generate rules that satisfy the confidence level. The execution time for each algorithm is measured to compare performance, and results are evaluated for consistency across methods. User inputs are collected to dynamically set parameters and select datasets, ensuring a flexible and interactive workflow.

Data Loading and Preprocessing

Using load_transactions_from_csv, which divides the data into item lists, transactions are loaded from CSV files. These transactions are transformed into a one-hot encoded matrix with encode_transactions for Apriori and FP-Growth, where each row denotes a transaction and each column an item. This stage makes sure the data is prepared so the algorithms can process it effectively.

Frequent Itemset Mining

Brute Force Method

The function "brute_force_frequent_itemsets" generates all possible item combinations from transactions and calculates their support. It filters frequent itemsets based on the minimum support threshold. For large datasets, this method is computationally costly because it thoroughly examines every subset.

Apriori Algorithm

Implemented using the "mlxtend.frequent_patterns.apriori" function, this algorithm optimizes mining by only considering itemsets whose subsets are frequent To handle transactions quickly, it makes use of one-hot encoding.

FP-Growth Algorithm

"mlxtend.frequent_patterns.fpgrowth" is used to build a compact FP-Tree, which avoids generating all candidate itemsets, making it faster than Apriori for larger datasets.

Support Calculation

This calculates how frequently a set of items appears in the dataset. The brute_force_frequent_itemsets function in the Brute Force technique divides the total number of transactions by the number of transactions that contain each candidate itemset (support = count / total_transactions * 100). Built-in functions (mlxtend.apriori and mlxtend.fpgrowth) for the Apriori and FP-Growth algorithms automatically determine support for each frequently occurring itemset as a percentage of transactions that contain the itemset.

Association rule

Brute Force Method

Using the antecedent and consequent divisions, support and confidence are calculated to generate rules from frequently occurring itemsets using the “brute_force_association_rules” function. The minimum confidence threshold is used to filter rules.

Apriori and FP-Growth

Both algorithms streamline the rule generation process by using the confidence threshold and automatically generating rules from frequently occurring itemsets using the mlxtend.frequent_patterns.association_rules function.

Confidence Calculation

Confidence is the probability that a consequent will happen if the antecedent has happened. In the Brute Force method, the brute_force_association_rules function calculates confidence by dividing the support of the entire rule by the support of the antecedent (confidence = support_rule / support_antecedent * 100). The association_rules function automatically calculates confidence for each rule for the Apriori and FP-Growth algorithms by dividing the support of the antecedent by the support of the entire itemset.

Performance Comparison

The code uses time.time() both before and after each algorithm's execution to determine how long it takes to complete. For both frequent itemsets and rules, results from Brute Force, Apriori, and FP-Growth are compared, demonstrating the disparities in their speed and correctness.

One-Hot Encoding for Transactional Data

One-hot encoding is applied to transactions in Apriori and FP-Growth algorithms to transform them into a binary matrix format, which is required for efficient processing by both algorithms.

User Interaction and Input Validation

The code validates user inputs for minimum support and confidence, ensuring they are within acceptable ranges before executing the algorithms.

Results and Evaluation

For small datasets, the brute-force method detects frequent itemsets effectively, but for larger datasets, it becomes computationally expensive. The Apriori algorithm increases performance by removing infrequent itemsets early on, but for huge datasets, its repetitive scans still result in prolonged execution times. The most effective method is FP-Growth, which considerably speeds up execution, especially for larger or more complicated datasets, by using a frequent-pattern tree to prevent repeated scans.

The FP-Growth method has proven to be suitable for large-scale association rule mining, as it routinely outperforms the other two in terms of execution time. The results emphasize the trade-off between simplicity and efficiency, where more sophisticated algorithms like FP-Growth provide better performance, especially in handling complex or large datasets.

CONCLUSION

This project successfully implemented three algorithms: Brute Force, Apriori, and FP-Growth, to find frequent itemsets and generate association rules from transactional data. By comparing the algorithms, we observed that FP-Growth was the most efficient in terms of execution time, especially with larger datasets. The Brute Force method, while effective for small datasets, became computationally expensive as the number of items increased. Apriori offered a balance between efficiency and simplicity, leveraging candidate generation and pruning techniques. The project demonstrates key data mining principles like support, confidence, and optimization, providing valuable insights for real-world market basket analysis.

SCREENSHOTS

Itemsets and Transactions

Transaction ID	Transaction
Trans1	Decorative Pillows, Quilts, Embroidered Bedspread
Trans2	Embroidered Bedspread, Shams, Kids Bedding, Bedding Collections, Bed Skirts, Bedspreads, Sheets
Trans3	Decorative Pillows, Quilts, Embroidered Bedspread, Shams, Kids Bedding, Bedding Collections
Trans4	Kids Bedding, Bedding Collections, Sheets, Bedspreads, Bed Skirts
Trans5	Decorative Pillows, Kids Bedding, Bedding Collections, Sheets, Bed Skirts, Bedspreads
Trans6	Bedding Collections, Bedspreads, Bed Skirts, Sheets, Shams, Kids Bedding
Trans7	Decorative Pillows, Quilts
Trans8	Decorative Pillows, Quilts, Embroidered Bedspread
Trans9	Bedspreads, Bed Skirts, Shams, Kids Bedding, Sheets
Trans10	Quilts, Embroidered Bedspread, Bedding Collections
Trans11	Bedding Collections, Bedspreads, Bed Skirts, Kids Bedding, Shams, Sheets
Trans12	Decorative Pillows, Quilts
Trans13	Embroidered Bedspread, Shams
Trans14	Sheets, Shams, Bed Skirts, Kids Bedding
Trans15	Decorative Pillows, Quilts
Trans16	Decorative Pillows, Kids Bedding, Bed Skirts, Shams
Trans17	Decorative Pillows, Shams, Bed Skirts
Trans18	Quilts, Sheets, Kids Bedding
Trans19	Shams, Bed Skirts, Kids Bedding, Sheets
Trans20	Decorative Pillows, Bedspreads, Shams, Sheets, Bed Skirts, Kids Bedding

Fig 1: K-Mart Dataset

Transaction ID	Transaction
Trans1	A Beginners Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans2	A Beginners Guide, Java: The Complete Reference, Java For Dummies
Trans3	A Beginners Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans4	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition , Beginning Programming with Java
Trans5	Android Programming: The Big Nerd Ranch, Beginning Programming with Java, Java 8 Pocket Guide
Trans6	A Beginners Guide, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans7	A Beginners Guide, Head First Java 2nd Edition , Beginning Programming with Java
Trans8	Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans9	Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition , Beginning Programming with Java
Trans10	Beginning Programming with Java, Java 8 Pocket Guide, C++ Programming in Easy Steps
Trans11	A Beginners Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans12	A Beginners Guide, Java: The Complete Reference, Java For Dummies, HTML and CSS: Design and Build Websites
Trans13	A Beginners Guide, Java: The Complete Reference, Java For Dummies, Java 8 Pocket Guide, HTML and CSS: Design and Build Websites
Trans14	Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans15	Java For Dummies, Android Programming: The Big Nerd Ranch
Trans16	A Beginners Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans17	A Beginners Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
Trans18	Head First Java 2nd Edition , Beginning Programming with Java, Java 8 Pocket Guide
Trans19	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans20	A Beginners Guide, Java: The Complete Reference, Java For Dummies

Fig 2: Amazon Dataset

Transaction ID	Transaction
Trans1	Desk Top, Printer, Flash Drive, Microsoft Office, Speakers, Anti-Virus
Trans2	Lab Top, Flash Drive, Microsoft Office, Lab Top Case, Anti-Virus
Trans3	Lab Top, Printer, Flash Drive, Microsoft Office, Anti-Virus, Lab Top Case, External Hard-Drive
Trans4	Lab Top, Printer, Flash Drive, Anti-Virus, External Hard-Drive, Lab Top Case
Trans5	Lab Top, Flash Drive, Lab Top Case, Anti-Virus
Trans6	Lab Top, Printer, Flash Drive, Microsoft Office
Trans7	Desk Top, Printer, Flash Drive, Microsoft Office
Trans8	Lab Top, External Hard-Drive, Anti-Virus
Trans9	Desk Top, Printer, Flash Drive, Microsoft Office, Lab Top Case, Anti-Virus, Speakers, External Hard-Drive
Trans10	Digital Camera , Lab Top, Desk Top, Printer, Flash Drive, Microsoft Office, Lab Top Case, Anti-Virus, External Hard-Drive, Speakers
Trans11	Lab Top, Desk Top, Lab Top Case, External Hard-Drive, Speakers, Anti-Virus
Trans12	Digital Camera , Lab Top, Lab Top Case, External Hard-Drive, Anti-Virus, Speakers
Trans13	Digital Camera , Speakers
Trans14	Digital Camera , Desk Top, Printer, Flash Drive, Microsoft Office
Trans15	Printer, Flash Drive, Microsoft Office, Anti-Virus, Lab Top Case, Speakers, External Hard-Drive
Trans16	Digital Camera, Flash Drive, Microsoft Office, Anti-Virus, Lab Top Case, External Hard-Drive, Speakers
Trans17	Digital Camera , Lab Top, Lab Top Case
Trans18	Digital Camera , Lab Top Case, Speakers
Trans19	Digital Camera , Lab Top, Printer, Flash Drive, Microsoft Office, Speakers, Lab Top Case, Anti-Virus
Trans20	Digital Camera , Lab Top, Speakers, Anti-Virus, Lab Top Case

Fig 3: Best Buy Dataset

Transaction ID	Transaction
Trans1	A, B, C
Trans2	A, B, C
Trans3	A, B, C, D
Trans4	A, B, C, D, E
Trans5	A, B, D, E
Trans6	A, D, E
Trans7	A, E
Trans8	A, E
Trans9	A, C, E
Trans10	A, C, E
Trans11	A, C, E

Fig 4: Generic Dataset

Transaction ID	Transaction
Trans1	Running Shoe, Socks, Sweatshirts, Modern Pants
Trans2	Running Shoe, Socks, Sweatshirts
Trans3	Running Shoe, Socks, Sweatshirts, Modern Pants
Trans4	Running Shoe, Sweatshirts, Modern Pants
Trans5	Running Shoe, Socks, Sweatshirts, Modern Pants, Soccer Shoe
Trans6	Running Shoe, Socks, Sweatshirts
Trans7	Running Shoe, Socks, Sweatshirts, Modern Pants, Tech Pants, Rash Guard, Hoodies
Trans8	Swimming Shirt, Socks, Sweatshirts
Trans9	Swimming Shirt, Rash Guard, Dry Fit V-Nick, Hoodies, Tech Pants
Trans10	Swimming Shirt, Rash Guard, Dry
Trans11	Swimming Shirt, Rash Guard, Dry Fit V-Nick
Trans12	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Hoodies, Tech Pants, Dry Fit V-Nick
Trans13	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Tech Pants, Dry Fit V-Nick, Hoodies
Trans14	Running Shoe, Swimming Shirt, Rash Guard, Tech Pants, Hoodies, Dry Fit V-Nick
Trans15	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Dry Fit V-Nick, Rash Guard, Tech Pants
Trans16	Swimming Shirt, Soccer Shoe, Hoodies, Dry Fit V-Nick, Tech Pants, Rash Guard
Trans17	Running Shoe, Socks
Trans18	Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Rash Guard, Tech Pants, Dry Fit V-Nick
Trans19	Running Shoe, Swimming Shirt, Rash Guard
Trans20	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Tech Pants, Rash Guard, Dry Fit V-Nick

Fig 5: Nike Dataset

Loading Transactions from CSV Files

Reads the CSV file, where the column 'Transaction' contains items separated by commas, and returns a list of transactions.

```
# Loading transactions from CSV files
def load_transactions_from_csv(file_path):
    df = pd.read_csv(file_path)
    transactions = df['Transaction'].apply(lambda x: x.split(',')).tolist()
    return transactions
```

Fig 6: Loading Transaction Function

Brute Force Algorithm

The code finds all possible itemsets and calculates their support, keeping only those that meet the minimum support threshold. It then generates association rules from these frequent itemsets, creating rules with confidence levels that meet or exceed the specified minimum confidence.

```python
# Brute-Force Algorithm

#Finding Itemsets
def brute_force_frequent_itemsets(transactions, min_support):
    unique_items = set(item for sublist in transactions for item in sublist)
    n_transactions = len(transactions)
    frequent_itemsets = []

    for size in range(1, len(unique_items) + 1):
        for itemset in itertools.combinations(unique_items, size):
            count = sum(1 for transaction in transactions if set(itemset).issubset(set(transaction)))
            support = count / n_transactions * 100
            if support >= min_support:
                frequent_itemsets.append((itemset, support))

    return frequent_itemsets
```

Fig 7: BFA to find Frequent Itemsets

```python
#Association Rules
def brute_force_association_rules(frequent_itemsets, transactions, min_confidence):
    n_transactions = len(transactions)
    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 = tuple(sorted(set(itemset) - set(antecedent)))
                antecedent_count = sum(1 for transaction in transactions if set(antecedent).issubset(set(transaction)))
                confidence = support / (antecedent_count / n_transactions * 100) * 100
                if confidence >= min_confidence:
                    rules.append((antecedent, consequent, support, confidence))

    return rules
```

Fig 8: BFA to generate Association Rules

Apriori Algorithm

Generates association rules by applying the Apriori algorithm to locate frequently occurring itemsets using the “mlxtend” package. The confidence and support thresholds are expressed as percentages.

```python
# Apriori Algorithm

def run_apriori_algorithm(transactions, min_support, min_confidence):
    df_encoded = encode_transactions(transactions)
    frequent_itemsets = apriori(df_encoded, min_support=min_support/100, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence/100)
    return frequent_itemsets, rules
```

Fig 9: Apriori Algorithm for Rule Generation

FP-Growth Algorithm

Generates association rules and locates frequently occurring itemsets using the FP-Growth algorithm from "mlxtend". Large datasets typically respond more quickly to this method.

```python
#FP-Growth Algorithm

def run_fpgrowth_algorithm(transactions, min_support, min_confidence):
    df_encoded = encode_transactions(transactions)
    frequent_itemsets = fpgrowth(df_encoded, min_support=min_support/100, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence/100)
    return frequent_itemsets, rules
```

Fig 10: FP-Growth for Rule Generation

Comparing Results Between Algorithms

Compares the association rules produced by each algorithm to check if they are consistent.

```python
# Compare results between algorithms
def compare_results(bf_rules, apriori_rules, fpgrowth_rules):
    bf_set = set((tuple(r[0]), tuple(r[1])) for r in bf_rules)
    apriori_set = set((tuple(r['antecedents']), tuple(r['consequents'])) for _, r in apriori_rules.iterrows())
    fpgrowth_set = set((tuple(r['antecedents']), tuple(r['consequents'])) for _, r in fpgrowth_rules.iterrows())

    print("\nAre the association rules the same between the algorithms?")
    print("Brute-Force vs Apriori: ", bf_set == apriori_set)
    print("Brute-Force vs FP-Growth: ", bf_set == fpgrowth_set)
    print("Apriori vs FP-Growth: ", apriori_set == fpgrowth_set)
```

Fig 11: Function to compare algorithms

Measuring Performance and Running Comparisons

Measures the execution time for each algorithm, runs them on the chosen dataset, and compares the results.

```python
# Main
def compare_algorithms(store, min_support, min_confidence):
    if store == 1:
        file_path = "amazon.csv"
    elif store == 2:
        file_path = "bestbuy.csv"
    elif store == 3:
        file_path = "kmart.csv"
    elif store == 4:
        file_path = "nike.csv"
    elif store == 5:
        file_path = "generic.csv"
    else:
        print("Invalid store selection.")
        return

    transactions = load_transactions_from_csv(file_path)

    #Brute Force Itemsets
    bf_frequent_itemsets = brute_force_frequent_itemsets(transactions, min_support)
    bf_rules = brute_force_association_rules(bf_frequent_itemsets, transactions, min_confidence)
    format_brute_force_output(bf_frequent_itemsets, bf_rules)

    #Apriori Itemsets
    apriori_frequent_itemsets, apriori_rules = run_apriori_algorithm(transactions, min_support, min_confidence)
    format_apriori_output(apriori_frequent_itemsets, apriori_rules)

    #FP-Growth Itemsets
    fpgrowth_frequent_itemsets, fpgrowth_rules = run_fpgrowth_algorithm(transactions, min_support, min_confidence)
    format_fpgrowth_output(fpgrowth_frequent_itemsets, fpgrowth_rules)
```

Fig 12: Displaying Itemsets

```python
    #TIME- Brute Force
    start_time = time.time()
    bf_frequent_itemsets = brute_force_frequent_itemsets(transactions, min_support)
    bf_rules = brute_force_association_rules(bf_frequent_itemsets, transactions, min_confidence)
    brute_force_time = time.time() - start_time
    print(f"\nBrute Force Execution Time: {brute_force_time:.4f} seconds")

    #TIME- Apriori
    start_time = time.time()
    apriori_frequent_itemsets, apriori_rules = run_apriori_algorithm(transactions, min_support, min_confidence)
    apriori_time = time.time() - start_time
    print(f"Apriori Execution Time: {apriori_time:.4f} seconds")

    #TIME-  FP-Growth
    start_time = time.time()
    fpgrowth_frequent_itemsets, fpgrowth_rules = run_fpgrowth_algorithm(transactions, min_support, min_confidence)
    fpgrowth_time = time.time() - start_time
    print(f"FP-Growth Execution Time: {fpgrowth_time:.4f} seconds")

    #Displaying the fastest algorithm
    display_fastest_algorithm(brute_force_time, apriori_time, fpgrowth_time)

    #Comparing results
    compare_results(bf_rules, apriori_rules, fpgrowth_rules)
```

Fig 13: Calculating Time and Comparing Algorithms

User Interaction

Takes user input for selecting the store and setting the support and confidence thresholds, then runs the comparison function.

```python
# Asking for user input
def main():
    while True:
        try:
            store = int(input("Choose the store-\n1. Amazon\n2. Best Buy\n3. K-mart\n4. Nike\n5. Generic\n"))
            if store < 1 or store > 5:
                raise ValueError
            break
        except ValueError:
            print("Invalid store selection. Please choose a number between 1 and 5.")

    while True:
        try:
            min_support = int(input("Enter the minimum support (1-100): "))
            if min_support < 1 or min_support > 100:
                raise ValueError
            break
        except ValueError:
            print("Invalid support value. Please enter a number between 1 and 100.")

    while True:
        try:
            min_confidence = int(input("Enter the minimum confidence (1-100): "))
            if min_confidence < 1 or min_confidence > 100:
                raise ValueError
            break
        except ValueError:
            print("Invalid confidence value. Please enter a number between 1 and 100.")

    compare_algorithms(store, min_support, min_confidence)

if __name__ == "__main__":
    main()
```

Fig 14: Function asking for User Input

Output

```
Choose the store-
1. Amazon
2. Best Buy
3. K-mart
4. Nike
5. Generic
 5
Enter the minimum support (1-100):  50
Enter the minimum confidence (1-100):  50

Brute Force Frequent Itemsets:
Itemset: (' E',), Support: 72.73%
Itemset: ('A',), Support: 100.00%
Itemset: (' C',), Support: 63.64%
Itemset: (' E', 'A'), Support: 72.73%
Itemset: ('A', ' C'), Support: 63.64%

Brute Force Association Rules:
Rule: (' E',) -> ('A',), Support: 72.73%, Confidence: 100.00%
Rule: ('A',) -> (' E',), Support: 72.73%, Confidence: 72.73%
Rule: ('A',) -> (' C',), Support: 63.64%, Confidence: 63.64%
Rule: (' C',) -> ('A',), Support: 63.64%, Confidence: 100.00%

Apriori Frequent Itemsets:
Itemset: (' E',), Support: 72.73%
Itemset: ('A',), Support: 100.00%
Itemset: (' C',), Support: 63.64%
Itemset: (' E', 'A'), Support: 72.73%
Itemset: (' C', 'A'), Support: 63.64%

Apriori Association Rules:
Rule: (' E',) -> ('A',), Support: 72.73%, Confidence: 100.00%
Rule: ('A',) -> (' E',), Support: 72.73%, Confidence: 72.73%
Rule: (' C',) -> ('A',), Support: 63.64%, Confidence: 100.00%
Rule: ('A',) -> (' C',), Support: 63.64%, Confidence: 63.64%
```

Fig 15: Output Part 1

```
FP-Growth Frequent Itemsets:
Itemset: ('A',), Support: 100.00%
Itemset: (' C',), Support: 63.64%
Itemset: (' E',), Support: 72.73%
Itemset: (' C', 'A'), Support: 63.64%
Itemset: (' E', 'A'), Support: 72.73%

FP-Growth Association Rules:
Rule: (' C',) -> ('A',), Support: 63.64%, Confidence: 100.00%
Rule: ('A',) -> (' C',), Support: 63.64%, Confidence: 63.64%
Rule: (' E',) -> ('A',), Support: 72.73%, Confidence: 100.00%
Rule: ('A',) -> (' E',), Support: 72.73%, Confidence: 72.73%

Brute Force Execution Time: 0.0002 seconds
Apriori Execution Time: 0.0037 seconds
FP-Growth Execution Time: 0.0026 seconds

Fastest Algorithm: Brute Force
Execution Time: 0.0002 seconds

Are the association rules the same between the algorithms?
Brute-Force vs Apriori:  True
Brute-Force vs FP-Growth:  True
Apriori vs FP-Growth:  True
```

Fig 16: Output Part 2

OTHER

Repository Link

https://github.com/AnjaliguptaRaghavendra/raghavendra_anjaligupta_midtermproject