NAME: **Avina Nakarmi**
NJIT UCID: **an778**
Email Address: **an778@njit.edu**
September 16, 2024
Professor: **Yasser Abduallah**
CS 634 101 Data Mining

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm Implementation in Retail Data Mining

Abstract:

Introduction

Apriori algorithm is an unsupervised machine learning algorithm used for association rule learning. It works by identifying frequent itemsets in transaction data and using them to generate association rules. This project uses Apriori algorithm to mine association rules in a retail context based on minimum support and confidence. It also explores the efficiency claims of the FP-Growth algorithm compared to the classic Apriori algorithm. The main goal of the project is to test the reliability and efficiency of different approaches (manual Apriori implementation and packaged Apriori/FP-Growth) to mine association rules in retail transaction databases.

Association rule mining is important in data analysis since it uncovers the hidden relationship between items in a very large dataset, thus helping decision-makers come up with meaningful patterns that may not have been obvious. It finds broad applications in areas such as retail, where the analysis of the market basket can identify sets of items that are usually bought together. The outcomes will lead to better decision-making on product placement, cross-selling opportunities, and targeted marketing strategies.

Since it is an unsupervised learning technique, association rule mining does not require labeled data; hence, it is flexible and can be easily applied to many kinds of transactional databases. Apriori is the most popular algorithm used for mining association rules due to the simplicity of its concept and efficiency in handling large datasets through iterative identification of frequent itemsets.

Problem Definition

The datasets used for the project contain a list of transactions in different retail stores; each transaction is uniquely identified by a transaction ID, which contains a set of items bought together. The dataset used in this analysis contains 20 transactions with 10 unique items, where each transaction represents one set of items that were purchased together.

These datasets were selected for their simplicity and suitability for illustrating association rule mining techniques, such as the Apriori and FP-Growth algorithms. Its size allows for easy experimentation and comparison between different algorithmic approaches while still reflecting the fundamental patterns seen in larger real-world retail datasets.

In this project, the user-defined parameters for generating association rules were the support and confidence thresholds. These parameters play a critical role in filtering the itemsets and rules that are considered relevant during the mining process.

Support measures how frequently an itemset appears in the dataset. It can be defined as:

$$\text{Support}(I1) = \frac{\text{Occurrences of } I1}{\text{Total number of transactions}}$$

Setting a high support threshold helps eliminate rare combinations of items that are not significant in the analysis.

Confidence measures the likelihood that a second item will appear in a transaction given the presence of the first. It can be mathematically defined as:

$$\text{Confidence}(I1, I2) = \frac{\text{Support }(I1, I2)}{\text{Support}(I1)}$$

Confidence ensures that only strong associations between itemsets are considered.

Support and confidence together define the goodness of a rule. Thresholds should be carefully selected: very high thresholds may filter out relevant associations, and lower thresholds may yield a very large number of rules where most are not important.

Methodology

Apriori Algorithm

1. **Dataset Selection**: The program begins by asking the user to choose a dataset from among several available options. Each dataset is represented by an input number.
2. **Input Parameters**: The user is then required to input the minimum support and minimum confidence thresholds, both of which should be values between 0 and 1. These parameters are used to filter the frequent itemsets and association rules.

3. **Data Loading**: The datasets, stored in CSV format, are read using the pandas library. The data is loaded into a DataFrame, which is then processed to extract transactions. Each transaction is represented as a list of items.
4. **Generating Frequent Itemsets**:
    a. **Initialization**: Begin by identifying individual items and their support in the dataset.
    b. **Iterative Generation**: Generate itemsets of size 2, 3, and so on, by combining frequent itemsets from the previous iteration.
    c. **Pruning**: For each iteration, calculate the support of the new itemsets. Itemsets with support below the minimum support threshold are discarded.
5. **Creating Association Rules**:
    a. **Rule Generation**: From the set of frequent itemsets, create association rules. Each rule is of the form A→B, where A and B are itemsets.
    b. **Confidence Calculation**: For each rule, compute the confidence as:

    $$\text{Confidence}(A, B) = \frac{\text{Support }(A, B)}{\text{Support}(A)}$$

    c. **Pruning**: Filter out rules with confidence below the minimum confidence threshold.
6. **Display Results**: Finally, the implementation displays the association rules that meet the specified support and confidence thresholds to the user. The rules are presented in a readable format.

Comparing Apriori Algorithm to FP-Growth Algorithm

The key theoretical difference between Apriori and FP-Growth algorithms lies in how they identify frequent itemsets: Apriori uses a candidate generation approach, iteratively creating and testing potential itemsets, while FP-Growth leverages a specialized tree structure called an FP-tree to efficiently discover frequent patterns without the need for explicit candidate generation, making it generally faster and more scalable for large datasets.

The transaction database was first cleaned and transformed to a dataframe with expected structure. The clean dataframe was then passed into `frequent_patterns.apriori` and `frequent_patterns.fpgrowth` functions with user-defined minimum support from the `mlxtend` package to generate frequent items using the respective algorithms. Lastly, the frequent itemsets along with user-defined confidence threshold were passed into the frequent_patterns.association_rules function to obtain a list of association rules.

Reliability and Accuracy Evaluation

Reliability:

The reliability of the custom implementation was evaluated by comparing the association rules it generated with those produced by the mlxtend package. Since the results matched exactly, it suggests that the custom implementation is accurate and aligns with established methods.

Efficiency:

Execution Time:

- FP-Growth (mlxtend): 0.0044 seconds
- Custom Algorithm: 0.0060 seconds
- Apriori (mlxtend): 0.0071 seconds

The efficiency analysis indicates that the FP-Growth algorithm is the fastest, followed by the custom implementation, and then the Apriori algorithm from mlxtend. This performance aligns with the theoretical expectation that FP-Growth should be faster than Apriori due to its more efficient data structure and algorithmic approach.

The custom implementation is reliable, as it produces results consistent with those from established libraries. The efficiency results also suggest that the implementation is competitive, though FP-Growth outperforms all methods in terms of execution time.

Performance Evaluation

Algorithm	Dataset Used	Min_support	Min_confidence	No. of Transaction	No. of Items	Execution Time (in secs)
Custom Apriori	K-Mart	0.25	0.5	20	10	0.0060
Apriori	K-Mart	0.25	0.5	20	10	0.0071
FP-Growth	K-Mart	0.25	0.5	20	10	0.0044
Custom Apriori	Generic	0.46	0.77	11	6	0.0053
Apriori	Generic	0.46	0.77	11	6	0.0055

FP-Growth	Generic	0.46	0.77	11	6	0.0033

Conclusion

The project aimed to implement the Apriori algorithm and carry out the efficiency analysis concerning the given Apriori and FP-Growth implementations of the mlxtend package. Such an efficiency analysis had to be carried out regarding the association rules obtained and computational times of the algorithms.

The results clearly depict that all three applied methodologies, our custom Apriori, Apriori from mlxtend, and FP-Growth returned the same set of rules. This means the custom implementation returned reliable and accurate results as well.

Regarding the performance metrics, execution time is from minimum to maximum. The FP-Growth algorithm was the fastest, taking only 0.0044 seconds to execute. However, in contrast, the Apriori function from mlxtend needed 0.0071 seconds of execution time. The custom implementation of the Apriori algorithm was also efficient: an execution time of 0.0060 seconds. This confirms that, of course, FP-Growth is more efficient compared to Apriori due to the usage of a compressed FP-tree representation which avoids the candidate generation step present in Apriori.

This difference in execution time increases when the size of the dataset is bigger because FP-Growth is designed to become more efficient with larger datasets. On this rather small dataset of 20 transactions of 10 items each, the custom implementation of Apriori performed well in both accuracy and execution speed, taking only slightly more time compared to the one from mlxtend.

In short, the implemented Apriori is reliable, but the FP-Growth Algorithm is more viable when it comes to handling bigger datasets because of the execution time. Hence, further work might be done on optimizing the implementation of Apriori or on the development of hybrid methods that can combine the advantages of both algorithms.

Appendix

Dataset Sample

6) Custom Data Example (Homework Example)

Item #	Item Name
1	ink
2	pen
3	cheese
4	bag
5	juice
6	milk

Table 10 Custom Item Names

ID	Transactions
Trans1	ink, pen, cheese, bag
Trans2	milk, pen, juice, cheese
Trans3	milk, juice
Trans4	juice, milk, cheese
Trans5	ink, pen, cheese, bag
Trans6	milk, pen, juice, cheese
Trans7	milk, ink, cheese
Trans8	pen, juice, bag
Trans9	milk, cheese, bag
Trans10	ink, milk, cheese, juice
Trans11	pen, cheese, bag
Trans12	juice, ink, pen
Trans13	milk, cheese, ink
Trans14	pen, juice, milk
Trans15	Cheese, bag, juice
Trans16	ink, milk, bag
Trans17	cheese, juice, pen
Trans18	milk, pen, cheese
Trans19	bag, juice, milk
Trans20	ink, cheese, juice

.

Code snippets with output

Data Selection

```
datasets = {
    '1': 'Amazon',
    '2': 'Best Buy',
    '3': 'K-Mart',
    '4': 'Nike',
    '5': 'Custom',
    '6': 'Generic',
}


dataset = 0
attempted = 0

def read_dataset_input():
    """
        - On the first attempt, prompts the user to select a dataset.
        - On subsequent attempts, informs the user of an invalid selection if needed.
        - If the user's input matches a key in the `datasets` dictionary, it prints the selected dataset.

        Returns:
        None
    """
    global attempted, dataset
    print("Invalid selection. Try again. \n") if attempted > 0 else print("Select a dataset: \n")
    attempted += 1
    dataset = input(" 1. Amazon \n 2. Best Buy \n 3. K-mart \n 4. Nike \n 5. Custom \n 6. Generic \n")
    if dataset in datasets.keys(): print("You selected: ", datasets[dataset])

input_read_condition = lambda: dataset not in datasets.keys()
do_while(input_read_condition, read_dataset_input)
```

[3] ✓ 9.2s

```
Select a dataset:

You selected:  Custom
```

Defining support and confidence

```python
support = None
confidence = None

def read_threshold():
  """
    - Prompts the user to enter a value for the support threshold.
    - Prompts the user to enter a value for the confidence threshold.

    Returns:
    None
  """
  global support, confidence
  support = input("Enter support threshold: ")
  confidence = input("Enter confidence threshold: ")

def threshold_read_condition():
  """
    - Attempts to convert `support` and `confidence` to floats.
    - If conversion fails (i.e., invalid input types), prints an error message and returns True to
      indicate the need for retrying.
    - Checks whether the values for `support` and `confidence` fall within the valid range [0, 1].
    - Returns False if both values are valid, otherwise returns True and prints an error message.

    Returns:
    bool: True if input is invalid (either due to type or out-of-range values), False otherwise.
  """
  global support, confidence
  try:
    support = float(support)
    confidence = float(confidence)
  except ValueError:
    print("Invalid input type. Try again.")
    return True
  if 0 <= support <= 1 and 0 <= confidence <= 1:
    return False
  print("Invalid input range. Try again.")
  return True
do_while(threshold_read_condition, read_threshold)

print("Generating association rules for ", datasets[dataset], " dataset with support: ", support, " and confidence: ", confidence)
```

[12] ✓ 6.7s

```
Generating association rules for  Custom  dataset with support:  0.25  and confidence:  0.5
```

Reading Transactions

```python
import pandas as pd

raw_dataset = pd.read_csv('./datasets/' + datasets[dataset] + '.csv', usecols=[1])
transactions_list = [transaction[0].split(',') for transaction in raw_dataset.values.tolist()]
transactions_list = [[item.strip() for item in transaction] for transaction in transactions_list]

print(raw_dataset)
```

[5] ✓ 0.5s

```
                 Transactions
0      ink, pen, cheese, bag
1    milk, pen, juice, cheese
2                 milk, juice
3         juice, milk, cheese
4      ink, pen, cheese, bag
5    milk, pen, juice, cheese
6           milk, ink, cheese
7             pen, juice, bag
8           milk, cheese, bag
9    ink, milk, cheese, juice
10           pen, cheese, bag
11            juice, ink, pen
12          milk, cheese, ink
13           pen, juice, milk
14         Cheese, bag, juice
15              ink, milk, bag
16         cheese, juice, pen
17          milk, pen, cheese
18           bag, juice, milk
19         ink, cheese, juice
```

Apriori using mlxtend

```python
from mlxtend.frequent_patterns import apriori, association_rules

start_time = time.time()

all_items = set(item for sublist in transactions_list for item in sublist)

df = pd.DataFrame([{item: (item in transaction) for item in all_items} for transaction in transactions_list])

frequent_itemsets = apriori(df, min_support=support, use_colnames=True)

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=confidence)

rules_sorted = rules.sort_values(by='lift', ascending=False)

rules_filtered = rules[['antecedents', 'consequents', 'support', 'confidence']]
rules_filtered.columns = ['Antecedents', 'Consequents', 'Support', 'Confidence']

print(rules_filtered)

end_time = time.time()
print("Time taken: ", end_time - start_time, " seconds")
```

[13] ✓ 0.0s

```
  Antecedents Consequents  Support  Confidence
0      (milk)    (cheese)     0.40    0.666667
1    (cheese)      (milk)     0.40    0.615385
2      (milk)     (juice)     0.35    0.583333
3     (juice)      (milk)     0.35    0.583333
4       (ink)    (cheese)     0.30    0.750000
5     (juice)    (cheese)     0.30    0.500000
6    (cheese)       (pen)     0.35    0.538462
7       (pen)    (cheese)     0.35    0.700000
8     (juice)       (pen)     0.30    0.500000
9       (pen)     (juice)     0.30    0.600000
Time taken:  0.010127067565917969  seconds
```

FP-growth Using mlxtend

```python
from mlxtend.frequent_patterns import fpgrowth, association_rules

start_time = time.time()
frequent_itemsets = fpgrowth(df, min_support=support, use_colnames=True)

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=confidence)

rules_filtered = rules[['antecedents', 'consequents', 'support', 'confidence']]
rules_filtered.columns = ['Antecedents', 'Consequents', 'Support', 'Confidence']

print(rules_filtered)

end_time = time.time()
print("Time taken: ", end_time - start_time, " seconds")
```

[14] ✓ 0.0s

```
  Antecedents Consequents  Support  Confidence
0    (cheese)       (pen)     0.35    0.538462
1       (pen)    (cheese)     0.35    0.700000
2     (juice)       (pen)     0.30    0.500000
3       (pen)     (juice)     0.30    0.600000
4       (ink)    (cheese)     0.30    0.750000
5     (juice)    (cheese)     0.30    0.500000
6      (milk)    (cheese)     0.40    0.666667
7    (cheese)      (milk)     0.40    0.615385
8      (milk)     (juice)     0.35    0.583333
9     (juice)      (milk)     0.35    0.583333
Time taken:  0.00843501091003418  seconds
```

Implementing Apiori

Generating frequent items

```
from itertools import combinations

min_sup = support * len(transactions_list)
freq_itemset_support = {}


def count_item_freq(itemsets):
    """
    - Iterates over each transaction in `transactions_list` and checks whether each itemset
      is present in the transaction.
    - If all items in an itemset are found within a transaction, increments the count for that
      itemset in the `itemset_support` dictionary.
    - Uses the `itemset_support` dictionary to store the frequency of each itemset.

    Returns:
    dict: A dictionary where the keys are itemsets and the values are their corresponding frequencies.
    """
    itemset_support = {}
    for transaction in transactions_list:
        for itemset in itemsets:
            for item in itemset:
                if item not in transaction:
                    break
            else:
                itemset_support[itemset] = itemset_support.get(itemset, 0) + 1
    return itemset_support


def prune_items(last_freq_itemset):
    """
    - Iterates over each itemset and its support in `last_freq_itemset`.
    - Filters out itemsets whose support is below `min_sup`.
    - For each retained itemset, calculates its relative support as the ratio of its count to the
      total number of transactions.

    Returns:
    dict: A dictionary where the keys are itemsets and the values are their relative support
          (calculated as support count divided by the total number of transactions), for itemsets
          that meet or exceed the minimum support threshold.
    """
    return {itemset:(sup/len(transactions_list)) for itemset,sup in last_freq_itemset.items() if sup >= min_sup}
```

```
def make_n_itemset(n_itemset):
    """
    - Extracts unique items from the provided `n_itemset`.
      - Creates (n+1)-itemsets by combining these unique items.
      - Returns a list of all possible (n+1)-itemsets.

      Example:
      If `n_itemset` contains itemsets like [('A', 'B'), ('A', 'C')], the function will generate
      (n+1)-itemsets like [('A', 'B', 'C')] if 'A', 'B', 'C' are the unique items.
    """
    n = len(n_itemset[0])
    return list(combinations(list(set(item for s in n_itemset for item in s)), n + 1))

start_time = time.time()

new_item_set_list = list(set((item,) for transaction in transactions_list for item in transaction))

while new_item_set_list:
    itemset_support = count_item_freq(new_item_set_list)
    freq_itemsets = prune_items(itemset_support)
    freq_itemset_support.update(freq_itemsets)
    if len(freq_itemsets) == 0:
        break
    new_item_set_list = make_n_itemset(list(freq_itemsets.keys()))

for itemset, sup in freq_itemset_support.items():
    print(itemset, sup)
```

[24] ✓ 0.0s

```
('cheese',) 0.65
('ink',) 0.4
('pen',) 0.5
('bag',) 0.4
('juice',) 0.6
('milk',) 0.6
('ink', 'cheese') 0.3
('cheese', 'pen') 0.35
('milk', 'cheese') 0.4
('milk', 'juice') 0.35
('cheese', 'juice') 0.3
('juice', 'pen') 0.3
```

Mining Association Rules

```
index = 1
for itemset, sup in freq_itemset_support.items():
    if len(itemset) < 2:
        continue
    for i in range(1, len(itemset)):
        for antecedent in combinations(itemset, i):
            consequent = tuple(set(itemset) - set(antecedent))
            conf = freq_itemset_support[itemset] / freq_itemset_support[antecedent]
            if conf >= confidence:
                print("Rule ", index, ": ", antecedent, "->", consequent)
                print("Confidence: ", conf*100, "%")
                print("Support: ", freq_itemset_support[itemset]*100, "%")
                print("\n")
                index += 1

end_time = time.time()
print("Time taken: ", end_time - start_time, " seconds")
```

[25] ✓ 0.0s

```
Rule  1 :  ('ink',) -> ('cheese',)
Confidence:  74.99999999999999 %
Support:  30.0 %


Rule  2 :  ('cheese',) -> ('pen',)
Confidence:  53.84615384615385 %
Support:  35.0 %


Rule  3 :  ('pen',) -> ('cheese',)
Confidence:  70.0 %
Support:  35.0 %


Rule  4 :  ('milk',) -> ('cheese',)
Confidence:  66.66666666666667 %
Support:  40.0 %


Rule  5 :  ('cheese',) -> ('milk',)
Confidence:  61.53846153846154 %
Support:  40.0 %


Rule  6 :  ('milk',) -> ('juice',)
Confidence:  58.333333333333336 %
Support:  35.0 %


Rule  7 :  ('juice',) -> ('milk',)
Confidence:  58.333333333333336 %
Support:  35.0 %


Rule  8 :  ('juice',) -> ('cheese',)
Confidence:  50.0 %
Support:  30.0 %


Rule  9 :  ('juice',) -> ('pen',)
Confidence:  50.0 %
Support:  30.0 %


Rule  10 :  ('pen',) -> ('juice',)
Confidence:  60.0 %
Support:  30.0 %


Time taken:  0.006052255630493164  seconds
```

<u>Generated rules with goodness of rule</u>

- Association rules for Amazon dataset with support: 0.5 and confidence: 0.5
    - Apriori (mlxtend)

```
                     Antecedents                   Consequents  Support  \
0  (Java: The Complete Reference)            (Java For Dummies)      0.5
1              (Java For Dummies)  (Java: The Complete Reference)      0.5

   Confidence
0    1.000000
1    0.769231
Time taken:  0.006206035614013672  seconds
```

    - FP-Growth

```
                     Antecedents                   Consequents  Support  \
0  (Java: The Complete Reference)            (Java For Dummies)      0.5
1              (Java For Dummies)  (Java: The Complete Reference)      0.5

   Confidence
0    1.000000
1    0.769231
Time taken:  0.0069310665130615234  seconds
```

    - Apriori (custom)

```
Rule  1 :  ('Java: The Complete Reference',) -> ('Java For Dummies',)
Confidence:  100.0 %
Support:  50.0 %


Rule  2 :  ('Java For Dummies',) -> ('Java: The Complete Reference',)
Confidence:  76.92307692307692 %
Support:  50.0 %


Time taken:  0.007889270782470703  seconds
```

- Association rules for Best Buy dataset with support: 0.6 and confidence: 0.3
    - Apriori (mlxtend)

```
     Antecedents     Consequents  Support  Confidence
0  (Lab Top Case)    (Anti-Virus)      0.6    0.857143
1    (Anti-Virus)  (Lab Top Case)      0.6    0.857143
Time taken:  0.0060689449310302734  seconds
```

    - FP-Growth

```
     Antecedents     Consequents  Support  Confidence
0  (Lab Top Case)    (Anti-Virus)      0.6    0.857143
1    (Anti-Virus)  (Lab Top Case)      0.6    0.857143
Time taken:  0.003200054168701172  seconds
```

- Apriori (custom)

```
Rule  1 :  ('Lab Top Case',) -> ('Anti-Virus',)
Confidence:  85.71428571428572 %
Support:  60.0 %


Rule  2 :  ('Anti-Virus',) -> ('Lab Top Case',)
Confidence:  85.71428571428572 %
Support:  60.0 %


Time taken:  0.006165266036987305  seconds
```

- Association rules for K-Mart dataset with support: 0.4 and confidence: 0.9
  - Apriori (mlxtend)

```
                Antecedents               Consequents  Support  Confidence
0                  (Sheets)              (Bed Skirts)     0.45    0.900000
1                  (Sheets)            (Kids Bedding)     0.50    1.000000
2              (Bed Skirts)            (Kids Bedding)     0.50    0.909091
3      (Sheets, Bed Skirts)            (Kids Bedding)     0.45    1.000000
4    (Sheets, Kids Bedding)              (Bed Skirts)     0.45    0.900000
5  (Bed Skirts, Kids Bedding)                (Sheets)     0.45    0.900000
6                  (Sheets)  (Bed Skirts, Kids Bedding)     0.45    0.900000
Time taken:  0.0065000057220458984  seconds
```

  - FP-Growth

```
                Antecedents               Consequents  Support  Confidence
0              (Bed Skirts)            (Kids Bedding)     0.50    0.909091
1                  (Sheets)            (Kids Bedding)     0.50    1.000000
2                  (Sheets)              (Bed Skirts)     0.45    0.900000
3      (Sheets, Bed Skirts)            (Kids Bedding)     0.45    1.000000
4    (Sheets, Kids Bedding)              (Bed Skirts)     0.45    0.900000
5  (Bed Skirts, Kids Bedding)                (Sheets)     0.45    0.900000
6                  (Sheets)  (Bed Skirts, Kids Bedding)     0.45    0.900000
Time taken:  0.008450746536254883  seconds
```

- Apriori (custom)

```
Rule  1 :  ('Sheets',) -> ('Bed Skirts',)
Confidence:  90.0 %
Support:  45.0 %


Rule  2 :  ('Sheets',) -> ('Kids Bedding',)
Confidence:  100.0 %
Support:  50.0 %


Rule  3 :  ('Bed Skirts',) -> ('Kids Bedding',)
Confidence:  90.9090909090909 %
Support:  50.0 %


Rule  4 :  ('Sheets',) -> ('Bed Skirts', 'Kids Bedding')
Confidence:  90.0 %
Support:  45.0 %


Rule  5 :  ('Sheets', 'Bed Skirts') -> ('Kids Bedding',)
Confidence:  100.0 %
Support:  45.0 %


Rule  6 :  ('Sheets', 'Kids Bedding') -> ('Bed Skirts',)
Confidence:  90.0 %
Support:  45.0 %

Rule  7 :  ('Bed Skirts', 'Kids Bedding') -> ('Sheets',)
Confidence:  90.0 %
Support:  45.0 %


Time taken:  0.007708072662353516  seconds
```

- Association rules for Nike dataset with support: 0.58 and confidence: 0.7
  - Apriori (mlxtend)

```
     Antecedents    Consequents  Support  Confidence
0        (Socks)  (Sweatshirts)      0.6    0.923077
1  (Sweatshirts)        (Socks)      0.6    0.923077
Time taken:  0.005515098571777344  seconds
```

  - FP-Growth

```
     Antecedents    Consequents  Support  Confidence
0        (Socks)  (Sweatshirts)      0.6    0.923077
1  (Sweatshirts)        (Socks)      0.6    0.923077
Time taken:  0.003181934356689453  seconds
```

- Apriori (custom)

```
Rule  1 :  ('Socks',) -> ('Sweatshirts',)
Confidence:  92.3076923076923 %
Support:  60.0 %


Rule  2 :  ('Sweatshirts',) -> ('Socks',)
Confidence:  92.3076923076923 %
Support:  60.0 %


Time taken:  0.0061359405517578125  seconds
```

- Association rules for Generic dataset with support: 0.46 and confidence: 0.77
  - Apriori (mlxtend)

```
  Antecedents Consequents  Support  Confidence
0         (C)          (A)     0.55    0.846154
Time taken:  0.005556106567382812 5  seconds
```

  - FP-Growth

```
  Antecedents Consequents  Support  Confidence
0         (C)          (A)     0.55    0.846154
Time taken:  0.0033380985260009766  seconds
```

  - Apriori (custom)

```
Rule  1 :  ('C',) -> ('A',)
Confidence:  84.61538461538461 %
Support:  55.00000000000001 %


Time taken:  0.005317211151123047  seconds
```

Execution Instructions

- Requirements:
  - Pandas
  - mlxtend (for Apriori and FP-Growth algorithms)
  - Itertools

  To install the necessary packages, you can use the following commands:

  pip install pandas mlxtend itertools
- Usage:
  - Dataset Selection: The notebook will prompt you to select a dataset from a list. Enter the corresponding number to choose the dataset.

    - Thresholds Input: Provide support and confidence thresholds when prompted. The values should be between 0 and 1.
    - Transaction Processing: The notebook will read and process the transactions from the selected dataset.
- Output:
  The notebook outputs the association rules generated by both the Apriori and FP-Growth algorithms, including metrics such as support, and confidence.
- Notes:
    - Ensure that the dataset CSV files are in the "./datasets/" directory.
    - The notebook is designed to be interactive and will prompt user inputs.

Link to GitHub Repository

https://github.com/avinanakarmi/CS634_MidTermProject_Apriori.git