CS 6220 FALL 2024: HOME WORK ASSIGNMENT 4

Name: Hima Varshini Parasa
GTID: 903945136
Task: Comparing two different frequent pattern mining algorithms (Problem 1.2)

DELIVERABLES:

1. **Source Code files:** "apfpGroceryDataset.py" and "apfpSNSDataset.py".
2. **Output Files:** "GroceryDatasetOutput.txt" and "SNSDatasetOutput.py".
3. **Source code files for visualization:** "VisualizationOfEachAlgo.py" and "ComparingVisualizations.py".
4. **How to Run:** Save the .py files and run the programs using the command: python <file-name>.py . Prerequisite: Ensure python and necessary packages are involved. List of used packages is given under "Implementation" section of this report.
5. **Setting of minimum support:** The threshold refers to the minimum support configured to test against both the algorithms. In the source code, this is set as 0.01 when the functions are called. This means the minimum support threshold is 0.01, or 1%. The frequency of the items must be at least 1% to be considered as a frequent dataitem, else the dataitem is discarded.

What does this report contain?

CONTEXT & TERMINOLOGY

Frequent pattern mining algorithms are designed to discover patterns, associations, correlations, or commonly occurring combinations among large data sets of items in transactional databases, relational databases, or other data repositories. These algorithms have become very important in data mining, finding considerable application in market basket analysis to identify products frequently appearing in the same transaction.

In this assignment, we implement most commonly used frequent pattern mining algorithms - Apriori and Frequent Pattern Growth (FP-Growth) and compare the results in terms of execution time and memory usage.

Apriori Algorithm:

This uses prior knowledge of frequent itemset properties. Frequent item sets are items in a dataset that appear together in a certain frequency. This frequency of the items is termed as "Support". Support represents the number of times a certain item occurred throughout the entire dataset. For example, in the output if the support for a certain item is given as 0.03, it implies that the dataset comprises 3% of this particular item.

Apriori Algorithm also returns apriori rules - which represents the popular combinations that frequently occurred together in the dataset. Suppose A → B is an apriori rule, 'A' in this relationship is called 'Antecedent' whereas 'B' is termed as 'Consequent'. For such rules, we measure the following metrics through this algorithm:

a) Antecedent Support: The frequency of the combination of A, B occurring together is termed as 'Antecedent support'.

b) Leverage: It represents the coincidence of apriori relationships. A higher leverage implies stronger relationship (i.e., more likely that the two items occur together and its not a mere coincidence). This metric is calculated by calculating the difference between the observed co-occurrence and the expected co-occurrence of the antecedents and the consequents.

c) Conviction: Measured by the ratio of the number of occurrences of antecedents and the number of occurrences of the antecedents and the consequents combined together. If the ratio is greater than 1, it implies a positive relationship - higher chances of the combination of items occurring together. If the ratio is equal to 1, no meaning inference can be deduced from the relationship. Whereas if the ratio is less than 1, then it implies that the occurrences in that combination is less likely to appear together in the dataset.

d) <u>Zhangs_metric:</u> This measures the strength of the relationship as well by evaluating the chances of occurrences of the consequent when the antecedent occurs and when it does not. This is calculated by dividing the probability of both the antecedent and consequent occurring together by the product of their individual probabilities,

These metrics together help evaluate the strength of apriori relationships, hence giving meaningful interpretation from the datasets.

FP-Growth Algorithm:

While FP-Growth essentially returns the same results and outputs are evaluated using the same metrics (support, leverage, conviction, zhang_metrics), the key difference between these algorithms is the relative efficiency improvement in the FP-Growth algorithm. In the Apriori algorithm, the algorithm scans the entire database at each step to select the candidate sets, making it inefficient and significantly slow. FP-Growth algorithm efficiently overcomes these drawbacks by implementing a tree data structure storing the items and its frequency in a key-value set and in descending order while removing those below a specified minimum support threshold. Here, each transaction is represented as a path in the tree, where nodes are items, and edges indicate the presence of items in a transaction. Nodes are linked together to represent common prefixes, allowing for a compact representation of the dataset. The frequent patterns are then mined from this data structure. Here, the entire data set is parsed only once making it faster than apriori as the dataset in that case is parsed large number of times, making the FP-Growth algorithm more appropriate for large-scale data sets.

DATASETS

For this implementation, we consider two datasets to effectively compare both the algorithms:

1. Open-source Grocery Dataset (9835 entries)

https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/groceries.csv

and another larger dataset (~ x3)

2. Open-source SNS data (30,003 entries)

https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/refs/heads/master/snsdata.csv

The grocery dataset consists of transactions made by multiple customers at a supermarket, with details of items bought together in one purchase. Each of the entries in the dataset represent a basket of items (milk, eggs, bread etc. among others)– this sort of data analysis usually gives interesting information on customer buying behavior through market basket analysis. The data set is a csv file with transactions of items separated by commas and the number of items across transactions is non uniform as different customers could buy different number of items. To preprocess the data, I created a one-hot encoded dataframe with the transactions represented as rows and the items as the columns in the dataframe, while the binary format (1 or 0) is used to represent if the particular item is bought in a particular transaction. Additionally, the entire data is completely transactional, no numerical data (number of items bought, cost etc.) is given. Hence, no further processing was necessary. The SNS (Social Networking Site) dataset represents user interaction related to their interests across various social activities such as dance, basketball, smoking etc. among others is curated. It is a CSV file with each row representing a particular users' interests while each column corresponds to the social activity. As these algorithms function on binary dataset, I used one-hot encoded dataframe to convert this data into 1s and 0s as well with 1 indicating the user is interested in the activity and 0 indicating not interested while replacing empty values with None.

IMPLEMENTATION

1. **Programming Language:** Python
2. **Libraries Used:**

- Pandas: used for loading and processing the dataset
- mlxtend.frequent_patterns : To implement the apriori, fpgrowth, and association_rules functions for frequent pattern mining
- time: To measure the execution time
- tracemalloc: To track usage of memory

3. **Source Code:**

- Files saves as apvsfp1.py and apvsfp2.py in the zip file submitted.

4. **Steps to Run the files:**

- Install required packages (if not already installed): Pandas, mlxtend (pip install pandas mlxtend).
- Run the source codes: python <filename.py> The dataset is accessed through the URL over the public internet and hence need not be downloaded on local machine for the implementation.

5. **Threshold for Support:**

The threshold refers to the minimum support configured to test against both the algorithms. In the source code, this is set as 0.01 when the functions are called. This means the minimum support threshold is 0.01, or 1%. The frequency of the items must be at least 1% to be considered as a frequent dataitem, else the dataitem is discarded.

6. **How to Run:**

- Save the python programs titled – “apfpGroceryDataset.py” and “apfpSNSDataset.py”.
- Run the programs in your local compiler with command:
  python <file-name>.py
- Repeat the same for visualization python programs as well, titled – “VisualizationOfEachAlgo.py” and “ComparingVisualizations.py”.

EXPECTED OUTPUTS

1. **Frequent Itemsets**

   The items which are noted to be frequently generated through the dataset, along with the support (frequency).

2. **Apriori Rules**

   Associations between the frequent itemsets is returned along with the support and other metrics explained earlier (leverage, conviction, zhangs_metric).

3. **Execution Time:**

   The time taken by each algorithm is returned in seconds.

4. **Memory usage:**

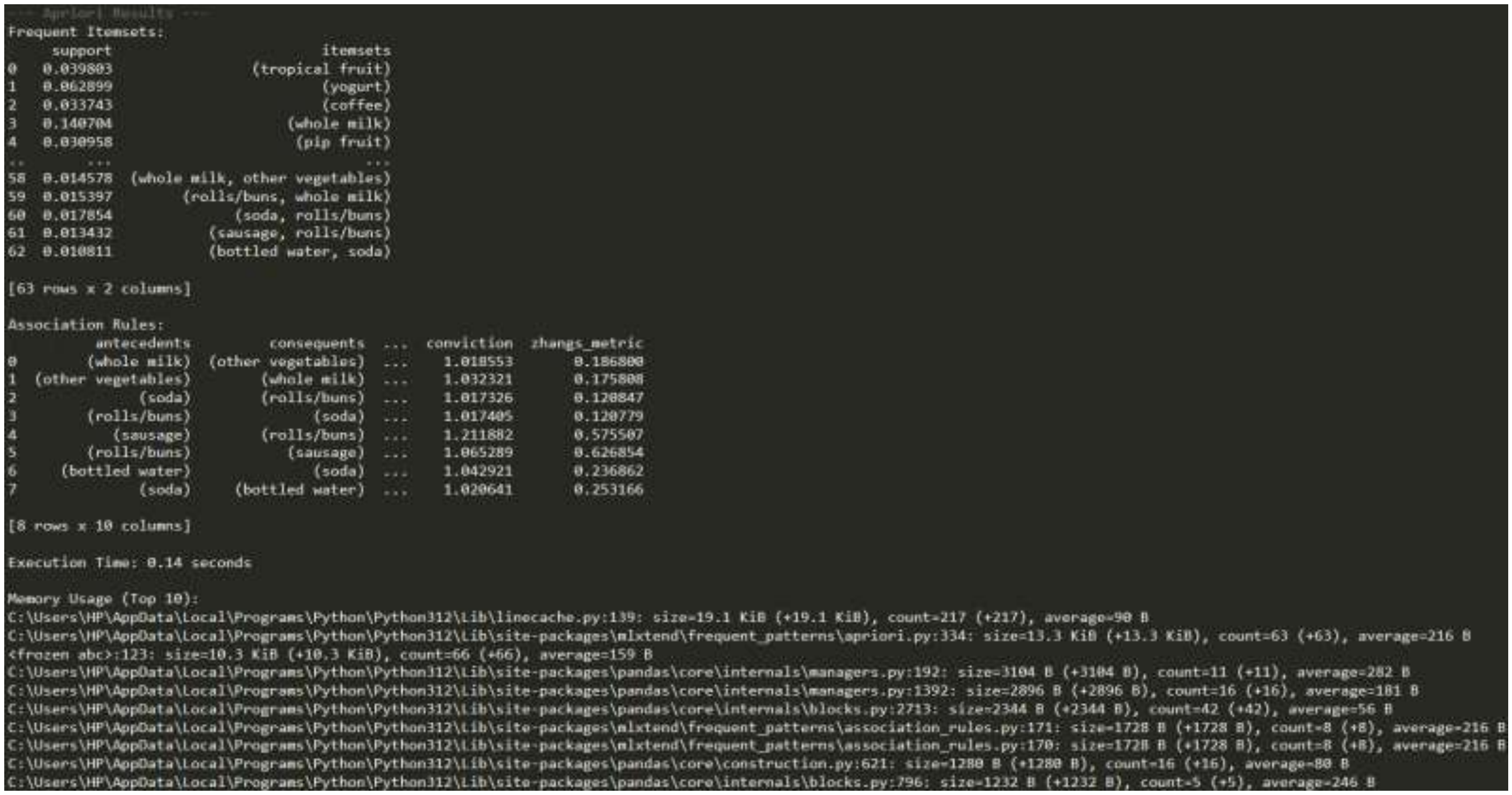
   Returns a list of top 10 most memory consumption of the lines in both algorithms, in descending order.

The outputs returned and compared in this report are stored in the zip file submitted with titles – "GroceryDatasetOutput.txt" and "SNSDatasetOutput.py".

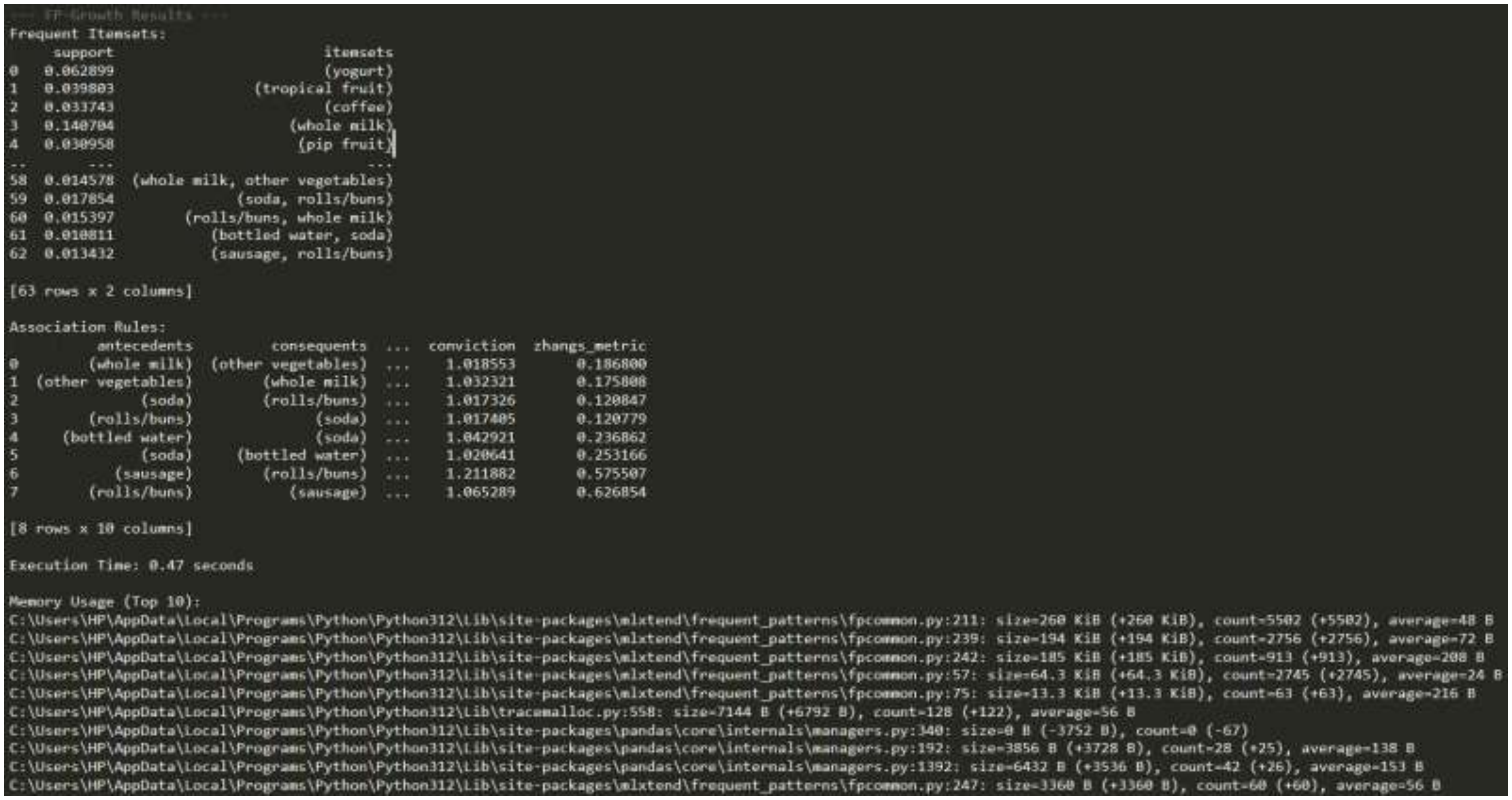
RESULTS & ANALYSIS

1. For Grocery Dataset

Output screenshots:



```
--- Apriori Results ---
Frequent Itemsets:
     support                     itemsets
0   0.039803              (tropical fruit)
1   0.062899                     (yogurt)
2   0.033743                     (coffee)
3   0.140704                 (whole milk)
4   0.030958                  (pip fruit)
..       ...                          ...
58  0.014578  (whole milk, other vegetables)
59  0.015397        (rolls/buns, whole milk)
60  0.017854              (soda, rolls/buns)
61  0.013432           (sausage, rolls/buns)
62  0.010811           (bottled water, soda)

[63 rows x 2 columns]

Association Rules:
          antecedents         consequents  ...  conviction  zhangs_metric
0        (whole milk)  (other vegetables)  ...    1.018553       0.186800
1  (other vegetables)        (whole milk)  ...    1.032321       0.175808
2              (soda)        (rolls/buns)  ...    1.017326       0.120847
3        (rolls/buns)              (soda)  ...    1.017405       0.120779
4           (sausage)        (rolls/buns)  ...    1.211882       0.575507
5        (rolls/buns)           (sausage)  ...    1.065289       0.626854
6     (bottled water)              (soda)  ...    1.042921       0.236862
7              (soda)     (bottled water)  ...    1.020641       0.253166

[8 rows x 10 columns]

Execution Time: 0.14 seconds

Memory Usage (Top 10):
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\linecache.py:139: size=19.1 KiB (+19.1 KiB), count=217 (+217), average=90 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\apriori.py:334: size=13.3 KiB (+13.3 KiB), count=63 (+63), average=216 B
<frozen abc>:123: size=10.3 KiB (+10.3 KiB), count=66 (+66), average=159 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\managers.py:192: size=3104 B (+3104 B), count=11 (+11), average=282 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\managers.py:1392: size=2896 B (+2896 B), count=16 (+16), average=181 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\blocks.py:2713: size=2344 B (+2344 B), count=42 (+42), average=56 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:171: size=1728 B (+1728 B), count=8 (+8), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:170: size=1728 B (+1728 B), count=8 (+8), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\construction.py:621: size=1280 B (+1280 B), count=16 (+16), average=80 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\blocks.py:796: size=1232 B (+1232 B), count=5 (+5), average=246 B
```

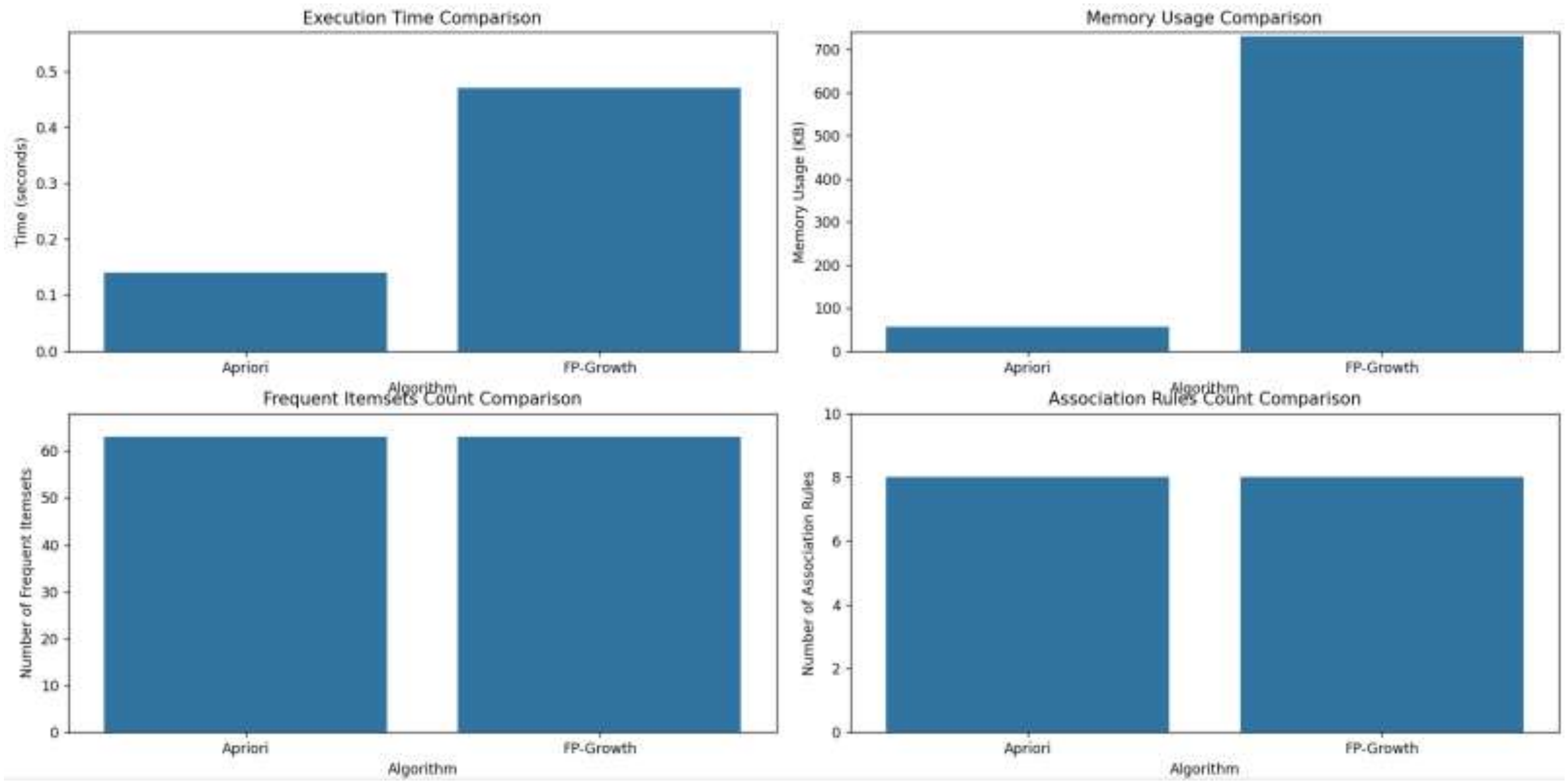


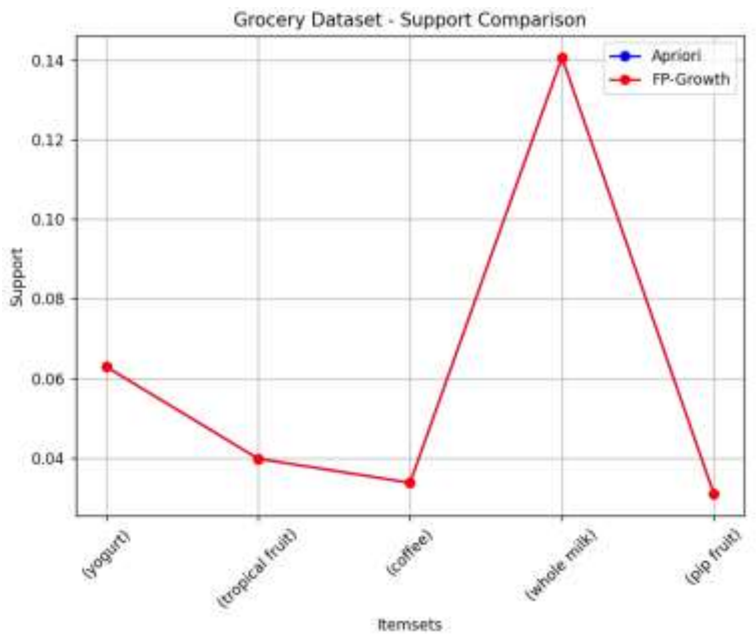
```
--- FP-Growth Results ---
Frequent Itemsets:
     support                     itemsets
0   0.062899                     (yogurt)
1   0.039803              (tropical fruit)
2   0.033743                     (coffee)
3   0.140704                 (whole milk)
4   0.030958                  (pip fruit)
..       ...                          ...
58  0.014578  (whole milk, other vegetables)
59  0.017854              (soda, rolls/buns)
60  0.015397        (rolls/buns, whole milk)
61  0.010811           (bottled water, soda)
62  0.013432           (sausage, rolls/buns)

[63 rows x 2 columns]

Association Rules:
          antecedents         consequents  ...  conviction  zhangs_metric
0        (whole milk)  (other vegetables)  ...    1.018553       0.186800
1  (other vegetables)        (whole milk)  ...    1.032321       0.175808
2              (soda)        (rolls/buns)  ...    1.017326       0.120847
3        (rolls/buns)              (soda)  ...    1.017405       0.120779
4     (bottled water)              (soda)  ...    1.042921       0.236862
5              (soda)     (bottled water)  ...    1.020641       0.253166
6           (sausage)        (rolls/buns)  ...    1.211882       0.575507
7        (rolls/buns)           (sausage)  ...    1.065289       0.626854

[8 rows x 10 columns]

Execution Time: 0.47 seconds

Memory Usage (Top 10):
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:211: size=260 KiB (+260 KiB), count=5502 (+5502), average=48 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:239: size=194 KiB (+194 KiB), count=2756 (+2756), average=72 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:242: size=185 KiB (+185 KiB), count=913 (+913), average=208 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:57: size=64.3 KiB (+64.3 KiB), count=2745 (+2745), average=24 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:75: size=13.3 KiB (+13.3 KiB), count=63 (+63), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\tracemalloc.py:558: size=7144 B (+6792 B), count=128 (+122), average=56 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\managers.py:340: size=0 B (-3752 B), count=0 (-67)
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\managers.py:192: size=3856 B (+3728 B), count=28 (+25), average=138 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\managers.py:1392: size=6432 B (+3536 B), count=42 (+26), average=153 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:247: size=3360 B (+3360 B), count=60 (+60), average=56 B
```

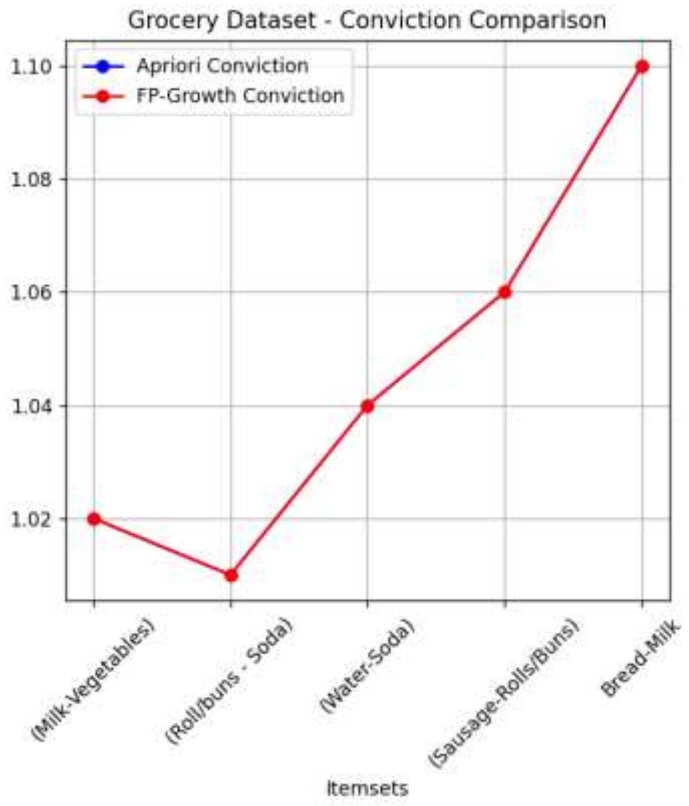
Tabular Representation:

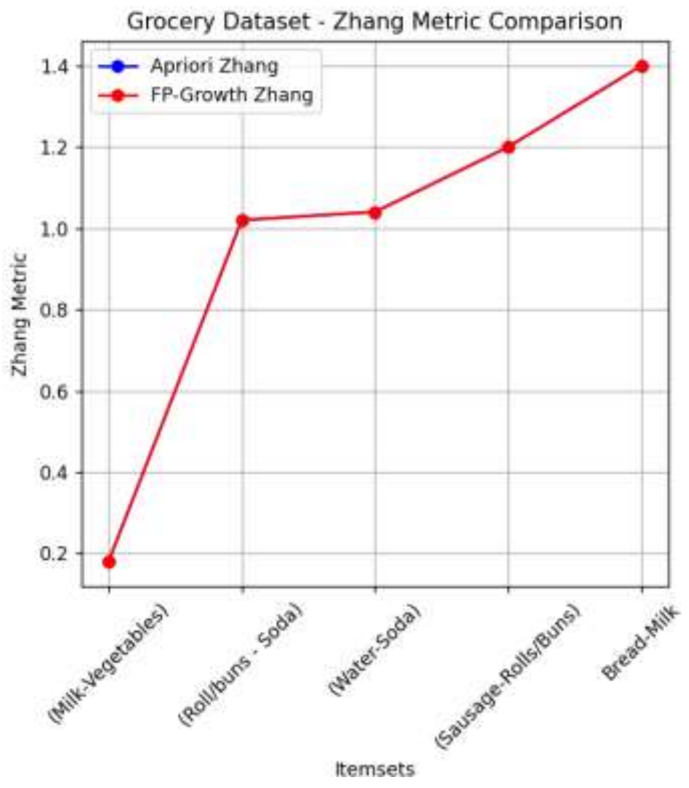
	Apriori	FP-Growth
Execution Time (in seconds)	0.14	0.47
Memory Used (in KiB)	55.6	729.7
Number of Frequent Items Returned	63	63
Number of Apriori Rules Generated	8	8

Graphical Representation:




Grocery Dataset - Support Comparison
Apriori
FP-Growth
Support
0.14
0.12
0.10
0.08
0.06
0.04
(yogurt)
(tropical fruit)
(coffee)
(whole milk)
(pip fruit)
Itemsets

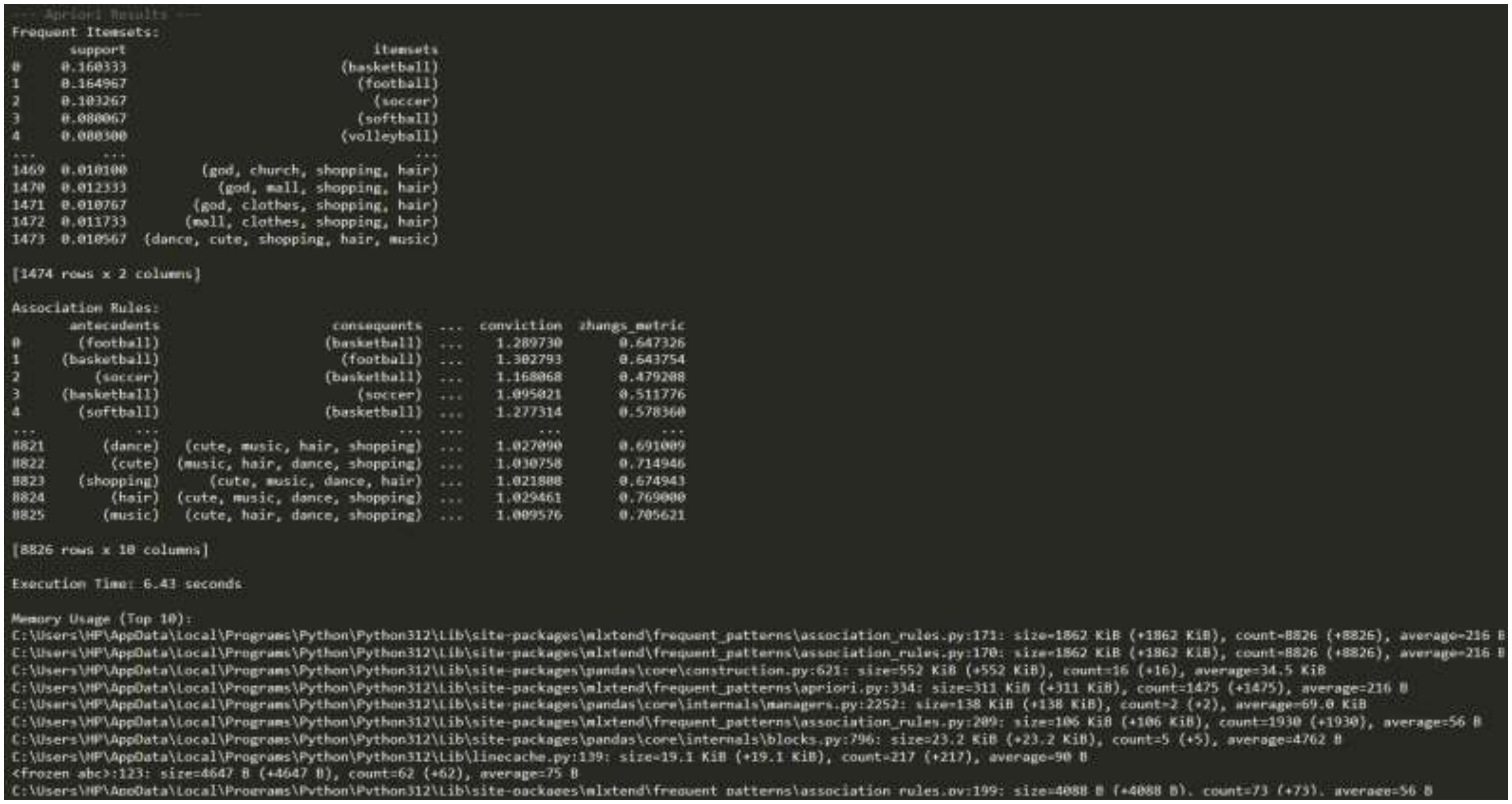

Grocery Dataset - Conviction Comparison
Apriori Conviction
FP-Growth Conviction
1.10
1.08
1.06
1.04
1.02
(Milk-Vegetables)
(Roll/buns - Soda)
(Water-Soda)
(Sausage-Rolls/Buns)
Bread-Milk
Itemsets



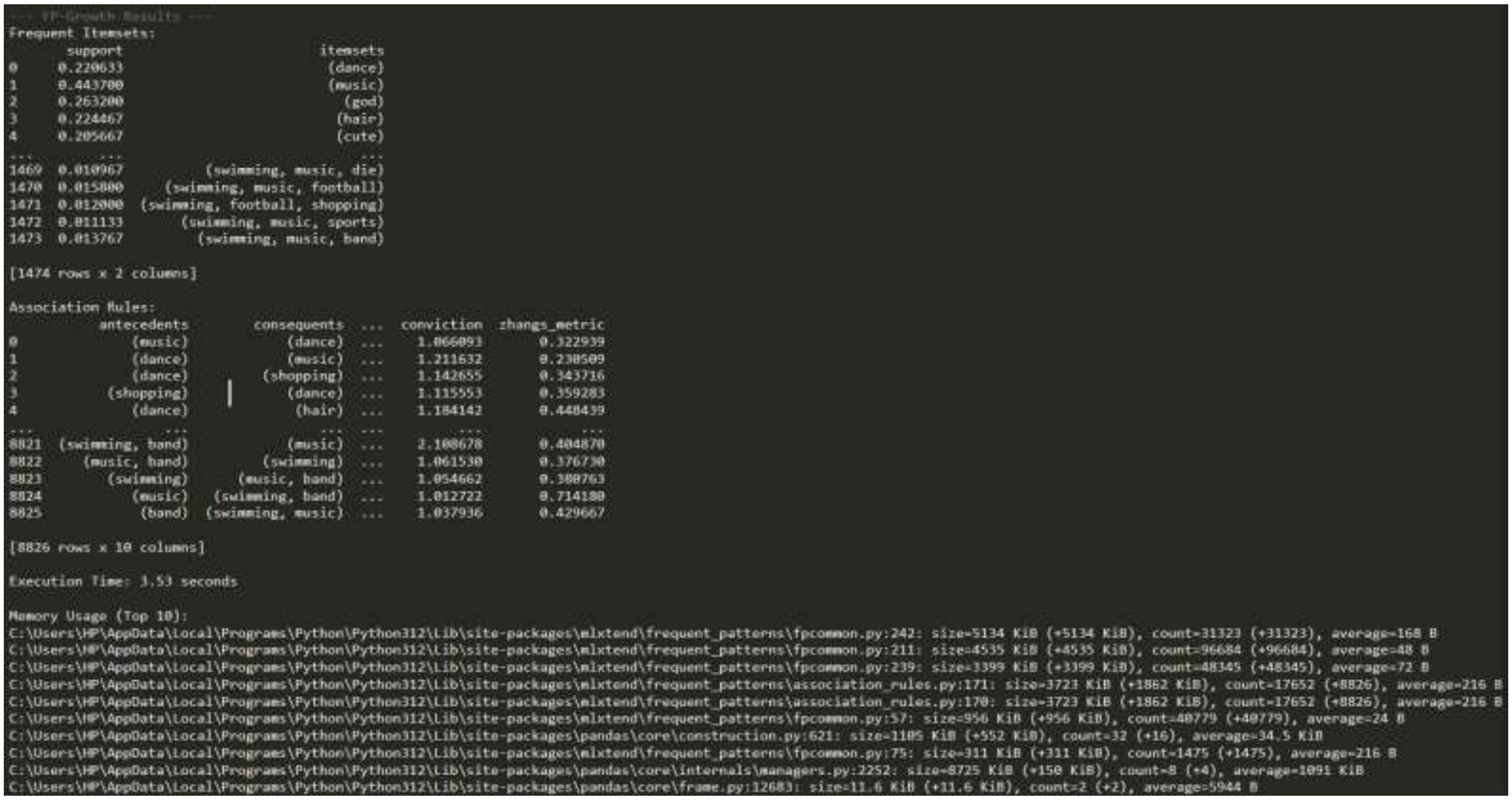
Here, it could be observed that the execution time (0.47 seconds) for FP-Growth Algorithm is higher than the execution time (0.14 seconds) recorded for the Apriori algorithm. While theoretically, FP-Growth is more efficient than Apriori, there are instances where Apriori is relatively more time-efficient, such as when handling smaller datasets. This is because, the complexity of creating a tree-like data structure in FP-Growth algorithm would bring in more overhead as compared to the overhead that might be introduced by computing the candidate sets in the Apriori algorithm. Also, the item with highest support returned by both the algorithms is Milk – indicating that it is the most commonly bought item. Other itemsets with higher support values included (yogurt), with a support value of 0.062899, (tropical fruit) at 0.039803, and (coffee) at 0.033743 which implies that these are frequently bought by customers. It could also be observed on similar lines that the memory usage (729.7 KiB) is relatively higher for FP-Growth algorithm than it is for Apriori (55.6 KiB). The above explanation could be applied here as well, as FP-Growth creates and stores relevant data in tree-like data structures which would occupy more space and there is a larger difference while working with smaller datasets. All in all, the results returned in terms of support, conviction and zhang_metrics is however the same.

2. For SNS Dataset

   Output screenshots:



```
--- Apriori Results ---
Frequent Itemsets:
       support                               itemsets
0     0.160333                           (basketball)
1     0.164967                             (football)
2     0.103267                               (soccer)
3     0.080067                             (softball)
4     0.080300                           (volleyball)
...        ...                                    ...
1469  0.010100        (god, church, shopping, hair)
1470  0.012333          (god, mall, shopping, hair)
1471  0.010767       (god, clothes, shopping, hair)
1472  0.011733      (mall, clothes, shopping, hair)
1473  0.010567  (dance, cute, shopping, hair, music)

[1474 rows x 2 columns]

Association Rules:
        antecedents                      consequents  ...  conviction  zhangs_metric
0        (football)                     (basketball)  ...    1.289730       0.647326
1      (basketball)                       (football)  ...    1.302793       0.643754
2          (soccer)                     (basketball)  ...    1.168068       0.479208
3      (basketball)                         (soccer)  ...    1.095021       0.511776
4        (softball)                     (basketball)  ...    1.277314       0.578360
...             ...                              ...  ...         ...            ...
8821        (dance)   (cute, music, hair, shopping)  ...    1.027090       0.691009
8822         (cute)  (music, hair, dance, shopping)  ...    1.030758       0.714946
8823     (shopping)      (cute, music, dance, hair)  ...    1.021808       0.674943
8824         (hair)  (cute, music, dance, shopping)  ...    1.029461       0.769000
8825        (music)   (cute, hair, dance, shopping)  ...    1.009576       0.705621

[8826 rows x 10 columns]

Execution Time: 6.43 seconds

Memory Usage (Top 10):
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:171: size=1862 KiB (+1862 KiB), count=8826 (+8826), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:170: size=1862 KiB (+1862 KiB), count=8826 (+8826), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\construction.py:621: size=552 KiB (+552 KiB), count=16 (+16), average=34.5 KiB
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\apriori.py:334: size=311 KiB (+311 KiB), count=1475 (+1475), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\managers.py:2252: size=138 KiB (+138 KiB), count=2 (+2), average=69.0 KiB
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:209: size=106 KiB (+106 KiB), count=1930 (+1930), average=56 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\blocks.py:796: size=23.2 KiB (+23.2 KiB), count=5 (+5), average=4762 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\linecache.py:139: size=19.1 KiB (+19.1 KiB), count=217 (+217), average=90 B
<frozen abc>:123: size=4647 B (+4647 B), count=62 (+62), average=75 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:199: size=4088 B (+4088 B), count=73 (+73), average=56 B
```

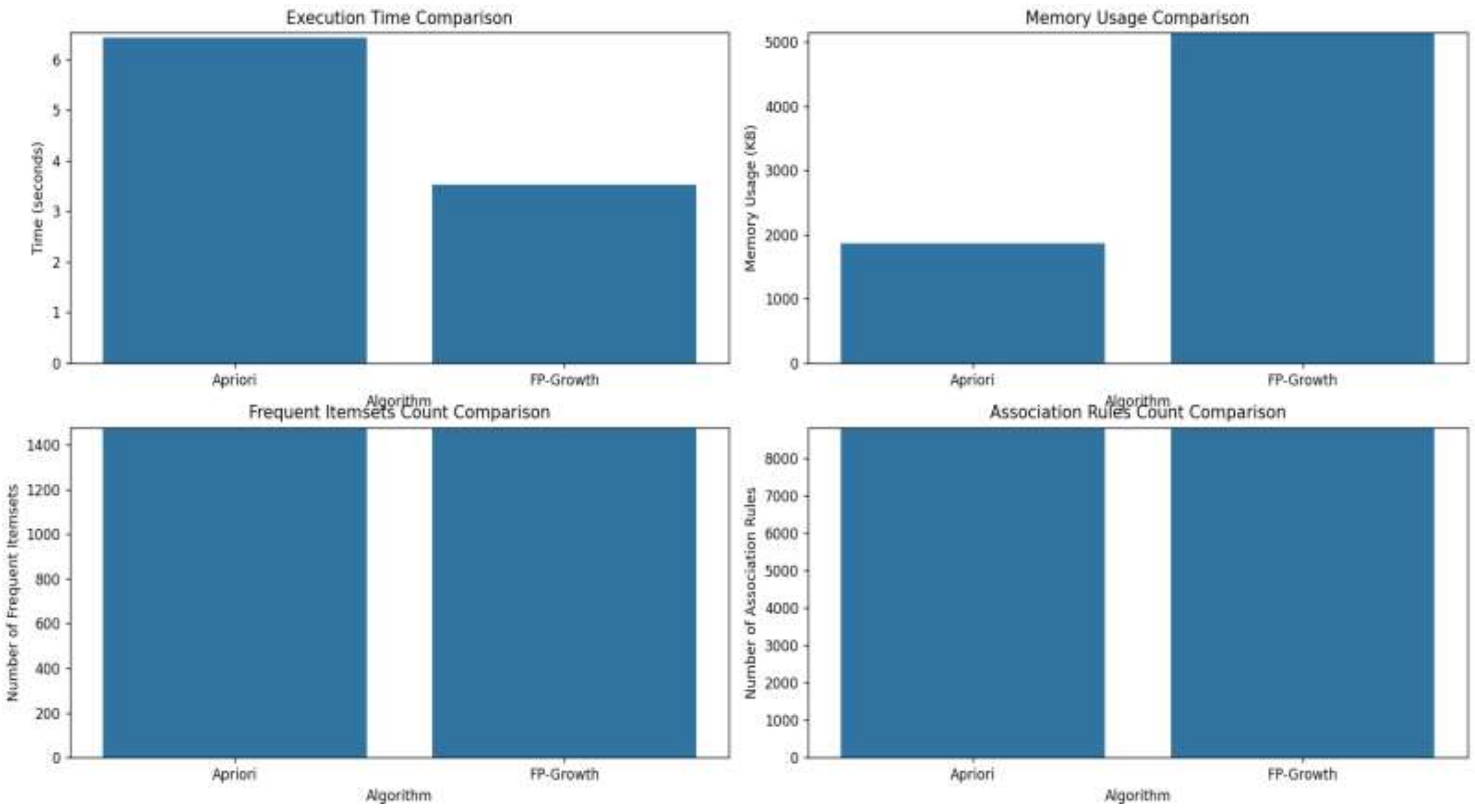


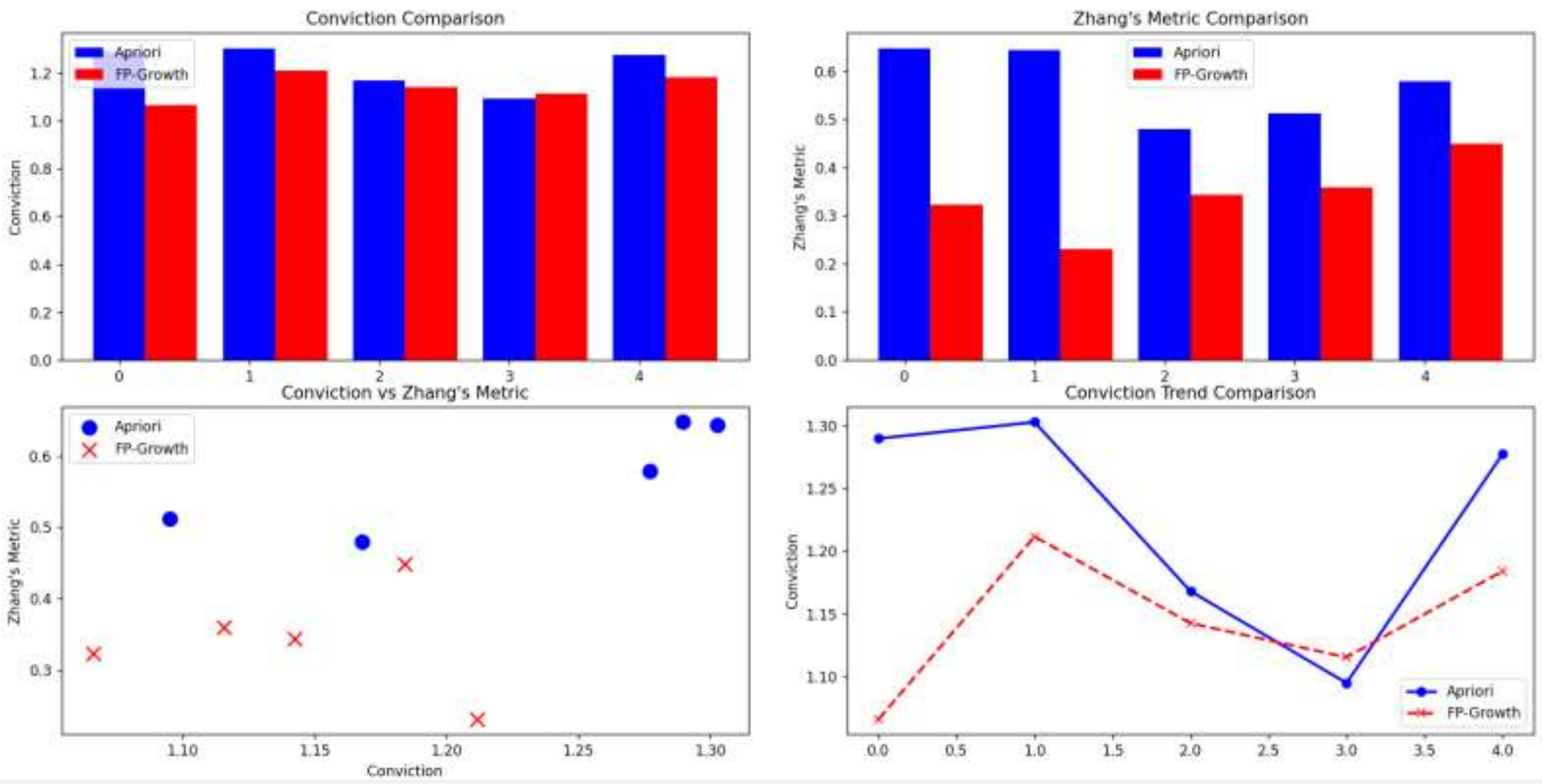
```
--- FP-Growth Results ---
Frequent Itemsets:
       support                         itemsets
0     0.220633                          (dance)
1     0.443700                          (music)
2     0.263200                            (god)
3     0.224467                           (hair)
4     0.205667                           (cute)
...        ...                              ...
1469  0.010967           (swimming, music, die)
1470  0.015800      (swimming, music, football)
1471  0.012000   (swimming, football, shopping)
1472  0.011133        (swimming, music, sports)
1473  0.013767          (swimming, music, band)

[1474 rows x 2 columns]

Association Rules:
            antecedents          consequents  ...  conviction  zhangs_metric
0               (music)              (dance)  ...    1.066093       0.322939
1               (dance)              (music)  ...    1.211632       0.230509
2               (dance)           (shopping)  ...    1.142655       0.343716
3            (shopping)              (dance)  ...    1.115553       0.359283
4               (dance)               (hair)  ...    1.184142       0.448439
...                 ...                  ...  ...         ...            ...
8821   (swimming, band)              (music)  ...    2.108678       0.404870
8822      (music, band)           (swimming)  ...    1.061530       0.376730
8823         (swimming)        (music, band)  ...    1.054662       0.380763
8824            (music)     (swimming, band)  ...    1.012722       0.714180
8825             (band)    (swimming, music)  ...    1.037936       0.429667

[8826 rows x 10 columns]

Execution Time: 3.53 seconds

Memory Usage (Top 10):
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:242: size=5134 KiB (+5134 KiB), count=31323 (+31323), average=168 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:211: size=4535 KiB (+4535 KiB), count=96684 (+96684), average=48 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:239: size=3399 KiB (+3399 KiB), count=48345 (+48345), average=72 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:171: size=3723 KiB (+1862 KiB), count=17652 (+8826), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:170: size=3723 KiB (+1862 KiB), count=17652 (+8826), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:57: size=956 KiB (+956 KiB), count=40779 (+40779), average=24 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\construction.py:621: size=1105 KiB (+552 KiB), count=32 (+16), average=34.5 KiB
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:75: size=311 KiB (+311 KiB), count=1475 (+1475), average=216 B
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\internals\managers.py:2252: size=8725 KiB (+150 KiB), count=8 (+4), average=1091 KiB
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\frame.py:12683: size=11.6 KiB (+11.6 KiB), count=2 (+2), average=5944 B
```

Tabular Representation:

	Apriori	FP-Growth
Execution Time (in seconds)	6.43	3.53
Memory Used (in KiB)	1,862	5,134
Number of Frequent Items Returned	1,474	1,474
Number of Apriori Rules Generated	8,826	8,826

Graphical Representation:



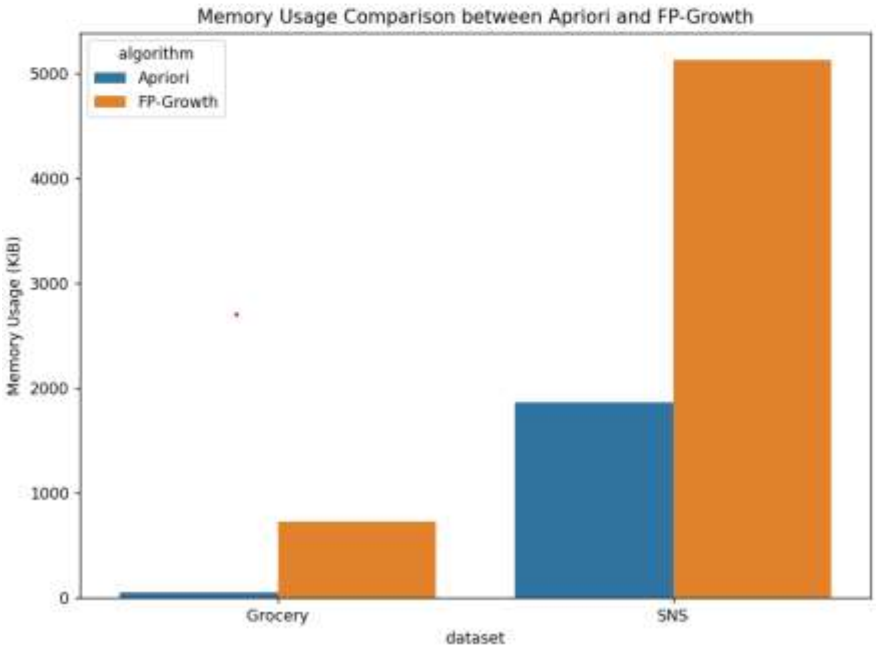


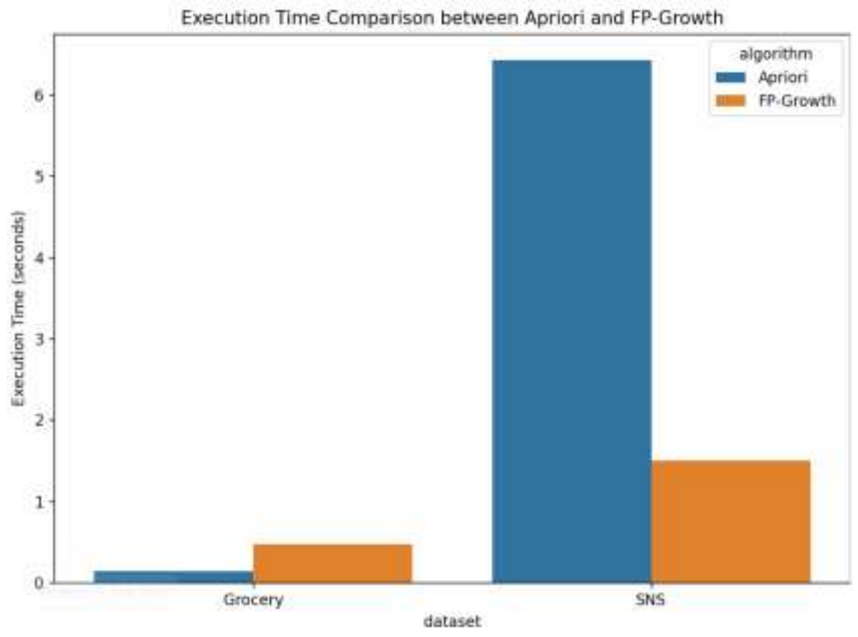
With an increased size in datasets, it could be observed that FP-Growth is faster than Apriori, finishing its task as it takes only 3.53 seconds whereas the Apriori algorithm takes 6.43 seconds. This difference in speed is due to the way the FP-Growth algorithm is implemented, using an FP-tree, a compact structure that helps it find frequent patterns more efficiently instead of parsing through the dataset multiple times. Unlike Apriori, FP-Growth doesn’t need to generate candidate itemsets, which saves time. In contrast, Apriori works by generating and testing candidate itemsets level by level, which can be slower, especially when dealing with larger datasets. This makes FP-Growth a better choice when speed is a priority and while dealing with larger datasets. In terms of memory however, FP-Growth algorithm still uses a larger memory blob relative to Apriori particularly as it grows to handle different combinations of itemsets. Its memory usage reaches up to about 5,134 KiB during frequent pattern mining with this particular dataset, while Apriori peaks at around 1,862 KiB when mining the association rules.

On the other hand, while both algorithms generate the same number of frequent itemsets, the type of items included and the support (confidence) values are different. While Apriori returned simpler (singleton) items (basketball) and (football), which have support values of 0.160 and 0.164, FP-

Growth returned more meaningful insights with more than one item-combinations like (swimming, music, band) along with a greater range of support such as (dance) at 0.220 and (music) at 0.443. This highlights FP-Growth's strength in finding a wider variety of item combinations compared to Apriori. However, it could be noted that the conviction and zhang’s metric scores are comparatively higher for results returned by Apriori. This could be explained through the implementation. The confidence gives the independence between the antecedent and consequent. Apriori may, due to the breadth of itemsets explored, generate rules showing stronger independence. FP-growth's tree-based, compressed approach might not always explore nuances of all itemsets to the same extent, leading to slightly weaker rules in terms of independence. Metric by czhang: In many cases, this could be informed by the confidence of the itemset and a balance of the supports across itemsets. Since Apriori does explicit rating for frequent itemsets and generates rules in steps by taking into consideration combinations of smaller itemsets, it may end up better optimizing rules that portray strong confidence and balance, thus increasing its scores for czhang's metrics

Comparing the results against the datasets for both the algorithms:





A significant improvement in execution time could be observed for FP-Growth while working with a larger dataset. But the memory usage increased proportionally. FP-trees created to mine the frequent patterns in the datasets could account to these metrics recorded. So choosing which algorithm to execute depends on the computational resources and the other constraints such as execution time and efficiency which are given. For greater computational resources with high emphasis on time constraint and while working with larger datasets, FP-Growth should be implemented. However, if computational resources are not as high and memory efficiency is relatively more important than execution time, then Apriori algorithm could be implemented. Note that the candidate items generated through Apriori algorithm through DFS might actually yield better conviction, zhang metrics scores which increased the confidence in decision making during analysis of the data.

CONCLUSION

The experimentation with varying sizes of datasets has yielded different performance results for both the algorithms in terms of execution time, memory usage, support values and overall efficiency in identifying frequent data items and the association rules in between. These behaviors clearly reflect the way the algorithms are implemented. Apriori algorithm works on the principle that the subset of frequent itemsets must also be frequent. Hence, it applies an iterative, level-wise search method (BFS), where it first identifies all frequent individual items (1-itemsets) and then extends them to larger itemsets by combining frequent (n-1)-itemsets. This is repeated n+1 times for ‘n’ dataitems in a worst-case scenario. At each iteration, the algorithm generates and identifies candidate items repeatedly until no other items repeating frequently are found. The number of candidate items generated is expected to grow exponentially while working with larger datasets. Hence, this is not a feasible approach for data mining on large datasets due to the increased overhead. FP-Growth on the other hand constructs FP-Tree after passing through the dataset once. This FP-tree is used to mine frequent patterns by the algorithm recursively with DFS approach. However, this introduces new computational costs as there should be enough memory to store the FP-Growth trees. Additionally, FP-Growth algorithm returned relatively more meaningful results with more complex association rules (combining more than one itemsets). Also, the execution time was drastically less compared to the that recorded for Apriori in larger datasets. However, while working with smaller dataset, it could be noted that Apriori algorithm is more suitable especially when the application is time-sensitive and memory constrained.

REFERENCES

[1] https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/groceries.csv

[2] https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/refs/heads/master/snsdata.csv

[3] https://www.geeksforgeeks.org/frequent-item-set-in-data-set-association-rule-mining/

[4] https://www.geeksforgeeks.org/frequent-pattern-growth-algorithm/

[5] https://www.geeksforgeeks.org/apriori-algorithm/

[6] http://fimi.cs.helsinki.fi/data/

[7] http://fimi.cs.helsinki.fi/

[8] https://www.kaggle.com/code/keitazoumana/comparative-analysis-between-apriori-and-fp-growth

[9] https://link.springer.com/chapter/10.1007/978-3-031-25847-3_13

[10] https://analyticsindiamag.com/developers-corner/apriori-vs-fp-growth-in-market-basket-analysis-a-comparative-guide/