Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.
Digital Object Identifier 10.1109/ACCESS.2023.0322000

Association Rule Mining using Apriori Algorithm on Grocery Shopping Data

KHADKA PILOT[1*] AND POUDEL KSHITIZ[2*]
[1]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: pilot.076bct025@tcioe.edu.np)
[2]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: kshitizpoudel18@gmail.com)
Corresponding author: Khadka Pilot (e-mail: pilot.076bct025@tcioe.edu.np).
* Authors contributed equally

ABSTRACT In the modern era of data-driven decision-making, business enterprises amass vast volumes of data from their daily operations. One prime example is the extensive troves of customer purchase data collected at grocery store checkout counters on a daily basis. This data deluge offers opportunities to extract meaningful insights. One such transformative technique is Association Rule Mining, a cornerstone of data mining, which uncovers intricate relationships and patterns hidden within the data. Association rule mining thrives in deciphering the underlying connections among items within large datasets. It allows understanding the co-occurrences and dependencies between products, behaviors, or events. From market basket analysis to recommendation engines, this technique fuels applications across various sectors, allowing efficient inventory management, personalized recommendations, and optimized business strategies.

INDEX TERMS Association rule mining, Apriori

I. INTRODUCTION

In the realm of data mining and machine learning, association rule mining stands as a fundamental technique that unveils hidden relationships within large datasets. The essence of this method lies in its ability to uncover associations, correlations, and patterns in data that might otherwise remain obscured. These extracted associations, often presented in the form of association rules, shed light on the co-occurrence and dependencies among various items in a dataset. Association rule mining finds its significance in a multitude of real-world applications, ranging from market basket analysis in retail to recommendation systems, healthcare analytics, and beyond. By identifying meaningful connections between items, it enables businesses and researchers to make informed decisions, develop personalized recommendations, optimize supply chain management, and enhance customer experiences.

At its core, the process involves sifting through transactional or relational datasets to discover associations among items that tend to occur together. The most prominent algorithm for association rule mining is the Apriori algorithm, which efficiently generates frequent itemsets and derives association rules. The output of association rule mining often comprises a list of rules, each consisting of an antecedent (the condition) and a consequent (the outcome). These rules are usually ranked based on their support and confidence values, allowing analysts to focus on the most relevant and trustworthy associations. As datasets continue to grow in size and complexity, association rule mining techniques have evolved to handle challenges like scalability and noise in the data. Advanced methods, including FP-Growth and Eclat, offer efficient alternatives to Apriori for discovering associations. Moreover, the integration of association rule mining with machine learning and deep learning techniques has opened up new avenues for extracting intricate patterns from diverse data sources.

II. METHODOLOGY

A. THEORY

In the context of market basket analysis, we define $I = \{i_1, i_2, \ldots, i_d\}$ to represent the set encompassing all available items. Correspondingly, $T = \{t_1, t_2, \ldots, t_N\}$ denotes the collection of transactions. Each individual transaction t_i consists of a subset of items selected from the item set I.

An 'itemset' refers to a collection of items, which can be composed of any number of elements, even none. Specifically, an itemset containing k distinct items is termed a 'k-itemset. A transaction t_j is considered to encompass an itemset X if X is a subset of t_j.

Support count is the frequency of occurrence of an itemset. Mathematically, the 'support count' $\sigma(X)$ for an itemset X is defined as:

$$\sigma(X) = |\{t_i \,|\, X \subseteq t_i,\, t_i \in T\}| \quad (1)$$

An association rule relates the two itemset and takes the form of $X \rightarrow Y$, where X and Y are sets of items that have no common elements ($X \cap Y = \emptyset$). The association rule is measured by its support and confidence. Support signifies how often a rule is applicable within a dataset, while confidence reveals how often items in set Y appear in transactions containing set X. This is so because it helps find out whether a rule occurs by random chance. A rule with extremely low support might arise coincidentally. Additionally, from a business standpoint, a rule with low support could lack practical value. For instance, promoting items rarely purchased together might not yield profitable outcomes.

The formula for support and confidence are:

$$s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \quad (2)$$

$$c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \quad (3)$$

Association rule mining involves two steps:

1) **Frequent Itemset Generation**: I involves identifying all itemsets that meet the minimum support requirement. Those itemset are called 'frequent itemset'.
2) **Rule Generation**: Extract high confidence rules from the generated itemset. These extracted rules are called strong rules.

B. FREQUENT ITEMSET GENERATION

In the bruteforce method, the process involves iterating through the itemset list to calculate the support and confidence values for each conceivable rule. Nevertheless, this approach becomes significantly computationally intensive. For a dataset with d distinct items:

- Total count of itemsets 2^d
- Total count of possible association rules: $R = 3^d \times 2^{d+1} + 1$

To address the challenges posed by the brute-force approach, the Apriori method was developed. This method operates on a fundamental principle: if an itemset is frequent, then all its subsets must also be frequent.

In the context of the Apriori method,
let C_k denote the collection of candidate k-itemsets,
and F_k denote the collection of frequent k-itemsets.

The algorithm calculates the support of each individual item once. After this step, the set of frequent 1-itemsets, F_1, is genereated. The algorithm then enters an iterative process wherein it generates new candidate k-itemsets by utilizing the frequent $(k-1)$-itemsets discovered in the previous iteration. Next, the algorithm counts the support of candidates to determine candidate itemset in Ck are contained in each transaction. It then removes candidate itemset with support less than the minimum support. This is followed iteratively until the there are no frequent itemset being generated.

C. RULE GENERATION

An association rule can be extracted by partitioning the itemset Y into two non-empty subsets, X and Y X, such that X $\rightarrow$ Y X satisfies the confidence threshold.

Initially, all the high-confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules.

D. FACTORS INFLUENCING THE COMPLEXITY OF THE APRIORI ALGORITHM

When working with the Apriori algorithm, several factors contribute to its overall complexity and performance:

1) Minimum Support Threshold: Adjusting the support threshold impacts the algorithm's outcomes. Lowering the threshold leads to discovering more frequent itemsets, yet it might increase the count of candidate itemsets and extend the length of frequent itemsets.
2) Dimensionality of the Data Set: The dimensionality, or the total number of distinct items in the dataset, plays a role in complexity. With more items, additional storage is needed to maintain support counts. Moreover, if frequent item counts rise, computational and I/O costs could escalate.
3) Database Size: As the Apriori algorithm makes multiple passes over the data, a larger number of transactions in the database may elongate its runtime.
4) Average Transaction Width: Datasets with denser patterns exhibit increased average transaction width. Consequently, this broader width could lead to longer frequent itemsets and greater hash tree traversals, due to the rise in subsets within each transaction.

E. ALGORITHM

Algorithm 1 Frequent itemset generation using Apriori algorithm

$k \leftarrow 1$
$F_k \leftarrow \{i \,|\, i \in I$ and $\sigma(\{i\}) \geq N \times$ minsup$\}$
repeat
  $k \leftarrow k + 1$
  $C_k \leftarrow$ apriori-gen(F_{k-1})
  for all $t \in T$ **do**
    $C_t \leftarrow$ subset(C_k, t)
    for all $c \in C_t$ **do**
      $\sigma(c) \leftarrow \sigma(c) + 1$
    end for
  end for
  $F_k \leftarrow \{c \,|\, c \in C_k$ and $\sigma(c) \geq N \times$ minsup$\}$
until $F_k = \emptyset$
Result $\leftarrow \bigcup F_k$

F. SYSTEM BLOCK DIAGRAM

Appendix

Algorithm 2 Rule generation

for each frequent k-itemset $F_k, k \geq 2$ **do**
  $H_1 \leftarrow \{i \mid i \in F_k\}$ ▷ 1-item consequents of the rule
  ap-genrules(F_k, H_1) ▷ Call ap-genrules function
end for

Algorithm 3 Procedure ap-genrules(F_k, H_m)

$k = |F_k|$ ▷ Size of frequent itemset.
$m = |H_m|$ ▷ Size of rule consequent.
if $k > m + 1$ **then**
  $H_{m+1} = \text{apriori-gen}(H_m)$
  for each $h_{m+1} \in H_{m+1}$ **do**
    conf $= \frac{\sigma(F_k)}{\sigma(F_k - h_{m+1})}$
    if conf $\geq$ minconf **then**
      output the rule $(F_k - h_{m+1}) \rightarrow h_{m+1}$
    else
      delete h_{m+1} from H_{m+1}
    end if
  end for
  ap-genrules(F_k, H_{m+1})
end if

G. INSTRUMENTATION

The following libraries were instrumental in completion of this project :

- **pd.read_csv**: This function is employed to read the CSV groceries dataset.
- **plt.violinplot**: Utilized for generating a violin plot that showcases the distribution of transaction sizes across various itemset counts.
- **plt.boxplot**: This function aids in identifying outliers within the grocery dataset by visualizing their distribution.
- **plt.hist**: It serves the purpose of plotting a histogram to depict the distribution of transaction sizes.
- **plt.barh**: Employed to create horizontal bar plots, useful for visualizing the ten most and least frequent itemsets.
- **mlxtend.preprocessing.TransactionEncoder**: This utility is used to encode the transactional data from a Python list of lists into a NumPy array.
- **mlxtend.frequent_patterns.apriori**: Applied to execute the Apriori algorithm on the transaction database, extracting frequent itemsets.
- **mlxtend.frequent_patterns.association_rules**: Utilized to generate association rules from the frequent itemsets obtained.

III. WORKING PRINCIPLE

A. DATA LOADING AND EXPLORATION

The grocery dataset was loaded using the `pd.read_csv` function. This dataset comprises 9,835 customer transactions involving grocery shopping. It encompasses a total of 169 distinct items, reflecting products commonly found in a grocery store. Among the items included are butter, milk, yogurt, cream cheese, spread cheese, rolls/buns, bottled water, soda, newspapers, and more.

B. ITEMSET ANALYSIS

Itemset analysis is a pivotal step in data mining, especially when applying algorithms like Apriori to transactional datasets such as grocery shopping records. This analysis focuses on understanding how often items appear together in transactions, revealing patterns in item combinations.

Visualization plays a critical role in itemset analysis as it helps to intuitively grasp the distribution and patterns of these item combinations. Common visualization techniques include:

- Violin Plots: These plots provide insights into transaction size distribution for varying itemset counts. They help identify variations in transaction sizes and their relationship with the number of items purchased.
- Box Plots: By illustrating transaction size distribution, box plots highlight potential outliers. These outliers could indicate unusually large or small transactions.
- Histograms: Histograms depict transaction size distribution, shedding light on the frequency of specific transaction sizes. This offers insights into transaction behaviors.

In our dataset, we observed that up the top six-item itemsets are more frequent, as revealed by the violin plot. Similarly, by visualizing transaction distribution, we could identify the most frequently occurring items. We found our 10 most frequent itemset to be: We identified the ten most frequent itemsets, which are as follows:

1) Most frequent itemset

- Whole milk
- Other vegetables
- Rolls/buns
- Soda
- Yogurt

These items are the ones most commonly purchased together in transactions. This finding aligns with common grocery shopping behavior where staples like milk, vegetables, rolls/buns, soda, and yogurt are often bought together. Customers might frequently purchase these items in a single shopping trip due to their everyday consumption or complementary nature.

2) Least Frequent Itemsets

The least frequent items in the dataset include:

- Sound storage medium
- Baby food
- Preservation products

These items have relatively low occurrence frequencies in transactions. Possible reasons for their infrequent appearance could be niche or specialized products. Customers might not need these items as frequently as others, leading to fewer occurrences in transactions.

Top One-Item Products

The top five one-item products, which are the most frequently purchased individual items, are:

- Canned beer
- Soda
- Whole milk
- Bottled beet
- Rolls/buns

These items suggest that beverages like beer, soda, and milk, along with staple food items like rolls/buns, have a high individual purchase rate. These items are commonly consumed and likely to be part of customers' regular grocery shopping lists.

The results suggest that the dataset's shopping patterns are characterized by the frequent purchase of staple items like milk, vegetables, rolls/buns, and beverages such as soda. Niche or specialized items like sound storage media, baby food, and preservation products appear less frequently, likely due to their specific use cases. These insights can help retailers optimize their product placements, marketing strategies, and inventory management to cater to customer preferences and boost sales.

TRANSACTION ENCODING

Transaction encoding is a fundamental preprocessing step in data mining, particularly when working with algorithms like the Apriori algorithm. It involves transforming transactional data into a suitable format that algorithms can process effectively. The primary objective is to convert the transactional information into a binary representation that reflects the presence or absence of specific items in each transaction.

In transactional datasets, each transaction represents a customer's purchase or interaction, and it contains a list of items bought together. For instance, in a grocery store dataset, each transaction could list the items a customer purchased in a single shopping instance.

Need for Encoding

Algorithms like Apriori work with structured data formats. However, transactional data is inherently unstructured, with varying items in each transaction. Transaction encoding addresses this disparity by transforming the dataset into a consistent matrix-like structure.

Binary Representation

Transaction encoding involves converting the transactional data into a binary matrix, where rows correspond to transactions, and columns correspond to unique items. Each cell in the matrix contains a binary value, indicating whether a specific item is present (1) or absent (0) in the corresponding transaction.

SUPPORT CALCULATION STEP

The support of an itemset in a dataset is a measure of how frequently that itemset appears in the transactions of the dataset. It is a fundamental metric in association rule mining, as it helps identify which itemsets are frequent and potentially interesting for generating association rules.

Here's how the support calculation step works:

1) **Transaction Count**: Count the total number of transactions in the dataset. This value is denoted as N.
2) **Itemset Count**: Count the number of transactions that contain the specific itemset . This count is denoted as n, where n represents the number of transactions that include the given itemset.
3) **Support Calculation**: The support of the itemset is calculated as the ratio of the number of transactions containing the itemset to the total number of transactions in the dataset:
$$\text{Support (s)} = \frac{n}{N}$$
The support value lies between 0 and 1. A higher support value indicates that the itemset is more frequently occurring in the transactions.
4) **Threshold Comparison**: Compare the calculated support value with a predefined minimum support threshold. This threshold determines the minimum level of support that an itemset must have to be considered frequent. If the calculated support is greater than or equal to the threshold, the itemset is considered frequent; otherwise, it's considered infrequent.

ASSOCIATION RULE GENERATION STEP

The association rule generation step is a crucial part of the Apriori algorithm, which aims to extract meaningful rules from frequent itemsets identified in the dataset. An association rule is an implication of the form "if X, then Y," where X and Y are itemsets, often referred to as the antecedent and consequent, respectively.

Here's how the association rule generation step works:

1) **Frequent Itemsets**: First, a set of frequent itemsets using the minimum support threshold are identified. These frequent itemsets act as the basis for generating association rules.
2) **Rule Evaluation Metrics**: An evaluation metrics is needed to determine the strength and significance of potential association rules. Common metrics include support, confidence, and lift. Confidence is particularly important. It measures the proportion of transactions containing the antecedent that also contain the consequent. Mathematically, it's defined as:
$$\text{Confidence (c)} = \frac{\text{Support (X} \cup \text{Y)}}{\text{Support (X)}}$$
The higher the confidence, the more likely the consequent will occur when the antecedent is present.
3) **Rule Generation**: For each frequent itemset with at least two items, association rules are generated by considering all possible combinations of the itemset as antecedent and the remaining items as consequent. If

the confidence of a rule exceeds a predefined minimum confidence threshold, it is considered a valid association rule.

4) **Pruning**: Not all generated rules are necessarily interesting or useful. To refine the rule set, further filtering based on additional metrics, such as lift or conviction can be applied to ensure that the rules discovered are meaningful.

RULE ANALYSIS FOR GROCERY DATASET

After applying the Apriori algorithm to a grocery store dataset and generating association rules, let's consider a few rules for analysis:

RULE 1

"If {Milk, Bread}, then {Eggs}"

- Support: 0.1 (10%)
- Confidence: 0.6 (60%)
- Lift: 1.5

Interpretation: This rule suggests that when customers buy both Milk and Bread, there's a 60% chance they will also buy Eggs. The lift of 1.5 indicates a positive association, meaning the occurrence of Milk, Bread, and Eggs together is 1.5 times more likely than if they were independent. The positive lift suggests a potential cross-promotion strategy for Milk, Bread, and Eggs.

IV. RESULT ANALYSIS

The preceding two plots offer insightful observations:

Transaction Distribution: The presented plots in Figure(3,4) highlight that a majority of transactions consist of a single item. This trend suggests that customers often prefer purchasing individual items rather than buying in bulk.

Most Bought Itemsets: The histogram in figure(5) provides a visual representation of the most frequently bought itemsets, along with their corresponding support counts. This visualization enables a quick identification of the pivotal items in the grocery store.

Least Frequent Itemsets: The plotted data of the least frequent itemsets in figure(6) implies that these particular sets may not hold significant importance. Consequently, these infrequent itemsets could potentially be considered for removal from the grocery store offerings.

Dominant Single Item: Despite whole milk being the most frequently bought item overall as seen in figure(5) , an intriguing insight emerges when considering single-item purchases as seen in figure(7). Specifically, the item "canned beer" emerges with the highest support count for such cases.

Variation with Minimum Support: The graph in figure(8) illustrates how the number of generated rules changes with varying minimum support levels (10%, 8%, 6%, 4%, 1%, and 0.5%). This visual aids in estimating the appropriate minimum support count for generating frequent itemsets, as it depends on the nature and size of the database. Notably, as the minimum support threshold increases, the number of generated rules experiences rapid growth. A higher support threshold ensures that only rules involving frequent items are generated. Conversely, setting a lower minimum support threshold results in numerous rules with high confidence. While such a scenario captures a multitude of associations, some of these rules may have lower frequency and lesser relevance for decision-making.

The insights drawn from these analyses contribute to a better understanding of customer behaviors, preferences, and shopping patterns, enabling informed business strategies and enhancing the overall shopping experience.

Support	Itemsets	Length
0.256	(whole milk)	1
0.193	(other vegetables)	1
0.184	(rolls/buns)	1
0.174	(soda)	1

TABLE 1. Frequent Itemsets generated with minimum support = 0.15

Support	Itemsets	Length
0.256	(whole milk)	1
0.193	(other vegetables)	1
0.184	(rolls/buns)	1
0.174	(soda)	1
0.140	(yogurt)	1
0.111	(bottled water)	1
0.109	(root vegetables)	1
0.105	(tropical fruit)	1
0.099	(shopping bags)	1
0.094	(sausage)	1
0.089	(pastry)	1
0.083	(citrus fruit)	1
0.081	(bottled beer)	1
0.080	(newspapers)	1
0.078	(canned beer)	1
0.076	(pip fruit)	1
0.075	(whole milk, other vegetables)	2
0.072	(fruit/vegetable juice)	1
0.072	(whipped/sour cream)	1
0.065	(brown bread)	1
0.063	(domestic eggs)	1
0.059	(frankfurter)	1
0.059	(margarine)	1
0.058	(coffee)	1
0.058	(pork)	1
0.057	(whole milk, rolls/buns)	2
0.056	(yogurt, whole milk)	2
0.055	(butter)	1
0.053	(curd)	1
0.052	(beef)	1
0.052	(napkins)	1

TABLE 2. Frequent Itemsets generated with minimum support = 0.05

Two distinct figures (9,10) showcase the generated association rules resulting from different frequent itemsets, which emerge due to varying choices of the minimum support threshold. Once these rules are generated, they can undergo further filtration based on a confidence threshold.

Key Concepts in Association Rule Analysis:

1) Antecedents: Refers to items or itemsets on the left-hand side (LHS) of the rule.

2) Consequents: Refers to items or itemsets on the right-hand side (RHS) of the rule.
3) Antecedent Support: Indicates how often the antecedents appear individually in the transactional database.
4) Consequent Support: Indicates how often the consequents appear individually in the transactional database.
5) Support: Denotes the frequency of appearance of both antecedents and consequents together.
6) Confidence: Measures how frequently the rule has proven true. Calculated as the support of the entire rule divided by the support of the antecedent.
7) Lift: Indicates how much more likely antecedent and consequent items are to appear together compared to if they were independent. Lift > 1 implies a positive association.
8) Leverage: Represents the difference between observed and expected co-occurrence of antecedents and consequents.
9) Conviction: Measures the strength of the rule's prediction reliance on dependency.
10) Zhang's Metric: A composite measure encompassing support, confidence, and the rule's structure.

Understanding these concepts aids in comprehending the significance of generated association rules. Further analysis and selection of rules based on these metrics empower businesses to uncover meaningful patterns and make informed decisions to optimize their strategies.

Visualization of Association Rules

The visualizations offer valuable insights into the generated association rules:

1) Lift-based Bar Chart: The horizontal bar chart in figure(11), portrays the top N association rules, arranged by their lift values. Higher lift values indicate more robust associations. Rules with elevated lift values imply that customers who purchase the antecedent items are more likely to also purchase the consequent items.
2) Network Graph: Figure(12) presents a network graph illustrating the association rules. Each node within the graph represents an item, and the edges between nodes signify associations between those items, as derived from the generated association rules. The thickness of the edges (represented by lift values) serves as an indicator of the strength of the association between the items.

V. DISCUSSION

A. BRUTE FORCE APPROACH VS APRIORI ALGORITHM

While both the brute force approach and the Apriori algorithm yield the same frequent itemsets, the Apriori algorithm boasts greater speed. The brute force approach involves considering all possible itemsets and computing their support and confidence values. On the other hand, the Apriori algorithm capitalizes on the Apriori property to decrease the count of itemsets under consideration, rendering it faster. Let's assume the following specifications:

- N: Total number of transactions in the dataset.
- M: Total number of unique items in the dataset.
- *min_sup*: Minimum support threshold.
- W: Maximum size of itemsets.

For the brute force approach, the complexity involves generating 2^n possible itemsets and scanning the database for each itemset, resulting in a complexity of $O(N \cdot M \cdot W)$. For the Apriori approach, candidate itemsets are generated by self-joining frequent itemsets from the previous level. The time and space complexity is lower due to this efficient generation process.

C) RULE GENERATION

Once frequent itemsets are derived, the generation of association rules becomes essential. For a dataset containing d items, using the brute force rule generation approach, the total number of generated rules (R) is given by:

$$R = 3^d - 2^{d+1} + 1.$$

The goal of frequent itemset generation is to reduce this number significantly by considering only those itemsets meeting the minimum support count criterion. For each frequent itemset with 'k' items, only $2^k - 2$ rules need to be generated, avoiding empty antecedents or consequents. This approach is faster due to the significant reduction in the number of itemsets. Generating rules involves calculating the confidence value for each rule. This task is also streamlined by not considering redundant rules. Additionally, generating rules from frequent itemsets eliminates the need for extra scans on the database, as the support count for both antecedents and consequents has already been calculated during frequent itemset generation. This efficiency arises from the fact that antecedents and consequents are subsets of frequent itemsets, enabling the calculation of their support counts in advance.

VI. CONCLUSION

Association analysis using the Apriori algorithm provides valuable insights into the purchasing patterns and relationships among items in the grocery dataset. Through the Apriori algorithm, we were able to identify frequent itemsets, which are combinations of items that occur together in a significant number of transactions. These frequent itemsets unveil hidden patterns and associations that can be leveraged for decision-making and business strategies.

The Apriori algorithm's efficiency in pruning the search space through the Apriori property played a crucial role in speeding up the process of generating frequent itemsets and association rules. By considering only the relevant itemsets and rules, we were able to gain meaningful insights without exhaustive calculations.

Visualizations, such as the lift-based bar chart and network graph, added an extra layer of understanding, enabling us to grasp the strength of associations and relationships among different items. These visualizations make it easier to interpret the results and identify items that are often purchased together.

In business contexts, the generated association rules can be used to enhance marketing strategies, optimize product placements, and even personalize recommendations for customers. By understanding the relationships between items, businesses can drive sales, improve customer satisfaction, and optimize inventory management. Association analysis using the Apriori algorithm is a powerful technique for uncovering valuable insights from transactional data, contributing to more informed and data-driven decision-making in various domains.

REFERENCES

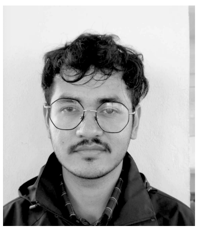
[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *"Deep Learning."* MIT Press.

[2] Daniel Jurafsky & James H. Martin. *Speech and Language Processing*.

[3] M. Narasimha Murty, V. Susheela Devi. *Pattern Recognition: An Algorithmic Approach*.

[4] Rakesh Agrawal, Tomasz Imieliński, Arun Swami, *Mining Association Rules between Sets of Items in Large Databases*, *ACM SIGMOD Record*, Vol. 22, Issue 2, June 1993, Publisher: Association for Computing Machinery.

[5] Rakesh Agrawal, Ramakrishnan Srikant, *Fast Algorithms for Mining Association Rules in Large Databases*, In *Proceedings of the 20th International Conference on Very Large Data Bases*, ISBN: 1558601538, Publisher: Morgan Kaufmann Publishers Inc., Address: San Francisco, CA, USA, Year: 1994.



PILOT KHADKA is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. During his time at Thapathali Campus, Pilot has actively engaged in various academic and extracurricular activities. He has participated in coding competitions, collaborated on software development projects, and demonstrated a keen interest in exploring new technologies.

KSHITIZ POUDEL is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. His relentless pursuit of knowledge and dedication to uplifting and empowering others make him an exceptional contributor and an invaluable asset to the academic community.

VII. APPENDIX

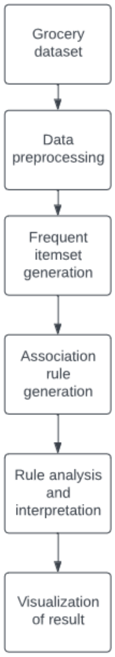


FIGURE 1. System Block Diagram

	Item(s)	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	...	Item 23	Item 24	Item 25	Item 26	Item 27	Item 28	Item 29	Item 30	Item 31	Item 32
0	4	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	3	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

FIGURE 2. Snapshot of dataset

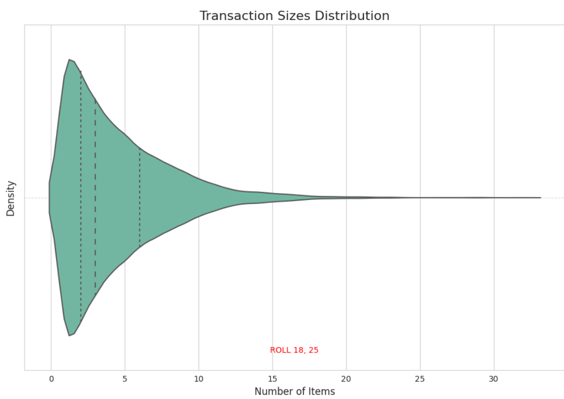


FIGURE 3. violin plot of transaction size distribution

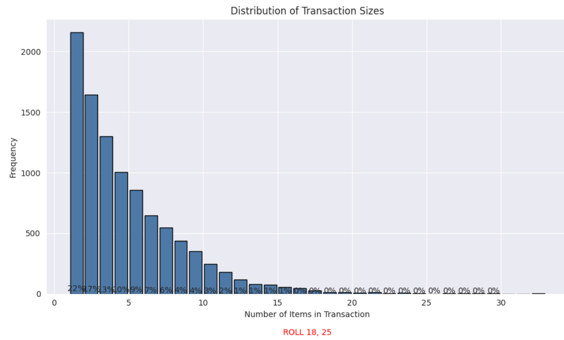


FIGURE 4. Frequency distribution of transaction size

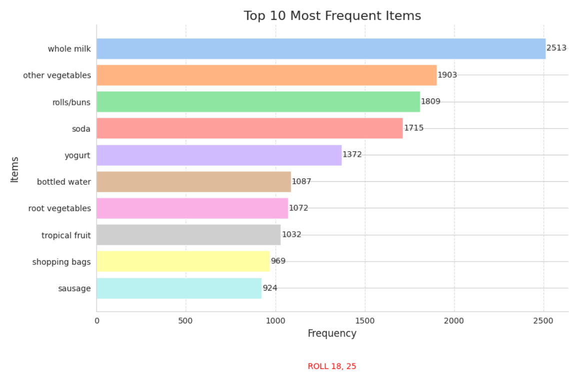


FIGURE 5. Top 10 most frequent items

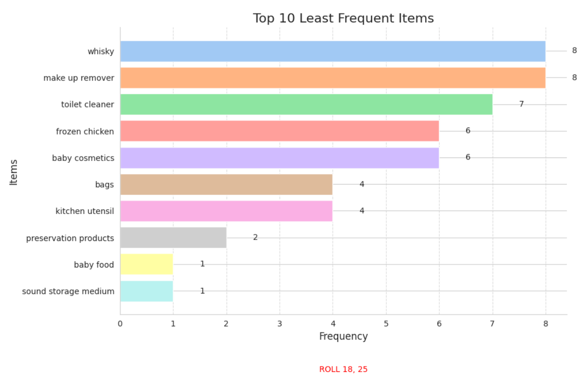


FIGURE 6. Top 10 least frequent items

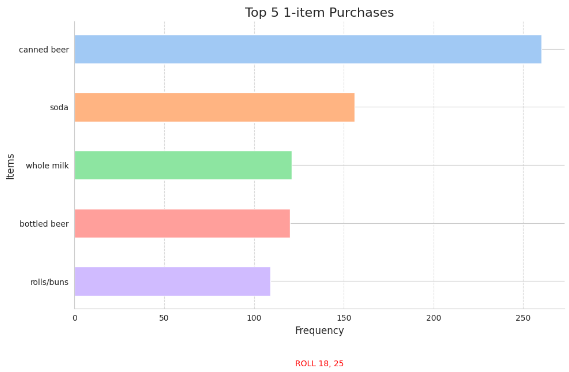


FIGURE 7. Top 5 1 item purchases

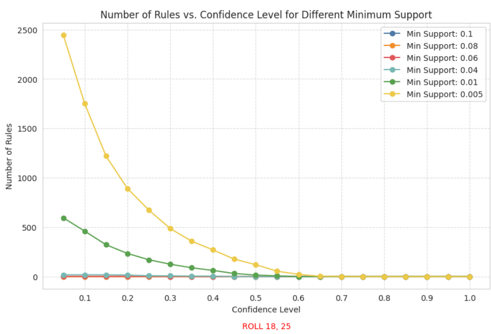


FIGURE 8. Number of rules vs confidence level

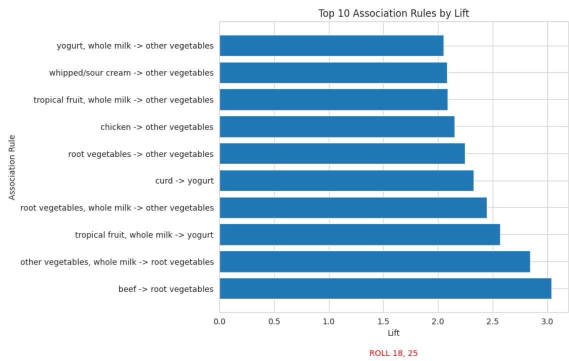


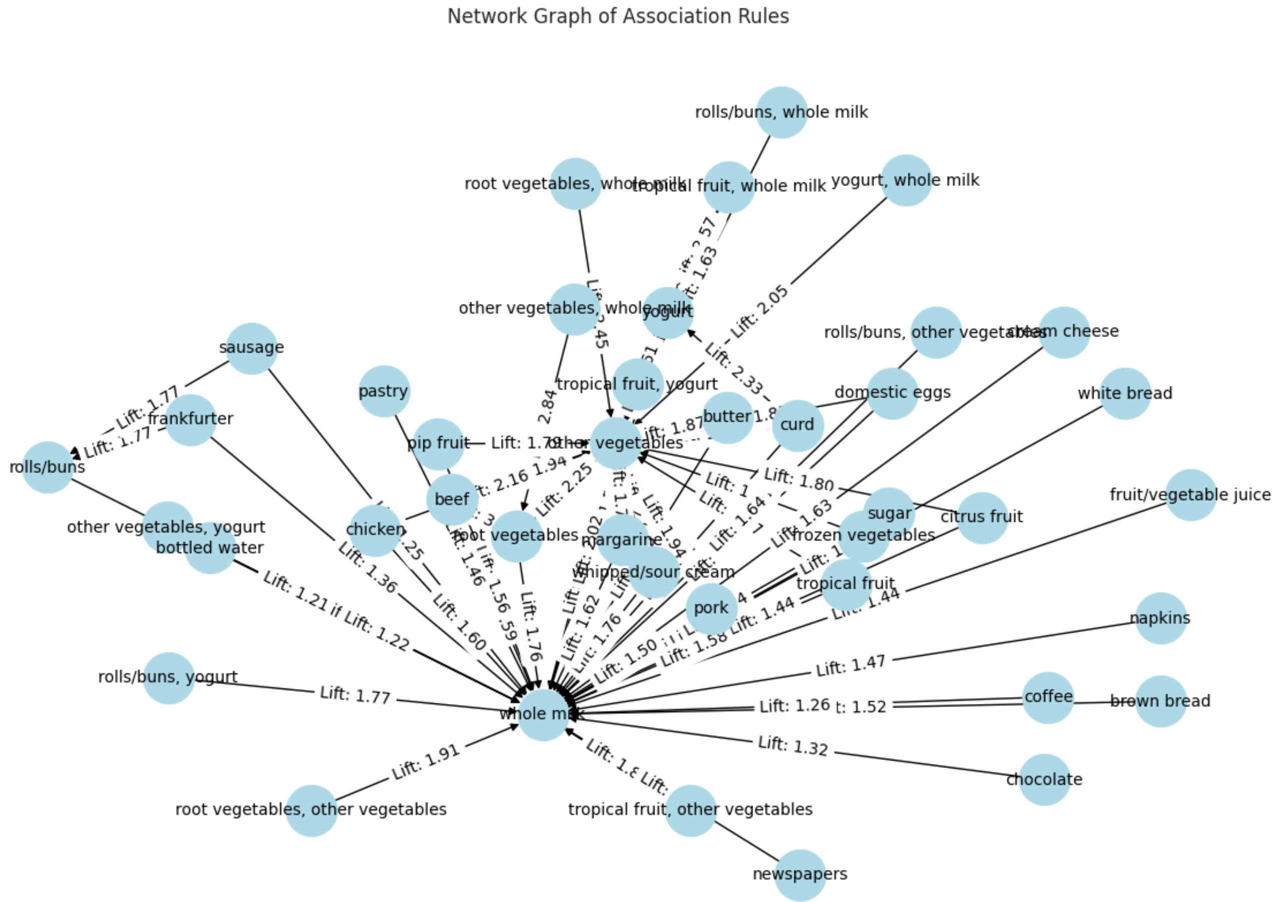
FIGURE 11. Top 10 association rules by lift

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(yogurt)	(whole milk)	0.139502	0.255516	0.056024	0.401603	1.571735	0.020379	1.244132	0.422732
1	(other vegetables)	(whole milk)	0.193493	0.255516	0.074835	0.386758	1.513634	0.025394	1.214013	0.420750
2	(rolls/buns)	(whole milk)	0.183935	0.255516	0.056634	0.307905	1.205032	0.009636	1.075696	0.208496

FIGURE 9. Association rules for minimum support=0.05

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(beef)	(root vegetables)	0.052466	0.108998	0.017387	0.331395	3.040367	0.011668	1.332628	0.708251
1	(other vegetables, whole milk)	(root vegetables)	0.074835	0.108998	0.023183	0.309783	2.842082	0.015026	1.290900	0.700572
2	(tropical fruit, whole milk)	(yogurt)	0.042298	0.139502	0.015150	0.358173	2.567516	0.009249	1.340701	0.637483
3	(root vegetables, whole milk)	(other vegetables)	0.048907	0.193493	0.023183	0.474012	2.449770	0.013719	1.533320	0.622230
4	(curd)	(yogurt)	0.053279	0.139502	0.017285	0.324427	2.325615	0.009853	1.273732	0.602085
5	(root vegetables)	(other vegetables)	0.108998	0.193493	0.047382	0.434701	2.246605	0.026291	1.426693	0.622764
6	(chicken)	(other vegetables)	0.042908	0.193493	0.017895	0.417062	2.155439	0.009593	1.383521	0.560090
7	(tropical fruit, whole milk)	(other vegetables)	0.042298	0.193493	0.017082	0.403846	2.087140	0.008898	1.352851	0.543880
8	(whipped/sour cream)	(other vegetables)	0.071683	0.193493	0.028876	0.402837	2.081924	0.015006	1.350565	0.559803
9	(yogurt, whole milk)	(other vegetables)	0.056024	0.193493	0.022267	0.397459	2.054131	0.011427	1.338511	0.543633
10	(tropical fruit, yogurt)	(whole milk)	0.029283	0.255516	0.015150	0.517361	2.024770	0.007668	1.542528	0.521384
11	(other vegetables, yogurt)	(whole milk)	0.043416	0.255516	0.022267	0.512881	2.007235	0.011174	1.528340	0.524577
12	(butter)	(whole milk)	0.055414	0.255516	0.027555	0.497248	1.946053	0.013395	1.480817	0.514659
13	(beef)	(other vegetables)	0.052466	0.193493	0.019725	0.375969	1.943066	0.009574	1.292416	0.512224
14	(pork)	(other vegetables)	0.057651	0.193493	0.021657	0.375661	1.941476	0.010502	1.291779	0.514595

FIGURE 10. Association rules for minimum support =0.02



ROLL 18, 25

FIGURE 12. Network graph of association rules

VIII. CODE

```
# -*- coding: utf-8 -*-
"""apriori.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1a8msdNSGQpLGM3lYE-6H2hct8_psDmcV
"""

import pandas as pd
import numpy as np
import math

from google.colab import drive
drive.mount('/content/drive')

path = '/content/drive/MyDrive/Colab_Notebooks/groceries2.csv'

data_df = pd.read_csv(path)
data_df

new_df = data_df.head(50)
# Display the new DataFrame
new_df

new_df.drop('Item(s)', axis=1, inplace=True)

new_df.head()

"""#Data visualization"""

import matplotlib.pyplot as plt
import seaborn as sns
num_items_per_transaction = data_df["Item(s)"]

# Set the style using Seaborn
sns.set_style("whitegrid")
plt.figure(figsize=(10, 7))

# Create a violin plot with custom colors
colors = ['#66c2a5', '#fc8d62', '#8da0cb', '#e78ac3']
sns.violinplot(x=num_items_per_transaction, palette=colors, inner="quartile")

# Customize the plot for better presentation
plt.title("Transaction Sizes Distribution", fontsize=16)
plt.xlabel("Number of Items", fontsize=12)
plt.ylabel("Density", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Add extra legend
extra_legend = "ROLL 18, 25"
plt.text(0.5, 0.05, extra_legend, ha='center', color='red', transform=plt.gca().transAxes)

# Show the plot
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))

# Use patch_artist=True to fill the box plot with colors
plt.boxplot(num_items_per_transaction, vert=False, patch_artist=True)

# Customize the plot for better presentation
plt.title("Distribution of Transaction Sizes")
plt.xlabel("Number of Items")
plt.grid(axis="x")  # Show grid lines on the x-axis for better readability
plt.yticks([])  # Remove the labels for the y-axis

# Add extra legend
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.15, extra_legend, ha='center', color='red', transform=plt.gca().transAxes)

plt.tight_layout()

# Show the plot
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns
num_items_per_transaction = data_df["Item(s)"]
min_size = min(num_items_per_transaction)
max_size = max(num_items_per_transaction)

# Set the style using Seaborn
sns.set_style("darkgrid")
plt.figure(figsize=(10, 6))

# Create bins with a size of 1
bins = list(range(min_size, max_size + 2))

# Create histograms for transaction sizes with a custom color
colors = ['#4e79a7']
plt.hist(num_items_per_transaction, bins=bins, edgecolor="black", color=colors, rwidth=0.85)
plt.title("Distribution of Transaction Sizes")
plt.xlabel("Number of Items in Transaction")
plt.ylabel("Frequency")

# Calculate the percentage of each number of items in the "Item(s)" column
item_count = data_df["Item(s)"].value_counts()
total_transactions = len(data_df["Item(s)"])
percentage_item_purchases = (item_count / total_transactions) * 100

# Add percentage labels above bars
for i, percentage in enumerate(percentage_item_purchases):
    plt.text(bins[i] + 0.5, percentage + 2, f"{round(percentage)}%", ha='center', fontsize=10)

# Add extra legend
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.15, extra_legend, ha='center', color='red', transform=plt.gca().transAxes)

plt.tight_layout()

# Show the plot
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns

# Set the style using Seaborn
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))

# Calculate the frequency of each item in the DataFrame
item_counts = data_df.iloc[:, 1:].stack().
```

```
    value_counts()

# Select the top 10 most frequent items
top_10_items = item_counts.head(10)

# Create a horizontal bar plot with a custom color
colors = sns.color_palette("pastel")
plt.barh(top_10_items.index, top_10_items.values,
    color=colors)

# Customize the plot for better presentation
plt.gca().invert_yaxis()  # Invert the y-axis
plt.title("Top 10 Most Frequent Items", fontsize
    =16)
plt.xlabel("Frequency", fontsize=12)
plt.ylabel("Items", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.gca().spines["right"].set_visible(False)  #
    Remove right border
plt.gca().spines["top"].set_visible(False)  #
    Remove top border
plt.tight_layout()

# Add values inside the bars
for index, value in enumerate(top_10_items.values)
    :
    plt.text(value + 3, index, str(value), va="
    center", fontsize=10)

# Add extra legend
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.2, extra_legend, ha='center',
    color='red', transform=plt.gca().transAxes)

# Show the plot
plt.show()

# Set the style using Seaborn
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))

# Calculate the frequency of each item in the
    DataFrame
item_counts = data_df.iloc[:, 1:].stack().
    value_counts()

# Select the bottom 10 least frequent items
bottom_10_items = item_counts.tail(10)

# Create a horizontal bar plot with a custom color
colors = sns.color_palette("pastel")
bars = plt.barh(bottom_10_items.index,
    bottom_10_items.values, color=colors)

# Customize the plot for better presentation
plt.gca().invert_yaxis()  # Invert the y-axis
plt.title("Top 10 Least Frequent Items", fontsize
    =16)
plt.xlabel("Frequency", fontsize=12)
plt.ylabel("Items", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.gca().spines["right"].set_visible(False)  #
    Remove right border
plt.gca().spines["top"].set_visible(False)  #
    Remove top border
plt.tight_layout()

# Add values inside the bars at the edge
for bar, value in zip(bars, bottom_10_items.values
    ):
    plt.text(bar.get_width() + 0.5, bar.get_y() +
    bar.get_height() / 2, str(value), va="center",
     fontsize=10)

# Add extra legend
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.2, extra_legend, ha='center',
    color='red', transform=plt.gca().transAxes)

# Show the plot
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns

# Set the style using Seaborn
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))

# Extract the item columns
item_columns = data_df.columns[1:33]

# Create a new DataFrame to store the standalone
    purchases (transactions with only one item)
standalone_purchases = data_df[data_df["Item(s)"]
    == 1][item_columns]

# Item Frequency Analysis for Standalone Purchases
standalone_item_counts = standalone_purchases.
    stack().value_counts()
top_standalone_items = standalone_item_counts.head
    (5)  # Get the top 5 most frequent standalone
    items

# Create a horizontal bar plot with a custom color
colors = sns.color_palette("pastel")
plt.barh(top_standalone_items.index,
    top_standalone_items.values, color=colors,
    height=0.5)

# Customize the plot for better presentation
plt.gca().invert_yaxis()  # Invert the y-axis
plt.title("Top 5 1-item Purchases", fontsize=16)
plt.xlabel("Frequency", fontsize=12)
plt.ylabel("Items", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.gca().spines["right"].set_visible(False)  #
    Remove right border
plt.gca().spines["top"].set_visible(False)  #
    Remove top border
plt.tight_layout()

# Add extra legend
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.2, extra_legend, ha='center',
    color='red', transform=plt.gca().transAxes)

# Show the plot
plt.show()

# Initialize a list to store transactions
transactions = []

# Iterate through each row in the DataFrame
for index, row in new_df.iterrows():
    # Convert row to a list and remove 'NaN' items
    transaction = [item for item in row if pd.
    notna(item)]
    transactions.append(transaction)

# Display the first two transactions
transactions[:2]
```

```python

# Calculate the number of non-missing values in each transaction
num_non_missing_per_transaction = data_df.iloc[:, 1:33].count(axis=1)

num_items_per_transaction = data_df["Item(s)"]

is_complete_transaction = (
    num_non_missing_per_transaction == num_items_per_transaction)

# Count the number of rows with missing values or mismatches in item counts
num_incomplete_transactions = (~is_complete_transaction).sum()

# Print the number of incomplete transactions
print("Number of Incomplete Transactions:", num_incomplete_transactions)

transactions_df = pd.DataFrame(transactions)
transactions_df

"""##Frequent Itemset Generation using Apriori Algorithm"""

from itertools import combinations
from collections import defaultdict

# Minimum support count
min_support = 2

# Generate candidate 1-itemsets
item_counts = defaultdict(int)
for transaction in transactions:
    for item in transaction:
        item_counts[item] += 1

# Prune infrequent items
frequent_1_itemsets = {item for item, count in item_counts.items() if count >= min_support}

# Generate frequent itemsets of size k
def generate_frequent_itemsets(itemsets, k):
    candidates = set()
    for itemset in itemsets:
        for item in frequent_1_itemsets:
            if item not in itemset:
                candidates.add(itemset + (item,))

    frequent_itemsets = set()
    for candidate in candidates:
        count = sum(1 for transaction in transactions if all(item in transaction for item in candidate))
        if count >= min_support:
            frequent_itemsets.add(candidate)

    return frequent_itemsets

# Generate frequent itemsets of increasing size
k = 2
frequent_itemsets = {(item,) for item in frequent_1_itemsets}
while frequent_itemsets:
    print(f"Frequent {k-1}-itemsets: {frequent_itemsets}")
    k += 1
    candidate_itemsets = generate_frequent_itemsets(frequent_itemsets, k)
    frequent_itemsets = candidate_itemsets

frequent_itemsets

"""##Rule Generation"""

# Generate association rules from frequent itemsets
def generate_association_rules(frequent_itemset):
    rules = []
    for itemset in frequent_itemset:
        for i in range(1, len(itemset)):
            for subset in combinations(itemset, i):
                remaining = tuple(item for item in itemset if item not in subset)
                support_itemset = sum(1 for transaction in transactions if all(item in transaction for item in itemset))
                support_subset = sum(1 for transaction in transactions if all(item in transaction for item in subset))
                confidence = support_itemset / support_subset
                if confidence >= 0.1:  # Minimum confidence threshold
                    rules.append((subset, remaining, confidence))
    return rules

# Generate association rules from frequent itemsets
association_rules = []
for frequent_itemset in frequent_itemsets:
    rules = generate_association_rules(frequent_itemset)
    association_rules.extend(rules)

# Print association rules
for antecedent, consequent, confidence in association_rules:
    print(f"Rule: {antecedent} => {consequent}, Confidence: {confidence:.2f}")

"""## Library Implementation"""

# Extract the item columns
item_columns = data_df.columns[1:33]

# Convert the data into a list of transactions
transactions = data_df[item_columns].apply(lambda row: row.dropna().tolist(), axis=1).tolist()

# Create a one-hot encoded DataFrame for the transactions
onehot_transactions = pd.DataFrame(transactions)

# Apply one-hot encoding
onehot_encoded = pd.get_dummies(onehot_transactions.unstack()).groupby(level=1).max()

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# List of minimum support values
min_support_values = [0.1, 0.08, 0.06, 0.04, 0.01, 0.005]  # Adding more values

# Confidence levels to evaluate
confidence_levels = list(np.arange(0.05, 1.05, 0.05))

# Empty lists to store results
```

```python
num_rules_lists = []

# Calculate and store the number of rules for each combination of minimum support and confidence level
for min_support in min_support_values:
    frequent_itemsets = apriori(onehot_encoded, min_support=min_support, use_colnames=True)
    rules_list = []
    for confidence_level in confidence_levels:
        rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=confidence_level)
        num_rules = len(rules)
        rules_list.append(num_rules)
    num_rules_lists.append(rules_list)

# Plot the results
plt.figure(figsize=(10, 6))

colors = ["#4e79a7", "#f28e2b", "#e15759", "#76b7b2", "#59a14f", "#edc948"]  # Changed line colors

for i, min_support in enumerate(min_support_values):
    plt.plot(confidence_levels, num_rules_lists[i], marker="o", color=colors[i], alpha=0.9, label=f"Min Support: {min_support}")

plt.xlabel("Confidence Level")
plt.ylabel("Number of Rules")
plt.title("Number of Rules vs. Confidence Level for Different Minimum Support")

# Set the desired x-axis labels
plt.xticks([0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1])

# Add grid lines for better readability
plt.grid(True, linestyle="--", alpha=0.3)

plt.legend()

# Add extra legend
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.15, extra_legend, ha='center', color='red', transform=plt.gca().transAxes)

plt.show()

# Run Apriori algorithm with a minimum support threshold of 0.05

from mlxtend.frequent_patterns import apriori, association_rules
frequent_itemsets = apriori(onehot_encoded, min_support=0.015, use_colnames=True)

# Sort the frequent itemsets DataFrame by 'support' in descending order
sorted_frequent_itemsets = frequent_itemsets.sort_values(by="support", ascending=False)

# Calculate the length of each itemset and add it as a new column 'length'
sorted_frequent_itemsets["length"] = sorted_frequent_itemsets["itemsets"].apply(len)

# Display the sorted frequent itemsets DataFrame with the desired formatting
with pd.option_context("display.max_rows", None, "display.max_columns", None, "display.float_format", '{:.3f}'.format):
    print(sorted_frequent_itemsets)

# Generate association rules
association_rules_df = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3)

# Sort the association rules by 'lift' metric in descending order and reset the index
sorted_association_rules = association_rules_df.sort_values(by="lift", ascending=False).reset_index(drop=True)

# Display the sorted association rules DataFrame
print("\nAssociation Rules:")
sorted_association_rules



association_rules_df = association_rules_df.sort_values(by='lift', ascending=False)

# Choose the top N rules to display
top_n = 10
top_df = association_rules_df.head(top_n)

# Convert frozenset objects to strings for concatenation
top_df['rule'] = top_df['antecedents'].apply(lambda x: ', '.join(x)) + ' -> ' + top_df['consequents'].apply(lambda x: ', '.join(x))

plt.figure(figsize=(10, 6))
plt.barh(top_df['rule'], top_df['lift'])
plt.xlabel('Lift')
plt.ylabel('Association Rule')
plt.title(f'Top {top_n} Association Rules by Lift')
plt.tight_layout()
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.15, extra_legend, ha='center', color='red', transform=plt.gca().transAxes)
plt.show()

import networkx as nx
G = nx.DiGraph()
# Add nodes and edges
for index, row in association_rules_df.iterrows():
    antecedent = ', '.join(row['antecedents'])
    consequent = ', '.join(row['consequents'])
    G.add_node(antecedent)
    G.add_node(consequent)
    G.add_edge(antecedent, consequent, weight=row['lift'])

# Position nodes using a spring layout algorithm
pos = nx.spring_layout(G)
plt.figure(figsize=(12, 8))

# Draw nodes and edges
nx.draw(G, pos, with_labels=True, font_size=10, node_size=1000, node_color='lightblue')
nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): f'Lift: {d["weight"]:.2f}' for u, v, d in G.edges(data=True)})
plt.title('Network Graph of Association Rules')
extra_legend = "ROLL 18, 25"
plt.text(0.5, -0.05, extra_legend, ha='center', color='red', transform=plt.gca().transAxes)

plt.show()
```

• • •