Asian MathSci League, Inc (AMSLI)
Partner: National Olympiad in Informatics Philippines (NOI.PH)
Email address: amsliphil@yahoo.com / ask@noi.ph
Contact Nos: +632-9254526 +63906-1186067

Student Copy
AIEP Send-off Python 2 Session 10

ASSOCIATION RULE MINING

> *"Without big data analytics, companies are blind and deaf, wandering out onto the web like deer on a freeway."*
>
> *~ Geoffrey Moore*

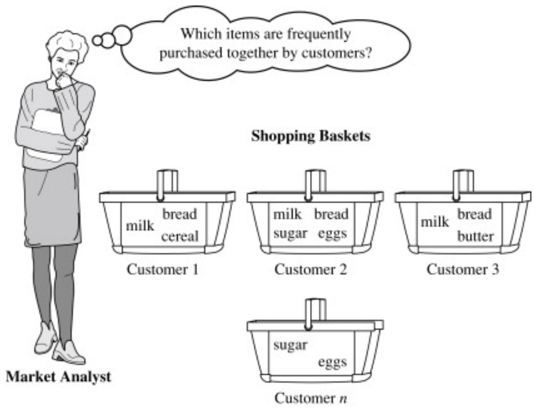
In the past sessions, we have dissected the facets of computational thinking, explored some algorithmic paradigms, and studied efficient data structures. To synthesize the skills and insights that you have acquired over the course of this ten-week program, it is only fitting to send you off with a real-world application of informatics.

We live in a data-driven world, and it is imperative that we make sense of these volumes of data and transform them into meaningful and actionable insights — this process is known as data mining[1]. In this handout, we will be learning an approach that is used extensively to unearth patterns across items in large datasets: association rule mining.

Association rule mining was popularized in a 1993 paper by Rakesh Agrawal, Tomasz Imieliński, and Arun Swami as a method for making retail-related assertions such as "People who buy bread and lettuce are likely to purchase ham and cheese as well." Formally referred to as association rules, these findings help inform high-impact decisions:

- If a store discontinues selling bread and lettuce, what products will be affected?
- A store is planning to hold a tie-in sale. Would it be prudent to lower the prices of bread and lettuce, and compensate for this by increasing the prices of ham and cheese — or would it be more strategic to do it the other way around?
- What products should be placed together on the same shelf or the same aisle to take advantage of customers' purchasing patterns? For instance, what items could be placed near the bread rack to increase the chances that shoppers would also go and grab some ham and cheese?

The final challenge will be to create a Python script for mining useful association rules from a grocery dataset with over 9,800 recorded transactions!



Step 0 MARKET BASKET MODEL

We start our discussion with an introduction to the market basket model, which is used to describe a many-to-many relationship between items and baskets. In the context of a grocery store, the items are the products while a basket is a customer's transaction for the day (i.e., their shopping basket). Ergo, in set-theoretic terms, a basket is a subset of the universal collection of items.

[1] A common misconception is to equate data mining with the process of extracting the data themselves.

This model is not limited to consumer behavior. In fact, it can be applied to a wide variety of research areas, some of which are presented in the table below:

Application	Items	Baskets	Remarks
Detecting plagiarism	Documents where the sentences appear	Sentences	Documents that appear together quite frequently may be indicative of plagiarism[2].
Identifying side effects of drugs	Drugs and side effects	Patients	Results can be used to study drug interactions or contraindications.
Determining biomarkers	Biomarkers and diseases	Patients	Results can help recommend appropriate tests (e.g., blood chemistry tests) for diseases.
Organizing search results	Pages that link to the web pages	Web pages	Pages with several common links most likely share similar topics.

Note that the market basket model assumes the following:

- There is a large collection of items.
- There is a large number of baskets.
- Each basket contains only a small subset of the items — much smaller than the universal collection of items.

While there are no formal or rigorous definitions as to what *large*, *small*, or *much smaller* mean, it fortunately does not need much convincing to believe that most grocery stores (and other item-basket relationships in the real world) satisfy these assumptions.

Step 1 FREQUENT ITEMSETS

An itemset is a shorthand for a set of items, and, as suggested by the applications of the market basket model, we are interested in finding the itemsets that appear in several of the baskets.

To formalize, the support of an itemset is defined as the number of baskets in which it appears as a subset. An itemset is considered frequent if its support is greater than or equal to the given support threshold.

To illustrate, suppose we have this collection of items {🍇, 🥬, 🥩, 🧀, 🍒} and 10 baskets:

B_1: {🥬, 🥩}

B_2: {🍇, 🥬, 🥩}

B_3: {🍒}

B_4: {🥬, 🥩, 🧀}

B_5: {🍇, 🥬}

B_6: {🍇, 🥬, 🥩, 🍒}

B_7: {🧀}

B_8: {🍇, 🥬, 🍒}

B_9: {🥩, 🧀}

B_{10}: {🍇, 🥬, 🥩, 🧀}

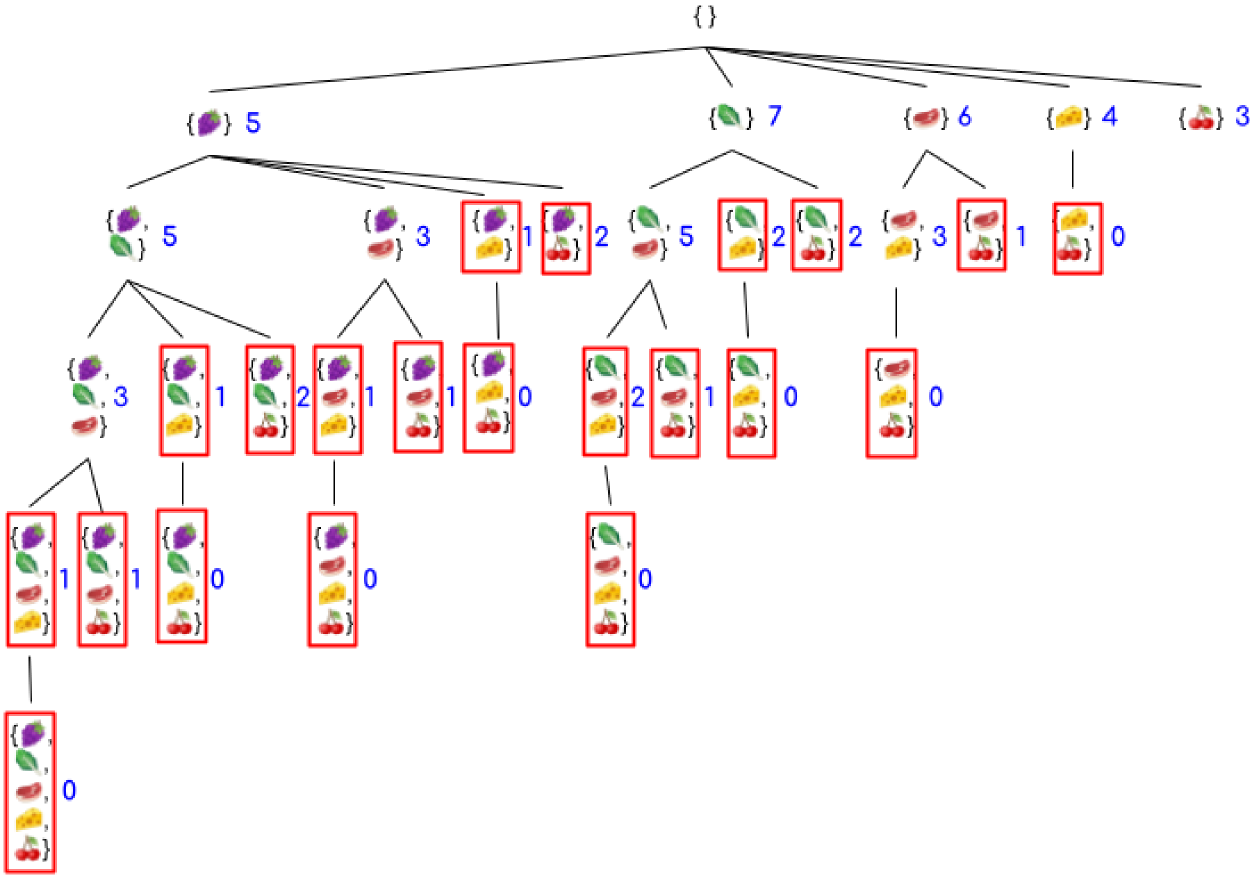
[2] As seen in this example, items and baskets are abstract concepts whose relationship does not necessarily correspond to that of their real-world counterparts. Although sentences are, in reality, inside documents, the market basket model for plagiarism detection frames documents (items) as being “inside” sentences (baskets). In this regard, it might make more sense to think of a basket as a grouping of items that appear together.

Suppose the support threshold is 3:

- {🍇, 🥬, 🥩} is frequent since its support is 3, appearing in B_2, B_6, and B_{10}.
- {🍇, 🍒} is not frequent since its support is only 2, appearing only in B_6 and B_8.

While it is fairly trivial to check whether an itemset is frequent, complications arise if our task changes to finding *all* the frequent itemsets. The naïve approach of generating all $2^n - 1$ non-empty itemsets and solving for their support is computationally expensive and impractical for real-world datasets, which may contain millions or even billions of rows.

But how can we do better? Let us jumpstart our investigation by representing the task of generating all the possible itemsets via a prefix tree (introduced in Session 8, during our discussion of Huffman coding); the number on the right of each itemset corresponds to its support. Additionally, itemsets that fall below the support threshold are boxed.



Below are some salient characteristics that can be noticed from this prefix tree:

- A parent is a subset of its children (or a child is a superset of its parent).
- The support of a node does not exceed the support of its parent.
- If a parent does not meet the support threshold, its children do not meet it as well.
- If a node meets the support threshold, its parent meets the threshold as well.

These observations are encapsulated in the anti-monotonicity property, also referred to as the downward closure property (its converse is not necessarily true):

If the set S is a frequent itemset, then all the subsets of S are frequent as well.

Its contrapositive also holds:
If at least one subset of S is not frequent, then S is not a frequent itemset.

This property motivates the apriori algorithm, introduced by Rakesh Agrawal (one of the authors of the 1993 association rule mining paper) and Ramakrishnan Srikant in a 1994 paper. Essentially a breadth-first search strategy:

- k-element itemsets are generated and the values of their support are computed.
- Subsets that fall below the support threshold are "pruned" (i.e., their descendants or supersets are also excluded), thereby reducing the search space.
- $(k + 1)$-element itemsets are then constructed from the items in the remaining sets.
- This cycle of pruning and generation continues until no new frequent itemset is found.

Worked Example

Let us walk through the algorithm; itemsets that fail to meet the support threshold of 3 are denoted with a strikethrough. We start with the singletons (single-element sets):

- {🍇} has a support of 5.
- {🥬} has a support of 7.
- {🥩} has a support of 6.
- {🧀} has a support of 4.
- {🍒} has a support of 3.

All the singletons are frequent. We proceed to enumerate 2-element itemsets:

- {🍇, 🥬} has a support of 5.
- {🍇, 🥩} has a support of 3.
- ~~{🍇, 🧀} has a support of 1~~.
- ~~{🍇, 🍒} has a support of 2~~.
- {🥬, 🥩} has a support of 5.
- ~~{🥬, 🧀} has a support of 2~~.
- ~~{🥬, 🍒} has a support of 2~~.
- {🥩, 🧀} has a support of 3.
- ~~{🥩, 🍒} has a support of 1~~.
- ~~{🧀, 🍒} has a support of 0~~.

The items that can be found in the remaining itemsets are 🍇, 🥬, 🥩, and 🧀. Using these, we generate the 3-element itemsets:

- {🍇, 🥬, 🥩} has a support of 3.
- ~~{🍇, 🥬, 🧀} has a support of 1~~.
  This can be discarded from the get-go since its subset {🍇, 🧀} is not frequent.
- ~~{🍇, 🥩, 🧀} has a support of 1~~.
  This can be discarded from the get-go since its subset {🍇, 🧀} is not frequent.
- ~~{🥬, 🥩, 🧀} has a support of 2~~.
  This can be discarded from the get-go since its subset {🥬, 🧀} is not frequent.

The items that can be found in the remaining itemset are 🍇, 🥬, and 🥩. No new itemset can be derived from this, and the apriori algorithm terminates. In summary, there are 5 frequent non-singleton itemsets:

- {🍇, 🥬}
- {🍇, 🥩}
- {🥬, 🥩}
- {🥩, 🧀}
- {🍇, 🥬, 🥩}

It may be fitting to point out that we had to compute the support of only 19 itemsets; we could have even reduced this to 16 by fully exploiting anti-monotonicity (although note that this also incurs additional space and time requirements). Compare this with the naïve approach, which would have forced us to get the support of all 31 non-empty itemsets.

Step 2 ASSOCIATION RULES

Finally, we are ready to dive into association rules! An association rule is an implication (if-then statement) of the form $X \rightarrow Y$, where X and Y are disjoint subsets[3] of the itemset I (i.e., they do not have any items in common). X and Y are called the antecedent and the consequent, respectively.

This rule is interpreted as "If all the items in X appear in some basket, then it is likely that all the items in Y will also appear in this basket." In the context of retail, this translates to "People who buy all the items in X are likely to purchase all the items in Y as well."

The direction of the association is important. $X \rightarrow Y$ is not the same as $Y \rightarrow X$.

Naturally, we would like some form of measurement of the strength of this association; this is formalized as the confidence metric, which is computed as

$$confidence(X \rightarrow Y) = \frac{support(X \cup Y)}{support(X)}.$$

In this regard, an association rule is considered "useful"[4] if it satisfies two conditions:

- The support of I is greater than or equal to the support threshold.
  (This is the reason for getting the frequent itemsets.)
- The confidence is greater than or equal to the confidence threshold.

A systematic scheme for generating all the candidate association rules is by partitioning each non-singleton frequent itemset. Rules that fall below the threshold are subsequently discarded.

Worked Example

Setting the confidence threshold to 60%, we proceed to list the candidate association rules and place a strikethrough on those that fail to meet the threshold:

- Frequent Itemset: {🍇, 🥬}
  - {🍇} → {🥬} has a confidence of $\frac{5}{5} = 1.$
  - {🥬} → {🍇} has a confidence of $\frac{5}{7} \approx 0.71.$
- Frequent Itemset: {🍇, 🥩}
  - {🍇} → {🥩} has a confidence of $\frac{3}{5} = 0.6.$
  - ~~{🥩} → {🍇} has a confidence of~~ $\frac{3}{6} = 0.5.$
- Frequent Itemset: {🥬, 🥩}
  - {🥬} → {🥩} has a confidence of $\frac{5}{7} \approx 0.71.$
  - {🥩} → {🥬} has a confidence of $\frac{5}{6} \approx 0.83.$
- Frequent Itemset: {🥩, 🧀}
  - ~~{🥩} → {🧀} has a confidence of~~ $\frac{3}{6} = 0.5.$
  - {🧀} → {🥩} has a confidence of $\frac{3}{4} = 0.75.$

[3] As a historical sidenote, the original 1993 paper that popularized association rule mining restricted Y to be a single item. However, in the 1994 paper where the apriori algorithm was introduced, it was relaxed to a set.

[4] "Useful" is not a technical term.

- Frequent Itemset: {🍇, 🥬, 🥩}
  - {🍇, 🥬} → {🥩} has a confidence of $\frac{3}{5} = 0.6$.
  - {🍇, 🥩} → {🥬} has a confidence of $\frac{3}{3} = 1$.
  - {🥬, 🥩} → {🍇} has a confidence of $\frac{3}{5} = 0.6$.
  - {🍇} → {🥬, 🥩} has a confidence of $\frac{3}{5} = 0.6$.
  - ~~{🥬} → {🍇, 🥩} has a confidence of~~ $\frac{3}{7} \approx 0.43$.
  - ~~{🥩} → {🍇, 🥬} has a confidence of~~ $\frac{3}{6} = 0.5$.

💭 Confidence is Not Enough

The confidence metric was introduced in the 1993 association rule mining paper, but it has its fair share of inadequacies. Various authors have proposed alternative metrics to evaluate and derive insights from association rules:

1. Interest[5]. This is computed as

$$interest(X \to Y) = \left| confidence(X \to Y) - \frac{support(Y)}{number\ of\ baskets} \right|.$$

   A value of 0 indicates that the fraction of baskets containing Y and the fraction of those containing $X \cup Y$ are identical. A high interest implies that the presence of X either "encourages" or "discourages" the presence of Y in the same basket. Examples of items that "discourage" co-occurrence include competing brands of the same product.

2. Lift[6]. This is computed as

$$lift(X \to Y) = \frac{confidence(X \to Y)}{\frac{support(Y)}{number\ of\ baskets}}.$$

   A value of 1 suggests that X and Y are statistically independent, i.e., they do not influence each other's presence. A value above 1 means that they "encourage" co-occurrence while a value below 1 implies that they "discourage" it.

3. Conviction[7]. This is computed as

$$conviction(X \to Y) = \frac{1 - \frac{support(Y)}{number\ of\ baskets}}{1 - confidence(X \to Y)}.$$

   This helps assess whether or not the rule is only a product of chance. A value of 1 indicates that X and Y are statistically independent. A high value suggests that the presence of Y in the same basket is, indeed, influenced by the presence of X.

For a more detailed discussion of these measures (along with other evaluation metrics), you may refer to this paper.

[5] Leskovec, J., Rajaraman, A., & Ullman, J. (2019). *Mining of massive datasets* (3rd ed.). Cambridge University Press. The original formulation in this book assumes Y to be a single item. We take the liberty of extending Y to be a set for consistency with our discussion.

[6] Brin, S., Motwani, R., Ullman, J.D., & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. *ACM SIGMOD Record, 26*(2), 225–264.

[7] Ibid.

🧑‍💻 EXERCISES

Concept Check[8]

1. A drawback of using confidence as a metric is that it ignores the probability of the occurrence of the consequent. Why is this a drawback? Explain why lift and conviction do not suffer from this drawback.

2. A measure is symmetrical if $measure(A \rightarrow B) = measure(B \rightarrow A)$. Which among lift, confidence, and conviction are symmetrical?

3. Perfect implications are rules that hold 100% of the time (or equivalently, the associated conditional probability is 1). A measure is desirable if it reaches its maximum achievable value for all perfect implications. This makes it easy to identify the best rules.

   Which among lift, confidence, and conviction have this property? You may ignore $0/0$ but not other infinity cases.

Programming

Following the principles of problem decomposition and abstraction, you are given a series of tasks to accomplish in order to create an association rule miner.

💡 Maximize Built-in Methods and Libraries

Python provides built-in methods for taking the union of sets and for checking if a set is a subset of another set; the details are found in the official documentation. The Python Standard Library also has a module for generating the subsets of a set: `itertools`.

However, the use of third-party libraries, like `mlxtend`, defeats this activity's purpose of serving as a practice for translating high-level algorithms to code.

A. Input Parsing

You are given a set of baskets encoded in transaction format[9]. The first line consists of three numbers separated by commas:

- The number of baskets (`int`)
- The support threshold (`int`)
- The threshold for evaluating the association rule (`float`)

Each succeeding line of input corresponds to one basket, with the items in each basket separated by commas as well.

[8] The questions were taken from Stanford University's CS246: Mining Massive Data Sets course.
[9] There are two other widely used formats: *binary format* and *vertical format*. For more information, you may refer to pages 220 to 221 of this book (these pages are included in the free preview).

Aside from the variables for the number of baskets and the thresholds, you should have the following lists at the end of input parsing:

List	Description
`items`	A one-dimensional list storing all the unique items
`baskets`	A two-dimensional list storing all the baskets. Each basket is a one-dimensional list containing the items in that basket.

A sample input is provided below:

```
10,3,0.6
lettuce,ham
grapes,lettuce,ham
cherry
lettuce,ham,cheese
grapes,lettuce
grapes,lettuce,ham,cherry
cheese
grapes,lettuce,cherry
ham,cheese
grapes,lettuce,ham,cheese
```

After parsing the input, the contents of `items` and `baskets`, along with the thresholds, should reflect the example in our discussion (🍇 is `grapes`, 🥬 is `lettuce`, 🥩 is `ham`, 🧀 is `cheese`, and 🍒 is `cherry`):

Variable/List	Contents
`num_baskets`	`10`
`support_threshold`	`3`
`eval_threshold`	`0.6`
`items`	`['grapes', 'cheese', 'lettuce', 'cherry', 'ham']`

`baskets`	`[['lettuce', 'ham'],`
	`['grapes', 'lettuce', 'ham'],`
	`['cherry'],`
	`['lettuce', 'ham', 'cheese'],`
	`['grapes', 'lettuce'],`
	`['grapes', 'lettuce', 'ham', 'cherry'],`
	`['cheese'],`
	`['grapes', 'lettuce', 'cherry'],`
	`['ham', 'cheese'],`
	`['grapes', 'lettuce', 'ham', 'cheese']]`

B. Frequent Itemsets

Using the apriori algorithm, implement the method `get_frequent_itemsets`. There is no need to incorporate the scheme wherein some itemsets are discarded at the get-go for containing an infrequent subset.

Parameters	Description
`items`	A one-dimensional list storing all the unique items
`baskets`	A two-dimensional list storing all the baskets
`support_threshold`	An integer denoting the minimum support for an itemset to be considered frequent

Return Value
A list of tuples corresponding to all the frequent non-singleton itemsets and their respective support. The tuples are arranged in order of decreasing support. The first element of each tuple is the itemset: a one-dimensional list containing the items in that set. The second element of each tuple is the support of the itemset.

Function Call	Result

`get_frequent_itemsets(items, baskets, support_threshold)`	`[(['grapes', 'lettuce'], 5),`
	`(['lettuce', 'ham'], 5),`
	`(['grapes', 'ham'], 3),`
	`(['cheese', 'ham'], 3),`
	`(['grapes', 'lettuce', 'ham'], 3)]`

C. Metrics

Implement methods for computing the four metrics discussed, namely `get_confidence`, `get_interest`, `get_lift`, and `get_conviction`.

Parameters	Description
`antecedent`	A one-dimensional list corresponding to the rule's antecedent
`consequent`	A one-dimensional list corresponding to the rule's consequent
`baskets`	A two-dimensional list storing all the baskets

Return Value
A floating-point number corresponding to the value of the metric

Function Call	Result
`get_confidence(['grapes', 'lettuce'], ['cherry'], baskets)`	`0.4`
`get_interest(['grapes', 'lettuce'], ['cherry'], baskets)`	`0.10000000000000003`
`get_lift(['grapes', 'lettuce'], ['cherry'], baskets)`	`1.3333333333333335`
`get_conviction(['grapes', 'lettuce'], ['cherry'], baskets)`	`1.1666666666666667`

D. Association Rules

Implement the method `get_useful_association_rules`.

Parameters	Description
`frequent_itemsets`	A list of tuples corresponding to all the frequent non-singleton itemsets and their respective support
`baskets`	A two-dimensional list storing all the baskets
`eval_threshold`	A floating-point number denoting the minimum evaluation for a rule to be considered "useful"
`metric`	Metric for evaluating the association rule (Possible values: `confidence`, `interest`, `lift`, `conviction`)

Return Value
A list of tuples corresponding to all the "useful" association rules and their respective evaluation. The tuples are arranged in order of decreasing evaluation. The first element of each tuple is the rule: a string of the form `X -> Y`, where `X` and `Y` are itemsets. For example, the association rule {ham, cheese} → {lettuce} translates to `{ham, cheese} -> {lettuce}`. The second element of each tuple is the evaluation based on the specified metric.

Function Call	Result

`get_useful_association_rules(`
  `get_frequent_itemsets(items,`
  `baskets, support_threshold),`
  `baskets, eval_threshold,`
  `"confidence")`
	`[('{grapes} -> {lettuce}', 1.0),`
	 `('{grapes, ham} -> {lettuce}', 1.0),`
	 `('{ham} -> {lettuce}',`
	`0.8333333333333334),`
	 `('{cheese} -> {ham}', 0.75),`
	 `('{lettuce} -> {grapes}',`
	`0.7142857142857143),`
	 `('{lettuce} -> {ham}',`
	`0.7142857142857143),`
	 `('{grapes} -> {ham}', 0.6),`
	 `('{grapes} -> {ham, lettuce}', 0.6),`
	 `('{grapes, lettuce} -> {ham}', 0.6),`
	 `('{lettuce, ham} -> {grapes}', 0.6)]`

🛒 Test Your Code With a Dataset

The 1993 paper that popularized association rule mining is titled "Mining association rules between sets of items in *large* databases." It may be interesting to try if your code can mine rules from a grocery dataset with 9,835 recorded transactions[10].

(Admittedly, this is still an extremely small dataset. In 2018, the *Business World*[11] reported that the average daily foot traffic in SM Megamall is already 179,139.)

[10] The dataset was taken from this GitHub repository.
[11] Soliman, M.A.P. (2018, July 16). Despite the advent of online retail, Filipinos still love the mall. *Business World*. https://www.bworldonline.com/despite-the-advent-of-online-retail-filipinos-still-love-the-mall/

Assuming that the dataset is inside the same folder as your program, you may use this code to read from a text file:

```
with open("grocery-dataset.txt") as file:
    transactions = file.read().splitlines()
```

As the name suggests, `transactions` is the list that will contain the 9,835 transactions.

Moving Forward

In these last five sessions, our analysis of the runtime of algorithms has been focused on the asymptotic rate of growth (big O and big theta). However, it is also important to pay attention to low-level (hardware) details. For instance, in data mining, the amount of time to read from and write on a hard drive is a major bottleneck; the disk's physical rotation introduces latency. Unfortunately, it is impossible to do away with secondary storage devices, as many real-world datasets are too large to fit inside the main memory.

To this end, strategies such as the triangular-matrix method and the triples method have been devised to reduce the space complexity of algorithms for itemset counting. Aside from the apriori algorithm, there are also other ways to enumerate the frequent itemsets. One alternative is ECLAT (Equivalence Class Clustering and Bottom-up Lattice Traversal), which employs depth-first search instead of apriori's breadth-first search.

This is just the tip of the iceberg! If you are interested to learn more about association rule mining and data mining in general, below are some suggested readings (these also served as the references for the contents of this handout):

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data,* 207–216. https://doi.org/10.1145/170035.170072

Agrawal, R., & Srikant, S. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Data Bases Conference*, 487–499.

Leskovec, J., & Ghashami, M. (2022). *Frequent itemset mining & association rules.* Stanford University. http://web.stanford.edu/class/cs246/slides/02-assocrules.pdf

Leskovec, J., Rajaraman, A., & Ullman, J. (2019). *Mining of massive datasets* (3rd ed.). Cambridge University Press.

Zaki, M.J., & Meira, W. (2020). *Data mining and machine learning: Fundamental concepts and algorithms* (2nd ed.). Cambridge University Press.

-- return 0; --