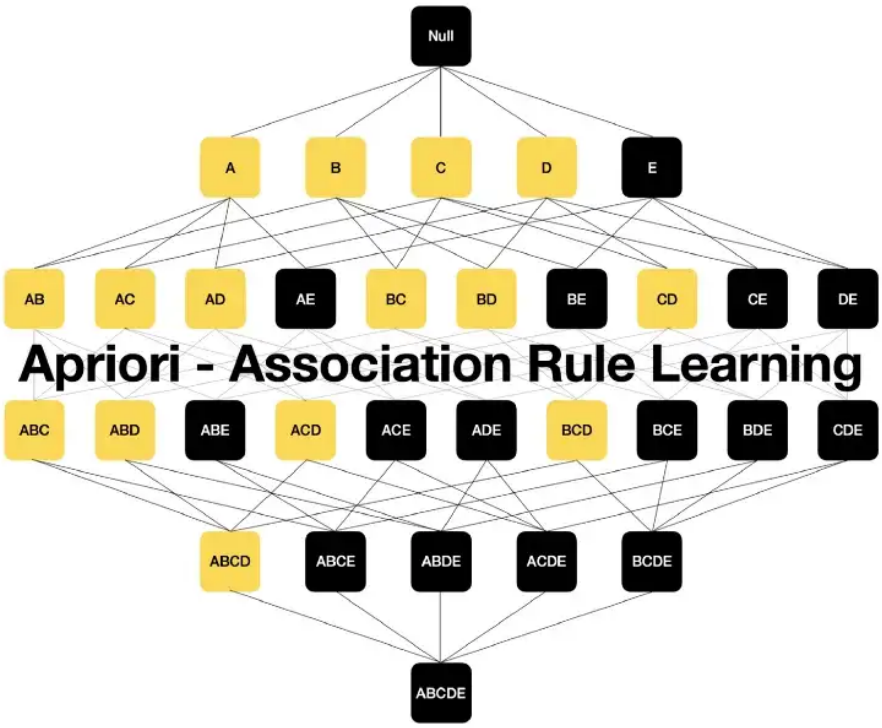
Apriori Algorithm for Association Rule Learning — How To Find Clear Links Between Transactions

Explanation and examples of frequent itemset mining and association rule learning over relational databases in Python



Apriori — Association Rule Learning. All possible itemsets formed from 5 items (A, B, C, D, E). Image by author.

Introduction

Most of you may already be familiar with clustering algorithms such as K-Means, HAC, or DBSCAN. However, clustering is not the only unsupervised way to find similarities between data points. You can also use association rule learning techniques to determine if certain data points (actions) are more likely to occur together.

A simple example would be the supermarket shopping basket analysis. If someone is buying ground beef, does it make them more likely to also buy spaghetti? We can answer these types of questions by using the Apriori algorithm.

Contents

- The category of algorithms Apriori belong to

- Introduction to Association Rule Learning and visual explanation of how Apriori algorithm works
- Python example of Apriori algorithm using real-life data
- Conclusions

What category of algorithms does Apriori belong to?

As stated earlier, Apriori is part of the association rule learning algorithms, which sit under the unsupervised branch of Machine Learning.

This is because Apriori does not require us to provide a target variable for the model. Instead, the algorithm identifies relationships between data points subject to our specified constraints.

The below graph is **interactive,** so please click on different categories to **enlarge and reveal more** 👇 **.**

Machine Learning algorithm classification. Interactive chart created by the author.

If you enjoy Data Science and Machine Learning, please subscribe to get an email whenever I publish a new story.

Association Rule Learning and Apriori algorithm

Association Rule Learning

As briefly mentioned in the introduction, association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. Let's use a simple supermarket shopping basket analysis to explain how the association rules are found.

	Item 1	Item 2	Item 3
Shopper 1	Eggs	Bacon	Soup
Shopper 2	Eggs	Bacon	Apple
Shopper 3	Eggs	Bacon	Apple
Shopper 4	Soup	Bacon	Banana
Shopper 5	Banana	Butter	-
Shopper 6	Butter	-	-

Supermarket purchase list by shoppers. Image by author.

Assume we analyze the above transaction data to find frequently bought items and determine if they are often purchased together. To help us find the answers, we will make use of the following 4 metrics:

- Support
- Confidence
- Lift
- Conviction

Support

The first step for us and the algorithm is to find frequently bought items. It is a

straightforward calculation that is based on frequency:

```
Support(A) = Transactions(A) / Total Transactions
```

So in our example:

```
Support(Eggs) = 3/6 = 1/2 = 0.5
Support(Bacon) = 4/6 = 2/3 = 0.667
Support(Apple) = 2/6 = 1/3 = 0.333
...
Support(Eggs&Bacon) = 3/6 = 0.5
...
```

Here we can set our first constraint by telling the algorithm the minimum support level we want to explore, which is useful when working with large datasets. We typically want to focus computing resources to search for associations between frequently bought items while discounting the infrequent ones.

For the sake of our example, let's **set minimum support to 0.5**, which leaves us to work with Eggs and Bacon for the rest of this example.

Important: while Support(Eggs) and Support(Bacon) individually satisfy our minimum support constraint, it is crucial to understand that we also need the combination of them (Eggs&Bacon) to pass this constraint. Otherwise, we would not have a single item pairing to progress forward to create association rules.

Confidence

Now that we have identified frequently bought items let's calculate confidence. This will tell us how confident (based on our data) we can be that an item will be purchased, given that another item has been purchased.

```
Confidence(A→B) = Probability(A & B) / Support(A)

Note, confidence is the same as what is also known as conditional
probability in statistics:
P(B|A) = P(A & B) / P(A)
```

Please beware of the notation. The above two equeations are equivalent, although the notations are in different order: ***(A→B)*** *is the same as* ***(B|A)*****.**

So, let's calculate confidence for our example:

Confidence(Eggs→Bacon) = P(Eggs & Bacon) / Support(Eggs) = (3/6) / (3/6) = 1

Confidence(Bacon→Eggs) = P(Eggs & Bacon) / Support(Bacon) = (3/6) / (2/3) = 3/4 = 0.75

The above tells us that whenever eggs are bought, bacon is also bought 100% of the time. Also, whenever bacon is bought, eggs are bought 75% of the time.

Lift

Given that different items are bought at different frequencies, how do we know that eggs and bacon really do have a strong association, and how do we measure it? You will be glad to hear that we have a way to evaluate this objectively using **lift.**

There are multiple ways to express the formula to calculate lift. Let me first show what the formulas look like, and then I will describe an intuitive way for you to think about it.

Lift(A→B) = Probability(A & B) / (Support(A) * Support(B))

You should be able to spot that we can simplify this formula by replacing P(A&B)/Sup(A) with Confidence(A→B). Hence, we have:

Lift(A→B) = Confidence(A→B) / Support(B)

Let's calculate lift for our associated items:

Lift(Eggs→Bacon) = Confidence(Eggs→Bacon) / Support(Bacon) = 1 / (2/3) = 1.5

Lift(Bacon→Eggs) = Confidence(Bacon→Eggs) / Support(Eggs) = (3/4) / (1/2) = 1.5

Lift for the two items is equal to 1.5. Note, lift>1 means that the two items are more likely to be bought together, while lift<1 means that the two items are more likely to be bought separately. Finally, lift=1 means that there is no association between the two items.

An intuitive way to understand this would be to first think about the probability of eggs being bought: `P(Eggs)=Support(Eggs)=0.5` as 3 out of 6 shoppers bought eggs.

Then think about the probability of eggs being bought whenever bacon was bought: `P(Eggs|Bacon)=Confidence(Bacon->Eggs)=0.75` since out of the 4 shoppers that bought bacon, 3 of them also bought eggs.

Now, lift is simply a measure that tells us whether the probability of buying eggs increases or decreases given the purchase of bacon. Since the probability of buying eggs in such a scenario goes up from 0.5 to 0.75, we see a positive lift of 1.5 times (0.75/0.5=1.5). This means you are 1.5 times (i.e., 50%) more likely to buy eggs if you have already put bacon into your basket.

See if your supermarket places these two items nearby. 😜

Conviction

Conviction is another way of measuring association, although it is a bit harder to get your head around. It compares the probability that A appears without B if they were independent with the actual frequency of the appearance of A without B. Let's take a look at the general formula first:

```
Conviction(A→B) = (1 - Support(B)) / (1 - Confidence(A→B))
```

In our example, this would be:

```
Conviction(Eggs→Bacon) = (1 - Sup(Bacon) / (1 - Conf(Eggs→Bacon)) = (1
- 2/3) / (1 - 1) = (1/3) / 0 = infinity

Conviction(Bacon→Eggs) = (1 - Sup(Eggs) / (1 - Conf(Bacon→Eggs)) = (1 -
1/2) / (1 - 3/4) = (1/2) / (1/4) = 2
```

As you can see, we had a division by 0 when calculating conviction for (Eggs→Bacon) and this is because we do not have a single instance of eggs being bought without bacon (confidence=100%).

In general, high confidence for A→B with low support for item B would yield a high conviction.

In contrast to lift, conviction is a directed measure. Hence, while lift is the same for both (Eggs→Bacon) and (Bacon→Eggs), conviction is different between the two, with Conv(Eggs→Bacon) being much higher. Thus, you can use conviction to evaluate the directional relationship between your items.

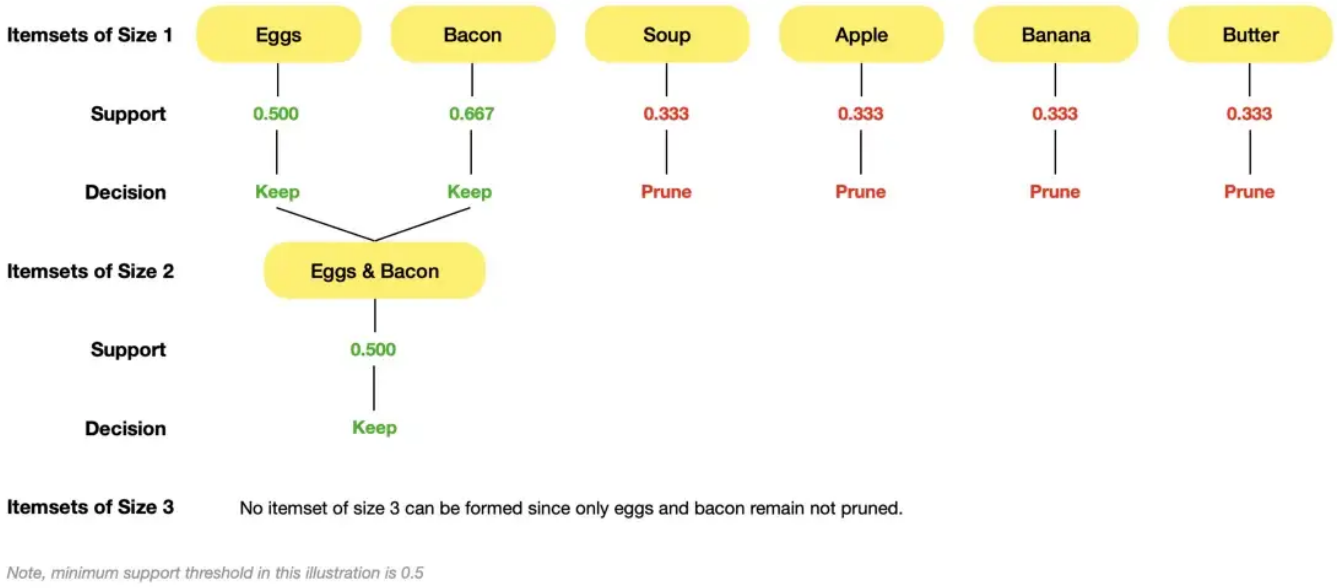
Finally, similar to lift, conviction=1 means that items are not associated, while conviction>1 indicates the relationship between the items (the higher the value, the stronger the relationship).

Apriori algorithm

Apriori is a pretty straightforward algorithm that performs the following sequence of calculations:

1. Calculate support for itemsets of size 1.
2. Apply the minimum support threshold and prune itemsets that do not meet the threshold.
3. Move on to itemsets of size 2 and repeat steps one and two.
4. Continue the same process until no additional itemsets satisfying the minimum threshold can be found.

To make the process more visual, here is a diagram that illustrates what the algorithm does:



Process tree of Apriori algorithm. Image by author.

As you can see, most itemsets in this example got pruned since they did not meet the minimum support threshold of 0.5.

It is important to realize that by setting a lower minimum support threshold we would produce many more itemsets of size 2. To be precise, with a minimum support threshold of 0.3, none of the itemsets of size 1 would get pruned giving us a total of 15 itemsets of size 2 (5+4+3+2+1=15).

This is not an issue when we have a small dataset, but it could become a bottleneck if you are working with a large dataset. E.g., 1,000 items can create as many as 499,500 item pairs. Hence, choose your minimum support threshold carefully.

Note, if you want more examples you can refer to the cover image that shows all possible itemsets that can be formed from just 5 individual items.

Python example of Apriori algorithm using real-life data

Let's now put theory behind us and run the analysis on real-life data in Python.

Setup

We will use the following data and libraries:

- Market Basket Optimisation data from Kaggle
- Apriori algorithm for association rule learning
- Pandas for data manipulation
- Matplotlib for drawing frequency distribution graph

Let's import all the libraries:

Then we download the Market_Basket_Optimisation.csv from Kaggle and ingest the data:

This is what the data looks like:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	fro: smoot
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

7501 rows × 20 columns

A snippet of the supermarket basket data. Image by author.

Exploration

Before we run the association rule analysis, let's first look at the items' frequency distribution.

	Item	Count	Percentage
0	asparagus	1	0.000034
112	water spray	3	0.000102
77	napkins	5	0.000170
34	cream	7	0.000238
11	bramble	14	0.000477
...	...	...	...
25	chocolate	1230	0.041889
43	french fries	1282	0.043660
100	spaghetti	1306	0.044478
37	eggs	1348	0.045908
72	mineral water	1788	0.060893

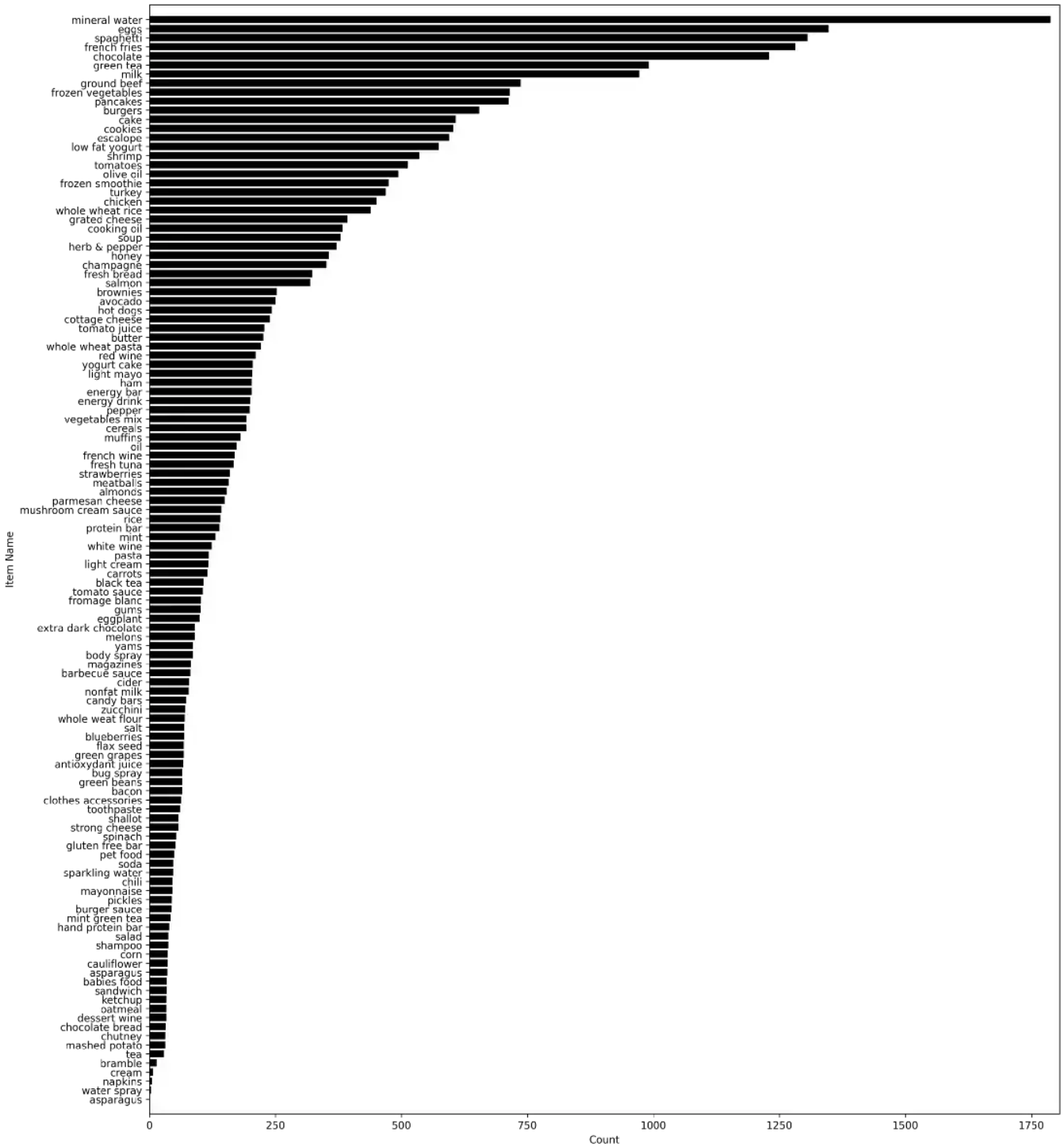
120 rows × 3 columns

A snippet of the frequency count of purchased supermarket items. Image by author.

We can see that mineral water (1,788) is by far the most popular item bought at this supermarket, followed by eggs (1,348), spaghetti (1,306), french fries (1,282), and chocolate (1,230). Meanwhile, poor old asparagus has only been bought once.

Also, it is important to note that even the most frequently bought item only appears in just over 6% of transactions. We can use this information as a guide when setting the minimum support threshold.

Let's also display frequency distribution in the bar chart.


mineral water
eggs
spaghetti
french fries
chocolate
green tea
milk
ground beef
frozen vegetables
pancakes
burgers
cake
cookies
escalope
low fat yogurt
shrimp
tomatoes
olive oil
frozen smoothie
turkey
chicken
whole wheat rice
grated cheese
cooking oil
soup
herb & pepper
honey
champagne
fresh bread
salmon
brownies
avocado
hot dogs
cottage cheese
tomato juice
butter
whole wheat pasta
red wine
yogurt cake
light mayo
ham
energy bar
energy drink
pepper
vegetables mix
cereals
muffins
oil
french wine
fresh tuna
strawberries
meatballs
almonds
parmesan cheese
mushroom cream sauce
rice
protein bar
mint
white wine
pasta
light cream
carrots
black tea
tomato sauce
fromage blanc
gums
eggplant
extra dark chocolate
melons
yams
body spray
magazines
barbecue sauce
cider
nonfat milk
candy bars
zucchini
whole weat flour
salt
blueberries
flax seed
green grapes
antioxydant juice
bug spray
green beans
bacon
clothes accessories
toothpaste
shallot
strong cheese
spinach
gluten free bar
pet food
soda
sparkling water
chili
mayonnaise
pickles
burger sauce
mint green tea
hand protein bar
salad
shampoo
corn
cauliflower
asparagus
babies food
sandwich
ketchup
oatmeal
dessert wine
chocolate bread
chutney
mashed potato
tea
bramble
cream
napkins
water spray
asparagus
Item Name
0
250
500
750
1000
1250
1500
1750
Count

Frequency distribution of purchased supermarket items. Image by author.

Running Apriori algorithm

Before we run the algorithm, let's put our data in the required format. It will be a **list of lists** that has all those "NaNs" you saw in Pandas DataFrame removed.

```
[['shrimp',
  'almonds',
  'avocado',
  'vegetables mix',
  'green grapes',
  'whole weat flour',
  'yams',
  'cottage cheese',
  'energy drink',
  'tomato juice',
  'low fat yogurt',
  'green tea',
  'honey',
  'salad',
  'mineral water',
  'salmon',
  'antioxydant juice',
  'frozen smoothie',
  'spinach',
  'olive oil'],
 ['burgers', 'meatballs', 'eggs'],
 ['chutney'],
 ['turkey', 'avocado'],
 ['mineral water', 'milk', 'energy bar', 'whole wheat rice', 'green tea'],
 ['low fat yogurt'],
 ['whole wheat pasta', 'french fries'],
```

A snippet of the transactions list of lists. Image by author.

Finally, let's run the Apriori algorithm and save itemsets and association rules.

The algorithm has found 36 itemsets of size 1 and 18 itemsets of size 2, which satisfy the minimum support threshold of 0.03. These are displayed below.

In [81]:
```
1 itemsets
```

Out[81]:
```
{1: {('mineral water',): 1788,
  ('low fat yogurt',): 574,
  ('green tea',): 991,
  ('avocado',): 250,
  ('frozen smoothie',): 475,
  ('tomato juice',): 228,
  ('shrimp',): 536,
  ('honey',): 356,
  ('salmon',): 319,
  ('olive oil',): 494,
  ('cottage cheese',): 239,
  ('burgers',): 654,
  ('eggs',): 1348,
  ('turkey',): 469,
  ('whole wheat rice',): 439,
  ('milk',): 972,
  ('french fries',): 1282,
  ('soup',): 379,
  ('frozen vegetables',): 715,
  ('spaghetti',): 1306,
  ('cookies',): 603,
  ('cooking oil',): 383,
  ('champagne',): 351,
  ('chicken',): 450,
  ('chocolate',): 1229,
  ('tomatoes',): 513,
  ('pancakes',): 713,
  ('grated cheese',): 393,
  ('fresh bread',): 323,
  ('ground beef',): 737,
  ('escalope',): 595,
  ('herb & pepper',): 371,
  ('cake',): 608,
  ('hot dogs',): 243,
  ('brownies',): 253,
  ('butter',): 226},
 2: {('green tea', 'mineral water'): 233,
  ('milk', 'mineral water'): 360,
  ('eggs', 'mineral water'): 382,
```

```
('eggs', 'spaghetti'): 274,
('mineral water', 'spaghetti'): 448,
('chocolate', 'eggs'): 249,
('mineral water', 'pancakes'): 253,
('milk', 'spaghetti'): 266,
('ground beef', 'mineral water'): 307,
('ground beef', 'spaghetti'): 294,
('chocolate', 'french fries'): 258,
('chocolate', 'mineral water'): 395,
('eggs', 'french fries'): 273,
('french fries', 'mineral water'): 253,
('frozen vegetables', 'mineral water'): 268,
('chocolate', 'spaghetti'): 294,
('chocolate', 'milk'): 241,
('eggs', 'milk'): 231}}
```

All itemsets satisfying minimum support threshold constraint. Image by author.

Let's now print the association rules found by the algorithm. Note, a few sets get excluded in the rule generation stage because they do not meet our specified minimum confidence requirement (in this example, min_confidence=0.2).

The above code prints all the association rules satisfying our constraints ordered by the highest lift and conviction:

```
{ground beef} -> {spaghetti} (conf: 0.399, supp: 0.039, lift: 2.291, conv: 1.374)
{spaghetti} -> {ground beef} (conf: 0.225, supp: 0.039, lift: 2.291, conv: 1.164)
{ground beef} -> {mineral water} (conf: 0.417, supp: 0.041, lift: 1.748, conv: 1.305)
{frozen vegetables} -> {mineral water} (conf: 0.375, supp: 0.036, lift: 1.572, conv: 1.21
{milk} -> {spaghetti} (conf: 0.274, supp: 0.035, lift: 1.572, conv: 1.137)
{spaghetti} -> {milk} (conf: 0.204, supp: 0.035, lift: 1.572, conv: 1.093)
{milk} -> {mineral water} (conf: 0.370, supp: 0.048, lift: 1.554, conv: 1.210)
{mineral water} -> {milk} (conf: 0.201, supp: 0.048, lift: 1.554, conv: 1.090)
{milk} -> {chocolate} (conf: 0.248, supp: 0.032, lift: 1.513, conv: 1.112)
{pancakes} -> {mineral water} (conf: 0.355, supp: 0.034, lift: 1.489, conv: 1.181)
{spaghetti} -> {mineral water} (conf: 0.343, supp: 0.060, lift: 1.439, conv: 1.159)
{mineral water} -> {spaghetti} (conf: 0.251, supp: 0.060, lift: 1.439, conv: 1.102)
{chocolate} -> {spaghetti} (conf: 0.239, supp: 0.039, lift: 1.374, conv: 1.086)
{spaghetti} -> {chocolate} (conf: 0.225, supp: 0.039, lift: 1.374, conv: 1.079)
{chocolate} -> {mineral water} (conf: 0.321, supp: 0.053, lift: 1.348, conv: 1.122)
{mineral water} -> {chocolate} (conf: 0.221, supp: 0.053, lift: 1.348, conv: 1.073)
{milk} -> {eggs} (conf: 0.238, supp: 0.031, lift: 1.322, conv: 1.076)
{chocolate} -> {french fries} (conf: 0.210, supp: 0.034, lift: 1.228, conv: 1.049)
{french fries} -> {chocolate} (conf: 0.201, supp: 0.034, lift: 1.228, conv: 1.047)
{eggs} -> {mineral water} (conf: 0.283, supp: 0.051, lift: 1.189, conv: 1.063)
{mineral water} -> {eggs} (conf: 0.214, supp: 0.051, lift: 1.189, conv: 1.043)
{french fries} -> {eggs} (conf: 0.213, supp: 0.036, lift: 1.185, conv: 1.042)
{eggs} -> {french fries} (conf: 0.203, supp: 0.036, lift: 1.185, conv: 1.040)
{spaghetti} -> {eggs} (conf: 0.210, supp: 0.037, lift: 1.167, conv: 1.038)
{eggs} -> {spaghetti} (conf: 0.203, supp: 0.037, lift: 1.167, conv: 1.037)
{chocolate} -> {eggs} (conf: 0.203, supp: 0.033, lift: 1.127, conv: 1.029)
{green tea} -> {mineral water} (conf: 0.235, supp: 0.031, lift: 0.986, conv: 0.996)
```

27 association rules produced by the Apriori algorithm based on our constraints. Image by author.

As we can see, the highest lift in this real-life data is for a combination of ground beef and spaghetti. However, it is more likely for the purchaser of ground beef to also buy spaghetti (confidence: 0.399, conviction: 1.374) than the other way round (confidence: 0.225, conviction: 1.164).

All in all, it looks like the visitors of this supermarket are big fans of spaghetti bolognese an mineral water.

Conclusions

Apriori is a straightforward algorithm that quickly learns association rules between items (data points). While I have taken you through its use for market basket analysis, there are al many other practical applications, including bioinformatics (protein sequencing), medical diagnosis (relationship between symptoms and disease), or Census data analysis.

One thing to be careful about when using Apriori on large datasets is the choice of minimur support threshold. If you are not careful, you can quickly run out of memory with a potentially huge number of itemsets of size 2.

I hope you found Apriori and association rule learning just as exciting as I did. Thanks for reading, and feel free to reach out if you have any questions or suggestions!