**Market Basket Analysis in Python**



Amazon, Netflix and many other popular companies rely on Market Basket Analysis to produce meaningful product recommendations. Market Basket Analysis is a powerful tool for translating vast amounts of customer transaction and viewing data into simple rules for product promotion and recommendation. In this notebook, we’ll learn how to perform Market Basket Analysis using the Apriori algorithm, standard and custom metrics, association rules, aggregation and pruning, and visualization.

**What is market basket analysis?**

1. Identify products frequently purchased together.

* Bookstore Ex:
  + Biography and history
  + Fiction and poetry

1. Construct recommendations based on these

* Bookstore Ex:
  + Place biography and history sections together.
  + Keep fiction and history apart

**The use cases of market basket analysis**

1. Build Netflix-style recommendations engine.
2. Improve product recommendations on an e-commerce store.
3. Cross-sell products in a retail setting.
4. Improve inventory management.
5. Upsell products.

* **Market basket analysis**
  + Construct association rules
  + Identify items frequently purchased together
* **Association rules**
  + {antecedent}→{consequent}
    - {fiction}→{biography}

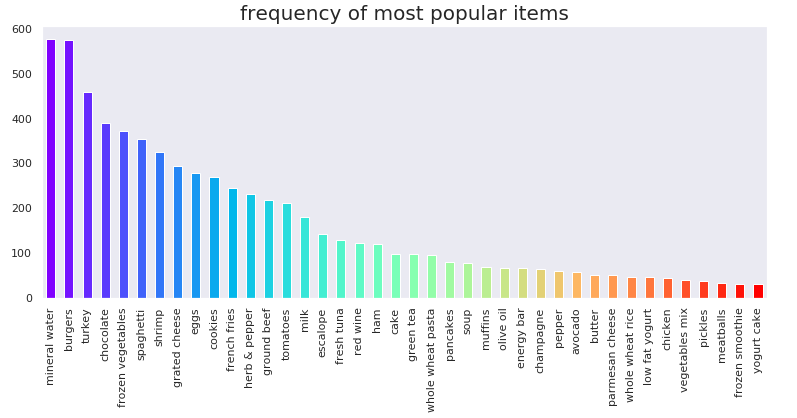
**Dataset**

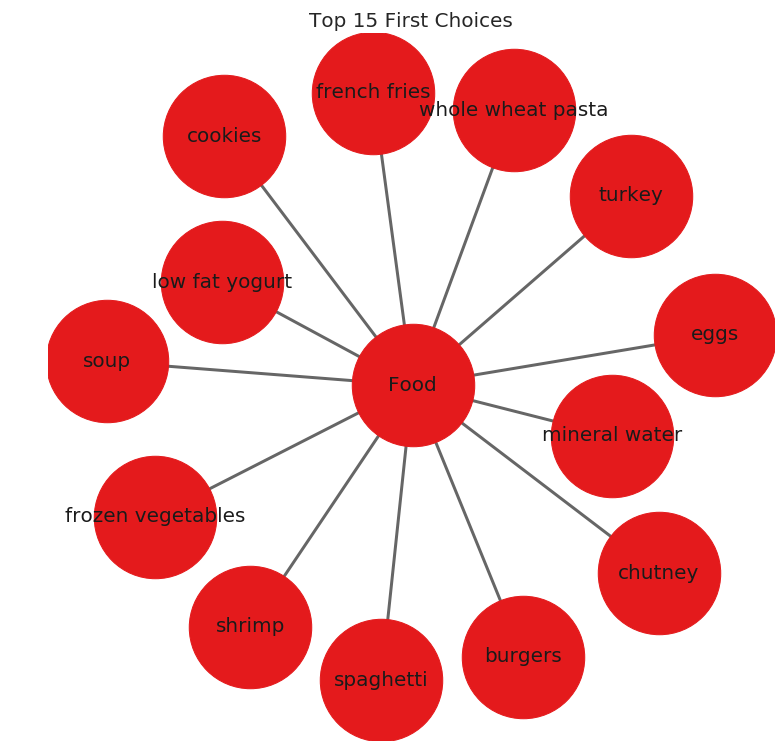
The contains information about customers buying different grocery items.

In [3]:

data = pd.read\_csv('Market\_Basket.csv', header = **None**)

**EDA**





**Getting the list of transactions**

Once we have read the dataset, we need to get the list of items in each transaction. SO we will run two loops here. One for the total number of transactions, and other for the total number of columns in each transaction. This list will work as a training set from where we can generate the list of association rules.

In [9]:

*# Getting the list of transactions from the dataset*

transactions = []

**for** i **in** range(0, len(data)):

transactions.append([str(data.values[i,j]) **for** j **in** range(0, len(data.columns))])

In [10]:

transactions[:1]

Out[10]:

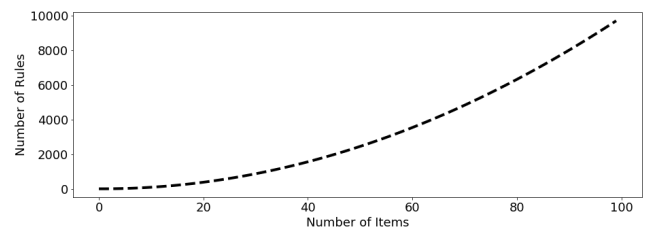
**Association rules**

* **Association rule**
  + Contains antecedent and consequent
    - {health} → {cooking}
* **Multi-antecedent rule**
  + {humor, travel} → {language}
* **Multi-consequent rule**
  + {biography} → {history, language}
* **Multi-antecedent and consequent rule**
  + {biography, non-fiction} → {history, language}

**Difficulty of selecting rules**

* Finding useful rules is difficult.
  + Set of all possible rules is large.
  + Most rules are not useful.
  + Must discard most rules.
* What if we restrict ourselves to simple rules?
  + One antecedent and one consequent.
  + Still challenging, even for small dataset.

**As the number of items increase the number of rules increases exponentially.**



In [11]:

**from** **itertools** **import** permutations

*# Extract unique items.*

flattened = [item **for** transaction **in** transactions **for** item **in** transaction]

items = list(set(flattened))

In [12]:

**One-hot encoding transaction data**

Throughout we will use a common pipeline for preprocessing data for use in market basket analysis. The first step is to import a pandas DataFrame and select the column that contains transactions. Each transaction in the column will be a string that consists of a number of items, each separated by a comma. The next step is to use a lambda function to split each transaction string into a list, thereby transforming the column into a list of lists. Then we will transform the transactions into a one-hot encoded DataFrame, where each column consists of TRUE and FALSE values that indicate whether an item was included in a transaction.

**Metrics and pruning**

* A **metric** is a measure of performance for rules.
  + {humor} → {poetry}
    - 0.81
  + {fiction} → {travel}
    - 0.23
* **Pruning** is the use of metrics to discard rules.
  + Retain: {humor} → {poetry}
  + Discard: { ction} → {travel}

**The simplest metric**

* The **support** metric measures the share of transactions that contain an itemset.



|  | **support** |
| --- | --- |
| **Food** | 1.000000 |
| **mineral water** | 0.238368 |
| **eggs** | 0.179709 |
| **spaghetti** | 0.174110 |
| **french fries** | 0.170911 |

**Confidence and lift**

**When support is misleading**

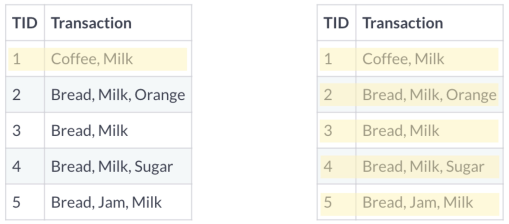
1. **Milk and bread frequently purchased together.**
   * Support: {Milk} → {Bread}
2. **Rule is not informative for marketing.**
   * Milk and bread are both *independently* popular items.

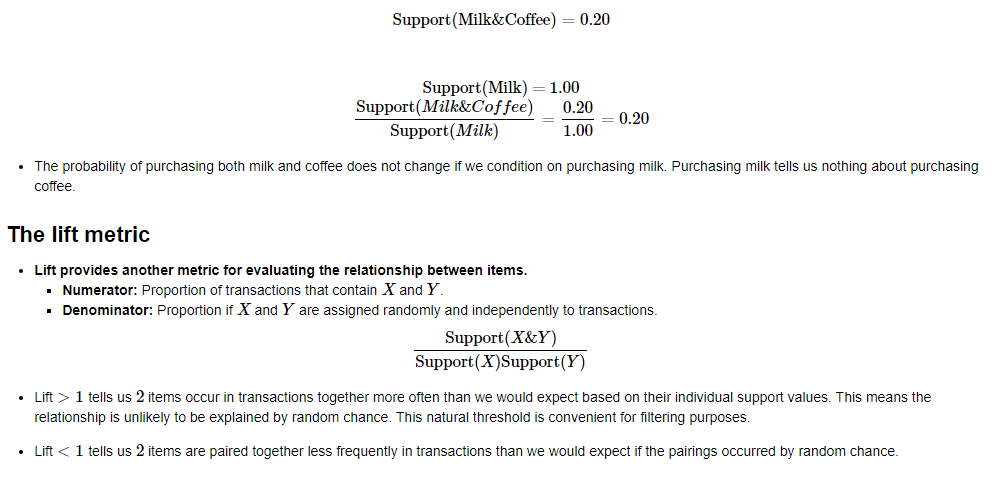
**The confidence metric**

1. Can improve over **support** with additional metrics.
2. Adding **confidence** provides a more complete picture.
3. **Confidence** gives us the *probability* we will purchase Y given we have purchased X.



**Interpreting the confidence metric**





**Recommending food with support**

A grocery-store wants to get members to eat more and has decided to use market basket analysis to figure out how. They approach you to do the analysis and ask that you use the five most highly-rated food items.

*# Compute support for burgers and french fries*

supportBF = np.logical\_and(onehot['burgers'], onehot['french fries']).mean()

*# Compute support for burgers and mineral water*

supportBM = np.logical\_and(onehot['burgers'], onehot['mineral water']).mean()

*# Compute support for french fries and mineral water*

supportFM = np.logical\_and(onehot['french fries'], onehot['mineral water']).mean()

*# Print support values*

print("burgers and french fries: **%.2f**" % supportBF)

print("burgers and mineral water: **%.2f**" % supportBM)

print("french fries and mineral water: **%.2f**" % supportFM)

burgers and french fries: 0.02

burgers and mineral water: 0.02

french fries and mineral water: 0.03

**Computing the support metric**

Previously we one-hot encoded a small grocery store's transactions as the DataFrame onehot. In this exercise, we'll make use of that DataFrame and the support metric to help the store's owner. First, she has asked us to identify frequently purchased items, which we'll do by computing support at the item-level. And second, she asked us to check whether the rule {mineral water} → {french fries} has a support of over 0.050.05.

mineral water+french fries support = 0.03372883615517931

**Refining support with confidence**

After reporting your findings from the previous exercise, the store's owner asks us about the direction of the relationship. Should they use mineral water to promote french fries or french fries to promote mineral water?

We decide to compute the confidence metric, which has a direction, unlike support. We'll compute it for both {mineral water} → {french fries} and {french fries} → {mineral water}.

*# Compute support for mineral water and french fries*

supportMF = np.logical\_and(onehot['mineral water'], onehot['french fries']).mean()

*# Compute support for mineral water*

supportM = onehot['mineral water'].mean()

*# Compute support for french fries*

supportF = onehot['french fries'].mean()

*# Compute confidence for both rules*

confidenceMM = supportMF / supportM

confidenceMF = supportMF / supportF

*# Print results*

print('mineral water = **{0:.2f}**, french fries = **{1:.2f}**'.format(confidenceMM, confidenceMF))

mineral water = 0.14, french fries = 0.20

Even though the support is identical for the two association rules, the confidence is much higher for french fries -> mineral water, since french fries has a higher support than mineral water.

**Further refinement with lift**

Once again, we report our results to the store's owner: Use french fries to promote mineral water, since the rule has a higher confidence metric. The store's owner thanks us for the suggestion, but asks us to confirm that this is a meaningful relationship using another metric.

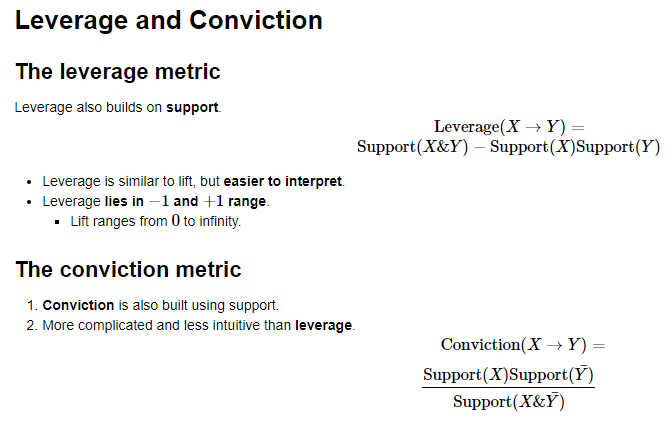
You recall that lift may be useful here. If lift is less than 11, this means that mineral water and french fries are paired together less frequently than we would expect if the pairings occurred by random chance.

lift = supportMF / (supportM \* supportF)

print("Lift: **%.2f**" % lift)

Lift: 0.83

As it turns out, lift is less than 1.01.0. This does not give us good confidence that the association rule we recommended did not arise by random chance.



**Computing conviction**

The store's owner asks us if we are able to compute conviction for the rule {burgers} → {french fries}, so she can decide whether to place the items next to each other on the company's website.

In [22]:

*# Compute support for burgers AND french fries*

supportBF = np.logical\_and(onehot['burgers'], onehot['french fries']).mean()

*# Compute support for burgers*

supportB = onehot['burgers'].mean()

*# Compute support for NOT french fries*

supportnF = 1.0 - onehot['french fries'].mean()

*# Compute support for burgers and NOT french fries*

supportBnF = supportB - supportBF

*# Compute and print conviction for burgers -> french fries*

conviction = supportB \* supportnF / supportBnF

print("Conviction: **%.2f**" % conviction)

Conviction: 1.11

Notice that the value of conviction was greater than 11, suggesting that the rule if burgers then french fries is supported.

**Computing conviction with a function**

The store's owner asks us if we are able to compute conviction for every pair of food items in the grocery-store dataset, so she can use that information to decide which food items to locate closer together on the website.

We agree to take the job, but realize that we a need more efficient way to compute conviction, since we will need to compute it many times. We decide to write a function that computes it. It will take two columns of a pandas DataFrame as an input, one antecedent and one consequent, and output the conviction metric.

In [23]:

**def** conviction(antecedent, consequent):

*# Compute support for antecedent AND consequent*

supportAC = np.logical\_and(antecedent, consequent).mean()

*# Compute support for antecedent*

supportA = antecedent.mean()

*# Compute support for NOT consequent*

supportnC = 1.0 - consequent.mean()

*# Compute support for antecedent and NOT consequent*

supportAnC = supportA - supportAC

*# Return conviction*

**return** supportA \* supportnC / supportAnC

**Computing leverage with a function**

In [24]:

**def** leverage(antecedent, consequent):

*# Compute support for antecedent AND consequent*

supportAB = np.logical\_and(antecedent, consequent).mean()

*# Compute support for antecedent*

supportA = antecedent.mean()

*# Compute support for consequent*

supportB = consequent.mean()

*# Return leverage*

**return** supportAB - supportB \* supportA

**Promoting food with conviction**

Previously we defined a function to compute conviction. We were asked to apply that function to all two-food items permutations of the grocery-store dataset. We'll test the function by applying it to the three most popular food items, which we used in earlier exercises: burgers, french fries, and mineral water.

In [25]:

*# Compute conviction for burgers -> french fries and french fries -> burgers*

convictionBF = conviction(onehot['burgers'], onehot['french fries'])

convictionFB = conviction(onehot['french fries'], onehot['burgers'])

*# Compute conviction for burgers -> mineral water and mineral water -> burgers*

convictionBM = conviction(onehot['burgers'], onehot['mineral water'])

convictionMB = conviction(onehot['mineral water'], onehot['burgers'])

*# Compute conviction for french fries -> mineral water and mineral water -> french fries*

convictionFM = conviction(onehot['french fries'], onehot['mineral water'])

convictionMF = conviction(onehot['mineral water'], onehot['french fries'])

*# Print results*

print('french fries -> burgers: ', convictionFB)

print('burgers -> french fries: ', convictionBF)

french fries -> burgers: 1.0476495106531305

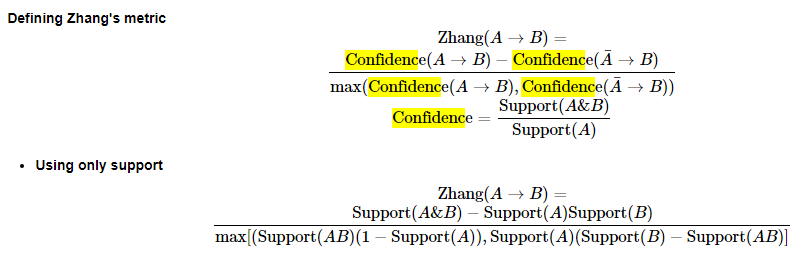
burgers -> french fries: 1.1088435652342468

**Association and Dissociation**



**Zhang's metric**

1. Introduced by Zhang(2000)
   * Takes values between −1−1 and +1+1
   * Value of +1+1 indicates perfect association
   * Value of −1−1 indicates perfect dissociation
2. **Comprehensive and interpretable**
3. **Constructed using support**



**Computing association and dissociation**

The store's owner has returned to you once again about your recommendation to promote french fries using burgers. They're worried that the two might be dissociated, which could have a negative impact on their promotional effort. They ask you to verify that this is not the case.

You immediately think of Zhang's metric, which measures association and dissociation continuously. Association is positive and dissociation is negative.

In [26]:

*# Compute the support of burgers and french fries*

supportT = onehot['burgers'].mean()

supportP = onehot['french fries'].mean()

*# Compute the support of both food items*

supportTP = np.logical\_and(onehot['burgers'], onehot['french fries']).mean()

*# Complete the expressions for the numerator and denominator*

numerator = supportTP - supportT\*supportP

denominator = max(supportTP\*(1-supportT), supportT\*(supportP-supportTP))

*# Compute and print Zhang's metric*

zhang = numerator / denominator

print(zhang)

0.3533836982354581

Once again, the association rule if burgers then french fries proved robust. It had a positive value for Zhang's metric, indicating that the two food items are not dissociated.

**Defining Zhang's metric**

In general, when we want to perform a task many times, we'll write a function, rather than coding up each individual instance. In this exercise, we'll define a function for Zhang's metric that takes an antecedent and consequent and outputs the metric itself.

In [27]:

*# Define a function to compute Zhang's metric*

**def** zhang(antecedent, consequent):

*# Compute the support of each book*

supportA = antecedent.mean()

supportC = consequent.mean()

*# Compute the support of both books*

supportAC = np.logical\_and(antecedent, consequent).mean()

*# Complete the expressions for the numerator and denominator*

numerator = supportAC - supportA\*supportC

denominator = max(supportAC\*(1-supportA), supportA\*(supportC-supportAC))

*# Return Zhang's metric*

**return** numerator / denominator

**Applying Zhang's metric**

The store's owner has sent you a list of itemsets she's investigating and has asked us to determine whether any of them contain items that are dissociated. When we're finished, she has asked that us to add the metric we use to a column in the rules DataFrame.

|  | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** | **zhang** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **542** | burgers | asparagus | 0.087188 | 0.000133 | 0.000133 | 0.001529 | 11.469419 | 0.000122 | 1.001398 | 1.0 |
| **13503** | ground beef | asparagus | 0.098254 | 0.000133 | 0.000133 | 0.001357 | 10.177748 | 0.000120 | 1.001225 | 1.0 |
| **5102** | energy bar | asparagus | 0.027063 | 0.000133 | 0.000133 | 0.004926 | 36.950739 | 0.000130 | 1.004817 | 1.0 |
| **1142** | shrimp | asparagus | 0.071457 | 0.000133 | 0.000133 | 0.001866 | 13.994403 | 0.000124 | 1.001736 | 1.0 |
| **5822** | soup | asparagus | 0.050527 | 0.000133 | 0.000133 | 0.002639 | 19.791557 | 0.000127 | 1.002512 | 1.0 |

|  | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** | **zhang** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 14520.000000 | 14520.000000 | 14520.000000 | 14520.000000 | 14520.000000 | 14520.000000 | 1.440000e+04 | 14400.000000 |
| **mean** | 0.040611 | 0.040611 | 0.001906 | 0.052663 | 1.467719 | 0.000335 | inf | -0.011728 |
| **std** | 0.097141 | 0.097141 | 0.007505 | 0.108745 | 1.864950 | 0.001148 | NaN | 0.621009 |
| **min** | 0.000133 | 0.000133 | 0.000000 | 0.000000 | 0.000000 | -0.011697 | 7.616318e-01 | -1.000000 |
| **25%** | 0.007732 | 0.007732 | 0.000133 | 0.004975 | 0.500009 | -0.000046 | 9.953340e-01 | -0.517778 |
| **50%** | 0.015731 | 0.015731 | 0.000400 | 0.021849 | 1.214494 | 0.000079 | 1.003948e+00 | 0.192710 |
| **75%** | 0.042528 | 0.042528 | 0.001333 | 0.058140 | 1.858384 | 0.000361 | 1.020828e+00 | 0.483074 |
| **max** | 1.000000 | 1.000000 | 0.238368 | 1.000000 | 45.460606 | 0.022088 | inf | 1.000000 |

In [30]:

Notice that most of the items were dissociated, which suggests that they would have been a poor choice to pair together for promotional purposes.

**Overview of market basket analysis**

**Standard procedure for market basket analysis.**

1. Generate large set of rules.
2. Filter rules using metrics.
3. Apply intuition and common sense.

**Filtering with support and conviction**

The store's owner has approached you with the DataFrame rules, which contains the work of a data scientist who was previously on staff. It includes columns for antecedents and consequents, along with the performance for each of those rules with respect to a number of metrics.

Our objective will be to perform multi-metric filtering on the dataset to identify potentially useful rules.

In [31]:

*# Select the subset of rules with antecedent support greater than 0.05*

rules\_filtered = rules\_[rules\_['antecedent support'] > 0.05]

*# Select the subset of rules with a consequent support greater than 0.01*

rules\_filtered = rules\_[rules\_['consequent support'] > 0.01]

*# Select the subset of rules with a conviction greater than 1.01*

rules\_filtered = rules\_[rules\_['conviction'] > 1.01]

*# Select the subset of rules with a lift greater than 1.0*

rules\_filtered = rules\_[rules\_['lift'] > 1.0]

*# Print remaining rules*

print(f'# of rules = {len(rules\_)}')

print(f'# of rules after filtering = {len(rules\_filtered)}')

print(rules\_filtered.head())

# of rules = 14520

# of rules after filtering = 8598

antecedents consequents antecedent support consequent support support \

0 ham champagne 0.02653 0.046794 0.001333

1 ham red wine 0.02653 0.028130 0.001866

2 ham asparagus 0.02653 0.004666 0.000133

3 ham burgers 0.02653 0.087188 0.005599

4 ham protein bar 0.02653 0.018531 0.000933

confidence lift leverage conviction zhang

0 0.050251 1.073888 0.000092 1.003640 0.070679

1 0.070352 2.500988 0.001120 1.045417 0.616514

2 0.005025 1.076956 0.000010 1.000361 0.073405

3 0.211055 2.420681 0.003286 1.157003 0.602888

4 0.035176 1.898232 0.000442 1.017252 0.486090

**Using multi-metric filtering to cross-promote food items**

As a final request, the store's owner asks us to perform additional filtering. Our previous attempt returned 85988598 rules, but she wanted much less.

In [32]:

*# Set the threshold for Zhang's rule to 0.65*

rules\_filtered = rules\_filtered[rules\_filtered['zhang'] > 0.65]

*# Print rule*

print(f'# of rules after filtering = {8598 - len(rules\_filtered)}')

print(rules\_filtered.head())

# of rules after filtering = 6911

antecedents consequents antecedent support consequent support \

23 ham bramble 0.02653 0.001866

38 ham whole wheat rice 0.02653 0.058526

59 ham carrots 0.02653 0.015331

74 ham light cream 0.02653 0.015598

78 ham mint green tea 0.02653 0.005599

support confidence lift leverage conviction zhang

23 0.000267 0.010050 5.384781 0.000217 1.008267 0.836483

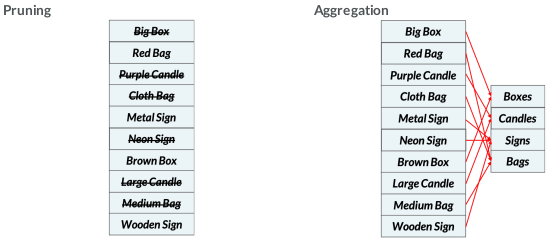
38 0.004266 0.160804 2.747588 0.002713 1.121877 0.653378

59 0.001600 0.060302 3.933231 0.001193 1.047856 0.766080

74 0.001200 0.045226 2.899497 0.000786 1.031032 0.672966

78 0.000533 0.020101 3.589854 0.000385 1.014799 0.741098

**Aggregation**



**Novelty gift data**

gifts = pd.read\_csv(url)

gifts.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 227760 entries, 0 to 227759

Data columns (total 3 columns):

InvoiceNo 227760 non-null object

StockCode 227760 non-null object

Description 227404 non-null object

dtypes: object(3)

memory usage: 5.2+ MB

In [34]:

gifts.head()

Out[34]:

|  | **InvoiceNo** | **StockCode** | **Description** |
| --- | --- | --- | --- |
| **0** | 562583 | 35637A | IVORY STRING CURTAIN WITH POLE |
| **1** | 562583 | 35638A | PINK AND BLACK STRING CURTAIN |
| **2** | 562583 | 84927F | PSYCHEDELIC TILE HOOK |
| **3** | 562583 | 22425 | ENAMEL COLANDER CREAM |
| **4** | 562583 | 16008 | SMALL FOLDING SCISSOR(POINTED EDGE) |

In [35]:

*# Stripping extra spaces in the description*

gifts['Description'] = gifts['Description'].str.strip()

In [36]:

*# Dropping the rows without any invoice number*

gifts.dropna(subset =['InvoiceNo'], inplace = **True**)

gifts['InvoiceNo'] = gifts['InvoiceNo'].astype('str')

In [37]:

*# Dropping all transactions which were done on credit*

gifts = gifts[~gifts['InvoiceNo'].str.contains('C')]

**EDA**

In [38]:

*# Print number of transactions.*

print(len(gifts['InvoiceNo'].unique()))

8410

In [39]:

*# Print number of items.*

print(len(gifts['Description'].unique()))

3447

In [40]:

*# Recover unique InvoiceNo's.*

InvoiceNo = gifts['InvoiceNo'].unique()

*# Create basket of items for each transaction.*

Transactions = [list(gifts[gifts['InvoiceNo'] == u].Description.astype(str)) **for** u **in** InvoiceNo]

In [41]:

*# Print example transaction.*

Transactions[0]

Out[41]:

['IVORY STRING CURTAIN WITH POLE',

'PINK AND BLACK STRING CURTAIN',

'PSYCHEDELIC TILE HOOK',

'ENAMEL COLANDER CREAM',

'SMALL FOLDING SCISSOR(POINTED EDGE)',

'JIGSAW TOADSTOOLS 3 PIECE']

In [42]:

*# Instantiate transaction encoder.*

encoder = TransactionEncoder()

*# One-hot encode transactions.*

onehot = encoder.fit(Transactions).transform(Transactions)

*# Use unique items as column headers.*

onehot = pd.DataFrame(onehot, columns = encoder.columns\_).drop('nan', axis=1)

*# Print onehot header.*

onehot.head()

**Performing aggregation**

After completing minor consulting jobs we've finally received our first big market basket analysis project: advising an online novelty gifts retailer on cross-promotions. Since the retailer has never previously hired a data scientist, they would like you to start the project by exploring its transaction data. They have asked us to perform aggregation for all signs in the dataset and also compute the support for this category.

In [43]:

*# Convert words to a list of words*

**def** convert\_str(string):

lst = list(string.split(' '))

**return** lst

In [44]:

*# Select the column headers for sign items*

sign\_headers = []

**for** i **in** onehot.columns:

wrd\_lst = convert\_str(str(i).lower())

**if** 'sign' **in** wrd\_lst:

sign\_headers.append(i)

In [45]:

*# Select columns of sign items*

sign\_columns = onehot[sign\_headers]

*# Perform aggregation of sign items into sign category*

signs = sign\_columns.sum(axis = 1) >= 1.0

*# Print support for signs*

print('Share of Signs: **%.2f**' % signs.mean())

Share of Signs: 0.20

**Defining an aggregation function**

Surprised by the high share of sign items in its inventory, the retailer decides that it makes sense to do further aggregation for different categories to explore the data better. This seems trivial to us, but the retailer has not previously been able to perform even a basic descriptive analysis of its transaction and items.

The retailer asks us to perform aggregation for the candles, bags, and boxes categories. To simplify the task, we decide to write a function. It will take a string that contains an item's category. It will then output a DataFrame that indicates whether each transaction includes items from that category.

In [46]:

**def** aggregate(item):

*# Select the column headers for sign items*

item\_headers = []

**for** i **in** onehot.columns:

wrd\_lst = convert\_str(str(i).lower())

**if** item **in** wrd\_lst:

item\_headers.append(i)

*# Select columns of sign items*

item\_columns = onehot[item\_headers]

*# Return category of aggregated items*

**return** item\_columns.sum(axis = 1) >= 1.0

In [47]:

*# Aggregate items for the bags, boxes, and candles categories*

bags = aggregate('bag')

boxes = aggregate('box')

candles = aggregate('candle')

print('Share of Bags: **%.2f**' % bags.mean())

print('Share of Boxes: **%.2f**' % boxes.mean())

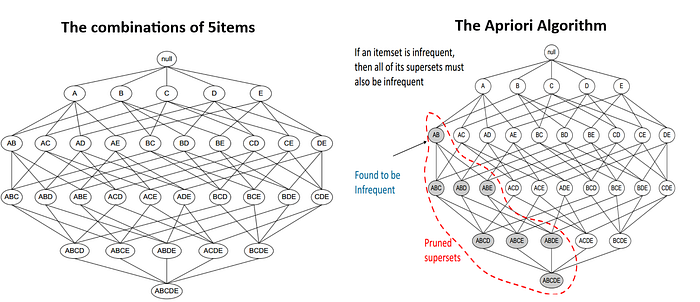
print('Share of Candles: **%.2f**' % candles.mean())

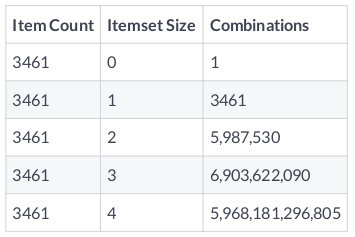
Share of Bags: 0.41

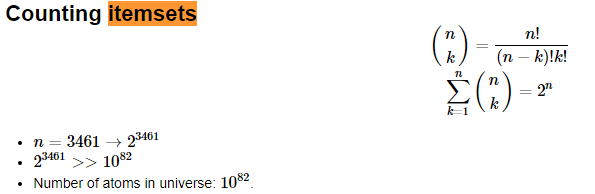
Share of Boxes: 0.39

Share of Candles: 0.11

**The Apriori algorithm**







**Reducing the number of itemsets**

* **Not possible to consider all itemsets**.
  + Not even possible to enumerate them.
* **How do we remove an itemset without even evaluating it**?
  + Could set maximum k� value.
* **Apriori algorithm offers alternative**.
  + Doesn't require enumeration of all itemsets.
  + Sensible rule for pruning.

**The Apriori principle**

* **Apriori principle.**
  + Subsets of frequent sets are frequent.
  + Retain sets known to be frequent.
  + Prune sets not known to be frequent.

**Ex:**

* **Candles = Infrequent**
  + -> {Candles, Signs} = Infrequent
* **{Candles, Signs} = Infrequent**
  + -> {Candles, Signs Boxes} = Infrequent
* **{Candles, Signs, Boxes} = Infrequent**
  + -> {Candles, Signs, Boxes, Bags} = Infrequent

**Identifying frequent itemsets with Apriori**

The aggregation exercise we performed for the online retailer proved helpful. It offered a starting point for understanding which categories of items appear frequently in transactions. The retailer now wants to explore the individual items themselves to find out which are frequent.

Here we'll apply the Apriori algorithm to the online retail dataset without aggregating first. Our objective will be to prune the itemsets using a minimum value of support and a maximum item number threshold.

In [48]:

*# Import apriori from mlxtend*

**from** **mlxtend.frequent\_patterns** **import** apriori

*# Compute frequent itemsets using the Apriori algorithm*

frequent\_itemsets = apriori(onehot,

min\_support = 0.05,

max\_len = 3,

use\_colnames = **True**)

|  | **support** | **itemsets** |
| --- | --- | --- |
| **0** | 0.054697 | (60 CAKE CASES VINTAGE CHRISTMAS) |
| **1** | 0.054459 | (ALARM CLOCK BAKELIKE GREEN) |
| **2** | 0.050535 | (ALARM CLOCK BAKELIKE RED) |
| **3** | 0.069203 | (ASSORTED COLOUR BIRD ORNAMENT) |
| **4** | 0.053983 | (BAKING SET 9 PIECE RETROSPOT) |

**Selecting a support threshold**

The manager of the online gift store looks at the results we provided from the previous exercise and commends us for the good work. She does, however, raise an issue: all of the itemsets we identified contain only one item. She asks whether it would be possible to use a less restrictive rule and to generate more itemsets, possibly including those with multiple items.

After agreeing to do this, we think about what might explain the lack of itemsets with more than 11 item. It can't be the max\_len parameter, since that was set to 33. We decide it must be support and decide to test two different values, each time checking how many additional itemsets are generated.

In [49]:

*# Import apriori from mlxtend*

**from** **mlxtend.frequent\_patterns** **import** apriori

*# Compute frequent itemsets using a support of 0.04 and length of 3*

frequent\_itemsets\_1 = apriori(onehot, min\_support = 0.04,

max\_len = 3, use\_colnames = **True**)

*# Compute frequent itemsets using a support of 0.05 and length of 3*

frequent\_itemsets\_2 = apriori(onehot, min\_support = 0.05,

max\_len = 3, use\_colnames = **True**)

*# Print the number of freqeuent itemsets*

print(len(frequent\_itemsets\_1), len(frequent\_itemsets\_2))

87 50

**Basic Apriori results pruning**

* **Apriori prunes itemsets.**
  + Applies minimum support threshold.
  + Modi ed version can prune by number of items.
  + Doesn't tell us about association rules.
* **Association rules.**
  + Many more association rules than itemsets.
  + {Bags, Boxes}: Bags -> Boxes OR Boxes -> Bags.

**How to compute association rules**

* **Computing rules from Apriori results.**
  + Difficult to enumerate for high n and k.
  + Could undo itemset pruning by Apriori.
* **Reducing number of association rules.**
  + mlxtend module offers means of pruning association rules.
  + association\_rules() takes frequent items, metric, and threshold.

**Generating association rules**

Previously we computed itemsets for the novelty gift store owner using the Apriori algorithm. You told the store owner that relaxing support from 0.05 to 0.04 increased the number of itemsets from 5050 to 8787. Satisfied with the descriptive work we've done, the store manager asks us to identify some association rules from those two sets of frequent itemsets we computed.

Our objective is to determine what association rules can be mined from these itemsets.

In [50]:

*# Import the association rule function from mlxtend*

**from** **mlxtend.frequent\_patterns** **import** association\_rules

*# Compute all association rules for frequent\_itemsets\_1*

rules\_1 = association\_rules(frequent\_itemsets\_1,

metric = "support",

min\_threshold = 0.001)

*# Compute all association rules for frequent\_itemsets\_2*

rules\_2 = association\_rules(frequent\_itemsets\_2,

metric = "support",

min\_threshold = 0.002)

*# Print the number of association rules generated*

print(len(rules\_1), len(rules\_2))

6 2

**Pruning with lift**

Once again, we report back to the novelty gift store manager. This time, we tell her that we identified 22 rules when you used a higher support threshold for the Apriori algorithm and only 66 rules when you used a lower threshold. She commends us for the good work, but asks you to consider using another metric to refine the two rules.

You remember that lift had a simple interpretation: values greater than 11 indicate that items co-occur more than we would expect if they were independently distributed across transactions. We decide to use lift, since that message will be simple to convey.

In [51]:

*# Import the association rules function*

**from** **mlxtend.frequent\_patterns** **import** association\_rules

*# Compute frequent itemsets using the Apriori algorithm*

frequent\_itemsets = apriori(onehot, min\_support = 0.03,

max\_len = 2, use\_colnames = **True**)

*# Compute all association rules for frequent\_itemsets*

rules = association\_rules(frequent\_itemsets,

metric = "lift",

min\_threshold = 1.0)

*# Print association rules*

rules.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 34 entries, 0 to 33

Data columns (total 9 columns):

antecedents 34 non-null object

consequents 34 non-null object

antecedent support 34 non-null float64

consequent support 34 non-null float64

support 34 non-null float64

confidence 34 non-null float64

lift 34 non-null float64

leverage 34 non-null float64

conviction 34 non-null float64

dtypes: float64(7), object(2)

memory usage: 2.5+ KB

In [52]:

rules.head()

|  | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | (ALARM CLOCK BAKELIKE GREEN) | (ALARM CLOCK BAKELIKE RED) | 0.054459 | 0.050535 | 0.036504 | 0.670306 | 13.264166 | 0.033752 | 2.879834 |
| **1** | (ALARM CLOCK BAKELIKE RED) | (ALARM CLOCK BAKELIKE GREEN) | 0.050535 | 0.054459 | 0.036504 | 0.722353 | 13.264166 | 0.033752 | 3.405550 |
| **2** | (HOT WATER BOTTLE KEEP CALM) | (CHOCOLATE HOT WATER BOTTLE) | 0.089893 | 0.062782 | 0.031510 | 0.350529 | 5.583238 | 0.025866 | 1.443048 |
| **3** | (CHOCOLATE HOT WATER BOTTLE) | (HOT WATER BOTTLE KEEP CALM) | 0.062782 | 0.089893 | 0.031510 | 0.501894 | 5.583238 | 0.025866 | 1.827135 |
| **4** | (HOT WATER BOTTLE TEA AND SYMPATHY) | (CHOCOLATE HOT WATER BOTTLE) | 0.062545 | 0.062782 | 0.033413 | 0.534221 | 8.509081 | 0.029486 | 2.012149 |

**Pruning with confidence**

We decide to see whether pruning by another metric might allow us to narrow things down even further.

What would be the right metric? Both lift and support are identical for all rules that can be generated from an itemset, so we decide to use confidence instead, which differs for rules produced from the same itemset.

In [53]:

*# Import the association rules function*

**from** **mlxtend.frequent\_patterns** **import** apriori, association\_rules

*# Compute frequent itemsets using the Apriori algorithm*

frequent\_itemsets = apriori(onehot, min\_support = 0.03,

max\_len = 2, use\_colnames = **True**)

*# Compute all association rules using confidence*

rules = association\_rules(frequent\_itemsets,

metric = "confidence",

min\_threshold = 0.4)

*# Print association rules*

rules.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 30 entries, 0 to 29

Data columns (total 9 columns):

antecedents 30 non-null object

consequents 30 non-null object

antecedent support 30 non-null float64

consequent support 30 non-null float64

support 30 non-null float64

confidence 30 non-null float64

lift 30 non-null float64

leverage 30 non-null float64

conviction 30 non-null float64

dtypes: float64(7), object(2)

memory usage: 2.2+ KB

In [54]:

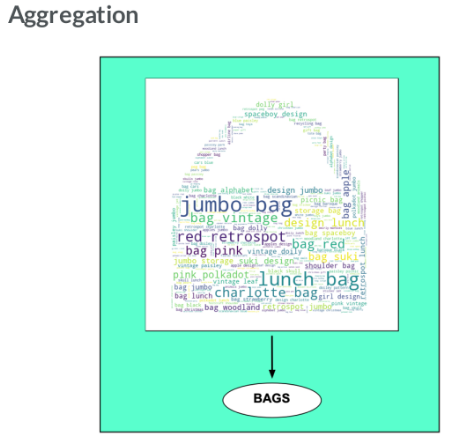
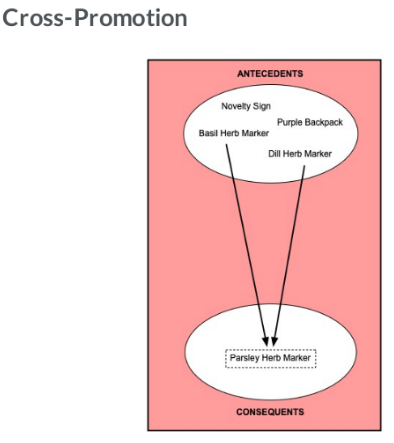
rules.head()

Out[54]:

|  | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | (ALARM CLOCK BAKELIKE GREEN) | (ALARM CLOCK BAKELIKE RED) | 0.054459 | 0.050535 | 0.036504 | 0.670306 | 13.264166 | 0.033752 | 2.879834 |
| **1** | (ALARM CLOCK BAKELIKE RED) | (ALARM CLOCK BAKELIKE GREEN) | 0.050535 | 0.054459 | 0.036504 | 0.722353 | 13.264166 | 0.033752 | 3.405550 |
| **2** | (CHOCOLATE HOT WATER BOTTLE) | (HOT WATER BOTTLE KEEP CALM) | 0.062782 | 0.089893 | 0.031510 | 0.501894 | 5.583238 | 0.025866 | 1.827135 |
| **3** | (HOT WATER BOTTLE TEA AND SYMPATHY) | (CHOCOLATE HOT WATER BOTTLE) | 0.062545 | 0.062782 | 0.033413 | 0.534221 | 8.509081 | 0.029486 | 2.012149 |
| **4** | (CHOCOLATE HOT WATER BOTTLE) | (HOT WATER BOTTLE TEA AND SYMPATHY) | 0.062782 | 0.062545 | 0.033413 | 0.532197 | 8.509081 | 0.029486 | 2.003953 |

**Advanced Apriori results pruning**

**Applications**



**Aggregation and filtering**

The store manager is now asking us to generate a floorplan proposal, where each pair of sections should contain one high support product and one low support product.

In [55]:

*# Aggregate items*

signs = aggregate('sign')

*# Concatenate aggregated items into 1 DataFrame*

aggregated = pd.concat([bags, boxes, candles, signs],axis=1)

aggregated.columns = ['bag','box','candle','sign']

*# Apply the apriori algorithm with a minimum support of 0.04*

frequent\_itemsets = apriori(aggregated, min\_support = 0.04, use\_colnames = **True**)

*# Generate the initial set of rules using a minimum support of 0.01*

rules = association\_rules(frequent\_itemsets,

metric = "support", min\_threshold = 0.01)

*# Set minimum antecedent support to 0.35*

rules = rules[rules['antecedent support'] > 0.35]

*# Set maximum consequent support to 0.35*

rules = rules[rules['consequent support'] < 0.35]

*# Print the remaining rules*

rules.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 9 entries, 2 to 29

Data columns (total 9 columns):

antecedents 9 non-null object

consequents 9 non-null object

antecedent support 9 non-null float64

consequent support 9 non-null float64

support 9 non-null float64

confidence 9 non-null float64

lift 9 non-null float64

leverage 9 non-null float64

conviction 9 non-null float64

dtypes: float64(7), object(2)

memory usage: 720.0+ bytes

In [56]:

rules.head()

Out[56]:

|  | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | (bag) | (candle) | 0.408680 | 0.112010 | 0.066587 | 0.162933 | 1.454634 | 0.020811 | 1.060835 |
| **4** | (bag) | (sign) | 0.408680 | 0.202021 | 0.123662 | 0.302589 | 1.497809 | 0.041100 | 1.144202 |
| **7** | (box) | (candle) | 0.385731 | 0.112010 | 0.082878 | 0.214858 | 1.918214 | 0.039672 | 1.130994 |
| **9** | (box) | (sign) | 0.385731 | 0.202021 | 0.126159 | 0.327065 | 1.618964 | 0.048233 | 1.185819 |
| **15** | (bag) | (candle, box) | 0.408680 | 0.082878 | 0.058026 | 0.141984 | 1.713182 | 0.024156 | 1.068888 |

**Applying Zhang's rule**

We learned that Zhang's rule is a continuous measure of association between two items that takes values in the −1,+1−1,+1 interval. A −1−1 value indicates a perfectly negative association and a +1+1 value indicates a perfectly positive association. In this exercise, we'll determine whether Zhang's rule can be used to refine a set of rules a gift store is currently using to promote products.

We will start by re-computing the original set of rules. After that, we will apply Zhang's metric to select only those rules with a high and positive association.

In [57]:

*# Funtion to compute Zhang's rule from mlxtend association\_rules output*

**def** zhangs\_rule(rules):

PAB = rules['support'].copy()

PA = rules['antecedent support'].copy()

PB = rules['consequent support'].copy()

NUMERATOR = PAB - PA\*PB

DENOMINATOR = np.max((PAB\*(1-PA).values,PA\*(PB-PAB).values), axis = 0)

**return** NUMERATOR / DENOMINATOR

In [58]:

*# Generate the initial set of rules using a minimum lift of 1.00*

rules = association\_rules(frequent\_itemsets, metric = "lift", min\_threshold = 1.00)

*# Set antecedent support to 0.04*

rules = rules[rules['antecedent support'] > 0.04]

*# Set consequent support to 0.04*

rules = rules[rules['consequent support'] > 0.04]

*# Compute Zhang's rule*

rules['zhang'] = zhangs\_rule(rules)

*# Set the lower bound for Zhang's rule to 0.5*

rules = rules[rules['zhang'] > 0.5]

rules[['antecedents', 'consequents']].info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 26 entries, 0 to 29

Data columns (total 2 columns):

antecedents 26 non-null object

consequents 26 non-null object

dtypes: object(2)

|  | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** | **zhang** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **25** | (candle, box) | (sign) | 0.082878 | 0.202021 | 0.040309 | 0.486370 | 2.407518 | 0.023566 | 1.553606 | 0.637466 |
| **26** | (sign, box) | (candle) | 0.126159 | 0.112010 | 0.040309 | 0.319510 | 2.852525 | 0.026178 | 1.304928 | 0.743194 |
| **27** | (candle) | (sign, box) | 0.112010 | 0.126159 | 0.040309 | 0.359873 | 2.852525 | 0.026178 | 1.365104 | 0.731352 |
| **28** | (sign) | (candle, box) | 0.202021 | 0.082878 | 0.040309 | 0.199529 | 2.407518 | 0.023566 | 1.145729 | 0.732644 |
| **29** | (box) | (candle, sign) | 0.385731 | 0.046254 | 0.040309 | 0.104501 | 2.259255 | 0.022467 | 1.065043 | 0.907382 |

**Advanced filtering with multiple metrics**

Earlier, we used data from an online novelty gift store to find antecedents that could be used to promote a targeted consequent. Since the set of potential rules was large, we had to rely on the Apriori algorithm and multi-metric filtering to narrow it down. In this exercise, we'll examine the full set of rules and find a useful one, rather than targeting a particular antecedent.

In this exercise, we'll apply the Apriori algorithm to identify frequent itemsets. We'll then recover the set of association rules from the itemsets and apply multi-metric filtering.

In [60]:

*# Apply the Apriori algorithm with a minimum support threshold of 0.04*

frequent\_itemsets = apriori(onehot, min\_support = 0.04, use\_colnames = **True**)

*# Recover association rules using a minium support threshold of 0.01*

rules = association\_rules(frequent\_itemsets, metric = 'support', min\_threshold = 0.01)

*# Apply a 0.002 antecedent support threshold, 0.01 confidence threshold, and 2.50 lift threshold*

filtered\_rules = rules[(rules['antecedent support'] > 0.002) &

(rules['consequent support'] > 0.01) &

(rules['confidence'] > 0.60) &

(rules['lift'] > 2.50)]

*# Print remaining rule*

filtered\_rules[['antecedents','consequents']]

Out[60]:

|  | **antecedents** | **consequents** |
| --- | --- | --- |
| **0** | (GARDENERS KNEELING PAD CUP OF TEA) | (GARDENERS KNEELING PAD KEEP CALM) |
| **2** | (PAPER CHAIN KIT VINTAGE CHRISTMAS) | (PAPER CHAIN KIT 50'S CHRISTMAS) |
| **4** | (WOODEN STAR CHRISTMAS SCANDINAVIAN) | (WOODEN HEART CHRISTMAS SCANDINAVIAN) |
| **5** | (WOODEN HEART CHRISTMAS SCANDINAVIAN) | (WOODEN STAR CHRISTMAS SCANDINAVIAN) |

**Movielens dataset**

The data consists of 105339105339 ratings applied over 1032910329 movies. There are 668668 users who has given their ratings for 149532149532 movies.

**Visualizing itemset support**

A content-streaming start-up has approached us for consulting services. To keep licensing fees low, they want to assemble a narrow library of movies that all appeal to the same audience. While they'll provide a smaller selection of content than the big players in the industry, they'll also be able to offer a low subscription fee.

We decide to use the MovieLens data and a heatmap for this project. Using a simple support-based heatmap will allow us to identify individual titles that have high support with other titles.

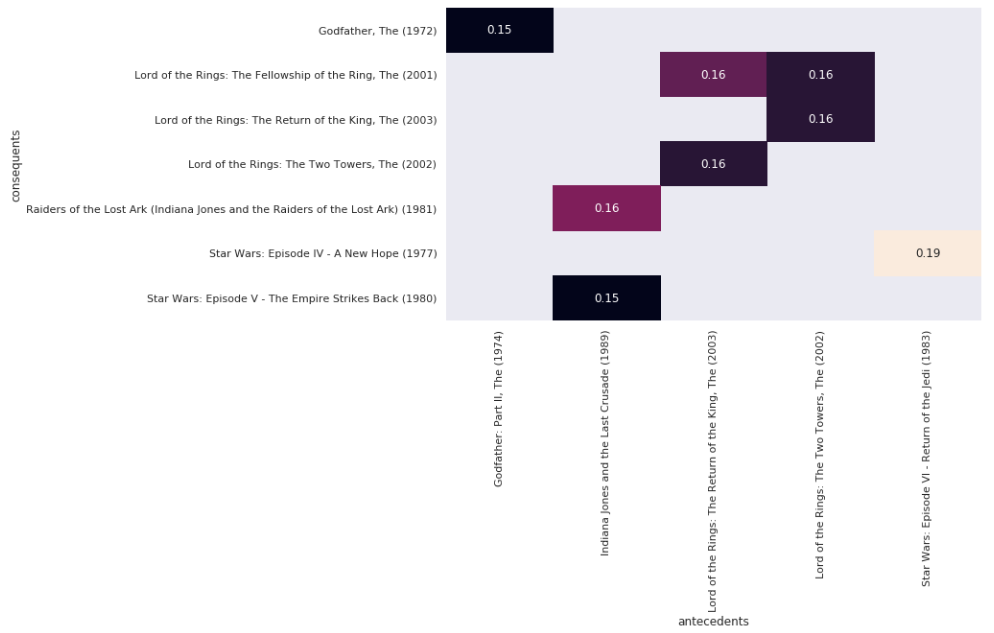
Out[66]:

|  | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | (Godfather: Part II, The (1974)) | (Godfather, The (1972)) | 0.169162 | 0.263473 | 0.151198 | 0.893805 | 3.392397 | 0.106628 | 6.935629 |
| **1** | (Indiana Jones and the Last Crusade (1989)) | (Raiders of the Lost Ark (Indiana Jones and th... | 0.184132 | 0.273952 | 0.164671 | 0.894309 | 3.264472 | 0.114227 | 6.869530 |
| **2** | (Indiana Jones and the Last Crusade (1989)) | (Star Wars: Episode V - The Empire Strikes Bac... | 0.184132 | 0.282934 | 0.151198 | 0.821138 | 2.902224 | 0.099100 | 4.009050 |
| **3** | (Lord of the Rings: The Return of the King, Th... | (Lord of the Rings: The Fellowship of the Ring... | 0.184132 | 0.220060 | 0.161677 | 0.878049 | 3.990045 | 0.121157 | 6.395509 |
| **4** | (Lord of the Rings: The Two Towers, The (2002)) | (Lord of the Rings: The Fellowship of the Ring... | 0.181138 | 0.220060 | 0.155689 | 0.859504 | 3.905774 | 0.115827 | 5.551338 |

**Heatmaps**

* Heatmaps help us understand a large number of rules between a small number of antecedents and consequents

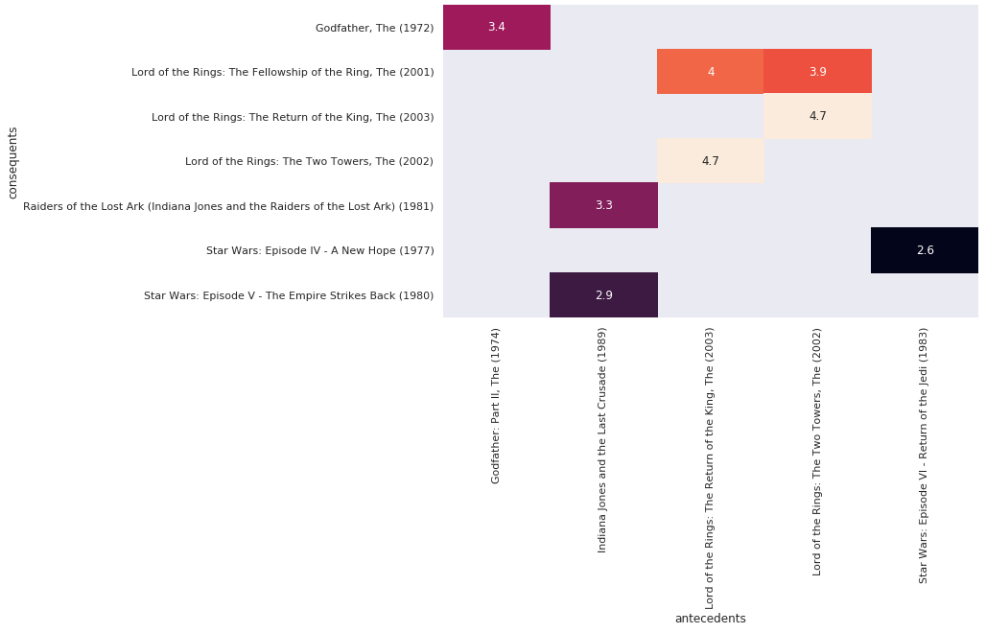
In [67]:



**Heatmaps with lift**

The founder likes the heatmap we've produced for her streaming service. After discussing the project further, however, we decide that that it is important to examine other metrics before making a final decision on which movies to license. In particular, the founder suggests that we select a metric that tells us whether the support values are higher than we would expect given the films' individual support values.

We recall that lift does this well and decide to use it as a metric. We also remember that lift has an important threshold at 1.01.0 and decide that it is important to replace the colorbar with annotations, so you can determine whether a value is greater than 1.01.0.



**Scatterplots**

* Scatter plots will help us to evaluate general tendencies and behaviors of rules between many antecedents and consequents but, without isolating any rule in particular.
* **A scatterplot displays pairs of values.**
  + Antecedent and consequent support.
  + Confidence and lift.
* **No model is assumed.**
  + No trend line or curve needed.
* **Can provide starting point for pruning.**
  + Identify patterns in data and rules.

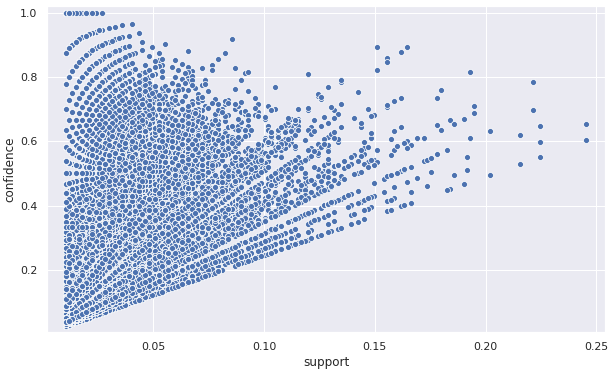
**What can we learn from scatterplots?**

* **Identify natural thresholds in data.**
  + Not possible with heatmaps or other visualizations.
* **Visualize entire dataset.**
  + Not limited to small number of rules.
* **Use findings to prune.**
  + Use natural thresholds and patterns to prune.

**Pruning with scatterplots**

After viewing your streaming service proposal from the previous exercise, the founder realizes that her initial plan may have been too narrow. Rather than focusing on initial titles, she asks you to focus on general patterns in the association rules and then perform pruning accordingly. Our goal should be to identify a large set of strong associations.

Fortunately, we've just learned how to generate scatterplots. We decide to start by plotting support and confidence, since all optimal rules according to many common metrics are located on the confidence-supply border.

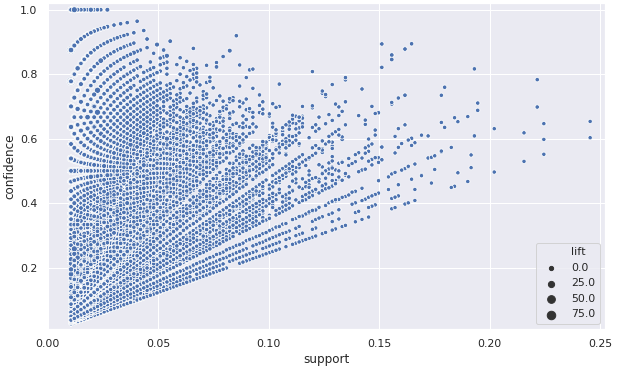


Notice that the confidence-support border roughly forms a triangle. This suggests that throwing out some low support rules would also mean that we would discard rules that are strong according to many common metrics.

**Optimality of the support-confidence border**

We return to the founder with the scatterplot produced in the previous exercise and ask whether she would like us to use pruning to recover the support-confidence border. We tell her about the Bayardo-Agrawal result, but she seems skeptical and asks whether we can demonstrate this in an example.

Recalling that scatterplots can scale the size of dots according to a third metric, we decide to use that to demonstrate optimality of the support-confidence border. We will show this by scaling the dot size using the lift metric, which was one of the metrics to which Bayardo-Agrawal applies.



If you look at the plot carefully, you'll notice that the highest values of lift are always at the support-confidence border for any given value of confidence.

**Parallel coordinates plot**

* The parallel coordinates plot will allow us to visualize whether a relationship exist between an antecedent and consequent. We can think of it as a directed network diagram. The plot shows connections between 22 objects that are related and indicates the direction of the relationship.

**When to use parallel coordinate plots**

* **Parallel coordinates vs. heatmap.**
  + Don't need intensity information.
  + Only want to know whether rule exists.
  + Want to reduce visual clutter.
* **Parallel coordinates vs. scatterplot.**
  + Want individual rule information.
  + Not interested in multiple metrics.
  + Only want to examine final rules.

**Using parallel coordinates to visualize rules**

Our visual demonstration in the previous exercise convinced the founder that the supply-confidence border is worthy of further exploration. She now suggests that we extract part of the border and visualize it. Since the rules that fall on the border are strong with respect to most common metrics, she argues that we should simply visualize whether a rule exists, rather than the intensity of the rule according to some metric. We realize that a parallel coordinates plot is ideal for such cases.

