**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

The use cases of market basket analysis

1. 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}

## Imports**1]:** Imports[¶](https://goldinlocks.github.io/Market-Basket-Analysis-in-Python/#Imports)

In [1]:

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

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

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

## In[2]:sns.set(style=”darkgrid”, color\_codes=True)

Pd.set\_option(‘display.max\_columns’, 75)

## Dataset[¶](https://goldinlocks.github.io/Market-Basket-Analysis-in-Python/#Dataset)

The contains information about customers buying different grocery items.

In[3]: data = pd.read\_csv('Market\_Basket.csv', header = **None**)

data.info()

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

RangeIndex: 7501 entries, 0 to 7500

Data columns (total 20 columns):

0 7501 non-null object

1 5747 non-null object

2 4389 non-null object

3 3345 non-null object

4 2529 non-null object

5 1864 non-null object

6 1369 non-null object

7 981 non-null object

8 654 non-null object

9 395 non-null object

10 256 non-null object

11 154 non-null object

12 87 non-null object

13 47 non-null object

14 25 non-null object

15 8 non-null object

16 4 non-null object

17 4 non-null object

18 3 non-null object

19 1 non-null object

dtypes: object(20)

memory usage: 1.1+ MB

In[4]:data.head()

Out[4]:

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 | frozen smoothie | spinach | olive oil |
| **1** | burgers | meatballs | eggs | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2** | chutney | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **3** | turkey | avocado | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **4** | mineral water | milk | energy |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

In[5] data.describe()

Out[4]:

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 | frozen smoothie | spinach | olive oil |
| **1** | burgers | meatballs | eggs | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2** | chutney | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **3** | turkey | avocado | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **4** | mineral water | milk | energy bar | whole wheat rice | green tea | NaN | NaN | NaN | NaN | NaN | NaN |  |  |  |  |  |  |  |  |  |

In[5]:data.describe()

Out[5]:

|  |
| --- |
|  |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
| **count** | 7501 | 5747 | 4389 | 3345 | 2529 | 1864 | 1369 | 981 | 654 | 395 | 256 | 154 | 87 | 47 | 25 | 8 | 4 | 4 | 3 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **unique** | 115 | 117 | 115 | 114 | 110 | 106 | 102 | 98 | 88 | 80 | 66 | 50 | 43 | 28 | 19 | 8 | 3 | 3 | 3 | 1 |
| **top** | mineral water | mineral water | mineral water | mineral water | green tea | french fries | green tea | green tea | green tea | green tea | low fat yogurt | green tea | green tea | green tea | magazines | salmon | frozen smoothie | protein bar | mayonnaise | olive oil |
| **freq** | 577 | 484 | 375 | 201 | 153 | 107 | 96 | 67 | 57 | 31 | 22 |  |  |  |  |  |  |  |  |  |

**EDQ**

**In[6]:** color = plt.cm.rainbow(np.linspace(0, 1, 40))

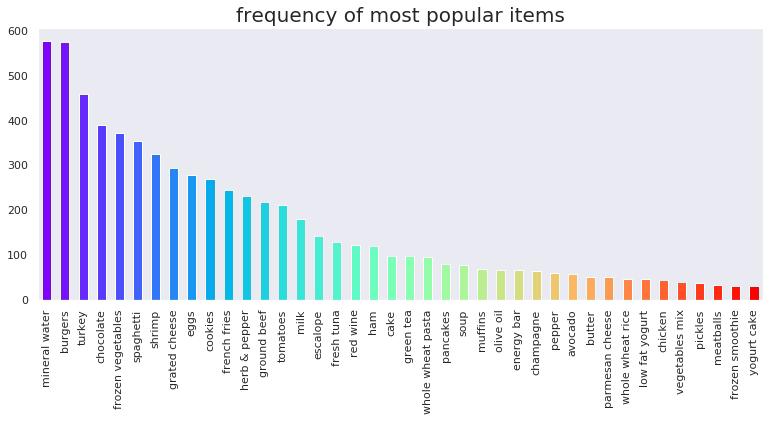
data[0].value\_counts().head(40).plot.bar(color = color, figsize=(13,5))

plt.title('frequency of most popular items', fontsize = 20)

plt.xticks(rotation = 90 )

plt.grid()

plt.show()

****

**In[7]:**

**Import networkx as nx**

**Data[‘food’] = ‘Food’**

**Food = data.truncate(before = -1, after = 15)**

**Food = nx.from\_pandas\_edgelist(food, source = ‘food’, target = 0, edge\_attr = True)**

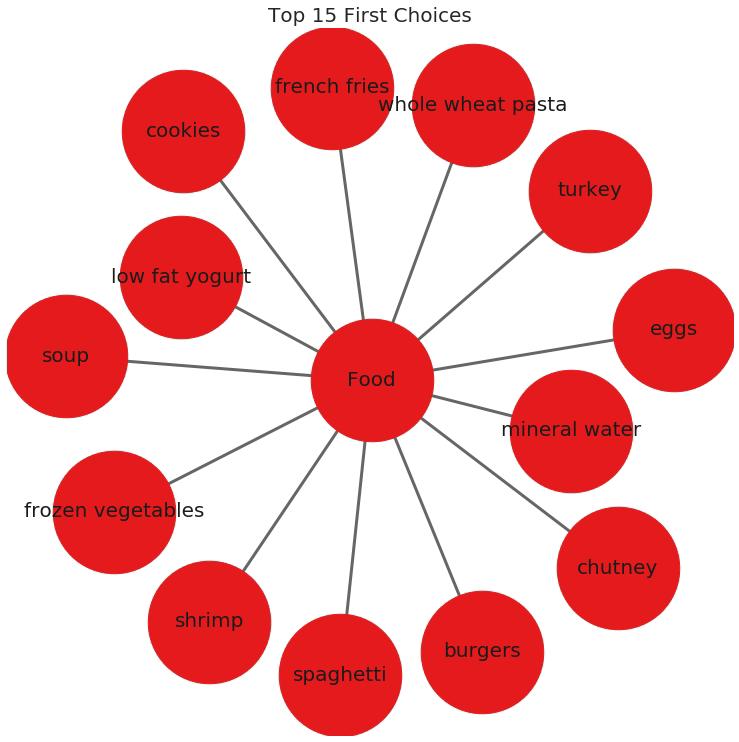
**In[8]: import** **warnings**

warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (13, 13)

pos = nxcolor = plt.cm.Set1(np.linspace(0, 15, 1))

nx.draw\_networkx\_nodes(food, pos, node\_size = 15000, node\_color = color)

****nx.draw\_networkx\_edges(food, pos, width = 3, alpha = 0.6, edge\_color = 'black')

nx.draw\_networkx\_labels(food, pos, font\_size = 20, font\_family = 'sans-serif'

## 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]: transaction**

**Out [10]:**

**[[‘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’,**

**‘Food’]]**

## Association rules[¶](https://goldinlocks.github.io/Market-Basket-Analysis-in-Python/#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[¶](https://goldinlocks.github.io/Market-Basket-Analysis-in-Python/#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))

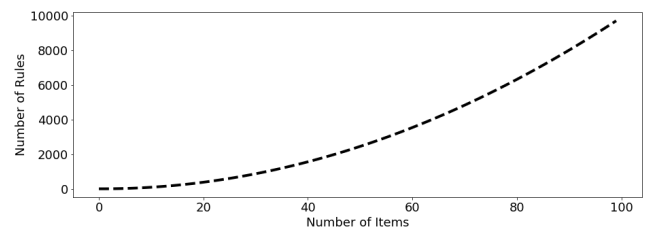
**Out[12]:**print(‘# of items:’,len(items))

print(list(items))

# of items: 122

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

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

Alt text that describes the graphic

**In[13]:**

**If ‘nan’ in items: items.remove(‘nan’)**

**Print(list(items))**

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

In[14]: *# Compute and print rules.*

rules = list(permutations(items, 2))

print('# of rules:',len(rules))

print(rules[:5])

# of rules: 14520

[('ham', 'champagne'), ('ham', 'red wine'), ('ham', 'asparagus'), ('ham', 'burgers'), ('ham', 'protein bar')]

*Compute the support*

support = onehot.mean()

support = pd.DataFrame(support, columns=['support']).sort\_values('support',ascending=**False**)

*# Print the support*

support.head()

Out[16]:

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

In [17]:

support.describe()

Out[17]:

|  | **support** |
| --- | --- |
| **count** | 121.000000 |
| **mean** | 0.040611 |
| **std** | 0.097542 |
| **min** | 0.000133 |
| **25%** | 0.007732 |
| **50%** | 0.015731 |
| **75%** | 0.042528 |
| **max** | 1.000000 |

# Confidence and lift¶

When support is misleading

Milk and bread frequently purchased together.

Support: {Milk} → {Bread}

Rule is not informative for marketing.

Milk and bread are both independently popular items.

The confidence metric

Can improve over support with additional metrics.

Adding confidence provides a more complete picture.

Confidence gives us the probability we will purchase Y

Given we have purchased X

.

SupportX&YSupportX

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

# 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 1

, this means that mineral water and french fries are paired together less frequently than we would expect if the pairings occurred by random chance.

# Compute lift

Lift = supportMF / (supportM \* supportF)

# Print lift

Print(“Lift: %.2f” % lift)

Lift: 0.83

As it turns out, lift is less than 1.0

. This does

# # Create rules DataFrame

Rules\_ = pd.DataFrame(rules, columns=[‘antecedents’,’consequents’])

# Define an empty list for metrics

Zhangs, conv, lev, antec\_supp, cons\_supp, suppt, conf, lft = [], [], [], [], [], [], [], []

# Loop over lists in itemsets

For itemset in rules:

# Extract the antecedent and consequent columns

Antecedent = onehot[itemset[0]]

Consequent = onehot[itemset[1]]

Antecedent\_support = onehot[itemset[0]].mean()

Consequent\_support = onehot[itemset[1]].mean()

Support = np.logical\_and(onehot[itemset[0]], onehot[itemset[1]]).mean()

Confidence = support / antecedent\_support

Lift = support / (antecedent\_support \* consequent\_support)

# Complete metrics and append it to the list

Antec\_supp.append(antecedent\_support)

Cons\_supp.append(consequent\_support)

Suppt.append(support)

Conf.append(confidence)

Lft.append(lift)

Lev.append(leverage(antecedent, consequent))

Conv.append(conviction(antecedent, consequent))

Zhangs.append(zhang(antecedent, consequent))

# Store results

Rules\_[‘antecedent support’] = antec\_supp

Rules\_[‘consequent support’] = cons\_supp

Rules\_[‘support’] = suppt

Rules\_[‘confidence’] = conf

Rules\_[‘lift’] = lft

Rules\_[‘leverage’] = lev

Rules\_[‘conviction’] = conv

Rules\_[‘zhang’] = zhangs

# Print results

Rules\_.sort\_values(‘zhang’,ascending=False).head()

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

Rules\_.describe()

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

Rules\_.info()

<class ‘pandas.core.frame.DataFrame’>

RangeIndex: 14520 entries, 0 to 14519

Data columns (total 10 columns):

Antecedents 14520 non-null object

Consequents 14520 non-null object

Antecedent support 14520 non-null float64

Consequent support 14520 non-null float64

Support 14520 non-null float64

Confidence 14520 non-null float64

Lift 14520 non-null float64

Leverage 14520 non-null float64

Conviction 14400 non-null float64

Zhang 14400 non-null float64

Dtypes: float64(8), object(2)

Memory usage: 1.1+ MB

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.

Generate large set of rules.

Filter rules using metrics.

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.

# 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 8598

Rules, but she wanted much less.

# 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

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.

## Apply the Apriori algorithm with a support value of 0.0095

Frequent\_itemsets = apriori(onehot, min\_support = 0.0095,

Use\_colnames = True, max\_len = 2)

# Generate association rules without performing additional pruning

Rules = association\_rules(frequent\_itemsets, metric=’support’,

Min\_threshold = 0.0)

# Generate scatterplot using support and confidence

Plt.figure(figsize=(10,6))

Sns.scatterplot(x = “support”, y = “confidence”, data = rules)

Plt.margins(0.01,0.01)

Plt.show()

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.

# Generate scatterplot using support and confidence

Plt.figure(figsize=(10,6))

Sns.scatterplot(x = “support”, y = “confidence”,

Size = “lift”, data = rules)

Plt.margins(0.01,0.01)

Plt.show()