MARKET BASKET INSIGHTS

# Introduction:

Apriori Algorithm is a widely-used and well-known Association Rule algorithm and is a popular algorithm used in market basket analysis. It is also considered accurate and overtop AIS and SETM algorithms. It helps to find frequent itemsets in transactions and identifies association rules between these items.

Market basket analysis is a [data mining](https://searchsqlserver.techtarget.com/definition/data-mining) technique used by retailers to increase sales by better understanding customer purchasing patterns. It involves analyzing large data sets, such as purchase history, to reveal product groupings, as well as products that are likely to be purchased together.

The adoption of market basket analysis was aided by the advent of electronic point-of-sale (POS) systems. Compared to handwritten records kept by store owners, the digital records generated by POS systems made it easier for applications to process and [analyze large volumes of purchase data](https://www.techtarget.com/searchcustomerexperience/tip/6-ways-to-use-analytics-to-improve-customer-engagement).

Implementation of market basket analysis requires a background in statistics and [data science](https://www.techtarget.com/searchenterpriseai/definition/data-science), as well as some algorithmic computer programming skills. For those without the needed technical skills, commercial, off-the-shelf tools exist.

## Overview

This notebook is part of a project focused on market basket analysis. We will begin by loading and preprocessing the dataset.

# Types of market basket analysis:

Retailers should understand the following types of market basket analysis:

* **Predictive market basket analysis.** This type considers items purchased in sequence to determine cross-sell.
* **Differential market basket analysis.** This type considers data across different stores, as well as purchases from different customer groups during different times of the day, month or year. If a rule holds in one dimension, such as store, time period or customer group, but does not hold in the others, analysts can determine the factors responsible for the exception. These insights can lead to new product offers that [drive higher sales](https://www.techtarget.com/searchcustomerexperience/tip/10-lead-scoring-best-practices-to-improve-sales-efficiency).

# Algorithms for market basket analysis:

In market basket analysis, [association rules](https://www.techtarget.com/searchbusinessanalytics/definition/association-rules-in-data-mining) are used to predict the likelihood of products being purchased together. Association rules count the frequency of items that occur together, seeking to find associations that occur far more often than expected.

Algorithms that use association rules include AIS, SETM and Apriori. The Apriori algorithm is commonly cited by data scientists in research articles about market basket analysis and is used to identify frequent items in the database, then evaluate their frequency as the datasets are expanded to larger sizes.

The [arules package for R](https://github.com/mhahsler/arules" \t "_blank) is an open source toolkit for association mining using the R programming language. This package supports the Apriori algorithm, along with the following other mining algorithms:

* arulesNBMiner
* Opusminer
* RKEEL
* RSarules

# Examples of market basket analysis:

Amazon's website uses a well-known example of market basket analysis. On a product page, Amazon presents users with related products, under the headings of "Frequently bought together" and "Customers who bought this item also bought."

Market basket analysis also applies to bricks-and-mortar stores. If analysis showed that magazine purchases often include the purchase of a bookmark, which could be considered an unexpected combination as the consumer did not purchase a book, then the bookstore might place a selection of bookmarks near the magazine rack.

# Benefits of market basket analysis:

Market basket analysis can increase sales and [customer satisfaction](https://www.techtarget.com/whatis/definition/customer-satisfaction-CSAT). Using data to determine that products are often purchased together, retailers can optimize product placement, offer special deals and create new product bundles to encourage further sales of these combinations.

These improvements can generate additional sales for the retailer, while making the shopping experience more productive and valuable for customers. By using market basket analysis, customers may feel a 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.

# Preprocessing

We'll preprocess the data to ensure it's ready for analysis.

# Visualization

## Visualizing Market Basket Analysis Results

We use matplotlib and seaborn libraries to create a scatterplot visualizing the results of the market basket analysis. The plot depicts the relationship between support, confidence, and lift for the generated association rules.

In [10]:

import matplotlib.pyplot as pltimport seaborn as sns

*# Plot scatterplot for Support vs. Confidence*plt.figure(figsize=(12, 8))sns.scatterplot(x="support", y="confidence", size="lift", data=rules, hue="lift", palette="viridis", sizes=(20, 200))plt.title('Market Basket Analysis - Support vs. Confidence (Size = Lift)')plt.xlabel('Support')plt.ylabel('Confidence')plt.legend(title='Lift', loc='upper right', bbox\_to\_anchor=(1.2, 1))plt.show()

## Interactive Market Basket Analysis Visualization

We leverage the Plotly Express library to create an interactive scatter plot visualizing the results of the market basket analysis. This plot provides an interactive exploration of the relationship between support, confidence, and lift for the generated association rules.

In [11]:

import plotly.express as px

*# Convert frozensets to lists for serialization*rules['antecedents'] = rules['antecedents'].apply(list)rules['consequents'] = rules['consequents'].apply(list)

*# Create an interactive scatter plot using plotly express*fig = px.scatter(rules, x="support", y="confidence", size="lift",

color="lift", hover\_name="consequents",

title='Market Basket Analysis - Support vs. Confidence',

labels={'support': 'Support', 'confidence': 'Confidence'})

*# Customize the layout*fig.update\_layout(

xaxis\_title='Support',

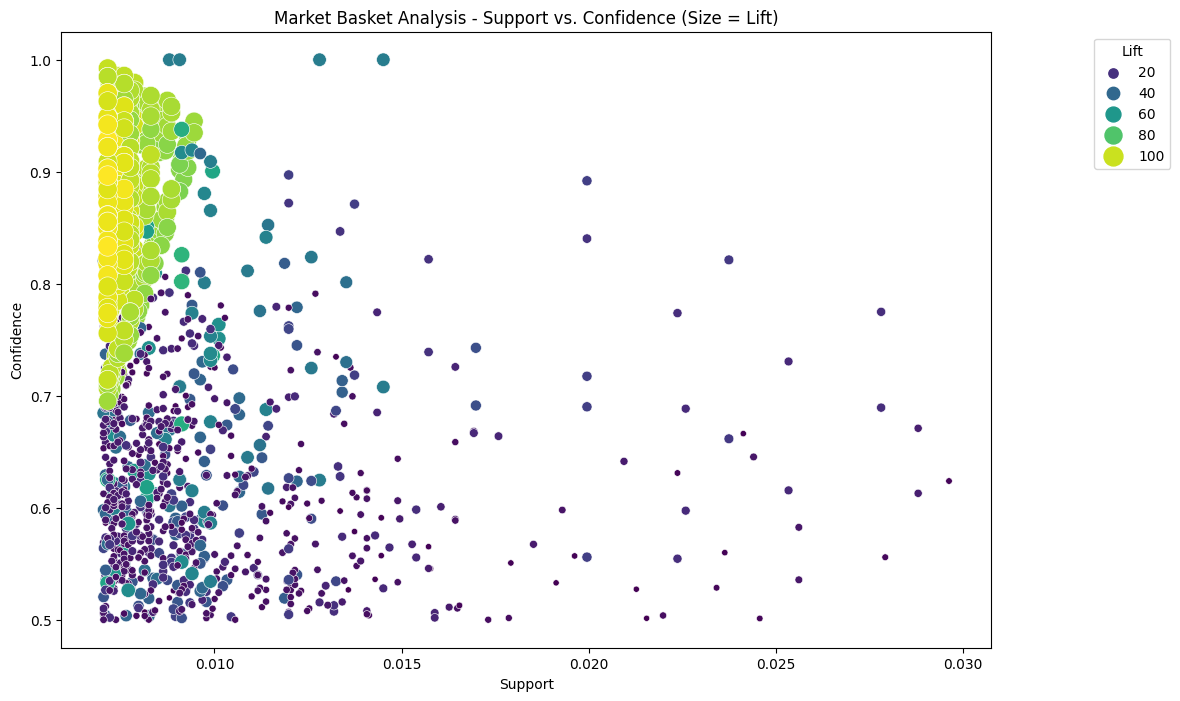
yaxis\_title='Confidence',

coloraxis\_colorbar\_title='Lift',

showlegend=True)

*# Show the interactive plot*fig.show()

0.010.020.030.50.60.70.80.9120406080100LiftMarket Basket Analysis - Support vs. ConfidenceSupportConfidence



# Imports

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

sns.set(style="darkgrid", color\_codes=True)

pd.set\_option('display.max\_columns', 75)

# Dataset

The contains information about customers buying different grocery items.

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

# input:

data.head()

# output:

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 NaN NaN NaN NaN NaN NaN NaN NaN NaN

# INPUT:

data.describe()

# OUTPUT:

0 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 15 8 4 3 1 2 2 1 1

# EDA

color = plt.cm.rainbow(np.linspace(0, 1, 40))

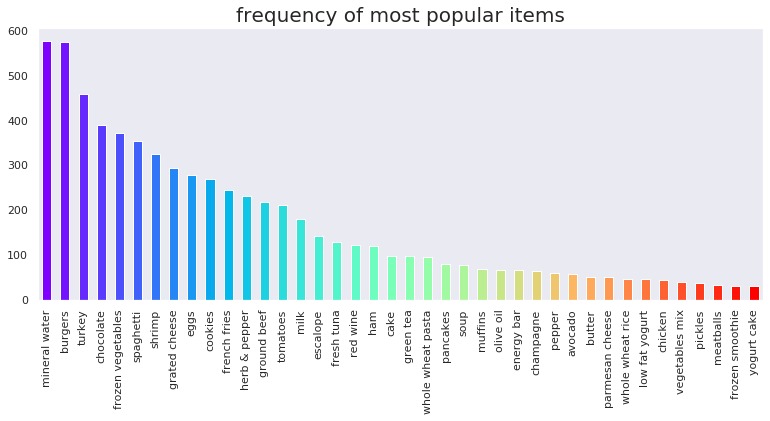
data[0].value\_counts().head(40).plot.bar(color = color, figsize=(13,5))

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

plt.xticks(rotation = 90 )

plt.grid()

plt.show()



INPUT:

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)

INPUT:

import warnings

warnings.filterwarnings('ignore')

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

pos = nx.spring\_layout(food)

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

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

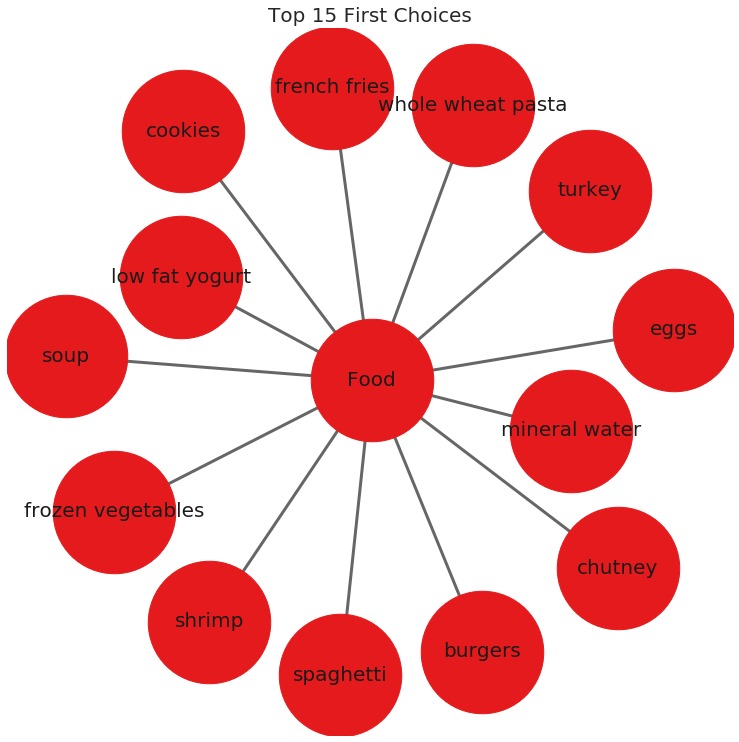
nx.draw\_networkx\_edges(food, pos, width = 3, alpha = 0.6, edge\_color = 'black')

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

plt.axis('off')

plt.grid()

lt.title('Top 15 First Choices', fontsize = 2plt.show()



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.

INPUT:

# 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))])

# INPUT:

transactions[:1]

# OUTPUT:

[['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

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.

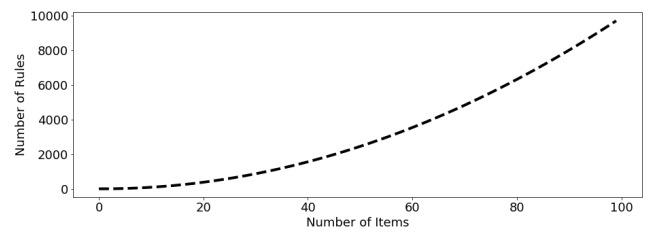
Must discard most rules.

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.



Alt text that describes the graphic

INPUT:

from itertools import permutations

# Extract unique items.

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

items = list(set(flattened))

INPUT:

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']

INPUT:

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']

INPUT:

# 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')]

# 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.

# INPUT:

# Import the transaction encoder function from mlxtend

from mlxtend.preprocessing import TransactionEncoder

# Instantiate transaction encoder and identify unique items

encoder = TransactionEncoder().fit(transactions)

# One-hot encode transactions

onehot = encoder.transform(transactions)

# Convert one-hot encoded data to DataFrame

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

# Print the one-hot encoded transaction dataset

onehot.head()

asparagus Food almonds antioxydant juice asparagus avocado babies food bacon barbecue sauce black tea blueberries body spray bramble brownies bug spray burger sauce burgers butter cake candy bars carrots cauliflower cereals champagne chicken chili chocolate chocolate bread chutney cider clothes accessories cookies cooking oil corn cottage cheese cream dessert wine ... parmesan cheese pasta pepper pet food pickles protein bar red wine rice salad salmon salt sandwich shallot shampoo shrimp soda soup spaghetti sparkling water spinach strawberries strong cheese tea tomato juice tomato sauce tomatoes toothpaste turkey vegetables mix water spray white wine whole weat flour whole wheat pasta whole wheat rice yams yogurt cake zucchini

0 False True True True False True False False False False False False False False False False False False False False False False False False False False False False False False False False False False True False False ... False False False False False False False False True True False False False False True False False False False True False False False True False False False False True False False True False False True False False

1 False True False False False False False False False False False False False False False False True False False False False False False False False False False False False False False False False False False False False ... False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False

2 False True False False False False False False False False False False False False False False False False False False False False False False False False False False True False False False False False False False False ... False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False

3 False True False False False True False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False ... False False False False False False False False False False False False False False False False False False False False False False False False False False False True False False False False False False False False False

4 False True False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False ... False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False False True False False False

5 rows × 121 columns

# 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.

number of transactions with items(s)number of transactions

# INPUT

# Compute the support

support = onehot.mean()

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

# Print the support

support.head()

# OUTPUT:

support

Food 1.000000

mineral water 0.238368

eggs 0.179709

spaghetti 0.174110

french fries 0.170911

# INPUT:

support.describe()

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

Alt text that describes the graphic

Support(Milk&Coffee)=0.20

Support(Milk) = 1.00

Support(Milk&Coffee)Support(Milk)=0.201.00=0.20

The probability of purchasing both milk and coffee does not change if we condition on purchasing milk. Purchasing milk tells us nothing about purchasing coffee.

The lift metric

Lift provides another metric for evaluating the relationship between items.

Numerator: Proportion of transactions that contain X

and Y

.

Denominator: Proportion if X

and Y

are assigned randomly and independently to transactions.

Support(X&Y)Support(X)Support(Y)

Lift >1

tells us 2

items occur in transactions together more often than we would expect based on their individual support values. This means the relationship is unlikely to be explained by random chance. This natural threshold is convenient for filtering purposes.

Lift <1

tells us 2

items are paired together less frequently in transactions than we would expect if the pairings occurred by random chance.

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.

# INPUT:

# 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 0.05

# INPUT:

# Add a mineral water+french fries column to the DataFrame onehot

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

# Compute the support

support = onehot.mean()

val = support.loc['mineral water+french fries']

# Print the support values

print(f'mineral water+french fries support = {val}')

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 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.

INPUT:

# 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 not give us good confidence that the association rule we recommended did not arise by random chance.

# Leverage and Conviction:

The leverage metric

Leverage also builds on support.

Leverage(X→Y)=Support(X&Y)−Support(X)Support(Y)

Leverage is similar to lift, but easier to interpret.

Leverage lies in −1 and +1 range.

Lift ranges from 0 to infinity.

# The conviction metric:

Conviction is also built using support.

More complicated and less intuitive than leverage.

Conviction(X→Y)=Support(X)Support(Y¯)

----------------------------------

Support(X&Y¯)

# 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.

# INPUT:

# 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 1

, 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.

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

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 0.741098

Aggregation

Alt text that describes the graphic

Novelty gift data

url = 'https://assets.datacamp.com/production/repositories/5654/datasets/5a3bc2ebccb77684a6d8a9f3fbec23fe04d4e3aa/online\_retail.csv'

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

CONCLUSION:

Finding items that buyers desire to buy is the major goal of market basket analysis.Market basket analysis may help sales and marketing teams develop more effective product placement,pricing,cross-sell,and up-sell tactics. 