# DATA SCIENCE

# FINAL TEST


**BY :**

NADHIFA SOFIA

19/448721/PPA/05804

**LECTURER :**

Dr. Sigit Priyanta, S.Si., M.Kom.

**MASTER OF COMPUTER SCIENCE**

**DEPARTMENT OF COMPUTER SCIENCE AND ELECTRONICS**

**FACULTY OF MATHEMATICS AND NATURAL SCIENCES**

**UNIVERSITAS GADJAH MADA**


**2020**

## I. Introduction

Market Basket Analysis is one of Machine Learning Techniques that is used by large retailers to uncover associations between items. It works by looking for combinations of items that occur together frequently in transactions. To put it another way, it allows retailers to identify relationships between the items that people buy.

Association Rules are widely used to analyze retail basket or transaction data, and are intended to identify strong rules discovered in transaction data using measures of interest, based on the concept of strong rules.

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness.

Based on the concept of strong rules, Rakesh Agrawal, Tomasz Imieliński and Arun Swami introduced association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule {X, Y} -> {Z} found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements.

In addition to the above example, market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection, continuous production, and bioinformatics. In contrast with sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

$$Rule:\ X \Rightarrow Y$$

$$Support = \frac{frq(X,Y)}{N}$$

$$Confidence = \frac{frq(X,Y)}{frq(X)}$$

$$Lift = \frac{Support}{Supp(X) \times Supp(Y)}$$

*Example:*


| Rule | Support | Confidence | Lift |
|---|---|---|---|
| $A \Rightarrow D$ | 2/5 | 2/3 | 10/9 |
| $C \Rightarrow A$ | 2/5 | 2/4 | 5/6 |
| $A \Rightarrow C$ | 2/5 | 2/3 | 5/6 |
| $B \& C \Rightarrow D$ | 1/5 | 1/3 | 5/9 |

**Figure 1.** Market Basket Analysis in a nutshell

Association Rule Mining is one of the ways to find patterns in data. It finds:

- Features (dimensions) which occur together
- Features (dimensions) which are "correlated"

What does the value of one feature tell us about the value of another feature?

For example, people who buy Chef Anton's Cajun Seasonings are likely to buy Tofu. Or we can rephrase the statement by saying: If (people buy Chef Anton's Cajun Seasonings), then (they buy Tofu). Note the if, then rule. This does not necessarily mean that if people buy Chef Anton's Cajun Seasoning, they buy Tofu. In General, we can say that if condition A tends to B it does not necessarily mean that B tends to A. From larger datasets (transactions), it will lead to a proper association rule.

By way of example we have table of items:

| TID | Items |
|---|---|
| 1 | Tofu, Chai |
| 2 | Tofu, Konbu, Chang, Pavlova |
| 3 | Chai, Konbu, Chang, Ikura |
| 4 | Tofu, Chai, Konbu, Chang |
| 5 | Tofu, Chai, Konbu, Ikura |

| | Chang | Tofu | Chai | Konbu | Pavlova | Ikura |
|---|---|---|---|---|---|---|
| **T1** | 0 | 1 | 1 | 0 | 0 | 0 |
| **T2** | 1 | 0 | 1 | 1 | 0 | 1 |
| **T3** | 1 | 1 | 1 | 1 | 0 | 1 |
| **T4** | 1 | 1 | 1 | 1 | 0 | 0 |
| **T5** | 0 | 1 | 1 | 1 | 0 | 1 |

**Rules 1**

- {Konbu, Chang} → {Chai} with Support = ⅖, Confidence = ⅔

**Rules 2**

- {Chai} → {Konbu, Chang} withSupport = ⅖, Confidence = 2/4

**Rules 3**

- {Chai, Konbu} → {Tofu}with Support = ⅖, Confidence = ⅔, etc.

**a. Support**

This measure gives an idea of how frequent an itemset is in all the transactions. Consider itemset1 = {tofu} and itemset2 = {pavlova}. There will be far more transactions containing bread than those containing shampoo. So as you rightly guessed, itemset1 will generally have a higher support than itemset2. Now consider itemset1 = {tofu, chai} and itemset2 = {tofu, konbu, chang, pavlova}. Many transactions will have both bread and butter on the cart but bread and shampoo? Not so much. So in this case, itemset1 will generally have a higher support than itemset2. Mathematically, support is the fraction of the total number of transactions in which the itemset occurs.

$$Support(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Total\ number\ of\ transactions}$$

Value of support helps us identify the rules worth considering for further analysis. For example, one might want to consider only the itemsets which occur at least 50 times out of a total of 10,000 transactions i.e. support = 0.005. If an itemset happens to have a very low support, we do not have enough information on the relationship between its items and hence no conclusions can be drawn from such a rule.

**2. Confidence**

This measure defines the likeliness of occurrence of consequent on the cart given that the cart already has the antecedents. That is to answer the question — of all the transactions containing say, {Konbu, Chang}, how many also had {Chai} on them? We can say by common knowledge that {Konbu Crunch} → {Chang} should be a high confidence rule. Technically, confidence is the conditional probability of occurrence of consequent given the antecedent.

$$Confidence(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Transactions\ containing\ X}$$

Let's consider a few more examples before moving ahead. What do you think would be the confidence for {Pavlova} → {Tofu}? That is, what fraction of transactions having butter also had bread? Very high i.e. a value close to 1? That’s right. What about {Ikura} → {Tofu}? High again. {Chang} → {Tofu}? Not so sure? Confidence for this rule will also be high since {Tofu} is such a frequent itemset and would be present in every other transaction.

It does not matter what you have in the antecedent for such a frequent consequent. The confidence for an association rule having a very frequent consequent will always be high.

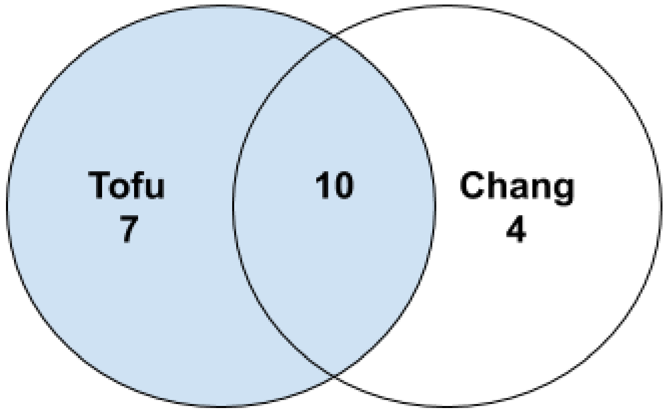



**Figure 2.** We have data with 10 tofu and chang, 70 data of tofu, and 4 data of chang

Consider the numbers from the figure on the left. Confidence for {Chang} → {Tofu} will be 10/(10+4) = 0.7 Looks like a high confidence value. But we know intuitively that these two products have a weak association and there is something misleading about this high confidence value. Lift is introduced to overcome this challenge.

**3. Lift**

Lift controls for the support (frequency) of consequent while calculating the conditional probability of occurrence of {Y} given {X}. Lift is a very literal term given to this measure. Think of it as the *lift* that {X} provides our confidence for having {Y} on the cart. To rephrase, lift is the rise in probability of having {Y} on the cart with the knowledge of {X} being present over the probability of having {Y} on the cart without any knowledge about presence of {X}. Mathematically,

$$Lift(\{X\} \rightarrow \{Y\}) = \frac{(Transactions\ containing\ both\ X\ and\ Y)/(Transactions\ containing\ X)}{Fraction\ of\ transactions\ containing\ Y}$$

In cases where {X} actually leads to {Y} on the cart, value of lift will be greater than 1. Let us understand this with an example which will be continuation of the {Chang} → {Tofu} rule. Probability of having tofu on the cart with the knowledge that chang is present (i.e. confidence) : 10/(10+4) = 0.7 Now to put this number in perspective, consider the probability of having milk on the cart without any knowledge about chang: 80/100 = 0.8

These numbers show that having a chang on the cart actually reduces the probability of having milk on the cart to 0.7 from 0.8! This will be a lift of 0.7/0.8 = 0.87. Now that's more like the real picture. A value of lift less than 1 shows that having chang on the cart does not increase the chances of occurrence of tofu on the cart in spite of the rule showing a high confidence value. A value of lift greater than 1 vouches for high association between {Y} and {X}. The greater the value of lift, greater are the chances of preference to buy {Y} if the customer has already bought {X}. Lift is the measure that will help store managers to decide product placements on aisle.

## II. Implementation

**Problem Solving of the Case**

The detailed code is at https://github.com/dhifaaans/uas_apriori/blob/master/uas_nadhifasofia_apriori.ipynb (which I create with my hard work to get A on this subject). In this project, I want to analyze what product will be ordered by the customers from some support, confidence, lift values. What products occur together? What features are correlated? For some reasons, I want to see can the discount product influence the customer to order more? So that I can arrange my future campaign marketing from certain products.

From the given dataset, I only used 2 of them which are order_details.csv which consists of orderID, productID, unitPrice, quantity, discount. So that, I tried a join clause to get the detailed products from products.csv to complete each other. Both of datasets are then combined by using WHERE clause like in MySQL, using 'productID' column.

```
In [23]: data1 = pd.read_csv('https://raw.githubusercontent.com/graphql-compose/graphql-compose-examples/master/examples/nor
         data1.head()
```

Out[23]:

| | orderID | productID | unitPrice | quantity | discount |
|---|---|---|---|---|---|
| **0** | 10248 | 11 | 14.0 | 12 | 0.0 |
| **1** | 10248 | 42 | 9.8 | 10 | 0.0 |
| **2** | 10248 | 72 | 34.8 | 5 | 0.0 |
| **3** | 10249 | 14 | 18.6 | 9 | 0.0 |
| **4** | 10249 | 51 | 42.4 | 40 | 0.0 |

```
In [24]: data2 = pd.read_csv('https://raw.githubusercontent.com/graphql-compose/graphql-compose-examples/master/examples/nor
         data2.head()
```

Out[24]:

| | productID | productName | supplierID | categoryID | quantityPerUnit | unitPrice | unitsInStock | unitsOnOrder | reorderLevel | discontinued |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Chai | 1 | 1 | 10 boxes x 20 bags | 18.00 | 39 | 0 | 10 | 0 |
| **1** | 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19.00 | 17 | 40 | 25 | 0 |
| **2** | 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10.00 | 13 | 70 | 25 | 0 |
| **3** | 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 22.00 | 53 | 0 | 0 | 0 |
| **4** | 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 21.35 | 0 | 0 | 0 | 1 |

Afterwards, I get the merged dataset like this :

```
In [2]: # Reading my UAS Dataset from Github (https://github.com/graphql-compose/graphql-compose-examples/blob/master/examp

        data1 = pd.read_csv('https://raw.githubusercontent.com/graphql-compose/graphql-compose-examples/master/examples/nor
        data2 = pd.read_csv('https://raw.githubusercontent.com/graphql-compose/graphql-compose-examples/master/examples/nor
        retaildata = pd.merge(data1, data2, on='productID')
        retaildata.head()
```

Out[2]:

| | orderID | productID | unitPrice_x | quantity | discount | productName | supplierID | categoryID | quantityPerUnit | unitPrice_y | unitsInStock | unitsOnOrder | reorderLev |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10248 | 11 | 14.0 | 12 | 0.0 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 3 |
| **1** | 10296 | 11 | 16.8 | 12 | 0.0 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 3 |
| **2** | 10327 | 11 | 16.8 | 50 | 0.2 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 3 |
| **3** | 10353 | 11 | 16.8 | 12 | 0.2 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 3 |
| **4** | 10365 | 11 | 16.8 | 24 | 0.0 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 3 |

As a Data Scientist wanna be, I tried to do Exploratory Data Analysis first to summarize main characteristics from datasets, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task :

## Do some Exploratory Data Analysis (EDA)

I did some Exploratory Data Analysis as most Data Scientist did to summarize main characteristics from datasets, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

In [3]:
```
retaildata.describe()
```

Out[3]:

| | orderID | productID | unitPrice_x | quantity | discount | supplierID | categoryID | unitPrice_y | unitsInStock | unitsOnOrder | reorderLevel | disc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 2155.000000 | 215 |
| **mean** | 10659.375870 | 40.793039 | 26.218520 | 23.812993 | 0.056167 | 14.488167 | 4.135499 | 27.948687 | 39.795824 | 8.074246 | 11.900232 | |
| **std** | 241.378032 | 22.159019 | 29.827418 | 19.022047 | 0.083450 | 8.228640 | 2.378567 | 31.610706 | 36.055668 | 19.748983 | 10.791421 | |
| **min** | 10248.000000 | 1.000000 | 2.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 2.500000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 10451.000000 | 22.000000 | 12.000000 | 10.000000 | 0.000000 | 7.000000 | 2.000000 | 12.500000 | 17.000000 | 0.000000 | 0.000000 | |
| **50%** | 10657.000000 | 41.000000 | 18.400000 | 20.000000 | 0.000000 | 14.000000 | 4.000000 | 19.450000 | 25.000000 | 0.000000 | 10.000000 | |
| **75%** | 10862.500000 | 60.000000 | 32.000000 | 30.000000 | 0.100000 | 21.000000 | 6.000000 | 34.000000 | 57.000000 | 0.000000 | 20.000000 | |
| **max** | 11077.000000 | 77.000000 | 263.500000 | 130.000000 | 0.250000 | 29.000000 | 8.000000 | 263.500000 | 125.000000 | 100.000000 | 30.000000 | |

In [4]:
```
retaildata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2155 entries, 0 to 2154
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   orderID          2155 non-null   int64
 1   productID        2155 non-null   int64
 2   unitPrice_x      2155 non-null   float64
 3   quantity         2155 non-null   int64
 4   discount         2155 non-null   float64
 5   productName      2155 non-null   object
 6   supplierID       2155 non-null   int64
 7   categoryID       2155 non-null   int64
 8   quantityPerUnit  2155 non-null   object
 9   unitPrice_y      2155 non-null   float64
```

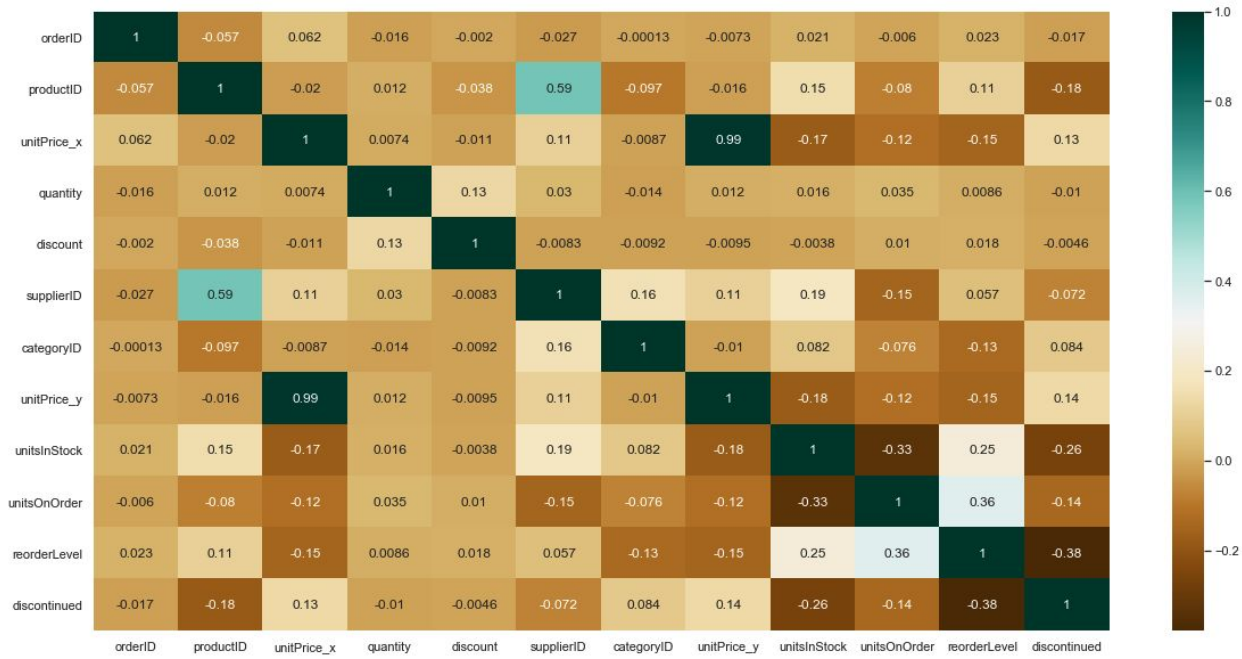
Then, I check the correlation for each feature

In [5]:
```
# To see every variables' correlation with HeatMap Visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)

plt.figure(figsize=(20,10))
c= retaildata.corr()
sns.heatmap(c,cmap='BrBG',annot=True)
c
```

Out[5]:

| | orderID | productID | unitPrice_x | quantity | discount | supplierID | categoryID | unitPrice_y | unitsInStock | unitsOnOrder | reorderLevel | discontinued |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **orderID** | 1.000000 | -0.057118 | 0.061743 | -0.016260 | -0.002041 | -0.027264 | -0.000129 | -0.007334 | 0.020603 | -0.005987 | 0.023311 | -0.016878 |
| **productID** | -0.057118 | 1.000000 | -0.019956 | 0.011932 | -0.037659 | 0.590308 | -0.097398 | -0.016404 | 0.151705 | -0.079977 | 0.110891 | -0.180518 |
| **unitPrice_x** | 0.061743 | -0.019956 | 1.000000 | 0.007366 | -0.011166 | 0.106456 | -0.008698 | 0.991273 | -0.174785 | -0.115969 | -0.150803 | 0.134009 |
| **quantity** | -0.016260 | 0.011932 | 0.007366 | 1.000000 | 0.128577 | 0.030246 | -0.014338 | 0.011677 | 0.016473 | 0.035448 | 0.008573 | -0.010183 |
| **discount** | -0.002041 | -0.037659 | -0.011166 | 0.128577 | 1.000000 | -0.008348 | -0.009170 | -0.009493 | -0.003835 | 0.010167 | 0.017835 | -0.004631 |
| **supplierID** | -0.027264 | 0.590308 | 0.106456 | 0.030246 | -0.008348 | 1.000000 | 0.160428 | 0.108567 | 0.186493 | -0.152965 | 0.056547 | -0.072309 |
| **categoryID** | -0.000129 | -0.097398 | -0.008698 | -0.014338 | -0.009170 | 0.160428 | 1.000000 | -0.010472 | 0.082043 | -0.076176 | -0.134834 | 0.084445 |
| **unitPrice_y** | -0.007334 | -0.016404 | 0.991273 | 0.011677 | -0.009493 | 0.108567 | -0.010472 | 1.000000 | -0.177929 | -0.115608 | -0.152413 | 0.136948 |
| **unitsInStock** | 0.020603 | 0.151705 | -0.174785 | 0.016473 | -0.003835 | 0.186493 | 0.082043 | -0.177929 | 1.000000 | -0.331962 | 0.246886 | -0.262432 |
| **unitsOnOrder** | -0.005987 | -0.079977 | -0.115969 | 0.035448 | 0.010167 | -0.152965 | -0.076176 | -0.115608 | -0.331962 | 1.000000 | 0.361687 | -0.140665 |
| **reorderLevel** | 0.023311 | 0.110891 | -0.150803 | 0.008573 | 0.017835 | 0.056547 | -0.134834 | -0.152413 | 0.246886 | 0.361687 | 1.000000 | -0.379406 |
| **discontinued** | -0.016878 | -0.180518 | 0.134009 | -0.010183 | -0.004631 | -0.072309 | 0.084445 | 0.136948 | -0.262432 | -0.140665 | -0.379406 | 1.000000 |




## Data Preprocessing

This is an important step in the data mining process. Data-gathering methods are often loosely controlled, resulting in out-of-range values, impossible data combinations, missing values, etc. Data Preprocessing that I did

- Remove spaces on productName from beginning and end of the product attributes
- Remove the redundant data from orderID since we want to list the unique value of it
- Convert the orderID into string, so the algorithm will understand it

## Data Preprocessing

**2. Preprocess existing data that you think is needed.**

- Remove spaces on productName from beginning and end of the product attributes
- Remove the redundance data from orderID since we want to list the unique value of it (no duplicates)
- Convert the orderID into string, so the algorithm will understand it

In [6]:
```
# Data Preparation
retaildata['productName'] = retaildata['productName'].str.strip()  # Remove spaces from beginning & end
retaildata.dropna(axis=0, subset=['orderID'], inplace=True)  # Remove duplicated order id
retaildata['orderID'] = retaildata['orderID'].astype('str')  # Convert order id -> string so algorithm will underst
retaildata.head() |
```

Out[6]:

| ·ID | productID | unitPrice_x | quantity | discount | productName | supplierID | categoryID | quantityPerUnit | unitPrice_y | unitsInStock | unitsOnOrder | reorderLevel | discon |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ·48 | 11 | 14.0 | 12 | 0.0 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 30 | |
| ·96 | 11 | 16.8 | 12 | 0.0 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 30 | |
| ·27 | 11 | 16.8 | 50 | 0.2 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 30 | |
| ·53 | 11 | 16.8 | 12 | 0.2 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 30 | |
| ·65 | 11 | 16.8 | 24 | 0.0 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21.0 | 22 | 30 | 30 | |

As I stated in the beginning, I want to analyze the association rule from discount products. Because market basket analysis is computationally expensive, so I just use products with price 0.25% off. If you want to see the whole association rules from every product lists, just use my code then tweak on the discount parameter.

```
In [8]: retaildata['discount'].value_counts()
```

```
Out[8]: 0.00    1317
0.05     185
0.10     173
0.20     161
0.15     157
0.25     154
0.03       3
0.02       2
0.01       1
0.06       1
0.04       1
Name: discount, dtype: int64
```

```
In [11]: # import warnings
# warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
In [12]: # Run market basket analysis to certain country, to avoid computationally expensive
# Create basket transactions
basket = (retaildata[retaildata['discount'] == 0.25]
          .groupby(['orderID', 'productName'])['quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('orderID'))
```

```
In [13]: basket.head()
```

Out[13]:

| productName | Alice Mutton | Boston Crab Meat | Camembert Pierrot | Carnarvon Tigers | Chai | Chang | Chartreuse verte | Chef Anton's Cajun Seasoning | Chef Anton's Gumbo Mix | Côte de Blaye | ... | Steeleye Stout | Tarte au sucre | Teatime Chocolate Biscuits | Thüringer Rostbratwurst | To |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| orderID | | | | | | | | | | | | | | | | |
| 10260 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 15.0 | 0.0 | 0.0 | C |
| 10263 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | C |
| 10279 | 15.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | C |
| 10284 | 0.0 | 0.0 | 20.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | C |
| 10298 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | C |

After that, I convert all positive values to 1 and everything else to 0, so that the apriori algorithm can work.

We can use Association Rules in any dataset where features take only two values i.e., 0/1. Some examples are listed below:

- Market Basket Analysis is a popular application of Association Rules.
- People who buy Chef Anton's Cajun Seasoning are likely to buy Tofu
- People who buy with price 0.25% off (like me who loves discount)

```
In [14]: # Converting all positive vaues to 1 and everything else to 0, so that the apriori algorithm can work
def my_encoding(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket.applymap(my_encoding)
```

After finishing some data mining, now let's take a look at how the Machine Learning Algorithm works. I proposed apriori algorithm to generate association rules. Apriori uses a breadth-first search strategy to count the support of itemsets and uses a candidate generation function which exploits the downward closure property of support.

## Training Model

### Support

Support is an indication of how frequently the itemset appears in the dataset. The support of X with respect to T is defined as the proportion of transactions t in the dataset which contains the itemset X.

supp(X) = |{t element T; X contains t}| / |T|

You can see the explanation from this url.

In [15]:
```
# Generatig frequent itemsets
frequent_itemsets = apriori(basket_sets, min_support=0.01, use_colnames=True)
```

In [16]:
```
frequent_itemsets.head()
```

Out[16]:

| | support | itemsets |
|---|---|---|
| 0 | 0.069444 | (Alice Mutton) |
| 1 | 0.013889 | (Boston Crab Meat) |
| 2 | 0.069444 | (Camembert Pierrot) |
| 3 | 0.041667 | (Carnarvon Tigers) |
| 4 | 0.069444 | (Chai) |

### Confidence

Confidence is an indication of how often the rule has been found to be true.

The confidence value of a rule, X -> Y , with respect to a set of transactions T, is the proportion of the transactions that contains X which also contains Y.

Confidence is defined as:

conf(X -> Y) = supp(X U Y) / supp(X)

supp(X U Y) means the support of the union of the items in X and Y. This is somewhat confusing since we normally think in terms of probabilities of events and not sets of items.

### Lift

Lift is a measure of the performance of a targeting model (association rule) at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model. A targeting model is doing a good job if the response within the target is much better than the average for the population as a whole. Lift is simply the ratio of these values: target response divided by average response.

For example, suppose a population has an average response rate of 5%, but a certain model (or rule) has identified a segment with a response rate of 20%. Then that segment would have a lift of 4.0 (20%/5%).

In [17]:
```
# Generating rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=0)
```

In [18]:
```
rules.head()
```

Out[18]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 0 | (Boston Crab Meat) | (Alice Mutton) | 0.013889 | 0.069444 | 0.013889 | 1.000000 | 14.4 | 0.012924 | inf |
| 1 | (Alice Mutton) | (Boston Crab Meat) | 0.069444 | 0.013889 | 0.013889 | 0.200000 | 14.4 | 0.012924 | 1.232639 |
| 2 | (Chang) | (Alice Mutton) | 0.083333 | 0.069444 | 0.013889 | 0.166667 | 2.4 | 0.008102 | 1.116667 |
| 3 | (Alice Mutton) | (Chang) | 0.069444 | 0.083333 | 0.013889 | 0.200000 | 2.4 | 0.008102 | 1.145833 |
| 4 | (Escargots de Bourgogne) | (Alice Mutton) | 0.027778 | 0.069444 | 0.013889 | 0.500000 | 7.2 | 0.011960 | 1.861111 |

Moreover, here is the association rules result that are generated by apriori algorithm. By using some sort of parameters (with support > 0.01, lift >= 0.3 and confidence >= 0.3), the result would be like :

## Making recommendations

In [19]:
```
basket_sets["Chef Anton's Cajun Seasoning"].sum()
```
Out[19]: 2

In [20]:
```
basket_sets['Tofu'].sum()
```
Out[20]: 2

In [21]:
```
# See the apriori algorithm by using certain parameters
rules[ (rules['lift'] >= 3) &
      (rules['confidence'] >= 0.3) ]
```
Out[21]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| **0** | (Boston Crab Meat) | (Alice Mutton) | 0.013889 | 0.069444 | 0.013889 | 1.0 | 14.4 | 0.012924 | inf |
| **4** | (Escargots de Bourgogne) | (Alice Mutton) | 0.027778 | 0.069444 | 0.013889 | 0.5 | 7.2 | 0.011960 | 1.861111 |
| **10** | (Zaanse koeken) | (Alice Mutton) | 0.055556 | 0.069444 | 0.027778 | 0.5 | 7.2 | 0.023920 | 1.861111 |
| **11** | (Alice Mutton) | (Zaanse koeken) | 0.069444 | 0.055556 | 0.027778 | 0.4 | 7.2 | 0.023920 | 1.574074 |
| **13** | (Boston Crab Meat) | (Zaanse koeken) | 0.013889 | 0.055556 | 0.013889 | 1.0 | 18.0 | 0.013117 | inf |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **747** | (Zaanse koeken, Alice Mutton) | (Gnocchi di nonna Alice, Chang, Escargots de B... | 0.027778 | 0.013889 | 0.013889 | 0.5 | 36.0 | 0.013503 | 1.972222 |
| **748** | (Chang, Escargots de Bourgogne) | (Gnocchi di nonna Alice, Zaanse koeken, Alice ... | 0.013889 | 0.013889 | 0.013889 | 1.0 | 72.0 | 0.013696 | inf |
| **749** | (Chang, Zaanse koeken) | (Gnocchi di nonna Alice, Escargots de Bourgogn... | 0.013889 | 0.013889 | 0.013889 | 1.0 | 72.0 | 0.013696 | inf |
| **750** | (Escargots de Bourgogne, Zaanse koeken) | (Gnocchi di nonna Alice, Chang, Alice Mutton) | 0.013889 | 0.013889 | 0.013889 | 1.0 | 72.0 | 0.013696 | inf |
| **754** | (Escargots de Bourgogne) | (Gnocchi di nonna Alice, Chang, Zaanse koeken,... | 0.027778 | 0.013889 | 0.013889 | 0.5 | 36.0 | 0.013503 | 1.972222 |

565 rows × 9 columns