*Srushti Shah*
*191071902*
*Final Year B. Tech. CE*
*Data Mining and Warehouse*

# ***Experiment No. 5***

***AIM***: To perform association rule mining

***THEORY:***

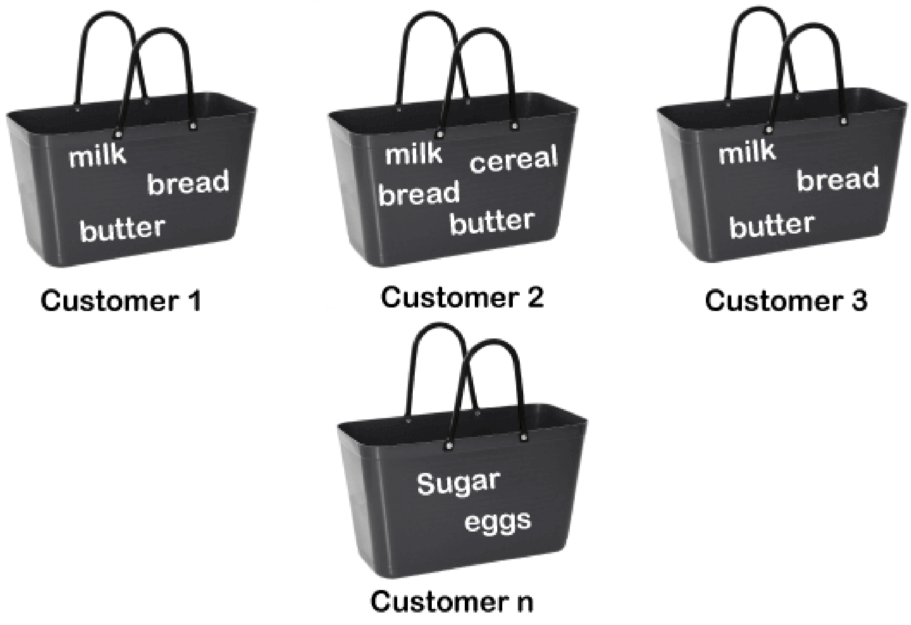
**Association Rule Mining**, as the name suggests, association rules are simple If/Then statements that help discover relationships between seemingly independent relational databases or other data repositories.

Most machine learning algorithms work with numeric datasets and hence tend to be mathematical. However, association rule mining is suitable for non-numeric, categorical data and requires just a little bit more than simple counting.

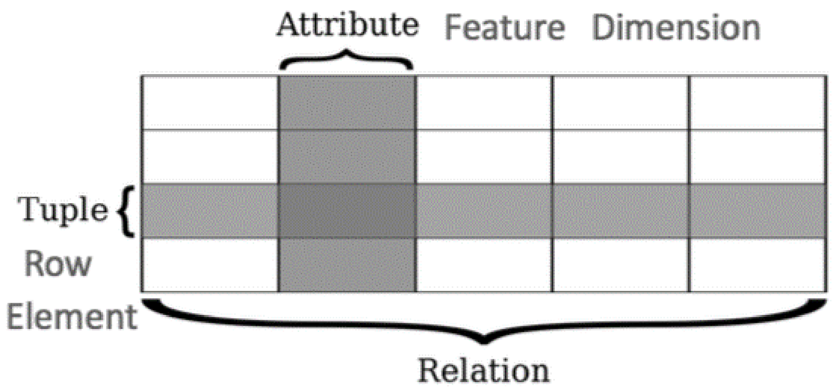
Association rule mining is a procedure which aims to observe frequently occurring patterns, correlations, or associations from datasets found in various kinds of databases such as relational databases, transactional databases, and other forms of repositories.

The association rule learning is one of the very important concepts of machine learning, and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:



The data that we are going to deal with looks like this:



$$\mathbf{x_i} = (x_{i1}, x_{i2}, \ldots, x_{ip}) \in \mathbb{R}^p$$

$$X \in \mathbb{R}^{n \times p}$$

**What Association Rule Mining Aims to Achieve?**

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 diapers are likely to buy baby powder. Or we can rephrase the statement by saying: If (people buy diaper), then (they buy baby powder). Note the if, then rule. This does not necessarily mean that if people buy baby powder, they buy diaper. In General, we can say that if condition A tends to B it does not necessarily mean that B tends to A. Watch the directionality!

### Measures of Effectiveness of the Rule

The measures of effectiveness of the rule are as Follows:

- Support
- Confidence
- Lift
- Others: Affinity, Leverage

**Measure 1: Support**. This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears. In Table 1 below, the support of {apple} is 4 out of 8, or 50%. Itemsets can also contain multiple items. For instance, the support of {apple, beer, rice} is 2 out of 8, or 25%.

Support {🍎} = $\frac{4}{8}$

| | |
|---|---|
| Transaction 1 | 🍎 🍺 🍚 🍗 |
| Transaction 2 | 🍎 🍺 🍚 |
| Transaction 3 | 🍎 🍺 |
| Transaction 4 | 🍎 🍐 |
| Transaction 5 | 🍼 🍺 🍚 🍗 |
| Transaction 6 | 🍼 🍺 🍚 |
| Transaction 7 | 🍼 🍺 |
| Transaction 8 | 🍼 🍐 |

Table 1. Example Transactions

If you discover that sales of items beyond a certain proportion tend to have a significant impact on your profits, you might consider using that proportion as your *support threshold*. You may then identify itemsets with support values above this threshold as significant itemsets.

**Measure 2: Confidence**. This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears. In Table 1, the confidence of {apple -> beer} is 3 out of 4, or 75%.

$$\text{Confidence}\ \{\text{🍎} \rightarrow \text{🍺}\} = \frac{\text{Support}\ \{\text{🍎}, \text{🍺}\}}{\text{Support}\ \{\text{🍎}\}}$$

One drawback of the confidence measure is that it might misrepresent the importance of an association. This is because it only accounts for how popular apples are, but not beers. If beers are also very popular in general, there will be a higher chance that a transaction containing apples will also contain beers, thus inflating the confidence measure. To account for the base popularity of both constituent items, we use a third measure called lift.

**Measure 3: Lift**. This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is. In Table 1, the lift of {apple -> beer} is 1,which implies no association between items. A lift value greater than 1 means that item Y is *likely* to be bought if item X is bought, while a value less than 1 means that item Y is *unlikely* to be bought if item X is bought.

$$\text{Lift}\ \{\text{🍎} \rightarrow \text{🍺}\} = \frac{\text{Support}\ \{\text{🍎}, \text{🍺}\}}{\text{Support}\ \{\text{🍎}\} \times \text{Support}\ \{\text{🍺}\}}$$

**An Illustration**

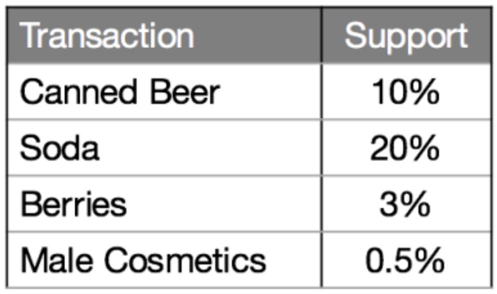
We use a dataset on grocery transactions from the *arules* R library. It contains actual transactions at a grocery outlet over 30 days. The network graph below shows associations between selected items. Larger circles imply higher support, while red circles imply higher lift:

One drawback of the confidence measure is that it tends to misrepresent the importance of an association. To demonstrate this, we go back to the main dataset to pick 3 association rules containing beer:

| Transaction | Support | Confidence | Lift |
|---|---|---|---|
| Canned Beer → Soda | 1% | 20% | 1.0 |
| Canned Beer → Berries | 0.1% | 1% | 0.3 |
| Canned Beer → Male Cosmetics | 0.1% | 1% | 2.6 |

Association measures for beer-related rules

The {beer -> soda} rule has the highest confidence at 20%. However, both beer and soda appear frequently across all transactions , so their association could simply be a fluke. This is confirmed by the lift value of {beer -> soda}, which is 1, implying no association between beer and soda.



| Transaction | Support |
|---|---|
| Canned Beer | 10% |
| Soda | 20% |
| Berries | 3% |
| Male Cosmetics | 0.5% |

Table 3. Support of individual items

On the other hand, the {beer -> male cosmetics} rule has a low confidence, due to few purchases of male cosmetics in general. However, whenever someone does buy male cosmetics, he is very likely to buy beer as well, as inferred from a high lift value of 2.6. The converse is true for {beer -> berries}. With a lift value below 1, we may conclude that if someone buys berries, he would likely be averse to beer.

It is easy to calculate the popularity of a single itemset, like {beer, soda}. However, a business owner would not typically ask about individual itemsets. Rather, the owner would be more interested in having a *complete* list of popular itemsets. To get this list, one needs to calculate the support values for every possible configuration of items, and then shortlist the itemsets that meet the minimum support threshold.
In a store with just 10 items, the total number of possible configurations to examine would be a whopping 1023. This number increases exponentially in a store with hundreds of items.

Is there a way to reduce the number of item configurations to consider?

## Apriori Algorithm

The *apriori principle* can reduce the number of itemsets we need to examine. Put simply, the apriori principle states that
if an itemset is infrequent, then all its **supersets** must also be infrequent
This means that if {beer} was found to be infrequent, we can expect {beer, pizza} to be equally or even more infrequent. So in consolidating the list of popular itemsets, we need not consider {beer, pizza}, nor any other itemset configuration that contains beer.

## Finding itemsets with high support

Using the apriori principle, the number of itemsets that have to be examined can be pruned, and the list of popular itemsets can be obtained in these steps:

**Step 0**. Start with itemsets containing just a single item, such as {apple} and {pear}.
**Step 1**. Determine the support for itemsets. Keep the itemsets that meet your minimum support threshold, and remove itemsets that do not.

**Step 2**. Using the itemsets you have kept from Step 1, generate all the possible itemset configurations.
**Step 3**. Repeat Steps 1 & 2 until there are no more new itemsets.
This iterative process is illustrated in the animated GIF below:

**Finding item rules with high confidence or lift**

We have seen how the apriori algorithm can be used to identify itemsets with high support. The same principle can also be used to identify item associations with high confidence or lift. Finding rules with high confidence or lift is less computationally taxing once high-support itemsets have been identified, because confidence and lift values are calculated using support values.

Take for example the task of finding high-confidence rules. If the rule

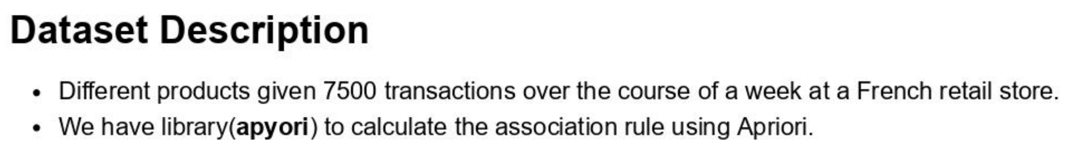
{beer, chips -> apple}

has low confidence, all other rules with the same constituent items and with apple on the right hand side would have low confidence too. Specifically, the rules

{beer -> apple, chips}
{chips -> apple, beer}

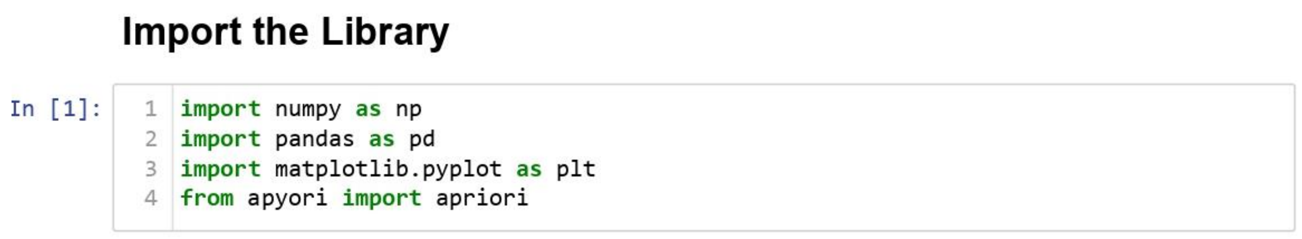
would have low confidence as well. As before, lower level candidate item rules can be pruned using the apriori algorithm, so that fewer candidate rules need to be examined

## ***IMPLEMENTATION AND OUTPUT:***



### Dataset Description

- Different products given 7500 transactions over the course of a week at a French retail store.
- We have library(**apyori**) to calculate the association rule using Apriori.



### Import the Library

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from apyori import apriori
```

## Read data and Display

In [2]:
```
store_data = pd.read_csv("store_data.csv", header=None)
display(store_data.head())
print(store_data.shape)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | shrimp | almonds | avocado | vegetables mix | green grapes | whole weat flour | yams | cottage cheese | energy drink | tomato juice | low fat yogurt | green tea | hone |
| **1** | burgers | meatballs | eggs | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | Nal |
| **2** | chutney | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | Nal |
| **3** | turkey | avocado | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | Nal |
| **4** | mineral water | milk | energy bar | whole wheat rice | green tea | NaN | NaN | NaN | NaN | NaN | NaN | NaN | Nal |

```
(7501, 20)
```

## Preprocessing on Data

- Here we need a data in form of list for Apriori Algorithm.

In [3]:
```
records = []
for i in range(1, 7501):
    records.append([str(store_data.values[i, j]) for j in range(0, 20)])
```

In [4]:
```
print(type(records))
```

```
<class 'list'>
```

## Apriori Algorithm

- Now time to apply algorithm on data.
- We have provide `min_support` , `min_confidence` , `min_lift` , and `min length` of sample-set for find rule.

In [9]:
```
association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_li
association_results = list(association_rules)
```

## How many relation derived

In [10]:
```
print("There are {} Relation derived.".format(len(association_results)))
```

```
There are 48 Relation derived.
```

### Association Rules Derived

```
In [11]: for i in range(0, len(association_results)):
             print(association_results[i][0])
```

```
frozenset({'chicken', 'light cream'})
frozenset({'mushroom cream sauce', 'escalope'})
frozenset({'escalope', 'pasta'})
frozenset({'herb & pepper', 'ground beef'})
frozenset({'tomato sauce', 'ground beef'})
frozenset({'olive oil', 'whole wheat pasta'})
frozenset({'shrimp', 'pasta'})
frozenset({'nan', 'chicken', 'light cream'})
frozenset({'shrimp', 'chocolate', 'frozen vegetables'})
frozenset({'spaghetti', 'cooking oil', 'ground beef'})
frozenset({'nan', 'mushroom cream sauce', 'escalope'})
frozenset({'nan', 'escalope', 'pasta'})
frozenset({'spaghetti', 'frozen vegetables', 'ground beef'})
frozenset({'olive oil', 'milk', 'frozen vegetables'})
frozenset({'shrimp', 'mineral water', 'frozen vegetables'})
frozenset({'olive oil', 'spaghetti', 'frozen vegetables'})
frozenset({'shrimp', 'spaghetti', 'frozen vegetables'})
frozenset({'spaghetti', 'tomatoes', 'frozen vegetables'})
frozenset({'spaghetti', 'ground beef', 'grated cheese'})
frozenset({'mineral water', 'herb & pepper', 'ground beef'})
frozenset({'nan', 'herb & pepper', 'ground beef'})
frozenset({'spaghetti', 'herb & pepper', 'ground beef'})
frozenset({'olive oil', 'milk', 'ground beef'})
frozenset({'nan', 'tomato sauce', 'ground beef'})
frozenset({'shrimp', 'spaghetti', 'ground beef'})
frozenset({'olive oil', 'milk', 'spaghetti'})
frozenset({'olive oil', 'mineral water', 'soup'})
frozenset({'nan', 'whole wheat pasta', 'olive oil'})
frozenset({'nan', 'shrimp', 'pasta'})
frozenset({'olive oil', 'spaghetti', 'pancakes'})
frozenset({'nan', 'shrimp', 'chocolate', 'frozen vegetables'})
frozenset({'nan', 'spaghetti', 'cooking oil', 'ground beef'})
frozenset({'nan', 'spaghetti', 'frozen vegetables', 'ground beef'})
frozenset({'spaghetti', 'milk', 'mineral water', 'frozen vegetables'})
frozenset({'nan', 'milk', 'frozen vegetables', 'olive oil'})
frozenset({'nan', 'shrimp', 'mineral water', 'frozen vegetables'})
frozenset({'nan', 'spaghetti', 'frozen vegetables', 'olive oil'})
frozenset({'nan', 'shrimp', 'spaghetti', 'frozen vegetables'})
frozenset({'nan', 'spaghetti', 'tomatoes', 'frozen vegetables'})
frozenset({'nan', 'spaghetti', 'ground beef', 'grated cheese'})
frozenset({'nan', 'mineral water', 'herb & pepper', 'ground beef'})
frozenset({'nan', 'spaghetti', 'herb & pepper', 'ground beef'})
frozenset({'nan', 'milk', 'olive oil', 'ground beef'})
frozenset({'nan', 'shrimp', 'spaghetti', 'ground beef'})
frozenset({'nan', 'milk', 'spaghetti', 'olive oil'})
frozenset({'nan', 'mineral water', 'olive oil', 'soup'})
frozenset({'nan', 'spaghetti', 'pancakes', 'olive oil'})
frozenset({'nan', 'mineral water', 'spaghetti', 'milk', 'frozen vegetables'})
```

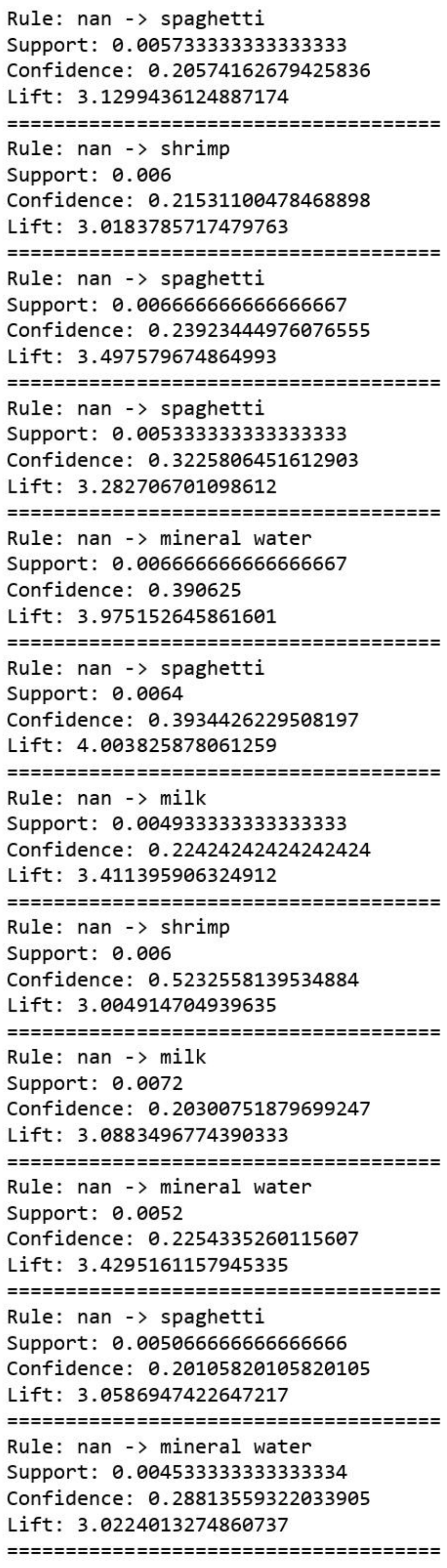
## Rules Generated

```
In [12]:
for item in association_results:
    # first index of the inner list
    # Contains base item and add item
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])

    # second index of the inner list
    print("Support: " + str(item[1]))

    # third index of the list located at 0th
    # of the third index of the inner list

    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=====================================")
```

```
Rule: chicken -> light cream
Support: 0.004533333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
=====================================
Rule: mushroom cream sauce -> escalope
Support: 0.005733333333333333
Confidence: 0.3006993006993007
Lift: 3.7903273197390845
=====================================
Rule: escalope -> pasta
Support: 0.005866666666666667
Confidence: 0.37288135593220345
Lift: 4.700185158809287
=====================================
Rule: herb & pepper -> ground beef
Support: 0.016
Confidence: 0.3234501347708895
Lift: 3.2915549671393096
=====================================
Rule: tomato sauce -> ground beef
Support: 0.005333333333333333
Confidence: 0.37735849056603776
Lift: 3.840147461662528
=====================================
Rule: olive oil -> whole wheat pasta
Support: 0.008
Confidence: 0.2714932126696833
Lift: 4.130221288078346
=====================================
Rule: shrimp -> pasta
Support: 0.005066666666666666
Confidence: 0.3220338983050848
Lift: 4.514493901473151
=====================================
Rule: nan -> chicken
Support: 0.004533333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
=====================================
Rule: shrimp -> chocolate
Support: 0.005333333333333333
Confidence: 0.23255813953488372
Lift: 3.260160834601174
=====================================
Rule: spaghetti -> cooking oil
Support: 0.0048
Confidence: 0.5714285714285714
```

```
Lift: 3.281557646029315
=====================================
Rule: nan -> mushroom cream sauce
Support: 0.005733333333333333
Confidence: 0.30069930069930073
Lift: 3.7903273197390845
=====================================
Rule: nan -> escalope
Support: 0.005866666666666667
Confidence: 0.37288135593220345
Lift: 4.700185158809287
=====================================
Rule: spaghetti -> frozen vegetables
Support: 0.008666666666666666
Confidence: 0.3110047846889952
Lift: 3.164906221394116
=====================================
Rule: olive oil -> milk
Support: 0.0048
Confidence: 0.20338983050847456
Lift: 3.094165778526489
=====================================
Rule: shrimp -> mineral water
Support: 0.0072
Confidence: 0.3068181818181818
Lift: 3.2183725365543547
=====================================
Rule: olive oil -> spaghetti
Support: 0.005733333333333333
Confidence: 0.20574162679425836
Lift: 3.1299436124887174
=====================================
Rule: shrimp -> spaghetti
Support: 0.006
Confidence: 0.21531100478468898
Lift: 3.0183785717479763
=====================================
Rule: spaghetti -> tomatoes
Support: 0.006666666666666667
Confidence: 0.23923444976076555
Lift: 3.497579674864993
=====================================
Rule: spaghetti -> ground beef
Support: 0.005333333333333333
Confidence: 0.3225806451612903
Lift: 3.282706701098612
=====================================
Rule: mineral water -> herb & pepper
Support: 0.006666666666666667
Confidence: 0.390625
Lift: 3.975152645861601
=====================================
Rule: nan -> herb & pepper
Support: 0.016
Confidence: 0.3234501347708895
Lift: 3.2915549671393096
=====================================
Rule: spaghetti -> herb & pepper
Support: 0.0064
Confidence: 0.3934426229508197
Lift: 4.003825878061259
=====================================
Rule: olive oil -> milk
Support: 0.004933333333333333
Confidence: 0.22424242424242424
Lift: 3.411395906324912
```

```
=====================================
Rule: nan -> tomato sauce
Support: 0.005333333333333333
Confidence: 0.37735849056603776
Lift: 3.840147461662528
=====================================
Rule: shrimp -> spaghetti
Support: 0.006
Confidence: 0.5232558139534884
Lift: 3.004914704939635
=====================================
Rule: olive oil -> milk
Support: 0.0072
Confidence: 0.20300751879699247
Lift: 3.0883496774390333
=====================================
Rule: olive oil -> mineral water
Support: 0.0052
Confidence: 0.2254335260115607
Lift: 3.4295161157945335
=====================================
Rule: nan -> whole wheat pasta
Support: 0.008
Confidence: 0.2714932126696833
Lift: 4.130221288078346
=====================================
Rule: nan -> shrimp
Support: 0.005066666666666666
Confidence: 0.3220338983050848
Lift: 4.514493901473151
=====================================
Rule: olive oil -> spaghetti
Support: 0.005066666666666666
Confidence: 0.20105820105820105
Lift: 3.0586947422647217
=====================================
Rule: nan -> shrimp
Support: 0.005333333333333333
Confidence: 0.23255813953488372
Lift: 3.260160834601174
=====================================
Rule: nan -> spaghetti
Support: 0.0048
Confidence: 0.5714285714285714
Lift: 3.281557646029315
=====================================
Rule: nan -> spaghetti
Support: 0.008666666666666666
Confidence: 0.3110047846889952
Lift: 3.164906221394116
=====================================
Rule: spaghetti -> milk
Support: 0.004533333333333334
Confidence: 0.28813559322033905
Lift: 3.0224013274860737
=====================================
Rule: nan -> milk
Support: 0.0048
Confidence: 0.20338983050847456
Lift: 3.094165778526489
=====================================
Rule: nan -> shrimp
Support: 0.0072
Confidence: 0.3068181818181818
Lift: 3.2183725365543547
=====================================
```



```
Rule: nan -> spaghetti
Support: 0.005733333333333333
Confidence: 0.20574162679425836
Lift: 3.1299436124887174
====================================
Rule: nan -> shrimp
Support: 0.006
Confidence: 0.21531100478468898
Lift: 3.0183785717479763
====================================
Rule: nan -> spaghetti
Support: 0.006666666666666667
Confidence: 0.23923444976076555
Lift: 3.497579674864993
====================================
Rule: nan -> spaghetti
Support: 0.005333333333333333
Confidence: 0.3225806451612903
Lift: 3.282706701098612
====================================
Rule: nan -> mineral water
Support: 0.006666666666666667
Confidence: 0.390625
Lift: 3.975152645861601
====================================
Rule: nan -> spaghetti
Support: 0.0064
Confidence: 0.3934426229508197
Lift: 4.003825878061259
====================================
Rule: nan -> milk
Support: 0.004933333333333333
Confidence: 0.22424242424242424
Lift: 3.411395906324912
====================================
Rule: nan -> shrimp
Support: 0.006
Confidence: 0.5232558139534884
Lift: 3.004914704939635
====================================
Rule: nan -> milk
Support: 0.0072
Confidence: 0.20300751879699247
Lift: 3.0883496774390333
====================================
Rule: nan -> mineral water
Support: 0.0052
Confidence: 0.2254335260115607
Lift: 3.4295161157945335
====================================
Rule: nan -> spaghetti
Support: 0.005066666666666666
Confidence: 0.20105820105820105
Lift: 3.0586947422647217
====================================
Rule: nan -> mineral water
Support: 0.004533333333333334
Confidence: 0.28813559322033905
Lift: 3.0224013274860737
====================================
```

***CONCLUSION:*** Thus, from this experiment I executed frequent pattern recognition using association rule mining using apriori algorithm to identify association rules of market based data analysis for various measure.