In [1]:
```
'''
    Author: A.Shrikant
'''
```

Out[1]:
```
'\n    Author: A.Shrikant\n'
```

Market Basket Analysis:

Association Rule Mining is a subset of **Data Mining**. In **Data Mining** we identify patterns from large datasets using **Statistics**, **Machine Learning** and **Database Systems**.

When **Association Rule Mining** is used in **Sales and Marketing** to analyze the relationship b/w products or itemsets, the kind of items that customers are frequently buying together and the frequency of itemsets over a period of time such an analysis is known as **Market Basket Analysis**.

Market Basket Analysis helps businesses identify the **purchase patterns** to a great extent and the **buying behavior** to some extent.

Purchase pattern involves analyzing the kind of products/services customers are purchasing, the frequency with which the products or services are being purchased and the time at which the purchases are made.

Buying behavior constitutes the decision-making process a customer goes through before making a purchasing choice. This decision-making is governed by various factors like psychological factors(needs, customer's perception of a product, customer's attitude and belief etc.), economic factors, social factors, etc.

Terminologies in **Association Rule Mining**:

1. **Itemset**: A collection of one or more items.
2. **Association Rule**: It is of the form **X => Y** i.e. X implies Y. Both X and Y are itemsets.
3. **Antecedent**: The itemset that appears to the left of the =>(implies) sign in the association rule.
4. **Consequent**: The itemset that appears to the right of the =>(implies) sign in the association rule.

Metrics used to determine the quality or effectiveness or strength of **Association Rules**:

1. **Support**: Support of an itemset X is defined as the number of times X occurred in the list of all transactions and is denoted as **support(X)**.

Similarly, support for an association rule X => Y is defined as the number of times $X \cup Y$ occurred in the list of all transactions and is denoted as **support(X => Y)** or $support(X \cup Y)$.

The range for support(X) is [0,1].

Interpretation of support(X): support(X) = P(X) = **Probability of finding X in a transaction.**

Interpretation of support(X => Y): support(X => Y) = $support(X \cup Y)$ = $P(X \cap Y)$ = **Probability of finding both X and Y in a transaction.**

2. **Confidence**: Confidence of an association rule **X => Y** is defined as the proportion of transactions that contain **Y(consequent)** among all the transactions that contain **X(antecedent)**.

$$\text{i.e. } confidence(X => Y) = \frac{support(X \cup Y)}{support(X)}$$

The range for confidence(X => Y) is [0,1].

Interpretation of confidence(X => Y): confidence(X => Y) = P(Y|X) = how likely is it that Y will occur in a transaction given that X has already occurred in that transaction.

3. **Lift**: Lift of an association rule **X => Y** is defined as the relative frequency with which X and Y occur together in comparison to the case if X and Y were independent.

$$\text{i.e. } lift(X => Y) = \frac{support(X \cup Y)}{support(X) \times support(Y)}$$

The range for lift(X => Y) is [0, infinity).

Interpretation of lift(X => Y): lift(X => Y) tells how much more likely is it that X and Y will occur together in a transaction w.r.t. X and Y occurring together in a transaction when they were independent.

lift(X => Y) tends to infinity when P(X) and P(Y) tends to 0 and $P(X \cap Y) = min(P(X), P(Y))$

4. **Leverage**: Leverage of an association rule **X => Y** is defined as:

$$leverage(X => Y) = support(X \cup Y) - support(X) \times support(Y)$$

The range for leverage(X => Y) is [-0.25, 0.25].

Interpretation of leverage(X => Y): leverage(X => Y) value greater than 0 indicates positive association i.e. **probability of(occurrence of Y in a transaction given X is already present)** is greater than the **probability of(occurrence of Y in a transaction)**.

Similarly, leverage(X => Y) value less than 0 indicates negative association i.e. **probability of(occurrence of Y in a transaction given X is already present)** is lower than the **probability of(occurrence of Y in a transaction)**.

leverage(X => Y) = 0 means X and Y are statistically independent i.e. $P(X \cap Y) = P(X) \times P(Y)$

leverage(X => Y) = 0.25 when $P(X \cap Y) = 0.5$, $P(X) = 0.5$ and $P(Y) = 0.5$

leverage(X => Y) = -0.25 when $P(X \cap Y) = 0$, $P(X) = 0.5$ and $P(Y) = 0.5$

5. **Conviction**: Conviction of an association rule **X => Y** is defined as:

$$conviction(X => Y) = \frac{1 - support(Y)}{1 - confidence(X => Y)}$$

The range for conviction(X => Y) is [0, infinity).

Interpretation of conviction(X => Y): conviction(X => Y) values is an indication of how much the consequent Y is dependent on the antecedent X.

conviction(X => Y) = 0, means that the association rule is incorrect because conviction(X => 0) = 0 means the support(Y) = 1 which means Y is always there in all the transactions so the occurrence or absence of X has almost no effect at all on the probability of occurrence of Y.

conviction(X => Y) < 1, means that association rule has negative association i.e. P(Y|X) < P(Y).

conviction(X => Y) = 1, means that X and Y are statistically independent of each other i.e. P(Y|X) = P(Y) and vice versa i.e. occurrence or absence of X does not affect the probability of occurrence of Y.

conviction(X => Y) > 1, means that association rule has positive association i.e. P(Y|X) > P(Y).

conviction(X => Y) = infinity, means that consequent(Y) is totally dependent on antecedent(X) i.e. whenever X occurs in a transaction then Y will also surely occur in that transaction. So the association rule becomes deterministic.

6. **zhangs metric**: zhangs metric of an association rule **X => Y** is defined as:

$$\text{zhangs metric}(X => Y) = \frac{confidence(X => Y) - confidence(X' => Y)}{max(confidence(X => Y), confidence(X' => Y))}$$

The range for zhangs metric(X => Y) is [-1, 1].

Interpretation of zhangs metric(X => Y):

zhangs metric(X => Y) > 0 indicates positive association i.e. P(Y|X) > P(Y) which says knowing X provides more information about Y than not knowing X.

zhangs metric(X => Y) < 0 indicates negative association i.e. P(Y|X) < P(Y) which says knowing X provides less information about Y than not knowing X.

zhangs metric(X => Y) = 1 indicates a complete dependence of Y on X i.e. P(Y|X) = 1 and P(Y|X') = 0.

zhangs metric(X => Y) = -1 indicates a complete dissociation or complete dependence of Y on X' i.e. P(Y|X) = 0 and P(Y|X') = 1.

zhangs metric(X => Y) = 0 indicates neither association nor dissociation i.e. P(Y|X) = P(Y|X') which says the knowledge of the occurrence of X provides the same amount of information about Y as the knowledge of the absence of X provides about Y.

Illustration of the Association rule metrics:

Let X = {biscuits} = itemset containing only biscuits and Y = {chocolate} = itemset containing only chocolate.

Let #transactions = N = 5000.

Let X occur in 500 transactions, Y occurs 800 times and $X \cup Y$ occur 200 times.

Consider the association rule X => Y.

support(X) = 500 / 5000 = **0.1**

support(Y) = 800 / 5000 = **0.16**

support($X \cup Y$) = 200 / 5000 = **0.04**

confidence(X => Y) = P(Y|X) = $\frac{P(X \cap Y)}{P(X)}$ = $\frac{support(X \cup Y)}{support(X)}$ = 0.04 / 0.1 = **0.4**

lift(X => Y) = $\frac{support(X \cup Y)}{support(X) support(Y)}$ = 0.04 / (0.1*0.16) = **2.5**

leverage(X => Y) = $support(X \cup Y) - support(X) support(Y)$ = 0.04 - (0.1*0.16) = **0.024**

conviction(X => Y) = $\frac{1 - support(Y)}{1 - confidence(X => Y)}$ = (1-0.16) / (1-0.4) = **1.4**

confidence(X' => Y) = P(Y|X') = $\frac{P(X' \cap Y)}{P(X')}$ = $\frac{support(X' \cup Y)}{support(X')}$ = $\frac{(support(Y) - support(X \cup Y))}{(1 - support(X))}$ = (0.16-0.04) / (1-0.1) = **0.1333**

zhangs metric(X => Y) = $\frac{confidence(X => Y) - confidence(X' => Y)}{max(confidence(X => Y), confidence(X' => Y))}$ = (0.4 - 0.1333) / max(0.4, 0.1333) = **0.6668**

From the confidence(X => Y) we can see that P(chocolate | biscuits) > P(chocolate)

From the lift(X => Y) > 1 we can see that the observed frequency of {chocolate, biscuits} occurring together is more probable than if they were independent.

leverage(X => Y) > 0 indicating a positive association for the rule.

conviction(X => Y) > 1 indicating that the probability of the occurrence of 'chocolate' in a transaction depends on the occurrence of 'biscuits' in that transaction.

zhangs metric(X => Y) is > 0 and close to 1 indicating that strong positive association for the rule X => Y.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv('dataset/Market_Basket_Optimisation.csv', header=None)
```

In [4]:
```
df.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 bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [5]:
```
df.shape
```

Out[5]: (7501, 20)

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       7501 non-null   object
 1   1       5747 non-null   object
 2   2       4389 non-null   object
 3   3       3345 non-null   object
 4   4       2529 non-null   object
 5   5       1864 non-null   object
 6   6       1369 non-null   object
 7   7       981 non-null    object
 8   8       654 non-null    object
 9   9       395 non-null    object
 10  10      256 non-null    object
 11  11      154 non-null    object
 12  12      87 non-null     object
 13  13      47 non-null     object
 14  14      25 non-null     object
 15  15      8 non-null      object
 16  16      4 non-null      object
 17  17      4 non-null      object
 18  18      3 non-null      object
 19  19      1 non-null      object
dtypes: object(20)
memory usage: 1.1+ MB
```

In [7]:
```
df.iloc[0,0]
```

Out[7]:
```
'shrimp'
```

Identifying the unique items along with their item count from the list of all transactions:

In [8]:
```
''' Creating a list containing all the transactions items. '''
transaction_items_list = []

for i in range(df.shape[0]):
    for j in range(df.shape[1]):
        transaction_items_list.append(df.iloc[i, j])
```

In [9]:
```python
transaction_items_list
```

Out[9]:
```
['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',
```

In [10]:
```python
''' Creating a dataframe containing all the transactions items. '''

df_all_items = pd.DataFrame(transaction_items_list, columns=["item name"])
```

In [11]:
```
df_all_items
```

Out[11]:

	item name
0	shrimp
1	almonds
2	avocado
3	vegetables mix
4	green grapes
...	...
150015	NaN
150016	NaN
150017	NaN
150018	NaN
150019	NaN

150020 rows × 1 columns

In [12]:
```
df_all_items["incident_count"] = 1
```

In [13]:
```
df_all_items
```

Out[13]:

	item name	incident_count
0	shrimp	1
1	almonds	1
2	avocado	1
3	vegetables mix	1
4	green grapes	1
...	...	...
150015	NaN	1
150016	NaN	1
150017	NaN	1
150018	NaN	1
150019	NaN	1

150020 rows × 2 columns

In [14]:
```
df_all_items.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150020 entries, 0 to 150019
Data columns (total 2 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   item name       29363 non-null   object
 1   incident_count  150020 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 2.3+ MB
```

In [15]:
```
df_all_items.dropna(inplace=True)
```

In [16]:
```
df_all_items.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29363 entries, 0 to 150003
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   item name       29363 non-null  object
 1   incident_count  29363 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 688.2+ KB
```

In [17]:
```
df_items_grouped = df_all_items.groupby('item name').sum()
df_items_grouped
```

Out[17]:

	incident_count
item name	
asparagus	1
almonds	153
antioxydant juice	67
asparagus	35
avocado	250
...	...
whole wheat pasta	221
whole wheat rice	439
yams	86
yogurt cake	205
zucchini	71

120 rows × 1 columns

In [18]:
```
# Setting the default row index in the dataframe 'df_unique_items' and making the row index of the
# dataframe 'df_items_grouped' as a column in 'df_unique_items'.

df_unique_items = df_items_grouped.sort_values('incident_count', ascending=False).reset_index()
df_unique_items
```

Out[18]:

	item name	incident_count
0	mineral water	1788
1	eggs	1348
2	spaghetti	1306
3	french fries	1282
4	chocolate	1230
...	...	...
115	bramble	14
116	cream	7
117	napkins	5
118	water spray	3
119	asparagus	1

120 rows × 2 columns

In [19]:
```
df_unique_items['item name'].values
```

Out[19]:
```
array(['mineral water', 'eggs', 'spaghetti', 'french fries', 'chocolate',
       'green tea', 'milk', 'ground beef', 'frozen vegetables',
       'pancakes', 'burgers', 'cake', 'cookies', 'escalope',
       'low fat yogurt', 'shrimp', 'tomatoes', 'olive oil',
       'frozen smoothie', 'turkey', 'chicken', 'whole wheat rice',
       'grated cheese', 'cooking oil', 'soup', 'herb & pepper', 'honey',
       'champagne', 'fresh bread', 'salmon', 'brownies', 'avocado',
       'hot dogs', 'cottage cheese', 'tomato juice', 'butter',
       'whole wheat pasta', 'red wine', 'yogurt cake', 'light mayo',
       'ham', 'energy bar', 'energy drink', 'pepper', 'vegetables mix',
       'cereals', 'muffins', 'oil', 'french wine', 'fresh tuna',
       'strawberries', 'meatballs', 'almonds', 'parmesan cheese',
       'mushroom cream sauce', 'rice', 'protein bar', 'mint',
       'white wine', 'pasta', 'light cream', 'carrots', 'black tea',
       'tomato sauce', 'fromage blanc', 'gums', 'eggplant', 'melons',
       'extra dark chocolate', 'body spray', 'yams', 'magazines',
       'barbecue sauce', 'cider', 'nonfat milk', 'candy bars', 'zucchini',
       'whole weat flour', 'salt', 'blueberries', 'flax seed',
       'green grapes', 'antioxydant juice', 'bacon', 'bug spray',
       'green beans', 'clothes accessories', 'toothpaste',
       'strong cheese', 'shallot', 'spinach', 'gluten free bar',
       'pet food', 'sparkling water', 'soda', 'mayonnaise', 'chili',
       'pickles', 'burger sauce', 'mint green tea', 'hand protein bar',
       'salad', 'shampoo', 'cauliflower', 'corn', 'asparagus', 'sandwich',
       'babies food', 'dessert wine', 'ketchup', 'oatmeal',
       'chocolate bread', 'chutney', 'mashed potato', 'tea', 'bramble',
       'cream', 'napkins', 'water spray', ' asparagus'], dtype=object)
```

In [20]:
```
df_unique_items['item name'].values.size
```

Out[20]:
```
120
```

Visualizing the frequently occurring items found from the list of all transactions:

In [21]:
```
import plotly.express as px
```

```
In [22]: ''' Showing only the Top 36 frequently occurring items using a Tree Map so that the Tree Map is
readable. '''

# Some of the values for the keyword argument 'color_continuous_scale' are:
# 'blues', 'greeens', 'reds', 'plasma', 'fall', 'magenta', 'thermal'

figure = px.treemap(df_unique_items.head(36),
                    path=[px.Constant('all items'), 'item name'],
                    values='incident_count',
                    color=df_unique_items["incident_count"].head(36),
                    color_continuous_scale="plasma",
                    hover_data = {'item name':True},
                    height = 500,
                    width = 790)
figure.show()
```

Pre-processing the transactions data:

In [23]:
```
''' Creating a list containing all the transactions and also filtering out the NaN values from each
transaction. '''

transactions_list = []

for i in range(df.shape[0]):
    transaction = [df.iloc[i,j] for j in range(df.shape[1])
                   if not(isinstance(df.iloc[i,j], float) and np.isnan(df.iloc[i,j]))]
    transactions_list.append(transaction)
```

In [24]:
```
transactions_list
```

Out[24]:
```
[['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',
```

In [25]:
```
''' Creating an array containing all the transactions. '''

transactions_array = np.array(transactions_list, dtype="object")
transactions_array
```

Out[25]:
```
array([list(['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes', 'whole weat flour', 'yams', 'cottage cheese', 'energy drink', 'tom
ato juice', 'low fat yogurt', 'green tea', 'honey', 'salad', 'mineral water', 'salmon', 'antioxydant juice', 'frozen smoothie', 'spinach', 'olive
oil']),
       list(['burgers', 'meatballs', 'eggs']), list(['chutney']), ...,
       list(['chicken']), list(['escalope', 'green tea']),
       list(['eggs', 'frozen smoothie', 'yogurt cake', 'low fat yogurt'])],
      dtype=object)
```

In [26]:
```
# !pip install mlxtend
```

In [27]:
```
from mlxtend.preprocessing import TransactionEncoder
```

In [28]:
```
transaction_encoder = TransactionEncoder()
```

In [29]:
```
''' Picking up the unique items from the 'transactions_array' and creating a OHE(One Hot Encoded)
boolean array out of it. '''

# transaction_encoder.fit_transform(transactions_array) is equivalent to
# transaction_encoder.fit(transactions_array).transaction_encoder.transform(transactions_array)
ohe_transactions_array = transaction_encoder.fit_transform(transactions_array)
ohe_transactions_array
```

Out[29]:
```
array([[False,  True,  True, ...,  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,  True, False]])
```

In [30]:
```
arr = transactions_array[1]
'nan' in arr
```

Out[30]:
```
False
```

In [31]:
```python
df_ohe_items_bool = pd.DataFrame(ohe_transactions_array, columns=transaction_encoder.columns_)
df_ohe_items_bool
```

Out[31]:

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	...	turkey	vegetables mix	water spray	white wine	whole weat flour	whole wheat pasta	whole wheat rice	yams	yogurt cake
0	False	True	True	False	True	False	False	False	False	False	...	False	True	False	False	True	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
3	False	False	False	False	True	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7496	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
7497	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
7498	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
7499	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
7500	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True

7501 rows × 120 columns

In [32]:
```python
top_70_item_names = df_unique_items['item name'].head(70)
top_70_item_names
```

Out[32]:
```
0           mineral water
1                    eggs
2               spaghetti
3            french fries
4               chocolate
              ...
65                   gums
66               eggplant
67                 melons
68    extra dark chocolate
69             body spray
Name: item name, Length: 70, dtype: object
```

In [33]:
```
df_ohe_top_70_items = df_ohe_items_bool.loc[:, top_70_item_names]
df_ohe_top_70_items
```

Out[33]:

	mineral water	eggs	spaghetti	french fries	chocolate	green tea	milk	ground beef	frozen vegetables	pancakes	...	light cream	carrots	black tea	tomato sauce	fromage blanc	gums	eggplant	melons	extra dark chocolate	bo spr
0	True	False	False	False	False	True	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
1	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
4	True	False	False	False	False	True	True	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7496	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
7497	False	True	False	True	False	True	False	False	True	False	...	False	False	False	False	False	False	False	False	False	Fal
7498	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
7499	False	False	False	False	False	True	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
7500	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal

7501 rows × 70 columns

Number of possible association rules:

Suppose from the transactions list we get d unique items. From this, we want to generate the association rules of the form(X => Y where X and Y are itemsets). For this, we need to create non-empty binary(two) partitions which we call Partition 1 and Partition 2 out of these d items. Partition 1 will be our antecedent itemset and Partition 2 the consequent itemset.

Let the Partition 1 be of size k i.e. it contains k items, then the Partition 2 will contain only d-k items.

ways Partition 1 can be created = $\binom{d}{k}$

ways Partition 2 can be created = # ways we can choose 1 from (d-k) + # ways we can choose 2 from (d-k) + ... + # ways we can choose (d-k) from (d-k)

$$= \sum_{j=1}^{d-k} \binom{d-k}{j}$$

ways in which both Partition 1 and Partition 2 can be created is:

$$\binom{d}{k} \times \sum_{j=1}^{d-k} \times \binom{d-k}{j}$$

But k can also vary from 1 to d-1 so the total #association rules possible out of d unique items:

$$\sum_{k=1}^{d-1}\left(\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right)$$

$$= 3^d - 2^{d+1} + 1$$

Apriori Principle:

The Apriori Algorithm is based on the Apriori principle. The **Apriori principle states that if an itemset is frequent then all its subsets are also frequent.**

This is because let Y be a frequent itemset in our list of transactions and X be any arbitrary subset of Y i.e. $X \subseteq Y$ then $support(X) \geq support(Y)$ because X can appear in other transactions where Y is not there.

Apriori Algorithm:

- Select a threshold for support of an itemset.
- Start with k = 1
- Generate frequent itemsets of size 1.
- Repeat until no more frequent itemsets are there:
  - Generate candidate itemsets of size (k+1) from the size k frequent itemsets.
  - Prune the size (k+1) itemsets containing subsets of size k that are infrequent.
  - Calculate the support for each of the remaining size (k+1) candidate itemsets.
  - Eliminate the size (k+1) candidate itemsets that are infrequent.

Generating frequently occurring itemsets using Apriori algorithm:

In [34]:
```
from mlxtend.frequent_patterns import apriori
```

In [35]:
```
df_apriori_res = apriori(df_ohe_top_70_items, min_support=0.01, use_colnames=True)
df_apriori_res
```

Out[35]:

	support	itemsets
0	0.238368	(mineral water)
1	0.179709	(eggs)
2	0.174110	(spaghetti)
3	0.170911	(french fries)
4	0.163845	(chocolate)
...	...	...
247	0.010932	(mineral water, chocolate, ground beef)
248	0.011065	(mineral water, ground beef, milk)
249	0.011065	(mineral water, frozen vegetables, milk)
250	0.010532	(chocolate, eggs, spaghetti)
251	0.010932	(chocolate, milk, spaghetti)

252 rows × 2 columns

In [36]:
```
df_apriori_res['itemsets'][251]
```

Out[36]:
```
frozenset({'chocolate', 'milk', 'spaghetti'})
```

In [37]:
```
len(df_apriori_res['itemsets'][251])
```

Out[37]:
```
3
```

In [38]:
```
df_apriori_res['itemset_length'] = df_apriori_res['itemsets'].apply(lambda el: len(el))
df_apriori_res
```

Out[38]:

	support	itemsets	itemset_length
0	0.238368	(mineral water)	1
1	0.179709	(eggs)	1
2	0.174110	(spaghetti)	1
3	0.170911	(french fries)	1
4	0.163845	(chocolate)	1
...	...	...	...
247	0.010932	(mineral water, chocolate, ground beef)	3
248	0.011065	(mineral water, ground beef, milk)	3
249	0.011065	(mineral water, frozen vegetables, milk)	3
250	0.010532	(chocolate, eggs, spaghetti)	3
251	0.010932	(chocolate, milk, spaghetti)	3

252 rows × 3 columns

Extracting association rules using the frequently occurring itemsets:

In [39]:
```
from mlxtend.frequent_patterns import association_rules
```

In [40]:
```python
rules = association_rules(df_apriori_res, metric='confidence', min_threshold=0.25)
rules
```

Out[40]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(eggs)	(mineral water)	0.179709	0.238368	0.050927	0.283383	1.188845	0.008090	1.062815	0.193648
1	(mineral water)	(spaghetti)	0.238368	0.174110	0.059725	0.250559	1.439085	0.018223	1.102008	0.400606
2	(spaghetti)	(mineral water)	0.174110	0.238368	0.059725	0.343032	1.439085	0.018223	1.159314	0.369437
3	(chocolate)	(mineral water)	0.163845	0.238368	0.052660	0.321400	1.348332	0.013604	1.122357	0.308965
4	(milk)	(mineral water)	0.129583	0.238368	0.047994	0.370370	1.553774	0.017105	1.209650	0.409465
...	...	...	...	...	...	...	...	...	...	...
90	(chocolate, spaghetti)	(eggs)	0.039195	0.179709	0.010532	0.268707	1.495234	0.003488	1.121700	0.344719
91	(eggs, spaghetti)	(chocolate)	0.036528	0.163845	0.010532	0.288321	1.759721	0.004547	1.174905	0.448096
92	(chocolate, milk)	(spaghetti)	0.032129	0.174110	0.010932	0.340249	1.954217	0.005338	1.251821	0.504495
93	(chocolate, spaghetti)	(milk)	0.039195	0.129583	0.010932	0.278912	2.152382	0.005853	1.207088	0.557239
94	(milk, spaghetti)	(chocolate)	0.035462	0.163845	0.010932	0.308271	1.881480	0.005122	1.208790	0.485728

95 rows × 10 columns

In [41]:
```python
# Sorting the rules based on the 'confidence' column.
rules.sort_values(by=['confidence'], ignore_index=True, inplace=True, ascending=False)
```

In [42]:
```
rules.head(20)
```

Out[42]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(ground beef, eggs)	(mineral water)	0.019997	0.238368	0.010132	0.506667	2.125563	0.005365	1.543848	0.540342
1	(ground beef, milk)	(mineral water)	0.021997	0.238368	0.011065	0.503030	2.110308	0.005822	1.532552	0.537969
2	(chocolate, ground beef)	(mineral water)	0.023064	0.238368	0.010932	0.473988	1.988472	0.005434	1.447937	0.508837
3	(frozen vegetables, milk)	(mineral water)	0.023597	0.238368	0.011065	0.468927	1.967236	0.005440	1.434136	0.503555
4	(soup)	(mineral water)	0.050527	0.238368	0.023064	0.456464	1.914955	0.011020	1.401255	0.503221
5	(pancakes, spaghetti)	(mineral water)	0.025197	0.238368	0.011465	0.455026	1.908923	0.005459	1.397557	0.488452
6	(olive oil, spaghetti)	(mineral water)	0.022930	0.238368	0.010265	0.447674	1.878079	0.004799	1.378954	0.478514
7	(milk, spaghetti)	(mineral water)	0.035462	0.238368	0.015731	0.443609	1.861024	0.007278	1.368879	0.479672
8	(chocolate, milk)	(mineral water)	0.032129	0.238368	0.013998	0.435685	1.827780	0.006340	1.349656	0.467922
9	(ground beef, spaghetti)	(mineral water)	0.039195	0.238368	0.017064	0.435374	1.826477	0.007722	1.348914	0.470957
10	(frozen vegetables, spaghetti)	(mineral water)	0.027863	0.238368	0.011998	0.430622	1.806541	0.005357	1.337656	0.459252
11	(milk, eggs)	(mineral water)	0.030796	0.238368	0.013065	0.424242	1.779778	0.005724	1.322834	0.452053
12	(olive oil)	(mineral water)	0.065858	0.238368	0.027596	0.419028	1.757904	0.011898	1.310962	0.461536
13	(mineral water, ground beef)	(spaghetti)	0.040928	0.174110	0.017064	0.416938	2.394681	0.009938	1.416470	0.607262
14	(ground beef)	(mineral water)	0.098254	0.238368	0.040928	0.416554	1.747522	0.017507	1.305401	0.474369
15	(chocolate, eggs)	(mineral water)	0.033196	0.238368	0.013465	0.405622	1.701663	0.005552	1.281394	0.426498
16	(chocolate, spaghetti)	(mineral water)	0.039195	0.238368	0.015865	0.404762	1.698053	0.006522	1.279541	0.427860
17	(salmon)	(mineral water)	0.042528	0.238368	0.017064	0.401254	1.683336	0.006927	1.272045	0.423972
18	(cereals)	(mineral water)	0.025730	0.238368	0.010265	0.398964	1.673729	0.004132	1.267198	0.413162
19	(ground beef)	(spaghetti)	0.098254	0.174110	0.039195	0.398915	2.291162	0.022088	1.373997	0.624943

Doing a Sanity Check of the generated association rules:

In [43]:
```
''' Function to get the total occurrences of an itemset from all the transactions. '''
def get_itemset_occurrences(transactions_array, items):
    itemset_count = 0
    itemset = frozenset(items)
    print(f'itemset: {itemset}')

    for transaction in transactions_array:
        if itemset.issubset(transaction):
            itemset_count += 1
    return itemset_count
```

In [44]:
```
''' Function to get the support for an itemset '''
def get_support(transactions_array, items):
    items_count = get_itemset_occurrences(transactions_array, items)
    N = len(transactions_array)

    return items_count/N
```

In [45]:
```
''' Considering the association rule: eggs => mineral water '''
```

Out[45]: ' Considering the association rule: eggs => mineral water '

In [46]:
```
items = ['ground beef', 'eggs', 'mineral water']
rule_support = get_support(transactions_array, items)
rule_support
```

```
itemset: frozenset({'mineral water', 'ground beef', 'eggs'})
```

Out[46]: 0.010131982402346354

In [47]:
```
antecedent_items = ['ground beef', 'eggs']
antecedent_support = get_support(transactions_array, antecedent_items)
antecedent_support
```

```
itemset: frozenset({'ground beef', 'eggs'})
```

Out[47]: 0.019997333688841486

In [48]:
```
consequent_items = ['mineral water']

consequent_support =  get_support(transactions_array, consequent_items)
consequent_support
```

```
itemset: frozenset({'mineral water'})
```

Out[48]: 0.23836821757099053

In [49]:
```
rule_confidence = rule_support/antecedent_support
rule_confidence
```

Out[49]: 0.5066666666666667

In [50]:
```
rule_lift = rule_support/(antecedent_support * consequent_support)
rule_lift
```

Out[50]: 2.125563012677107

In [51]:
```
rule_leverage = rule_support - (antecedent_support * consequent_support)
rule_leverage
```

Out[51]: 0.005365253614764888

In [52]:
```
rule_conviction = (1-consequent_support)/(1-rule_confidence)
rule_conviction
```

Out[52]: 1.5438482076263709

In [53]:
```
confidence_not_ant_implies_con = (consequent_support - rule_support)/(1-antecedent_support)

rule_zhangs_metric = ((rule_confidence - confidence_not_ant_implies_con) /
max(rule_confidence, confidence_not_ant_implies_con))

rule_zhangs_metric
```

Out[53]: 0.5403418081320838

Conclusion:

For generating frequently occurring itemsets we have chosen the Top 70 items out of the 120 unique items found from the list of

all transactions.

Using these Top 70 items we generated the frequently occurring itemsets using the Apriori algorithm using min_support as 0.01 i.e. 1%.

For generating the association rules out of the generated frequently occurring itemsets we used the 'Confidence' metric with min_threshold as 0.25 i.e. 25%.

As a result, we obtained 95 association rules with the Top 5 listed below:

- (eggs, ground beef) => (mineral water) confidence: 0.5067, lift: 2.1256
- (ground beef, milk) => (mineral water) confidence: 0.5030, lift: 2.1103
- (chocolate, ground beef) => (mineral water) confidence: 0.4740, lift: 1.9885
- (milk, frozen vegetables) => (mineral water) confidence: 0.4690, lift: 1.9672
- (soup) => (mineral water) confidence: 0.4565, lift: 1.9150