In [1]:
```
import os
import math
import scipy
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sys
from itertools import combinations, groupby
from collections import Counter
from IPython.display import display
import mlxtend
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth
from sklearn.preprocessing import LabelBinarizer
```

1. Importing the dataset

In [2]:
```
products = pd.read_csv(r'C:\Users\Aparna Akula\Documents\
                         Data Mining\Project\products.csv')
display(products.head())
print(products.shape)
```

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13

```
(49688, 4)
```

In [3]:
```
aisles = pd.read_csv(r'C:\Users\Aparna Akula\Documents\
                       Data Mining\Project\aisles.csv')
display(aisles.head())
print(aisles.shape)
```

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation

```
(134, 2)
```

In [4]:
```
departments = pd.read_csv(r'C:\Users\Aparna Akula\Documents\
                       Data Mining\Project\departments.csv')
display(departments.head())
print(departments.shape)
```

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

```
(21, 2)
```

In [5]:
```
order_products__train = pd.read_csv(r'C:\Users\Aparna Akula\Documents\
                       Data Mining\Project\order_products__train.csv')
display(order_products__train.head())
print(order_products__train.shape)
```

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1

```
(1384617, 4)
```

In [6]:
```
order_products__prior = pd.read_csv(r'C:\Users\Aparna Akula\Documents\
                        Data Mining\Project\order_products__prior.csv')
display(order_products__prior.head())
print(order_products__prior.shape)
```

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

```
(32434489, 4)
```

In [7]:
```
orders = pd.read_csv(r'C:\Users\Aparna Akula\Documents\
                        Data Mining\Project\orders.csv')
display(orders.head())
print(orders.shape)
```

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior
0	2539329	1	prior	1	2	8	
1	2398795	1	prior	2	3	7	
2	473747	1	prior	3	3	12	
3	2254736	1	prior	4	4	7	
4	431534	1	prior	5	4	15	

```
(3421083, 7)
```

In [8]:
```
# joining orders and prior
train_and_prior = pd.concat([order_products__prior,
            order_products__train],ignore_index=True)
train_and_prior
```

Out[8]:

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0
...	...	...	...	...
33819101	3421063	14233	3	1
33819102	3421063	35548	4	1
33819103	3421070	35951	1	1
33819104	3421070	16953	2	1
33819105	3421070	4724	3	1

33819106 rows × 4 columns

2. Combining datasets

In [9]:
```
df1 = pd.merge(aisles, products, how='inner', on='aisle_id')
df2 = pd.merge(df1, departments, how='inner', on='department_id')
df3 = pd.merge(train_and_prior, df2, how='inner', on='product_id')
df4 = pd.merge(orders, df3, how='inner', on='order_id')
```

In [10]:
```
#df = df4.drop(['aisle_id','department_id','eval_set','order_number','product
df = df4.drop(['aisle_id','department_id','product_id'],axis=1)
display(df.head())
print('The Dataset consists of {} features & {} samples.'.format(df.shape[1],
```

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior
0	2539329	1	prior	1	2	8	
1	2539329	1	prior	1	2	8	
2	2539329	1	prior	1	2	8	
3	2539329	1	prior	1	2	8	
4	2539329	1	prior	1	2	8	

```
The Dataset consists of 12 features & 33819106 samples.
```

Exploratory Data Analysis

In [11]:
```
df.dtypes
```

Out[11]:
```
order_id                  int64
user_id                   int64
eval_set                 object
order_number              int64
order_dow                 int64
order_hour_of_day         int64
days_since_prior_order  float64
add_to_cart_order         int64
reordered                 int64
aisle                    object
product_name             object
department               object
dtype: object
```

In [12]:
```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

```
number of duplicate rows:  (0, 12)
```

In [13]:
```
df.count()
```

Out[13]:
```
order_id                  33819106
user_id                   33819106
eval_set                  33819106
order_number              33819106
order_dow                 33819106
order_hour_of_day         33819106
days_since_prior_order    31741038
add_to_cart_order         33819106
reordered                 33819106
aisle                     33819106
product_name              33819106
department                33819106
dtype: int64
```

In [14]:
```
print(df.isnull().sum())
```

```
order_id                        0
user_id                         0
eval_set                        0
order_number                    0
order_dow                       0
order_hour_of_day               0
days_since_prior_order    2078068
add_to_cart_order               0
reordered                       0
aisle                           0
product_name                    0
department                      0
dtype: int64
```

In [15]:
```
df_null = df.dropna()
df_null.count()
```

Out[15]:
```
order_id                  31741038
user_id                   31741038
eval_set                  31741038
order_number              31741038
order_dow                 31741038
order_hour_of_day         31741038
days_since_prior_order    31741038
add_to_cart_order         31741038
reordered                 31741038
aisle                     31741038
product_name              31741038
department                31741038
dtype: int64
```

Ref: https://towardsdatascience.com/exploratory-data-analysis-in-python-c9a77dfa39ce [(https://towardsdatascience.com/exploratory-data-analysis-in-python-c9a77dfa39ce)](https://towardsdatascience.com/exploratory-data-analysis-in-python-c9a77dfa39ce)

In [16]:
```
df.describe()
```

Out[16]:

	order_id	user_id	order_number	order_dow	order_hour_of_day	days_sin
count	3.381911e+07	3.381911e+07	3.381911e+07	3.381911e+07	3.381911e+07	
mean	1.710566e+06	1.029444e+05	1.713998e+01	2.737285e+00	1.343123e+01	
std	9.874008e+05	5.946733e+04	1.749829e+01	2.093296e+00	4.246149e+00	
min	1.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	
25%	8.554130e+05	5.143500e+04	5.000000e+00	1.000000e+00	1.000000e+01	
50%	1.710660e+06	1.026260e+05	1.100000e+01	3.000000e+00	1.300000e+01	
75%	2.565587e+06	1.544120e+05	2.400000e+01	5.000000e+00	1.600000e+01	
max	3.421083e+06	2.062090e+05	1.000000e+02	6.000000e+00	2.300000e+01	

In [17]:
```
a = len(df['order_id'].unique())
b = len(df['user_id'].unique())
print(a)
print(b)
```

```
3346083
206209
```

In [18]:
```
a = df['aisle'].unique()
len(a)
```

Out[18]: 134

In [19]:
```
a = df['product_name'].unique()
len(a)
```

Out[19]: 49685

In [20]:
```
a = df['department'].unique()
len(a)
print(a)
```

```
['beverages' 'dairy eggs' 'snacks' 'household' 'produce' 'breakfast'
 'pantry' 'deli' 'frozen' 'personal care' 'meat seafood' 'international'
 'bakery' 'canned goods' 'dry goods pasta' 'alcohol' 'pets' 'babies'
 'other' 'missing' 'bulk']
```

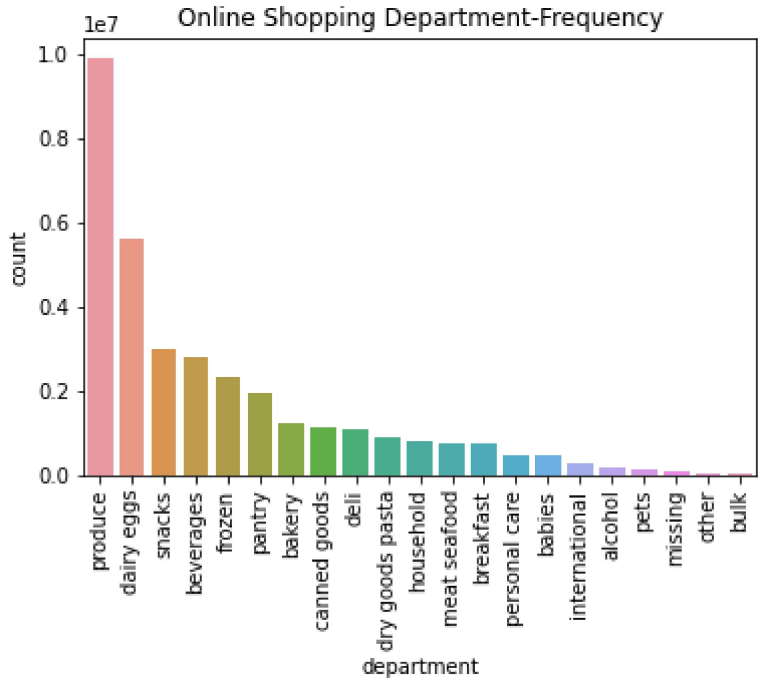
Visuals

In [21]:
```
from matplotlib.pyplot import figure

figure(figsize=(10, 7))
```

Out[21]:
```
<Figure size 720x504 with 0 Axes>

<Figure size 720x504 with 0 Axes>
```

In [22]:
```
sns.countplot(df.department, order=df.department.value_counts().index)
plt.title('Online Shopping Department-Frequency')
plt.xticks(rotation=90)
plt.show()
```

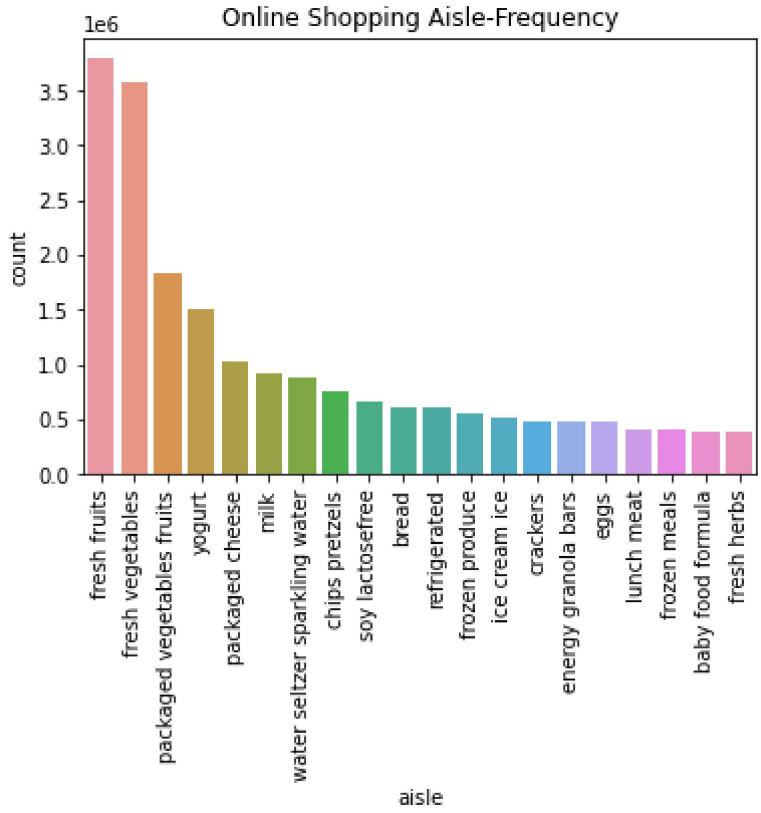
```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing o
ther arguments without an explicit keyword will result in an error or misin
terpretation.
  warnings.warn(
```



In [23]:

```
sns.countplot(df.aisle, order=df.aisle.value_counts().index[:20])
plt.title('Online Shopping Aisle-Frequency')
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing o
ther arguments without an explicit keyword will result in an error or misin
terpretation.
  warnings.warn(
```



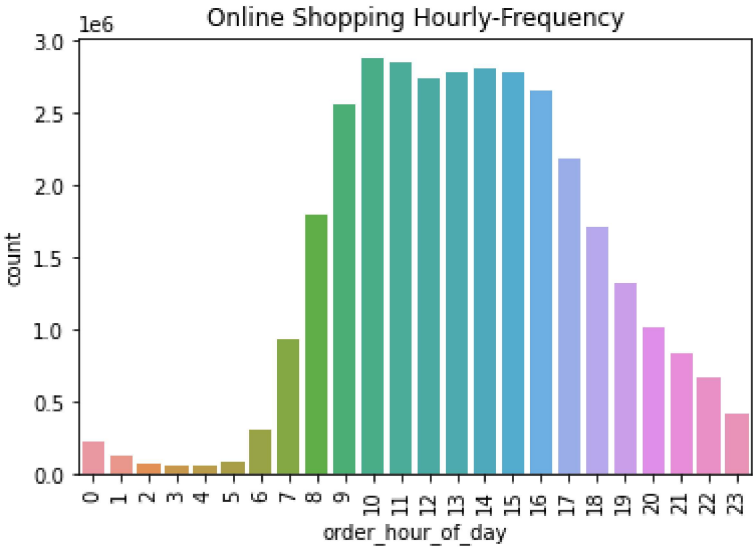
In [24]:

```
sns.countplot(df.product_name, order=df.product_name.value_counts().index[:20
plt.title('Online Shopping Products-Frequency')
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing o
ther arguments without an explicit keyword will result in an error or misin
terpretation.
  warnings.warn(
```

In [25]:
```
sns.countplot(df.order_hour_of_day)#, order=df.aisle.value_counts().index[:20
plt.title('Online Shopping Hourly-Frequency')
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing o
ther arguments without an explicit keyword will result in an error or misin
terpretation.
  warnings.warn(
```



1.Apriori

In [26]:
```
#sample
sample1 = df[df['order_id']<30000]
```

In [27]:
```
df_item = sample1[['order_id','product_name']].copy()
df_item.rename(columns={'order_id':'order','product_name':'items'},inplace=Tr
df_item['temp']=1
```

In [28]:
```
df_2 = df_item.groupby(['order','items'])['temp'].sum().unstack().fillna(0)
```

In [29]:
```
def myencoder(i):
    if i <= 0:
        return 0
    elif i>=1:
        return 1
```

In [30]:

```
from datetime import datetime
start = datetime.now()

df_3 = df_2.applymap(myencoder)

from mlxtend.frequent_patterns import apriori
frequent_itemsets = apriori(df_3,
                            min_support = 0.01,
                            max_len = 2,
                            use_colnames = True)
print(frequent_itemsets.head())

frequent_itemsets.shape

rules_1 = association_rules(frequent_itemsets,
                            metric="confidence", min_threshold=0)
print(rules_1)

end = datetime.now()
print(end - start)
```

```
    support                    itemsets
0  0.011755    (100% Raw Coconut Water)
1  0.019114    (100% Whole Wheat Bread)
2  0.012504       (2% Reduced Fat Milk)
3  0.026099  (Apple Honeycrisp Organic)
4  0.020102                 (Asparagus)
                 antecedents                consequents  antecedent support
\
0   (Bag of Organic Bananas)    (Organic Baby Spinach)            0.119557
1     (Organic Baby Spinach)  (Bag of Organic Bananas)            0.074787
2   (Bag of Organic Bananas)    (Organic Hass Avocado)            0.119557
3     (Organic Hass Avocado)  (Bag of Organic Bananas)            0.066474
4   (Bag of Organic Bananas)     (Organic Raspberries)            0.119557
5      (Organic Raspberries)  (Bag of Organic Bananas)            0.043237
6     (Organic Strawberries)  (Bag of Organic Bananas)            0.080511
7   (Bag of Organic Bananas)    (Organic Strawberries)            0.119557
8                   (Banana)          (Cucumber Kirby)            0.148109
9           (Cucumber Kirby)                  (Banana)            0.030664
10                  (Banana)             (Large Lemon)            0.148109
11             (Large Lemon)                  (Banana)            0.046985
12          (Organic Avocado)                  (Banana)            0.054310
13                  (Banana)         (Organic Avocado)            0.148109
14                  (Banana)    (Organic Baby Spinach)            0.148109
15    (Organic Baby Spinach)                  (Banana)            0.074787
16                  (Banana)      (Organic Fuji Apple)            0.148109
17      (Organic Fuji Apple)                  (Banana)            0.027768
18                  (Banana)    (Organic Hass Avocado)            0.148109
19    (Organic Hass Avocado)                  (Banana)            0.066474
20    (Organic Strawberries)                  (Banana)            0.080511
21                  (Banana)    (Organic Strawberries)            0.148109
22                  (Banana)           (Strawberries)            0.148109
23             (Strawberries)                  (Banana)            0.045247
24    (Organic Hass Avocado)    (Organic Baby Spinach)            0.066474
25    (Organic Baby Spinach)    (Organic Hass Avocado)            0.074787
26    (Organic Strawberries)    (Organic Baby Spinach)            0.080511
```

27	(Organic Baby Spinach)	(Organic Strawberries)					0.074787
28	(Organic Strawberries)	(Organic Hass Avocado)					0.080511
29	(Organic Hass Avocado)	(Organic Strawberries)					0.066474

	consequent support	support	confidence	lift	leverage	convictio n
0	0.074787	0.016320	0.136506	1.825264	0.007379	1.07147 6
1	0.119557	0.016320	0.218223	1.825264	0.007379	1.12620 8
2	0.066474	0.019012	0.159020	2.392223	0.011065	1.11004 5
3	0.119557	0.019012	0.286007	2.392223	0.011065	1.23312 6
4	0.043237	0.013186	0.110288	2.550786	0.008016	1.07536 3
5	0.119557	0.013186	0.304965	2.550786	0.008016	1.26676 0
6	0.119557	0.018910	0.234871	1.964509	0.009284	1.15071 2
7	0.080511	0.018910	0.158165	1.964509	0.009284	1.09224 3
8	0.030664	0.010358	0.069933	2.280602	0.005816	1.04222 2
9	0.148109	0.010358	0.337778	2.280602	0.005816	1.28641 3
10	0.046985	0.012061	0.081435	1.733235	0.005102	1.03750 5
11	0.148109	0.012061	0.256708	1.733235	0.005102	1.14610 5
12	0.148109	0.016286	0.299875	2.024688	0.008242	1.21676 9
13	0.054310	0.016286	0.109961	2.024688	0.008242	1.06252 6
14	0.074787	0.016491	0.111341	1.488776	0.005414	1.04113 4
15	0.148109	0.016491	0.220501	1.488776	0.005414	1.09287 0
16	0.027768	0.010392	0.070163	2.526741	0.006279	1.04559 4
17	0.148109	0.010392	0.374233	2.526741	0.006279	1.36135 5
18	0.066474	0.010426	0.070393	1.058967	0.000581	1.00421 7
19	0.148109	0.010426	0.156843	1.058967	0.000581	1.01035 8
20	0.148109	0.017615	0.218790	1.477220	0.005691	1.09047 6
21	0.080511	0.017615	0.118933	1.477220	0.005691	1.04360 8
22	0.045247	0.012709	0.085806	1.896397	0.006007	1.04436 6
23	0.148109	0.012709	0.280873	1.896397	0.006007	1.18461 9
24	0.074787	0.010051	0.151205	2.021801	0.005080	1.09003 0
25	0.066474	0.010051	0.134396	2.021801	0.005080	1.07846

```
9
26            0.074787  0.011925    0.148117  1.980514  0.005904    1.08608
0
27            0.080511  0.011925    0.159453  1.980514  0.005904    1.09391
8
28            0.066474  0.012675    0.157427  2.368264  0.007323    1.10794
7
29            0.080511  0.012675    0.190671  2.368264  0.007323    1.13611
3
0:02:49.771294
```

2.FP_Growth

In [32]:

```
from datetime import datetime
start = datetime.now()

df_3 = df_2.applymap(myencoder)

frequent_itemsets_fp=fpgrowth(df_3, min_support=0.01,use_colnames=True)
rules_fp = association_rules(frequent_itemsets_fp,
            metric="confidence", min_threshold=0)

rules_fp

end = datetime.now()
print(end - start)
```

```
0:13:37.774186
```

In [33]:
```
print(rules_fp)
```

```
   antecedents consequents  antecedent support  consequent support   supp
ort  \
0      (14907)      (1646)            0.066474            0.119557  0.019
012
1       (1646)     (14907)            0.119557            0.066474  0.019
012
2       (1752)     (14907)            0.148109            0.066474  0.010
426
3      (14907)      (1752)            0.066474            0.148109  0.010
426
4      (13930)     (14907)            0.074787            0.066474  0.010
051
5      (14907)     (13930)            0.066474            0.074787  0.010
051
6      (14907)     (15980)            0.066474            0.080511  0.012
675
7      (15980)     (14907)            0.080511            0.066474  0.012
675
8       (1752)      (5655)            0.148109            0.030664  0.010
358
9       (5655)      (1752)            0.030664            0.148109  0.010
358
10      (1752)     (13930)            0.148109            0.074787  0.016
491
11     (13930)      (1752)            0.074787            0.148109  0.016
491
12     (13930)     (15980)            0.074787            0.080511  0.011
925
13     (15980)     (13930)            0.080511            0.074787  0.011
925
14     (13930)      (1646)            0.074787            0.119557  0.016
320
15      (1646)     (13930)            0.119557            0.074787  0.016
320
16     (15597)      (1646)            0.043237            0.119557  0.013
186
17      (1646)     (15597)            0.119557            0.043237  0.013
186
18      (1752)     (15980)            0.148109            0.080511  0.017
615
19     (15980)      (1752)            0.080511            0.148109  0.017
615
20     (15980)      (1646)            0.080511            0.119557  0.018
910
21      (1646)     (15980)            0.119557            0.080511  0.018
910
22     (13904)      (1752)            0.054310            0.148109  0.016
286
23      (1752)     (13904)            0.148109            0.054310  0.016
286
24      (1752)     (10821)            0.148109            0.046985  0.012
061
25     (10821)      (1752)            0.046985            0.148109  0.012
061
```

```
26      (1752)     (14691)            0.148109            0.027768  0.010
392
27     (14691)      (1752)            0.027768            0.148109  0.010
392
28      (1752)     (21530)            0.148109            0.045247  0.012
709
29     (21530)      (1752)            0.045247            0.148109  0.012
709

    confidence      lift  leverage  conviction
0     0.286007  2.392223  0.011065    1.233126
1     0.159020  2.392223  0.011065    1.110045
2     0.070393  1.058967  0.000581    1.004217
3     0.156843  1.058967  0.000581    1.010358
4     0.134396  2.021801  0.005080    1.078469
5     0.151205  2.021801  0.005080    1.090030
6     0.190671  2.368264  0.007323    1.136113
7     0.157427  2.368264  0.007323    1.107947
8     0.069933  2.280602  0.005816    1.042222
9     0.337778  2.280602  0.005816    1.286413
10    0.111341  1.488776  0.005414    1.041134
11    0.220501  1.488776  0.005414    1.092870
12    0.159453  1.980514  0.005904    1.093918
13    0.148117  1.980514  0.005904    1.086080
14    0.218223  1.825264  0.007379    1.126208
15    0.136506  1.825264  0.007379    1.071476
16    0.304965  2.550786  0.008016    1.266760
17    0.110288  2.550786  0.008016    1.075363
18    0.118933  1.477220  0.005691    1.043608
19    0.218790  1.477220  0.005691    1.090476
20    0.234871  1.964509  0.009284    1.150712
21    0.158165  1.964509  0.009284    1.092243
22    0.299875  2.024688  0.008242    1.216769
23    0.109961  2.024688  0.008242    1.062526
24    0.081435  1.733235  0.005102    1.037505
25    0.256708  1.733235  0.005102    1.146105
26    0.070163  2.526741  0.006279    1.045594
27    0.374233  2.526741  0.006279    1.361355
28    0.085806  1.896397  0.006007    1.044366
29    0.280873  1.896397  0.006007    1.184619
```

3.ECLAT

In [34]:
```
sample2 = df[df['order_id']<5000]
```

In [35]:
```
e_df = sample2.groupby(['order_id'])['product_name'].apply(', '.join).reset_i
e_df.head()

transactions = e_df['product_name'].apply(lambda t: t.split(','))
transactions = list(transactions)
len(transactions)
```

Out[35]: 4910

In [36]:
```
transactions
```

Out[36]:
```
[['Bag of Organic Bananas',
  ' Organic Hass Avocado',
  ' Cucumber Kirby',
  ' Organic Whole String Cheese',
  ' Organic Celery Hearts',
  ' Organic 4% Milk Fat Whole Milk Cottage Cheese',
  ' Lightly Smoked Sardines in Olive Oil',
  ' Bulgarian Yogurt'],
 ['Organic Egg Whites',
  ' Michigan Organic Kale',
  ' Garlic Powder',
  ' Coconut Butter',
  ' Natural Sweetener',
  ' Carrots',
  ' Original Unflavored Gelatine Mix',
  ' All Natural No Stir Creamy Almond Butter',
  ' Classic Blend Cole Slaw'],
 ['Total 2% with Strawberry Lowfat Greek Strained Yogurt',
  ' Unsweetened Almondmilk',
```

In [37]:
```
!pip install pyECLAT
```

```
Requirement already satisfied: pyECLAT in c:\users\aparna akula\anaconda3\l
ib\site-packages (1.0.2)
Requirement already satisfied: tqdm>=4.41.1 in c:\users\aparna akula\anacon
da3\lib\site-packages (from pyECLAT) (4.62.3)
Requirement already satisfied: pandas>=0.25.3 in c:\users\aparna akula\anac
onda3\lib\site-packages (from pyECLAT) (1.3.4)
Requirement already satisfied: numpy>=1.17.4 in c:\users\aparna akula\anaco
nda3\lib\site-packages (from pyECLAT) (1.20.3)
Requirement already satisfied: pytz>=2017.3 in c:\users\aparna akula\anacon
da3\lib\site-packages (from pandas>=0.25.3->pyECLAT) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\aparna ak
ula\anaconda3\lib\site-packages (from pandas>=0.25.3->pyECLAT) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\aparna akula\anaconda3
\lib\site-packages (from python-dateutil>=2.7.3->pandas>=0.25.3->pyECLAT)
(1.16.0)
Requirement already satisfied: colorama in c:\users\aparna akula\anaconda3
\lib\site-packages (from tqdm>=4.41.1->pyECLAT) (0.4.4)
```

In [38]:
```
list2 = pd.DataFrame(transactions)
```

In [39]:
```
list2.head(3)
```

Out[39]:

	0	1	2	3	4	5	6	7
0	Bag of Organic Bananas	Organic Hass Avocado	Cucumber Kirby	Organic Whole String Cheese	Organic Celery Hearts	Organic 4% Milk Fat Whole Milk Cottage Cheese	Lightly Smoked Sardines in Olive Oil	Bulgarian Yogurt
1	Organic Egg Whites	Michigan Organic Kale	Garlic Powder	Coconut Butter	Natural Sweetener	Carrots	Original Unflavored Gelatine Mix	All Natural No Stir Creamy Almond Butter
2	Total 2% with Strawberry Lowfat Greek Strained...	Unsweetened Almondmilk	Lemons	Organic Baby Spinach	Unsweetened Chocolate Almond Breeze Almond Milk	Organic Ginger Root	Air Chilled Organic Boneless Skinless Chicken...	Organic Ezekiel 49 Bread Cinnamon Raisin

3 rows × 69 columns

In [40]:
```
from pyECLAT import ECLAT
eclat = ECLAT(data=list2)
```

In [ ]:
```
from datetime import datetime

start = datetime.now()


min_support = 1/100

# start from transactions containing at least 2 items
min_combination = 2

# up to maximum items per transaction
max_combination = 2

rule_indices, rule_supports = eclat.fit(min_support=min_support,
                                        min_combination=min_combination,
                                        max_combination=max_combination,
                                        separator=' & ',verbose=True)

end = datetime.now()
print(end - start)
```

In [ ]:
```
result = pd.DataFrame(rule_supports.items(),columns=['Item', 'Support'])
result.sort_values(by=['Support'], ascending=False)
```

4.FAST APRIORI

In [41]:
```
sample3 = df[df['order_id']<50000]

df_item2 = sample3[['order_id','product_name']].copy()
df_item2.rename(columns={'order_id':'order','product_name':'items'},inplace=T
df_item2['temp']=1

df_fa = df_item2.groupby(['order','items'])['temp'].sum().unstack().fillna(0)
```

In [43]:
```
def fast_apriori(df_FA):
  frequent = apriori(df_FA,min_support=0.01,
                     use_colnames=True)
  rules_apriori = association_rules(frequent,metric='support',
                    support_only=True,min_threshold=0.004)
  output = rules_apriori[['antecedents','consequents','support']]
  output['antecedents'].duplicated().sum()
  cols = ['antecedents','consequents']
  output[cols] = output[cols].applymap(lambda x: tuple(x))
  output = output.explode('antecedents').reset_index(drop=True).
                    explode('consequents').reset_index(drop=True)
  output.sort_values('support',ascending=False)
  print(output)
```

In [44]:

```
from datetime import datetime
start = datetime.now()

fast_apriori(df_fa)

end = datetime.now()
print(end - start)
```

```
                antecedents            consequents   support
0    Bag of Organic Bananas    Organic Baby Spinach  0.016185
1      Organic Baby Spinach  Bag of Organic Bananas  0.016185
2    Bag of Organic Bananas    Organic Hass Avocado  0.019418
3      Organic Hass Avocado  Bag of Organic Bananas  0.019418
4    Bag of Organic Bananas     Organic Raspberries  0.013423
5       Organic Raspberries  Bag of Organic Bananas  0.013423
6      Organic Strawberries  Bag of Organic Bananas  0.019030
7    Bag of Organic Bananas    Organic Strawberries  0.019030
8                    Banana              Large Lemon  0.012891
9                Large Lemon                   Banana  0.012891
10           Organic Avocado                   Banana  0.016738
11                   Banana         Organic Avocado  0.016738
12                   Banana    Organic Baby Spinach  0.016615
13     Organic Baby Spinach                   Banana  0.016615
14                   Banana      Organic Fuji Apple  0.010088
15       Organic Fuji Apple                   Banana  0.010088
16                   Banana    Organic Hass Avocado  0.010272
17     Organic Hass Avocado                   Banana  0.010272
18     Organic Strawberries                   Banana  0.018518
19                   Banana    Organic Strawberries  0.018518
20                   Banana             Strawberries  0.012625
21               Strawberries                   Banana  0.012625
22     Organic Hass Avocado    Organic Baby Spinach  0.010436
23     Organic Baby Spinach    Organic Hass Avocado  0.010436
24     Organic Strawberries    Organic Baby Spinach  0.012114
25     Organic Baby Spinach    Organic Strawberries  0.012114
26     Organic Strawberries    Organic Hass Avocado  0.012686
27     Organic Hass Avocado    Organic Strawberries  0.012686
0:01:26.761411
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\pandas\core\frame.py:364
1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self[k1] = value[k2]
```

5.Super Fast Apriori

In [60]:
```
orders1 = order_products__prior.set_index('order_id')['product_id'].rename('i
display(orders1.head(10))
type(orders1)
```

```
order_id
2    33120
2    28985
2     9327
2    45918
2    30035
2    17794
2    40141
2     1819
2    43668
3    33754
Name: item_id, dtype: int64
```

Out[60]: pandas.core.series.Series

In [61]:
```
def size(obj):
    return "{0:.2f} MB".format(sys.getsizeof(obj) / (1000 * 1000))
print('dimensions: {0};   size: {1};   unique_orders: {2};   unique_items: {3
      .format(orders1.shape, size(orders), len(orders1.index.unique()), len(o
```

```
dimensions: (32434489,);   size: 376.24 MB;   unique_orders: 3214874;   uni
que_items: 49677
```

In [62]:

```
# Returns frequency counts for items and item pairs
def freq(iterable):
    if type(iterable) == pd.core.series.Series:
        return iterable.value_counts().rename("freq")
    else:
        return pd.Series(Counter(iterable)).rename("freq")


# Returns number of unique orders
def order_count(order_item):
    return len(set(order_item.index))


# Returns generator that yields item pairs, one at a time
def get_item_pairs(order_item):
    order_item = order_item.reset_index().to_numpy()
    for order_id, order_object in groupby(order_item, lambda x: x[0]):
        item_list = [item[1] for item in order_object]

        for item_pair in combinations(item_list, 2):
            yield item_pair


# Returns frequency and support associated with item
def merge_item_stats(item_pairs, item_stats):
    return (item_pairs
                .merge(item_stats.rename(columns={'freq': 'freqA', 'support':
                .merge(item_stats.rename(columns={'freq': 'freqB', 'support':


# Returns name associated with item
def merge_item_name(rules, item_name):
    columns = ['itemA','itemB','freqAB','supportAB','freqA','supportA','freqB
               'confidenceAtoB','confidenceBtoA','lift']
    rules = (rules
                .merge(item_name.rename(columns={'item_name': 'itemA'}), left
                .merge(item_name.rename(columns={'item_name': 'itemB'}), left
    return rules[columns]
```

In [63]:

```
def association_rules(order_item, min_support):

    print("Starting order_item: {:22d}".format(len(order_item)))

    # Calculate item frequency and support
    item_stats             = freq(order_item).to_frame("freq")
    item_stats['support']  = item_stats['freq'] / order_count(order_item) * 1

    # Filter from order_item items below min support
    qualifying_items       = item_stats[item_stats['support'] >= min_support]
    order_item             = order_item[order_item.isin(qualifying_items)]

    print("Items with support >= {}: {:15d}".format(min_support, len(qualifyi
    print("Remaining order_item: {:21d}".format(len(order_item)))

    # Filter from order_item orders with less than 2 items
    order_size             = freq(order_item.index)
    qualifying_orders      = order_size[order_size >= 2].index
    order_item             = order_item[order_item.index.isin(qualifying_orde

    print("Remaining orders with 2+ items: {:11d}".format(len(qualifying_orde
    print("Remaining order_item: {:21d}".format(len(order_item)))

    # Recalculate item frequency and support
    item_stats             = freq(order_item).to_frame("freq")
    item_stats['support']  = item_stats['freq'] / order_count(order_item) * 1

    # Get item pairs generator
    item_pair_gen          = get_item_pairs(order_item)

    # Calculate item pair frequency and support
    item_pairs              = freq(item_pair_gen).to_frame("freqAB")
    item_pairs['supportAB'] = item_pairs['freqAB'] / len(qualifying_orders) *

    print("Item pairs: {:31d}".format(len(item_pairs)))

    # Filter from item_pairs those below min support
    item_pairs              = item_pairs[item_pairs['supportAB'] >= min_suppo

    print("Item pairs with support >= {}: {:10d}\n".format(min_support, len(i

    # Create table of association rules and compute relevant metrics
    item_pairs = item_pairs.reset_index().rename(columns={'level_0': 'item_A'
    item_pairs = merge_item_stats(item_pairs, item_stats)

    item_pairs['confidenceAtoB'] = item_pairs['supportAB'] / item_pairs['supp
    item_pairs['confidenceBtoA'] = item_pairs['supportAB'] / item_pairs['supp
    item_pairs['lift']           = item_pairs['supportAB'] / (item_pairs['sup

    # Return association rules sorted by lift in descending order
    return item_pairs.sort_values('lift', ascending=False)
```

In [64]:

```
orders1 = orders1[0:50000,]
```

In [65]:

```
#dataset with 50,000 records
from datetime import datetime
start = datetime.now()

rules = association_rules(orders1, 0.01)

end = datetime.now()
print(end - start)
```

```
Starting order_item:                  50000
Items with support >= 0.01:           11616
Remaining order_item:                 50000
Remaining orders with 2+ items:        4752
Remaining order_item:                 49774
Item pairs:                          320078
Item pairs with support >= 0.01:     320078

0:00:02.965432
```

In [ ]:

```
from datetime import datetime
start = datetime.now()

rules = association_rules(df_3, 0.01)

end = datetime.now()
print(end - start)
```

In [66]:

```
# Replace item ID with item name and display association rules
item_name    = pd.read_csv(r'C:\Users\Aparna Akula\Documents\Data Mining\Proje
item_name    = item_name.rename(columns={'product_id':'item_id', 'product_name
rules_final = merge_item_name(rules, item_name).sort_values('lift', ascending
display(rules_final)
```

	itemA	itemB	freqAB	supportAB	freqA	supportA	freqB	supportB	co
0	Unstopables Downy Unstopables Fresh In-Wash Sc...	Pistachios, Premium Blend, Pomegranate, with A...	1	0.021044	1	0.021044	1	0.021044	
245100	Organic Seedless Raisins	Pure Wildflower Raw Honey	1	0.021044	1	0.021044	1	0.021044	
245231	Quince	Lightly Toasted Almond Butter Crunchy	1	0.021044	1	0.021044	1	0.021044	
245230	Dragon Fruit	Lightly Toasted Almond Butter Crunchy	1	0.021044	1	0.021044	1	0.021044	
245220	Dragon Fruit	Quince	1	0.021044	1	0.021044	1	0.021044	
...	...	...	...	...	...	...	...	...	
15527	Organic Red Bell Pepper	Banana	1	0.021044	99	2.083333	750	15.782828	
5999	Honeycrisp Apple	Bag of Organic Bananas	1	0.021044	143	3.009259	613	12.899832	
3393	Organic Hass Avocado	Organic Avocado	1	0.021044	333	7.007576	274	5.765993	
18093	Organic Avocado	Organic Hass Avocado	1	0.021044	274	5.765993	333	7.007576	
6017	Banana	Bag of Organic Bananas	1	0.021044	750	15.782828	613	12.899832	

320078 rows × 11 columns

In [67]:
```
print(rules_final.sort_values(by=['freqAB']).tail(15))
```

```
                        itemA                    itemB  freqAB  supportAB  \
78427  Bag of Organic Bananas    Organic Yellow Onion      36   0.757576
72801                  Banana     Organic Raspberries      36   0.757576
15561         Organic Avocado                  Banana      36   0.757576
51426                  Banana      Organic Fuji Apple      36   0.757576
66012                  Banana          Cucumber Kirby      37   0.778620
6010     Organic Hass Avocado  Bag of Organic Bananas      40   0.841751
34283                  Banana        Honeycrisp Apple      40   0.841751
4551                   Banana    Organic Baby Spinach      44   0.925926
3397                   Banana         Organic Avocado      46   0.968013
72799  Bag of Organic Bananas     Organic Raspberries      46   0.968013
8408                   Banana            Strawberries      52   1.094276
4550   Bag of Organic Bananas    Organic Baby Spinach      53   1.115320
96841  Bag of Organic Bananas    Organic Strawberries      60   1.262626
18097  Bag of Organic Bananas    Organic Hass Avocado      68   1.430976
96842                  Banana    Organic Strawberries      72   1.515152

       freqA   supportA  freqB   supportB  confidenceAtoB  confidenceBtoA
\
78427    613  12.899832    177   3.724747        0.058728        0.203390
72801    750  15.782828    223   4.692761        0.048000        0.161435
15561    274   5.765993    750  15.782828        0.131387        0.048000
51426    750  15.782828    146   3.072391        0.048000        0.246575
66012    750  15.782828    137   2.882997        0.049333        0.270073
6010     333   7.007576    613  12.899832        0.120120        0.065253
34283    750  15.782828    143   3.009259        0.053333        0.279720
4551     750  15.782828    345   7.260101        0.058667        0.127536
3397     750  15.782828    274   5.765993        0.061333        0.167883
72799    613  12.899832    223   4.692761        0.075041        0.206278
8408     750  15.782828    217   4.566498        0.069333        0.239631
4550     613  12.899832    345   7.260101        0.086460        0.153623
96841    613  12.899832    415   8.733165        0.097879        0.144578
18097    613  12.899832    333   7.007576        0.110930        0.204204
96842    750  15.782828    415   8.733165        0.096000        0.173494

           lift
78427  0.015767
72801  0.010229
15561  0.008325
51426  0.015623
66012  0.017112
6010   0.009312
34283  0.017723
4551   0.008081
3397   0.010637
72799  0.015991
8408   0.015183
4550   0.011909
96841  0.011208
18097  0.015830
96842  0.010993
```