Market Basket Challenge Solution

In this notebook, we conduct market basket analysis which allows us to identify the relationship between different combinations of products that users buy.

Challenge Objective:

1. Calculate confidence, tendency that given a particular product(s) is(/are) purchased, than (an) other product(s) will also be purchased, between two or three products.
2. Give understanding of users' tastes and preferences for decisions about marketing activities such as recommendations, pricing, promotions for bundling product, etc.
3. (Bonus) Determine association rule having high confidence

In [1]:
```
import pandas as pd
import numpy as np
```
executed in 21.6s, finished 15:39:36 2020-12-18

Importing Data and Their Early Exploration

In [2]:
```
transc_order0 = pd.read_csv('association_order.csv')
asc_rules = pd.read_csv('rules.csv')
```
executed in 925ms, finished 15:39:36 2020-12-18

In [3]:
```
print('Number of transaction order:', len(transc_order0))
print('Transaction list:')
transc_order0
```

executed in 1.10s, finished 15:39:38 2020-12-18

```
Number of transaction order: 423861
Transaction list:
```

Out[3]:

	orderid	itemid
0	31379820545759	719740607
1	31378575577269	1825360194
2	31369591568249	1108903291
3	31369836201769	4507360843
4	31372360246729	1821888475
...	...	...
423856	31348804008384	2174769495
423857	31379932877729	6912791179
423858	31368033346399	1340457527
423859	31381864883559	2515423948
423860	31358790557954	1488272669

423861 rows × 2 columns

In [4]:
```
print('Number of association rules whose confidence will be counted:', len(asc_rules))
print('Association rules list:')
asc_rules
```

executed in 120ms, finished 15:39:38 2020-12-18

```
Number of association rules whose confidence will be counted: 14238
Association rules list:
```

Out[4]:

	rule
0	100242812>80361758
1	100242812>89031406
2	1003153762>1016449477
3	1006024995>2727415265
4	1006024995>866012366
...	...
14233	995073047>3202007524
14234	995073047>651958908
14235	995073047>7902698606
14236	995073047>922394800
14237	995073047>991988255

14238 rows × 1 columns

This tabel provides association rules of many products on Shopee.

Example: In row 1, association rule of 100242812 > 80361758 indicate customers buying product 100242812 would buying product 80361758 as well. In this challenge, we compute tendency of this association rule.

Process and Split Association Rules Data

Splitting Item

In [5]:
```
#1. Splitting based on '>' to get product(s) in left and right group
asc_rules['item_rule'] = asc_rules['rule'].apply(lambda x:x.split('>')[:])
asc_rules['pre_group_A'] = asc_rules['item_rule'].apply(lambda x:x[0])
asc_rules['pre_group_B'] = asc_rules['item_rule'].apply(lambda x:x[1])
asc_rules = asc_rules.drop('item_rule',axis=1)
asc_rules
```

executed in 560ms, finished 15:39:38 2020-12-18

Out[5]:

	rule	pre_group_A	pre_group_B
0	100242812>80361758	100242812	80361758
1	100242812>89031406	100242812	89031406
2	1003153762>1016449477	1003153762	1016449477
3	1006024995>2727415265	1006024995	2727415265
4	1006024995>866012366	1006024995	866012366
...	...	...	...
14233	995073047>3202007524	995073047	3202007524
14234	995073047>651958908	995073047	651958908
14235	995073047>7902698606	995073047	7902698606
14236	995073047>922394800	995073047	922394800
14237	995073047>991988255	995073047	991988255

14238 rows × 3 columns

In [6]:
```
#2. Splitting based on '&' to get item in both groups
asc_rules['group_A'] = asc_rules['pre_group_A'].apply(lambda x:x.split('&')[:])
asc_rules['group_B'] = asc_rules['pre_group_B'].apply(lambda x:x.split('&')[:])
asc_rules['len_A'] = asc_rules['group_A'].apply(lambda x:len(x))
asc_rules['len_B'] = asc_rules['group_B'].apply(lambda x:len(x))
asc_rules
```

executed in 248ms, finished 15:39:39 2020-12-18

Out[6]:

	rule	pre_group_A	pre_group_B	group_A	group_B	len_A	len_B
0	100242812>80361758	100242812	80361758	[100242812]	[80361758]	1	1
1	100242812>89031406	100242812	89031406	[100242812]	[89031406]	1	1
2	1003153762>1016449477	1003153762	1016449477	[1003153762]	[1016449477]	1	1
3	1006024995>2727415265	1006024995	2727415265	[1006024995]	[2727415265]	1	1
4	1006024995>866012366	1006024995	866012366	[1006024995]	[866012366]	1	1
...	...	...	...	...	...	...	...
14233	995073047>3202007524	995073047	3202007524	[995073047]	[3202007524]	1	1
14234	995073047>651958908	995073047	651958908	[995073047]	[651958908]	1	1
14235	995073047>7902698606	995073047	7902698606	[995073047]	[7902698606]	1	1
14236	995073047>922394800	995073047	922394800	[995073047]	[922394800]	1	1
14237	995073047>991988255	995073047	991988255	[995073047]	[991988255]	1	1

14238 rows × 7 columns

Split Association Rules Data

In this stage, we split association rules data based on how many item in left and right side. As explained on Kaggle competition page, there are three type of association rules structured in data.

In [7]:
```
asc_rules.groupby(['len_A','len_B'])['rule'].count().reset_index()
```

executed in 280ms, finished 15:39:39 2020-12-18

Out[7]:

	len_A	len_B	rule
0	1	1	7272
1	1	2	3483
2	2	1	3483

In [8]:
```
# Manipulate groups whose have 1 product with append them with stuff that different with itemid.
# We append with '0'. The aim is to get all the data having two 'products' in both groups and to split them later.
for i in asc_rules.index:
    if asc_rules['len_A'][i]==1:
        asc_rules.loc[i,'group_A'].append(0)
for i in asc_rules.index:
    if asc_rules['len_B'][i]==1:
        asc_rules.loc[i,'group_B'].append(0)
```

executed in 4.04s, finished 15:39:43 2020-12-18

In [9]:
```
asc_rules['item_A1'] = asc_rules['group_A'].apply(lambda x:x[0])
asc_rules['item_A2'] = asc_rules['group_A'].apply(lambda x:x[1])
asc_rules['item_B1'] = asc_rules['group_B'].apply(lambda x:x[0])
asc_rules['item_B2'] = asc_rules['group_B'].apply(lambda x:x[1])
asc_rules = asc_rules.drop(['pre_group_A','pre_group_B','group_A','group_B','len_A','len_B'],axis=1)
asc_rules.head(30)
```

executed in 208ms, finished 15:39:43 2020-12-18

Out[9]:

	rule	item_A1	item_A2	item_B1	item_B2
0	100242812>80361758	100242812	0	80361758	0
1	100242812>89031406	100242812	0	89031406	0
2	1003153762>1016449477	1003153762	0	1016449477	0
3	1006024995>2727415265	1006024995	0	2727415265	0
4	1006024995>866012366	1006024995	0	866012366	0
5	1006149508>2867088619	1006149508	0	2867088619	0
6	1006149508>599643179	1006149508	0	599643179	0
7	1006149508>943964984	1006149508	0	943964984	0
8	1008435393&1008455319>454021998	1008435393	1008455319	454021998	0
9	1008435393&1336146681>454021998	1008435393	1336146681	454021998	0
10	1008435393&1492482010>454021998	1008435393	1492482010	454021998	0
11	1008435393&1939300455>454021998	1008435393	1939300455	454021998	0
12	1008435393>1008455319	1008435393	0	1008455319	0
13	1008435393>1336146681	1008435393	0	1336146681	0
14	1008435393>1492482010	1008435393	0	1492482010	0
15	1008435393>1939300455	1008435393	0	1939300455	0
16	1008435393>2121131157	1008435393	0	2121131157	0
17	1008435393>454021930	1008435393	0	454021930	0
18	1008435393>454021936	1008435393	0	454021936	0
19	1008435393>454021937	1008435393	0	454021937	0

	rule	item_A1	item_A2	item_B1	item_B2
20	1008435393>454021955	1008435393	0	454021955	0
21	1008435393>454021998	1008435393	0	454021998	0
22	1008435393>454021998&1008455319	1008435393	0	454021998	1008455319
23	1008435393>454021998&1336146681	1008435393	0	454021998	1336146681
24	1008435393>454021998&1492482010	1008435393	0	454021998	1492482010
25	1008435393>454021998&1939300455	1008435393	0	454021998	1939300455
26	1008435393>454021998&454021999	1008435393	0	454021998	454021999
27	1008435393>454021999	1008435393	0	454021999	0
28	1008455319>1008435393	1008455319	0	1008435393	0
29	1008455319>1336146681	1008455319	0	1336146681	0

In [10]:
```
# Splitting association data based on how many item in each group
asc_rules1 = asc_rules[(asc_rules.item_A2==0)&(asc_rules.item_B2==0)][['rule','item_A1','item_B1']]
asc_rules2 = asc_rules[(asc_rules.item_A2==0)&(asc_rules.item_B2!=0)][['rule','item_A1','item_B1','item_B2']]
asc_rules3 = asc_rules[(asc_rules.item_A2!=0)&(asc_rules.item_B2==0)][['rule','item_A1','item_A2','item_B1']]
```

executed in 297ms, finished 15:39:43 2020-12-18

In [11]:
```
asc_rules1.columns = ['rule','product_A','product_B']
asc_rules2.columns = ['rule','product_A','product_B','product_C']
asc_rules3.columns = ['rule','product_A','product_B','product_C']
```

executed in 24ms, finished 15:39:43 2020-12-18

In [12]:
```
display(asc_rules1)
display(asc_rules2)
display(asc_rules3)
```

executed in 312ms, finished 15:39:44 2020-12-18

	rule	product_A	product_B
0	100242812>80361758	100242812	80361758
1	100242812>89031406	100242812	89031406
2	1003153762>1016449477	1003153762	1016449477
3	1006024995>2727415265	1006024995	2727415265
4	1006024995>866012366	1006024995	866012366
...	...	...	...
14233	995073047>3202007524	995073047	3202007524
14234	995073047>651958908	995073047	651958908
14235	995073047>7902698606	995073047	7902698606
14236	995073047>922394800	995073047	922394800
14237	995073047>991988255	995073047	991988255

7272 rows × 3 columns

	rule	product_A	product_B	product_C
22	1008435393>454021998&1008455319	1008435393	454021998	1008455319
23	1008435393>454021998&1336146681	1008435393	454021998	1336146681
24	1008435393>454021998&1492482010	1008435393	454021998	1492482010
25	1008435393>454021998&1939300455	1008435393	454021998	1939300455
26	1008435393>454021998&454021999	1008435393	454021998	454021999
...	...	...	...	...

	rule	product_A	product_B	product_C
14156	959273129>959269904&1167299260	959273129	959269904	1167299260
14183	982727004>1512054476&2188141395	982727004	1512054476	2188141395
14184	982727004>1512054476&2523757071	982727004	1512054476	2523757071
14188	982727004>2188141395&2523757071	982727004	2188141395	2523757071
14213	991988255>926159335&2057880096	991988255	926159335	2057880096

3483 rows × 4 columns

	rule	product_A	product_B	product_C
8	1008435393&1008455319>454021998	1008435393	1008455319	454021998
9	1008435393&1336146681>454021998	1008435393	1336146681	454021998
10	1008435393&1492482010>454021998	1008435393	1492482010	454021998
11	1008435393&1939300455>454021998	1008435393	1939300455	454021998
43	1019107054&1697382283>684827047	1019107054	1697382283	684827047
...	...	...	...	...
14176	982727004&2188141395>1512054476	982727004	2188141395	1512054476
14177	982727004&2188141395>2523757071	982727004	2188141395	2523757071
14178	982727004&2523757071>1512054476	982727004	2523757071	1512054476
14179	982727004&2523757071>2188141395	982727004	2523757071	2188141395
14196	991988255&2057880096>926159335	991988255	2057880096	926159335

3483 rows × 4 columns

Drop Unnecessary Transaction

On Kaggle's competition page, it is explained that items listed in associated rules data was chosen because they are bought in quite numerous transactions. In the next step, we drop transaction data whose items are not listed on associated rules data. Reducing transaction data will reduce counting process time in several next step. We are reduce number of transaction by merging products listed in with products listed in association

rules.

In [13]:
```
# Collect product in association rules data that needed in all counts
bcd0 = asc_rules1.product_A.unique().tolist()
bcd1 = asc_rules1.product_B.unique().tolist()
bcd2 = asc_rules2.product_A.unique().tolist()
bcd3 = asc_rules2.product_B.unique().tolist()
bcd4 = asc_rules2.product_C.unique().tolist()
bcd5 = asc_rules3.product_A.unique().tolist()
bcd6 = asc_rules3.product_B.unique().tolist()
bcd7 = asc_rules3.product_C.unique().tolist()

cde0 = bcd0+bcd1+bcd2+bcd3+bcd4+bcd5+bcd6+bcd7
cde = list(set(cde0))
print('Number of product listed on before:', transc_order0['itemid'].nunique())
print('Number of product listed on after:', len(cde))
```

executed in 320ms, finished 15:39:44 2020-12-18

```
Number of product listed on before: 239702
Number of product listed on after: 2175
```

Data type of products in association rule are string, while data type of products in transaction are integer. In order to merge products/itemid in these data, they need to have same data type.

In [14]:
```
print('Data type of products are',type(cde[0]))
print('Data type of products are',type(transc_order0.itemid[0]))
```

executed in 195ms, finished 15:39:44 2020-12-18

```
Data type of products are <class 'str'>
Data type of products are <class 'numpy.int64'>
```

In [15]:
```
# Change data type
transc_order0.itemid = transc_order0.itemid.astype(str)
```

executed in 1.86s, finished 15:39:46 2020-12-18

Reduce number of transaction by merging it with products listed in association rules

In [16]:
```
barang = pd.DataFrame({'itemid':cde})
transc_order = transc_order0.merge(barang,on='itemid')
```
executed in 2.28s, finished 15:39:48 2020-12-18

In [17]:
```
print('Number of transaction order before reduction:', len(transc_order0))
print('Number of transaction order after reduction:', len(transc_order))
print('Proportion of transaction needed in Market Basket Analysis:', round(100*len(transc_order)/len(transc_order0),2),

print('Transaction list after reduction:')
transc_order
```
executed in 142ms, finished 15:39:49 2020-12-18

```
Number of transaction order before reduction: 423861
Number of transaction order after reduction: 45542
Proportion of transaction needed in Market Basket Analysis: 10.74 %
Transaction list after reduction:
```

Out[17]:

	orderid	itemid
0	31374190005914	1593652521
1	31375996453960	1593652521
2	31384428279063	1593652521
3	31372497811514	1593652521
4	31365714895571	1593652521
...	...	...
45537	31358838837234	2683332748
45538	31372338200519	2683332748
45539	31375800432851	2683332748
45540	31374806941627	2683332748
45541	31383294796850	2683332748

45542 rows × 2 columns

Calculate Frequency of Product Purchases

Step to get frequency of purchasing of two (or three) products each row:

1. For each row, define all products
2. For each product in one row, query buyer
3. Get intersection of buyer from each product (in one row) and count the number of it.

Counting Frequency of Product A

We count frequency for each product using count method. This count method is used to count the frequency of product A

In [18]:
```
belanja = transc_order.groupby('itemid')['orderid'].count().reset_index()
belanja.columns = ['product_A','freq_left']
belanja
```

executed in 622ms, finished 15:39:49 2020-12-18

Out[18]:

	product_A	freq_left
0	100242812	17
1	1003153762	18
2	1006024995	35
3	1006149508	18
4	1008435393	55
...	...	...
2170	982727004	28
2171	98467905	13
2172	988591528	27
2173	991988255	72
2174	995073047	107

2175 rows × 2 columns

In [19]:
```
asc_rules1a = asc_rules1.merge(belanja,on='product_A')
asc_rules2a = asc_rules2.merge(belanja,on='product_A')
display(asc_rules1a)
display(asc_rules2a)
```

executed in 605ms, finished 15:39:50 2020-12-18

	rule	product_A	product_B	freq_left
0	100242812>80361758	100242812	80361758	17
1	100242812>89031406	100242812	89031406	17
2	1003153762>1016449477	1003153762	1016449477	18
3	1006024995>2727415265	1006024995	2727415265	35
4	1006024995>866012366	1006024995	866012366	35
...	...	...	...	...
7267	995073047>3202007524	995073047	3202007524	107
7268	995073047>651958908	995073047	651958908	107
7269	995073047>7902698606	995073047	7902698606	107
7270	995073047>922394800	995073047	922394800	107
7271	995073047>991988255	995073047	991988255	107

7272 rows × 4 columns

	rule	product_A	product_B	product_C	freq_left
0	1008435393>454021998&1008455319	1008435393	454021998	1008455319	55
1	1008435393>454021998&1336146681	1008435393	454021998	1336146681	55
2	1008435393>454021998&1492482010	1008435393	454021998	1492482010	55
3	1008435393>454021998&1939300455	1008435393	454021998	1939300455	55
4	1008435393>454021998&454021999	1008435393	454021998	454021999	55
...	...	...	...	...	...
3478	959273129>959269904&1167299260	959273129	959269904	1167299260	30

	rule	product_A	product_B	product_C	freq_left
3479	982727004>1512054476&2188141395	982727004	1512054476	2188141395	28
3480	982727004>1512054476&2523757071	982727004	1512054476	2523757071	28
3481	982727004>2188141395&2523757071	982727004	2188141395	2523757071	28
3482	991988255>926159335&2057880096	991988255	926159335	2057880096	72

3483 rows x 5 columns

Counting Frequency of Product A&B

In [20]:
```
def two_items(df):
    frequency = []
    for i in df.index:
        j,k = (df['product_A'][i],df['product_B'][i])
        efg1 = set(transc_order[transc_order.itemid==j]['orderid'])
        efg2 = set(transc_order[transc_order.itemid==k]['orderid'])
        frequency.append(len(efg1.intersection(efg2)))
    return frequency
```

executed in 560ms, finished 15:39:50 2020-12-18

In [21]:
```
asc_rules1a['freq_all'] = two_items(asc_rules1a)
asc_rules3['freq_left'] = two_items(asc_rules3)
```

executed in 6m 36s, finished 15:46:26 2020-12-18

Counting Frequency of Product A,B,&C

In [22]:
```
def three_items(df):
    frequency = []
    for i in df.index:
        j,k,l = (df['product_A'][i],df['product_B'][i],df['product_C'][i])
        fgh1 = set(transc_order[transc_order.itemid==j]['orderid'])
        fgh2 = set(transc_order[transc_order.itemid==k]['orderid'])
        fgh3 = set(transc_order[transc_order.itemid==l]['orderid'])
        fgh4 = fgh2.intersection(fgh3)
        frequency.append(len(fgh1.intersection(fgh4)))
    return frequency
```

executed in 26ms, finished 15:46:27 2020-12-18

In [23]:
```
asc_rules2a['freq_all'] = three_items(asc_rules2a)
asc_rules3['freq_all'] = three_items(asc_rules3)
```

executed in 6m 15s, finished 15:52:42 2020-12-18

Preview of Counting Result

In [24]:
```
display(asc_rules1a)
display(asc_rules2a)
display(asc_rules3)
```

executed in 209ms, finished 15:52:42 2020-12-18

	rule	product_A	product_B	freq_left	freq_all
0	100242812>80361758	100242812	80361758	17	8
1	100242812>89031406	100242812	89031406	17	6
2	1003153762>1016449477	1003153762	1016449477	18	7
3	1006024995>2727415265	1006024995	2727415265	35	6
4	1006024995>866012366	1006024995	866012366	35	6
...	...	...	...	...	...
7267	995073047>3202007524	995073047	3202007524	107	11
7268	995073047>651958908	995073047	651958908	107	14
7269	995073047>7902698606	995073047	7902698606	107	7
7270	995073047>922394800	995073047	922394800	107	6
7271	995073047>991988255	995073047	991988255	107	6

7272 rows × 5 columns

	rule	product_A	product_B	product_C	freq_left	freq_all
0	1008435393>454021998&1008455319	1008435393	454021998	1008455319	55	6
1	1008435393>454021998&1336146681	1008435393	454021998	1336146681	55	18
2	1008435393>454021998&1492482010	1008435393	454021998	1492482010	55	7
3	1008435393>454021998&1939300455	1008435393	454021998	1939300455	55	9
4	1008435393>454021998&454021999	1008435393	454021998	454021999	55	7
...	...	...	...	...	...	...
3478	959273129>959269904&1167299260	959273129	959269904	1167299260	30	6
3479	982727004>1512054476&2188141395	982727004	1512054476	2188141395	28	6

	rule	product_A	product_B	product_C	freq_left	freq_all
3480	982727004>1512054476&2523757071	982727004	1512054476	2523757071	28	6
3481	982727004>2188141395&2523757071	982727004	2188141395	2523757071	28	6
3482	991988255>926159335&2057880096	991988255	926159335	2057880096	72	8

3483 rows × 6 columns

	rule	product_A	product_B	product_C	freq_left	freq_all
8	1008435393&1008455319>454021998	1008435393	1008455319	454021998	17	6
9	1008435393&1336146681>454021998	1008435393	1336146681	454021998	22	18
10	1008435393&1492482010>454021998	1008435393	1492482010	454021998	8	7
11	1008435393&1939300455>454021998	1008435393	1939300455	454021998	11	9
43	1019107054&1697382283>684827047	1019107054	1697382283	684827047	8	6
...	...	...	...	...	...	...
14176	982727004&2188141395>1512054476	982727004	2188141395	1512054476	16	6
14177	982727004&2188141395>2523757071	982727004	2188141395	2523757071	16	6
14178	982727004&2523757071>1512054476	982727004	2523757071	1512054476	8	6
14179	982727004&2523757071>2188141395	982727004	2523757071	2188141395	8	6
14196	991988255&2057880096>926159335	991988255	2057880096	926159335	14	8

3483 rows × 6 columns

In [25]:
```
part1 = asc_rules1a[['rule','freq_left','freq_all']]
part2 = asc_rules2a[['rule','freq_left','freq_all']]
part3 = asc_rules3[['rule','freq_left','freq_all']]
part4 = pd.concat([part1,part2,part3]).reset_index(drop=True)
```

executed in 722ms, finished 15:52:43 2020-12-18

Calculate Confidence

In [26]:
```
part4['score'] = 1000*(part4['freq_all']/part4['freq_left'])
part4['confidence'] = part4['score'].apply(lambda x:int(x))
part4
```

executed in 4.27s, finished 15:52:47 2020-12-18

Out[26]:

	rule	freq_left	freq_all	score	confidence
0	100242812>80361758	17	8	470.588235	470
1	100242812>89031406	17	6	352.941176	352
2	1003153762>1016449477	18	7	388.888889	388
3	1006024995>2727415265	35	6	171.428571	171
4	1006024995>866012366	35	6	171.428571	171
...	...	...	...	...	...
14233	982727004&2188141395>1512054476	16	6	375.000000	375
14234	982727004&2188141395>2523757071	16	6	375.000000	375
14235	982727004&2523757071>1512054476	8	6	750.000000	750
14236	982727004&2523757071>2188141395	8	6	750.000000	750
14237	991988255&2057880096>926159335	14	8	571.428571	571

14238 rows × 5 columns

In [27]:
```
pre_conf_rule = asc_rules.merge(part4,on='rule')
pre_conf_rule
```

executed in 210ms, finished 15:52:47 2020-12-18

Out[27]:

	rule	item_A1	item_A2	item_B1	item_B2	freq_left	freq_all	score	confidence
0	100242812>80361758	100242812	0	80361758	0	17	8	470.588235	470
1	100242812>89031406	100242812	0	89031406	0	17	6	352.941176	352
2	1003153762>1016449477	1003153762	0	1016449477	0	18	7	388.888889	388
3	1006024995>2727415265	1006024995	0	2727415265	0	35	6	171.428571	171
4	1006024995>866012366	1006024995	0	866012366	0	35	6	171.428571	171
...	...	...	...	...	...	...	...	...	...
14233	995073047>3202007524	995073047	0	3202007524	0	107	11	102.803738	102
14234	995073047>651958908	995073047	0	651958908	0	107	14	130.841121	130
14235	995073047>7902698606	995073047	0	7902698606	0	107	7	65.420561	65
14236	995073047>922394800	995073047	0	922394800	0	107	6	56.074766	56
14237	995073047>991988255	995073047	0	991988255	0	107	6	56.074766	56

14238 rows × 9 columns

In [28]:
```
conf_rule = asc_rules[['rule']].merge(part4[['rule','confidence']],on='rule')
conf_rule
```

executed in 256ms, finished 15:52:47 2020-12-18

Out[28]:

	rule	confidence
0	100242812>80361758	470
1	100242812>89031406	352
2	1003153762>1016449477	388
3	1006024995>2727415265	171
4	1006024995>866012366	171
...	...	...
14233	995073047>3202007524	102
14234	995073047>651958908	130
14235	995073047>7902698606	65
14236	995073047>922394800	56
14237	995073047>991988255	56

14238 rows × 2 columns

In [29]:
```
#conf_rule.to_csv('market_basket.csv',index=False)
```

executed in 25ms, finished 15:52:47 2020-12-18

Bonus

In [30]:
```
pre_conf_rule[(pre_conf_rule.confidence>800)&(pre_conf_rule.freq_left<15)]
```

executed in 220ms, finished 15:52:48 2020-12-18

Out[30]:

	rule	item_A1	item_A2	item_B1	item_B2	freq_left	freq_all	score	confidence
10	1008435393&1492482010>454021998	1008435393	1492482010	454021998	0	8	7	875.000000	875
11	1008435393&1939300455>454021998	1008435393	1939300455	454021998	0	11	9	818.181818	818
72	1040759990>238045246	1040759990	0	238045246	0	6	6	1000.000000	1000
84	105461158>67334309	105461158	0	67334309	0	7	6	857.142857	857
85	105461158>67372772	105461158	0	67372772	0	7	6	857.142857	857
...	...	...	...	...	...	...	...	...	...
14096	937914384&1400873429>1400976291	937914384	1400873429	1400976291	0	7	6	857.142857	857
14134	953197492>55976260	953197492	0	55976260	0	7	6	857.142857	857
14150	959273129&1167325068>1167299260	959273129	1167325068	1167299260	0	11	9	818.181818	818
14163	967624910>967178365	967624910	0	967178365	0	8	7	875.000000	875
14166	97408609>6507846415	97408609	0	6507846415	0	8	7	875.000000	875

2423 rows × 9 columns

In [31]:
```
pre_conf_rule[(pre_conf_rule.confidence>800)&(pre_conf_rule.freq_left>40)]
```

executed in 176ms, finished 15:52:48 2020-12-18

Out[31]:

	rule	item_A1	item_A2	item_B1	item_B2	freq_left	freq_all	score	confidence
1350	1469688069>631161363	1469688069	0	631161363	0	57	53	929.824561	929
2164	1629699394&2697336272>2399078400	1629699394	2697336272	2399078400	0	41	33	804.878049	804
2325	1629823151&2697336272>1897730934	1629823151	2697336272	1897730934	0	45	39	866.666667	866
4222	1897730934&2399078400>2697336272	1897730934	2399078400	2697336272	0	177	167	943.502825	943
4233	1897730934&2697336272>2399078400	1897730934	2697336272	2399078400	0	188	167	888.297872	888
4278	1897730934>2399078400	1897730934	0	2399078400	0	215	177	823.255814	823
4280	1897730934>2697336272	1897730934	0	2697336272	0	215	188	874.418605	874
5332	2109291307&2399078400>2697336272	2109291307	2399078400	2697336272	0	63	54	857.142857	857
5337	2109291307&2697336272>1897730934	2109291307	2697336272	1897730934	0	62	51	822.580645	822
5340	2109291307&2697336272>2399078400	2109291307	2697336272	2399078400	0	62	54	870.967742	870
6390	2399078400&2697336272>1897730934	2399078400	2697336272	1897730934	0	190	167	878.947368	878
6418	2399078400>1897730934	2399078400	0	1897730934	0	204	177	867.647059	867
6424	2399078400>1897730934&2697336272	2399078400	0	1897730934	2697336272	204	167	818.627451	818
6437	2399078400>2697336272	2399078400	0	2697336272	0	204	190	931.372549	931
6803	2559166447>2559166449	2559166447	0	2559166449	0	42	37	880.952381	880
7155	2697336272>1897730934	2697336272	0	1897730934	0	217	188	866.359447	866
7173	2697336272>2399078400	2697336272	0	2399078400	0	217	190	875.576037	875
7693	2847684782>2847551273	2847684782	0	2847551273	0	42	37	880.952381	880

Analysis: Combination (two or three) products of **1897730934, 2399078400, and 2697336272** purchased simultaneously in large portion of one transaction. Giving pricing promotion of these products may interest same user to purchase often. Maybe it would attracts another user too.

In [ ]: