方法一: 使用apyori套件

In [1]:

```python
## Import package
from apyori import apriori
## Data 自行定義數據
market_data = [['T-Shirt','Pants','Jeans','Jersy','Socks','Basketball','Bottle','Shorts'],
 ['T-Shirt','Jeans'],
 ['Jersy','Basketball','Socks','Bottle'],
 ['Jeans','Pants','Bottle'],
 ['Shorts','Basketball'],
 ['Shorts','Jersy'],
 ['T-Shirt'],
 ['Basketball','Jersy'],
 ]
association_rules = apriori(market_data, min_support=0.2, min_confidence=0.2, min_lift=2, max_length=2)
association_results = list(association_rules)
##print(association_results )
for product in association_results:
 #print(product) # ex. RelationRecord(items=frozenset({'Basketball', 'Socks'}), support=0.25, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Basketball'}), items_add=frozenset({'Socks'}), confidence=0.5, lift=2.0), OrderedStatistic(items_base=frozenset({'Socks'}), items_add=frozenset({'Basketball'}), confidence=1.0, lift=2.0)])
 pair = product[0]
 ##print(pair) ## ex. frozenset({'Basketball', 'Socks'})
 products = [x for x in pair]
 print(products) # ex. ['Basketball', 'Socks']
 print("Rule: " + products[0] + " →" + products[1])
 print("Support: " + str(product[1]))
 print("Lift: " + str(product[2][0][3]))
 print("==================================")
```

```
['Basketball', 'Socks']
Rule: Basketball →Socks
Support: 0.25
Lift: 2.0
==================================
['Pants', 'Bottle']
Rule: Pants →Bottle
Support: 0.25
Lift: 2.6666666666666665
==================================
['Bottle', 'Socks']
Rule: Bottle →Socks
Support: 0.25
Lift: 2.6666666666666665
==================================
['Pants', 'Jeans']
Rule: Pants →Jeans
Support: 0.25
Lift: 2.6666666666666665
==================================
['Jersy', 'Socks']
Rule: Jersy →Socks
Support: 0.25
Lift: 2.0
==================================
```

方法二: 使用mlxtend套件，將數據轉換成one-hot編碼

In [1]:

```
## Import Package
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
## Data 自行定義數據
market_data = {
 'Transaction ID': [1,2,3,4,5,6,7,8],
 'Items':[['T-Shirt','Pants','Jeans','Jersy','Socks','Basketball','Bottle','Shorts'],
 ['T-Shirt','Jeans'],
 ['Jersy','Basketball','Socks','Bottle'],
 ['Jeans','Pants','Bottle'],
 ['Shorts','Basketball'],
 ['Shorts','Jersy'],
 ['T-Shirt'],
 ['Basketball','Jersy'],
 ]}
## 轉成DataFrame
data = pd.DataFrame(market_data)
## 讓DataFrame 能呈現的寬度大一點
pd.options.display.max_colwidth = 100
## 轉成數值編碼，目前都是字串的組合
data_id = data.drop('Items', 1)
data_items = data.Items.str.join(',')
## 轉成數值
data_items = data_items.str.get_dummies(',')
## 接上Transaction ID
data = data_id.join(data_items)
## 計算支持度 Support
Support_items = apriori(data[['T-Shirt','Pants','Jeans','Jersy','Socks','Basketball','Bottle','Shorts']], min_support=0.20, use_colnames = True)
## 計算關聯規則 Association Rule
Association_Rules = association_rules(Support_items, metric = 'lift', min_threshold=1)

Association Rules
```

Out[1]:

	antecedents	consequents	antecedent support	consequent support	support
0	(Jeans)	(T-Shirt)	0.375	0.375	0.25
1	(T-Shirt)	(Jeans)	0.375	0.375	0.25
2	(Pants)	(Jeans)	0.250	0.375	0.25
3	(Jeans)	(Pants)	0.375	0.250	0.25
4	(Pants)	(Bottle)	0.250	0.375	0.25
...	...	...	...	...	..
63	(Basketball, Bottle)	(Jersy, Socks)	0.250	0.250	0.25
64	(Jersy)	(Socks, Basketball, Bottle)	0.500	0.250	0.25
65	(Socks)	(Jersy, Basketball, Bottle)	0.250	0.250	0.25
66	(Basketball)	(Jersy, Socks, Bottle)	0.500	0.250	0.25
67	(Bottle)	(Jersy, Socks, Basketball)	0.375	0.250	0.25

68 rows × 9 columns

In [13]:

```
## Import Package
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
## Data 自行定義數據
market_data = {
 'Transaction ID': [1,2,3,4,5,6,7,8],
 'Items':[['T-Shirt','Pants','Jeans','Jersy','Socks','Basketball','Bottle','S
horts'],
  ['T-Shirt','Jeans'],
  ['Jersy','Basketball','Socks','Bottle'],
  ['Jeans','Pants','Bottle'],
  ['Shorts','Basketball'],
  ['Shorts','Jersy'],
  ['T-Shirt'],
  ['Basketball','Jersy'],
  ]}
## 轉成DataFrame
data = pd.DataFrame(market_data)
## 讓DataFrame 能呈現的寬度大一點
pd.options.display.max_colwidth = 100
## 轉成數值編碼，目前都是字串的組合
data_id = data.drop('Items', 1)
data_items = data.Items.str.join(',')
data_items = data_items.str.get_dummies(',')
data_id.join(data_items).head()
```

Out[13]:

	Transaction ID	Basketball	Bottle	Jeans	Jersy	Pants	Shorts	S
0	1	1	1	1	1	1	1	
1	2	0	0	1	0	0	0	
2	3	1	1	0	1	0	0	
3	4	0	1	1	0	1	0	
4	5	1	0	0	0	0	1	

In [ ]:

In [2]:

```
## Import Package
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
## Data 自行定義數據
market_data = {
 'Transaction ID': [1,2,3,4,5,6,7,8],
 'Items':[['T-Shirt','Pants','Jeans','Jersy','Socks','Basketball','Bottle','S
horts'],
  ['T-Shirt','Jeans'],
  ['Jersy','Basketball','Socks','Bottle'],
  ['Jeans','Pants','Bottle'],
  ['Shorts','Basketball'],
  ['Shorts','Jersy'],
  ['T-Shirt'],
  ['Basketball','Jersy'],
 ]}
```

In [4]:

```
df = pd.DataFrame(market_data)
df.head()
```

Out[4]:

	Transaction ID	Items
0	1	[T-Shirt, Pants, Jeans, Jersy, Socks, Basketball, Bottle, Shorts]
1	2	[T-Shirt, Jeans]
2	3	[Jersy, Basketball, Socks, Bottle]
3	4	[Jeans, Pants, Bottle]
4	5	[Shorts, Basketball]

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
## 轉成DataFrame
```

方法三: 基本分析方法

In [14]:

```
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

In [15]:

```
data = {'ID': [1,2,3,4,5,6],
        'Onion':[1,0,0,1,1,1],
        'Potato': [1,1,0,1,1,1],
        'Burger':[1,1,0,0,1,1],
        'Milk':[0,1,1,1,0,1],
        'Beer': [0,0,1,0,1,0]}
```

In [16]:

```
df = pd.DataFrame(data)
```

In [17]:

```
df = df[['ID', 'Onion', 'Potato', 'Burger', 'Milk', 'Beer']]
```

In [19]:

```
df.head(3)
```

Out[19]:

	ID	Onion	Potato	Burger	Milk	Beer
0	1	1	1	1	0	0
1	2	0	1	1	1	0
2	3	0	0	0	1	1

In [20]:

```
df = df.drop('ID',axis=1)
```

In [21]:

```
frequent_itemsets = apriori(df, min_support =0.50, use_colnames=True)
```

In [10]:

```
frequent_itemsets
```

Out[10]:

	support	itemsets
0	0.666667	(Onion)
1	0.833333	(Potato)
2	0.666667	(Burger)
3	0.666667	(Milk)
4	0.666667	(Onion, Potato)
5	0.500000	(Burger, Onion)
6	0.666667	(Burger, Potato)
7	0.500000	(Potato, Milk)
8	0.500000	(Burger, Onion, Potato)

In [11]:

```
# min_threshlod = 1 最小的lift值須等於1 不然沒有意義
rules = association_rules(frequent_itemsets, metric = 'lift', min_threshold=1
)
```

In [12]:

```
rules
```

Out[12]:

	antecedents	consequents	antecedent support	consequent support	suppor
0	(Onion)	(Potato)	0.666667	0.833333	0.66666
1	(Potato)	(Onion)	0.833333	0.666667	0.66666
2	(Burger)	(Onion)	0.666667	0.666667	0.50000
3	(Onion)	(Burger)	0.666667	0.666667	0.50000
4	(Burger)	(Potato)	0.666667	0.833333	0.66666
5	(Potato)	(Burger)	0.833333	0.666667	0.66666
6	(Burger, Onion)	(Potato)	0.500000	0.833333	0.50000
7	(Burger, Potato)	(Onion)	0.666667	0.666667	0.50000
8	(Onion, Potato)	(Burger)	0.666667	0.666667	0.50000
9	(Burger)	(Onion, Potato)	0.666667	0.666667	0.50000
10	(Onion)	(Burger, Potato)	0.666667	0.666667	0.50000
11	(Potato)	(Burger, Onion)	0.833333	0.500000	0.50000

In [13]:

```
rules [( rules['lift']>1.125) & (rules['confidence']>0.8)]
```

Out[13]:

	antecedents	consequents	antecedent support	consequent support	support
0	(Onion)	(Potato)	0.666667	0.833333	0.666667
4	(Burger)	(Potato)	0.666667	0.833333	0.666667
6	(Burger, Onion)	(Potato)	0.500000	0.833333	0.500000

ONE HOT ENCODER

方法二: 如何把項目集資料轉化為可以分析的格式(ONE HOT ENCODER)

In [11]:

```
retail_shopping_basket = {'ID':[1,2,3,4,5,6],
                          'Basket':[['Beer', 'Diaper', 'Pretzels', 'Chips', 'As
pirin'],
                          ['Diaper', 'Beer', 'Chips', 'Lotion', 'Juice', 'Baby
food', 'Milk'],
                          ['Soda', 'Chips', 'Milk'],
                          ['Soup', 'Beer', 'Diaper', 'Milk', 'IceCream'],
                          ['Soda', 'Coffee', 'Milk', 'Bread'],
                          ['Beer', 'Chips']
                                    ]
                          }

retail = pd.DataFrame(retail_shopping_basket)
```

In [12]:
```
retail
```
Out[12]:

	ID	Basket
0	1	[Beer, Diaper, Pretzels, Chips, Aspirin]
1	2	[Diaper, Beer, Chips, Lotion, Juice, Babyfood,...
2	3	[Soda, Chips, Milk]
3	4	[Soup, Beer, Diaper, Milk, IceCream]
4	5	[Soda, Coffee, Milk, Bread]
5	6	[Beer, Chips]

In [13]:
```
retail = retail[['ID', 'Basket']]
```
In [14]:
```
pd.options.display.max_colwidth = 100
```
In [15]:
```
retail
```
Out[15]:

	ID	Basket
0	1	[Beer, Diaper, Pretzels, Chips, Aspirin]
1	2	[Diaper, Beer, Chips, Lotion, Juice, Babyfood, Milk]
2	3	[Soda, Chips, Milk]
3	4	[Soup, Beer, Diaper, Milk, IceCream]
4	5	[Soda, Coffee, Milk, Bread]
5	6	[Beer, Chips]

In [16]:

```
retail_id = retail.drop('Basket', axis =1)
retail_id
```

Out[16]:

	ID
0	1
1	2
2	3
3	4
4	5
5	6

In [17]:

```
retail_Basket = retail.Basket.str.join(',')
retail_Basket
```

Out[17]:

```
0               Beer,Diaper,Pretzels,Chips,Aspirin
1    Diaper,Beer,Chips,Lotion,Juice,Babyfood,Milk
2                                  Soda,Chips,Milk
3                   Soup,Beer,Diaper,Milk,IceCream
4                           Soda,Coffee,Milk,Bread
5                                       Beer,Chips
Name: Basket, dtype: object
```

In [18]:

```
retail_Basket = retail_Basket.str.get_dummies(',')
retail_Basket
```

Out[18]:

	Aspirin	Babyfood	Beer	Bread	Chips	Coffee	Diaper	IceCre
0	1	0	1	0	1	0	1	
1	0	1	1	0	1	0	1	
2	0	0	0	0	1	0	0	
3	0	0	1	0	0	0	1	
4	0	0	0	1	0	1	0	
5	0	0	1	0	1	0	0	

In [19]:

```
retail = retail_id.join(retail_Basket)
retail
```

Out[19]:

	ID	Aspirin	Babyfood	Beer	Bread	Chips	Coffee	Diaper	Ice
0	1	1	0	1	0	1	0	1	
1	2	0	1	1	0	1	0	1	
2	3	0	0	0	0	1	0	0	
3	4	0	0	1	0	0	0	1	
4	5	0	0	0	1	0	1	0	
5	6	0	0	1	0	1	0	0	

In [20]:

```
frequent_itemsets_2 = apriori(retail.drop('ID',1), use_colnames=True)
```

In [21]:

```
frequent itemsets 2
```

Out[21]:

	support	itemsets
0	0.666667	(Beer)
1	0.666667	(Chips)
2	0.500000	(Diaper)
3	0.666667	(Milk)
4	0.500000	(Beer, Chips)
5	0.500000	(Beer, Diaper)

In [22]:

```
association rules(frequent itemsets 2, metric='lift')
```

Out[22]:

	antecedents	consequents	antecedent support	consequent support	support
0	(Beer)	(Chips)	0.666667	0.666667	0.5
1	(Chips)	(Beer)	0.666667	0.666667	0.5
2	(Beer)	(Diaper)	0.666667	0.500000	0.5
3	(Diaper)	(Beer)	0.500000	0.666667	0.5

方法三: 電影題材關聯 導入數據庫CSV檔案進行分析

In [15]:

```
movies = pd.read csv('movies.csv')
```

In [16]:

```
movies.head(10)
```

Out[16]:

	movieId	title	ger
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fan
1	2	Jumanji (1995)	Adventure|Children|Fan
2	3	Grumpier Old Men (1995)	Comedy|Roma
3	4	Waiting to Exhale (1995)	Comedy|Drama|Roma
4	5	Father of the Bride Part II (1995)	Com
5	6	Heat (1995)	Action|Crime|Thr
6	7	Sabrina (1995)	Comedy|Roma
7	8	Tom and Huck (1995)	Adventure|Chilc
8	9	Sudden Death (1995)	Ac
9	10	GoldenEye (1995)	Action|Adventure|Thr

In [17]:

```
movies_ohe = movies.drop('genres', 1).join(movies.genres.str.get_dummies())
```

In [18]:

```
pd.options.display.max_columns=100
```

In [19]:

```
movies_ohe.head()
```

Out[19]:

	movieId	title	(no genres listed)	Action	Adventure	Animation	Ch
0	1	Toy Story (1995)	0	0	1	1	
1	2	Jumanji (1995)	0	0	1	0	
2	3	Grumpier Old Men (1995)	0	0	0	0	
3	4	Waiting to Exhale (1995)	0	0	0	0	
4	5	Father of the Bride Part II (1995)	0	0	0	0	

In [23]:

```
movies_ohe.shape
```

Out[23]:

```
(9742, 20)
```

In [36]:

```
movies_ohe.columns
movies_ohe.set_index(['movieId','title'],inplace=True)
## inplace = True : 不創建新的對象，直接對原本的對象進行修改
## inplace = False : 對數據進行修改，創建並返回新的對象承載它的修改結果
```

Out[36]:

```
Index(['(no genres listed)', 'Action', 'Adventure', 'Animatio
n', 'Children',
       'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
'Film-Noir',
       'Horror', 'IMAX', 'Musical', 'Mystery', 'Romance', 'Sci
-Fi', 'Thriller',
       'War', 'Western'],
      dtype='object')
```

In [26]:

```
movies_ohe.head()
```

Out[26]:

		(no genres listed)	Action	Adventure	Animation	Child
movieId	**title**					
1	**Toy Story (1995)**	0	0	1	1	
2	**Jumanji (1995)**	0	0	1	0	
3	**Grumpier Old Men (1995)**	0	0	0	0	
4	**Waiting to Exhale (1995)**	0	0	0	0	
5	**Father of the Bride Part II (1995)**	0	0	0	0	

In [27]:

```
frequent_itemsets_movies = apriori(movies_ohe,use_colnames=True, min_support=
0.025)
```

In [28]:

```
frequent_itemsets_movies
```

Out[28]:

	support	itemsets
0	0.187641	(Action)
1	0.129645	(Adventure)
2	0.062718	(Animation)
3	0.068158	(Children)
4	0.385547	(Comedy)
5	0.123075	(Crime)
6	0.045165	(Documentary)
7	0.447649	(Drama)
8	0.079963	(Fantasy)
9	0.100390	(Horror)
10	0.034285	(Musical)
11	0.058817	(Mystery)
12	0.163827	(Romance)
13	0.100595	(Sci-Fi)
14	0.194416	(Thriller)
15	0.039212	(War)
16	0.062615	(Action, Adventure)
17	0.044036	(Action, Comedy)
18	0.042907	(Action, Crime)
19	0.054301	(Drama, Action)
20	0.046294	(Sci-Fi, Action)
21	0.067235	(Thriller, Action)
22	0.025354	(Adventure, Animation)
23	0.032026	(Adventure, Children)
24	0.040957	(Adventure, Comedy)
25	0.031924	(Drama, Adventure)

	support	itemsets
26	0.034285	(Fantasy, Adventure)
27	0.031410	(Sci-Fi, Adventure)
28	0.031000	(Animation, Children)
29	0.027612	(Animation, Comedy)
30	0.037159	(Comedy, Children)
31	0.035414	(Crime, Comedy)
32	0.103983	(Drama, Comedy)
33	0.030076	(Fantasy, Comedy)
34	0.090741	(Romance, Comedy)
35	0.065387	(Drama, Crime)
36	0.058407	(Thriller, Crime)
37	0.029563	(Drama, Mystery)
38	0.095874	(Drama, Romance)
39	0.085403	(Drama, Thriller)
40	0.030589	(Drama, War)
41	0.047116	(Thriller, Horror)
42	0.036338	(Thriller, Mystery)
43	0.031000	(Sci-Fi, Thriller)
44	0.035106	(Drama, Romance, Comedy)
45	0.031718	(Drama, Thriller, Crime)

In [29]:

```
rules_movies = association_rules(frequent_itemsets_movies, metric='lift', min
 threshold=1.25)
```

In [30]:

```
rules movies
```

Out[30]:

	antecedents	consequents	antecedent support	consequent support	suppor
0	(Action)	(Adventure)	0.187641	0.129645	0.06261
1	(Adventure)	(Action)	0.129645	0.187641	0.06261
2	(Action)	(Crime)	0.187641	0.123075	0.04290
3	(Crime)	(Action)	0.123075	0.187641	0.04290
4	(Sci-Fi)	(Action)	0.100595	0.187641	0.04629
5	(Action)	(Sci-Fi)	0.187641	0.100595	0.04629
6	(Thriller)	(Action)	0.194416	0.187641	0.06723
7	(Action)	(Thriller)	0.187641	0.194416	0.06723
8	(Adventure)	(Animation)	0.129645	0.062718	0.02535
9	(Animation)	(Adventure)	0.062718	0.129645	0.02535
10	(Adventure)	(Children)	0.129645	0.068158	0.03202
11	(Children)	(Adventure)	0.068158	0.129645	0.03202
12	(Fantasy)	(Adventure)	0.079963	0.129645	0.03428
13	(Adventure)	(Fantasy)	0.129645	0.079963	0.03428
14	(Sci-Fi)	(Adventure)	0.100595	0.129645	0.03141
15	(Adventure)	(Sci-Fi)	0.129645	0.100595	0.03141
16	(Animation)	(Children)	0.062718	0.068158	0.03100
17	(Children)	(Animation)	0.068158	0.062718	0.03100
18	(Comedy)	(Children)	0.385547	0.068158	0.03715
19	(Children)	(Comedy)	0.068158	0.385547	0.03715
20	(Romance)	(Comedy)	0.163827	0.385547	0.09074
21	(Comedy)	(Romance)	0.385547	0.163827	0.09074
22	(Thriller)	(Crime)	0.194416	0.123075	0.05840
23	(Crime)	(Thriller)	0.123075	0.194416	0.05840
24	(Drama)	(Romance)	0.447649	0.163827	0.09587

	antecedents	consequents	antecedent support	consequent support	suppor
25	(Romance)	(Drama)	0.163827	0.447649	0.09587
26	(Drama)	(War)	0.447649	0.039212	0.03058
27	(War)	(Drama)	0.039212	0.447649	0.03058
28	(Thriller)	(Horror)	0.194416	0.100390	0.04711
29	(Horror)	(Thriller)	0.100390	0.194416	0.04711
30	(Thriller)	(Mystery)	0.194416	0.058817	0.03633
31	(Mystery)	(Thriller)	0.058817	0.194416	0.03633
32	(Sci-Fi)	(Thriller)	0.100595	0.194416	0.03100
33	(Thriller)	(Sci-Fi)	0.194416	0.100595	0.03100
34	(Drama, Comedy)	(Romance)	0.103983	0.163827	0.03510
35	(Romance)	(Drama, Comedy)	0.163827	0.103983	0.03510
36	(Drama, Thriller)	(Crime)	0.085403	0.123075	0.03171
37	(Drama, Crime)	(Thriller)	0.065387	0.194416	0.03171
38	(Thriller)	(Drama, Crime)	0.194416	0.065387	0.03171
39	(Crime)	(Drama, Thriller)	0.123075	0.085403	0.03171

In [31]:

```
rules_movies[(rules_movies.lift>4)].sort_values(by=['lift'], ascending=False)
```

Out[31]:

	antecedents	consequents	antecedent support	consequent support	support
16	(Animation)	(Children)	0.062718	0.068158	0.031
17	(Children)	(Animation)	0.068158	0.062718	0.031

In [32]:

```
movies[(movies.genres.str.contains('Children')) & (movies.genres.str.contains('Animation'))]
```

Out[32]:

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy
12	13	Balto (1995)	Adventure|Animation|
44	48	Pocahontas (1995)	Animation|Children|Drama|Musical|F
205	239	Goofy Movie, A (1995)	Animation|Children|Comedy|F
272	313	Swan Princess, The (1994)	Animation|
...	...	...	
9629	178827	Paddington 2 (2017)	Adventure|Animation|Children
9657	180987	Ferdinand (2017)	Animation|Children
9664	182293	Hare-um Scare-um (1939)	Animation|Children
9666	182299	Porky's Hare Hunt (1938)	Animation|Children
9708	187541	Incredibles 2 (2018)	Action|Adventure|Animation|

302 rows × 3 columns

In [68]:

```
movies[(movies.genres.str.contains('Children')) & (~movies.genres.str.contains('Animation'))]
```

Out[68]:

	movieId	title	genres
1	2	Jumanji (1995)	Adventure|Children|Fantasy
7	8	Tom and Huck (1995)	Adventure|Children
26	27	Now and Then (1995)	Children|Drama
32	34	Babe (1995)	Children|Drama
34	38	It Takes Two (1995)	Children|Comedy
...	...	...	...
9636	179401	Jumanji: Welcome to the Jungle (2017)	Action|Adventure|Children
9670	182731	Pixel Perfect (2004)	Children|Comedy|Sci-Fi
9679	183301	The Tale of the Bunny Picnic (1986)	Children
9697	184987	A Wrinkle in Time (2018)	Adventure|Children|Fantasy|Sci-Fi
9710	187595	Solo: A Star Wars Story (2018)	Action|Adventure|Children|Sci-Fi

362 rows × 3 columns