Appriori Algorithm

In [1]:
```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
```

In [2]:
```
df = pd.read_csv('BreadBasket.csv', sep = ',')
df.head()
```

Out[2]:

	Tx	products
0	0	MILK,BREAD,BISCUIT
1	1	BREAD,MILK,BISCUIT,CORNFLAKES
2	2	BREAD,TEA,BOURNVITA
3	3	JAM,MAGGI,BREAD,MILK
4	4	MAGGI,TEA,BISCUIT

In [3]:
```
data = list(df["products"].apply(lambda x:x.split(",") ))
data
```

Out[3]:
```
[['MILK', 'BREAD', 'BISCUIT'],
 ['BREAD', 'MILK', 'BISCUIT', 'CORNFLAKES'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['JAM', 'MAGGI', 'BREAD', 'MILK'],
 ['MAGGI', 'TEA', 'BISCUIT'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['MAGGI', 'TEA', 'CORNFLAKES'],
 ['MAGGI', 'BREAD', 'TEA', 'BISCUIT'],
 ['JAM', 'MAGGI', 'BREAD', 'TEA'],
 ['BREAD', 'MILK'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'COCK'],
 ['BREAD', 'SUGER', 'BISCUIT'],
 ['COFFEE', 'SUGER', 'CORNFLAKES'],
 ['BREAD', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['TEA', 'MILK', 'COFFEE', 'CORNFLAKES']]
```

In [4]:
```
from mlxtend.preprocessing import TransactionEncoder
a = TransactionEncoder()
a_data = a.fit(data).transform(data)
df = pd.DataFrame(a_data,columns=a.columns_)
df
```

Out[4]:

	BISCUIT	BOURNVITA	BREAD	COCK	COFFEE	CORNFLAKES	JAM	MAGGI	MILK	SUGER	TEA
0	True	False	True	False	False	False	False	False	True	False	False
1	True	False	True	False	False	True	False	False	True	False	False
2	False	True	True	False	False	False	False	False	False	False	True
3	False	False	True	False	False	False	True	True	True	False	False
4	True	False	False	False	False	False	False	True	False	False	True
5	False	True	True	False	False	False	False	False	False	False	True
6	False	False	False	False	False	True	False	True	False	False	True
7	True	False	True	False	False	False	False	True	False	False	True
8	False	False	True	False	False	False	True	True	False	False	True
9	False	False	True	False	False	False	False	False	True	False	False
10	True	False	False	True	True	True	False	False	False	False	False
11	True	False	False	True	True	True	False	False	False	False	False
12	False	True	False	False	True	False	False	False	False	True	False
13	False	False	True	True	True	False	False	False	False	False	False
14	True	False	True	False	False	False	False	False	False	True	False
15	False	False	False	False	True	True	False	False	False	True	False
16	False	True	True	False	False	False	False	False	False	True	False
17	False	False	True	False	True	False	False	False	False	True	False
18	False	False	True	False	True	False	False	False	False	True	False
19	False	False	False	False	True	True	False	False	True	False	True

In [5]:
```
from mlxtend.frequent_patterns import apriori

apriori(df, min_support=0.2)
```

Out[5]:

	support	itemsets
0	0.35	(0)
1	0.20	(1)
2	0.65	(2)
3	0.40	(4)
4	0.30	(5)
5	0.25	(7)
6	0.25	(8)
7	0.30	(9)
8	0.35	(10)
9	0.20	(0, 2)
10	0.20	(8, 2)
11	0.20	(9, 2)
12	0.20	(2, 10)
13	0.20	(4, 5)
14	0.20	(9, 4)
15	0.20	(10, 7)

In [6]:
```
apriori(df, min_support=0.2, use_colnames=True)
```

Out[6]:

	support	itemsets
0	0.35	(BISCUIT)
1	0.20	(BOURNVITA)
2	0.65	(BREAD)
3	0.40	(COFFEE)
4	0.30	(CORNFLAKES)
5	0.25	(MAGGI)
6	0.25	(MILK)
7	0.30	(SUGER)
8	0.35	(TEA)
9	0.20	(BISCUIT, BREAD)
10	0.20	(BREAD, MILK)
11	0.20	(BREAD, SUGER)
12	0.20	(BREAD, TEA)
13	0.20	(COFFEE, CORNFLAKES)
14	0.20	(COFFEE, SUGER)
15	0.20	(MAGGI, TEA)

In [7]:
```
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

Out[7]:

	support	itemsets	length
0	0.35	(BISCUIT)	1
1	0.20	(BOURNVITA)	1
2	0.65	(BREAD)	1
3	0.40	(COFFEE)	1
4	0.30	(CORNFLAKES)	1
5	0.25	(MAGGI)	1
6	0.25	(MILK)	1
7	0.30	(SUGER)	1
8	0.35	(TEA)	1
9	0.20	(BISCUIT, BREAD)	2
10	0.20	(BREAD, MILK)	2
11	0.20	(BREAD, SUGER)	2
12	0.20	(BREAD, TEA)	2
13	0.20	(COFFEE, CORNFLAKES)	2
14	0.20	(COFFEE, SUGER)	2
15	0.20	(MAGGI, TEA)	2

In [8]:
```
from mlxtend.frequent_patterns import association_rules

association_rules(frequent_itemsets)
```

Out[8]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(MILK)	(BREAD)	0.25	0.65	0.2	0.8	1.230769	0.0375	1.75	0.25
1	(MAGGI)	(TEA)	0.25	0.35	0.2	0.8	2.285714	0.1125	3.25	0.75

FP Grouth Algorithm

In [9]:
```
from mlxtend.frequent_patterns import fpgrowth

fpgrowth(df, min_support=0.2)
```

Out[9]:

	support	itemsets
0	0.65	(2)
1	0.35	(0)
2	0.25	(8)
3	0.30	(5)
4	0.35	(10)
5	0.20	(1)
6	0.25	(7)
7	0.40	(4)
8	0.30	(9)
9	0.20	(0, 2)
10	0.20	(8, 2)
11	0.20	(4, 5)
12	0.20	(10, 2)
13	0.20	(10, 7)
14	0.20	(9, 4)
15	0.20	(9, 2)

In [10]:
```
fpgrowth(df, min_support=0.2, use_colnames=True)
```

Out[10]:

	support	itemsets
0	0.65	(BREAD)
1	0.35	(BISCUIT)
2	0.25	(MILK)
3	0.30	(CORNFLAKES)
4	0.35	(TEA)
5	0.20	(BOURNVITA)
6	0.25	(MAGGI)
7	0.40	(COFFEE)
8	0.30	(SUGER)
9	0.20	(BISCUIT, BREAD)
10	0.20	(BREAD, MILK)
11	0.20	(COFFEE, CORNFLAKES)
12	0.20	(BREAD, TEA)
13	0.20	(MAGGI, TEA)
14	0.20	(COFFEE, SUGER)
15	0.20	(BREAD, SUGER)

In [11]:
```
from mlxtend.frequent_patterns import association_rules

association_rules(frequent_itemsets)
```

Out[11]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(MILK)	(BREAD)	0.25	0.65	0.2	0.8	1.230769	0.0375	1.75	0.25
1	(MAGGI)	(TEA)	0.25	0.35	0.2	0.8	2.285714	0.1125	3.25	0.75

Comparitive study of Appriori and FP Grouth

In [12]:
```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()
te_ary = te.fit(data).transform(data)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

In [13]:
```
from mlxtend.frequent_patterns import apriori

%timeit -n 100 -r 10 apriori(df, min_support=0.6)
```

```
3.9 ms ± 599 µs per loop (mean ± std. dev. of 10 runs, 100 loops each)
```

In [14]:
```
%timeit -n 100 -r 10 apriori(df, min_support=0.6, low_memory=True)
```

```
3.97 ms ± 1.01 ms per loop (mean ± std. dev. of 10 runs, 100 loops each)
```

In [15]:
```
from mlxtend.frequent_patterns import fpgrowth

%timeit -n 100 -r 10 fpgrowth(df, min_support=0.6)
```

```
1.59 ms ± 113 µs per loop (mean ± std. dev. of 10 runs, 100 loops each)
```

FP Grouth Algoritm is faster than Appriori