HW 4

1) Unique Items: Milk, Beer, Diaper, Bread, Butter, Cookies.

Unique Items = 6.

(a) Possible association rules = $3^d - 2^{d+1} + 1$

$= 3^6 - 2^{6+1} + 1$

$= 729 - 128 + 1$

$= \underline{602}$

(b) Maximum size of frequent items that can be extracted.

∵ The no of unique items is 6, it must be less than 6.

& Checking the given transaction data, TID = 6, 9 has itemsets of 4 length. Hence, max size of freq items = $\boxed{4}$.

(c) For 3 itemsets, same as choose 3 from 6 without repitition. i.e ${}^6P_3$ or $\frac{6!}{3!(6-3)!} = \frac{6!}{3!3!} =$

$= \frac{6 \times 5 \times 4}{3 \times 2} = \boxed{20}$.

(d) - Milk, Beer - [1]
Milk, Diaper - [4]
Milk, Bread [3]
Milk, Butter [3]
Milk, Cookies. [1]
- Beer, Diaper [2]
Beer, Bread [0]
Beer, Butter [0]
Beer, Cookies [2]
- Diaper, Bread [3]
Diaper, Butter [3]
Diaper, Cookies [2]
- Bread, Butter [5]
- Bread, Cookies [1]
- Butter, Cookies. [1]

→ Milk, Beer, Diaper [1].
→ ~~Milk, Diaper, Cookies [1].~~
→ Milk, Diaper, Bread [2].
→ Milk, Diaper, Butter [2].
→ Milk, Diaper, Cookies [1].
↑ This was not needed too.

→ Bread, Butter, Milk. [3]
→ Bread, Butter, Beer [0]
→ Bread, Butter, Diaper [3]
→ Bread, Butter, Cookies [1].
↑ [supersets will have support ≤ subset]

∴ The only 4 length itemset has support of 2, considering only {Bread, Butter} & its supersets because goal is to find itemsets with maximum support. ∴ No subsets of {Bread, Butter} with more than 2 items have support ≥ 5, {Butter, Bread} has the highest support with ≥ 2 items.

(e) $c(\{a\} \rightarrow \{b\}) = \frac{s(a \cup b)}{s(a)}$ ⟺ $c(\{b\} \rightarrow \{a\}) = \frac{s(a \cup b)}{s(b)}$.

i.e we want $s(a) = s(b)$ ✗

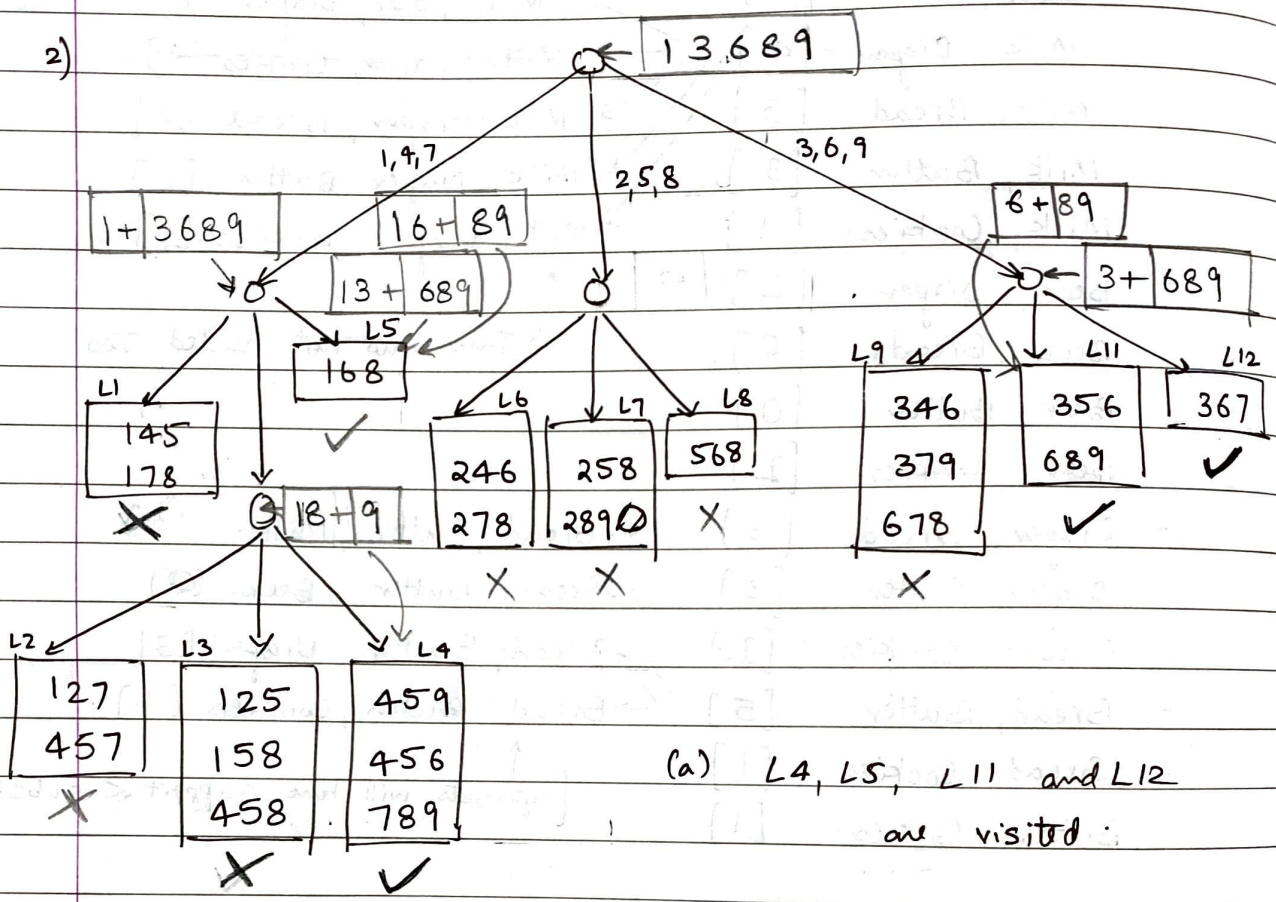
s(Milk) = 5
s(Beer) = 4
s(Diaper) = 7
s(Bread) = 5
s(Butter) = 5
s(Cookies) = 4.

{Milk} → {Bread}, {Bread} → {Milk} will have same confidence,
c_1 c_2

$c_1 = \frac{3}{5} = c_2 = \frac{3}{5}$.

$\therefore c_1 = c_2$

2)



(a) L4, L5, L11 and L12 are visited.

(b) L4: { }
L5: {168}
L11: {689}
L12: { }

Hence, {168} and {689} are supported by transaction from leaf nodes L5 & L11 respectively.

3)

	Coffee	$\overline{\text{Coffee}}$	
Tea	150 $_{11}$	50 $_{12}$	200
$\overline{\text{Tea}}$	650 $_{21}$	150 $_{22}$	800
	800 $_{01}$	200 $_{02}$	1000

10, 20

Expected contingency table :—

E_{11}	E_{12}
E_{21}	E_{22}

$$E_{ij}^{\circ\circ} = \frac{C_{i0} \cdot C_{0j}}{\text{Total}}$$

160 $_{11}$	40 $_{12}$
640 $_{21}$	160 $_{22}$

$$E_{11} = \frac{C_{10} \cdot C_{01}}{\text{Total}} = \frac{200 \times 800}{1000} = 160$$

$$E_{12} = \frac{C_{10} \cdot C_{02}}{\text{Total}} = \frac{200 \times 200}{1000} = 40$$

$$E_{21} = \frac{C_{20} \cdot C_{01}}{\text{Total}} = \frac{800 \times 800}{\text{Total}} = 640$$

$$E_{22} = \frac{C_{20} \cdot C_{02}}{\text{Total}} = \frac{800 \times 200}{1000} = 160.$$

$$\chi^2 = \frac{\Sigma (O_{ij} - E_{ij})^2}{E_{ij}}$$

$$= \frac{(C_{11} - E_{11})^2}{E_{11}} + \frac{(C_{12} - E_{12})^2}{E_{12}} + \frac{(C_{21} - E_{21})^2}{E_{21}} + \frac{(C_{22} - E_{22})^2}{E_{22}}$$

$$= \frac{(150 - 160)^2}{160} + \frac{(50 - 40)^2}{40} + \frac{(650 - 640)^2}{640} + \frac{(150 - 160)^2}{160}$$

$$\chi^2 = \frac{100}{160} + \frac{100}{40} + \frac{100}{640} + \frac{100}{160}$$

$$= \frac{10}{16} + \frac{10}{4} + \frac{10}{64} + \frac{10}{16}$$

$$= 0.625 + 2.5 + 0.156 + 0.625$$

$$= \underline{3.9}.$$

4) Consider the data :- for schools A & B. given the ensem results in percentage.

	Male.		Female.		
School.	num	Avg	num	Avg.	Total Avg
A	80	84	20	80	83.2
B	20	85	80	81	81.8.

Here, Average of Average is not the AVERAGE.

(A) Here, we can observe from the Total Average that school A's students perform better than School B's (83.2 > 81.8).

(B) But we havent considered a factor or ~~gender~~ variable. i.e gender while sampling the data. But if we check the interval distribution of the total students, we can see that for school A, both the averages of male students & female students are less than the average for male & female students from school B. (84 < 85) AND (80 < 81).

This level of detail or considering one more variable led to reversal of our conclusion. So here gener is an lurking variable. that affects the data but we did not considered it while checking only the average scores (Total).

(C) Also, this is happening because the internal distribution for the students (male or female) is different for both schools. i.e 80 male & 20 female students score are sampled ~~while~~ for school A while 20 male & 80 female students are ~~scor~~ sampled for school B.

That's why I think Simpsons paradox occur.

(D) To avoid Simpsons paradox, we should consider sampling equally as well as randomly. & make sure that there arent any underlying lurking variables that influence certain statistical results.

(D) One more example, related to previous. [To elabonate more on lurking variables].

consider school A:

male = 80 , avg = 84

female = 20 , avg = 80.

lets assume these students are in ~~courses.~~ / streams.

	Engineering.	Math	Political Science	
M:	50	20	10	⇐ Male
F:	5	0	15	⇐ Female

- So, why does it makes sense that male students average is less than female in school A?

To answer this question we might consider several underlying factors.

~~Case 1: male students tend to take harder subjects since they~~

Case 1: for school A, its hard or competetive to get into math & engineering programmes. & comparatively easier to get into pol-sci.
So, in this case more smart students must get into engg or math, hence they will score more than the students in pol-sci. ~~Thats why~~ And since there are more students in engg + math > pol-sci this increases the average.

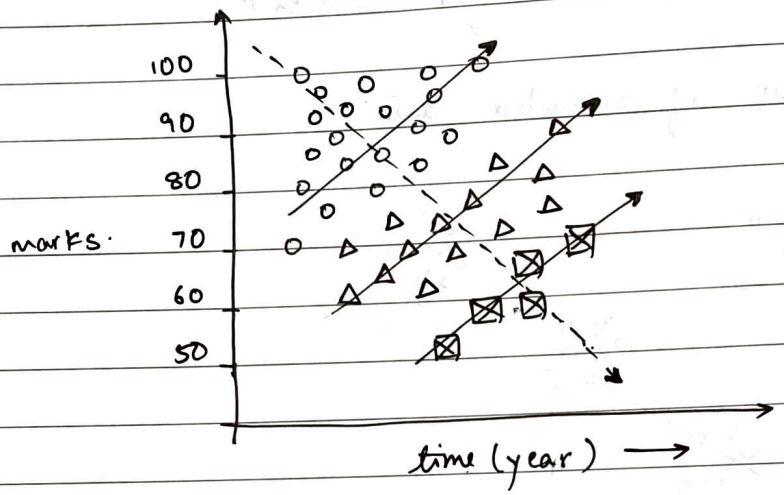
So here, the lurking variable was that school A has more competetive engg & math program & since male students tend to go in such programmes.
avg(male) > avg(female).

(E) There could be many more underlying factors responsible. Hence, we must first decide the level of detail to which we want our analysis to be done. ~~beco~~ & decide or study relation of underlying variables on the analyzed measure..

(F) Otherwise. if any lurking variable is ignored, it might reverse the meaning and finally led to false conclusions.

(G) One more related example.

consider scores of 3 subjects. engg (○), math (△) and pol-sci (⊠) over years.



when we do not consider the subjects, we see that as time passes by, more students have lower grades than previous year i.e performance is decreasing. (negative correlation).

But when we consider the individual groups, in each of them as time passes by, the scores are increasing. (positive correlation).
Hence when adding / considering one more factor, we see that average performance is increasing over each subject so the performance is increasing. This completely reverses the final conclusion

A Quick overview of what I have done

I have used dictionaries to store my results. I have written some functions and then clubbed them all together in another functions. Method 1 represents to (Fk-1 -- F1) and Method 2 is (Fk-1 -- Fk-1). In the few last cells I have created a table which has comparison between methods computed on fewer data. before that I have ran the algorithm on 25% of the total data to just check out what are associations I am getting when running on more data. I have ran the algorithm multiple times tinkering the data size, support and confidence but was not able to save all the results, so I will have only textual results in very small number of stated observations.

(A) Imports, Read Data and process it for better handling

In [1]:
```
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm
```

In [2]:
```
df = pd.read_csv('data/groceries.csv')
```

In [3]:
```
df.iloc[0][:10]
```

Out[3]:
```
Item(s)                       4
Item 1              citrus fruit
Item 2     semi-finished bread
Item 3                margarine
Item 4             ready soups
Item 5                     NaN
Item 6                     NaN
Item 7                     NaN
Item 8                     NaN
Item 9                     NaN
Name: 0, dtype: object
```

Create new Dataframe, just changing the format a bit

In [4]:
```
items = dict(n_items=[], items=[])
for i in range(df.shape[0]):
    row = df.iloc[i]
    arr = []
    n_items = row[0]
    for j in range(1,n_items+1):
        arr.append(row[j].lower())
    items['n_items'].append(row[0])
    items['items'].append(arr)
```

In [5]:
```
# items
```

In [6]:
```
data = pd.DataFrame(items)
data = data[:500]
data
```

Out[6]:

	n_items	items
0	4	[citrus fruit, semi-finished bread, margarine,...
1	3	[tropical fruit, yogurt, coffee]
2	1	[whole milk]
3	4	[pip fruit, yogurt, cream cheese, meat spreads]
4	4	[other vegetables, whole milk, condensed milk,...
...	...	...
495	2	[bottled beer, nut snack]
496	2	[pet care, bottled beer]
497	3	[citrus fruit, other vegetables, soda]
498	4	[pork, other vegetables, frozen vegetables, hy...
499	2	[other vegetables, shopping bags]

500 rows × 2 columns

In [8]:
```
# Just checking if the n_items match with length of data in items for each rows
for i in range(data.shape[0]):
    if data.iloc[i]['n_items'] != len(data.iloc[i]['items']):
        print(i)
```

There are functions below to

1. calculate the support given the dataframe and dictionary,
2. generate itemset pairs given 2 dictionraries, f1 and fk-1 in this case
3. prune the itemset pairs given the dictionary with itempairs and required support, it returns the pruned dictionary
4. To print some of the starting and ending elements of dictionary to get a quick glance over the dictionary

In [9]:
```
def calculate_support(data, dic):
    # iterate over items in dictionary(item-pairs)
    for itemset in tqdm(list(dic.keys())):
        count=0
        items_list = itemset.split('||')
        # iterate over each transaction if the item-pairs are present in it
        for row in range(data.shape[0]):
            transaction = data.iloc[row]['items']
            # increment the counter itself in the dictionary if all of the item
```

```
            check = all(item in transaction for item in items_list)
            if check:
                # print(items_list)
                # print(data.iloc[row]['n_items'], transaction)
                # print()
                count+=1
        dic[itemset] = count
```

In [10]:
```
def generate_itemsets(dic_s, dic_l):
    itemsets = dict()
    for item in tqdm(dic_s.keys()):
        for i in dic_l.keys():
            i_arr = i.split('||')
            if item < i_arr[0]:
                itemsets[item+'||'+i]=0
    return itemsets
```

In [11]:
```
def prune_itemset(dic, support):
    to_keep = []
    for key in dic.keys():
        if dic[key]>=support:
            to_keep.append(key)
    dic_pruned = {key: dic[key] for key in dic if key in to_keep}
    return dic_pruned
```

In [12]:
```
def dic_glance(dic):
    keys = list(dic.keys())
    if len(keys)>6:
        print('___size: {}___\n{} : {}\n{} : {}\n{} : {}\n...\n{} : {}\n{} : {}
              keys[1], dic[keys[1]], keys[2], dic[keys[2]], keys[-3], dic[key
    else:
        print('___size: {}___\n{}\n'.format(len(keys),dic))

# I happened to find out that this was not required, I can simply use len(dic)
def dic_len(dic):
    return len(list(dic.keys()))
```

F(k-1) - F1 method:

How this works is, first I calculate the unique items(f1) then calculate the support for it, prune f1, then calculate itemsets with size=2(f2) from the pruned f1. Then I prune the f2 and run a loop to calculate further itemsets with size=[3,k), where I follow [mix&match -> support count -> prune] method and keep on appending the frequent itemsets from f2, f3, ...fk into a dictionary(total_frequent_itemsets). Note: since I already have itempairs of length=1 in f1_pruned, I will not append this to the total_frequent_itemsets dictionary.

In [102…
```
def method1(support):
    print('Input Shape: ',data.shape)
    total_frequent_itemsets = dict()

    print('Initialize f1')
    # find out unique items
```

```
    items = []
    for i in range(data.shape[0]):
        transaction = data.iloc[i]['items']
        for item in transaction:
            if item not in items:
                items.append(item)
    items.sort()
    items[:5]+['........Many More']
    # unique_items = list(map(str.lower,items))
    unique_items = items.copy()
    unique_items.sort()
    len(unique_items)

    # save unique items as dictionary where key is item and value is its count
    f1 = dict()
    for item in unique_items:
        f1[item]=0
    calculate_support(data, f1)
    dic_glance(f1)
    print('Pruning f1')
    f1_pruned = prune_itemset(f1, support)
    dic_glance(f1_pruned)
    #total_frequent_itemsets.update(f1_pruned)

    print('Initialize f2')
    f2 = dict()
    items = list(f1_pruned.keys())
    for i in range(len(items)):
        primary_items = items[i]
        for item in items:
            if primary_items != item:
                if primary_items < item:
                    f2[primary_items+'||'+item]=0
    calculate_support(data, f2)
    dic_glance(f2)
    print('Pruning f2')
    f2_pruned = prune_itemset(f2, support)
    dic_glance(f2_pruned)
    total_frequent_itemsets.update(f2_pruned)
    print('Total Frequent Itemsets: ',len(total_frequent_itemsets))
    print('_'*5)

    fk_pruned = f2_pruned.copy()
    size=3
    len_frequent_pairs = 1

    while len_frequent_pairs!=0:
        print('initialize f'+str(size))
        fk = generate_itemsets(f1_pruned, fk_pruned)
        calculate_support(data, fk)
        dic_glance(fk)
        fk_pruned = prune_itemset(fk, support)
        print('Pruning f'+str(size))
        dic_glance(fk_pruned)
        len_frequent_pairs = dic_len(fk_pruned)
        total_frequent_itemsets.update(fk_pruned)
        print('Total Frequent Itemsets: ',len(total_frequent_itemsets))
        print('_'*5)
        size+=1
    return [f1,total_frequent_itemsets]
```

In [18]:
```
%%time
total_frequent_itemsets = method1(support = 5)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 82___
baking powder : 5
beef : 26
berries : 23
...
white wine : 5
whole milk : 122
white wine : 56

Initialize f2
  0%|          | 0/3321 [00:00<?, ?it/s]
___size: 3321___
baking powder||beef : 0
baking powder||berries : 0
baking powder||beverages : 0
...
white wine||whole milk : 0
white wine||yogurt : 0
white wine||whole milk : 21

Pruning f2
___size: 215___
beef||other vegetables : 6
beef||rolls/buns : 10
beef||root vegetables : 9
...
whipped/sour cream||whole milk : 14
whipped/sour cream||yogurt : 6
whipped/sour cream||whole milk : 21

Total Frequent Itemsets:  215

_____
initialize f3
  0%|          | 0/82 [00:00<?, ?it/s]
  0%|          | 0/6905 [00:00<?, ?it/s]
```

```
___size: 6905___
baking powder||beef||other vegetables : 0
baking powder||beef||rolls/buns : 0
baking powder||beef||root vegetables : 0
...
whipped/sour cream||whole milk||yogurt : 4
white bread||whole milk||yogurt : 1
whipped/sour cream||whole milk||yogurt : 0

Pruning f3
___size: 51___
bottled beer||other vegetables||whole milk : 5
bottled water||other vegetables||rolls/buns : 6
bottled water||other vegetables||whole milk : 6
...
root vegetables||whole milk||yogurt : 5
sausage||whole milk||yogurt : 5
root vegetables||whole milk||yogurt : 6

Total Frequent Itemsets:  266

_____
initialize f4
  0%|          | 0/82 [00:00<?, ?it/s]
  0%|          | 0/2032 [00:00<?, ?it/s]
___size: 2032___
baking powder||bottled beer||other vegetables||whole milk : 0
baking powder||bottled water||other vegetables||rolls/buns : 0
baking powder||bottled water||other vegetables||whole milk : 0
...
specialty chocolate||tropical fruit||whole milk||yogurt : 0
spices||tropical fruit||whole milk||yogurt : 0
specialty chocolate||tropical fruit||whole milk||yogurt : 2

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  266

_____
CPU times: user 7min 6s, sys: 3.03 s, total: 7min 9s
Wall time: 7min 9s
```

From the above cells, I ran the Fk-1 and Fk1 method and it gives us frequent pairs of length 266(not including f1_pruned), based on the support=5. Time taken is 7 minutes for 500 rows/transactions of data

---

F(k-1) - F(k-1) method:

The process is again the same as above, I just swap out the function which generates the itempairs

In [14]:
```
def generate_itemsets2(dic):
    itemsets = dict()
    keys = list(dic.keys())
```

```
    for i in tqdm(range(len(keys))):
        prior = keys[i]
        prior_arr = prior.split('||')
        for j in range(i+1, len(keys)):
            later = keys[j]
            later_arr = later.split('||')
            # print(prior_arr,' + ', later_arr)
            # print(prior_arr[:-1], later_arr[:-1])
            if prior_arr[:-1]==later_arr[:-1]:
                # print(prior_arr, later_arr[-1])
                itemset = '||'.join(prior_arr+[later_arr[-1]])
                itemsets[itemset]=0
    return itemsets
```

In [15]:

```
def method2(support):
    print('Input Shape: ',data.shape)
    total_frequent_itemsets = dict()

    print('Initialize f1')
    # find out unique items
    items = []
    for i in range(data.shape[0]):
        transaction = data.iloc[i]['items']
        for item in transaction:
            if item not in items:
                items.append(item)
    items.sort()
    items[:5]+['........Many More']
    # unique_items = list(map(str.lower,items))
    unique_items = items.copy()
    unique_items.sort()
    len(unique_items)

    # save unique items as dictionary where key is item and value is its count
    f1 = dict()
    for item in unique_items:
        f1[item]=0
    calculate_support(data, f1)
    dic_glance(f1)
    print('Pruning f1')
    f1_pruned = prune_itemset(f1, support)
    dic_glance(f1_pruned)
    #total_frequent_itemsets.update(f1_pruned)

    print('Initialize f2')
    f2 = dict()
    items = list(f1_pruned.keys())
    for i in range(len(items)):
        primary_items = items[i]
        for item in items:
            if primary_items != item:
                if primary_items < item:
                    f2[primary_items+'||'+item]=0
    calculate_support(data, f2)
    dic_glance(f2)
    print('Pruning f2')
    f2_pruned = prune_itemset(f2, support)
    dic_glance(f2_pruned)
    total_frequent_itemsets.update(f2_pruned)
```

```
    print('Total Frequent Itemsets: ',len(total_frequent_itemsets))
    print('_'*5)

    fk_pruned = f2_pruned.copy()
    size=3
    len_frequent_pairs = 1

    while len_frequent_pairs!=0:
        print('initialize f'+str(size))
        fk = generate_itemsets2(fk_pruned)
        calculate_support(data, fk)
        dic_glance(fk)
        fk_pruned = prune_itemset(fk, support)
        print('Pruning f'+str(size))
        dic_glance(fk_pruned)
        len_frequent_pairs = dic_len(fk_pruned)
        total_frequent_itemsets.update(fk_pruned)
        print('Total Frequent Itemsets: ',len(total_frequent_itemsets))
        print('_'*5)
        size+=1
    return [f1,total_frequent_itemsets]
```

In [63]:
```
%%time

total_frequent_itemsets = method2(support=5)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 82___
baking powder : 5
beef : 26
berries : 23
...
white wine : 5
whole milk : 122
white wine : 56

Initialize f2
  0%|          | 0/3321 [00:00<?, ?it/s]
```

```
___size: 3321___
baking powder||beef : 0
baking powder||berries : 0
baking powder||beverages : 0
...
white wine||whole milk : 0
white wine||yogurt : 0
white wine||whole milk : 21

Pruning f2
___size: 215___
beef||other vegetables : 6
beef||rolls/buns : 10
beef||root vegetables : 9
...
whipped/sour cream||whole milk : 14
whipped/sour cream||yogurt : 6
whipped/sour cream||whole milk : 21

Total Frequent Itemsets:  215

_____
initialize f3
  0%|          | 0/215 [00:00<?, ?it/s]
  0%|          | 0/674 [00:00<?, ?it/s]
___size: 674___
beef||other vegetables||rolls/buns : 4
beef||other vegetables||root vegetables : 2
beef||other vegetables||sausage : 1
...
tropical fruit||whole milk||yogurt : 6
waffles||whipped/sour cream||whole milk : 4
tropical fruit||whole milk||yogurt : 4

Pruning f3
___size: 51___
bottled beer||other vegetables||whole milk : 5
bottled water||other vegetables||rolls/buns : 6
bottled water||other vegetables||whole milk : 6
...
root vegetables||whole milk||yogurt : 5
sausage||whole milk||yogurt : 5
root vegetables||whole milk||yogurt : 6

Total Frequent Itemsets:  266

_____
initialize f4
  0%|          | 0/51 [00:00<?, ?it/s]
  0%|          | 0/16 [00:00<?, ?it/s]
```

```
___size: 16___
bottled water||other vegetables||rolls/buns||whole milk : 3
bottled water||rolls/buns||whole milk||yogurt : 3
other vegetables||rolls/buns||root vegetables||soda : 0
...
rolls/buns||sausage||soda||yogurt : 1
rolls/buns||sausage||whole milk||yogurt : 3
rolls/buns||sausage||soda||yogurt : 2

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  266

_____
CPU times: user 2min 23s, sys: 998 ms, total: 2min 24s
Wall time: 2min 23s
```

As we can see from the above cell, Fk-1 Fk-1 method is way more faster than Fk-1 F1 method(I am not 100% percent sure because to make such a claim I need to compare the runtimes much more times)

- *Fk-1 Fk-1 takes around 2.5 minutes while Fk-1 F1 takes 7 minutes to give the same results when ran on first 500 rows with support=5*
- *So the later is (2.5-7)/7 = 64 percent faster than the prior in this case*

In [296…

```
# import json
# with open('total_frequqnt_itemsets.json', 'w') as fp:
#     json.dump(total_frequent_itemsets, fp)

# with open('f1.json', 'w') as f:
#     json.dump(f1, f)
```

Functions to generate association pairs and Function to count confidence: confidence(X ⇒ Y) = support(X ∪ Y) / support(X)

In [85]:

```
import itertools
# to remove some elements from an array
def remove_elements(arr, to_remove):
    final = arr.copy()
    for elem in to_remove:
        if elem in arr:
            final.remove(elem)
    return sorted(final)

# returns nested array containing association rules like: [['beef->other vegeta
```

```
def recur_subset( s, l=None ):
    rules = []
    if l == None:
        l = len(s)-1
    if l > 0:
        for x in itertools.combinations( s, l ):
            #print(list(x),'->',remove_elements(s,x))
            #print('||'.join(list(x))+'->'+'||'.join(remove_elements(s,x)))
            rules.append('||'.join(list(x))+'->'+'||'.join(remove_elements(s,x)
            # check.append(list(x))
            # check.append(remove_elements(s,x))
        recur_subset( s, l-1 )
    return rules

def calculate_confidence(dic, all_frequent_itemsets):
    for rule in tqdm(dic.keys()):
        a = rule.split('->')[0]
        a_arr = a.split('||')
        b = rule.split('->')[1]
        b_arr = b.split('||')
        union_arr=a_arr.copy()
        union_arr.extend(b_arr)
        union_arr.sort()
        union='||'.join(union_arr)
        #print(union)
        #confidence=all_frequent_itemsets[union]/all_frequent_itemsets[b]
        try:
            confidence=all_frequent_itemsets[union]/all_frequent_itemsets[b]
            dic[rule]=confidence
            #print(confidence)
        except:
            pass
            #print(union,'+', b)
        #check.append('||'.join(union))
```

Function to return association rules(actually it returns a dictionary with confidence counts for all generated rules) after that previously developed function "prune_itemsets" can be used to get association rules with desired confidence

1. *Initialize rules dict i.e generate rules*
2. *Calculate confidence for each rule*
3. *Remove rules with confidence less than min-confidence*

In [81]:
```
def generate_rules(f1, total_frequent_itemsets, confidence):
    rules = {}
    # initialize the rules dictionary with key=rule and value=confidence
    print('Initialize Rules')
    for itemset in tqdm(total_frequent_itemsets.keys()):
        itemset_arr = itemset.split('||')
        # print(itemset_arr)
        # recur_subset(itemset_arr)
        # print()
        r = recur_subset(itemset_arr)
        for i in r:
            rules[i]=0
```

```
    #rules
    dic_glance(rules)
    print('_'*5)

    print('Calculate Confidence')
    all_frequent_itemsets = f1.copy()
    all_frequent_itemsets.update(total_frequent_itemsets)
    #print(all_frequent_itemsets)
    calculate_confidence(rules, all_frequent_itemsets)
    dic_glance(rules)
    print('_'*5)

    print('Pruning rules')
    rules_pruned = prune_itemset(rules, confidence)
    rules_pruned
    dic_glance(rules_pruned)
    return rules
```

(B)Running on more data, first 2500 rows from dataset with min(support=10, confidence=5)

In [348…
```
data = pd.DataFrame(items)
data = data[:2500]
data
```

Out[348]:

	n_items	items
0	4	[citrus fruit, semi-finished bread, margarine,...
1	3	[tropical fruit, yogurt, coffee]
2	1	[whole milk]
3	4	[pip fruit, yogurt, cream cheese, meat spreads]
4	4	[other vegetables, whole milk, condensed milk,...
...	...	...
2495	1	[sausage]
2496	1	[other vegetables]
2497	1	[herbs]
2498	6	[ham, pip fruit, other vegetables, cream chees...
2499	4	[root vegetables, yogurt, beverages, flour]

2500 rows × 2 columns

In [375…
```
def trial(f1, total_frequent_itemsets):
    return f1, total_frequent_itemsets

f1, total_frequqnt_itemsets=trial(f1, total_frequent_itemsets)
```

In [377…
```
total_frequent_itemsets
```

Out[377]:
```
{'bottled water||other vegetables': 19,
 'bottled water||rolls/buns': 19,
 'bottled water||whole milk': 15,
 'citrus fruit||whole milk': 15,
 'frankfurter||rolls/buns': 16,
 'other vegetables||rolls/buns': 25,
 'other vegetables||root vegetables': 20,
 'other vegetables||soda': 15,
 'other vegetables||tropical fruit': 17,
 'other vegetables||whole milk': 35,
 'rolls/buns||sausage': 23,
 'rolls/buns||soda': 22,
 'rolls/buns||whole milk': 32,
 'rolls/buns||yogurt': 16,
 'root vegetables||whole milk': 22,
 'sausage||whole milk': 15,
 'soda||whole milk': 19,
 'tropical fruit||whole milk': 18,
 'whole milk||yogurt': 21}
```

In [349…
```
%%time

f1, total_frequent_itemsets = method2(support=10)
rules = generate_rules(f1, total_frequent_itemsets, confidence=5)
```

```
Input Shape:  (2500, 2)
Initialize f1
  0%|          | 0/166 [00:00<?, ?it/s]
___size: 166___
abrasive cleaner : 11
artif. sweetener : 10
baby cosmetics : 3
...
whole milk : 642
yogurt : 357
whole milk : 18

Pruning f1
___size: 131___
abrasive cleaner : 11
artif. sweetener : 10
baking powder : 47
...
whole milk : 642
yogurt : 357
whole milk : 18

Initialize f2
  0%|          | 0/8515 [00:00<?, ?it/s]
```

```
___size: 8515___
abrasive cleaner||artif. sweetener : 1
abrasive cleaner||baking powder : 0
abrasive cleaner||beef : 4
...
whole milk||yogurt : 145
whole milk||zwieback : 4
whole milk||yogurt : 2

Pruning f2
___size: 889___
baking powder||domestic eggs : 11
baking powder||other vegetables : 14
baking powder||rolls/buns : 10
...
white bread||whole milk : 42
white bread||yogurt : 25
white bread||whole milk : 145

Total Frequent Itemsets:  889

_____
initialize f3
  0%|          | 0/889 [00:00<?, ?it/s]
  0%|          | 0/8549 [00:00<?, ?it/s]
___size: 8549___
baking powder||domestic eggs||other vegetables : 4
baking powder||domestic eggs||rolls/buns : 2
baking powder||domestic eggs||sugar : 5
...
whipped/sour cream||white bread||yogurt : 10
whipped/sour cream||whole milk||yogurt : 28
whipped/sour cream||white bread||yogurt : 13

Pruning f3
___size: 642___
baking powder||other vegetables||whole milk : 11
baking powder||whipped/sour cream||whole milk : 11
beef||brown bread||whole milk : 10
...
whipped/sour cream||white bread||yogurt : 10
whipped/sour cream||whole milk||yogurt : 28
whipped/sour cream||white bread||yogurt : 13

Total Frequent Itemsets:  1531

_____
initialize f4
  0%|          | 0/642 [00:00<?, ?it/s]
  0%|          | 0/849 [00:00<?, ?it/s]
```

```
___size: 849___
beef||citrus fruit||other vegetables||rolls/buns : 7
beef||citrus fruit||other vegetables||root vegetables : 6
beef||citrus fruit||other vegetables||whole milk : 8
...
soda||tropical fruit||whole milk||yogurt : 6
tropical fruit||whipped/sour cream||whole milk||yogurt : 11
soda||tropical fruit||whole milk||yogurt : 4

Pruning f4
___size: 79___
beef||other vegetables||rolls/buns||whole milk : 12
bottled water||rolls/buns||soda||yogurt : 11
bottled water||rolls/buns||whole milk||yogurt : 10
...
root vegetables||whipped/sour cream||whole milk||yogurt : 11
sausage||tropical fruit||whole milk||yogurt : 10
root vegetables||whipped/sour cream||whole milk||yogurt : 11

Total Frequent Itemsets:  1610

_____
initialize f5
  0%|          | 0/79 [00:00<?, ?it/s]
  0%|          | 0/9 [00:00<?, ?it/s]
___size: 9___
citrus fruit||other vegetables||root vegetables||tropical fruit||whole milk :
7
curd||other vegetables||root vegetables||whole milk||yogurt : 7
fruit/vegetable juice||other vegetables||pip fruit||whole milk||yogurt : 7
...
other vegetables||pip fruit||whipped/sour cream||whole milk||yogurt : 8
other vegetables||root vegetables||tropical fruit||whole milk||yogurt : 8
other vegetables||pip fruit||whipped/sour cream||whole milk||yogurt : 7

Pruning f5
___size: 1___
{'fruit/vegetable juice||other vegetables||root vegetables||whole milk||yogur
t': 10}

Total Frequent Itemsets:  1611

_____
initialize f6
  0%|          | 0/1 [00:00<?, ?it/s]
0it [00:00, ?it/s]
___size: 0___
{}

Pruning f6
___size: 0___
{}

Total Frequent Itemsets:  1611

_____
Initialize Rules
  0%|          | 0/1611 [00:00<?, ?it/s]
```

```
___size: 4025___
baking powder->domestic eggs : 0
domestic eggs->baking powder : 0
baking powder->other vegetables : 0
...
fruit/vegetable juice||other vegetables||whole milk||yogurt->root vegetables :
0
fruit/vegetable juice||root vegetables||whole milk||yogurt->other vegetables :
0
fruit/vegetable juice||other vegetables||whole milk||yogurt->root vegetables :
0


_____
Calculate Confidence
  0%|          | 0/4025 [00:00<?, ?it/s]
___size: 4025___
baking powder->domestic eggs : 0
domestic eggs->baking powder : 0
baking powder->other vegetables : 0
...
fruit/vegetable juice||other vegetables||whole milk||yogurt->root vegetables :
0
fruit/vegetable juice||root vegetables||whole milk||yogurt->other vegetables :
0
fruit/vegetable juice||other vegetables||whole milk||yogurt->root vegetables :
0


_____
Pruning rules
___size: 4___
{'curd->dessert': 7.0, 'rolls/buns->sliced cheese': 8.0, 'sausage->sliced chee
se': 5.0, 'rolls/buns||sausage->sliced cheese': 5.0}

CPU times: user 34min 26s, sys: 11.6 s, total: 34min 37s
Wall time: 34min 35s
```

In [347…

```
rules = dict(sorted(rules.items(), key=lambda item: item[1]))
association_rules = prune_itemset(rules, 2)
association_rules
```

Out[347]:

```
{'other vegetables->root vegetables': 2.2222222222222223,
 'sausage->shopping bags': 2.3333333333333335,
 'other vegetables->whipped/sour cream': 2.4,
 'rolls/buns->whipped/sour cream': 2.4,
 'bottled water->hygiene articles': 2.5,
 'bottled water->packaged fruit/vegetables': 2.5,
 'fruit/vegetable juice->pip fruit': 2.5,
 'rolls/buns->sausage': 2.875,
 'bottled water->napkins': 3.0,
 'other vegetables->pip fruit': 3.0,
 'other vegetables->pork': 3.0,
 'rolls/buns->shopping bags': 3.0,
 'other vegetables->shopping bags': 3.6666666666666665,
 'hygiene articles->napkins': 4.0,
 'sausage->sliced cheese': 5.0,
 'curd->dessert': 7.0,
 'rolls/buns->sliced cheese': 8.0}
```

As we can see from above, association rules logically makes sense hence the algorithm is workign properly. Example if a shopper buys rolls/buns theres high

probablity that he/she will buy sliced cheese OR we can see that probablity of buying shopping bags after buying any item is high, this also seems logically valid

---

Trying out different values of support and confidence(using only Fk-1 -- Fk-1 since it is faster, to measure how they perform and compare the results. (considering only first 500 rows to reduce computational time)

In [18]:
```
data = pd.DataFrame(items)
data = data[:500]
data
```

Out[18]:

	n_items	items
0	4	[citrus fruit, semi-finished bread, margarine,...
1	3	[tropical fruit, yogurt, coffee]
2	1	[whole milk]
3	4	[pip fruit, yogurt, cream cheese, meat spreads]
4	4	[other vegetables, whole milk, condensed milk,...
...	...	...
495	2	[bottled beer, nut snack]
496	2	[pet care, bottled beer]
497	3	[citrus fruit, other vegetables, soda]
498	4	[pork, other vegetables, frozen vegetables, hy...
499	2	[other vegetables, shopping bags]

500 rows × 2 columns

In [106…
```
%%time
# support=3   Fk-1 -- F-1

arr = method1(support=5)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.5)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
```

```
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 82___
baking powder : 5
beef : 26
berries : 23
...
white wine : 5
whole milk : 122
white wine : 56

Initialize f2
  0%|          | 0/3321 [00:00<?, ?it/s]
___size: 3321___
baking powder||beef : 0
baking powder||berries : 0
baking powder||beverages : 0
...
white wine||whole milk : 0
white wine||yogurt : 0
white wine||whole milk : 21

Pruning f2
___size: 215___
beef||other vegetables : 6
beef||rolls/buns : 10
beef||root vegetables : 9
...
whipped/sour cream||whole milk : 14
whipped/sour cream||yogurt : 6
whipped/sour cream||whole milk : 21

Total Frequent Itemsets:  215

_____
initialize f3
  0%|          | 0/82 [00:00<?, ?it/s]
  0%|          | 0/6905 [00:00<?, ?it/s]
```

```
___size: 6905___
baking powder||beef||other vegetables : 0
baking powder||beef||rolls/buns : 0
baking powder||beef||root vegetables : 0
...
whipped/sour cream||whole milk||yogurt : 4
white bread||whole milk||yogurt : 1
whipped/sour cream||whole milk||yogurt : 0

Pruning f3
___size: 51___
bottled beer||other vegetables||whole milk : 5
bottled water||other vegetables||rolls/buns : 6
bottled water||other vegetables||whole milk : 6
...
root vegetables||whole milk||yogurt : 5
sausage||whole milk||yogurt : 5
root vegetables||whole milk||yogurt : 6

Total Frequent Itemsets:  266

_____
initialize f4
  0%|          | 0/82 [00:00<?, ?it/s]
  0%|          | 0/2032 [00:00<?, ?it/s]
___size: 2032___
baking powder||bottled beer||other vegetables||whole milk : 0
baking powder||bottled water||other vegetables||rolls/buns : 0
baking powder||bottled water||other vegetables||whole milk : 0
...
specialty chocolate||tropical fruit||whole milk||yogurt : 0
spices||tropical fruit||whole milk||yogurt : 0
specialty chocolate||tropical fruit||whole milk||yogurt : 2

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  266

_____
Initialize Rules
  0%|          | 0/266 [00:00<?, ?it/s]
___size: 583___
beef->other vegetables : 0
other vegetables->beef : 0
beef->rolls/buns : 0
...
tropical fruit||whole milk->yogurt : 0
tropical fruit||yogurt->whole milk : 0
tropical fruit||whole milk->yogurt : 0


_____
Calculate Confidence
  0%|          | 0/583 [00:00<?, ?it/s]
```

```
___size: 583___
beef->other vegetables : 0.0625
other vegetables->beef : 0.23076923076923078
beef->rolls/buns : 0.09433962264150944
...
tropical fruit||whole milk->yogurt : 0.10714285714285714
tropical fruit||yogurt->whole milk : 0.04918032786885246
tropical fruit||whole milk->yogurt : 0.13333333333333333


_____
Pruning rules
___size: 14___
whole milk->cereals : 0.8333333333333334
rolls/buns->chewing gum : 0.5555555555555556
whole milk->chicken : 0.5555555555555556
...
rolls/buns->sliced cheese : 0.7272727272727273
whole milk->sugar : 0.5
rolls/buns->sliced cheese : 0.5333333333333333

CPU times: user 7min 7s, sys: 2.69 s, total: 7min 10s
Wall time: 7min 9s
```

In [107…

```
%%time
# support=5   Fk-1 -- F-1

arr = method1(support=7)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.5)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 68___
beef : 26
berries : 23
beverages : 17
...
white bread : 12
whole milk : 122
white bread : 56

Initialize f2
  0%|          | 0/2278 [00:00<?, ?it/s]
```

```
___size: 2278___
beef||berries : 4
beef||beverages : 2
beef||bottled beer : 1
...
white bread||whole milk : 3
white bread||yogurt : 1
white bread||whole milk : 21

Pruning f2
___size: 107___
beef||rolls/buns : 10
beef||root vegetables : 9
beef||whole milk : 8
...
waffles||whole milk : 8
whipped/sour cream||whole milk : 14
waffles||whole milk : 21

Total Frequent Itemsets:  107

_____
initialize f3
  0%|          | 0/68 [00:00<?, ?it/s]
  0%|          | 0/3306 [00:00<?, ?it/s]
___size: 3306___
beef||berries||root vegetables : 3
beef||berries||whole milk : 2
beef||beverages||whole milk : 1
...
waffles||whole milk||yogurt : 3
whipped/sour cream||whole milk||yogurt : 4
waffles||whole milk||yogurt : 1

Pruning f3
___size: 8___
citrus fruit||rolls/buns||whole milk : 7
curd||whole milk||yogurt : 7
margarine||rolls/buns||whole milk : 8
...
other vegetables||soda||whole milk : 7
rolls/buns||sausage||whole milk : 7
other vegetables||soda||whole milk : 7

Total Frequent Itemsets:  115

_____
initialize f4
  0%|          | 0/68 [00:00<?, ?it/s]
  0%|          | 0/292 [00:00<?, ?it/s]
```

```
___size: 292___
beef||citrus fruit||rolls/buns||whole milk : 1
beef||curd||whole milk||yogurt : 0
beef||margarine||rolls/buns||whole milk : 3
...
pork||rolls/buns||whole milk||yogurt : 0
red/blush wine||rolls/buns||sausage||whole milk : 0
pork||rolls/buns||whole milk||yogurt : 0

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  115

_____
Initialize Rules
  0%|          | 0/115 [00:00<?, ?it/s]
___size: 238___
beef->rolls/buns : 0
rolls/buns->beef : 0
beef->root vegetables : 0
...
rolls/buns||whole milk->yogurt : 0
rolls/buns||yogurt->whole milk : 0
rolls/buns||whole milk->yogurt : 0


_____
Calculate Confidence
  0%|          | 0/238 [00:00<?, ?it/s]
___size: 238___
beef->rolls/buns : 0.09433962264150944
rolls/buns->beef : 0.38461538461538464
beef->root vegetables : 0.18
...
rolls/buns||whole milk->yogurt : 0.125
rolls/buns||yogurt->whole milk : 0.05737704918032787
rolls/buns||whole milk->yogurt : 0.0660377358490566


_____
Pruning rules
___size: 8___
root vegetables->detergent : 0.5384615384615384
whole milk->domestic eggs : 0.5555555555555556
whole milk->ham : 0.6666666666666666
...
rolls/buns->sliced cheese : 0.7272727272727273
whole milk->sugar : 0.5
rolls/buns->sliced cheese : 0.5333333333333333

CPU times: user 3min 27s, sys: 1.02 s, total: 3min 28s
Wall time: 3min 27s
```

In [109…

```
%%time
# support=7   Fk-1 -- F-1

arr = method1(support=10)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.75)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 52___
beef : 26
berries : 23
beverages : 17
...
white bread : 12
whole milk : 122
white bread : 56

Initialize f2
  0%|          | 0/1326 [00:00<?, ?it/s]
___size: 1326___
beef||berries : 4
beef||beverages : 2
beef||bottled beer : 1
...
white bread||whole milk : 3
white bread||yogurt : 1
white bread||whole milk : 21

Pruning f2
___size: 49___
beef||rolls/buns : 10
bottled beer||whole milk : 10
bottled water||other vegetables : 19
...
tropical fruit||whole milk : 18
whipped/sour cream||whole milk : 14
tropical fruit||whole milk : 21

Total Frequent Itemsets:  49

_____
initialize f3
  0%|          | 0/52 [00:00<?, ?it/s]
  0%|          | 0/1308 [00:00<?, ?it/s]
```

```
___size: 1308___
beef||bottled beer||whole milk : 1
beef||bottled water||other vegetables : 0
beef||bottled water||rolls/buns : 0
...
waffles||whole milk||yogurt : 3
whipped/sour cream||whole milk||yogurt : 4
waffles||whole milk||yogurt : 1

Pruning f3
___size: 1___
{'other vegetables||rolls/buns||whole milk': 12}

Total Frequent Itemsets:  50

_____
initialize f4
  0%|          | 0/52 [00:00<?, ?it/s]
  0%|          | 0/32 [00:00<?, ?it/s]
___size: 32___
beef||other vegetables||rolls/buns||whole milk : 3
berries||other vegetables||rolls/buns||whole milk : 0
beverages||other vegetables||rolls/buns||whole milk : 1
...
newspapers||other vegetables||rolls/buns||whole milk : 2
oil||other vegetables||rolls/buns||whole milk : 1
newspapers||other vegetables||rolls/buns||whole milk : 1

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  50

_____
Initialize Rules
  0%|          | 0/50 [00:00<?, ?it/s]
___size: 101___
beef->rolls/buns : 0
rolls/buns->beef : 0
bottled beer->whole milk : 0
...
other vegetables||rolls/buns->whole milk : 0
other vegetables||whole milk->rolls/buns : 0
other vegetables||rolls/buns->whole milk : 0


_____
Calculate Confidence
  0%|          | 0/101 [00:00<?, ?it/s]
```

```
___size: 101___
beef->rolls/buns : 0.09433962264150944
rolls/buns->beef : 0.38461538461538464
bottled beer->whole milk : 0.08196721311475409
...
other vegetables||rolls/buns->whole milk : 0.09836065573770492
other vegetables||whole milk->rolls/buns : 0.11320754716981132
other vegetables||rolls/buns->whole milk : 0.125


_____
Pruning rules
___size: 0___
{}

CPU times: user 1min 37s, sys: 691 ms, total: 1min 38s
Wall time: 1min 38s
```

In [108…

```
%%time
# support=10   Fk-1 -- F-1

arr = method1(support=15)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.25)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 42___
beef : 26
berries : 23
beverages : 17
...
whipped/sour cream : 29
whole milk : 122
whipped/sour cream : 56

Initialize f2
  0%|          | 0/861 [00:00<?, ?it/s]
```

```
___size: 861___
beef||berries : 4
beef||beverages : 2
beef||bottled beer : 1
...
whipped/sour cream||whole milk : 14
whipped/sour cream||yogurt : 6
whipped/sour cream||whole milk : 21

Pruning f2
___size: 19___
bottled water||other vegetables : 19
bottled water||rolls/buns : 19
bottled water||whole milk : 15
...
soda||whole milk : 19
tropical fruit||whole milk : 18
soda||whole milk : 21

Total Frequent Itemsets:  19

_____
initialize f3
  0%|          | 0/42 [00:00<?, ?it/s]
  0%|          | 0/452 [00:00<?, ?it/s]
___size: 452___
beef||bottled water||other vegetables : 0
beef||bottled water||rolls/buns : 0
beef||bottled water||whole milk : 1
...
tropical fruit||whole milk||yogurt : 6
waffles||whole milk||yogurt : 3
tropical fruit||whole milk||yogurt : 4

Pruning f3
___size: 0___
{}

Total Frequent Itemsets:  19

_____
Initialize Rules
  0%|          | 0/19 [00:00<?, ?it/s]
___size: 38___
bottled water->other vegetables : 0
other vegetables->bottled water : 0
bottled water->rolls/buns : 0
...
whole milk->tropical fruit : 0
whole milk->yogurt : 0
whole milk->tropical fruit : 0


_____
Calculate Confidence
  0%|          | 0/38 [00:00<?, ?it/s]
```

```
___size: 38___
bottled water->other vegetables : 0.19791666666666666
other vegetables->bottled water : 0.2835820895522388
bottled water->rolls/buns : 0.1792452830188679
...
whole milk->tropical fruit : 0.4
whole milk->yogurt : 0.375
whole milk->tropical fruit : 0.1721311475409836


_____
Pruning rules
___size: 18___
other vegetables->bottled water : 0.2835820895522388
rolls/buns->bottled water : 0.2835820895522388
whole milk->citrus fruit : 0.32608695652173914
...
whole milk->sausage : 0.3191489361702128
whole milk->tropical fruit : 0.4
whole milk->sausage : 0.375

CPU times: user 50.5 s, sys: 262 ms, total: 50.8 s
Wall time: 50.7 s
```

In [91]:
```
%%time
# support=3   Fk-1 -- Fk-1

arr = method2(support=5)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.5)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 82___
baking powder : 5
beef : 26
berries : 23
...
white wine : 5
whole milk : 122
white wine : 56

Initialize f2
  0%|          | 0/3321 [00:00<?, ?it/s]
```

```
___size: 3321___
baking powder||beef : 0
baking powder||berries : 0
baking powder||beverages : 0
...
white wine||whole milk : 0
white wine||yogurt : 0
white wine||whole milk : 21

Pruning f2
___size: 215___
beef||other vegetables : 6
beef||rolls/buns : 10
beef||root vegetables : 9
...
whipped/sour cream||whole milk : 14
whipped/sour cream||yogurt : 6
whipped/sour cream||whole milk : 21

Total Frequent Itemsets:  215

_____
initialize f3
  0%|          | 0/215 [00:00<?, ?it/s]
  0%|          | 0/674 [00:00<?, ?it/s]
___size: 674___
beef||other vegetables||rolls/buns : 4
beef||other vegetables||root vegetables : 2
beef||other vegetables||sausage : 1
...
tropical fruit||whole milk||yogurt : 6
waffles||whipped/sour cream||whole milk : 4
tropical fruit||whole milk||yogurt : 4

Pruning f3
___size: 51___
bottled beer||other vegetables||whole milk : 5
bottled water||other vegetables||rolls/buns : 6
bottled water||other vegetables||whole milk : 6
...
root vegetables||whole milk||yogurt : 5
sausage||whole milk||yogurt : 5
root vegetables||whole milk||yogurt : 6

Total Frequent Itemsets:  266

_____
initialize f4
  0%|          | 0/51 [00:00<?, ?it/s]
  0%|          | 0/16 [00:00<?, ?it/s]
```

```
___size: 16___
bottled water||other vegetables||rolls/buns||whole milk : 3
bottled water||rolls/buns||whole milk||yogurt : 3
other vegetables||rolls/buns||root vegetables||soda : 0
...
rolls/buns||sausage||soda||yogurt : 1
rolls/buns||sausage||whole milk||yogurt : 3
rolls/buns||sausage||soda||yogurt : 2

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  266

_____
Initialize Rules
  0%|          | 0/266 [00:00<?, ?it/s]
___size: 583___
beef->other vegetables : 0
other vegetables->beef : 0
beef->rolls/buns : 0
...
tropical fruit||whole milk->yogurt : 0
tropical fruit||yogurt->whole milk : 0
tropical fruit||whole milk->yogurt : 0


_____
Calculate Confidence
  0%|          | 0/583 [00:00<?, ?it/s]
___size: 583___
beef->other vegetables : 0.0625
other vegetables->beef : 0.23076923076923078
beef->rolls/buns : 0.09433962264150944
...
tropical fruit||whole milk->yogurt : 0.10714285714285714
tropical fruit||yogurt->whole milk : 0.04918032786885246
tropical fruit||whole milk->yogurt : 0.13333333333333333


_____
Pruning rules
___size: 14___
whole milk->cereals : 0.8333333333333334
rolls/buns->chewing gum : 0.5555555555555556
whole milk->chicken : 0.5555555555555556
...
rolls/buns->sliced cheese : 0.7272727272727273
whole milk->sugar : 0.5
rolls/buns->sliced cheese : 0.5333333333333333

CPU times: user 1min 28s, sys: 518 ms, total: 1min 29s
Wall time: 1min 29s
```

In [110…

```
%%time
# support=7   Fk-1 -- Fk-1

arr = method2(support=7)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.5)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 68___
beef : 26
berries : 23
beverages : 17
...
white bread : 12
whole milk : 122
white bread : 56

Initialize f2
  0%|          | 0/2278 [00:00<?, ?it/s]
___size: 2278___
beef||berries : 4
beef||beverages : 2
beef||bottled beer : 1
...
white bread||whole milk : 3
white bread||yogurt : 1
white bread||whole milk : 21

Pruning f2
___size: 107___
beef||rolls/buns : 10
beef||root vegetables : 9
beef||whole milk : 8
...
waffles||whole milk : 8
whipped/sour cream||whole milk : 14
waffles||whole milk : 21

Total Frequent Itemsets:  107
_____
initialize f3
  0%|          | 0/107 [00:00<?, ?it/s]
  0%|          | 0/221 [00:00<?, ?it/s]
```

```
___size: 221___
beef||rolls/buns||root vegetables : 3
beef||rolls/buns||whole milk : 4
beef||root vegetables||whole milk : 3
...
soda||tropical fruit||yogurt : 4
soda||whole milk||yogurt : 4
soda||tropical fruit||yogurt : 6

Pruning f3
___size: 8___
citrus fruit||rolls/buns||whole milk : 7
curd||whole milk||yogurt : 7
margarine||rolls/buns||whole milk : 8
...
other vegetables||soda||whole milk : 7
rolls/buns||sausage||whole milk : 7
other vegetables||soda||whole milk : 7

Total Frequent Itemsets:  115

_____
initialize f4
  0%|          | 0/8 [00:00<?, ?it/s]
0it [00:00, ?it/s]
___size: 0___
{}

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  115

_____
Initialize Rules
  0%|          | 0/115 [00:00<?, ?it/s]
___size: 238___
beef->rolls/buns : 0
rolls/buns->beef : 0
beef->root vegetables : 0
...
rolls/buns||whole milk->yogurt : 0
rolls/buns||yogurt->whole milk : 0
rolls/buns||whole milk->yogurt : 0


_____
Calculate Confidence
  0%|          | 0/238 [00:00<?, ?it/s]
```

```
___size: 238___
beef->rolls/buns : 0.09433962264150944
rolls/buns->beef : 0.38461538461538464
beef->root vegetables : 0.18
...
rolls/buns||whole milk->yogurt : 0.125
rolls/buns||yogurt->whole milk : 0.05737704918032787
rolls/buns||whole milk->yogurt : 0.0660377358490566


_____
Pruning rules
___size: 8___
root vegetables->detergent : 0.5384615384615384
whole milk->domestic eggs : 0.5555555555555556
whole milk->ham : 0.6666666666666666
...
rolls/buns->sliced cheese : 0.7272727272727273
whole milk->sugar : 0.5
rolls/buns->sliced cheese : 0.5333333333333333

CPU times: user 1min 31s, sys: 579 ms, total: 1min 32s
Wall time: 1min 32s
```

In [99]:

```
%%time
# support=5   Fk-1 -- Fk-1

arr = method2(support=7)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.5)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 68___
beef : 26
berries : 23
beverages : 17
...
white bread : 12
whole milk : 122
white bread : 56

Initialize f2
  0%|          | 0/2278 [00:00<?, ?it/s]
```

```
___size: 2278___
beef||berries : 4
beef||beverages : 2
beef||bottled beer : 1
...
white bread||whole milk : 3
white bread||yogurt : 1
white bread||whole milk : 21

Pruning f2
___size: 107___
beef||rolls/buns : 10
beef||root vegetables : 9
beef||whole milk : 8
...
waffles||whole milk : 8
whipped/sour cream||whole milk : 14
waffles||whole milk : 21

Total Frequent Itemsets:  107

_____
initialize f3
  0%|          | 0/107 [00:00<?, ?it/s]
  0%|          | 0/221 [00:00<?, ?it/s]
___size: 221___
beef||rolls/buns||root vegetables : 3
beef||rolls/buns||whole milk : 4
beef||root vegetables||whole milk : 3
...
soda||tropical fruit||yogurt : 4
soda||whole milk||yogurt : 4
soda||tropical fruit||yogurt : 6

Pruning f3
___size: 8___
citrus fruit||rolls/buns||whole milk : 7
curd||whole milk||yogurt : 7
margarine||rolls/buns||whole milk : 8
...
other vegetables||soda||whole milk : 7
rolls/buns||sausage||whole milk : 7
other vegetables||soda||whole milk : 7

Total Frequent Itemsets:  115

_____
initialize f4
  0%|          | 0/8 [00:00<?, ?it/s]
0it [00:00, ?it/s]
___size: 0___
{}

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  115

_____
Initialize Rules
  0%|          | 0/115 [00:00<?, ?it/s]
```

```
___size: 238___
beef->rolls/buns : 0
rolls/buns->beef : 0
beef->root vegetables : 0
...
rolls/buns||whole milk->yogurt : 0
rolls/buns||yogurt->whole milk : 0
rolls/buns||whole milk->yogurt : 0


_____
Calculate Confidence
  0%|          | 0/238 [00:00<?, ?it/s]
___size: 238___
beef->rolls/buns : 0.09433962264150944
rolls/buns->beef : 0.38461538461538464
beef->root vegetables : 0.18
...
rolls/buns||whole milk->yogurt : 0.125
rolls/buns||yogurt->whole milk : 0.05737704918032787
rolls/buns||whole milk->yogurt : 0.0660377358490566


_____
Pruning rules
___size: 8___
root vegetables->detergent : 0.5384615384615384
whole milk->domestic eggs : 0.5555555555555556
whole milk->ham : 0.6666666666666666
...
rolls/buns->sliced cheese : 0.7272727272727273
whole milk->sugar : 0.5
rolls/buns->sliced cheese : 0.5333333333333333

CPU times: user 1min 31s, sys: 587 ms, total: 1min 32s
Wall time: 1min 32s
```

In [100…

```
%%time
#support=10   Fk-1 -- Fk-1

arr = method2(support=10)
f1, total_frequent_itemsets = arr[0], arr[1]
rules = generate_rules(f1, total_frequent_itemsets, confidence=0.75)
```

```
Input Shape:  (500, 2)
Initialize f1
  0%|          | 0/147 [00:00<?, ?it/s]
```

```
___size: 147___
abrasive cleaner : 4
artif. sweetener : 1
baby cosmetics : 1
...
whole milk : 122
yogurt : 56
whole milk : 4

Pruning f1
___size: 52___
beef : 26
berries : 23
beverages : 17
...
white bread : 12
whole milk : 122
white bread : 56

Initialize f2
  0%|          | 0/1326 [00:00<?, ?it/s]
___size: 1326___
beef||berries : 4
beef||beverages : 2
beef||bottled beer : 1
...
white bread||whole milk : 3
white bread||yogurt : 1
white bread||whole milk : 21

Pruning f2
___size: 49___
beef||rolls/buns : 10
bottled beer||whole milk : 10
bottled water||other vegetables : 19
...
tropical fruit||whole milk : 18
whipped/sour cream||whole milk : 14
tropical fruit||whole milk : 21

Total Frequent Itemsets:  49

_____
initialize f3
  0%|          | 0/49 [00:00<?, ?it/s]
  0%|          | 0/73 [00:00<?, ?it/s]
```

```
___size: 73___
bottled water||other vegetables||rolls/buns : 6
bottled water||other vegetables||soda : 4
bottled water||other vegetables||whole milk : 6
...
sausage||soda||yogurt : 4
sausage||whole milk||yogurt : 5
sausage||soda||yogurt : 4

Pruning f3
___size: 1___
{'other vegetables||rolls/buns||whole milk': 12}

Total Frequent Itemsets:  50

_____
initialize f4
  0%|          | 0/1 [00:00<?, ?it/s]
0it [00:00, ?it/s]
___size: 0___
{}

Pruning f4
___size: 0___
{}

Total Frequent Itemsets:  50

_____
Initialize Rules
  0%|          | 0/50 [00:00<?, ?it/s]
___size: 101___
beef->rolls/buns : 0
rolls/buns->beef : 0
bottled beer->whole milk : 0
...
other vegetables||rolls/buns->whole milk : 0
other vegetables||whole milk->rolls/buns : 0
other vegetables||rolls/buns->whole milk : 0


_____
Calculate Confidence
  0%|          | 0/101 [00:00<?, ?it/s]
___size: 101___
beef->rolls/buns : 0.09433962264150944
rolls/buns->beef : 0.38461538461538464
bottled beer->whole milk : 0.08196721311475409
...
other vegetables||rolls/buns->whole milk : 0.09836065573770492
other vegetables||whole milk->rolls/buns : 0.11320754716981132
other vegetables||rolls/buns->whole milk : 0.125


_____
Pruning rules
___size: 0___
{}

CPU times: user 53.7 s, sys: 381 ms, total: 54 s
Wall time: 53.9 s
```

The following table has times which are computed by considering first 500 transactions(to save time) and then changing the values for support and confidence

Support	F(k-1) - F1 (method 1)	F(k-1) - F(k-1) (method 2)
5	7.2 min	2.3 min
7	3.5 min	1.5 min
10	1.7 min	1.5 min
15	0.9 min	0.8 min

---

1. *According to the above table, we can see than method2 is faster than method1*
2. *Also, we can see that Fk-1 -- F1-1 method saves time in frequent timeset generation because it does fewer comparisons or iterations(this can be verified from the print statements)*
3. *The no of itemsets generated in both of the methods remains th same, thats why final association rules will also be the same. but method 2 is significantly faster than first*
4. *Also, I observed that as the support increases frequent itemsets decreases and speed increases.*
5. *As the support increases, size of itemsets in frequent itemsets and ultimately size of association rule decreases. So for more detailed analysis support and confidence shoud be lower, but again this increases the computing time.*
6. *Confidence can also be reduced to get rules with more no of items in it*
7. *And as the size of data increases, we can have higher support or higher confidence. Along with this, as size of data increases size of rules generated can also increase.*

In [ ]: