Problem Set 5

CED18I039 - Paleti Krishnasai

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Question 1

Extend the Apriori Algorithm discussed in the class supporting Transaction Reduction approach to improve the time complexity issue as a result of the repeated scans limitation of Apriori. You may compare this extended version with the earlier implementations in (1) over the same benchmark dataset.

In [2]:
```
import pandas as pd
import time
#Use Market Data
df=pd.read_csv("Market_Basket_Optimisation.csv",header=None)
df
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmo
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

In [3]:

```
#Store the dataframe into list of lists/transactions
records = [[y for y in x if pd.notna(y)] for x in df.values.tolist()]
print("Sample of a record:\n",records[0])
```

```
Sample of a record:
 ['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes', 'whole weat flour', 'yams', 'cottage che
ese', 'energy drink', 'tomato juice', 'low fat yogurt', 'green tea', 'honey', 'salad', 'mineral water', 'sal
mon', 'antioxydant juice', 'frozen smoothie', 'spinach', 'olive oil']
```

In [4]:

```
Database={}
for i in range(len(records)):
    Database["T"+str(i+1)]=records[i]

Itemset={}
for i in range(len(records)):
    for j in range(len(records[i])):
        if(frozenset([records[i][j]]) not in Itemset):
            Itemset[frozenset([records[i][j]])]=1
        else:
            Itemset[frozenset([records[i][j]])]+=1
```

In [5]:

```
def get_items(Itemset,no_of_items,cur):
    for key,val in Itemset.items():
        print(key," ",val)

Min_Sup_Count=0.005*len(records)

Li={}
for key,val in Itemset.items():
    if(val>=Min_Sup_Count):
        Li[key]=val
Itemset=Li

def check(miniset,Database):
    count=0
    #print(miniset)
    for key,val in Database.items():

        if(frozenset(val).intersection(miniset)==miniset):
            count+=1
    return count

def get_c(Li,Database,itert):
    c={}
    for key,vals in Li.items():
        for key1,vals1 in Li.items():
            if (key1!=key):
                miniset=key1.union(key)
                if(len(miniset)>sot+1):
                    continue
                count=check(miniset,Database)
                c[miniset]=count
    return c

def remove_transaction(Database,sot):
    #print(Database)
    rem_keys=[]
    for key,val in Database.items():
        #print(key,val)
        if(len(val)<=sot):
            rem_keys.append(key)
    for key in rem_keys:
        Database.pop(key)
    return Database
```

```python
def remove_items(c,Min_Sup_Count):
    #print(Database)
    #print("Li=",c)
    rem_keys=[]
    for key,val in c.items():
        if(val<Min_Sup_Count):
            #print(key,val)
            rem_keys.append(key)
    for key in rem_keys:
        c.pop(key)
    #print(c)
    return c


def remove_item(Li,Database):
    miniset=set()
    for key,val in Li.items():
        miniset=miniset.union(key)
    for key,val in Database.items():
        Database[key]=list(set(val) & miniset)

    return Database

countitem=0
for key,val in Itemset.items():
    print(key," ",val)
    countitem+=1
    if(countitem>10):
        break
```

```
frozenset({'shrimp'})   536
frozenset({'almonds'})   153
frozenset({'avocado'})   250
frozenset({'vegetables mix'})   193
frozenset({'green grapes'})   68
frozenset({'whole weat flour'})   70
frozenset({'yams'})   86
frozenset({'cottage cheese'})   239
frozenset({'energy drink'})   200
frozenset({'tomato juice'})   228
frozenset({'low fat yogurt'})   574
```

In [6]:

```
import time
start = time.process_time()


Final_List=[]
sot=1
final_c={}
while(1):
    print("Iteration No: ",sot)
    Database=remove_transaction(Database,sot)
    c=get_c(Li,Database,sot+1)
    #print(c)
    sot+=1

    Li=remove_items(c,Min_Sup_Count)
    Database=remove_item(Li,Database)
    if(len(Li)==0):
        break
    else:
        final_c=Li

time_taken=time.process_time() - start
print("\n Time Taken for Mining the itemset with minimum support of "+str(Min_Sup_Count)+" = "+str(time_take
```

```
Iteration No:  1
Iteration No:  2
Iteration No:  3

 Time Taken for Mining the itemset with minimum support of 37.505 = 103.84487366900001 seconds
```

In [7]:

```
#partial results
countitem=0
for key,val in final_c.items():
    print(key," ",val)
    countitem+=1
    if(countitem>10):
        break
```

```
frozenset({'eggs', 'mineral water', 'shrimp'})   39
frozenset({'milk', 'mineral water', 'shrimp'})   59
frozenset({'mineral water', 'shrimp', 'frozen vegetables'})   54
```

```
frozenset({'mineral water', 'shrimp', 'spaghetti'})   64
frozenset({'mineral water', 'chocolate', 'shrimp'})   57
frozenset({'mineral water', 'shrimp', 'ground beef'})   38
frozenset({'milk', 'chocolate', 'shrimp'})   41
frozenset({'shrimp', 'frozen vegetables', 'spaghetti'})   45
frozenset({'chocolate', 'shrimp', 'frozen vegetables'})   40
frozenset({'chocolate', 'shrimp', 'spaghetti'})   48
```

In [8]:

```
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder

import pandas as pd
te = TransactionEncoder()
te_ary = te.fit(records).transform(records)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Out[8]:

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	...	turkey	vegetables mix	water spray
0	False	True	True	False	True	False	False	False	False	False	...	False	True	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False
3	False	False	False	False	True	False	False	False	False	False	...	True	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7496	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7497	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7498	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7499	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7500	False	False	False	False	False	False	False	False	False	False	...	False	False	False

7501 rows × 120 columns

In [9]:

```
import time
start = time.process_time()

print(apriori(df, min_support=0.005,use_colnames=True))

time_taken=time.process_time() - start
print("\n Time Taken for mining using Apriori =",time_taken)
```

```
      support                                        itemsets
0    0.020397                                       (almonds)
1    0.008932                           (antioxydant juice)
2    0.033329                                     (avocado)
3    0.008666                                       (bacon)
4    0.010799                                (barbecue sauce)
..        ...                                             ...
720  0.007466             (mineral water, soup, spaghetti)
721  0.009332          (mineral water, spaghetti, tomatoes)
722  0.006399            (mineral water, turkey, spaghetti)
723  0.006266  (mineral water, whole wheat rice, spaghetti)
724  0.005066             (olive oil, spaghetti, pancakes)

[725 rows x 2 columns]

 Time Taken for mining using Apriori = 1.1855476629999941
```

Time Taken for Mining using Apriori = 0.921875 seconds Time Taken for Mining using Apriori(Transaction Reduction Approach) = 95.796875 seconds The time taken for TR Approach might be higher owing to the efficient implementation of the apriori module present in mlxtend

Question 2

Test drive any one implementation in (1) or (2) adopting a Vertical Transaction Database format.

In [10]:

```
records=[[100,400,500,700,800,900],[100,200,300,400,600,800,900],[300,500,600,700,800,900],[200,400],[100,80
```

In [11]:

```
Database={}
for i in range(len(records)):
    Database["T"+str(i+1)]=records[i]

Itemset={}
for i in range(len(records)):
    for j in range(len(records[i])):
        if(frozenset([records[i][j]]) not in Itemset):
            Itemset[frozenset([records[i][j]])]=1
        else:
            Itemset[frozenset([records[i][j]])]+=1

from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder

import pandas as pd
te = TransactionEncoder()
te_ary = te.fit(records).transform(records)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Out[11]:

	100	200	300	400	500	600	700	800	900
0	True	False	False	True	True	False	True	True	True
1	True	True	True	True	False	True	False	True	True
2	False	False	True	False	True	True	True	True	True
3	False	True	False	True	False	False	False	False	False
4	True	False	False	False	False	False	False	True	False

In [12]:

```
Database_vdf={}
for key,val in Database.items():
    for x in val:
        if(frozenset([x]) not in Database_vdf):
            Database_vdf[frozenset([x])]=frozenset([key])
        else:
            Database_vdf[frozenset([x])]=frozenset([key]).union(Database_vdf[frozenset([x])])

records_vdf=[]
for key,val in Database_vdf.items():
    records_vdf.append(val)
```

In [13]:

```
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder

import pandas as pd
te = TransactionEncoder()
te_ary = te.fit(records_vdf).transform(records_vdf)
df_vdf = pd.DataFrame(te_ary, columns=te.columns_)
df_vdf
```

Out[13]:

	T1	T2	T3	T4	T5
0	True	True	False	False	True
1	True	True	False	True	False
2	True	False	True	False	False
3	True	False	True	False	False
4	True	True	True	False	True
5	True	True	True	False	False
6	False	True	False	True	False
7	False	True	True	False	False
8	False	True	True	False	False

In [14]:

```
import time
start = time.process_time()

print(apriori(df, min_support=0.003,use_colnames=True))

time_taken=time.process_time() - start
print("\n Time Taken for Mining using Apriori =",time_taken)
```

```
     support                              itemsets
0        0.6                                 (100)
1        0.4                                 (200)
2        0.4                                 (300)
3        0.6                                 (400)
4        0.4                                 (500)
..       ...                                   ...
206      0.2       (800, 100, 900, 300, 400, 600)
207      0.2       (800, 100, 900, 400, 500, 700)
208      0.2       (800, 900, 200, 300, 400, 600)
209      0.2       (800, 900, 300, 500, 600, 700)
210      0.2  (800, 100, 900, 200, 300, 400, 600)

[211 rows x 2 columns]

 Time Taken for Mining using Apriori = 0.03592329199999256
```

In [15]:

```
import time
start = time.process_time()

print(apriori(df_vdf, min_support=0.003,use_colnames=True))

time_taken=time.process_time() - start
print("\n Time Taken for Mining using Apriori (Vertical Transaction Database format)  = "+str(time_taken)+"
```

```
     support          itemsets
0   0.666667              (T1)
1   0.777778              (T2)
2   0.666667              (T3)
3   0.222222              (T4)
4   0.222222              (T5)
5   0.444444          (T1, T2)
6   0.444444          (T1, T3)
7   0.111111          (T1, T4)
```

```
8   0.222222          (T1, T5)
9   0.444444          (T2, T3)
10  0.222222          (T4, T2)
11  0.222222          (T2, T5)
12  0.111111          (T5, T3)
13  0.222222      (T1, T2, T3)
14  0.111111      (T1, T2, T4)
15  0.222222      (T1, T2, T5)
16  0.111111      (T1, T5, T3)
17  0.111111      (T2, T5, T3)
18  0.111111  (T1, T2, T5, T3)
```

The time taken for mining using VTD format is similar to the time taken for normal format .

Question 3

Using a vertical transaction database notation, generate the FI's following the intersection approach (basic ECLAT) discussed in the class. Use earlier benchmark datasets in (1).

In [16]:

```
import pandas as pd
import time
#Use Market Data
df=pd.read_csv("Market_Basket_Optimisation.csv",header=None)
df
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmo
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

In [17]:

```
records = [[y for y in x if pd.notna(y)] for x in df.values.tolist()]

Database={}
for i in range(len(records)):
    Database["T"+str(i+1)]=records[i]

Database_vdf={}
for key,val in Database.items():
    for x in val:
        if(frozenset([x]) not in Database_vdf):
            Database_vdf[frozenset([x])]=frozenset([key])
        else:
            Database_vdf[frozenset([x])]=frozenset([key]).union(Database_vdf[frozenset([x])])


records_vdf=[]
for key,val in Database_vdf.items():
    records_vdf.append(val)
```

In [18]:

```
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder

import pandas as pd
te = TransactionEncoder()
te_ary = te.fit(records_vdf).transform(records_vdf)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Out[18]:

	T1	T10	T100	T1000	T1001	T1002	T1003	T1004	T1005	T1006	...	T990	T991	T992	T993	T994	T995	T996	T997	T99
0	True	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	True	Fals
1	True	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
2	True	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
3	True	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
4	True	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
115	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
116	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
117	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
118	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals
119	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	Fals

120 rows × 7501 columns

In [19]:

```
#convert to vertical
Database_vdf={}
for key,val in Database.items():
    for x in val:
        #print(Database_vdf)
        if(frozenset([x]) not in Database_vdf):
            #print(frozenset([x]))
            Database_vdf[frozenset([x])]=frozenset([key])
        else:
            #print(x)
            Database_vdf[frozenset([x])]=frozenset([key]).union(Database_vdf[frozenset([x])])

def remove_items_vdf(Database_vdf,Min_Sup):
    rem_keys=[]
    for key,val in Database_vdf.items():
        if(len(val)<Min_Sup):
            rem_keys.append(key)
    for key in rem_keys:
        Database_vdf.pop(key)
    return Database_vdf

def get_vdf(Li,iteration):
    New_Li={}
    for key1,val1 in Li.items():
        for key2,val2 in Li.items():
            if(key1!=key2):
                new_key=key1.union(key2)
                if(len(new_key)>iteration):
                    continue
                else:
                    New_Li[new_key]=val1.intersection(val2)
    return New_Li


import time
start = time.process_time()
Min_Sup_Count_vdf=0.005*len(records)

Li=remove_items_vdf(Database_vdf,Min_Sup_Count_vdf)
iteration=1
while(1):
    iteration+=1
    c=get_vdf(Li,iteration)
```

```
    print("Iteration No: ",iteration)
    Li=remove_items_vdf(c,Min_Sup_Count_vdf)
    if(len(Li)==0):
        break
    else:
        final_vdf=Li

time_taken=time.process_time() - start
print("\n Time Taken for Mining the itemset with min_support of "+str(Min_Sup_Count_vdf)+" = "+str(time_take
```

```
Iteration No:  2
Iteration No:  3
Iteration No:  4

 Time Taken for Mining the itemset with min_support of 37.505 = 0.2825049150000041 seconds
```

In [20]:

```
countitem=0
for key,val in final_vdf.items():
    print(key," ",val,"\n")
    countitem+=1
    if(countitem>10):
        break
```

```
frozenset({'eggs', 'mineral water', 'shrimp'})   frozenset({'T6973', 'T7468', 'T1726', 'T92', 'T1327', 'T369
6', 'T4993', 'T111', 'T6101', 'T667', 'T143', 'T657', 'T7475', 'T1817', 'T2262', 'T2008', 'T3869', 'T1226',
'T2179', 'T976', 'T478', 'T237', 'T4095', 'T1608', 'T1894', 'T3528', 'T1214', 'T4925', 'T3880', 'T1059', 'T6
776', 'T108', 'T126', 'T5062', 'T3616', 'T2357', 'T745', 'T7370', 'T144'})

frozenset({'milk', 'mineral water', 'shrimp'})   frozenset({'T5219', 'T7131', 'T3660', 'T5698', 'T2125', 'T8
09', 'T3041', 'T2242', 'T7265', 'T789', 'T4993', 'T5019', 'T5466', 'T2335', 'T6716', 'T667', 'T504', 'T2105
', 'T796', 'T7475', 'T3260', 'T3629', 'T3869', 'T5277', 'T2179', 'T2783', 'T5991', 'T3755', 'T2198', 'T621',
'T2156', 'T1605', 'T4121', 'T5792', 'T2796', 'T1059', 'T6157', 'T7344', 'T2430', 'T6022', 'T108', 'T126', 'T
5062', 'T1606', 'T4933', 'T3616', 'T2633', 'T3481', 'T2510', 'T2065', 'T5746', 'T3643', 'T801', 'T3242', 'T5
639', 'T2359', 'T745', 'T7370', 'T5016'})

frozenset({'mineral water', 'shrimp', 'frozen vegetables'})   frozenset({'T5219', 'T6973', 'T2173', 'T7131',
'T3660', 'T2668', 'T6523', 'T5607', 'T6101', 'T2335', 'T143', 'T3447', 'T2105', 'T5247', 'T5460', 'T3059', '
T3981', 'T2618', 'T3579', 'T1959', 'T1700', 'T1226', 'T2179', 'T2783', 'T5785', 'T2198', 'T472', 'T5084', 'T
1038', 'T1894', 'T1605', 'T3370', 'T4925', 'T3880', 'T2820', 'T1393', 'T6776', 'T6022', 'T984', 'T126', 'T50
62', 'T1606', 'T4933', 'T3616', 'T2633', 'T3481', 'T1993', 'T5746', 'T3643', 'T3242', 'T2357', 'T5639', 'T13
66', 'T700'})
```

frozenset({'mineral water', 'shrimp', 'spaghetti'}) frozenset({'T6973', 'T2173', 'T7468', 'T3660', 'T4932', 'T4830', 'T92', 'T809', 'T2668', 'T6544', 'T1327', 'T3696', 'T6523', 'T5019', 'T3723', 'T5607', 'T676', 'T4494', 'T504', 'T6716', 'T1389', 'T4096', 'T2105', 'T796', 'T1346', 'T3059', 'T3981', 'T2618', 'T3579', 'T5855', 'T4914', 'T2783', 'T5991', 'T462', 'T2198', 'T4710', 'T4095', 'T1894', 'T5479', 'T1657', 'T3528', 'T1214', 'T1605', 'T4925', 'T3693', 'T2820', 'T1059', 'T2430', 'T126', 'T1606', 'T1993', 'T2633', 'T2065', 'T2309', 'T3643', 'T6554', 'T801', 'T3242', 'T2357', 'T5639', 'T2359', 'T2638', 'T5016', 'T144'})

frozenset({'mineral water', 'chocolate', 'shrimp'}) frozenset({'T2173', 'T2670', 'T7468', 'T1726', 'T3660', 'T4830', 'T3696', 'T6523', 'T1785', 'T5019', 'T891', 'T3723', 'T676', 'T4494', 'T143', 'T4766', 'T7475', 'T3260', 'T5460', 'T3059', 'T2618', 'T3629', 'T3579', 'T5302', 'T1700', 'T1226', 'T5277', 'T2198', 'T472', 'T621', 'T2696', 'T1608', 'T1894', 'T5479', 'T2156', 'T3528', 'T1214', 'T2293', 'T1657', 'T5792', 'T2796', 'T1393', 'T2430', 'T6022', 'T126', 'T5062', 'T1606', 'T3616', 'T2633', 'T5746', 'T3424', 'T2357', 'T5639', 'T745', 'T1366', 'T660', 'T5016'})

frozenset({'mineral water', 'shrimp', 'ground beef'}) frozenset({'T6973', 'T2173', 'T4932', 'T2668', 'T828', 'T1327', 'T3696', 'T4993', 'T6523', 'T5607', 'T143', 'T4766', 'T2008', 'T3981', 'T4914', 'T5991', 'T462', 'T3544', 'T4133', 'T4245', 'T1894', 'T5084', 'T3693', 'T1605', 'T2820', 'T5792', 'T2796', 'T7344', 'T1606', 'T2633', 'T2510', 'T3424', 'T3643', 'T3242', 'T2359', 'T2638', 'T3124', 'T5395'})

frozenset({'milk', 'chocolate', 'shrimp'}) frozenset({'T634', 'T161', 'T3660', 'T1741', 'T937', 'T4289', 'T5019', 'T7475', 'T3260', 'T3195', 'T3629', 'T5277', 'T4326', 'T3378', 'T3686', 'T2198', 'T1835', 'T621', 'T566', 'T469', 'T2156', 'T1329', 'T5792', 'T2796', 'T6439', 'T7324', 'T2430', 'T6022', 'T126', 'T5062', 'T1606', 'T3616', 'T2925', 'T2633', 'T1607', 'T5746', 'T3029', 'T1375', 'T5639', 'T745', 'T5016'})

frozenset({'shrimp', 'frozen vegetables', 'spaghetti'}) frozenset({'T1717', 'T6973', 'T2173', 'T3660', 'T3269', 'T2668', 'T3509', 'T6523', 'T6280', 'T5607', 'T2105', 'T3085', 'T3864', 'T3059', 'T3981', 'T2618', 'T3560', 'T3579', 'T1784', 'T2783', 'T727', 'T2198', 'T1480', 'T3913', 'T7331', 'T1894', 'T1605', 'T4839', 'T4925', 'T2820', 'T2896', 'T6578', 'T2384', 'T1460', 'T126', 'T1606', 'T4074', 'T1993', 'T2633', 'T270', 'T3643', 'T480', 'T3242', 'T2357', 'T5639'})

frozenset({'chocolate', 'shrimp', 'frozen vegetables'}) frozenset({'T634', 'T2173', 'T2079', 'T3660', 'T3509', 'T6523', 'T1399', 'T143', 'T6986', 'T3412', 'T5460', 'T3059', 'T2618', 'T3579', 'T1700', 'T1226', 'T6176', 'T3378', 'T2198', 'T472', 'T469', 'T5400', 'T1894', 'T6578', 'T7324', 'T1393', 'T6022', 'T126', 'T5062', 'T1606', 'T3616', 'T2925', 'T2633', 'T5746', 'T3029', 'T2357', 'T4921', 'T5639', 'T1366', 'T1886'})

frozenset({'chocolate', 'shrimp', 'spaghetti'}) frozenset({'T6060', 'T2173', 'T916', 'T7468', 'T3660', 'T4998', 'T3448', 'T4830', 'T1741', 'T1941', 'T5697', 'T2734', 'T3696', 'T3509', 'T6523', 'T5019', 'T3723', 'T4494', 'T676', 'T3195', 'T3059', 'T2320', 'T2618', 'T3579', 'T1471', 'T7482', 'T2198', 'T3576', 'T772', 'T3006', 'T1894', 'T5479', 'T1657', 'T3528', 'T1214', 'T4988', 'T1702', 'T6578', 'T5487', 'T4936', 'T2430', 'T126', 'T1606', 'T2633', 'T2357', 'T1531', 'T5639', 'T5016'})

frozenset({'shrimp', 'spaghetti', 'ground beef'}) frozenset({'T6973', 'T2173', 'T4998', 'T4932', 'T2668', 'T6718', 'T5697', 'T3696', 'T1327', 'T6523', 'T4907', 'T5607', 'T1558', 'T3085', 'T3864', 'T3195', 'T6086', 'T3981', 'T2740', 'T3560', 'T4914', 'T2382', 'T5991', 'T3493', 'T462', 'T3576', 'T1894', 'T2811', 'T3693', 'T1605', 'T4988', 'T2820', 'T2384', 'T1880', 'T1606', 'T4074', 'T5385', 'T2633', 'T3643', 'T3400', 'T3242', '

```
T1531', 'T2359', 'T2638', 'T2734'})
```

Question 4

Extend the basic Apriori algorithm to generate Frequent Patterns which differentiate ab from ba (ordered patterns generation).

In [21]:
```
sequences={}
sequences[10]="<a(abc)(ac)d(cf)>"
sequences[20]="<(ad)c(abc)(ae)>"
sequences[30]="<(ef)(ab)(df)cb>"
sequences[40]="<eg(af)cbc>"
```

In [22]:
```
for key,val in sequences.items():
    sequences[key]=val.replace('<','').replace('(','').replace(')','').replace('>','')
sequences
```

Out[22]:
```
{10: 'aabcacdcf', 20: 'adcabcae', 30: 'efabdfcb', 40: 'egafcbc'}
```

In [23]:
```
c0={}
for key in sequences.keys():
    for x in sequences[key]:
        if(x!='(' and x!=')' and x!='<' and x!='>'):
            if(x not in c0):
                c0[x]=1
            else:
                c0[x]+=1
c0
```

Out[23]:
```
{'a': 8, 'b': 5, 'c': 8, 'd': 3, 'f': 4, 'e': 3, 'g': 1}
```

In [24]:
```
l0={}
Min_Sup_Count=1
for key in c0.keys():
    if(c0[key]>=Min_Sup_Count):
        l0[key]=c0[key]
l0
```

Out[24]: {'a': 8, 'b': 5, 'c': 8, 'd': 3, 'f': 4, 'e': 3, 'g': 1}

In [25]:
```
import re
def get_sequence(l0,sequences,count):
    c1={}
    for key1 in l0.keys():
        for key2 in l0.keys():
            if(key1!=key2):
                new_key=key1+key2
                if(len(new_key)==count):
                    for key in sequences.keys():
                        if(new_key not in c1):
                            c1[new_key]=len(re.findall(new_key,sequences[key]))
                        else:
                            c1[new_key]+=len(re.findall(new_key,sequences[key]))
    return c1


def remove_sequence(c,Min_Sup_Count):
    Li={}
    for key in c.keys():
        if(c[key]>=Min_Sup_Count):
            Li[key]=c[key]
    return Li
```

In [26]:
```
sot=1
final_sequence={}
Min_Sup_Count=3
while(1):
    print("Iteration No: ",sot)

    c=get_sequence(l0,sequences,sot+1)

    sot+=1
    Li=remove_sequence(c,Min_Sup_Count)
    if(len(Li)==0):
        break
    else:
        final_sequence=Li
```

```
Iteration No:  1
```

In [27]:
```
print("Frequent Patterns are:",[x for x in final_sequence.keys()] )
```

```
Frequent Patterns are: ['ab', 'bc', 'ca']
```

Question 5

Implement following extensions to Apriori Algorithm (discussed / to be discussed in the class): Hash based strategy, Partitioning Approach. You may refer to online tutorials for a formal pseudocode description.

Hash Variant

In [28]:
```
transactions = [{'A','C','D'},{'B','C','E'},{'A','B','C','E'},{'B','E'}]
```

In [29]:
```
database_hash={}
count=0
for i in transactions:
    count+=1
    database_hash["T"+str(count)]=i
```

In [30]:
```
c0={}
for i in transactions:
    for j in i:
        if(j in c0):
            c0[j]+=1
        else:
            c0[j]=1
c0
```

Out[30]:
```
{'D': 1, 'C': 3, 'A': 2, 'B': 3, 'E': 3}
```

In [31]:
```
order={}
count=0
for key in sorted(c0.keys()):
    count+=1
    order[key]=count
order
```

Out[31]:
```
{'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5}
```

In [32]:
```
Min_Sup_Count=2
rem_keys=[]
Li={}
for key,val in c0.items():
    if(val>=Min_Sup_Count):
        Li[key]=val
        rem_keys.append(key)

print(Li)
```

```
{'C': 3, 'A': 2, 'B': 3, 'E': 3}
```

In [33]:
```
#Generate Set of all 2 transactions in Database
import itertools
for key,val in database_hash.items():
    val=[set(i) for i in itertools.combinations(val, 2)]
    sets=[]
    for i in range(len(val)):
        sets.append(sorted(val[i]))

    database_hash[key]=sets
database_hash
```

Out[33]:
```
{'T1': [['C', 'D'], ['A', 'D'], ['A', 'C']],
 'T2': [['B', 'C'], ['C', 'E'], ['B', 'E']],
 'T3': [['B', 'C'],
  ['A', 'C'],
  ['C', 'E'],
  ['A', 'B'],
  ['B', 'E'],
```

```
 ['A', 'E']],
```

In [34]:

```
#Generate hash table
Hash_Table={}
for key,items in database_hash.items():
    for x in items:
        val=(order[x[0]]*10+order[x[1]])%7
        if(val in Hash_Table):
            Hash_Table[val].append(x)
        else:
            Hash_Table[val]=[x]
Hash_Table
```

Out[34]:

```
{6: [['C', 'D'], ['A', 'C'], ['A', 'C']],
 0: [['A', 'D'], ['C', 'E'], ['C', 'E']],
 2: [['B', 'C'], ['B', 'C']],
 4: [['B', 'E'], ['B', 'E'], ['B', 'E']],
 5: [['A', 'B']],
 1: [['A', 'E']]}
```

In [35]:

```
print(Li)
```

```
{'C': 3, 'A': 2, 'B': 3, 'E': 3}
```

In [36]:

```
#Generate C2
C2={}
keys=sorted(Li.keys())
for i in range(len(keys)):
    for j in range(i+1,len(keys)):
        New_key=frozenset(set(keys[i]).union(set(keys[j])))
        New_val=(order[keys[i]]*10+order[keys[j]])%7
        C2[New_key]=len(Hash_Table[New_val])
print(C2)
```

```
{frozenset({'B', 'A'}): 1, frozenset({'C', 'A'}): 3, frozenset({'E', 'A'}): 1, frozenset({'C', 'B'}): 2, fro
zenset({'B', 'E'}): 3, frozenset({'C', 'E'}): 3}
```

In [37]:

```
#Generate L2
L2={}
for key,val in C2.items():
    if(val>=Min_Sup_Count):
        L2[key]=val

print(L2)
```

```
{frozenset({'C', 'A'}): 3, frozenset({'C', 'B'}): 2, frozenset({'B', 'E'}): 3, frozenset({'C', 'E'}): 3}
```

Partitioning Approach

In [38]:

```
df=pd.read_csv("Market_Basket_Optimisation.csv",header=None)
df
```

Out[38]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmo
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

7501 rows × 20 columns

In [39]:
```
records = [[y for y in x if pd.notna(y)] for x in df.values.tolist()]

Database={}
for i in range(len(records)):
    Database["T"+str(i+1)]=records[i]
```

In [40]:
```
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder

import pandas as pd
te = TransactionEncoder()
te_ary = te.fit(records).transform(records)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Out[40]:

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	...	turkey	vegetables mix	water spray
0	False	True	True	False	True	False	False	False	False	False	...	False	True	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False
3	False	False	False	False	True	False	False	False	False	False	...	True	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7496	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7497	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7498	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7499	False	False	False	False	False	False	False	False	False	False	...	False	False	False
7500	False	False	False	False	False	False	False	False	False	False	...	False	False	False

7501 rows × 120 columns

In [41]:
```
def split_dfs(df,no):
    dfs=[]
    for i in range(0,df.shape[0],int(df.shape[0]/no)):
        dfs.append(df.iloc[i:i+int(df.shape[0]/no)])
    return dfs
dfs=split_dfs(df,13)
```

In [42]:
```
results=[]
for i in dfs:
    results.append(apriori(i, min_support=0.01,use_colnames=True))
```

In [43]:
```
import math
final_candidate_set={}
for i in results:
    for j in range(i.shape[0]):
        item=i.iloc[j][1]
        if(item in final_candidate_set):
            final_candidate_set[item]+=(i.iloc[j][0]*int(df.shape[0]/13))
        else:
            final_candidate_set[item]=(i.iloc[j][0]*int(df.shape[0]/13))
```

In [44]:
```
final_results={}
Min_Sup_Count=int(df.shape[0]*(0.1))
for key,val in final_candidate_set.items():
    if(val>=Min_Sup_Count):
        final_results[key]=val
final_results
```

Out[44]:
```
{frozenset({'chocolate'}): 1229.0,
 frozenset({'eggs'}): 1348.0,
 frozenset({'french fries'}): 1282.0,
 frozenset({'green tea'}): 991.0,
 frozenset({'milk'}): 972.0,
 frozenset({'mineral water'}): 1788.0,
 frozenset({'spaghetti'}): 1306.0}
```

Question 6

Implement the Dynamic Itemset Counting Algorithm for Frequent Itemset Generation.

In [45]:

```
database = [[1,1,0],[1,0,0],[0,1,1],[0,0,0]]
unique_itemset =[{1},{2},{3}]
min_supp = 1
M = 2
size = len(database)
```

In [46]:

```
import numpy as np
import itertools
import copy

def get_subset(S,n):
    a = itertools.combinations(S,n)
    results = []
    for i in a:
        results.append(set(i))
    return(results)

def get_superset(S,unique_itemset):
    #print(S)
    result = []
    a = set()
    for i in unique_itemset:
        if i.intersection(S)==set():
            a = i.union(S)
            result.append(a)
            a = set()

    return(result)
```

In [47]:

```
def check_subset(Set,frequent_set):
    subset = get_subset(Set,len(Set)-1)
    flag = 1
    temp = []

    for i in frequent_set:
        temp.append(i[0])

    frequent_set = temp
    for i in subset:
        if i not in frequent_set:
            flag=0
            break

    if flag:
        return(True)
    else:
        return(False)

def get_itemset(T):
    result = set()
    for i in range(len(T)):
        if T[i]!=0:
            result.add(i+1)

    return(result)
```

In [48]:

```
DC = []
DS = []
SC = []
SS = []

for i in unique_itemset:
    DC.append([i,0,0])

print("Initial DC:",DC,"\n")

counter = 0
T = []
while len(DC)!=0 or len(DS)!=0:

    for i in range(counter,counter+M):
        index = i%size
        T = get_itemset(database[index])
        print("Transaction :",T)

        for item in DC:
            item[2]+=1
            if item[0].issubset(T):
                item[1]+=1
        for item in DS:
            item[2]+=1
            if item[0].issubset(T):
                item[1]+=1

    for item in copy.copy(DC):
        if(item[1]>=min_supp):
            DS.append(item)
            DC.remove(item)

    for item in copy.copy(DS):
        if(item[2]==size):
            SS.append(item)
            DS.remove(item)
    for item in copy.copy(DC):
        if(item[2]==size):
            SC.append(item)
            DC.remove(item)

    frequent_set = copy.copy(DS)
```

```
    frequent_set.extend(SS)
    for item in frequent_set:
        S = get_superset(item[0],unique_itemset)
        for i in S:
            if (check_subset(i,frequent_set)):
                flag=1
                for x in DC:
                    if x[0]==i:
                        flag=0
                for x in DS:
                    if x[0]==i:
                        flag=0
                for x in SC:
                    if x[0]==i:
                        flag=0
                for x in SS:
                    if x[0]==i:
                        flag=0
                if flag:
                    DC.append([i,0,0])

    counter+=M
    print("DS: ",DS)
    print("DC: ",DC)
    print("SS: ",SS)
    print("SC: ",SC,"\n")
```

```
Initial DC: [[{1}, 0, 0], [{2}, 0, 0], [{3}, 0, 0]]

Transaction : {1, 2}
Transaction : {1}
DS:  [[{1}, 2, 2], [{2}, 1, 2]]
DC:  [[{3}, 0, 2], [{1, 2}, 0, 0]]
SS:  []
SC:  []

Transaction : {2, 3}
Transaction : set()
DS:  []
DC:  [[{1, 2}, 0, 2], [{1, 3}, 0, 0], [{2, 3}, 0, 0]]
SS:  [[{1}, 2, 4], [{2}, 2, 4], [{3}, 1, 4]]
SC:  []

Transaction : {1, 2}
```

```
Transaction : {1}
DS:  []
DC:  [[{1, 3}, 0, 2], [{2, 3}, 0, 2]]
SS:  [[{1}, 2, 4], [{2}, 2, 4], [{3}, 1, 4], [{1, 2}, 1, 4]]
SC:  []

Transaction : {2, 3}
Transaction : set()
DS:  []
DC:  []
SS:  [[{1}, 2, 4], [{2}, 2, 4], [{3}, 1, 4], [{1, 2}, 1, 4], [{2, 3}, 1, 4]]
SC:  [[{1, 3}, 0, 4]]
```

Question 7 (OPTIONAL)

(not working on my laptop due to packages version mismatch)

In [ ]: