MRIDUL HARISH, CED18I034, PROBLEM SET 5

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder
import re
import itertools
import math
import copy
```

In [ ]:
```
market_df = pd.read_csv("Market_Basket_Optimisation.csv",header=None)
```

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 [ ]:
```
records = [[y for y in x if pd.notna(y)] for x in market_df.values.tolist()]
print("Sample : ",  records[0])
```

```
Sample :  ['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes', 'whole w
eat flour', 'yams', 'cottage cheese', 'energy drink', 'tomato juice', 'low fat yogurt
', 'green tea', 'honey', 'salad', 'mineral water', 'salmon', 'antioxydant juice', 'fr
ozen smoothie', 'spinach', 'olive oil']
```

In [ ]:

```
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

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

def check(miniset,Database):
  count = 0
  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)>itert):
          continue
        count = check(miniset,Database)
        c[miniset] = count
  return c

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

  for key in rem_keys:
    Database.pop(key)
  return Database

def remove_items(c,min_sup_count):
  rem_keys = []
  for key,val in c.items():
    if(val < min_sup_count):
      rem_keys.append(key)

  for key in rem_keys:
    c.pop(key)

  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

min_sup_count = 0.005*len(records)
Li = {}
for key,val in Itemset.items():
  if(val >= min_sup_count):
    Li[key] = val

Itemset = Li

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 [ ]:

```
start = time.process_time()
Final_List = []
sot=1
final_c={}
while(1):
  print("Iteration Num ",sot)
  Database = remove_transaction(Database,sot)
  c = get_c(Li,Database,sot+1)
  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("\nTime taken = ",str(time_taken),"seconds")
```

```
Iteration Num  1
Iteration Num  2
Iteration Num  3

Time taken =  146.109375 seconds
```

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

```
frozenset({'mineral water', 'shrimp', 'eggs'})  =  39
frozenset({'mineral water', 'shrimp', 'milk'})  =  59
frozenset({'mineral water', 'frozen vegetables', 'shrimp'})  =  54
frozenset({'mineral water', 'spaghetti', 'shrimp'})  =  64
frozenset({'shrimp', 'mineral water', 'chocolate'})  =  57
frozenset({'shrimp', 'mineral water', 'ground beef'})  =  38
frozenset({'shrimp', 'chocolate', 'milk'})  =  41
frozenset({'spaghetti', 'frozen vegetables', 'shrimp'})  =  45
frozenset({'shrimp', 'frozen vegetables', 'chocolate'})  =  40
frozenset({'shrimp', 'spaghetti', 'chocolate'})  =  48
frozenset({'shrimp', 'spaghetti', 'ground beef'})  =  45
```

In [ ]:
```
te = TransactionEncoder()
te_ary = te.fit(records).transform(records)
df = pd.DataFrame(te_ary,columns = te.columns_)
print(df.shape)
```

```
(7501, 120)
```

In [ ]:
```
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              (spaghetti, mineral water, soup)
721  0.009332          (spaghetti, tomatoes, mineral water)
722  0.006399            (spaghetti, mineral water, turkey)
723  0.006266  (whole wheat rice, spaghetti, mineral water)
724  0.005066              (spaghetti, olive oil, pancakes)

[725 rows x 2 columns]

 Time taken for Mining using Apriori =  1.328125
```

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

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

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 [ ]:
```
te = TransactionEncoder()
te_ary = te.fit(records).transform(records)
df = pd.DataFrame(te_ary, columns=te.columns_)
print(df)
```

```
     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 [ ]:
```
te = TransactionEncoder()
te_ary = te.fit(records_vdf).transform(records_vdf)
df_vdf = pd.DataFrame(te_ary,columns=te.columns_)
print(df_vdf)
```

```
      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 [ ]:
```
start = time.process_time()
print(apriori(df,min_support=0.01,use_colnames=True))
time_taken = time.process_time() - start
print("Time Taken normally = ",str(time_taken)," seconds")
```

```
     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 normally =  0.03125  seconds
```

In [ ]:
```
start = time.process_time()
print(apriori(df_vdf,min_support=0.01,use_colnames=True))
time_taken = time.process_time() - start
print("Time Taken using Vertical Database Format = ",str(time_taken)," seconds")
```

```
     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          (T2, T1)
6   0.444444          (T3, T1)
7   0.111111          (T4, T1)
8   0.222222          (T5, T1)
9   0.444444          (T3, T2)
10  0.222222          (T2, T4)
11  0.222222          (T5, T2)
12  0.111111          (T3, T5)
13  0.222222      (T3, T2, T1)
14  0.111111      (T4, T2, T1)
15  0.222222      (T5, T2, T1)
16  0.111111      (T3, T5, T1)
17  0.111111      (T3, T5, T2)
18  0.111111  (T3, T5, T2, T1)
Time Taken using Vertical Database Format =  0.015625  seconds
```

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 [ ]:
```
records = [[y for y in x if pd.notna(y)] for x in market_df.values.tolist()]
print(records[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', 'salmon', 'antioxydant juice', 'frozen smoot
hie', 'spinach', 'olive oil']
```

In [ ]:

```
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[frozense

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

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

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[frozense

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
```

In [ ]:

```
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 Num ",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_v
```

```
Iteration Num  2
Iteration Num  3
Iteration Num  4

 Time Taken for Mining the itemset with min_support of 37.505 = 0.375 seconds
```

In [ ]:

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

```
frozenset({'mineral water', 'shrimp', 'eggs'})   frozenset({'T126', 'T108', 'T1327',
'T4095', 'T745', 'T2179', 'T2357', 'T237', 'T2008', 'T3528', 'T7468', 'T3869', 'T1214
', 'T4925', 'T92', 'T667', 'T3616', 'T657', 'T3880', 'T6776', 'T478', 'T4993', 'T111
', 'T1894', 'T976', 'T7475', 'T6973', 'T1817', 'T2262', 'T3696', 'T1726', 'T7370', 'T
1059', 'T1608', 'T6101', 'T144', 'T1226', 'T143', 'T5062'})

frozenset({'mineral water', 'shrimp', 'milk'})   frozenset({'T126', 'T108', 'T5016',
'T745', 'T2179', 'T2242', 'T3629', 'T7265', 'T7344', 'T2125', 'T2796', 'T5466', 'T160
5', 'T4933', 'T2198', 'T3242', 'T3260', 'T4121', 'T3869', 'T5219', 'T2156', 'T6022',
'T2105', 'T2633', 'T667', 'T3616', 'T789', 'T3481', 'T7131', 'T5746', 'T2065', 'T2359
', 'T796', 'T2510', 'T809', 'T3041', 'T1606', 'T6157', 'T4993', 'T5639', 'T7475', 'T5
019', 'T3755', 'T2430', 'T3643', 'T5698', 'T801', 'T5991', 'T3660', 'T504', 'T2335',
'T5792', 'T5277', 'T7370', 'T1059', 'T621', 'T2783', 'T5062', 'T6716'})

frozenset({'mineral water', 'frozen vegetables', 'shrimp'})   frozenset({'T126', 'T10
38', 'T5607', 'T2179', 'T2357', 'T3447', 'T1605', 'T1366', 'T4933', 'T2198', 'T3579',
'T3242', 'T3981', 'T5219', 'T2668', 'T5460', 'T2820', 'T6022', 'T984', 'T2633', 'T492
5', 'T2105', 'T3616', 'T5247', 'T5084', 'T3481', 'T2618', 'T1700', 'T3880', 'T7131',
'T5746', 'T6776', 'T472', 'T1606', 'T2173', 'T1894', 'T5785', 'T5639', 'T1959', 'T697
3', 'T3643', 'T3660', 'T2335', 'T6523', 'T6101', 'T1993', 'T3370', 'T143', 'T3059', '
T2783', 'T1226', 'T1393', 'T700', 'T5062'})

frozenset({'spaghetti', 'mineral water', 'shrimp'})   frozenset({'T126', 'T1389', 'T2
638', 'T1327', 'T4494', 'T4932', 'T5016', 'T4095', 'T2357', 'T5607', 'T676', 'T1657',
'T3528', 'T1605', 'T2198', 'T3579', 'T7468', 'T3242', 'T3981', 'T2668', 'T1214', 'T28
20', 'T4925', 'T2633', 'T92', 'T2105', 'T4710', 'T2618', 'T4096', 'T6544', 'T2065', '
T2359', 'T796', 'T809', 'T462', 'T1606', 'T2173', 'T6554', 'T1894', 'T1346', 'T5639',
'T6973', 'T5019', 'T2430', 'T5855', 'T3723', 'T3643', 'T5991', 'T801', 'T3660', 'T504
```

```
', 'T3696', 'T4914', 'T6523', 'T5479', 'T1059', 'T1993', 'T3693', 'T2783', 'T3059', '
T144', 'T4830', 'T2309', 'T6716'})

frozenset({'chocolate', 'mineral water', 'shrimp'})   frozenset({'T126', 'T4494', 'T5
016', 'T4766', 'T2357', 'T745', 'T676', 'T3629', 'T2796', 'T1657', 'T3528', 'T1366',
'T2198', 'T3579', 'T7468', 'T3260', 'T3424', 'T1214', 'T5460', 'T2156', 'T6022', 'T26
33', 'T5302', 'T3616', 'T1700', 'T1785', 'T2618', 'T5746', 'T472', 'T1606', 'T2173',
'T1894', 'T5639', 'T891', 'T7475', 'T5019', 'T2430', 'T3723', 'T3696', 'T3660', 'T172
6', 'T6523', 'T5479', 'T5277', 'T2696', 'T1608', 'T621', 'T660', 'T3059', 'T2293', 'T
4830', 'T1226', 'T1393', 'T2670', 'T143', 'T5062', 'T5792'})

frozenset({'ground beef', 'mineral water', 'shrimp'})   frozenset({'T2638', 'T1327',
'T4932', 'T4766', 'T5607', 'T2008', 'T7344', 'T2796', 'T3544', 'T4133', 'T1605', 'T32
42', 'T3424', 'T3124', 'T3981', 'T2668', 'T2820', 'T2633', 'T5084', 'T4245', 'T2359',
'T2510', 'T462', 'T1606', 'T5395', 'T4993', 'T2173', 'T1894', 'T6973', 'T828', 'T3643
', 'T5991', 'T3696', 'T4914', 'T6523', 'T3693', 'T143', 'T5792'})

frozenset({'chocolate', 'shrimp', 'milk'})   frozenset({'T126', 'T5016', 'T745', 'T36
29', 'T2796', 'T1329', 'T3378', 'T2925', 'T2198', 'T469', 'T6439', 'T1835', 'T3260',
'T634', 'T2156', 'T6022', 'T2633', 'T4289', 'T3616', 'T1607', 'T5746', 'T4326', 'T160
6', 'T3029', 'T5639', 'T7475', 'T1741', 'T5019', 'T2430', 'T566', 'T1375', 'T3195', '
T3660', 'T3686', 'T5277', 'T937', 'T621', 'T7324', 'T161', 'T5062', 'T5792'})

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

frozenset({'chocolate', 'frozen vegetables', 'shrimp'})   frozenset({'T126', 'T2357',
'T6176', 'T6986', 'T1393', 'T4921', 'T3412', 'T3378', 'T1366', 'T2925', 'T2198', 'T35
79', 'T469', 'T634', 'T5460', 'T6022', 'T2633', 'T3616', 'T1700', 'T2618', 'T5746', '
T472', 'T3509', 'T1606', 'T2173', 'T3029', 'T1894', 'T1886', 'T5639', 'T6578', 'T3660
', 'T1399', 'T6523', 'T2079', 'T7324', 'T3059', 'T1226', 'T5400', 'T143', 'T5062'})

frozenset({'chocolate', 'spaghetti', 'shrimp'})   frozenset({'T126', 'T4494', 'T3006
', 'T5016', 'T2357', 'T676', 'T1531', 'T4936', 'T5697', 'T1657', 'T772', 'T3528', 'T2
198', 'T3579', 'T7468', 'T1214', 'T2320', 'T2633', 'T2618', 'T1941', 'T1702', 'T3448
', 'T6060', 'T5487', 'T3509', 'T1606', 'T2173', 'T2734', 'T4988', 'T1894', 'T5639', '
T6578', 'T1741', 'T5019', 'T3576', 'T2430', 'T3723', 'T3195', 'T3696', 'T3660', 'T652
3', 'T5479', 'T3059', 'T4998', 'T4830', 'T7482', 'T1471', 'T916'})

frozenset({'ground beef', 'spaghetti', 'shrimp'})   frozenset({'T2638', 'T1327', 'T49
32', 'T4074', 'T5607', 'T1558', 'T1531', 'T6718', 'T5697', 'T5385', 'T1605', 'T3493',
'T3242', 'T1880', 'T3981', 'T2668', 'T2820', 'T2633', 'T2384', 'T3400', 'T4907', 'T35
60', 'T2359', 'T2740', 'T462', 'T1606', 'T2173', 'T2734', 'T4988', 'T1894', 'T2382',
'T6086', 'T6973', 'T3576', 'T3643', 'T3195', 'T5991', 'T3696', 'T4914', 'T6523', 'T28
11', 'T3085', 'T3693', 'T4998', 'T3864'})
```

Question 4 - Extend the basic Apriori algorithm to generate Frequent Patterns which differentiate ab from ba (ordered patterns generation).

In [ ]:

```
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>"

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

print(sequences)
```

```
{10: 'aabcacdcf', 20: 'adcabcae', 30: 'efabdfcb', 40: 'egafcbc'}
```

In [ ]:

```
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

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

In [ ]:

```
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

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

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

```
Iteration No:  1
Iteration No:  2
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.

In [ ]:

```
#HASH VARIANT
transactions = [{'A','C','D'},{'B','C','E'},{'A','B','C','E'},{'B','E'}]

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

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

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

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)


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

print(database_hash)
```

```
{'T1': [['A', 'C'], ['C', 'D'], ['A', 'D']], 'T2': [['C', 'E'], ['B', 'C'], ['B', 'E
']], 'T3': [['C', 'E'], ['B', 'C'], ['A', 'C'], ['B', 'E'], ['A', 'E'], ['A', 'B']],
'T4': [['B', 'E']]}
```

In [ ]:
```
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]

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])

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({'E', 'B'}): 3, frozen
set({'C', 'E'}): 3}
```

In [ ]:
```
#PARTITIONING APPROACH
records = [[y for y in x if pd.notna(y)] for x in market_df.values.tolist()]
Database={}
for i in range(len(records)):
    Database["T"+str(i+1)]=records[i]

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

```
      asparagus  almonds  antioxydant juice  asparagus  avocado  babies food  \
0         False     True               True      False     True        False
1         False    False              False      False    False        False
2         False    False              False      False    False        False
3         False    False              False      False     True        False
4         False    False              False      False    False        False
...         ...      ...                ...        ...      ...          ...
7496      False    False              False      False    False        False
7497      False    False              False      False    False        False
7498      False    False              False      False    False        False
7499      False    False              False      False    False        False
7500      False    False              False      False    False        False

      bacon  barbecue sauce  black tea  blueberries  ...  turkey  \
0     False           False      False        False  ...   False
1     False           False      False        False  ...   False
2     False           False      False        False  ...   False
3     False           False      False        False  ...    True
4     False           False      False        False  ...   False
...     ...             ...        ...          ...  ...     ...
```

```
7496  False           False      False        False  ...   False
7497  False           False      False        False  ...   False
7498  False           False      False        False  ...   False
7499  False           False      False        False  ...   False
7500  False           False      False        False  ...   False

      vegetables mix  water spray  white wine  whole weat flour  \
0               True        False       False              True
1              False        False       False             False
2              False        False       False             False
3              False        False       False             False
4              False        False       False             False
...              ...          ...         ...               ...
7496           False        False       False             False
7497           False        False       False             False
7498           False        False       False             False
7499           False        False       False             False
7500           False        False       False             False

      whole wheat pasta  whole wheat rice   yams  yogurt cake  zucchini
0                 False             False   True        False     False
1                 False             False  False        False     False
2                 False             False  False        False     False
3                 False             False  False        False     False
4                 False              True  False        False     False
...                 ...               ...    ...          ...       ...
7496              False             False  False        False     False
7497              False             False  False        False     False
7498              False             False  False        False     False
7499              False             False  False        False     False
7500              False             False  False         True     False

[7501 rows x 120 columns]
```

In [ ]:

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

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))

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

print(final_results)
```

```
{frozenset({'chocolate'}): 1229.0, frozenset({'eggs'}): 1348.0, frozenset({'french fr
ies'}): 1282.0, frozenset({'green tea'}): 991.0, frozenset({'milk'}): 972.0, frozense
t({'mineral water'}): 1788.0, frozenset({'spaghetti'}): 1306.0}
```

Question 6 - Implement the Dynamic Itemset Counting Algorithm for Frequent Itemset Generation.

In [ ]:

```
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)

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)

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 [ ]:

```
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]]
```