In [141…
```
import numpy as np
import pandas as pd
import math

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules
from mlxtend.frequent_patterns import fpgrowth
import time
```

In [ ]:
```
```

Test drive the basic version of FP Growth algorithms for Frequent Itemset Mining using the package / library support in the platform of your choice. Test it with various support and confidence measures and generate a time comparison for varied data set sizes. To do the performance comparison you may use benchmark datasets provided for FIM such as the FIMI workshop or other sources.

Testing on test data..

In [142…
```
# https://coderspacket.com/implementing-fp-growth-algorithm-in-machine-learning-using-python#:~:text=Confidence%20%3A%20C%20%28i-%3Ej%29%20%3D%20S,%28i
# https://www.kaggle.com/code/rjmanoj/fp-growth-algorithm-frequent-itemset-pattern



data = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]


TE = TransactionEncoder()
TE_arr = TE.fit(data).transform(data)
TE_arr
```

Out[142]:
```
array([[False, False, False,  True, False,  True,  True,  True,  True,
        False,  True],
       [False, False,  True,  True, False,  True, False,  True,  True,
        False,  True],
       [ True, False, False,  True, False,  True,  True, False, False,
        False, False],
       [False,  True, False, False, False,  True,  True, False, False,
         True,  True],
       [False,  True, False,  True,  True,  True, False, False,  True,
        False, False]])
```

In [143…
```
TE_arr.astype("int")
```

Out[143]:
```
array([[0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1],
       [0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
       [1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1],
       [0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0]])
```

In [144…
```
df = pd.DataFrame(TE_arr, columns=TE.columns_)
df
```

Out[144]:

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

In [145…
```
frequent_itemsets_fp = fpgrowth(df, min_support=0.6, use_colnames=True)
frequent_itemsets_fp
```

Out[145]:

	support	itemsets
0	1.0	(Kidney Beans)
1	0.8	(Eggs)
2	0.6	(Yogurt)
3	0.6	(Onion)
4	0.6	(Milk)
5	0.8	(Kidney Beans, Eggs)
6	0.6	(Kidney Beans, Yogurt)
7	0.6	(Onion, Eggs)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Onion, Kidney Beans, Eggs)
10	0.6	(Kidney Beans, Milk)

In [146…
```
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=0.8)
rules_fp.iloc[:,:6]
```

Out[146]:

	antecedents	consequents	antecedent support	consequent support	support	confidence
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.8
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.0
2	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.0
3	(Onion)	(Eggs)	0.6	0.8	0.6	1.0
4	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.0
5	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.0
6	(Onion, Eggs)	(Kidney Beans)	0.6	1.0	0.6	1.0
7	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.0
8	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.0

In [ ]:

Running FP-growth on mushroom FIMI dataset

In [147…

```
df = pd.read_csv("mushroom.dat", delimiter=" ")
df.drop(['Unnamed: 23'], axis=1, inplace=True)
df
```

Out[147]:

	1	3	9	13	23	25	34	36	38	40	...	63	67	76	85	86	90	93	98	107	113
0	2	3	9	14	23	26	34	36	39	40	...	63	67	76	85	86	90	93	99	108	114
1	2	4	9	15	23	27	34	36	39	41	...	63	67	76	85	86	90	93	99	108	115
2	1	3	10	15	23	25	34	36	38	41	...	63	67	76	85	86	90	93	98	107	113
3	2	3	9	16	24	28	34	37	39	40	...	63	67	76	85	86	90	94	99	109	114
4	2	3	10	14	23	26	34	36	39	41	...	63	67	76	85	86	90	93	98	108	114
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8118	2	7	9	13	24	28	35	36	39	50	...	63	73	83	85	88	90	93	106	112	119
8119	2	3	9	13	24	28	35	36	39	50	...	63	73	83	85	87	90	93	106	110	119
8120	2	6	9	13	24	28	35	36	39	41	...	63	73	83	85	88	90	93	106	112	119
8121	1	7	10	13	24	31	34	36	38	48	...	66	67	76	85	86	90	94	102	110	119
8122	2	3	9	13	24	28	35	36	39	50	...	63	73	83	85	88	90	93	104	112	119

8123 rows × 23 columns

In [148…

```
TE = TransactionEncoder()
data = df.values.tolist()
TE_arr = TE.fit(data).transform(data)

df = pd.DataFrame(TE_arr, columns=TE.columns_)
df
```

Out[148]:

	1	2	3	4	5	6	7	8	9	10	...	110	111	112	113	114	115	116	117	118	119
0	False	True	True	False	False	False	False	False	True	False	...	False	False	False	False	True	False	False	False	False	False
1	False	True	False	True	False	False	False	False	True	False	...	False	False	False	False	False	True	False	False	False	False
2	True	False	True	False	False	False	False	False	False	True	...	False	False	False	True	False	False	False	False	False	False
3	False	True	True	False	False	False	False	False	True	False	...	False	False	False	False	True	False	False	False	False	False
4	False	True	True	False	False	False	False	False	False	True	...	False	False	False	False	True	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8118	False	True	False	False	False	False	True	False	True	False	...	False	False	True	False	False	False	False	False	False	True
8119	False	True	True	False	False	False	False	False	True	False	...	True	False	False	False	False	False	False	False	False	True
8120	False	True	False	False	False	True	False	False	True	False	...	False	False	True	False	False	False	False	False	False	True
8121	True	False	False	False	False	False	True	False	False	True	...	True	False	False	False	False	False	False	False	False	True
8122	False	True	True	False	False	False	False	False	True	False	...	False	False	True	False	False	False	False	False	False	True

8123 rows × 119 columns

In [150…

```
import warnings
warnings.filterwarnings('ignore')

min_supports = (30, 40, 50)
confidences = (70, 80, 90)

print("========================================================================================\n\n")

for each in min_supports:
    for percentage in confidences:
        start_time = time.process_time()
        # Building the model
        support = each/100
        frequent_itemsets_fp = fpgrowth(df, min_support=support, use_colnames=True)

        # Collecting the inferred rules in a dataframe
        minThreshold = percentage/100
        rules = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=minThreshold)
        rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])

        end_time = time.process_time()
        time_taken = end_time - start_time
        print(f"Time Taken for min_support = {each}% and min_threshold = {minThreshold} is {time_taken}sec \n")
        print("========================================================================================\n\n\n")
```

```
================================================================================

Time Taken for min_support = 30% and min_threshold = 0.7 is 0.546875sec

================================================================================

Time Taken for min_support = 30% and min_threshold = 0.8 is 0.390625sec

================================================================================

Time Taken for min_support = 30% and min_threshold = 0.9 is 0.265625sec

================================================================================

Time Taken for min_support = 40% and min_threshold = 0.7 is 0.1875sec

================================================================================

Time Taken for min_support = 40% and min_threshold = 0.8 is 0.0625sec

================================================================================

Time Taken for min_support = 40% and min_threshold = 0.9 is 0.09375sec

================================================================================

Time Taken for min_support = 50% and min_threshold = 0.7 is 0.078125sec

================================================================================

Time Taken for min_support = 50% and min_threshold = 0.8 is 0.046875sec

================================================================================
```

```
Time Taken for min_support = 50% and min_threshold = 0.9 is 0.0625sec

================================================================================
```

In [54]:
```
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=0.8)
rules_fp.iloc[:,:6]
```

Out[54]:

	antecedents	consequents	antecedent support	consequent support	support	confidence
0	(85)	(86)	1.000000	0.975379	0.975379	0.975379
1	(86)	(85)	0.975379	1.000000	0.975379	1.000000
2	(34)	(85)	0.974147	1.000000	0.974147	1.000000
3	(85)	(34)	1.000000	0.974147	0.974147	0.974147
4	(34)	(86)	0.974147	0.975379	0.973163	0.998989
...	...	...	...	...	...	...
199	(34, 59)	(85, 86)	0.613443	0.975379	0.613443	1.000000
200	(59, 85)	(34, 86)	0.637080	0.973163	0.613443	0.962899
201	(59, 86)	(34, 85)	0.613443	0.974147	0.613443	1.000000
202	(59)	(34, 85, 86)	0.637080	0.973163	0.613443	0.962899
203	(63)	(85)	0.607534	1.000000	0.607534	1.000000

204 rows × 6 columns

In [ ]:

In [ ]:

2. 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 [151…]:
```
data = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

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

Out[151]:

	0	1	2	3	4	5
0	Milk	Onion	Nutmeg	Kidney Beans	Eggs	Yogurt
1	Dill	Onion	Nutmeg	Kidney Beans	Eggs	Yogurt
2	Milk	Apple	Kidney Beans	Eggs	None	None
3	Milk	Unicorn	Corn	Kidney Beans	Yogurt	None
4	Corn	Onion	Onion	Kidney Beans	Ice cream	Eggs

In [152…]:
```
# converting data into list and dropping all NAN
records = [[each for each in row if pd.notna(each)] for row in df.values.tolist()]
records
```

Out[152]:
```
[['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
 ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
 ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
 ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
 ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

In [153…]:
```
Database={}
for i in range(len(records)):
    Database["T"+str(i+1)]=records[i]

Database
```

Out[153]:
```
{'T1': ['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
 'T2': ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
 'T3': ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
 'T4': ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
 'T5': ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']}
```

In [ ]:

In [154…

```
Itemset={}
for row in records:
    for item in row:
        Itemset[frozenset([item])] = Itemset.get(frozenset([item]), 0) + 1

Itemset
```

Out[154]:

```
{frozenset({'Milk'}): 3,
 frozenset({'Onion'}): 4,
 frozenset({'Nutmeg'}): 2,
 frozenset({'Kidney Beans'}): 5,
 frozenset({'Eggs'}): 4,
 frozenset({'Yogurt'}): 3,
 frozenset({'Dill'}): 1,
 frozenset({'Apple'}): 1,
 frozenset({'Unicorn'}): 1,
 frozenset({'Corn'}): 2,
 frozenset({'Ice cream'}): 1}
```

In [155…

```
# counting the number of occurances of ecah items and filtering based on that..
Min_Sup_Count = 1
L = {}

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

Itemset = L
Itemset
```

Out[155]:

```
{frozenset({'Milk'}): 3,
 frozenset({'Onion'}): 4,
 frozenset({'Nutmeg'}): 2,
 frozenset({'Kidney Beans'}): 5,
 frozenset({'Eggs'}): 4,
 frozenset({'Yogurt'}): 3,
 frozenset({'Dill'}): 1,
 frozenset({'Apple'}): 1,
 frozenset({'Unicorn'}): 1,
 frozenset({'Corn'}): 2,
 frozenset({'Ice cream'}): 1}
```

In [156…

```
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, sot):
    c={}
    for key in Li:
        for key1 in Li:
            if (key1 != key):
                miniset = key1.union(key)

                if(len(miniset) > (sot+1)):
                    continue

                count = check(miniset, Database)
                c[miniset] = count

#     display(c)
    return c


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

    for key in rem_keys:
        Database.pop(key)

#     display(Database)
    return Database

# removing based on the min_support_count
def remove_min_support(c, Min_Sup_Count):
    removed_keys = []
    for key, val in c.items():
        if(val < Min_Sup_Count):
            removed_keys.append(key)

    for key in removed_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
```

In [ ]:

In [157…

```
sot = 1
final_c = {}

while(1):
    print("Iteration No: ",sot)
    Database = remove_transaction(Database, sot)

    sot += 1
    c = get_c(L, Database, sot+1)
    #print(c)

    L = remove_min_support(c, Min_Sup_Count)
    Database = remove_item(L, Database)

    if(len(L)==0):
        break
    else:
        final_c = L

    display(final_c)
```

```
Iteration No:  1
```

```
{frozenset({'Milk', 'Onion'}): 1,
 frozenset({'Milk', 'Nutmeg'}): 1,
 frozenset({'Kidney Beans', 'Milk'}): 3,
 frozenset({'Eggs', 'Milk'}): 2,
 frozenset({'Milk', 'Yogurt'}): 2,
 frozenset({'Apple', 'Milk'}): 1,
 frozenset({'Milk', 'Unicorn'}): 1,
 frozenset({'Corn', 'Milk'}): 1,
 frozenset({'Nutmeg', 'Onion'}): 2,
 frozenset({'Kidney Beans', 'Onion'}): 3,
 frozenset({'Eggs', 'Onion'}): 3,
 frozenset({'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Onion'}): 1,
 frozenset({'Corn', 'Onion'}): 1,
 frozenset({'Ice cream', 'Onion'}): 1,
 frozenset({'Kidney Beans', 'Nutmeg'}): 2,
 frozenset({'Eggs', 'Nutmeg'}): 2,
 frozenset({'Nutmeg', 'Yogurt'}): 2,
 frozenset({'Dill', 'Nutmeg'}): 1,
 frozenset({'Eggs', 'Kidney Beans'}): 4,
 frozenset({'Kidney Beans', 'Yogurt'}): 3,
 frozenset({'Dill', 'Kidney Beans'}): 1,
 frozenset({'Apple', 'Kidney Beans'}): 1,
 frozenset({'Kidney Beans', 'Unicorn'}): 1,
 frozenset({'Corn', 'Kidney Beans'}): 2,
 frozenset({'Ice cream', 'Kidney Beans'}): 1,
 frozenset({'Eggs', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs'}): 1,
 frozenset({'Apple', 'Eggs'}): 1,
 frozenset({'Corn', 'Eggs'}): 1,
 frozenset({'Eggs', 'Ice cream'}): 1,
 frozenset({'Dill', 'Yogurt'}): 1,
 frozenset({'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Unicorn'}): 1,
 frozenset({'Corn', 'Ice cream'}): 1}
Iteration No:  2
```

```
{frozenset({'Milk', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Onion'}): 1,
 frozenset({'Eggs', 'Milk', 'Onion'}): 1,
 frozenset({'Milk', 'Onion', 'Yogurt'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Eggs', 'Milk', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Onion'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Milk', 'Onion', 'Yogurt'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Nutmeg'}): 1,
 frozenset({'Eggs', 'Milk', 'Nutmeg'}): 1,
 frozenset({'Milk', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Milk', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk'}): 2,
 frozenset({'Kidney Beans', 'Milk', 'Yogurt'}): 2,
 frozenset({'Apple', 'Kidney Beans', 'Milk'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Unicorn'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Milk'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Yogurt'}): 1,
 frozenset({'Apple', 'Eggs', 'Kidney Beans', 'Milk'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Milk', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Milk', 'Unicorn'}): 1,
 frozenset({'Eggs', 'Milk', 'Yogurt'}): 1,
 frozenset({'Apple', 'Eggs', 'Milk'}): 1,
 frozenset({'Milk', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Milk', 'Yogurt'}): 1,
 frozenset({'Corn', 'Milk', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Milk', 'Unicorn'}): 1,
 frozenset({'Kidney Beans', 'Nutmeg', 'Onion'}): 2,
 frozenset({'Eggs', 'Nutmeg', 'Onion'}): 2,
 frozenset({'Nutmeg', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion'}): 2,
 frozenset({'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Kidney Beans', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Eggs', 'Nutmeg', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Dill', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Onion'}): 3,
 frozenset({'Kidney Beans', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Ice cream', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Corn', 'Eggs', 'Kidney Beans', 'Onion'}): 1,
```

```
 frozenset({'Eggs', 'Ice cream', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Dill', 'Kidney Beans', 'Onion', 'Yogurt'}): 1,
 frozenset({'Corn', 'Ice cream', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Eggs', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Onion'}): 1,
 frozenset({'Corn', 'Eggs', 'Onion'}): 1,
 frozenset({'Eggs', 'Ice cream', 'Onion'}): 1,
 frozenset({'Dill', 'Eggs', 'Onion', 'Yogurt'}): 1,
 frozenset({'Corn', 'Eggs', 'Ice cream', 'Onion'}): 1,
 frozenset({'Dill', 'Onion', 'Yogurt'}): 1,
 frozenset({'Corn', 'Ice cream', 'Onion'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg'}): 2,
 frozenset({'Kidney Beans', 'Nutmeg', 'Yogurt'}): 2,
 frozenset({'Dill', 'Kidney Beans', 'Nutmeg'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg'}): 1,
 frozenset({'Dill', 'Kidney Beans', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Nutmeg', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Nutmeg'}): 1,
 frozenset({'Dill', 'Eggs', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Dill', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Kidney Beans'}): 1,
 frozenset({'Apple', 'Eggs', 'Kidney Beans'}): 1,
 frozenset({'Corn', 'Eggs', 'Kidney Beans'}): 1,
 frozenset({'Eggs', 'Ice cream', 'Kidney Beans'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Yogurt'}): 1,
 frozenset({'Corn', 'Eggs', 'Ice cream', 'Kidney Beans'}): 1,
 frozenset({'Dill', 'Kidney Beans', 'Yogurt'}): 1,
 frozenset({'Kidney Beans', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Unicorn'}): 1,
 frozenset({'Corn', 'Ice cream', 'Kidney Beans'}): 1,
 frozenset({'Dill', 'Eggs', 'Yogurt'}): 1,
 frozenset({'Corn', 'Eggs', 'Ice cream'}): 1,
 frozenset({'Corn', 'Unicorn', 'Yogurt'}): 1}
Iteration No:  3
```

```
{frozenset({'Kidney Beans', 'Milk', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Eggs', 'Milk', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Onion'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Milk', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Milk', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Yogurt'}): 1,
 frozenset({'Apple', 'Eggs', 'Kidney Beans', 'Milk'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Milk', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Milk', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Milk', 'Unicorn'}): 1,
 frozenset({'Corn', 'Milk', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion'}): 2,
 frozenset({'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Kidney Beans', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Dill', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Nutmeg', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Dill', 'Eggs', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Dill', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Corn', 'Eggs', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Eggs', 'Ice cream', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Onion', 'Yogurt'}): 1,
 frozenset({'Corn', 'Eggs', 'Ice cream', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Dill', 'Kidney Beans', 'Onion', 'Yogurt'}): 1,
 frozenset({'Corn', 'Ice cream', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Dill', 'Eggs', 'Onion', 'Yogurt'}): 1,
 frozenset({'Corn', 'Eggs', 'Ice cream', 'Onion'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Dill', 'Kidney Beans', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Dill', 'Eggs', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Yogurt'}): 1,
 frozenset({'Corn', 'Eggs', 'Ice cream', 'Kidney Beans'}): 1,
```

```
 frozenset({'Corn', 'Kidney Beans', 'Unicorn', 'Yogurt'}): 1}
Iteration No:  4
{frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Kidney Beans', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Onion', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Yogurt'}): 1,
 frozenset({'Corn', 'Kidney Beans', 'Milk', 'Unicorn', 'Yogurt'}): 1,
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 2,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Dill', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Dill', 'Eggs', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Onion', 'Yogurt'}): 1,
 frozenset({'Corn', 'Eggs', 'Ice cream', 'Kidney Beans', 'Onion'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Yogurt'}): 1}
Iteration No:  5
{frozenset({'Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'}): 1,
 frozenset({'Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}): 1}
Iteration No:  6
```

In [ ]:

In [ ]:

In [ ]:

Question 3 -Test drive any one implementation in (1) or (2) adopting a Vertical Transaction Database format.

In [158…

```
records=[['a','d','e','g','h','i'],
         ['a','b','c','d','f','h','i'],
         ['c','e','f','g','h','i'],
         ['b','d'],
         ['a','h']
        ]
```

In [159…

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

Database
```

Out[159]:

```
{'T1': ['a', 'd', 'e', 'g', 'h', 'i'],
 'T2': ['a', 'b', 'c', 'd', 'f', 'h', 'i'],
 'T3': ['c', 'e', 'f', 'g', 'h', 'i'],
 'T4': ['b', 'd'],
 'T5': ['a', 'h']}
```

In [160…

```
TE = TransactionEncoder()
TE_arr = TE.fit(records).transform(records)
df = pd.DataFrame(TE_arr, columns=TE.columns_)
df
```

Out[160]:

	a	b	c	d	e	f	g	h	i
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 [161…

```
# converting into vertical database..
# counting occurances
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])])

Database_vdf
```

Out[161]:

```
{frozenset({'a'}): frozenset({'T1', 'T2', 'T5'}),
 frozenset({'d'}): frozenset({'T1', 'T2', 'T4'}),
 frozenset({'e'}): frozenset({'T1', 'T3'}),
 frozenset({'g'}): frozenset({'T1', 'T3'}),
 frozenset({'h'}): frozenset({'T1', 'T2', 'T3', 'T5'}),
 frozenset({'i'}): frozenset({'T1', 'T2', 'T3'}),
 frozenset({'b'}): frozenset({'T2', 'T4'}),
 frozenset({'c'}): frozenset({'T2', 'T3'}),
 frozenset({'f'}): frozenset({'T2', 'T3'})}
```

In [162…

```
records_vdf = []
for val in Database_vdf.values():
    records_vdf.append(val)

records_vdf
```

Out[162]:
```
[frozenset({'T1', 'T2', 'T5'}),
 frozenset({'T1', 'T2', 'T4'}),
 frozenset({'T1', 'T3'}),
 frozenset({'T1', 'T3'}),
 frozenset({'T1', 'T2', 'T3', 'T5'}),
 frozenset({'T1', 'T2', 'T3'}),
 frozenset({'T2', 'T4'}),
 frozenset({'T2', 'T3'}),
 frozenset({'T2', 'T3'})]
```

In [163…
```
TE = TransactionEncoder()
TE_arr = TE.fit(records_vdf).transform(records_vdf)
df_vdf = pd.DataFrame(TE_arr, columns=TE.columns_)
df_vdf
```

Out[163]:

	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 [164…
```
from mlxtend.frequent_patterns import apriori
import time

startTime = time.process_time()

# Building the model
frquent_items = apriori(df, min_support = 0.2, use_colnames = True)
rules = association_rules(frquent_items, metric ="confidence", min_threshold = 0.8)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])

endTime = time.process_time()
print("\n Time Taken of Vertical Apriori: ", endTime-startTime)

display(rules)
```

```
 Time Taken of Vertical Apriori:  0.0
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
138	(c, d)	(a, b)	0.2	0.2	0.2	1.0	5.00	0.16	inf
139	(a, b)	(c, d)	0.2	0.2	0.2	1.0	5.00	0.16	inf
144	(a, c)	(f, b)	0.2	0.2	0.2	1.0	5.00	0.16	inf
145	(c, b)	(f, a)	0.2	0.2	0.2	1.0	5.00	0.16	inf
147	(f, a)	(c, b)	0.2	0.2	0.2	1.0	5.00	0.16	inf
...	...	...	...	...	...	...	...	...	...
1206	(f, i, d, a, c)	(h)	0.2	0.8	0.2	1.0	1.25	0.04	inf
1242	(i, d, a, e, g)	(h)	0.2	0.8	0.2	1.0	1.25	0.04	inf
1277	(f, i, d, b, c)	(h)	0.2	0.8	0.2	1.0	1.25	0.04	inf
1319	(f, i, e, g, c)	(h)	0.2	0.8	0.2	1.0	1.25	0.04	inf
1352	(f, i, d, a, b, c)	(h)	0.2	0.8	0.2	1.0	1.25	0.04	inf

1448 rows × 9 columns

Testing on FIMI dataset

In [165…

```
df = pd.read_csv("mushroom.dat", delimiter=" ")
df.drop(['Unnamed: 23'], axis=1, inplace=True)
df
```

Out[165]:

	1	3	9	13	23	25	34	36	38	40	...	63	67	76	85	86	90	93	98	107	113
0	2	3	9	14	23	26	34	36	39	40	...	63	67	76	85	86	90	93	99	108	114
1	2	4	9	15	23	27	34	36	39	41	...	63	67	76	85	86	90	93	99	108	115
2	1	3	10	15	23	25	34	36	38	41	...	63	67	76	85	86	90	93	98	107	113
3	2	3	9	16	24	28	34	37	39	40	...	63	67	76	85	86	90	94	99	109	114
4	2	3	10	14	23	26	34	36	39	41	...	63	67	76	85	86	90	93	98	108	114
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8118	2	7	9	13	24	28	35	36	39	50	...	63	73	83	85	88	90	93	106	112	119
8119	2	3	9	13	24	28	35	36	39	50	...	63	73	83	85	87	90	93	106	110	119
8120	2	6	9	13	24	28	35	36	39	41	...	63	73	83	85	88	90	93	106	112	119
8121	1	7	10	13	24	31	34	36	38	48	...	66	67	76	85	86	90	94	102	110	119
8122	2	3	9	13	24	28	35	36	39	50	...	63	73	83	85	88	90	93	104	112	119

8123 rows × 23 columns

In [166…

```
# converting data into list and dropping all NAN
records = [[each for each in row if pd.notna(each)] for row in df.values.tolist()]
print(records[:2])
```

```
[[2, 3, 9, 14, 23, 26, 34, 36, 39, 40, 52, 55, 59, 63, 67, 76, 85, 86, 90, 93, 99, 108, 114], [2, 4, 9, 15, 23, 27, 34, 36, 39, 41, 52, 55, 59, 63, 67,
76, 85, 86, 90, 93, 99, 108, 115]]
```

In [167…

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

TE = TransactionEncoder()
TE_arr = TE.fit(records).transform(records)
df = pd.DataFrame(TE_arr, columns=TE.columns_)

# converting into vertical database..
# counting occurances
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 val in Database_vdf.values():
    records_vdf.append(val)


TE = TransactionEncoder()
TE_arr = TE.fit(records_vdf).transform(records_vdf)
df_vdf = pd.DataFrame(TE_arr, columns=TE.columns_)


startTime = time.process_time()

# Building the model
frquent_items = apriori(df, min_support = 0.6, use_colnames = True)
rules = association_rules(frquent_items, metric ="confidence", min_threshold = 0.8)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])

endTime = time.process_time()
print("\n Time Taken of Vertical Apriori: ", endTime-startTime)

display(rules)
```

```
 Time Taken of Vertical Apriori:  0.015625
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
55	(59, 86)	(34)	0.613443	0.974147	0.613443	1.000000	1.026539	0.015859	inf
70	(90, 86)	(34)	0.897082	0.974147	0.897082	1.000000	1.026539	0.023192	inf
132	(90, 36, 86)	(34)	0.772005	0.974147	0.772005	1.000000	1.026539	0.019958	inf
151	(59, 85, 86)	(34)	0.613443	0.974147	0.613443	1.000000	1.026539	0.015859	inf
154	(59, 86)	(34, 85)	0.613443	0.974147	0.613443	1.000000	1.026539	0.015859	inf
...	...	...	...	...	...	...	...	...	...
115	(85, 86)	(34, 36)	0.975379	0.812631	0.812631	0.833144	1.025243	0.020008	1.12294
119	(86)	(34, 36, 85)	0.975379	0.812631	0.812631	0.833144	1.025243	0.020008	1.12294
79	(85)	(36, 86)	1.000000	0.814847	0.814847	0.814847	1.000000	0.000000	1.00000
33	(85)	(34, 36)	1.000000	0.812631	0.812631	0.812631	1.000000	0.000000	1.00000
118	(85)	(34, 36, 86)	1.000000	0.812631	0.812631	0.812631	1.000000	0.000000	1.00000

204 rows × 9 columns

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Question - 4 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 [168…

```
df = pd.read_csv("mushroom.dat", delimiter=" ")
df.drop(['Unnamed: 23'], axis=1, inplace=True)

# converting data into list and dropping all NAN
records = [[each for each in row if pd.notna(each)] for row in df.values.tolist()]

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

# converting into vertical database..
# counting occurances
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])])




def remove_min_support_vdf(Database_vdf, Min_Sup_count):
    remove_keys=[]
    for key,val in Database_vdf.items():
        if(len(val) < Min_Sup_count):
            remove_keys.append(key)

    for key in remove_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
```

```
startTtime = time.process_time()
support = 60
support_count = support*len(records)/100

L = remove_min_support_vdf(Database_vdf, support_count)
sot = 1

while(1):
    print("Iteration No: ",sot)
    sot += 1
    c = get_vdf(L, sot)
    L = remove_min_support_vdf(c, support_count)

    if(len(L) == 0):
        break
    else:
        final_vdf = Li

endTime = time.process_time()

print('Time Taken: ', endTime-startTime)
```

```
Iteration No:  1
Iteration No:  2
Iteration No:  3
Iteration No:  4
Iteration No:  5
Time Taken:  11.921875
```

In [ ]:

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

In [169…

```
sequences = {
    1: 'aabacacdckf',
    2: 'adckabackae',
    3: 'efckabadfccb',
    4: 'egckafccback'
}
```

In [170…

```
# generate c0
C0 = {}

for each in sequences.values():
    for x in each:
        C0[x] = C0.get(x, 0) + 1

C0
```

Out[170]:

```
{'a': 12, 'b': 5, 'c': 12, 'd': 3, 'k': 6, 'f': 4, 'e': 3, 'g': 1}
```

In [171…

```
# generate l0

l0 = {}
min_support_count = 2

for key, val in C0.items():
    if(val >= min_support_count):
        l0[key] = val

l0
```

Out[171]:

```
{'a': 12, 'b': 5, 'c': 12, 'd': 3, 'k': 6, 'f': 4, 'e': 3}
```

In [172…

```
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_support_count):
    Li={}
    for key, val in C.items():
        if(val >= min_support_count):
            Li[key] = val

    return Li
```

In [173…
```
sot = 1
final_sequence = {}
Min_Sup_Count = 3
while(1):
    print("Iteration No: ",sot)

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

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

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

In [174…
```
final_sequence
```

Out[174]:
```
{'ab': 3, 'ac': 4, 'ba': 4, 'ck': 6, 'ka': 4, 'fc': 3}
```

In [ ]:

In [ ]:

Question 6 - Implement following extensions to Apriori Algorithm (discussed / to be discussed in the class): Hash based strategy, Partitioning Approach and Sampling strategies. You may refer to online tutorials for a formal pseudocode description.

Hash-based approach

In [185…
```
transactions = [{'1','3','4'},{'2','3','5'},{'1','2','3','5'},{'2','5'}, {'1','2','4','5'}]
transactions
```

Out[185]:
```
[{'1', '3', '4'},
 {'2', '3', '5'},
 {'1', '2', '3', '5'},
 {'2', '5'},
 {'1', '2', '4', '5'}]
```

In [186…

```
hash_transactions = {}
count = 0
for each in transactions:
    count += 1
    hash_transactions["T"+str(count)] = each

hash_transactions
```

Out[186]:

```
{'T1': {'1', '3', '4'},
 'T2': {'2', '3', '5'},
 'T3': {'1', '2', '3', '5'},
 'T4': {'2', '5'},
 'T5': {'1', '2', '4', '5'}}
```

In [187…

```
C0 = {}

for each in transactions:
    for elem in each:
        C0[elem] = C0.get(elem, 0) + 1
C0
```

Out[187]:

```
{'4': 2, '1': 3, '3': 3, '2': 4, '5': 4}
```

In [188…

```
order={}
count=0
for key in sorted(C0.keys()):
    count+=1
    order[key]=count

order
```

Out[188]:

```
{'1': 1, '2': 2, '3': 3, '4': 4, '5': 5}
```

In [189…

```
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)
```

```
{'4': 2, '1': 3, '3': 3, '2': 4, '5': 4}
```

In [190…

```
import itertools

for key,val in hash_transactions.items():
    val = [set(i) for i in itertools.combinations(val, 2)]
    sets=[]
    for i in range(len(val)):
        sets.append(sorted(val[i]))

    hash_transactions[key] = sets

hash_transactions
```

Out[190]:

```
{'T1': [['1', '4'], ['3', '4'], ['1', '3']],
 'T2': [['2', '3'], ['3', '5'], ['2', '5']],
 'T3': [['2', '3'],
  ['1', '3'],
  ['3', '5'],
  ['1', '2'],
  ['2', '5'],
  ['1', '5']],
 'T4': [['2', '5']],
 'T5': [['2', '4'],
  ['1', '4'],
  ['4', '5'],
  ['1', '2'],
  ['2', '5'],
  ['1', '5']]}
```

In [191…

```
# hash table generation..
hashTable = {}

for key,val in hash_transactions.items():
    for x in val:
        tempVal = (order[list(x)[0]]*10 + order[x[1]]) % 7
        if(tempVal in hashTable):
            hashTable[tempVal].append(x)
        else:
            hashTable[tempVal] = [x]

hashTable
```

Out[191]:

```
{0: [['1', '4'], ['3', '5'], ['3', '5'], ['1', '4']],
 6: [['3', '4'], ['1', '3'], ['1', '3']],
 2: [['2', '3'], ['2', '3']],
 4: [['2', '5'], ['2', '5'], ['2', '5'], ['2', '5']],
 5: [['1', '2'], ['1', '2']],
 1: [['1', '5'], ['1', '5']],
 3: [['2', '4'], ['4', '5']]}
```

In [192…

```
# generating C2
C2 = {}

keys = sorted(Li.keys())
n = len(keys)
for i in range(n):
    for j in range(i+1, n):
        newKey = frozenset(set(keys[i]).union(set(keys[j])))
        newVal = (order[keys[i]]*10 + order[keys[j]]) % 7
        C2[newKey] = len(hashTable[newVal])

C2
```

Out[192]:

```
{frozenset({'1', '2'}): 2,
 frozenset({'1', '3'}): 3,
 frozenset({'1', '4'}): 4,
 frozenset({'1', '5'}): 2,
 frozenset({'2', '3'}): 2,
 frozenset({'2', '4'}): 2,
 frozenset({'2', '5'}): 4,
 frozenset({'3', '4'}): 3,
 frozenset({'3', '5'}): 4,
 frozenset({'4', '5'}): 2}
```

In [193…

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

L2
```

Out[193]:

```
{frozenset({'1', '2'}): 2,
 frozenset({'1', '3'}): 3,
 frozenset({'1', '4'}): 4,
 frozenset({'1', '5'}): 2,
 frozenset({'2', '3'}): 2,
 frozenset({'2', '4'}): 2,
 frozenset({'2', '5'}): 4,
 frozenset({'3', '4'}): 3,
 frozenset({'3', '5'}): 4,
 frozenset({'4', '5'}): 2}
```

In [ ]:

In [ ]:

In [ ]:

Question - 7

In [194…

```
import numpy as np
import itertools
import copy

def subset_generator(S,n):
    a = itertools.combinations(S,n)
    result = []
    for i in a:
        result.append(set(i))

    return result

def superset_generator(S,unique_itemset):
    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 subset_checker(S,FIS):
    subset = subset_generator(S,len(S)-1)
    flag = 1
    temp = []

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

    FIS = temp
    for i in subset:
        if(i not in FIS):
            flag=0
            break

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

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

    return(result)
```

```
database = [['B','B','A'],['B','A','A'],['A','B','B'],['A','A','A']]
unique_itemset =[{1},{2},{3}]

min_supp = 1
M = 2
size = len(database)

suspected_infreq = []
suspected_freq = []
confirmed_infreq = []
confirmed_freq = []

suspected_infreq = [[i,0,0] for i in unique_itemset]
print("Initial suspected_infreq:",suspected_infreq,"\n")

counter = 0
T = []
while(suspected_infreq != [] or suspected_freq != []):
    for i in range(counter,counter+M):
        index = i%size
        T = transaction_to_itemset(database[index])
        print("Transaction :",T)

        for item in suspected_infreq:
            item[2]+=1
            if item[0].issubset(T):
                item[1]+=1
        for item in suspected_freq:
            item[2]+=1
            if item[0].issubset(T):
                item[1]+=1
    test=suspected_infreq
    # add and remove based on min support
    for item in test:
        if(item[1]>=min_supp):
            suspected_freq.append(item)
            suspected_infreq.remove(item)

    for item in copy.copy(suspected_freq):
        if(item[2]==size):
            confirmed_freq.append(item)
            suspected_freq.remove(item)
    for item in copy.copy(suspected_infreq):
        if(item[2]==size):
            confirmed_infreq.append(item)
            suspected_infreq.remove(item)

    FIS = copy.copy(suspected_freq)
    FIS.extend(confirmed_freq)
```

```
    FIS.extend(confirmed_freq)
    for item in FIS:
        S = superset_generator(item[0],unique_itemset)
        for i in S:
            if subset_checker(i,FIS):
                flag=1
                for x in suspected_infreq:
                    if x[0]==i:
                        flag=0
                for x in suspected_freq:
                    if x[0]==i:
                        flag=0
                for x in confirmed_infreq:
                    if x[0]==i:
                        flag=0
                for x in confirmed_freq:
                    if x[0]==i:
                        flag=0
                if flag:
                    suspected_infreq.append([i,0,0])

counter+=M
print("suspected_freq: ",suspected_freq)
print("suspected_infreq: ",suspected_infreq)
print("confirmed_freq: ",confirmed_freq)
print("confirmed_infreq: ",confirmed_infreq,"\n")
```

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

Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
Transaction : {1, 2, 3}
suspected_freq:  []
suspected_infreq:  []
confirmed_freq:  [[{1}, 4, 4], [{3}, 4, 4], [{2}, 4, 4], [{1, 3}, 4, 4], [{2, 3}, 4, 4], [{1, 2}, 4, 4], [{1, 2, 3}, 4, 4]]
confirmed_infreq:  []
```

In [ ]: