1. Test Drive ACLOSE algorithm to mine closed frequent patterns on a sample dataset of your choice. Test the same on a FIMI benchmark dataset which you have used for Apriori/FP-growth implementations.

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

In [2]:
```
data = [['Apple', 'Beer', 'Rice', 'Chicken'],
          ['Apple', 'Beer', 'Rice'],
          ['Apple', 'Beer'],
          ['Apple', 'Bananas'],
          ['Milk', 'Beer', 'Rice', 'Chicken'],
          ['Milk', 'Beer', 'Rice'],
          ['Milk', 'Beer'],
          ['Apple', 'Bananas']]
```

In [4]:
```
def pruneItemsets(itemsets, transactions, min_support):
    item_counts = {}
    for transaction in transactions:
        for itemset in itemsets:
            if itemset.issubset(transaction):
                item_counts[itemset] = item_counts.get(itemset, 0) + 1

        num_transactions = float(len(transactions))

        pruned_item_counts = {}
        for itemset, count in item_counts.items():
            support = count / num_transactions
            if support >= min_support:
                pruned_item_counts[itemset] = count

    return pruned_item_counts


def isClosed(itemset, all_frequent_itemsets, support_lookup):
    for frequent_itemset in all_frequent_itemsets:
        if(itemset != frequent_itemset and set(itemset).issubset(set(frequent_itemset))):
            if(support_lookup[itemset] == support_lookup[frequent_itemset]):
                return False

    return True
```

In [5]:
```
def ACLOSE(transactions, min_support):
    all_frequent_itemsets = []
    closed_itemsets = []
    support_lookup = {}

    k = 1
    while True:
        candidate_itemsets = set()
        if(k == 1):
            for transaction in transactions:
                for item in transaction:
                    candidate_itemsets.add(frozenset([item]))

        else:
            for itemset1 in frequent_itemsets:
                for itemset2 in frequent_itemsets:
                    union = itemset1.union(itemset2)
                    if(len(union) == k and union not in candidate_itemsets):
                        candidate_itemsets.add(union)

        item_counts = pruneItemsets(candidate_itemsets, transactions, min_support)
        frequent_itemsets = list(item_counts.keys())
        if(len(frequent_itemsets) == 0):
            break

        all_frequent_itemsets.extend(frequent_itemsets)
        support_lookup = support_lookup | item_counts
        if(k > 2):
            for itemset in frequent_itemsets:
                if(isClosed(itemset, all_frequent_itemsets, support_lookup)):
                    closed_itemsets.append(itemset)
        k += 1

    return closed_itemsets, support_lookup


min_support = 0.2
closed_itemsets, support_lookup = ACLOSE(data, min_support)
closed_itemsets
```

Out[5]:
```
[frozenset({'Apple', 'Beer', 'Rice'}),
 frozenset({'Beer', 'Chicken', 'Rice'}),
 frozenset({'Beer', 'Milk', 'Rice'})]
```

Testing on FIMI dataset

In [6]:
```
df = pd.read_csv("mushroom.dat", delimiter=" ")
df.drop(['Unnamed: 23'], axis=1, inplace=True)
df
```

Out[6]:

	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 [16]:
```
df_list = df.values.tolist()

min_support = 0.6
closed_itemsets, support_lookup = ACLOSE(df_list, min_support)
closed_itemsets
```

Out[16]:
```
[frozenset({34, 36, 90}),
 frozenset({59, 85, 86}),
 frozenset({34, 86, 90}),
 frozenset({34, 39, 85}),
 frozenset({36, 86, 90}),
 frozenset({34, 36, 85}),
 frozenset({36, 85, 86}),
 frozenset({34, 39, 86}),
 frozenset({34, 85, 86}),
 frozenset({36, 85, 90}),
 frozenset({34, 85, 90}),
 frozenset({34, 59, 86}),
 frozenset({85, 86, 90}),
 frozenset({34, 59, 85}),
 frozenset({34, 36, 86}),
 frozenset({39, 85, 86}),
 frozenset({39, 85, 90}),
 frozenset({34, 36, 85, 86}),
 frozenset({34, 59, 85, 86}),
 frozenset({36, 85, 86, 90}),
 frozenset({34, 36, 86, 90}),
 frozenset({34, 36, 85, 90}),
 frozenset({34, 39, 85, 86}),
 frozenset({34, 85, 86, 90}),
 frozenset({34, 36, 85, 86, 90})]
```

In [ ]:

2. Test Drive Pincer search to mine maximal frequent patterns on a sample dataset of your choice. Test the same on a FIMI benchmark dataset which you have used for Apriori/FP-growth implementations.

In [18]:

```python
from itertools import combinations

def mfcs_prune(old_ckplus1,curr_mfcs):
    new_ckplus1=[]
    for c in old_ckplus1:
        for itemset in curr_mfcs:
            if set(c).issubset(set(itemset)):
                new_ckplus1.append(c)

    return new_ckplus1

def mfs_prune(old_ck,curr_mfs):
    new_ck=old_ck.copy()
    for c in old_ck.copy():
        for itemset in curr_mfs:
            if set(c).issubset(set(itemset)):
                new_ck.remove(c)

    return new_ck

def mfcs_gen(sk,mfcs):
    mfcs = mfcs.copy()
    for infrequent_itemset in sk:
        for mfcs_itemset in mfcs.copy():
            # If infrequent itemset is a subset of mfcs itemset
            if all(_item in mfcs_itemset for _item in infrequent_itemset):
                mfcs.remove(mfcs_itemset)

                for item in infrequent_itemset:
                    updated_mfcs_itemset = mfcs_itemset.copy()
                    updated_mfcs_itemset.remove(item)

                    if not any(all(item in _mfcs_itemset for item in updated_mfcs_itemset) for _mfcs_itemset in mfcs):
                        mfcs.append(updated_mfcs_itemset)

    return mfcs
```

In [19]:

```
def gen_next_ck(ck,k):
    num_freq=len(ck)
    newck=[]
    for i in range(num_freq):
        j=i+1
        while((j<num_freq) and (ck[i][:k-1]==ck[j][:k-1])):
            new_itemset=ck[i][:k-1]+[ck[i][k-1]]+[ck[j][k-1]]
            insert_in_new=False

            if k==1:
                insert_in_new=True
            elif k==2 and (new_itemset[-2:] in ck):
                insert_in_new=True
            else:
                for a in combinations(ck[:-2],k-2):
                    if(list(a)+ck[-2:] not in ck):
                        insert_in_new=False

            if insert_in_new:
                newck.append(new_itemset)
            j+=1

    return newck
```

In [21]:

```
def pincerSearch(txn,min_supp):
    items = set()
    for transaction in txn:
        items.update(transaction)

    level_k=1
    cand_freq_itemsets=[[item] for item in items]
    level_freq_itemsets=[]
    level_infreq_itemsets=[]
    mfcs=[items.copy()]
    mfs=[]

    print(f"MFCS={mfcs}\n")
    print(f"MFS={mfs}\n")

    while len(cand_freq_itemsets)!=0:
        print(f"Level {level_k}")
        print(f"C{level_k} = {cand_freq_itemsets}")

        cand_freq_itemsets_cnt=[0]*len(cand_freq_itemsets)
        mfcs_itemsets_cnt=[0]*len(mfcs)

        # step 1- read txn from db and get support for ck and mfcs
        for each in txn:
            for i,itemset in enumerate(cand_freq_itemsets):
                if set(itemset).issubset(each):
                    cand_freq_itemsets_cnt[i]+=1

            for i,itemset in enumerate(mfcs):
                if set(itemset).issubset(each):
                    mfcs_itemsets_cnt[i]+=1


        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
            print(f"{itemset} - {supp}")

        print('\n')
        for itemset,supp in zip(mfcs,mfcs_itemsets_cnt):
            print(f"{itemset} - {supp}")

        print('\n')

        # step 2 - add freq itemsets from mfcs to mfs
        for itemset,supp in zip(mfcs,mfcs_itemsets_cnt):
            if (itemset not in mfs) and supp>=min_supp:
                mfs.append(itemset)

        print(f"MFS - {mfs}")
        level_freq_itemsets=[]
        level_infreq_itemsets=[]
```

```
        # step 3 - infreq itemsets in ck makes sk
        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
            if supp>=min_supp:
                level_freq_itemsets.append(itemset)

            if supp<min_supp:
                level_infreq_itemsets.append(itemset)

        print(f"L{level_k} - {level_freq_itemsets}")
        print(f"S{level_k} - {level_infreq_itemsets}")

        # step 4 - mfcs-gen if sk is non empty
        mfcs=mfcs_gen(level_infreq_itemsets,mfcs)
        print(f"MFCS - {mfcs}")

        # step 5 - pruning cand using mfs
        print(f"C{level_k} was - {level_freq_itemsets}")
        level_freq_itemsets=mfs_prune(level_freq_itemsets,mfs)
        print(f"After pruning,L{level_k} - {level_freq_itemsets}")

        # step 6 - gen next ck from old ck
        cand_freq_itemsets=gen_next_ck(cand_freq_itemsets,level_k)

        # step 7 - prune new ck with mfcs
        cand_freq_itemsets=mfcs_prune(cand_freq_itemsets,mfcs)
        level_k+=1

    return mfs


transactions = [
        {1, 5, 6, 8},
        {2, 4, 8},
        {4, 5, 7},
        {2, 3},
        {5, 6, 7},
        {2, 3, 4},
        {2, 6, 7, 9},
        {5},
        {8},
        {3, 5, 7},
        {3, 5, 7},
        {5, 6, 8},
        {2, 4, 6, 7},
        {1, 3, 5, 7},
        {2, 3, 9} ]

min_support_count = 4
MFS = pincerSearch(transactions, min_support_count)
print("MFS = {}".format(MFS))
```

```
print('MFS = {}'.format(MFS))
```

```
MFCS=[{1, 2, 3, 4, 5, 6, 7, 8, 9}]

MFS=[]

Level 1
C1 = [[1], [2], [3], [4], [5], [6], [7], [8], [9]]
[1] - 2
[2] - 6
[3] - 6
[4] - 4
[5] - 8
[6] - 5
[7] - 7
[8] - 4
[9] - 2


{1, 2, 3, 4, 5, 6, 7, 8, 9} - 0


MFS - []
L1 - [[2], [3], [4], [5], [6], [7], [8]]
S1 - [[1], [9]]
MFCS - [{2, 3, 4, 5, 6, 7, 8}]
C1 was - [[2], [3], [4], [5], [6], [7], [8]]
After pruning,L1 - [[2], [3], [4], [5], [6], [7], [8]]
Level 2
C2 = [[2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [4, 5], [4, 6], [4, 7], [4, 8], [5, 6], [5, 7], [5, 8],
[6, 7], [6, 8], [7, 8]]
[2, 3] - 3
[2, 4] - 3
[2, 5] - 0
[2, 6] - 2
[2, 7] - 2
[2, 8] - 1
[3, 4] - 1
[3, 5] - 3
[3, 6] - 0
[3, 7] - 3
[3, 8] - 0
[4, 5] - 1
[4, 6] - 1
[4, 7] - 2
[4, 8] - 1
[5, 6] - 3
[5, 7] - 5
[5, 8] - 2
[6, 7] - 3
[6, 8] - 2
```

```
[7, 8] - 0


{2, 3, 4, 5, 6, 7, 8} - 0


MFS - []
L2 - [[5, 7]]
S2 - [[2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [4, 5], [4, 6], [4, 7], [4, 8], [5, 6], [5, 8], [6, 7],
[6, 8], [7, 8]]
MFCS - [{2}, {3}, {4}, {5, 7}, {6}, {8}]
C2 was - [[5, 7]]
After pruning,L2 - [[5, 7]]
MFS = []
```

In [ ]:

In [ ]: