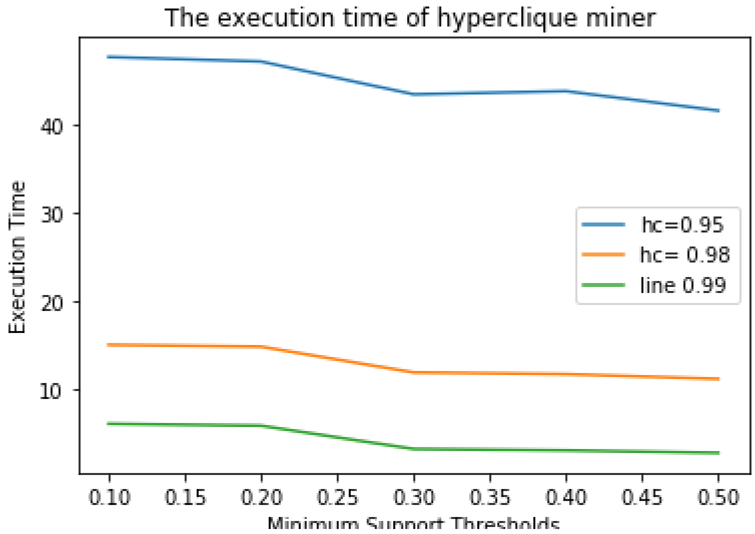
```
#This block of code takes the dataset and  apply transaction encoding on it

import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
import numpy as np
from mlxtend.frequent_patterns import apriori


ds=pd.read_csv("pumsb_sample1.csv")
ds=ds.values.tolist()
te = TransactionEncoder()
te_ary = te.fit(ds).transform(ds)
df = pd.DataFrame(te_ary, columns=te.columns_)
#print(df)

###############################################


def calc_sup(item):
    count =0
    for row in range(0,len(df)):
        l=len(item)
        c=0
        for i in range(0,l):
            if df.get_value(row,item[i])==True:
                c=c+1
        if c==l:
            count=count+1

    return(count/len(df))


def calc_hc(item):

    subset=list(itertools.combinations(item,1))
    l=[]

    for i in range(len(subset)):
        temp=list(subset[i])
        l.append(calc_sup(temp))



        maximum=max(l)
    return(calc_sup(item)/maximum)
        #hc_dict.update({item:sup_dict[item]/maximum})
```

```
###Execution time plot

#Step 2 ---> Iteration over i=2 to k-1
#inside the iteration all the pruning functions are called and final result is printed by this funct

import time



def myfunc(min_sup,hc):
  start_time=time.time()
  ck=[]
  count=0
  for i in list(df.columns):
    col=df.loc[:,i]
    col=list(col)
    support_count=0
    for item in col:
        if item==True:
            support_count+=1


    support=support_count/len(df)
    if support >= min_sup :
        x=[]
        x.append(i)
        ck.append(x)



  ck=list(map(frozenset,ck))
  print(ck)
  count+=len(ck)

#################################################




  k=len(df.columns)

  Lk=ck   # ck from previous step 1



  for i in range(2,k):

    print(i)
    CK1=aprioriGen(Lk,i-1)   #i-1

    ck1=CK1


    ck1=antimonotone(Lk,ck1,i-1) #i-1

    ck1=cross_support(ds,ck1,hc)



    #code for step 4 here
    ck_updated=[]
    for item in ck1:
      dt=list(map(int,item))

      if(calc_sup(dt)>min_sup):
        ck_updated.append(item)


    ck_updated1=[]

    for item in ck_updated:
      dt=list(map(int,item))
      if(calc_hc(dt)>hc):
        ck_updated1.append(item)

    print(set(ck_updated1))
    count+=len(ck_updated1)
    if len(ck_updated1)==0:
      print("=================================")
      break
    else:
      Lk=ck_updated1
  end_time=time.time()
  return end_time-start_time
#code to check if ck1 is empty if not the Lk=ck1
```

```
#myfunc(0.5,0.99)

ms=[0.1,0.2,0.3,0.4,0.5]
hct=[0.95,0.98,0.99]
y1=[]
y2=[]
y3=[]

for i in ms:
  y1.append(myfunc(i,0.95))
  y2.append(myfunc(i,0.98))
  y3.append(myfunc(i,0.99))


import matplotlib.pyplot as plt

x=ms
plt.plot(x, y1, label = "hc=0.95")
plt.plot(x, y2, label = "hc= 0.98")
plt.plot(x, y3, label = "line 0.99")
plt.xlabel('Minimum Support Thresholds')
plt.ylabel('Execution Time')
plt.title('The execution time of hyperclique miner ')

plt.legend()

plt.show()
```

```
set()
================================
[frozenset({14}), frozenset({15}), frozenset({17}), frozenset({66}), frozenset({84}), fr
2
{frozenset({4440, 4414}), frozenset({4527, 4727}), frozenset({4432, 188}), frozenset({68
3
{frozenset({4426, 180, 4428}), frozenset({4426, 188, 4428}), frozenset({4680, 4780, 4518
4
set()
================================
[frozenset({14}), frozenset({15}), frozenset({17}), frozenset({66}), frozenset({84}), fr
2
{frozenset({4440, 4414}), frozenset({4527, 4727}), frozenset({6856, 5946}), frozenset({1
3
{frozenset({4785, 4527, 4727}), frozenset({4785, 4627, 4727}), frozenset({6856, 4953, 59
4
set()
================================
[frozenset({15}), frozenset({17}), frozenset({66}), frozenset({84}), frozenset({111}), f
2
{frozenset({4527, 4727}), frozenset({4432, 188}), frozenset({4434, 7092}), frozenset({17
3
{frozenset({4426, 188, 7062}), frozenset({188, 180, 4428}), frozenset({17, 3404, 4404}),
4
set()
================================
[frozenset({15}), frozenset({17}), frozenset({66}), frozenset({84}), frozenset({111}), f
2
{frozenset({4440, 4414}), frozenset({4527, 4727}), frozenset({4432, 188}), frozenset({17
3
{frozenset({4426, 180, 4428}), frozenset({4426, 188, 4428}), frozenset({170, 188, 4426})
4
set()
================================
[frozenset({15}), frozenset({17}), frozenset({66}), frozenset({84}), frozenset({111}), f
2
{frozenset({4440, 4414}), frozenset({161, 84}), frozenset({188, 4438}), frozenset({4527,
3
{frozenset({4785, 4627, 4727}), frozenset({4527, 4627, 4727}), frozenset({4785, 4627, 45
4
set()
================================
```



Minimum Support Thresholds

```
# Apriori Gen function

def aprioriGen(Lk, k):
    ck1=[]

    for i in range(len(Lk)):
        for j in range(i+1, len(Lk)):
            L1 = list(Lk[i])
            L1=L1[0:k-1]
            L2 = list(Lk[j])
            L2=L2[0:k-1]
            L1.sort()
            L2.sort()
            if L1==L2:
                ck1.append(Lk[i] | Lk[j])
    return ck1



#Anti Monotone function

import itertools
def antimonotone(prev_ck,current_ck,k):

    ck_updated=[]
    for item in current_ck:    #ck
        subset=list(itertools.combinations(item, k))
        subset=list(map(frozenset,subset))
        count=0
        L=len(subset)
        for item1 in subset:
            for item2 in prev_ck:
                if item1==item2:
                    count=count+1

        #print(L)
        #print(count)
        if L == count:
            ck_updated.append(item)
            #print(item)

    ck_updated=list(map(frozenset,ck_updated))
    return ck_updated
```

```
import itertools
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
import numpy as np

support_dict={}
def cross_support(CK1,ck,hc):
    te = TransactionEncoder()
    te_ary = te.fit(CK1).transform(CK1)
    df = pd.DataFrame(te_ary, columns=te.columns_)
   # print(df)
   # print(len(df))


    for i in list(df.columns):
      col=df.loc[:,i]
      col=list(col)
      support_count=0
      for item in col:

        if item==True:
          support_count+=1

      support_dict.update({i:support_count/len(df)})

    ck=list(map(list,ck))
    ck_updated=[]
    #print(support_dict)
    for item in ck:
        subset=list(itertools.combinations(item, 2))

        for i in range(0,len(subset)):
            temp=subset[i]
            #print(2*support_dict[subset[i][0]])
            #print(support_dict[subset[i][1]])
            flag=0
            if support_dict[subset[i][0]]<(support_dict[subset[i][1]]*hc):
                #print((item))
                #ck.remove(item)
                flag=1
            if support_dict[subset[i][1]]<(support_dict[subset[i][0]]*hc):
                #print((item))
                #ck.remove(item)
                flag=1

            if flag!=1:
                ck_updated.append(item)

    ck_updated=list(map(frozenset,ck_updated))
```