```
#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.csv")

L=[0.80,0.85,0.90,0.95]
attr=[250,500,1000,1500]

y1=[]
y2=[]
y3=[]
y4=[]


for attribute in attr:
  ds1=ds[:attribute]
  ds1=ds1.values.tolist()
  te = TransactionEncoder()
  te_ary = te.fit(ds1).transform(ds1)
  df1 = pd.DataFrame(te_ary, columns=te.columns_)
  y1.append(myfunc(ds1,df1,0.01,0.80))

for attribute in attr:
  ds2=ds[:attribute]
  ds2=ds2.values.tolist()
  te = TransactionEncoder()
  te_ary = te.fit(ds2).transform(ds2)
  df2 = pd.DataFrame(te_ary, columns=te.columns_)
  y2.append(myfunc(ds2,df2,0.01,0.85))


for attribute in attr:
  ds3=ds[:attribute]
  ds3=ds3.values.tolist()
  te = TransactionEncoder()
  te_ary = te.fit(ds3).transform(ds3)
  df3 = pd.DataFrame(te_ary, columns=te.columns_)
  y3.append(myfunc(ds3,df3,0.01,0.90))

for attribute in attr:
  ds4=ds[:attribute]
  ds4=ds4.values.tolist()
  te = TransactionEncoder()
  te_ary = te.fit(ds4).transform(ds4)
  df4 = pd.DataFrame(te_ary, columns=te.columns_)
  y4.append(myfunc(ds4,df4,0.01,0.95))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:22: FutureWarning: get_va
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
4
================================
2
3
```

```
3
4
================================
2
3
4
================================
```

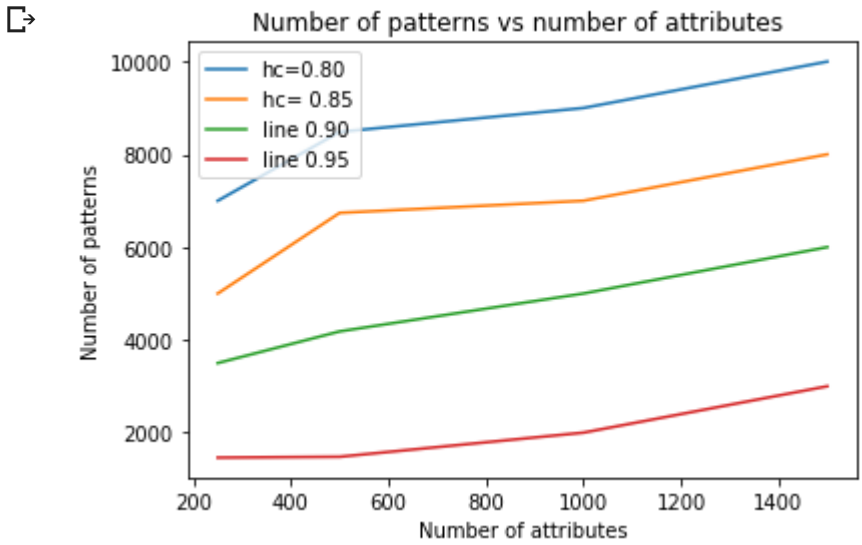
```
import matplotlib.pyplot as plt

x=attr
```

```
plt.plot(x, y1, label = "hc=0.80")
plt.plot(x, y2, label = "hc= 0.85")
plt.plot(x, y3, label = "line 0.90")
plt.plot(x, y4, label = "line 0.95")
plt.xlabel('Number of attributes')
plt.ylabel('Number of patterns')
plt.title('Number of patterns vs number of attributes ')

plt.legend()

plt.show()
```




```
def myfunc(ds,df,min_sup,hc):
  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:
        #print((item))
        dt=list(map(int,item))
        #print(dt)
        #print(calc_sup(item[0]))
        if(calc_sup(dt,df)>min_sup):
          ck_updated.append(item)


      ck_updated1=[]

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

     # print(set(ck_updated1))
      count+=len(ck_updated1)
      if len(ck_updated1)==0:
        print("================================")
        break
      else:
        Lk=ck_updated1
    return count
```

```
def calc_hc(item,df):

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

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



        maximum=max(l)
    return(calc_sup(item,df)/maximum)


def calc_sup(item,df):
    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))







# 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))
    #print(ck_updated)
    return ck_updated
```