Dataset:

cream cheese,detergent,newspapers,processed cheese,tropical fruit
bathroom cleaner,candy,flour,frozen dessert,other vegetables,rolls/buns,root vegetables,salty snack,sweet spreads,tropical fruit,waffles
bottled water,canned beer
yogurt
chocolate,rolls/buns,sausage,soda
other vegetables
brown bread,canned beer,fruit/vegetable juice,newspapers,shopping bags,soda
beverages,bottled water,specialty bar,yogurt
bottled water,hamburger meat,hygiene articles,napkins,other vegetables,rolls/buns,spices
beverages,other vegetables,root vegetables,sugar,whole milk
abrasive cleaner,artif. sweetener,berries,other vegetables,pork,soda,whipped/sour cream,whole milk
beef,detergent,grapes
pastry,soda
fruit/vegetable juice
canned beer
dessert,other vegetables,root vegetables,whole milk
citrus fruit,newspapers,zwieback
canned beer,rolls/buns,sausage,shopping bags,soda,specialty bar
brown bread,candy,cereals,coffee,domestic eggs,pastry,root vegetables,soda,sugar,tropical fruit,waffles,whole milk,yogurt
berries,yogurt
canned beer
bottled water,butter milk,cream cheese,newspapers,rolls/buns,soda,spread
cheese,yogurt
coffee
bottled water,pastry
rolls/buns
misc. beverages
UHT-milk,bottled water,butter,curd,hard cheese,long life bakery product,other vegetables,rolls/buns,root vegetables,whipped/sour cream
cat food,newspapers,rolls/buns,sausage
canned beer
grapes,ham,other vegetables,whole milk
baking powder,brown bread,curd,domestic eggs,fruit/vegetable juice,margarine,other vegetables,semi-finished bread,tropical fruit,turkey

pickled vegetables,processed cheese,soda,whole milk,yogurt
curd,pastry,whole milk,yogurt
brown bread,canned beer,packaged fruit/vegetables
bottled water,chewing gum,chocolate marshmallow,hygiene
articles,napkins,oil,rolls/buns
beef,cat food,ham,ice cream,rolls/buns,whipped/sour cream
pastry,rolls/buns,sugar
canned fish,detergent,frozen vegetables,other vegetables,salty snack,seasonal
products,whole milk
pastry,sausage
beef,sausage,whole milk

Code:

```
import csv
import itertools

DataFile = open('groceries2.csv', 'r')
minsup = 0.02
f2 = "Rules.txt"
f1 = "FItems.txt"
minconf = 0.45


def L1():

  DataCaptured = csv.reader(DataFile, delimiter=',')
  data = list(DataCaptured)
  for e in data:
     e = sorted(e)
  count = {}
  for items in data:
     for item in items:
        if item not in count:
           count[(item)] = 1
        else:
           count[(item)] = count[(item)] + 1
```

```
    count2 = {k: v for k, v in count.items() if v >= minsup*9835}
    return count2, data


def generateCk(Lk_1, flag, data):
    Ck = []

    if flag == 1:
        flag = 0
        for item1 in Lk_1:
            for item2 in Lk_1:
                if item2 > item1:
                    Ck.append((item1, item2))



    else:
        for item in Lk_1:
            k = len(item)
        for item1 in Lk_1:
            for item2 in Lk_1:
                if (item1[:-1] == item2[:-1]) and (item1[-1] != item2[-1]):
                    if item1[-1] > item2[-1]:
                        Ck.append(item2 + (item1[-1],))
                    else:
                        Ck.append(item1 + (item2[-1],))
        # print("C" + str(k+1) + ": ", Ck[1:3])
        # print()
    L = generateLk(set(Ck), data)
    return L, flag


def generateLk(Ck, data):
    count = {}
    for itemset in Ck:
        #print(itemset)
        for transaction in data:
            if all(e in transaction for e in itemset):
                if itemset not in count:
                    count[itemset] = 1
```

```
            else:
                count[itemset] = count[itemset] + 1



    count2 = {k: v for k, v in count.items() if v >= minsup*9835}

    return count2


def rulegenerator(fitems):
    counter = 0
    print("Rules:-")
    for itemset in fitems.keys():
        if isinstance(itemset, str):
            continue
        length = len(itemset)

        union_support = fitems[tuple(itemset)]
        for i in range(1, length):

            lefts = map(list, itertools.combinations(itemset, i))
            for left in lefts:
                if len(left) == 1:
                    if ''.join(left) in fitems:
                        leftcount = fitems[''.join(left)]
                        conf = union_support / leftcount
                else:
                    if tuple(left) in fitems:
                        leftcount = fitems[tuple(left)]
                        conf = union_support / leftcount
                if conf >= minconf:
                    fo = open(f2, "a+")
                    right = list(itemset[:])
                    for e in left:
                        right.remove(e)
                    fo.write(str(left) + ' (' + str(leftcount) + ')' + ' -> ' + str(right) + ' (' +
str(fitems[''.join(right)]) + ')' + ' [' + str(conf) + ']' + '\n')
                    print(str(left) + ' -> ' + str(right) + ' (Confidence=' + str(conf) + ')')
                    counter = counter + 1
```

```
                #Greater than 1???
                fo.close()
    print(counter, "rules generated")


def apriori():

    L, data = L1()
    flag = 1
    FreqItems = dict(L)
    while(len(L) != 0):
        fo = open(f1, "a+")
        for k, v in L.items():
            fo.write(str(k) + ' >>> ' + str(v) + '\n\n')
        fo.close()

        L, flag = generateCk(L, flag, data)
        FreqItems.update(L)
    rulegenerator(FreqItems)



if __name__ == '__main__':
    apriori()
```

Output:

```
Rules:-
['domestic eggs'] -> ['whole milk'] (Confidence=0.47275641025641024)
['curd'] -> ['whole milk'] (Confidence=0.4904580152671756)
['butter'] -> ['whole milk'] (Confidence=0.4972477064220184)
['other vegetables', 'root vegetables'] -> ['whole milk']
(Confidence=0.4892703862660944)
['root vegetables', 'whole milk'] -> ['other vegetables']
(Confidence=0.47401247401247404)
['other vegetables', 'yogurt'] -> ['whole milk'] (Confidence=0.5128805620608899)
6 rules generated
```