**CSE-3019**

**DATA MINING**

LAB: L47 +L48

**Name: Nimit Agrawal**

**REG. NO: 16BCE0170**

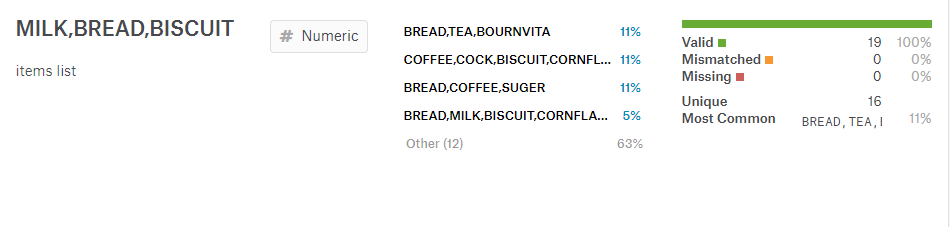
**Lab-4**

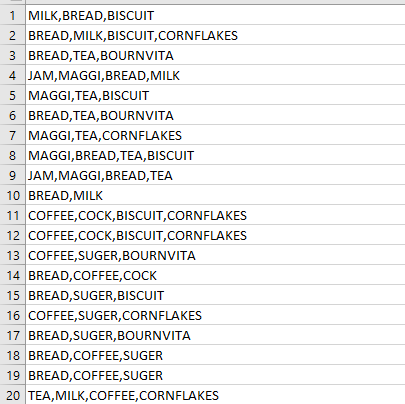
Apply Apriori Algorithm to a dataset and perform market basket analysis with varying support and confidence values.

Output should have the association rule with higher confidence**.**

**Dataset:**

**https://www.kaggle.com/shazadudwadia/supermarket**



**Dataset**

**Apriori Code**:

import sys

import operator

from itertools import chain, combinations

from collections import defaultdict

from optparse import OptionParser

def subsets(arr):

return chain(\*[combinations(arr, i + 1) for i, a in enumerate(arr)])

def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):

\_itemSet = set()

localSet = defaultdict(int)

for item in itemSet:

for transaction in transactionList:

if item.issubset(transaction):

freqSet[item] += 1

localSet[item] += 1

for item, count in localSet.items():

support = float(count)/len(transactionList)

if support >= minSupport:

\_itemSet.add(item)

return \_itemSet

def joinSet(itemSet, length):

return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length])

def getItemSetTransactionList(data\_iterator):

transactionList = list()

itemSet = set()

for record in data\_iterator:

transaction = frozenset(record)

transactionList.append(transaction)

for item in transaction:

itemSet.add(frozenset([item])) # Generate 1-itemSets

return itemSet, transactionList

def runApriori(data\_iter, minSupport, minConfidence):

itemSet, transactionList = getItemSetTransactionList(data\_iter)

freqSet = defaultdict(int)

largeSet = dict()

# Global dictionary which stores (key=n-itemSets,value=support)

# which satisfy minSupport

assocRules = dict()

# Dictionary which stores Association Rules

oneCSet = returnItemsWithMinSupport(itemSet,

transactionList,

minSupport,

freqSet)

currentLSet = oneCSet

k = 2

while(currentLSet != set([])):

largeSet[k-1] = currentLSet

currentLSet = joinSet(currentLSet, k)

currentCSet = returnItemsWithMinSupport(currentLSet,

transactionList,

minSupport,

freqSet)

currentLSet = currentCSet

k = k + 1

def getSupport(item):

"""local function which Returns the support of an item"""

return float(freqSet[item])/len(transactionList)

toRetItems = []

for key, value in largeSet.items():

toRetItems.extend([(tuple(item), getSupport(item))

for item in value])

toRetRules = []

for key, value in list(largeSet.items())[1:]:

for item in value:

\_subsets = map(frozenset, [x for x in subsets(item)])

for element in \_subsets:

remain = item.difference(element)

if len(remain) > 0:

confidence = getSupport(item)/getSupport(element)

if confidence >= minConfidence:

toRetRules.append(((tuple(element), tuple(remain)),

confidence))

return toRetItems, toRetRules

def printResults(items, rules):

for item, support in sorted(items, key=operator.itemgetter(1)):

print("item: %s , %.3f" % (str(item), support))

print("\n------------------------ RULES:")

for rule, confidence in sorted(rules, key=operator.itemgetter(1)):

pre, post = rule

print("Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence))

def dataFromFile(fname):

"""Function which reads from the file and yields a generator"""

file\_iter = open(fname, 'rU')

for line in file\_iter:

line = line.strip().rstrip(',') # Remove trailing comma

record = frozenset(line.split(','))

yield record

if \_\_name\_\_ == "\_\_main\_\_":

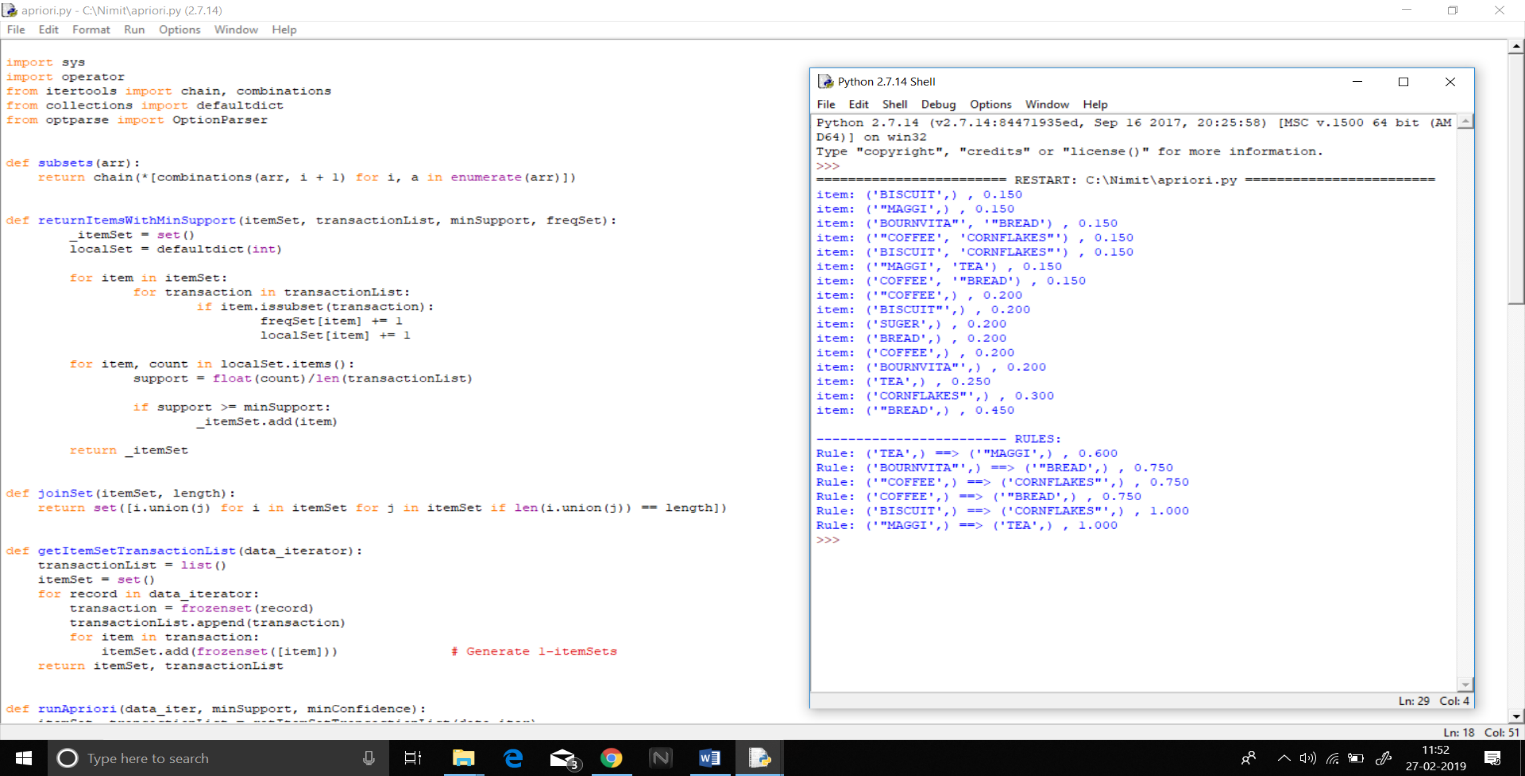
inFile=dataFromFile("GroceryStoreDataSet.csv")

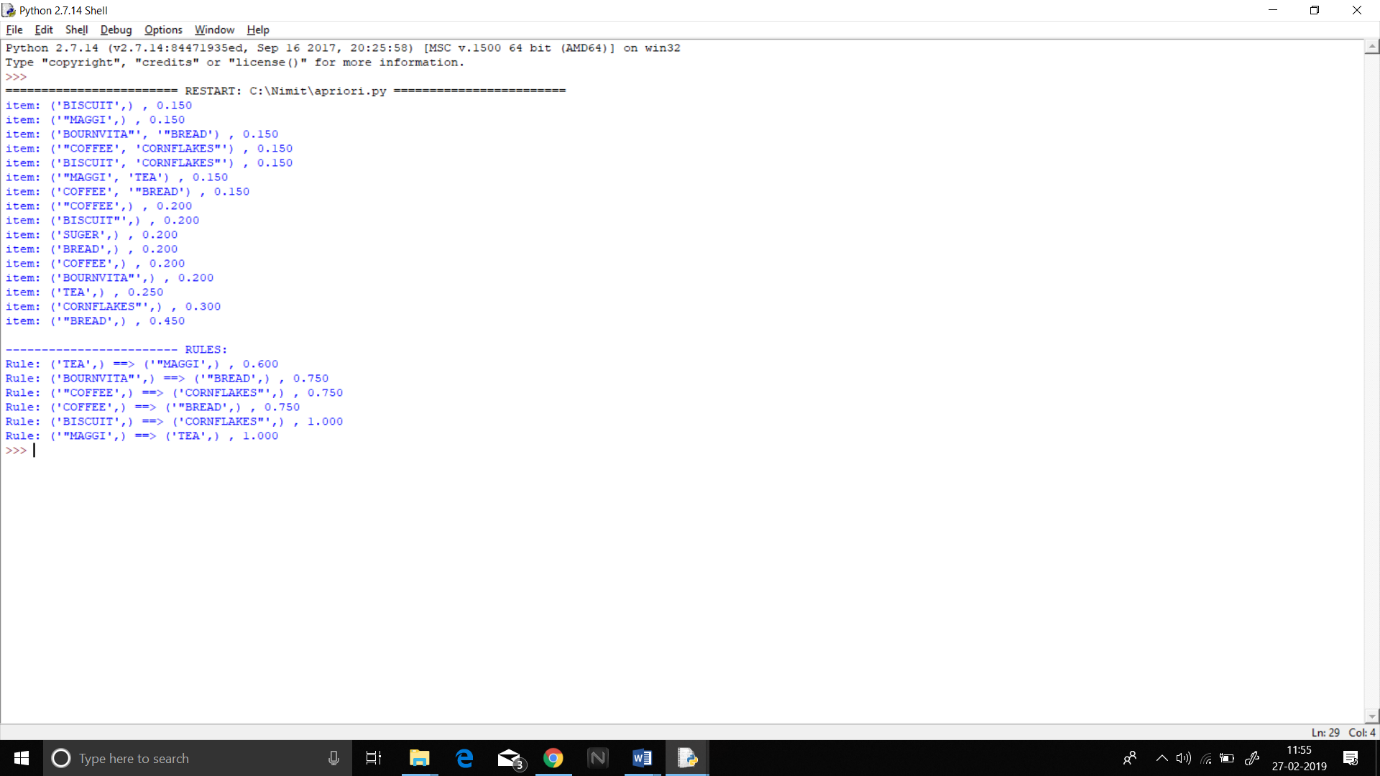
minSupport = 0.15

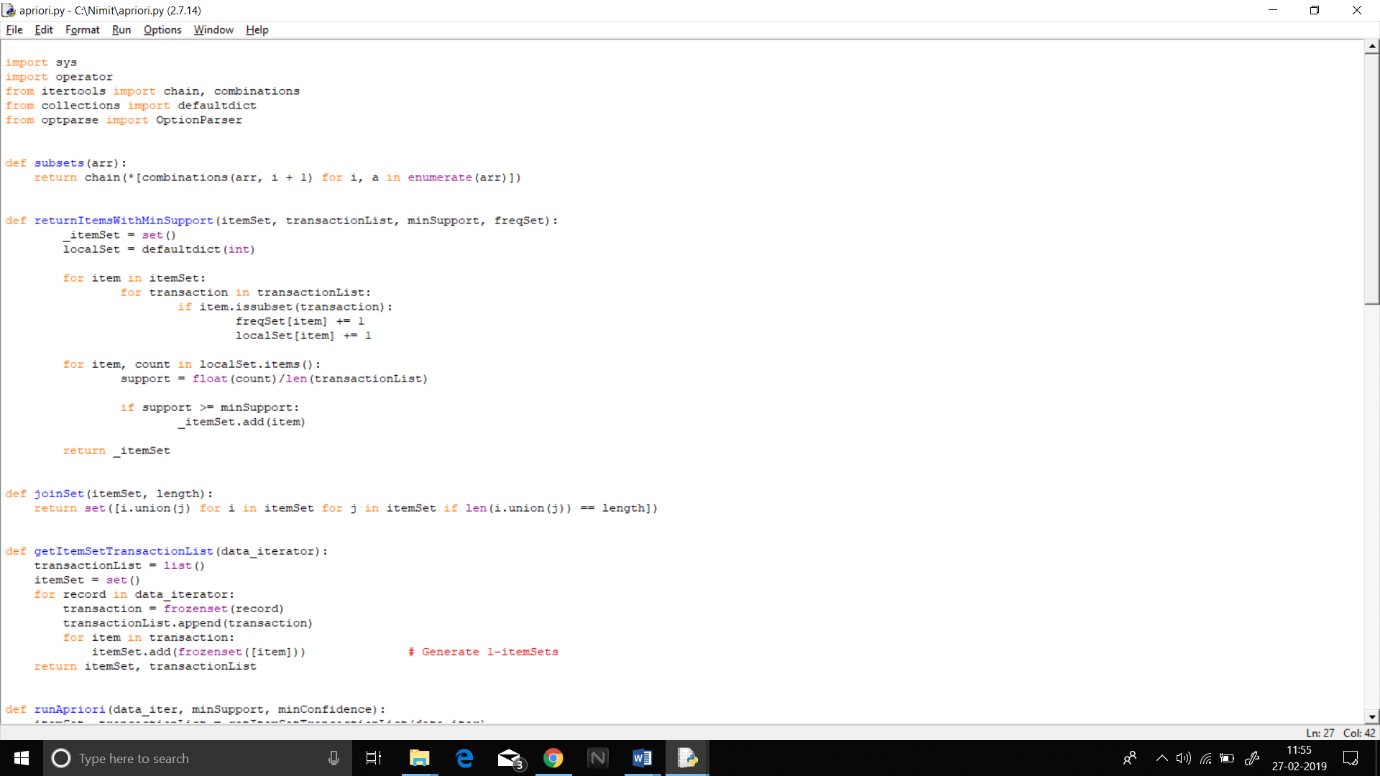
minConfidence = 0.6

items, rules = runApriori(inFile, minSupport, minConfidence)

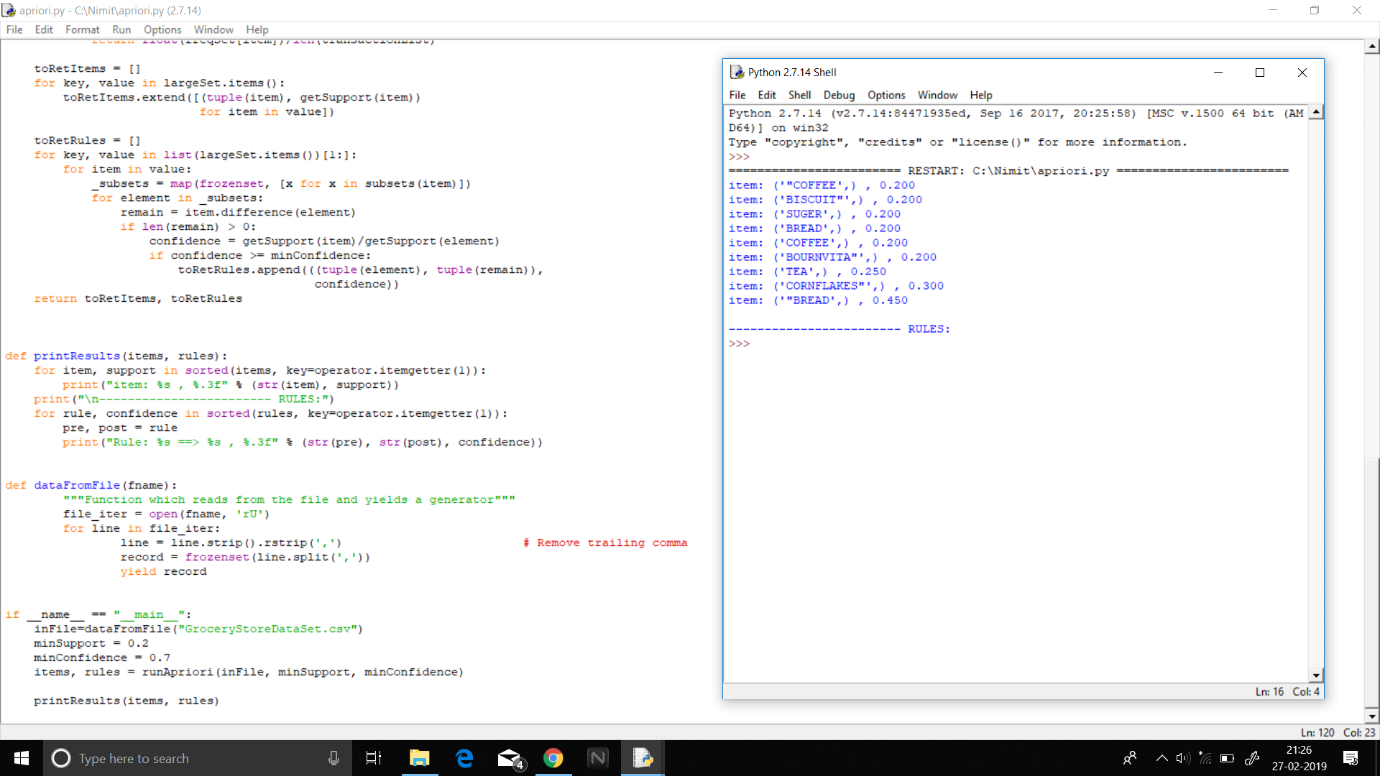
printResults(items, rules)





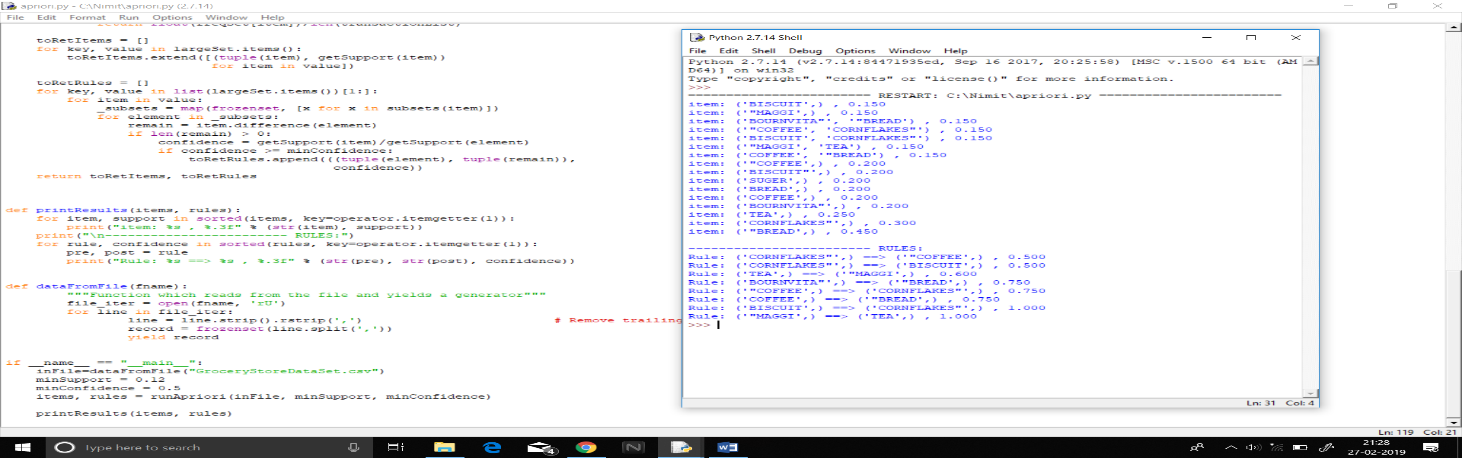


Support:0.2

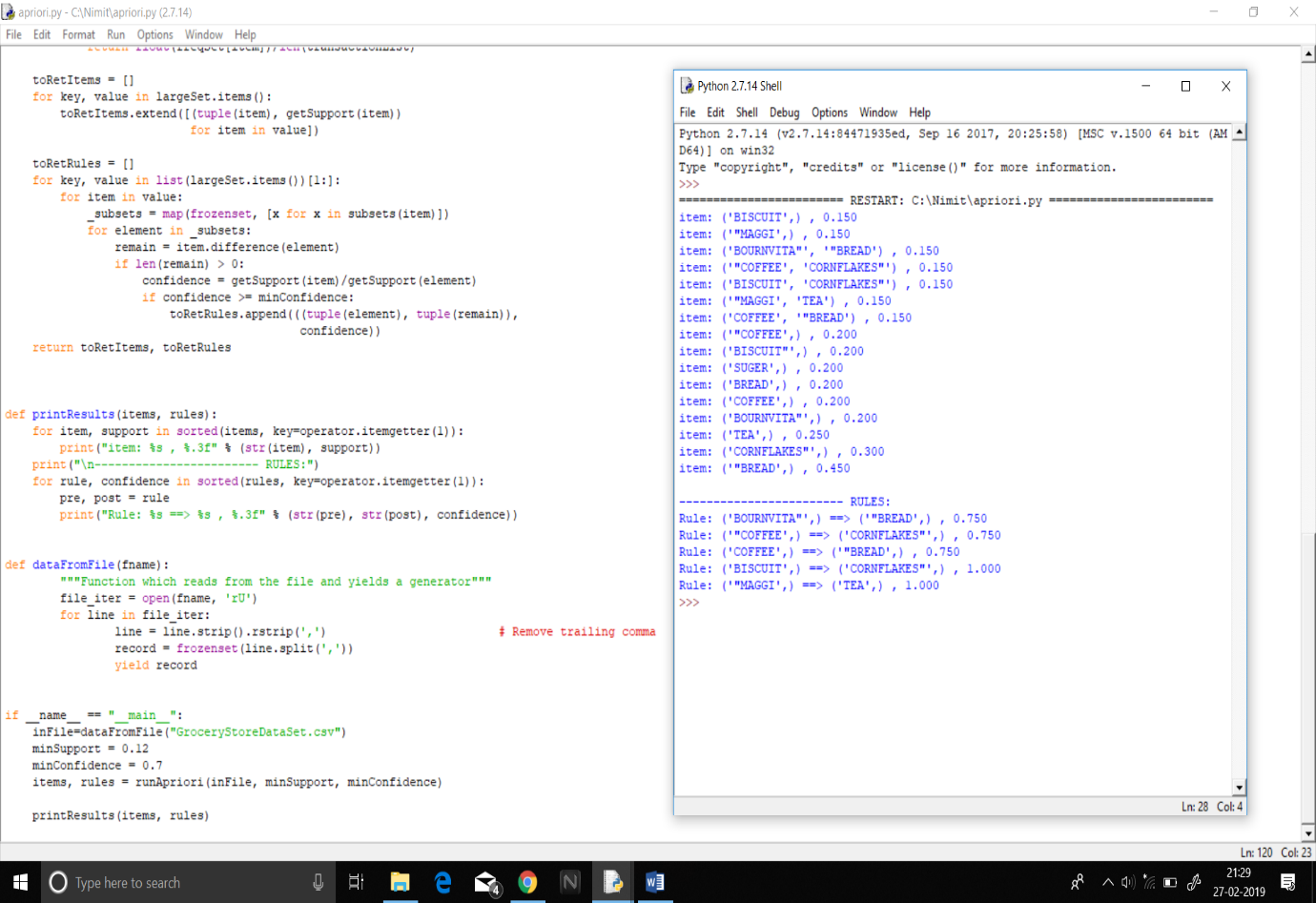
Confidence:0.7

Support: 0.12

Confidence: 0.5



Support:0.12

Confidence:0.7