Name: Mareena Fernandes

TE IT	Roll number : 8669
Expt number : 7	Date of implementation:10/05/2021
Aim : To implement frequent pattern mining algorithm	
Programming language used : Python	
Related Course outcome : CO2 Upon completion of this course students will be able to Implement the appropriate data mining methods like classification, clustering or association mining on large data sets	

Theory : Frequent patterns are patterns (such as itemsets, subsequences or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a frequent sequential pattern. A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructures occurs frequently, it is called a frequent structured pattern. Finding such frequent patterns plays an essential role in mining associations, correlations and many other interesting relationships among data. A typical example of frequent itemset mining is **market basket analysis**. This process analyzes customer buying habits by finding association between different items that customers place in their shopping basket. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. In general, association rule mining can be viewed as a two-step process:

1. Find all frequent itemsets : By definition, each of these itemsets will occur atleast as frquently as a predetermined minimum support count.

2. Generate strong association rules fro the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

An itemset X is closed in a data set S if there exists no proper super-itemset Y such that Y has the same support count as X in S. An itemset X is a closed frequent itemset in set S if X is both closed and frequent in S. An itemset X is a maximal frequent itemset in set S if X is frequent, and there exists no super-itemset Y such that X is subset of Y and Y is frequent in S.

Apriori is a seminal algorithm proposed by R. Agrawal and R.Srikant in 1994 for mining frequent itemsets for boolean association rule. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties.

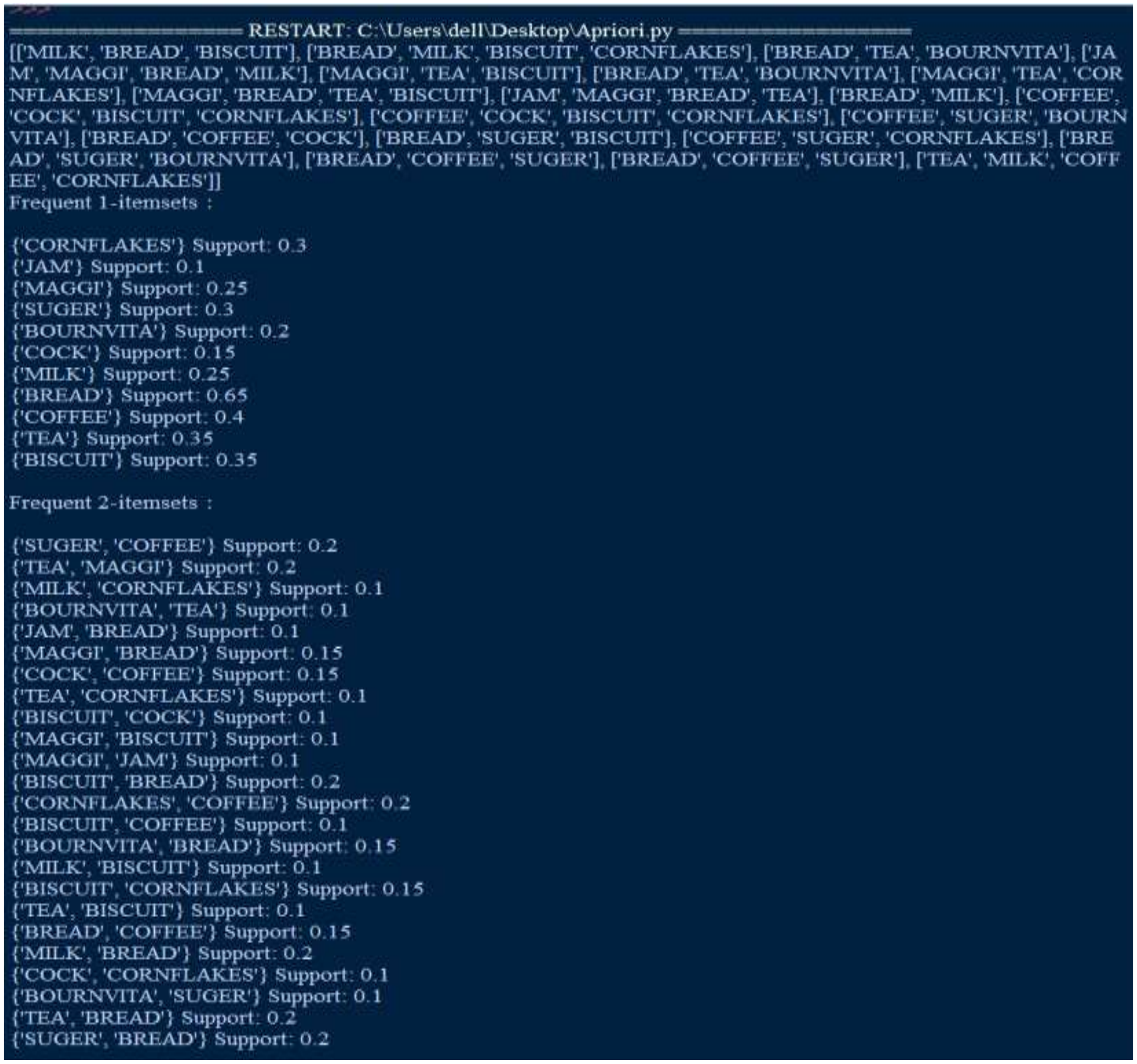
Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*

Code:

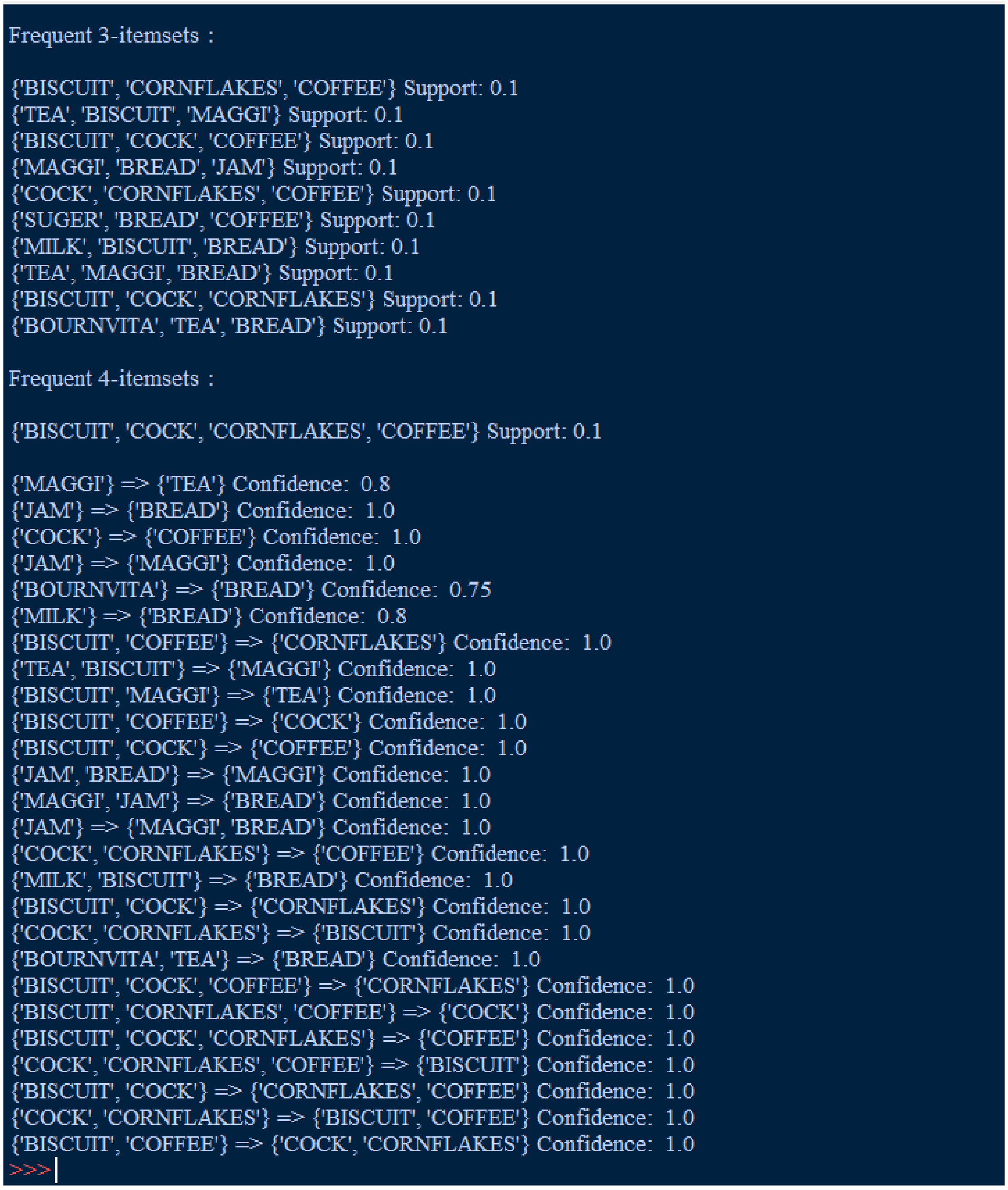
```
import numpy as np import pandas as pd
data = pd.read_csv('C:/Users/dell/Desktop/GroceryStoreDataSet.csv', header=None)
data.head()
transactions = []

for i in range(len(data)): transactions.append(data.values[i, 0].split(','))
print(transactions)

class Apriori:
def  init (self, transactions, min_support, min_confidence): self.transactions = transactions
self.min_support = min_support # The minimum support. self.min_confidence =
min_confidence # The minimum confidence.
self.support_data = {} # A dictionary. The key is frequent itemset and the value is support.

def create_C1(self): C1 = set()
for transaction in self.transactions: for item in transaction:
C1.add(frozenset([item])) return C1

def create_Ck(self, Lksub1, k): Ck = set()
len_Lksub1 = len(Lksub1) list_Lksub1 = list(Lksub1) for i in range(len_Lksub1):
for j in range(i+1, len_Lksub1): l1 = list(list_Lksub1[i])
l2 = list(list_Lksub1[j]) l1.sort()
l2.sort()
if l1[0:k-2] == l2[0:k-2]:
# TODO: self joining Lk-1 Ck_tmp = list(set(l1) | (set(l2))) # TODO: pruning
flag = 1
for k in range(len(Ck_tmp)): tmp = Ck_tmp.copy() tmp.pop(k)
if not set(tmp) in Lksub1: flag = 0
break if flag:
Ck.add(frozenset(Ck_tmp))
return Ck

def generate_Lk_from_Ck(self, Ck): Lk = set()
item_count = {}

for transaction in self.transactions: for item in Ck:
if item.issubset(transaction): if item not in item_count:
item_count[item] = 1 else:
item_count[item] += 1 t_num = float(len(self.transactions)) for item in item_count:
support = item_count[item] / t_num if support >= self.min_support:
Lk.add(item) self.support_data[item] = support
return Lk
```

```
def generate_L(self): self.support_data = {}

C1 = self.create_C1()
L1 = self.generate_Lk_from_Ck(C1) Lksub1 = L1.copy()
L = []
L.append(Lksub1) i = 2
while True:
Ci = self.create_Ck(Lksub1, i)
Li = self.generate_Lk_from_Ck(Ci) if Li:

Lksub1 = Li.copy() L.append(Lksub1) i += 1
else:
break return L

def generate_rules(self): L = self.generate_L()

big_rule_list = [] sub_set_list = []
for i in range(0, len(L)): for freq_set in L[i]:
for sub_set in sub_set_list:
if sub_set.issubset(freq_set):
# TODO : compute the confidence
conf = float(format(self.support_data[freq_set] / self.support_data[freq_set -

sub_set],'.3f'))

big_rule = (set(freq_set - sub_set), set(sub_set), conf)
if conf >= self.min_confidence and big_rule not in big_rule_list: big_rule_list.append(big_rule)

sub_set_list.append(freq_set) return big_rule_list
model = Apriori(transactions, min_support=0.1, min_confidence=0.75) L = model.generate_L()

for Lk in L:
print('Frequent {}-itemsets ： \n'.format(len(list(Lk)[0])))

for freq_set in Lk:
print(set(freq_set), 'Support:', model.support_data[freq_set])

print()
rule_list = model.generate_rules()


for item in rule_list:
print(item[0], "=>", item[1], "Confidence: ", item[2])
```

Output:



```
===================== RESTART: C:\Users\dell\Desktop\Apriori.py =====================
[['MILK', 'BREAD', 'BISCUIT'], ['BREAD', 'MILK', 'BISCUIT', 'CORNFLAKES'], ['BREAD', 'TEA', 'BOURNVITA'], ['JA
M', 'MAGGI', 'BREAD', 'MILK'], ['MAGGI', 'TEA', 'BISCUIT'], ['BREAD', 'TEA', 'BOURNVITA'], ['MAGGI', 'TEA', 'COR
NFLAKES'], ['MAGGI', 'BREAD', 'TEA', 'BISCUIT'], ['JAM', 'MAGGI', 'BREAD', 'TEA'], ['BREAD', 'MILK'], ['COFFEE',
'COCK', 'BISCUIT', 'CORNFLAKES'], ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'], ['COFFEE', 'SUGER', 'BOURN
VITA'], ['BREAD', 'COFFEE', 'COCK'], ['BREAD', 'SUGER', 'BISCUIT'], ['COFFEE', 'SUGER', 'CORNFLAKES'], ['BRE
AD', 'SUGER', 'BOURNVITA'], ['BREAD', 'COFFEE', 'SUGER'], ['BREAD', 'COFFEE', 'SUGER'], ['TEA', 'MILK', 'COFF
EE', 'CORNFLAKES']]
Frequent 1-itemsets :

 {'CORNFLAKES'} Support: 0.3
 {'JAM'} Support: 0.1
 {'MAGGI'} Support: 0.25
 {'SUGER'} Support: 0.3
 {'BOURNVITA'} Support: 0.2
 {'COCK'} Support: 0.15
 {'MILK'} Support: 0.25
 {'BREAD'} Support: 0.65
 {'COFFEE'} Support: 0.4
 {'TEA'} Support: 0.35
 {'BISCUIT'} Support: 0.35

Frequent 2-itemsets :

 {'SUGER', 'COFFEE'} Support: 0.2
 {'TEA', 'MAGGI'} Support: 0.2
 {'MILK', 'CORNFLAKES'} Support: 0.1
 {'BOURNVITA', 'TEA'} Support: 0.1
 {'JAM', 'BREAD'} Support: 0.1
 {'MAGGI', 'BREAD'} Support: 0.15
 {'COCK', 'COFFEE'} Support: 0.15
 {'TEA', 'CORNFLAKES'} Support: 0.1
 {'BISCUIT', 'COCK'} Support: 0.1
 {'MAGGI', 'BISCUIT'} Support: 0.1
 {'MAGGI', 'JAM'} Support: 0.1
 {'BISCUIT', 'BREAD'} Support: 0.2
 {'CORNFLAKES', 'COFFEE'} Support: 0.2
 {'BISCUIT', 'COFFEE'} Support: 0.1
 {'BOURNVITA', 'BREAD'} Support: 0.15
 {'MILK', 'BISCUIT'} Support: 0.1
 {'BISCUIT', 'CORNFLAKES'} Support: 0.15
 {'TEA', 'BISCUIT'} Support: 0.1
 {'BREAD', 'COFFEE'} Support: 0.15
 {'MILK', 'BREAD'} Support: 0.2
 {'COCK', 'CORNFLAKES'} Support: 0.1
 {'BOURNVITA', 'SUGER'} Support: 0.1
 {'TEA', 'BREAD'} Support: 0.2
 {'SUGER', 'BREAD'} Support: 0.2
```



```
Frequent 3-itemsets :

{'BISCUIT', 'CORNFLAKES', 'COFFEE'} Support: 0.1
{'TEA', 'BISCUIT', 'MAGGI'} Support: 0.1
{'BISCUIT', 'COCK', 'COFFEE'} Support: 0.1
{'MAGGI', 'BREAD', 'JAM'} Support: 0.1
{'COCK', 'CORNFLAKES', 'COFFEE'} Support: 0.1
{'SUGER', 'BREAD', 'COFFEE'} Support: 0.1
{'MILK', 'BISCUIT', 'BREAD'} Support: 0.1
{'TEA', 'MAGGI', 'BREAD'} Support: 0.1
{'BISCUIT', 'COCK', 'CORNFLAKES'} Support: 0.1
{'BOURNVITA', 'TEA', 'BREAD'} Support: 0.1

Frequent 4-itemsets :

{'BISCUIT', 'COCK', 'CORNFLAKES', 'COFFEE'} Support: 0.1

{'MAGGI'} => {'TEA'} Confidence:  0.8
{'JAM'} => {'BREAD'} Confidence:  1.0
{'COCK'} => {'COFFEE'} Confidence:  1.0
{'JAM'} => {'MAGGI'} Confidence:  1.0
{'BOURNVITA'} => {'BREAD'} Confidence:  0.75
{'MILK'} => {'BREAD'} Confidence:  0.8
{'BISCUIT', 'COFFEE'} => {'CORNFLAKES'} Confidence:  1.0
{'TEA', 'BISCUIT'} => {'MAGGI'} Confidence:  1.0
{'BISCUIT', 'MAGGI'} => {'TEA'} Confidence:  1.0
{'BISCUIT', 'COFFEE'} => {'COCK'} Confidence:  1.0
{'BISCUIT', 'COCK'} => {'COFFEE'} Confidence:  1.0
{'JAM', 'BREAD'} => {'MAGGI'} Confidence:  1.0
{'MAGGI', 'JAM'} => {'BREAD'} Confidence:  1.0
{'JAM'} => {'MAGGI', 'BREAD'} Confidence:  1.0
{'COCK', 'CORNFLAKES'} => {'COFFEE'} Confidence:  1.0
{'MILK', 'BISCUIT'} => {'BREAD'} Confidence:  1.0
{'BISCUIT', 'COCK'} => {'CORNFLAKES'} Confidence:  1.0
{'COCK', 'CORNFLAKES'} => {'BISCUIT'} Confidence:  1.0
{'BOURNVITA', 'TEA'} => {'BREAD'} Confidence:  1.0
{'BISCUIT', 'COCK', 'COFFEE'} => {'CORNFLAKES'} Confidence:  1.0
{'BISCUIT', 'CORNFLAKES', 'COFFEE'} => {'COCK'} Confidence:  1.0
{'BISCUIT', 'COCK', 'CORNFLAKES'} => {'COFFEE'} Confidence:  1.0
{'COCK', 'CORNFLAKES', 'COFFEE'} => {'BISCUIT'} Confidence:  1.0
{'BISCUIT', 'COCK'} => {'CORNFLAKES', 'COFFEE'} Confidence:  1.0
{'COCK', 'CORNFLAKES'} => {'BISCUIT', 'COFFEE'} Confidence:  1.0
{'BISCUIT', 'COFFEE'} => {'COCK', 'CORNFLAKES'} Confidence:  1.0
>>>
```

Post-lab Questions:

1. Explain any two methods to improve the efficiency of Apriori algorithm.

Ans:

a. Among mining algorithms based on association rules, Apriori technique, mining frequent itemset and interesting associations in transaction database, is not only the first used association rule mining technique but also the most popular one. It has been found out that the traditional Apriori algorithms has two major bottlenecks: scanning the database frequently; generating a large number of candidate sets. Based on the inherent defects of Apriori algorithm, some related improvements are carried out:
    - using new database mapping way to avoid scanning the database repeatedly.
    - further pruning frequent itemsets and candidate itemsets in order to improve joining efficiency.
    - using overlap strategy to count support to achieve high efficiency.

b. Under the same conditions, the results illustrate that the proposed improved Apriori algorithm improves the operating efficiency compared with other improved algorithms.

2. Give the steps to generate association rules from frequent itemsets which are generated by apriori algorithm.

Ans:

a. Once the frequent item-sets have been discovered by the apriori algorithm, generating association rules comes into picture.
b. For that, we calculate the confidence of each rule.
c. This confidence is calculated with the formula:
   Confidence(A->B)=Support_count(A∪B)/Support_count(A)
d. For example, for an itemset {I1, I2, I3} from L3 the rules can be formed as such:
e. [I1^I2]=>[I3] //confidence = sup(I1^I2^I3)/sup(I1^I2) = 2/4*100=50%
   [I1^I3]=>[I2] //confidence = sup(I1^I2^I3)/sup(I1^I3) = 2/4*100=50%
   [I2^I3]=>[I1] //confidence = sup(I1^I2^I3)/sup(I2^I3) = 2/4*100=50%
   [I1]=>[I2^I3] //confidence = sup(I1^I2^I3)/sup(I1) = 2/6*100=33%
   [I2]=>[I1^I3] //confidence = sup(I1^I2^I3)/sup(I2) = 2/7*100=28%
   [I3]=>[I1^I2] //confidence = sup(I1^I2^I3)/sup(I3) = 2/6*100=33%
f. And here if the minimum confidence to be considered is 50%, then only the first 3 rules are considered as strong association rules.