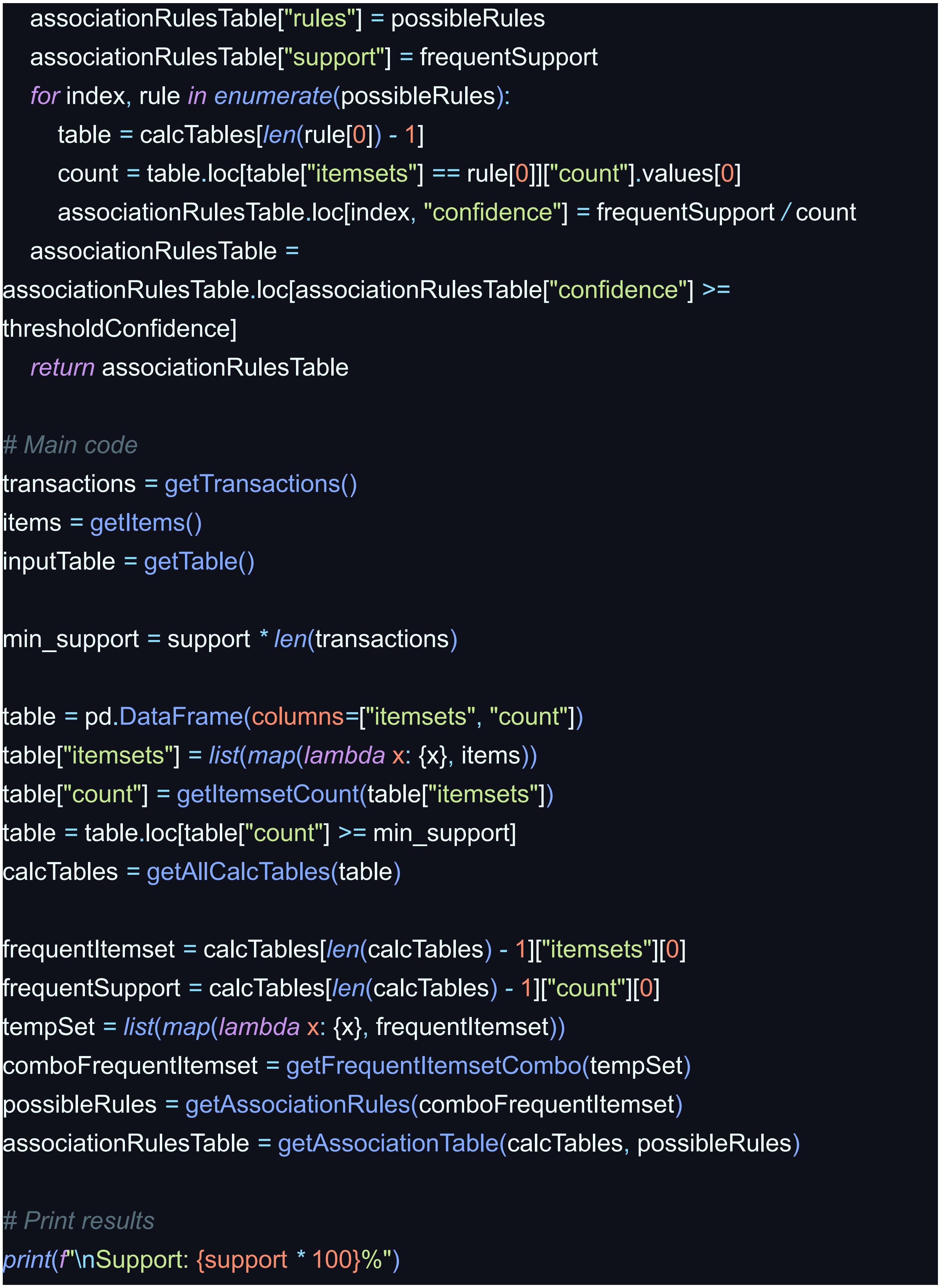
Name: Hardika Patil **Roll No: 66**
Class:COMPS B **Batch: B3**

Experiment No: 8

Aim: Implementation of Apriori Algorithm

Code:

```
import pandas as pd
import itertools as iter

# Input data
inputMatrix = [
    [1, 1, 1, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 1],
    [1, 1, 0, 1, 0],
    [1, 1, 1, 0, 1],
    [1, 1, 1, 1, 0],
]
support = 0.5
thresholdConfidence = 0.6

# Function to get transaction labels
def getTransactions():
    return ["T" + str(i + 1) for i in range(len(inputMatrix))]

# Function to get item labels
def getItems():
    return ["I" + str(i + 1) for i in range(len(inputMatrix[0]))]

# Function to create the initial DataFrame table
def getTable():
    temp = pd.DataFrame(index=transactions, columns=items)
    for i in range(len(transactions)):
        for j in range(len(items)):
            temp.loc["T" + str(i + 1)]["I" + str(j + 1)] = inputMatrix[i][j]
    return temp

# Function to count the occurrences of itemsets in transactions
def getItemsetCount(itemsets):
    temp = []
    for itemset in itemsets:
        count = 0
        for transaction in transactions:
            flag = True
            for item in itemset:
                if inputTable.loc[transaction][item] != 1:
                    flag = False
                    break
            if flag:
                count += 1
        temp.append(count)
    return temp

# Function to create unique combinations of items
def cleanCombinations(items):
    temp = []
    for item in items:
        temp.append(set().union(*item))
    return temp

# Function to generate all calculation tables
def getAllCalcTables(initialTable):
    calcTables = [initialTable]
    for x in range(len(transactions)):
        if calcTables[x].shape[0] <= 1:
            break
        table = pd.DataFrame(columns=["itemsets", "count"])
        items = list(iter.combinations(calcTables[x]["itemsets"], x + 2))
        items = cleanCombinations(items)
        table["itemsets"] = items
        count = getItemsetCount(table["itemsets"])
        table["count"] = count
        table = table.loc[table["count"] >= min_support]
        calcTables.append(table)
    return calcTables

# Function to get combinations of frequent itemsets
def getFrequentItemsetCombo(tempSet):
    combo = []
    for x in range(len(frequentItemset) - 1):
        combo.extend(list(iter.combinations(tempSet, x + 1)))
    return cleanCombinations(combo)

# Function to generate association rules
def getAssociationRules(comboFrequentItemset):
    rules = []
    for index in range(len(comboFrequentItemset)):
        item = comboFrequentItemset[index]
        for x in comboFrequentItemset:
            a = len(item)
            b = len(x)
            if a == 1 and b == 1:
                pass
            elif len(item.union(x)) == (a + b):
                rules.append((item, x))
    return rules

# Function to generate association table
def getAssociationTable(calcTables, possibleRules):
    associationRulesTable = pd.DataFrame(columns=["rules", "support",
"confidence"])
```



```
    associationRulesTable["rules"] = possibleRules
    associationRulesTable["support"] = frequentSupport
    for index, rule in enumerate(possibleRules):
        table = calcTables[len(rule[0]) - 1]
        count = table.loc[table["itemsets"] == rule[0]]["count"].values[0]
        associationRulesTable.loc[index, "confidence"] = frequentSupport / count
    associationRulesTable = associationRulesTable.loc[associationRulesTable["confidence"] >= thresholdConfidence]
    return associationRulesTable

# Main code
transactions = getTransactions()
items = getItems()
inputTable = getTable()

min_support = support * len(transactions)

table = pd.DataFrame(columns=["itemsets", "count"])
table["itemsets"] = list(map(lambda x: {x}, items))
table["count"] = getItemsetCount(table["itemsets"])
table = table.loc[table["count"] >= min_support]
calcTables = getAllCalcTables(table)

frequentItemset = calcTables[len(calcTables) - 1]["itemsets"][0]
frequentSupport = calcTables[len(calcTables) - 1]["count"][0]
tempSet = list(map(lambda x: {x}, frequentItemset))
comboFrequentItemset = getFrequentItemsetCombo(tempSet)
possibleRules = getAssociationRules(comboFrequentItemset)
associationRulesTable = getAssociationTable(calcTables, possibleRules)

# Print results
print(f"\nSupport: {support * 100}%")
print(f"Min-Support: {min_support}")
print(f"Threshold Confidence: {thresholdConfidence * 100}")
print("\nInput Table:\n")
print(inputTable)

print("\n---------Iterations to get frequent itemset    ")
for index, table in enumerate(calcTables):
    print(f"\niteration: {index}")
    print(table)

print(" \n")
print("Frequent itemset: ", frequentItemset, "\n")
print(" Association Rules   \n")
for x in possibleRules:
    print(f"{x[0]} > {x[1]}")
```

Output:

```
Jay@ASUS-TUF-F15 MINGW64 /d/Study
material/Degree/5th_sem/DWM/Expi/4th
$ python -u "d:\Study material\Deg"
e\5th_sem\DWM\Expi\4th\apriori.py"
Support: 50.0%
Min-Support: 3.0
Threshold Confidence: 60.0

Input Table:

   I1 I2 I3 I4 I5
T1  1  1  1  0  0
T2  0  1  1  1  0
T3  0  0  0  1  1

---------Iterations to get frequent itemset

iteration: 0
```

```
  itemsets  count
0    {I1}      4
1    {I2}      5
2    {I3}      4
3    {I4}      4

iteration: 1
   itemsets  count
0  {I1, I2}      4
1  {I1, I3}      3
3  {I3, I2}      4
4  {I4, I2}      3

iteration: 2
       itemsets  count
0  {I1, I3, I2}      3


Frequent itemset:  {'I1', 'I3', 'I2'}

        Association Rules

{'I1'} > {'I3', 'I2'}
{'I3'} > {'I1', 'I2'}
{'I2'} > {'I1', 'I3'}
{'I1', 'I3'} > {'I2'}
{'I1', 'I2'} > {'I3'}
{'I3', 'I2'} > {'I1'}
```