CSE 601
Data Mining and Bioinformatics
Project 1
Dimensionality Reduction and Association Analysis

Part 2: Association Analysis

Submitted By:
Karan Manchandia
UB Name: karanman
Person No.: 50290755

Divya Srivastava
UB Name: divyasri
Person No.: 50290383

Varsha Lakshman
UB Name: varshala
Person No.: 50288138

OBJECTIVE:

The objective of Part 2 of this project is to implement Apriori algorithm to find all frequent item sets and generate association rules and query the generated association rules based on the three given templates. Apriori algorithm is used for mining frequent item sets and rule generation algorithm for generating relevant association rules. The process is started by identifying the frequent individual items in the given data file and extending them to larger and larger item sets.

Apriori Algorithm:

The Apriori algorithm uses support-based pruning to control the exponential growth of candidate item sets. The pruning is achieved because of the anti-monotone property of the support measure which states that the support for an itemset never exceeds the support for its subsets. Conversely if an itemset is infrequent then all of its supersets will also be infrequent. In the first iteration, we identify the frequent item sets. Frequent item sets are the items whose support is greater than the minimum support. In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets. As we have already eliminated the infrequent 1- item sets, we need not compute support for the combinations generated from the infrequent item sets. Because the anti-monotone property ensures that all supersets of the infrequent 1-itemsets must be infrequent. The iterations are continued until the number of frequent itemsets generated is zero.

The following are the steps followed to implement apriori algorithm:

1. Frequent Itemset Generation:
   - Identifying the frequent itemsets. Frequent itemsets are the items whose support is greater than the minimum support.
   - An iterative approach is applied where the previously found k-frequent itemsets are used to find k+1 itemsets.
2. Rule Generation:
   - Rules are generated from the frequent itemsets generated in the previous step. The confidence is calculated from the support of the frequent itemsets. The itemsets whose confidence is more than the minimum confidence provided will be selected for rule generation. All the associated rules generated from the same frequent itemset have the same support.

Task1:

Part 1: Frequent Itemset Generation:

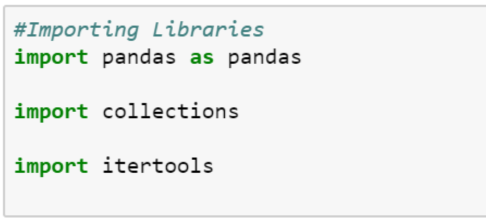
- From the given data set, our aim is to find the frequent itemsets whose support is 30%,40%,50%,60% and 70%. The frequent item sets are selected if their support is above the given support value.
- For length “1” frequent item sets we select only those items whose support is greater than minimum support.

- New candidate item sets are generated from the length “1” frequent item sets using combinations.
- The algorithm of frequent itemset generation is referred from the text book.

Flow of Apriori Algorithm for Frequent Item Set Generation:

Step1: The first part of the implementation is importing the required python libraries. The libraries used are:

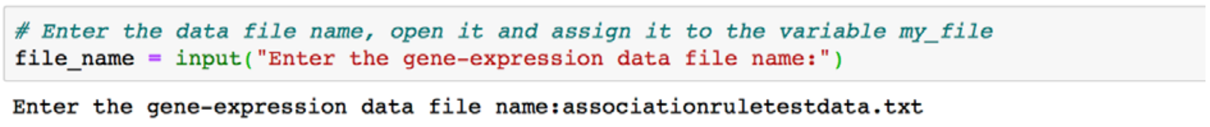
- Pandas: Used for pandas Data Frame
- Collections: Used for counting number of values in a column to generate frequent 1-itemsets.
- Itertools: Combinations functions is used for generating combinations of frequent itemsets.



```
#Importing Libraries
import pandas as pandas

import collections

import itertools
```

Screenshot1

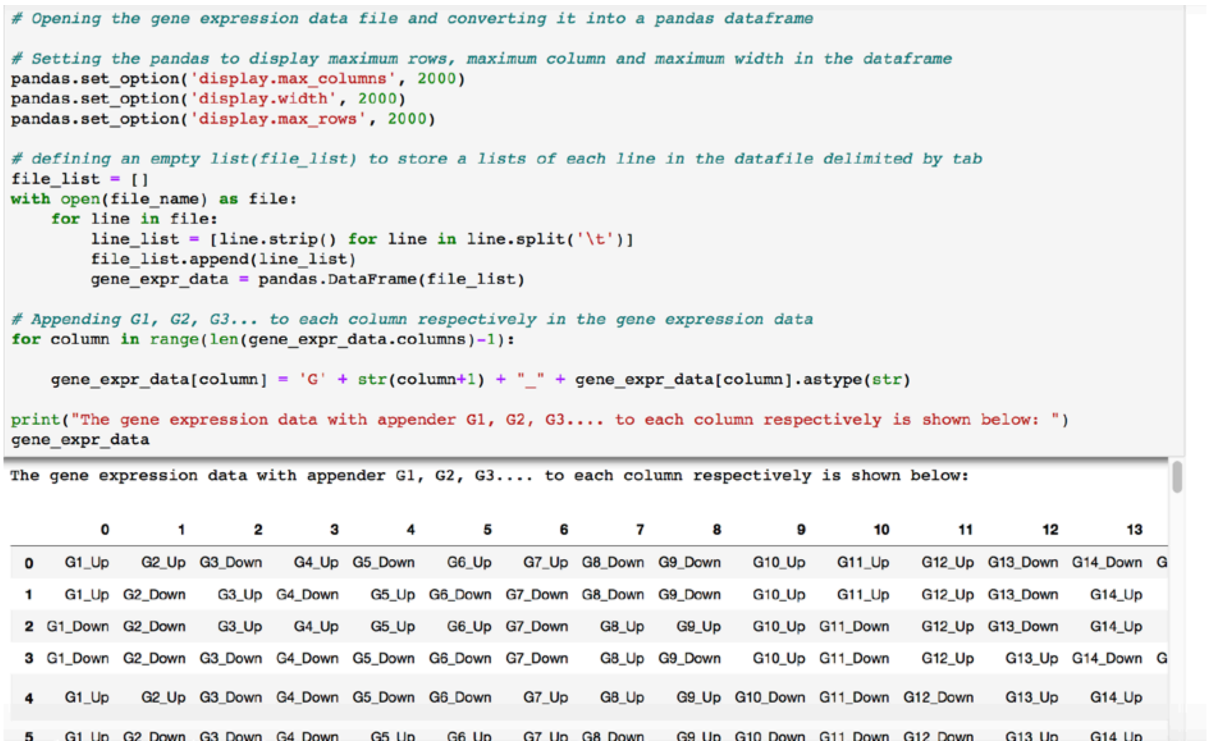
Step 2: Enter the name of the data file:



```
# Enter the data file name, open it and assign it to the variable my_file
file_name = input("Enter the gene-expression data file name:")

Enter the gene-expression data file name:associationruletestdata.txt
```

Screenshot 2

Step 3: We have converted the gene expression data file into a pandas data frame and appended G1, G2, G3... to each column respectively in the gene expression data.



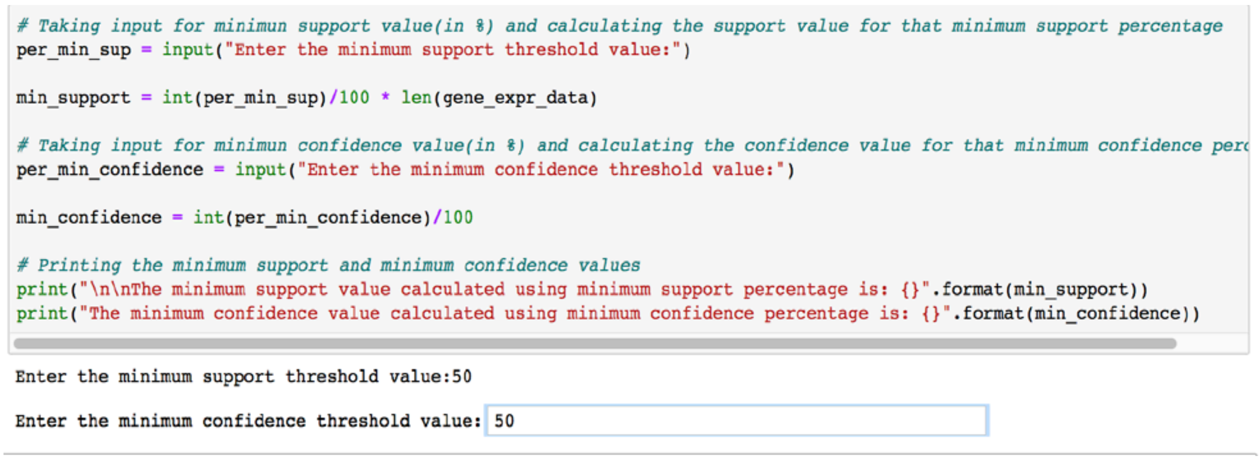
```
# Opening the gene expression data file and converting it into a pandas dataframe

# Setting the pandas to display maximum rows, maximum column and maximum width in the dataframe
pandas.set_option('display.max_columns', 2000)
pandas.set_option('display.width', 2000)
pandas.set_option('display.max_rows', 2000)

# defining an empty list(file_list) to store a lists of each line in the datafile delimited by tab
file_list = []
with open(file_name) as file:
    for line in file:
        line_list = [line.strip() for line in line.split('\t')]
        file_list.append(line_list)
        gene_expr_data = pandas.DataFrame(file_list)

# Appending G1, G2, G3... to each column respectively in the gene expression data
for column in range(len(gene_expr_data.columns)-1):

    gene_expr_data[column] = 'G' + str(column+1) + "_" + gene_expr_data[column].astype(str)

print("The gene expression data with appender G1, G2, G3.... to each column respectively is shown below: ")
gene_expr_data
```

```
The gene expression data with appender G1, G2, G3.... to each column respectively is shown below:
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	G1_Up	G2_Up	G3_Down	G4_Up	G5_Down	G6_Up	G7_Up	G8_Down	G9_Down	G10_Up	G11_Up	G12_Up	G13_Down	G14_Down	G
1	G1_Up	G2_Down	G3_Up	G4_Down	G5_Up	G6_Down	G7_Down	G8_Down	G9_Down	G10_Up	G11_Up	G12_Up	G13_Down	G14_Up	
2	G1_Down	G2_Down	G3_Up	G4_Up	G5_Up	G6_Up	G7_Down	G8_Up	G9_Up	G10_Up	G11_Down	G12_Up	G13_Down	G14_Up	
3	G1_Down	G2_Down	G3_Down	G4_Down	G5_Down	G6_Down	G7_Down	G8_Up	G9_Down	G10_Up	G11_Down	G12_Up	G13_Up	G14_Down	G
4	G1_Up	G2_Up	G3_Down	G4_Down	G5_Down	G6_Down	G7_Up	G8_Up	G9_Up	G10_Down	G11_Down	G12_Down	G13_Up	G14_Up	
5	G1_Up	G2_Down	G3_Down	G4_Down	G5_Up	G6_Up	G7_Up	G8_Down	G9_Up	G10_Down	G11_Down	G12_Down	G13_Up	G14_Up	

Screenshot 3

Step 4: Enter the minimum support and confidence.

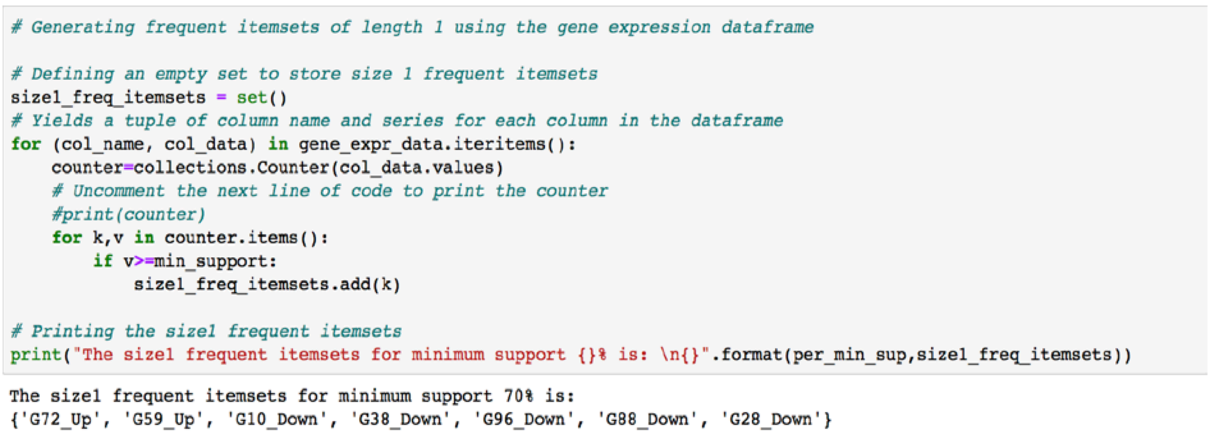


```
# Taking input for minimun support value(in %) and calculating the support value for that minimum support percentage
per_min_sup = input("Enter the minimum support threshold value:")

min_support = int(per_min_sup)/100 * len(gene_expr_data)

# Taking input for minimun confidence value(in %) and calculating the confidence value for that minimum confidence perc
per_min_confidence = input("Enter the minimum confidence threshold value:")

min_confidence = int(per_min_confidence)/100

# Printing the minimum support and minimum confidence values
print("\n\nThe minimum support value calculated using minimum support percentage is: {}".format(min_support))
print("The minimum confidence value calculated using minimum confidence percentage is: {}".format(min_confidence))
```

```
Enter the minimum support threshold value:50

Enter the minimum confidence threshold value: 50
```

Screenshot 4

Step 5: We have generated frequent item sets of length 1 for the items whose support is greater than minimum support.

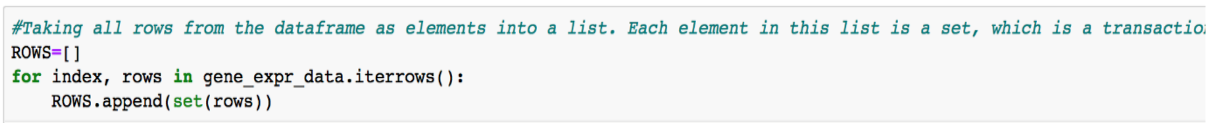


```
# Generating frequent itemsets of length 1 using the gene expression dataframe

# Defining an empty set to store size 1 frequent itemsets
size1_freq_itemsets = set()
# Yields a tuple of column name and series for each column in the dataframe
for (col_name, col_data) in gene_expr_data.iteritems():
    counter=collections.Counter(col_data.values)
    # Uncomment the next line of code to print the counter
    #print(counter)
    for k,v in counter.items():
        if v>=min_support:
            size1_freq_itemsets.add(k)

# Printing the size1 frequent itemsets
print("The size1 frequent itemsets for minimum support {}% is: \n{}".format(per_min_sup,size1_freq_itemsets))
```

```
The size1 frequent itemsets for minimum support 70% is:
{'G72_Up', 'G59_Up', 'G10_Down', 'G38_Down', 'G96_Down', 'G88_Down', 'G28_Down'}
```

Screenshot 5

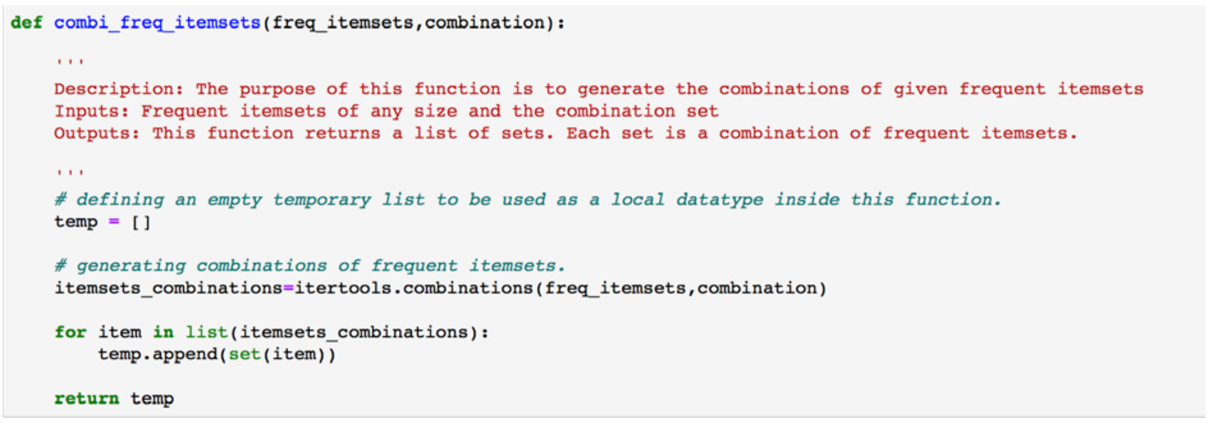
Step 6: We are iterating the rows from the data frame and adding it into a list. Each element in this list is a set, which is a transaction.



```
#Taking all rows from the dataframe as elements into a list. Each element in this list is a set, which is a transactio
ROWS=[]
for index, rows in gene_expr_data.iterrows():
    ROWS.append(set(rows))
```

Screenshot 6

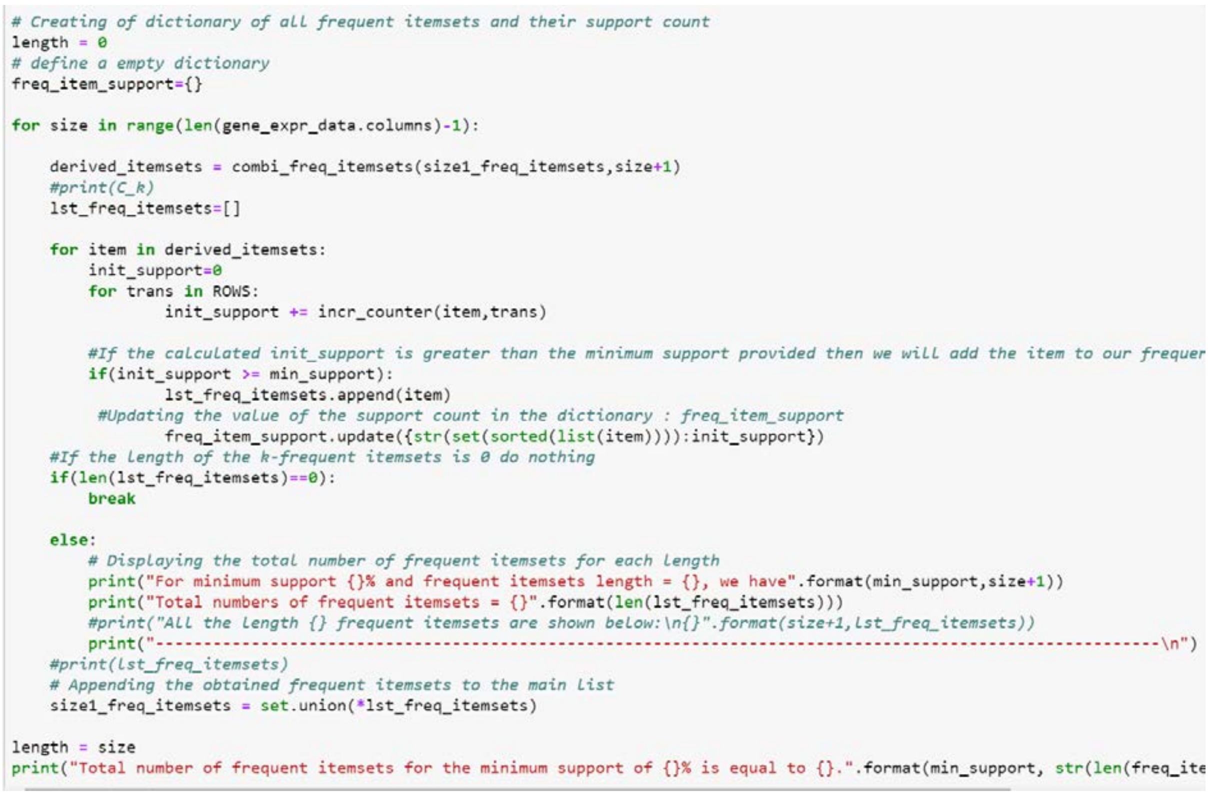
Step 7: We have generated new candidate itemsets using the frequent length 1 itemsets found in the previous step. We have used itertools library. The below is the function which generates k items using the frequent length (k-1) item sets.



```
def combi_freq_itemsets(freq_itemsets,combination):

    '''
    Description: The purpose of this function is to generate the combinations of given frequent itemsets
    Inputs: Frequent itemsets of any size and the combination set
    Outputs: This function returns a list of sets. Each set is a combination of frequent itemsets.

    '''
    # defining an empty temporary list to be used as a local datatype inside this function.
    temp = []

    # generating combinations of frequent itemsets.
    itemsets_combinations=itertools.combinations(freq_itemsets,combination)

    for item in list(itemsets_combinations):
        temp.append(set(item))

    return temp
```

Screenshot 7

Step 8: From the newly generated candidate item sets we check if it is a subset of transactions [the list which has all the rows]. If it is a subset and the support of the item is greater than the minimum support provided, we will add the item to our frequent itemsets collection.

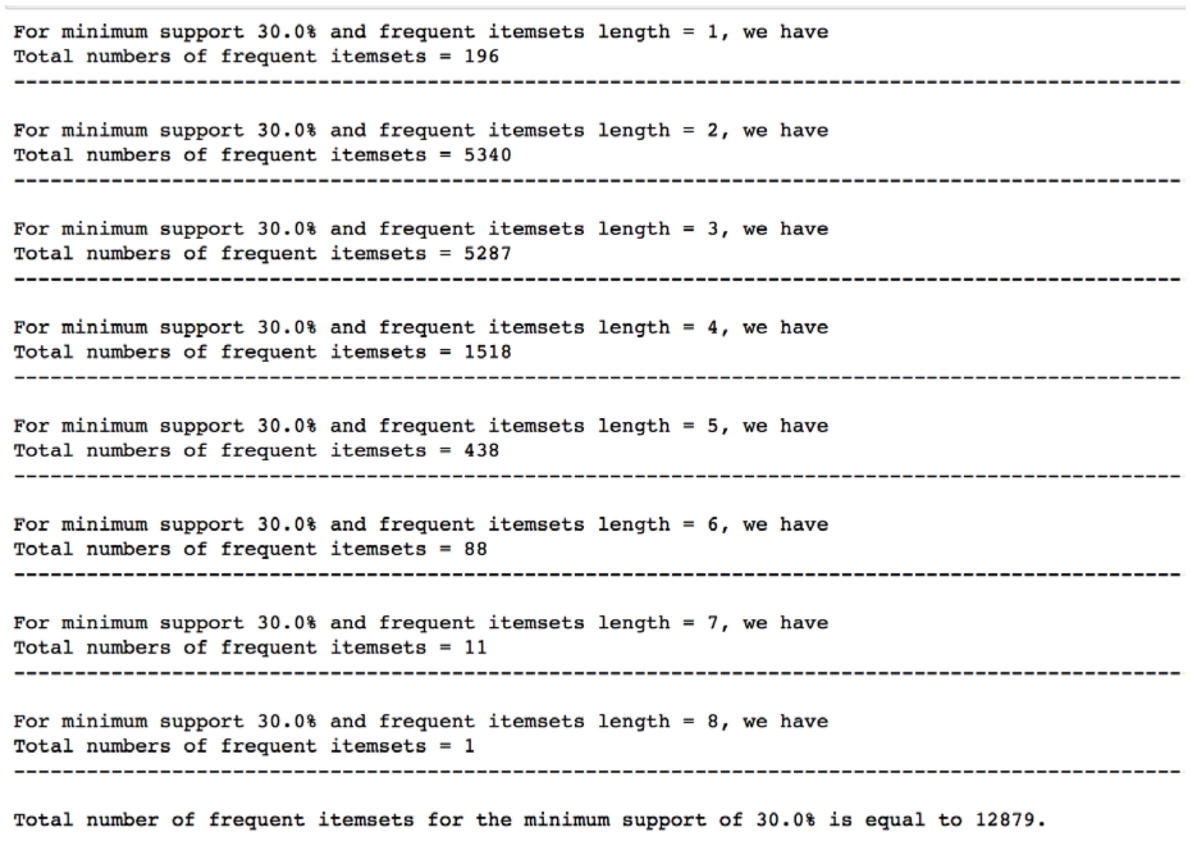


```
# Creating of dictionary of all frequent itemsets and their support count
length = 0
# define a empty dictionary
freq_item_support={}

for size in range(len(gene_expr_data.columns)-1):

    derived_itemsets = combi_freq_itemsets(size1_freq_itemsets,size+1)
    #print(C_k)
    lst_freq_itemsets=[]

    for item in derived_itemsets:
        init_support=0
        for trans in ROWS:
                init_support += incr_counter(item,trans)

        #If the calculated init_support is greater than the minimum support provided then we will add the item to our frequer
        if(init_support >= min_support):
                lst_freq_itemsets.append(item)
         #Updating the value of the support count in the dictionary : freq_item_support
                freq_item_support.update({str(set(sorted(list(item)))):init_support})
    #If the length of the k-frequent itemsets is 0 do nothing
    if(len(lst_freq_itemsets)==0):
        break

    else:
        # Displaying the total number of frequent itemsets for each length
        print("For minimum support {}% and frequent itemsets length = {}, we have".format(min_support,size+1))
        print("Total numbers of frequent itemsets = {}".format(len(lst_freq_itemsets)))
        #print("All the length {} frequent itemsets are shown below:\n{}".format(size+1,lst_freq_itemsets))
        print("-----------------------------------------------------------------------------------------------------------\n")
    #print(lst_freq_itemsets)
    # Appending the obtained frequent itemsets to the main list
    size1_freq_itemsets = set.union(*lst_freq_itemsets)

length = size
print("Total number of frequent itemsets for the minimum support of {}% is equal to {}.".format(min_support, str(len(freq_ite
```

Screenshot 8

Below are the screenshots attached for the frequent item sets:

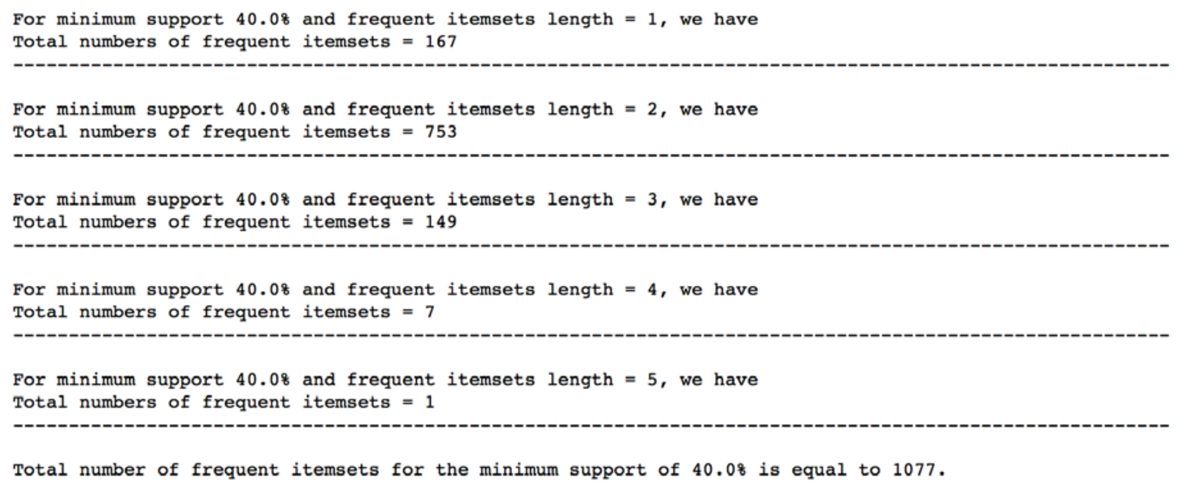
Frequent item sets for Support = 30%:



```
For minimum support 30.0% and frequent itemsets length = 1, we have
Total numbers of frequent itemsets = 196
------------------------------------------------------------------------------------------------

For minimum support 30.0% and frequent itemsets length = 2, we have
Total numbers of frequent itemsets = 5340
------------------------------------------------------------------------------------------------

For minimum support 30.0% and frequent itemsets length = 3, we have
Total numbers of frequent itemsets = 5287
------------------------------------------------------------------------------------------------

For minimum support 30.0% and frequent itemsets length = 4, we have
Total numbers of frequent itemsets = 1518
------------------------------------------------------------------------------------------------

For minimum support 30.0% and frequent itemsets length = 5, we have
Total numbers of frequent itemsets = 438
------------------------------------------------------------------------------------------------

For minimum support 30.0% and frequent itemsets length = 6, we have
Total numbers of frequent itemsets = 88
------------------------------------------------------------------------------------------------

For minimum support 30.0% and frequent itemsets length = 7, we have
Total numbers of frequent itemsets = 11
------------------------------------------------------------------------------------------------

For minimum support 30.0% and frequent itemsets length = 8, we have
Total numbers of frequent itemsets = 1
------------------------------------------------------------------------------------------------

Total number of frequent itemsets for the minimum support of 30.0% is equal to 12879.
```

Screenshot 9

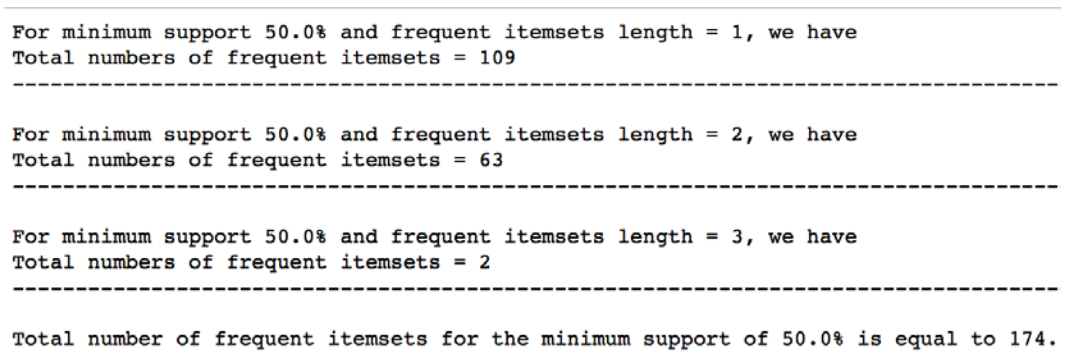
Frequent Item set generation for Support = 40%:



```
For minimum support 40.0% and frequent itemsets length = 1, we have
Total numbers of frequent itemsets = 167
------------------------------------------------------------------------------------------------

For minimum support 40.0% and frequent itemsets length = 2, we have
Total numbers of frequent itemsets = 753
------------------------------------------------------------------------------------------------

For minimum support 40.0% and frequent itemsets length = 3, we have
Total numbers of frequent itemsets = 149
------------------------------------------------------------------------------------------------

For minimum support 40.0% and frequent itemsets length = 4, we have
Total numbers of frequent itemsets = 7
------------------------------------------------------------------------------------------------

For minimum support 40.0% and frequent itemsets length = 5, we have
Total numbers of frequent itemsets = 1
------------------------------------------------------------------------------------------------

Total number of frequent itemsets for the minimum support of 40.0% is equal to 1077.
```

Screenshot 10

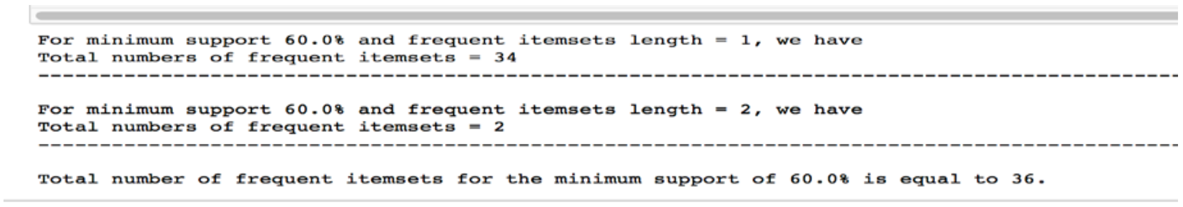
Frequent Item set generation for Support = 50%:



```
For minimum support 50.0% and frequent itemsets length = 1, we have
Total numbers of frequent itemsets = 109
------------------------------------------------------------------------------

For minimum support 50.0% and frequent itemsets length = 2, we have
Total numbers of frequent itemsets = 63
------------------------------------------------------------------------------

For minimum support 50.0% and frequent itemsets length = 3, we have
Total numbers of frequent itemsets = 2
------------------------------------------------------------------------------

Total number of frequent itemsets for the minimum support of 50.0% is equal to 174.
```

Screenshot 11

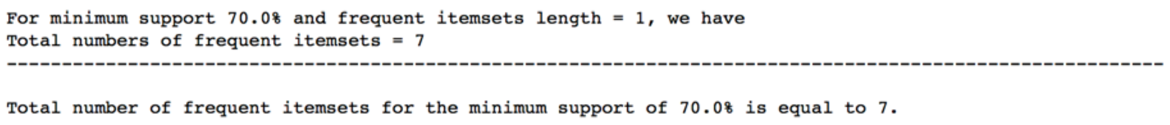
Frequent Item set generation for Support = 60%:



```
For minimum support 60.0% and frequent itemsets length = 1, we have
Total numbers of frequent itemsets = 34
------------------------------------------------------------------------------------------

For minimum support 60.0% and frequent itemsets length = 2, we have
Total numbers of frequent itemsets = 2
------------------------------------------------------------------------------------------

Total number of frequent itemsets for the minimum support of 60.0% is equal to 36.
```

Screenshot 12

Frequent Item set generation for Support = 70%:



```
For minimum support 70.0% and frequent itemsets length = 1, we have
Total numbers of frequent itemsets = 7
------------------------------------------------------------------------------------------

Total number of frequent itemsets for the minimum support of 70.0% is equal to 7.
```

Screenshot 13

Frequent Itemset Generation Results:

- **For Support = 30%**:
  - Number of length-1 frequent itemsets: 196
  - Number of length-2 frequent itemsets: 5340
  - Number of length-3 frequent itemsets: 5287
  - Number of length-4 frequent itemsets: 1518
  - Number of length-5 frequent itemsets: 438
  - Number of length-6 frequent itemsets: 88
  - Number of length-7 frequent itemsets: 11
  - Number of length-8 frequent itemsets: 1
  - Total frequent itemsets: 12879

- **For Support = 40%**:

  - Number of length-1 frequent itemsets: 167
  - Number of length-2 frequent itemsets: 753
  - Number of length-3 frequent itemsets: 149
  - Number of length-4 frequent itemsets: 7
  - Number of length-5 frequent itemsets: 1
  - Total frequent itemsets: 1077

- **For Support = 50%**:
  - Number of length-1 frequent itemsets: 109
  - Number of length-2 frequent itemsets: 63
  - Number of length-3 frequent itemsets: 2
  - Total frequent itemsets: 174

- **For Support = 60%**:
  - Number of length-1 frequent itemsets: 34
  - Number of length-2 frequent itemsets: 2
  - Total frequent itemsets: 36

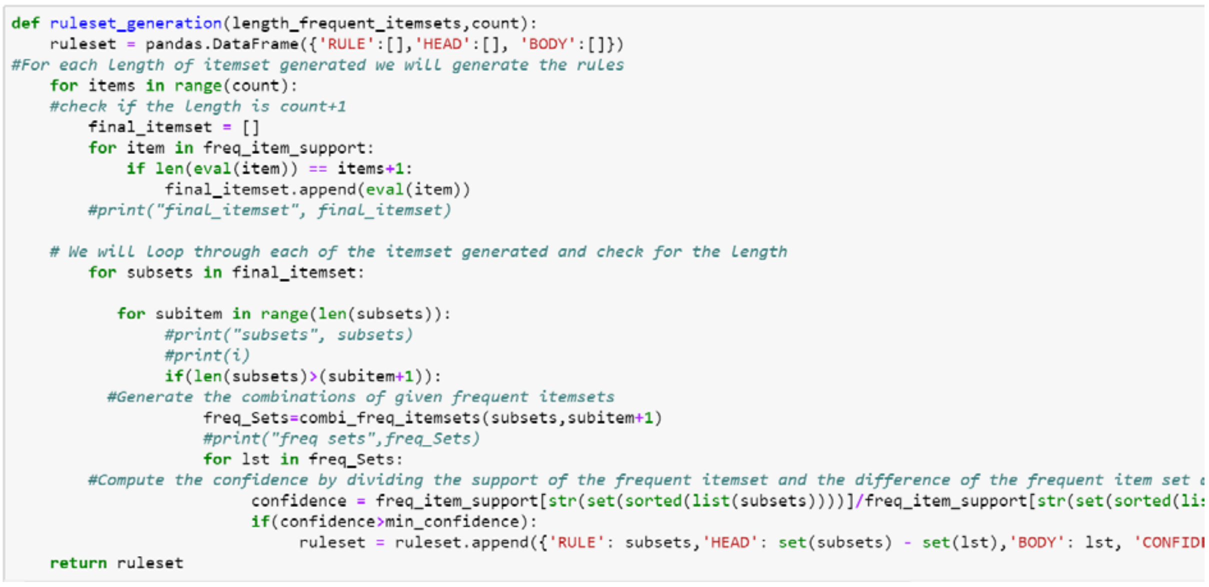
- **For Support = 70%**:
  - Number of length-1 frequent itemsets: 7
  - Total frequent itemsets: 7

Part 2: Rule Generation:

- From the frequent itemsets obtained from the previous step we will generate rules. To find the confidence we have to divide the support of the items in the frequent item set list and the support of the XOR of the combination of items in the frequent item set list and the itemset. The rules for which the confidence is greater than the minimum confidence, are added to ruleset. The code snippet is attached below.
- The rules generated from the same frequent itemset have the following property: if $X \rightarrow Y-X$ does not satisfy the confidence threshold then no rule $X' \rightarrow Y-X'$, where X' is a subset of X, satisfies the confidence threshold. This property follows from the anti-monotone property of the support count $\sigma(X') \geq \sigma(X)$, therefore

$$c(X \rightarrow Y-X) = \sigma(X \cup Y-X)/\sigma(X) \geq \sigma(X' \cup Y-X')/\sigma(X') = c(X' \rightarrow Y-X')$$

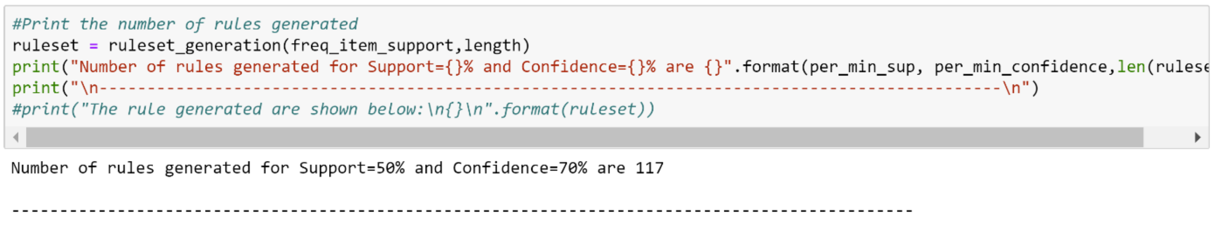
- As Apriori uses level-wise search for rule generation, it starts from empty right-hand side and all items in the left-hand side. To generate a rule in the next level it merges the left-hand sides of two confident rules on the previous level.
- The algorithm for rule generation is referred from the text book.



```
def ruleset_generation(length_frequent_itemsets,count):
    ruleset = pandas.DataFrame({'RULE':[],'HEAD':[], 'BODY':[]})
#For each length of itemset generated we will generate the rules
    for items in range(count):
    #check if the length is count+1
        final_itemset = []
        for item in freq_item_support:
            if len(eval(item)) == items+1:
                final_itemset.append(eval(item))
        #print("final_itemset", final_itemset)

    # We will loop through each of the itemset generated and check for the length
        for subsets in final_itemset:

           for subitem in range(len(subsets)):
                #print("subsets", subsets)
                #print(i)
                if(len(subsets)>(subitem+1)):
          #Generate the combinations of given frequent itemsets
                    freq_Sets=combi_freq_itemsets(subsets,subitem+1)
                    #print("freq sets",freq_Sets)
                    for lst in freq_Sets:
        #Compute the confidence by dividing the support of the frequent itemset and the difference of the frequent item set
                         confidence = freq_item_support[str(set(sorted(list(subsets))))]/freq_item_support[str(set(sorted(li
                         if(confidence>min_confidence):
                              ruleset = ruleset.append({'RULE': subsets,'HEAD': set(subsets) - set(lst),'BODY': lst, 'CONFID
    return ruleset
```

Screenshot 14: Rule Generation Algorithm

Rule Generation Result Screenshot for Support = 50% and confidence = 70%:



```
#Print the number of rules generated
ruleset = ruleset_generation(freq_item_support,length)
print("Number of rules generated for Support={}% and Confidence={}% are {}".format(per_min_sup, per_min_confidence,len(rulese
print("\n------------------------------------------------------------------------------------------\n")
#print("The rule generated are shown below:\n{}\n".format(ruleset))
```

```
Number of rules generated for Support=50% and Confidence=70% are 117

---------------------------------------------------------------------------------------------
```

Screenshot 15

Results for rule generation:

- Support is set to be 30% and confidence 70%.
  Number of rules generated: 31576

- Support is set to be 40% and confidence 70%
  Number of rules generated: 1130

- **Support is set to be 50% and confidence 70%**
  Number of rules generated: 117

- Support is set to be 60% and confidence 70%
  Number of rules generated: 4

- Support is set to be 70% and confidence 70%
  Number of rules generated: 0

Task 2: Association Rules Generation Based on the Templates

Template Descriptions:

Template1:

- The given template1 is of the following form- {RULE|HEAD|BODY} HAS ({ANY|NUMBER|NONE}) OF (ITEM1, ITEM2, ..., ITEMn).
- The first parameter depicts the part of the RULE where the matching itemset should be searched for.
- RULE defines the frequent itemsets generated in the above steps
- HEAD defines the left hand side of the RULE which depicts all the combinations of itemsets that tends to the right hand side of the RULE i.e. BODY
- BODY defines the right hand side of the RULE which depicts all the combinations of itemsets that the combinations of left hand side of RULE tends to.
- The second parameter depicts the constraint on the rule selection i.e. "ANY", "NONE" or "NUMBER".
- ANY, filters the rules if the given item appears in the part of the RULE passed in first parameter.
- NONE filters the rules if the given item appears in none of the itemsets in the part of the RULE that was passed in the first parameters.
- NUMBER filters the rules if only the exact number of times, the itemset appears in the part of the RULE that was passed in the first parameter.
- The third parameter contains the list of items that are to be searched in the part of the RULE, passed in the first parameter.
- Eg- asso_rule.template1("RULE", "ANY", ['G59_Up'])

Template2:

- The given template2 is of the following form-SizeOf({HEAD|BODY|RULE}) ≥ NUMBER
- The first parameter depicts the part of the RULE where the matching itemset should be searched for. They are RULE, BODY and HEAD as described above.
- The second parameter, depicts the minimum number of items that should be present in the part of the rule, passed in the first parameter of the query.
- Eg- asso_rule.template2("RULE", 3)

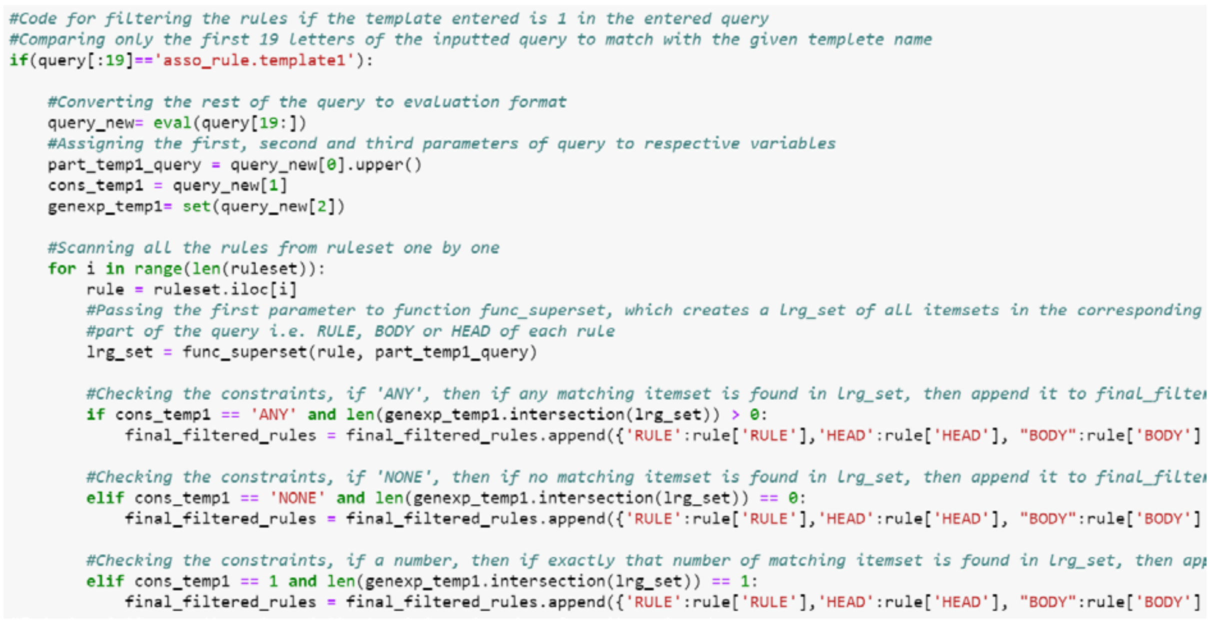
Template3:

- The given template3 is the combination of template1 and template2 queries combined with 'OR' or 'AND' logics.
- Here the first parameter consists of the combination rule for both the types of the queries where the 'OR/AND' logic gets defined and the templates on which this logic is to be applied, is given i.e. template1 or template2.
- The other parameters of the query constitute of the parameters of template1 or template2 as defined in the first parameter of this query.

- Eg- asso_rule.template3("1or1", "HEAD", "ANY",['G10_Down'], "BODY", 1, ['G59_Up'])

Steps followed to implement the Association Rule Generation Based on Template:

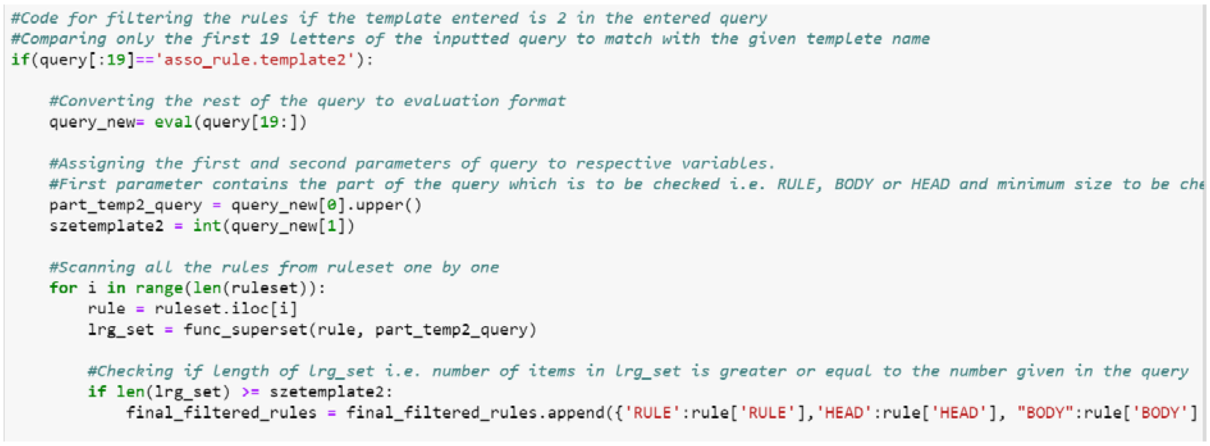
- Input the query from command line and separate the template name and the query parameters and make the query parameters executable.
- Go to the code written for the respective templates i.e. template1, template2 and template3.
- For template1, separate the list of parameters and match the given item in the part of the rule specified as per the given constraint, “ANY”, “NONE” or “NUMBER”. Append the resulting rules into a list and display it along with the count of the rules filtered.



```
#Code for filtering the rules if the template entered is 1 in the entered query
#Comparing only the first 19 letters of the inputted query to match with the given templete name
if(query[:19]=='asso_rule.template1'):

    #Converting the rest of the query to evaluation format
    query_new= eval(query[19:])
    #Assigning the first, second and third parameters of query to respective variables
    part_temp1_query = query_new[0].upper()
    cons_temp1 = query_new[1]
    genexp_temp1= set(query_new[2])

    #Scanning all the rules from ruleset one by one
    for i in range(len(ruleset)):
        rule = ruleset.iloc[i]
        #Passing the first parameter to function func_superset, which creates a lrg_set of all itemsets in the corresponding
        #part of the query i.e. RULE, BODY or HEAD of each rule
        lrg_set = func_superset(rule, part_temp1_query)

        #Checking the constraints, if 'ANY', then if any matching itemset is found in lrg_set, then append it to final_filte
        if cons_temp1 == 'ANY' and len(genexp_temp1.intersection(lrg_set)) > 0:
            final_filtered_rules = final_filtered_rules.append({'RULE':rule['RULE'],'HEAD':rule['HEAD'], "BODY":rule['BODY']

        #Checking the constraints, if 'NONE', then if no matching itemset is found in lrg_set, then append it to final_filte
        elif cons_temp1 == 'NONE' and len(genexp_temp1.intersection(lrg_set)) == 0:
            final_filtered_rules = final_filtered_rules.append({'RULE':rule['RULE'],'HEAD':rule['HEAD'], "BODY":rule['BODY']

        #Checking the constraints, if a number, then if exactly that number of matching itemset is found in lrg_set, then ap
        elif cons_temp1 == 1 and len(genexp_temp1.intersection(lrg_set)) == 1:
            final_filtered_rules = final_filtered_rules.append({'RULE':rule['RULE'],'HEAD':rule['HEAD'], "BODY":rule['BODY']
```

Screenshot16

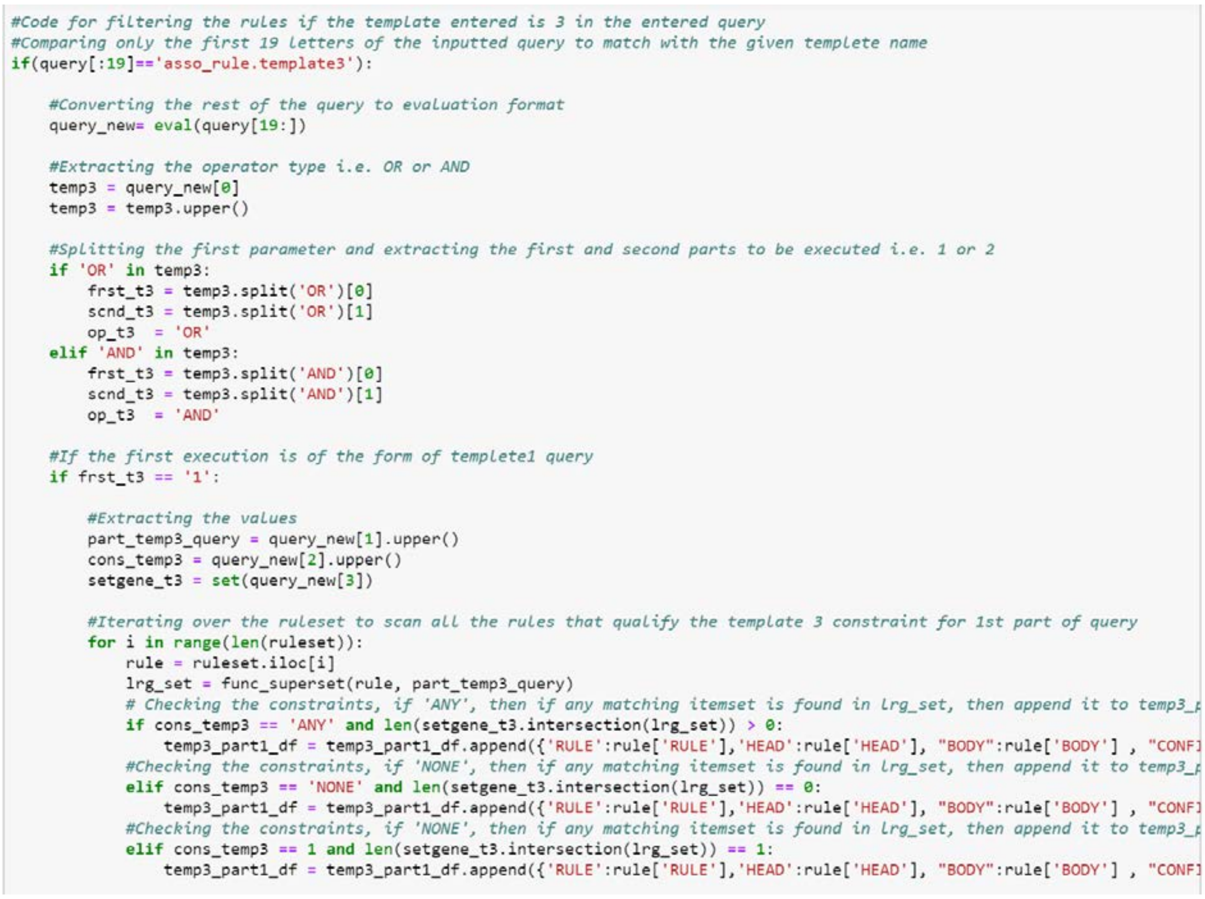
- For template2, separate the list of parameters and filter all those rules that contains atleast the given number of items, specified in the second parameter of the query. Append the resulting rules into a list and display it along with the count of the rules filtered.



```
#Code for filtering the rules if the template entered is 2 in the entered query
#Comparing only the first 19 letters of the inputted query to match with the given templete name
if(query[:19]=='asso_rule.template2'):

    #Converting the rest of the query to evaluation format
    query_new= eval(query[19:])

    #Assigning the first and second parameters of query to respective variables.
    #First parameter contains the part of the query which is to be checked i.e. RULE, BODY or HEAD and minimum size to be che
    part_temp2_query = query_new[0].upper()
    szetemplate2 = int(query_new[1])

    #Scanning all the rules from ruleset one by one
    for i in range(len(ruleset)):
        rule = ruleset.iloc[i]
        lrg_set = func_superset(rule, part_temp2_query)

        #Checking if length of lrg_set i.e. number of items in lrg_set is greater or equal to the number given in the query
        if len(lrg_set) >= szetemplate2:
            final_filtered_rules = final_filtered_rules.append({'RULE':rule['RULE'],'HEAD':rule['HEAD'], "BODY":rule['BODY']
```

Screenshot 17

- For template3, extract the first parameter that contains the combination of template details i.e. template1 or template2 as well as the combination logic i.e. OR or AND.
- Generate the results for both the queries as per the specified template types and combine them as per the combination logic. Append all the filtered rules to a list and display it along with the count of filtered rules.



```
#Code for filtering the rules if the template entered is 3 in the entered query
#Comparing only the first 19 letters of the inputted query to match with the given templete name
if(query[:19]=='asso_rule.template3'):

    #Converting the rest of the query to evaluation format
    query_new= eval(query[19:])

    #Extracting the operator type i.e. OR or AND
    temp3 = query_new[0]
    temp3 = temp3.upper()

    #Splitting the first parameter and extracting the first and second parts to be executed i.e. 1 or 2
    if 'OR' in temp3:
        frst_t3 = temp3.split('OR')[0]
        scnd_t3 = temp3.split('OR')[1]
        op_t3  = 'OR'
    elif 'AND' in temp3:
        frst_t3 = temp3.split('AND')[0]
        scnd_t3 = temp3.split('AND')[1]
        op_t3  = 'AND'

    #If the first execution is of the form of templete1 query
    if frst_t3 == '1':

        #Extracting the values
        part_temp3_query = query_new[1].upper()
        cons_temp3 = query_new[2].upper()
        setgene_t3 = set(query_new[3])

        #Iterating over the ruleset to scan all the rules that qualify the template 3 constraint for 1st part of query
        for i in range(len(ruleset)):
            rule = ruleset.iloc[i]
            lrg_set = func_superset(rule, part_temp3_query)
            # Checking the constraints, if 'ANY', then if any matching itemset is found in lrg_set, then append it to temp3_p
            if cons_temp3 == 'ANY' and len(setgene_t3.intersection(lrg_set)) > 0:
                temp3_part1_df = temp3_part1_df.append({'RULE':rule['RULE'],'HEAD':rule['HEAD'], "BODY":rule['BODY'] , "CONFI
            #Checking the constraints, if 'NONE', then if any matching itemset is found in lrg_set, then append it to temp3_p
            elif cons_temp3 == 'NONE' and len(setgene_t3.intersection(lrg_set)) == 0:
                temp3_part1_df = temp3_part1_df.append({'RULE':rule['RULE'],'HEAD':rule['HEAD'], "BODY":rule['BODY'] , "CONFI
            #Checking the constraints, if 'NONE', then if any matching itemset is found in lrg_set, then append it to temp3_p
            elif cons_temp3 == 1 and len(setgene_t3.intersection(lrg_set)) == 1:
                temp3_part1_df = temp3_part1_df.append({'RULE':rule['RULE'],'HEAD':rule['HEAD'], "BODY":rule['BODY'] , "CONFI
```

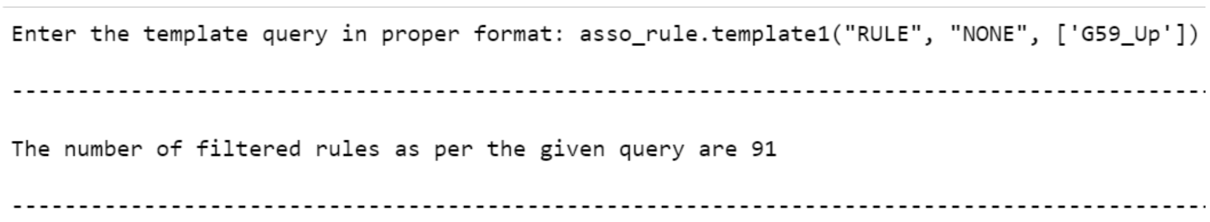
Screenshot 18

Screenshots of Results of Association Rule Generation Based on Templates: (For Support 50% and Confidence 70%)

- **asso_rule.template1("RULE", "ANY", ['G59_Up'])**

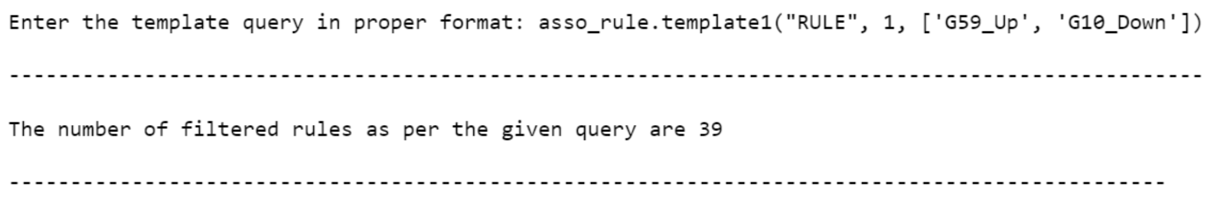
```
Enter the template query in proper format: asso_rule.template1("RULE", "ANY", ['G59_Up'])

-----------------------------------------------------------------------------------

The number of filtered rules as per the given query are 26

-----------------------------------------------------------------------------------
```

- **asso_rule.template1("RULE", "NONE", ['G59_Up'])**



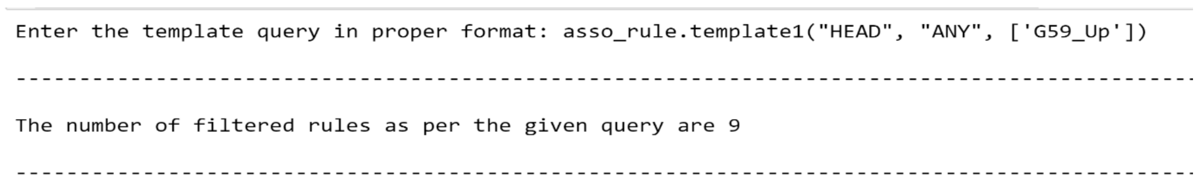
```
Enter the template query in proper format: asso_rule.template1("RULE", "NONE", ['G59_Up'])

-----------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 91

-----------------------------------------------------------------------------------------
```

- **asso_rule.template1("RULE", 1, ['G59_Up', 'G10_Down'])**



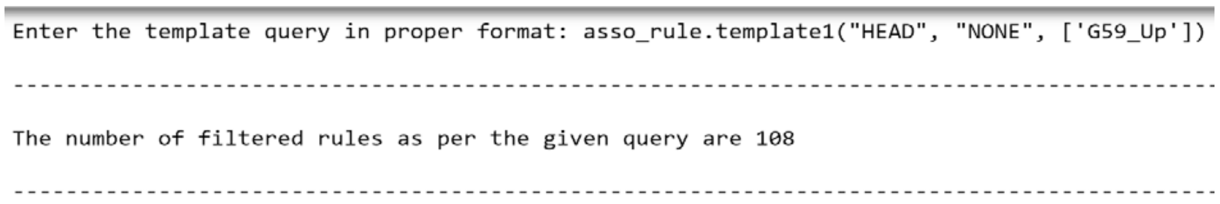
```
Enter the template query in proper format: asso_rule.template1("RULE", 1, ['G59_Up', 'G10_Down'])

-----------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 39

------------------------------------------------------------------------------------
```

- **asso_rule.template1("HEAD", "ANY", ['G59_Up'])**



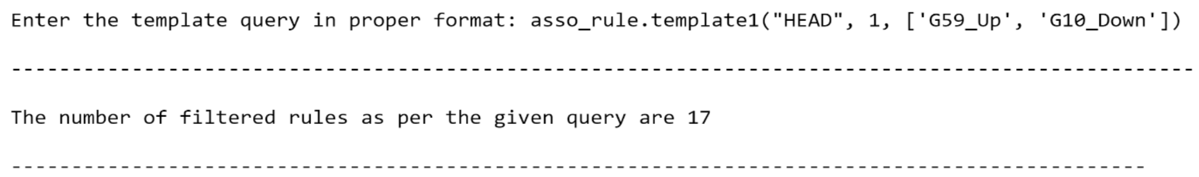
```
Enter the template query in proper format: asso_rule.template1("HEAD", "ANY", ['G59_Up'])

-----------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 9

-----------------------------------------------------------------------------------------
```

- **asso_rule.template1("HEAD", "NONE", ['G59_Up'])**



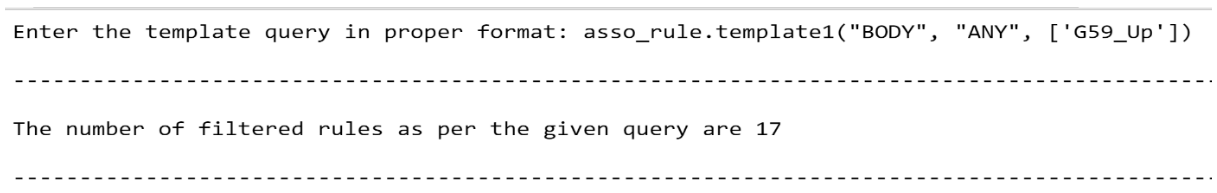
```
Enter the template query in proper format: asso_rule.template1("HEAD", "NONE", ['G59_Up'])

-----------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 108

-----------------------------------------------------------------------------------------
```

- **asso_rule.template1("HEAD", 1, ['G59_Up', 'G10_Down'])**



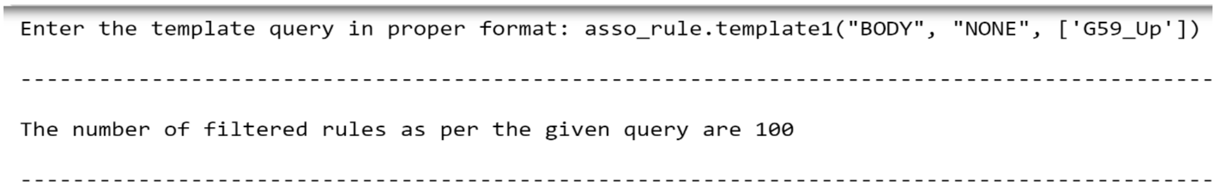
```
Enter the template query in proper format: asso_rule.template1("HEAD", 1, ['G59_Up', 'G10_Down'])

-----------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 17

------------------------------------------------------------------------------------
```

- **asso_rule.template1("BODY", "ANY", ['G59_Up'])**



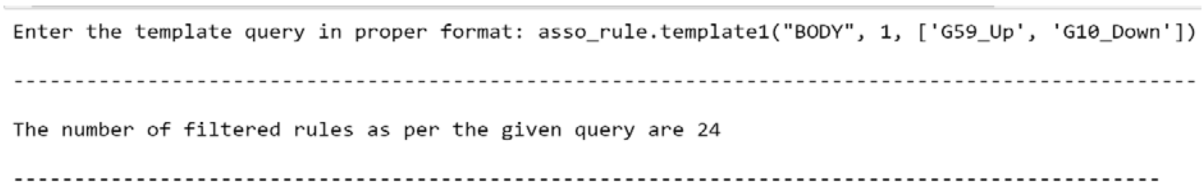
```
Enter the template query in proper format: asso_rule.template1("BODY", "ANY", ['G59_Up'])

-----------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 17

-----------------------------------------------------------------------------------------
```

- **asso_rule.template1("BODY", "NONE", ['G59_Up'])**



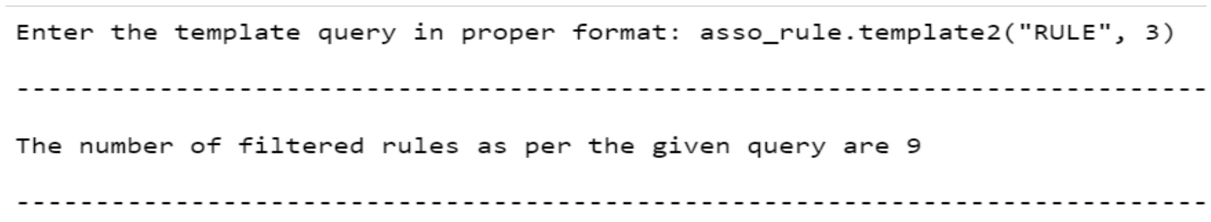
```
Enter the template query in proper format: asso_rule.template1("BODY", "NONE", ['G59_Up'])

-------------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 100

-------------------------------------------------------------------------------------------
```

- **asso_rule.template1("BODY", 1, ['G59_Up', 'G10_Down'])**



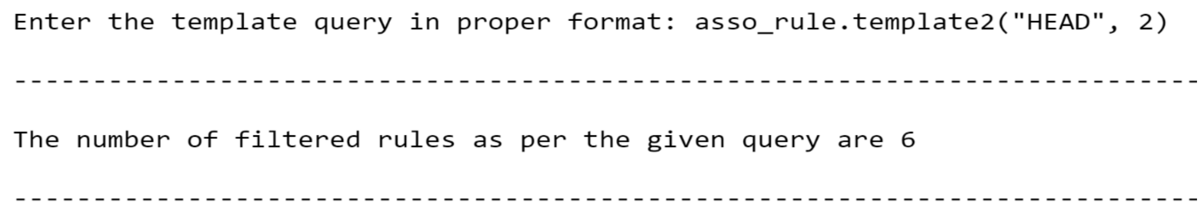
```
Enter the template query in proper format: asso_rule.template1("BODY", 1, ['G59_Up', 'G10_Down'])

------------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 24

------------------------------------------------------------------------------------------
```

- **asso_rule.template2("RULE", 3)**



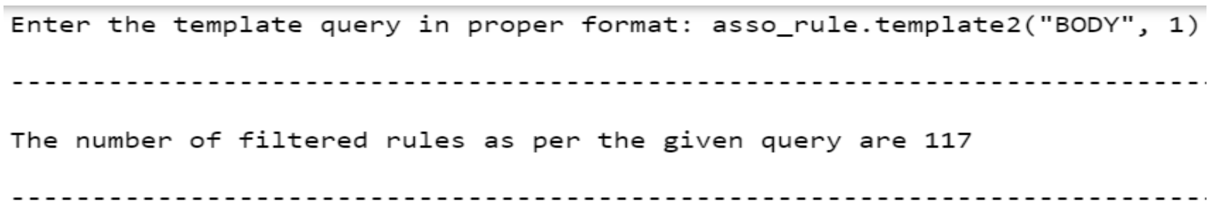
```
Enter the template query in proper format: asso_rule.template2("RULE", 3)

--------------------------------------------------------------------------

The number of filtered rules as per the given query are 9

--------------------------------------------------------------------------
```

- **asso_rule.template2("HEAD", 2)**



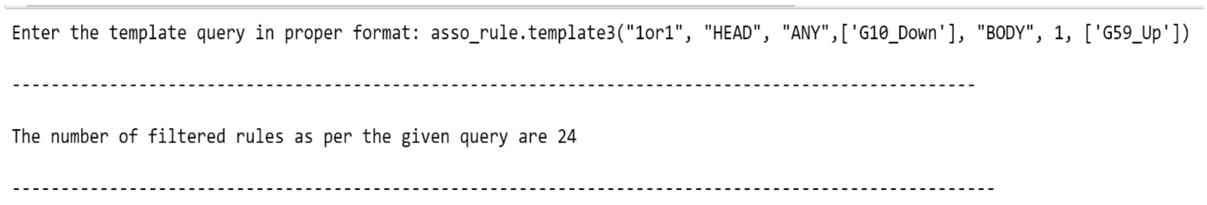
```
Enter the template query in proper format: asso_rule.template2("HEAD", 2)

--------------------------------------------------------------------------

The number of filtered rules as per the given query are 6

--------------------------------------------------------------------------
```

- **asso_rule.template2("BODY", 1)**



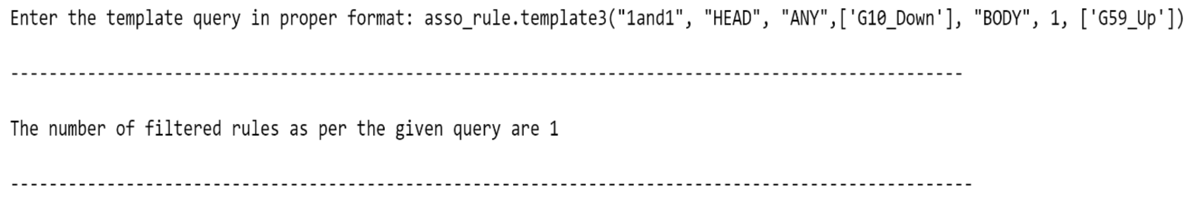
```
Enter the template query in proper format: asso_rule.template2("BODY", 1)

--------------------------------------------------------------------------

The number of filtered rules as per the given query are 117

--------------------------------------------------------------------------
```

- **asso_rule.template3("1or1", "HEAD", "ANY",['G10_Down'], "BODY", 1, ['G59_Up'])**



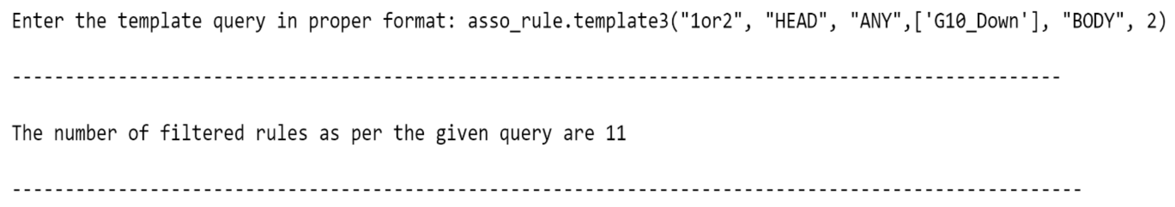
```
Enter the template query in proper format: asso_rule.template3("1or1", "HEAD", "ANY",['G10_Down'], "BODY", 1, ['G59_Up'])

----------------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 24

----------------------------------------------------------------------------------------------
```

- **asso_rule.template3("1and1", "HEAD", "ANY",['G10_Down'], "BODY", 1, ['G59_Up'])**



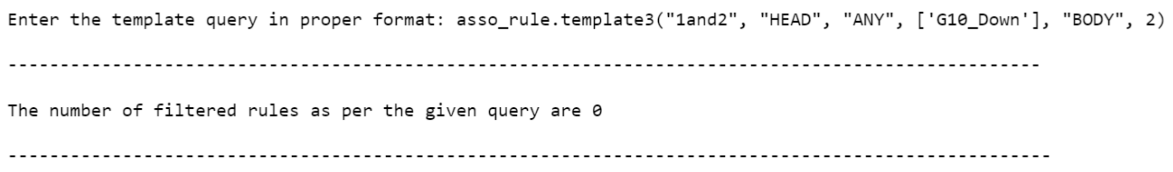
```
Enter the template query in proper format: asso_rule.template3("1and1", "HEAD", "ANY",['G10_Down'], "BODY", 1, ['G59_Up'])

----------------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 1

----------------------------------------------------------------------------------------------
```

- **asso_rule.template3("1or2", "HEAD", "ANY",['G10_Down'], "BODY", 2)**



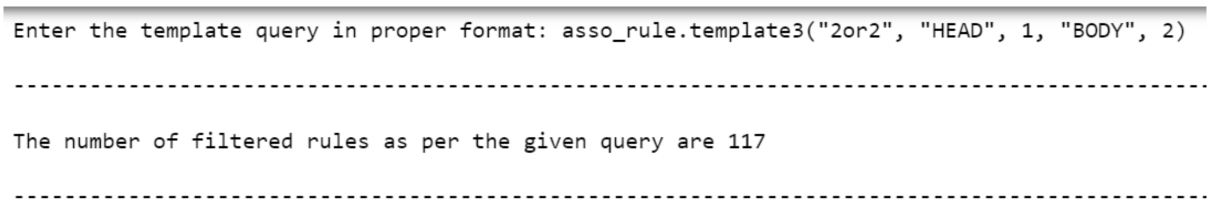
```
Enter the template query in proper format: asso_rule.template3("1or2", "HEAD", "ANY",['G10_Down'], "BODY", 2)

----------------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 11

----------------------------------------------------------------------------------------------
```

- **asso_rule.template3("1and2", "HEAD", "ANY", ['G10_Down'], "BODY", 2)**



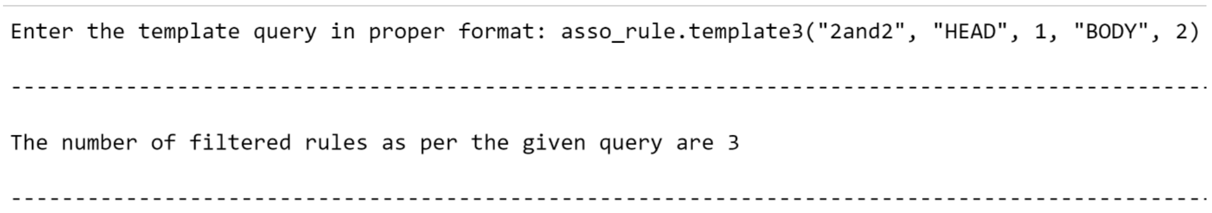
```
Enter the template query in proper format: asso_rule.template3("1and2", "HEAD", "ANY", ['G10_Down'], "BODY", 2)
----------------------------------------------------------------------------------------------
The number of filtered rules as per the given query are 0
----------------------------------------------------------------------------------------------
```

- **asso_rule.template3("2or2", "HEAD", 1, "BODY", 2)**



```
Enter the template query in proper format: asso_rule.template3("2or2", "HEAD", 1, "BODY", 2)

----------------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 117

----------------------------------------------------------------------------------------------
```

- **asso_rule.template3("2and2", "HEAD", 1, "BODY", 2)**



```
Enter the template query in proper format: asso_rule.template3("2and2", "HEAD", 1, "BODY", 2)

----------------------------------------------------------------------------------------------

The number of filtered rules as per the given query are 3

----------------------------------------------------------------------------------------------
```

Results Summarization:

- (result11, cnt) = asso_rule.template1("RULE", "ANY", ['G59_Up']): 26
- (result12, cnt) = asso_rule.template1("RULE", "NONE", ['G59_Up']): 91
- (result13, cnt) = asso_rule.template1("RULE", 1, ['G59_Up', 'G10_Down']): 39
- (result14, cnt) = asso_rule.template1("HEAD", "ANY", ['G59_Up']): 9
- (result15, cnt) = asso_rule.template1("HEAD", "NONE", ['G59_Up']): 108

- (result16, cnt) = asso_rule.template1("HEAD", 1, ['G59_Up', 'G10_Down']): 17
- (result17, cnt) = asso_rule.template1("BODY", "ANY", ['G59_Up']): 17
- (result18, cnt) = asso_rule.template1("BODY", "NONE", ['G59_Up']): 100
- (result19, cnt) = asso_rule.template1("BODY", 1, ['G59_Up', 'G10_Down']): 24
- (result21, cnt) = asso_rule.template2("RULE", 3): 9
- (result22, cnt) = asso_rule.template2("HEAD", 2): 6
- (result23, cnt) = asso_rule.template2("BODY", 1): 117
- (result31, cnt) = asso_rule.template3("1or1", "HEAD", "ANY",['G10_Down'], "BODY", 1, ['G59_Up']): 24
- (result32, cnt) = asso_rule.template3("1and1", "HEAD", "ANY",['G10_Down'], "BODY", 1, ['G59_Up']): 1
- (result33, cnt) = asso_rule.template3("1or2", "HEAD", "ANY",['G10_Down'], "BODY", 2): 11
- (result34, cnt) = asso_rule.template3("1and2", "HEAD", "ANY", ['G10_Down'], "BODY", 2): 0
- (result35, cnt) = asso_rule.template3("2or2", "HEAD", 1, "BODY", 2): 117
- (result36, cnt) = asso_rule.template3("2and2", "HEAD", 1, "BODY", 2): 3

Conclusion:

- Support is an indication of how frequently the items appear in the data.
- Confidence indicates the number of times; the prediction is true.
- If support is high, it indicates that the rule applies to large number of cases
- If confidence is high, it indicates there are higher chances of the prediction to be true.

References:

- https://www.hackerearth.com/blog/developers/beginners-tutorial-apriori-algorithm-data-mining-r-implementation/
- https://www.cs.helsinki.fi/group/bioinfo/teaching/dami_s10/dami_lecture3.pdf
- https://www.saedsayad.com/association_rules.htm
- Introduction to Data Mining. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, Addison Wesley.