**Data Science (ITE4005)**

**Programming Assignment #1**

**Apriori Pruning Algorithm**



컴퓨터공학부 컴퓨터전공

2009004065 유건열

목차

[1. Environment 2](#_Toc477997063)

[2. Summary of algorithm 2](#_Toc477997064)

[(1) Frequent pattern generation 2](#_Toc477997065)

[(2) Making association rules 4](#_Toc477997066)

[(3) Support and confidence 5](#_Toc477997067)

[3. Detailed description of codes 5](#_Toc477997068)

[(1) Main flow 5](#_Toc477997069)

[(2) Detailed description 6](#_Toc477997070)

[a. Imported libraries 6](#_Toc477997071)

[b. Handling user’s argument 6](#_Toc477997072)

[c. Opening files 6](#_Toc477997073)

[d. Reading transactions from the input file 7](#_Toc477997074)

[e. Getting the number of transactions 7](#_Toc477997075)

[f. Dictionary for counting support number 7](#_Toc477997076)

[h. Dictionary for length 1 candidate patterns 9](#_Toc477997077)

[i. Finding dictionary of frequent patterns 9](#_Toc477997078)

[j. Pruning 10](#_Toc477997079)

[k. superset list 11](#_Toc477997080)

[l. superset dictionary 11](#_Toc477997081)

[m. removing length 1 frequent patterns 12](#_Toc477997082)

[n. making association rules 12](#_Toc477997083)

[o. making subsets for association rules 13](#_Toc477997084)

[p. making combinations 14](#_Toc477997085)

[q. printing in set format 15](#_Toc477997086)

[r. closing files 15](#_Toc477997087)

[4. Instructions for building exe 16](#_Toc477997088)

[5. Testing without exe file 21](#_Toc477997089)

# **1. Environment**

- OS : Windows 10

- Language : Python 2.7.12

# **2. Summary of algorithm**

## **(1) Frequent pattern generation**

This assignment is to make the association rules with frequent patterns in input data. Assume that we have some input data like this.

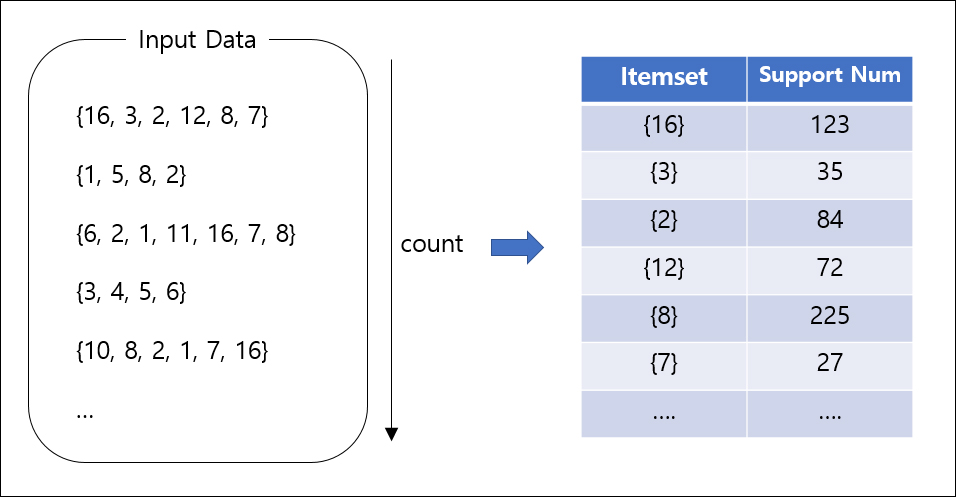


Figure 1

First of my algorithm, count the support number of the item sets that have length 1 as you see in Figure 1. The support number is how many transactions have that item set within.



Figure 2

Assume that we have 10% for the minimum support. So, we have to remove item sets that have the support less than the minimum support like Figure 2. Then, we can get ‘length-1 frequent items’. After this, we’ll make length-2 frequent items.



Figure 3

We can make length-2 candidate sets by making combinations with length-1 frequent items like Figure 3. And then, we can also remove item sets with support less than the minimum support.

After this, we have to use pruning algorithm. That says “If the length-k+1 item set is frequent, all of the subsets with length-k are frequent.” So, if {1,2}, and {1,3} are frequent, but {2,3} is not frequent, you can generate {1,2,3} with {1,2}, and {1,3}, but {1,2,3} is not frequent because {2,3} is not frequent.

However, in length-2 case, every subset of length-2 item sets are frequent. So, we don’t need pruning process this step. (We would generate {16,2} by {16}, {2}. So, {16} and {2} are frequent. And they are all of the subsets of {16,2}.) I’ll show you about pruning process in next step.

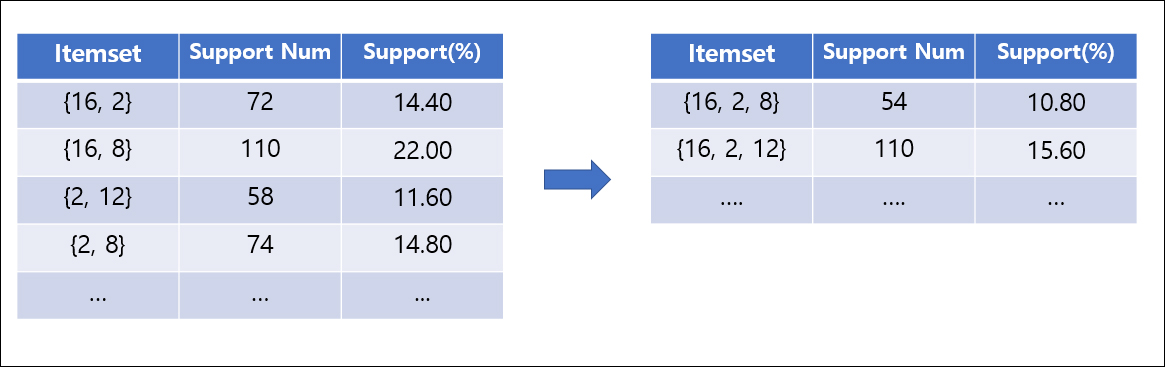


Figure 4

Figure 4 shows how to make length-3 candidate sets with frequent patterns that have length 2. We can make length-3 item sets by getting unions of length 2 item sets that have length 3. (ex. {16, 2} ∪ {16, 8} = {16, 2, 8}) This means we can get the union {16,8,2,12} with {16,8}, and {2,12}. But its length is 4. So, we don’t need to consider it in this step.

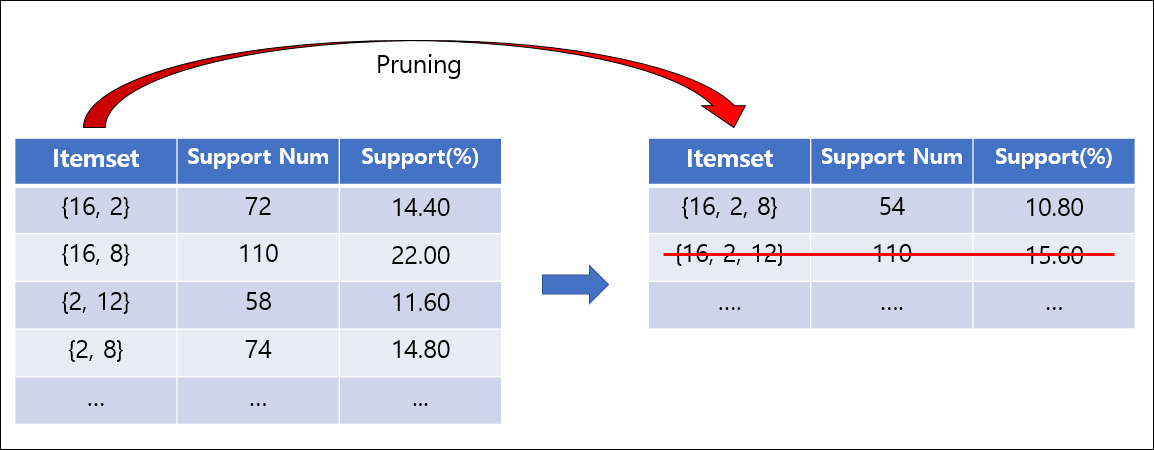


Figure 5

It’s ‘Pruning’ time. As I said, pruning means “to remove the length-k+1 item sets that don’t have all of subsets of them that have length k.” In Figure 5, we can see a candidate set {16,2,12}. Its length-2 subsets are {16,2}, {16,12}, and {2,12}. But we don’t have {16,12} in length-2 frequent sets. So, by pruning rule, {16,2,12} cannot be frequent set, even though it has support higher than the minimum support.

We should iterate this step until there is no frequent or candidate set possible to generate. This is the end of the process for getting frequent patterns. Next, we’ll go to the step for association rules.

## **(2) Making association rules**

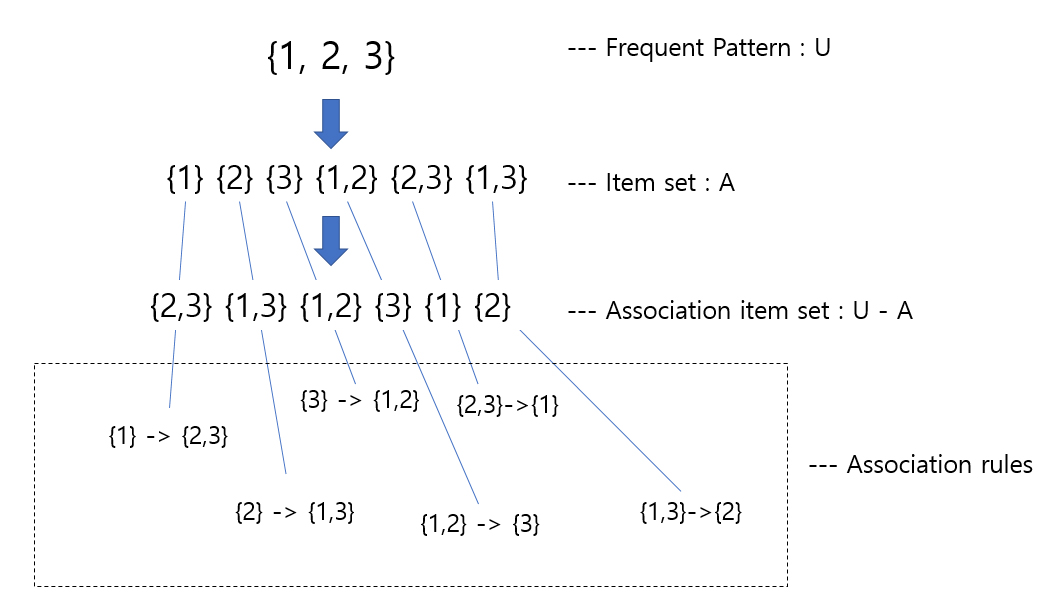


Figure 6

In this assignment, we have to make association rules with the subsets of the frequent patterns like Figure 6. If we have frequent pattern {1,2,3}, U, we can find subsets of {1,2,3} (Of course, except empty set). Let’s call these sets A. Then you can get U-A for association item set. Finally, we can make association rules with A and U-A.

Let’s talk about length-1 frequent patterns. We use the subsets of frequent patterns for making association rules. But, in length-1 case, we cannot make any subsets. That’s why we don’t need consider length-1 frequent patterns.

## **(3) Support and confidence**

After making all association rules, we have to find support and confidence of each rules. If we have association rule A -> B, A = {1,2,3}, B = {4,5}, support and confidence can be calculated by :

In this case, A∪B is same with the frequent pattern of the rule. (ex. If the association rule is {1,2,3} -> {4,5}, {1,2,3} ∪{4,5} = {1,2,3,4,5} is the frequent pattern of that rule.) So, we already have the n(A∪B)! Because when we generate the frequent patterns, we already count it. Calculating support and confidence is the end of this algorithm.

# **3. Detailed description of codes**

## **(1) Main flow**

First of all, it opens file to read and write. Next big step is making frequent patterns. To make frequent patterns, it makes 1-length subsets dictionary like this.

{

{16} : 244

{8} : 232

...

}

With this dictionary, we can check if each item set's support is bigger than the minimum support. If not, that item will be removed.

After this, we can get 1-length frequent patterns with support value dictionary.

With this one, we can make 2-length candidate patterns by using union, and repeat former step (make support value dictionary, and remove patterns with support less than the minimum support).

We can repeat until there is no candidate patterns.

But we should take pruning rule from 3-length candidate patterns.

For example, we can make candidate pattern {1,2,3} with {1,2}, {2,3}. But if there is not {1,3} in 2-length frequent patterns, we cannot say {1,2,3} is frequent pattern by pruning rule.

So, after finding N+1 length candidate patterns, we should check if the candidate patterns have all of its subsets in N-length frequent patterns.

After this step, we finally get all of the frequent patterns.

Next, we'll make association rules with the subsets of frequent patterns. So, as I said in 'Summary of algorithm', I removed 1-length frequent patterns for making association rules.

And we have to make combinations of each frequent pattern. After making these combinations, I calculated support and confidence, and printed them with set format (ex. {1,2,3}).

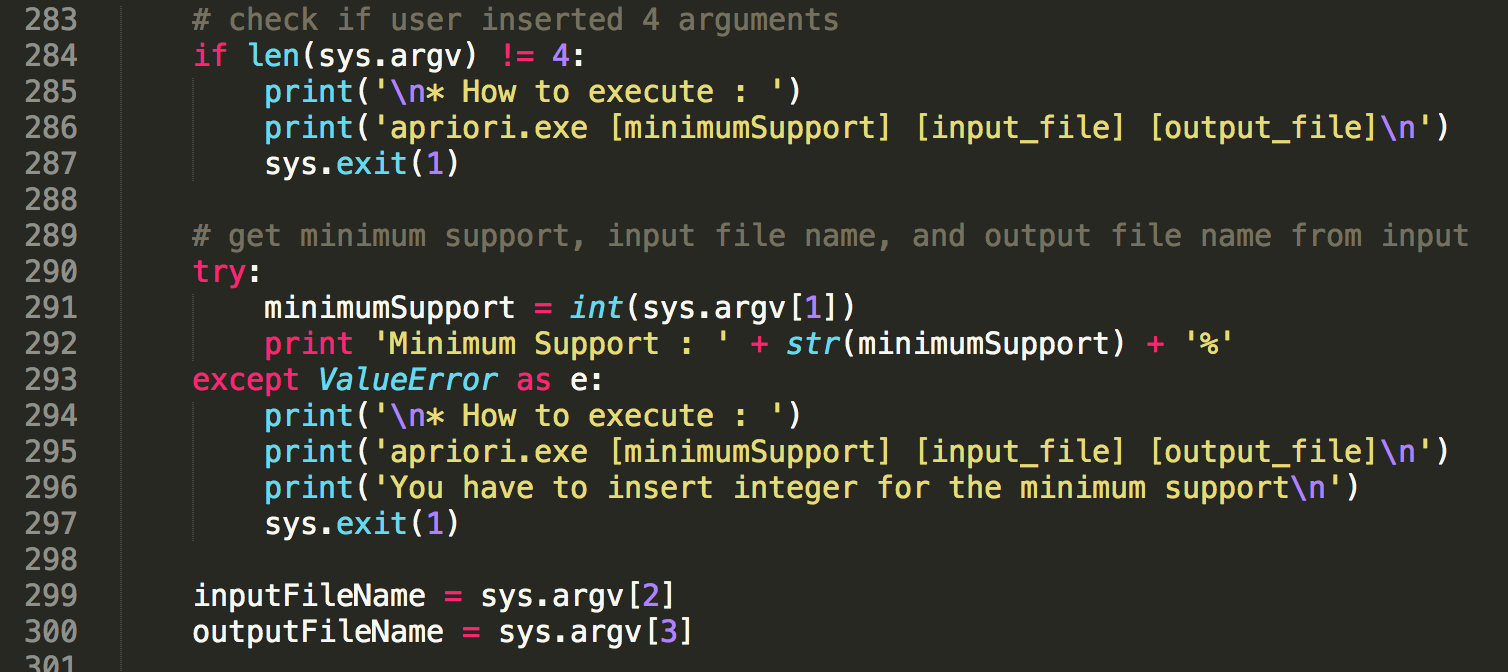
## **(2) Detailed description**

### **a. Imported libraries**



I imported sys for ‘argv’ that is for user's argument, and for exit.

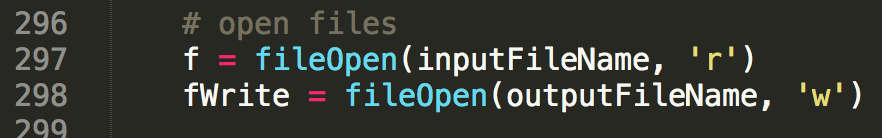
### **b. Handling user’s argument**

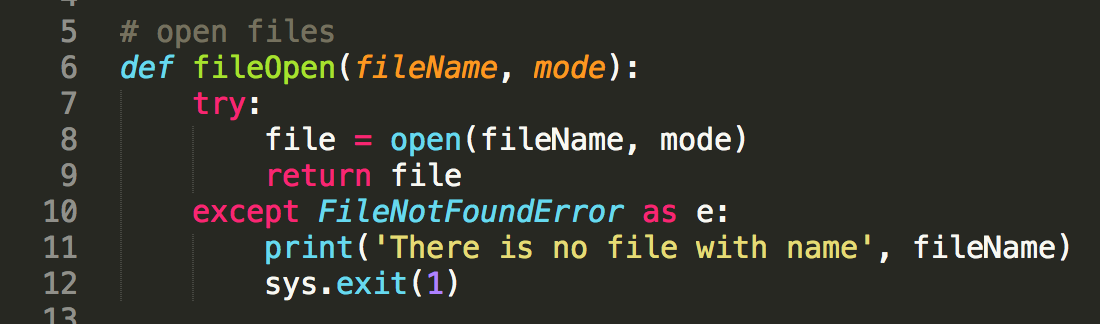


In main function, it checks the argument's number first. we need four argument include file name. (minimum support, input file name, output file name). If the argument is not 4, it shows 'how to execute' and the program will be exit.

If user insert 4 arguments exactly, we can get minimum support, input file name, and output file name.

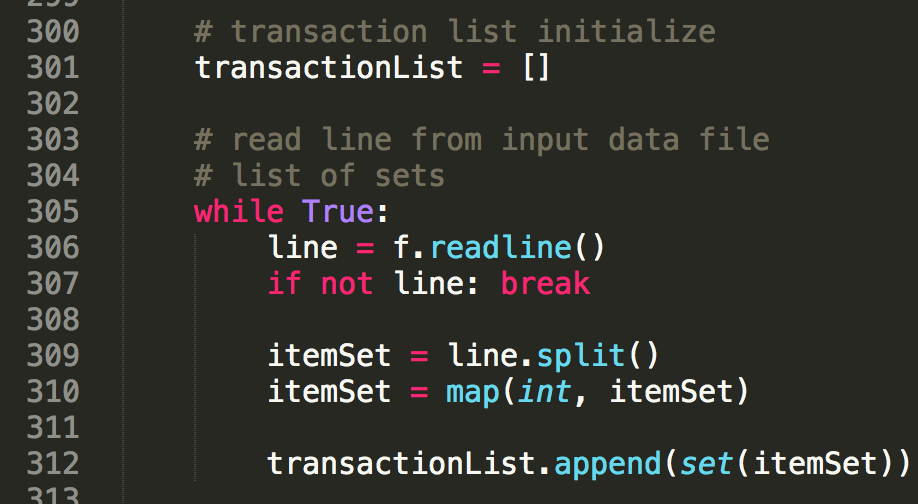
### **c. Opening files**





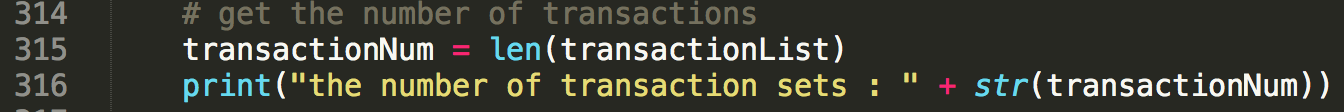
And, it opens files for input and output. ‘fileOpen’ function will do that. If ‘FileNotFoundError’ occurs, program will be exit. (If user insert file name is not in directory.)

### **d. Reading transactions from the input file**

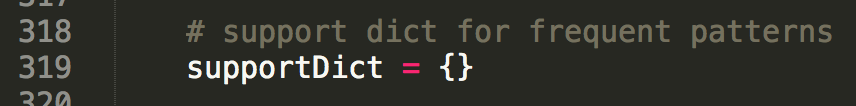


After opening files, it will read transaction lines from the file. The items are int. So, it will convert the string from the file to integer, and append the item sets after making 'set' format.

### **e. Getting the number of transactions**

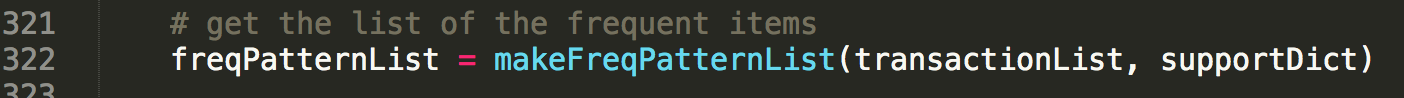
Then, we can get the number of the transactions. it will use it for counting support.

### **f. Dictionary for counting support number**



It makes dictionary for counting support number of each frequent pattern.

**g. Making frequent patterns**



In this step, it makes the list of frequent patterns.



This method is for making frequent pattern list. At first, it gets 1-length subsets dictionary. After that, by comparing each support with the minimum support, it makes frequent pattern dictionary. We can easily get frequent list from the keys of the frequent pattern dictionary. and it saves the frequent pattern dictionary into ‘supportDict’. Because it will use the support number of each frequent patterns to calculate support and confidence later.

After making 1-length frequent patterns, we should generate N-length frequent patterns continuously. It has repeating steps like this :

(1) N+1 length candidate patterns will be generated by N length frequent patterns.

(2) Making support value dictionary of N+1 length candidate patterns

(3) Getting the dictionary of only frequent patterns by removing candidate patterns that have support less than the minimum support.

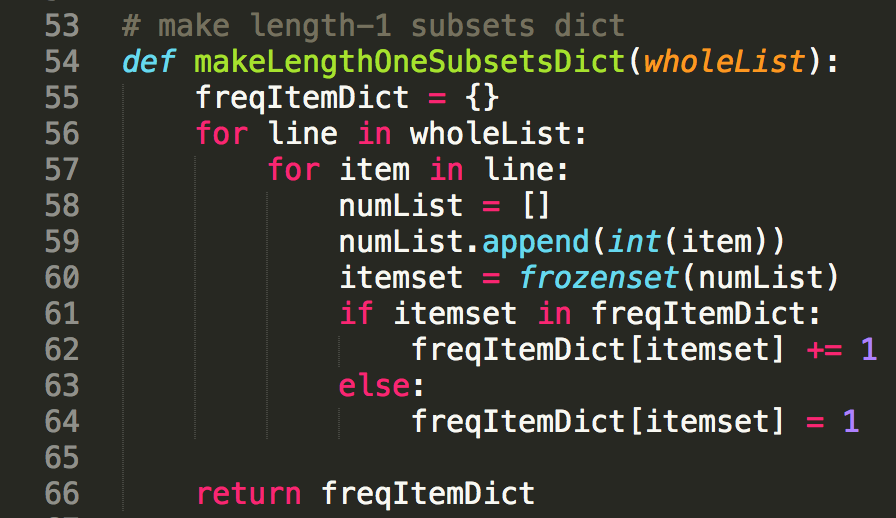
(4) Updating support value dictionary for the frequent patterns.

(5) Adding N+1 length frequent patterns to the frequent pattern list

(6) Saving N+1 frequent patterns to generate N+2 frequent patterns

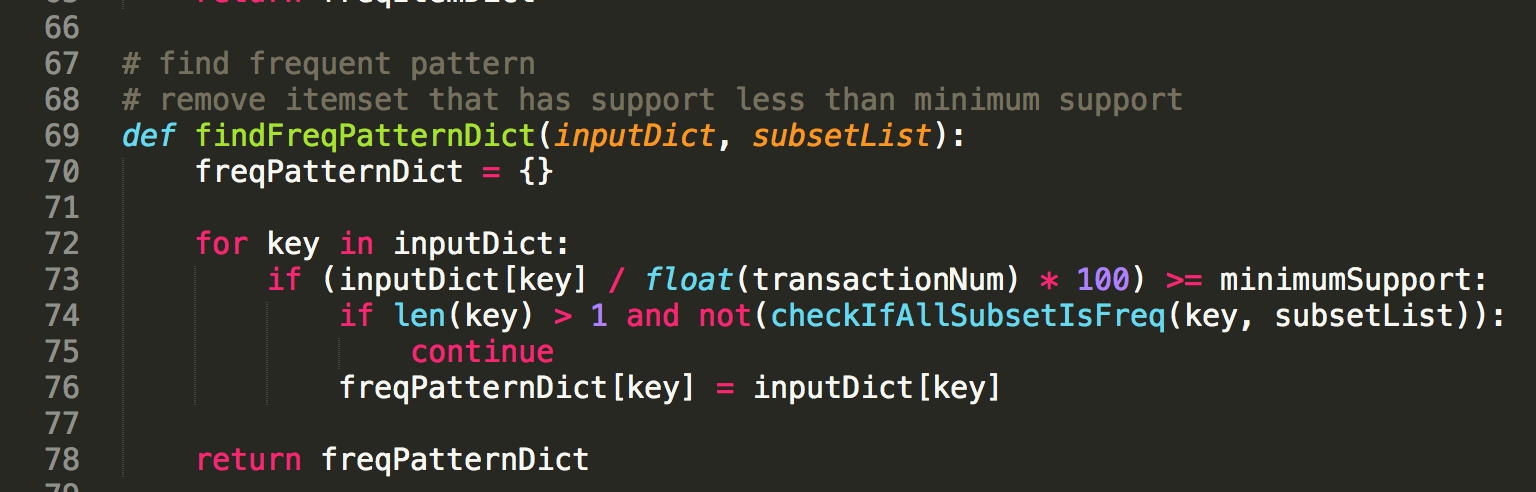
I will show the next depth's function of this.

### **h. Dictionary for length 1 candidate patterns**



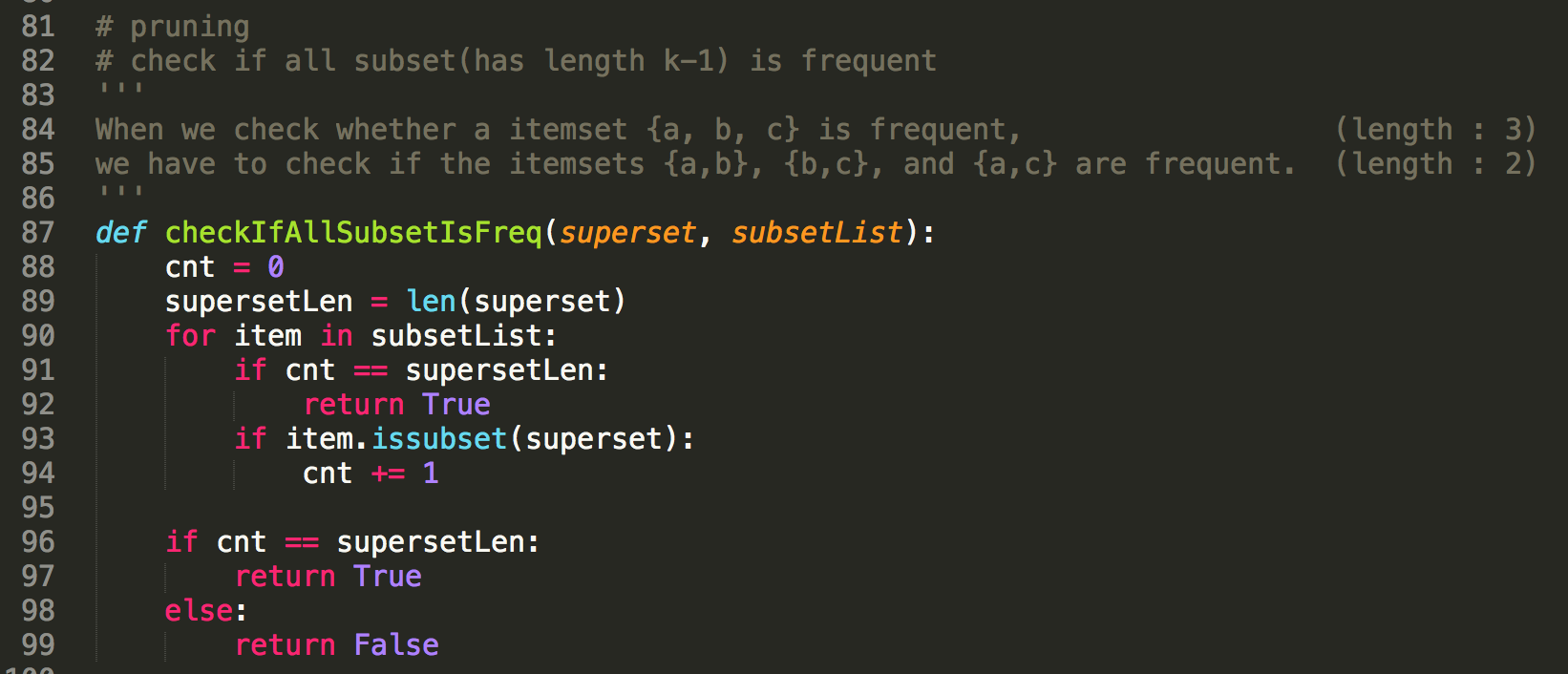
This method is for getting 1 length candidate patterns. It just gets numbers one by one from the transactions. It will use the sets for the key of support value dictionary. But key of python's dictionary is should be hashable. So, it had to make 'frozenset' instead of 'set'. If there is no key that is same with item set, it means the item set appears at first. So, in that case, it initializes value to 1. Otherwise, it just adds 1 to the value.

### **i. Finding dictionary of frequent patterns**



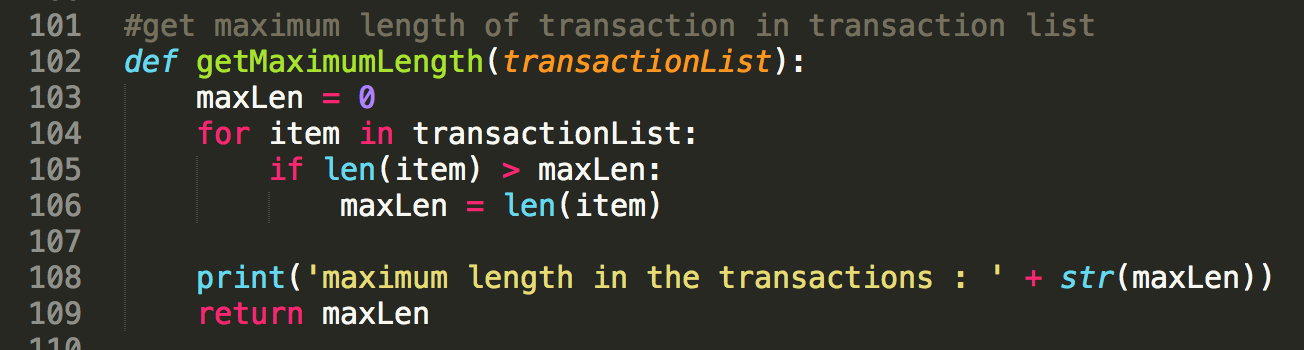
This method checks if the candidate patterns are frequent or not by comparing support with the minimum support, and pruning. ‘checkIfAllSubsetIsFreq’ method will do the ‘pruning’.

### **j. Pruning**



Unless all the subsets of the candidate pattern, the candidate pattern is not frequent. But when the length is N-1, we checked that rule with N-2 subsets! By iteration, when the length is N, we should check only N-1 length subsets. And, the number of N-1 length subset of N length set is N. (ex. {1,2,3}’s subsets that have length 2 are {1,2}, {2,3}, {1,3}.)

So, it scanned just the N-1 length frequent patterns, and counted the subsets of the superset. If the count value is same with the length of superset N, that superset is frequent. If not, it is not frequent.



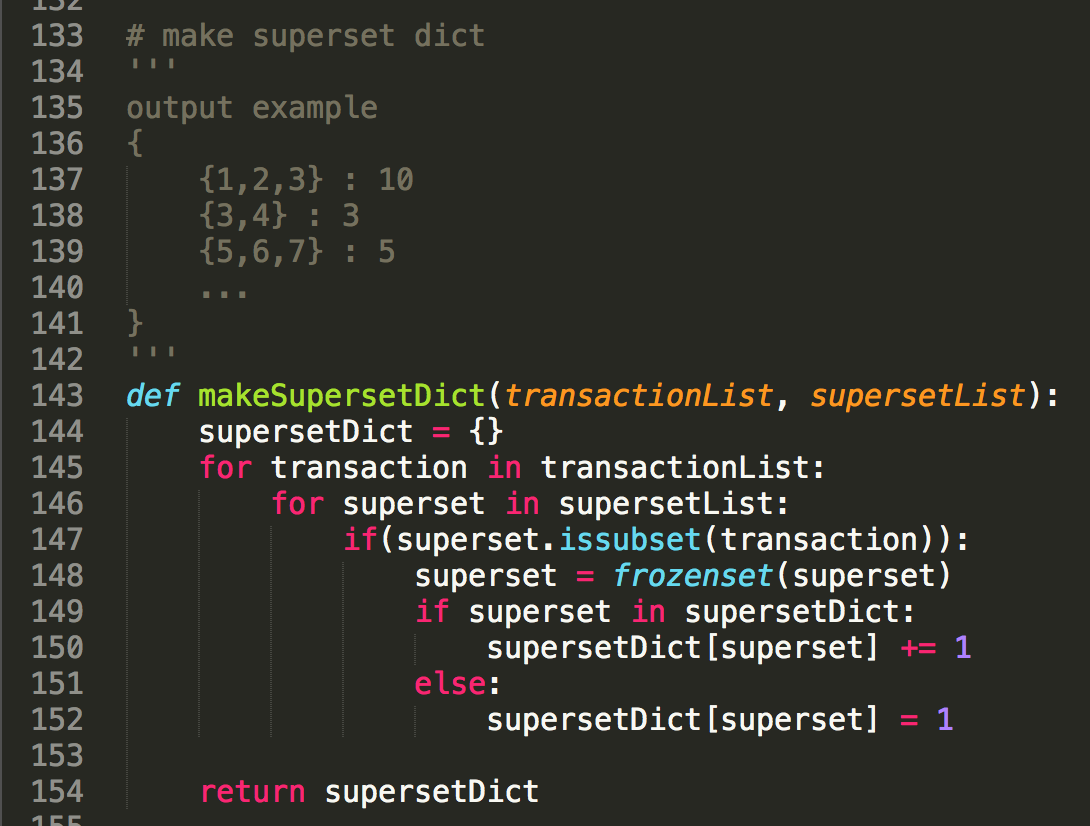
This method is for getting maximum length in the transaction list. That value will be used as a limited number in pruning method.

### **k. superset list**



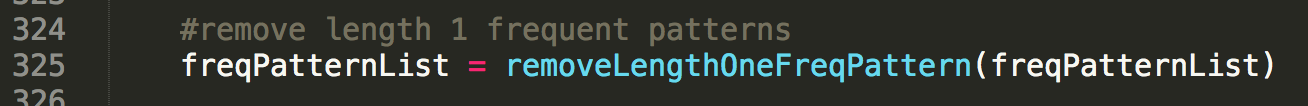
So we've found length 1 frequent patterns. Next, we have to find length 2,3,4... frequent patterns. I called this candidate list 'superset list'. It will find union of all of length N-1 frequent patterns' combinations. And if the length of the union is N, they will be put into the superset list.

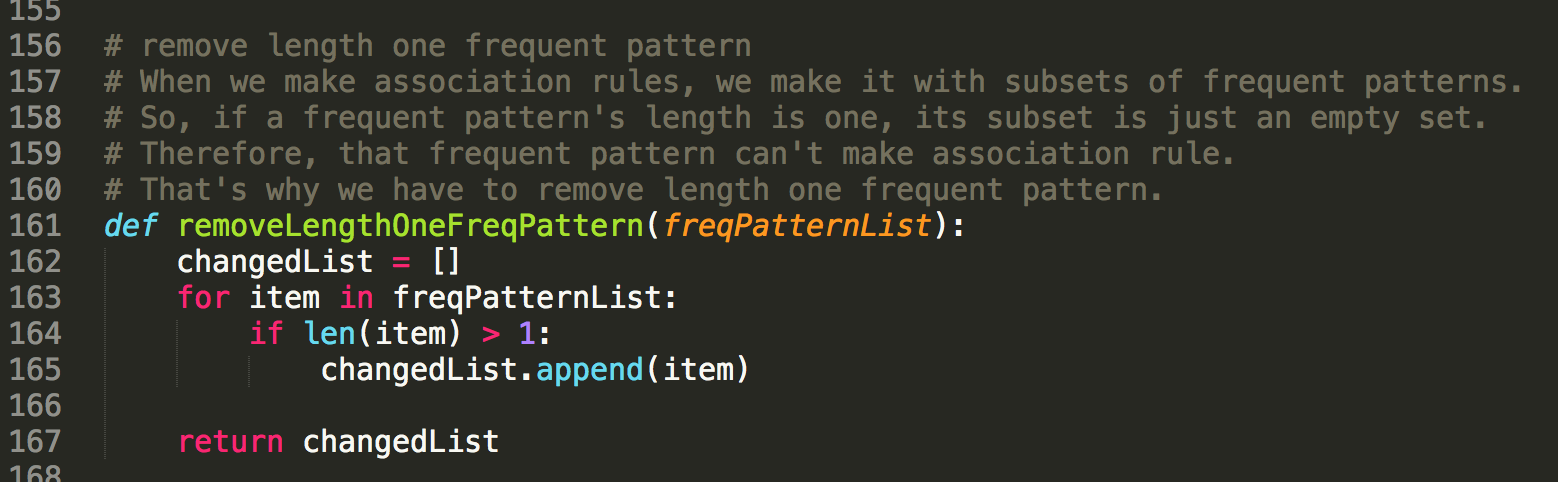
### **l. superset dictionary**



With the candidate patterns, we can make the dictionary for the support value. Scanning the transactions, it will just count the candidate pattern in the transactions. And this will be used to find the frequent patterns.

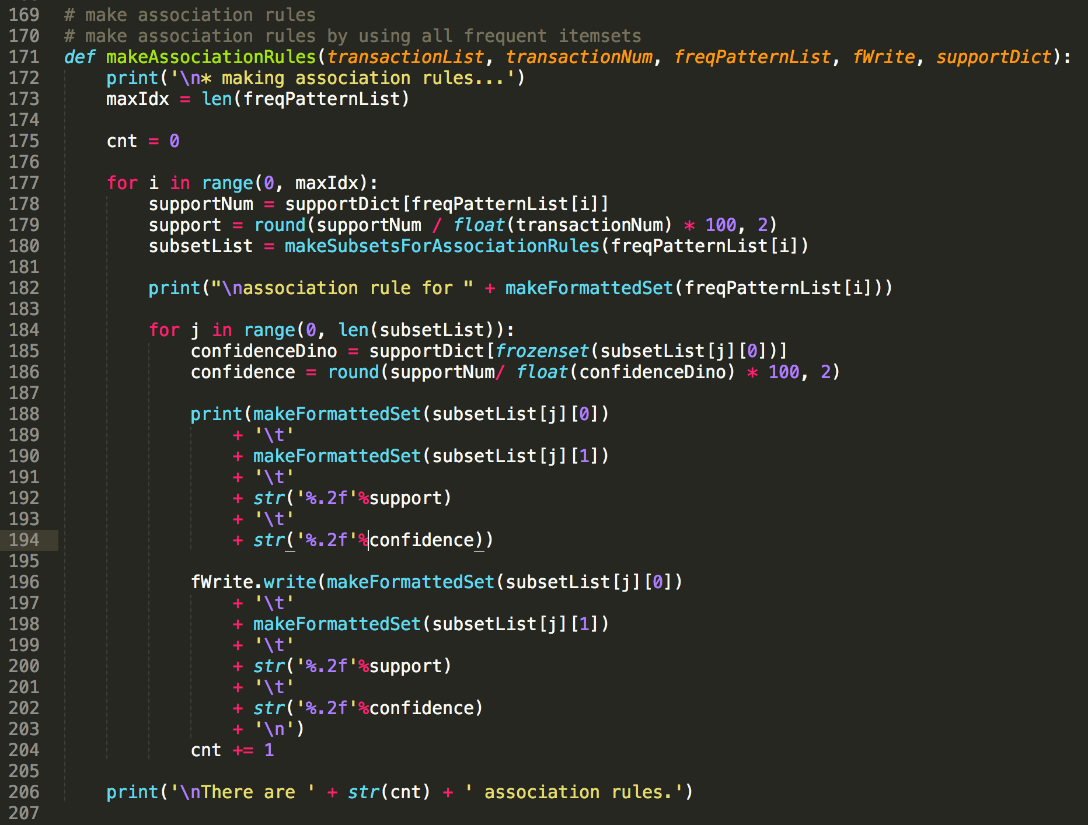
### **m. removing length 1 frequent patterns**





Before making association rules, we have to remove length 1 Frequent patterns. Because length 1 frequent patterns don't have any subsets except empty set. So, they cannot make any association rules. After this, we'll get the frequent pattern list without length 1 frequent patterns.

### **n. making association rules**



Finally, It's time to make association rules.

This method has a for statement by frequent patterns. When we count the support and the confidence, we need the support number of that frequent pattern : n(A U B).

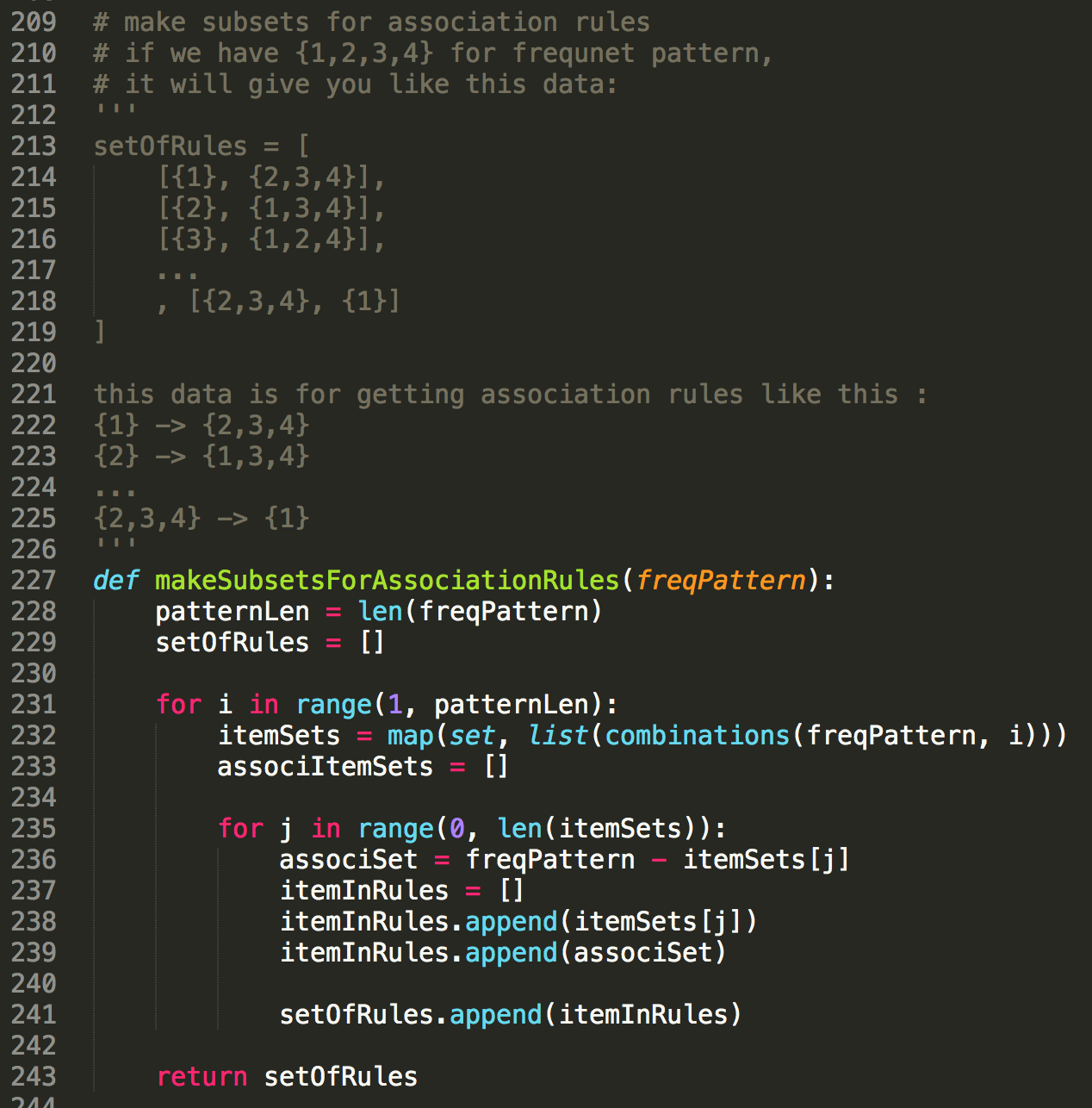
But we don't need to scan the DB again! Because we have all of the support number of the frequent patterns.

Calculating support is very easy. Just divide support number by the transactions' number. And then, we have to make conditions for association rules.

After finding the conditions, we have to count the condition. Like the preceding, you don't need to scan the DB for counting the condition. Because we have all of the support number of the frequent patterns in 'supportDict'.

After calculating support and confidence, it prints the conditions, support, and confidence and write them into output file. I used % formatted printing for zero filling until second floating point number.

### **o. making subsets for association rules**



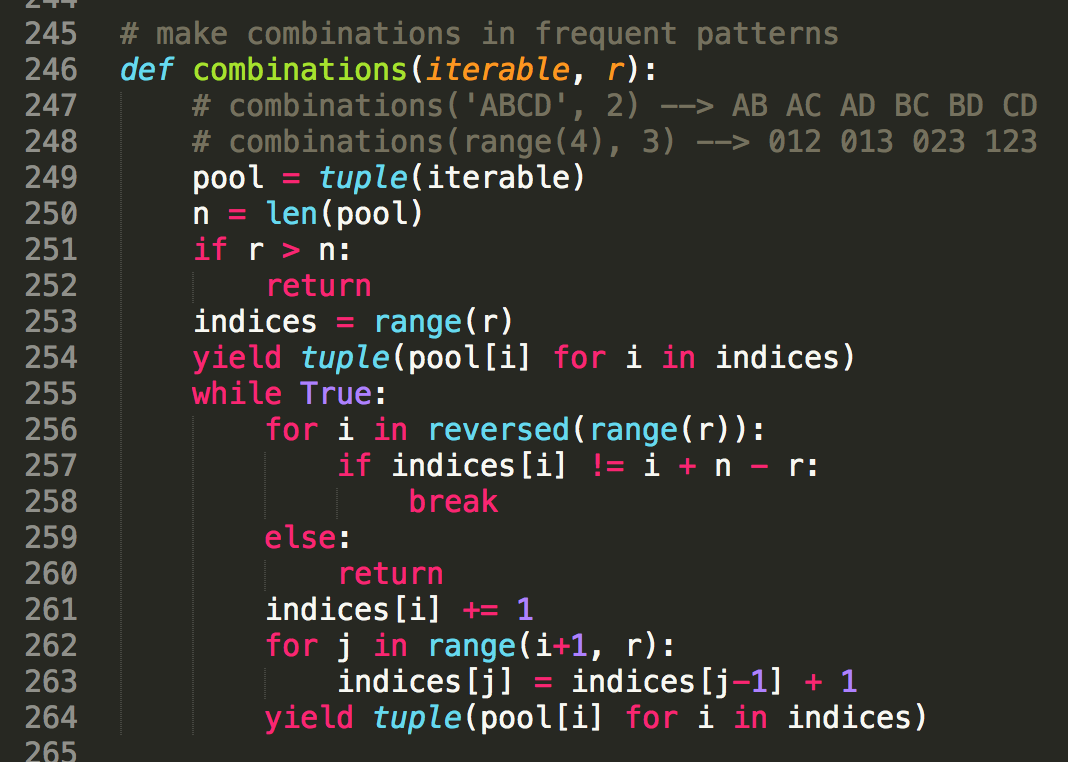
This method is to make subsets for association rules. I used 'combinations' method for get all the combinations of subsets.

For example, in for statement, if you get the frequent pattern {1,2,3}. In that case, you can get combinations like this :

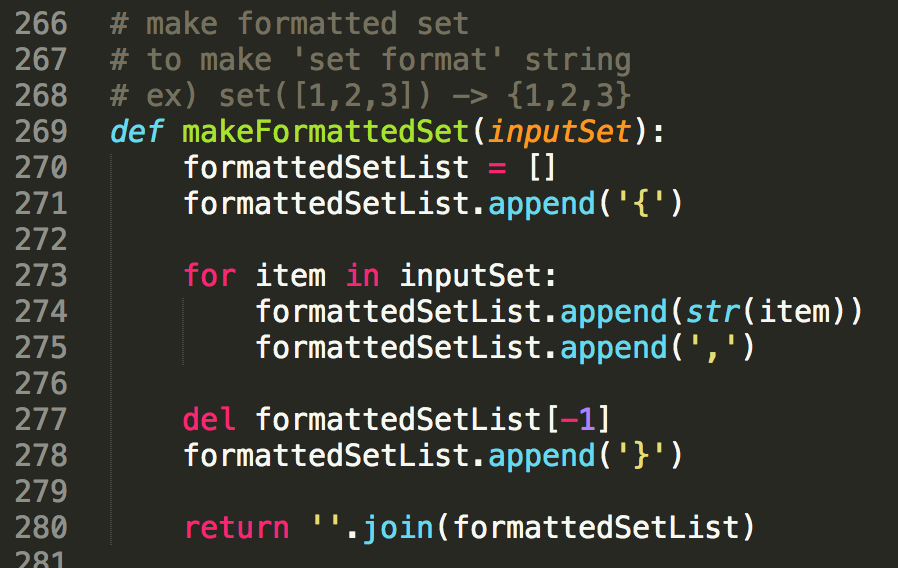
{1}, {2}, {3}, {1,2}, {1,3} {2,3}

And if you calculate [frequent pattern] - [combination], you can get the remainder in the frequent pattern set. So, you can make an association rule this. [combination] -> [reminder]

### **p. making combinations**

 This is the combination method. When you gives it iterable something and the combination number, you can get all of the combinations of it.

### **q. printing in set format**

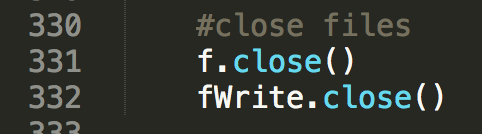


In this homework, we have to print the set like this format {1,2,3}. But in python, when I use print(mySet), it will show like set([1,2,3]).

So I needed a method to print set in set format.

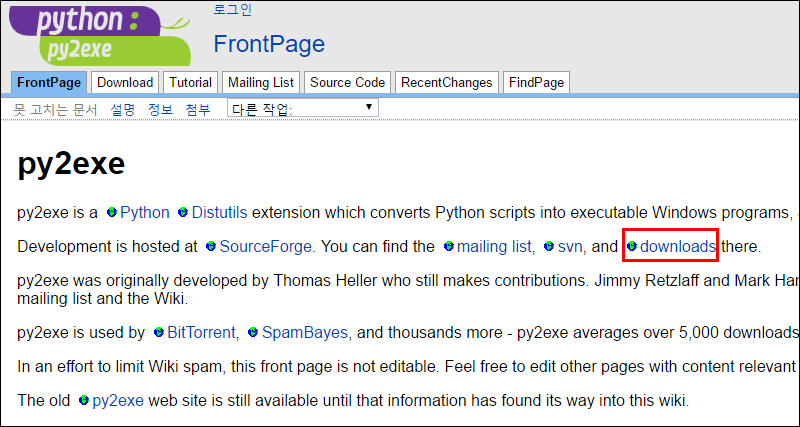
I just put the set items and '{','}',',' int to the list. At last, I joined the list things to string.

### **r. closing files**



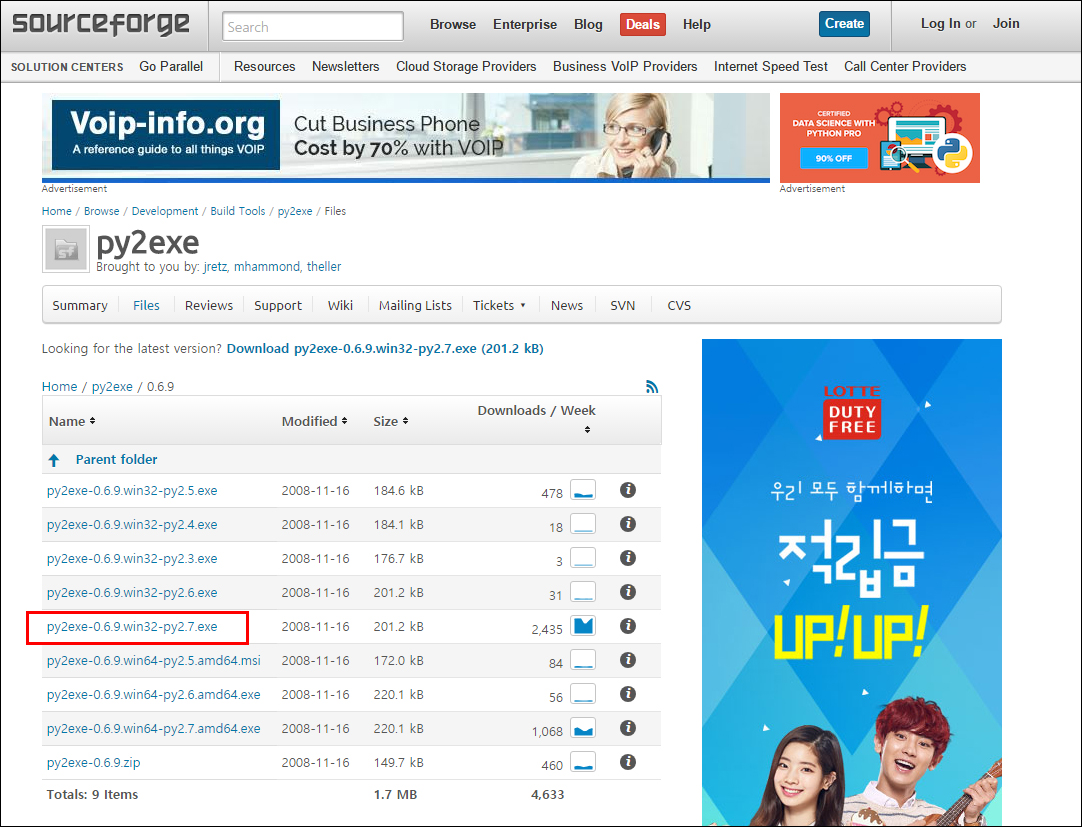
At the end of the program, it closes the files.

# **4. Instructions for building exe**

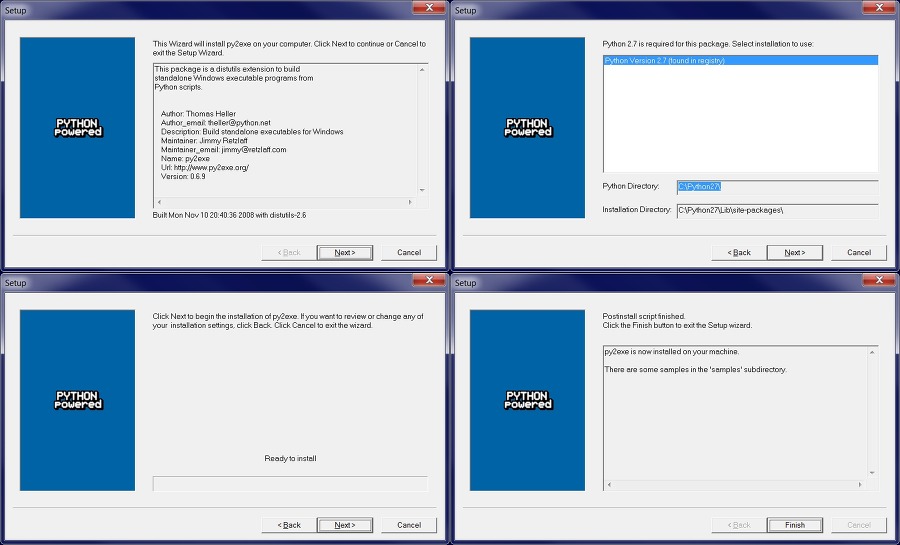


I used py2exe to make the exe file with python. So if you want to build exe file, you have to install py2exe first.

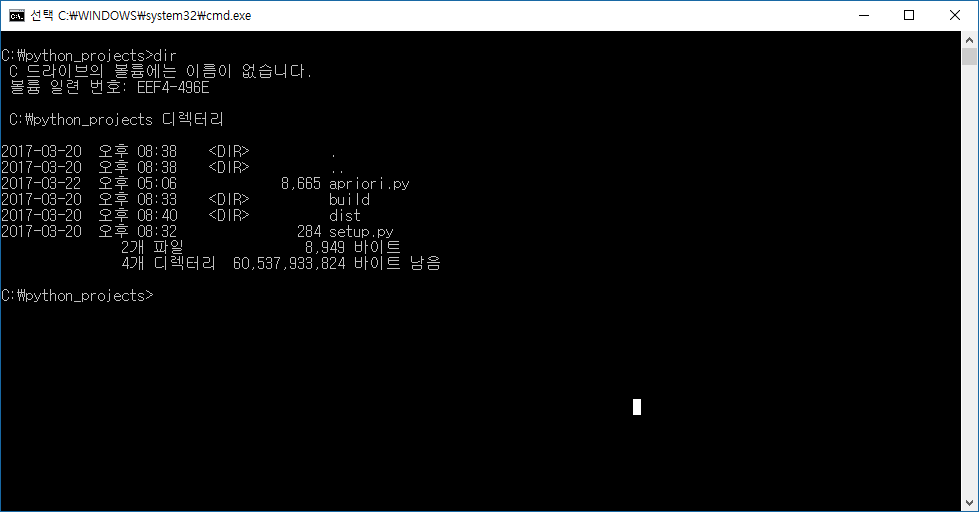
First, please connect to <http://py2exe.org> , and click downloads.



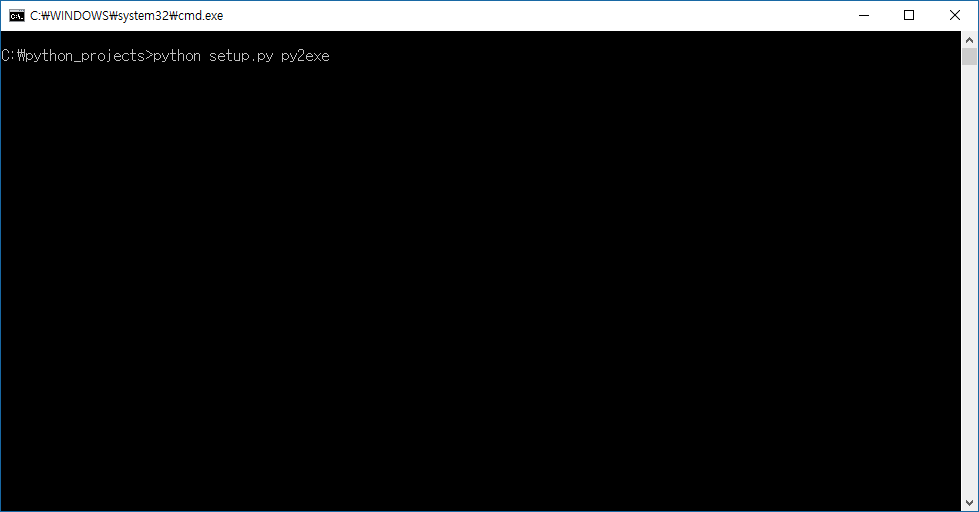
My python version is 2.7.12. So I installed py2exe-0.6.9.win32-py2.7.exe.



You can install it easily by clicking ‘next’ button.

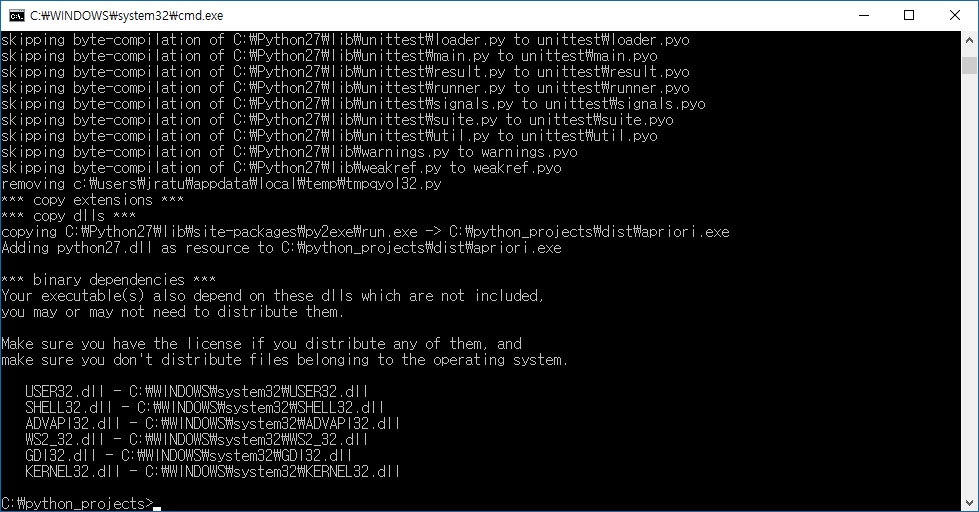


After installing py2exe, turn on the command prompt. And move to directory with apriori.py and setup.py. I uploaded setup.py on the git lab. So, you can use it like a makefile.

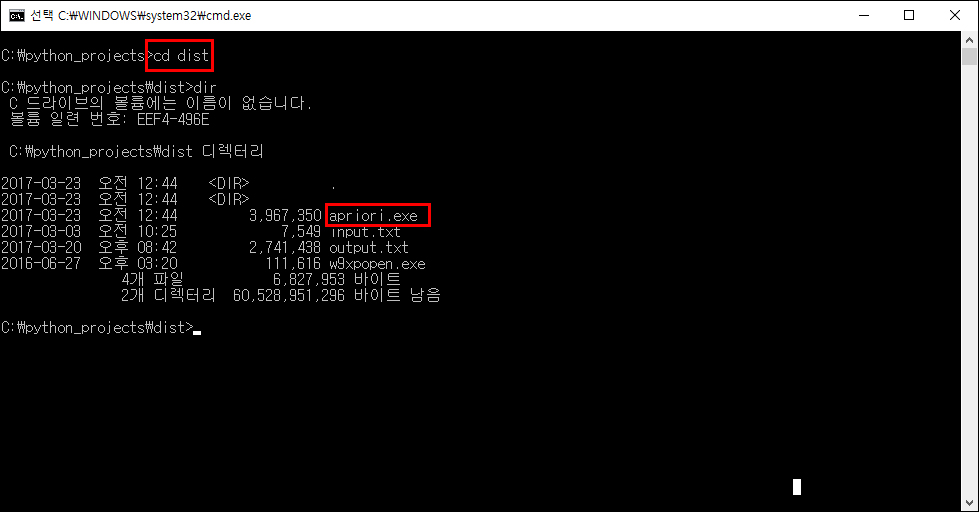


And then, just insert this command

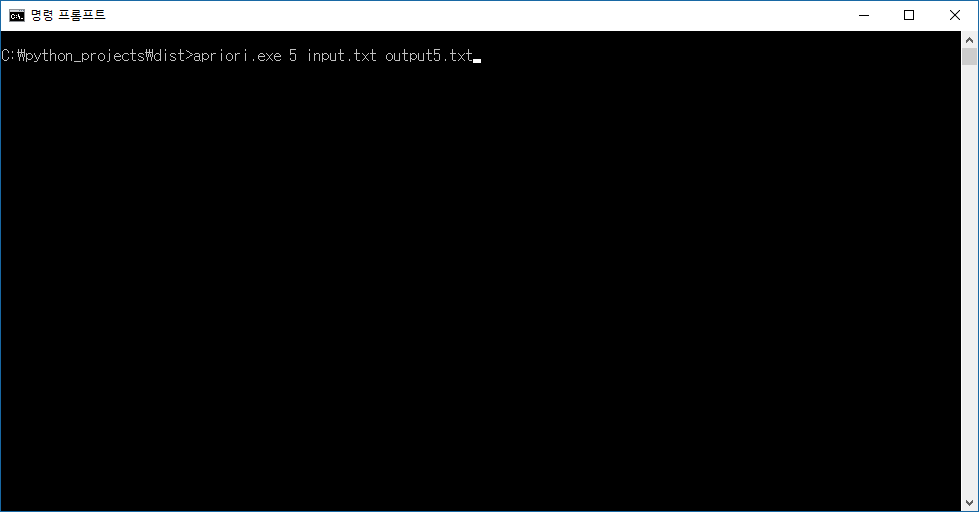
$ python setup.py py2exe



It’s done!

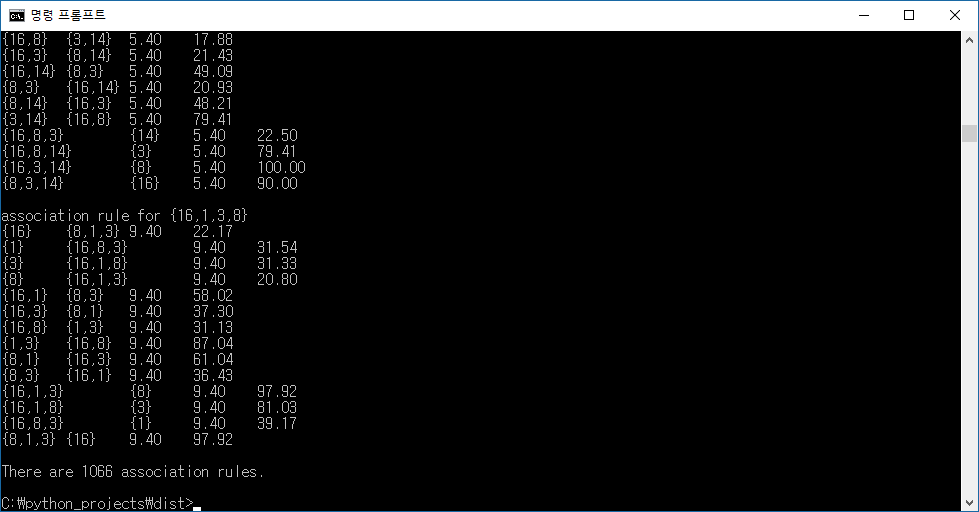


Moving to the ‘dist’ directory, you can notice the apriori.exe file.

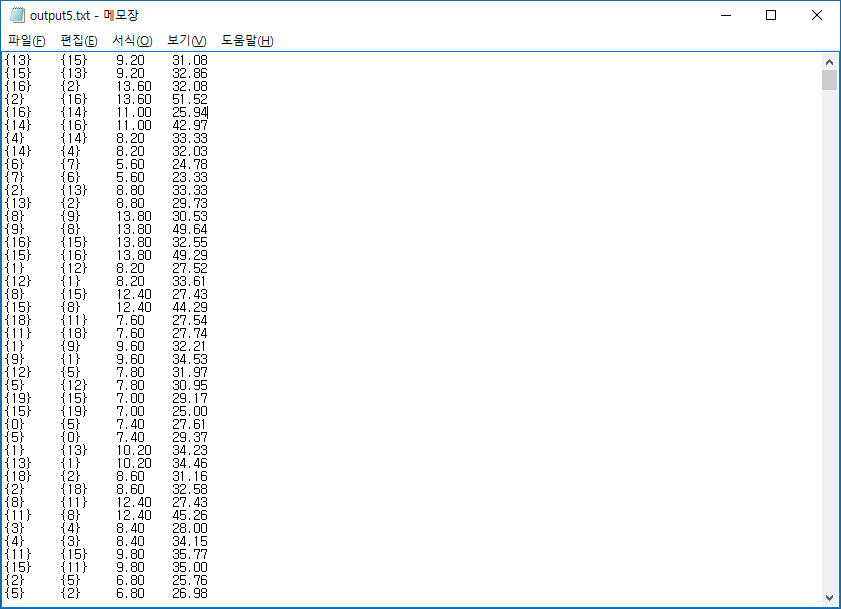


You can execute the file by this command.

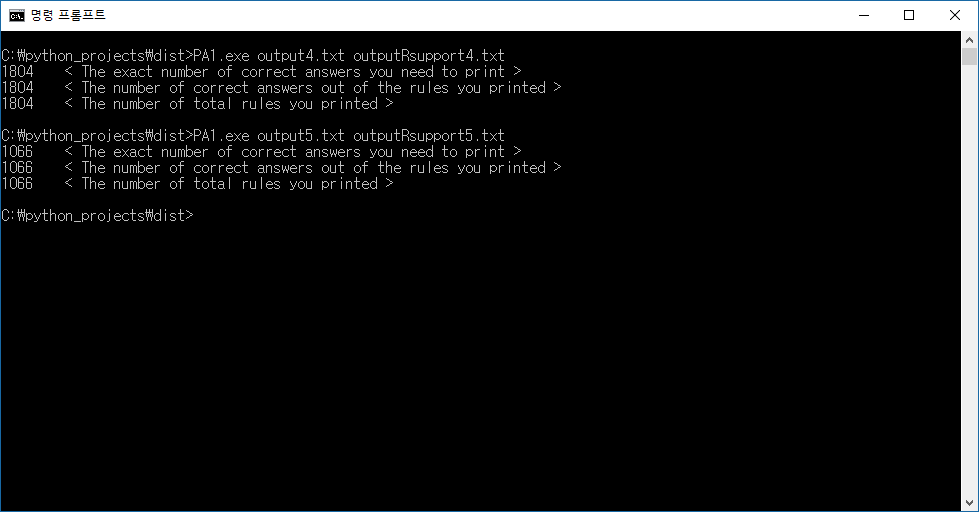
$ apriori.exe [minimum\_support] [input\_file\_name] [output\_file\_name]



The answer will be printed on the command prompt.

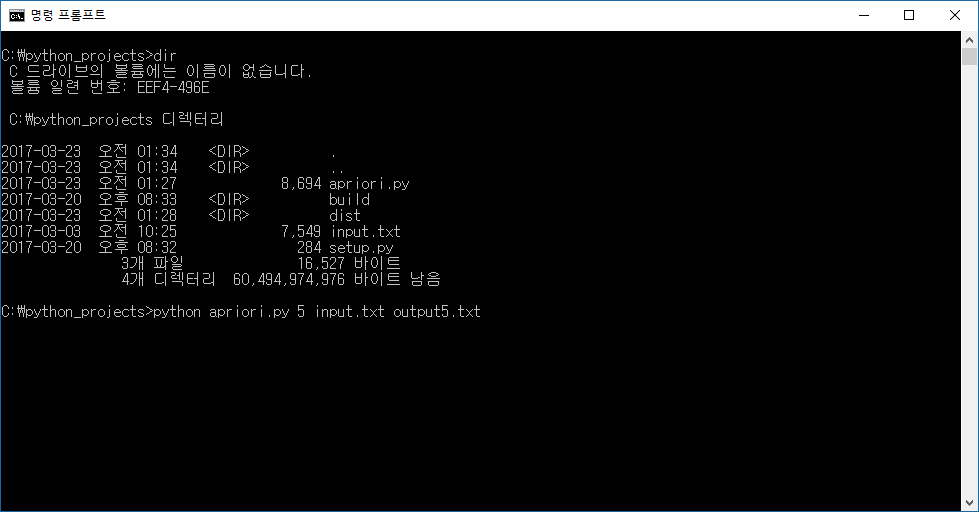


And you can check it in the output file as well.



Finally, my work passed the testing process.

# 5. Testing without exe file



If you don’t want to install py2exe, or your environment can’t execute exe file (ex. OS X), you can easily test it by python command. Move to the directory with apriori.py and input.txt, and just put this command.

$ python apriori.py [minimum\_support] [input\_file\_name] [output\_file\_name]