Student Information
Name - Amber Sautner
UCID - As4393
Email - As4393@njit.edu
NJIT-ID 31609002

Apriori Algorithm vs. Brute Force Method

Project Overview

This project is designed to compare two popular association mining techniques: Apriori algorithm and Brute Force method. Although both algorithms are used to find frequent itemsets in a dataset, through this project we will see the differences in efficiency and scalability between the two. Throughout the files, certain functions may overlap in both methods and could be removed for optimization, but to better explain the difference between Apriori vs Brute I created separate files. These files include count.py, powerset.py, and association.py. All functions will be colored in green and variables will be colored in red.

Apriori Algorithm

Apriori Algorithm Overview

The Apriori algorithm uses a 'bottom-up' approach in discovering frequent itemsets and their corresponding association rules from a dataset. It is done by generating k itemsets and determining their frequencies, all while pruning out any infrequent itemsets. Based off the idea that all subsets of an infrequent itemset will also be infrequent, the algorithm will discard the unnecessary data and only continue with frequent itemsets. Therefore, this method is known to be efficient and scalable for larger datasets.

Code Analysis

To begin, we will investigate the steps in creating the Apriori Algorithm. Within each file, there will be a general overview, the documentation, as well as source code for final clarification.

Load_apriori_data.py

General Overview

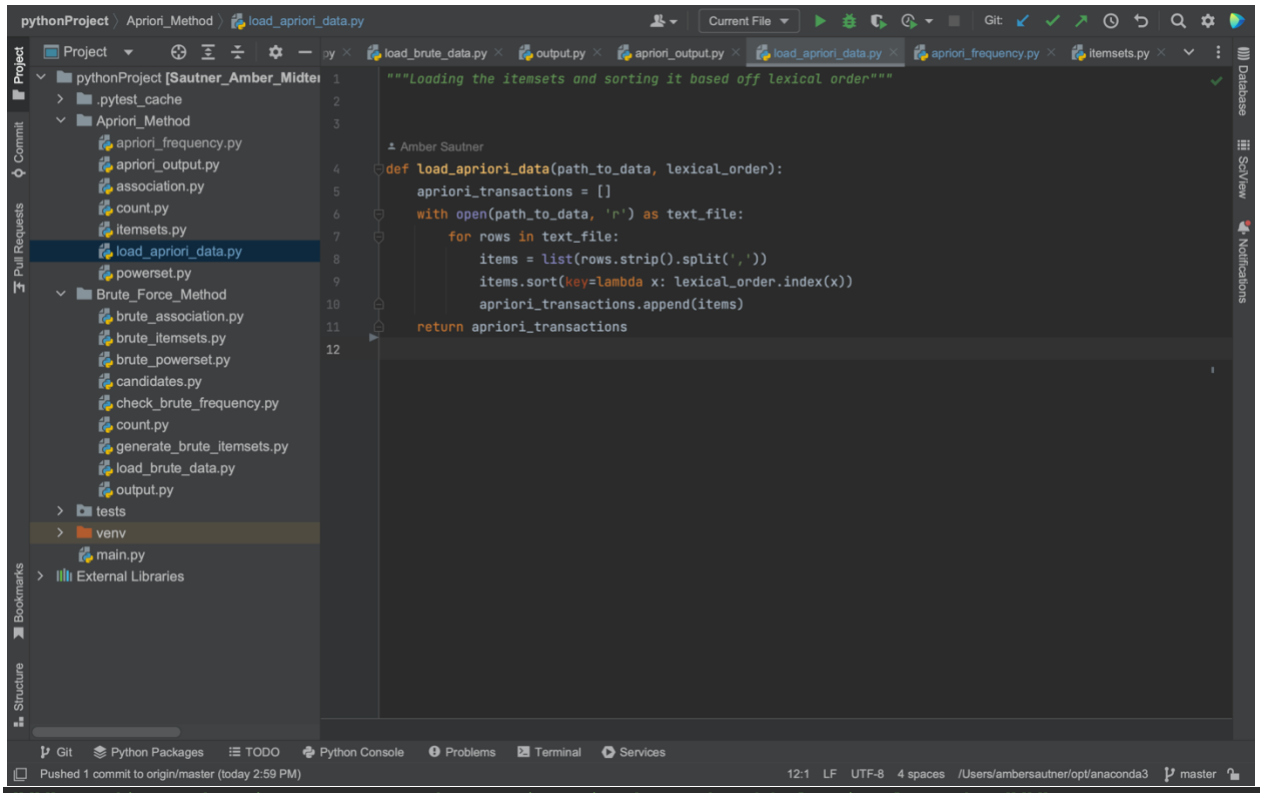
This file is the first step in the Apriori method. It will define the function load_apriori_data that will take in a list of unordered, un-cleaned data and return a newly processed list with the transactional data now in lexical order.

Documentation

This function takes two parameters, path_to_data and lexical_order. It starts by initializing an empty list for the future transactions to be added to called apriori_transactions. (Line 5) Once the file containing the data is opened in read-only mode, a 'for loop' will begin to iterate through

each line to create a new list that contains only the product names separated by a comma. (Line 7-8) This was done to create a more uniformed, easily understood list of data called items.
A key is also used to sort the items so it will be in lexical order. (Line 9) Finally, we append the transformed data to the originally empty apriori_transactions list to update it with the newly processed items. (Line 11)

Source Code




```
"""Loading the itemsets and sorting it based off lexical order"""


def load_apriori_data(path_to_data, lexical_order):
    apriori_transactions = []
    with open(path_to_data, 'r') as text_file:
        for rows in text_file:
            items = list(rows.strip().split(','))
            items.sort(key=lambda x: lexical_order.index(x))
            apriori_transactions.append(items)
    return apriori_transactions
```

Apriori_frequency.py

General Overview

In the last file, we created a function to open our datafile and create an ordered list with the transactional data. Next, we will focus on finding the frequent itemsets from the transactional data and returning those itemsets, their support count, and the newly discarded itemsets that were infrequent. It is done by generating itemsets of increasing size and then pruning them based off their support value. It terminates when no more frequent itemsets are found.

Documentation

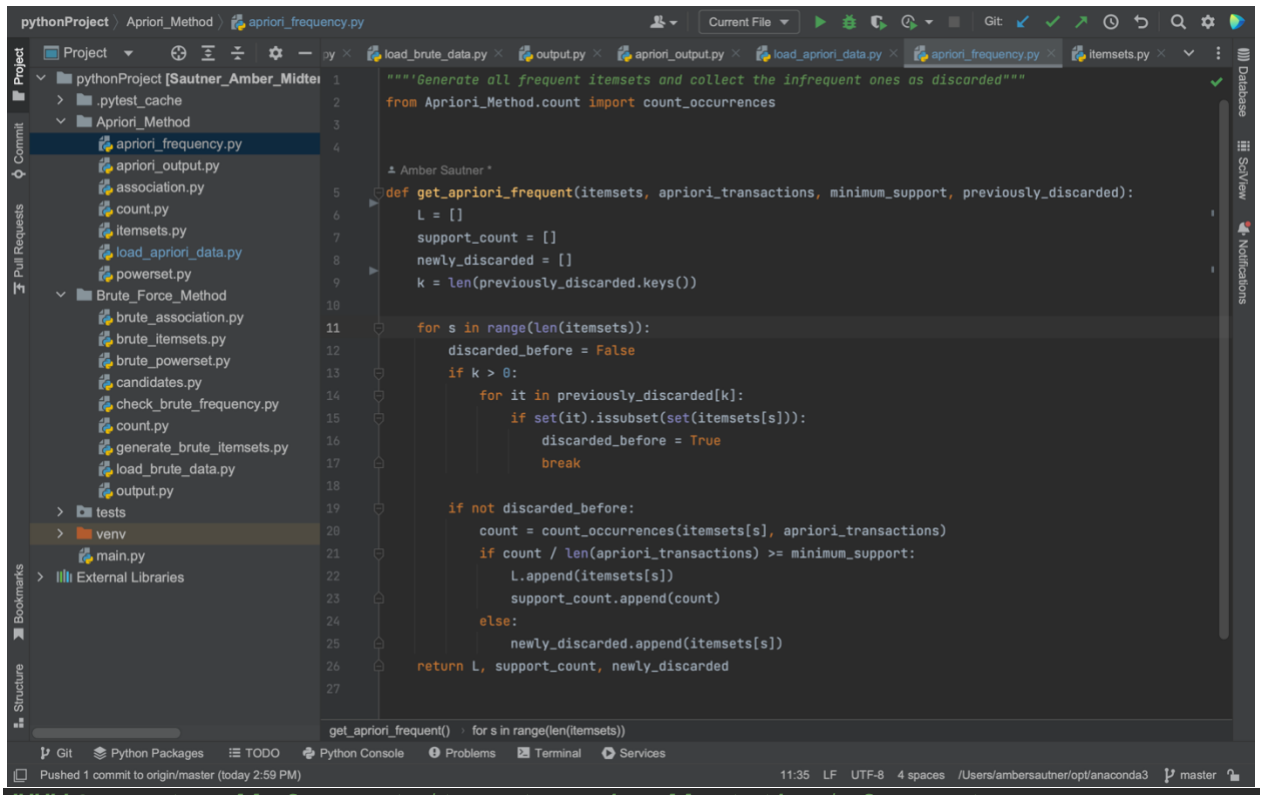
The get_apriori_frequency will load 4 parameters: itemsets, apriori_transactions, minimum_support, and previously_discarded. To begin, we must initialize an empty list to hold the frequent itemsets once we find them, which is L = []. (Line 6) We do the same for the support_count, which is how many times the itemsets occurs, and newly_discarded, which are the pruned itemsets from the iteration. (Lines 7-8) We also set k as the length of the previously discarded itemsets. The len() function will get the number of itemsets that were pruned in the previous iteration, which is assigned to k in order to determine the size of the itemsets to b generated in the current iteration.

In the Apriori algorithm, with each iteration we must keep track of the discarded itemsets because once an itemset is deemed infrequent, any subsets are classified as also infrequent. Therefore, we will not need to check those subsets. We see this process in the first 'for loop' where we iterate through each itemset and check if we have any previously discarded itemsets or not. (Line 11-12) In this loop, it is logical to first check if any of the itemsets we are looking at are previously discarded, because then we do not need to waste computation time by continuing. For clarification, we do this because the Apriori algorithm is based off the idea that if the itemset is a subset of the current subset/itemset, then it is infrequent. Finally, we break because we do not need to waste CPU when we know it will be infrequent.

The second part of the 'for loop' tells us that the itemset was not discarded before, and therefore is a frequent subset. (Line 19) The count, which represents the number of times the itemset appeared in the total transactions, begins to be counted through the count_occurrences function. Furthermore, an 'if statement' is added to state that if the itemsets frequency meets the requirements of the minimum support then it will be appended to L. (Lines 21-23) Otherwise it will be discarded. (Line 25)

Finally the updated list with the frequent itemsets is returned as L, along with their respective support counts as support_count, and the newly discarded itemsets to keep track of as newly_discarded. (Line 26)

Source Code




```python
"""'Generate all frequent itemsets and collect the infrequent ones as
discarded"""
from Apriori_Method.count import count_occurrences


def get_apriori_frequent(itemsets, apriori_transactions, minimum_support,
previously_discarded):
    L = []
    support_count = []
    newly_discarded = []
    k = len(previously_discarded.keys())

    for s in range(len(itemsets)):
        discarded_before = False
        if k > 0:
            for it in previously_discarded[k]:
                if set(it).issubset(set(itemsets[s])):
                    discarded_before = True
                    break

        if not discarded_before:
            count = count_occurrences(itemsets[s], apriori_transactions)
            if count / len(apriori_transactions) >= minimum_support:
                L.append(itemsets[s])
                support_count.append(count)
            else:
                newly_discarded.append(itemsets[s])
    return L, support_count, newly_discarded
```

Itemsets.py

General Overview

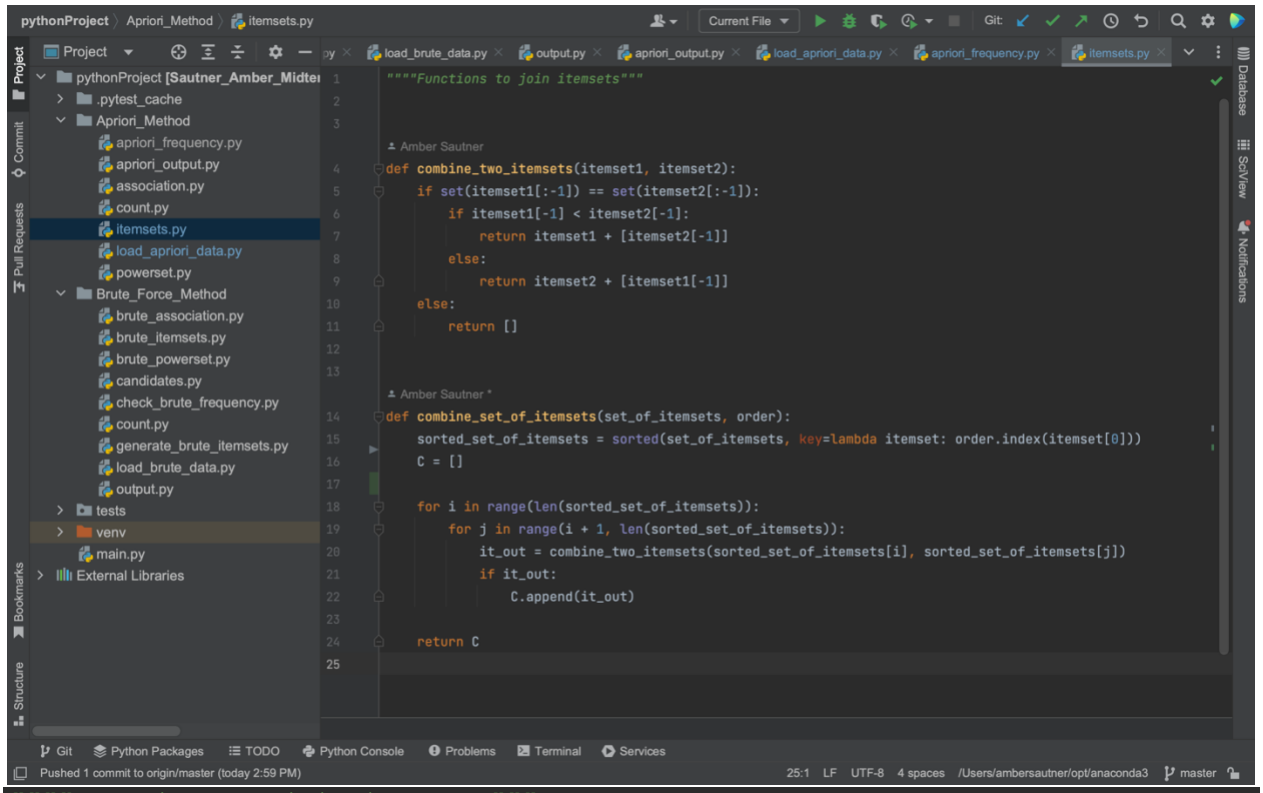
After creating functions to load the data and check their frequencies, the next step would be to join itemsets together as we increase in itemset size. The following two functions are an integral step in the Apriori process because they will combine itemsets in a set of itemsets.

Documentation

The function combine_two_itemsets is designed to take two parameters, itemset1 and itemset2, and check if they are join-able. They are deemed joinable if all elements are the same except for the last one since we are discarding that one. The first 'if statement' will check to see if they meet this requirement by stating if itemset1's last element is equal to itemset2's last element, then join them. (Line 5) To join them, we add an 'if…else' statement to append the last element of the larger itemset to the smaller one. (Line 6-9) If the sets are not joinable, the 'else' will return an empty list. (Line 11)

The second function combine_set_of_itemsets takes two parameters, set_of_itemsets and order, because it will return a new set of itemsets that contains all the possible combinations from those two itemsets in lexical order. It will start by sorting the itemsets by their first element, since this leads to faster lookups and optimizes the code. (Line 15) Next, we create an empty list, C, which will later hold the generated combinations. The 'for loop' will iterate through the length of the set_of_itemsets that were just sorted with the order. (Line 18) The variable i represents the index that is being iterated over, which is why we previously had sorted the itemsets by order.index(itemset[0]). Next, it will iterate over all pairs of itemsets in the given sorted set starting after i. (Line 19) By calling the combine_two_itemsets function, we check if the two itemsets are join-able and if so, they are returned as it_out and then appended to the empty list C. (Line 20-22) Finally C is returned with all the possible combinations of join-able itemsets from the input set. (Line 24)

Source Code




```
"""Functions to join itemsets"""


def combine_two_itemsets(itemset1, itemset2):
    if set(itemset1[:-1]) == set(itemset2[:-1]):
        if itemset1[-1] < itemset2[-1]:
            return itemset1 + [itemset2[-1]]
        else:
            return itemset2 + [itemset1[-1]]
    else:
        return []


def combine_set_of_itemsets(set_of_itemsets, order):
    sorted_set_of_itemsets = sorted(set_of_itemsets, key=lambda itemset:
order.index(itemset[0]))
    C = []

    for i in range(len(sorted_set_of_itemsets)):
        for j in range(i + 1, len(sorted_set_of_itemsets)):
            it_out = combine_two_itemsets(sorted_set_of_itemsets[i],
sorted_set_of_itemsets[j])
            if it_out:
                C.append(it_out)

    return C
```

Powerset.py

General Overview

This file contains 3 functions that serve in place of the itertool package. They are repeated in the Brute Force method.

Documentation

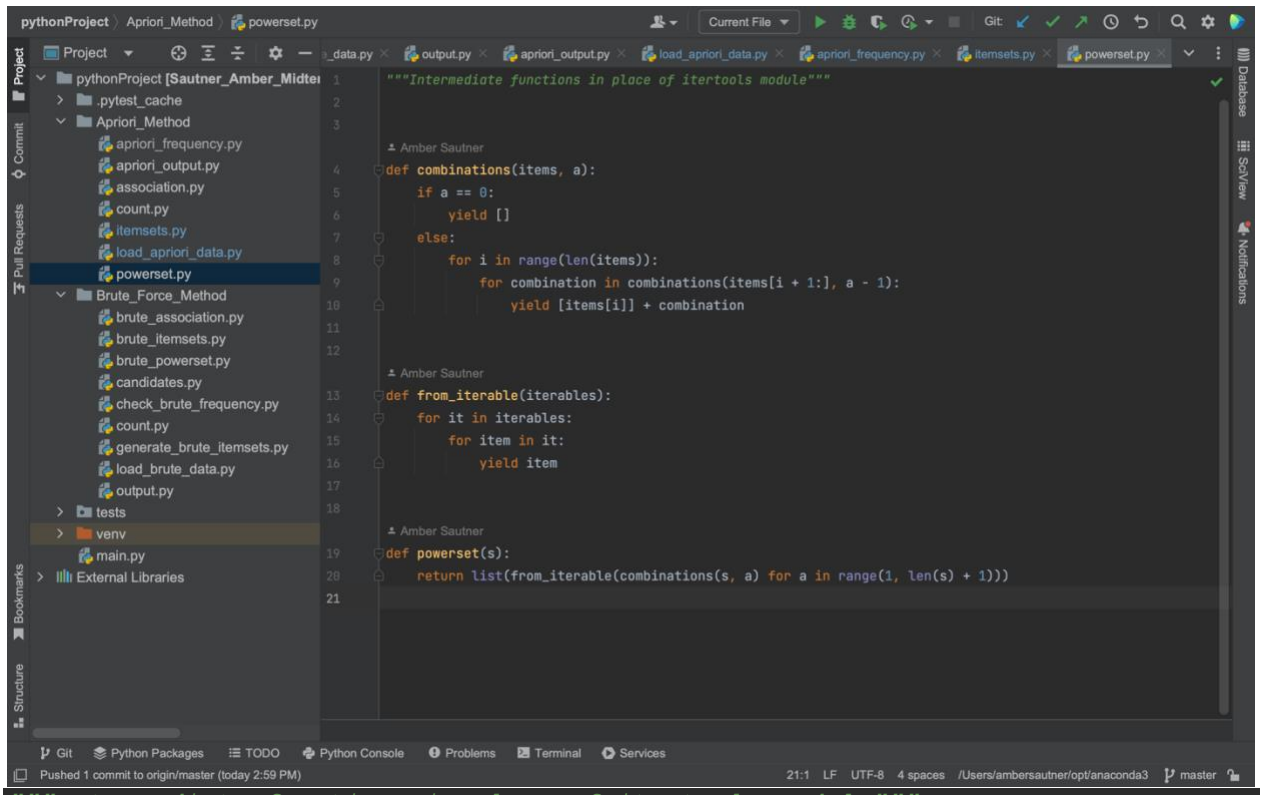
The first function, combinations, is designed to generate all possible combinations of 'a' items from the 'items' list. It will take in two parameters, items and a, and uses recursion to find all possible combinations of length 'a' from the 'items' list. The Apriori Algorithm will create all possible combinations of the transactional data to find which itemsets are frequent, which is why this function is vital. The function will first set a equal to 0 and then follow a simple 'if…else' statement.

The 'if' snippet will show that if a is 0, the function yields an empty list. This will represent the instance where we want to find all combinations of length 0, which is just the empty combination. In the 'else' snippet, a 'for loop' is created to iterate over the items list. At each loop, the code will start another 'for loop' to go through each item and recursively find all combinations at length 'a-1' from the remaining items list. (Line 9) For each combination found, the function will yield a new combination with the current item at the front of the list. By doing this, it will exhaust all possible combinations with all items.

The second function, from_iterable, is designed to take the parameter iterables and iterate over each iterable to yield the items. The 'for loop' will iterate over the 'iterables' parameter and assign each iterable to the variable 'it'. (Line 14) The second 'for loop' will iterate over each item in the 'it' and assign each item to the variable item. (Line 15) Finally, the items are yielded back one at a time with all the items in the original iterables. I created this function to flatten the nested lists instead of have large lists of data.

The third function, powerset, aims to generate all possible subsets of the given set 's'. The combinations function generates all combinations of s of length a for all a in the range from 1 to the length of s. The from_iterable function then turns this into a single sequence of all combinations of s, including the empty set. Finally, the entire sequence is converted to a list using the list() function and returned. (Line 20)

Source Code




```
"""Intermediate functions in place of itertools module"""


def combinations(items, a):
    if a == 0:
        yield []
    else:
        for i in range(len(items)):
            for combination in combinations(items[i + 1:], a - 1):
                yield [items[i]] + combination


def from_iterable(iterables):
    for it in iterables:
        for item in it:
            yield item


def powerset(s):
    return list(from_iterable(combinations(s, a) for a in range(1, len(s) +
1)))
```

Count.py

General Overview

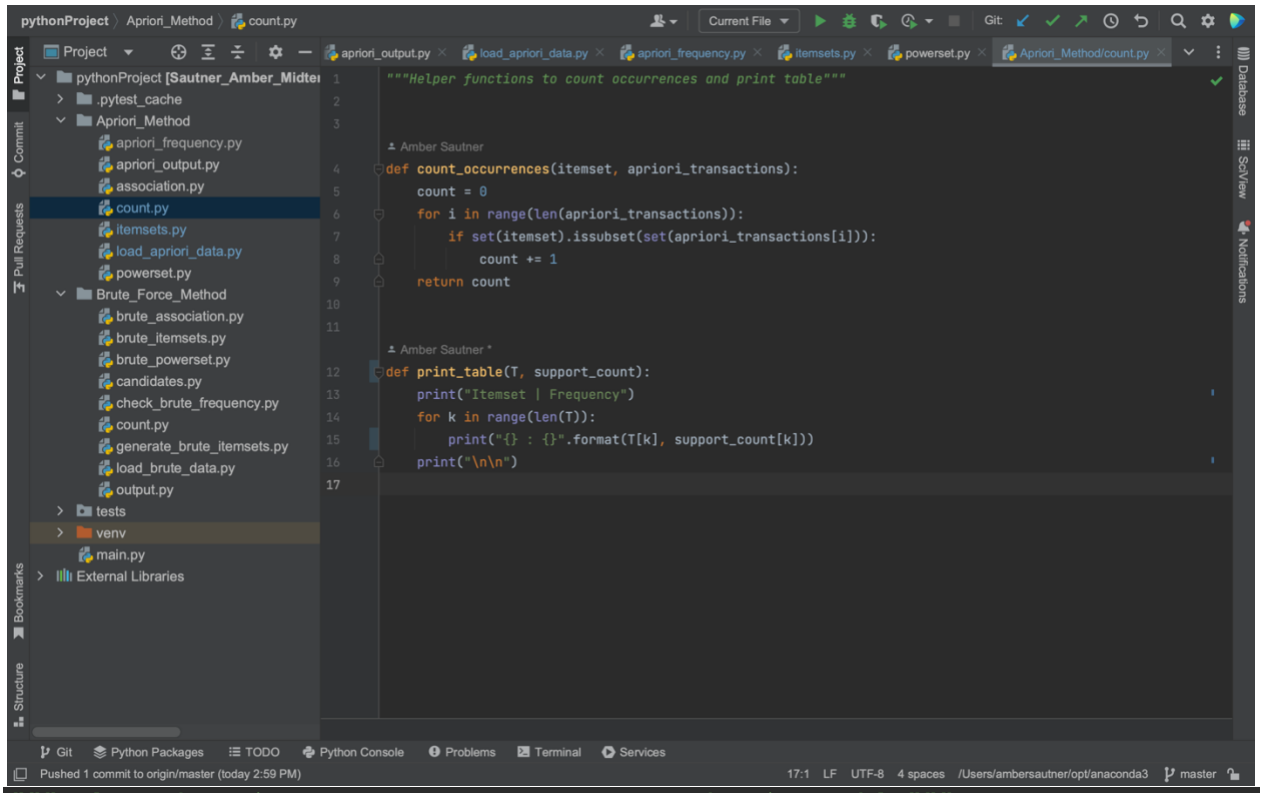
This file contains two helper functions that overlap in the Apriori Algorithm and Brute Force Method. The first function will count the number of times an itemset occurs in a transaction and the second will print a table with the itemset and its respective frequency.

Documentation

The first function, count_occurences, takes in two parameters, itemset and apriori_transactions, and initializes the variable 'count' to 0. The 'for loop' will iterate over the apriori_transactions to see how many times one itemset occurs in one transaction. (Line 6) We need to know this because depending on the minimum support, an itemset's count can either deem it frequent or infrequent. The following 'if statement' confirms that if an item is a subset of the set we are checking, then we will increment the count by 1 and return the count. (Line 7-9) This count is the number of times the itemset appears in the transactions.

The second function, print_table, takes in two parameters, T and support_count, to print a neat table comparing the itemset with its respective frequency. The 'for loop' goes through T, aka the list of itemsets, and their support count to print out the information as a formatted string.

Source Code




```
"""Helper functions to count occurrences and print table"""


def count_occurrences(itemset, apriori_transactions):
    count = 0
    for i in range(len(apriori_transactions)):
        if set(itemset).issubset(set(apriori_transactions[i])):
            count += 1
    return count


def print_table(T, support_count):
    print("Itemset | Frequency")
    for k in range(len(T)):
```

```
        print("{} : {}".format(T[k], support_count[k]))
    print("\n\n")
```

Association.py

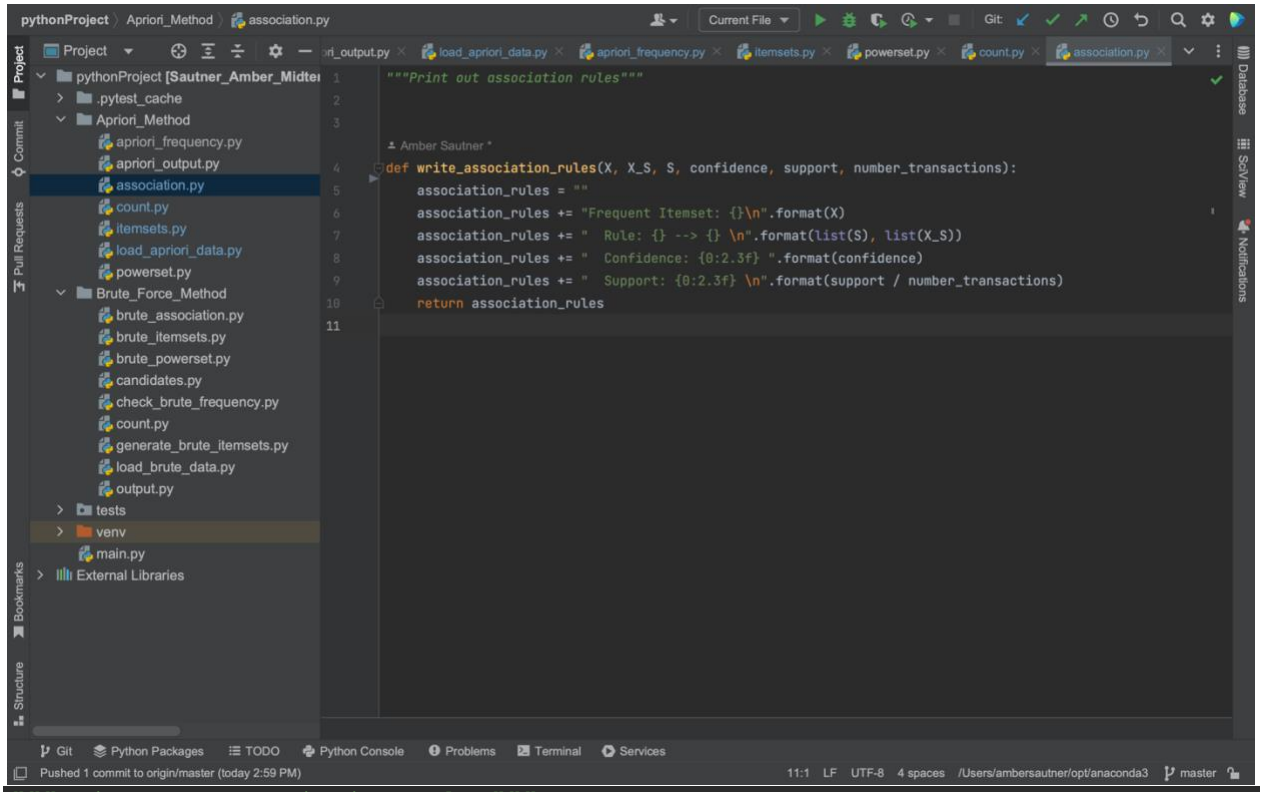
General Overview

This function was created to return the association rules as a formatted string that is easily understood by the user. It is the final file that overlaps with the Brute Force method.

Documentation

The function, write_association_rules, takes in 5 parameters; X, X_S, S, confidence, support, and number_transactions. Each line will generate the respective association rules between the itemset X and its subset X_S given the support count from itemset X, the support count from X_S, and the number_transactions. The association_rules variable is returned as a string with these 3 lines of information.

Source Code




```
"""Print out association rules"""


def write_association_rules(X, X_S, S, confidence, support,
number_transactions):
    association_rules = ""
    association_rules += "Frequent Itemset: {}\n".format(X)
    association_rules += "  Rule: {} --> {} \n".format(list(S), list(X_S))
    association_rules += "  Confidence: {0:2.3f} ".format(confidence)
    association_rules += "  Support: {0:2.3f} \n".format(support /
number_transactions)
    return association_rules
```

Apriori_output.py

General Overview

In this file we will outputting the association rules based off the frequent itemsets.

Documentation

The first step in this file is to set up some of our key variables. We will create a user input variable for minimum_support and minimum_confidence. Next, we will define the path_to_data, order, Apriori_transactions, and number_transactions. After this, we initialize 'C' as an empty dictionary to put all possible candidate itemsets in. The itemset_size is set to 1 and discarded is initialized to store infrequent itemsets that will be disregarded in future iterations. The final line initializes C with itemset_size = 1 and then updates it as an ordered list. (Line 26)

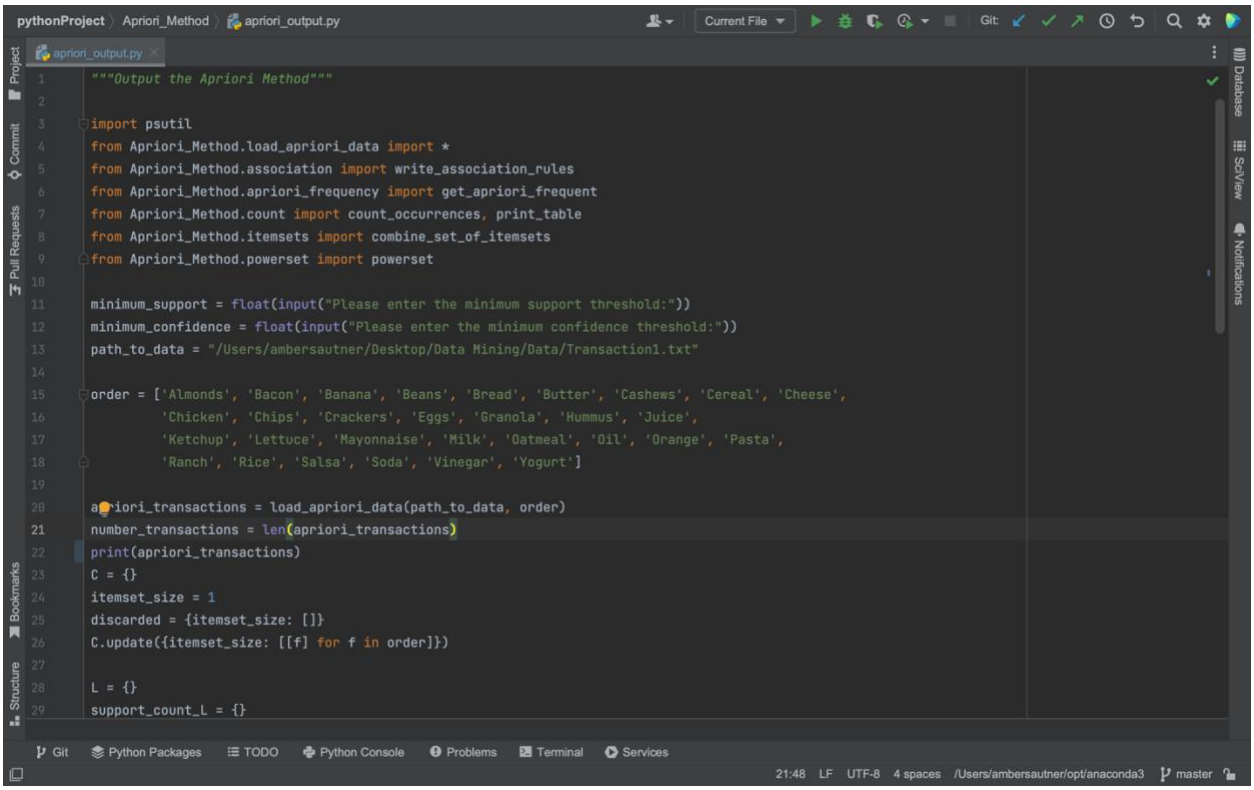
Next, we will create a similar list for L, but this will contain the updated frequent itemsets that do not contain the pruned data. (Line 28) We initialize the support_count_L as a dictionary for each frequent itemsets support count. We call the get_apriori_frequent function in order to return three values of frequent itemset_size 1. (Line 30) This returns f, the support, and the newly discarded itemsets. Lastly, we update L with f that was just returned, as well as update the support_count_L.
I then printed a table of L to show the corresponding frequency of each item.

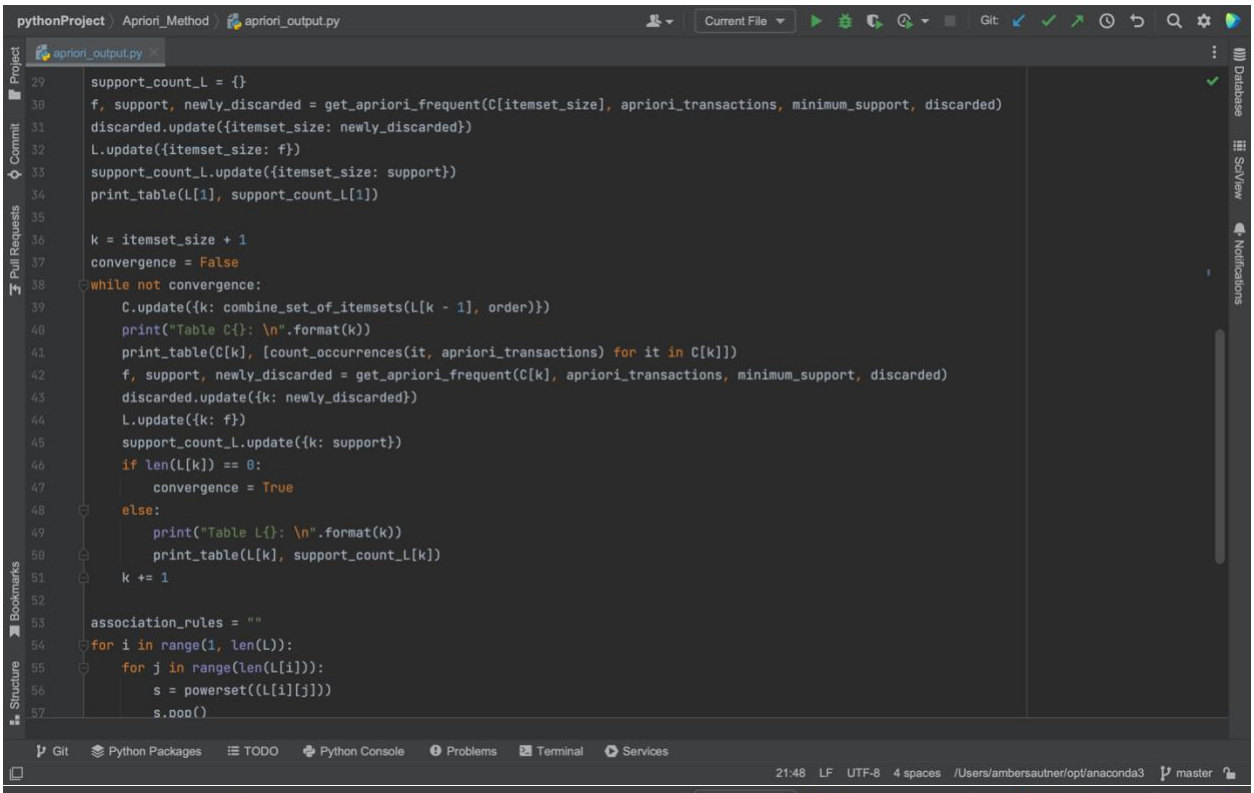
Next step is to initialize k and set it as the itemset_size + 1. (Line 36) The Apriori method works by evaluating the itemsets at k size, and continuously increasing the k by 1 at each iteration until no more frequent itemsets are found. To do this, we will create a loop using 'while'. (Line 38) We set a variable named convergence as False because we want this loop to run repeatedly until it is True.
The first step in the loop is to update the frequent itemsets through the combine_set_of_itemsets function with frequent itemsets of size 'k-1' and the order list. This will generate the candidates in iteration k. (Line 39) Since we previously initialized k as "k = itemset_size(1) +1", we must subtract 1 to get the original k value. This is stored in C[k] and printed using the print_table function to neatly display the frequent itemsets with their respective support count. (Line 40-41) With the get_apriori_frequent function, we are returning the frequent itemsets of size k. This function will store the frequent itemsets in L, the support counts in support, and update the discarded itemsets in newly_discarded. (Line 42) We then update L, support_count, and discarded. Next, we put in an 'if statement' to check if there are no more possible frequent items, and if yes then we set convergence to True and the loop breaks. (Line 46-47) If not, we will print a table with the frequent itemsets and their respective support values at k and finally update k. (line 48-51) Overall, this 'if…else' snippet is there to continuously print L until there is nothing left.

The final section is to generate the association rules with the frequent itemsets and their support values. We begin by starting off with an empty string for association_rules so we can update it later. (Line 53) The first 'for loop' will begin by looping over the frequent itemsets in L. (Line 54) The second one will go into each frequent itemset and loop over each itemset within it. (Line 55) We use a variable 's' to find all the combinations from L and then we remove the last one. (Line 56-57) We remove the last element because it represents the entire itemset, but since we just finished reviewing all the information we no longer need it. The third 'for loop' will go

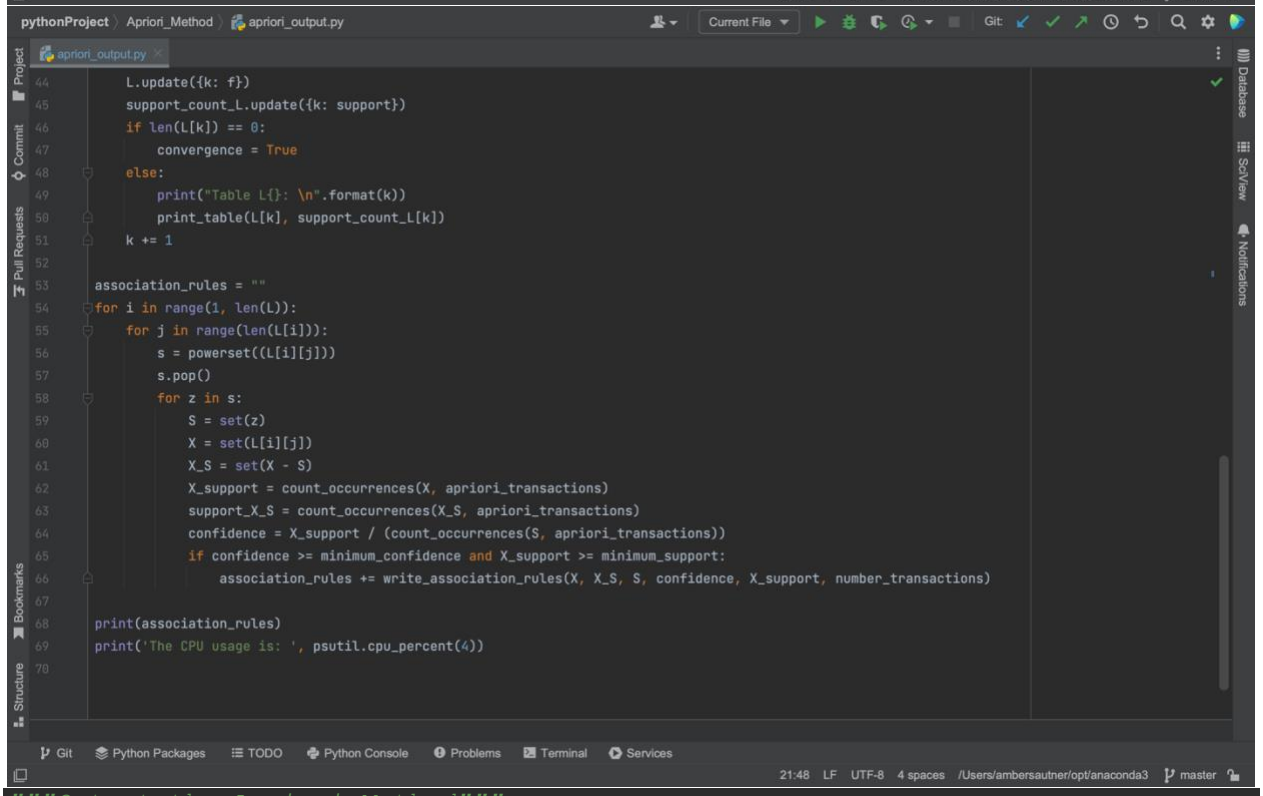
through each element in the subset to find the support for the itemset, support for the complement of the subset z in X, and the confidence of the rule. (Lines 59-64)
We add an 'if statement' to say that if the returned confidence and X_support is greater than or equal to the minimum_confidence and minimum_support both previously set by the user, then the rule is interesting. When interesting, we use the write_association_rules() function to print out the information, which is then added and printed as association_rules. (Lines 65-66) Lastly, we print these interesting association rules as well as the CPU time for this process. I imported psutil to clearly show the different in CPU time when comparing Apriori vs Brute Force.

Source Code





```
    support_count_L = {}
    f, support, newly_discarded = get_apriori_frequent(C[itemset_size], apriori_transactions, minimum_support, discarded)
    discarded.update({itemset_size: newly_discarded})
    L.update({itemset_size: f})
    support_count_L.update({itemset_size: support})
    print_table(L[1], support_count_L[1])

    k = itemset_size + 1
    convergence = False
while not convergence:
        C.update({k: combine_set_of_itemsets(L[k - 1], order)})
        print("Table C{}: \n".format(k))
        print_table(C[k], [count_occurrences(it, apriori_transactions) for it in C[k]])
        f, support, newly_discarded = get_apriori_frequent(C[k], apriori_transactions, minimum_support, discarded)
        discarded.update({k: newly_discarded})
        L.update({k: f})
        support_count_L.update({k: support})
        if len(L[k]) == 0:
            convergence = True
        else:
            print("Table L{}: \n".format(k))
            print_table(L[k], support_count_L[k])
        k += 1

    association_rules = ""
for i in range(1, len(L)):
        for j in range(len(L[i])):
            s = powerset((L[i][j]))
            s.pop()
```



```
        L.update({k: f})
        support_count_L.update({k: support})
        if len(L[k]) == 0:
            convergence = True
        else:
            print("Table L{}: \n".format(k))
            print_table(L[k], support_count_L[k])
        k += 1

    association_rules = ""
for i in range(1, len(L)):
        for j in range(len(L[i])):
            s = powerset((L[i][j]))
            s.pop()
            for z in s:
                S = set(z)
                X = set(L[i][j])
                X_S = set(X - S)
                X_support = count_occurrences(X, apriori_transactions)
                support_X_S = count_occurrences(X_S, apriori_transactions)
                confidence = X_support / (count_occurrences(S, apriori_transactions))
                if confidence >= minimum_confidence and X_support >= minimum_support:
                    association_rules += write_association_rules(X, X_S, S, confidence, X_support, number_transactions)

    print(association_rules)
    print('The CPU usage is: ', psutil.cpu_percent(4))
```

```
"""Output the Apriori Method"""

import psutil
from Apriori_Method.load_apriori_data import *
from Apriori_Method.association import write_association_rules
```

```
from Apriori_Method.apriori_frequency import get_apriori_frequent
from Apriori_Method.count import count_occurrences, print_table
from Apriori_Method.itemsets import combine_set_of_itemsets
from Apriori_Method.powerset import powerset

minimum_support = float(input("Please enter the minimum support threshold:"))
minimum_confidence = float(input("Please enter the minimum confidence
threshold:"))
path_to_data = "/Users/ambersautner/Desktop/Data
Mining/Data/Transaction1.txt"

order = ['Almonds', 'Bacon', 'Banana', 'Beans', 'Bread', 'Butter', 'Cashews',
'Cereal', 'Cheese',
         'Chicken', 'Chips', 'Crackers', 'Eggs', 'Granola', 'Hummus',
'Juice',
         'Ketchup', 'Lettuce', 'Mayonnaise', 'Milk', 'Oatmeal', 'Oil',
'Orange', 'Pasta',
         'Ranch', 'Rice', 'Salsa', 'Soda', 'Vinegar', 'Yogurt']

apriori_transactions = load_apriori_data(path_to_data, order)
number_transactions = len(apriori_transactions)
print(apriori_transactions)
C = {}
itemset_size = 1
discarded = {itemset_size: []}
C.update({itemset_size: [[f] for f in order]})

L = {}
support_count_L = {}
f, support, newly_discarded = get_apriori_frequent(C[itemset_size],
apriori_transactions, minimum_support, discarded)
discarded.update({itemset_size: newly_discarded})
L.update({itemset_size: f})
support_count_L.update({itemset_size: support})
print_table(L[1], support_count_L[1])

k = itemset_size + 1
convergence = False
while not convergence:
    C.update({k: combine_set_of_itemsets(L[k - 1], order)})
    print("Table C{}: \n".format(k))
    print_table(C[k], [count_occurrences(it, apriori_transactions) for it in
C[k]])
    f, support, newly_discarded = get_apriori_frequent(C[k],
apriori_transactions, minimum_support, discarded)
    discarded.update({k: newly_discarded})
    L.update({k: f})
    support_count_L.update({k: support})
    if len(L[k]) == 0:
        convergence = True
    else:
        print("Table L{}: \n".format(k))
        print_table(L[k], support_count_L[k])
    k += 1

association_rules = ""
for i in range(1, len(L)):
```

```
    for j in range(len(L[i])):
        s = powerset((L[i][j]))
        s.pop()
        for z in s:
            S = set(z)
            X = set(L[i][j])
            X_S = set(X - S)
            X_support = count_occurrences(X, apriori_transactions)
            support_X_S = count_occurrences(X_S, apriori_transactions)
            confidence = X_support / (count_occurrences(S,
apriori_transactions))
            if confidence >= minimum_confidence and X_support >=
minimum_support:
                association_rules += write_association_rules(X, X_S, S,
confidence, X_support, number_transactions)

print(association_rules)
print('The CPU usage is: ', psutil.cpu_percent(4))
```

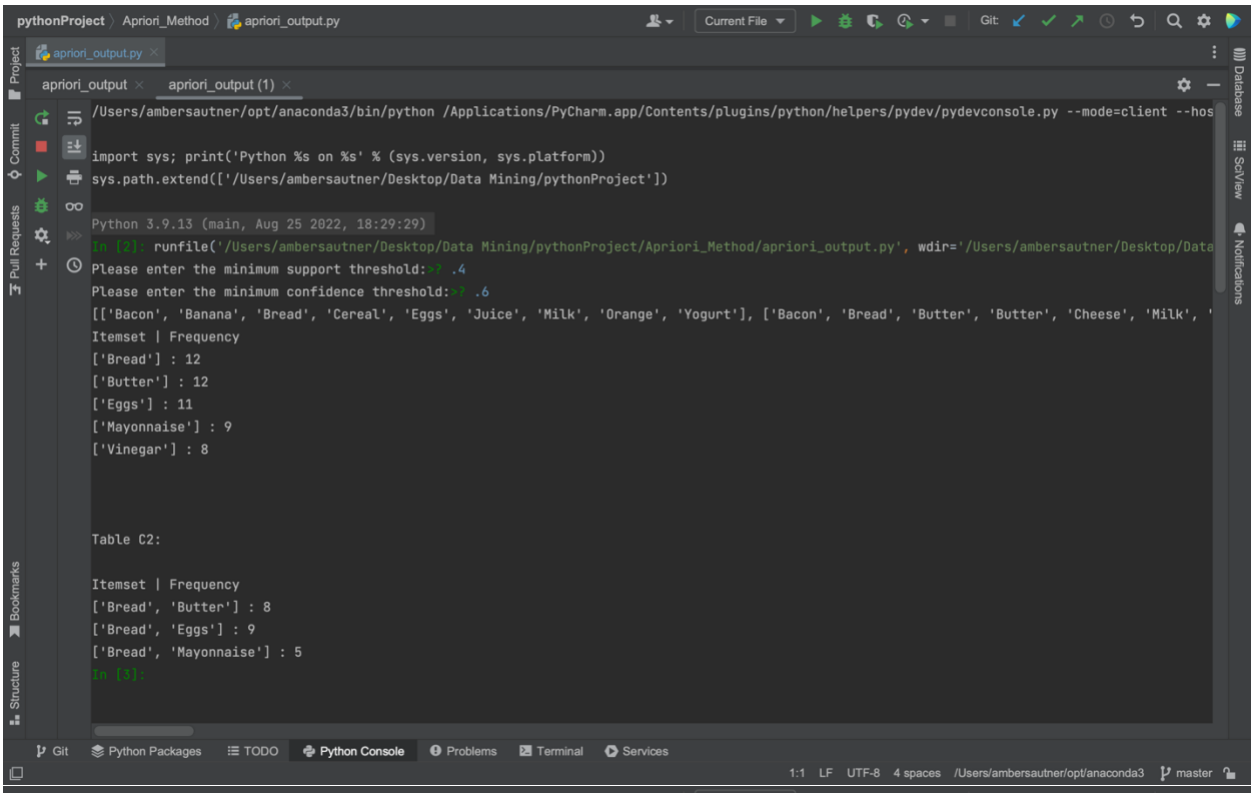
User Application

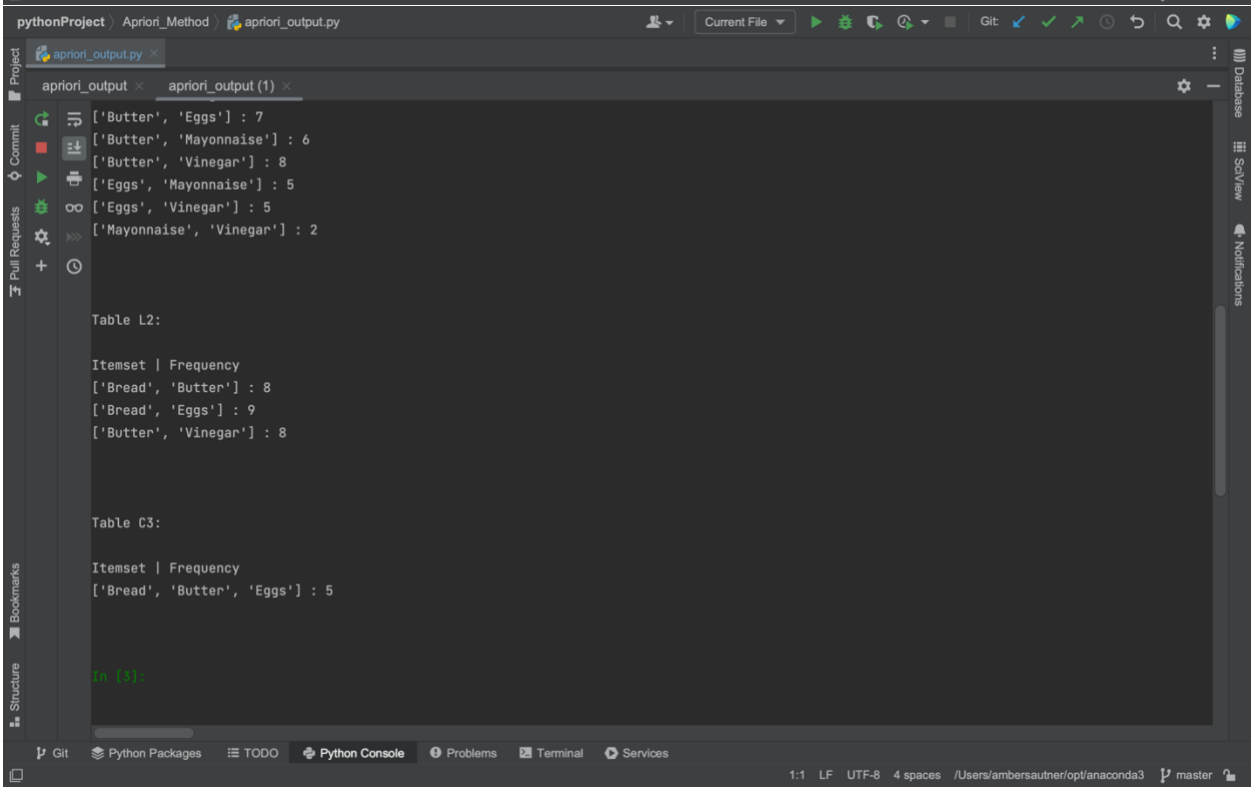
After explaining the code, I will not show various examples of different user input with various transactional data. For the first example of Transaction1.txt example 1 for both the Apriori Method and Brute Force Method, I included the tables of the frequent itemsets and association rules to provide extra clarification. However, to reduce the file storage I will print out only the ordered transactions and association rules for the rest of the examples.

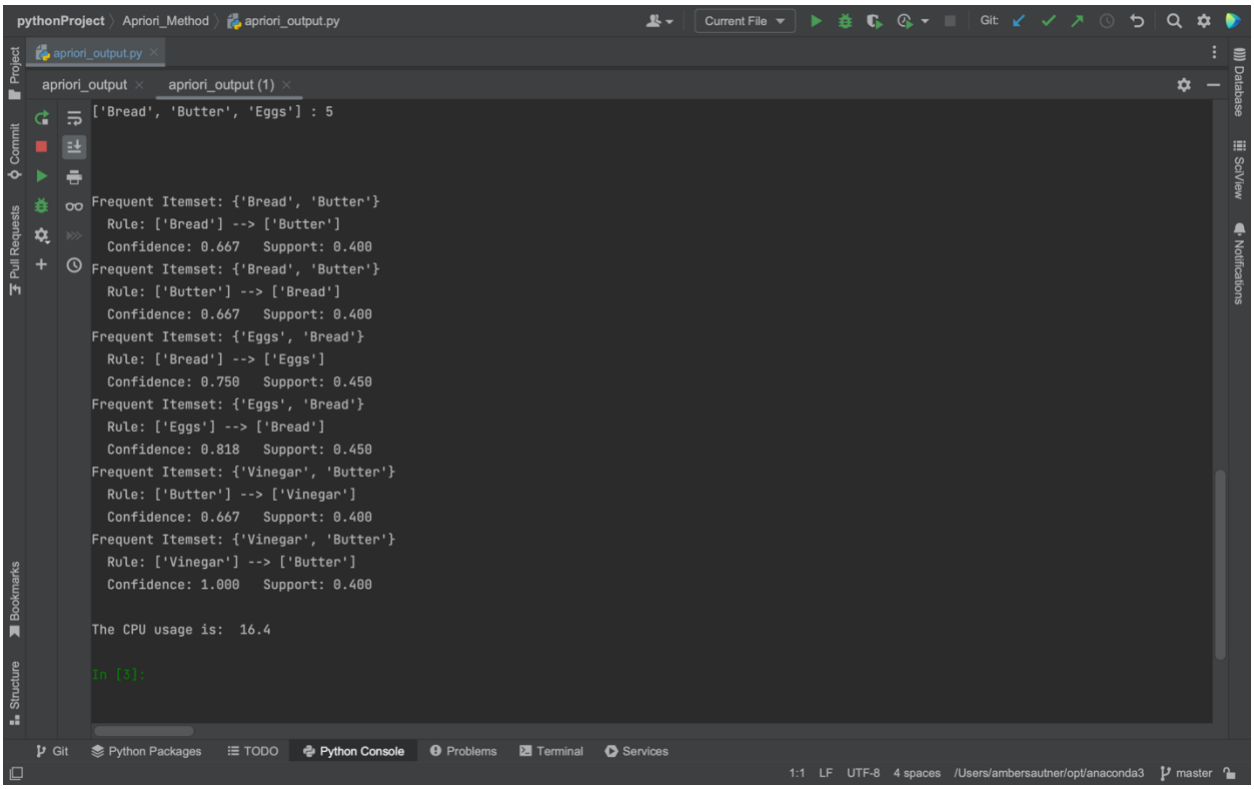
Transaction1.txt Examples

Example 1

Minimum Support = .4
Minimum Confidence = .6
CPU = 16.4


pythonProject
Apriori_Method
apriori_output.py
Current File
Git:
apriori_output
apriori_output (1)
/Users/ambersautner/opt/anaconda3/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=client --hos
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/ambersautner/Desktop/Data Mining/pythonProject'])
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
In [2]: runfile('/Users/ambersautner/Desktop/Data Mining/pythonProject/Apriori_Method/apriori_output.py', wdir='/Users/ambersautner/Desktop/Data
Please enter the minimum support threshold:?? .4
Please enter the minimum confidence threshold:?? .6
[['Bacon', 'Banana', 'Bread', 'Cereal', 'Eggs', 'Juice', 'Milk', 'Orange', 'Yogurt'], ['Bacon', 'Bread', 'Butter', 'Butter', 'Cheese', 'Milk', '
Itemset | Frequency
['Bread'] : 12
['Butter'] : 12
['Eggs'] : 11
['Mayonnaise'] : 9
['Vinegar'] : 8
Table C2:
Itemset | Frequency
['Bread', 'Butter'] : 8
['Bread', 'Eggs'] : 9
['Bread', 'Mayonnaise'] : 5
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1
LF
UTF-8
4 spaces
/Users/ambersautner/opt/anaconda3
master


pythonProject
Apriori_Method
apriori_output.py
Current File
Git:
apriori_output
apriori_output (1)
['Butter', 'Eggs'] : 7
['Butter', 'Mayonnaise'] : 6
['Butter', 'Vinegar'] : 8
['Eggs', 'Mayonnaise'] : 5
['Eggs', 'Vinegar'] : 5
['Mayonnaise', 'Vinegar'] : 2
Table L2:
Itemset | Frequency
['Bread', 'Butter'] : 8
['Bread', 'Eggs'] : 9
['Butter', 'Vinegar'] : 8
Table C3:
Itemset | Frequency
['Bread', 'Butter', 'Eggs'] : 5
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1
LF
UTF-8
4 spaces
/Users/ambersautner/opt/anaconda3
master

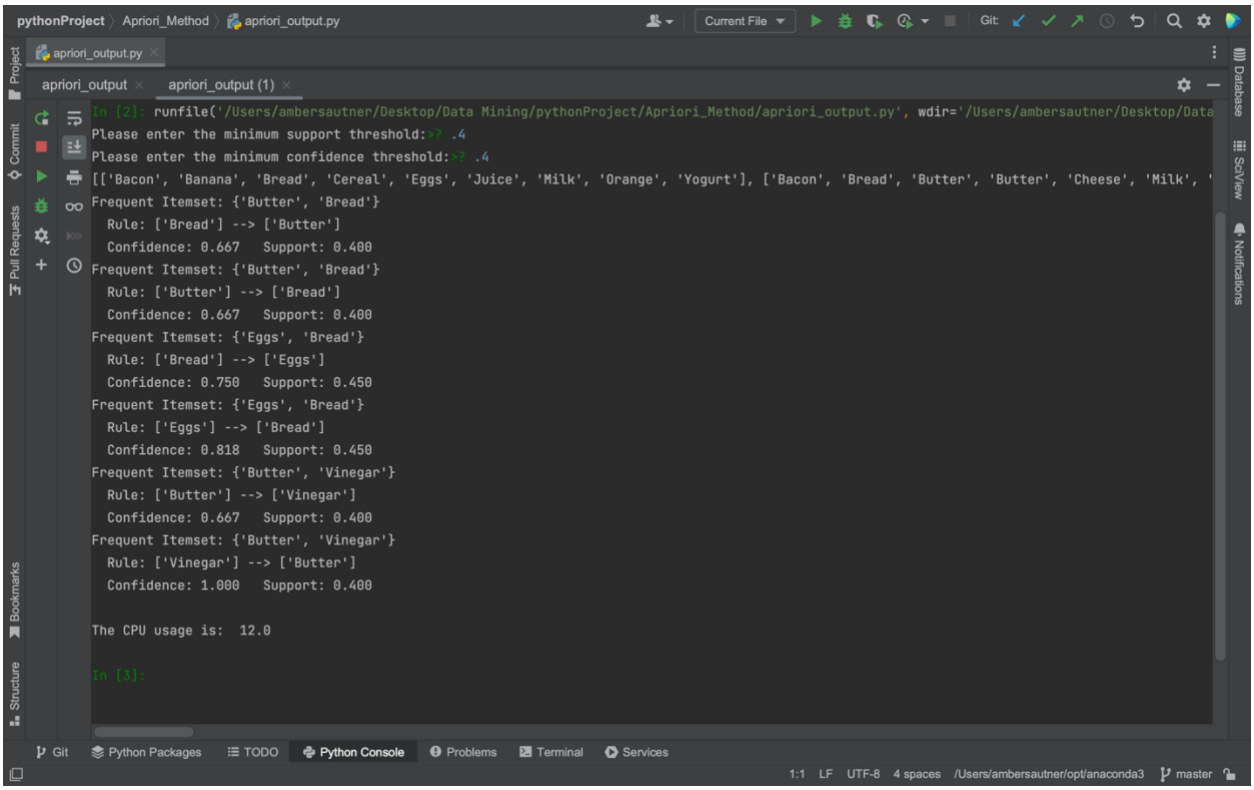


Example 2

Minimum Support = .4

Minimum Confidence =.4

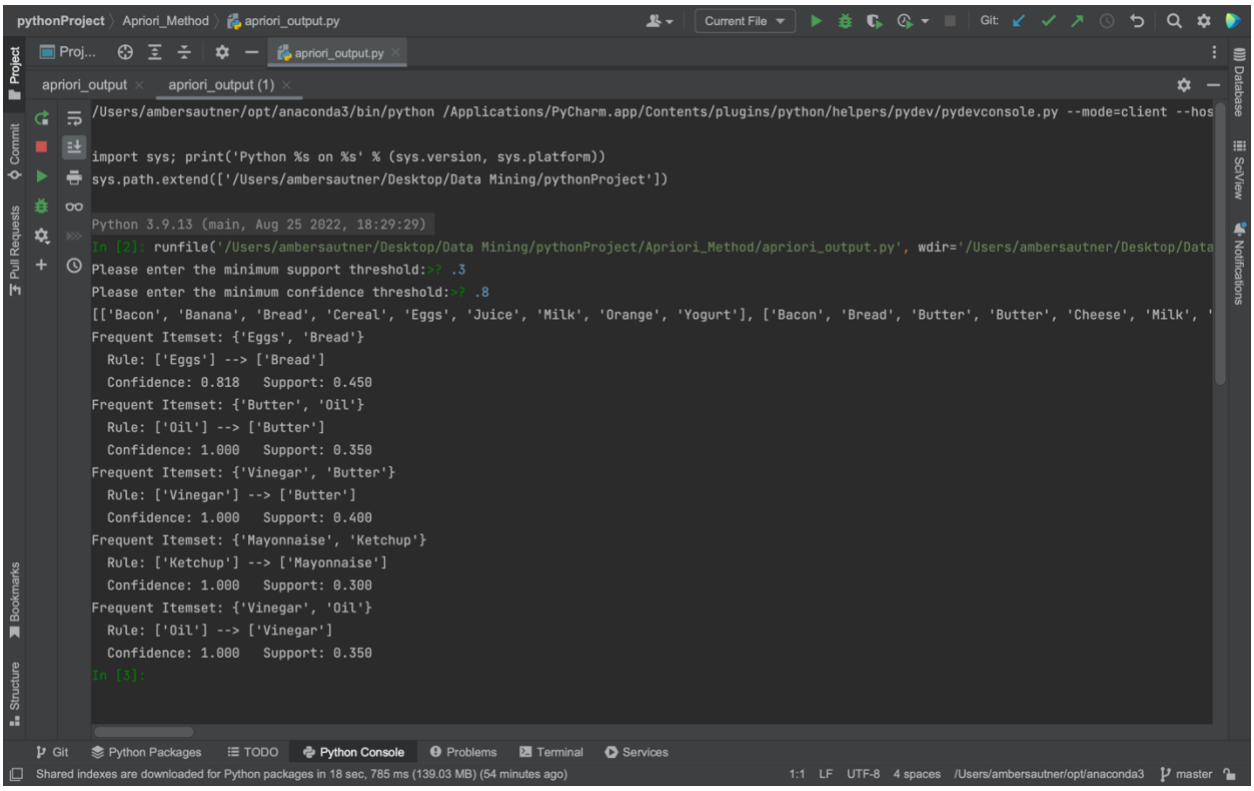
CPU = 12.0

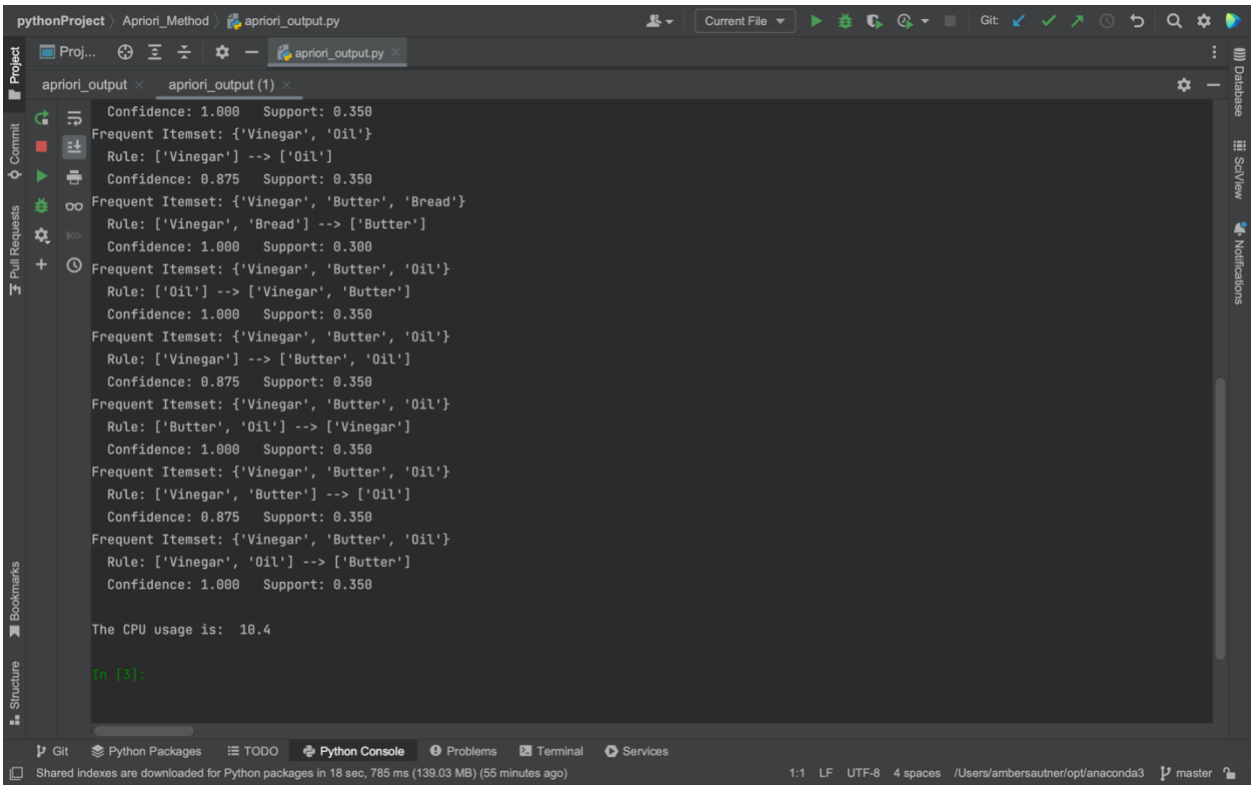


Transaction2.txt Examples

Example 1

Minimum Support = .3
Minimum Confidence = .8
CPU = 10.4





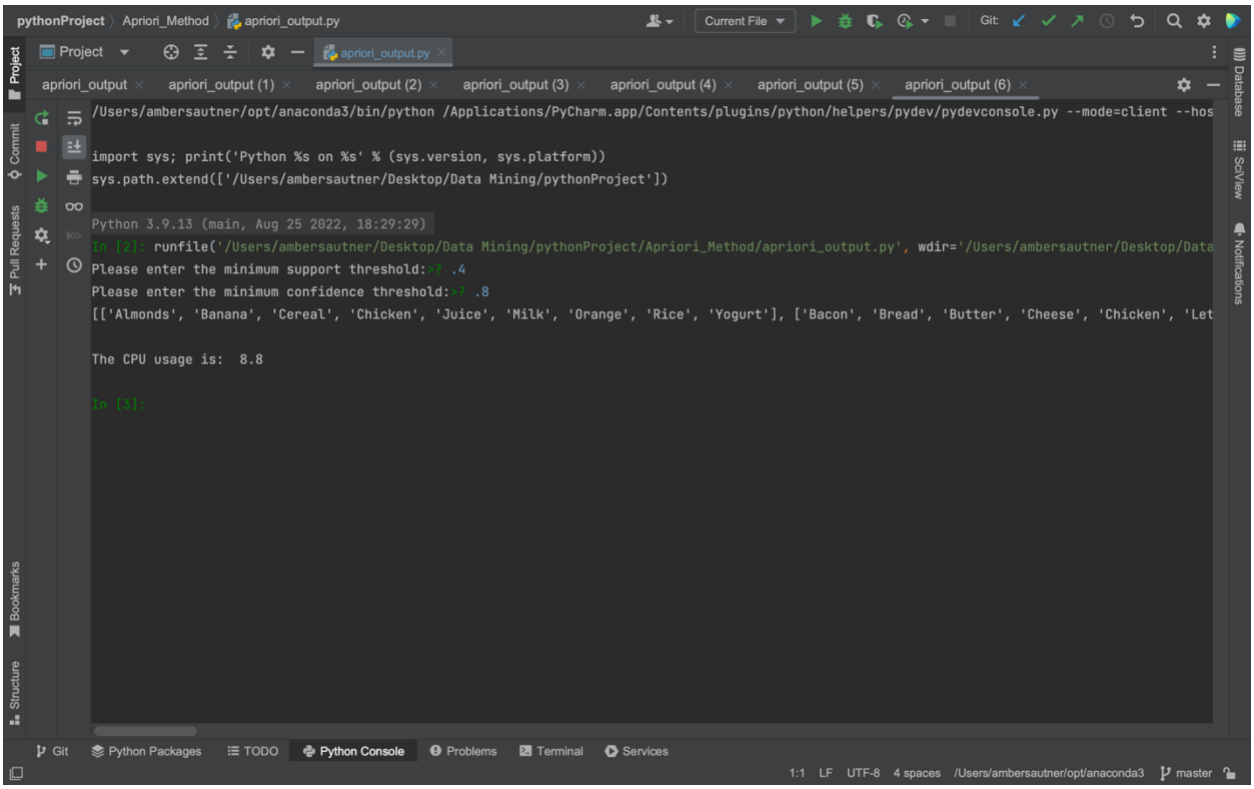
Example 2

Minimum Support = .4
Minimum Confidence = .7
CPU = 11.3

Transaction3.txt Examples

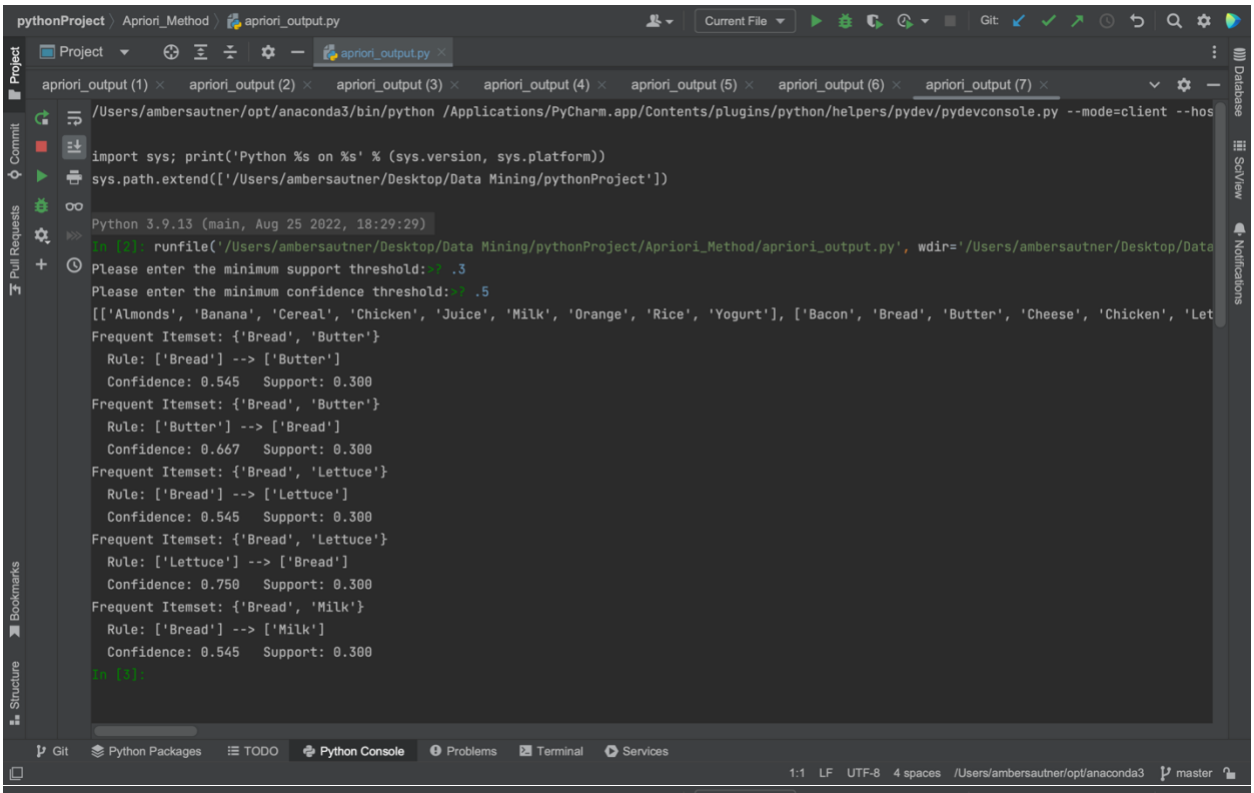
Example 1

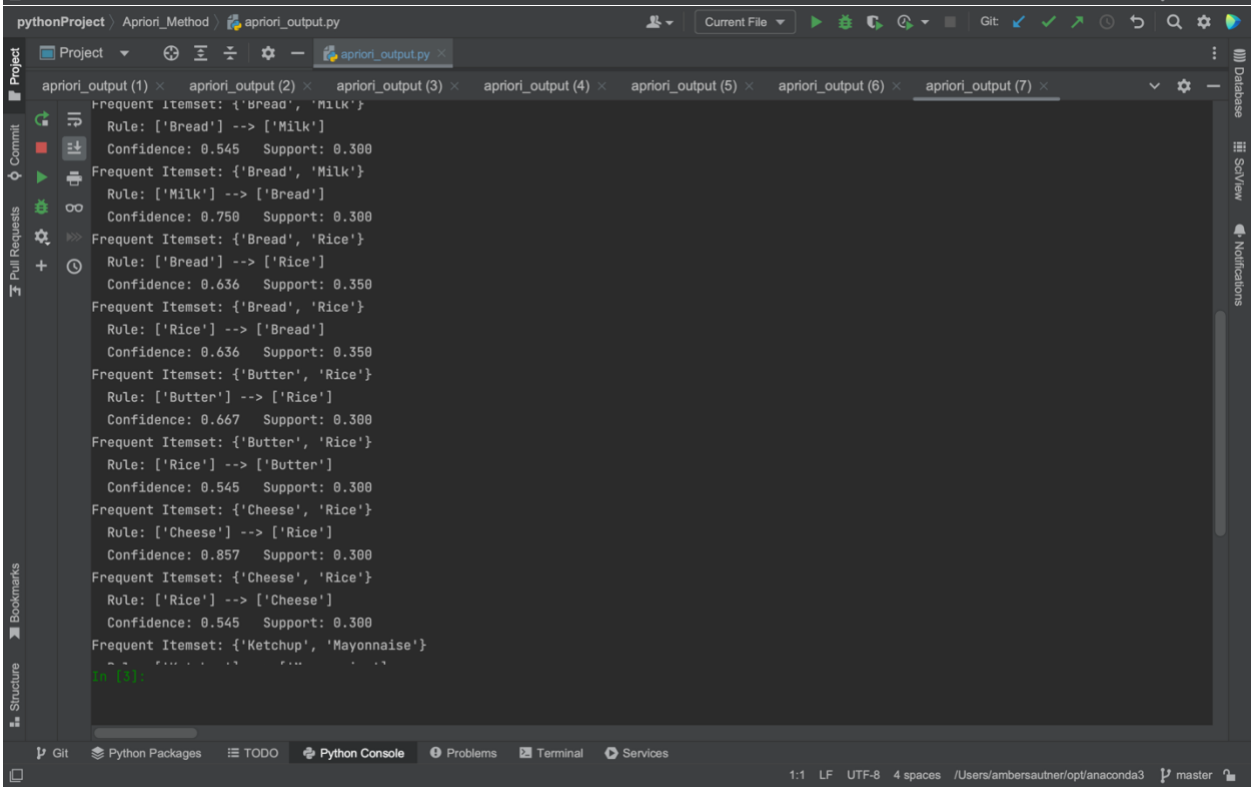
Minimum Support = .4
Minimum Confidence = .8
CPU = 8.8

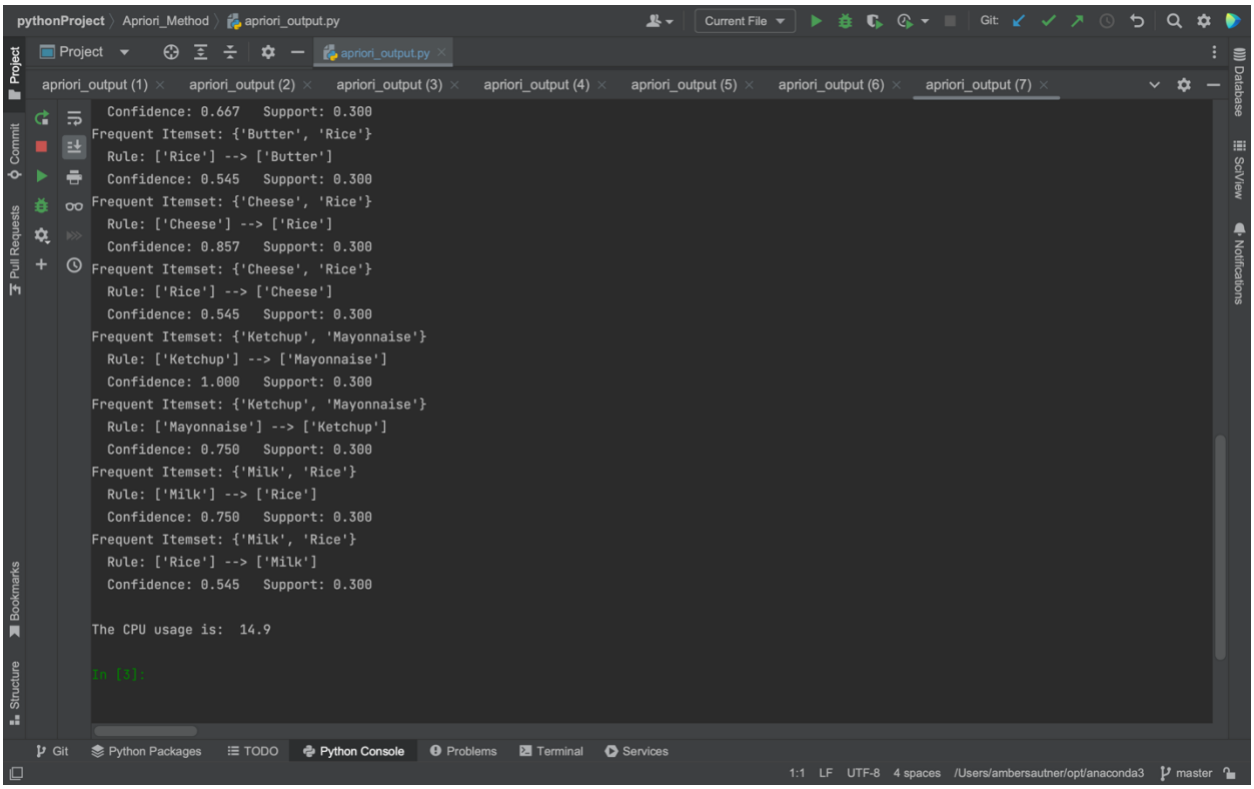


Example 2

Minimum Support = .3
Minimum Confidence = .5
CPU = 14.9


pythonProject
Apriori_Method
apriori_output.py
Current File
Git:
Project
apriori_output (1)
apriori_output (2)
apriori_output (3)
apriori_output (4)
apriori_output (5)
apriori_output (6)
apriori_output (7)
/Users/ambersautner/opt/anaconda3/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=client --hos
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/ambersautner/Desktop/Data Mining/pythonProject'])
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
In [2]: runfile('/Users/ambersautner/Desktop/Data Mining/pythonProject/Apriori_Method/apriori_output.py', wdir='/Users/ambersautner/Desktop/Data
Please enter the minimum support threshold:?? .3
Please enter the minimum confidence threshold:?? .5
[['Almonds', 'Banana', 'Cereal', 'Chicken', 'Juice', 'Milk', 'Orange', 'Rice', 'Yogurt'], ['Bacon', 'Bread', 'Butter', 'Cheese', 'Chicken', 'Let
Frequent Itemset: {'Bread', 'Butter'}
Rule: ['Bread'] --> ['Butter']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Bread', 'Butter'}
Rule: ['Butter'] --> ['Bread']
Confidence: 0.667 Support: 0.300
Frequent Itemset: {'Bread', 'Lettuce'}
Rule: ['Bread'] --> ['Lettuce']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Bread', 'Lettuce'}
Rule: ['Lettuce'] --> ['Bread']
Confidence: 0.750 Support: 0.300
Frequent Itemset: {'Bread', 'Milk'}
Rule: ['Bread'] --> ['Milk']
Confidence: 0.545 Support: 0.300
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master

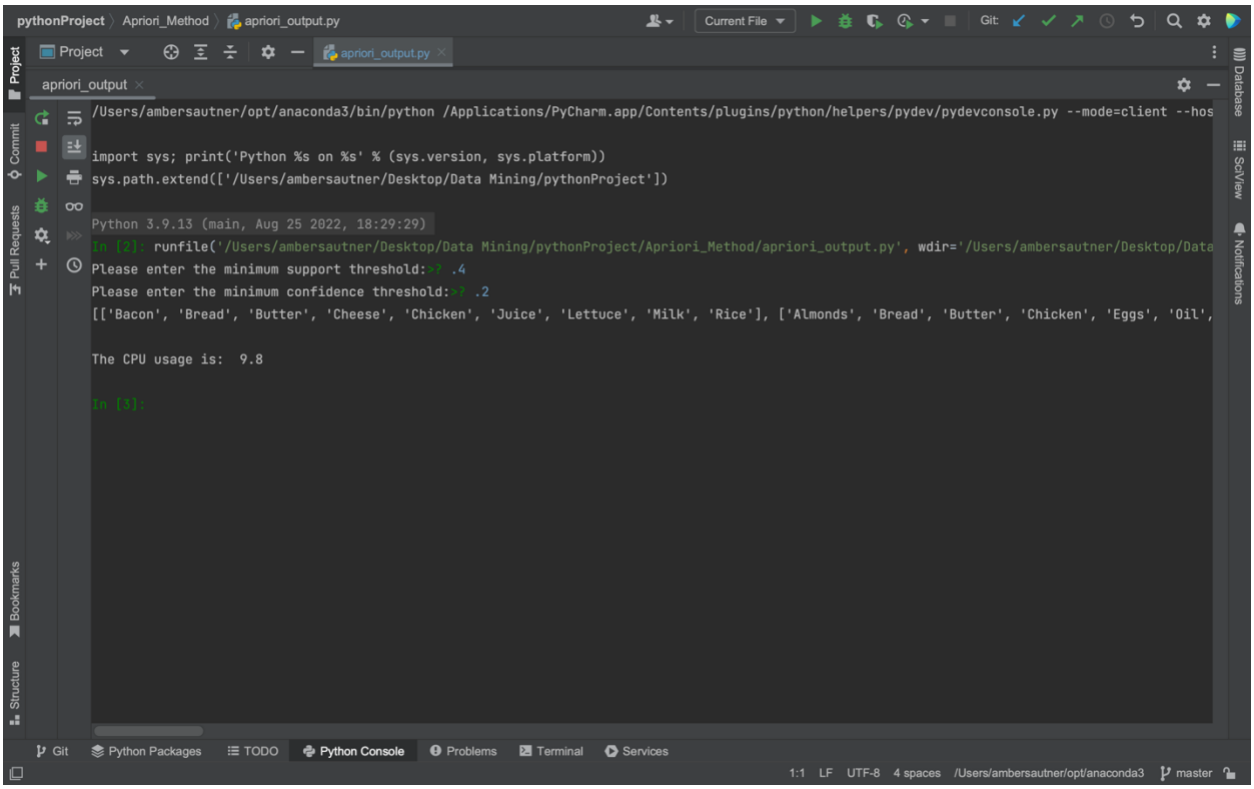

pythonProject
Apriori_Method
apriori_output.py
Current File
Git:
Project
apriori_output (1)
apriori_output (2)
apriori_output (3)
apriori_output (4)
apriori_output (5)
apriori_output (6)
apriori_output (7)
Rule: ['Bread'] --> ['Milk']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Bread', 'Milk'}
Rule: ['Milk'] --> ['Bread']
Confidence: 0.750 Support: 0.300
Frequent Itemset: {'Bread', 'Rice'}
Rule: ['Bread'] --> ['Rice']
Confidence: 0.636 Support: 0.350
Frequent Itemset: {'Bread', 'Rice'}
Rule: ['Rice'] --> ['Bread']
Confidence: 0.636 Support: 0.350
Frequent Itemset: {'Butter', 'Rice'}
Rule: ['Butter'] --> ['Rice']
Confidence: 0.667 Support: 0.300
Frequent Itemset: {'Butter', 'Rice'}
Rule: ['Rice'] --> ['Butter']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Cheese', 'Rice'}
Rule: ['Cheese'] --> ['Rice']
Confidence: 0.857 Support: 0.300
Frequent Itemset: {'Cheese', 'Rice'}
Rule: ['Rice'] --> ['Cheese']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Ketchup', 'Mayonnaise'}
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master



Transaction4.txt Examples

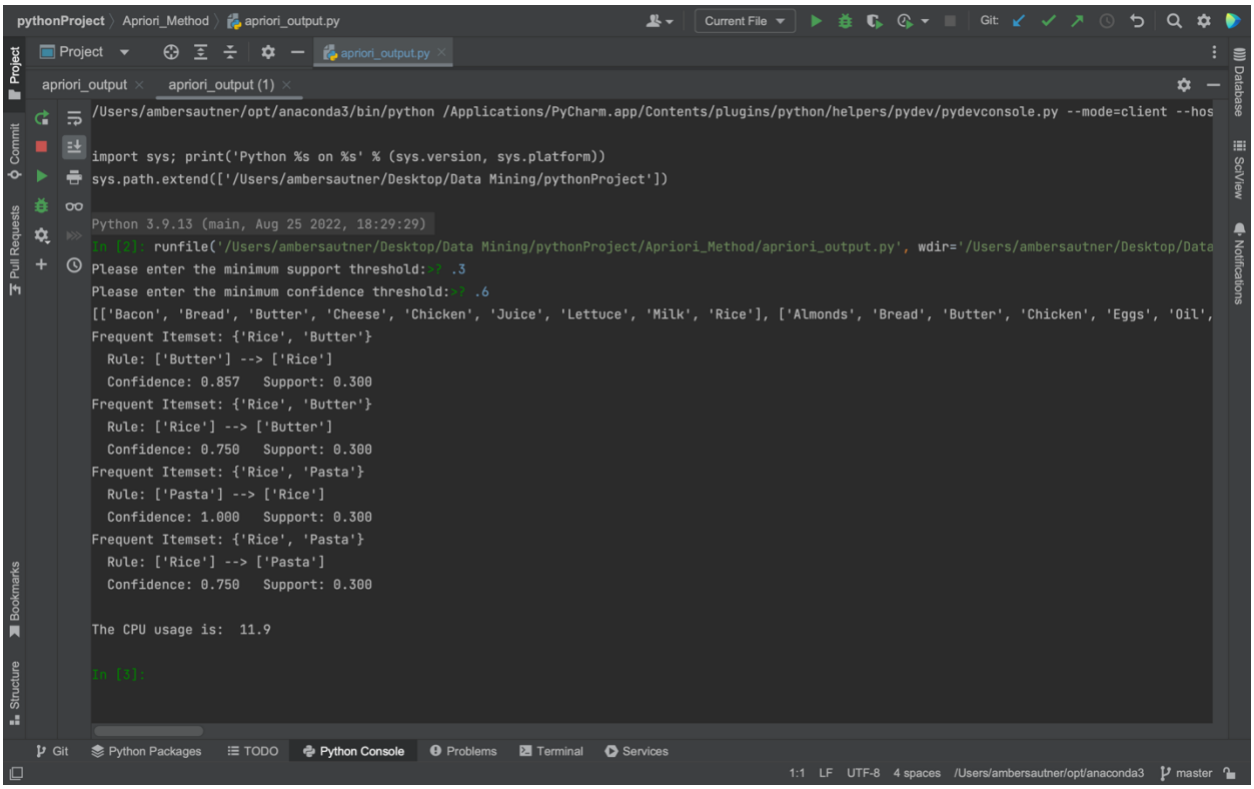
Example 1

Minimum Support = .4
Minimum Confidence = .2
CPU = 9.8



Example 2

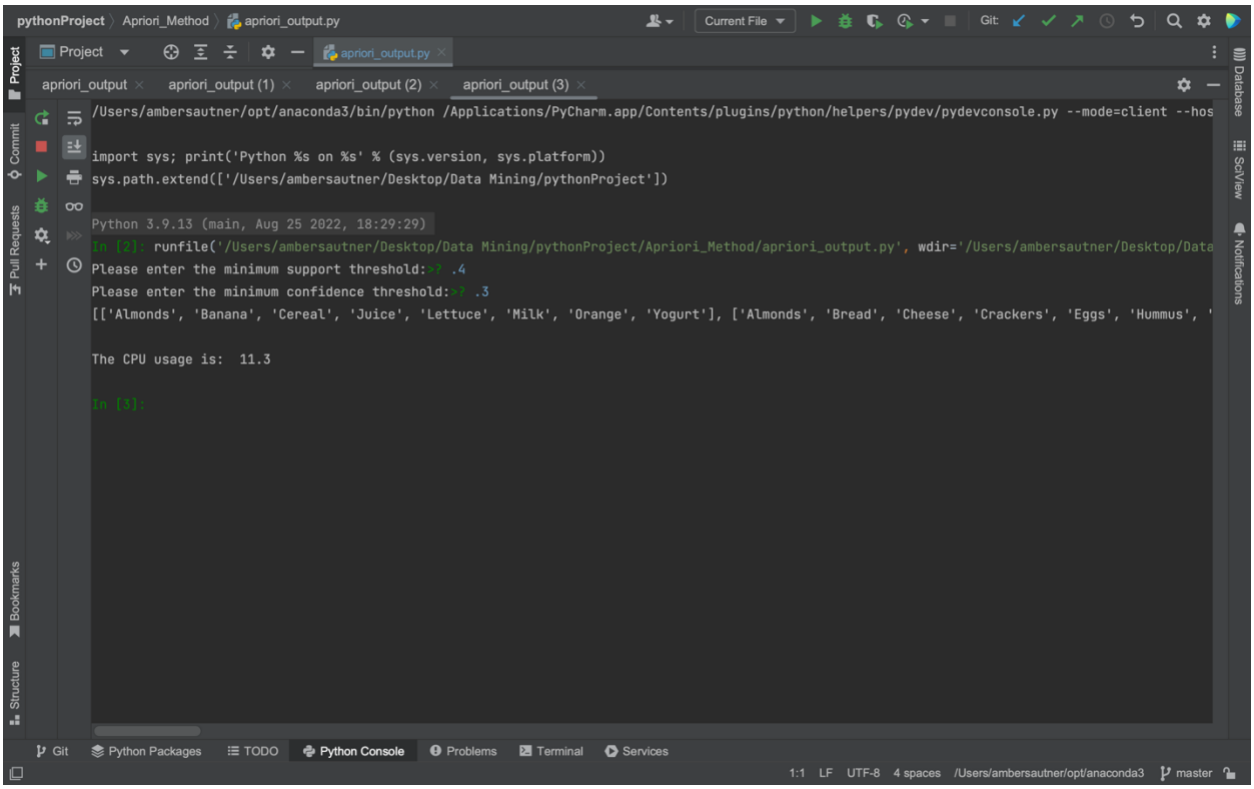
Minimum Support = .3
Minimum Confidence = .6
CPU = 11.9



Transaction5.txt Examples

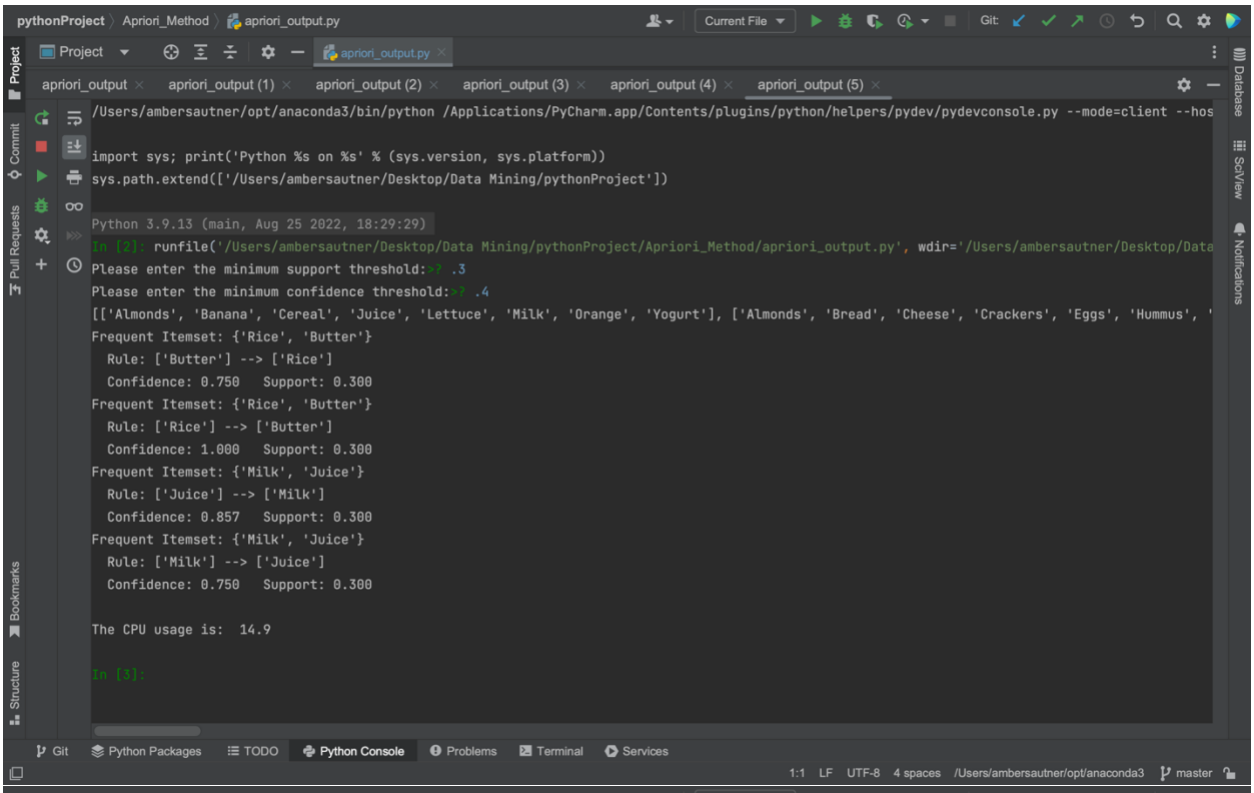
Example 1

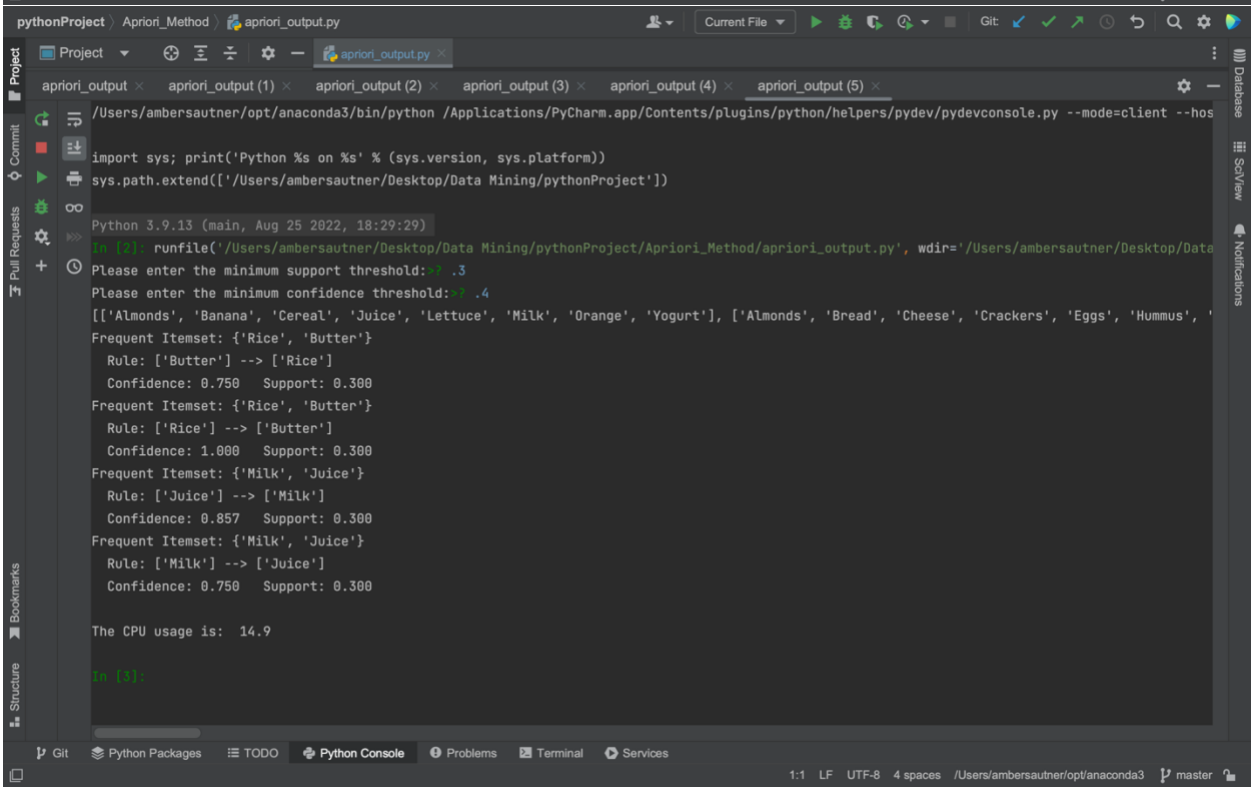
Minimum Support = .4
Minimum Confidence = .3
CPU = 11.3



Example 2

Minimum Support = .3
Minimum Confidence = .4
CPU = 14.9


pythonProject
Apriori_Method
apriori_output.py
Current File
Git:
Project
apriori_output.py
apriori_output
apriori_output (1)
apriori_output (2)
apriori_output (3)
apriori_output (4)
apriori_output (5)
/Users/ambersautner/opt/anaconda3/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=client --hos
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/ambersautner/Desktop/Data Mining/pythonProject'])
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
In [2]: runfile('/Users/ambersautner/Desktop/Data Mining/pythonProject/Apriori_Method/apriori_output.py', wdir='/Users/ambersautner/Desktop/Data
Please enter the minimum support threshold:>? .3
Please enter the minimum confidence threshold:>? .4
[['Almonds', 'Banana', 'Cereal', 'Juice', 'Lettuce', 'Milk', 'Orange', 'Yogurt'], ['Almonds', 'Bread', 'Cheese', 'Crackers', 'Eggs', 'Hummus',
Frequent Itemset: {'Rice', 'Butter'}
Rule: ['Butter'] --> ['Rice']
Confidence: 0.750 Support: 0.300
Frequent Itemset: {'Rice', 'Butter'}
Rule: ['Rice'] --> ['Butter']
Confidence: 1.000 Support: 0.300
Frequent Itemset: {'Milk', 'Juice'}
Rule: ['Juice'] --> ['Milk']
Confidence: 0.857 Support: 0.300
Frequent Itemset: {'Milk', 'Juice'}
Rule: ['Milk'] --> ['Juice']
Confidence: 0.750 Support: 0.300
The CPU usage is: 14.9
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master


pythonProject
Apriori_Method
apriori_output.py
Current File
Git:
Project
apriori_output.py
apriori_output
apriori_output (1)
apriori_output (2)
apriori_output (3)
apriori_output (4)
apriori_output (5)
/Users/ambersautner/opt/anaconda3/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=client --hos
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/ambersautner/Desktop/Data Mining/pythonProject'])
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
In [2]: runfile('/Users/ambersautner/Desktop/Data Mining/pythonProject/Apriori_Method/apriori_output.py', wdir='/Users/ambersautner/Desktop/Data
Please enter the minimum support threshold:>? .3
Please enter the minimum confidence threshold:>? .4
[['Almonds', 'Banana', 'Cereal', 'Juice', 'Lettuce', 'Milk', 'Orange', 'Yogurt'], ['Almonds', 'Bread', 'Cheese', 'Crackers', 'Eggs', 'Hummus',
Frequent Itemset: {'Rice', 'Butter'}
Rule: ['Butter'] --> ['Rice']
Confidence: 0.750 Support: 0.300
Frequent Itemset: {'Rice', 'Butter'}
Rule: ['Rice'] --> ['Butter']
Confidence: 1.000 Support: 0.300
Frequent Itemset: {'Milk', 'Juice'}
Rule: ['Juice'] --> ['Milk']
Confidence: 0.857 Support: 0.300
Frequent Itemset: {'Milk', 'Juice'}
Rule: ['Milk'] --> ['Juice']
Confidence: 0.750 Support: 0.300
The CPU usage is: 14.9
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master

Brute Force Method

Brute Force Overview

The Brute Force method works by generating all possible combinations of itemsets from the given dataset and searching through each to find the frequent itemsets. Unlike the Apriori method which discards infrequent items, the brute force method will compute each frequent and infrequent combination. This makes it less efficient and scalable for large datasets. Like before, the functions will be in green and variable in red.

Code Analysis

Load_brute_data.py

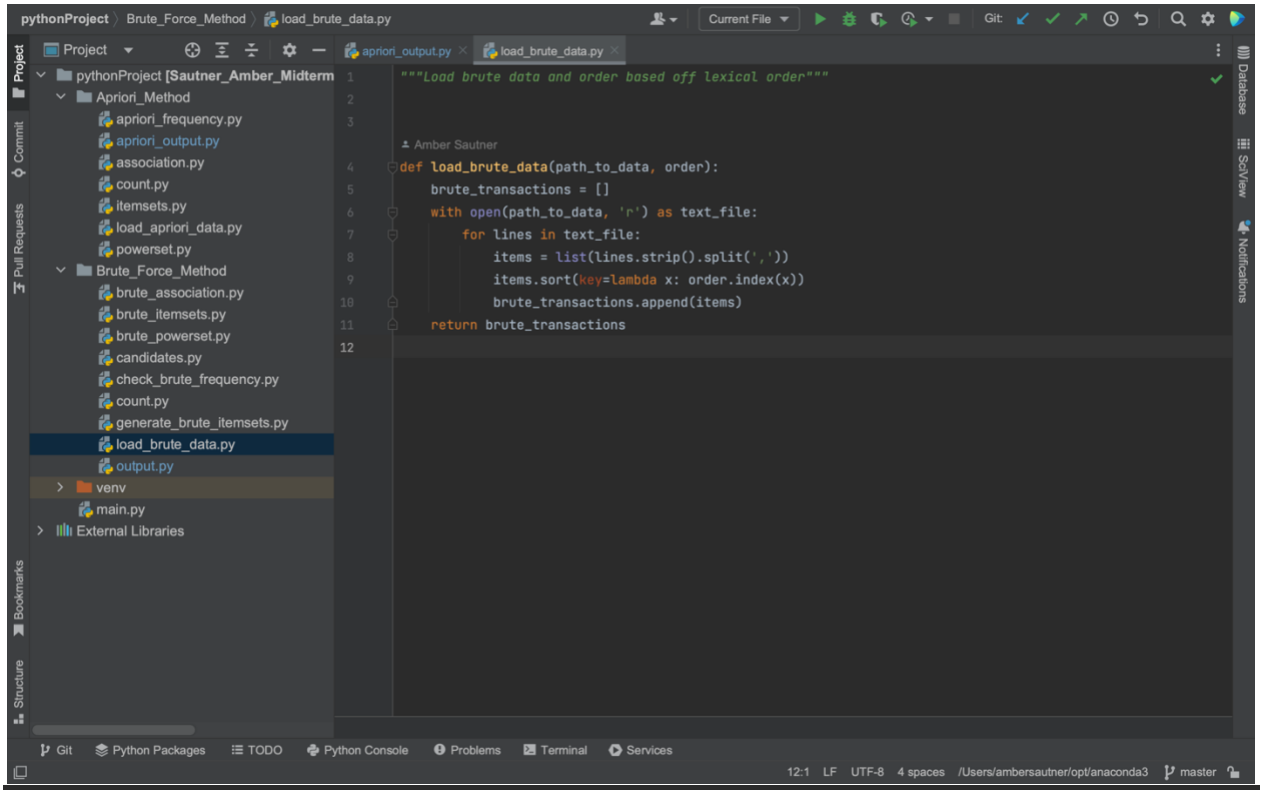
General Overview

Similarly to the load_apriori_data function, this file will load the transactional data and return it as an ordered list.

Documentation

The function, load_brute_data, takes two parameters, path_to_data and order. It begins by initializing an empty list for brute_transactions, which we later will update. (Line 5) The code then opens the inputted data file as read-only and begins a 'for loop' to iterate through each line to strip it and split the items by a comma. A list is returned based off the given order and then appended to the brute_transactions list that we initialized before.

Source Code




```
"""Load brute data and order based off lexical order"""


def load_brute_data(path_to_data, order):
    brute_transactions = []
```

```
    with open(path_to_data, 'r') as text_file:
        for lines in text_file:
            items = list(lines.strip().split(','))
            items.sort(key=lambda x: order.index(x))
            brute_transactions.append(items)
    return brute_transactions
```

Generate_brute_itemsets.py

General Overview

In this file, we are generating all the brute itemsets based off the lexical order and the user-defined minimum support. We return the frequent itemsets and their respective support counts.

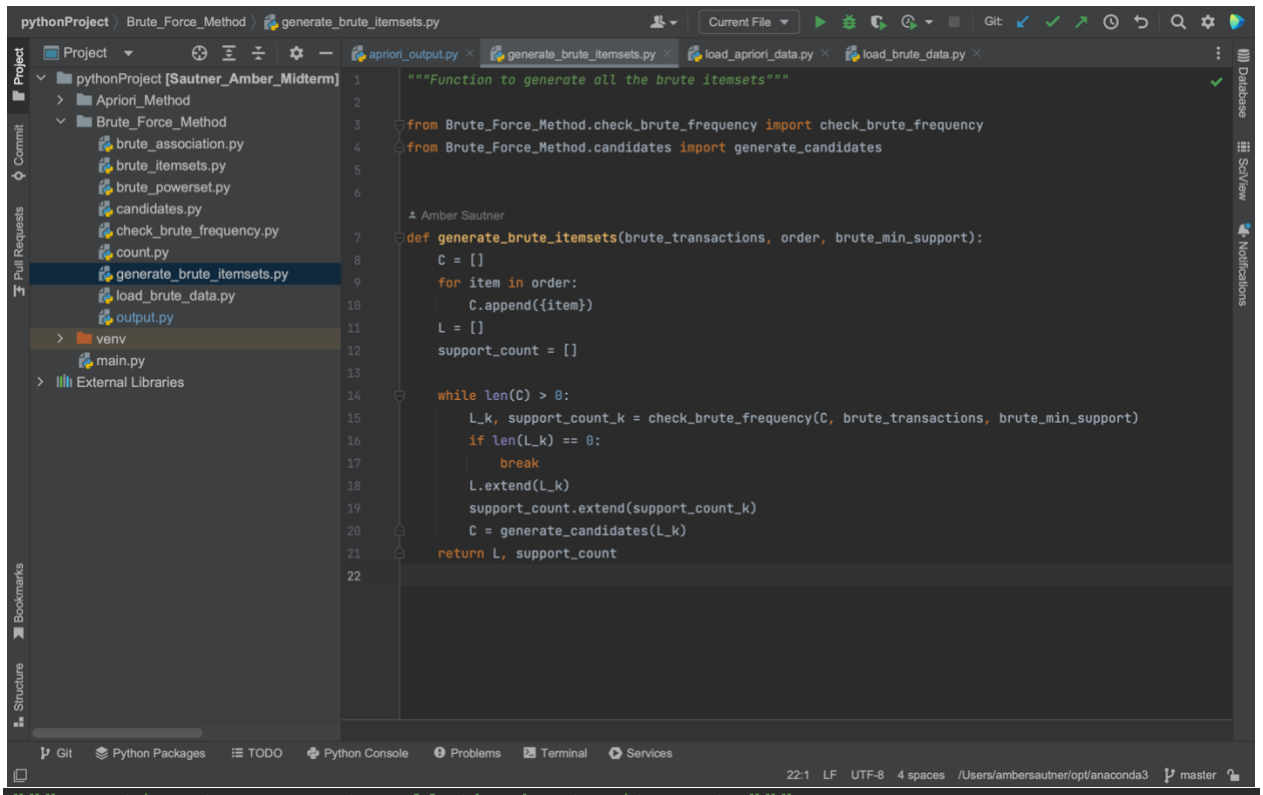
Documentation

This function, generate_brute_itemsets, takes in three parameters, brute_transactions, order, and brute_min_support. We begin by initializing an empty list for C which will hold all the generated candidates itemsets. The next line will start a 'for loop' to iterate over each item in the order list. In this loop, we will create a set with only one item from the order list and append it to C. (Line 9-10) This will be the list of candidate itemsets used later.

Next we will initialize an empty list for the frequent itemsets as well as their support counts. (Line 11-12) The 'while' loop is set to run when there are itemsets within the C list. We then call the check_brute_frequency function to find the frequent itemsets in C, which returns the frequent itemsets as L_k and their support counts as support_count_k. (Line 15) We add an 'if statement' to check whether there are more frequent itemsets to be found and if not, the loop is broken. (Line 16-17)

When there are more frequent itemsets, we will start by appending all the frequent itemsets in L_k to L and their respective support counts to support_count. (Line 18-19) After, we call the generate_candidates function on L_k and set the output to C. Finally, we return the frequent itemsets and their support counts as L and support_count.

Source Code




```
"""Function to generate all the brute itemsets"""

from Brute_Force_Method.check_brute_frequency import check_brute_frequency
from Brute_Force_Method.candidates import generate_candidates


def generate_brute_itemsets(brute_transactions, order, brute_min_support):
    C = []
    for item in order:
        C.append({item})
    L = []
    support_count = []

    while len(C) > 0:
        L_k, support_count_k = check_brute_frequency(C, brute_transactions,
brute_min_support)
        if len(L_k) == 0:
            break
        L.extend(L_k)
        support_count.extend(support_count_k)
        C = generate_candidates(L_k)
    return L, support_count
```

Check_brute_frequency.py

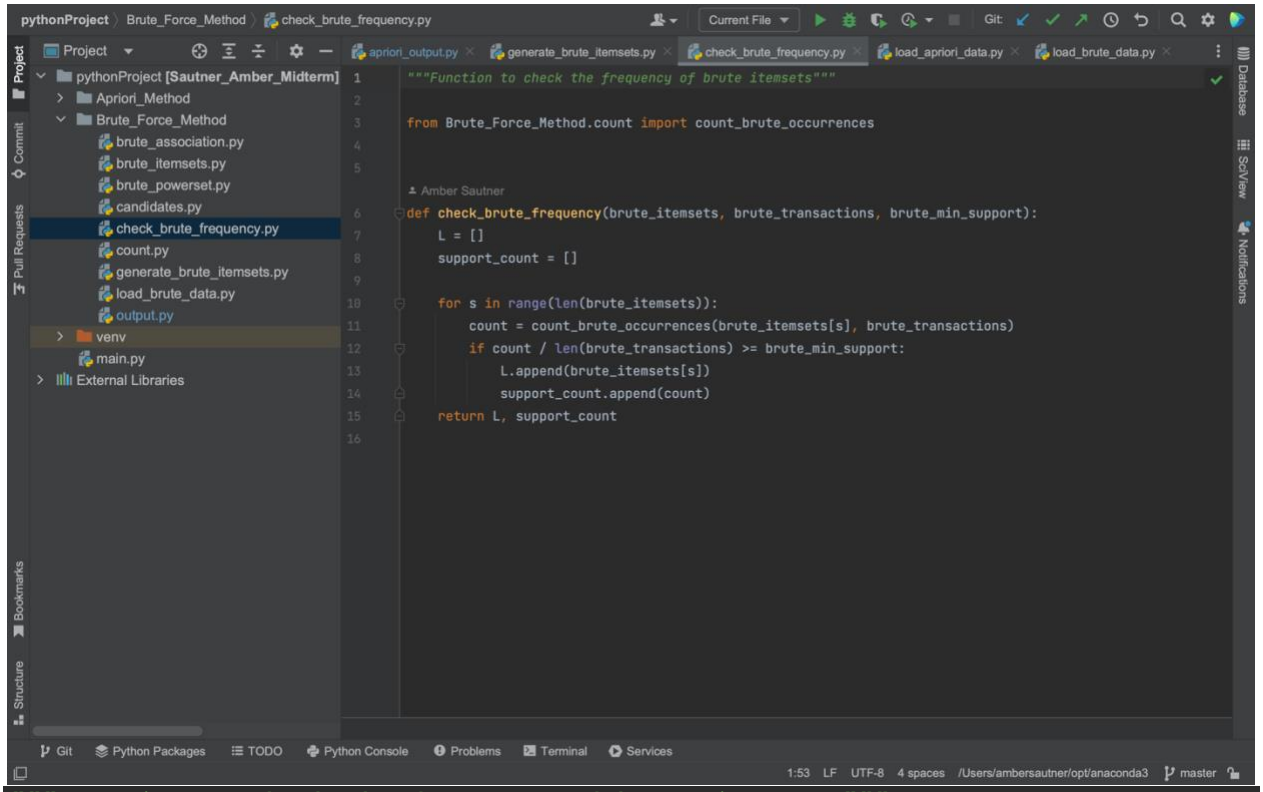
General Overview

This function will iterate through the itemsets to check the frequency of the itemsets.The frequent itemsets and their support counts are returned.

Documentation

This function, check_brute_frequency, takes in three parameters, brute_itemsets, brute_transactions, and brute_min_support. It will start by initializing an empty list L for the frequent itemsets and their respective support counts to be added later. (Line 7-8) Then, a 'for loop' is started to iterate through the brute itemsets and start a count based off the count_brute_occurrences function. Count will represent how many times the brute itemsets appeared in the transactions. (Line 11) An 'if statement' will check whether the itemset meets the requirements of the user-defined brute minimum support. If so, then the brute itemsets are appended to L and the count appended to support_count. Finally, L and support_count are returned.

Source Code




```
"""Function to check the frequency of brute itemsets"""

from Brute_Force_Method.count import count_brute_occurrences


def check_brute_frequency(brute_itemsets, brute_transactions,
brute_min_support):
    L = []
    support_count = []

    for s in range(len(brute_itemsets)):
        count = count_brute_occurrences(brute_itemsets[s],
brute_transactions)
        if count / len(brute_transactions) >= brute_min_support:
            L.append(brute_itemsets[s])
```

```
            support_count.append(count)
    return L, support_count
```

Count.py

General Overview

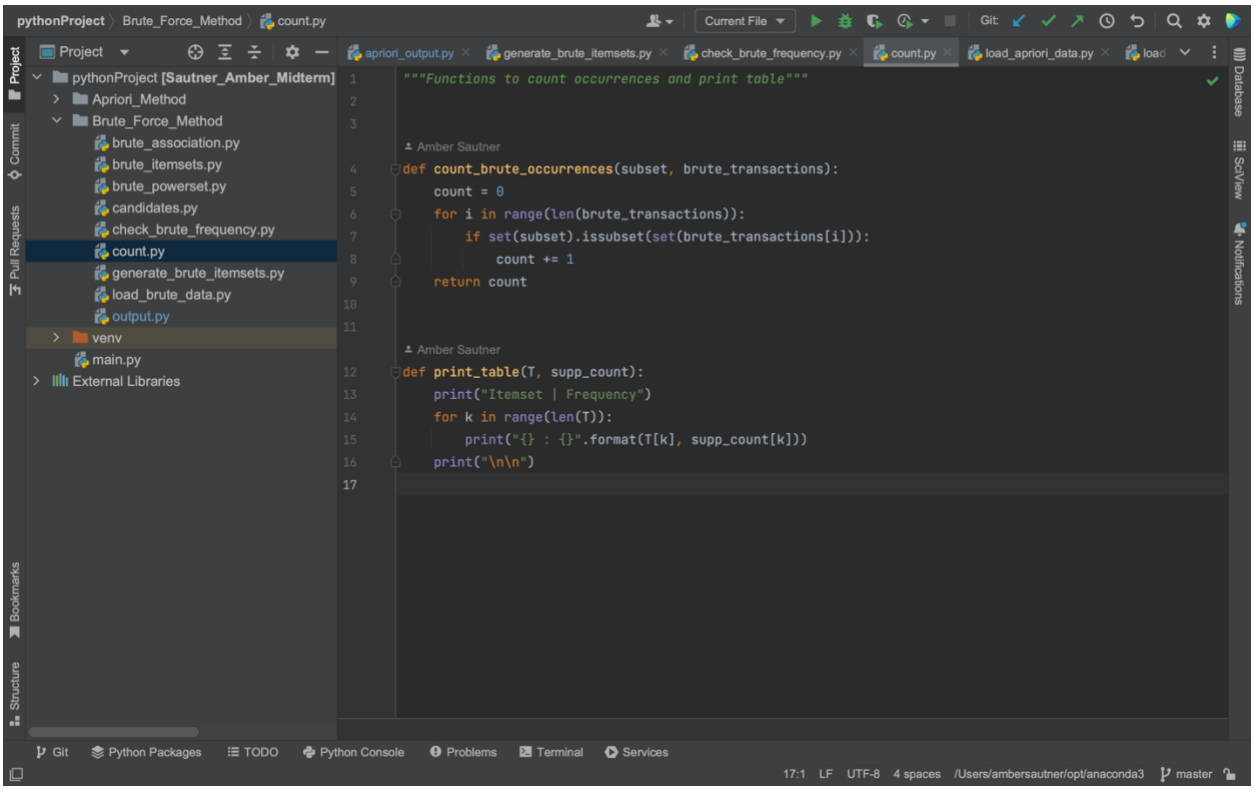
This file will contain two functions, count_brute_occurrences and print_table.

Documentation

The first function, count_brute_occurrences, will help the brute force method's process by counting the amount of times an itemsets occurs in one transaction. It takes two parameters, subset and brute_transaction, and returns the updated count. The count is incremented in a 'for loop' that iterates through the brute_transactions to see if the item is a subset of the set we are checking. If so, the count is incremented by 1.

The print_table function will neatly print the itemset and its frequency in a formatted string.

Source Code



Brute_association.py

General Overview

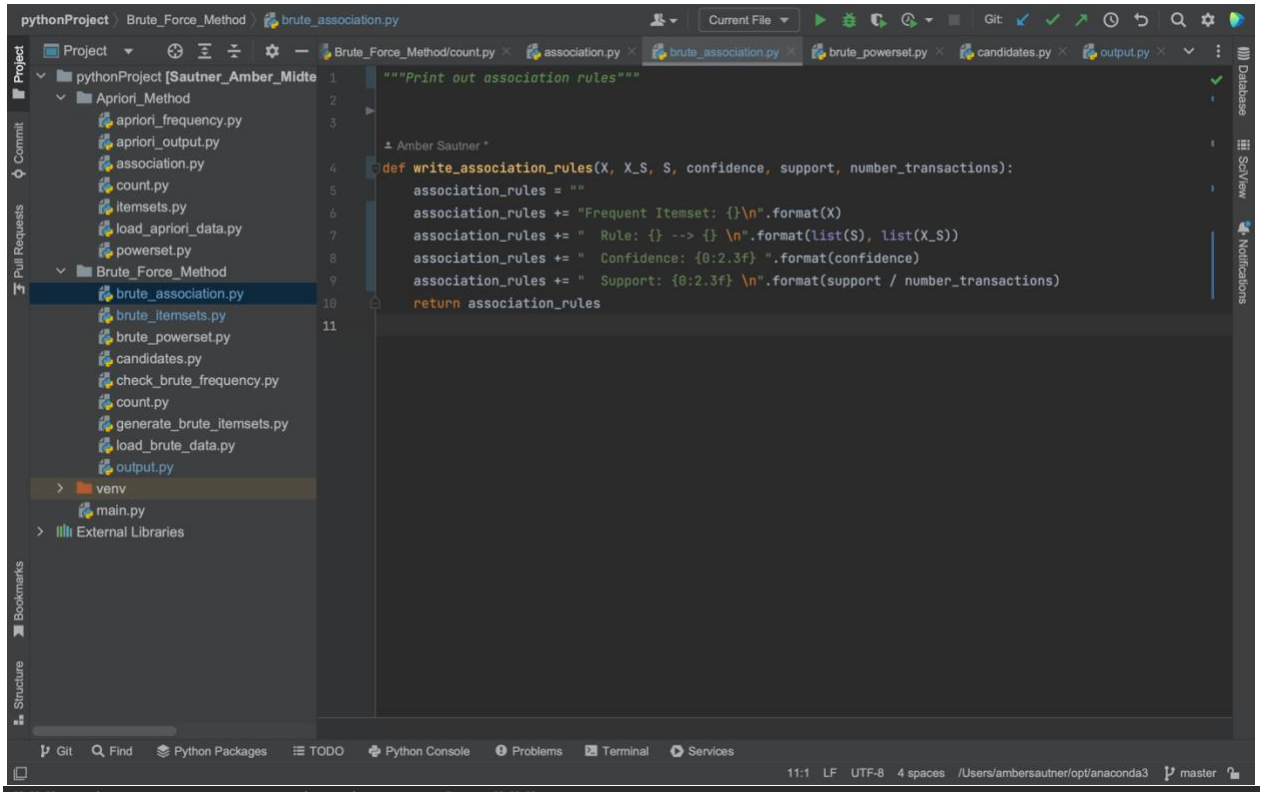
This function is the same as the write_association function in the Apriori method. It was created to return the association rules as a formatted string that is easily understood by the user.

Documentation

The function, write_association_rules, takes in 5 parameters; X, X_S, S, confidence, support, and number_transactions. Each line will generate the respective association rules between the itemset X and its subset X_S given the support count from itemset X, the support count from

X_S, and the number_transactions. The association_rules variable is returned as a string with these 3 lines of information.

Source Code




```
"""Print out association rules"""


def write_association_rules(X, X_S, S, confidence, support,
number_transactions):
    association_rules = ""
    association_rules += "Frequent Itemset: {}\n".format(X)
    association_rules += "  Rule: {} --> {} \n".format(list(S), list(X_S))
    association_rules += "  Confidence: {0:2.3f} ".format(confidence)
    association_rules += "  Support: {0:2.3f} \n".format(support /
number_transactions)
    return association_rules
```

Brute_itemsets.py

General Overview

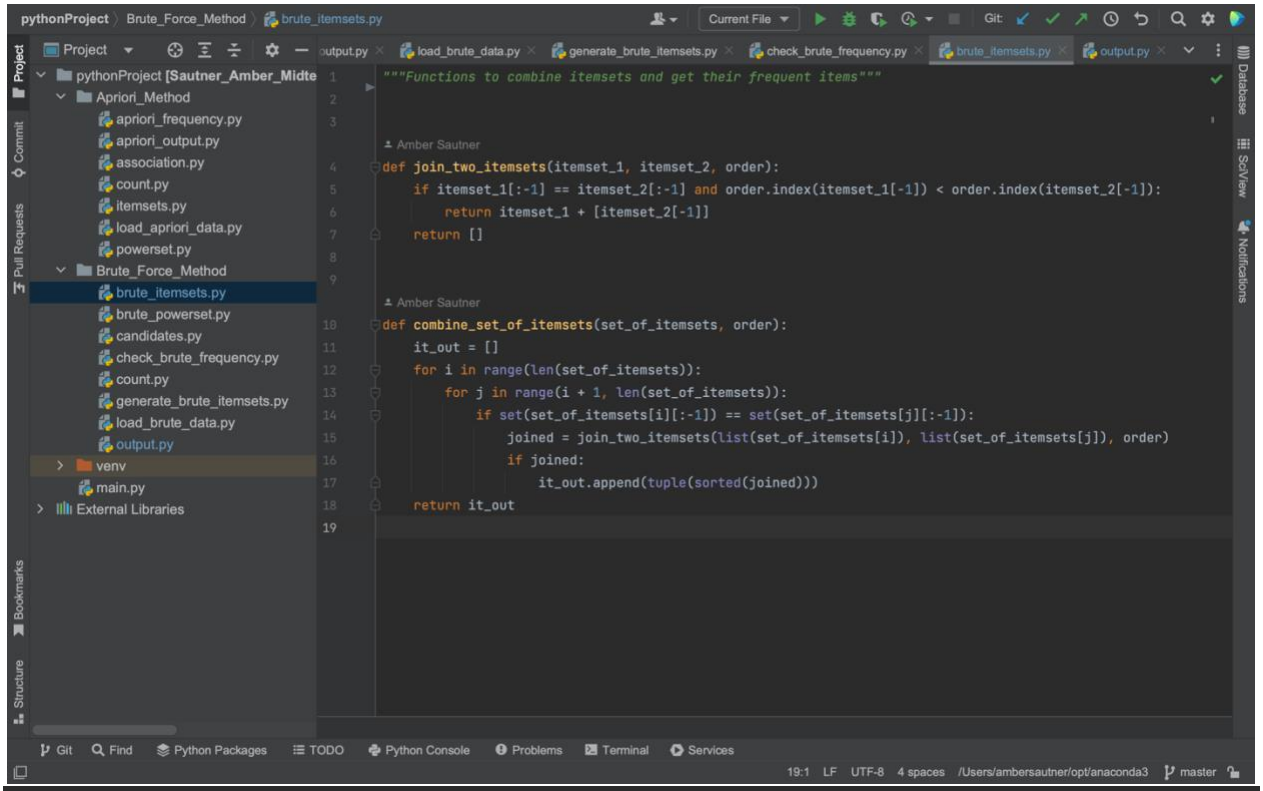
This file contains two functions that will help join itemsets.

Documentation

The first function join_two_itemsets will take three parameters; itemset_1, itemset_2, and order. It shows that when excluding the last element in each itemset, if itemset_1 is equal to itemset_2 and the order of itemset_1 is less than itemset_2, we will return a joined itemset between the two. This joined itemset is a combination of all the elements in itemset_1 and all except for the last element in itemset_2. If these two conditions are not met, then an empty list is returned.

The join_set_of_itemsets function takes two parameters, set_of_itemsets and order. It initializes an empty list it_out that will be used to store the joined itemsets.
The function then uses two nested loops to compare each pair of itemsets in set_of_itemsets and check if they are joinable. If the two itemsets are joinable, the join_two_itemsets function is called to create a new itemset which combines the two joinable itemsets. If a new itemset is created, it is added to the it_out list.
Finally, the function returns the it_out list containing all the joinable itemsets.

Source Code




```
"""Functions to combine itemsets and get their frequent items"""


def join_two_itemsets(itemset_1, itemset_2, order):
    if itemset_1[:-1] == itemset_2[:-1] and order.index(itemset_1[-1]) <
order.index(itemset_2[-1]):
        return itemset_1 + [itemset_2[-1]]
    return []


def combine_set_of_itemsets(set_of_itemsets, order):
    it_out = []
    for i in range(len(set_of_itemsets)):
        for j in range(i + 1, len(set_of_itemsets)):
            if set(set_of_itemsets[i][:-1]) == set(set_of_itemsets[j][:-1]):
                joined = join_two_itemsets(list(set_of_itemsets[i]),
list(set_of_itemsets[j]), order)
                if joined:
                    it_out.append(tuple(sorted(joined)))
    return it_out
```

Brute_powerset.py

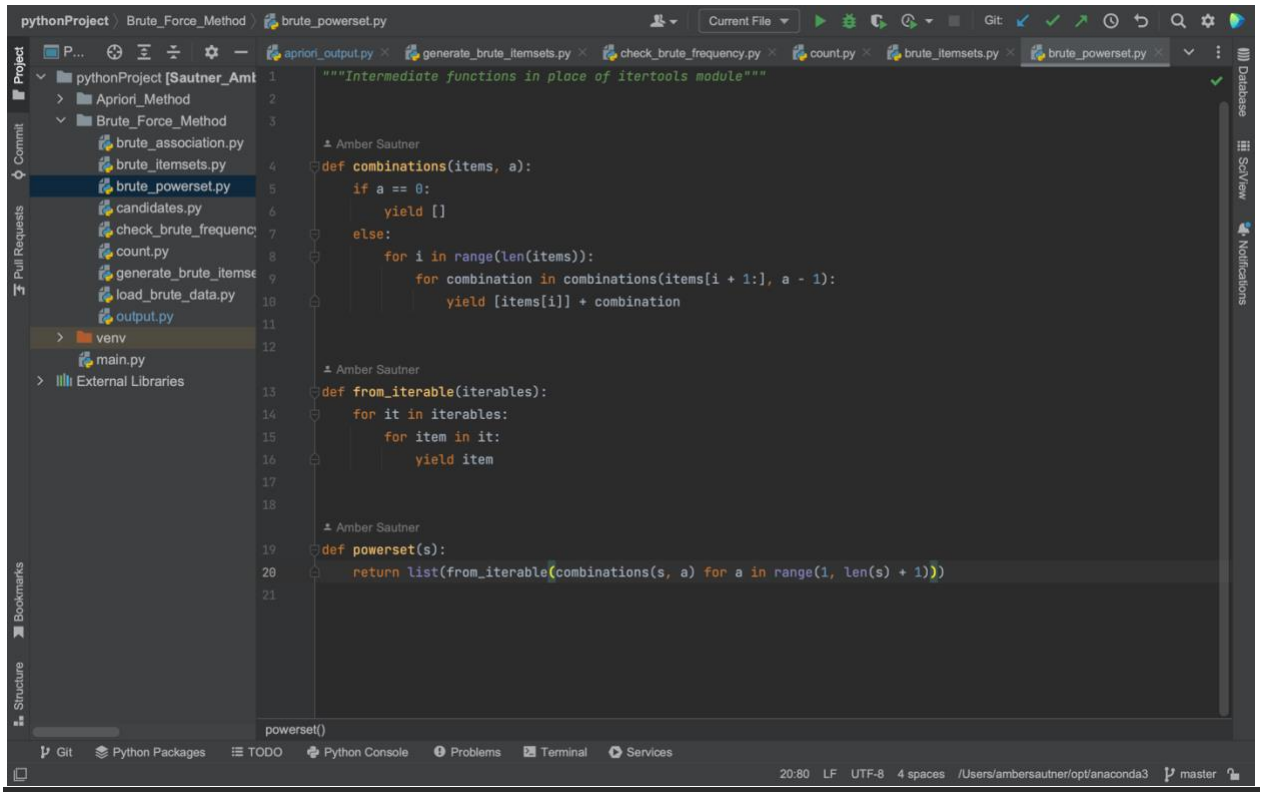
General Overview

This file contains three functions that serve to help the brute force method's combination process.

Documentation

This file is the same as the Apriori_powerset but was separated to created to explain the brute force method better. Look to the Apriori_powerset.py for the proper documentation.

Source Code




```
"""Intermediate functions in place of itertools module"""


def combinations(items, a):
    if a == 0:
        yield []
    else:
        for i in range(len(items)):
            for combination in combinations(items[i + 1:], a - 1):
                yield [items[i]] + combination


def from_iterable(iterables):
    for it in iterables:
        for item in it:
            yield item
```

```
def powerset(s):
    return list(from_iterable(combinations(s, a) for a in range(1, len(s) +
1)))
```

Candidates.py

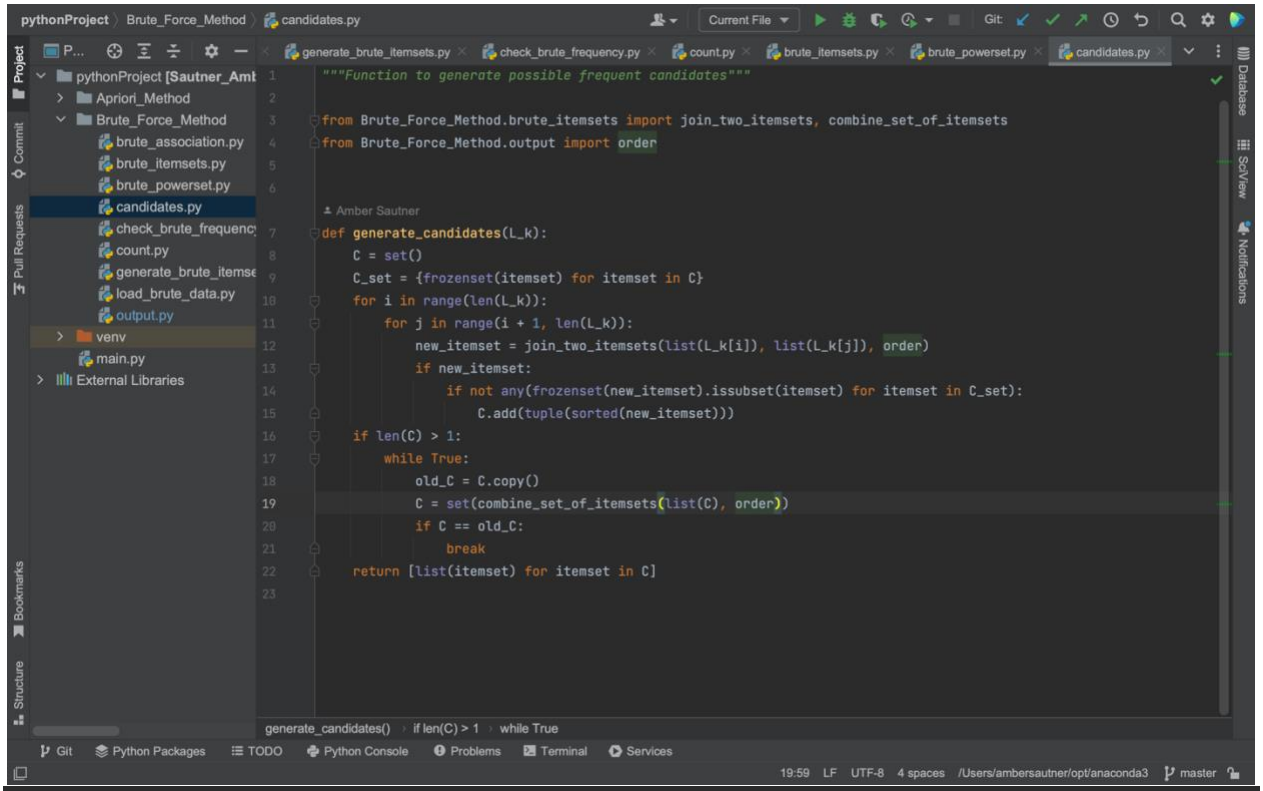
General Overview

This function will generate the possible frequent candidates.

Documentation

The function, generate_candidates, takes only one parameter, L_k, and begins by initializing C as an empty set and C_set as a set of frozensets that each contain an itemset from C. (Line 8-9) The two 'for loops' will iterate through all the pairs of itemsets in L_k and create a variable new_itemset. (Line 10-12) This variable calls the join_two_itemsets function on the two itemsets and uses the 'order' from load_brute_data. The following 'if statement' will check if the new itemset has any duplications and if it doesn't, then it is added to the sorted new_itemset. (Line 13-15) The next part will check if there are at least two itemsets in C, and start an infinite loop to save this copy. (Line 17-18) C will be updated to represent the joined pair of itemsets in C and if C is equal to the previously saved copy of C, then it breaks. (Line 20-21) Finally, a list of the itemsets in C is returned.

Source Code




```
"""Function to generate possible frequent candidates"""

from Brute_Force_Method.brute_itemsets import join_two_itemsets,
combine_set_of_itemsets
from Brute_Force_Method.output import order
```

```
def generate_candidates(L_k):
    C = set()
    C_set = {frozenset(itemset) for itemset in C}
    for i in range(len(L_k)):
        for j in range(i + 1, len(L_k)):
            new_itemset = join_two_itemsets(list(L_k[i]), list(L_k[j]),
order)
            if new_itemset:
                if not any(frozenset(new_itemset).issubset(itemset) for
itemset in C_set):
                    C.add(tuple(sorted(new_itemset)))
    if len(C) > 1:
        while True:
            old_C = C.copy()
            C = set(combine_set_of_itemsets(list(C), order))
            if C == old_C:
                break
    return [list(itemset) for itemset in C]
```

Output.py

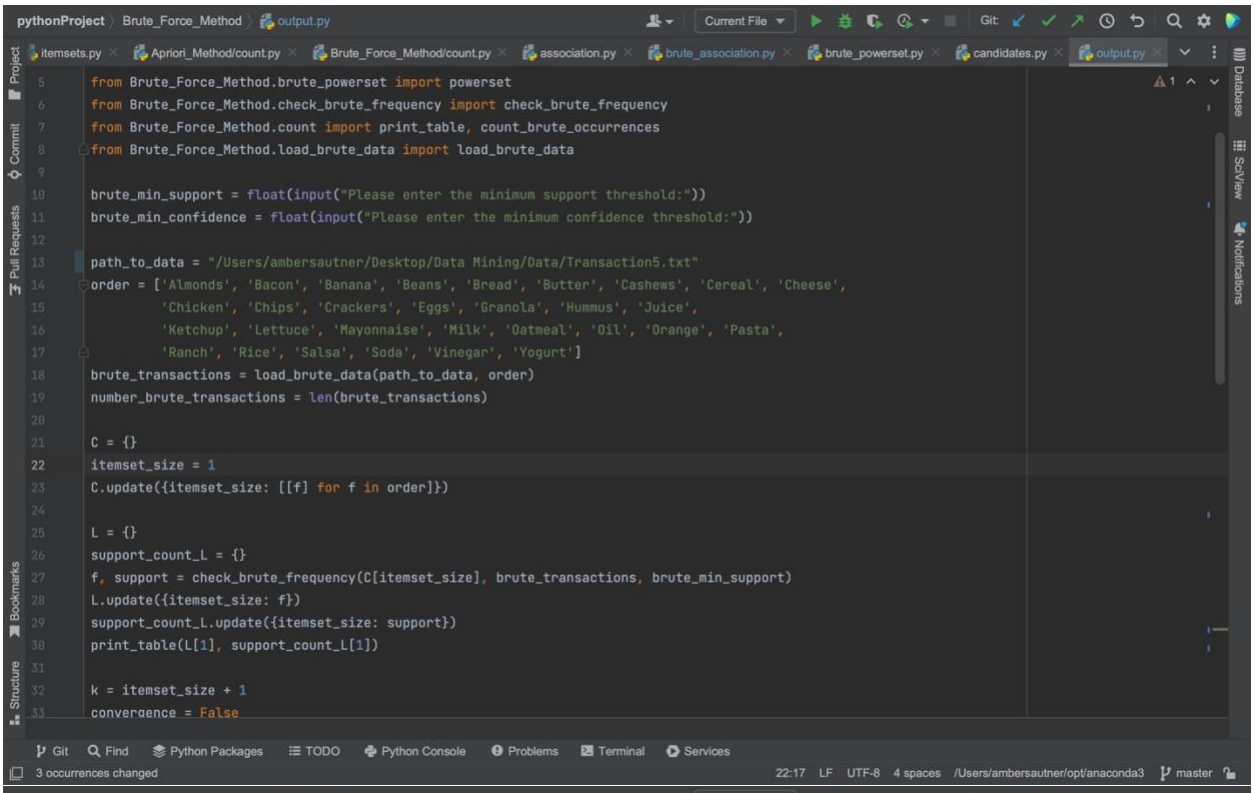
General Overview

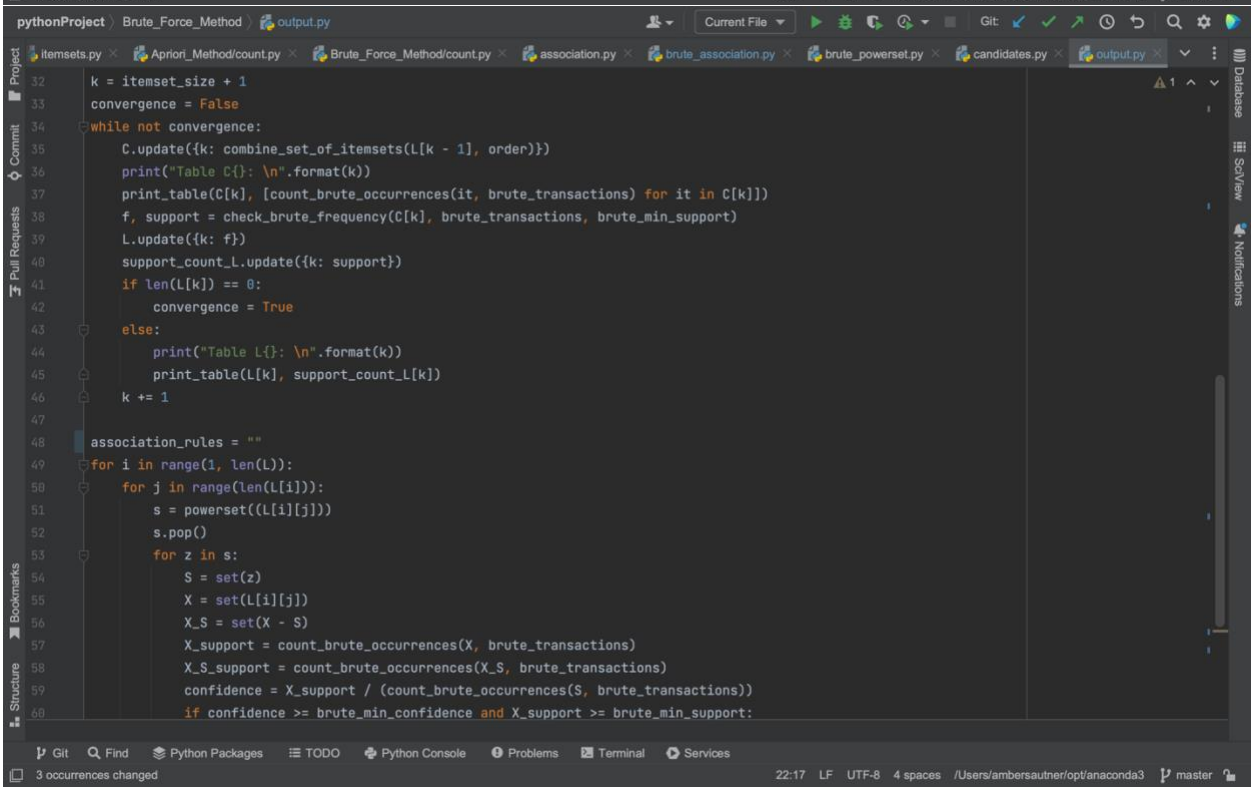
The output of this file will match the output from the Apriori method but is different in the code. It will combine all the previous functions to generate the association rules after looking at each itemset.

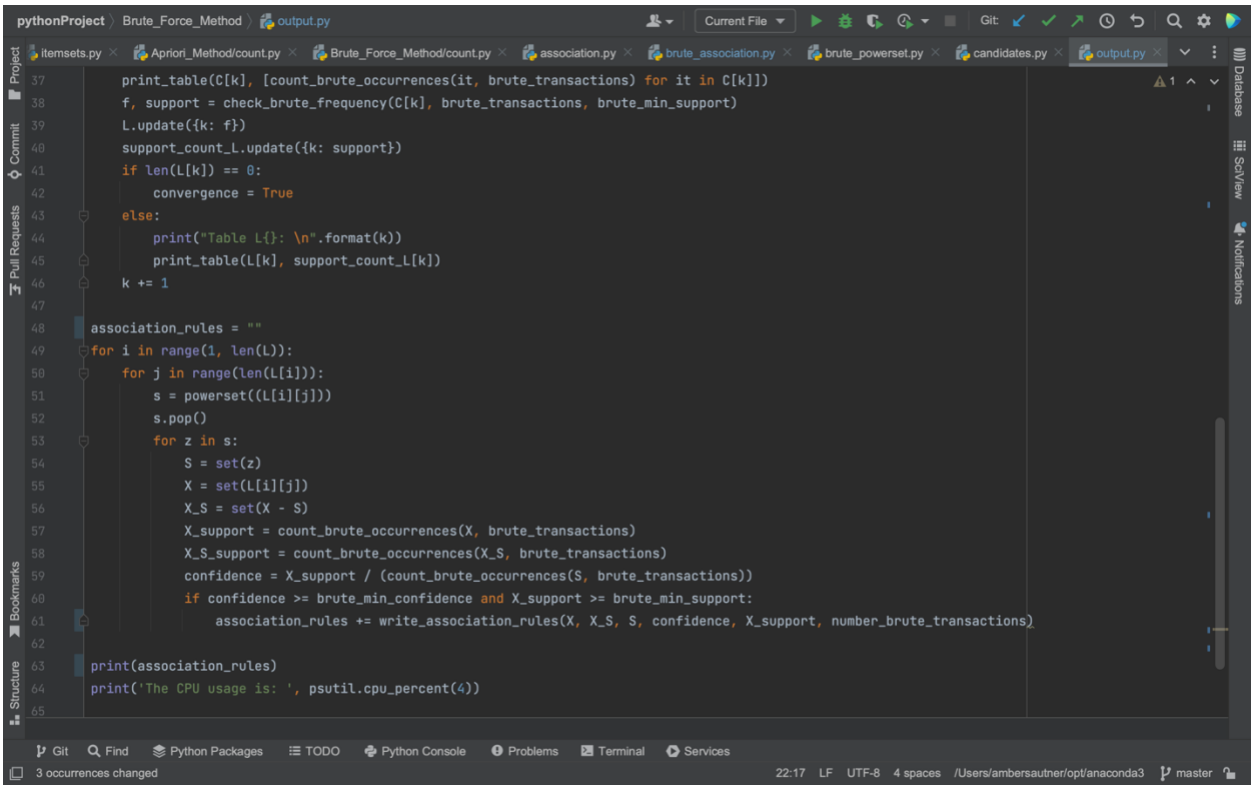
Documentation

This file is the same as the code in Apriori_output except for some changes. Since the brute force method generates all possible combinations of itemsets and does not prune anything, we can remove the discarded and newly_discarded variables. Like in the Apriori algorithm, it starts by generating all possible 1-itemsets and checks whether they meet the minimum support threshold to be deemed frequent. However, it does not discard them and continues onto generating itemsets of size 2 from the frequent 1-itemsets and checking their frequency. This will continue for larger itemset sizes until no more frequent itemsets can be found. (Line 34-46) With the frequent itemsets now obtained, we can generate the association rules by iterating through all possible subsets of each frequent itemset and calculating the confidence of each rule. The rules that meet the user-defined minimum support and confidence thresholds will be printed out. Finally, we print out the CPU usage as a percentage to compare to Apriori.

Source Code


pythonProject
Brute_Force_Method
output.py
Current File
Git:
itemsets.py
Apriori_Method/count.py
Brute_Force_Method/count.py
association.py
brute_association.py
brute_powerset.py
candidates.py
output.py
from Brute_Force_Method.brute_powerset import powerset
from Brute_Force_Method.check_brute_frequency import check_brute_frequency
from Brute_Force_Method.count import print_table, count_brute_occurrences
from Brute_Force_Method.load_brute_data import load_brute_data
brute_min_support = float(input("Please enter the minimum support threshold:"))
brute_min_confidence = float(input("Please enter the minimum confidence threshold:"))
path_to_data = "/Users/ambersautner/Desktop/Data Mining/Data/Transaction5.txt"
order = ['Almonds', 'Bacon', 'Banana', 'Beans', 'Bread', 'Butter', 'Cashews', 'Cereal', 'Cheese',
'Chicken', 'Chips', 'Crackers', 'Eggs', 'Granola', 'Hummus', 'Juice',
'Ketchup', 'Lettuce', 'Mayonnaise', 'Milk', 'Oatmeal', 'Oil', 'Orange', 'Pasta',
'Ranch', 'Rice', 'Salsa', 'Soda', 'Vinegar', 'Yogurt']
brute_transactions = load_brute_data(path_to_data, order)
number_brute_transactions = len(brute_transactions)
C = {}
itemset_size = 1
C.update({itemset_size: [[f] for f in order]})
L = {}
support_count_L = {}
f, support = check_brute_frequency(C[itemset_size], brute_transactions, brute_min_support)
L.update({itemset_size: f})
support_count_L.update({itemset_size: support})
print_table(L[1], support_count_L[1])
k = itemset_size + 1
convergence = False
Git
Find
Python Packages
TODO
Python Console
Problems
Terminal
Services
3 occurrences changed
22:17
LF
UTF-8
4 spaces
/Users/ambersautner/opt/anaconda3
master


pythonProject
Brute_Force_Method
output.py
Current File
Git:
itemsets.py
Apriori_Method/count.py
Brute_Force_Method/count.py
association.py
brute_association.py
brute_powerset.py
candidates.py
output.py
k = itemset_size + 1
convergence = False
while not convergence:
C.update({k: combine_set_of_itemsets(L[k - 1], order)})
print("Table C{}: \n".format(k))
print_table(C[k], [count_brute_occurrences(it, brute_transactions) for it in C[k]])
f, support = check_brute_frequency(C[k], brute_transactions, brute_min_support)
L.update({k: f})
support_count_L.update({k: support})
if len(L[k]) == 0:
convergence = True
else:
print("Table L{}: \n".format(k))
print_table(L[k], support_count_L[k])
k += 1
association_rules = ""
for i in range(1, len(L)):
for j in range(len(L[i])):
s = powerset((L[i][j]))
s.pop()
for z in s:
S = set(z)
X = set(L[i][j])
X_S = set(X - S)
X_support = count_brute_occurrences(X, brute_transactions)
X_S_support = count_brute_occurrences(X_S, brute_transactions)
confidence = X_support / (count_brute_occurrences(S, brute_transactions))
if confidence >= brute_min_confidence and X_support >= brute_min_support:
Git
Find
Python Packages
TODO
Python Console
Problems
Terminal
Services
3 occurrences changed
22:17
LF
UTF-8
4 spaces
/Users/ambersautner/opt/anaconda3
master




```
"""Output the brute force method"""
import psutil
from Brute_Force_Method.brute_association import write_association_rules
from Brute_Force_Method.brute_itemsets import combine_set_of_itemsets
from Brute_Force_Method.brute_powerset import powerset
from Brute_Force_Method.check_brute_frequency import check_brute_frequency
from Brute_Force_Method.count import print_table, count_brute_occurrences
from Brute_Force_Method.load_brute_data import load_brute_data

brute_min_support = float(input("Please enter the minimum support
threshold:"))
brute_min_confidence = float(input("Please enter the minimum confidence
threshold:"))

path_to_data = "/Users/ambersautner/Desktop/Data
Mining/Data/Transaction5.txt"
order = ['Almonds', 'Bacon', 'Banana', 'Beans', 'Bread', 'Butter', 'Cashews',
'Cereal', 'Cheese',
         'Chicken', 'Chips', 'Crackers', 'Eggs', 'Granola', 'Hummus',
'Juice',
         'Ketchup', 'Lettuce', 'Mayonnaise', 'Milk', 'Oatmeal', 'Oil',
'Orange', 'Pasta',
         'Ranch', 'Rice', 'Salsa', 'Soda', 'Vinegar', 'Yogurt']
brute_transactions = load_brute_data(path_to_data, order)
number_brute_transactions = len(brute_transactions)

C = {}
itemset_size = 1
C.update({itemset_size: [[f] for f in order]})
```

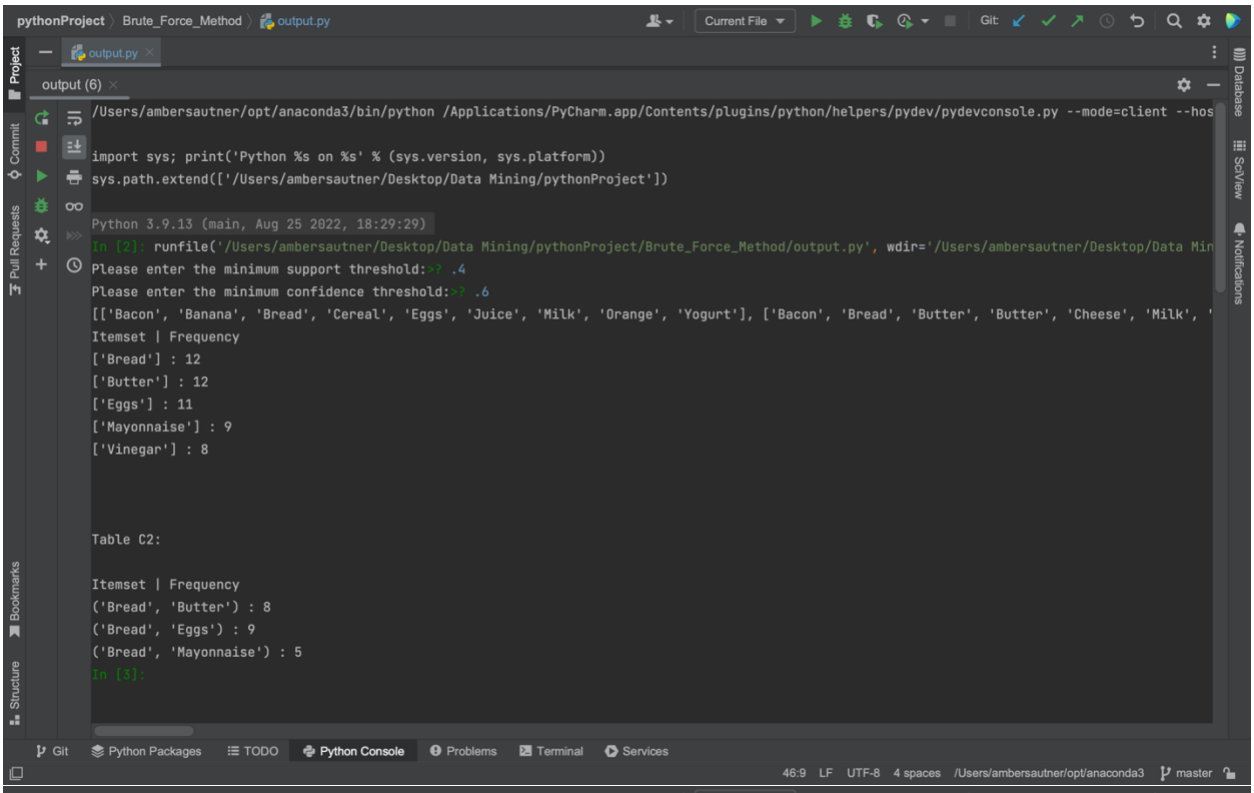
```
L = {}
support_count_L = {}
f, support = check_brute_frequency(C[itemset_size], brute_transactions,
brute_min_support)
L.update({itemset_size: f})
support_count_L.update({itemset_size: support})
print_table(L[1], support_count_L[1])

k = itemset_size + 1
convergence = False
while not convergence:
    C.update({k: combine_set_of_itemsets(L[k - 1], order)})
    print("Table C{}: \n".format(k))
    print_table(C[k], [count_brute_occurrences(it, brute_transactions) for it
in C[k]])
    f, support = check_brute_frequency(C[k], brute_transactions,
brute_min_support)
    L.update({k: f})
    support_count_L.update({k: support})
    if len(L[k]) == 0:
        convergence = True
    else:
        print("Table L{}: \n".format(k))
        print_table(L[k], support_count_L[k])
    k += 1

association_rules = ""
for i in range(1, len(L)):
    for j in range(len(L[i])):
        s = powerset((L[i][j]))
        s.pop()
        for z in s:
            S = set(z)
            X = set(L[i][j])
            X_S = set(X - S)
            X_support = count_brute_occurrences(X, brute_transactions)
            X_S_support = count_brute_occurrences(X_S, brute_transactions)
            confidence = X_support / (count_brute_occurrences(S,
brute_transactions))
            if confidence >= brute_min_confidence and X_support >=
brute_min_support:
                association_rules += write_association_rules(X, X_S, S,
confidence, X_support, number_brute_transactions)

print(association_rules)
print('The CPU usage is: ', psutil.cpu_percent(4))
```

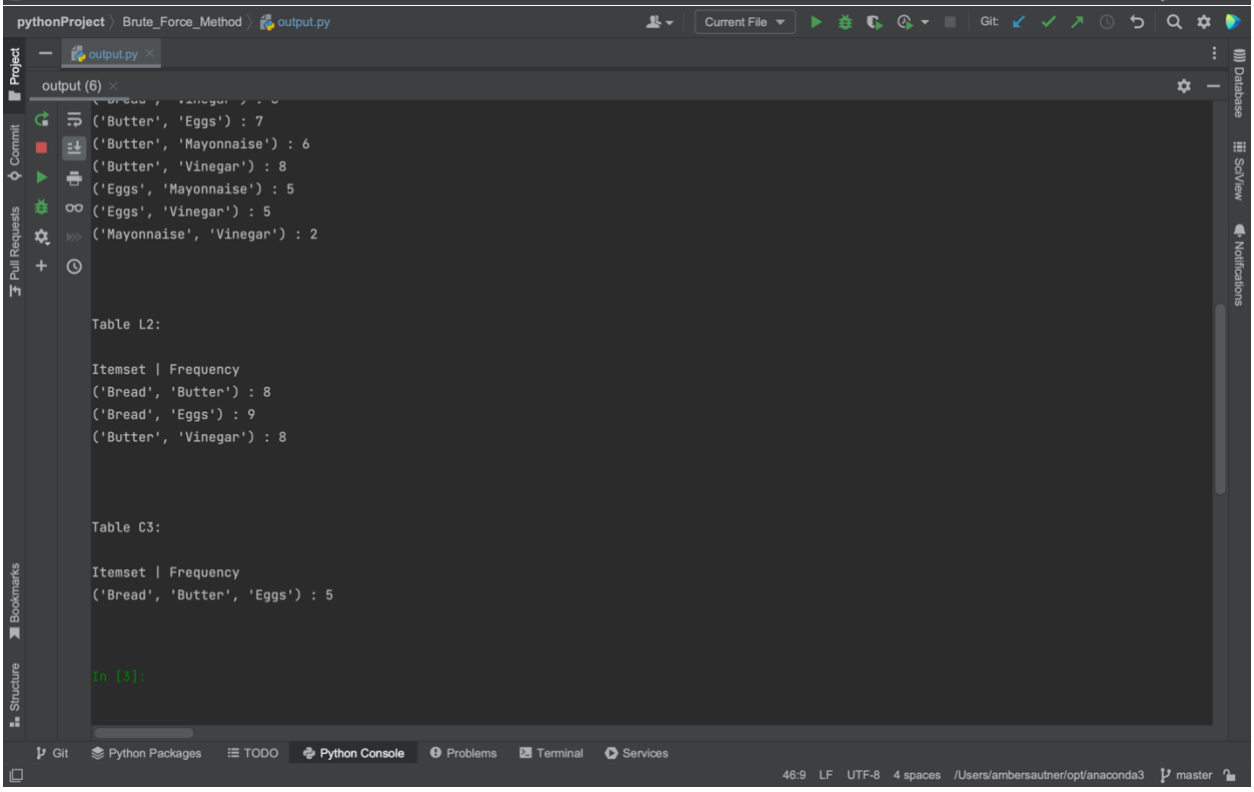
User Application

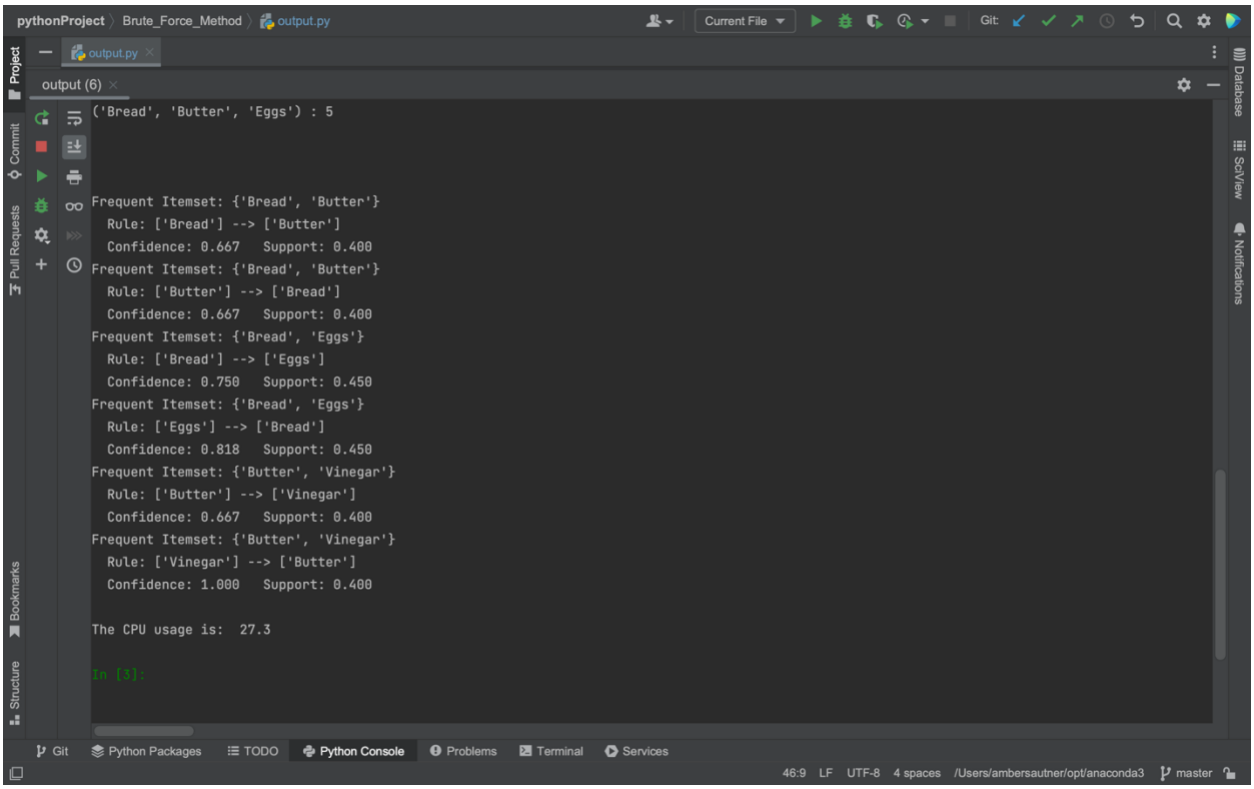
Transaction1.txt Examples

Example 1

Minimum Support = .4
Minimum Confidence = .6
CPU = 27.3

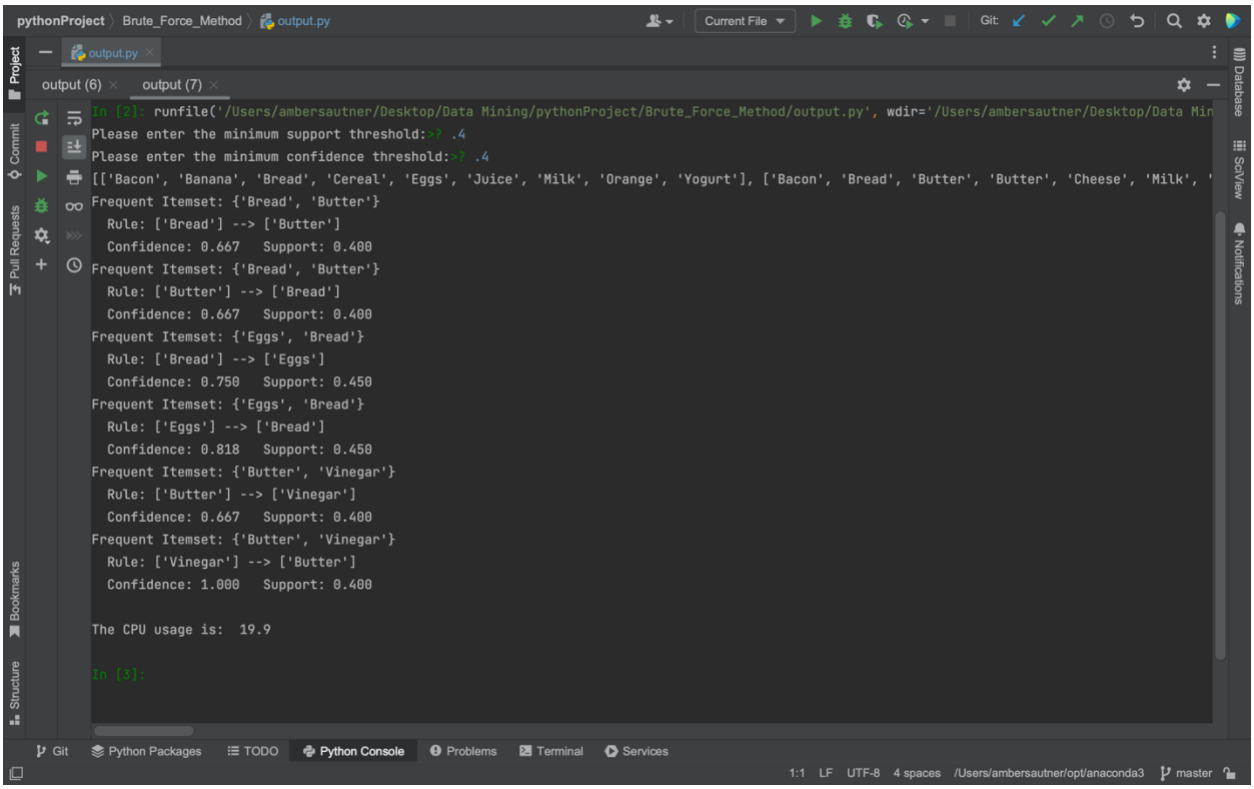

pythonProject
Brute_Force_Method
output.py
Current File
Git:
output (6)
/Users/ambersautner/opt/anaconda3/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=client --hos
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/ambersautner/Desktop/Data Mining/pythonProject'])
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
In [2]: runfile('/Users/ambersautner/Desktop/Data Mining/pythonProject/Brute_Force_Method/output.py', wdir='/Users/ambersautner/Desktop/Data Min
Please enter the minimum support threshold:.4
Please enter the minimum confidence threshold:.6
[['Bacon', 'Banana', 'Bread', 'Cereal', 'Eggs', 'Juice', 'Milk', 'Orange', 'Yogurt'], ['Bacon', 'Bread', 'Butter', 'Butter', 'Cheese', 'Milk', '
Itemset | Frequency
['Bread'] : 12
['Butter'] : 12
['Eggs'] : 11
['Mayonnaise'] : 9
['Vinegar'] : 8
Table C2:
Itemset | Frequency
('Bread', 'Butter') : 8
('Bread', 'Eggs') : 9
('Bread', 'Mayonnaise') : 5
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
46:9 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master


pythonProject
Brute_Force_Method
output.py
Current File
Git:
output (6)
('Butter', 'Eggs') : 7
('Butter', 'Mayonnaise') : 6
('Butter', 'Vinegar') : 8
('Eggs', 'Mayonnaise') : 5
('Eggs', 'Vinegar') : 5
('Mayonnaise', 'Vinegar') : 2
Table L2:
Itemset | Frequency
('Bread', 'Butter') : 8
('Bread', 'Eggs') : 9
('Butter', 'Vinegar') : 8
Table C3:
Itemset | Frequency
('Bread', 'Butter', 'Eggs') : 5
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
46:9 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master



Example 2

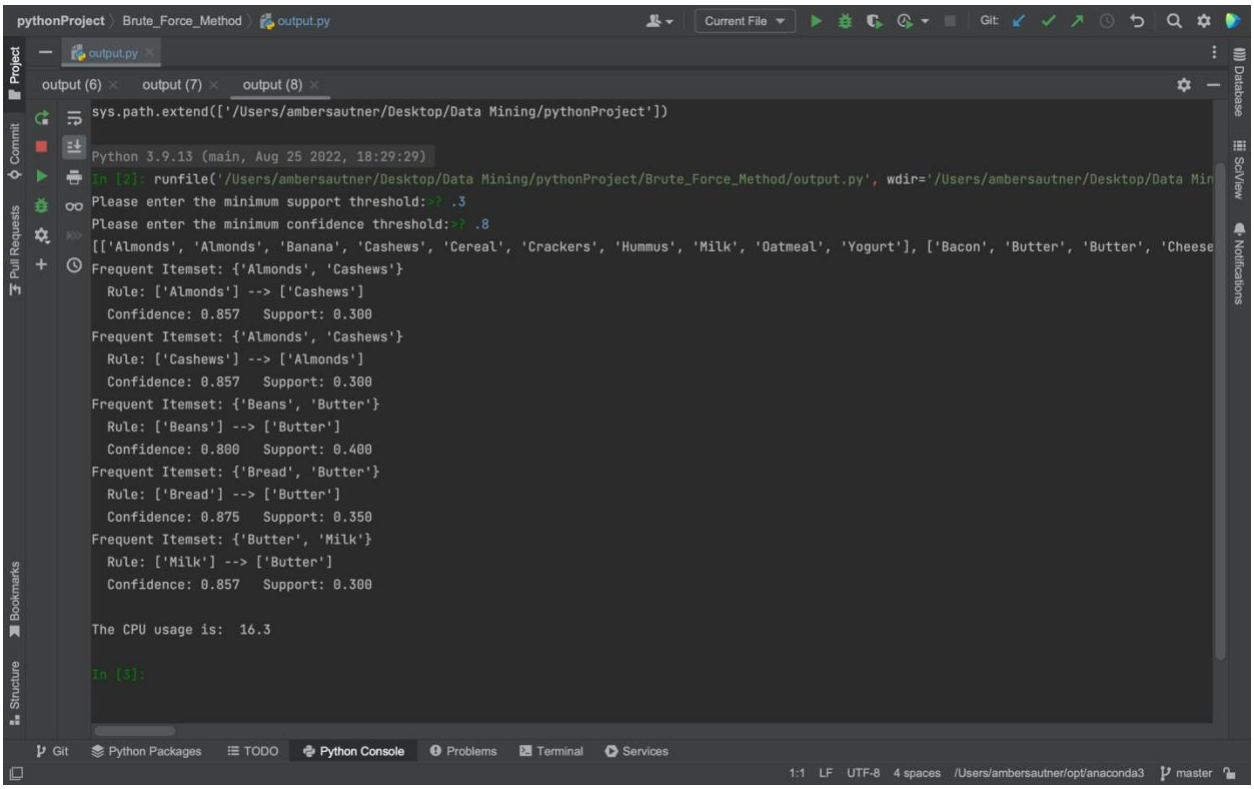
Minimum Support = .4
Minimum Confidence =.4
CPU = 19.9



Transaction2.txt Examples

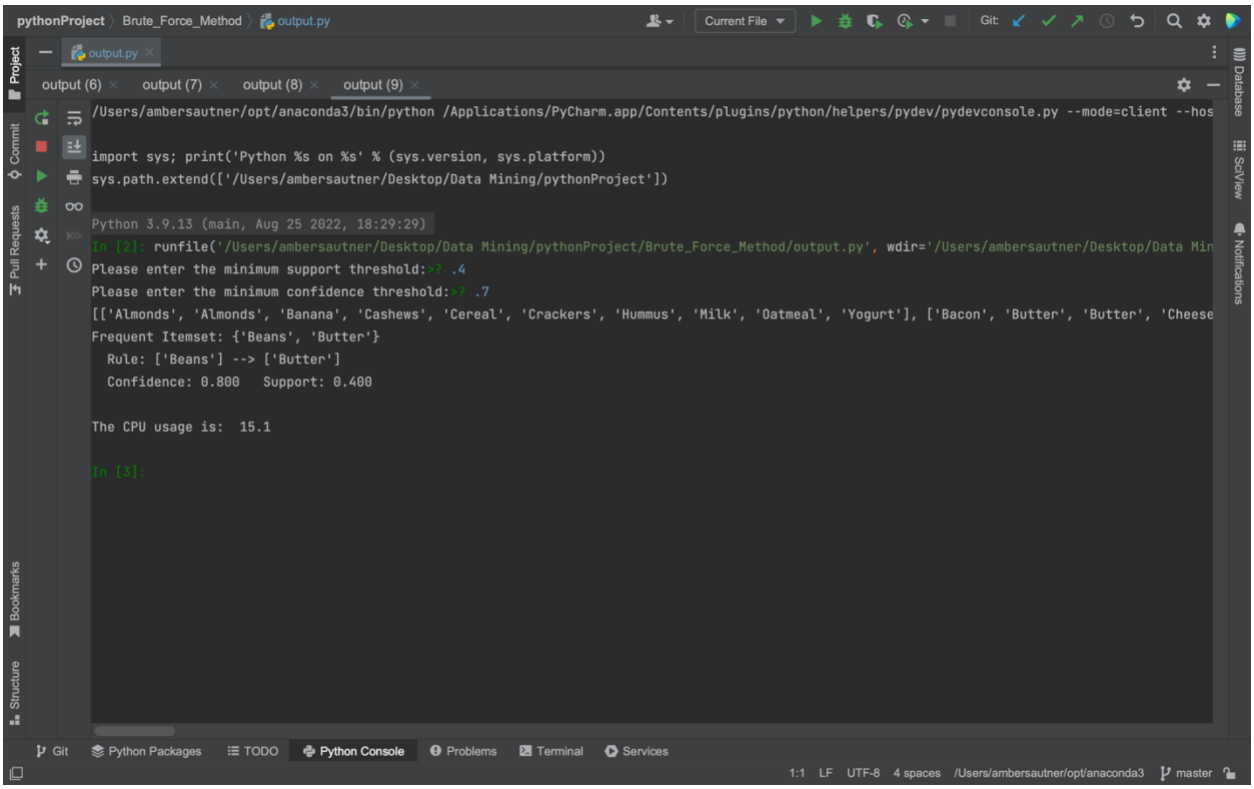
Example 1

Minimum Support = .3
Minimum Confidence = .8
CPU = 16.3



Example 2

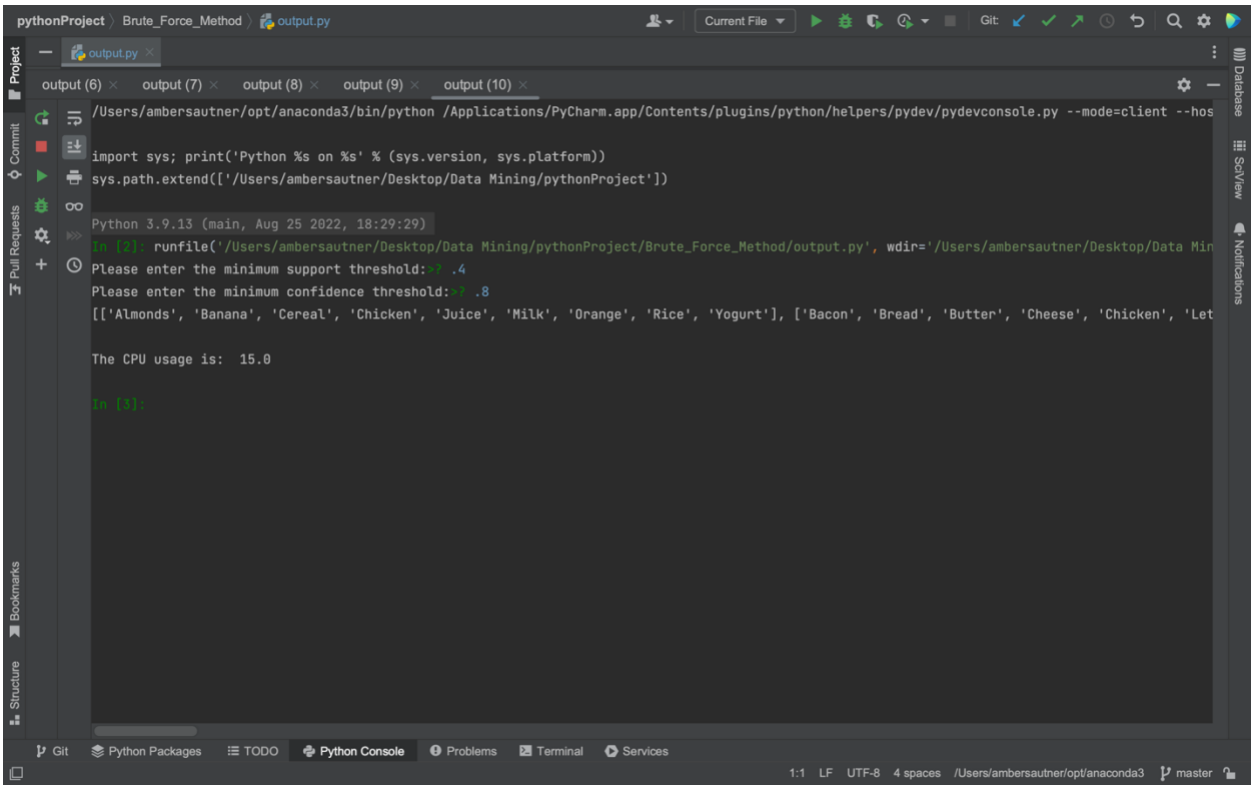
Minimum Support = .4
Minimum Confidence =.7
CPU = 15.1



Transaction3.txt Examples

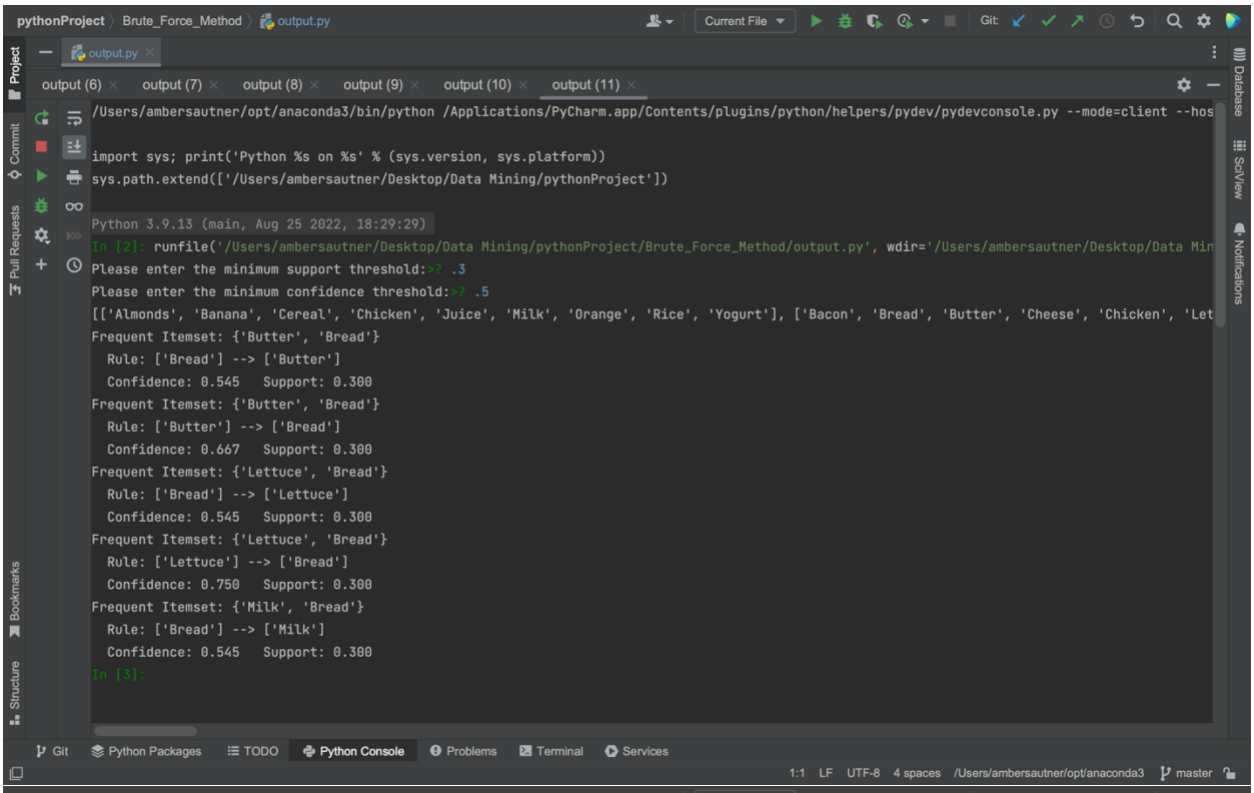
Example 1

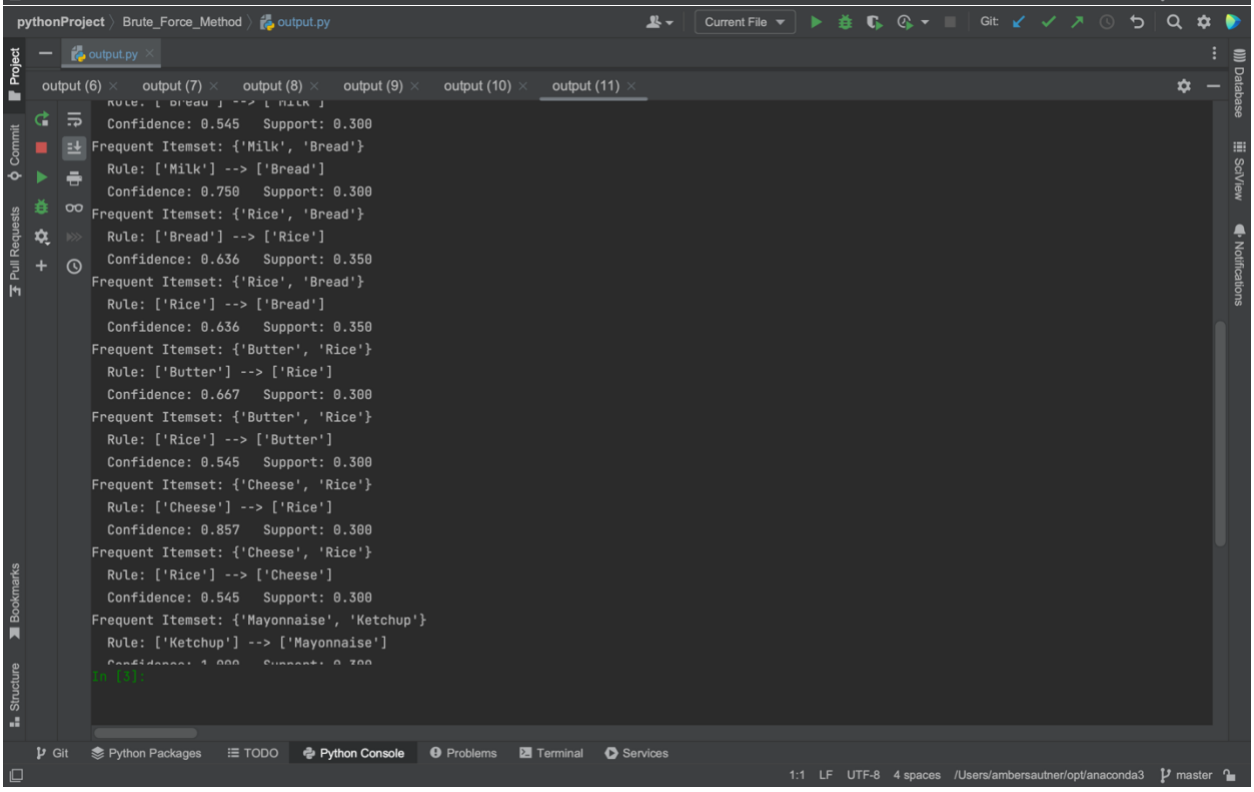
Minimum Support = .4
Minimum Confidence = .8
CPU = 15.0

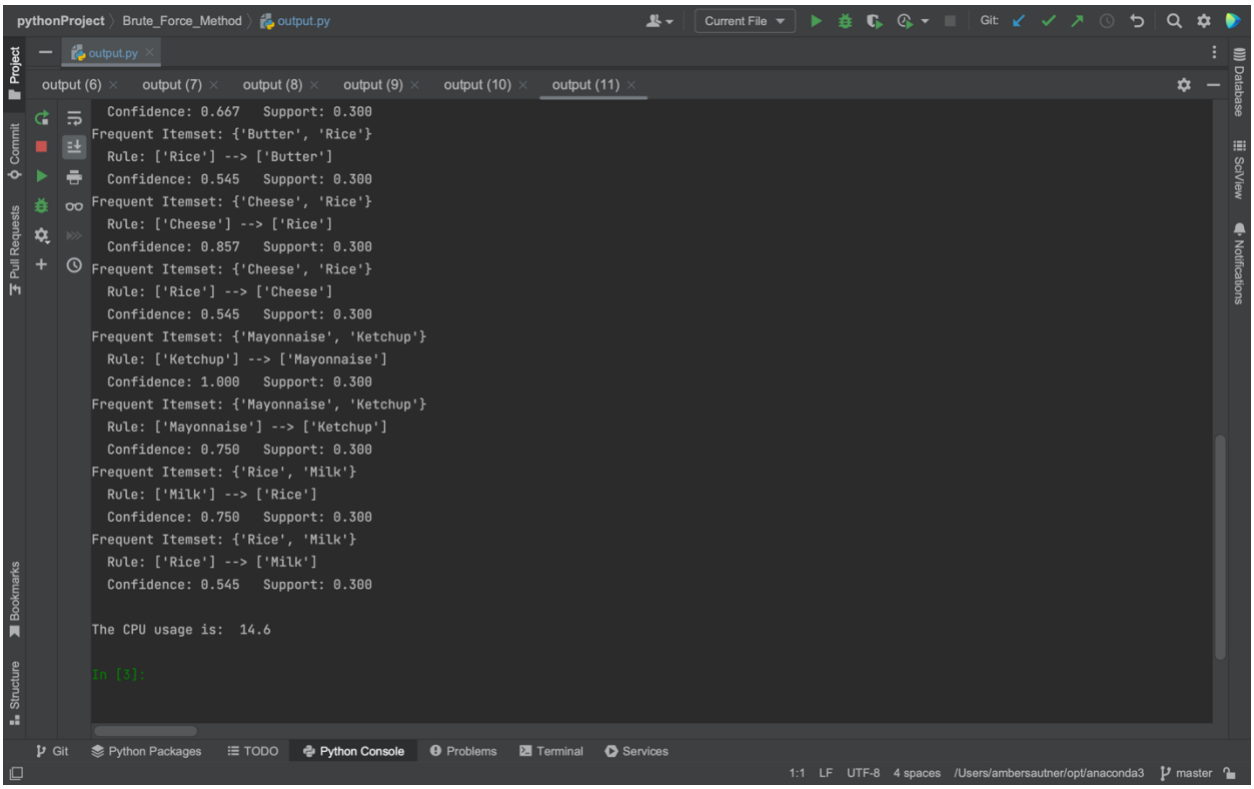


Example 2

Minimum Support = .3
Minimum Confidence = .5
CPU = 14.6


pythonProject
Brute_Force_Method
output.py
Current File
Git:
output (6)
output (7)
output (8)
output (9)
output (10)
output (11)
/Users/ambersautner/opt/anaconda3/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=client --hos
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/ambersautner/Desktop/Data Mining/pythonProject'])
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
In [2]: runfile('/Users/ambersautner/Desktop/Data Mining/pythonProject/Brute_Force_Method/output.py', wdir='/Users/ambersautner/Desktop/Data Min
Please enter the minimum support threshold:?? .3
Please enter the minimum confidence threshold:?? .5
[['Almonds', 'Banana', 'Cereal', 'Chicken', 'Juice', 'Milk', 'Orange', 'Rice', 'Yogurt'], ['Bacon', 'Bread', 'Butter', 'Cheese', 'Chicken', 'Let
Frequent Itemset: {'Butter', 'Bread'}
Rule: ['Bread'] --> ['Butter']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Butter', 'Bread'}
Rule: ['Butter'] --> ['Bread']
Confidence: 0.667 Support: 0.300
Frequent Itemset: {'Lettuce', 'Bread'}
Rule: ['Bread'] --> ['Lettuce']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Lettuce', 'Bread'}
Rule: ['Lettuce'] --> ['Bread']
Confidence: 0.750 Support: 0.300
Frequent Itemset: {'Milk', 'Bread'}
Rule: ['Bread'] --> ['Milk']
Confidence: 0.545 Support: 0.300
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master

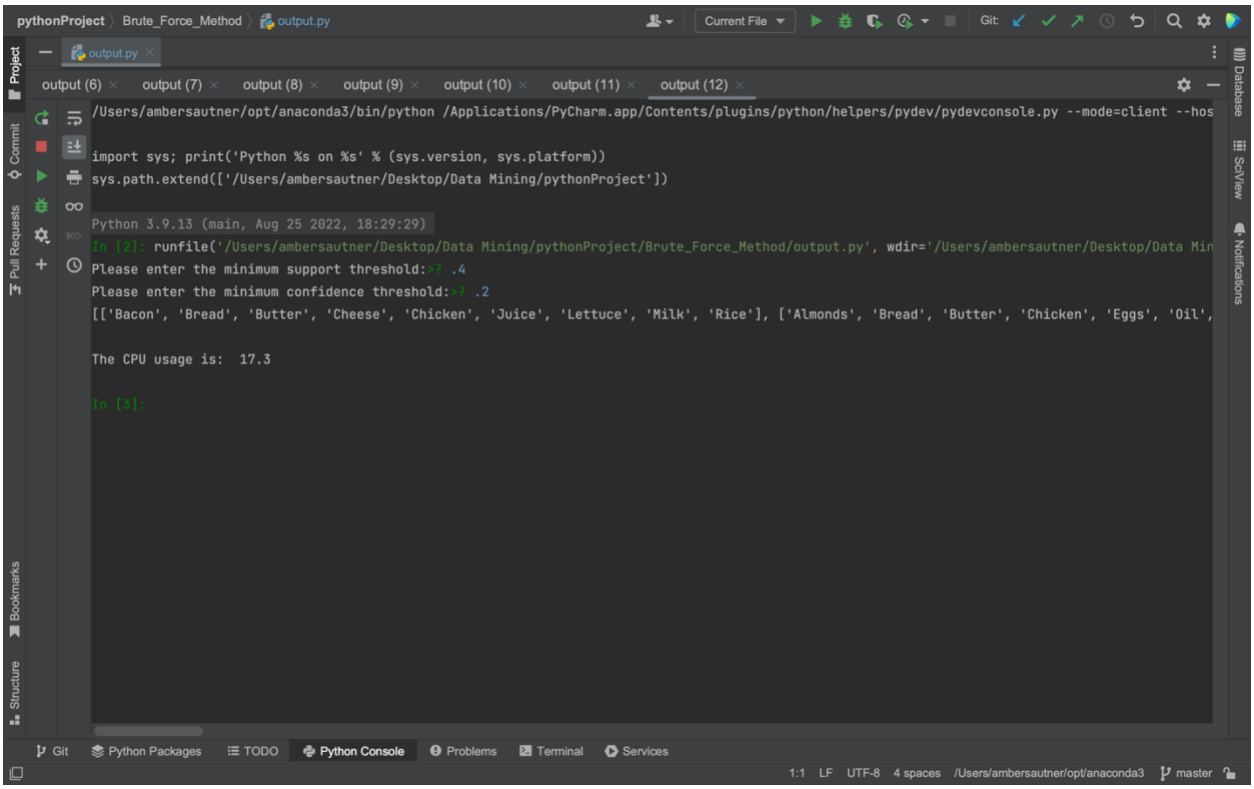

pythonProject
Brute_Force_Method
output.py
Current File
Git:
output (6)
output (7)
output (8)
output (9)
output (10)
output (11)
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Milk', 'Bread'}
Rule: ['Milk'] --> ['Bread']
Confidence: 0.750 Support: 0.300
Frequent Itemset: {'Rice', 'Bread'}
Rule: ['Bread'] --> ['Rice']
Confidence: 0.636 Support: 0.350
Frequent Itemset: {'Rice', 'Bread'}
Rule: ['Rice'] --> ['Bread']
Confidence: 0.636 Support: 0.350
Frequent Itemset: {'Butter', 'Rice'}
Rule: ['Butter'] --> ['Rice']
Confidence: 0.667 Support: 0.300
Frequent Itemset: {'Butter', 'Rice'}
Rule: ['Rice'] --> ['Butter']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Cheese', 'Rice'}
Rule: ['Cheese'] --> ['Rice']
Confidence: 0.857 Support: 0.300
Frequent Itemset: {'Cheese', 'Rice'}
Rule: ['Rice'] --> ['Cheese']
Confidence: 0.545 Support: 0.300
Frequent Itemset: {'Mayonnaise', 'Ketchup'}
Rule: ['Ketchup'] --> ['Mayonnaise']
In [3]:
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
1:1 LF UTF-8 4 spaces /Users/ambersautner/opt/anaconda3 master



Transaction4.txt Examples

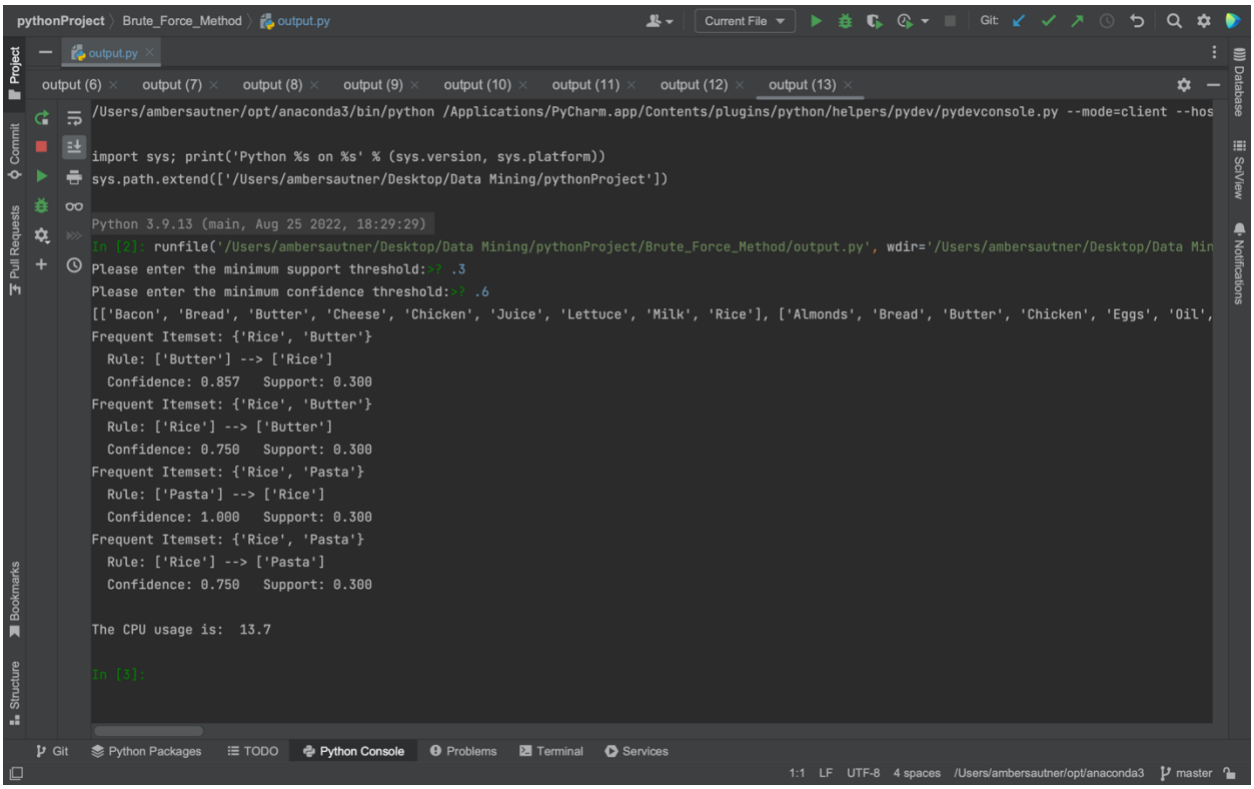
Example 1

Minimum Support = .4
Minimum Confidence = .2
CPU = 17.3



Example 2

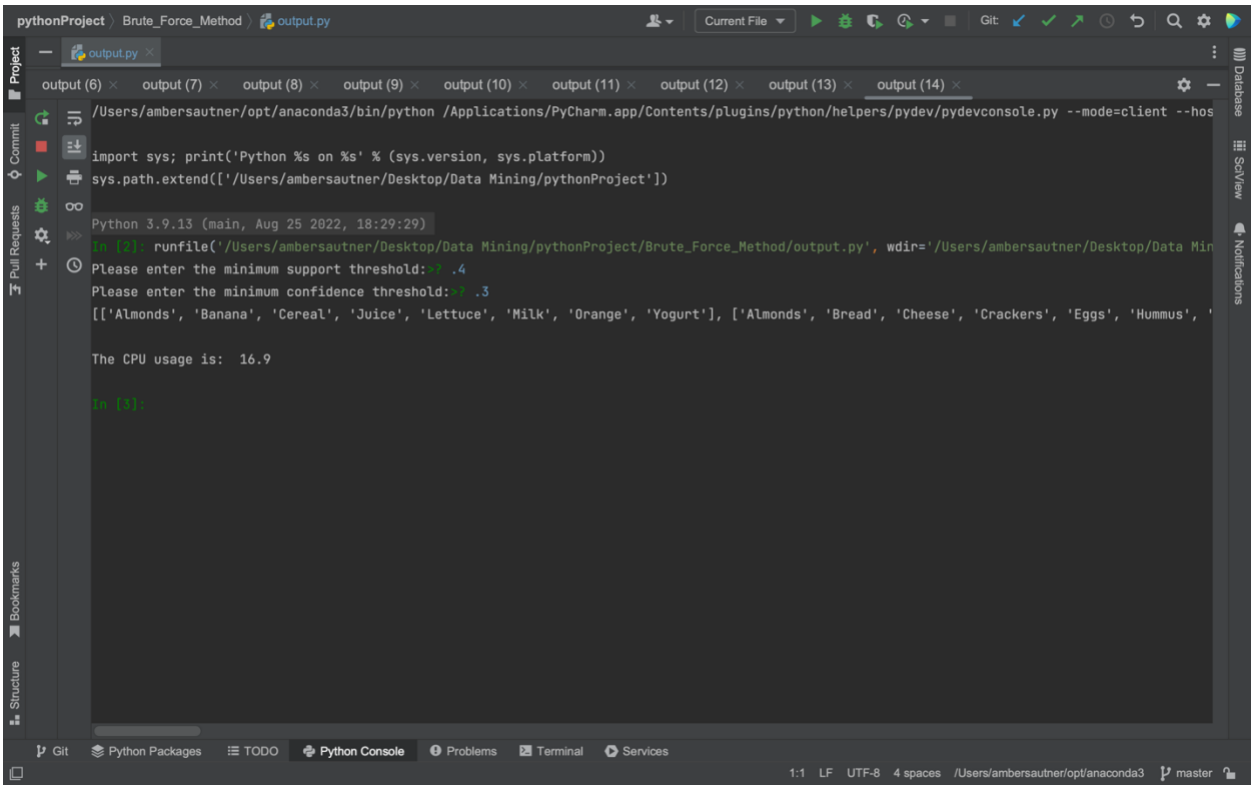
Minimum Support = .3
Minimum Confidence = .6
CPU = 13.7



Transaction5.txt Examples

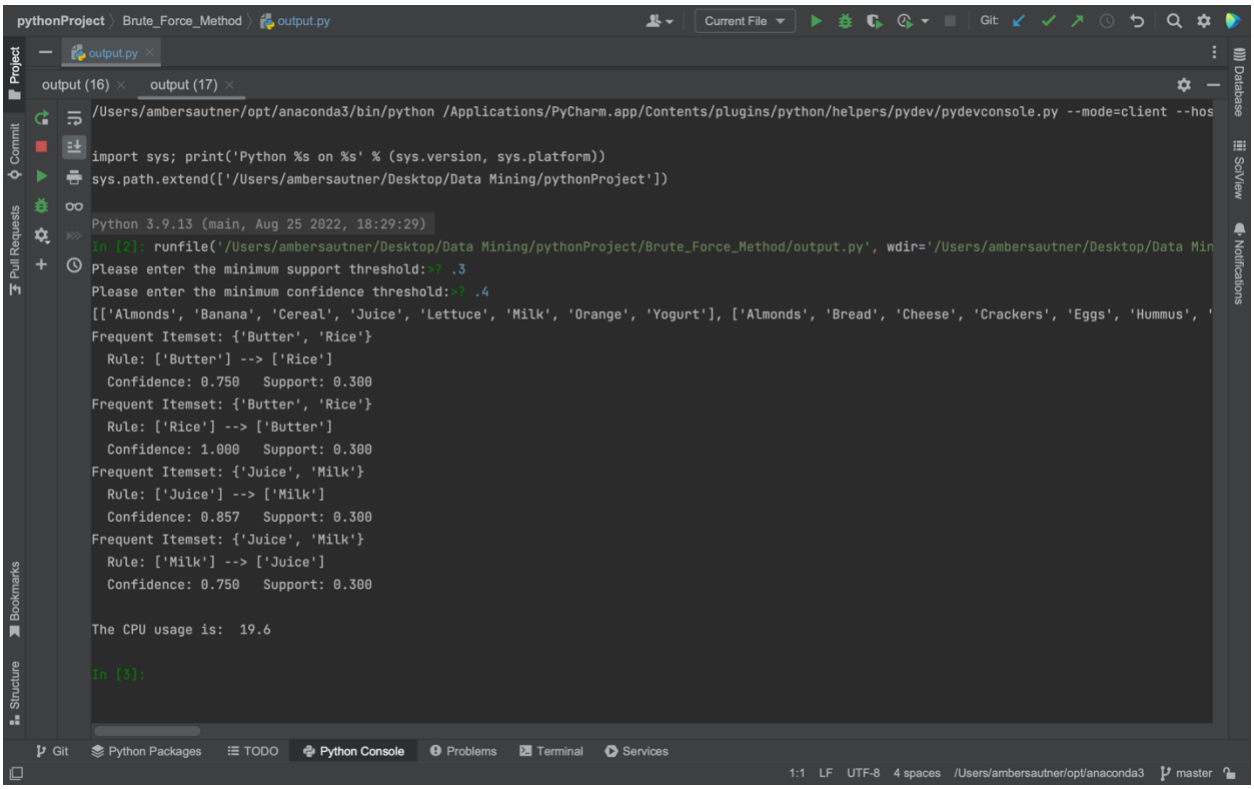
Example 1

Minimum Support = .4
Minimum Confidence = .3
CPU = 16.9



Example 2

Minimum Support = .3
Minimum Confidence = .4
CPU = 19.6



Data Files:

Below are the 5 different transactions with 30 items sourced from FoodItems.txt

FoodItems.txt

```
Contains 30 different items one would find at a grocery store.
Almonds
Bacon
Banana
Beans
Bread
Butter
Cereal
Cheese
Chicken
Chips
Crackers
Eggs
Granola
Hummus
Juice
Ketchup
Lettuce
Mayonnaise
Milk
Oil
Orange
Oatmeal
```

Pasta
Ranch
Rice
Salsa
Soda
Vinegar
Yogurt

Transaction1.txt

Contains 20 unique transactions using the 30 items from FoodItems.txt

Milk,Cereal,Yogurt,Banana,Orange,Juice,Bread,Eggs,Bacon
Bread,Butter,Cheese,Bacon,Milk,Butter,Oil,Vinegar
Eggs,Butter,Lettuce,Chicken,Pasta,Mayonnaise,Ketchup
Orange,Almonds,Yogurt,Banana,Juice,Mayonnaise,Ketchup,Yogurt
Crackers,Hummus,Chips,Salsa,Soda,Juice,Bacon
Pasta,Cheese,Butter,Bread,Eggs,Oil,Vinegar
Chicken,Pasta,Butter,Bread,Eggs,Mayonnaise,Ketchup
Bread,Butter,Oil,Vinegar,Butter,Lettuce,Cheese
Oil,Vinegar,Lettuce,Butter,Ranch,Cheese,Bacon
Cereal,Cashews,Banana,Orange,Butter,Mayonnaise,Ketchup
Ketchup,Mayonnaise,Butter,Bread,Eggs,Oil,Vinegar
Eggs,Bacon,Bread,Milk,Butter,Juice,Oil,Vinegar
Chicken,Lettuce,Ranch,Mayonnaise,Cheese,Bacon
Oatmeal,Milk,Banana,Orange,Yogurt,Cereal,Bacon
Cheese,Crackers,Hummus,Salsa,Bread,Eggs,Mayonnaise,Ketchup
Salsa,Chips,Crackers,Juice,Soda,Bread,Eggs
Bread,Granola,Cheese,Mayonnaise,Cheese,Butter
Lettuce,Chicken,Vinegar,Mayonnaise,Butter,Bread,Eggs
Milk,Eggs,Oatmeal,Juice,Butter,Ranch,Oil,Vinegar
Yogurt,Banana,Orange,Cereal,Juice,Bread,Eggs

Transaction2.txt

Contains 20 unique transactions using the 30 items from FoodItems.txt

Almonds,Cashews,Cereal,Milk,Oatmeal,Almonds,Banana,Yogurt,Hummus,Crackers
Milk,Cheese,Butter,Pasta,Bacon,Yogurt,Granola,Eggs,Butter,Crackers
Bacon,Ranch,Chicken,Bread,Butter,Ranch,Mayonnaise,Hummus,Beans
Yogurt,Cereal,Milk,Juice,Bacon,Butter,Cheese,Lettuce,Bread,Mayonnaise
Eggs,Bread,Butter,Chicken,Cheese,Milk,Chicken,Hummus,Beans
Milk,Juice,Soda,Chips,Crackers,Hummus,Eggs,Butter,Beans
Ketchup,Lettuce,Mayonnaise,Vinegar,Butter,Hummus,Crackers
Rice,Bread,Pasta,Oatmeal,Almonds,Cashews,Butter,Beans
Oatmeal,Juice,Eggs,Banana,Almonds,Yogurt,Ketchup,Orange
Soda,Salsa,Ranch,Almonds,Cashews,Crackers,Hummus
Oil,Vinegar,Mayonnaise,Bread,Lettuce,Butter,Butter,Crackers,Beans
Orange,Banana,Lettuce,Cashews,Almonds,Eggs,Hummus,Crackers
Chips,Crackers,Soda,Salsa,Lettuce,Chicken,Ranch,Rice,Beans
Lettuce,Almonds,Cashews,Butter,Bread,Butter,Ranch,Ketchup
Vinegar,Oil,Butter,Ketchup,Mayonnaise,Eggs,Butter,Beans
Eggs,Ketchup,Bacon,Bread,Juice,Orange,Banana,Hummus,Beans
Rice,Salsa,Crackers,Oil,Chips,Soda,Eggs,Butter,Hummus,Beans
Yogurt,Banana,Cashews,Almonds,Milk,Cereal,Butter,Crackers
Chicken,Lettuce,Eggs,Milk,Butter,Oil,Vinegar,Bread,Hummus,Crackers
Pasta,Oil,Eggs,Vinegar,Chicken,Salsa,Cashews,Granola,Butter,Beans

Transaction3.txt

Contains 20 unique transactions using the 30 items from FoodItems.txt

```
Milk,Cereal,Yogurt,Banana,Orange,Juice,Almonds,Rice,Chicken
Bread,Butter,Cheese,Bacon,Milk,Lettuce,Rice,Chicken
Eggs,Butter,Chicken,Pasta,Bacon,Mayonnaise,Ketchup
Crackers,Hummus,Chips,Salsa,Soda,Cashews
Rice,Chicken,Pasta,Butter,Lettuce
Ketchup,Mayonnaise,Rice,Ranch,Cheese
Bread,Oil,Vinegar,Butter,Lettuce,Rice
Oatmeal,Milk,Banana,Yogurt,Cereal,Orange
Cheese,Crackers,Rice,Juice,Soda,Milk,Eggs,Bread
Salsa,Chips,Bacon,Soda,Juice,Mayonnaise,Ketchup
Lettuce,Chicken,Vinegar,Mayonnaise,Almonds
Yogurt,Banana,Orange,Cereal,Cereal,Bread
Milk,Eggs,Bread,Juice,Butter,Cheese,Rice,Oil
Chicken,Lettuce,Ranch,Mayonnaise,Bread
Cheese,Bacon,Pasta,Rice,Butter,Mayonnaise,Ketchup
Orange,Almonds,Yogurt,Banana,Juice,Bread,Rice,Chicken
Hummus,Chips,Rice,Soda,Lettuce,Banana,Milk,Eggs,Bread,Salsa
Cereal,Cashews,Orange,Juice,Lettuce,Milk,Eggs,Bread,Butter
Bread,Butter,Cheese,Mayonnaise,Lettuce,Oil,Ketchup
Pasta,Cheese,Rice,Butter,Rice,Mayonnaise,Ketchup,Milk,Eggs,Bread
```

Transaction4.txt

Contains 20 unique transactions using the 30 items from FoodItems.txt

```
Bread,Butter,Cheese,Bacon,Milk,Lettuce,Juice,Rice,Chicken
Eggs,Butter,Chicken,Pasta,Oil,Bread,Rice,Almonds,Yogurt
Cheese,Crackers,Hummus,Salsa,Almonds
Rice,Chicken,Pasta,Butter,Lettuce,Juice
Milk,Cereal,Yogurt,Banana,Orange,Almonds
Salsa,Chips,Beans,Juice,Soda,Milk
Lettuce,Chicken,Vinegar,Mayonnaise,Almonds,Bread
Oatmeal,Milk,Banana,Yogurt,Cereal,Orange
Cheese,Bacon,Pasta,Rice,Butter,Almonds,Yogurt,Cashews
Yogurt,Banana,Orange,Cereal,Milk,Rice,Chicken,Almonds
Crackers,Hummus,Chips,Salsa,Soda,Cashews
Bread,Butter,Cheese,Mayonnaise,Oil,Rice,Pasta,Eggs
Milk,Eggs,Bread,Juice,Butter,Cheese
Ketchup,Mayonnaise,Beans,Ranch,Cheese,Almonds
Chicken,Lettuce,Ranch,Mayonnaise,Bread,Soda
Orange,Almonds,Yogurt,Banana,Juice,Bread,Oil
Cereal,Cashews,Orange,Juice,Lettuce,Milk
Pasta,Cheese,Rice,Butter,Beans,Lettuce
Lettuce,Chicken,Pasta,Mayonnaise,Soda,Rice,Pasta
Oil,Vinegar,Lettuce,Beans,Ranch,Cheese,Bread
```

Transaction5.txt

Contains 20 unique transactions using the 30 items from FoodItems.txt

```
Milk,Cereal,Yogurt,Banana,Orange,Juice,Almonds,Lettuce
Cheese,Crackers,Hummus,Salsa,Almonds,Eggs,Bread
```

```
Bread,Butter,Cheese,Bacon,Milk,Lettuce,Rice,Pasta,Lettuce
Chicken,Lettuce,Ranch,Mayonnaise,Bread,Soda
Ketchup,Mayonnaise,Vinegar,Ranch,Cheese,Almonds
Salsa,Chips,Butter,Juice,Soda,Rice,Chicken,Milk
Oatmeal,Milk,Banana,Yogurt,Juice,Orange,Almonds,Cashews
Rice,Chicken,Pasta,Butter,Lettuce,Juice,Lettuce
Lettuce,Chicken,Vinegar,Mayonnaise,Almonds,Bread
Cereal,Cashews,Orange,Juice,Lettuce,Milk,Lettuce
Pasta,Cheese,Rice,Butter,Vinegar,Lettuce
Orange,Almonds,Yogurt,Banana,Juice,Bread,Milk
Bread,Butter,Cheese,Mayonnaise,Lettuce
Cheese,Bacon,Pasta,Rice,Butter,Crackers,Bread
Crackers,Hummus,Chips,Salsa,Soda,Cashews
Eggs,Butter,Chicken,Pasta,Lettuce,Bread
Milk,Eggs,Bread,Juice,Butter,Cheese,Rice,Chicken,Milk
Lettuce,Chicken,Pasta,Mayonnaise,Soda
Oil,Vinegar,Lettuce,Vinegar,Ranch,Cheese
Yogurt,Banana,Orange,Cereal,Milk,Cheese
```

Conclusion

Based on the association rules discovered by applying various minimum support and confidence values to the 5 transactional datasets with both methods, we can conclude that the Apriori algorithm is generally more efficient and scalable than the Brute Force Method due to its shorter CPU time.

Whilst the Apriori algorithm works best with large datasets that require extensive computation, the Brute Force Method has the potential to uncover more association rules because it considers all possible itemsets. Therefore, depending on the conditions and requirements of the dataset being analyzed, both methods are successful in mining association rules.

Thank you