NAME: **SUBHODEEP GHOSH**  
NJIT UCID: **SG2646**  
Email Address: sg2646@njit.edu  
10/01/2024  
Professor: Yasser Abduallah  
CS 634 **101** Data Mining

**Midterm Project Report**

*Apriori Algorithm Implementation in Association Rule Mining:*

**Abstract:**

This project explores the use of association rule mining in finding frequent patters or associations among sets of items in transactional databases. The algorithm that was employed to do this task is Brute Force algorithm. The results obtained were verified against existing python libraries for Apriori and FPTree algorithms. I explore how the three methods compare against each other in terms of execution speed and the accuracy of result.

**Introduction:**

In Data Mining, patterns are sets of items that frequently occur together in a dataset. They represent important and intrinsic properties of datasets. Thus pattern discovery is a vital task in data mining. Association rule mining in one such task which finds frequent correlations among sets of items in transactional database. For example, customers frequently buying milk and cookies together gets registered in a shop’s database and with association rule mining we find the rule that when milk is being bought, cookie is also being bought – thus we can use this information to suggest cookies to other customers whenever they buy milk. Ultimate goal is thus customer satisfaction which in turn leads to profit.

There are various algorithms to do the task of association rule mining – brute force (the one I implemented in this project), apriori, FPT (the two algorithms my model was compared against).

5 distinct transactional databases were used to implement this project – each containing 10 items and 20 transactions.

Before diving into the database descriptions and implementation details, lets look at some of the core concepts involved in this project.

**Core Concepts:**

Two most important concepts of association rule mining are support and confidence. Based on them all the algorithms have been designed.

**Support**: It shows the popularity of the itemset (may include one or more items. More items usually mean those 2 items are bought together, e.g., when milk is bought cookie is also bought) in relation to the transactions. It is represented by the number of transactions (frequency) that include the itemset divided by the total number of transactions in the database.

**Confidence:** It shows how likely the items in an itemset are purchased together. Say we have an itemset of {Milk, Cookies}. The confidence for Milk -> Cookie is given by the support of the itemset {Milk, Cookie} divided by the support of itemset {Milk}. Thus we notice, confidence helps in uncovering association rules that whenever Milk is bought, how likely are customers to buy Cookie with it.

In pattern analysis, we are interested in association rules that dominate the database. Support and confidence ensure the popularity and correlation of items being bought together. We only consider items that are equal to or above a minimum support or confidence defined by the user. All others are not popular or correlated enough to be considered a rule.

**Concept behind Brute Force, Apriori and FPTree:**

**The Brute Force algorithm** generates all possible itemsets for a particular k>=1 (integers) where k defines the size of the itemsets. Thus, when k = 1, our itemsets have only a single item in them like {Milk}, {Cookie}, etc. If there are 10 items, there will be 10 itemsets for k=1.

We calculate the support for all these itemsets and remove the ones that have support less than the user defined min support. We then generate all possible itemsets for k = 2, which would 10C2 total itemsets or 45 itemsets. We then calculate the support again for these 2-itemsets and remove the ones with less than min support. We store the itemsets and their respective supports for future calculations.

We repeat this process until we reach a certain k when none of the generated k-itemsets has support >= min support. We then stop the process of finding the support for k-itemsets and do not generate (k+1)-itemset.

We then use the support for the itemsets we have to compute the confidence for a certain rule. Say we have an itemset {A,B,C} with support = 60%. If we want to find the confidence of the rule A -> B,C then we just divide the support of {A,B,C} by the support of {A}. Similarly for A,B -> C we calculate the confidence as support for {A,B,C} divided by the support of {A,B}.

Based on the calculated confidences, we remove the rules that are below the user defined min confidence and print the rest.

**The Apriori Algorithm** proceeds by identifying the frequence individual items in the database. Then extends the identified frequent items to larger itemsets. The resulting frequent itemsets are used to build and finalize the association rules. The Apriori principle states that any superset of a non-frequent itemset must also be non-frequent. Likewise, any subset of frequent itemset must also be frequent.

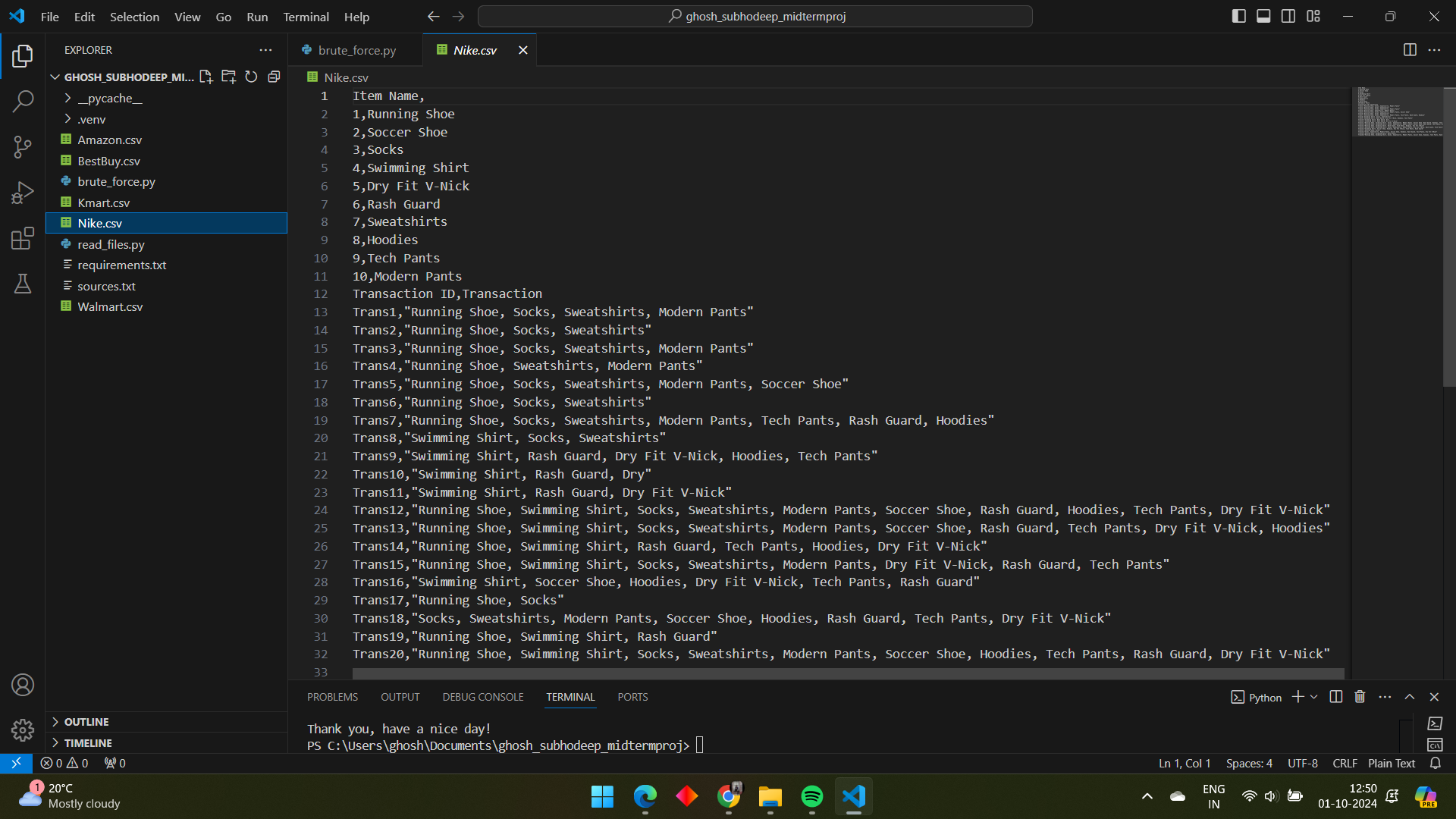
**The FPTree Algorithm** creates a tree-like structure made of the original itemsets. The purpose of the tree is to mine the most frequent patterns. The root node represents null while the lower nodes represent the itemsets.

**Datasets used:**

The following 5 datasets were used as transactional databases – Amazon.csv, BestBuy.csv, Kmart.csv, Nike.csv and Walmart.csv.

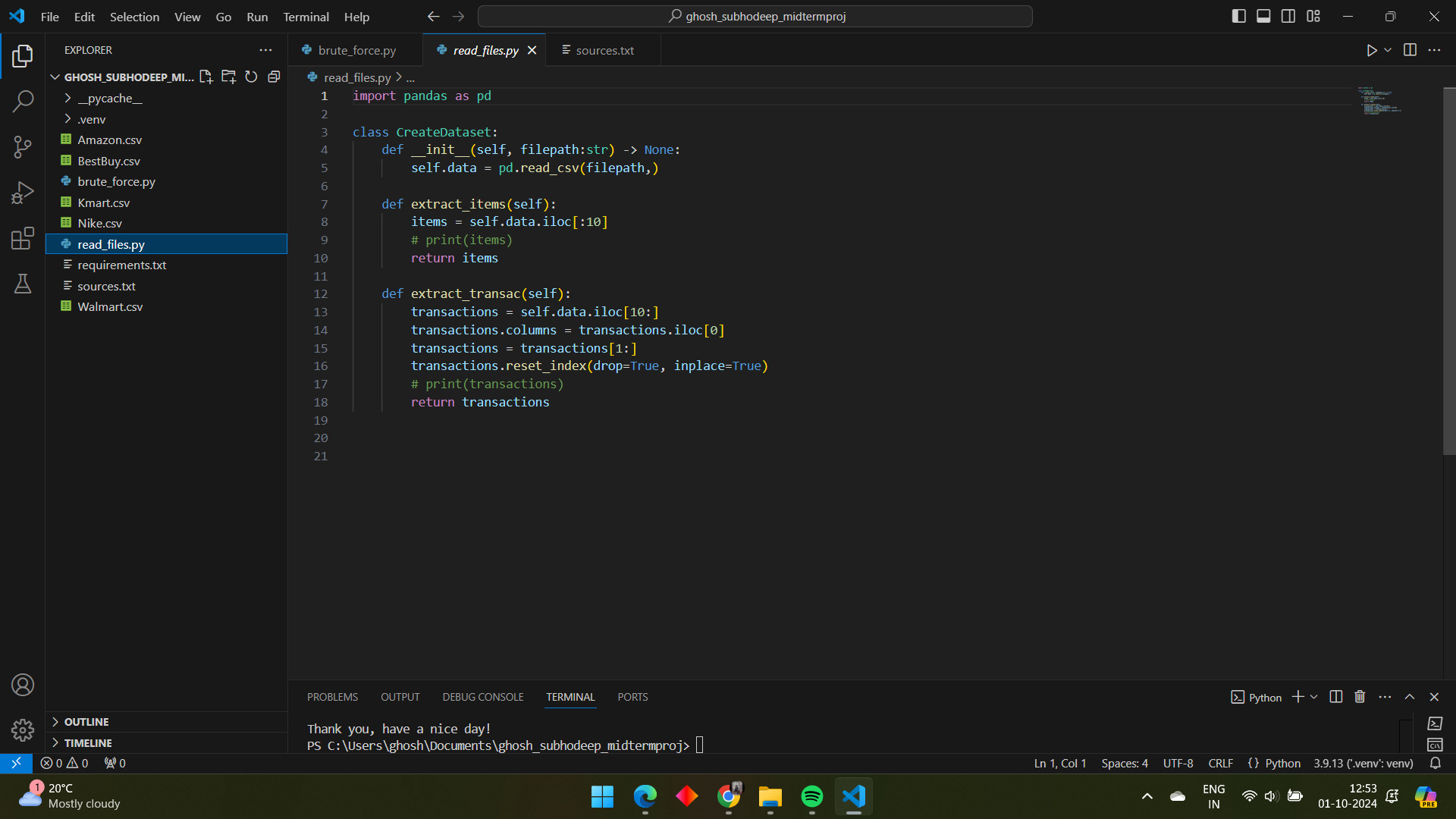
Walmart.csv lacked the required number of items and transactions, so I used chatpgt-4 to extend the number of items to 10 and transactions to 20.

Following is a screenshot of how the .csv files look. The one in the screenshot is Nike.csv.

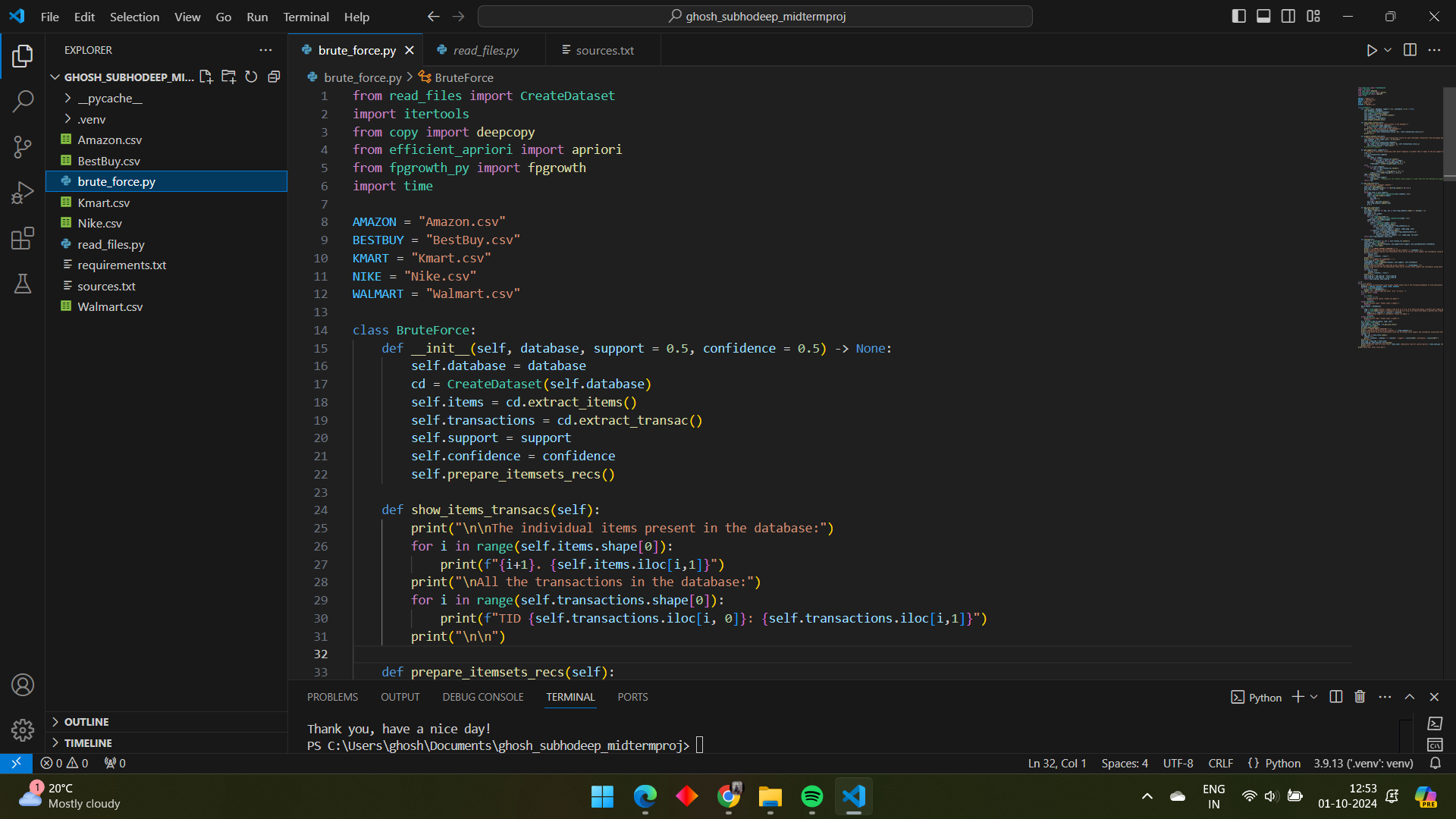


**Implementation Details:**

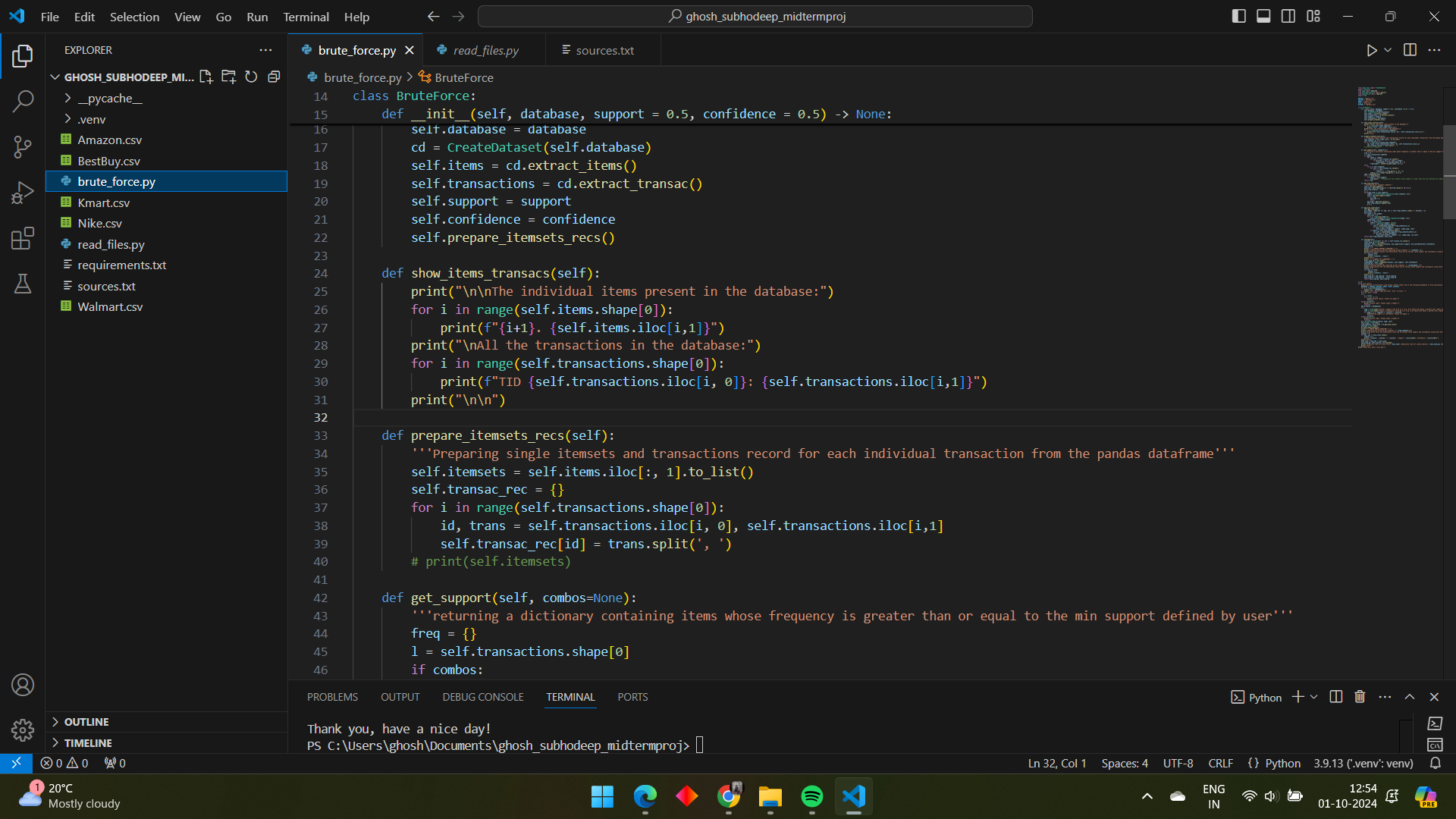
1. Pre-processing the data: The CreateDataset class in the read\_files.py is responsible for reading the .csv files using and then converting them into pandas dataframe. Then with the extract\_items() function it extracts all the items in the dataset into a pandas series, and with extract\_transac() function it extracts all the transactions with their ids into another dataframe.



1. The brute\_force.py is the main file and the one that should be run to generate results. First I make some important imports for my code and define some constants that contain the path to the .csv files.



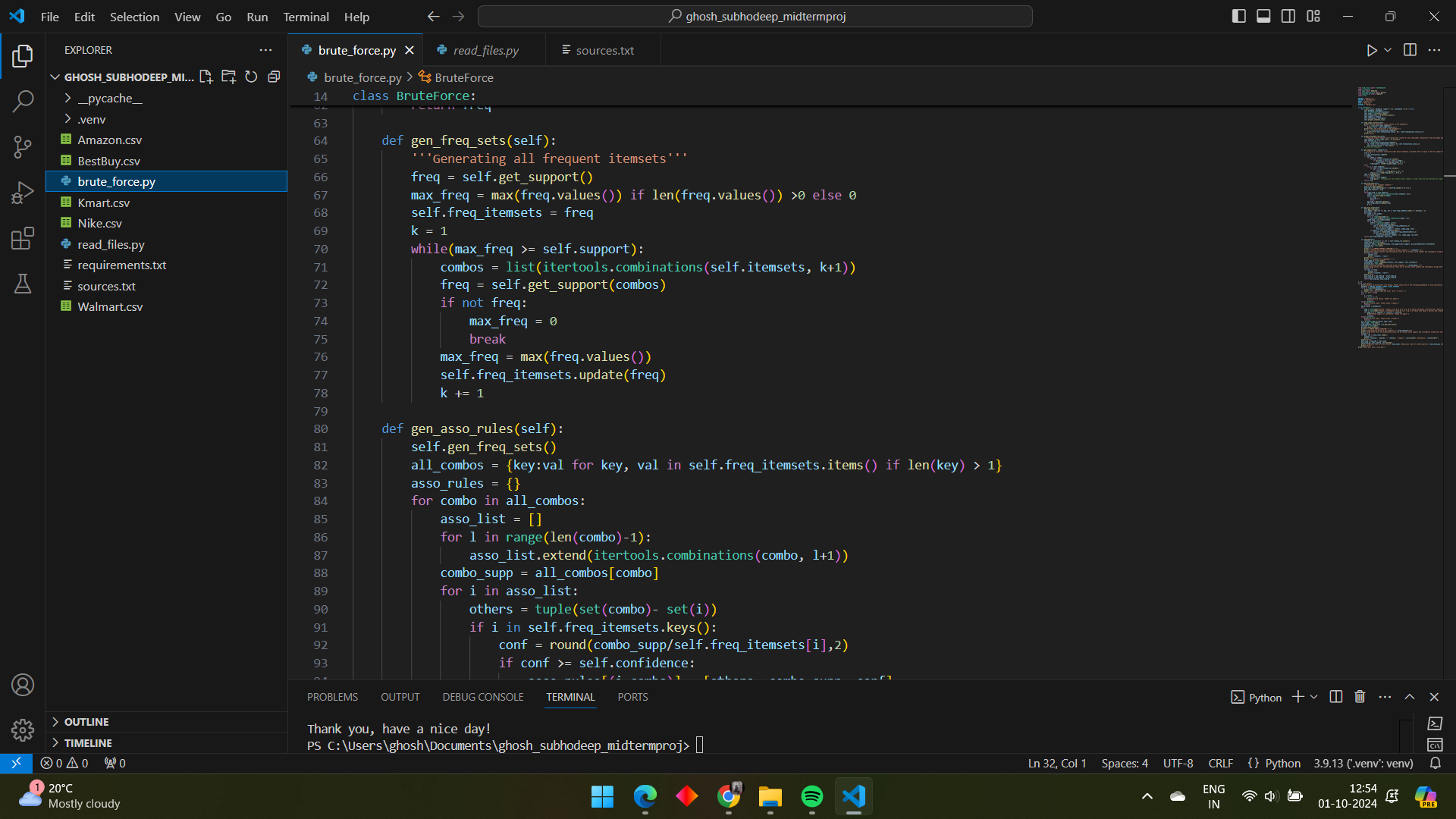
1. I then define the BruteForce class, that will contain all the important functions to run our brute force algorithm.



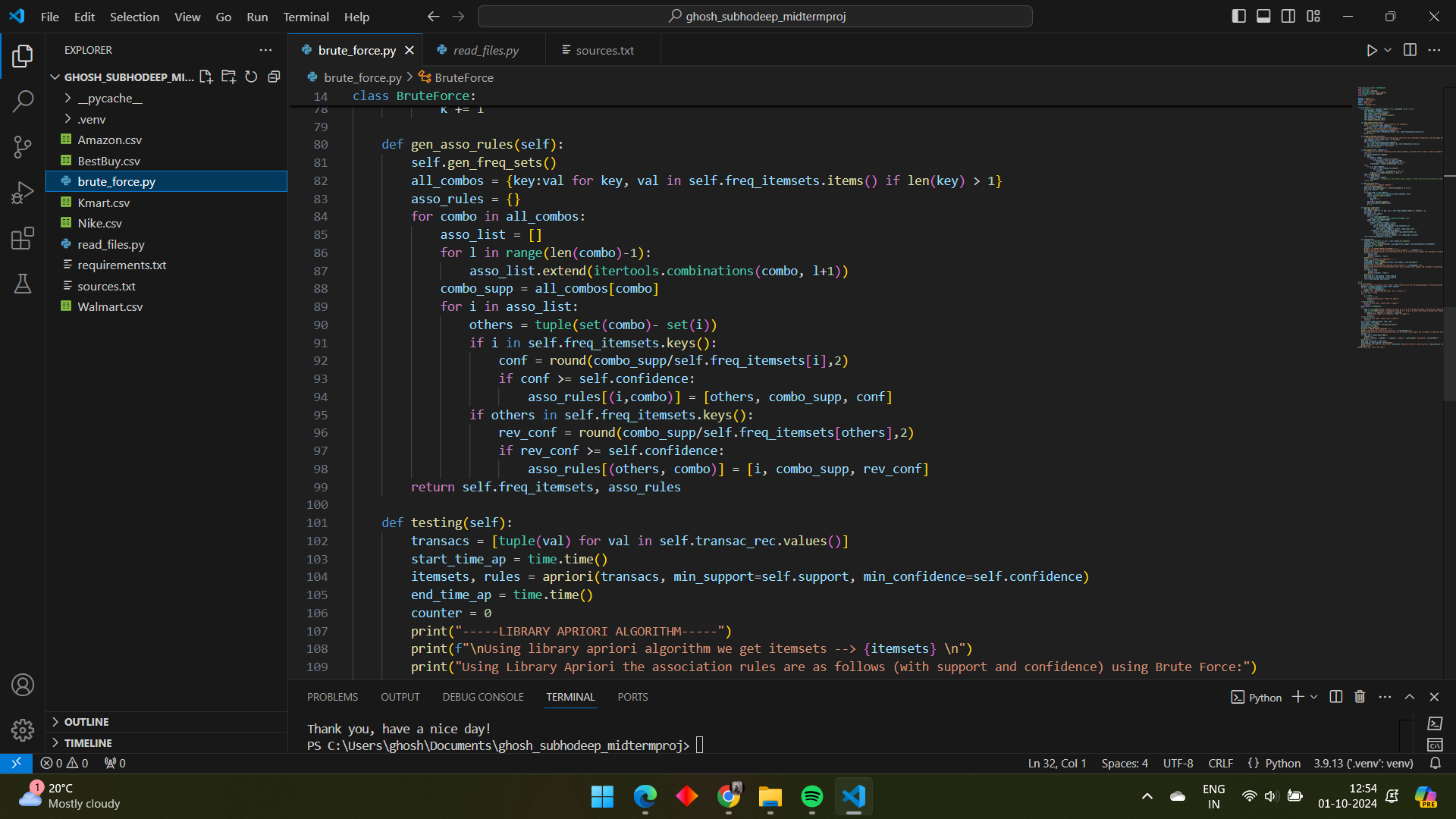
1. The get\_support() function responsible for counting the support of each itemset and removing those with support < min support



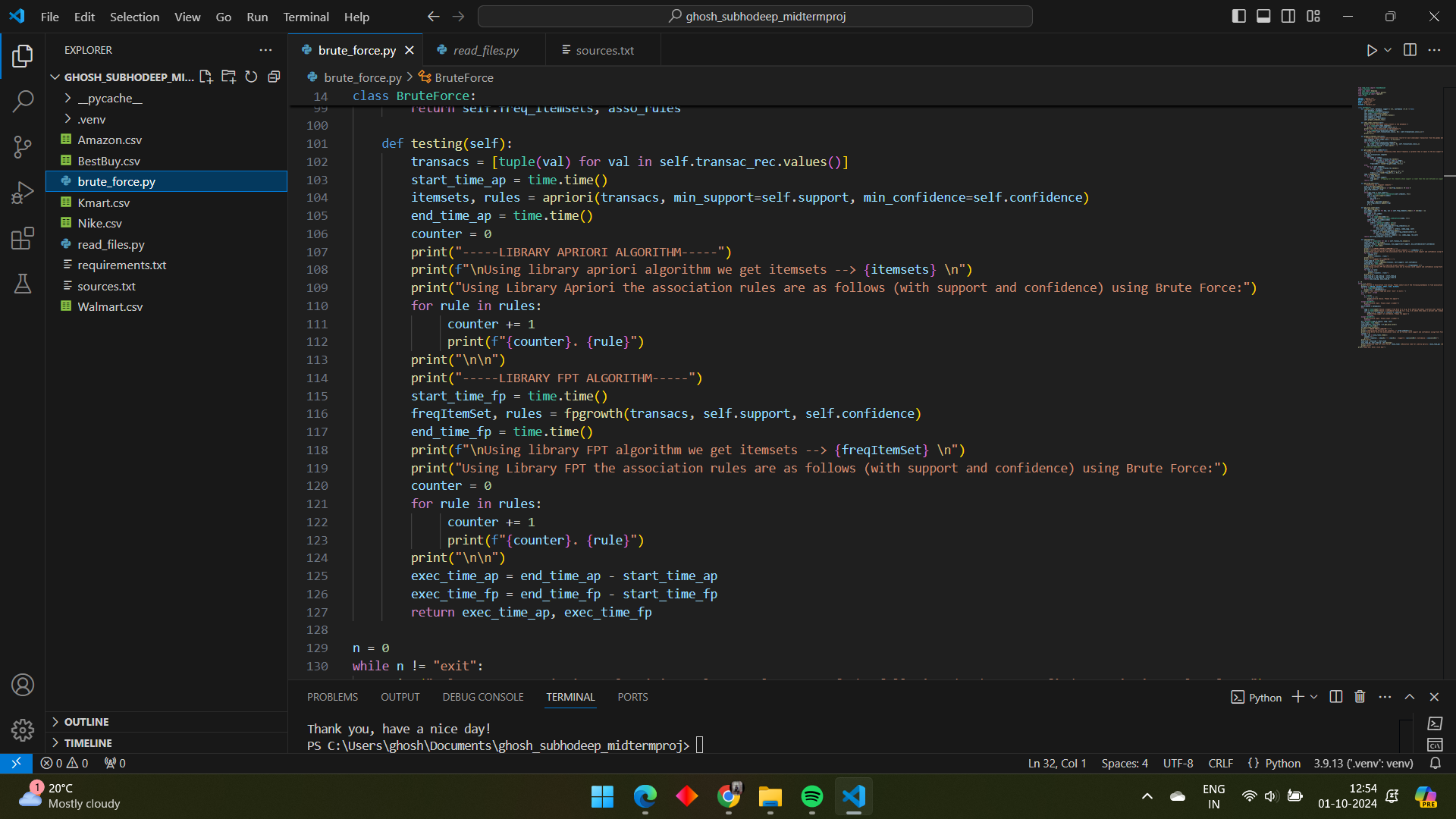
1. The gen\_freq\_sets() function keeps generating all the frequent itemsets till k for which support exists. It stops at k+1 when it finds there is no support of any itemset for k.



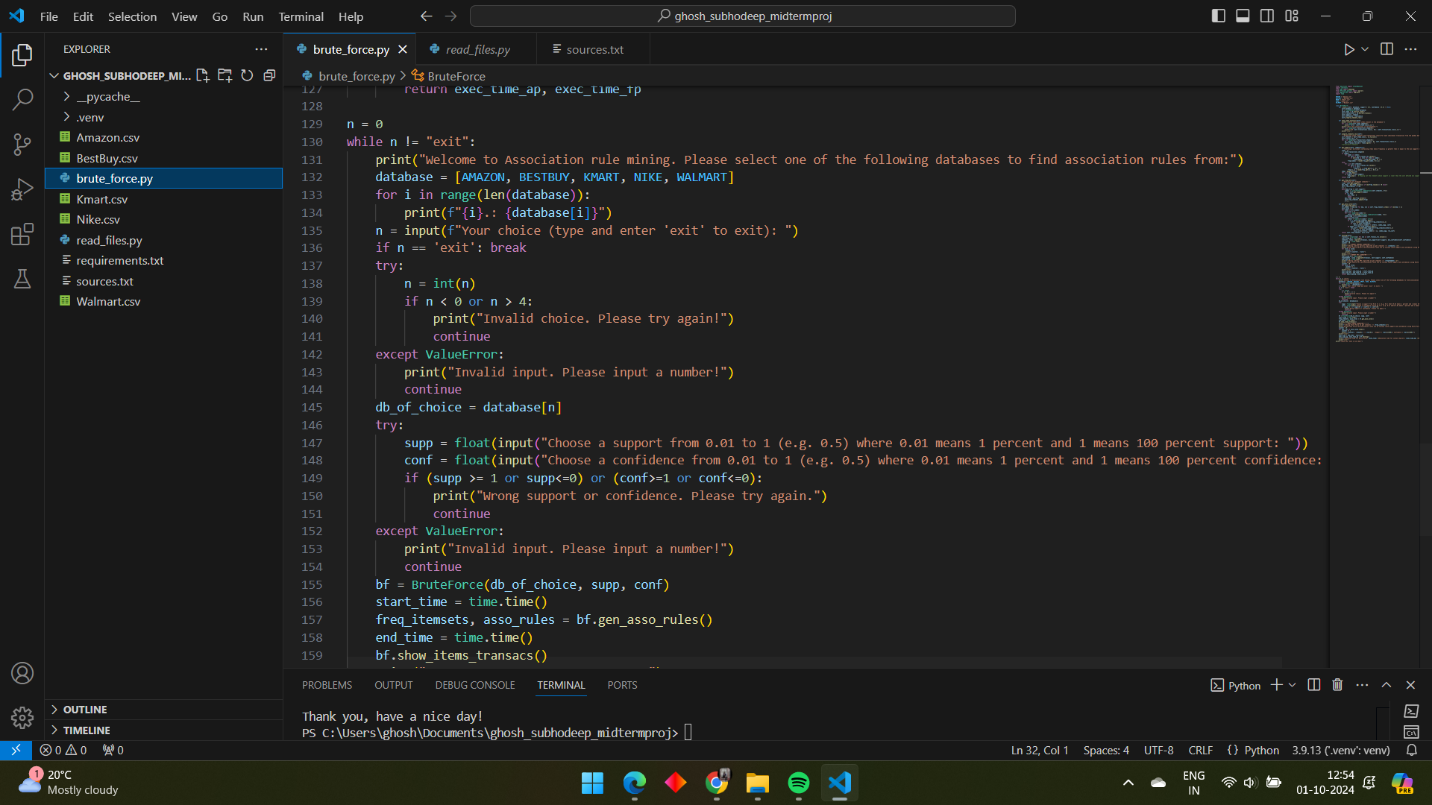
1. The get\_asso\_rules() function is responsible for generating and checking all the possible association rules for a given k-itemset, k>=2, against the min confidence. It returns a dictionary of those association rules that suffice the min confidence as keys and their respective confidence as values.

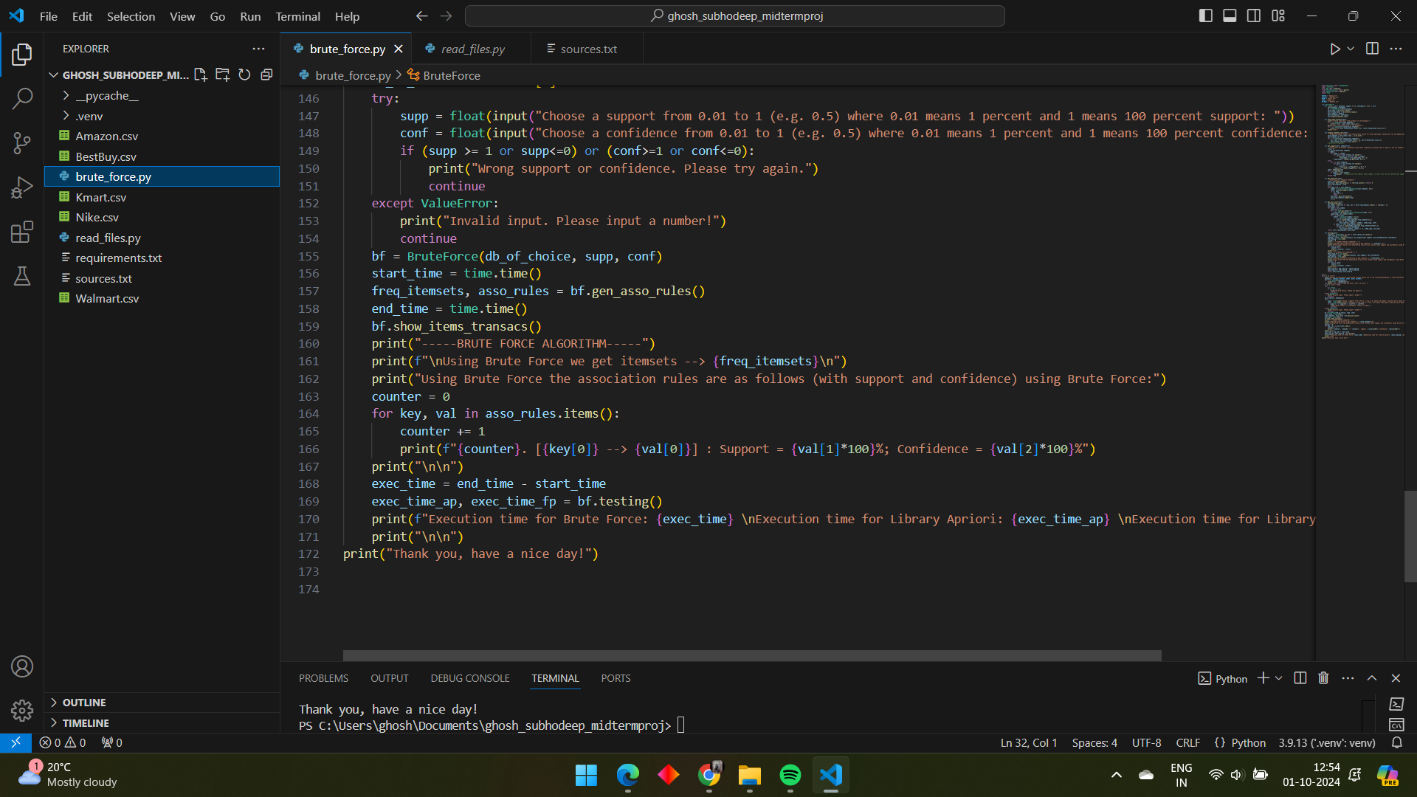


1. The testing() function is used to check the results generated by the python library Apriori algorithm and FPTree algorithm to verify my results using Brute Force. The execution time of these 2 algorithms are also calculated to compare against the brute force.



1. This part of the code represents how the object of the Brute Force class is created and how the functions are called, how error handling is done in case of invalid inputs, and how the simple user interface is designed.



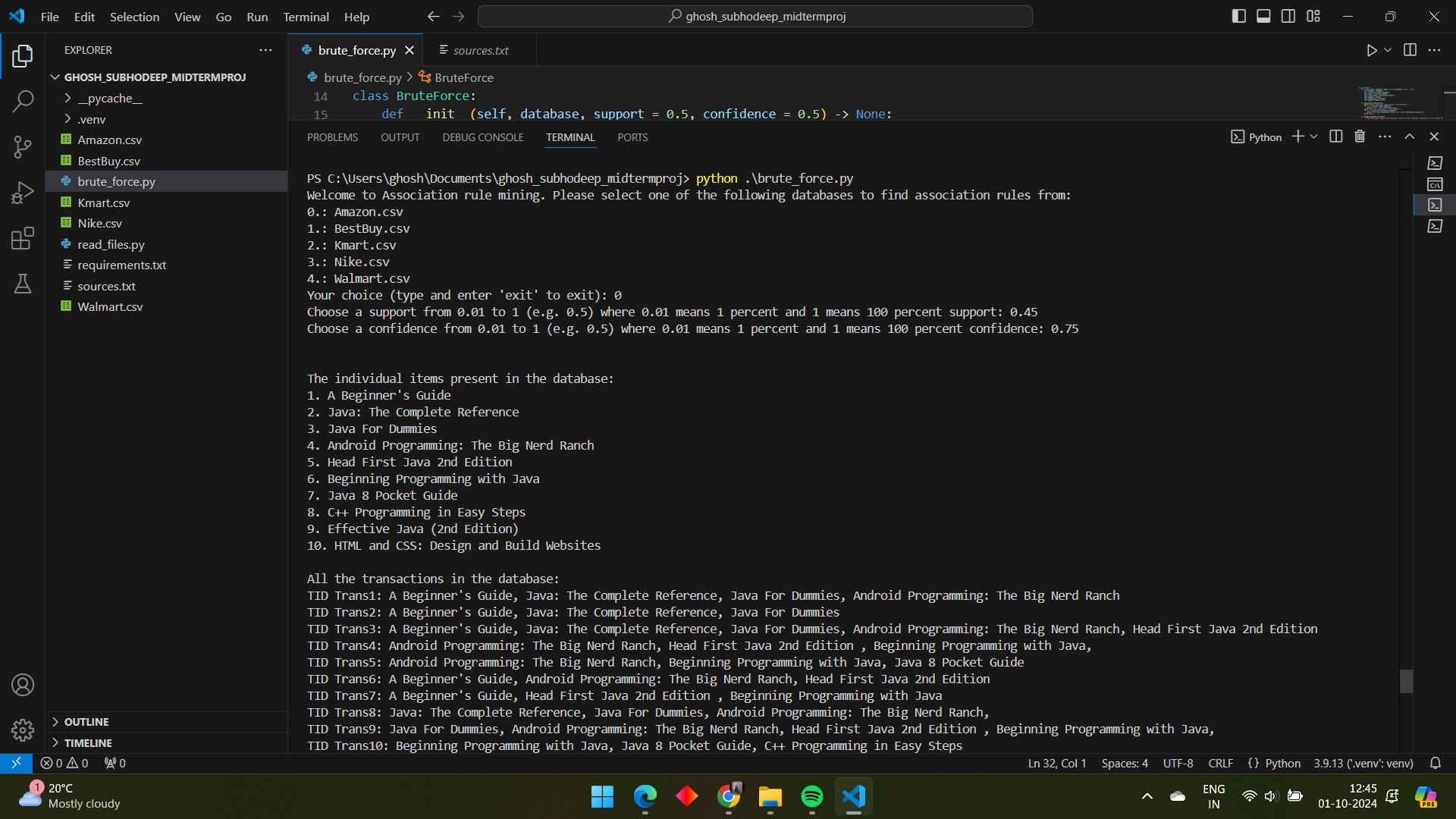


**How To Run The Code:**

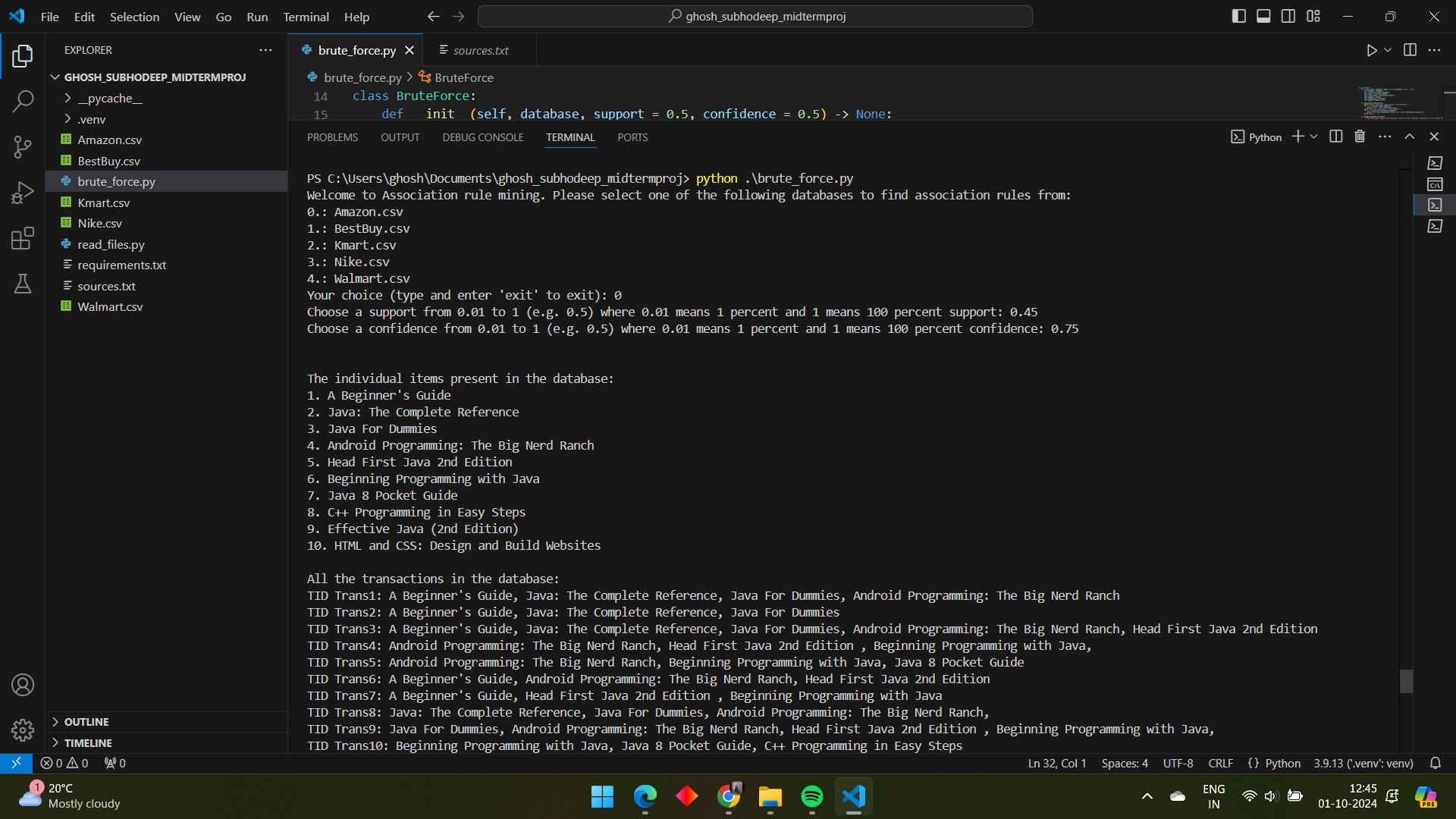
1. Make sure you have python installed in the system.
2. Download the zip file and extract the contents in a single folder. Make sure none of the files get separated from the parent folder ghosh\_subhodeep\_midtermproj, otherwise the code will not work.
3. Open the project folder from terminal
4. Install the prerequisites using the terminal:

pip install -r requirements.txt

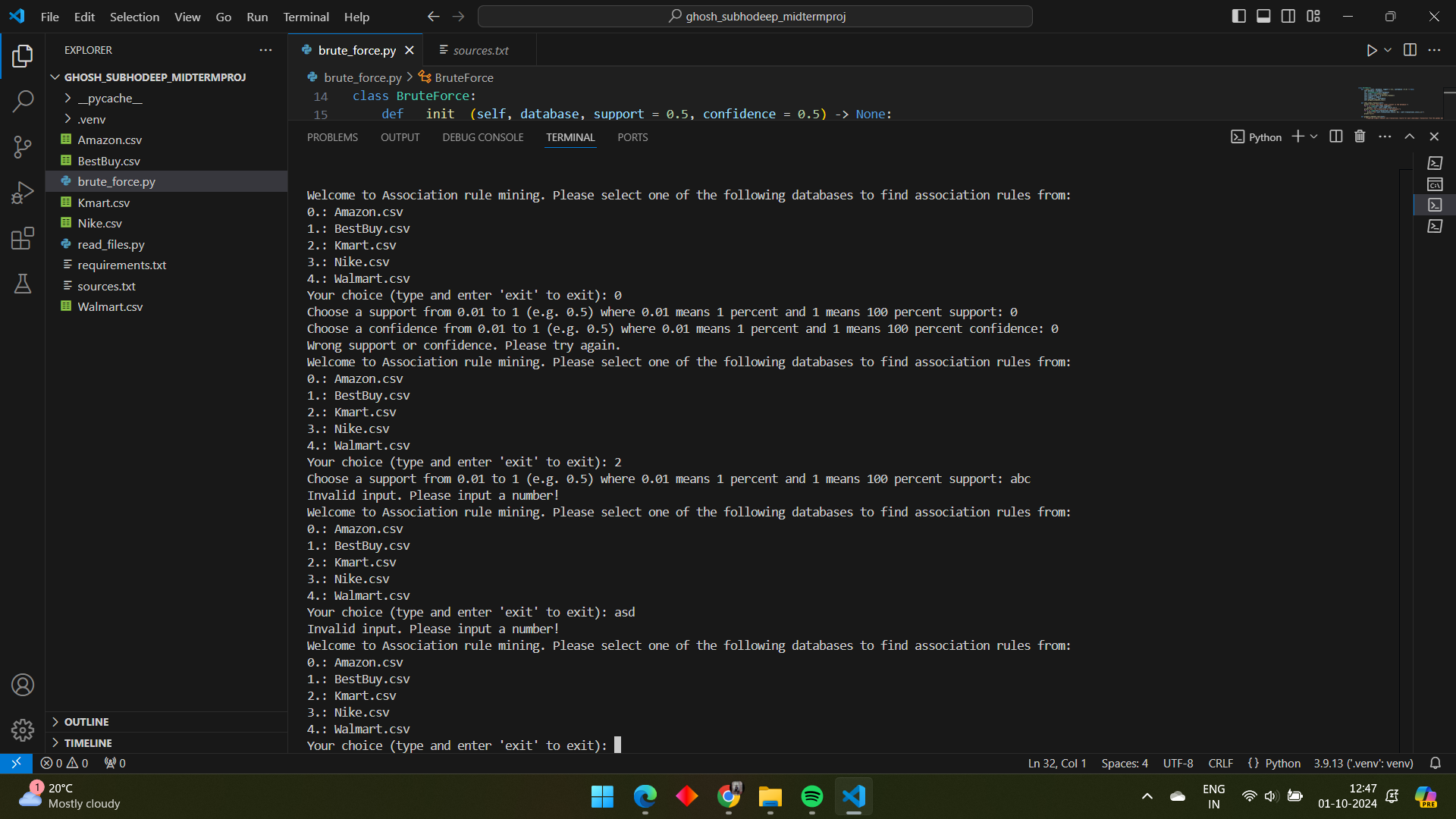
1. Once they are installed, run the brute\_force.py file. You should get something like this once it runs:



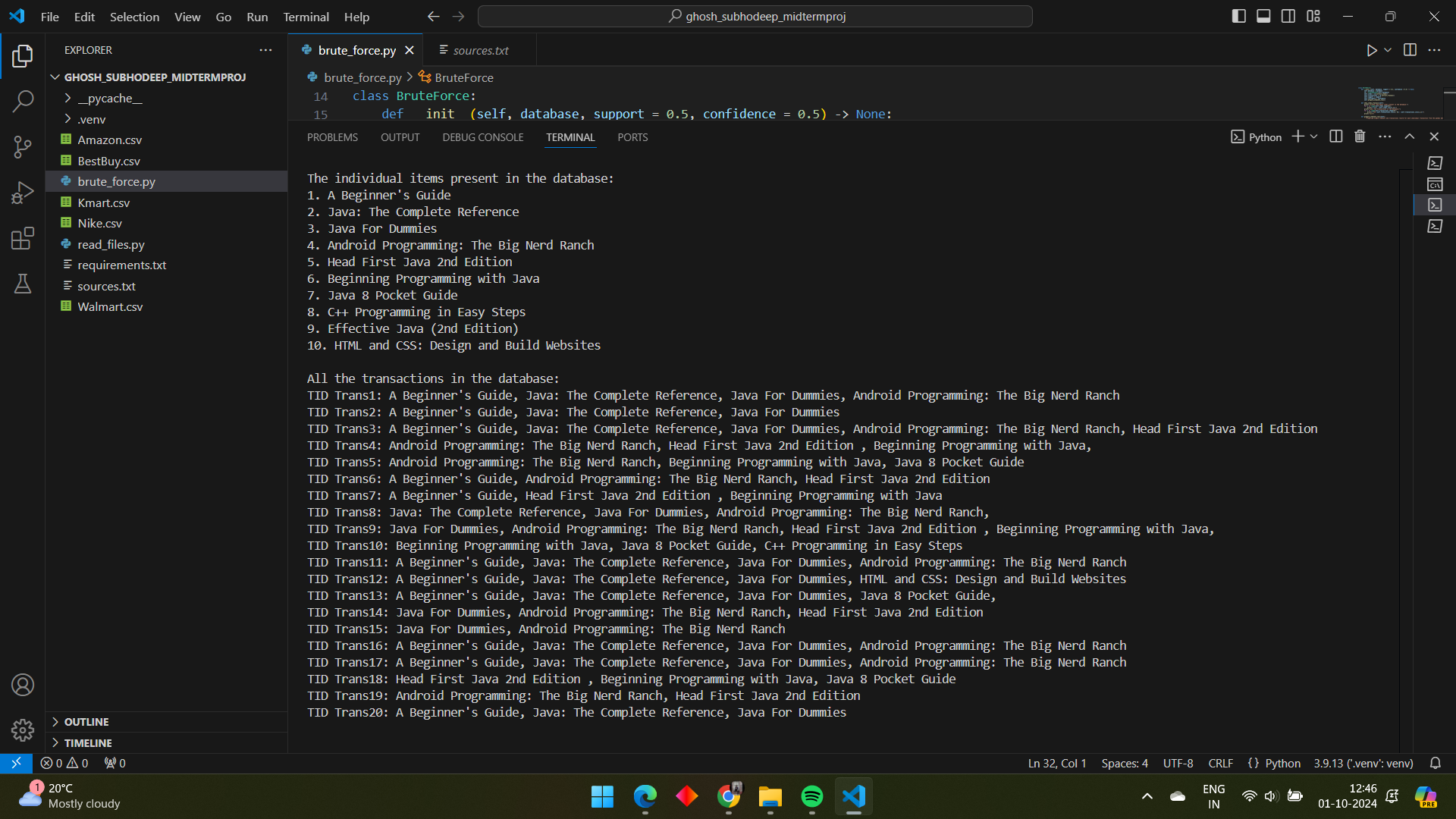
1. Enter your choice (0-4). I entered 0 to demonstrate. Then you will be prompted for support and confidence in the range (0,1] where if you input 0.45, it is interpreted as 0.45\*100% = 45%. Do not input 0 as your support or confidence otherwise it will throw an error and ask you to input again

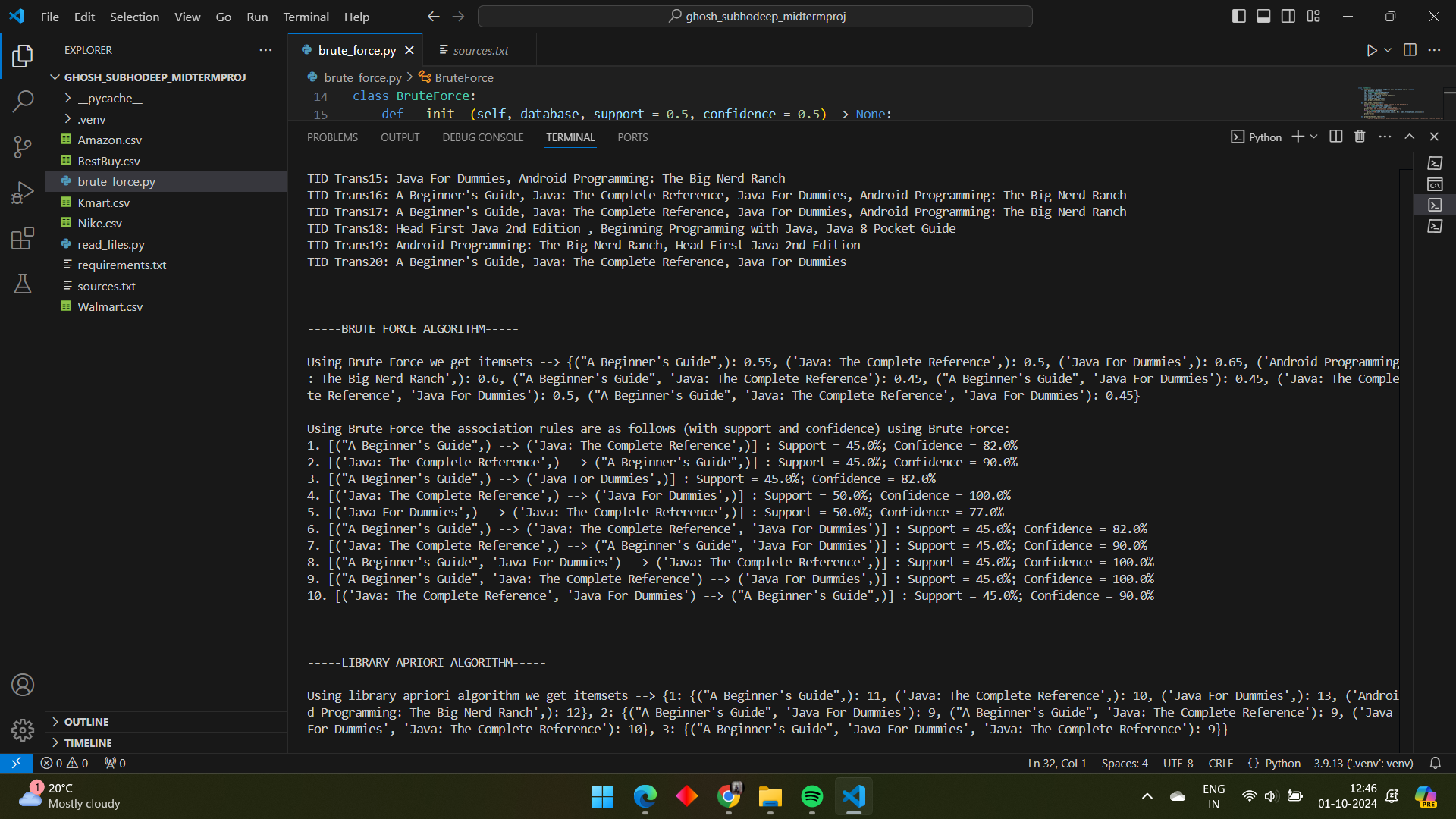


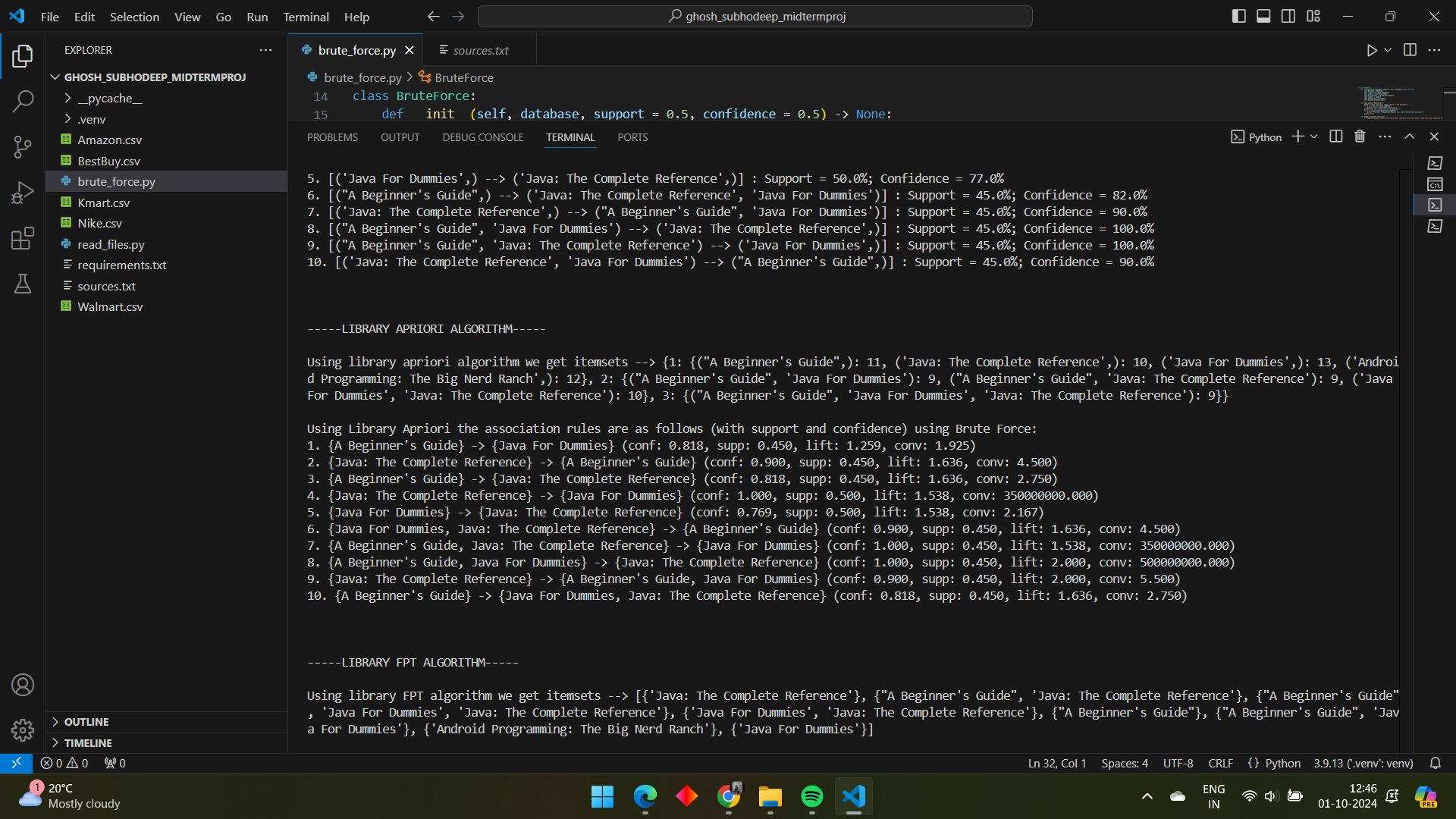
How wrong inputs are handled, an example:

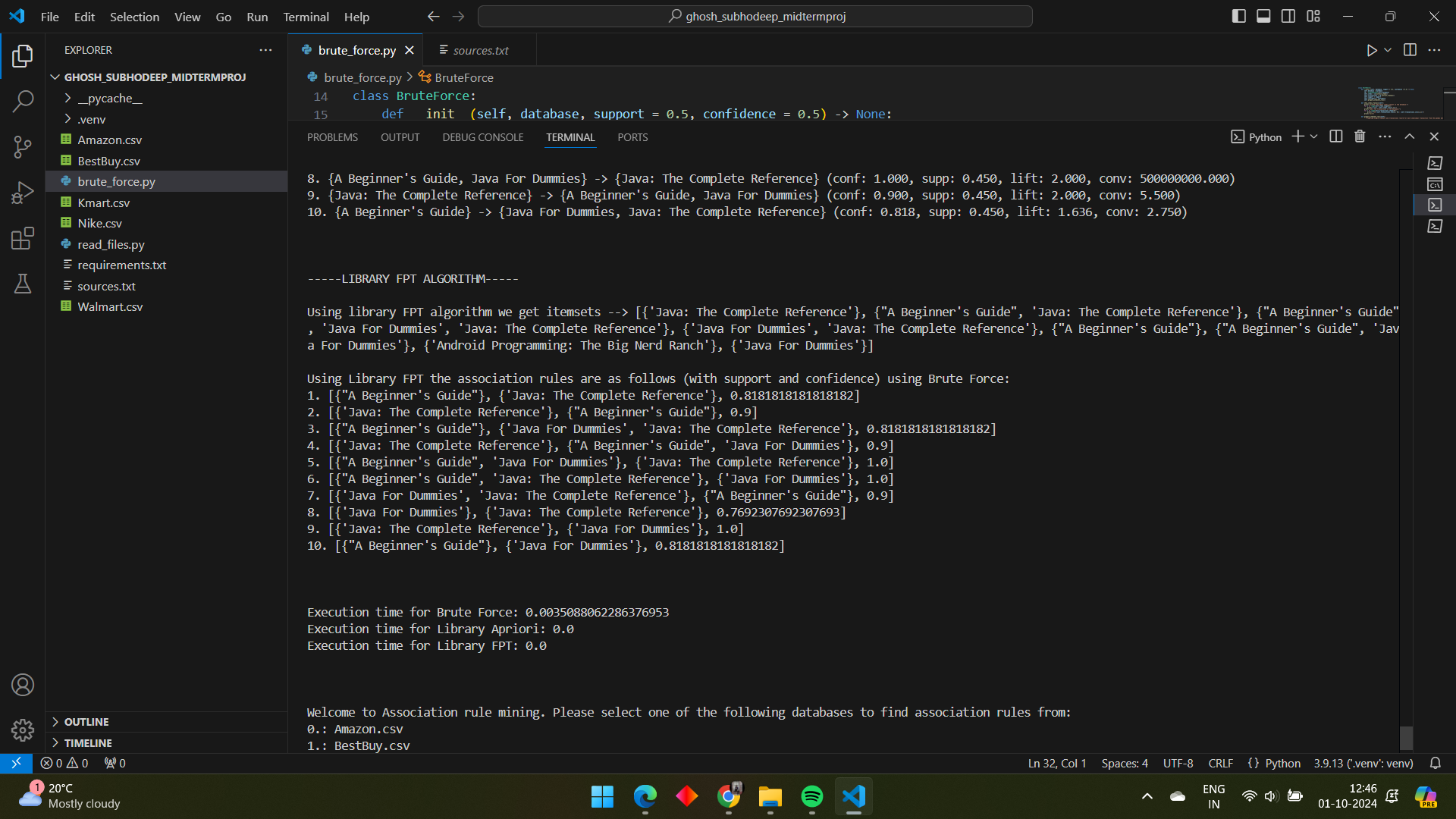


1. Once you have inputted the values for min support and confidence, you will be showed the items and transactions present in the database of your choice, the frequent itemsets and association rules discovered by the brute force algorithm, the library Apriori algorithm and the library FPTree Algorithm along with their respective execution times in the end for comparison of performance.

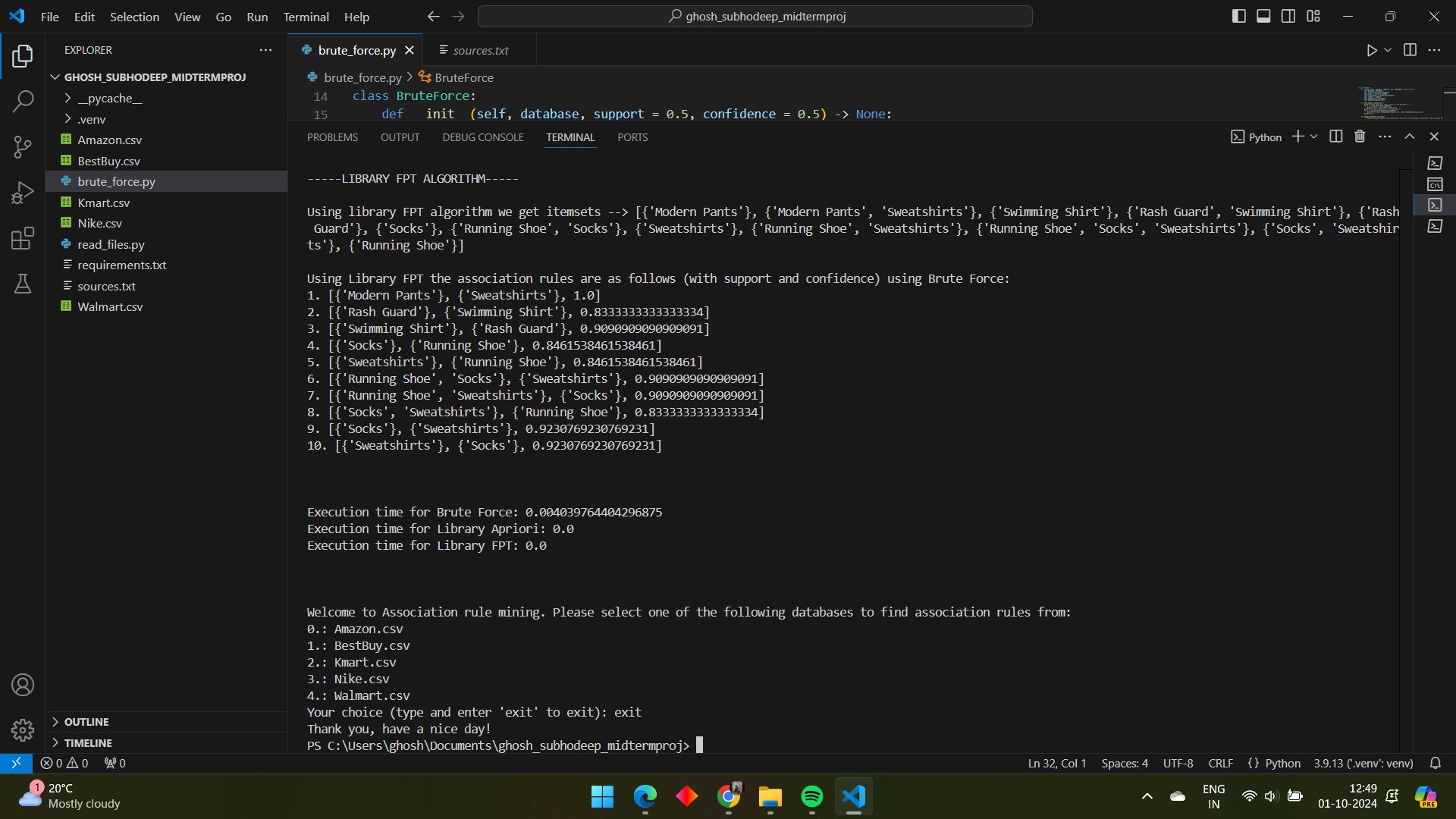








1. After the results are generated you will once again be given the choice of choosing a database for the next iteration. This will go on until you type “exit” and enter which will finally terminate the code.



**Conclusion:**

With this project I learned the concepts of association rule mining and the various algorithms that are employed to do this task. I learned the concepts of support and confidence. From the results I also inferred that although the results of the brute force algorithm is 100% accurate and consistent with library Apriori and FPTree algorithm, its still relatively much slower than them. Even though it takes only 1000th of a second to run compared to instant results by Apriori and FPTree, we have to remember the databases in this project are all very small compared to real world databases which have hundreds of thousands of items and millions of transactions. Thus brute force approach is not optimal to use on real world databases.

**Sources:**

All the sources are listed in sources.txt.



All the .csv files are already included in the project folder along with requirements.txt listing the requirements to be installed.