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.

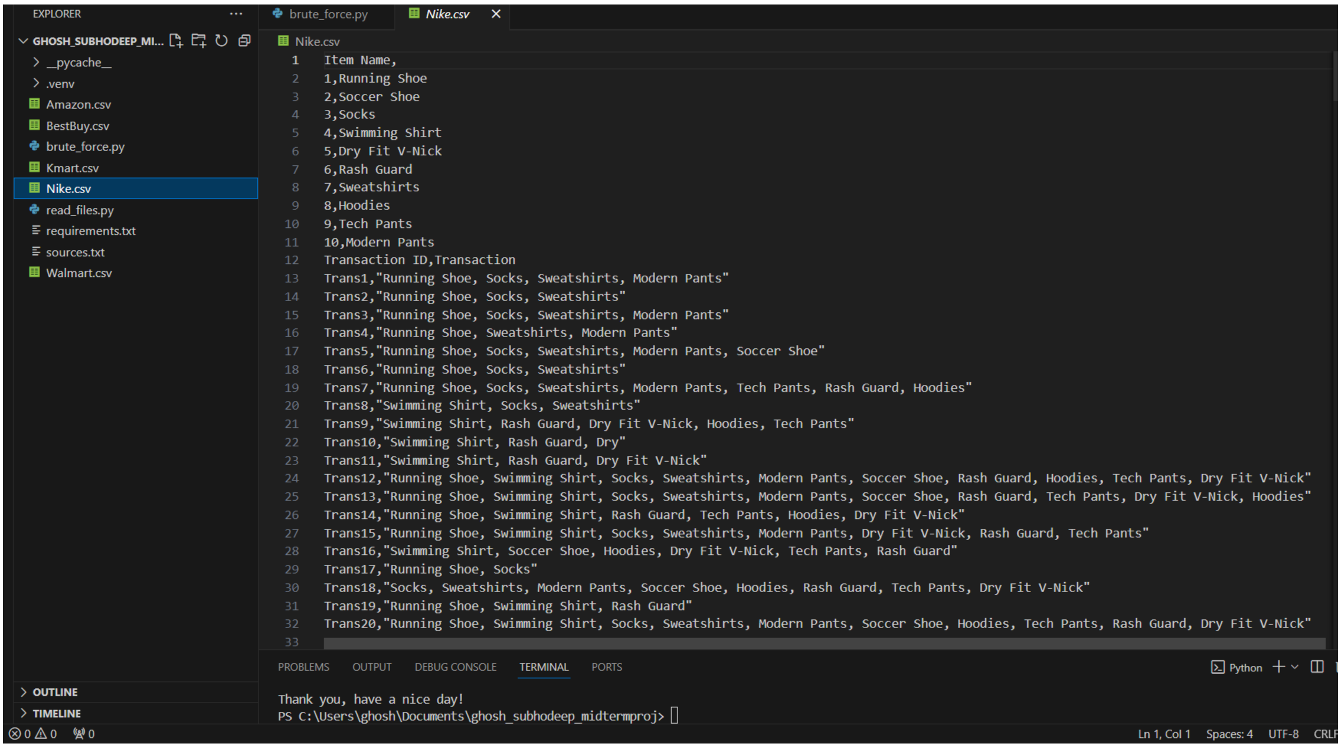
The FPTree Algorithm creates a tree-like structure made of the original itemsets. The purpose of the tree is to mine the most frequent patterns. The root node represents null while the lower nodes represent the itemsets.

Datasets used:

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

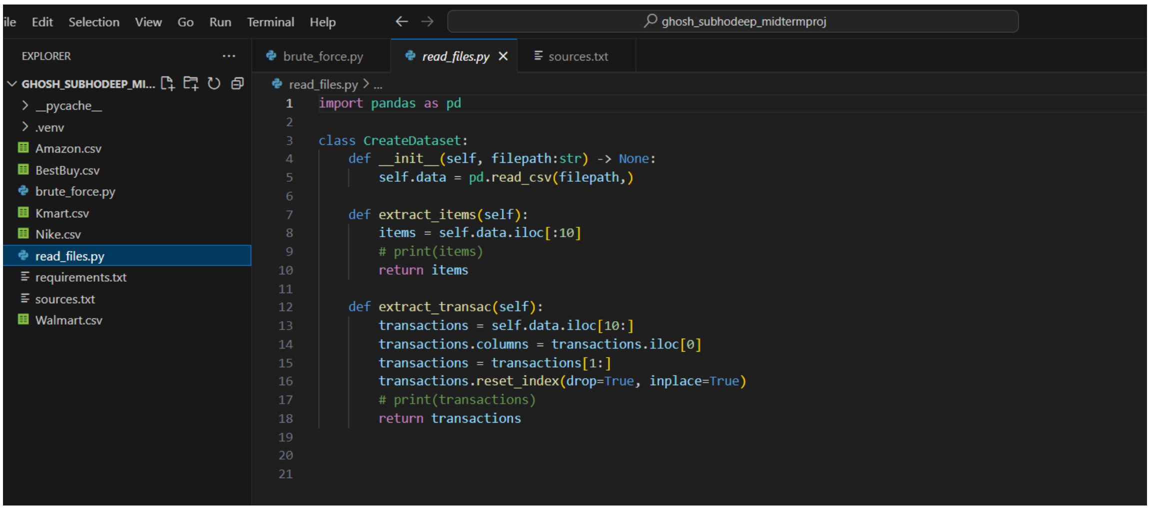
Walmart.csv lacked the required number of items and transactions, so I used chatpgt-4 to extend the number of items to 10 and transactions to 20.

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

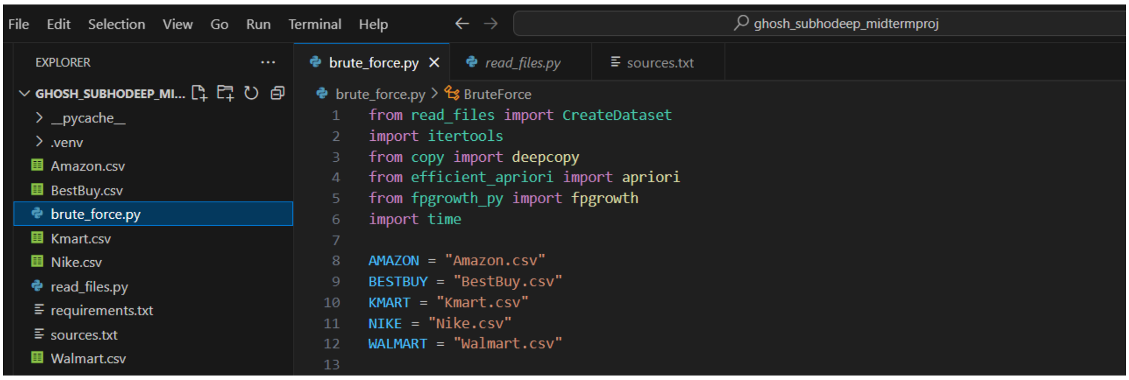


Implementation Details:

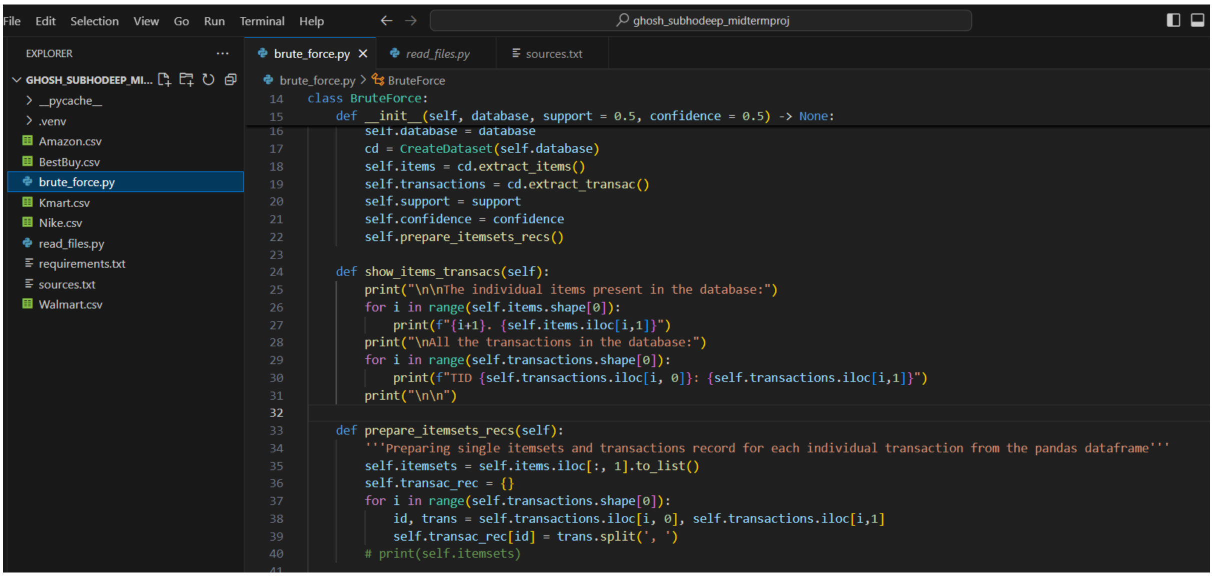
1. Pre-processing the data: The CreateDataset class in the read_files.py is responsible for reading the .csv files using and then converting them into pandas dataframe. Then with the extract_items() function it extracts all the items in the dataset into a pandas series, and with extract_transac() function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



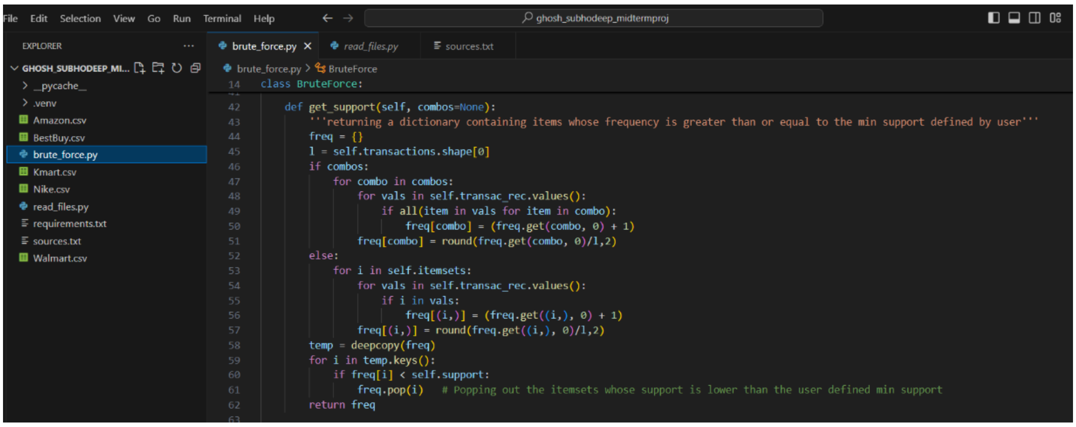
2. The brute_force.py is the main file and the one that should be run to generate results. First I make some important imports for my code and define some constants that contain the path to the .csv files.



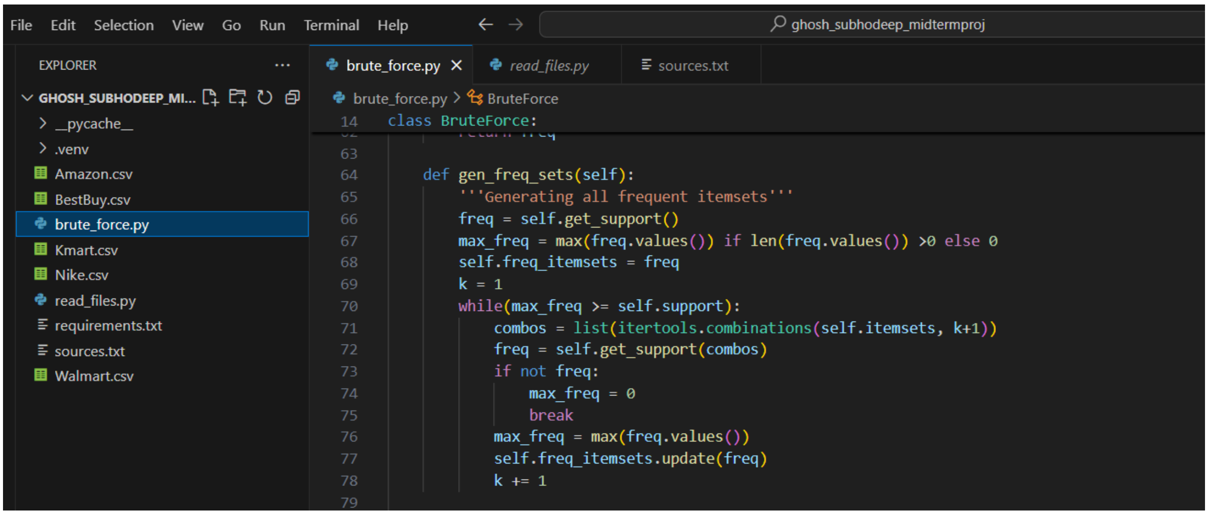
3. I then define the BruteForce class, that will contain all the important functions to run our brute force algorithm.



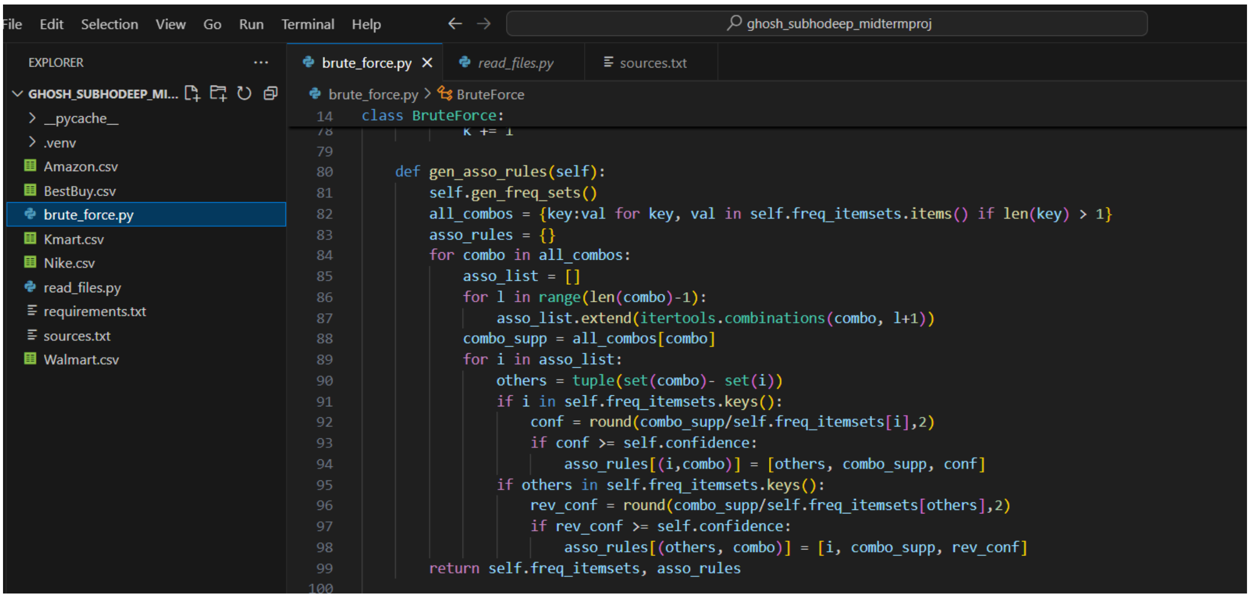
4. The get_support() function responsible for counting the support of each itemset and removing those with support < min support



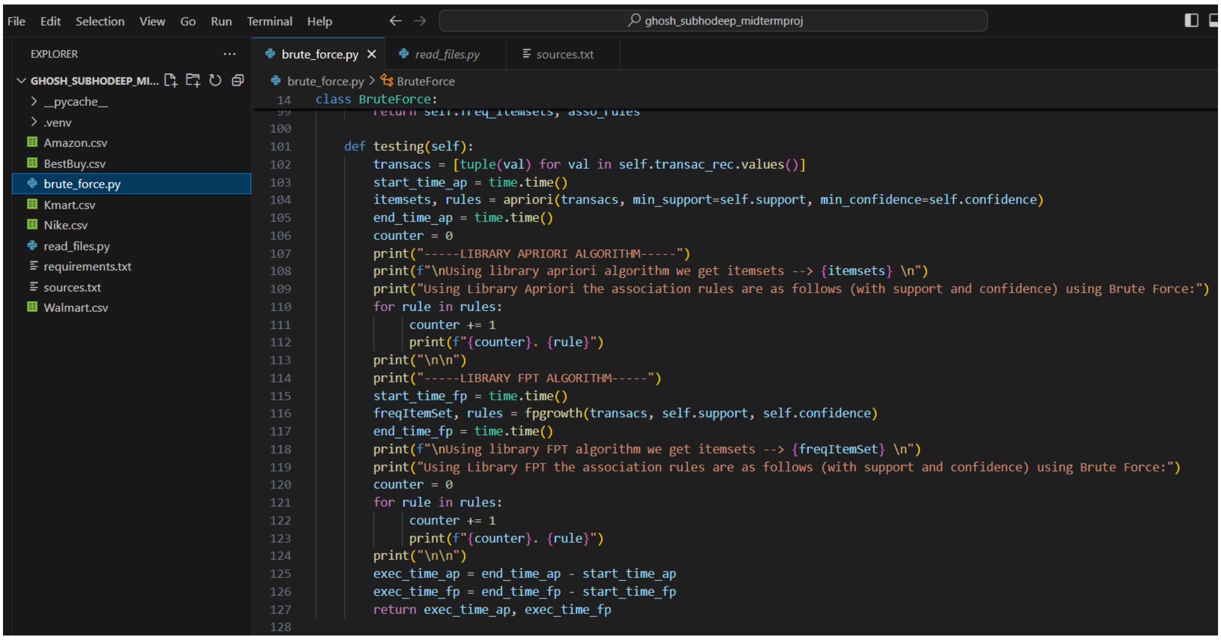
5. The gen_freq_sets() function keeps generating all the frequent itemsets till k for which support exists. It stops at k+1 when it finds there is no support of any itemset for k.



6. 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.



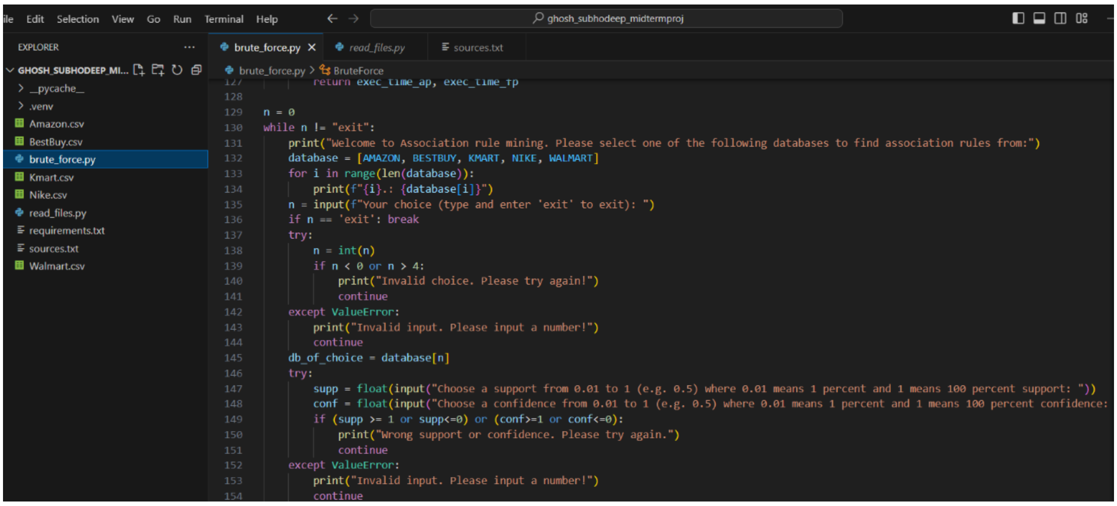
7. The testing() function is used to check the results generated by the python library Apriori algorithm and FPTree algorithm to verify my results using Brute Force. The execution time of these 2 algorithms are also calculated to compare against the brute force.



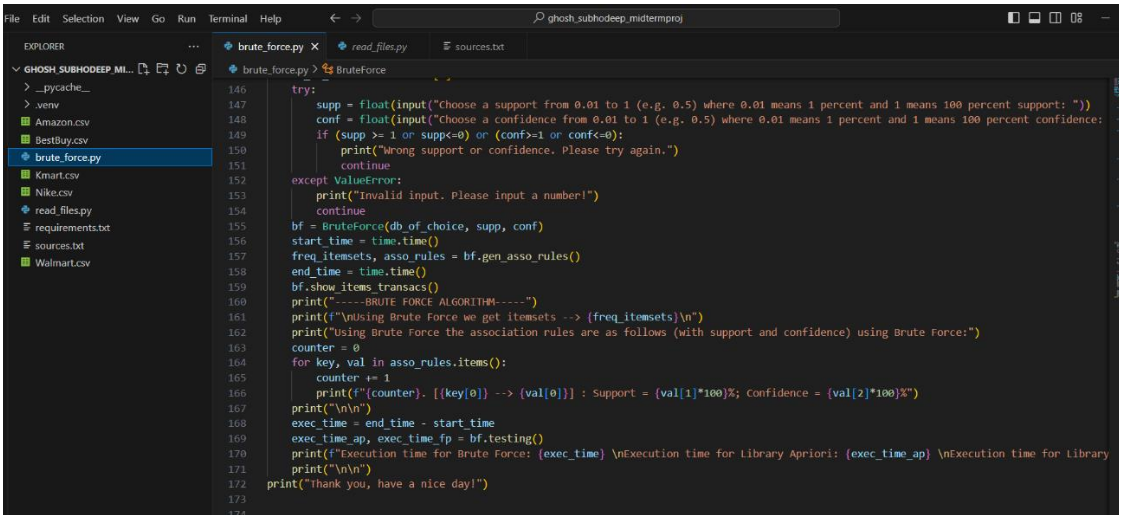
```python
class BruteForce:

    def testing(self):
        transacs = [tuple(val) for val in self.transac_rec.values()]
        start_time_ap = time.time()
        itemsets, rules = apriori(transacs, min_support=self.support, min_confidence=self.confidence)
        end_time_ap = time.time()
        counter = 0
        print("-----LIBRARY APRIORI ALGORITHM-----")
        print(f"\nUsing library apriori algorithm we get itemsets --> {itemsets} \n")
        print("Using Library Apriori the association rules are as follows (with support and confidence) using Brute Force:")
        for rule in rules:
            counter += 1
            print(f"{counter}. {rule}")
        print("\n\n")
        print("-----LIBRARY FPT ALGORITHM-----")
        start_time_fp = time.time()
        freqItemSet, rules = fpgrowth(transacs, self.support, self.confidence)
        end_time_fp = time.time()
        print(f"\nUsing library FPT algorithm we get itemsets --> {freqItemSet} \n")
        print("Using Library FPT the association rules are as follows (with support and confidence) using Brute Force:")
        counter = 0
        for rule in rules:
            counter += 1
            print(f"{counter}. {rule}")
        print("\n\n")
        exec_time_ap = end_time_ap - start_time_ap
        exec_time_fp = end_time_fp - start_time_fp
        return exec_time_ap, exec_time_fp
```

8. 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.



```python
n = 0
while n != "exit":
    print("Welcome to Association rule mining. Please select one of the following databases to find association rules from:")
    database = [AMAZON, BESTBUY, KMART, NIKE, WALMART]
    for i in range(len(database)):
        print(f"{i}.: {database[i]}")
    n = input(f"Your choice (type and enter 'exit' to exit): ")
    if n == 'exit': break
    try:
        n = int(n)
        if n < 0 or n > 4:
            print("Invalid choice. Please try again!")
            continue
    except ValueError:
        print("Invalid input. Please input a number!")
        continue
    db_of_choice = database[n]
    try:
        supp = float(input("Choose a support from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent support: "))
        conf = float(input("Choose a confidence from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent confidence:
        if (supp >= 1 or supp<=0) or (conf>=1 or conf<=0):
            print("Wrong support or confidence. Please try again.")
            continue
    except ValueError:
        print("Invalid input. Please input a number!")
        continue
```



```python
    try:
        supp = float(input("Choose a support from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent support: "))
        conf = float(input("Choose a confidence from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent confidence:
        if (supp >= 1 or supp<=0) or (conf>=1 or conf<=0):
            print("Wrong support or confidence. Please try again.")
            continue
    except ValueError:
        print("Invalid input. Please input a number!")
        continue
    bf = BruteForce(db_of_choice, supp, conf)
    start_time = time.time()
    freq_itemsets, asso_rules = bf.gen_asso_rules()
    end_time = time.time()
    bf.show_items_transacs()
    print("-----BRUTE FORCE ALGORITHM-----")
    print(f"\nUsing Brute Force we get itemsets --> {freq_itemsets}\n")
    print("Using Brute Force the association rules are as follows (with support and confidence) using Brute Force:")
    counter = 0
    for key, val in asso_rules.items():
        counter += 1
        print(f"{counter}. [{key[0]} --> {val[0]}] : Support = {val[1]*100}%; Confidence = {val[2]*100}%")
    print("\n\n")
    exec_time = end_time - start_time
    exec_time_ap, exec_time_fp = bf.testing()
    print(f"Execution time for Brute Force: {exec_time} \nExecution time for Library Apriori: {exec_time_ap} \nExecution time for Library
    print("\n\n")
print("Thank you, have a nice day!")
```

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
5. Once they are installed, run the brute_force.py file. You should get something like this once it runs:

```
PS C:\Users\ghosh\Documents\ghosh_subhodeep_midtermproj> python .\brute_force.py
Welcome to Association rule mining. Please select one of the following databases to find association rules from:
0.: Amazon.csv
1.: BestBuy.csv
2.: Kmart.csv
3.: Nike.csv
4.: Walmart.csv
Your choice (type and enter 'exit' to exit): 0
```

6. 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

```
Your choice (type and enter 'exit' to exit): 0
Choose a support from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent support: 0.45
Choose a confidence from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent confidence: 0.75
```

How wrong inputs are handled, an example:

```
Welcome to Association rule mining. Please select one of the following databases to find association rules from:
0.: Amazon.csv
1.: BestBuy.csv
2.: Kmart.csv
3.: Nike.csv
4.: Walmart.csv
Your choice (type and enter 'exit' to exit): 0
Choose a support from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent support: 0
Choose a confidence from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent confidence: 0
Wrong support or confidence. Please try again.
Welcome to Association rule mining. Please select one of the following databases to find association rules from:
0.: Amazon.csv
1.: BestBuy.csv
2.: Kmart.csv
3.: Nike.csv
4.: Walmart.csv
Your choice (type and enter 'exit' to exit): 2
Choose a support from 0.01 to 1 (e.g. 0.5) where 0.01 means 1 percent and 1 means 100 percent support: abc
Invalid input. Please input a number!
Welcome to Association rule mining. Please select one of the following databases to find association rules from:
0.: Amazon.csv
1.: BestBuy.csv
2.: Kmart.csv
3.: Nike.csv
4.: Walmart.csv
Your choice (type and enter 'exit' to exit): asd
Invalid input. Please input a number!
Welcome to Association rule mining. Please select one of the following databases to find association rules from:
0.: Amazon.csv
1.: BestBuy.csv
2.: Kmart.csv
3.: Nike.csv
4.: Walmart.csv
Your choice (type and enter 'exit' to exit):
```

7. 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.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                   Python

The individual items present in the database:
1. A Beginner's Guide
2. Java: The Complete Reference
3. Java For Dummies
4. Android Programming: The Big Nerd Ranch
5. Head First Java 2nd Edition
6. Beginning Programming with Java
7. Java 8 Pocket Guide
8. C++ Programming in Easy Steps
9. Effective Java (2nd Edition)
10. HTML and CSS: Design and Build Websites

All the transactions in the database:
TID Trans1: A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
TID Trans2: A Beginner's Guide, Java: The Complete Reference, Java For Dummies
TID Trans3: A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
TID Trans4: Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition , Beginning Programming with Java,
TID Trans5: Android Programming: The Big Nerd Ranch, Beginning Programming with Java, Java 8 Pocket Guide
TID Trans6: A Beginner's Guide, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
TID Trans7: A Beginner's Guide, Head First Java 2nd Edition , Beginning Programming with Java
TID Trans8: Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch,
TID Trans9: Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition , Beginning Programming with Java,
TID Trans10: Beginning Programming with Java, Java 8 Pocket Guide, C++ Programming in Easy Steps
TID Trans11: A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
TID Trans12: A Beginner's Guide, Java: The Complete Reference, Java For Dummies, HTML and CSS: Design and Build Websites
TID Trans13: A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Java 8 Pocket Guide,
TID Trans14: Java For Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
TID Trans15: Java For Dummies, Android Programming: The Big Nerd Ranch
TID Trans16: A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
TID Trans17: A Beginner's Guide, Java: The Complete Reference, Java For Dummies, Android Programming: The Big Nerd Ranch
TID Trans18: Head First Java 2nd Edition , Beginning Programming with Java, Java 8 Pocket Guide
TID Trans19: Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
TID Trans20: A Beginner's Guide, Java: The Complete Reference, Java For Dummies
```

```
-----BRUTE FORCE ALGORITHM-----

Using Brute Force we get itemsets --> {("A Beginner's Guide",): 0.55, ('Java: The Complete Reference',): 0.5, ('Java For Dummies',): 0.65, ('Android Programming
: The Big Nerd Ranch',): 0.6, ("A Beginner's Guide", 'Java: The Complete Reference'): 0.45, ("A Beginner's Guide", 'Java For Dummies'): 0.45, ('Java: The Comple
te Reference', 'Java For Dummies'): 0.5, ("A Beginner's Guide", 'Java: The Complete Reference', 'Java For Dummies'): 0.45}

Using Brute Force the association rules are as follows (with support and confidence) using Brute Force:
1. [("A Beginner's Guide",) --> ('Java: The Complete Reference',)] : Support = 45.0%; Confidence = 82.0%
2. [('Java: The Complete Reference',) --> ("A Beginner's Guide",)] : Support = 45.0%; Confidence = 90.0%
3. [("A Beginner's Guide",) --> ('Java For Dummies',)] : Support = 45.0%; Confidence = 82.0%
4. [('Java: The Complete Reference',) --> ('Java For Dummies',)] : Support = 50.0%; Confidence = 100.0%
5. [('Java For Dummies',) --> ('Java: The Complete Reference',)] : Support = 50.0%; Confidence = 77.0%
6. [("A Beginner's Guide",) --> ('Java: The Complete Reference', 'Java For Dummies')] : Support = 45.0%; Confidence = 82.0%
7. [('Java: The Complete Reference',) --> ("A Beginner's Guide", 'Java For Dummies')] : Support = 45.0%; Confidence = 90.0%
8. [("A Beginner's Guide", 'Java For Dummies') --> ('Java: The Complete Reference',)] : Support = 45.0%; Confidence = 100.0%
9. [("A Beginner's Guide", 'Java: The Complete Reference') --> ('Java For Dummies',)] : Support = 45.0%; Confidence = 100.0%
10. [('Java: The Complete Reference', 'Java For Dummies') --> ("A Beginner's Guide",)] : Support = 45.0%; Confidence = 90.0%
```

```
-----LIBRARY APRIORI ALGORITHM-----

Using library apriori algorithm we get itemsets --> {1: {("A Beginner's Guide",): 11, ('Java: The Complete Reference',): 10, ('Java For Dummies',): 13, ('Androi
d Programming: The Big Nerd Ranch',): 12}, 2: {("A Beginner's Guide", 'Java For Dummies'): 9, ("A Beginner's Guide", 'Java: The Complete Reference'): 9, ('Java
For Dummies', 'Java: The Complete Reference'): 10}, 3: {("A Beginner's Guide", 'Java For Dummies', 'Java: The Complete Reference'): 9}}

Using Library Apriori the association rules are as follows (with support and confidence) using Brute Force:
1. {A Beginner's Guide} -> {Java For Dummies} (conf: 0.818, supp: 0.450, lift: 1.259, conv: 1.925)
2. {Java: The Complete Reference} -> {A Beginner's Guide} (conf: 0.900, supp: 0.450, lift: 1.636, conv: 4.500)
3. {A Beginner's Guide} -> {Java: The Complete Reference} (conf: 0.818, supp: 0.450, lift: 1.636, conv: 2.750)
4. {Java: The Complete Reference} -> {Java For Dummies} (conf: 1.000, supp: 0.500, lift: 1.538, conv: 350000000.000)
5. {Java For Dummies} -> {Java: The Complete Reference} (conf: 0.769, supp: 0.500, lift: 1.538, conv: 2.167)
6. {Java For Dummies, Java: The Complete Reference} -> {A Beginner's Guide} (conf: 0.900, supp: 0.450, lift: 1.636, conv: 4.500)
7. {A Beginner's Guide, Java: The Complete Reference} -> {Java For Dummies} (conf: 1.000, supp: 0.450, lift: 1.538, conv: 350000000.000)
8. {A Beginner's Guide, Java For Dummies} -> {Java: The Complete Reference} (conf: 1.000, supp: 0.450, lift: 2.000, conv: 500000000.000)
9. {Java: The Complete Reference} -> {A Beginner's Guide, Java For Dummies} (conf: 0.900, supp: 0.450, lift: 2.000, conv: 5.500)
10. {A Beginner's Guide} -> {Java For Dummies, Java: The Complete Reference} (conf: 0.818, supp: 0.450, lift: 1.636, conv: 2.750)
```

```
-----LIBRARY FPT ALGORITHM-----

Using library FPT algorithm we get itemsets --> [{'Java: The Complete Reference'}, {"A Beginner's Guide", 'Java: The Complete Reference'}, {"A Beginner's Guide"
, 'Java For Dummies', 'Java: The Complete Reference'}, {'Java For Dummies', 'Java: The Complete Reference'}, {"A Beginner's Guide"}, {"A Beginner's Guide", 'Jav
a For Dummies'}, {'Android Programming: The Big Nerd Ranch'}, {'Java For Dummies'}]

Using Library FPT the association rules are as follows (with support and confidence) using Brute Force:
1. [{"A Beginner's Guide"}, {'Java: The Complete Reference'}, 0.8181818181818182]
2. [{'Java: The Complete Reference'}, {"A Beginner's Guide"}, 0.9]
3. [{"A Beginner's Guide"}, {'Java For Dummies', 'Java: The Complete Reference'}, 0.8181818181818182]
4. [{'Java: The Complete Reference'}, {"A Beginner's Guide", 'Java For Dummies'}, 0.9]
5. [{"A Beginner's Guide", 'Java For Dummies'}, {'Java: The Complete Reference'}, 1.0]
6. [{"A Beginner's Guide", 'Java: The Complete Reference'}, {'Java For Dummies'}, 1.0]
7. [{'Java For Dummies', 'Java: The Complete Reference'}, {"A Beginner's Guide"}, 0.9]
8. [{'Java For Dummies'}, {'Java: The Complete Reference'}, 0.7692307692307693]
9. [{'Java: The Complete Reference'}, {'Java For Dummies'}, 1.0]
10. [{"A Beginner's Guide"}, {'Java For Dummies'}, 0.8181818181818182]



Execution time for Brute Force: 0.0035088062286376953
Execution time for Library Apriori: 0.0
Execution time for Library FPT: 0.0
```

8. 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.

```
Welcome to Association rule mining. Please select one of the following databases to find association rules from:
0.: Amazon.csv
1.: BestBuy.csv
2.: Kmart.csv
3.: Nike.csv
4.: Walmart.csv
Your choice (type and enter 'exit' to exit): exit
Thank you, have a nice day!
PS C:\Users\ghosh\Documents\ghosh_subhodeep_midtermproj>
                                                              Ln 32, Col 1   Spaces: 4   UT
```

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.

sources.txt

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