

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 patterns 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 FPGrowth 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 is 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, FPGrowth (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, let's 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  $k \geq 1$  (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 be  $10C2$  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  $\geq$  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  $A \rightarrow B,C$  then we just divide the support of {A,B,C} by the support of {A}. Similarly for  $A,B \rightarrow C$  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 frequent 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.

```

1 Item Name,
2 1,Running Shoe
3 2,Soccer Shoe
4 3,Socks
5 4,Swimming Shirt
6 5,Dry Fit V-Nick
7 6,Rash Guard
8 7,Sweatshirts
9 8,Hoodies
10 9,Tech Pants
11 10,Modern Pants
12 Transaction ID,Transaction
13 Trans1,"Running Shoe, Socks, Sweatshirts, Modern Pants"
14 Trans2,"Running Shoe, Socks, Sweatshirts"
15 Trans3,"Running Shoe, Socks, Sweatshirts, Modern Pants"
16 Trans4,"Running Shoe, Sweatshirts, Modern Pants"
17 Trans5,"Running Shoe, Socks, Sweatshirts, Modern Pants, Soccer Shoe"
18 Trans6,"Running Shoe, Socks, Sweatshirts"
19 Trans7,"Running Shoe, Socks, Sweatshirts, Modern Pants, Tech Pants, Rash Guard, Hoodies"
20 Trans8,"Swimming Shirt, Socks, Sweatshirts"
21 Trans9,"Swimming Shirt, Rash Guard, Dry Fit V-Nick, Hoodies, Tech Pants"
22 Trans10,"Swimming Shirt, Rash Guard, Dry"
23 Trans11,"Swimming Shirt, Rash Guard, Dry Fit V-Nick"
24 Trans12,"Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Hoodies, Tech Pants, Dry Fit V-Nick"
25 Trans13,"Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Tech Pants, Dry Fit V-Nick, Hoodies"
26 Trans14,"Running Shoe, Swimming Shirt, Rash Guard, Tech Pants, Hoodies, Dry Fit V-Nick"
27 Trans15,"Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Dry Fit V-Nick, Rash Guard, Tech Pants"
28 Trans16,"Swimming Shirt, Soccer Shoe, Hoodies, Dry Fit V-Nick, Tech Pants, Rash Guard"
29 Trans17,"Running Shoe, Socks"
30 Trans18,"Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Rash Guard, Tech Pants, Dry Fit V-Nick"
31 Trans19,"Running Shoe, Swimming Shirt, Rash Guard"
32 Trans20,"Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Tech Pants, Rash Guard, Dry Fit V-Nick"
33

```

Thank you, have a nice day!  
PS C:\Users\ghosh\Documents\ghosh\_subhodeep\_midtermproj>

## 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 import pandas as pd
2
3 class Createdataset:
4     def __init__(self, filepath:str) -> None:
5         self.data = pd.read_csv(filepath,)
6
7     def extract_items(self):
8         items = self.data.iloc[:10]
9         # print(items)
10        return items
11
12    def extract_transac(self):
13        transactions = self.data.iloc[10:]
14        transactions.columns = transactions.iloc[0]
15        transactions = transactions[1:]
16        transactions.reset_index(drop=True, inplace=True)
17        # print(transactions)
18        return transactions
19
20
21

```

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.

```

File Edit Selection View Go Run Terminal Help
ghosh_subhodeep_midtermproj

EXPLORER
GHOSH_SUBHODEEP_MI...
  > __pycache__
  > .venv
  Amazon.csv
  BestBuy.csv
  brute_force.py
  Kmart.csv
  Nike.csv
  read_files.py
  requirements.txt
  sources.txt
  Walmart.csv

brute_force.py > BruteForce
1 from read_files import CreateDataset
2 import itertools
3 from copy import deepcopy
4 from efficient_apriori import apriori
5 from fpgrowth_py import fpgrowth
6 import time
7
8 AMAZON = "Amazon.csv"
9 BESTBUY = "BestBuy.csv"
10 KMART = "Kmart.csv"
11 NIKE = "Nike.csv"
12 WALMART = "Walmart.csv"
13

```

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

```

File Edit Selection View Go Run Terminal Help
ghosh_subhodeep_midtermproj

EXPLORER
GHOSH_SUBHODEEP_MI...
  > __pycache__
  > .venv
  Amazon.csv
  BestBuy.csv
  brute_force.py
  Kmart.csv
  Nike.csv
  read_files.py
  requirements.txt
  sources.txt
  Walmart.csv

brute_force.py > BruteForce
14 class BruteForce:
15     def __init__(self, database, support = 0.5, confidence = 0.5) -> None:
16         self.database = database
17         cd = CreateDataset(self.database)
18         self.items = cd.extract_items()
19         self.transactions = cd.extract_transac()
20         self.support = support
21         self.confidence = confidence
22         self.prepare_itemsets_recs()
23
24     def show_items_transacs(self):
25         print("\n\nThe individual items present in the database:")
26         for i in range(self.items.shape[0]):
27             print(f"{i+1}. {self.items.iloc[i,1]}")
28         print("\n\nAll the transactions in the database:")
29         for i in range(self.transactions.shape[0]):
30             print(f"TID {self.transactions.iloc[i, 0]}: {self.transactions.iloc[i,1]}")
31         print("\n\n")
32
33     def prepare_itemsets_recs(self):
34         '''Preparing single itemsets and transactions record for each individual transaction from the pandas dataframe'''
35         self.itemsets = self.items.iloc[:, 1].to_list()
36         self.transac_rec = {}
37         for i in range(self.transactions.shape[0]):
38             id, trans = self.transactions.iloc[i, 0], self.transactions.iloc[i,1]
39             self.transac_rec[id] = trans.split(',')
40         # print(self.itemsets)
41

```

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

```

File Edit Selection View Go Run Terminal Help
ghosh_subhodeep_midtermproj

EXPLORER
GHOSH_SUBHODEEP_MI...
  > __pycache__
  > .venv
  Amazon.csv
  BestBuy.csv
  brute_force.py
  Kmart.csv
  Nike.csv
  read_files.py
  requirements.txt
  sources.txt
  Walmart.csv

brute_force.py > BruteForce
14 class BruteForce:
15     def __init__(self, database, support = 0.5, confidence = 0.5) -> None:
16         self.database = database
17         cd = CreateDataset(self.database)
18         self.items = cd.extract_items()
19         self.transactions = cd.extract_transac()
20         self.support = support
21         self.confidence = confidence
22         self.prepare_itemsets_recs()
23
24     def show_items_transacs(self):
25         print("\n\nThe individual items present in the database:")
26         for i in range(self.items.shape[0]):
27             print(f"{i+1}. {self.items.iloc[i,1]}")
28         print("\n\nAll the transactions in the database:")
29         for i in range(self.transactions.shape[0]):
30             print(f"TID {self.transactions.iloc[i, 0]}: {self.transactions.iloc[i,1]}")
31         print("\n\n")
32
33     def prepare_itemsets_recs(self):
34         '''Preparing single itemsets and transactions record for each individual transaction from the pandas dataframe'''
35         self.itemsets = self.items.iloc[:, 1].to_list()
36         self.transac_rec = {}
37         for i in range(self.transactions.shape[0]):
38             id, trans = self.transactions.iloc[i, 0], self.transactions.iloc[i,1]
39             self.transac_rec[id] = trans.split(',')
40         # print(self.itemsets)
41
42     def get_support(self, combos=None):
43         '''returning a dictionary containing items whose frequency is greater than or equal to the min support defined by user'''
44         freq = {}
45         l = self.transactions.shape[0]
46         if combos:
47             for combo in combos:
48                 for vals in self.transac_rec.values():
49                     if all(item in vals for item in combo):
50                         freq[combo] = (freq.get(combo, 0) + 1)
51             freq[combo] = round(freq.get(combo, 0)/l,2)
52         else:
53             for i in self.itemsets:
54                 for vals in self.transac_rec.values():
55                     if i in vals:
56                         freq[(i,)] = (freq.get((i,), 0) + 1)
57             freq[(i,)] = round(freq.get((i,), 0)/l,2)
58         temp = deepcopy(freq)
59         for i in temp.keys():
60             if freq[i] < self.support:
61                 freq.pop(i) # Popping out the itemsets whose support is lower than the user defined min support
62         return freq
63

```

- 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.

```

File Edit Selection View Go Run Terminal Help
ghosh_subhodeep_midtermproj

EXPLORER
GHOSH_SUBHODEEP_MI...
  > __pycache__
  > .venv
  Amazon.csv
  BestBuy.csv
  brute_force.py
  Kmart.csv
  Nike.csv
  read_files.py
  requirements.txt
  sources.txt
  Walmart.csv

brute_force.py x read_files.py sources.txt
class BruteForce:
    def gen_freq_sets(self):
        '''Generating all frequent itemsets'''
        freq = self.get_support()
        max_freq = max(freq.values()) if len(freq.values()) > 0 else 0
        self.freq_itemsets = freq
        k = 1
        while(max_freq >= self.support):
            combos = list(itertools.combinations(self.itemsets, k+1))
            freq = self.get_support(combos)
            if not freq:
                max_freq = 0
                break
            max_freq = max(freq.values())
            self.freq_itemsets.update(freq)
            k += 1

```

- The `get_asso_rules()` function is responsible for generating and checking all the possible association rules for a given  $k$ -itemset,  $k \geq 2$ , 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.

```

File Edit Selection View Go Run Terminal Help
ghosh_subhodeep_midtermproj

EXPLORER
GHOSH_SUBHODEEP_MI...
  > __pycache__
  > .venv
  Amazon.csv
  BestBuy.csv
  brute_force.py
  Kmart.csv
  Nike.csv
  read_files.py
  requirements.txt
  sources.txt
  Walmart.csv

brute_force.py x read_files.py sources.txt
class BruteForce:
    def gen_asso_rules(self):
        self.gen_freq_sets()
        all_combos = {key:val for key, val in self.freq_itemsets.items() if len(key) > 1}
        asso_rules = {}
        for combo in all_combos:
            asso_list = []
            for l in range(len(combo)-1):
                asso_list.extend(itertools.combinations(combo, l+1))
            combo_supp = all_combos[combo]
            for i in asso_list:
                others = tuple(set(combo) - set(i))
                if i in self.freq_itemsets.keys():
                    conf = round(combo_supp/self.freq_itemsets[i],2)
                    if conf >= self.confidence:
                        asso_rules[(i,combo)] = [others, combo_supp, conf]
                if others in self.freq_itemsets.keys():
                    rev_conf = round(combo_supp/self.freq_itemsets[others],2)
                    if rev_conf >= self.confidence:
                        asso_rules[(others, combo)] = [i, combo_supp, rev_conf]
            return self.freq_itemsets, asso_rules

```

- 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.

```

14 class BruteForce:
15     def __init__(self, transac_rec, support, confidence):
16         self.transac_rec = transac_rec
17         self.support = support
18         self.confidence = confidence
19         self.freq_itemsets, self.asso_rules = self.gen_asso_rules()
20
21     def testing(self):
22         transacs = [tuple(val) for val in self.transac_rec.values()]
23         start_time_ap = time.time()
24         itemsets, rules = apriori(transacs, min_support=self.support, min_confidence=self.confidence)
25         end_time_ap = time.time()
26         counter = 0
27         print("-----LIBRARY APRIORI ALGORITHM-----")
28         print(f"\nUsing library apriori algorithm we get itemsets --> {itemsets} \n")
29         print("Using Library Apriori the association rules are as follows (with support and confidence) using Brute Force:")
30         for rule in rules:
31             counter += 1
32             print(f"{counter}. {rule}")
33         print("\n\n")
34         print("-----LIBRARY FPT ALGORITHM-----")
35         start_time_fp = time.time()
36         freqItemSet, rules = fpgrowth(transacs, self.support, self.confidence)
37         end_time_fp = time.time()
38         print(f"\nUsing Library FPT algorithm we get itemsets --> {freqItemSet} \n")
39         print("Using Library FPT the association rules are as follows (with support and confidence) using Brute Force:")
40         counter = 0
41         for rule in rules:
42             counter += 1
43             print(f"{counter}. {rule}")
44         print("\n\n")
45         exec_time_ap = end_time_ap - start_time_ap
46         exec_time_fp = end_time_fp - start_time_fp
47         return exec_time_ap, exec_time_fp

```

- 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.

```

127 return exec_time_ap, exec_time_fp
128
129 n = 0
130 while n != "exit":
131     print("Welcome to Association rule mining. Please select one of the following databases to find association rules from:")
132     database = [AMAZON, BESTBUY, KMART, NIKE, WALMART]
133     for i in range(len(database)):
134         print(f"{i}.: {database[i]}")
135     n = input("Your choice (type and enter 'exit' to exit): ")
136     if n == 'exit': break
137     try:
138         n = int(n)
139         if n < 0 or n > 4:
140             print("Invalid choice. Please try again!")
141             continue
142     except ValueError:
143         print("Invalid input. Please input a number!")
144         continue
145     db_of_choice = database[n]
146     try:
147         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: "))
148         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: "))
149         if (supp >= 1 or supp<=0) or (conf>=1 or conf<=0):
150             print("Wrong support or confidence. Please try again.")
151             continue
152     except ValueError:
153         print("Invalid input. Please input a number!")
154         continue

```

```

146 try:
147     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: "))
148     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: "))
149     if (supp >= 1 or supp<=0) or (conf>=1 or conf<=0):
150         print("Wrong support or confidence. Please try again.")
151         continue
152 except ValueError:
153     print("Invalid input. Please input a number!")
154     continue
155 bf = BruteForce(db_of_choice, supp, conf)
156 start_time = time.time()
157 freq_itemsets, asso_rules = bf.gen_asso_rules()
158 end_time = time.time()
159 bf.show_items(transacs())
160 print("-----BRUTE FORCE ALGORITHM-----")
161 print(f"\nUsing Brute Force we get itemsets --> {freq_itemsets}\n")
162 print("Using Brute Force the association rules are as follows (with support and confidence) using Brute Force:")
163 counter = 0
164 for key, val in asso_rules.items():
165     counter += 1
166     print(f"{counter}. [{key[0]}] --> {val[0]} : Support = {val[1]*100}%; Confidence = {val[2]*100}%")
167 print("\n\n")
168 exec_time = end_time - start_time
169 exec_time_ap, exec_time_fp = bf.testing()
170 print(f"Execution time for Brute Force: {exec_time} \nExecution time for Library Apriori: {exec_time_ap} \nExecution time for Library FPT: {exec_time_fp}")
171 print("\n\n")
172 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  $0.45 \times 100\% = 45\%$ . 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):
```



- 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.

```

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 Complete 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:

- | Index | Input  | Output   | Support          | Confidence          |
|-------|--|--|------------------|---------------------|
| 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, {'Android 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:

- 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", "Java For Dummies", "Java: The Complete Reference"}, {"A Beginner's Guide", "Java 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: 0.0035888062286376953
```

```
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 - UTF

## 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 1000<sup>th</sup> 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.