NAME: Anagha Bharadwaj
NJIT UCID: AB2929
Email Address: ab2929@njit.edu
10/13/2024
Professor: Yasser Abduallah
CS 634101 Data Mining

MIDTERM PROJECT

Title: Implementation and Comparison of Apriori, FPGrowth and Brute force Method

Abstract: In this project, I implement data mining techniques to identify frequent items and association rules using the brute force method, the Apriori algorithm, and the FP-Growth algorithm across multiple datasets. Through this project I compare between the algorithms to find the most efficient and accurate one using several retail shop data.

Introduction:

Data mining refers to sorting through collected data to come up with key information or patterns that are used to solve a problem or grow businesses.
Brute force method is a systematic standard procedure where it digs through all the possible outcomes and provides the best solution. The Apriori algorithm is a machine learning algorithm used to identify frequent itemsets and association rules in relational databases. It's a common algorithm in data mining and is used in a variety of applications, such as market basket analysis, identifying diseases, and finding what customers frequently purchase. It is the perfect way to find the probability of outcomes. The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance. It uses a simple divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

Implementation and workflow:

1. Get user's input of minimum support, minimum confidence and the dataset. Dataset used here is a combination of clothing and other retail items. To ensure data accuracy, data is preprocessed, filtering unique items and sorting them based on a predefined order.
2. Create dictionaries and find the frequent items and their count

3. Iterate through the dictionary until all the frequent items till Nth frequent items.Starts with single items (itemset size K = 1) and proceed to K = 2, K = 3, and so on.
4. Through brute force method calculate association rules. Association rules that satisfy both the minimum support and minimum confidence requirements are extracted. These rules reveal valuable insights into which items are often purchased together
5. Perform the apriori algorithm and fp growth algorithm.
6. Analyze and compare between the three models to find the best outcome.

Core Concepts and Principle:

Association rules: Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases.

Support and confidence:The support of an itemset is the number of transactions in which the itemset appears, divided by the total number of transactions. Confidence is a measure of the likelihood that an itemset will appear if another itemset appears

Frequent Items Discovery: Discovery of the set of items that frequently occur together in a dataset. It describes the relationship between the data.

Results and Evaluation:

The FP-tree algorithm proved to be the most efficient method for finding frequent items and association rules, with the shortest execution time of approximately 0.03 seconds.

The Apriori algorithm showed a moderate performance, taking around 0.04 seconds. While it is more efficient than the brute force method, it still faces challenges related to the generation of candidate itemsets, especially in large datasets.

The Brute Force method had the longest execution time at approximately 0.06 seconds, making it the slowest and less efficient for larger datasets.

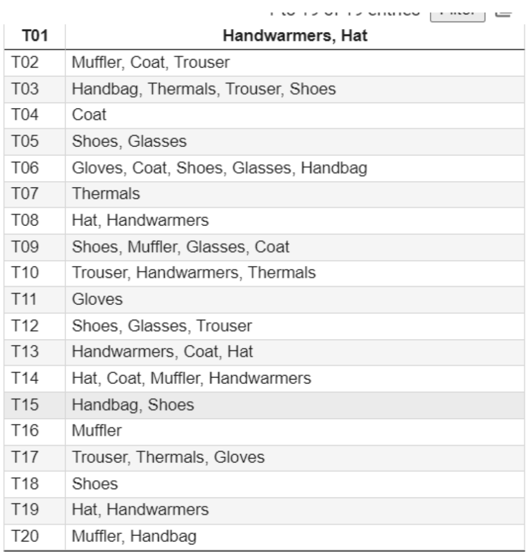
Conclusion:

A successful demonstration of data mining concepts, principles, and methods are applied on data and comparison and analysis takes place between the three methods to identify the best method to find frequent datasets and association rules.

Screenshots:

Here are what the csv files look like .

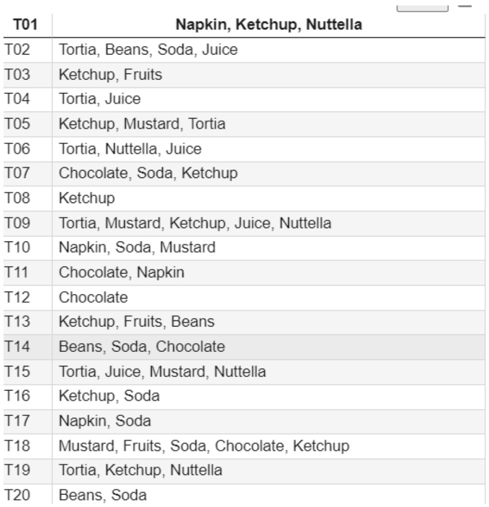
Figure 1 : Burlington



1 to 19 of 19 entries Filter

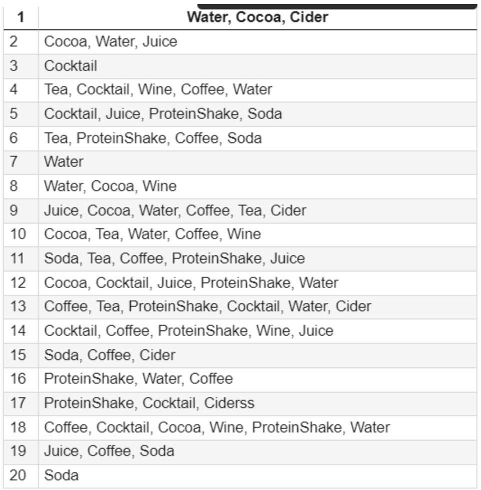
T01	Handwarmers, Hat
T02	Muffler, Coat, Trouser
T03	Handbag, Thermals, Trouser, Shoes
T04	Coat
T05	Shoes, Glasses
T06	Gloves, Coat, Shoes, Glasses, Handbag
T07	Thermals
T08	Hat, Handwarmers
T09	Shoes, Muffler, Glasses, Coat
T10	Trouser, Handwarmers, Thermals
T11	Gloves
T12	Shoes, Glasses, Trouser
T13	Handwarmers, Coat, Hat
T14	Hat, Coat, Muffler, Handwarmers
T15	Handbag, Shoes
T16	Muffler
T17	Trouser, Thermals, Gloves
T18	Shoes
T19	Hat, Handwarmers
T20	Muffler, Handbag

Figure 2 : Costco dataset



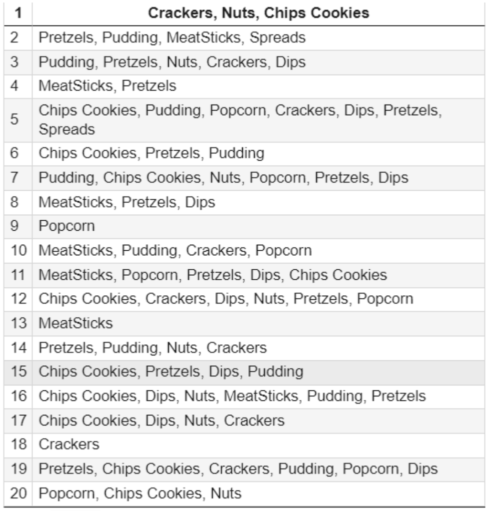
T01	Napkin, Ketchup, Nuttella
T02	Tortia, Beans, Soda, Juice
T03	Ketchup, Fruits
T04	Tortia, Juice
T05	Ketchup, Mustard, Tortia
T06	Tortia, Nuttella, Juice
T07	Chocolate, Soda, Ketchup
T08	Ketchup
T09	Tortia, Mustard, Ketchup, Juice, Nuttella
T10	Napkin, Soda, Mustard
T11	Chocolate, Napkin
T12	Chocolate
T13	Ketchup, Fruits, Beans
T14	Beans, Soda, Chocolate
T15	Tortia, Juice, Mustard, Nuttella
T16	Ketchup, Soda
T17	Napkin, Soda
T18	Mustard, Fruits, Soda, Chocolate, Ketchup
T19	Tortia, Ketchup, Nuttella
T20	Beans, Soda

Figure 3: juicebar



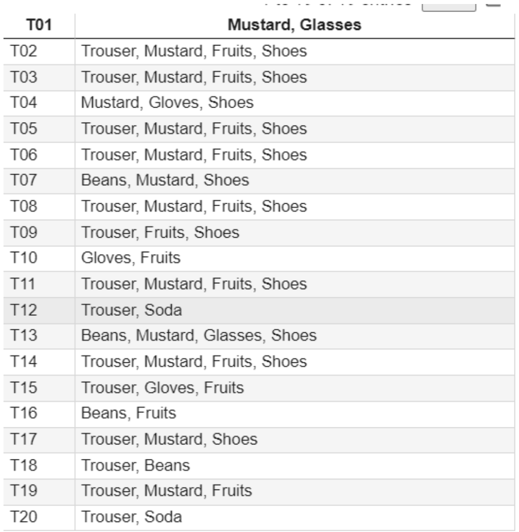
1	Water, Cocoa, Cider
2	Cocoa, Water, Juice
3	Cocktail
4	Tea, Cocktail, Wine, Coffee, Water
5	Cocktail, Juice, ProteinShake, Soda
6	Tea, ProteinShake, Coffee, Soda
7	Water
8	Water, Cocoa, Wine
9	Juice, Cocoa, Water, Coffee, Tea, Cider
10	Cocoa, Tea, Water, Coffee, Wine
11	Soda, Tea, Coffee, ProteinShake, Juice
12	Cocoa, Cocktail, Juice, ProteinShake, Water
13	Coffee, Tea, ProteinShake, Cocktail, Water, Cider
14	Cocktail, Coffee, ProteinShake, Wine, Juice
15	Soda, Coffee, Cider
16	ProteinShake, Water, Coffee
17	ProteinShake, Cocktail, Ciderss
18	Coffee, Cocktail, Cocoa, Wine, ProteinShake, Water
19	Juice, Coffee, Soda
20	Soda

Figure4: Shoprite



1	Crackers, Nuts, Chips Cookies
2	Pretzels, Pudding, MeatSticks, Spreads
3	Pudding, Pretzels, Nuts, Crackers, Dips
4	MeatSticks, Pretzels
5	Chips Cookies, Pudding, Popcorn, Crackers, Dips, Pretzels, Spreads
6	Chips Cookies, Pretzels, Pudding
7	Pudding, Chips Cookies, Nuts, Popcorn, Pretzels, Dips
8	MeatSticks, Pretzels, Dips
9	Popcorn
10	MeatSticks, Pudding, Crackers, Popcorn
11	MeatSticks, Popcorn, Pretzels, Dips, Chips Cookies
12	Chips Cookies, Crackers, Dips, Nuts, Pretzels, Popcorn
13	MeatSticks
14	Pretzels, Pudding, Nuts, Crackers
15	Chips Cookies, Pretzels, Dips, Pudding
16	Chips Cookies, Dips, Nuts, MeatSticks, Pudding, Pretzels
17	Chips Cookies, Dips, Nuts, Crackers
18	Crackers
19	Pretzels, Chips Cookies, Crackers, Pudding, Popcorn, Dips
20	Popcorn, Chips Cookies, Nuts

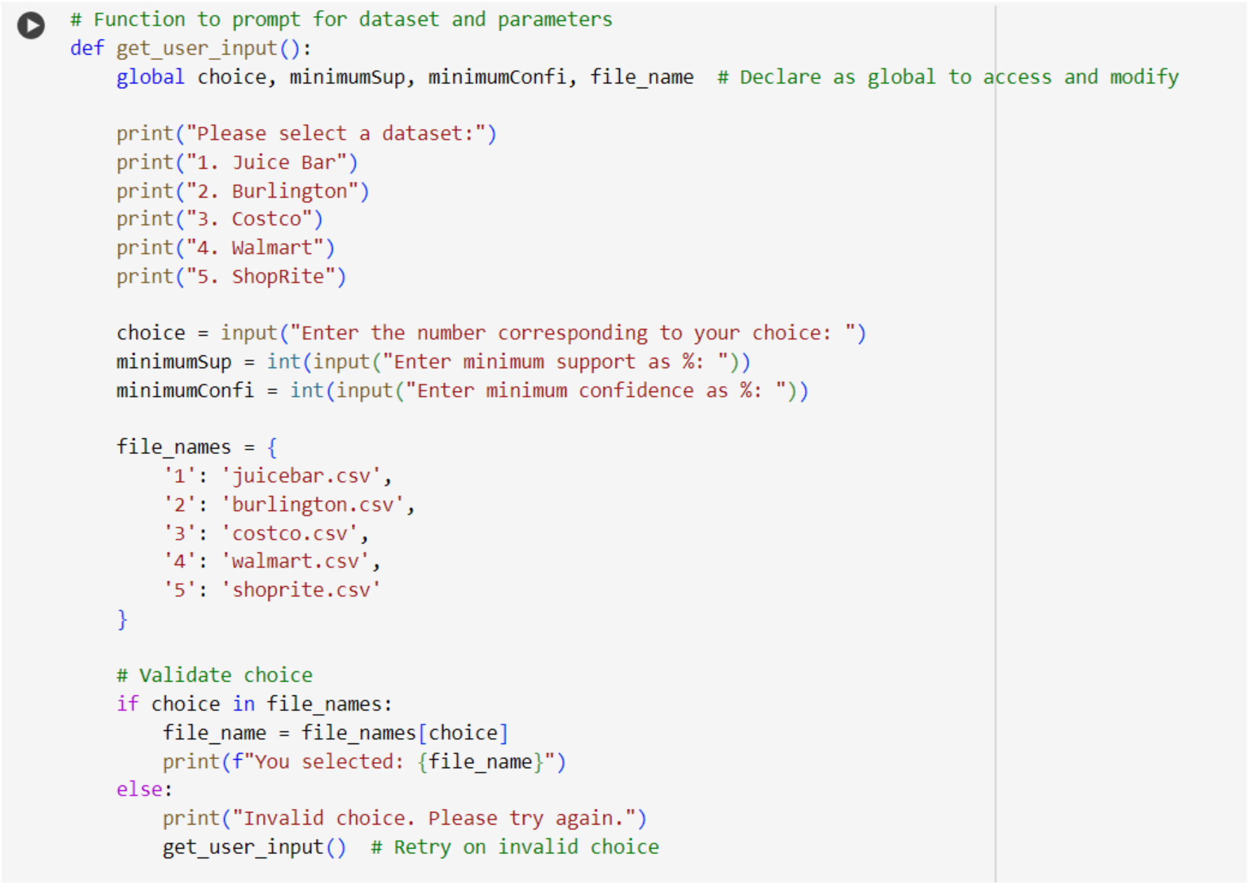
Figure 5: Walmart



T01	Mustard, Glasses
T02	Trouser, Mustard, Fruits, Shoes
T03	Trouser, Mustard, Fruits, Shoes
T04	Mustard, Gloves, Shoes
T05	Trouser, Mustard, Fruits, Shoes
T06	Trouser, Mustard, Fruits, Shoes
T07	Beans, Mustard, Shoes
T08	Trouser, Mustard, Fruits, Shoes
T09	Trouser, Fruits, Shoes
T10	Gloves, Fruits
T11	Trouser, Mustard, Fruits, Shoes
T12	Trouser, Soda
T13	Beans, Mustard, Glasses, Shoes
T14	Trouser, Mustard, Fruits, Shoes
T15	Trouser, Gloves, Fruits
T16	Beans, Fruits
T17	Trouser, Mustard, Shoes
T18	Trouser, Beans
T19	Trouser, Mustard, Fruits
T20	Trouser, Soda

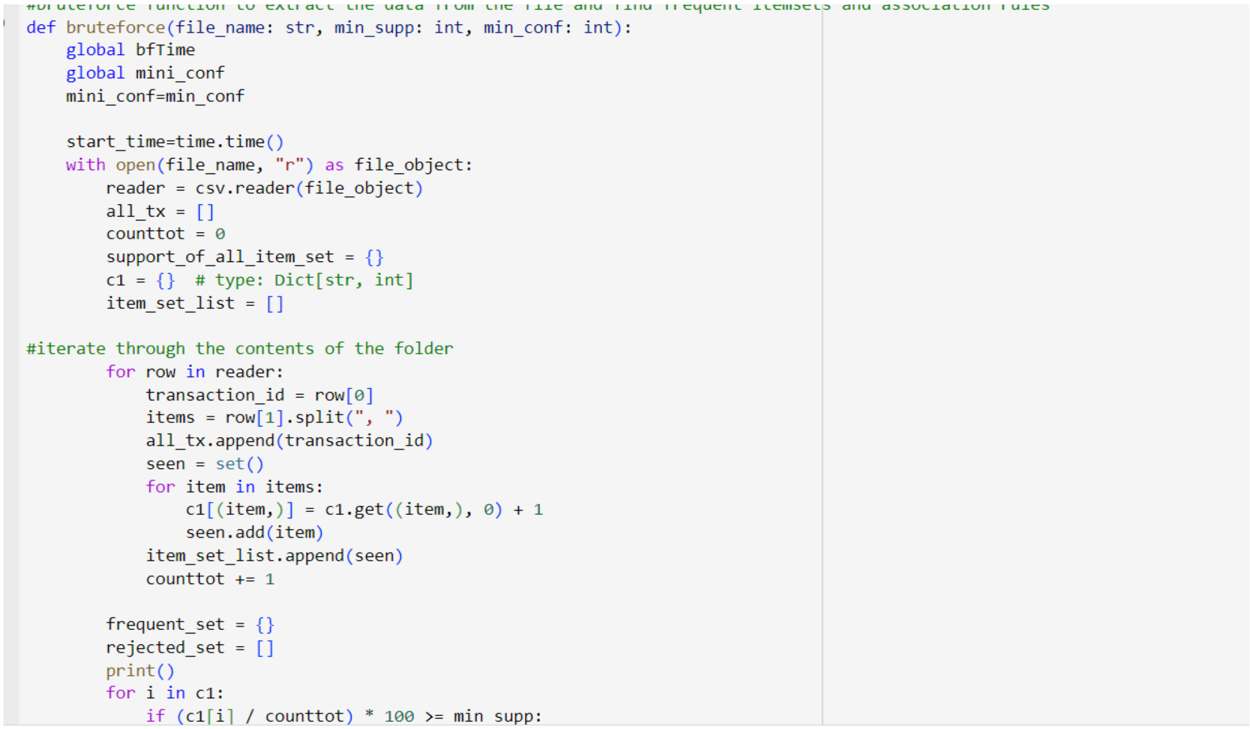
Below are screenshots from the codefiles:

Users are requested to enter details regarding minimum confidence, support and to select what dataset to be used.



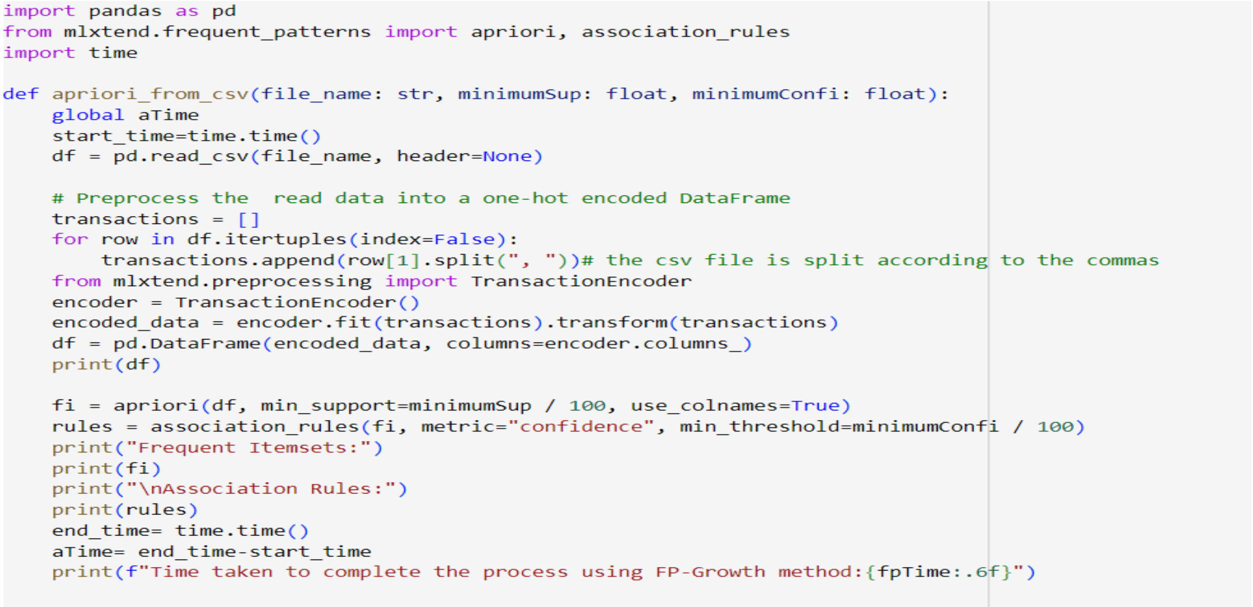
```python
# Function to prompt for dataset and parameters
def get_user_input():
    global choice, minimumSup, minimumConfi, file_name  # Declare as global to access and modify

    print("Please select a dataset:")
    print("1. Juice Bar")
    print("2. Burlington")
    print("3. Costco")
    print("4. Walmart")
    print("5. ShopRite")

    choice = input("Enter the number corresponding to your choice: ")
    minimumSup = int(input("Enter minimum support as %: "))
    minimumConfi = int(input("Enter minimum confidence as %: "))

    file_names = {
        '1': 'juicebar.csv',
        '2': 'burlington.csv',
        '3': 'costco.csv',
        '4': 'walmart.csv',
        '5': 'shoprite.csv'
    }

    # Validate choice
    if choice in file_names:
        file_name = file_names[choice]
        print(f"You selected: {file_name}")
    else:
        print("Invalid choice. Please try again.")
        get_user_input()  # Retry on invalid choice
```

The input from the user is globally saved and is used throughout the model execution.
Using the brute force method individual items , frequent items and all the transactions are displayed.
Association rules are calculated based on the confidence
Confidence(A→B)=Support(A)Support(A∪B). Compare the calculated confidence against a minimum confidence threshold to determine strong rules.

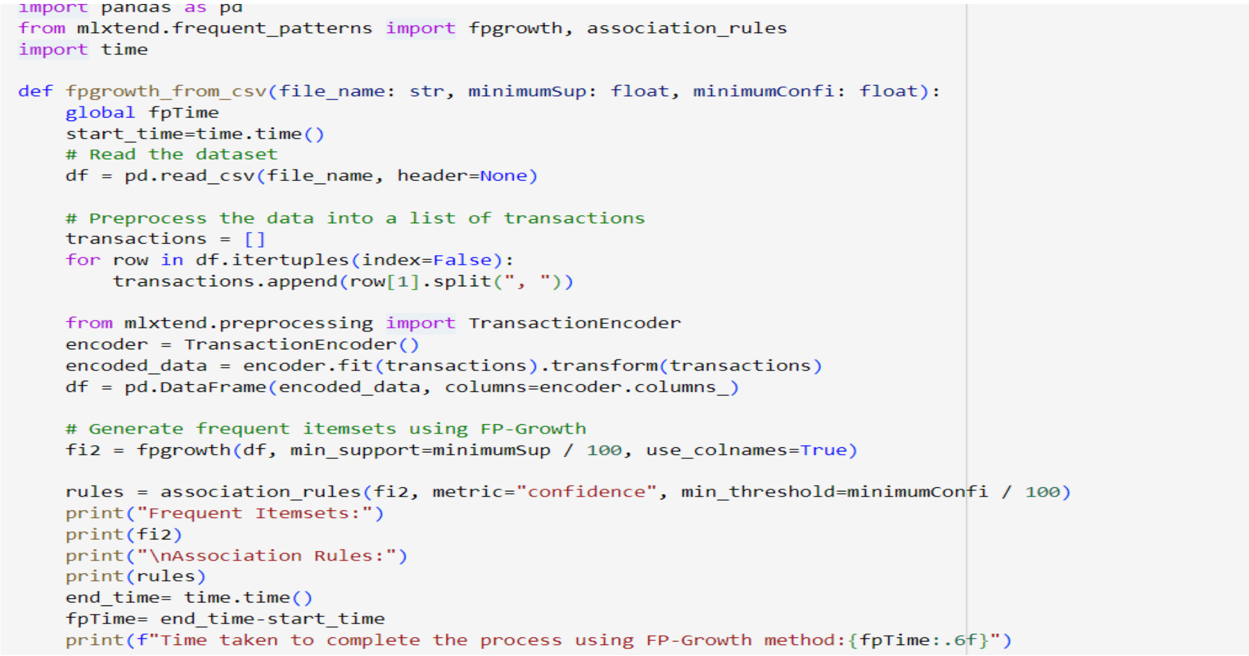


```
#bruteforce function to extract the data from the file and find frequent itemsets and association rules
def bruteforce(file_name: str, min_supp: int, min_conf: int):
    global bfTime
    global mini_conf
    mini_conf=min_conf

    start_time=time.time()
    with open(file_name, "r") as file_object:
        reader = csv.reader(file_object)
        all_tx = []
        counttot = 0
        support_of_all_item_set = {}
        c1 = {}  # type: Dict[str, int]
        item_set_list = []

#iterate through the contents of the folder
        for row in reader:
            transaction_id = row[0]
            items = row[1].split(", ")
            all_tx.append(transaction_id)
            seen = set()
            for item in items:
                c1[(item,)] = c1.get((item,), 0) + 1
                seen.add(item)
            item_set_list.append(seen)
            counttot += 1

        frequent_set = {}
        rejected_set = []
        print()
        for i in c1:
            if (c1[i] / counttot) * 100 >= min_supp:
```

Using the inbuilt libraries for FpGrowth Algorithm and the Apriori Algorithm is used to get association rules.



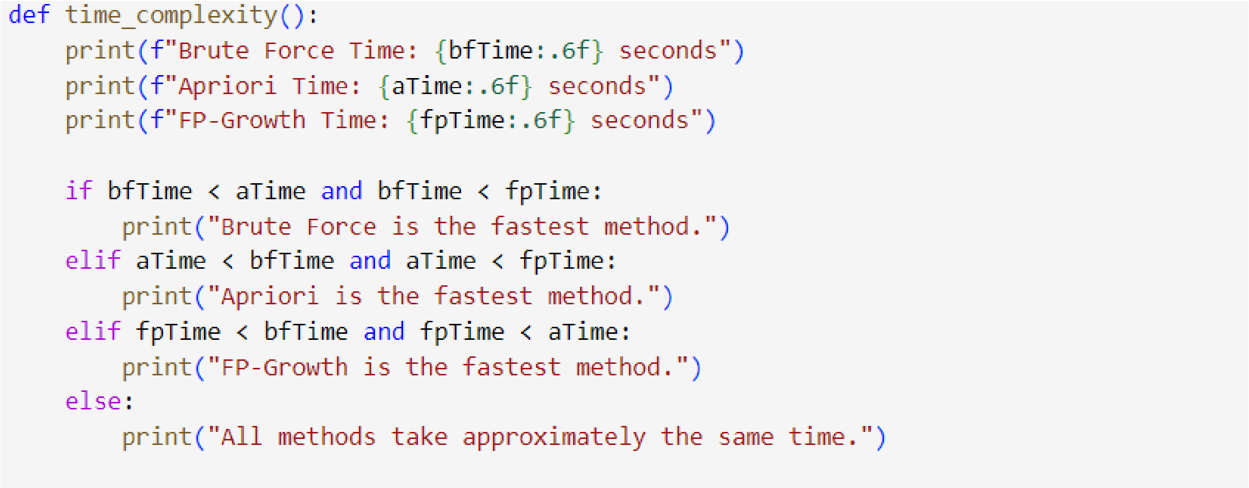
```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import time

def apriori_from_csv(file_name: str, minimumSup: float, minimumConfi: float):
    global aTime
    start_time=time.time()
    df = pd.read_csv(file_name, header=None)

    # Preprocess the  read data into a one-hot encoded DataFrame
    transactions = []
    for row in df.itertuples(index=False):
        transactions.append(row[1].split(", "))# the csv file is split according to the commas
    from mlxtend.preprocessing import TransactionEncoder
    encoder = TransactionEncoder()
    encoded_data = encoder.fit(transactions).transform(transactions)
    df = pd.DataFrame(encoded_data, columns=encoder.columns_)
    print(df)

    fi = apriori(df, min_support=minimumSup / 100, use_colnames=True)
    rules = association_rules(fi, metric="confidence", min_threshold=minimumConfi / 100)
    print("Frequent Itemsets:")
    print(fi)
    print("\nAssociation Rules:")
    print(rules)
    end_time= time.time()
    aTime= end_time-start_time
    print(f"Time taken to complete the process using FP-Growth method:{fpTime:.6f}")
```



```python
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
import time

def fpgrowth_from_csv(file_name: str, minimumSup: float, minimumConfi: float):
    global fpTime
    start_time=time.time()
    # Read the dataset
    df = pd.read_csv(file_name, header=None)

    # Preprocess the data into a list of transactions
    transactions = []
    for row in df.itertuples(index=False):
        transactions.append(row[1].split(", "))

    from mlxtend.preprocessing import TransactionEncoder
    encoder = TransactionEncoder()
    encoded_data = encoder.fit(transactions).transform(transactions)
    df = pd.DataFrame(encoded_data, columns=encoder.columns_)

    # Generate frequent itemsets using FP-Growth
    fi2 = fpgrowth(df, min_support=minimumSup / 100, use_colnames=True)

    rules = association_rules(fi2, metric="confidence", min_threshold=minimumConfi / 100)
    print("Frequent Itemsets:")
    print(fi2)
    print("\nAssociation Rules:")
    print(rules)
    end_time= time.time()
    fpTime= end_time-start_time
    print(f"Time taken to complete the process using FP-Growth method:{fpTime:.6f}")
```

The final analysis is done by comparing the time taken to each of the algorithm to execute the entire code.

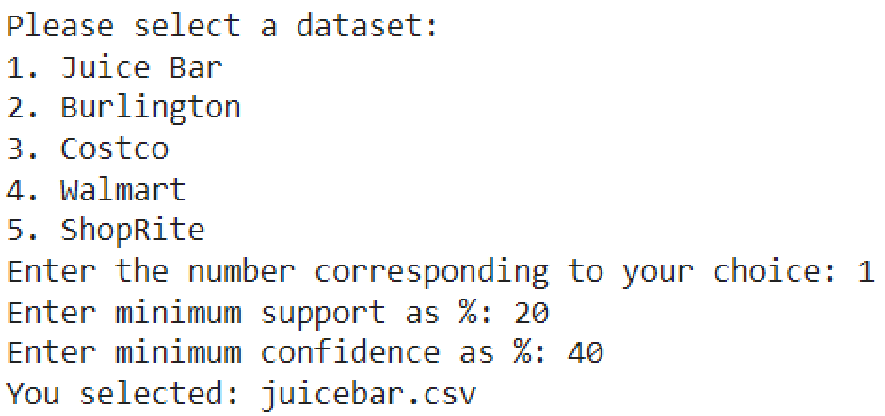
The above image shows the imported libraries utilized to complete this project.



```python
def time_complexity():
    print(f"Brute Force Time: {bfTime:.6f} seconds")
    print(f"Apriori Time: {aTime:.6f} seconds")
    print(f"FP-Growth Time: {fpTime:.6f} seconds")

    if bfTime < aTime and bfTime < fpTime:
        print("Brute Force is the fastest method.")
    elif aTime < bfTime and aTime < fpTime:
        print("Apriori is the fastest method.")
    elif fpTime < bfTime and fpTime < aTime:
        print("FP-Growth is the fastest method.")
    else:
        print("All methods take approximately the same time.")
```

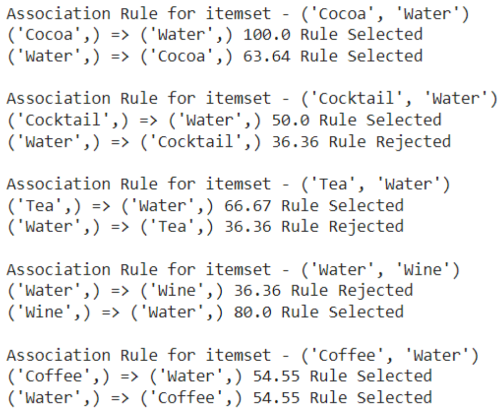
Outputs:

The user is asked to enter details and here juice bar dataset is selected along with the support and confidence of 20 and 40 percent.

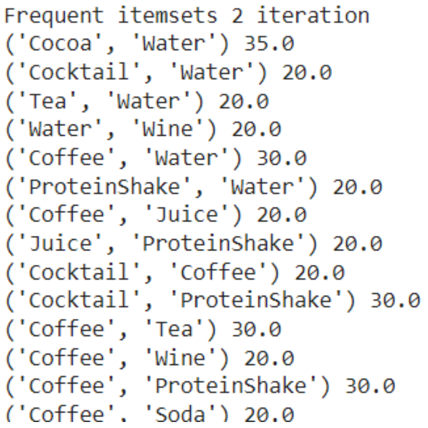


```
Please select a dataset:
1. Juice Bar
2. Burlington
3. Costco
4. Walmart
5. ShopRite
Enter the number corresponding to your choice: 1
Enter minimum support as %: 20
Enter minimum confidence as %: 40
You selected: juicebar.csv
```

Brute Force method is called and computation takes place to return the frequent itemsets and the association rules.

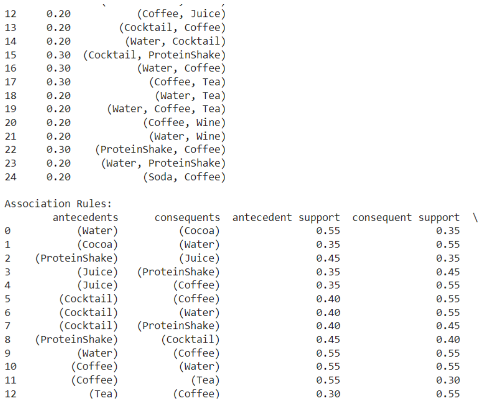


```
Association Rule for itemset - ('Cocoa', 'Water')
('Cocoa',) => ('Water',) 100.0 Rule Selected
('Water',) => ('Cocoa',) 63.64 Rule Selected

Association Rule for itemset - ('Cocktail', 'Water')
('Cocktail',) => ('Water',) 50.0 Rule Selected
('Water',) => ('Cocktail',) 36.36 Rule Rejected

Association Rule for itemset - ('Tea', 'Water')
('Tea',) => ('Water',) 66.67 Rule Selected
('Water',) => ('Tea',) 36.36 Rule Rejected

Association Rule for itemset - ('Water', 'Wine')
('Water',) => ('Wine',) 36.36 Rule Rejected
('Wine',) => ('Water',) 80.0 Rule Selected

Association Rule for itemset - ('Coffee', 'Water')
('Coffee',) => ('Water',) 54.55 Rule Selected
('Water',) => ('Coffee',) 54.55 Rule Selected
```

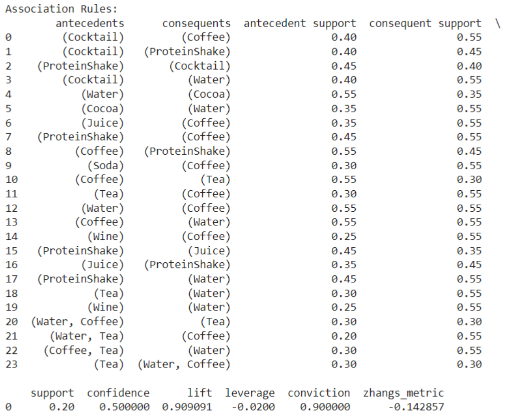


```
Frequent itemsets 2 iteration
('Cocoa', 'Water') 35.0
('Cocktail', 'Water') 20.0
('Tea', 'Water') 20.0
('Water', 'Wine') 20.0
('Coffee', 'Water') 30.0
('ProteinShake', 'Water') 20.0
('Coffee', 'Juice') 20.0
('Juice', 'ProteinShake') 20.0
('Cocktail', 'Coffee') 20.0
('Cocktail', 'ProteinShake') 30.0
('Coffee', 'Tea') 30.0
('Coffee', 'Wine') 20.0
('Coffee', 'ProteinShake') 30.0
('Coffee', 'Soda') 20.0
```

Apriori Algorithm and FP growth algorithm is called and its respective outputs are generated.

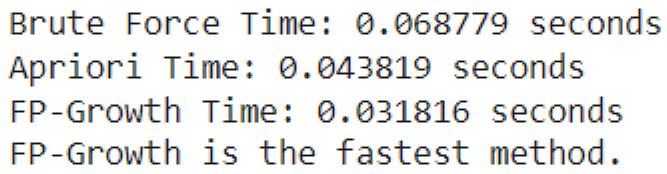


```
12     0.20          (Coffee, Juice)
13     0.20        (Cocktail, Coffee)
14     0.20         (Water, Cocktail)
15     0.30  (Cocktail, ProteinShake)
16     0.30           (Water, Coffee)
17     0.30             (Coffee, Tea)
18     0.20              (Water, Tea)
19     0.20      (Water, Coffee, Tea)
20     0.20            (Coffee, Wine)
21     0.20             (Water, Wine)
22     0.30    (ProteinShake, Coffee)
23     0.20     (Water, ProteinShake)
24     0.20            (Soda, Coffee)

Association Rules:
       antecedents      consequents  antecedent support  consequent support  \
0          (Water)          (Cocoa)                0.55                0.35
1          (Cocoa)          (Water)                0.35                0.55
2   (ProteinShake)          (Juice)                0.45                0.35
3          (Juice)   (ProteinShake)                0.35                0.45
4          (Juice)         (Coffee)                0.35                0.55
5       (Cocktail)         (Coffee)                0.40                0.55
6       (Cocktail)          (Water)                0.40                0.55
7       (Cocktail)   (ProteinShake)                0.40                0.45
8   (ProteinShake)       (Cocktail)                0.45                0.40
9          (Water)         (Coffee)                0.55                0.55
10        (Coffee)          (Water)                0.55                0.55
11        (Coffee)            (Tea)                0.55                0.30
12           (Tea)         (Coffee)                0.30                0.55
```



```
Association Rules:
       antecedents      consequents  antecedent support  consequent support  \
0       (Cocktail)         (Coffee)                0.40                0.55
1       (Cocktail)   (ProteinShake)                0.40                0.45
2   (ProteinShake)       (Cocktail)                0.45                0.40
3       (Cocktail)          (Water)                0.40                0.55
4          (Water)          (Cocoa)                0.55                0.35
5          (Cocoa)          (Water)                0.35                0.55
6          (Juice)         (Coffee)                0.35                0.55
7   (ProteinShake)         (Coffee)                0.45                0.55
8         (Coffee)   (ProteinShake)                0.55                0.45
9           (Soda)         (Coffee)                0.30                0.55
10        (Coffee)            (Tea)                0.55                0.30
11           (Tea)         (Coffee)                0.30                0.55
12         (Water)         (Coffee)                0.55                0.55
13        (Coffee)          (Water)                0.55                0.55
14          (Wine)         (Coffee)                0.25                0.55
15  (ProteinShake)          (Juice)                0.45                0.35
16         (Juice)   (ProteinShake)                0.35                0.45
17  (ProteinShake)          (Water)                0.45                0.55
18           (Tea)          (Water)                0.30                0.55
19          (Wine)          (Water)                0.25                0.55
20 (Water, Coffee)            (Tea)                0.30                0.30
21    (Water, Tea)         (Coffee)                0.20                0.55
22   (Coffee, Tea)          (Water)                0.30                0.55
23           (Tea)  (Water, Coffee)                0.30                0.30

   support  confidence      lift  leverage  conviction  zhangs_metric
0     0.20    0.500000  0.909091   -0.0200    0.900000      -0.142857
```

Lastly time complexities of the three algorithms are compared and analyzed:



```
Brute Force Time: 0.068779 seconds
Apriori Time: 0.043819 seconds
FP-Growth Time: 0.031816 seconds
FP-Growth is the fastest method.
```

Extras:

Some of the dataset was downloaded from an online source and some generated through chatgpt.

The source code (.py file) and data sets (.csv files) will be attached to the zip file. Link to Git.
Repository https://github.com/ana-bharadwaj/dmsproject

Things to do to run the file:
In order to run the file please install `mlxtend.To do so please perform`
`pip install mlxtend.`
`Detailed readme file attached to github. Requirements.txt too.`