CS 634-Data Mining Midterm Project

10.19.2025

—

Chike Egbuchulam
coe5@njit.edu
CS634 - Data Mining
Instructor: Dr.Yasser Abduallah

Github link: https://github.com/naijatek21/Egbuchulam_Chike_midtermproject.git

Introduction

The goal of this project is to implement a brute force algorithm to determine the frequent itemsets and association rules from a transaction database given a minimum support and confidence values. We then took the results of this algorithm, ran it on 5 datatables and compared it to two popular association rule mining algorithms: Apriori and FP-Growth.

Setup

In order to run this project you must have a machine that is able to run python3. After downloading the and unzipping the entire code repository, in the terminal your local or virtual machine environments you must then install the following python libraries the commands below work an a WSL/Ubuntu systems.

```
Pip install pandas

Pip install mlxtend

Pip install os
```

To run the program (while in Egubuchulam_Chike_midtermproject directory) :

```
    python midterm.py
```

Dataset Creation

I. Data Source

Bakery Sales Dataset: https://www.kaggle.com/datasets/akashdeepkuila/bakery/data

For this project I sourced the data from a Kaggle dataset named *Bakery Sales Dataset* which comprises the transaction details of baked good orders of a bakery in Edinburgh,Scotland . Due to the diverse collection of items purchased around the same time, the dataset proved to be a reasonably good source of data to model association rules.

II. Reformatting

Since the dataset in its original form contained only one item purchased per row and no clear segmentation of it, the dataset required some manipulation to be suitable for observations. To do this, I downloaded the dataset as a csv file and from there used a personal mySQL server (or you can use an online tools such as csvFiddle) to run a series of a custom SQL script that used queries to create the 5 datables needed for our goals . The script `bakerysetup.sql` does the following

1. Groups each row by transaction number so we are able to see all the items that are purchased together
2. Constrict the the resulting rows to those that contain the 15 most popular items purchased so we are better able to find rules
3. Divided the resulting set int 5 tables of of 40 transactions to simulate the sales of 5 bakeries in the *Midterm Baking District*
    - Alice's Bakey
    - Bob's Breakfast
    - Charlie's Cafe
    - Desayuno de David
    - Eddie's Eats

After running the script the resulting tables were saved as csv files and exported. Notre: The full script is in the project folder and is not needed for program execution as the the tables are already formed.

III. Other Considerations

In the cases where a customer would order multiples of the same thing (two coffees, two bagels, etc.) the aforementioned operations would simply put the duplicate item twice on the same line. Since we are only considering single item associations, when such instances occur the main program described in the next section will disregard the duplicates and treat the time as a single entity.

Program Description

midterm.py

The main midterm.py program when run will introduce the user to a menu screen where they will have a choice of which bakery they want to investigate. After selecting the bakery they are prompted to enter a minimum support and confidence level as a number between 0-100 and from there the program reads the csv file for their chosen bakery, ignores the transaction number and removes duplicates in every row. Then loads that data as a list of sets, a list of all unique items that appear in the list, and the the prompted values and passes them as parameters to our brute force association rule finder bfassoc().

bfassoc.py

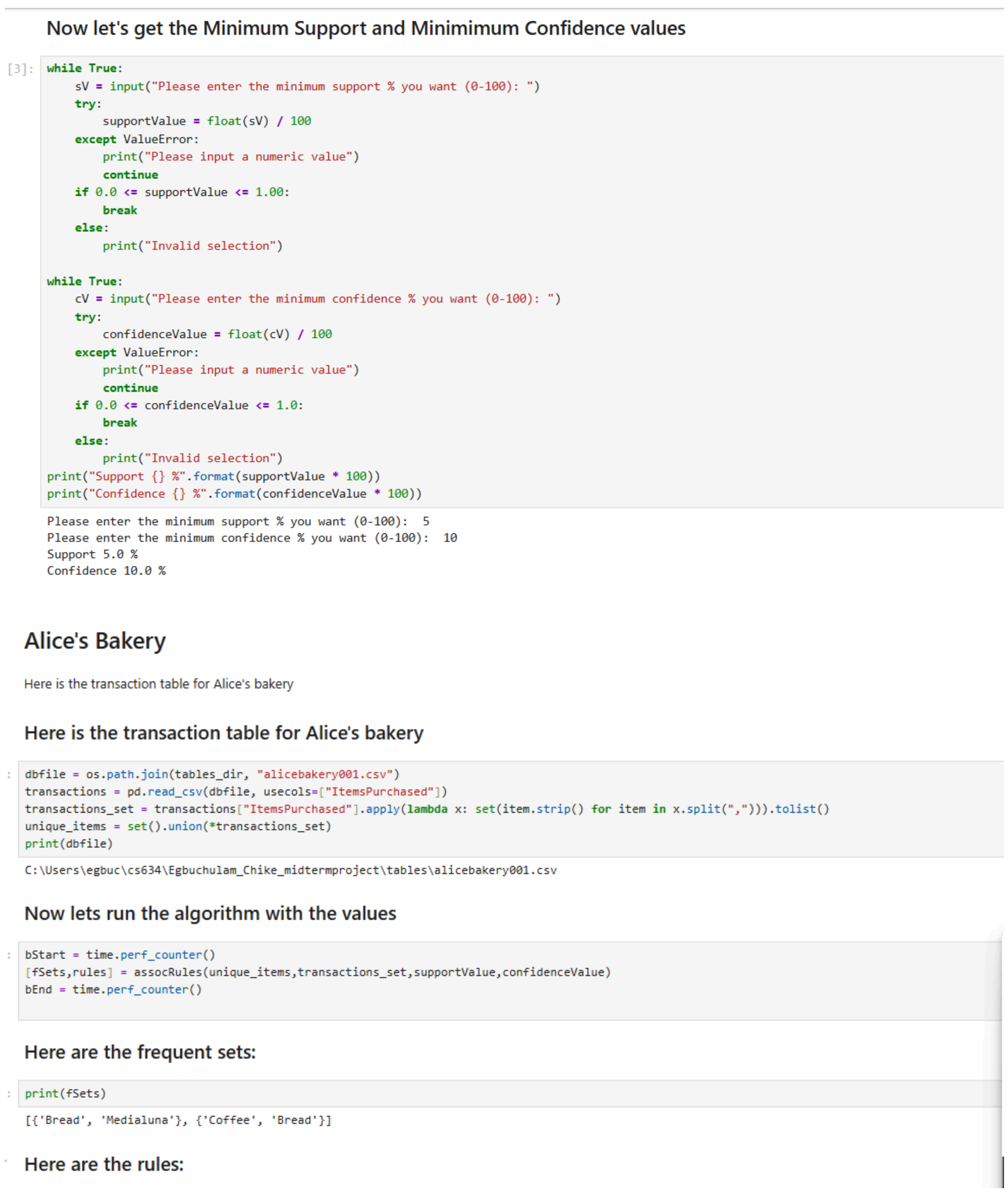
The bfassoc.py program takes in the user input from the main function and outputs both the frequent item sets found after running the algorithm as well as a tuple list that is used to describe each association rule along with its confidence.   The algorithm starts by taking the list of all unique items and a puts each item in its own set and traverses the data table keeping track of every row it appears in. The set is then put into a dedicated list if it reaches the minimum support threshold and the cycle loops with all possible subsets of size k=2,3 ,4 … until no sets or items have a minimum support threshold. Form there that list is traversed to see what items reach the minimum confidence level and those results are added to a tuple list that is outputted to the console

Special considerations

In the cases where support = 0% in order to prevent the program from crashing, the program simply outputs the transaction list as all possible two item combinations would be considered rules.

Screenshots

Alice ‘s Bakery with support level 5% and confidence level 10%



Now let's get the Minimum Support and Minimimum Confidence values

```
[3]: while True:
         sV = input("Please enter the minimum support % you want (0-100): ")
         try:
             supportValue = float(sV) / 100
         except ValueError:
             print("Please input a numeric value")
             continue
         if 0.0 <= supportValue <= 1.00:
             break
         else:
             print("Invalid selection")

     while True:
         cV = input("Please enter the minimum confidence % you want (0-100): ")
         try:
             confidenceValue = float(cV) / 100
         except ValueError:
             print("Please input a numeric value")
             continue
         if 0.0 <= confidenceValue <= 1.0:
             break
         else:
             print("Invalid selection")
     print("Support {} %".format(supportValue * 100))
     print("Confidence {} %".format(confidenceValue * 100))
```

```
Please enter the minimum support % you want (0-100):  5
Please enter the minimum confidence % you want (0-100):  10
Support 5.0 %
Confidence 10.0 %
```

Alice's Bakery

Here is the transaction table for Alice's bakery

Here is the transaction table for Alice's bakery

```
dbfile = os.path.join(tables_dir, "alicebakery001.csv")
transactions = pd.read_csv(dbfile, usecols=["ItemsPurchased"])
transactions_set = transactions["ItemsPurchased"].apply(lambda x: set(item.strip() for item in x.split(","))).tolist()
unique_items = set().union(*transactions_set)
print(dbfile)
```

```
C:\Users\egbuc\cs634\Egbuchulam_Chike_midtermproject\tables\alicebakery001.csv
```

Now lets run the algorithm with the values

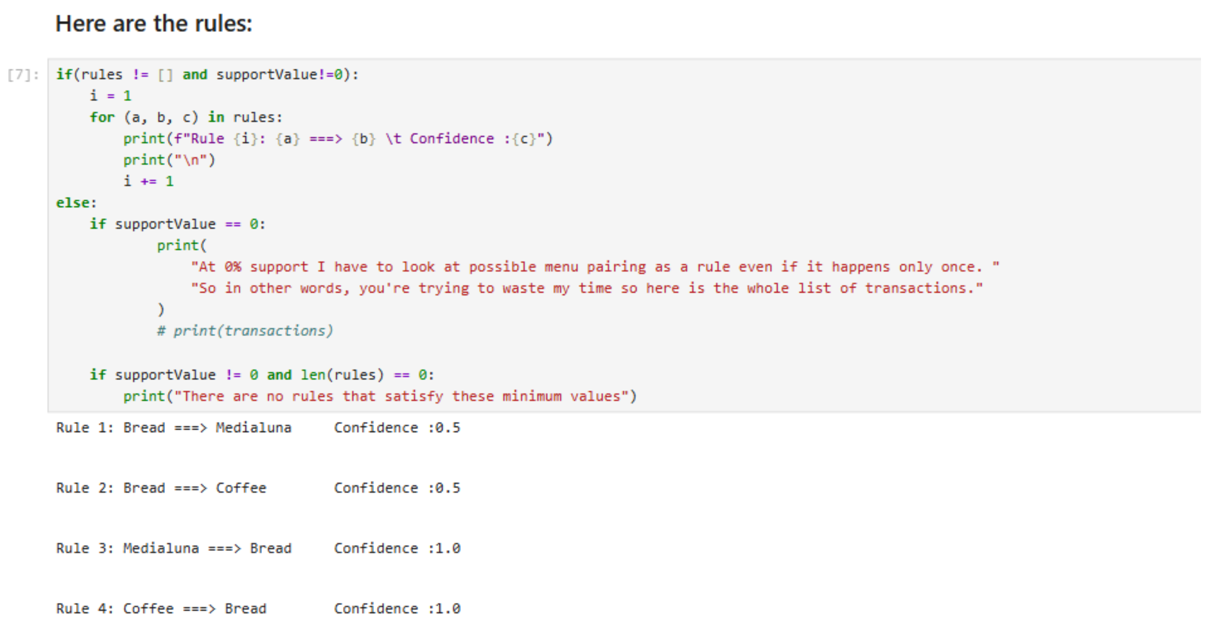
```
bStart = time.perf_counter()
[fSets,rules] = assocRules(unique_items,transactions_set,supportValue,confidenceValue)
bEnd = time.perf_counter()
```

Here are the frequent sets:

```
print(fSets)
```

```
[{'Bread', 'Medialuna'}, {'Coffee', 'Bread'}]
```

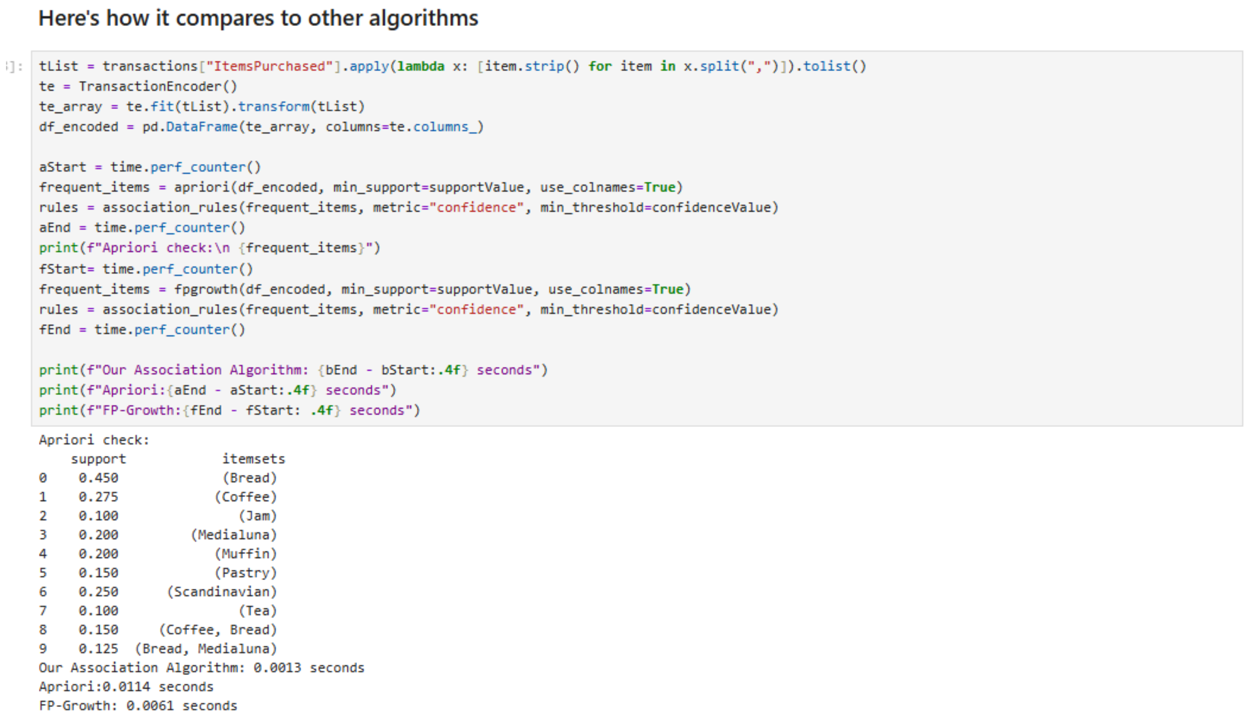
Here are the rules:



Here are the rules:

```
if(rules != [] and supportValue!=0):
    i = 1
    for (a, b, c) in rules:
        print(f"Rule {i}: {a} ===> {b} \t Confidence :{c}")
        print("\n")
        i += 1
else:
    if supportValue == 0:
            print(
                "At 0% support I have to look at possible menu pairing as a rule even if it happens only once. "
                "So in other words, you're trying to waste my time so here is the whole list of transactions."
            )
            # print(transactions)

    if supportValue != 0 and len(rules) == 0:
        print("There are no rules that satisfy these minimum values")
```

```
Rule 1: Bread ===> Medialuna     Confidence :0.5


Rule 2: Bread ===> Coffee        Confidence :0.5


Rule 3: Medialuna ===> Bread     Confidence :1.0


Rule 4: Coffee ===> Bread        Confidence :1.0
```



Here's how it compares to other algorithms

```
tList = transactions["ItemsPurchased"].apply(lambda x: [item.strip() for item in x.split(",")]).tolist()
te = TransactionEncoder()
te_array = te.fit(tList).transform(tList)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

aStart = time.perf_counter()
frequent_items = apriori(df_encoded, min_support=supportValue, use_colnames=True)
rules = association_rules(frequent_items, metric="confidence", min_threshold=confidenceValue)
aEnd = time.perf_counter()
print(f"Apriori check:\n {frequent_items}")
fStart= time.perf_counter()
frequent_items = fpgrowth(df_encoded, min_support=supportValue, use_colnames=True)
rules = association_rules(frequent_items, metric="confidence", min_threshold=confidenceValue)
fEnd = time.perf_counter()

print(f"Our Association Algorithm: {bEnd - bStart:.4f} seconds")
print(f"Apriori:{aEnd - aStart:.4f} seconds")
print(f"FP-Growth:{fEnd - fStart: .4f} seconds")
```

```
Apriori check:
    support            itemsets
0     0.450             (Bread)
1     0.275            (Coffee)
2     0.100               (Jam)
3     0.200         (Medialuna)
4     0.200            (Muffin)
5     0.150            (Pastry)
6     0.250      (Scandinavian)
7     0.100               (Tea)
8     0.150     (Coffee, Bread)
9     0.125  (Bread, Medialuna)
Our Association Algorithm: 0.0013 seconds
Apriori:0.0114 seconds
FP-Growth: 0.0061 seconds
```

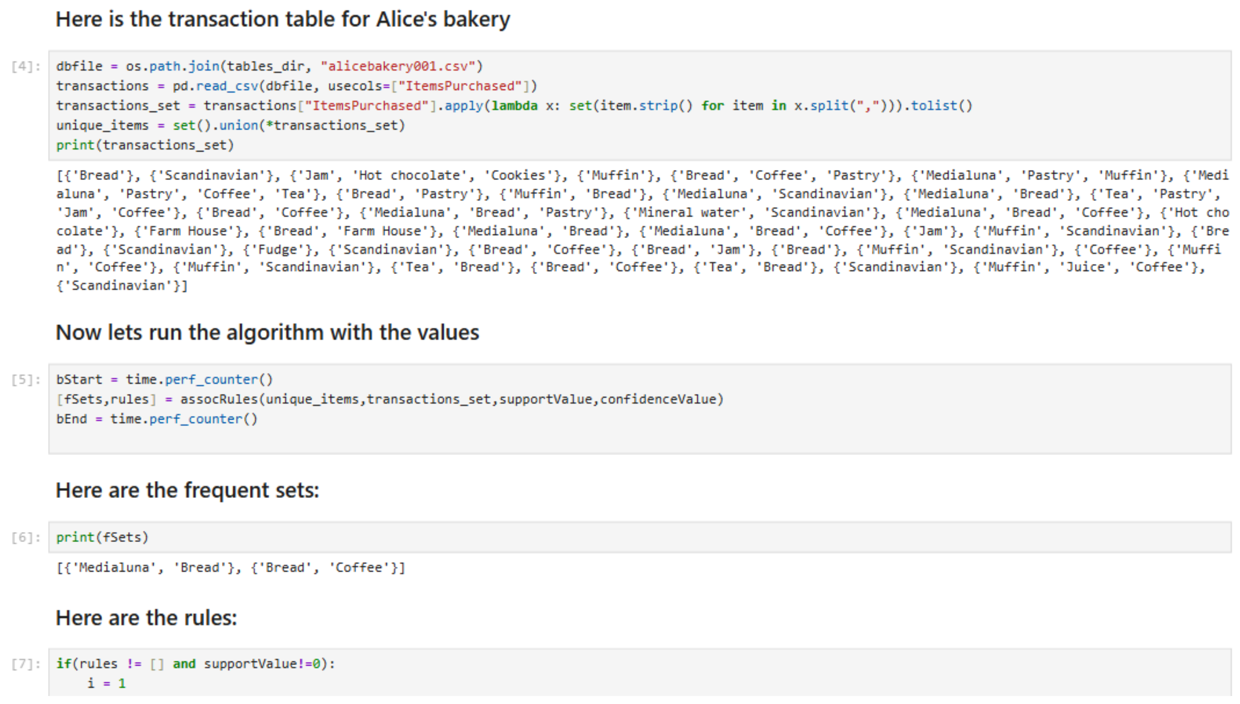
Alice’s Bakery Support 10% Confidence 20%

```
            break
    else:
        print("Invalid selection")
print("Support {} %".format(supportValue * 100))
print("Confidence {} %".format(confidenceValue * 100))
```

```
Please enter the minimum support % you want (0-100):  10
Please enter the minimum confidence % you want (0-100):  20
Support 10.0 %
Confidence 20.0 %
```

Now we'll run theses values for each database with our algorithm while also running the aprioiri and FP growth algorithms for comparison

Alice's Bakery



Here is the transaction table for Alice's bakery

```
[4]: dbfile = os.path.join(tables_dir, "alicebakery001.csv")
     transactions = pd.read_csv(dbfile, usecols=["ItemsPurchased"])
     transactions_set = transactions["ItemsPurchased"].apply(lambda x: set(item.strip() for item in x.split(","))).tolist()
     unique_items = set().union(*transactions_set)
     print(transactions_set)
```

```
[{'Bread'}, {'Scandinavian'}, {'Jam', 'Hot chocolate', 'Cookies'}, {'Muffin'}, {'Bread', 'Coffee', 'Pastry'}, {'Medialuna', 'Pastry', 'Muffin'}, {'Medi
aluna', 'Pastry', 'Coffee', 'Tea'}, {'Bread', 'Pastry'}, {'Muffin', 'Bread'}, {'Medialuna', 'Scandinavian'}, {'Medialuna', 'Bread'}, {'Tea', 'Pastry',
'Jam', 'Coffee'}, {'Bread', 'Coffee'}, {'Medialuna', 'Bread', 'Pastry'}, {'Mineral water', 'Scandinavian'}, {'Medialuna', 'Bread', 'Coffee'}, {'Hot cho
colate'}, {'Farm House'}, {'Bread', 'Farm House'}, {'Medialuna', 'Bread'}, {'Medialuna', 'Bread', 'Coffee'}, {'Jam'}, {'Muffin', 'Scandinavian'}, {'Bre
ad'}, {'Scandinavian'}, {'Fudge'}, {'Scandinavian'}, {'Bread', 'Coffee'}, {'Bread', 'Jam'}, {'Bread'}, {'Muffin', 'Scandinavian'}, {'Coffee'}, {'Muffi
n', 'Coffee'}, {'Muffin', 'Scandinavian'}, {'Tea', 'Bread'}, {'Bread', 'Coffee'}, {'Tea', 'Bread'}, {'Scandinavian'}, {'Muffin', 'Juice', 'Coffee'},
{'Scandinavian'}]
```

Now lets run the algorithm with the values

```
[5]: bStart = time.perf_counter()
     [fSets,rules] = assocRules(unique_items,transactions_set,supportValue,confidenceValue)
     bEnd = time.perf_counter()
```

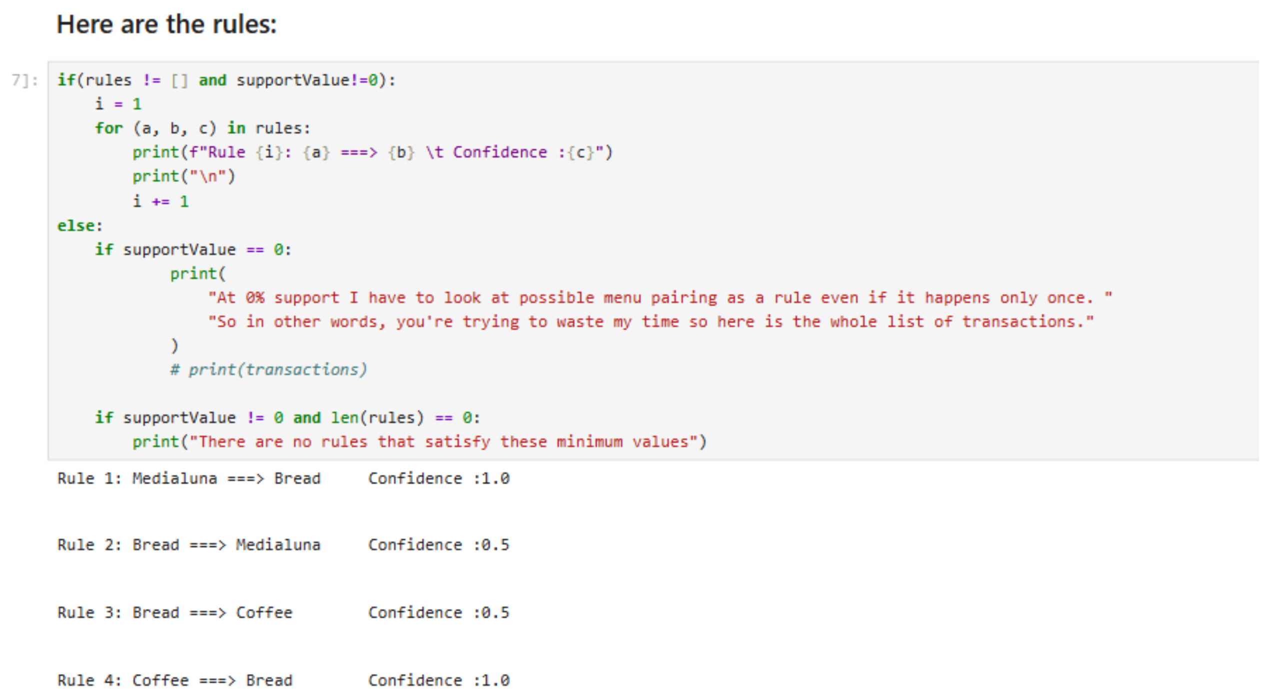
Here are the frequent sets:

```
[6]: print(fSets)
```

```
[{'Medialuna', 'Bread'}, {'Bread', 'Coffee'}]
```

Here are the rules:

```
[7]: if(rules != [] and supportValue!=0):
         i = 1
```



Here are the rules:

```
7]: if(rules != [] and supportValue!=0):
        i = 1
        for (a, b, c) in rules:
            print(f"Rule {i}: {a} ===> {b} \t Confidence :{c}")
            print("\n")
            i += 1
    else:
        if supportValue == 0:
                print(
                    "At 0% support I have to look at possible menu pairing as a rule even if it happens only once. "
                    "So in other words, you're trying to waste my time so here is the whole list of transactions."
                )
                # print(transactions)

        if supportValue != 0 and len(rules) == 0:
            print("There are no rules that satisfy these minimum values")
```

```
Rule 1: Medialuna ===> Bread      Confidence :1.0


Rule 2: Bread ===> Medialuna      Confidence :0.5


Rule 3: Bread ===> Coffee         Confidence :0.5


Rule 4: Coffee ===> Bread         Confidence :1.0
```

Here's how it compares to other algorithms

```python
tList = transactions["ItemsPurchased"].apply(lambda x: [item.strip() for item in x.split(",")]).tolist()
te = TransactionEncoder()
te_array = te.fit(tList).transform(tList)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

aStart = time.perf_counter()
frequent_items = apriori(df_encoded, min_support=supportValue, use_colnames=True)
rules = association_rules(frequent_items, metric="confidence", min_threshold=confidenceValue)
aEnd = time.perf_counter()
print(f"Apriori check:\n {frequent_items}")
fStart= time.perf_counter()
frequent_items = fpgrowth(df_encoded, min_support=supportValue, use_colnames=True)
rules = association_rules(frequent_items, metric="confidence", min_threshold=confidenceValue)
fEnd = time.perf_counter()

print(f"Our Association Algorithm: {bEnd - bStart:.4f} seconds")
print(f"Apriori:{aEnd - aStart:.4f} seconds")
print(f"FP-Growth:{fEnd - fStart: .4f} seconds")
```

```
Apriori check:
    support            itemsets
0    0.450              (Bread)
1    0.275             (Coffee)
2    0.100                (Jam)
3    0.200          (Medialuna)
4    0.200             (Muffin)
5    0.150             (Pastry)
6    0.250        (Scandinavian)
7    0.100                (Tea)
8    0.150      (Bread, Coffee)
9    0.125   (Medialuna, Bread)
Our Association Algorithm: 0.0033 seconds
Apriori:0.0204 seconds
FP-Growth: 0.0161 seconds
```