NAME: Amish Faldu
NJIT UCID: af557
Email Address: af557@njit.edu
03/03/2024
Professor: Yasser Abduallah
CS 634 104 Data Mining

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm Implementation in Retail Data Mining

Abstract:
In this project, I delve into the Apriori Algorithm, a fundamental technique in data mining, to discover associations within retail transactions. Through implementing the algorithm and applying diverse data mining concepts, principles, and methodologies, I evaluate its efficiency and correctness. I've created algorithm for mining valuable insights from transaction data.

Introduction:
Data mining helps find hidden patterns in big datasets. Our project looks at the Apriori and FP-Growth Algorithm, a classic method for finding associations, especially in retail. We'll explain the basic ideas behind data mining used in our project.

The Apriori Algorithm is about finding frequent patterns and creating association rules. To do this, we first find out which items are most common in the list of transactions. Then, depending on the user's chosen support level, we calculate the support for each item. Items that don't meet the support level are removed. This algorithm uses a brute force method to find frequent item sets using minimum support threshold and generating association rules by filtering out those that don't meet a minimum confidence threshold.

In this implementation, I've applied the brute force algorithm to a custom dataset associated with a retail store, allowing us to find frequent item-sets and association rules. Key steps in this process included:

- Loading the dataset and item-sets from CSV files.
- Pre-processing the dataset to ensure item order.
- Prompting user input for minimum support and confidence thresholds.
- Iteratively generating candidate item-sets and updating frequent item-sets using the brute force algorithm.

Core Concepts and Principles:

Frequent Itemset Discovery:
The Apriori Algorithm is used to discover frequent item-sets, i.e., sets of items that frequently occur in transactions. These item-sets provide insights into customer purchase behaviour and item preferences.

Support and Confidence:
Two key metrics in frequent pattern algorithms are support and confidence. Support measures how frequently an item or itemset occurs, while confidence assesses the likelihood of items being purchased together.

Association Rules:
By determining strong association rules, we identify which items are commonly purchased together. These rules helps in optimizing sales strategies, such as product recommendations.

Project Workflow:

Our project follows a structured workflow involving various stages and the application of the Apriori

Data Loading and Pre-processing:
We begin by loading transaction data from a retail store dataset. Each transaction consists of a list of items bought by a customer. To ensure data accuracy, we pre-process the dataset, i.e. sorting them based on a alphabetic order.

Determination of Minimum Support and Confidence:
We collect the user's input for minimum support and confidence levels to filter out less frequent patterns.

Brute Force Algorithm Implementation

Iteration Through Candidate Item-sets:
The iterative application of the Apriori Algorithm involves generating candidate item-sets of increasing sizes. We start with single items (itemset size K = 1) and proceed to K = 2, K = 3, and so on until K + 1 item-set is not possible. This iterative process involves a "brute force" method of generating all possible itemset combinations.

Support Count Calculation:
For each candidate itemset, we calculate its support by counting how many transactions contain the itemset. Item-sets that meet the minimum support threshold are retained, while others are discarded.

Confidence Calculation:
We evaluate the confidence of association rules, by counting transactions containing items in itemset dividing by transactions containing items in left-hand of the association rule.

Association Rule Generation:
Association rules that satisfy the minimum confidence are extracted. These rules reveal valuable insights into which items are often purchased together.

Results and Evaluation:
The project's effectiveness and efficiency are evaluated based on performance measures such as support, confidence, and the resulting association rules. We also compare brute force algorithm with the Apriori library to assess its reliability.

Conclusion:

In conclusion, my project demonstrates the application of data mining concepts, principles, and methods. I've successfully implemented the Apriori Algorithm to extract meaningful association rules from retail transaction data. The iterative, "brute force" approach, custom algorithm design, and adherence to user-defined parameters exemplify the power of data mining in revealing valuable patterns for decision-making in the retail industry.

Directory Structure:

-------- data/ => This contains all the data files needed to run the code
-------- docs/ => This contains images for documentation
-------- .gitignore => Ignore files / folders to include in git history
-------- bruteforce.py => Implementation of the brute force algorithm
-------- library.py => Contains function that invoke library frequent pattern functions
-------- main.ipynb => This is the main Jupyter notebook that contains the code to run brute force and library's algorithm
-------- Midterm-Project-Report.pdf => Obviously, project report
-------- README.md => This contains all the instructions to setup the project locally
-------- requirements.txt => This contains all the packages to run the code

---

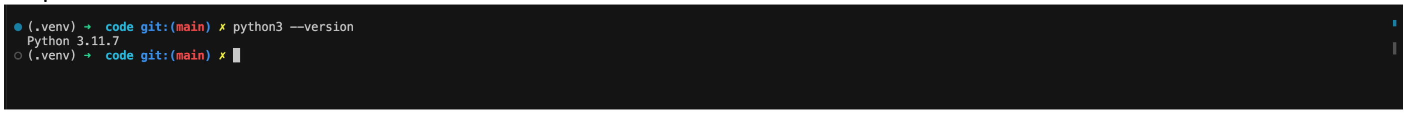
Steps to run the program:

1. Make sure that you've python installed on your machine. To download and install Python, go to https://www.python.org/. To check if you've python installed, run `python3 --version`.
   NOTE - I have used python version 3.11.7 to run the program
2. Create a virtual environment using command `python3 -m venv .venv`.
3. Activate the virtual environment using command `source .venv/bin/activate`.
4. Install all the packages in **requirements.txt** using `pip install -r requirements.txt`.
5. Select existing `.venv` python environment as the kernel for the `main.ipynb` Jupyter notebook.
6. Run cells in `main.ipynb` Jupyter notebook.

Screenshots:

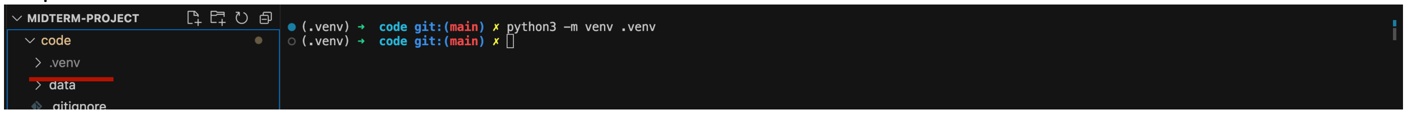
Pre-requisite steps

Step - 1



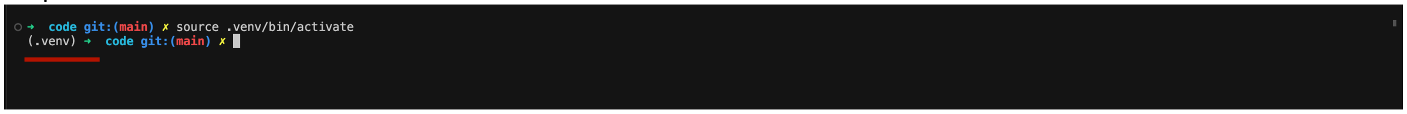
```
(.venv) ➜  code git:(main) ✗ python3 --version
Python 3.11.7
(.venv) ➜  code git:(main) ✗
```

Step - 2



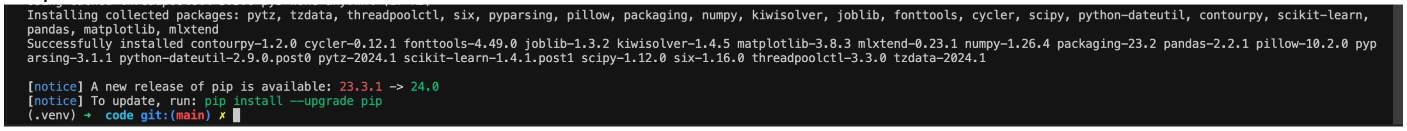
```
(.venv) ➜  code git:(main) ✗ python3 -m venv .venv
(.venv) ➜  code git:(main) ✗
```

Step - 3



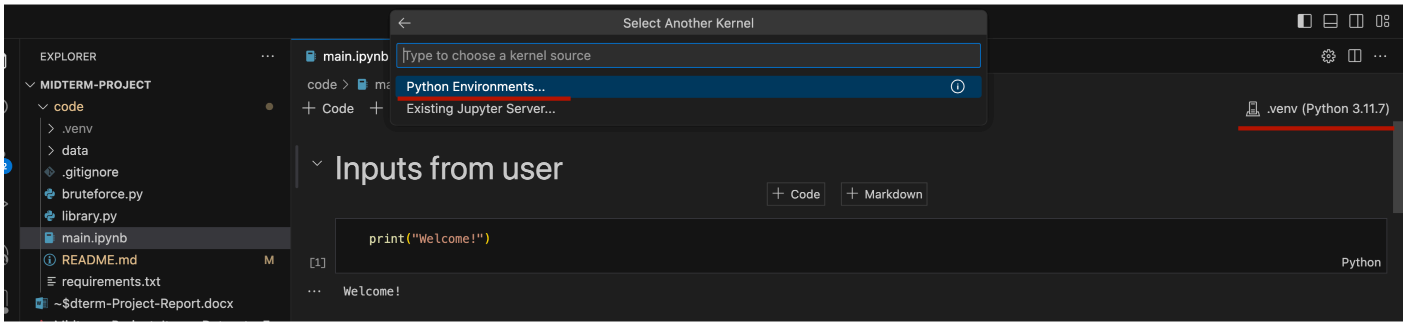
```
➜  code git:(main) ✗ source .venv/bin/activate
(.venv) ➜  code git:(main) ✗
```

Step – 4



```
Installing collected packages: pytz, tzdata, threadpoolctl, six, pyparsing, pillow, packaging, numpy, kiwisolver, joblib, fonttools, cycler, scipy, python-dateutil, contourpy, scikit-learn,
pandas, matplotlib, mlxtend
Successfully installed contourpy-1.2.0 cycler-0.12.1 fonttools-4.49.0 joblib-1.3.2 kiwisolver-1.4.5 matplotlib-3.8.3 mlxtend-0.23.1 numpy-1.26.4 packaging-23.2 pandas-2.2.1 pillow-10.2.0 pyp
arsing-3.1.1 python-dateutil-2.9.0.post0 pytz-2024.1 scikit-learn-1.4.1.post1 scipy-1.12.0 six-1.16.0 threadpoolctl-3.3.0 tzdata-2024.1

[notice] A new release of pip is available: 23.3.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
(.venv) ➜  code git:(main) ✗
```

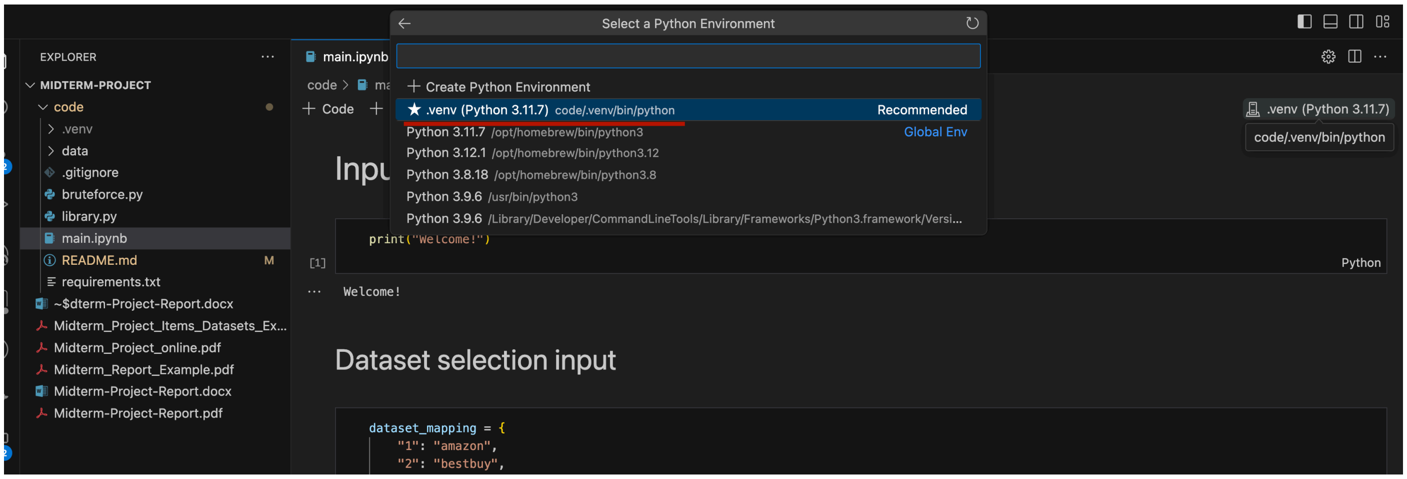
Step – 5
Click on the top-right of the notebook where it would say “Select Kernel”. By clicking it will prompt you to select a env.



```
print("Welcome!")
```

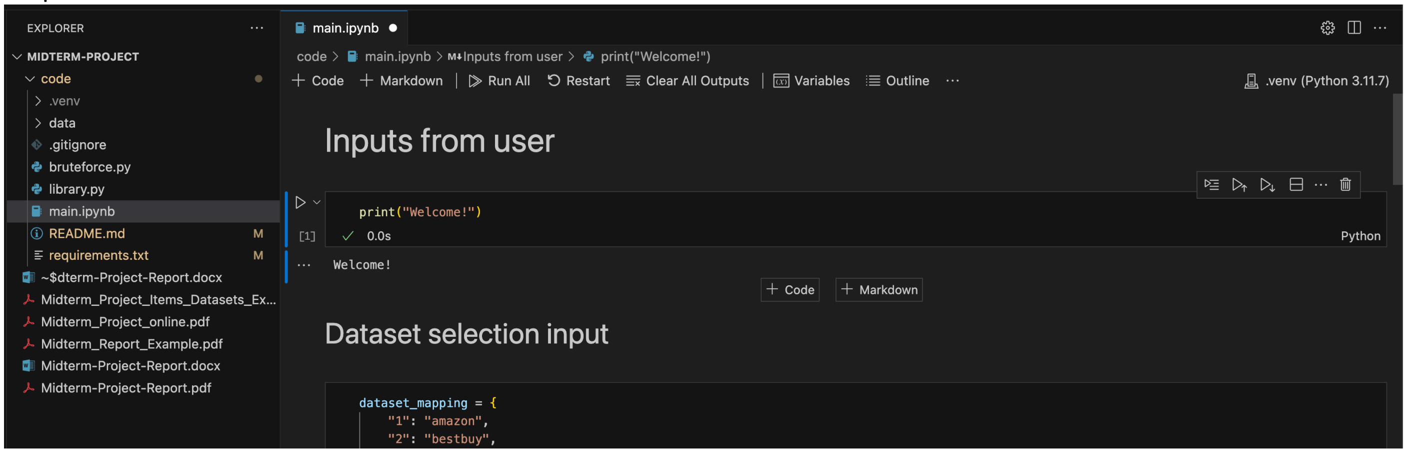
```
Welcome!
```

Now select the virtual we created in step – 2 as the environment. After that, it would show you the selected kernel in top-right section.


Select a Python Environment
Create Python Environment
★ .venv (Python 3.11.7) code/.venv/bin/python Recommended
Python 3.11.7 /opt/homebrew/bin/python3 Global Env
Python 3.12.1 /opt/homebrew/bin/python3.12
Python 3.8.18 /opt/homebrew/bin/python3.8
Python 3.9.6 /usr/bin/python3
Python 3.9.6 /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versi...
.venv (Python 3.11.7)
code/.venv/bin/python
Dataset selection input

```
dataset_mapping = {
    "1": "amazon",
    "2": "bestbuy",
```

Step – 6

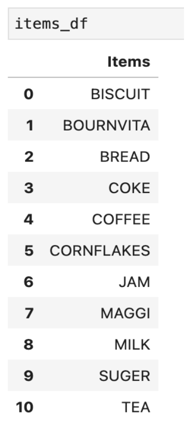


```
print("Welcome!")
```

```
Welcome!
```

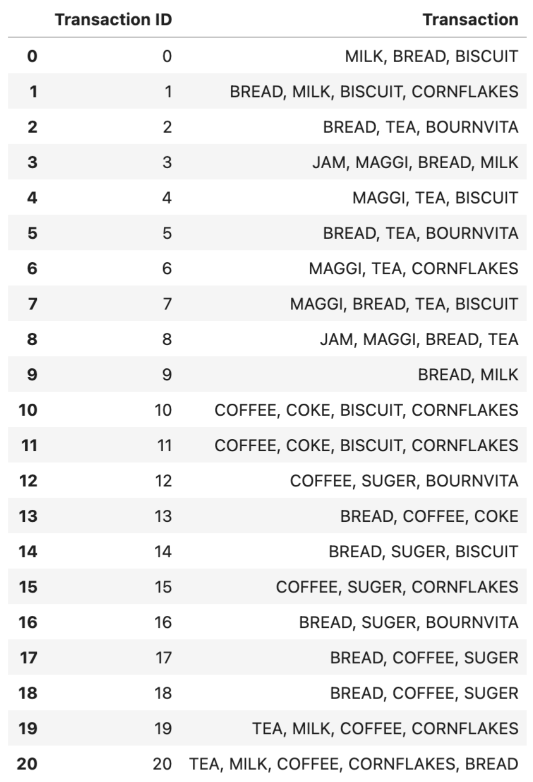
```
dataset_mapping = {
    "1": "amazon",
    "2": "bestbuy",
```

Here are what the csv files (This program takes in two separate csv files: Items & Transactions).
Figure 1 : Custom Items CSV file
Figure 2 : Custom Transactions CSV file



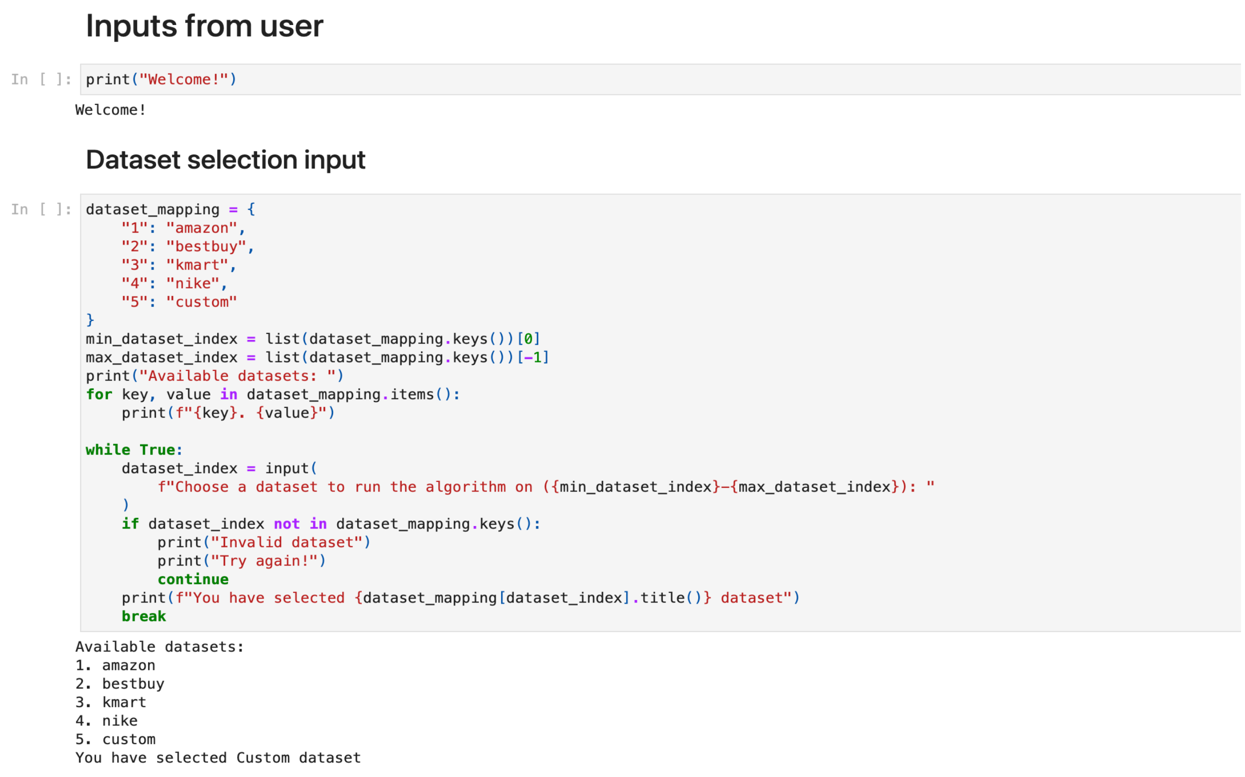
```
items_df
```

	Items
0	BISCUIT
1	BOURNVITA
2	BREAD
3	COKE
4	COFFEE
5	CORNFLAKES
6	JAM
7	MAGGI
8	MILK
9	SUGER
10	TEA



	Transaction ID	Transaction
0	0	MILK, BREAD, BISCUIT
1	1	BREAD, MILK, BISCUIT, CORNFLAKES
2	2	BREAD, TEA, BOURNVITA
3	3	JAM, MAGGI, BREAD, MILK
4	4	MAGGI, TEA, BISCUIT
5	5	BREAD, TEA, BOURNVITA
6	6	MAGGI, TEA, CORNFLAKES
7	7	MAGGI, BREAD, TEA, BISCUIT
8	8	JAM, MAGGI, BREAD, TEA
9	9	BREAD, MILK
10	10	COFFEE, COKE, BISCUIT, CORNFLAKES
11	11	COFFEE, COKE, BISCUIT, CORNFLAKES
12	12	COFFEE, SUGER, BOURNVITA
13	13	BREAD, COFFEE, COKE
14	14	BREAD, SUGER, BISCUIT
15	15	COFFEE, SUGER, CORNFLAKES
16	16	BREAD, SUGER, BOURNVITA
17	17	BREAD, COFFEE, SUGER
18	18	BREAD, COFFEE, SUGER
19	19	TEA, MILK, COFFEE, CORNFLAKES
20	20	TEA, MILK, COFFEE, CORNFLAKES, BREAD

Below are screenshots of the running code from Jupyter notebook:
Prompts user to choose the dataset, min support, min confidence. Then it validates user input to check if it is acceptable or not.



Inputs from user

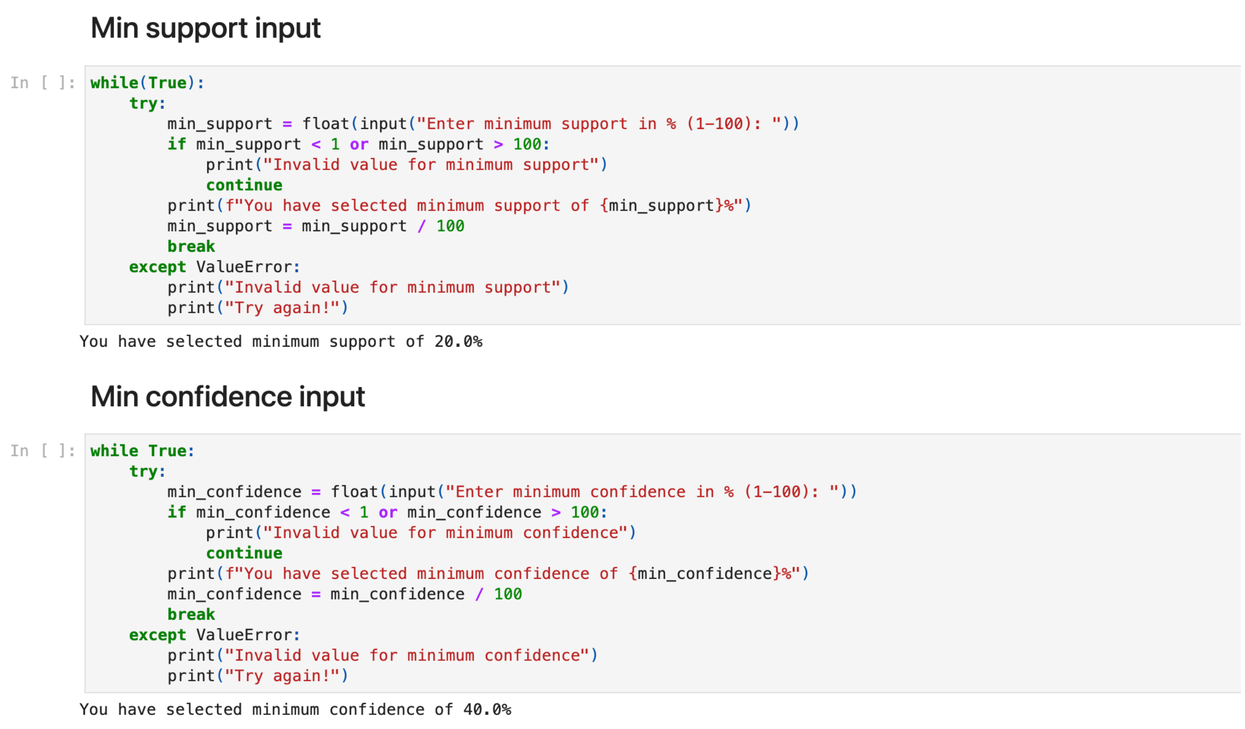
```
In [ ]: print("Welcome!")
```

```
Welcome!
```

Dataset selection input

```
In [ ]: dataset_mapping = {
            "1": "amazon",
            "2": "bestbuy",
            "3": "kmart",
            "4": "nike",
            "5": "custom"
        }
        min_dataset_index = list(dataset_mapping.keys())[0]
        max_dataset_index = list(dataset_mapping.keys())[-1]
        print("Available datasets: ")
        for key, value in dataset_mapping.items():
            print(f"{key}. {value}")

        while True:
            dataset_index = input(
                f"Choose a dataset to run the algorithm on ({min_dataset_index}-{max_dataset_index}): "
            )
            if dataset_index not in dataset_mapping.keys():
                print("Invalid dataset")
                print("Try again!")
                continue
            print(f"You have selected {dataset_mapping[dataset_index].title()} dataset")
            break
```

```
Available datasets:
1. amazon
2. bestbuy
3. kmart
4. nike
5. custom
You have selected Custom dataset
```



Min support input

```
In [ ]: while(True):
            try:
                min_support = float(input("Enter minimum support in % (1-100): "))
                if min_support < 1 or min_support > 100:
                    print("Invalid value for minimum support")
                    continue
                print(f"You have selected minimum support of {min_support}%")
                min_support = min_support / 100
                break
            except ValueError:
                print("Invalid value for minimum support")
                print("Try again!")
```

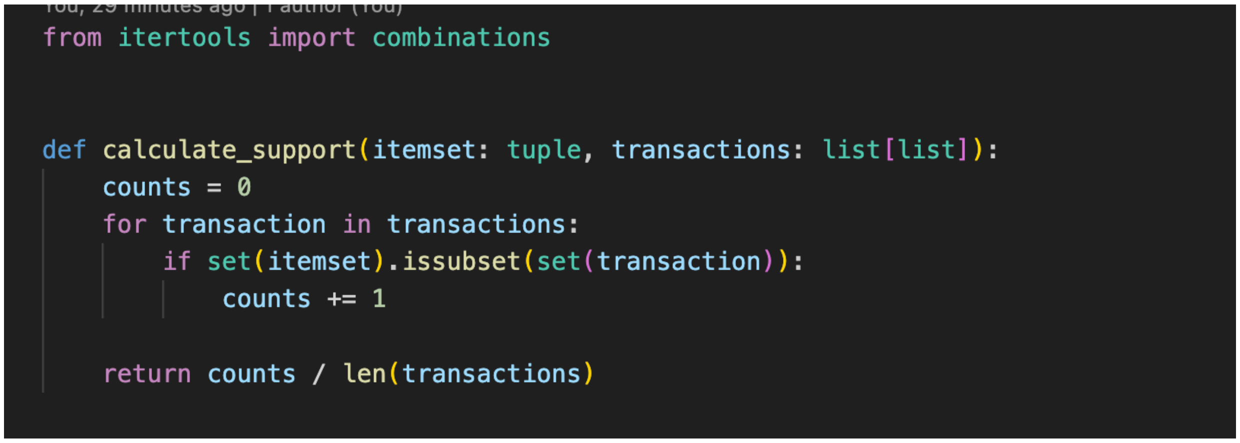
```
You have selected minimum support of 20.0%
```

Min confidence input

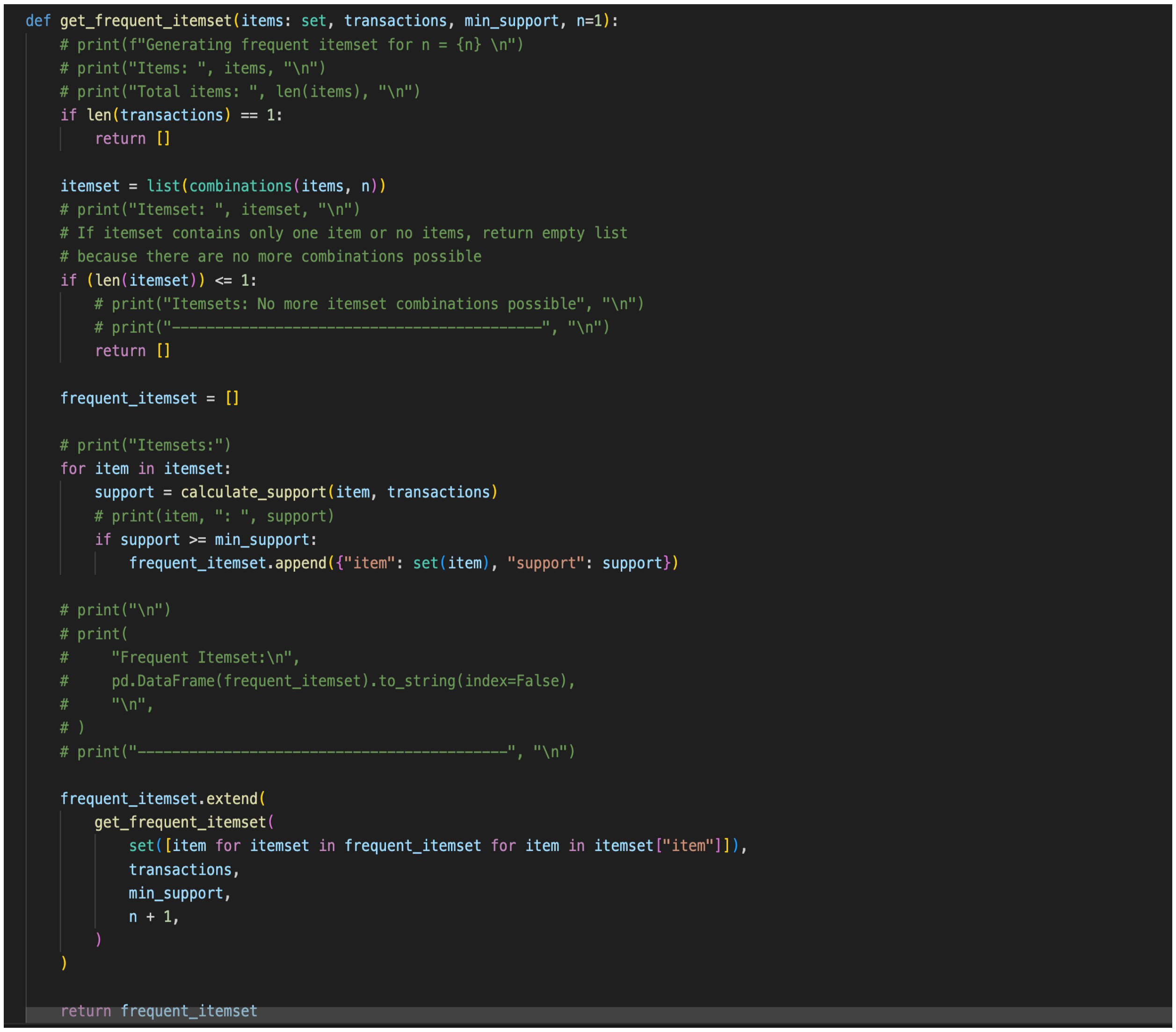
```
In [ ]: while True:
            try:
                min_confidence = float(input("Enter minimum confidence in % (1-100): "))
                if min_confidence < 1 or min_confidence > 100:
                    print("Invalid value for minimum confidence")
                    continue
                print(f"You have selected minimum confidence of {min_confidence}%")
                min_confidence = min_confidence / 100
                break
            except ValueError:
                print("Invalid value for minimum confidence")
                print("Try again!")
```

```
You have selected minimum confidence of 40.0%
```

Counting Items and finding Frequent Item-sets

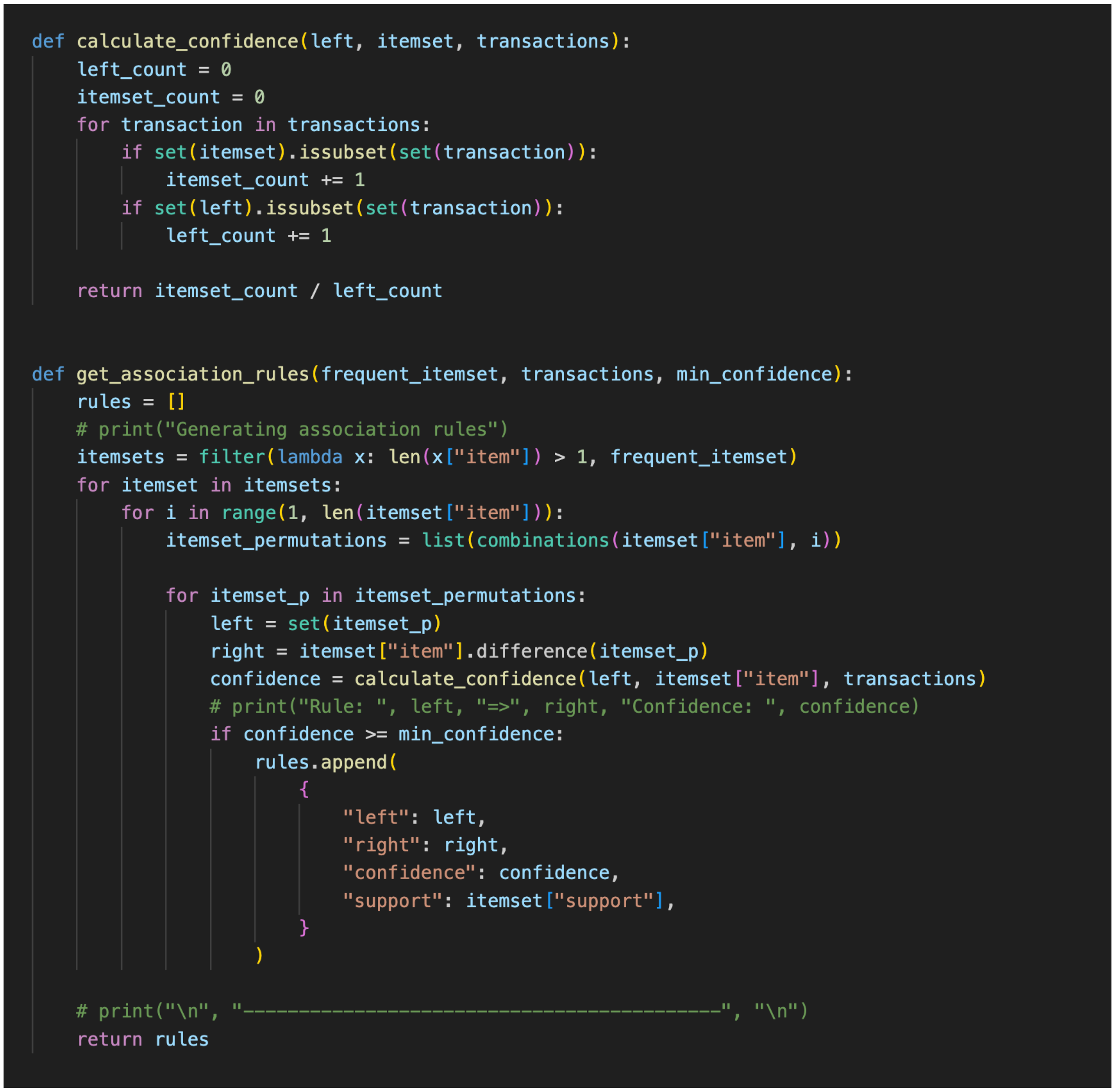


```
You, 29 minutes ago | 1 author (You)
from itertools import combinations


def calculate_support(itemset: tuple, transactions: list[list]):
    counts = 0
    for transaction in transactions:
        if set(itemset).issubset(set(transaction)):
            counts += 1

    return counts / len(transactions)
```



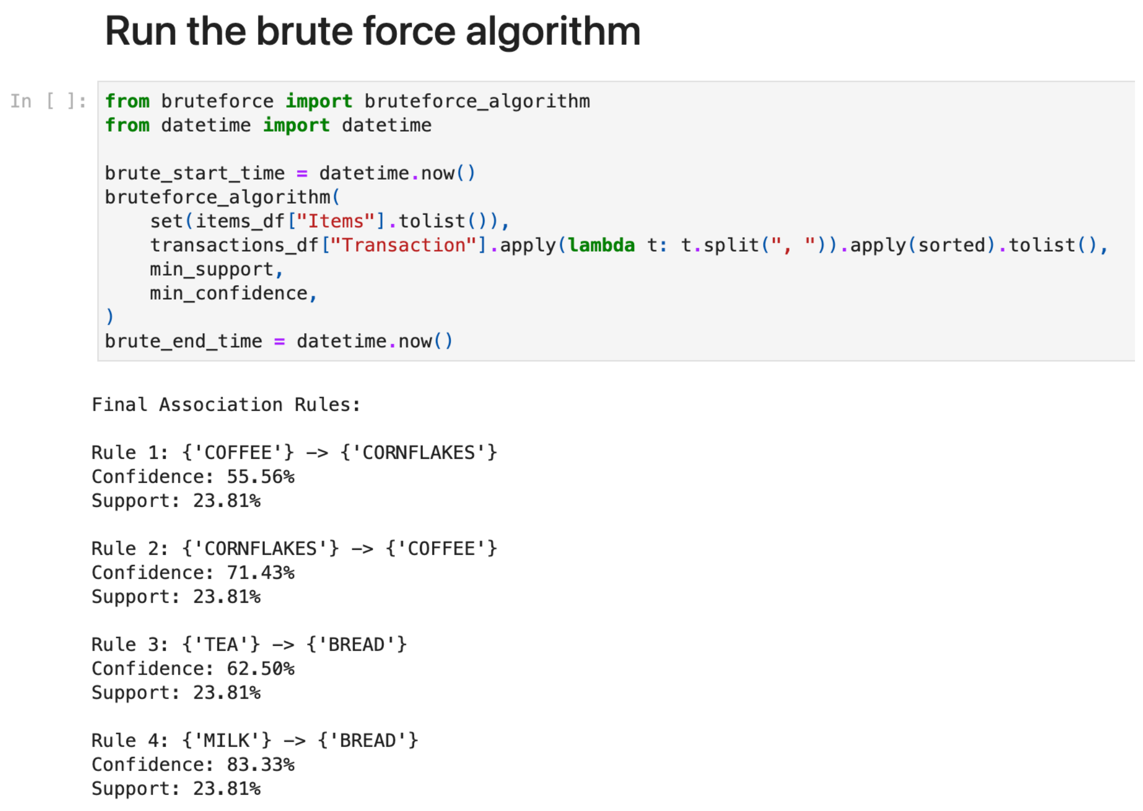
```
def get_frequent_itemset(items: set, transactions, min_support, n=1):
    # print(f"Generating frequent itemset for n = {n} \n")
    # print("Items: ", items, "\n")
    # print("Total items: ", len(items), "\n")
    if len(transactions) == 1:
        return []

    itemset = list(combinations(items, n))
    # print("Itemset: ", itemset, "\n")
    # If itemset contains only one item or no items, return empty list
    # because there are no more combinations possible
    if (len(itemset)) <= 1:
        # print("Itemsets: No more itemset combinations possible", "\n")
        # print("----------------------------------------------", "\n")
        return []

    frequent_itemset = []

    # print("Itemsets:")
    for item in itemset:
        support = calculate_support(item, transactions)
        # print(item, ": ", support)
        if support >= min_support:
            frequent_itemset.append({"item": set(item), "support": support})

    # print("\n")
    # print(
    #     "Frequent Itemset:\n",
    #     pd.DataFrame(frequent_itemset).to_string(index=False),
    #     "\n",
    # )
    # print("--------------------------------------------", "\n")

    frequent_itemset.extend(
        get_frequent_itemset(
            set([item for itemset in frequent_itemset for item in itemset["item"]]),
            transactions,
            min_support,
            n + 1,
        )
    )

    return frequent_itemset
```

Finding association rules



```python
def calculate_confidence(left, itemset, transactions):
    left_count = 0
    itemset_count = 0
    for transaction in transactions:
        if set(itemset).issubset(set(transaction)):
            itemset_count += 1
        if set(left).issubset(set(transaction)):
            left_count += 1

    return itemset_count / left_count


def get_association_rules(frequent_itemset, transactions, min_confidence):
    rules = []
    # print("Generating association rules")
    itemsets = filter(lambda x: len(x["item"]) > 1, frequent_itemset)
    for itemset in itemsets:
        for i in range(1, len(itemset["item"])):
            itemset_permutations = list(combinations(itemset["item"], i))

            for itemset_p in itemset_permutations:
                left = set(itemset_p)
                right = itemset["item"].difference(itemset_p)
                confidence = calculate_confidence(left, itemset["item"], transactions)
                # print("Rule: ", left, "=>", right, "Confidence: ", confidence)
                if confidence >= min_confidence:
                    rules.append(
                        {
                            "left": left,
                            "right": right,
                            "confidence": confidence,
                            "support": itemset["support"],
                        }
                    )

    # print("\n", "-------------------------------------------", "\n")
    return rules
```

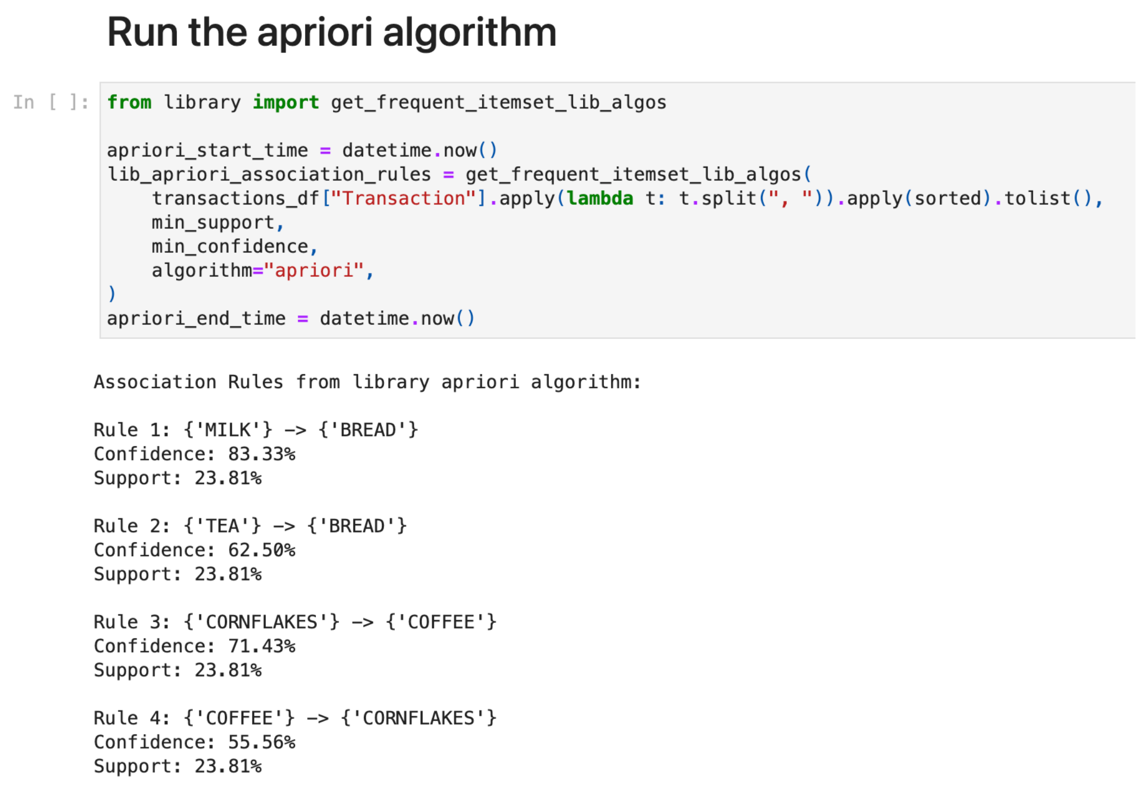
Verified Results with the built-in python package.



Run the brute force algorithm

```
In [ ]: from bruteforce import bruteforce_algorithm
        from datetime import datetime

        brute_start_time = datetime.now()
        bruteforce_algorithm(
            set(items_df["Items"].tolist()),
            transactions_df["Transaction"].apply(lambda t: t.split(", ")).apply(sorted).tolist(),
            min_support,
            min_confidence,
        )
        brute_end_time = datetime.now()
```

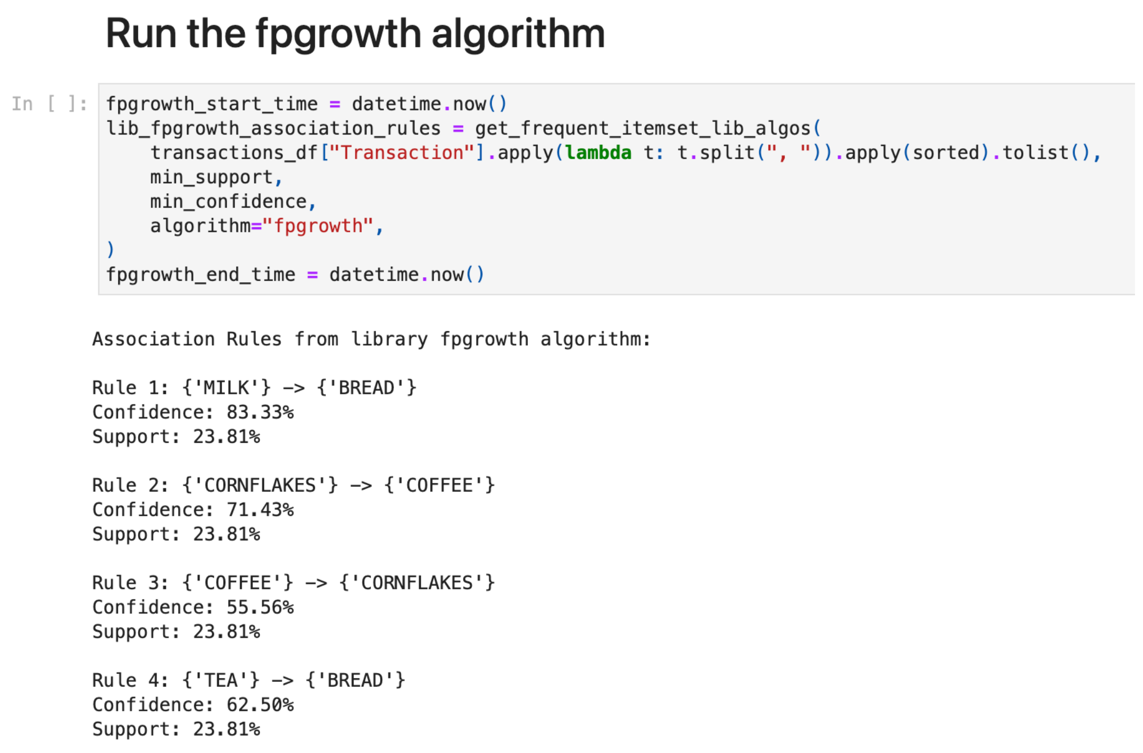
```
Final Association Rules:

Rule 1: {'COFFEE'} -> {'CORNFLAKES'}
Confidence: 55.56%
Support: 23.81%

Rule 2: {'CORNFLAKES'} -> {'COFFEE'}
Confidence: 71.43%
Support: 23.81%

Rule 3: {'TEA'} -> {'BREAD'}
Confidence: 62.50%
Support: 23.81%

Rule 4: {'MILK'} -> {'BREAD'}
Confidence: 83.33%
Support: 23.81%
```



Run the apriori algorithm

```
In [ ]: from library import get_frequent_itemset_lib_algos

        apriori_start_time = datetime.now()
        lib_apriori_association_rules = get_frequent_itemset_lib_algos(
            transactions_df["Transaction"].apply(lambda t: t.split(", ")).apply(sorted).tolist(),
            min_support,
            min_confidence,
            algorithm="apriori",
        )
        apriori_end_time = datetime.now()
```

```
Association Rules from library apriori algorithm:

Rule 1: {'MILK'} -> {'BREAD'}
Confidence: 83.33%
Support: 23.81%

Rule 2: {'TEA'} -> {'BREAD'}
Confidence: 62.50%
Support: 23.81%

Rule 3: {'CORNFLAKES'} -> {'COFFEE'}
Confidence: 71.43%
Support: 23.81%

Rule 4: {'COFFEE'} -> {'CORNFLAKES'}
Confidence: 55.56%
Support: 23.81%
```

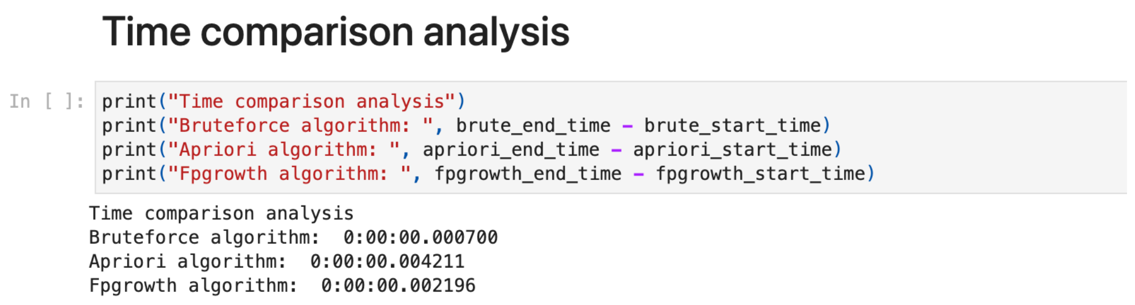


Run the fpgrowth algorithm

```
In [ ]: fpgrowth_start_time = datetime.now()
        lib_fpgrowth_association_rules = get_frequent_itemset_lib_algos(
            transactions_df["Transaction"].apply(lambda t: t.split(", ")).apply(sorted).tolist(),
            min_support,
            min_confidence,
            algorithm="fpgrowth",
        )
        fpgrowth_end_time = datetime.now()
```

```
Association Rules from library fpgrowth algorithm:

Rule 1: {'MILK'} -> {'BREAD'}
Confidence: 83.33%
Support: 23.81%

Rule 2: {'CORNFLAKES'} -> {'COFFEE'}
Confidence: 71.43%
Support: 23.81%

Rule 3: {'COFFEE'} -> {'CORNFLAKES'}
Confidence: 55.56%
Support: 23.81%

Rule 4: {'TEA'} -> {'BREAD'}
Confidence: 62.50%
Support: 23.81%
```

Time comparison analysis



Time comparison analysis

```
In [ ]: print("Time comparison analysis")
        print("Bruteforce algorithm: ", brute_end_time - brute_start_time)
        print("Apriori algorithm: ", apriori_end_time - apriori_start_time)
        print("Fpgrowth algorithm: ", fpgrowth_end_time - fpgrowth_start_time)
```

```
Time comparison analysis
Bruteforce algorithm:  0:00:00.000700
Apriori algorithm:  0:00:00.004211
Fpgrowth algorithm:  0:00:00.002196
```

Link to Git Repository: https://github.com/amishfaldu-njit/cs-634104-mid-term-project