

Implementation and Code Usage

Apriori Algorithm Implementation

Abstract:

I studied and implemented the Apriori Algorithm, a data mining method for identifying association rules, in this midterm assignment. By designing and developing the code, I developed a unique approach to extract meaningful information from transaction data.

Introduction:

Data mining is the process of discovering patterns, correlations, and insights from large datasets to extract valuable information. It uses a variety of methods and algorithms to analyze data and find hidden patterns that can be predicted and utilized in decision-making. Association rules are one of the fundamental concepts in data mining, used to uncover relationships between variables in large datasets. They are typically represented as "if-then" statements that describe the co-occurrence of items in transactions. For example, an association rule might indicate that if a customer buys item A, they are likely to buy item B as well.

The Apriori algorithm is a popular algorithm used to mine association rules from transactional datasets. It works by generating frequent itemsets, which are sets of items that frequently occur together in transactions. The algorithm then creates association rules that satisfy user-specified requirements, including minimum support and minimum confidence, using these frequently occurring itemsets.

The algorithm scans the dataset to identify frequent itemsets, i.e., sets of items that occur together with a frequency greater than or equal to a specified minimum support. Based on the frequent itemsets found, the algorithm generates candidate itemsets by combining items to form larger itemsets. The algorithm prunes the candidate itemsets that cannot possibly be frequent based on the downward closure property. This reduces the number of candidate itemsets that need to be considered in subsequent iterations and is repeated iteratively until no new frequent itemsets can be found. Once all frequent itemsets are identified, the algorithm generates association rules from these itemsets by calculating their support and confidence. Association rules that meet the user-defined thresholds (minimum support and minimum confidence) are considered significant and are returned as the final output.

I implemented Apriori algorithm, FP-growth algorithm and Brute Force algorithm in this project. Following is the implementation of Apriori algorithm:

- We begin the implementation by finding itemsets containing only one item.
- Then, we verify the items and check their support, as it should be greater than or equal to minimum support. We eliminate all the items that have support less than minimum support.
- Then iteratively, we generate larger itemsets from the itemsets found in previous iteration.
- Candidates for the next iteration are generated by joining pairs of frequent itemsets found in previous iteration.
- Once all frequent itemsets are found, association rules are generated from these itemsets.
- The confidence of rule $A \rightarrow B$ is defined as the support of the itemset $\{A,B\}$ divided by the support of the antecedent $\{A\}$.

Core Concepts and Principles:

Frequent Itemset Discovery:

In the Apriori algorithm, discovering frequent itemsets involves iteratively finding itemsets that satisfy a minimum support threshold.

Support and Confidence:

In the Apriori algorithm, support and confidence are two key metrics used to identify meaningful patterns in transactional datasets. Support measures the frequency of occurrence of an itemset in the dataset. Confidence measures the reliability of the association between two itemsets in an association rule.

Association Rule:

In the Apriori algorithm, association rules are generated from frequent itemsets. An association rule is a relationship between two sets of items in a transactional dataset, often represented as "if-then" statements. The rule suggests that if the antecedent set of items is present in a transaction, then the consequent set of items is likely to be present as well.

Data Loading and Processing:

The code reads the transaction data of a retail store from a csv file. Each transaction consists of a list of items bought by a customer.

Determination of Minimum Support and Confidence:

The user is asked to enter values for minimum support and minimum confidence to filter out less significant patterns.

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 our custom Apriori Algorithm implementation with the Apriori library to assess its reliability.

Conclusion:

To sum up, this project shows how data mining ideas, techniques, and principles can be applied. I successfully implemented the FP-growth algorithm, the Apriori method, and the Brute Force technique to extract useful association rules from transaction data from retail stores. The customization of algorithms, iterative "brute force" methodology, and strict adherence to user-specified criteria demonstrate the effectiveness of data mining in identifying significant patterns that support retail industry decision-making.

Screenshots:

Below are the screenshots of the csv files.

Figure 1: Amazon Database.

The screenshot shows a CSV file titled "Amazon_Database". The file has two columns: "Transaction ID" and "Transaction". The "Transaction ID" column contains numbers from 1 to 20, and the "Transaction" column contains lists of items. A green box highlights the first row (header). The data is as follows:

Transaction ID	Transaction
1	
2	T1 Notebook, Pen, Pencil, Eraser
3	T2 Notebook, Pen, Pencil
4	T3 Notebook, Pen, Pencil, Eraser, Sharpener
5	T4 Eraser, Sharpener, Pen refill
6	T5 Eraser, Pen refill, Diary
7	T6 Notebook, Eraser, Sharpener
8	T7 Notebook, Sharpener, Pen refill
9	T8 Pen, Pencil, Eraser
10	T9 Pencil, Eraser, Sharpener, Pen refill
11	T10 Pen refill, Diary, File
12	T11 Notebook, Pen, Pencil, Eraser
13	T12 Notebook, Pen, Pencil, Push pins
14	T13 Notebook, Pen, Pencil, Diary, Push pins
15	T14 Pencil, Eraser, Sharpener
16	T15 Pencil, Eraser
17	T16 Notebook, Pen, Pencil, Eraser
18	T17 Notebook, Pen, Pencil, Eraser
19	T18 Sharpener, Pen refill, Diary
20	T19 Eraser, Sharpener
21	T20 Notebook, Pen, Pencil

Figure 2: Best Buy Database.

The screenshot shows a Microsoft Excel spreadsheet with the title "Best_Buy_Database" centered above the table. The table has 21 rows, indexed from 1 to 21 on the left. Row 1 is a header row with two columns: "Transaction ID" and "Transaction". Rows 2 through 20 list individual transactions, each containing a Transaction ID (e.g., T1, T2, T3, ..., T20) and a list of items purchased. Row 21 is an empty row at the bottom. The columns are labeled "A" and "B" at the top, and there are circular icons with arrows in the top corners.

	A	B
1	Transaction ID	Transaction
2	T1	Desktop,Printer,Flash Drive,Microsoft Office,Speakers,Anti-Virus
3	T2	Laptop,Flash Drive,Microsoft Office,Laptop Case,Anti-Virus
4	T3	Laptop,Printer,Flash Drive,Microsoft Office,Anti-Virus,Laptop Case,External Hard-Drive
5	T4	Laptop,Printer,Flash Drive,Anti-Virus,External Hard-Drive,Laptop Case
6	T5	Laptop,Flash Drive,Laptop Case,Anti-Virus
7	T6	Laptop,Printer,Flash Drive,Microsoft Office
8	T7	Desktop,Printer,Flash Drive,Microsoft Office
9	T8	Laptop,External Hard-Drive,Anti-Virus
10	T9	Desktop,Printer,Flash Drive,Microsoft Office,Laptop Case,Anti-Virus,Speakers,External Hard-Drive
11	T10	Digital Camera,Laptop,Desktop,Printer,Flash Drive,Microsoft Office,Laptop Case,Anti-Virus,External Hard-Drive,Speakers
12	T11	Laptop,Desktop,Laptop Case,External Hard-Drive,Speakers,Anti-Virus
13	T12	Digital Camera,Laptop,Laptop Case,External Hard-Drive,Anti-Virus,Speakers
14	T13	Digital Camera,Speakers
15	T14	Digital Camera,Desktop,Printer,Flash Drive,Microsoft Office
16	T15	Printer,Flash Drive,Microsoft Office,Anti-Virus,Laptop Case,Speakers,External Hard-Drive
17	T16	Digital Camera,Flash Drive,Microsoft Office,Anti-Virus,Laptop Case,External Hard-Drive,Speakers
18	T17	Digital Camera,Laptop,Laptop Case
19	T18	Digital Camera,Laptop Case,Speakers
20	T19	Digital Camera,Laptop,Printer,Flash Drive,Microsoft Office,Speakers,Laptop Case,Anti-Virus
21	T20	Digital Camera,Laptop,Speakers,Anti-Virus,Laptop Case

Figure 3: Walmart Database.

Walmart_Database

Transaction ID	Transaction
T1	Bananas,Pears,Apples
T2	Bananas,Pears,Apples
T3	Bananas,Pears,Apples,Oranges,Cucumber
T4	Bananas,Pears,Apples,Oranges,Blueberries
T5	Bananas,Pears,Oranges,Blueberries
T6	Bananas,Oranges,Blueberries
T7	Bananas,Blueberries
T8	Bananas,Blueberries,Cucumber
T9	Bananas,Apples,Blueberries
T10	Bananas,Apples,Blueberries
T11	Bananas,Apples,Blueberries,Cucumber

Figure 4: K-Mart Database.

The screenshot shows a table titled "K_Mart_Database" with 21 rows. The columns are "Transaction ID" and "Transaction". The transactions are as follows:

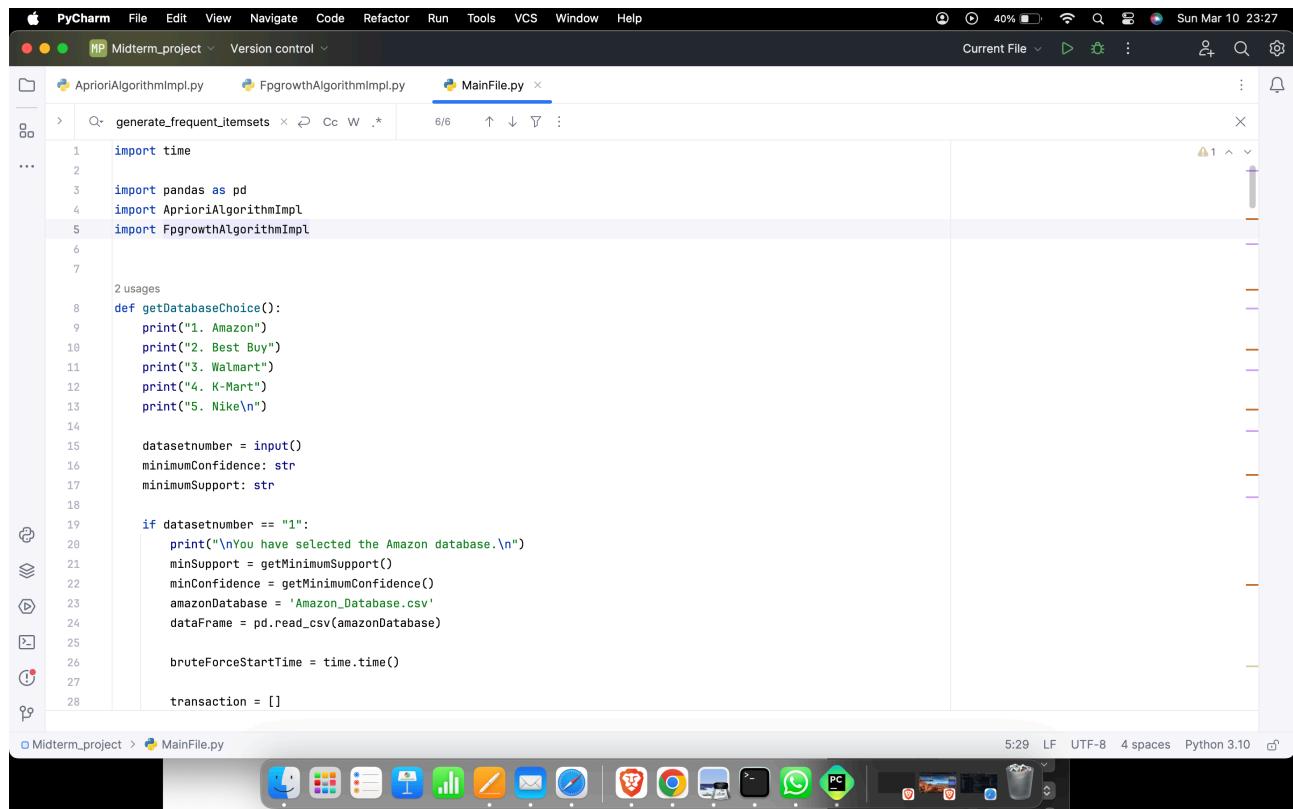
Transaction ID	Transaction
T1	Decorative Pillows,Quilts,Embroidered Bedspread
T2	Embroidered Bedspread,Shams,Kids Bedding,Bedding Collections,Bed Skirts,Bedsheets,Sheets
T3	Decorative Pillows,Quilts,Embroidered Bedspread,Shams,Kids Bedding,Bedding Collections
T4	Kids Bedding,Bedding Collections,Sheets,Bedsheets,Bed Skirts
T5	Decorative Pillows,Kids Bedding,Bedding Collections,Sheets,Bed Skirts,Bedsheets
T6	Bedding Collections,Bedsheets,Bed Skirts,Sheets,Shams,Kids Bedding
T7	Decorative Pillows,Quilts
T8	Decorative Pillows,Quilts,Embroidered Bedspread
T9	Bedsheets,Bed Skirts,Shams,Kids Bedding,Sheets
T10	Quilts,Embroidered Bedspread,Bedding Collections
T11	Bedding Collections,Bedsheets,Bed Skirts,Kids Bedding,Shams,Sheets
T12	Decorative Pillows,Quilts
T13	Embroidered Bedspread,Shams
T14	Sheets,Shams,Bed Skirts,Kids Bedding
T15	Decorative Pillows,Quilts
T16	Decorative Pillows,Kids Bedding,Bed Skirts,Shams
T17	Decorative Pillows,Shams,Bed Skirts
T18	Quilts,Sheets,Kids Bedding
T19	Shams,Bed Skirts,Kids Bedding,Sheets
T20	Decorative Pillows,Bedsheets,Shams,Sheets,Bed Skirts,Kids Bedding

Figure 5: Nike Database.

1	Transaction ID	Transaction
2	T1	Running Shoe,Socks,Sweatshirts,Modern Pants
3	T2	Running Shoe,Socks,Sweatshirts
4	T3	Running Shoe,Socks,Sweatshirts,Modern Pants
5	T4	Running Shoe,Sweatshirts,Modern Pants
6	T5	Running Shoe,Socks,Sweatshirts,Modern Pants,Soccer Shoe
7	T6	Running Shoe,Socks,Sweatshirts
8	T7	Running Shoe,Socks,Sweatshirts,Modern Pants,Tech Pants,Rash Guard,Hoodies
9	T8	Swimming Shirt,Socks,Sweatshirts
10	T9	Swimming Shirt,Rash Guard,Dry Fit V-Nick,Hoodies,Tech Pants
11	T10	Swimming Shirt,Rash Guard,Dry
12	T11	Swimming Shirt,Rash Guard,Dry Fit V-Nick
13	T12	Running Shoe,Swimming Shirt,Socks,Sweatshirts,Modern Pants,Soccer Shoe,Rash Guard,Hoodies,Tech Pants,Dry Fit V-Nick
14	T13	Running Shoe,Swimming Shirt,Socks,Sweatshirts,Modern Pants,Soccer Shoe,Rash Guard,Tech Pants,Dry Fit V-Nick,Hoodies
15	T14	Running Shoe,Swimming Shirt,Rash Guard,Tech Pants,Hoodies,Dry Fit V-Nick
16	T15	Running Shoe,Swimming Shirt,Socks,Sweatshirts,Modern Pants,Dry Fit V-Nick,Rash Guard,Tech Pants
17	T16	Swimming Shirt,Soccer Shoe,Hoodies,Dry Fit V-Nick,Tech Pants,Rash Guard
18	T17	Running Shoe,Socks
19	T18	Socks,Sweatshirts,Modern Pants,Soccer Shoe,Hoodies,Rash Guard,Tech Pants,Dry Fit V-Nick
20	T19	Running Shoe,Swimming Shirt,Rash Guard
21	T20	Running Shoe,Swimming Shirt,Socks,Sweatshirts,Modern Pants,Soccer Shoe,Hoodies,Tech Pants,Rash Guard,Dry Fit V-Nick

Below are screenshots from my code:

We have to execute the MainFile.py in command prompt using command
python MainFile.py



The screenshot shows the PyCharm IDE interface with the 'MainFile.py' tab selected. The code editor displays the following Python script:

```
import time
import pandas as pd
import AprioriAlgorithmImpl
import FpgrowthAlgorithmImpl

def getDatabaseChoice():
    print("1. Amazon")
    print("2. Best Buy")
    print("3. Walmart")
    print("4. K-Mart")
    print("5. Nike\n")

datasetnumber = input()
minimumConfidence: str
minimumSupport: str

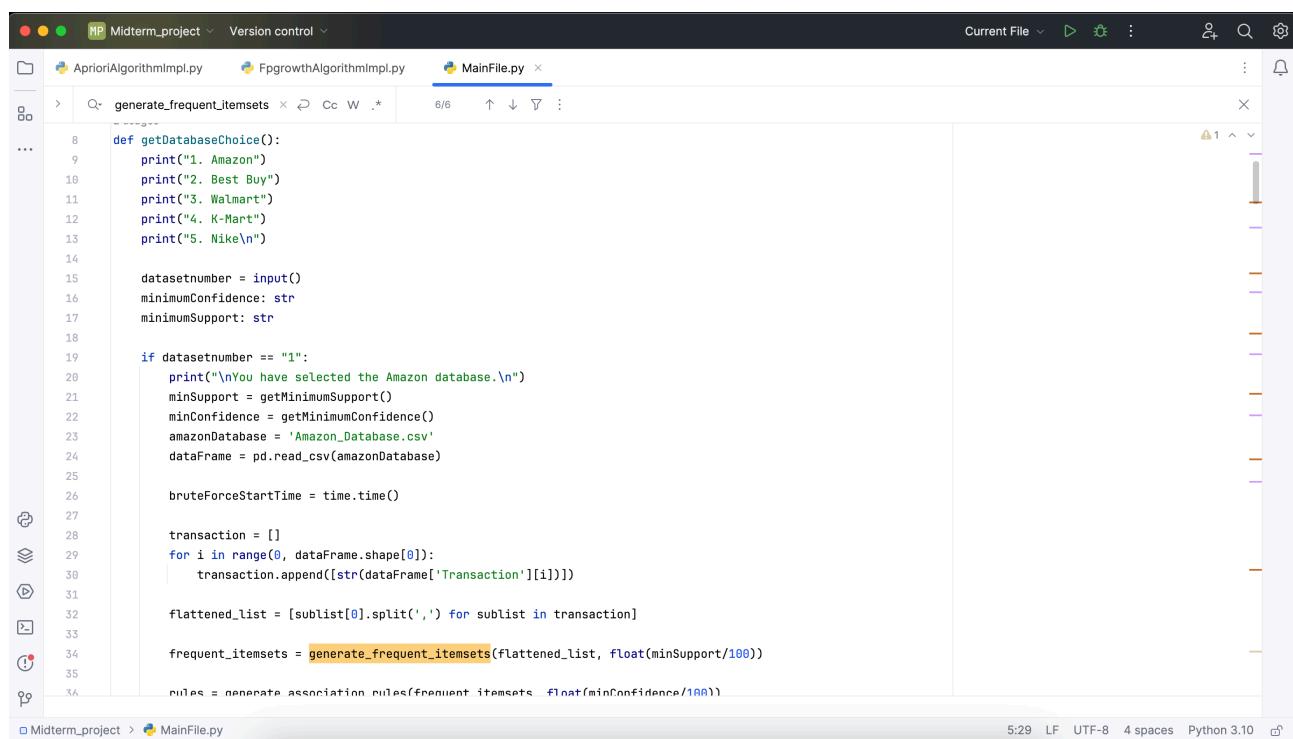
if datasetnumber == "1":
    print("\nYou have selected the Amazon database.\n")
    minSupport = getMinimumSupport()
    minConfidence = getMinimumConfidence()
    amazonDatabase = 'Amazon_Database.csv'
    dataFrame = pd.read_csv(amazonDatabase)

    bruteForceStartTime = time.time()

    transaction = []
```

The status bar at the bottom indicates the file is saved, the encoding is UTF-8, and the Python version is 3.10.

Selecting Database and entering minimum support and confidence:



The screenshot shows the PyCharm IDE interface with the 'MainFile.py' tab selected. The code editor displays the following Python script, which includes logic for selecting a database and generating association rules:

```
def getDatabaseChoice():
    print("1. Amazon")
    print("2. Best Buy")
    print("3. Walmart")
    print("4. K-Mart")
    print("5. Nike\n")

datasetnumber = input()
minimumConfidence: str
minimumSupport: str

if datasetnumber == "1":
    print("\nYou have selected the Amazon database.\n")
    minSupport = getMinimumSupport()
    minConfidence = getMinimumConfidence()
    amazonDatabase = 'Amazon_Database.csv'
    dataFrame = pd.read_csv(amazonDatabase)

    bruteForceStartTime = time.time()

    transaction = []
    for i in range(0, dataFrame.shape[0]):
        transaction.append([str(dataFrame['Transaction'][i])])

    flattened_list = [sublist[0].split(',') for sublist in transaction]

    frequent_itemsets = generate_frequent_itemsets(flattened_list, float(minSupport/100))

    rules = generate_association_rules(frequent_itemsets, float(minConfidence/100))
```

The status bar at the bottom indicates the file is saved, the encoding is UTF-8, and the Python version is 3.10.

Implementing Brute force algorithm for amazon database:

The screenshot shows a code editor window titled "Midterm_project". The current file is "MainFile.py". The code implements a brute-force algorithm to find frequent itemsets and association rules from an Amazon database. It starts by selecting the Amazon database and reading it into a pandas DataFrame. It then initializes variables for support and confidence thresholds, and starts timing the process. It iterates through each transaction in the DataFrame, splitting it into individual items. These items are then used to generate frequent itemsets. Finally, it generates association rules from these frequent itemsets and prints them out.

```
if datasetnumber == "1":
    print("\nYou have selected the Amazon database.\n")
    minSupport = getMinimumSupport()
    minConfidence = getMinimumConfidence()
    amazonDatabase = 'Amazon_Database.csv'
    dataFrame = pd.read_csv(amazonDatabase)

    bruteForceStartTime = time.time()

    transaction = []
    for i in range(0, dataFrame.shape[0]):
        transaction.append([str(dataFrame['Transaction'][i])])

    flattened_list = [sublist[0].split(',') for sublist in transaction]

    frequent_itemsets = generate_frequent_itemsets(flattened_list, float(minSupport/100))

    rules = generate_association_rules(frequent_itemsets, float(minConfidence/100))
    print("\nAssociation Rules:")
    ruleCounter = 1
    for antecedent, consequent, confidence, support in rules:
        print(f"Rule {ruleCounter}: {set(antecedent)} -> {set(consequent)}")
        print("Support: ", support)
        print("Confidence: ", confidence)
        print("-" * 75)
        ruleCounter += 1

    bruteForceEndTime = time.time()

    print(f"\nTime taken by Apriori algorithm to find association rules using apriori algorithm is {bruteForceEndTime - bruteForceStartTime} seconds")
```

Implementing Apriori algorithm and FP-growth algorithm:

Apriori Algorithm using package:

The screenshot shows a code editor window titled "Midterm_project". The current file is "AprioriAlgorithmImpl.py". The code uses a package to implement the Apriori algorithm. It starts by defining a function to generate association rules using the Apriori algorithm. It then loads a dataset from a CSV file and preprocesses it. It uses the package's functionality to find frequent itemsets and then generates association rules based on those itemsets.

```
def association_rules_using_apriori(databaseName, minSupport, minConfidence):
    start_time = time.time()

    # Suppress DeprecationWarnings
    warnings.filterwarnings(action="ignore", category=DeprecationWarning)

    # Load the dataset
    df = pd.read_csv(databaseName)
    pd.set_option('display.max_rows', None)

    # Check the structure and content of the DataFrame
    print("\n")

    # Preprocess the data

    df['Transaction'] = df['Transaction'].str.split(',')
    one_hot_encoded = df['Transaction'].str.join('|').str.get_dummies()

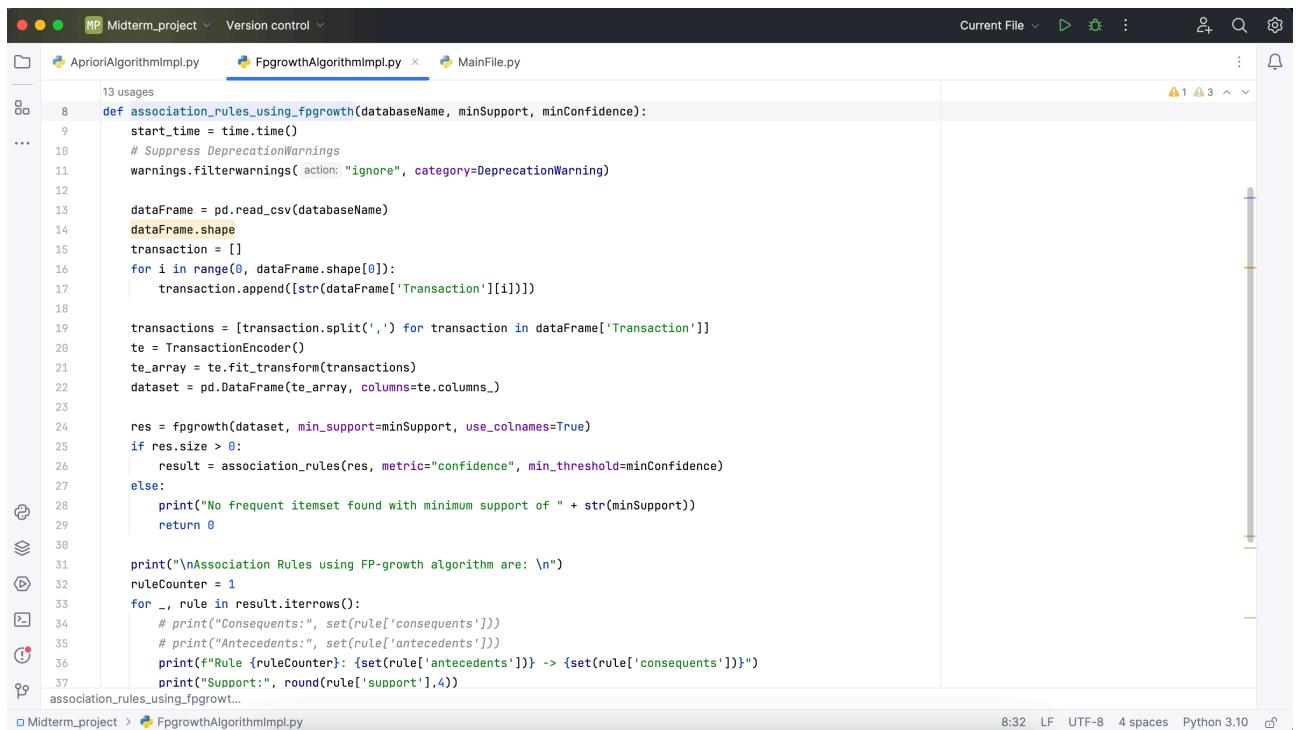
    # Use Apriori algorithm to find frequent itemsets

    frequent_itemsets = apriori(one_hot_encoded, min_support=minSupport, use_colnames=True)

    if frequent_itemsets.size == 0:
        print("No frequent itemset found with minimum support of " + str(minSupport))
        exit()

    print("\n\nAssociation Rules using Apriori algorithm are: \n")
    # Generate association rules
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=minConfidence)
```

Implementing FP-growth algorithm using package:



```
def association_rules_using_fprowth(databaseName, minSupport, minConfidence):
    start_time = time.time()
    # Suppress DeprecationWarnings
    warnings.filterwarnings(action="ignore", category=DeprecationWarning)

    dataFrame = pd.read_csv(databaseName)
    dataFrame.shape
    transaction = []
    for i in range(0, dataFrame.shape[0]):
        transaction.append([str(dataFrame['Transaction'][i])])

    transactions = [transaction.split(',') for transaction in dataFrame['Transaction']]
    te = TransactionEncoder()
    te_array = te.fit_transform(transactions)
    dataset = pd.DataFrame(te_array, columns=te.columns_)

    res = fpgrowth(dataset, min_support=minSupport, use_colnames=True)
    if res.size > 0:
        result = association_rules(res, metric="confidence", min_threshold=minConfidence)
    else:
        print("No frequent itemset found with minimum support of " + str(minSupport))
        return 0

    print("\nAssociation Rules using FP-growth algorithm are: \n")
    ruleCounter = 1
    for _, rule in result.iterrows():
        # print("Consequents:", set(rule['consequents']))
        # print("Antecedents:", set(rule['antecedents']))
        print(f"Rule {ruleCounter}: {set(rule['antecedents'])} -> {set(rule['consequents'])}")
        print("Support:", round(rule['support'], 4))
    association_rules_using_fprowth...
```

Accepting and validating minimum support and minimum confidence:

```
def getMinimumSupport():
    print("\nPlease enter minimum support in % (value from 1 to 100): \n")
    minimumSupport = input()
    minSupport = int(minimumSupport)
    if 0 < minSupport < 100:
        return minSupport
    else:
        return getMinimumSupport()
```

6 usages

```
def getMinimumConfidence():
    print("\nPlease enter minimum Confidence in % (value from 1 to 100): \n")
    minimumConfidence = input()
    minConfidence = int(minimumConfidence)
    if 0 < minConfidence < 100:
        return minConfidence
    else:
        return getMinimumConfidence()
```

Generating frequent itemsets:

Calculating association rules:

```
def generate_association_rules(frequent_itemsets, min_confidence):
    rules = []
    for itemset, support in frequent_itemsets:
        if len(itemset) >= 1:
            for i in range(1, len(itemset)):
                antecedent = itemset[:i]
                consequent = itemset[i:]
                antecedent_support = calculate_support(antecedent, frequent_itemsets)
                if antecedent_support > 0:
                    confidence = support / antecedent_support
                    if confidence >= min_confidence:
                        rules.append((antecedent, consequent, confidence, antecedent_support))
    return rules
```

Result:

Brute force algorithm output:

```
def generate_frequent_itemsets(transactions, min_support):
    items = {}
    for transaction in transactions:
        for item in transaction:
            if item in items:
                items[item] += 1
            else:
                items[item] = 1

    frequent_itemsets = []
    n = len(transactions)
    for item, count in items.items():
        support = count / n
        if support >= min_support:
            frequent_itemsets.append(([item], support))

    k = 2
    while True:
        candidates = generate_candidates([itemset[0] for itemset in frequent_itemsets], k)
        if not candidates:
            break

        frequent_candidates = []
        for candidate in candidates:
            count = 0
            for transaction in transactions:
                if set(candidate).issubset(set(transaction)):
```

Apriori Algorithm Output:

Terminal Local × + ▾

(base) zeal@zeals-Air Midterm_project % python MainFile.py

Which dataset would you like to use Zeal?

1. Amazon
2. Best Buy
3. Walmart
4. K-Mart
5. Nike

1

You have selected the Amazon database.

Please enter minimum support in % (value from 1 to 100):

40

Please enter minimum Confidence in % (value from 1 to 100):

40

Please enter minimum Confidence in % (value from 1 to 100):

40

Association Rules:

Rule 1: {'Notebook'} -> {'Pen'}

Support: 0.55

Confidence: 0.81818181818181

Rule 2: {'Notebook'} -> {'Pencil'}

Support: 0.55

Confidence: 0.81818181818181

Rule 3: {'Pen'} -> {'Pencil'}

Support: 0.5

Confidence: 1.0

Rule 4: {'Eraser'} -> {'Pencil'}

Support: 0.65

Confidence: 0.6923076923076923

Rule 5: {'Notebook'} -> {'Pencil', 'Pen'}

Support: 0.55

Confidence: 0.81818181818181

Rule 6: {'Notebook', 'Pen'} -> {'Pencil'}

Association Rules using Apriori algorithm are:

Rule 1: {'Pencil'} -> {'Eraser'}

Support: 0.45

Confidence: 0.6923076923076923

Rule 2: {'Eraser'} -> {'Pencil'}

Support: 0.45

Confidence: 0.6923076923076923

Rule 3: {'Notebook'} -> {'Pen'}

Support: 0.45

Confidence: 0.8181818181818181

Rule 4: {'Pen'} -> {'Notebook'}

Support: 0.45

Confidence: 0.9

Rule 5: {'Notebook'} -> {'Pencil'}

Support: 0.45

Confidence: 0.8181818181818181

Rule 6: {'Pencil'} -> {'Notebook'}

Support: 0.45

Confidence: 0.6923076923076923

Execution of all 3 algorithms:

Total execution time of Brute Force algorithm is 0.002 seconds.

Total execution time of Apriori algorithm is 0.0097 seconds.

Total execution time of FP-growth algorithm is 0.0035 seconds.

(base) zeal@zeals-Air Midterm_project % █

Association Rules using FP-growth algorithm are:

Rule 1: {'Pencil'} -> {'Eraser'}

Support: 0.45

Confidence: 0.6923

Rule 2: {'Eraser'} -> {'Pencil'}

Support: 0.45

Confidence: 0.6923

Rule 3: {'Notebook'} -> {'Pencil'}

Support: 0.45

Confidence: 0.8182

Rule 4: {'Pencil'} -> {'Notebook'}

Support: 0.45

Confidence: 0.6923

Rule 5: {'Pencil'} -> {'Pen'}

Support: 0.5

Confidence: 0.7692

Rule 6: {'Pen'} -> {'Pencil'}

Support: 0.5

Confidence: 1.0
