NAME: Narmada Peddi

NJIT UCID: np863

Email Address: np863@njit.edu

03-10-2024

Professor: Yasser Abduallah

CS 634-104 Data Mining

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm

Abstract:

In this project, I implemented an Apriori algorithm that generates frequent item pattern sets and association rules. The algorithm has user defined minimum support and confidence and generates useful insights based on the above parameters.

Introduction:

Data Mining has bunch of algorithms that generate frequent item pattern sets and association rules. One such is Apriori Algorithm. To demonstrate the Apriori algorithm I have chosen 5 datasets in total. At first, the algorithm looks for individual items that are frequently bought together. The frequent item sets are then generated. Only those are kept which satisfy the minimum support value meaning that the frequent item sets that do not satisfy the minimum support value are discarded. Now, association rules are generated based on the confidence value.

To sum up I have carried out the following steps in the algorithm:

- Loading the datasets, asking for user prompts which includes datasets, minimum support and confidence.
- Defined functions that calculate the frequent item sets and generate association rules.
- Printing the association rules along with their support and confidence.
- Displaying the execution time of the program at the end.

Core Concepts and Principles:

Frequent Item Discovery:

The frequent item sets are the ones the items that are most frequently bought together. This help in understanding us the customer pattern, behaviour and how likely certain items are purchased.

Support and Confidence:

Support defines how frequently an item occurs whereas confidence describes the chance of items that are frequently bought together.

Association Rules:

Association rules help in discovering important patterns and relationships in items that are frequently together.

Project Workflow:

The implementation structure is as follows:

Data Loading and Preprocessing:

There are 5 datasets in total, each dataset containing transaction ID, and Transaction List (Items).

Minimum Support and Confidence:

The user is prompted to enter the minimum support and confidence value. This aids in eliminating less frequent item sets.

Candidate item sets Iteration:

The candidate item sets are iterated over to find frequent item sets which increase in size.

Support Calculation:

Support is calculated to identify how frequent an item occurs in the dataset. Item sets that satisfy the minimum support are retained, whereas the rest are discarded.

Confidence Calculation:

Calculating confidence will help us in determining the strength of a particular association rule.

Association Rules:

Association rules are printed when both the support and confidence values are satisfied.

Results and Evaluation:

The projects accuracy is determined by certain factors such as support, confidence, and association rules.

Conclusion: The project demonstrates how Apriori algorithm can be used to generate frequent pattern item sets, and association rules.

All the .CSV Files are downloaded from the google.

Screenshots

.CSV File example of Cars Dataset.

```
In [6]: import pandas as pd
        df = pd.read_csv('Cars_List.csv')
        df
```

	TransactionID	TransactionList
0	1	Sedan,Coupe,Wagon,Convertibles,SUV
1	2	Hybrid,Convertibles,SUV
2	3	SUV,Minivans,Trucks,Convertibles
3	4	Luxary,Convertibles,SUV
4	5	Sports,Hybrid,Coupe,Sedan,Convertibles,SUV
5	6	Wagon,SUV,Trucks,Convertibles
6	7	Sedan,Coupe,SUV
7	8	Sedan,Convertibles,SUV
8	9	SUV,Sedan,Hybrid,Luxary,Convertibles
9	10	Trucks,Minivans,SUV
10	11	SUV,Sedan,Convertibles
11	12	Luxary,Convertibles,Sports,SUV
12	13	Sports,Sedan,SUV
13	14	SUV,Convertibles
14	15	Sedan,Sports,Convertibles,SUV
15	16	SUV,Coupe,Luxary,Convertibles
16	17	Sedan,Wagon,Trucks,Convertibles,SUV
17	18	Sedan,SUV,Convertibles
18	19	SUV,Convertibles,Sports
19	20	Sports,Convertibles

Screenshots of the code:

1. **User Prompt**

```
print("Welcome to the apriori algorithms. \n Please chose the dataset you want: \n 1. Nike \n 2.
while True:
    choice_of_data=input()
    if(choice_of_data=='1'):
        dataList=pd.read_csv('Nike.csv')
        print('User chose Test dataset')
        break
    elif(choice_of_data=='2'):
        dataList=pd.read_csv('Kmart - Sheet1.csv')
        print('User chose GroceryStore dataset')
        break
    elif(choice_of_data=='3'):
        dataList=pd.read_csv('Cars_List.csv')
        print('User chose Cars List dataset')
        break
    elif(choice_of_data=='4'):
        dataList=pd.read_csv('Games_Transaction_List.csv')
        print('User chose Games dataset')
        break
    elif(choice_of_data=='5'):
        dataList=pd.read_csv('Costco.csv')
        print('User chose Costco dataset')
        break
    else:
        print("Invalid data, please enter the number corresponding to the data")
        break



print("Enter the Minimum Support (in percentage) : ", end=" ")
minsupport = input()
print("Enter the Minimum Confidence (in percentage) : ", end=" ")
minconfidence = input()
min_support = float(minsupport)/100
min_conf = float(minconfidence)/100
print('\n')
print("The minimum support is :", minsupport)
```

2. **Loading Transactions from Dataset**

```
def load_transactions(dataList):
    Transactions = []
    df_items = dataList['TransactionList']
    comma_splitted_df = df_items.apply(lambda x: x.split(','))
    for i in comma_splitted_df:
        Transactions.append(i)
    return Transactions
load_transactions(dataList)

Transactions = load_transactions(dataList)
Transactions
```

3. **Generating candidate Item sets**

```
def calculateCandidate(Lk):
    res = []
    print("len Lk", len(Lk))
    for i in range(len(Lk)):
        for j in range(i+1, len(Lk)):
            it1 = Lk[i]
            it2 = Lk[j]
            it11 = list(it1)
            it22 = list(it2)
            it11.sort()
            it22.sort()
            print("it11",it11)
            print("it22",it22)
            if it11[:len(it1)-1] == it22[:len(it1)-1]:
                res.append(it1 | it2)
                print("Res: ", res)
    return res
```

4. Calculating Frequency Support

```
def calculate_frequency_support():
    support = {}
    candidate = [[]]
    Lk = [[]]
    C1 = set()
    for t in Transactions:
        for item in t:
            C1.add(frozenset([item]))
            #print(C1)
            #print("*****")
    print("----------------------------------------")
    print("C1: ", C1)
    candidate.append(C1)
    #print(candidate)
    print("----------------------------------------")
    print("Transactions: ",Transactions)
    count = scan(Transactions, C1)
    print("----------------------------------------")
    print("Count: ", count)
    Lk.append(list(count.keys()))
    print("----------------------------------------")
    print("Lk: ", Lk)
    support.update(count)
    print("----------------------------------------")
    print("support: ", support)
    print("----------------------------------------")
    print("candidate: ",candidate)
    k = 1
    while len(Lk[k]) > 0:
        print("++++++++++++++++++++++++++++++++++++++")
        print("k=", k)
        print("Lk[k]: ", Lk[k])
        candidate.append(calculateCandidate(Lk[k]))
        print("candidate: ", candidate)
        print("candidate[k+1]: ",candidate[k+1])
        count = scan(Transactions, candidate[k+1])
        support.update(count)
        Lk.append(list(count.keys()))
        k += 1
    return Lk, support
```

5. **Printing Association Rules**

```
def EvaluateAssociationRules(frequent, support):
    fresult = []
    for i in range(2, len(frequent)):
        if len(frequent[i]) == 0:
            break
        freq_sets = frequent[i]

        for fs in freq_sets:
            for right in [frozenset([x]) for x in fs]:
                left = fs-right
                confidence = support[fs] / support[left]
                if confidence >= min_conf:
                    fresult.append([Rule(left, right, fs), support[fs], con

        if len(freq_sets[0]) != 2:

            for fs in freq_sets:
                right = [frozenset([x]) for x in fs]
                EvaluateSecondaryRules(fs, right, fresult, support)

    fresult.sort(key=lambda x: str(x[0]))
    return fresult
```

Output

(Dataset:4)

- **Association rules and Frequent Pattern Item Sets with Support= 30 and Confidence= 40**

- **Association Rules**

```
Enter your choice: 2
User chose Grocery Store dataset
Enter the Minimum Support (in percentage): 30
Enter the Minimum Confidence (in percentage): 40

Frequent itemsets found with FP-Growth algorithm:
Itemset: ('Kids Bedding',), Support: 1
Itemset: (' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', ' Bedspreads', 'Kids Bedding'), Support: 1
```

```
User chose Sports dataset
Enter the Minimum Support (in percentage) :  30
Enter the Minimum Confidence (in percentage) :  40


The minimum support is : 30
The minimum Confidence is : 40
----------------------------------------
C1:  {frozenset({'Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'FIFA'}), frozens
et({'Halo'}), frozenset({'Mario'}), frozenset({'Skyrim'}), frozenset({'NFS'}), frozenset({'Dota2'})}
----------------------------------------
Transactions:  [['Assasins_Creed', 'Mario', 'Counter_Strike'], ['Dota2', 'Counter_Strike'], ['Counter_Strike', 'NFS',
'FIFA', 'Assasins_Creed', 'Dota2'], ['Halo', 'Dota2'], ['FIFA', 'Skyrim', 'Dota2', 'Assasins_Creed'], ['PUBG', 'Count
er_Strike', 'Skyrim', 'Dota2'], ['Dota2', 'PUBG', 'Counter_Strike', 'NFS'], ['Skyrim', 'Assasins_Creed', 'Dota2', 'Co
unter_Strike'], ['NFS', 'Skyrim'], ['Counter_Strike', 'Dota2', 'Mario'], ['Counter_Strike', 'PUBG', 'Mario', 'Dota
2'], ['NFS', 'FIFA', 'Dota2'], ['Counter_Strike', 'Dota2', 'Halo'], ['Counter_Strike', 'Dota2'], ['Halo', 'Mario', 'C
ounter_Strike'], ['Skyrim', 'PUBG', 'Counter_Strike'], ['FIFA', 'NFS', 'Counter_Strike'], ['Counter_Strike', 'NFS'],
['Mario', 'Dota2', 'Counter_Strike'], ['Dota2', 'Counter_Strike']]
----------------------------------------
Count:  {frozenset({'Counter_Strike'}): 0.8, frozenset({'NFS'}): 0.3, frozenset({'Dota2'}): 0.7}
----------------------------------------
Lk:  [[], [frozenset({'Counter_Strike'}), frozenset({'NFS'}), frozenset({'Dota2'})]]
----------------------------------------
support:  {frozenset({'Counter_Strike'}): 0.8, frozenset({'NFS'}): 0.3, frozenset({'Dota2'}): 0.7}
----------------------------------------
candidate:  [[], {frozenset({'Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'FIF
A'}), frozenset({'Halo'}), frozenset({'Mario'}), frozenset({'Skyrim'}), frozenset({'NFS'}), frozenset({'Dota2'})}]
++++++++++++++++++++++++++++++++++++++
k= 1
Lk[k]:  [frozenset({'Counter_Strike'}), frozenset({'NFS'}), frozenset({'Dota2'})]
len Lk 3
it11 ['Counter_Strike']
it22 ['NFS']
Res:  [frozenset({'NFS', 'Counter_Strike'})]
it11 ['Counter_Strike']
it22 ['Dota2']
Res:  [frozenset({'NFS', 'Counter_Strike'}), frozenset({'Dota2', 'Counter_Strike'})]
it11 ['NFS']
it22 ['Dota2']
Res:  [frozenset({'NFS', 'Counter_Strike'}), frozenset({'Dota2', 'Counter_Strike'}), frozenset({'NFS', 'Dota2'})]
candidate:  [[], {frozenset({'Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'FIF
A'}), frozenset({'Halo'}), frozenset({'Mario'}), frozenset({'Skyrim'}), frozenset({'NFS'}), frozenset({'Dota2'})}, [f
rozenset({'NFS', 'Counter_Strike'}), frozenset({'Dota2', 'Counter_Strike'}), frozenset({'NFS', 'Dota2'})]]
candidate[k+1]:  [frozenset({'NFS', 'Counter_Strike'}), frozenset({'Dota2', 'Counter_Strike'}), frozenset({'NFS', 'Do
ta2'})]
++++++++++++++++++++++++++++++++++++++
k= 2
Lk[k]:  [frozenset({'Dota2', 'Counter_Strike'})]
len Lk 1
candidate:  [[], {frozenset({'Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'FIF
A'}), frozenset({'Halo'}), frozenset({'Mario'}), frozenset({'Skyrim'}), frozenset({'NFS'}), frozenset({'Dota2'})}, [f
rozenset({'NFS', 'Counter_Strike'}), frozenset({'Dota2', 'Counter_Strike'}), frozenset({'NFS', 'Dota2'})], []]
candidate[k+1]:  []
Frequency:  [[], [frozenset({'Counter_Strike'}), frozenset({'NFS'}), frozenset({'Dota2'})], [frozenset({'Dota2', 'Cou
nter_Strike'})], []]
Support:  {frozenset({'Counter_Strike'}): 0.8, frozenset({'NFS'}): 0.3, frozenset({'Dota2'}): 0.7, frozenset({'Dota
2', 'Counter_Strike'}): 0.55}
```

Association Rules for Dataset 1:

```
----- > Association With Support and Confidence: < -------

Rule:  Dry_FitV-Nick ==> Rash_Guard
Support:  0.45
Confidence:  1.0


Rule:  Dry_FitV-Nick ==> Rash_Guard,Swimming_Shirt
Support:  0.4
Confidence:  0.888888888888889


Rule:  Dry_FitV-Nick ==> Rash_Guard,Tech_Pants
Support:  0.4
Confidence:  0.888888888888889


Rule:  Dry_FitV-Nick ==> Swimming_Shirt
Support:  0.4
Confidence:  0.888888888888889


Rule:  Dry_FitV-Nick ==> Tech_Pants
```

Association Rules for Dataset 2:

```
----- > Association With Support and Confidence: < -------

Rule:   Bed Skirts ==>  Kids Bedding
Support:  0.45
Confidence:  0.8181818181818181


Rule:   Bed Skirts ==>  Kids Bedding, Shams
Support:  0.4
Confidence:  0.7272727272727273


Rule:   Bed Skirts ==>  Kids Bedding, Sheets
Support:  0.4
Confidence:  0.7272727272727273


Rule:   Bed Skirts ==>  Shams
Support:  0.45
Confidence:  0.8181818181818181
```

Association Rules for Dataset 3:

```
----- > Association With Support and Confidence: < -------

Rule:  Convertibles ==> SUV
Support:  0.8
Confidence:  0.9411764705882354


Rule:  Convertibles,Luxary ==> SUV
Support:  0.2
Confidence:  1.0


Rule:  Convertibles,SUV ==> Sedan
Support:  0.4
Confidence:  0.5


Rule:  Convertibles,Sedan ==> SUV
Support:  0.4
Confidence:  1.0
```

Association Rules for Dataset 5:

```
----- > Association With Support and Confidence: < -------

Rule:  milk ==> whipped_cream
Support:  0.4
Confidence:  0.6153846153846154


Rule:  whipped_cream ==> milk
Support:  0.4
Confidence:  0.888888888888889
```

Association Rules and Frequent Pattern Item sets (Dataset:4) with Support = 50 and Confidence= 60

```
----- > Association With Support and Confidence: < -------

Rule:  Counter_Strike ==> Dota2
Support:  0.55
Confidence:  0.6875


Rule:  Dota2 ==> Counter_Strike
Support:  0.55
Confidence:  0.7857142857142858


------------------------- RUNNING TIME:-----------------------------------
The Runtime of the program is: 0.00039482116699218 75seconds
-------------------------------------------------------------------------
```

Association Rules and Frequent Pattern Item sets (Dataset:4) with Support = 50 and Confidence= 60

```
 3.Vehicles
 4. Sports
 5.Costco
4
User chose Sports dataset
Enter the Minimum Support (in percentage) :  50
Enter the Minimum Confidence (in percentage) :  60


The minimum support is : 50
The minimum Confidence is : 60
----------------------------------------
C1:  {frozenset({'Skyrim'}), frozenset({'Halo'}), frozenset({'NFS'}), frozenset({'FIFA'}), frozenset({'Mario'}), frozenset({'
Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'Dota2'})}
----------------------------------------
Transactions:  [['Assasins_Creed', 'Mario', 'Counter_Strike'], ['Dota2', 'Counter_Strike'], ['Counter_Strike', 'NFS', 'FIFA',
'Assasins_Creed', 'Dota2'], ['Halo', 'Dota2'], ['FIFA', 'Skyrim', 'Dota2', 'Assasins_Creed'], ['PUBG', 'Counter_Strike', 'Sky
rim', 'Dota2'], ['Dota2', 'PUBG', 'Counter_Strike', 'NFS'], ['Skyrim', 'Assasins_Creed', 'Dota2', 'Counter_Strike'], ['NFS',
'Skyrim'], ['Counter_Strike', 'Dota2', 'Mario'], ['Counter_Strike', 'PUBG', 'Mario', 'Dota2'], ['NFS', 'FIFA', 'Dota2'], ['Co
unter_Strike', 'Dota2', 'Halo'], ['Counter_Strike', 'Dota2'], ['Halo', 'Mario', 'Counter_Strike'], ['Skyrim', 'PUBG', 'Counte
r_Strike'], ['FIFA', 'NFS', 'Counter_Strike'], ['Counter_Strike', 'NFS'], ['Mario', 'Dota2', 'Counter_Strike'], ['Dota2', 'Co
unter_Strike']]
----------------------------------------
Count:  {frozenset({'Counter_Strike'}): 0.8, frozenset({'Dota2'}): 0.7}
----------------------------------------
Lk:  [[], [frozenset({'Counter_Strike'}), frozenset({'Dota2'})]]
----------------------------------------
support:  {frozenset({'Counter_Strike'}): 0.8, frozenset({'Dota2'}): 0.7}
----------------------------------------
candidate:  [[], {frozenset({'Skyrim'}), frozenset({'Halo'}), frozenset({'NFS'}), frozenset({'FIFA'}), frozenset({'Mario'}),
frozenset({'Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'Dota2'})}]
++++++++++++++++++++++++++++++++++++++
k= 1
Lk[k]:  [frozenset({'Counter_Strike'}), frozenset({'Dota2'})]
len Lk 2
it11 ['Counter_Strike']
it22 ['Dota2']
Res:  [frozenset({'Dota2', 'Counter_Strike'})]
candidate:  [[], {frozenset({'Skyrim'}), frozenset({'Halo'}), frozenset({'NFS'}), frozenset({'FIFA'}), frozenset({'Mario'}),
frozenset({'Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'Dota2'})}, [frozenset({'Dota
2', 'Counter_Strike'})]]
candidate[k+1]:  [frozenset({'Dota2', 'Counter_Strike'})]
++++++++++++++++++++++++++++++++++++++
k= 2
Lk[k]:  [frozenset({'Dota2', 'Counter_Strike'})]
len Lk 1
candidate:  [[], {frozenset({'Skyrim'}), frozenset({'Halo'}), frozenset({'NFS'}), frozenset({'FIFA'}), frozenset({'Mario'}),
frozenset({'Counter_Strike'}), frozenset({'PUBG'}), frozenset({'Assasins_Creed'}), frozenset({'Dota2'})}, [frozenset({'Dota
2', 'Counter_Strike'})], []]
candidate[k+1]:  []
Frequency:  [[], [frozenset({'Counter_Strike'}), frozenset({'Dota2'})], [frozenset({'Dota2', 'Counter_Strike'})], []]
Support:  {frozenset({'Counter_Strike'}): 0.8, frozenset({'Dota2'}): 0.7, frozenset({'Dota2', 'Counter_Strike'}): 0.55}
```

FP GROWTH ALGORITHM

Abstract:

In this project, I implemented an FP- Growth algorithm that generates frequent item pattern sets and association rules. The algorithm has user defined minimum support and confidence and generates useful insights based on the above parameters.

Introduction:

Data Mining has bunch of algorithms that generate frequent item pattern sets and association rules. One such is FP-Growth Algorithm. To demonstrate the FP-Growth algorithm I have chosen 5 datasets in

total. Initially, the datasets are scanned to find the frequency of each item. Items that meet the minimum support are retained. An FP tree is built in which the items are inserted into the tree. A conditional pattern base is generated for each frequent item. The process in continued until no more frequent item sets are generated.

To sum up I have carried out the following steps in the algorithm:

- Loading the datasets, asking for user prompts which includes datasets, minimum support and confidence.
- Converting Transactions into Dictionary.
- Applying FP-growth algorithm to find frequent pattern Item sets
- Generated association rules based on Frequent Pattern Item sets.
- Defined functions that calculate the frequent item sets and generate association rules.
- Printing the association rules along with their support and confidence.
- Displaying the execution time of the program at the end.

Core Concepts and Principles:

Frequent Item Discovery:

The frequent item sets are the ones the items that are most frequently bought together. This help in understanding us the customer pattern, behaviour and how likely certain items are purchased.

FP-Tree:

It is a data structure used by FP-Tree to represent the items in a dataset which stores information such as support counts

Header Table:

It stores the value of each unique item including support count and a link to the tree.

Conditional Pattern Base:

It generates conditional FP-trees for mining frequent items.

Pattern Growth:

Unlike Apriori, FP-growth generates conditional FP trees to identify frequent patterns.

Support Calculation:

Support is calculated to identify how frequent an item occurs in the dataset. Item sets that satisfy the minimum support are retained, whereas the rest are discarded.

Confidence Calculation:

Calculating confidence will help us in determining the strength of a particular association rule.

Association Rules:

Association rules are printed when both the support and confidence values are satisfied.

Results and Evaluation:

The projects accuracy is determined by certain factors such as support, confidence, and association rules.

Conclusion:

The project demonstrates how FP-Growth algorithm can be used to generate frequent pattern item sets, and association rules.

Screenshots

Code Snippets:

1. **Representing a Node:**

```
class TreeNode:
    def __init__(self, item, count, parent):
        self.item = item
        self.count = count
        self.parent = parent
        self.children = {}
        self.link = None
```

2. **Inserting Transactions to FP-tree**

```python
def insert_tree(transaction, tree, header_table, count):
    if transaction[0] in tree.children:
        tree.children[transaction[0]].count += count
    else:
        tree.children[transaction[0]] = TreeNode(transaction[0], count, tree)
        if header_table[transaction[0]][1] is None:
            header_table[transaction[0]][1] = tree.children[transaction[0]]
        else:
            update_header(header_table[transaction[0]][1], tree.children[transaction[0]])
    if len(transaction) > 1:
        insert_tree(transaction[1:], tree.children[transaction[0]], header_table, count)
```

3. **Updating Header**

```python
def update_header(node, target_node):
    while node.link is not None:
        node = node.link
    node.link = target_node
```

4. **Constructing FP-Tree**

```python
def construct_fp_tree(data, min_support):
    item_counts = {}
    for transaction, count in data.items():
        for item in transaction:
            item_counts[item] = item_counts.get(item, 0) + count
    item_counts = {k: v for k, v in item_counts.items() if v >= min_support}
    header_table = {item: [count, None] for item, count in item_counts.items()}
    tree_root = TreeNode(None, 1, None)
    for transaction, count in data.items():
        sorted_items = [item for item in transaction if item in item_counts]
        sorted_items.sort(key=lambda x: item_counts[x], reverse=True)
        if len(sorted_items) > 0:
            insert_tree(sorted_items, tree_root, header_table, count)
    return tree_root, header_table
```

5. Mining Tree

```
def mine_tree(header_table, min_support, prefix, frequent_itemsets):
    sorted_items = sorted(list(header_table.items()), key=lambda p: p[1][0])
    for item, (count, node) in sorted_items:
        new_prefix = prefix.copy()
        new_prefix.add(item)
        frequent_itemsets[tuple(sorted(new_prefix))] = count
        conditional_pattern_base = find_conditional_pattern_base(node)
        conditional_tree_data = {}
        for pattern, count in conditional_pattern_base:
            conditional_tree_data[pattern] = count
        conditional_tree_root, conditional_header_table = construct_fp_tree(conditional_tree_data
        if conditional_header_table:
            mine_tree(conditional_header_table, min_support, new_prefix, frequent_itemsets)
```

6. Conditional Pattern Base

```
def find_conditional_pattern_base(node):
    patterns = []
    while node is not None:
        prefix_path = ascend_tree(node)
        if len(prefix_path) > 1:
            patterns.append((tuple(prefix_path[1:]), node.count))
        node = node.link
    return patterns
```

7. Generating Association Rules

```
def generate_association_rules(frequent_itemsets, min_confidence):
    association_rules = []
    for itemset in frequent_itemsets.keys():
        if len(itemset) > 1:
            rules_from_itemset = generate_rules_from_itemset(itemset, frequent_itemsets, min_conf
            association_rules.extend(rules_from_itemset)
    return association_rules
```

Outputs:

Association Rules Dataset1:

```
Association rules found with FP-Growth algorithm:
Rule: Dry -> Swimming_Shirt, Confidence: 1.0
Rule: Dry -> Rash_Guard, Confidence: 1.0
Rule: Dry -> Rash_Guard, Swimming_Shirt, Confidence: 1.0
Rule: Dry, Rash_Guard -> Swimming_Shirt, Confidence: 1.0
Rule: Dry, Swimming_Shirt -> Rash_Guard, Confidence: 1.0
Rule: RunningShoe -> Tech_Pants, Confidence: 1.0
Rule: RunningShoe -> Dry_FitV-Nick, Confidence: 1.0
Rule: RunningShoe -> Tech_Pants, Dry_FitV-Nick, Confidence: 1.0
Rule: Dry_FitV-Nick, RunningShoe -> Tech_Pants, Confidence: 1.0
Rule: RunningShoe, Tech_Pants -> Dry_FitV-Nick, Confidence: 1.0
Rule: RunningShoe -> Modern_Pants, Confidence: 1.0
Rule: RunningShoe -> Dry_FitV-Nick, Modern_Pants, Confidence: 1.0
Rule: Dry_FitV-Nick, RunningShoe -> Modern_Pants, Confidence: 1.0
Rule: Modern_Pants, RunningShoe -> Dry_FitV-Nick, Confidence: 1.0
Rule: RunningShoe -> Tech_Pants, Modern_Pants, Confidence: 1.0
Rule: Modern_Pants, RunningShoe -> Tech_Pants, Confidence: 1.0
Rule: RunningShoe, Tech_Pants -> Modern_Pants, Confidence: 1.0
Rule: RunningShoe -> Tech_Pants, Dry_FitV-Nick, Modern_Pants, Confidence: 1.0
Rule: Dry_FitV-Nick, RunningShoe -> Tech_Pants, Modern_Pants, Confidence: 1.0
```

Association Rules Dataset2:

```
Association rules found with FP-Growth algorithm:
Rule: Kids Bedding ->  Bedspreads, Confidence: 1.0
Rule: Kids Bedding ->  Bedding Collections, Confidence: 1.0
Rule: Kids Bedding ->  Bedspreads,  Bedding Collections, Confidence: 1.0
Rule:  Bedding Collections, Kids Bedding ->  Bedspreads, Confidence: 1.0
Rule:  Bedspreads, Kids Bedding ->  Bedding Collections, Confidence: 1.0
Rule: Kids Bedding ->  Sheets, Confidence: 1.0
Rule: Kids Bedding ->  Sheets,  Bedding Collections, Confidence: 1.0
Rule:  Bedding Collections, Kids Bedding ->  Sheets, Confidence: 1.0
Rule:  Sheets, Kids Bedding ->  Bedding Collections, Confidence: 1.0
Rule: Kids Bedding ->  Sheets,  Bedspreads, Confidence: 1.0
Rule:  Bedspreads, Kids Bedding ->  Sheets, Confidence: 1.0
Rule:  Sheets, Kids Bedding ->  Bedspreads, Confidence: 1.0
Rule: Kids Bedding ->  Sheets,  Bedspreads,  Bedding Collections, Confidence: 1.0
Rule:  Bedding Collections, Kids Bedding ->  Sheets,  Bedspreads, Confidence: 1.0
Rule:  Bedspreads, Kids Bedding ->  Sheets,  Bedding Collections, Confidence: 1.0
Rule:  Sheets, Kids Bedding ->  Bedspreads,  Bedding Collections, Confidence: 1.0
Rule:  Bedding Collections,  Bedspreads, Kids Bedding ->  Sheets, Confidence: 1.0
Rule:  Bedding Collections,  Sheets, Kids Bedding ->  Bedspreads, Confidence: 1.0
Rule:  Bedspreads,  Sheets, Kids Bedding ->  Bedding Collections, Confidence: 1.0
Rule: Kids Bedding ->  Bed Skirts, Confidence: 1.0
```

Association Rules Dataset3:

```
Association rules found with FP-Growth algorithm:
Rule: Minivans -> Convertibles, Confidence: 0.5
Rule: Minivans -> Convertibles, SUV, Confidence: 0.5
Rule: Convertibles, Minivans -> SUV, Confidence: 1.0
Rule: Minivans, SUV -> Convertibles, Confidence: 0.5
Rule: Minivans -> Convertibles, Trucks, Confidence: 0.5
Rule: Convertibles, Minivans -> Trucks, Confidence: 1.0
Rule: Minivans, Trucks -> Convertibles, Confidence: 0.5
Rule: Minivans -> Convertibles, Trucks, SUV, Confidence: 0.5
Rule: Convertibles, Minivans -> SUV, Trucks, Confidence: 1.0
Rule: Minivans, SUV -> Convertibles, Trucks, Confidence: 0.5
Rule: Minivans, Trucks -> Convertibles, SUV, Confidence: 0.5
Rule: Convertibles, Minivans, SUV -> Trucks, Confidence: 1.0
Rule: Convertibles, Minivans, Trucks -> SUV, Confidence: 1.0
Rule: Minivans, SUV, Trucks -> Convertibles, Confidence: 0.5
Rule: Minivans -> Trucks, Confidence: 1.0
Rule: Trucks -> Minivans, Confidence: 0.5
Rule: Minivans -> SUV, Confidence: 1.0
Rule: Minivans -> Trucks, SUV, Confidence: 1.0
Rule: Trucks -> Minivans, SUV, Confidence: 0.5
Rule: Minivans, SUV -> Trucks, Confidence: 1.0
Rule: Minivans, Trucks -> SUV, Confidence: 1.0
Rule: SUV, Trucks -> Minivans, Confidence: 0.5
Rule: Coupe, Wagon -> Sedan, Confidence: 1.0
Rule: Sedan, Wagon -> Coupe, Confidence: 0.5
```

Association Rules Dataset4:

```
Association rules found with FP-Growth algorithm:
Rule: Counter_Strike, Halo -> Mario, Confidence: 0.5
Rule: Halo, Mario -> Counter_Strike, Confidence: 1.0
Rule: Halo -> Dota2, Confidence: 0.6666666666666666
Rule: Halo -> Counter_Strike, Confidence: 0.6666666666666666
Rule: Counter_Strike, Halo -> Dota2, Confidence: 0.5
Rule: Dota2, Halo -> Counter_Strike, Confidence: 0.5
Rule: Assasins_Creed, Mario -> Counter_Strike, Confidence: 1.0
Rule: Assasins_Creed, NFS -> Counter_Strike, Confidence: 1.0
Rule: Assasins_Creed, NFS -> Dota2, Confidence: 1.0
Rule: Assasins_Creed, NFS -> Counter_Strike, Dota2, Confidence: 1.0
Rule: Assasins_Creed, Counter_Strike, Dota2 -> NFS, Confidence: 0.5
Rule: Assasins_Creed, Counter_Strike, NFS -> Dota2, Confidence: 1.0
Rule: Assasins_Creed, Dota2, NFS -> Counter_Strike, Confidence: 1.0
Rule: Counter_Strike, Dota2, NFS -> Assasins_Creed, Confidence: 0.5
Rule: Assasins_Creed -> Skyrim, Confidence: 0.5
Rule: Assasins_Creed, Skyrim -> Counter_Strike, Confidence: 0.5
Rule: Assasins_Creed, Skyrim -> Counter_Strike, Dota2, Confidence: 0.5
Rule: Assasins_Creed, Counter_Strike, Dota2 -> Skyrim, Confidence: 0.5
Rule: Assasins_Creed, Counter_Strike, Skyrim -> Dota2, Confidence: 1.0
Rule: Assasins_Creed, Dota2, Skyrim -> Counter_Strike, Confidence: 0.5
Rule: Counter_Strike, Dota2, Skyrim -> Assasins_Creed, Confidence: 0.5
Rule: Assasins_Creed -> Skyrim, Dota2, Confidence: 0.5
Rule: Assasins_Creed, Dota2 -> Skyrim, Confidence: 0.6666666666666666
Rule: Assasins_Creed, Skyrim -> Dota2, Confidence: 1.0
```

Association Rules Dataset5:

```
Association rules found with FP-Growth algorithm:
Rule: cheetos -> popcorn, Confidence: 1.0
Rule: cheetos -> chips, Confidence: 1.0
Rule: cheetos -> popcorn, chips, Confidence: 1.0
Rule: cheetos, chips -> popcorn, Confidence: 1.0
Rule: cheetos, popcorn -> chips, Confidence: 1.0
Rule: cheetos -> chocolate, Confidence: 1.0
Rule: cheetos -> chips, chocolate, Confidence: 1.0
Rule: cheetos, chips -> chocolate, Confidence: 1.0
Rule: cheetos, chocolate -> chips, Confidence: 1.0
Rule: cheetos -> popcorn, chocolate, Confidence: 1.0
Rule: cheetos, chocolate -> popcorn, Confidence: 1.0
Rule: cheetos, popcorn -> chocolate, Confidence: 1.0
Rule: cheetos -> popcorn, chips, chocolate, Confidence: 1.0
Rule: cheetos, chips -> popcorn, chocolate, Confidence: 1.0
Rule: cheetos, chocolate -> popcorn, chips, Confidence: 1.0
Rule: cheetos, popcorn -> chocolate, chips, Confidence: 1.0
Rule: cheetos, chips, chocolate -> popcorn, Confidence: 1.0
Rule: cheetos, chips, popcorn -> chocolate, Confidence: 1.0
Rule: cheetos, chocolate, popcorn -> chips, Confidence: 1.0
Rule: chips, chocolate, popcorn -> cheetos, Confidence: 1.0
Rule: honey -> egg, Confidence: 1.0
Rule: honey -> whipped_cream, Confidence: 1.0
Rule: honey -> whipped_cream, egg, Confidence: 1.0
Rule: egg, honey -> whipped_cream, Confidence: 1.0
```

Output for Dataset 2 with support= 50 and confidence= 60:

```
Enter your choice: 2
User chose Grocery Store dataset
Enter the Minimum Support (in percentage): 50
Enter the Minimum Confidence (in percentage): 60

Frequent itemsets found with FP-Growth algorithm:
Itemset: ('Kids Bedding',), Support: 1
Itemset: (' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', ' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', ' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: ('Embroidered Bedspread',), Support: 1
Itemset: (' Shams', 'Embroidered Bedspread'), Support: 1
Itemset: ('Decorative Pillows',), Support: 3
```

Output for Dataset 2 with support= 30 and confidence= 40:

```
Enter your choice: 2
User chose Grocery Store dataset
Enter the Minimum Support (in percentage): 30
Enter the Minimum Confidence (in percentage): 40

Frequent itemsets found with FP-Growth algorithm:
Itemset: ('Kids Bedding',), Support: 1
Itemset: (' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bedding Collections', ' Bedspreads', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', ' Sheets', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedspreads', 'Kids Bedding'), Support: 1
Itemset: (' Bed Skirts', ' Bedding Collections', ' Bedspreads', 'Kids Bedding'), Support: 1
```

Brute Force Algorithm

Abstract:

In this project, I implemented Brute Force algorithm that generates frequent item pattern sets and association rules. The algorithm has user defined minimum support and confidence and generates useful insights based on the above parameters.

Introduction:

The Brute Force Algorithm generates all the frequent pattern item sets and association rules by enumerating all the possible item sets along with support count and confidence.

To sum up I have carried out the following steps in the algorithm:

- Loading the datasets, asking for user prompts which includes datasets, minimum support and confidence.
- Defined functions that calculate the frequent item sets and generate association rules.
- Generated association rules based on Frequent Pattern Item sets.
- Printing the association rules along with their support and confidence.
- Displaying the execution time of the program at the end.

Core Concepts and Principles:

Frequent Item Discovery:

The frequent item sets are the ones the items that are most frequently bought together. This help in understanding us the customer pattern, behaviour and how likely certain items are purchased.

Support and Confidence:

Support defines how frequently an item occurs whereas confidence describes the chance of items that are frequently bought together.

Association Rules:

Association rules help in discovering important patterns and relationships in items that are frequently together.

Project Workflow:

The implementation structure is as follows:

Data Loading and Preprocessing:

There are 5 datasets in total, each dataset containing transaction ID, and Transaction List (Items).

Minimum Support and Confidence:

The user is prompted to enter the minimum support and confidence value. This aids in eliminating less frequent item sets.

Candidate item sets Iteration:

The candidate item sets are iterated over to find frequent item sets which increase in size.

Support Calculation:

Support is calculated to identify how frequent an item occurs in the dataset. Item sets that satisfy the minimum support are retained, whereas the rest are discarded.

Confidence Calculation:

Calculating confidence will help us in determining the strength of a particular association rule.

Association Rules:

Association rules are printed when both the support and confidence values are satisfied.

Results and Evaluation:

The projects accuracy is determined by certain factors such as support, confidence, and association rules.

Conclusion:

The project demonstrates how Brute Force Algorithm can be used to generate frequent pattern item sets, and association rules.

Screenshots

---

1. **Reading from CSV File and Loading Transactions**

```
def read_transactions_from_csv(file_path):
    try:
        data = pd.read_csv(file_path)
        return data
    except FileNotFoundError:
        print("File not found. Please provide a valid file path.")
        return None

def load_transactions(dataList):
    Transactions = []
    df_items = dataList['TransactionList']
    comma_splitted_df = df_items.apply(lambda x: x.split(','))
    for i in comma_splitted_df:
        Transactions.append(i)
    return Transactions
```

2. **Generating Frequent Pattern Item sets**

```python
def generate_frequent_itemsets_brute_force(transactions, min_support):
    itemsets = set()
    for transaction in transactions:
        for item in transaction:
            itemsets.add(frozenset([item]))

    frequent_itemsets = {}
    total_transactions = len(transactions)
    for itemset in itemsets:
        count = 0
        for transaction in transactions:
            if itemset.issubset(transaction):
                count += 1
        support = count / total_transactions
        if support >= min_support:
            frequent_itemsets[itemset] = support

    return frequent_itemsets
```

3. Generating Association Rules

```python
def generate_association_rules(frequent_itemsets, min_confidence):
    association_rules = []
    for itemset in frequent_itemsets.keys():
        if len(itemset) > 1:
            rules_from_itemset = generate_rules_from_itemset(itemset, frequent_itemsets, min_conf
            association_rules.extend(rules_from_itemset)
    return association_rules
```

Outputs:

```
Rule: NFS, PUBG -> Counter_Strike, Confidence: 1.0
Rule: NFS, PUBG -> Counter_Strike, Dota2, Confidence: 1.0
Rule: Counter_Strike, NFS, PUBG -> Dota2, Confidence: 1.0
Rule: Dota2, NFS, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario, PUBG -> Dota2, Confidence: 1.0
Rule: Mario, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario, PUBG -> Counter_Strike, Dota2, Confidence: 1.0
Rule: Counter_Strike, Mario, PUBG -> Dota2, Confidence: 1.0
Rule: Dota2, Mario, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Dota2, PUBG, Skyrim -> Counter_Strike, Confidence: 1.0
Rule: Counter_Strike, Skyrim -> PUBG, Confidence: 0.6666666666666666
Rule: PUBG, Skyrim -> Counter_Strike, Confidence: 1.0
Rule: PUBG -> Dota2, Confidence: 0.75
Rule: PUBG -> Counter_Strike, Dota2, Confidence: 0.75
Rule: Counter_Strike, PUBG -> Dota2, Confidence: 0.75
Rule: Dota2, PUBG -> Counter_Strike, Confidence: 1.0
Rule: PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario -> Dota2, Confidence: 0.6
Rule: Mario -> Counter_Strike, Dota2, Confidence: 0.6
Rule: Counter_Strike, Mario -> Dota2, Confidence: 0.6
Rule: Dota2, Mario -> Counter_Strike, Confidence: 1.0
Rule: Mario -> Counter_Strike, Confidence: 1.0
Rule: Skyrim -> Dota2, Confidence: 0.6
```

Performance:

- **Execution time for Apriori:** 0.00036692619323730 47seconds
- **Execution time for FP-Growth:** 0.0012137889862060547seconds
- **Execution time for Brute Force:** 0.0003368854522705078seconds
- The Fastest Algorithm in terms of Execution is Brute Force.

Output Comparison:

- **FP-Growth**

```
Rule: NFS, PUBG -> Counter_Strike, Confidence: 1.0
Rule: NFS, PUBG -> Counter_Strike, Dota2, Confidence: 1.0
Rule: Counter_Strike, NFS, PUBG -> Dota2, Confidence: 1.0
Rule: Dota2, NFS, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario, PUBG -> Dota2, Confidence: 1.0
Rule: Mario, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario, PUBG -> Counter_Strike, Dota2, Confidence: 1.0
Rule: Counter_Strike, Mario, PUBG -> Dota2, Confidence: 1.0
Rule: Dota2, Mario, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Dota2, PUBG, Skyrim -> Counter_Strike, Confidence: 1.0
Rule: Counter_Strike, Skyrim -> PUBG, Confidence: 0.6666666666666666
Rule: PUBG, Skyrim -> Counter_Strike, Confidence: 1.0
Rule: PUBG -> Dota2, Confidence: 0.75
Rule: PUBG -> Counter_Strike, Dota2, Confidence: 0.75
Rule: Counter_Strike, PUBG -> Dota2, Confidence: 0.75
Rule: Dota2, PUBG -> Counter_Strike, Confidence: 1.0
Rule: PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario -> Dota2, Confidence: 0.6
Rule: Mario -> Counter_Strike, Dota2, Confidence: 0.6
Rule: Counter_Strike, Mario -> Dota2, Confidence: 0.6
Rule: Dota2, Mario -> Counter_Strike, Confidence: 1.0
Rule: Mario -> Counter_Strike, Confidence: 1.0
Rule: Skyrim -> Dota2, Confidence: 0.6
```

- **Apriori**

```
----- > Association With Support and Confidence: < -------

Rule:  Counter_Strike ==> Dota2
Support:  0.55
Confidence:  0.6875


Rule:  Dota2 ==> Counter_Strike
Support:  0.55
Confidence:  0.7857142857142858
```

- **Brute Force**

```
Rule: NFS, PUBG -> Counter_Strike, Confidence: 1.0
Rule: NFS, PUBG -> Counter_Strike, Dota2, Confidence: 1.0
Rule: Counter_Strike, NFS, PUBG -> Dota2, Confidence: 1.0
Rule: Dota2, NFS, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario, PUBG -> Dota2, Confidence: 1.0
Rule: Mario, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario, PUBG -> Counter_Strike, Dota2, Confidence: 1.0
Rule: Counter_Strike, Mario, PUBG -> Dota2, Confidence: 1.0
Rule: Dota2, Mario, PUBG -> Counter_Strike, Confidence: 1.0
Rule: Dota2, PUBG, Skyrim -> Counter_Strike, Confidence: 1.0
Rule: Counter_Strike, Skyrim -> PUBG, Confidence: 0.6666666666666666
Rule: PUBG, Skyrim -> Counter_Strike, Confidence: 1.0
Rule: PUBG -> Dota2, Confidence: 0.75
Rule: PUBG -> Counter_Strike, Dota2, Confidence: 0.75
Rule: Counter_Strike, PUBG -> Dota2, Confidence: 0.75
Rule: Dota2, PUBG -> Counter_Strike, Confidence: 1.0
Rule: PUBG -> Counter_Strike, Confidence: 1.0
Rule: Mario -> Dota2, Confidence: 0.6
Rule: Mario -> Counter_Strike, Dota2, Confidence: 0.6
Rule: Counter_Strike, Mario -> Dota2, Confidence: 0.6
Rule: Dota2, Mario -> Counter_Strike, Confidence: 1.0
Rule: Mario -> Counter_Strike, Confidence: 1.0
Rule: Skyrim -> Dota2, Confidence: 0.6
```

GitHub Link: https://github.com/NarmadaPeddi77/peddi_narmada_cs634_104