NAME: Suraj Bhardwaj
NJIT UCID: srb34
Email Address: srb34@njit.edu
Github Link: https://github.com/xxsurajbxx/CS634MidtermProject
Github Username: xxsurajbxx
10/7/2024
Professor: Yasser Abduallah
CS 634 101 Data Mining

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm Implementation In Retail Data Mining

Abstract:

In this research project, I use the Apriori, FP Growth, and brute force algorithms, basic data mining techniques, to investigate correlations in retail transactions. I evaluate the algorithm's efficacy and efficiency by putting it into practice and using a variety of data mining methods, methodologies, and principles. In addition, I create and design unique tools to construct a model that draws insightful conclusions from the transaction data.

Introduction:

A potent technique for locating hidden correlations and patterns in big datasets is data mining. The Algorithms studied in this research, well-known association rule mining methods, are the focus of this research; specifically how they can be applied in a retail setting. We examine the fundamental ideas and principles of data mining that underpin our methodology, such as the creation and presentation of association rules.

In this implementation, I created association rules and identified frequently occurring itemsets using a bespoke retail dataset and various association rule mining methods. The following were the primary steps in this process:

- Fabricating data which will soon be mined for association rules and saving the information in CSV files.

- The datasets are loaded from CSV files.

- Preserving item uniqueness and order by preprocessing the collection.

- Obtaining user feedback to determine confidence and minimal support thresholds.
- Candidate item sets are iteratively generated and often updated using the brute force, Apriori, and FP Growth algorithms.

Core Concepts and Principles:

Frequent Itemset Discovery:
The brute force algorithm, Apriori, and FP Growth algorithms are algorithms to find frequent itemsets from a transactional dataset. These itemsets represent items that are frequently purchased and have an association with each other. They help to recognize patterns in consumer preferences and behaviors.

Support and Confidence:
Two key metrics in data mining are support and confidence. Support measures how frequently an item or itemset occurs, while confidence assesses the likelihood of items being purchased together. These metrics guide our analysis.

Association Rules:
By determining strong association rules, I identify which items are commonly purchased together. These rules are instrumental for optimizing sales strategies, such as recommendations.

Project Workflow:

This project follows a structured methodology with several

Data Loading and Preprocessing:
Initially the transactional data is loaded in from a user selected retail store dataset. Every transaction consists of the items that one customer has bought in a transaction.

Determination of Minimum Support and Confidence:
The user is prompted to give input regarding the minimum support and confidence values to only produce significant patterns.

Iteration Through all the Possible Itemsets:
The brute force algorithm entails generating a set of candidate k-itemsets which is the set of all possible k length combinations of items. K iteratively gets larger, starting at 1 until none of the k length itemsets are frequent.

Support Count Calculation:
We determine the support of each potential itemset by calculating the number of transactions that contain the itemset. While some itemsets are discarded, those that fulfill the minimum support criterion are kept.

Confidence Calculation:
We assess the degree of association between itemsets by calculating the confidence of association rules. Carefully comparing the support values for individual products and itemsets is necessary in this stage.

Association Rule Generation:
Association rules that meet the standards for minimal confidence and minimum support are extracted. These guidelines provide insightful information about which products are frequently bought together.

Results and Evaluation:
The brute force method is evaluated for its effectiveness and efficiency compared to the Apriori and FP Growth algorithm implemented in the mlxtend python library. The runtime of the brute force algorithm is compared to these two algorithms, measured through a python library called timeit. It should be noted that in timing the algorithms, the Apriori and FP growth algorithms use a function from mlxtend to generate the association rules from the input frequent itemsets while the brute force algorithm uses a different function to do this. Their runtimes are quite similar so it does not affect the results very much however should still be noted.

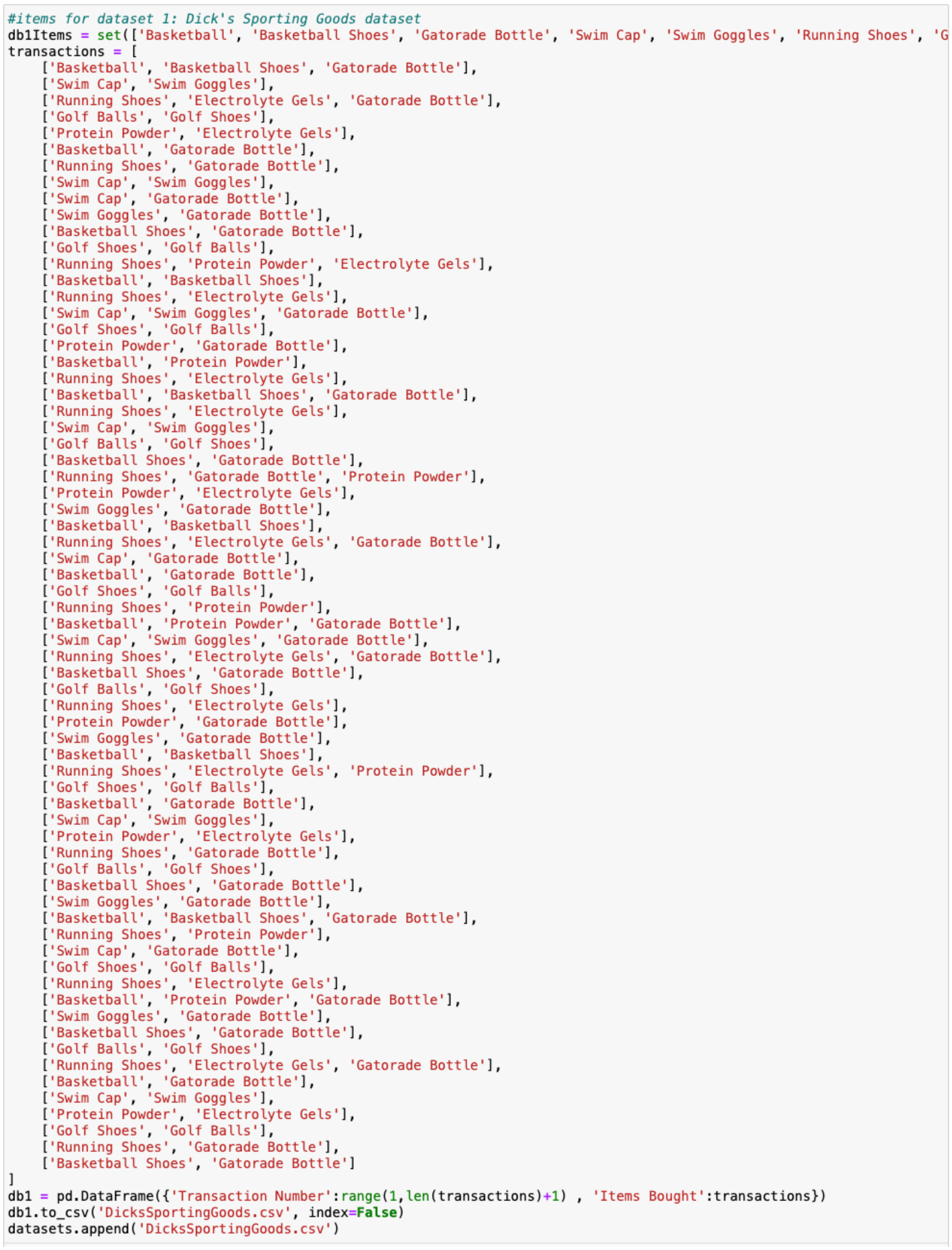
Conclusion:

In summary, our effort shows how data mining ideas, tenets, and techniques can be used. Using retail transaction data, we successfully applied the Apriori Algorithm to derive significant association rules. 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

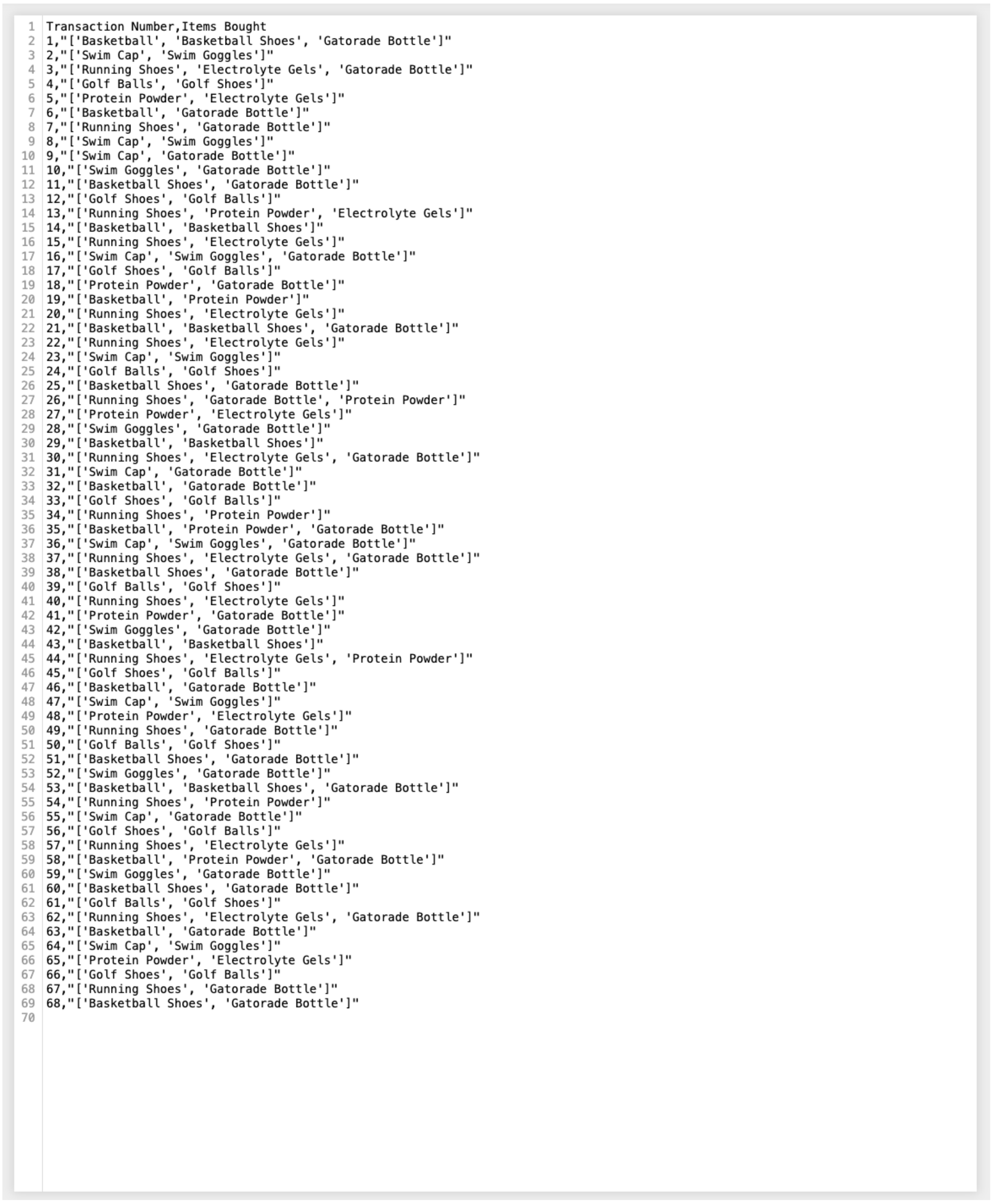
The first part of my program generates the csv files.

Here is the code that generates the csv files.



```python
#items for dataset 1: Dick's Sporting Goods dataset
db1Items = set(['Basketball', 'Basketball Shoes', 'Gatorade Bottle', 'Swim Cap', 'Swim Goggles', 'Running Shoes', 'G
transactions = [
    ['Basketball', 'Basketball Shoes', 'Gatorade Bottle'],
    ['Swim Cap', 'Swim Goggles'],
    ['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle'],
    ['Golf Balls', 'Golf Shoes'],
    ['Protein Powder', 'Electrolyte Gels'],
    ['Basketball', 'Gatorade Bottle'],
    ['Running Shoes', 'Gatorade Bottle'],
    ['Swim Cap', 'Swim Goggles'],
    ['Swim Cap', 'Gatorade Bottle'],
    ['Swim Goggles', 'Gatorade Bottle'],
    ['Basketball Shoes', 'Gatorade Bottle'],
    ['Golf Shoes', 'Golf Balls'],
    ['Running Shoes', 'Protein Powder', 'Electrolyte Gels'],
    ['Basketball', 'Basketball Shoes'],
    ['Running Shoes', 'Electrolyte Gels'],
    ['Swim Cap', 'Swim Goggles', 'Gatorade Bottle'],
    ['Golf Shoes', 'Golf Balls'],
    ['Protein Powder', 'Gatorade Bottle'],
    ['Basketball', 'Protein Powder'],
    ['Running Shoes', 'Electrolyte Gels'],
    ['Basketball', 'Basketball Shoes', 'Gatorade Bottle'],
    ['Running Shoes', 'Electrolyte Gels'],
    ['Swim Cap', 'Swim Goggles'],
    ['Golf Balls', 'Golf Shoes'],
    ['Basketball Shoes', 'Gatorade Bottle'],
    ['Running Shoes', 'Gatorade Bottle', 'Protein Powder'],
    ['Protein Powder', 'Electrolyte Gels'],
    ['Swim Goggles', 'Gatorade Bottle'],
    ['Basketball', 'Basketball Shoes'],
    ['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle'],
    ['Swim Cap', 'Gatorade Bottle'],
    ['Basketball', 'Gatorade Bottle'],
    ['Golf Shoes', 'Golf Balls'],
    ['Running Shoes', 'Protein Powder'],
    ['Basketball', 'Protein Powder', 'Gatorade Bottle'],
    ['Swim Cap', 'Swim Goggles', 'Gatorade Bottle'],
    ['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle'],
    ['Basketball Shoes', 'Gatorade Bottle'],
    ['Golf Balls', 'Golf Shoes'],
    ['Running Shoes', 'Electrolyte Gels'],
    ['Protein Powder', 'Gatorade Bottle'],
    ['Swim Goggles', 'Gatorade Bottle'],
    ['Basketball', 'Basketball Shoes'],
    ['Running Shoes', 'Electrolyte Gels', 'Protein Powder'],
    ['Golf Shoes', 'Golf Balls'],
    ['Basketball', 'Gatorade Bottle'],
    ['Swim Cap', 'Swim Goggles'],
    ['Protein Powder', 'Electrolyte Gels'],
    ['Running Shoes', 'Gatorade Bottle'],
    ['Golf Balls', 'Golf Shoes'],
    ['Basketball Shoes', 'Gatorade Bottle'],
    ['Swim Goggles', 'Gatorade Bottle'],
    ['Basketball', 'Basketball Shoes', 'Gatorade Bottle'],
    ['Running Shoes', 'Protein Powder'],
    ['Swim Cap', 'Gatorade Bottle'],
    ['Golf Shoes', 'Golf Balls'],
    ['Running Shoes', 'Electrolyte Gels'],
    ['Basketball', 'Protein Powder', 'Gatorade Bottle'],
    ['Swim Goggles', 'Gatorade Bottle'],
    ['Basketball Shoes', 'Gatorade Bottle'],
    ['Golf Balls', 'Golf Shoes'],
    ['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle'],
    ['Basketball', 'Gatorade Bottle'],
    ['Swim Cap', 'Swim Goggles'],
    ['Protein Powder', 'Electrolyte Gels'],
    ['Golf Shoes', 'Golf Balls'],
    ['Running Shoes', 'Gatorade Bottle'],
    ['Basketball Shoes', 'Gatorade Bottle']
]
db1 = pd.DataFrame({'Transaction Number':range(1,len(transactions)+1) , 'Items Bought':transactions})
db1.to_csv('DicksSportingGoods.csv', index=False)
datasets.append('DicksSportingGoods.csv')
```

Here is what the CSV files look like



```
Transaction Number,Items Bought
1,"['Basketball', 'Basketball Shoes', 'Gatorade Bottle']"
2,"['Swim Cap', 'Swim Goggles']"
3,"['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle']"
4,"['Golf Balls', 'Golf Shoes']"
5,"['Protein Powder', 'Electrolyte Gels']"
6,"['Basketball', 'Gatorade Bottle']"
7,"['Running Shoes', 'Gatorade Bottle']"
8,"['Swim Cap', 'Swim Goggles']"
9,"['Swim Cap', 'Gatorade Bottle']"
10,"['Swim Goggles', 'Gatorade Bottle']"
11,"['Basketball Shoes', 'Gatorade Bottle']"
12,"['Golf Shoes', 'Golf Balls']"
13,"['Running Shoes', 'Protein Powder', 'Electrolyte Gels']"
14,"['Basketball', 'Basketball Shoes']"
15,"['Running Shoes', 'Electrolyte Gels']"
16,"['Swim Cap', 'Swim Goggles', 'Gatorade Bottle']"
17,"['Golf Shoes', 'Golf Balls']"
18,"['Protein Powder', 'Gatorade Bottle']"
19,"['Basketball', 'Protein Powder']"
20,"['Running Shoes', 'Electrolyte Gels']"
21,"['Basketball', 'Basketball Shoes', 'Gatorade Bottle']"
22,"['Running Shoes', 'Electrolyte Gels']"
23,"['Swim Cap', 'Swim Goggles']"
24,"['Golf Balls', 'Golf Shoes']"
25,"['Basketball Shoes', 'Gatorade Bottle']"
26,"['Running Shoes', 'Gatorade Bottle', 'Protein Powder']"
27,"['Protein Powder', 'Electrolyte Gels']"
28,"['Swim Goggles', 'Gatorade Bottle']"
29,"['Basketball', 'Basketball Shoes']"
30,"['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle']"
31,"['Swim Cap', 'Gatorade Bottle']"
32,"['Basketball', 'Gatorade Bottle']"
33,"['Golf Shoes', 'Golf Balls']"
34,"['Running Shoes', 'Protein Powder']"
35,"['Basketball', 'Protein Powder', 'Gatorade Bottle']"
36,"['Swim Cap', 'Swim Goggles', 'Gatorade Bottle']"
37,"['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle']"
38,"['Basketball Shoes', 'Gatorade Bottle']"
39,"['Golf Balls', 'Golf Shoes']"
40,"['Running Shoes', 'Electrolyte Gels']"
41,"['Protein Powder', 'Gatorade Bottle']"
42,"['Swim Goggles', 'Gatorade Bottle']"
43,"['Basketball', 'Basketball Shoes']"
44,"['Running Shoes', 'Electrolyte Gels', 'Protein Powder']"
45,"['Golf Shoes', 'Golf Balls']"
46,"['Basketball', 'Gatorade Bottle']"
47,"['Swim Cap', 'Swim Goggles']"
48,"['Protein Powder', 'Electrolyte Gels']"
49,"['Running Shoes', 'Gatorade Bottle']"
50,"['Golf Balls', 'Golf Shoes']"
51,"['Basketball Shoes', 'Gatorade Bottle']"
52,"['Swim Goggles', 'Gatorade Bottle']"
53,"['Basketball', 'Basketball Shoes', 'Gatorade Bottle']"
54,"['Running Shoes', 'Protein Powder']"
55,"['Swim Cap', 'Gatorade Bottle']"
56,"['Golf Shoes', 'Golf Balls']"
57,"['Running Shoes', 'Electrolyte Gels']"
58,"['Basketball', 'Protein Powder', 'Gatorade Bottle']"
59,"['Swim Goggles', 'Gatorade Bottle']"
60,"['Basketball Shoes', 'Gatorade Bottle']"
61,"['Golf Balls', 'Golf Shoes']"
62,"['Running Shoes', 'Electrolyte Gels', 'Gatorade Bottle']"
63,"['Basketball', 'Gatorade Bottle']"
64,"['Swim Cap', 'Swim Goggles']"
65,"['Protein Powder', 'Electrolyte Gels']"
66,"['Golf Shoes', 'Golf Balls']"
67,"['Running Shoes', 'Gatorade Bottle']"
68,"['Basketball Shoes', 'Gatorade Bottle']"
```

In Part 2 I define some helper functions to make the brute force code more readable

```python
import itertools
import ast
```

```python
#Provided a some candidate itemsets as well as the transaction data and the minimum support, this function returns a
def findFrequentItemsets(transactions, itemsets, minSupport):
    frequentItemsets = {}
    for itemset in itemsets.keys():
            # counts the number of times that the itemset can be found in the transactions array
            for transaction in transactions:
                if isinstance(itemset, str):
                    if itemset in set(transaction):
                        itemsets[itemset] += 1
                elif set(itemset).issubset(set(transaction)):
                    itemsets[itemset] += 1
            # check to see if it is frequent or not. if it is then add the itemset to the frequent sets array
            support = itemsets[itemset]/len(transactions)
            if support >= minSupport:
                frequentItemsets[itemset] = support
    return frequentItemsets
```

```python
def generateAssociationRules(frequentItemsets, minConfidence):
    associationRules = set()
    for itemSet in frequentItemsets.keys():
        if not isinstance(itemSet, str):
            sets = []
            for i in range(1, len(itemSet)):
                sets.extend(list(itertools.combinations(itemSet, i)))
            for s in sets:
                confidence = 0
                if len(s)==1:
                    confidence = frequentItemsets[itemSet]/frequentItemsets[s[0]]
                else:
                    confidence = frequentItemsets[itemSet]/frequentItemsets[s]
                if confidence >= minConfidence:
                    associationRules.add(AssociationRule(s, tuple(set(itemSet)-set(list(s))), confidence, frequentIt
    return associationRules
```

I also define a class for the association rules

```python
#This is a class for the association rule. It is used by me after running the brute force algorithm by generate Asso
class AssociationRule:
    def __init__(self, a, b, confidence, support):
        self.a = a
        self.b = b
        self.confidence = confidence
        self.support = support
    def __str__(self):
        retstr = ''
        if len(self.a)==1:
            retstr+=self.a[0]
        else:
            retstr+=str(self.a)
        retstr += " implies "
        if len(self.b)==1:
            retstr+=self.b[0]
        else:
            retstr+=str(self.b)
        retstr+=f' with {self.confidence} confidence and {self.support} support'
        return retstr
```

In part 3 I import some additional classes

```python
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import timeit
```

I also gather user input

```python
#gather user input in regards to the dataset, minimum support and minimum confidence
for i in range(len(datasets)):
    print(f'({i+1}) {datasets[i][:-4]}')
d=8
while d>5 or d<1:
    d = int(input('Choose a number corresponding to a dataset: '))
file_path = f'./{datasets[d-1]}'
minimumSupport = 100
while minimumSupport>1 or minimumSupport<0:
    minimumSupport = float(input('Enter a minimum support between 0 and 1: '))
minimumConfidence = 100
while minimumConfidence>1 or minimumConfidence<0:
    minimumConfidence = float(input('Enter a minimum confidence between 0 and 1: '))
```

```
(1) DicksSportingGoods
(2) BestBuy
(3) HomeDepot
(4) Staples
(5) GameStop
Choose a number corresponding to a dataset: 1
Enter a minimum support between 0 and 1: 0.05
Enter a minimum confidence between 0 and 1: 0.5
```

The data is then read in

```python
#reads the chosen dataset in and formats the data
df = pd.read_csv(file_path)
transactions = df['Items Bought'].apply(ast.literal_eval).tolist()
rules = None
```

A wrapper function is used to time the functions runtimes

```python
#wrapper function so that the association rules can be returned and the algorithms can be timed
def wrapperFunction(func):
    global rules
    if func==1:
        rules = bruteForce(transactions, minimumSupport, minimumConfidence)
    elif func==2:
        frequentItemsets = apriori(formattedData, min_support=minimumSupport, use_colnames=True)
        rules = association_rules(frequentItemsets, metric="confidence", min_threshold=minimumConfidence)
    elif func==3:
        frequentItemsets = fpgrowth(formattedData, min_support=minimumSupport, use_colnames=True)
        rules = association_rules(frequentItemsets, metric="confidence", min_threshold=minimumConfidence)
```

The brute force algorithm is ran and timed

```python
#this runs, times and prints the output of the brute force algorithm
args = tuple([1])
executionTime = timeit.timeit(lambda: wrapperFunction(*args), number=1)
for rule in rules:
    print(rule)
print(f'\nExecution Time: {executionTime}')
```

```
Basketball implies Gatorade Bottle with 0.6923076923076924 confidence and 0.1323529411764706 support
Running Shoes implies Electrolyte Gels with 0.6470588235294118 confidence and 0.16176470588235295 support
Golf Balls implies Golf Shoes with 1.0 confidence and 0.16176470588235295 support
Basketball Shoes implies Basketball with 0.5 confidence and 0.08823529411764706 support
('Gatorade Bottle', 'Electrolyte Gels') implies Running Shoes with 1.0 confidence and 0.058823529411764705 support
Swim Cap implies Gatorade Bottle with 0.5 confidence and 0.07352941176470588 support
Basketball Shoes implies Gatorade Bottle with 0.75 confidence and 0.1323529411764706 support
Swim Goggles implies Gatorade Bottle with 0.5833333333333333 confidence and 0.10294117647058823 support
Swim Cap implies Swim Goggles with 0.7 confidence and 0.10294117647058823 support
('Gatorade Bottle', 'Running Shoes') implies Electrolyte Gels with 0.5 confidence and 0.058823529411764705 support
Electrolyte Gels implies Running Shoes with 0.7333333333333334 confidence and 0.16176470588235295 support
Golf Shoes implies Golf Balls with 1.0 confidence and 0.16176470588235295 support
Swim Goggles implies Swim Cap with 0.5833333333333333 confidence and 0.10294117647058823 support

Execution Time: 0.012043460999848321
```

The data is then reformatted for the Apriori and FP Growth algorithms

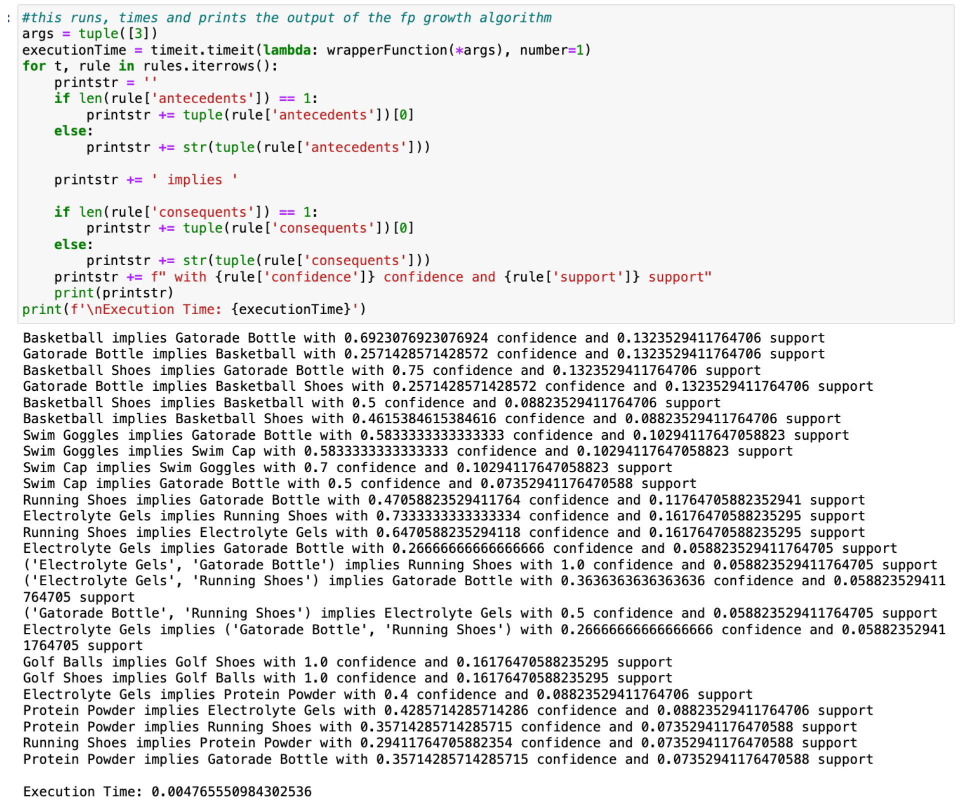
```
#formats the data for the apriori and fp tree algorithms
encoder = TransactionEncoder()
formattedData = pd.DataFrame(encoder.fit(transactions).transform(transactions), columns=encoder.columns_)
```

The Apriori algorithm is ran and timed, and the output is formatted clearly

```
#this runs, times and prints the output of the apriori algorithm
args = tuple([2])
executionTime = timeit.timeit(lambda: wrapperFunction(*args), number=1)
for t, rule in rules.iterrows():
    printstr = ''
    if len(rule['antecedents']) == 1:
        printstr += tuple(rule['antecedents'])[0]
    else:
        printstr += str(tuple(rule['antecedents']))

    printstr += ' implies '

    if len(rule['consequents']) == 1:
        printstr += tuple(rule['consequents'])[0]
    else:
        printstr += str(tuple(rule['consequents']))
    printstr += f" with {rule['confidence']} confidence and {rule['support']} support"
    print(printstr)
print(f'\nExecution Time: {executionTime}')
```

```
Basketball Shoes implies Basketball with 0.5 confidence and 0.08823529411764706 support
Basketball implies Gatorade Bottle with 0.6923076923076924 confidence and 0.1323529411764706 support
Basketball Shoes implies Gatorade Bottle with 0.75 confidence and 0.1323529411764706 support
Running Shoes implies Electrolyte Gels with 0.6470588235294118 confidence and 0.16176470588235295 support
Electrolyte Gels implies Running Shoes with 0.7333333333333334 confidence and 0.16176470588235295 support
Swim Cap implies Gatorade Bottle with 0.5 confidence and 0.07352941176470588 support
Swim Goggles implies Gatorade Bottle with 0.5833333333333333 confidence and 0.10294117647058823 support
Golf Balls implies Golf Shoes with 1.0 confidence and 0.16176470588235295 support
Golf Shoes implies Golf Balls with 1.0 confidence and 0.16176470588235295 support
Swim Goggles implies Swim Cap with 0.5833333333333333 confidence and 0.10294117647058823 support
Swim Cap implies Swim Goggles with 0.7 confidence and 0.10294117647058823 support
('Running Shoes', 'Gatorade Bottle') implies Electrolyte Gels with 0.5 confidence and 0.058823529411764705 support
('Electrolyte Gels', 'Gatorade Bottle') implies Running Shoes with 1.0 confidence and 0.058823529411764705 support

Execution Time: 0.008864415001880843
```

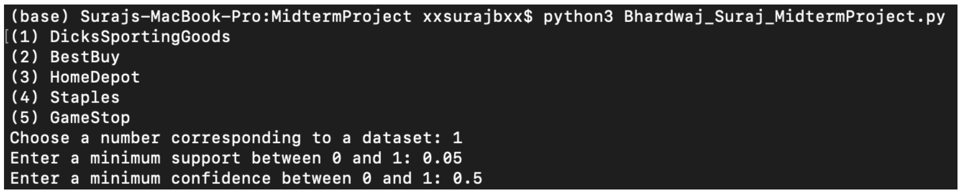
The FP growth algorithm is ran and timed, and the output is formatted clearly



```python
#this runs, times and prints the output of the fp growth algorithm
args = tuple([3])
executionTime = timeit.timeit(lambda: wrapperFunction(*args), number=1)
for t, rule in rules.iterrows():
    printstr = ''
    if len(rule['antecedents']) == 1:
        printstr += tuple(rule['antecedents'])[0]
    else:
        printstr += str(tuple(rule['antecedents']))

    printstr += ' implies '

    if len(rule['consequents']) == 1:
        printstr += tuple(rule['consequents'])[0]
    else:
        printstr += str(tuple(rule['consequents']))
    printstr += f" with {rule['confidence']} confidence and {rule['support']} support"
    print(printstr)
print(f'\nExecution Time: {executionTime}')
```

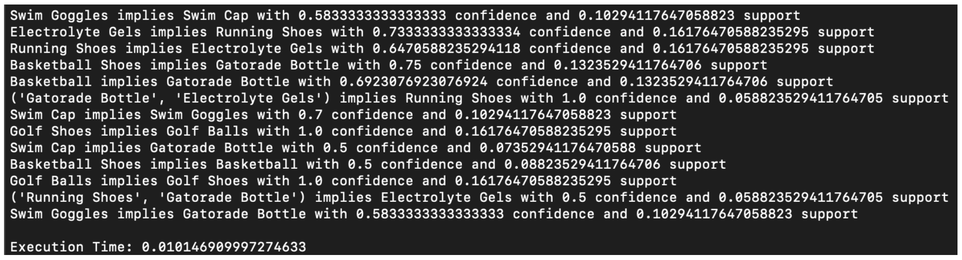
```
Basketball implies Gatorade Bottle with 0.6923076923076924 confidence and 0.1323529411764706 support
Gatorade Bottle implies Basketball with 0.2571428571428572 confidence and 0.1323529411764706 support
Basketball Shoes implies Gatorade Bottle with 0.75 confidence and 0.1323529411764706 support
Gatorade Bottle implies Basketball Shoes with 0.2571428571428572 confidence and 0.1323529411764706 support
Basketball Shoes implies Basketball with 0.5 confidence and 0.08823529411764706 support
Basketball implies Basketball Shoes with 0.4615384615384616 confidence and 0.08823529411764706 support
Swim Goggles implies Gatorade Bottle with 0.5833333333333333 confidence and 0.10294117647058823 support
Swim Goggles implies Swim Cap with 0.5833333333333333 confidence and 0.10294117647058823 support
Swim Cap implies Swim Goggles with 0.7 confidence and 0.10294117647058823 support
Swim Cap implies Gatorade Bottle with 0.5 confidence and 0.07352941176470588 support
Running Shoes implies Gatorade Bottle with 0.47058823529411764 confidence and 0.11764705882352941 support
Electrolyte Gels implies Running Shoes with 0.7333333333333334 confidence and 0.16176470588235295 support
Running Shoes implies Electrolyte Gels with 0.6470588235294118 confidence and 0.16176470588235295 support
Electrolyte Gels implies Gatorade Bottle with 0.26666666666666666 confidence and 0.058823529411764705 support
('Electrolyte Gels', 'Gatorade Bottle') implies Running Shoes with 1.0 confidence and 0.058823529411764705 support
('Electrolyte Gels', 'Running Shoes') implies Gatorade Bottle with 0.3636363636363636 confidence and 0.058823529411
764705 support
('Gatorade Bottle', 'Running Shoes') implies Electrolyte Gels with 0.5 confidence and 0.058823529411764705 support
Electrolyte Gels implies ('Gatorade Bottle', 'Running Shoes') with 0.26666666666666666 confidence and 0.05882352941
1764705 support
Golf Balls implies Golf Shoes with 1.0 confidence and 0.16176470588235295 support
Golf Shoes implies Golf Balls with 1.0 confidence and 0.16176470588235295 support
Electrolyte Gels implies Protein Powder with 0.4 confidence and 0.08823529411764706 support
Protein Powder implies Electrolyte Gels with 0.4285714285714286 confidence and 0.08823529411764706 support
Protein Powder implies Running Shoes with 0.35714285714285715 confidence and 0.07352941176470588 support
Running Shoes implies Protein Powder with 0.29411764705882354 confidence and 0.07352941176470588 support
Protein Powder implies Gatorade Bottle with 0.35714285714285715 confidence and 0.07352941176470588 support

Execution Time: 0.004765550984302536
```

Here is what the program looks like when ran in the terminal.



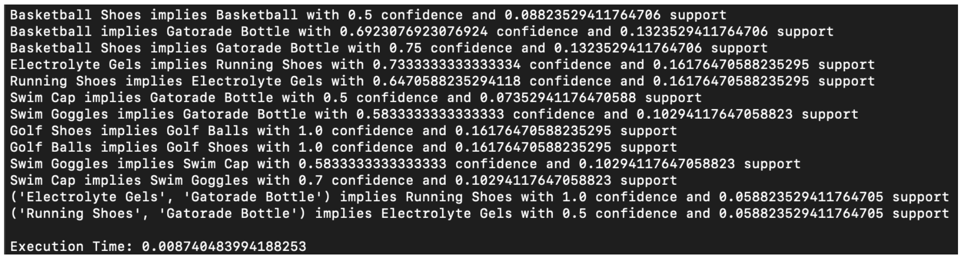
```
(base) Surajs-MacBook-Pro:MidtermProject xxsurajbxx$ python3 Bhardwaj_Suraj_MidtermProject.py
(1) DicksSportingGoods
(2) BestBuy
(3) HomeDepot
(4) Staples
(5) GameStop
Choose a number corresponding to a dataset: 1
Enter a minimum support between 0 and 1: 0.05
Enter a minimum confidence between 0 and 1: 0.5
```

Here is the output from the brute force algorithm.



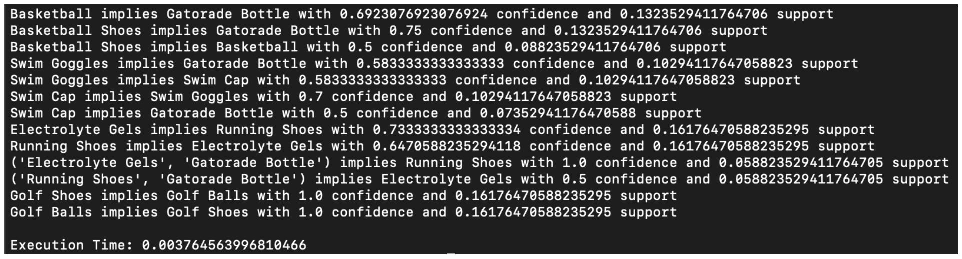
```
Swim Goggles implies Swim Cap with 0.5833333333333333 confidence and 0.10294117647058823 support
Electrolyte Gels implies Running Shoes with 0.7333333333333334 confidence and 0.16176470588235295 support
Running Shoes implies Electrolyte Gels with 0.6470588235294118 confidence and 0.16176470588235295 support
Basketball Shoes implies Gatorade Bottle with 0.75 confidence and 0.1323529411764706 support
Basketball implies Gatorade Bottle with 0.6923076923076924 confidence and 0.1323529411764706 support
('Gatorade Bottle', 'Electrolyte Gels') implies Running Shoes with 1.0 confidence and 0.058823529411764705 support
Swim Cap implies Swim Goggles with 0.7 confidence and 0.10294117647058823 support
Golf Shoes implies Golf Balls with 1.0 confidence and 0.16176470588235295 support
Swim Cap implies Gatorade Bottle with 0.5 confidence and 0.07352941176470588 support
Basketball Shoes implies Basketball with 0.5 confidence and 0.08823529411764706 support
Golf Balls implies Golf Shoes with 1.0 confidence and 0.16176470588235295 support
('Running Shoes', 'Gatorade Bottle') implies Electrolyte Gels with 0.5 confidence and 0.058823529411764705 support
Swim Goggles implies Gatorade Bottle with 0.5833333333333333 confidence and 0.10294117647058823 support

Execution Time: 0.010146909997274633
```

Here is the output from the Apriori algorithm



```
Basketball Shoes implies Basketball with 0.5 confidence and 0.08823529411764706 support
Basketball implies Gatorade Bottle with 0.6923076923076924 confidence and 0.1323529411764706 support
Basketball Shoes implies Gatorade Bottle with 0.75 confidence and 0.1323529411764706 support
Electrolyte Gels implies Running Shoes with 0.7333333333333334 confidence and 0.16176470588235295 support
Running Shoes implies Electrolyte Gels with 0.6470588235294118 confidence and 0.16176470588235295 support
Swim Cap implies Gatorade Bottle with 0.5 confidence and 0.07352941176470588 support
Swim Goggles implies Gatorade Bottle with 0.5833333333333333 confidence and 0.10294117647058823 support
Golf Shoes implies Golf Balls with 1.0 confidence and 0.16176470588235295 support
Golf Balls implies Golf Shoes with 1.0 confidence and 0.16176470588235295 support
Swim Goggles implies Swim Cap with 0.5833333333333333 confidence and 0.10294117647058823 support
Swim Cap implies Swim Goggles with 0.7 confidence and 0.10294117647058823 support
('Electrolyte Gels', 'Gatorade Bottle') implies Running Shoes with 1.0 confidence and 0.058823529411764705 support
('Running Shoes', 'Gatorade Bottle') implies Electrolyte Gels with 0.5 confidence and 0.058823529411764705 support

Execution Time: 0.008740483994188253
```

Here is the output from the FP-Growth algorithm



```
Basketball implies Gatorade Bottle with 0.6923076923076924 confidence and 0.1323529411764706 support
Basketball Shoes implies Gatorade Bottle with 0.75 confidence and 0.1323529411764706 support
Basketball Shoes implies Basketball with 0.5 confidence and 0.08823529411764706 support
Swim Goggles implies Gatorade Bottle with 0.5833333333333333 confidence and 0.10294117647058823 support
Swim Goggles implies Swim Cap with 0.5833333333333333 confidence and 0.10294117647058823 support
Swim Cap implies Swim Goggles with 0.7 confidence and 0.10294117647058823 support
Swim Cap implies Gatorade Bottle with 0.5 confidence and 0.07352941176470588 support
Electrolyte Gels implies Running Shoes with 0.7333333333333334 confidence and 0.16176470588235295 support
Running Shoes implies Electrolyte Gels with 0.6470588235294118 confidence and 0.16176470588235295 support
('Electrolyte Gels', 'Gatorade Bottle') implies Running Shoes with 1.0 confidence and 0.058823529411764705 support
('Running Shoes', 'Gatorade Bottle') implies Electrolyte Gels with 0.5 confidence and 0.058823529411764705 support
Golf Shoes implies Golf Balls with 1.0 confidence and 0.16176470588235295 support
Golf Balls implies Golf Shoes with 1.0 confidence and 0.16176470588235295 support

Execution Time: 0.003764563996810466
```

Other

The source code (.py file) and data sets (.csv files) will be attached to the zip file. *Link to Git Repository*

https://github.com/mahumabid/Apriori_Algorithm