Diagram

Description automatically generated

Course: CS634 - 002

Professor: Jason Wang

Student: Shanmuga Sankar Balamurugan

UCID: sb387

Email: [sb387@njit.edu](mailto:sb387@njit.edu)

Subject: Mid-Term Project Report

Topic: Association Rule Mining

**ASSOCIATION RULE MINING ALGORITHMS**



Contents

[1. Introduction 4](#_Toc96704463)

[2. Description 4](#_Toc96704464)

[2.1. Apriori Algorithm 4](#_Toc96704465)

[2.2. Brute Force Algorithm 4](#_Toc96704466)

[2.3. Database 4](#_Toc96704467)

[3. Input Databases 4](#_Toc96704468)

[3.1. Database 1 4](#_Toc96704469)

[3.2. Database 2 5](#_Toc96704470)

[3.3. Database 3 5](#_Toc96704471)

[3.4. Database 4 6](#_Toc96704472)

[3.5. Database 5 6](#_Toc96704473)

[4. Apriori Algorithm 7](#_Toc96704474)

[4.1. Import Libraries 7](#_Toc96704475)

[4.2. Read Data Function 7](#_Toc96704476)

[4.3. Frequency Function 7](#_Toc96704477)

[4.4. Support Function 8](#_Toc96704478)

[4.5. Confidence Function 9](#_Toc96704479)

[4.6. Apriori Algorithm Function 9](#_Toc96704480)

[5. Brute Force Algorithm 11](#_Toc96704481)

[5.1. Import Libraries 11](#_Toc96704482)

[5.2. Read Data Function 11](#_Toc96704483)

[5.3. Frequency Function 11](#_Toc96704484)

[5.4. Support Function 12](#_Toc96704485)

[5.5. Confidence Function 13](#_Toc96704486)

[5.6. Brute Force Algorithm Function 14](#_Toc96704487)

[6. Results 15](#_Toc96704488)

[6.1. Apriori vs. Brute Force – Database 1 15](#_Toc96704489)

[6.1.1. Apriori Results 15](#_Toc96704490)

[6.1.2. Brute Force Results 16](#_Toc96704491)

[6.2. Apriori vs Brute Force – Database 2 17](#_Toc96704492)

[6.2.1. Apriori Results 17](#_Toc96704493)

[6.2.2. Brute Force Results 18](#_Toc96704494)

[6.3. Apriori vs. Brute Force – Database 3 19](#_Toc96704495)

[6.3.1. Apriori Results 19](#_Toc96704496)

[6.3.2. Brute Force Results 20](#_Toc96704497)

[6.4. Apriori vs. Brute Force – Database 4 21](#_Toc96704498)

[6.4.1. Apriori Results 21](#_Toc96704499)

[6.4.2. Brute Force Results 22](#_Toc96704500)

[6.5. Apriori vs Brute Force – Database 5 23](#_Toc96704501)

[6.5.1. Apriori Results 23](#_Toc96704502)

[6.5.2. Brute Force Results 24](#_Toc96704503)

[7. Conclusion 24](#_Toc96704504)

# Introduction

The goal of the project is to apply and compare association rule technique such as Apriori algorithm against Brute Force algorithm, to determine efficiency as a matter of time, i.e., how much faster is Apriori to Brute Force algorithm.

# Description

## Apriori Algorithm

Find the sets which meet minimum support requirement, starting from 1-itemsets to k-itemsets; if j-itemset is not frequent we do not need to consider any of its supersets. The identified frequent sets are used to generate association rules.

## Brute Force Algorithm

Generate all 1-itemsets and 2-itemsets and check if they are frequent. Then generate 3-itemsets; follow this procedure to generate k-itemsets, and flag all that meets the minimum support requirement during the entire process. These frequent sets are used to generate association rules.

## Database

We create five databases in a csv file format, each with 20 transactions containing between 15-20 items (overall 30 unique items), commonly seen on platforms such as Amazon, K-Mart, etc. This transaction data is used as input on which our algorithms are applied. We limit the no. items per database as execution of Brute Force algorithm is not feasible; we are more interested in executing Apriori and comparing their efficiencies.

# Input Databases

The images of the five databases with 20 transactions with 15-20 items will be found below.

## Database 1

Table

Description automatically generated

## Database 2

A screenshot of a computer

Description automatically generated with medium confidence

## Database 3

Text, table

Description automatically generated

## Database 4

Text, table

Description automatically generated

## Database 5

Text

Description automatically generated with medium confidence

# Apriori Algorithm

The code written for the executing Apriori Algorithm.

## Import Libraries

List of packages used in the program

|  |
| --- |
| **import** pandas **as** pd  **import** numpy **as** np  **from** itertools **import** chain**,**combinations  **import** time |

## Read Data Function

Function to read the csv file containing transaction data

|  |
| --- |
| **def** readData**(**path**):**  '''  Function to read csv file containing the transactions    Parameters:-  path - Location of the input file    '''  transactionData **=** pd**.**read\_csv**(**path**,** header **=** **None)**  **return** transactionData |

## Frequency Function

Function to identify the frequent 1-itemsets in the data

|  |
| --- |
| **def** frequency**(**transactionData**,**support**):**  '''Fucntion to determine the frequent items in the transaction database    Parameters:-  transactionData - Single column dataframe containing all the transactions  support - User determined support level for generating itemsets    '''    ## Extract transaction information from dataframe to a list  Transactions **=** **[]**  **for** i **in** **range(len(**transactionData**)):**  Transactions**.**append**(**transactionData**[**0**][**i**].**split**(**","**))**    ## Initialize dictionaries to store frequent itemsets  FrequentItemSets **=** **{}**  AprioriResults **=** **{}**    ## Calculate total no. of occurrences of items among all transactions  **for** i**,**items **in** **enumerate(**Transactions**):**  **for** j **in** **range(len(**items**)):**  **if** items**[**j**]** **in** FrequentItemSets**:**  FrequentItemSets**[**items**[**j**]]** **+=** 1  **else:**  FrequentItemSets**[**items**[**j**]]** **=** 1  association **=** **[]**  nonFrequent **=** **[]**    ## Assign items to the lists depending on whether they meet the minimum support  **for** i **in** FrequentItemSets**:**  **if** FrequentItemSets**[**i**]/len(**Transactions**)** **>=** support**:**  association**.**append**(**i**)**  **else:**  nonFrequent**.**append**(**i**)**    ## Delete all items which do not meet the minimum support requirement  **for** i **in** nonFrequent**:**  **del** FrequentItemSets**[**i**]**  n\_combinations **=** **list(**combinations**(**association**,**2**))**  **return** n\_combinations**,**Transactions**,**FrequentItemSets**,**AprioriResults |

## Support Function

Function to get all frequent itemsets which meet minimum support level, and break when no superset can be created based on existing frequent itemsets

|  |
| --- |
| **def** support\_level**(**n\_combinations**,**support**,**Transactions**,**AprioriResults**):**  '''  Function to determine all itemset combinations which meet the minimum support    Parameters:-  n\_combinations - List of all itemset combinations meeting the support requirement  support - User determined support level for generating itemsets  Transaction - List containing all transactions  AprioriResults - Dictionary containing all itemsets meeting support level    '''    ## Loop to generate itemsets as long as they meet support requirements  **while** **len(**n\_combinations**)** **>** 0**:**  itemSets **=** **[]**  **for** i **in** n\_combinations**:**  count **=** 0  **for** j **in** **range(len(**Transactions**)):**  **if** **set(**i**).**issubset**(**Transactions**[**j**])** **==** **True:** ## for itemsets present in transactions, increase count  count **+=** 1  **if** count**/len(**Transactions**)** **>=** support**:** ## if itemset meets minimum support add it to dictionary  itemSets**.**append**(**i**)**  AprioriResults**[tuple(sorted(**i**))]** **=** count    addition **=** **[]**    ## Loop to generate (n+1)th itemset  **for** i **in** **range(len(**itemSets**)):**  j **=** **len(**itemSets**)** **-** 1  **while** j **>** i**:** ## Generating (n+1)th itemset, eg. (A,B),(A,C) => (A,B,C)  **if** **len(list(set(**itemSets**[**i**])** **-** **set(**itemSets**[**j**])))** **==** 1 **and** **set(**itemSets**[**i**]).**intersection**(set(**itemSets**[**j**]))** **!=** **set():**  addition**.**append**(tuple(sorted(set(**itemSets**[**i**]).**intersection**(set(**itemSets**[**j**])).**union**(set(**itemSets**[**i**]).**symmetric\_difference**(set(**itemSets**[**j**]))))))**  j **-=** 1  ## List of new itemsets for which support levels need to be checked  n\_combinations **=** **list(frozenset(sorted(**sub**))** **for** sub **in** **set(sorted(**addition**)))**    **return** AprioriResults |

## Confidence Function

Function to print association rules of frequent itemsets which meet the minimum confidence level

|  |
| --- |
| **def** confidence\_level**(**AprioriResults**,**FrequentItemSets**,**Transactions**,**confidence**,**support**):**  '''  Function to generate the support and confidence levels of itemsets which meet user defined requirements    Parameters:-  ApriroiResults - Dictionary containing frequent itemsets  FrequentItemSets - Dictionary containing frequent items  Transactions - List containing all transactions  confidence - User determined confidence level for generating itemsets  support - User determined support level for generating itemsets    '''  ## iterate over itemsets which are greater than minimum support level  **for** i**,**combination **in** **enumerate(**AprioriResults**):**  ## iterate to get association of 1 item to the rest of the set  **for** j **in** combination**:**  ## for 2-itemsets  **if** **len(set(**combination**)** **-** **set((**j**,)))** **==** 1**:**  BaseGroup **=** **list(set(**combination**)** **-** **set((**j**,)),)[**0**]**  ## calculate support and confidence level  confidenceCalculation **=** AprioriResults**.**get**(**combination**)/**FrequentItemSets**.**get**(**BaseGroup**)**  supportLevel **=** AprioriResults**.**get**(**combination**)/len(**Transactions**)**  ## print valid associations  **if** confidenceCalculation **>=** confidence **and** supportLevel **>=** support**:**  **print(set((**BaseGroup**,)),**"=>"**,**"{"**,**j**,**"}"**,**"("**,**supportLevel**\***100**,**"%,"**,**confidenceCalculation**\***100**,**"%"**,**")"**)**  ## for n-itemsets, where n > 2  **else:**  BaseGroup **=** **tuple(set(**combination**)** **-** **set((**j**,)))**  ## calculate support and confidence level  confidenceCalculation **=** AprioriResults**.**get**(**combination**)/**AprioriResults**.**get**(tuple(sorted(**BaseGroup**)))**  supportLevel **=** AprioriResults**.**get**(**combination**)/len(**Transactions**)**  ## print valid associations  **if** confidenceCalculation **>=** confidence **and** supportLevel **>=** support**:**  **print(set(**BaseGroup**),**"=>"**,**"{"**,**j**,**"}"**,**"("**,**supportLevel**\***100**,**"%,"**,**confidenceCalculation**\***100**,**"%"**,**")"**)**    FrequentItemSets**[**combination**]** **=** AprioriResults**.**get**(**combination**)** |

## Apriori Algorithm Function

Function which takes user input of support and confidence level for transaction data, on which Apriori Algorithm is executed.

|  |
| --- |
| **def** Apriori**(**support**,**confidence**,**path**):**  '''  Function to execute Apriori Algorithim    Parameters:-  support - User determined support level for generating itemsets  confidence - User determined confidence level for generating itemsets  path - Location of the input file    '''    transactionData **=** readData**(**path**)**  n\_combinations**,**Transactions**,**FrequentItemSets**,**AprioriResults **=** frequency**(**transactionData**,**support**)**  AprioriResults **=** support\_level**(**n\_combinations**,**support**,**Transactions**,**AprioriResults**)**  **return** confidence\_level**(**AprioriResults**,**FrequentItemSets**,**Transactions**,**confidence**,**support**)** |

# Brute Force Algorithm

Code written for executing Brute Force Algorithm

## Import Libraries

List of packages used in the program

|  |
| --- |
| **import** pandas **as** pd  **import** numpy **as** np  **from** itertools **import** chain**,**combinations  **import** time |

## Read Data Function

Function to read the csv file containing transaction data

|  |
| --- |
| **def** readData**(**path**):**  '''  Function to read csv file containing the transactions    Parameters:-  path - Location of the input file    '''  transactionData **=** pd**.**read\_csv**(**path**,** header **=** **None)**  **return** transactionData |

## Frequency Function

Function to identify the frequent 1-itemsets in the data

|  |
| --- |
| **def** frequency**(**transactionData**,**support**):**  '''Fucntion to determine the frequent items in the transaction database    Parameters:-  transactionData - Single column dataframe containing all the transactions  support - User determined support level for generating itemsets    '''    ## Extract transaction information from dataframe to a list  Transactions **=** **[]**  **for** i **in** **range(len(**transactionData**)):**  Transactions**.**append**(**transactionData**[**0**][**i**].**split**(**","**))**    ## Initialize dictionaries to store frequent itemsets  FrequentItemSets **=** **{}**  BruteForce **=** **{}**    ## Calculate total no. of occurrences of items among all transactions  **for** i**,**items **in** **enumerate(**Transactions**):**  **for** j **in** **range(len(**items**)):**  **if** items**[**j**]** **in** FrequentItemSets**:**  FrequentItemSets**[**items**[**j**]]** **+=** 1  **else:**  FrequentItemSets**[**items**[**j**]]** **=** 1  association **=** **[]**  nonFrequent **=** **[]**    ## Assign items to the lists depending on whether they meet the minimum support  **for** i **in** FrequentItemSets**:**  **if** FrequentItemSets**[**i**]/len(**Transactions**)** **>=** support**:**  association**.**append**(**i**)**  **else:**  nonFrequent**.**append**(**i**)**    ## List of all items  **for** x **in** nonFrequent**:**  association**.**append**(**x**)**    ## Delete all items which do not meet the minimum support requirement  **for** i **in** nonFrequent**:**  **del** FrequentItemSets**[**i**]**  n\_combinations **=** **list(**combinations**(**association**,**2**))**  **return** n\_combinations**,**Transactions**,**FrequentItemSets**,**BruteForce |

## Support Function

Function to enumerate all possible itemsets and give all frequent itemsets which meet minimum support level

|  |
| --- |
| **def** support\_level**(**n\_combinations**,**support**,**Transactions**,**BruteForce**):**  '''  Function to determine all itemset combinations which meet the minimum support    Parameters:-  n\_combinations - List of all itemset combinations meeting the support requirement  support - User determined support level for generating itemsets  Transaction - List containing all transactions  BruteForce - Dictionary containing all itemsets meeting support level    '''    ## Loop to generate itemsets as long as they meet support requirements  **while** **len(**n\_combinations**)** **>** 0**:**  itemSets **=** **[]**  **for** i **in** n\_combinations**:**  count **=** 0  **for** j **in** **range(len(**Transactions**)):**  **if** **set(**i**).**issubset**(**Transactions**[**j**])** **==** **True:** ## for itemsets present in transactions, increase count  count **+=** 1  **if** count**/len(**Transactions**)** **>=** support**:** ## if itemset meets minimum support add it to dictionary  itemSets**.**append**(**i**)**  BruteForce**[tuple(sorted(**i**))]** **=** count    itemSets **=** n\_combinations    addition **=** **[]**    ## Loop to generate (n+1)th itemset  **for** i **in** **range(len(**itemSets**)):**  j **=** **len(**itemSets**)** **-** 1  **while** j **>** i**:** ## Generating (n+1)th itemset, eg. (A,B),(A,C) => (A,B,C)  **if** **len(list(set(**itemSets**[**i**])** **-** **set(**itemSets**[**j**])))** **==** 1 **and** **set(**itemSets**[**i**]).**intersection**(set(**itemSets**[**j**]))** **!=** **set():**  addition**.**append**(tuple(sorted(set(**itemSets**[**i**]).**intersection**(set(**itemSets**[**j**])).**union**(set(**itemSets**[**i**]).**symmetric\_difference**(set(**itemSets**[**j**]))))))**  j **-=** 1  ## List of new itemsets for which support levels need to be checked  n\_combinations **=** **list(frozenset(sorted(**sub**))** **for** sub **in** **set(sorted(**addition**)))**    **return** BruteForce |

## Confidence Function

Function to print association rules of frequent itemsets which meet the minimum confidence level

|  |
| --- |
| **def** confidence\_level**(**BruteForce**,**FrequentItemSets**,**Transactions**,**confidence**,**support**):**  '''  Function to generate the support and confidence levels of itemsets which meet user defined requirements    Parameters:-  BruteForce - Dictionary containing frequent itemsets  FrequentItemSets - Dictionary containing frequent items  Transactions - List containing all transactions  confidence - User determined confidence level for generating itemsets  support - User determined support level for generating itemsets    '''  ## iterate over itemsets which are greater than minimum support level  **for** i**,**combination **in** **enumerate(**BruteForce**):**  ## iterate to get association of 1 item to the rest of the set  **for** j **in** combination**:**  ## for 2-itemsets  **if** **len(set(**combination**)** **-** **set((**j**,)))** **==** 1**:**  BaseGroup **=** **list(set(**combination**)** **-** **set((**j**,)),)[**0**]**  ## calculate support and confidence level  confidenceCalculation **=** BruteForce**.**get**(**combination**)/**FrequentItemSets**.**get**(**BaseGroup**)**  supportLevel **=** BruteForce**.**get**(**combination**)/len(**Transactions**)**  ## print valid associations  **if** confidenceCalculation **>=** confidence **and** supportLevel **>=** support**:**  **print(set((**BaseGroup**,)),**"=>"**,**"{"**,**j**,**"}"**,**"("**,**supportLevel**\***100**,**"%,"**,**confidenceCalculation**\***100**,**"%"**,**")"**)**  ## for n-itemsets, where n > 2  **else:**  BaseGroup **=** **tuple(set(**combination**)** **-** **set((**j**,)))**  ## calculate support and confidence level  confidenceCalculation **=** BruteForce**.**get**(**combination**)/**BruteForce**.**get**(tuple(sorted(**BaseGroup**)))**  supportLevel **=** BruteForce**.**get**(**combination**)/len(**Transactions**)**  ## print valid associations  **if** confidenceCalculation **>=** confidence **and** supportLevel **>=** support**:**  **print(set(**BaseGroup**),**"=>"**,**"{"**,**j**,**"}"**,**"("**,**supportLevel**\***100**,**"%,"**,**confidenceCalculation**\***100**,**"%"**,**")"**)**    FrequentItemSets**[**combination**]** **=** BruteForce**.**get**(**combination**)** |

## Brute Force Algorithm Function

Function which takes user input of support and confidence level for transaction data, on which Brute Force Algorithm is executed.

|  |
| --- |
| **def** BruteForce**(**support**,**confidence**,**path**):**  '''  Function to execute Brute Force Algorithim    Parameters:-  support - User determined support level for generating itemsets  confidence - User determined confidence level for generating itemsets  path - Location of the input file    '''    transactionData **=** readData**(**path**)**  n\_combinations**,**Transactions**,**FrequentItemSets**,**AprioriResults **=** frequency**(**transactionData**,**support**)**  AprioriResults **=** support\_level**(**n\_combinations**,**support**,**Transactions**,**AprioriResults**)**  **return** confidence\_level**(**AprioriResults**,**FrequentItemSets**,**Transactions**,**confidence**,**support**)** |

# Results

The association rule and time taken to execute both algorithms at various support and confidence levels.

## Apriori vs. Brute Force – Database 1

At 15% support and 70% confidence Apriori and Brute Force execute at 0.0089s and 1629.50s respectively. At 10% support and 70% confidence Apriori and Brute Force execute at 0.04587s and 1770.33s respectively.

### Apriori Results

Graphical user interface

Description automatically generated with medium confidence

Text

Description automatically generated

### Brute Force Results

Graphical user interface

Description automatically generated

Text

Description automatically generated

## Apriori vs Brute Force – Database 2

At 15% support and 65% confidence Apriori and Brute Force execute at 0.01499s and 401.06s respectively. At 15% support and 70% confidence Apriori and Brute Force execute at 0.0159s and 437.49s respectively.

### Apriori Results

A picture containing scatter chart

Description automatically generated

Text

Description automatically generated

### Brute Force Results

Text

Description automatically generated with medium confidence

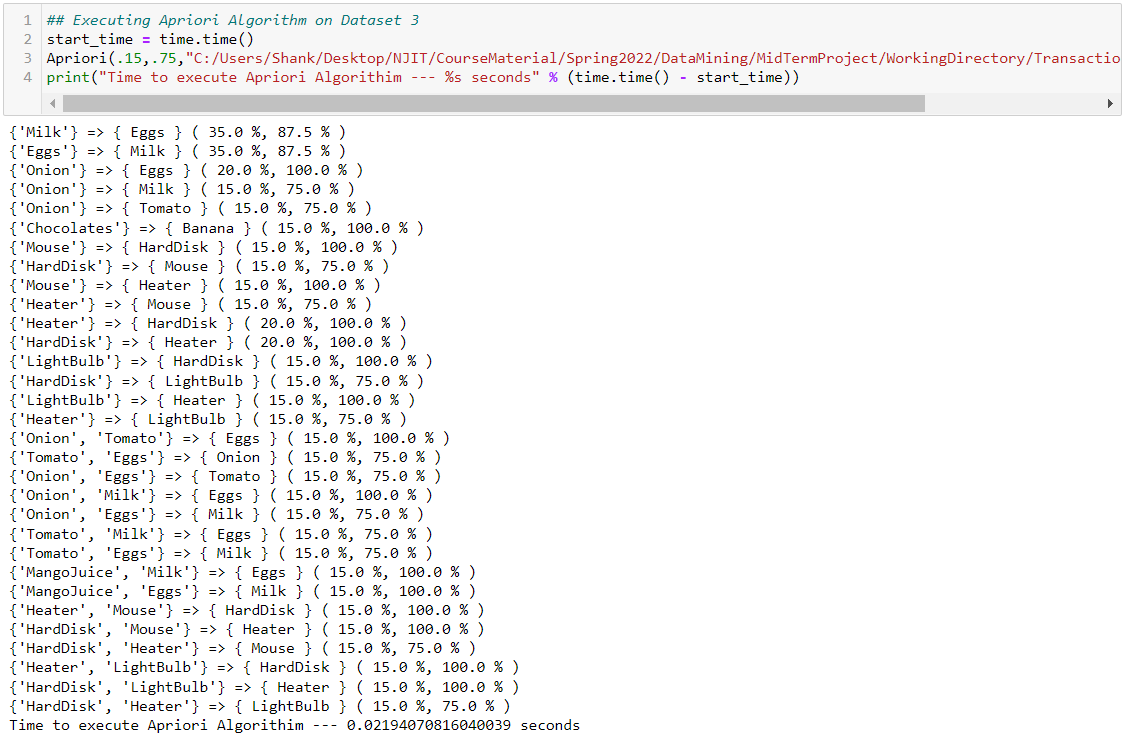
Text

Description automatically generated

## Apriori vs. Brute Force – Database 3

At 15% support and 75% confidence Apriori and Brute Force execute at 0.0219s and 99.77s respectively. At 10% support and 90% confidence Apriori and Brute Force execute at 0.0229s and 105.38s respectively.

### Apriori Results



Text

Description automatically generated

### Brute Force Results

Text

Description automatically generated with medium confidence

Text

Description automatically generated

## Apriori vs. Brute Force – Database 4

At 15% support and 60% confidence Apriori and Brute Force execute at 0.0139s and 1572.45s respectively. At 10% support and 85% confidence Apriori and Brute Force execute at 0.0269s and 1694.82s respectively.

### Apriori Results

Text

Description automatically generated

Text

Description automatically generated

### Brute Force Results

Text

Description automatically generated

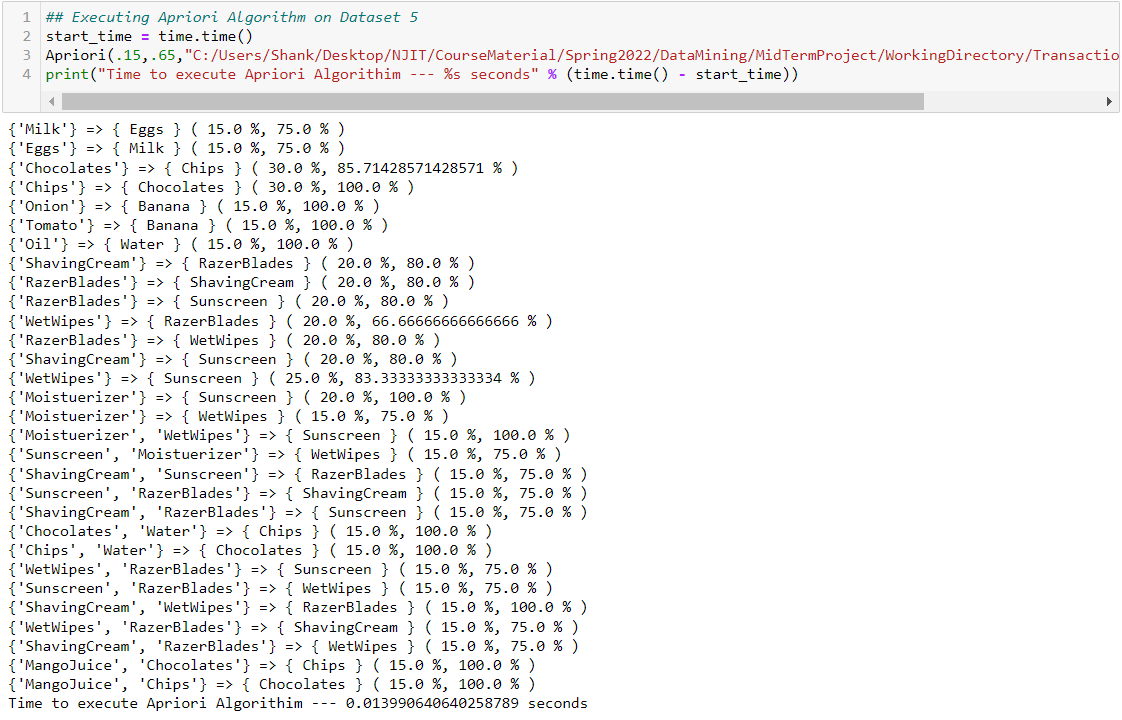
Text

Description automatically generated

## Apriori vs Brute Force – Database 5

At 15% support and 65% confidence Apriori and Brute Force execute at 0.01399s and 98.03srespectively. At 10% support and 83% confidence Apriori and Brute Force execute at 0.0759s and 106.19s respectively.

### Apriori Results



Text

Description automatically generated

### Brute Force Results

Text

Description automatically generated

Text

Description automatically generated

# Conclusion

We have conducted two different types of association rule mining algorithms on 5 different databases, and in each case, across different support and confidence level combinations, we have determined that Apriori algorithm is faster than the Brute Force algorithm by a considerable margin.