CSE 601: Data Mining & Bioinformatics

Association Project Report

SUBMITTED BY:

MALINI ANBAZHAGAN (50289383)

MANDYAM BHARAT REDDY (50289569)

MONISHA BALAJI (50290962)

Table of Contents

INTRODUCTTION

Apriori Algorithm has been widely used to help work with relational databases that involve a collection of data items forming transactions. The primary goal of this algorithm is to aid with frequent item set detection and Association rule learning and generation over the given databases. This algorithm is consecutive such that the frequent item sets generation help with the association rule formation. This project involves the implementation of the Apriori algorithm to generate rules from any database provided.

APRIORI ALGORITHM

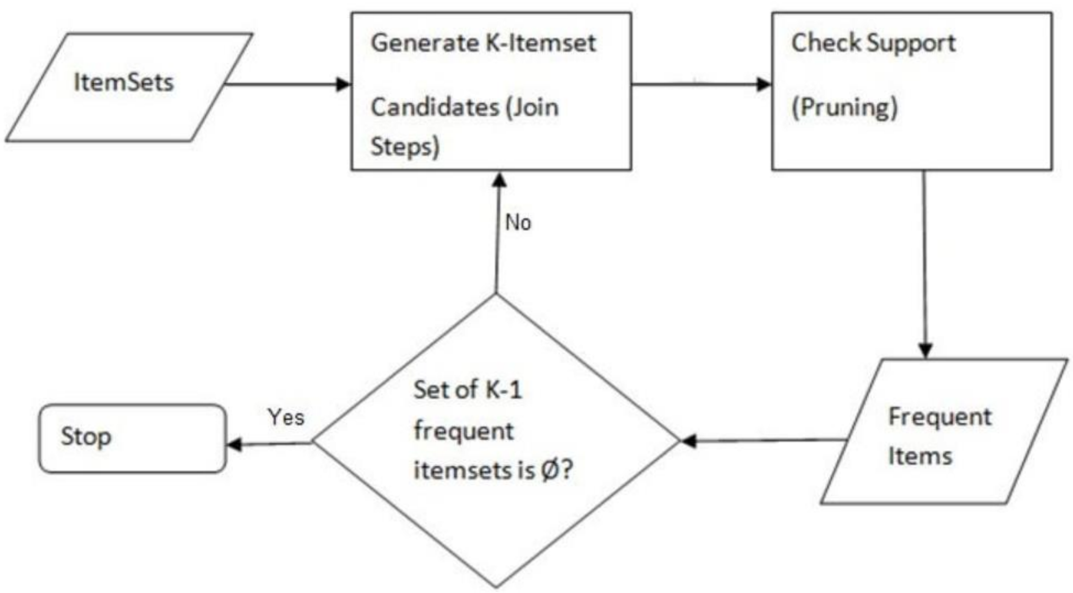
The Apriori Algorithm implementation involves two stages:

i.) Frequent Item set Generation

ii.) Association Rule Formation

During these stages, there are two parameters that are of vital importance; Support and Confidence. Support refers to the proportion of transactions in which the candidate items appear. While Confidence is defined as the likelihood of one item being purchased with respect to another. These parameter scores help determine the production of item sets and rules.

Frequent Item set Generation



Given a relational database consisting of transactions, the first aim is to identify all combinations of frequent item sets of varying length. The steps involved to generate the item sets are:

- Identify the items in the transactions. Start with length set to 1. Minimum support score is calculated.

- For each of these candidate items, calculate support and check if scores are greater than the minimum support given. For each of the items that have a lower support value than the given minimum support is considered infrequent and ignored.
- Now from the initial set of frequent items of length 1, form all possible combinations of candidate items with an increment of length by 1.
- Repeat the process of calculating and checking the support value. Prune the infrequent items by comparing it with minimum support given.
- Generation of candidate item sets by length increment is repeated until no further frequent items is possible.
- Halt after identifying all frequent item sets.

Association Rule Formation

Moving on to the second phase of the apriori algorithm, the rules are generated from the frequent item sets. The steps are as follows:

- Ignore items of length 1 as rules with either of the sides equal to null is ineffective. Minimum confidence value is noted.
- For item sets starting from length 2, form all possible combinations of rules and calculate the confidence for each.
- Confidence is calculated using the below formula:

  Rule: S → (L – S)

  Where S is subset of L

  Confidence of S = Support_Count(L) / Support_Count(S)

- Ignore the ones with scores less than the given minimum confidence.
- Halt after generating all possible rules from the frequent item sets.

ALGORITHM IMPLEMENTATION

This project requires implementing the algorithm without the use of existing functions and packages. Given for this project was a database about gene expressions where the last column gives the name of the disease. A brief flow of our Apriori algorithm implementation is given below:

Part-1 Flow

- Load the dataset and convert into a dataframe

In [2]:

```python
# User Input for Dataset
file_name = input("Enter file name: ")
```

```
Enter file name: assos.txt
```

In [3]:

```python
# Reading file into Dataframe
Data = pd.read_csv(file_name, sep='\t', lineterminator='\n', header=None)
Data
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	91	92	93	94	95	96	97	98	99	100
0	Up	Up	Down	Up	Down	Up	Up	Down	Down	Up	...	Up	Up	Down	Up	Down	Down	Down	Up	Down	ALL\r
1	Up	Down	Up	Down	Up	Down	Down	Down	Down	Up	...	Up	Up	Down	Down	Down	Down	Down	Up	Up	ALL\r
2	Down	Down	Up	Up	Up	Up	Down	Up	Up	Up	...	Up	Up	Down	Up	Down	Down	Up	Up	Up	ALL\r
3	Down	Down	Down	Down	Down	Down	Down	Up	Down	Up	...	Up	Down	Up	Down	Down	Up	Down	Up	Down	AML\r
4	Up	Up	Down	Down	Down	Down	Up	Up	Up	Down	...	Up	Up	Down	Down	Down	Down	Down	Up	Up	Breast Cancer\r

- Accept Minimum Support from user

In [5]:

```python
# User Input for Support
Support = int(input("Enter Support Percentage: "))
Support = Support / 100 * len(Data)
print("Support is set to be ",int(Support),"%")
```

```
Enter Support Percentage: 50
Support is set to be  50 %
```

- Create a list of unique candidate items

In [6]:

```python
freq_set = set()
l = len(Data.columns)
# Extracting unique items from each column
for i in range(l):
    d = Data[i]
    for j in d:
        if j not in freq_set:
            freq_set.add(j)
```

In [7]:

```python
freq_set
```

```
 'G91_Down',
 'G91_Up',
 'G92_Down',
 'G92_Up',
 'G93_Down',
 'G93_Up',
 'G94_Down',
 'G94_Up',
 'G95_Down',
 'G95_Up',
 'G96_Down',
 'G96_Up',
 'G97_Down',
 'G97_Up'
```

- Generating combinations of different lengths (1,2,3,4…) candidate item sets

In [9]:
```
# Function to generate all combinations of lengths 1,2,3,4,.. etc.
# itertools.combinations returns a list of possible combinations of desired length
def allcombinations(items, c):
    combinations = itertools.combinations(items, c)
    return [set(i) for i in list(combinations)]
```

- Compute support scores and compare with Minimum support to calculate frequent itemsets

In [10]:
```
# Initializing s as 0 for calculating total number of frequent itemsets of all lengths
s = 0
final_list=[]
for i in range(1, l):
    # Getting combinations for each length
    candidates = allcombinations(freq_set, i)
    set_ap = []
# Count frequency of each combination present in the superset and storing those that have greater value than support provided by u
    for k in candidates:
        sup_count = 0
        for j in Superset:
            if k.issubset(j):
                sup_count = sup_count + 1
        if sup_count >= Support:
            set_ap.append(k)
            final_list.append(k)
# Exiting the loop if no new itemsets are found for a particular length
    if len(set_ap) == 0:
        break
    else:
        print("number of length-" + str(i), "frequent itemsets: " + str(len(set_ap)))
```

- Display final results

```python
print("number of all lengths frequent itemsets: ", s)
print("Final list of frequent itemsets", final_list)
```

```
number of length-1 frequent itemsets: 109
number of length-2 frequent itemsets: 63
number of length-3 frequent itemsets: 2
number of all lengths frequent itemsets:  174
Final list of frequent itemsets [{'G91_Up'}, {'G68_Down'}, {'G62_Down'}, {'G6_Up'}, {'G61_Down'}, {'G37_Up'}, {'G11_Down'},
{'G3_Down'}, {'G65_Down'}, {'G31_Up'}, {'G73_Down'}, {'G23_Up'}, {'G41_Down'}, {'G49_Down'}, {'G4_Up'}, {'G18_Down'}, {'G43_Do
wn'}, {'G29_Down'}, {'G88_Down'}, {'G28_Down'}, {'G27_Up'}, {'G66_Up'}, {'G15_Down'}, {'G64_Up'}, {'G59_Up'}, {'G21_Up'}, {'G6
3_Up'}, {'G57_Down'}, {'G48_Up'}, {'G25_Up'}, {'G82_Down'}, {'G87_Up'}, {'G99_Up'}, {'G30_Down'}, {'G83_Up'}, {'G38_Down'},
{'G74_Down'}, {'G20_Down'}, {'G50_Down'}, {'G95_Down'}, {'G36_Up'}, {'G86_Down'}, {'G24_Down'}, {'G84_Up'}, {'G97_Down'}, {'G1
0_Down'}, {'G35_Down'}, {'G79_Down'}, {'G69_Down'}, {'G47_Up'}, {'G70_Down'}, {'G83_Down'}, {'G8_Up'}, {'G44_Down'}, {'G63_Dow
n'}, {'G12_Up'}, {'G14_Up'}, {'G80_Down'}, {'G90_Up'}, {'G92_Down'}, {'G32_Down'}, {'G3_Up'}, {'G56_Up'}, {'G93_Up'}, {'G5_U
p'}, {'G67_Up'}, {'G17_Up'}, {'G75_Up'}, {'G53_Up'}, {'G96_Down'}, {'G58_Down'}, {'G60_Up'}, {'G2_Down'}, {'G53_Down'}, {'G77_
Up'}, {'G7_Down'}, {'G33_Down'}, {'G52_Down'}, {'G19_Down'}, {'G85_Down'}, {'G9_Up'}, {'G46_Down'}, {'G76_Down'}, {'G98_Up'},
{'G81_Up'}, {'G40_Down'}, {'G22_Up'}, {'G13_Down'}, {'G55_Up'}, {'G39_Up'}, {'G16_Down'}, {'G43_Up'}, {'G45_Up'}, {'G94_Up'},
{'G1_Up'}, {'G71_Up'}, {'G54_Up'}, {'G64_Down'}, {'G51_Down'}, {'G34_Down'}, {'G72_Up'}, {'G50_Up'}, {'G26_Up'}, {'G100_Dow
n'}, {'G78_Up'}, {'G42_Down'}, {'G9_Down'}, {'G89_Up'}, {'G4_Down'}, {'G38_Down', 'G91_Up'}, {'G6_Up', 'G28_Down'}, {'G6_Up',
'G59_Up'}, {'G6_Up', 'G38_Down'}, {'G6_Up', 'G32_Down'}, {'G6_Up', 'G13_Down'}, {'G65_Down', 'G38_Down'}, {'G88_Down', 'G41_Do
wn'}, {'G41_Down', 'G28_Down'}, {'G38_Down', 'G41_Down'}, {'G88_Down', 'G28_Down'}, {'G88_Down', 'G59_Up'}, {'G88_Down', 'G87_
Up'}, {'G88_Down', 'G38_Down'}, {'G88_Down', 'G24_Down'}, {'G88_Down', 'G10_Down'}, {'G88_Down', 'G8_Up'}, {'G88_Down', 'G54_U
p'}, {'G28_Down', 'G59_Up'}, {'G87_Up', 'G28_Down'}, {'G38_Down', 'G28_Down'}, {'G28_Down', 'G10_Down'}, {'G47_Up', 'G28_Dow
n'}, {'G32_Down', 'G28_Down'}, {'G28_Down', 'G2_Down'}, {'G52_Down', 'G28_Down'}, {'G28_Down', 'G13_Down'}, {'G82_Down', 'G59_
Up'}, {'G87_Up', 'G59_Up'}, {'G38_Down', 'G59_Up'}, {'G59_Up', 'G10_Down'}, {'G32_Down', 'G59_Up'}, {'G59_Up', 'G96_Down'},
{'G59_Up', 'G13_Down'}, {'G1_Up', 'G59_Up'}, {'G72_Up', 'G59_Up'}, {'G97_Down', 'G82_Down'}, {'G82_Down', 'G96_Down'}, {'G82_D
```

With the minimum support value set to 50%, for our dataset, a total of 174 frequent item sets of all lengths were created. The count of length-1 item sets was equal to 109. The count of length-2 item sets was 63 and finally the count of the longest length item sets, of length equal to 3, was 2.

Part-2 Flow

- Accept Minimum Confidence as user input

```python
In [10]: # User Input for Confidence
conf = int(input("Enter confidence Percentage: "))
conf = conf / 100 * len(Data)
print("Minimum Confidence is set to be ",int(conf),"%")
conf=conf/100
```

```
Enter confidence Percentage: 70
Minimum Confidence is set to be  70 %
```

- Calculate confidence and check with minimum confidence value and generate rules

```
for k in final_list[i]:
    sup_count = 0
    rhs = 0
    k = {k}
    for j in Superset:
        if final_list[i].issubset(j):
            sup_count = sup_count + 1
        if k.issubset(j):
            rhs = rhs + 1
    # Computing Confidence for each rule
    final_conf = sup_count/rhs
    #print(final_conf,k,"-->",str(final_list[i]-k),"Sup:", sup_count," rhs:",rhs)
    # Check if candidate rule has confidence greater than or equal to minconf
    if final_conf>=conf:
        #print(final_conf,k)
        rule_form = str(k) + "-->" + str(final_list[i]-k)
        rules.append(rule_form)
```

- Display final result

```
print("Total number of rules generated: ",len(rules))
print("Rules:")
for r in rules:
    print(r)
```

```
Total number of rules generated:  117
Rules:
{'G82_Down'}-->{'G97_Down'}
{'G97_Down'}-->{'G82_Down'}
{'G97_Down'}-->{'G72_Up'}
{'G67_Up'}-->{'G38_Down'}
{'G67_Up'}-->{'G1_Up'}
{'G1_Up'}-->{'G67_Up'}
{'G28_Down'}-->{'G87_Up'}
{'G87_Up'}-->{'G28_Down'}
{'G28_Down'}-->{'G59_Up'}
{'G28_Down'}-->{'G6_Up'}
{'G6_Up'}-->{'G28_Down'}
{'G28_Down'}-->{'G52_Down'}
{'G52_Down'}-->{'G28_Down'}
{'G28_Down'}-->{'G88_Down'}
{'G88_Down'}-->{'G28_Down'}
{'G28_Down'}-->{'G38_Down'}
{'G38_Down'}-->{'G28_Down'}
{'G28_Down'}-->{'G10_Down'}
```

As can be seen from the above snapshots, when Support is set to 50% and Confidence is set to 70%, 117 association rules are generated for the gene expression dataset.

This project involves producing the results in specific Template formats. Results of each query of any template type produces the count and the list of association rules that abide by the conditions of the query parameters. To help in generating the results as per template requirements, the association rules are split into two: HEAD and BODY.

TEMPLATE 1

- In Template 1, we have 3 parameters,
    - First parameter is one of RULE, HEAD, BODY. HEAD is the left hand side of arrow in Rule, BODY is right hand side of arrow in Rule
    - Second parameter is one of ANY, NONE , NUMBER(1,2,3..)
        - ANY means return all rule/body/head that have any of the given item/s
        - NONE means return all rule/body/head that do not have the given item/s

- NUMBER means return all rule/body/head that have exactly the number of items given by the user. For example, if 1 is entered, then return all rules/head/body that contain exactly one of the given items
  - Third parameter is the list of items to be queried against the rules generated

A snapshot of one of the mentioned required queries is given below:

```
Enter Query Or "exit" for Exiting: asso_rule.template1("BODY","ANY",['G59_Up'])
Query:  ("BODY","ANY",['G59_Up'])
Count: 17
Rules Queried:
{'G28_Down'}-->{'G59_Up'}
{'G87_Up'}-->{'G59_Up'}
{'G82_Down'}-->{'G59_Up'}
{'G6_Up'}-->{'G59_Up'}
{'G88_Down'}-->{'G59_Up'}
{'G38_Down'}-->{'G59_Up'}
{'G10_Down'}-->{'G59_Up'}
{'G96_Down'}-->{'G59_Up'}
{'G1_Up'}-->{'G59_Up'}
{'G32_Down'}-->{'G59_Up'}
{'G72_Up'}-->{'G59_Up'}
{'G13_Down'}-->{'G59_Up'}
{'G96_Down'}-->{'G59_Up', 'G72_Up'}
{'G96_Down', 'G72_Up'}-->{'G59_Up'}
{'G82_Down'}-->{'G59_Up', 'G72_Up'}
{'G72_Up'}-->{'G82_Down', 'G59_Up'}
{'G82_Down', 'G72_Up'}-->{'G59_Up'}
Enter Query Or "exit" for Exiting: exit
```

As mentioned, it produces the count and rule list for the entered query.

TEMPLATE 2

- In Template 2, we have 2 parameters,
  - First parameter is RULE/HEAD/BODY similar to Template1
  - Second parameter is a number
- In Template 2, all Rule/Body/Head that have size greater than or equal to the number entered are returned

A snapshot of the results for template 2 as mentioned in the handouts is given below:

```
Enter Query Or "exit" for Exiting: asso_rule.template2("BODY",2)
Query:  ("BODY",2)
Count: 3
Rules Queried:
{'G96_Down'}-->{'G59_Up', 'G72_Up'}
{'G82_Down'}-->{'G59_Up', 'G72_Up'}
{'G72_Up'}-->{'G82_Down', 'G59_Up'}

Enter Query Or "exit" for Exiting:
```

TEMPLATE 3

- Template 3 is basically a combination of Templates 1 & 2 using Logical AND or OR operators
- Return Rules/Head/Body that satisfy AND or OR operations of Templates 1 or 2 queries

A snapshot of the results for template 3 is shown:

```
Enter Query Or "exit" for Exiting: asso_rule.template3("1or2","HEAD","ANY",['G10_Down'],"BODY",2)
Query:  ("1or2","HEAD","ANY",['G10_Down'],"BODY",2)
Count: 11
Rules Queried:
{'G10_Down'}-->{'G28_Down'}
{'G10_Down'}-->{'G59_Up'}
{'G10_Down'}-->{'G88_Down'}
{'G10_Down'}-->{'G38_Down'}
{'G10_Down'}-->{'G70_Down'}
{'G10_Down'}-->{'G1_Up'}
{'G10_Down'}-->{'G47_Up'}
{'G10_Down'}-->{'G94_Up'}
{'G96_Down'}-->{'G59_Up', 'G72_Up'}
{'G82_Down'}-->{'G59_Up', 'G72_Up'}
{'G72_Up'}-->{'G82_Down', 'G59_Up'}

Enter Query Or "exit" for Exiting:
```

RESULTS

Part 1

Support =30%

```
Support is set to be  30 %
number of length-1 frequent itemsets: 196
number of length-2 frequent itemsets: 5340
number of length-3 frequent itemsets: 5287
number of length-4 frequent itemsets: 1518
number of length-5 frequent itemsets: 438
number of length-6 frequent itemsets: 88
number of length-7 frequent itemsets: 11
number of length-8 frequent itemsets: 1
number of all lengths frequent itemsets:  12879
```

Support = 40%

```
Support is set to be  40 %
number of length-1 frequent itemsets: 167
number of length-2 frequent itemsets: 753
number of length-3 frequent itemsets: 149
number of length-4 frequent itemsets: 7
number of length-5 frequent itemsets: 1
number of all lengths frequent itemsets:  1077
```

Support = 50%

```
Support is set to be  50 %
number of length-1 frequent itemsets: 109
number of length-2 frequent itemsets: 63
number of length-3 frequent itemsets: 2
number of all lengths frequent itemsets:  174
```

Support = 60%

```
Support is set to be  60 %
number of length-1 frequent itemsets: 34
number of length-2 frequent itemsets: 2
number of all lengths frequent itemsets:  36
```

Support = 70%

```
Support is set to be  70 %
number of length-1 frequent itemsets: 7
number of all lengths frequent itemsets:  7
```

Part 2

```
Enter Support Percentage: 50
Support is set to be  50 %
number of length-1 frequent itemsets: 109
number of length-2 frequent itemsets: 63
number of length-3 frequent itemsets: 2
number of all lengths frequent itemsets:  174
Enter confidence Percentage: 70
Minimum Confidence is set to be  70 %
Total number of rules generated:  117
```

Template 1

1.) *asso_rule.template1("RULE", "ANY", ['G59_Up'])*

```
Enter Query Or "exit" for Exiting: asso_rule.template1("RULE", "ANY", ['G59_Up'])
Count: 26
```

2.) asso_rule.template1("RULE", "NONE", ['G59_Up'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("RULE", "NONE", ['G59_Up'])
Count: 91
```

3.) asso_rule.template1("RULE", 1, ['G59_Up', 'G10_Down'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("RULE", 1, ['G59_Up', 'G10_Down'])
Count: 39
```

4.) asso_rule.template1("HEAD", "ANY", ['G59_Up'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("HEAD", "ANY", ['G59_Up'])
Count: 9
```

5.) asso_rule.template1("HEAD", "NONE", ['G59_Up'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("HEAD", "NONE", ['G59_Up'])
Count: 108
```

6.) asso_rule.template1("HEAD", 1, ['G59_Up', 'G10_Down'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("HEAD", 1, ['G59_Up', 'G10_Down'])
Count: 17
```

7.) asso_rule.template1("BODY", "ANY", ['G59_Up'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("BODY", "ANY", ['G59_Up'])
Count: 17
```

8.) **a**sso_rule.template1("BODY", "NONE", ['G59_Up'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("BODY", "NONE", ['G59_Up'])
Count: 100
```

9.) asso_rule.template1("BODY", 1, ['G59_Up', 'G10_Down'])

```
Enter Query Or "exit" for Exiting: asso_rule.template1("BODY", 1, ['G59_Up', 'G10_Down'])
Count: 24
```

Template 2

1.) asso_rule.template2("RULE", 3)

```
Enter Query Or "exit" for Exiting: asso_rule.template2("RULE", 3)
Count: 9
```

2.) asso_rule.template2("HEAD", 2)

```
Enter Query Or "exit" for Exiting: asso_rule.template2("HEAD", 2)
Count: 6
```

3.) asso_rule.template2("BODY", 1)

```
Enter Query Or "exit" for Exiting: asso_rule.template2("BODY", 1)
Count: 117
```

Template 3

1.) asso_rule.template3("1or1", "HEAD", "ANY", ['G10_Down'], "BODY", 1, ['G59_Up'])

```
Enter Query Or "exit" for Exiting: asso_rule.template3("1or1", "HEAD", "ANY", ['G10_Down'], "BODY", 1, ['G59_Up'])
Count: 24
```

2.) asso_rule.template3("1and1", "HEAD", "ANY", ['G10_Down'], "BODY", 1, ['G59_Up'])

```
Enter Query Or "exit" for Exiting: asso_rule.template3("1and1", "HEAD", "ANY", ['G10_Down'], "BODY", 1, ['G59_Up'])
Count: 1
```

3.) asso_rule.template3("1or2", "HEAD", "ANY", ['G10_Down'], "BODY", 2)

```
Enter Query Or "exit" for Exiting: asso_rule.template3("1or2", "HEAD", "ANY", ['G10_Down'], "BODY", 2)
Count: 11
```

4.) asso_rule.template3("1and2", "HEAD", "ANY", ['G10_Down'], "BODY", 2)

```
Enter Query Or "exit" for Exiting: asso_rule.template3("1and2", "HEAD", "ANY", ['G10_Down'], "BODY", 2)
Count: 0
```

5.) asso_rule.template3("2or2", "HEAD", 1, "BODY", 2)

```
Enter Query Or "exit" for Exiting: asso_rule.template3("2or2", "HEAD", 1, "BODY", 2)
Count: 117
```

6.) asso_rule.template3("2and2", "HEAD", 1, "BODY", 2)

```
Enter Query Or "exit" for Exiting: asso_rule.template3("2and2", "HEAD", 1, "BODY", 2)
Count: 3
```

CONCLUSION

Thus, this project assisted us, in gaining clear insights on the Apriori algorithm and the Association Rule Generation algorithm. This project involved implementing our very own Apriori algorithm that acknowledge the conditions and properties of support and confidence parameters; generating frequent item sets bearing these properties and thereby producing the association rules.