CS 634-854 Data Mining
Midterm Project

Hollins Justin Jose
NJIT UCID: *hj84*
ID#: *31401785*
Email: *hj84@njit.edu*

Taught By: Dr. Jason T.L. Wang
Department of Computer Science
New Jersey Institute of Technology

Table of Contents

Databases Used

DB1.csv

```
1  Banana,Apple,Orange
2  Apple,Orange,Banana
3  Tomato,Garlic,Onion
4  Banana,Orange,Apple
5  Apple,Banana,Orange
6  Tomato,Garlic,Onion
7  Onion,Garlic,Tomato
8  Onion,Banana,Orange
9  Onion,Apple,Orange
10 Apple,Banana,Orange
11 Apple,Banana,Orange
12 Tomato,Banana,Apple
13 Tomato,Apple
14 Apple,Tomato,Garlic
15 Garlic,Tomato,Apple
16 Tomato,Garlic,Onion
17 Garlic,Onion,Tomato
18 Onion,Garlic,Apple
19 Orange,Apple
20 Tomato,Garlic,Onion
```

DB2.csv

```
1  Poster,Thumbtacks,Tape
2  Poster,Tape,Thumbtacks
3  Tape,Thumbtacks,Poster
4  Marker,Poster,Pencil
5  Poster,Tape,Thumbtacks
6  Tape,Poster,Thumbtacks
7  Tape,Poster,Thumbtacks
8  Pencil,Marker,Calculator
9  Marker,Poster,Pencil
10 Thumbtacks,Poster,Tape
11 Marker,Pencil,Calculator
12 Tape,Thumbtacks,Poster
13 Thumbtacks,Poster,Tape
14 Poster,Tape
15 Poster,Thumbtacks,Tape
16 Calculator,Tape,Pencil
17 Calculator,Pencil
18 Pencil,Poster,Marker
19 Marker,Poster,Pencil
20 Poster,Marker,Thumbtacks
```

DB3.csv

```
1  Jeans,Shirt,Sweater
2  Shoes,Underwear,Socks
3  Jeans,Shirt,Sweater
4  Shirt,Jeans,Sweater
5  Jeans,Shirt,Sweater
6  Underwear,Socks,Jeans
7  Socks,Underwear,Sweater
8  Underwear,Socks,Shirt
9  Socks,Underwear
10 Underwear,Socks,Jeans
11 Jeans,Shirt,Sweater
12 Sweater,Jeans,Socks
13 Underwear,Socks,Shoes
14 Shoes,Jeans,Shirt
15 Jeans,Sweater,Shirt
16 Shirt,Sweater,Socks
17 Sweater,Shirt,Jeans
18 Socks,Shirt,Underwear
19 Underwear,Socks
20 Socks,Underwear
```

DB4.csv

```
Fishing rod,Bucket,Bait
Tent,Hat
Fishing rod,Bucket,Bait
Bucket,Fishing rod,Bait
Hat,Boots,Bucket
Bucket,Fishing rod,Bait
Fishing rod,Bait,Boots
Hat,Bucket,Boots
Boots,Hat,Tent
Fishing rod,Bait,Boots
Bait,Bucket,Hat
Boots,Tent,Bucket
Bait,Fishing rod,Tent
Bucket,Fishing rod,Tent
Bait,Fishing rod,Boots
Fishing rod,Bait,Bucket
Hat,Boots,Tent
Fishing rod,Bait,Bucket
Bucket,Bait,Boots
Bait,Fishing rod
```

DB5.csv

```
Sanitizer,Clorox wipes,Paper towels
Clorox wipes,Paper towels,Sanitizer
Sanitizer,Clorox wipes
Clorox wipes,Sanitizer,Paper towels
Sanitizer,Clorox wipes
Soap,Febreeze,Paper towels
Febreeze,Sanitizer,Soap
Soap,Paper towels,Deodorant
Deodorant,Soap
Sanitizer,Clorox wipes,Soap
Clorox wipes,Sanitizer
Sanitizer,Clorox wipes,Deodorant
Sanitizer,Clorox wipes
Clorox wipes,Sanitizer,Paper towels
Febreeze,Paper towels,Clorox wipes
Paper towels,Febreeze
Soap,Deodorant,Paper towels
Deodorant,Soap,Febreeze
Soap,Deodorant
Deodorant,Febreeze,Sanitizer
```

DBMaster.csv

```
Banana,Apple,Orange
Apple,Orange,Banana
Tomato,Garlic,Onion
Banana,Orange,Apple
Apple,Banana,Orange
Tomato,Garlic,Onion
Onion,Garlic,Tomato
Onion,Banana,Orange
Onion,Apple,Orange
Apple,Banana,Orange
Apple,Banana,Orange
Tomato,Banana,Apple
Tomato,Apple
Apple,Tomato,Garlic
Garlic,Tomato,Apple
Tomato,Garlic,Onion
Garlic,Onion,Tomato
Onion,Garlic,Apple
Orange,Apple
Tomato,Garlic,Onion
Poster,Thumbtacks,Tape
Poster,Tape,Thumbtacks
Tape,Thumbtacks,Poster
Marker,Poster,Pencil
Poster,Tape,Thumbtacks
Tape,Poster,Thumbtacks
Tape,Poster,Thumbtacks
Pencil,Marker,Calculator
Marker,Poster,Pencil
Thumbtacks,Poster,Tape
Marker,Pencil,Calculator
Tape,Thumbtacks,Poster
Thumbtacks,Poster,Tape
Poster,Tape
Poster,Thumbtacks,Tape
Calculator,Tape,Pencil
Calculator,Pencil
Pencil,Poster,Marker
Marker,Poster,Pencil
Poster,Marker,Thumbtacks
Jeans,Shirt,Sweater
Shoes,Underwear,Socks
Jeans,Shirt,Sweater
Shirt,Jeans,Sweater
Jeans,Shirt,Sweater
Underwear,Socks,Jeans
Socks,Underwear,Sweater
Underwear,Socks,Shirt
Socks,Underwear
Underwear,Socks,Jeans
Jeans,Shirt,Sweater
Sweater,Jeans,Socks
Underwear,Socks,Shoes
Shoes,Jeans,Shirt
Jeans,Sweater,Shirt
Shirt,Sweater,Socks
Sweater,Shirt,Jeans
Socks,Shirt,Underwear
Underwear,Socks
Socks,Underwear
Fishing rod,Bucket,Bait
Tent,Hat
Fishing rod,Bucket,Bait
Bucket,Fishing rod,Bait
```

DBMaster.csv

```
65   Hat,Boots,Bucket
66   Bucket,Fishing rod,Bait
67   Fishing rod,Bait,Boots
68   Hat,Bucket,Boots
69   Boots,Hat,Tent
70   Fishing rod,Bait,Boots
71   Bait,Bucket,Hat
72   Boots,Tent,Bucket
73   Bait,Fishing rod,Tent
74   Bucket,Fishing rod,Tent
75   Bait,Fishing rod,Boots
76   Fishing rod,Bait,Bucket
77   Hat,Boots,Tent
78   Fishing rod,Bait,Bucket
79   Bucket,Bait,Boots
80   Bait,Fishing rod
81   Sanitizer,Clorox wipes,Paper towels
82   Clorox wipes,Paper towels,Sanitizer
83   Sanitizer,Clorox wipes
84   Clorox wipes,Sanitizer,Paper towels
85   Sanitizer,Clorox wipes
86   Soap,Febreeze,Paper towels
87   Febreeze,Sanitizer,Soap
88   Soap,Paper towels,Deodorant
89   Deodorant,Soap
90   Sanitizer,Clorox wipes,Soap
91   Clorox wipes,Sanitizer
92   Sanitizer,Clorox wipes,Deodorant
93   Sanitizer,Clorox wipes
94   Clorox wipes,Sanitizer,Paper towels
95   Febreeze,Paper towels,Clorox wipes
96   Paper towels,Febreeze
97   Soap,Deodorant,Paper towels
98   Deodorant,Soap,Febreeze
99   Soap,Deodorant
100  Deodorant,Febreeze,Sanitizer
```

Comparison of Algorithms

Apriori and Brute Force Algorithm Results, Support = 25% and Confidence = 30% (DB1)

```
Input Support, hit Enter, and input Confidence (in %): 25
30
Association Rules for Apriori Algorithm:
Garlic -> Tomato [40%, 88%]
Tomato -> Garlic [40%, 80%]
Onion -> Tomato [30%, 66%]
Tomato -> Onion [30%, 60%]
Apple -> Banana [35%, 53%]
Banana -> Apple [35%, 87%]
Apple -> Orange [40%, 61%]
Orange -> Apple [40%, 88%]
Banana -> Orange [35%, 87%]
Orange -> Banana [35%, 77%]
Garlic -> Onion [35%, 77%]
Onion -> Garlic [35%, 77%]
Apple Banana -> Orange [30%, 85%]
Banana Orange -> Apple [30%, 85%]
Apple Orange -> Banana [30%, 75%]
Garlic Onion -> Tomato [30%, 85%]
Onion Tomato -> Garlic [30%, 100%]
Garlic Tomato -> Onion [30%, 75%]
Total execution time for Apriori Algorithm: 37ms
```

Total Time of Execution for Apriori Algorithm is: **37 ms**

```
Input Support, hit Enter, and input Confidence (in %): 25
30
Association Rules for Brute Force Algorithm:
Garlic -> Tomato [40%, 88%]
Tomato -> Garlic [40%, 80%]
Onion -> Tomato [30%, 66%]
Tomato -> Onion [30%, 60%]
Apple -> Banana [35%, 53%]
Banana -> Apple [35%, 87%]
Apple -> Orange [40%, 61%]
Orange -> Apple [40%, 88%]
Banana -> Orange [35%, 87%]
Orange -> Banana [35%, 77%]
Garlic -> Onion [35%, 77%]
Onion -> Garlic [35%, 77%]
Apple Banana -> Orange [30%, 85%]
Banana Orange -> Apple [30%, 85%]
Apple Orange -> Banana [30%, 75%]
Garlic Onion -> Tomato [30%, 85%]
Onion Tomato -> Garlic [30%, 100%]
Garlic Tomato -> Onion [30%, 75%]
Total execution time for Brute Pass: 45ms
```

Total Time of Execution for Brute Force Algorithm is: **45 ms**

The Association Rules with the highest confidence values indicate that:

- **when a customer buys an apple and an orange, they typically also buy a banana**
- **when a customer buys a garlic and an onion, they typically also buy a tomato**

Using these findings, the store can design its layout to place these items next to each other to increase revenue and enhance the customer buying experience.

Apriori and Brute Force Algorithm Results, Support = 40% and Confidence = 60% (DB2)

```
Input Support, hit Enter, and input Confidence (in %): 40
60
Association Rules for Apriori Algorithm:
Poster -> Thumbtacks [55%, 68%]
Thumbtacks -> Poster [55%, 100%]
Poster -> Tape [55%, 68%]
Tape -> Poster [55%, 91%]
Tape -> Thumbtacks [50%, 83%]
Thumbtacks -> Tape [50%, 90%]
Poster Tape -> Thumbtacks [50%, 90%]
Tape Thumbtacks -> Poster [50%, 100%]
Poster Thumbtacks -> Tape [50%, 90%]
Total execution time for Apriori Algorithm: 24ms
```

```
Input Support, hit Enter, and input Confidence (in %): 40
60
Association Rules for Brute Force Algorithm:
Poster -> Thumbtacks [55%, 68%]
Thumbtacks -> Poster [55%, 100%]
Tape -> Thumbtacks [50%, 83%]
Thumbtacks -> Tape [50%, 90%]
Poster -> Tape [55%, 68%]
Tape -> Poster [55%, 91%]
Poster Tape -> Thumbtacks [50%, 90%]
Tape Thumbtacks -> Poster [50%, 100%]
Poster Thumbtacks -> Tape [50%, 90%]
Total execution time for Brute Pass: 22ms
```

The Association Rules with the highest confidence values indicate that:

- **when a customer buys thumbtacks, they typically also buy a poster**
- **when a customer buys a poster and tape, they typically also buy thumbtacks**
- **when a customer buys tape, they typically also buy a poster**

Using these findings, the store can design its layout to place these items next to each other to increase revenue and enhance the customer buying experience.

For instance, the poster, thumbtacks, and tape can be placed in the same aisle so that customers are likely to buy more than just one item.

Apriori and Brute Force Algorithm Results, Support = 30% and Confidence = 75% (DB3)

```
Input Support, hit Enter, and input Confidence (in %): 30
75
Association Rules for Apriori Algorithm:
Sweater -> Jeans [40%, 80%]
Socks -> Underwear [50%, 83%]
Underwear -> Socks [50%, 100%]
Sweater -> Shirt [40%, 80%]
Jeans Shirt -> Sweater [35%, 87%]
Shirt Sweater -> Jeans [35%, 87%]
Jeans Sweater -> Shirt [35%, 87%]
Total execution time for Apriori Algorithm: 19ms
```

```
Input Support, hit Enter, and input Confidence (in %): 30
75
Association Rules for Brute Force Algorithm:
Sweater -> Jeans [40%, 80%]
Socks -> Underwear [50%, 83%]
Underwear -> Socks [50%, 100%]
Sweater -> Shirt [40%, 80%]
Jeans Shirt -> Sweater [35%, 87%]
Shirt Sweater -> Jeans [35%, 87%]
Jeans Sweater -> Shirt [35%, 87%]
Total execution time for Brute Pass: 21ms
```

The Association Rules with the highest confidence values indicate that:

- **when a customer buys socks, they typically also buy a underwear**
- **when a customer buys a shirt and jeans, they typically also buy a sweater**

Using these findings, the store can design its layout to place these items next to each other to increase revenue and enhance the customer buying experience.

For instance, the socks and underwear can be place directly opposite each other so that customers are likely to buy more than just one item.

Apriori and Brute Force Algorithm Results, Support = 38.4% and Confidence = 69.7% (DB4)

```
Input Support, hit Enter, and input Confidence (in %): 38.4
69.7
Association Rules for Apriori Algorithm:
Bait -> Fishing rod [55%, 84%]
Fishing rod -> Bait [55%, 91%]
Total execution time for Apriori Algorithm: 14ms
```

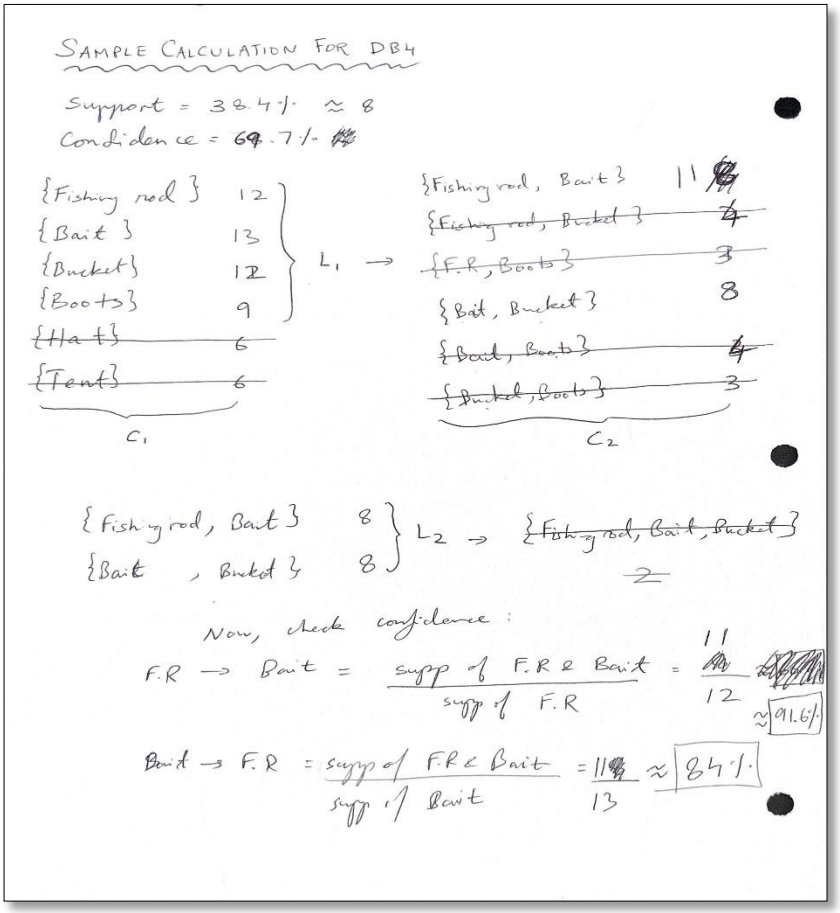
```
Input Support, hit Enter, and input Confidence (in %): 38.4
69.7
Bait -> Fishing rod [55%, 84%]
Fishing rod -> Bait [55%, 91%]
Total execution time for Brute Pass: 16ms
```

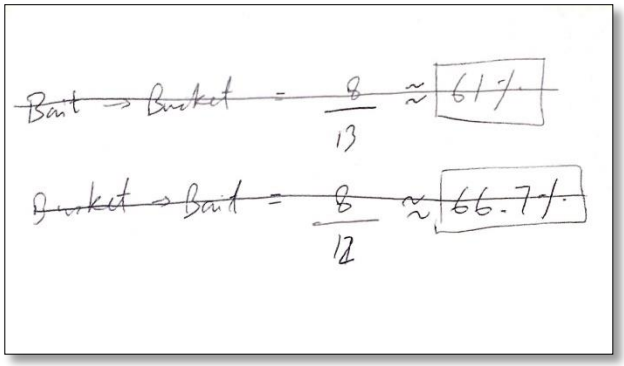
The Association Rules with the highest confidence values indicate that:

- **when a customer buys bait, they typically also buy a fishing rod**

Using these findings, the store can place the fishing rods, bait in one section of the store.

Hand Calculations Support = 38.4% and Confidence = 69.7% (DB4)





From the Hand Calculations, it can be observed that the same association rules are drawn

Apriori and Brute Force Algorithm Results, Support = 10% and Confidence = 45% (DB5)

```
Input Support, hit Enter, and input Confidence (in %): 10 45
Association Rules for Apriori Algorithm:
Clorox wipes -> Sanitizer [50%, 90%]
Sanitizer -> Clorox wipes [50%, 83%]
Clorox wipes -> Paper towels [25%, 45%]
Paper towels -> Clorox wipes [25%, 55%]
Deodorant -> Soap [25%, 71%]
Soap -> Deodorant [25%, 62%]
Febreeze -> Soap [15%, 50%]
Febreeze -> Paper towels [15%, 50%]
Clorox wipes Paper towels -> Sanitizer [20%, 80%]
Paper towels Sanitizer -> Clorox wipes [20%, 100%]
Deodorant Paper towels -> Soap [10%, 100%]
Paper towels Soap -> Deodorant [10%, 66%]
Total execution time for Apriori Algorithm: 25ms
```

```
Input Support, hit Enter, and input Confidence (in %): 10 45
Association Rules for Brute Force Algorithm:
Clorox wipes -> Sanitizer [50%, 90%]
Sanitizer -> Clorox wipes [50%, 83%]
Clorox wipes -> Paper towels [25%, 45%]
Paper towels -> Clorox wipes [25%, 55%]
Deodorant -> Soap [25%, 71%]
Soap -> Deodorant [25%, 62%]
Febreeze -> Soap [15%, 50%]
Febreeze -> Paper towels [15%, 50%]
Clorox wipes Paper towels -> Sanitizer [20%, 80%]
Paper towels Sanitizer -> Clorox wipes [20%, 100%]
Deodorant Paper towels -> Soap [10%, 100%]
Paper towels Soap -> Deodorant [10%, 66%]
Total execution time for Brute Pass: 33ms
```

The Association Rules with the highest confidence values indicate that:

- **when a customer buys Clorox wipes, they typically also buy a sanitizer**
- **when a customer buys paper towels and deodorant, they typically also buy soap**

Using these findings, especially with the pandemic afflicting the nation, the store can place Clorox wipes, sanitizer and paper towels in the front of the store as they are essentials according to the rules.

Apriori and Brute Force Algorithm Results, Support = 40% and Confidence = 60% (DBMaster)

```
Input Support, hit Enter, and input Confidence (in %): 40
60
Association Rules for Apriori Algorithm:
Jeans -> Sweater [40%, 72%]
Sweater -> Jeans [40%, 80%]
Clorox wipes -> Sanitizer [50%, 90%]
Sanitizer -> Clorox wipes [50%, 83%]
Garlic -> Tomato [40%, 88%]
Tomato -> Garlic [40%, 80%]
Socks -> Underwear [50%, 83%]
Underwear -> Socks [50%, 100%]
Poster -> Thumbtacks [55%, 68%]
Thumbtacks -> Poster [55%, 100%]
Bait -> Fishing rod [55%, 84%]
Fishing rod -> Bait [55%, 91%]
Tape -> Thumbtacks [50%, 83%]
Thumbtacks -> Tape [50%, 90%]
Poster -> Tape [55%, 68%]
Tape -> Poster [55%, 91%]
Apple -> Orange [40%, 61%]
Orange -> Apple [40%, 88%]
Jeans -> Shirt [40%, 72%]
Shirt -> Jeans [40%, 72%]
Shirt -> Sweater [40%, 72%]
Sweater -> Shirt [40%, 80%]
Bait -> Bucket [40%, 61%]
Bucket -> Bait [40%, 66%]
Poster Tape -> Thumbtacks [50%, 90%]
Tape Thumbtacks -> Poster [50%, 100%]
Poster Thumbtacks -> Tape [50%, 90%]
Total execution time for Apriori Algorithm: 36ms
```

Total Time of Execution for Apriori Algorithm is: **36 ms**

```
Input Support, hit Enter, and input Confidence (in %): 40 60
Jeans -> Sweater [40%, 72%]
Sweater -> Jeans [40%, 80%]
Clorox wipes -> Sanitizer [50%, 90%]
Sanitizer -> Clorox wipes [50%, 83%]
Garlic -> Tomato [40%, 88%]
Tomato -> Garlic [40%, 80%]
Socks -> Underwear [50%, 83%]
Underwear -> Socks [50%, 100%]
Poster -> Thumbtacks [55%, 68%]
Thumbtacks -> Poster [55%, 100%]
Bait -> Fishing rod [55%, 84%]
Fishing rod -> Bait [55%, 91%]
Tape -> Thumbtacks [50%, 83%]
Thumbtacks -> Tape [50%, 90%]
Poster -> Tape [55%, 68%]
Tape -> Poster [55%, 91%]
Apple -> Orange [40%, 61%]
Orange -> Apple [40%, 88%]
Jeans -> Shirt [40%, 72%]
Shirt -> Jeans [40%, 72%]
Shirt -> Sweater [40%, 72%]
Sweater -> Shirt [40%, 80%]
Bait -> Bucket [40%, 61%]
Bucket -> Bait [40%, 66%]
Poster Tape -> Thumbtacks [50%, 90%]
Tape Thumbtacks -> Poster [50%, 100%]
Poster Thumbtacks -> Tape [50%, 90%]
Total execution time for Brute Pass: 8990ms
```

Total Time of Execution for Apriori Algorithm is: **8990 ms**

Apriori and Brute Force Algorithm Results, Support = 30% and Confidence = 85% (DBMaster)

```
Input Support, hit Enter, and input Confidence (in %): 30
85
Association Rules for Apriori Algorithm:
Garlic -> Tomato [40%, 88%]
Underwear -> Socks [50%, 100%]
Fishing rod -> Bait [55%, 91%]
Thumbtacks -> Tape [50%, 90%]
Banana -> Orange [35%, 87%]
Marker -> Pencil [30%, 85%]
Clorox wipes -> Sanitizer [50%, 90%]
Thumbtacks -> Poster [55%, 100%]
Banana -> Apple [35%, 87%]
Tape -> Poster [55%, 91%]
Orange -> Apple [40%, 88%]
Apple Banana -> Orange [30%, 85%]
Banana Orange -> Apple [30%, 85%]
Poster Tape -> Thumbtacks [50%, 90%]
Tape Thumbtacks -> Poster [50%, 100%]
Poster Thumbtacks -> Tape [50%, 90%]
Jeans Shirt -> Sweater [35%, 87%]
Shirt Sweater -> Jeans [35%, 87%]
Jeans Sweater -> Shirt [35%, 87%]
Garlic Onion -> Tomato [30%, 85%]
Onion Tomato -> Garlic [30%, 100%]
Bucket Fishing rod -> Bait [30%, 85%]
Total execution time for Apriori Algorithm: 40ms
```

```
Input Support, hit Enter, and input Confidence (in %): 30 85
Garlic -> Tomato [40%, 88%]
Underwear -> Socks [50%, 100%]
Fishing rod -> Bait [55%, 91%]
Thumbtacks -> Tape [50%, 90%]
Banana -> Orange [35%, 87%]
Marker -> Pencil [30%, 85%]
Clorox wipes -> Sanitizer [50%, 90%]
Thumbtacks -> Poster [55%, 100%]
Banana -> Apple [35%, 87%]
Tape -> Poster [55%, 91%]
Orange -> Apple [40%, 88%]
Poster Tape -> Thumbtacks [50%, 90%]
Tape Thumbtacks -> Poster [50%, 100%]
Poster Thumbtacks -> Tape [50%, 90%]
Apple Banana -> Orange [30%, 85%]
Banana Orange -> Apple [30%, 85%]
Jeans Shirt -> Sweater [35%, 87%]
Shirt Sweater -> Jeans [35%, 87%]
Jeans Sweater -> Shirt [35%, 87%]
Garlic Onion -> Tomato [30%, 85%]
Onion Tomato -> Garlic [30%, 100%]
Bucket Fishing rod -> Bait [30%, 85%]
Total execution time for Brute Pass: 9626ms
```

From these screenshots, it can be observed that the Brute Force algorithm takes approximately 250 times as long to run than the Apriori algorithm. This is because Brute Force generates all possible combinations of itemsets before pruning to generate association rules.

Apriori Algorithm Source Code

```
import java.io.IOException;
import java.util.*;
import java.io.*;

public class firstpass {

    public static void main( String[] args ) throws IOException {

    //Get Support and Confidence Input from user
    System.out.print("Input Support, hit Enter, and input Confidence (in %): ");
    Scanner sc = new Scanner(System.in);
    double support = sc.nextDouble();
    int support_actual = (int) (support * 20) / 100;
    int counter = 0;
    double confidence = sc.nextDouble();

    sc.close();

    if( support == 0 ) {
        System.out.println("INVALID SUPPORT");
        System.exit(0);
    }

    if( confidence == 0 ) {
        System.out.println("INVALID SUPPORT");
        System.exit(0);
    }

    String filePath = "DBMaster.csv";
    long startTime = System.nanoTime();
    //Master Hashmap to store frequent itemsets as keys with support values as values
    HashMap<List<String>, Integer> map_master = new HashMap<List<String>, Integer>();
    //Hashmaps to store itemsets
    HashMap<String, Integer> map = new HashMap<String, Integer>();
    HashMap<List<String>, Integer> map_double = new HashMap<List<String>, Integer>();
    HashMap<List<String>, Integer> map_triple = new HashMap<List<String>, Integer>();


    String line;
    BufferedReader reader = new BufferedReader(new FileReader(filePath));

    //---------------pass through data-----------------//

    while ( (line = reader.readLine()) != null ) {

            String[] parts = line.split(",");

            /* Append to Hashmap to store single Candidate sets */

            for( int i = 0; i<parts.length; i++ ) {
                if( !map.containsKey(parts[i]) ) {
                    map.put(parts[i], 1);
                    counter++;
                }
                else {
```

```
                map.put(parts[i], map.get(parts[i])+1);
            }
        }

    }


//iterate through the hashmap to see if the support is upheld
Set<String> hash_Set_single = new HashSet<String>();
for (String key : map.keySet()) {

        List<String> single_itemset = new Vector<String>();
        single_itemset.add(key);


        if( map.get(key) >= support_actual ) {

            /*add the qualified itemsets into a set and
            update master Hashmap*/
            hash_Set_single.add(key);
            map_master.put(single_itemset, map.get(key));

        }

    }

    reader.close();

//---------------pass through data-----------------//
String filePath2 = "DBMaster.csv";
String line2;
BufferedReader reader2 = new BufferedReader(new FileReader(filePath2));
while ( (line2 = reader2.readLine()) != null )
{

    String[] parts = line2.split(",");

    for( int i = 0; i<parts.length; i++ ) {

        //check if element is in frequent 1-itemset
        if( !hash_Set_single.contains(parts[i]) ) {
            continue;
        }

        else {

            for( int j = i+1; j<parts.length; j++ ) {

                //check if element in frequent 1-itemset
                if( !hash_Set_single.contains(parts[j]) ) {
                    continue;
                }

                else {

                    //create, add, and sort vector to store 2-itemset
                    List<String> double_itemset = new Vector<String>();
                    double_itemset.add(parts[i]);
                    double_itemset.add(parts[j]);
                    Collections.sort(double_itemset);
```

```
                        //add vector of double itemset to double hashmap
                        if( !map_double.containsKey(double_itemset) )
                            map_double.put(double_itemset, 1);
                        else
                            map_double.put( double_itemset,
map_double.get(double_itemset)+1 );
                    }

                }

            }

        }

    //Set to hold frequent 2-itemsets
    Set<List<String>> hash_Set_double = new HashSet<List<String>>();

    for (Map.Entry<List<String>,Integer> entry : map_double.entrySet()) {

        if( entry.getValue() >= support_actual ) {
            map_master.put(entry.getKey(), entry.getValue());
            hash_Set_double.add(entry.getKey());
        }

    }

    reader2.close();


    //---------------------pass through data---------------------------//

    String filePath3 = "DBMaster.csv";
    String line3;
    BufferedReader reader3 = new BufferedReader(new FileReader(filePath3));
    while ( (line3 = reader3.readLine()) != null )
    {

        String[] parts = line3.split(",");

        //first pointer
        for( int i = 0; i<parts.length; i++ ) {

            //check if in frequent 1-itemset
            if( !hash_Set_single.contains(parts[i]) ) {
                continue;
            }

            else {

                //second pointer
                for( int j = i+1; j<parts.length; j++ ) {

                    //check if in frequent 1-itemset
                    if( !hash_Set_single.contains(parts[j]) ) {
                        continue;
                    }
                        //create pair of first two items
                        List<String> first_double_itemset = new Vector<String>();
                        first_double_itemset.add(parts[i]);
```

```
                        first_double_itemset.add(parts[j]);
                        Collections.sort(first_double_itemset);

                    //check if in frequent 2-itemset
                    if( !hash_Set_double.contains(first_double_itemset) ) {
                        continue;
                    }


                    else {

                        //third pointer
                        for( int k = j+1; k<parts.length; k++ ) {

                            if( !hash_Set_single.contains(parts[k]) ) {
                                continue;
                            }

                            //create pair of second two items
                            List<String> second_double_itemset = new Vector<String>();
                            second_double_itemset.add(parts[k]);
                            second_double_itemset.add(parts[j]);
                            Collections.sort(second_double_itemset);

                            if( !hash_Set_double.contains(second_double_itemset) ) {
                                continue;
                            }

                            //create pair of first and third items
                            List<String> third_double_itemset = new Vector<String>();
                            third_double_itemset.add(parts[k]);
                            third_double_itemset.add(parts[i]);
                            Collections.sort(third_double_itemset);

                            if( !hash_Set_double.contains(third_double_itemset) ) {
                                continue;
                            }

                            else {

                                //append to hashtable with triple itemsets
                                List<String> triple_itemset = new Vector<String>();
                                triple_itemset.add(parts[k]);
                                triple_itemset.add(parts[i]);
                                triple_itemset.add(parts[j]);
                                Collections.sort(triple_itemset);

                                if( !map_triple.containsKey(triple_itemset) )
                                    map_triple.put(triple_itemset, 1);
                                else
                                    map_triple.put( triple_itemset,
map_triple.get(triple_itemset)+1 );

                            }

                        }

                    }

                }
```

```
            }

        }

    }

     //Set to hold frequent 3-itemsets
     Set<List<String>> hash_Set_triple = new HashSet<List<String>>();

     for (Map.Entry<List<String>,Integer> entry : map_triple.entrySet()) {

         if( entry.getValue() >= support_actual ) {
             map_master.put(entry.getKey(), entry.getValue());
             hash_Set_triple.add(entry.getKey());
         }

     }

    reader3.close();

    //------------generate association rules for double itemsets-------------//

    for( List<String> temp: hash_Set_double ) {

        //temp holds a list of two objects

        int temp_supp = 0; //holds temp's support value
        int left_val = 0; //hold temp's left support value
        int temp_conf = 0; //holds temp's confidence value

        //iterate through the master hashmap to find the support value associated with
the double itemsets
        for( Map.Entry<List<String>,Integer> entry : map_master.entrySet() ) {

            if( entry.getKey() == temp ) {
                temp_supp = entry.getValue();
            }

        }

        //iterate through temp to find the support values associated with the single
items within the double itemset
        for( int i = 0; i<temp.size(); i++ ) {

            //create a list to hold the values in the temp
            //list is used instead of string for type agreement
            List<String> temp_objects = new Vector<String>();
            temp_objects.add(temp.get(i));

            //assign the appropriate support values to the items within the double
itemset
            left_val = map_master.get(temp_objects);

            //calculate confidence
            temp_conf = (temp_supp * 100)/left_val;

            //perform confidence check with user inputted values
            if( temp_conf >= confidence ) {

                //for the first association rule
                if( i == 0 )
```

```
                    System.out.print(temp.get(i)+" -> "+temp.get(i+1));
                //for the second association rule
                else if( i == 1 )
                    System.out.print(temp.get(i)+" -> "+temp.get(i-1));

                System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%] ");
            }

        }

    }

    //--------------generate association rules for triple itemsets-------------//
    for( List<String> temp: hash_Set_triple ) {

        //temp holds a list of two objects

        int temp_supp = 0; //holds temp's support value
        int left_val = 0; //hold temp's left support value
        int temp_conf = 0; //holds temp's confidence value

        //iterate through the master hashmap to find the support value associated with
the double itemsets
        for( Map.Entry<List<String>,Integer> entry : map_master.entrySet() ) {

            if( entry.getKey() == temp ) {
                temp_supp = entry.getValue();
            }

        }

        //iterate through temp to find the support values associated with the single
items within the double itemset
        for( int i = 0; i<temp.size(); i++ ) {

            //create a list to hold the values in the temp
            //list is used instead of string for type agreement
            List<String> temp_objects = new Vector<String>();
            if( i == 0 ) {
                temp_objects.add(temp.get(i));
                temp_objects.add(temp.get(i+1));
                left_val = map_master.get(temp_objects);
                temp_conf = (temp_supp * 100)/left_val;
                if( temp_conf >= confidence ) {
                    System.out.print(temp.get(i)+" "+temp.get(i+1)+" ->
"+temp.get(i+2));
                    System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%]
");
                }
            }

            if( i == 1 ) {
                temp_objects.add(temp.get(i));
                temp_objects.add(temp.get(i+1));
                left_val = map_master.get(temp_objects);
                temp_conf = (temp_supp * 100)/left_val;
                if( temp_conf >= confidence ) {
                    System.out.print(temp.get(i)+" "+temp.get(i+1)+" -> "+temp.get(i-
1));
                    System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%]
");
```

```
                }
            }

            if( i == 2 ) {
                temp_objects.add(temp.get(i-2));
                temp_objects.add(temp.get(i));
                left_val = map_master.get(temp_objects);
                temp_conf = (temp_supp * 100)/left_val;
                if( temp_conf >= confidence ) {
                    System.out.print(temp.get(i-2)+" "+temp.get(i)+" -> "+temp.get(i-
1));
                    System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%]
");
                }
            }

        }

    }

    long elapsedTime = System.nanoTime() - startTime;
    System.out.println("Total execution time for Apriori Algorithm: "
                + elapsedTime/1000000+"ms");

    }

}
```

Brute Force Algorithm Source Code

```
import java.io.IOException;
import java.util.*;
import java.io.*;

public class brutepass1 {
    public static void main( String[] args ) throws IOException {

    //Get Support Input
    System.out.print("Input Support, hit Enter, and input Confidence (in %): ");
    Scanner sc = new Scanner(System.in);
    double support = sc.nextDouble();
    int support_actual = (int) (support * 20) / 100;
    int counter = 0;
    int counter1 =0;

    double confidence = sc.nextDouble();

    sc.close();

    String filePath = "DBMaster11.csv";
    long startTime = System.nanoTime();
    HashMap<List<String>, Integer> map_master = new HashMap<List<String>, Integer>();
//Hashmap to store frequent itemsets with support values
    HashMap<String, Integer> map = new HashMap<String, Integer>(); //Hashmap to store
single itemsets
    HashMap<List<String>, Integer> map_double = new HashMap<List<String>, Integer>();
//Hashmap to store double itemsets
    HashMap<List<String>, Integer> map_triple = new HashMap<List<String>, Integer>();
//Hashmap to store triple itemsets
    Set<List<String>> freq_itemset_brute = new HashSet<List<String>>();
    Set<List<String>> candidateKey = new HashSet<List<String>>();
    ArrayList<List<String>> transactions = new ArrayList<List<String>>();
    HashMap<List<String>, Integer> map_mastery_dummy = new HashMap<List<String>,
Integer>();

    String line;
    BufferedReader reader = new BufferedReader(new FileReader(filePath));
    int max = 0;


    //--------------pass through data-----------------//

    while ( (line = reader.readLine()) != null )
        {
            int maxtest = 0;
            String[] parts = line.split(",");
            List<String> holder_list_transactions = new Vector<String>();

            for( int i = 0; i<parts.length; i++ ) {
                holder_list_transactions.add(parts[i]);
                if( !map.containsKey(parts[i]) ) {
                    map.put(parts[i], 1);
                    maxtest++;

                }
```

```
                else {
                    map.put(parts[i], map.get(parts[i])+1);
                    maxtest++;
                }
            }
            Collections.sort(holder_list_transactions);

            transactions.add(holder_list_transactions);

            //get  size of largest itemset
            if( max < maxtest ) {
                max = maxtest;
            }

        }

    //iterate through the hashmap to see if the support is upheld
    Set<String> hash_Set_single = new HashSet<String>();
    List<String> hash_Set_single_list = new Vector<String>();
    for (String key : map.keySet()) {

            List<String> single_itemset = new Vector<String>();
            single_itemset.add(key);
            hash_Set_single_list.add(key);
            if( map.get(key) >= support_actual ) {


                //add the qualified itemsets into a set
                //frequent single itemset
                hash_Set_single.add(key);
                map_master.put(single_itemset, map.get(key));
            }

        }


    reader.close();


    //---------------pass through data-----------------//

            //first pointer
            for( int i = 0; i<hash_Set_single_list.size(); i++ ) {

                //second pointer
                for( int j = i+1; j<hash_Set_single_list.size(); j++ ) {

                    //create, add, and sort vector to store double itemset
                    List<String> double_itemset = new Vector<String>();
                    double_itemset.add(hash_Set_single_list.get(i));
                    double_itemset.add(hash_Set_single_list.get(j));
                    Collections.sort(double_itemset);

                    map_double.put(double_itemset, 0);
                    counter++;
                }

        }

    String filePath2 = "DBMaster11.csv";
    String line2;
```

```
    BufferedReader reader2 = new BufferedReader(new FileReader(filePath2));

    while ( (line2 = reader2.readLine()) != null )
    {

        String[] parts = line2.split(",");

        //first pointer
        for( int i = 0; i<parts.length; i++ ) {

                //second pointer
                for( int j = i+1; j<parts.length; j++ ) {

                        //create, add, and sort vector to store double itemset
                        List<String> double_itemset = new Vector<String>();
                        double_itemset.add(parts[i]);
                        double_itemset.add(parts[j]);
                        Collections.sort(double_itemset);

                        //add vector of double itemset to double hashmap C2
                        if( !map_double.containsKey(double_itemset) ) {
                            map_double.put(double_itemset, 1);
                        }

                        else {
                            map_double.put( double_itemset,
map_double.get(double_itemset)+1 );
                        }

                }

        }

    }

    //Set to hold frequent two itemsets
    Set<List<String>> hash_Set_double = new HashSet<List<String>>();

    for (Map.Entry<List<String>,Integer> entry : map_double.entrySet()) {

        freq_itemset_brute.add(entry.getKey());
        if( entry.getValue() >= support_actual ) {
            map_master.put(entry.getKey(), entry.getValue());
            hash_Set_double.add(entry.getKey());

        }

    }


    reader2.close();


    //---------------------subsequent passes---------------------------//
    Integer ki = 3;

    /*while loop to generate all possible itemsets as long as support is upheld
    if k-itemsets has no freqent sets, the loop terminates without generating k+1-
itemsets*/

    Boolean nomoreadditions = false;
```

```
    while( nomoreadditions == false ) {


        candidateKey = generateCanKey(freq_itemset_brute, ki);
        map_mastery_dummy = addtomap(candidateKey, transactions);

        HashMap<List<String>, Integer> map_master_temp = new HashMap<List<String>,
Integer>();
        for (Map.Entry<List<String>,Integer> entry : map_mastery_dummy.entrySet()) {

            if( entry.getValue() >= support_actual ) {
                map_master.put(entry.getKey(), entry.getValue());
                map_master_temp.put(entry.getKey(), entry.getValue());

            }

        }
        if( map_master_temp.isEmpty() ) {
            nomoreadditions = true;
        }


        freq_itemset_brute.clear();

        for( List<String> entry : candidateKey ) {
            freq_itemset_brute.add(entry);
        }

        ki++;

    }

        //first pointer
        for( int i = 0; i<hash_Set_single_list.size(); i++ ) {

            //second pointer
            for( int j = i+1; j<hash_Set_single_list.size(); j++ ) {

                //third pointer
                    for( int k = j+1; k<hash_Set_single_list.size(); k++ ) {

                            //append to hashtable with triple itemsets C3
                            List<String> triple_itemset = new Vector<String>();
                            triple_itemset.add(hash_Set_single_list.get(i));
                            triple_itemset.add(hash_Set_single_list.get(j));
                            triple_itemset.add(hash_Set_single_list.get(k));
                            Collections.sort(triple_itemset);
                            map_triple.put(triple_itemset, 0);
                            counter1++;

                    }

            }

    }


    String filePath3 = "DBMaster11.csv";
    String line3;
    BufferedReader reader3 = new BufferedReader(new FileReader(filePath3));
    while ( (line3 = reader3.readLine()) != null )
```

```
    {

        String[] parts = line3.split(",");

        //first pointer
        for( int i = 0; i<parts.length; i++ ) {

                //second pointer
                for( int j = i+1; j<parts.length; j++ ) {

                    //third pointer
                        for( int k = j+1; k<parts.length; k++ ) {

                                //append to hashtable with triple itemsets C3
                                List<String> triple_itemset = new Vector<String>();
                                triple_itemset.add(parts[k]);
                                triple_itemset.add(parts[i]);
                                triple_itemset.add(parts[j]);
                                Collections.sort(triple_itemset);

                                if( !map_triple.containsKey(triple_itemset) )
                                    map_triple.put(triple_itemset, 1);
                                else
                                    map_triple.put( triple_itemset,
map_triple.get(triple_itemset)+1 );

                        }

                }

        }

    }

     //Set to hold frequent three itemsets
     Set<List<String>> hash_Set_triple = new HashSet<List<String>>();

     for (Map.Entry<List<String>,Integer> entry : map_triple.entrySet()) {

         if( entry.getValue() >= support_actual ) {
             //map_master.put(entry.getKey(), entry.getValue());
             hash_Set_triple.add(entry.getKey());
         }

     }

     reader3.close();

    // //------------generate association rules for double itemsets-------------//

    for( List<String> temp: hash_Set_double ) {

        //temp holds a list of two objects

        int temp_supp = 0; //holds temp's support value
        int left_val = 0; //hold temp's left support value
        int temp_conf = 0; //holds temp's confidence value

        //iterate through the master hashmap to find the support value associated with
the double itemsets
        for( Map.Entry<List<String>,Integer> entry : map_master.entrySet() ) {
```

```
            if( entry.getKey() == temp ) {
                temp_supp = entry.getValue();
            }

        }

        //iterate through temp to find the support values associated with the single
items within the double itemset
        for( int i = 0; i<temp.size(); i++ ) {

            //create a list to hold the values in the temp
            //list is used instead of string for type agreement
            List<String> temp_objects = new Vector<String>();
            temp_objects.add(temp.get(i));

            //assign the appropriate support values to the items within the double
itemset
            left_val = map_master.get(temp_objects);

            //calculate confidence
            temp_conf = (temp_supp * 100)/left_val;

            //perform confidence check with user inputted values
            if( temp_conf >= confidence ) {
                //for the first association rule
                if( i == 0 )
                    System.out.print(temp.get(i)+" -> "+temp.get(i+1));
                //for the second association rule
                else if( i == 1 )
                    System.out.print(temp.get(i)+" -> "+temp.get(i-1));

                System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%] ");
            }

        }

    }

    //--------------generate association rules for triple itemsets-------------//
    for( List<String> temp: hash_Set_triple ) {

        //temp holds a list of two objects

        int temp_supp = 0; //holds temp's support value
        int left_val = 0; //hold temp's left support value
        int temp_conf = 0; //holds temp's confidence value

        temp_supp = map_master.get(temp);

        //iterate through the master hashmap to find the support value associated with
the double itemsets
        // for( Map.Entry<List<String>,Integer> entry : map_master.entrySet() ) {

        //     if( entry.getKey() == temp ) {
        //         temp_supp = entry.getValue();
        //     }

        // }
```

```
        //iterate through temp to find the support values associated with the single
items within the double itemset
        for( int i = 0; i<temp.size(); i++ ) {

            //create a list to hold the values in the temp
            //list is used instead of string for type agreement
            List<String> temp_objects = new Vector<String>();
            if( i == 0 ) {
                temp_objects.add(temp.get(i));
                temp_objects.add(temp.get(i+1));
                left_val = map_master.get(temp_objects);
                temp_conf = (temp_supp * 100)/left_val;
                if( temp_conf >= confidence ) {
                    System.out.print(temp.get(i)+" "+temp.get(i+1)+" ->
"+temp.get(i+2));
                    System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%]
");
                }
            }

            if( i == 1 ) {
                temp_objects.add(temp.get(i));
                temp_objects.add(temp.get(i+1));
                left_val = map_master.get(temp_objects);
                temp_conf = (temp_supp * 100)/left_val;
                if( temp_conf >= confidence ) {
                    System.out.print(temp.get(i)+" "+temp.get(i+1)+" -> "+temp.get(i-
1));
                    System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%]
");
                }
            }

            if( i == 2 ) {
                temp_objects.add(temp.get(i-2));
                temp_objects.add(temp.get(i));
                left_val = map_master.get(temp_objects);
                temp_conf = (temp_supp * 100)/left_val;
                if( temp_conf >= confidence ) {
                    System.out.print(temp.get(i-2)+" "+temp.get(i)+" -> "+temp.get(i-
1));
                    System.out.println(" ["+(temp_supp*100)/20+"%"+", "+temp_conf+"%]
");
                }
            }

        }

    }

    long elapsedTime = System.nanoTime() - startTime;
    System.out.println("Total execution time for Brute Pass: "
                + elapsedTime/1000000+"ms");

    }

    public static Set<List<String>> generateCanKey(Set<List<String>> seedKey, Integer
setSize) {
        int counter2 = 0;
        Set<List<String>> canKey = new HashSet<List<String>>();
        for( List<String> canKeyItemA : seedKey ) {
```

```
            for( List<String> canKeyItemB : seedKey ) {

                List<String> temp = new Vector<String>();

                if( canKeyItemA != canKeyItemB ) {
                    temp.addAll(canKeyItemA);
                    temp.addAll(canKeyItemB);
                    Set<String> removedups = new HashSet<>(temp);
                    temp = new Vector<String>(removedups); //to remove duplicates
                    Collections.sort(temp);
                }
                if( !canKey.contains(temp) && temp.size() == setSize ) {
                    canKey.add(temp);
                    counter2++;
                }
            }
        }

        return canKey;

    }

    public static HashMap<List<String>, Integer> addtomap( Set<List<String>> canKey,
ArrayList<List<String>> trans  ) {

        HashMap<List<String>, Integer> map_mastery_dummy = new HashMap<List<String>,
Integer>();
        for( List<String> canKeyitem : canKey  ) {
            map_mastery_dummy.put( canKeyitem, 0);
        }

        for( List<String> transitem: trans ) {


            for( List<String> canKeyitem : canKey  ) {
                Boolean canKeyitemexists = true;
                for( String indiv_item : canKeyitem ) {
                    if( !transitem.contains(indiv_item) ) {
                        canKeyitemexists = false;
                        break;
                    }
                }

                if( canKeyitemexists ) {
                    map_mastery_dummy.put( canKeyitem, map_mastery_dummy.get(canKeyitem)
+ 1 );
                }

            }
        }
        return map_mastery_dummy;

    }


}
```