CS634 midterm project

Wen Jiao

February 20, 2021

Part 1: Apriori algorithm:

Source code:

```
package cs634_midterm;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;


public class Apriori {
     private static double min_support;
     private static double min_confidence;
     static List<List<String>> record=new
ArrayList<List<String>>();
     static List<List<String>> frequentIS=new
ArrayList<List<String>>();
     static boolean endTag;
     static int times=2;

     public static void main(String[] args) {
          System.out.println("please input the min_support
value");
          Scanner in=new Scanner(System.in);
          min_support=in.nextDouble();
          System.out.println("the min_support value is
"+min_support);
          System.out.println("please input the min_condidence
value");
          min_confidence=in.nextDouble();
          System.out.println("the min_confidence value is
"+min_confidence);
          System.out.println();
          in.close();
          long start=System.currentTimeMillis();//count time
from here
```

```
          record=getrecord("./DataSet01.txt");//scan data-set
          //showData(record); for test

          apriori_alg();//use Apriori algorithm to generate
frequent item-sets
          //System.out.println("Frequent item-sets is
following:"); for test
          //showData(frequentIS); for test
          System.out.println();


          AssociationRulesMining();//generate all the
association rules
          long end=System.currentTimeMillis();//count time until
here
          System.out.println("Executing time: "+(end-start)
+"ms");


     }

     /**
      * method description: read transaction records from
data_set file
      * @param path
      * @return
      */
     private static List<List<String>> getrecord(String path){
          try {
               File dataset=new File(path);
               if(dataset.isFile()&&dataset.exists()) {
                    InputStreamReader read=new
InputStreamReader(new FileInputStream(dataset));
                    BufferedReader buffered_reader=new
BufferedReader(read);
                    String one_line;
                    while((one_line=buffered_reader.readLine())!
=null) {
                         String[] one_record=one_line.split(",
|:");
                         List<String> lineList=new
ArrayList<String>();
                         for(int i=1;i<one_record.length;i++) {
                              lineList.add(one_record[i]);
                         }
```

```
                    Collections.sort(lineList);
                    record.add(lineList);
                }
                read.close();
            }
            else if(dataset.isDirectory()&&dataset.exists())
{
                System.out.println("Path is a directory!");
            }

            else System.out.println("Can't find the file");
        }
        catch(Exception e){
            e.printStackTrace();
        }
        System.out.println("The input data is following:");
        showData(record);
        System.out.println();
        return record;
    }

    /**
     * method description: implementation of Apriori Algorithm
     */
    private static void apriori_alg() {
        List<List<String>> tempFList;

        //definition of OneD_CandidateSet: all the candidate
item-sets only have one item
        List<List<String>>
OneD_CandidateSet=get1stCandidateSet();
        //showData(OneD_CandidateSet)
        //definition of OneD_FrequentSet: all the frequent
item-sets only have one item
        List<List<String>>
OneD_FrequentSet=getNthFrequentSet(OneD_CandidateSet);
        frequentIS.addAll(OneD_FrequentSet);
        //System.out.println("1D frequent item-set:"); for
test
        //showData(OneD_FrequentSet); for test
        tempFList=OneD_FrequentSet;


        while(endTag==false) {
            List<List<String>>
nextCandidateSet=getNextCandidateSet(tempFList);
```

```
                List<List<String>>
nextFrequentSet=getNthFrequentSet(nextCandidateSet);
                frequentIS.addAll(nextFrequentSet);
                if(endTag==false) {
                     System.out.println(times+"D frequent item-
set:");
                     showData(nextFrequentSet);
                     times++;
                     tempFList=nextFrequentSet;
                }

           }
     }

     /**
      * method description: find the 1st candidate set(all the
unique items)
      * @return
      */
     private static List<List<String>> get1stCandidateSet(){
          List<List<String>> tablelist=new
ArrayList<List<String>>();
          HashSet<String> hs=new HashSet<String>();
          for(int i=0;i<record.size();i++) {
               for(int j=0;j<record.get(i).size();j++) {
                    hs.add(record.get(i).get(j));
               }
          }

          Iterator<String> iterator=hs.iterator();
          while(iterator.hasNext()) {
               List<String> tempList=new ArrayList<String>();

               tempList.add(iterator.next());
               tablelist.add(tempList);
          }
          return tablelist;
     }

     /**
      * get frequent-sets from nth candidate-set
      * @param NthcandidateSet
      * @return
      */
     private static List<List<String>>
getNthFrequentSet(List<List<String>> NthcandidateSet){
```

```
            int count;
            boolean endTag_=true;
            List<List<String>> nthFrequentSet=new
ArrayList<List<String>>();

            for(int i=0;i<NthcandidateSet.size();i++) {
                  count=countFrequency(NthcandidateSet.get(i));
                  if(count>=min_support*record.size()) {
                        nthFrequentSet.add(NthcandidateSet.get(i));
                        //map.add(new
Mymap(NthcandidateSet.get(i),count));
                        endTag_=false;
                  }
            }
            endTag=endTag_;
            if(endTag==true) {
                  System.out.println("can't find more frequent-
sets!");
            }
            return nthFrequentSet;
      }

      /**
       * method description: count the times that one item-set in
nth-candidateSet occurs
       * @param candidateSet
       * @return
       */
      private static int countFrequency(List<String>
candidateSet) {
            int count=0;
            for(int i=0;i<record.size();i++) {
                  boolean countTag=true;
                  for(int j=0;j<candidateSet.size();j++) {
                        String temp=candidateSet.get(j);
                        if(!record.get(i).contains(temp)) {
                              countTag=false;
                              break;
                        }
                  }
                  if(countTag==true) count++;
            }
            return count;
      }

      /**
```

```
      * method description: use the last frequent item-sets to
generate the next candidate item-sets
      * @param LastFrequentSet
      * @return
      */
     private static List<List<String>>
getNextCandidateSet(List<List<String>> LastFrequentSet){
          List<List<String>> nextCandidateSet=new
ArrayList<List<String>>();
          List<List<String>> afterJoining=new
ArrayList<List<String>>();
          int dimension=LastFrequentSet.get(0).size();
          for(int i=0;i<LastFrequentSet.size();i++) {//join
               for(int j=i+1;j<LastFrequentSet.size();j++) {
                    boolean joinable=true;
                    for(int k=0;k<dimension-1;k++) {//Step 1 of
checking if two item-set can be joined
                         if(LastFrequentSet.get(i).get(k)!
=LastFrequentSet.get(j).get(k)) {
                              joinable=false;
                         }
                    }

if(LastFrequentSet.get(i).get(dimension-1)==LastFrequentSet.get(
j).get(dimension-1)) {//Step 2 of checking two item-set can be
joined
                         joinable=false;
                    }
                    if(joinable==true) {
                         List<String> temp=new
ArrayList<String>();
                         temp.addAll(LastFrequentSet.get(i));

temp.add(LastFrequentSet.get(j).get(dimension-1));
                         Collections.sort(temp);
                         afterJoining.add(temp);
                    }
               }
          }
          //showData(afterJoining); For test

          for(int i=0;i<afterJoining.size();i++) {//remove item-
set whose subsets are not all in the last frequent item-set
               boolean allSubsetOccur=true;
               for(int j=0;j<dimension;j++) {
                    String item=afterJoining.get(i).remove(j);
```

```
                    List<String> temp=afterJoining.get(i);
                    if(!LastFrequentSet.contains(temp)) {
                         allSubsetOccur=false;
                    }
                    afterJoining.get(i).add(j,item);//return
afterJoining to the original
               }
               if(allSubsetOccur==true) {
                    nextCandidateSet.add(afterJoining.get(i));
               }
          }
          if(nextCandidateSet.isEmpty()) {
               System.out.println("Can't generate next
candidateSet!");
          }
          return nextCandidateSet;
     }

     private static void showData(List<List<String>> list) {
          for(int i=0;i<list.size();i++) {
               int temp=list.get(i).size()-1;
               for(int j=0;j<temp;j++) {
                    System.out.print(list.get(i).get(j)+", ");
               }
               System.out.print(list.get(i).get(temp));
               System.out.println();
          }
     }

     private static void AssociationRulesMining() {
          for(int i=0;i<frequentIS.size();i++) {
               List<String> temp=frequentIS.get(i);
               if(temp.size()>1) {
                    List<String> tempClone=new
ArrayList<String>();
                    tempClone.addAll(temp);
                    List<List<String>>
allSubset=getSubsets(tempClone);

                    for(int j=0;j<allSubset.size();j++) {
                         List<String> s1=allSubset.get(j);
                         List<String> s2=getRest(temp,s1);
                         double
confidence=isAssociationRule(s1,s2,temp);
                         if(confidence>=min_confidence) {
```

```
                                double support=getCount(temp)*1.0/
record.size();
                                System.out.println("[Support=
"+support+", confidence= "+confidence+"]");
                           }
                      }
                 }
           }

     }

     private static List<List<String>> getSubsets(List<String>
list) {
           List<List<String>> result=new
ArrayList<List<String>>();
           int length=list.size();
           int numberOfSubsets=length==0?0:1<<(length);

           for(int i=1;i<numberOfSubsets-1;i++) {//Starting from
1 because we don't need empty set
                List<String> subset=new ArrayList<String>();

                int index=i;
                for(int j=0;j<list.size();j++) {
                     if((index&1)==1) {
                          subset.add(list.get(j));
                     }
                     index=index>>1;
                }
                Collections.sort(subset);
                result.add(subset);

           }
           return result;
     }

     /**
      * method description: extract s1 from temp
      * @param temp
      * @param s1
      * @return
      */
     private static List<String> getRest(List<String> temp,
List<String> s1){
           List<String> result=new ArrayList<String>();
```

```
            for(int i=0;i<temp.size();i++) {
                String item=temp.get(i);
                if(!s1.contains(item))
                    result.add(item);
            }
            return result;
        }

        /**
         * method description: check if a subset of a item-set can
generate an association rule
         * @param s1
         * @param s2
         * @param temp
         * @return
         */
        private static double isAssociationRule(List<String>
s1,List<String> s2,List<String> temp) {
            double confidence;
            int counts1;
            int counttemp;

            if(temp.size()!=0&&temp!=null) {
                counts1=getCount(s1);
                //System.out.println(counts1);
                counttemp=getCount(temp);
                //System.out.println(counttemp);
                confidence=counttemp*1.0/counts1;

                if(confidence>=min_confidence) {
                    System.out.print("Association rule:
"+s1.toString()+"-->"+s2.toString()+"  ");
                    return confidence;
                }
                else return 0;
            }
            else return 0;
        }

        private static int getCount(List<String> list) {
            int count=0;
            for(int i=0;i<record.size();i++) {
                boolean tag=true;
                for(int j=0;j<list.size();j++) {
                    String temp=list.get(j);
                    if(!record.get(i).contains(temp)) {
```

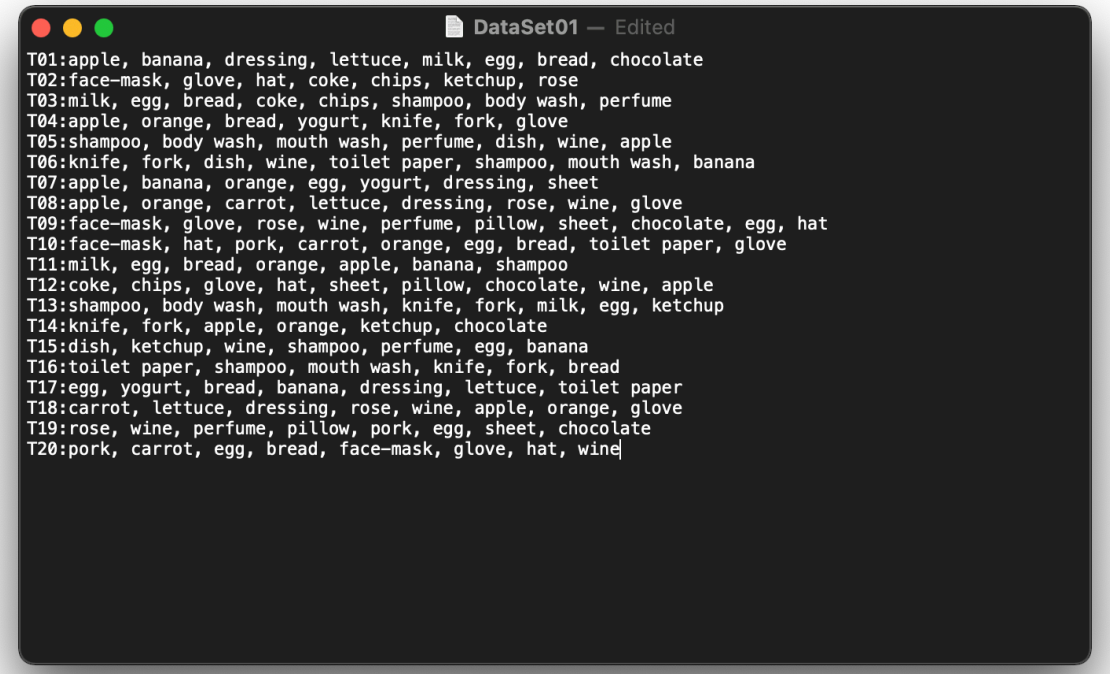
```
                        tag=false;
                        break;
                    }
                }
                if(tag==true) count++;
            }
            return count;
        }

}
```

Screenshots:

Data set 01:

Scenario: a normal day in Walmart

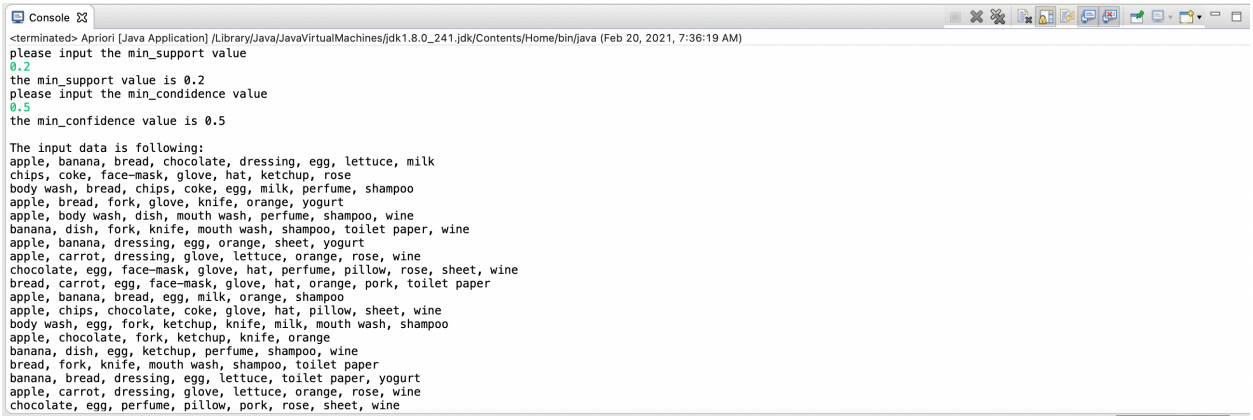
Screenshot of DataSet01.txt:

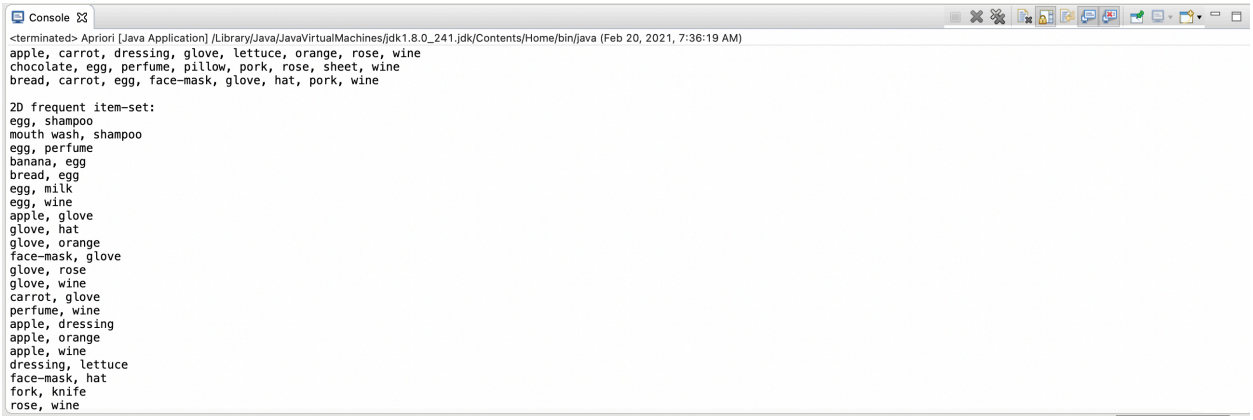


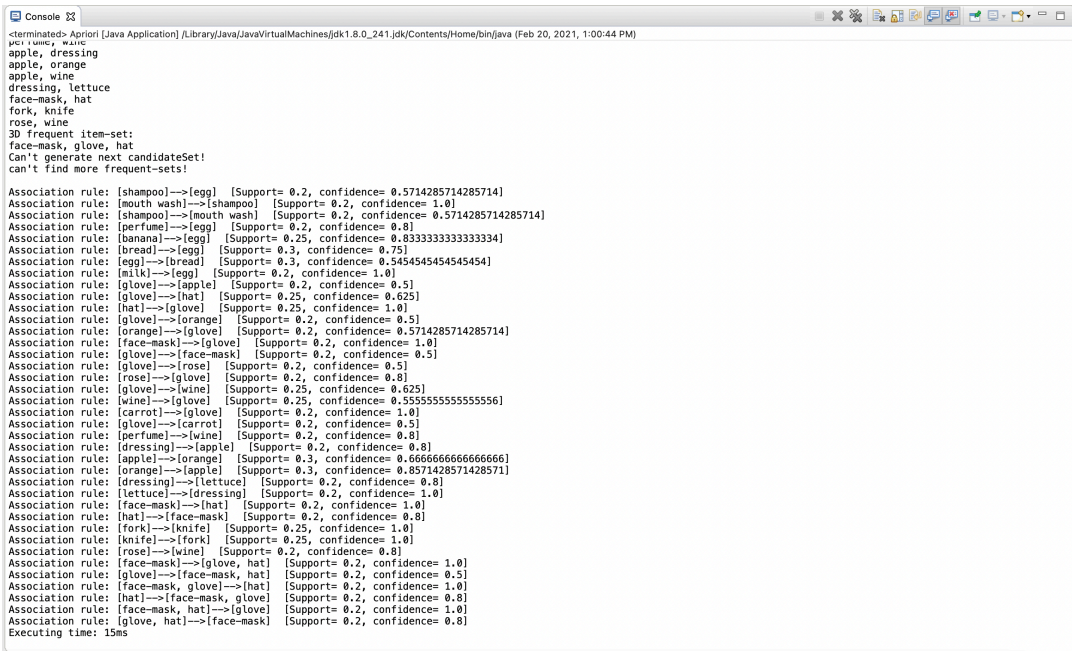
The min_support value which is decided by user is 0.2

The min_confidence value which is decided by user is 0.5

Screenshots of output:



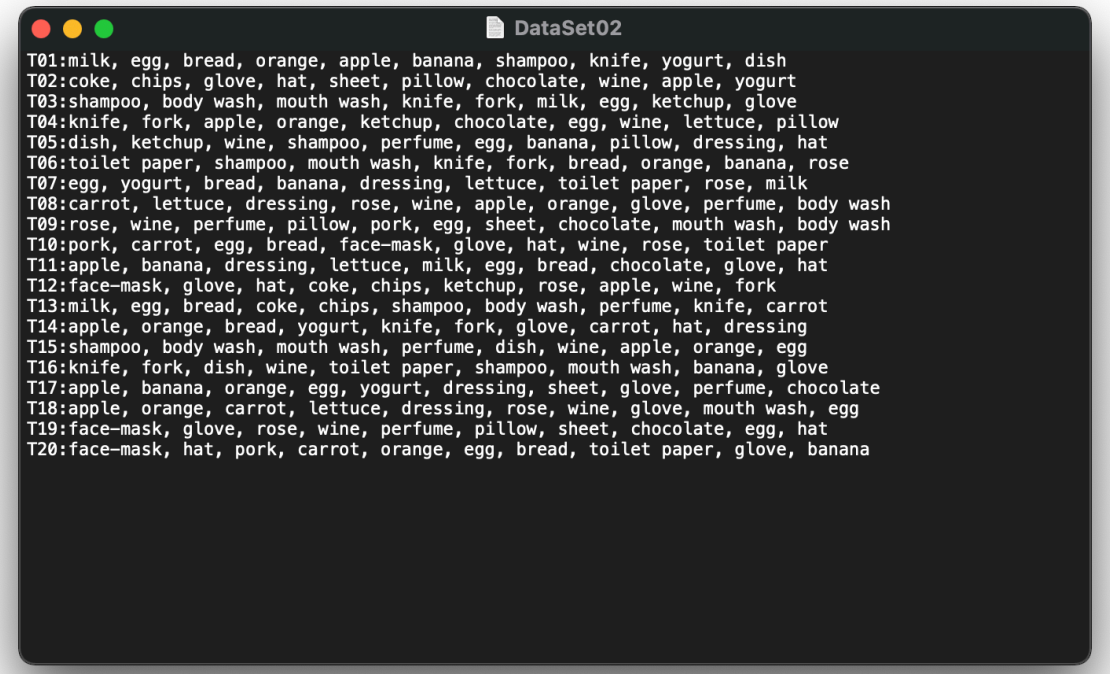




Data set 02:

Scenario: another normal day in Walmart

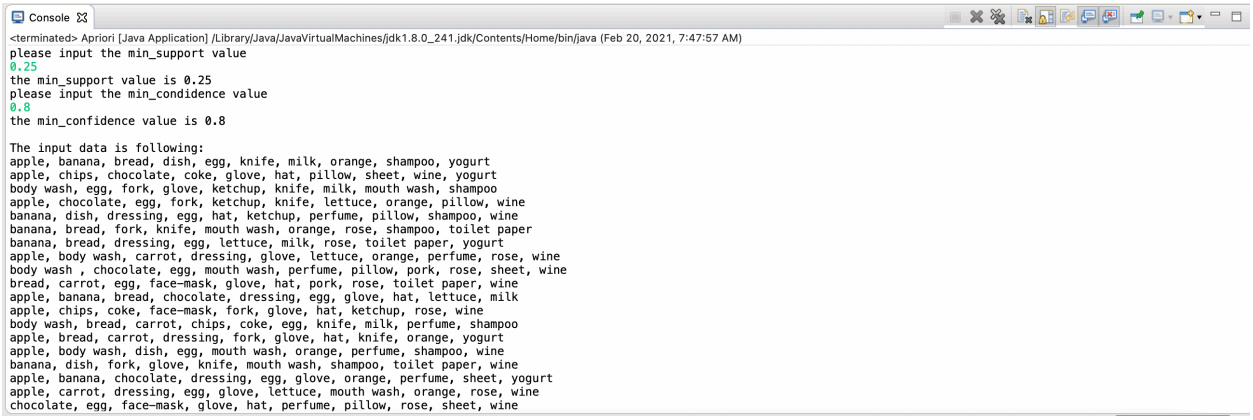
Screenshot of DataSet02.txt

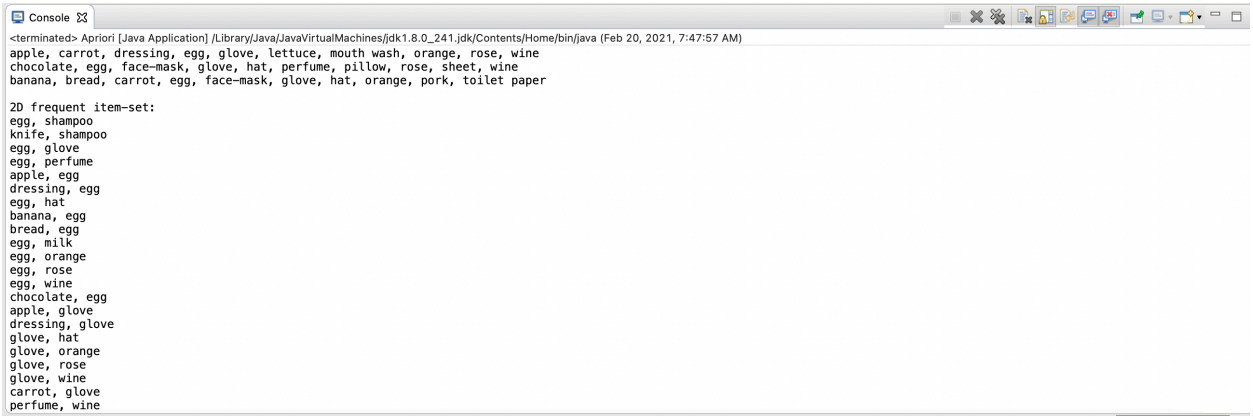


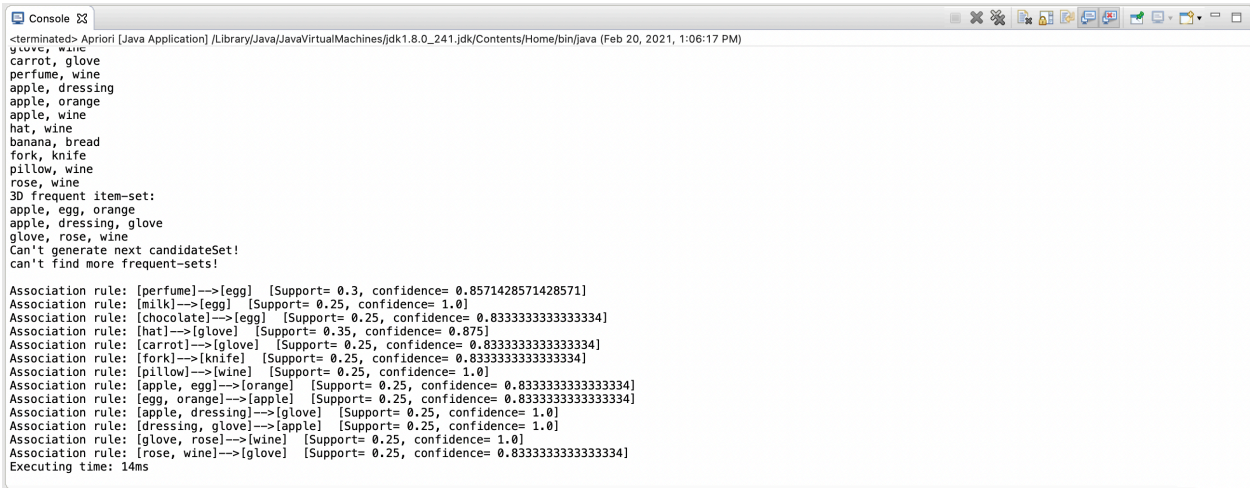
The min_support value is 0.25

The min_confidence value is 0.8

Screenshots of output:



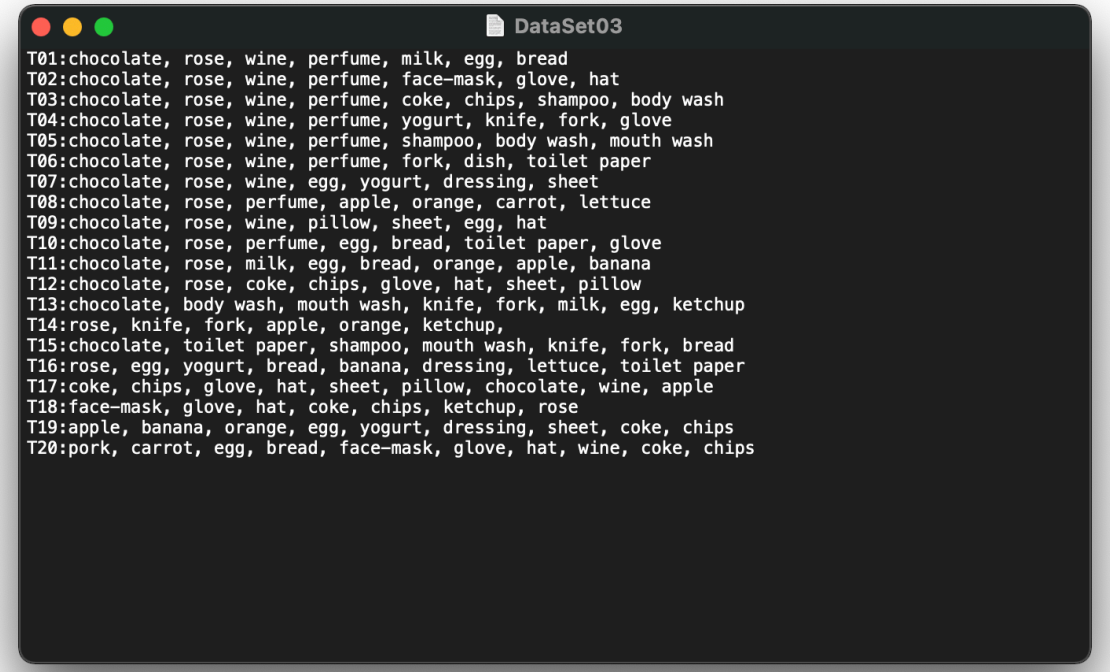




Data set 03:

Scenario: Valentine’s day in Walmart

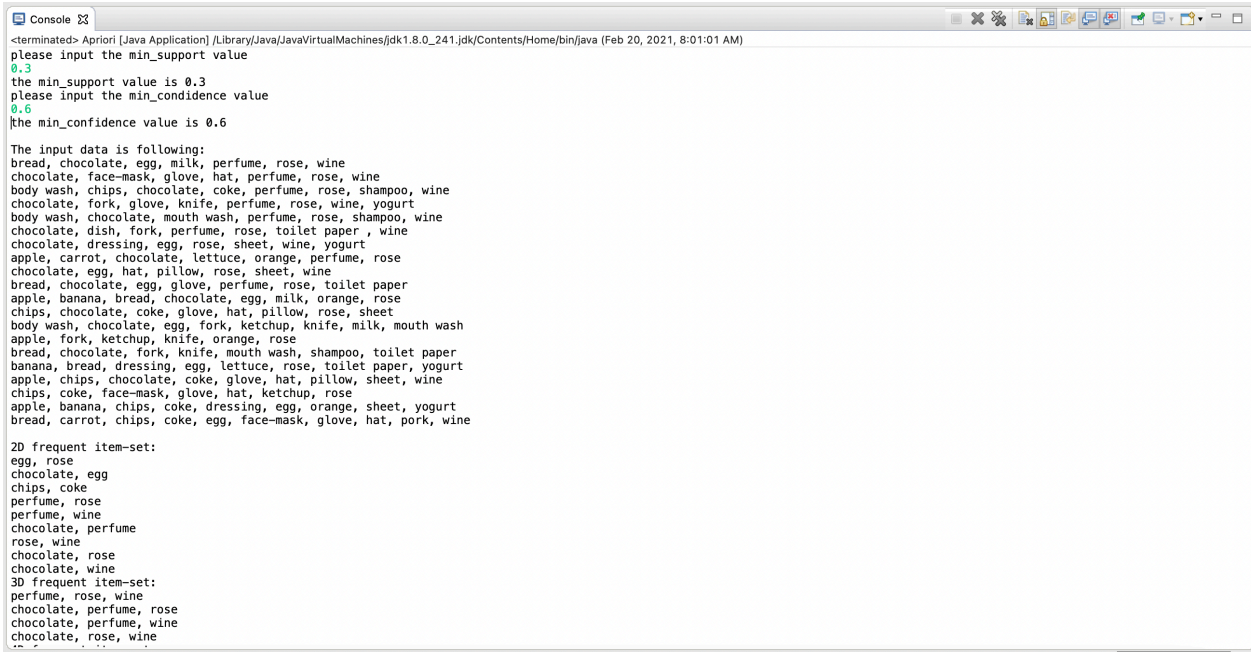
Screenshot of DataSet03.txt:



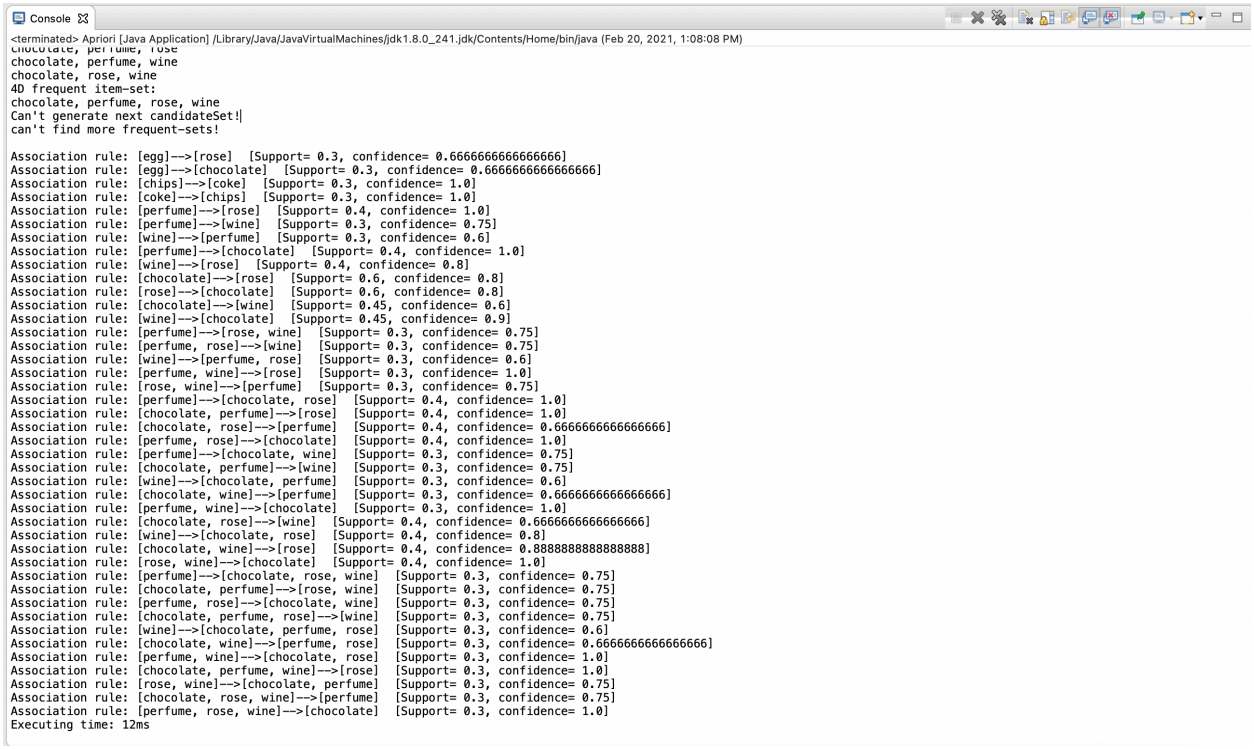
The min_support value is 0.3

The min_confidence value is 0.6

Screenshots of output:



```
Console
<terminated> Apriori [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 8:01:01 AM)
please input the min_support value
0.3
the min_support value is 0.3
please input the min_condidence value
0.6
the min_confidence value is 0.6

The input data is following:
bread, chocolate, egg, milk, perfume, rose, wine
chocolate, face-mask, glove, hat, perfume, rose, wine
body wash, chips, chocolate, coke, perfume, rose, shampoo, wine
chocolate, fork, glove, knife, perfume, rose, wine, yogurt
body wash, chocolate, mouth wash, perfume, rose, shampoo, wine
chocolate, dish, fork, perfume, rose, toilet paper , wine
chocolate, dressing, egg, rose, sheet, wine, yogurt
apple, carrot, chocolate, lettuce, orange, perfume, rose
chocolate, egg, hat, pillow, rose, sheet, wine
bread, chocolate, egg, glove, perfume, rose, toilet paper
apple, banana, bread, chocolate, egg, milk, orange, rose
chips, chocolate, coke, glove, hat, pillow, rose, sheet
body wash, chocolate, egg, fork, ketchup, knife, milk, mouth wash
apple, fork, ketchup, knife, orange, rose
bread, chocolate, fork, knife, mouth wash, shampoo, toilet paper
banana, bread, dressing, egg, lettuce, rose, toilet paper, yogurt
apple, chips, chocolate, coke, glove, hat, pillow, sheet, wine
chips, coke, face-mask, glove, hat, ketchup, rose
apple, banana, chips, coke, dressing, egg, orange, sheet, yogurt
bread, carrot, chips, coke, egg, face-mask, glove, hat, pork, wine

2D frequent item-set:
egg, rose
chocolate, egg
chips, coke
perfume, rose
perfume, wine
chocolate, perfume
rose, wine
chocolate, rose
chocolate, wine
3D frequent item-set:
perfume, rose, wine
chocolate, perfume, rose
chocolate, perfume, wine
chocolate, rose, wine
```

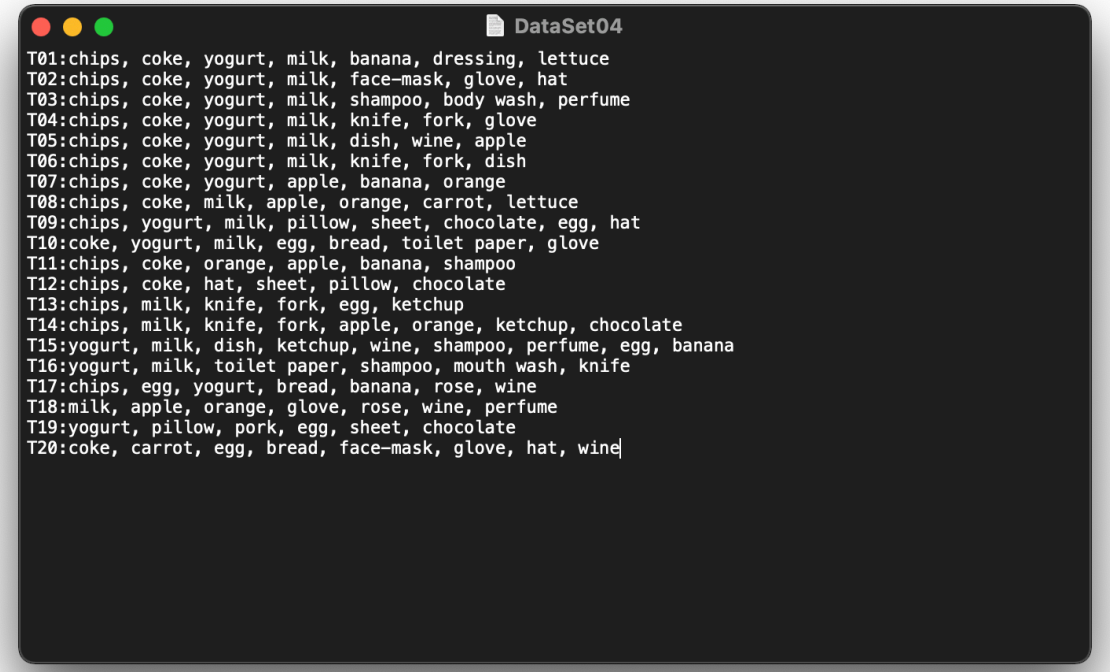


```
Console
<terminated> Apriori [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 1:08:08 PM)
chocolate, perfume, rose
chocolate, perfume, wine
chocolate, rose, wine
4D frequent item-set:
chocolate, perfume, rose, wine
Can't generate next candidateSet!
can't find more frequent-sets!

Association rule: [egg]-->[rose]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [egg]-->[chocolate]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [chips]-->[coke]  [Support= 0.3, confidence= 1.0]
Association rule: [coke]-->[chips]  [Support= 0.3, confidence= 1.0]
Association rule: [perfume]-->[rose]  [Support= 0.4, confidence= 1.0]
Association rule: [perfume]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[perfume]  [Support= 0.3, confidence= 0.6]
Association rule: [perfume]-->[chocolate]  [Support= 0.4, confidence= 1.0]
Association rule: [wine]-->[rose]  [Support= 0.4, confidence= 0.8]
Association rule: [chocolate]-->[rose]  [Support= 0.6, confidence= 0.8]
Association rule: [rose]-->[chocolate]  [Support= 0.6, confidence= 0.8]
Association rule: [chocolate]-->[wine]  [Support= 0.45, confidence= 0.6]
Association rule: [wine]-->[chocolate]  [Support= 0.45, confidence= 0.9]
Association rule: [perfume]-->[rose, wine]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume, rose]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[perfume, rose]  [Support= 0.3, confidence= 0.6]
Association rule: [perfume, wine]-->[rose]  [Support= 0.3, confidence= 1.0]
Association rule: [rose, wine]-->[perfume]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume]-->[chocolate, rose]  [Support= 0.4, confidence= 1.0]
Association rule: [chocolate, perfume]-->[rose]  [Support= 0.4, confidence= 1.0]
Association rule: [chocolate, rose]-->[perfume]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [perfume, rose]-->[chocolate]  [Support= 0.4, confidence= 1.0]
Association rule: [perfume]-->[chocolate, wine]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, perfume]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[chocolate, perfume]  [Support= 0.3, confidence= 0.6]
Association rule: [chocolate, wine]-->[perfume]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [perfume, wine]-->[chocolate]  [Support= 0.3, confidence= 1.0]
Association rule: [chocolate, rose]-->[wine]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [wine]-->[chocolate, rose]  [Support= 0.4, confidence= 0.8]
Association rule: [chocolate, wine]-->[rose]  [Support= 0.4, confidence= 0.8888888888888888]
Association rule: [rose, wine]-->[chocolate]  [Support= 0.4, confidence= 1.0]
Association rule: [perfume]-->[chocolate, rose, wine]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, perfume]-->[rose, wine]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume, rose]-->[chocolate, wine]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, perfume, rose]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[chocolate, perfume, rose]  [Support= 0.3, confidence= 0.6]
Association rule: [chocolate, wine]-->[perfume, rose]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [perfume, wine]-->[chocolate, rose]  [Support= 0.3, confidence= 1.0]
Association rule: [chocolate, perfume, wine]-->[rose]  [Support= 0.3, confidence= 1.0]
Association rule: [rose, wine]-->[chocolate, perfume]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, rose, wine]-->[perfume]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume, rose, wine]-->[chocolate]  [Support= 0.3, confidence= 1.0]
Executing time: 12ms
```

Data set 04:

Scenario: 4 items (chips, coke, yogurt, milk) are on sale in Walmart

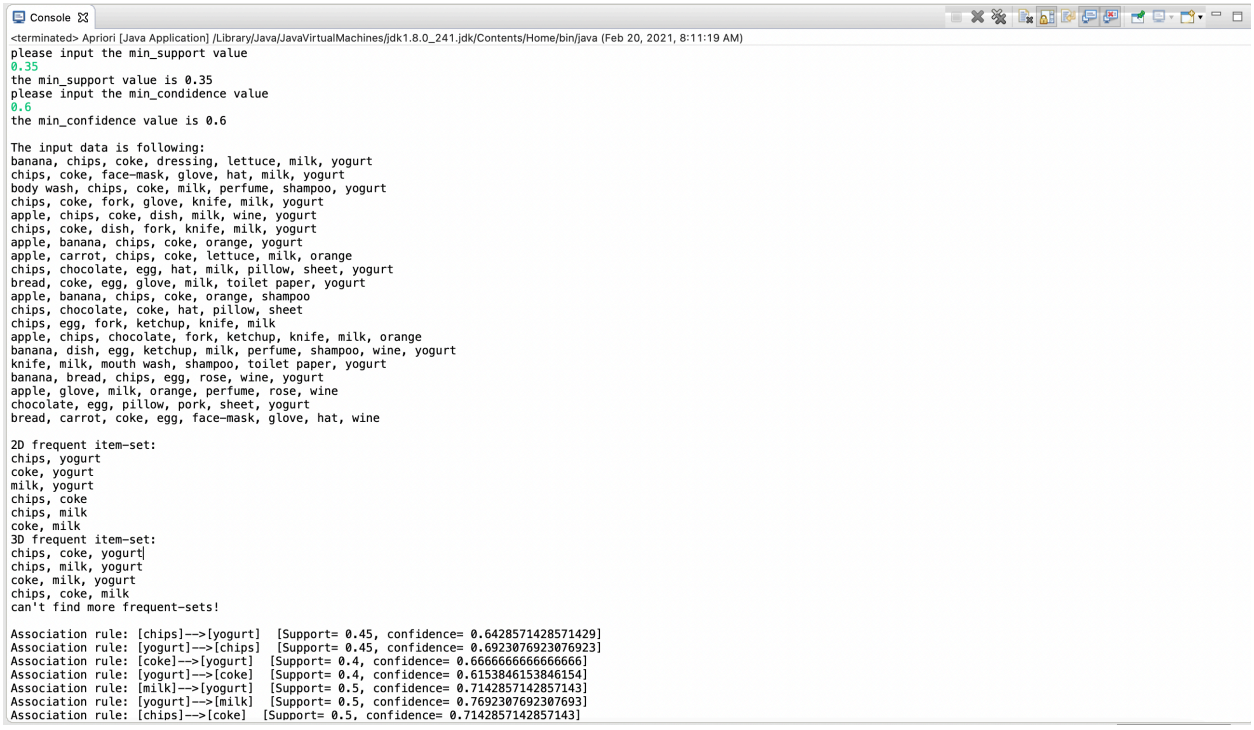
Screenshot of DataSet04.txt:



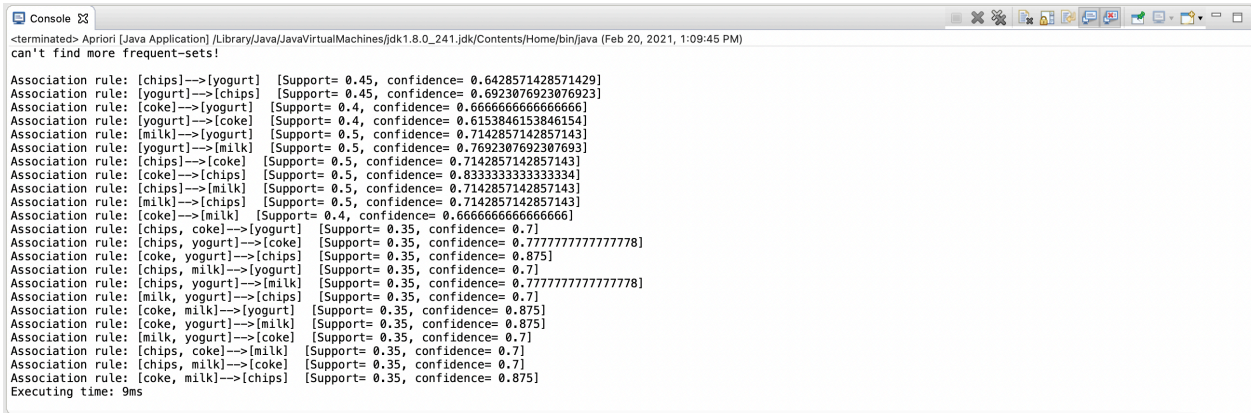
The min_support value is 0.35

The min_confidence value is 0.6

Screenshots of output:



```
Console
<terminated> Apriori [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 8:11:19 AM)
please input the min_support value
0.35
the min_support value is 0.35
please input the min_condidence value
0.6
the min_confidence value is 0.6

The input data is following:
banana, chips, coke, dressing, lettuce, milk, yogurt
chips, coke, face-mask, glove, hat, milk, yogurt
body wash, chips, coke, milk, perfume, shampoo, yogurt
chips, coke, fork, glove, knife, milk, yogurt
apple, chips, coke, dish, milk, wine, yogurt
chips, coke, dish, fork, knife, milk, yogurt
apple, banana, chips, coke, orange, yogurt
apple, carrot, chips, coke, lettuce, milk, orange
chips, chocolate, egg, hat, milk, pillow, sheet, yogurt
bread, coke, egg, glove, milk, toilet paper, yogurt
apple, banana, chips, coke, orange, shampoo
chips, chocolate, coke, hat, pillow, sheet
chips, egg, fork, ketchup, knife, milk
apple, chips, chocolate, fork, ketchup, knife, milk, orange
banana, dish, egg, ketchup, milk, perfume, shampoo, wine, yogurt
knife, milk, mouth wash, shampoo, toilet paper, yogurt
banana, bread, chips, egg, rose, wine, yogurt
apple, glove, milk, orange, perfume, rose, wine
chocolate, egg, pillow, pork, sheet, yogurt
bread, carrot, coke, egg, face-mask, glove, hat, wine

2D frequent item-set:
chips, yogurt
coke, yogurt
milk, yogurt
chips, coke
chips, milk
coke, milk
3D frequent item-set:
chips, coke, yogurt
chips, milk, yogurt
coke, milk, yogurt
chips, coke, milk
can't find more frequent-sets!

Association rule: [chips]-->[yogurt]  [Support= 0.45, confidence= 0.6428571428571429]
Association rule: [yogurt]-->[chips]  [Support= 0.45, confidence= 0.6923076923076923]
Association rule: [coke]-->[yogurt]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [yogurt]-->[coke]  [Support= 0.4, confidence= 0.6153846153846154]
Association rule: [milk]-->[yogurt]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [yogurt]-->[milk]  [Support= 0.5, confidence= 0.7692307692307693]
Association rule: [chips]-->[coke]  [Support= 0.5, confidence= 0.7142857142857143]
```



```
Console
<terminated> Apriori [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 1:09:45 PM)
can't find more frequent-sets!

Association rule: [chips]-->[yogurt]  [Support= 0.45, confidence= 0.6428571428571429]
Association rule: [yogurt]-->[chips]  [Support= 0.45, confidence= 0.6923076923076923]
Association rule: [coke]-->[yogurt]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [yogurt]-->[coke]  [Support= 0.4, confidence= 0.6153846153846154]
Association rule: [milk]-->[yogurt]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [yogurt]-->[milk]  [Support= 0.5, confidence= 0.7692307692307693]
Association rule: [chips]-->[coke]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [coke]-->[chips]  [Support= 0.5, confidence= 0.8333333333333334]
Association rule: [chips]-->[milk]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [milk]-->[chips]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [coke]-->[milk]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [chips, coke]-->[yogurt]  [Support= 0.35, confidence= 0.7]
Association rule: [chips, yogurt]-->[coke]  [Support= 0.35, confidence= 0.7777777777777778]
Association rule: [coke, yogurt]-->[chips]  [Support= 0.35, confidence= 0.875]
Association rule: [chips, milk]-->[yogurt]  [Support= 0.35, confidence= 0.7]
Association rule: [chips, yogurt]-->[milk]  [Support= 0.35, confidence= 0.7777777777777778]
Association rule: [milk, yogurt]-->[chips]  [Support= 0.35, confidence= 0.7]
Association rule: [coke, milk]-->[yogurt]  [Support= 0.35, confidence= 0.875]
Association rule: [coke, yogurt]-->[milk]  [Support= 0.35, confidence= 0.875]
Association rule: [milk, yogurt]-->[coke]  [Support= 0.35, confidence= 0.7]
Association rule: [chips, coke]-->[milk]  [Support= 0.35, confidence= 0.7]
Association rule: [chips, milk]-->[coke]  [Support= 0.35, confidence= 0.7]
Association rule: [coke, milk]-->[chips]  [Support= 0.35, confidence= 0.875]
Executing time: 9ms
```

Data set 05:

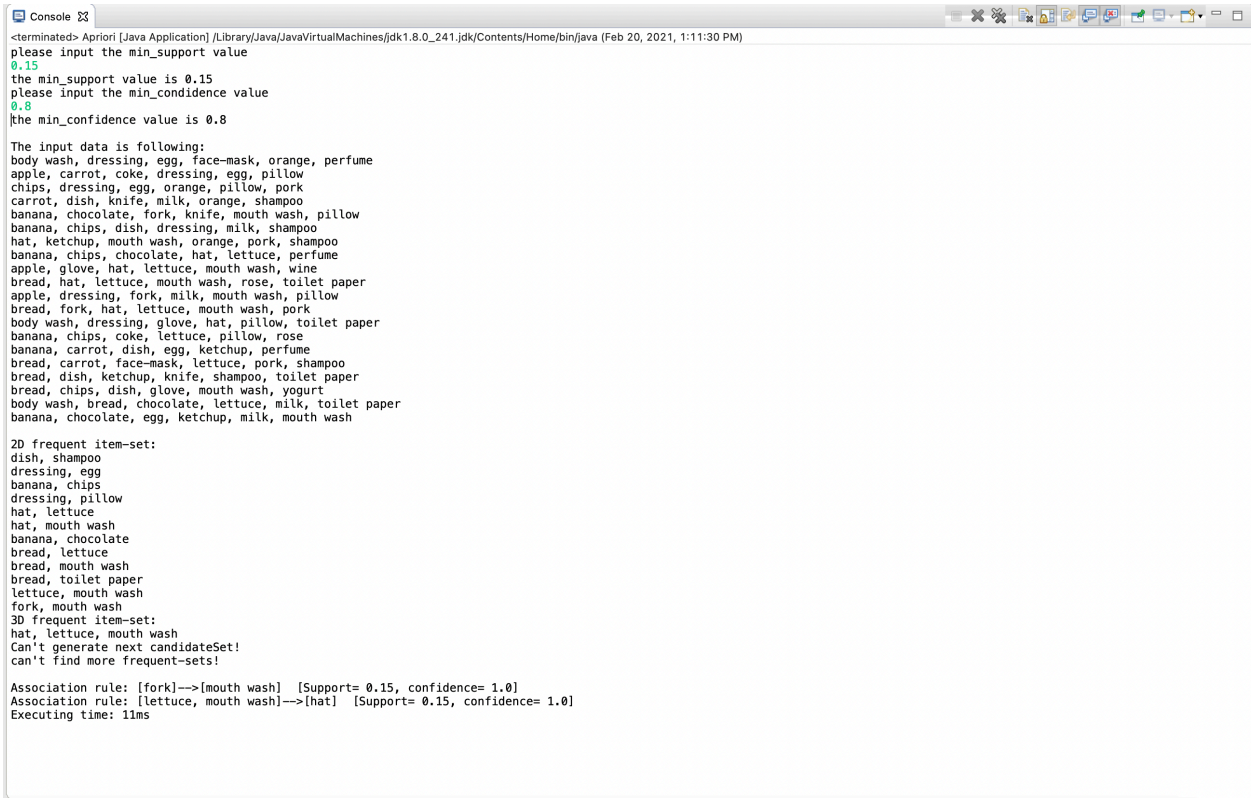
Scenario: a random data set (The reason why I generate a random data set although I know it's not proposed is, I want to find out what's the difference between a random data set and a artificial data set like each of the other 4 data sets)

Screenshot of DataSet05.txt:

The min_support value is 0.15

The min_confidence value is 0.8

Screenshots of output:



Analysis: It's very obvious that generating association rules from a random dataset is more difficult than generating association rules from an artificial dataset. In oder to generating association rules, we need to make either the value of support or the value of confidence lower. In this case, I lower the value of support which is 0.15.

Part2: brute force algorithm

Source code:

```
package cs634_midterm;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

public class Brute_force {
     private static double min_support;
     private static double min_confidence;
     static List<List<String>> record=new
ArrayList<List<String>>();
     static List<Set<String>> frequentIS=new
ArrayList<Set<String>>();
     static List<String> itemset=new ArrayList<String>();
     static boolean endTag=false;
     static int times=1;

     public static void main(String[] args) {
          System.out.println("please input the min_support
value");
          Scanner in=new Scanner(System.in);
          min_support=in.nextDouble();
          System.out.println("the min_support value is
"+min_support);
          System.out.println("please input the min_condidence
value");
          min_confidence=in.nextDouble();
          System.out.println("the min_confidence value is
"+min_confidence);
          System.out.println();
          in.close();
          long start=System.currentTimeMillis();

          record=getrecord("./DataSet01.txt");//scan data-set
```

```
            //showData(record); for test

            for(int i=0;i<record.size();i++) {
                 for(int j=0;j<record.get(i).size();j++) {
                      String temp=record.get(i).get(j);
                      if(!itemset.contains(temp)) {
                           itemset.add(temp);
                      }
                 }
            }
            Collections.sort(itemset);

            while(!endTag) {
                 List<Set<String>> temp=new ArrayList<>();
                 boolean haveFS=false;
                 temp=brute_force(itemset,times);
                 for(int i=0;i<temp.size();i++) {
                      int count=0;
                      for(int j=0;j<record.size();j++) {
                           List<String> temp_=new
ArrayList<String>();
                           temp_=record.get(j);
                           if(temp_.containsAll(temp.get(i))) {
                                count++;
                           }
                      /*if(count>=min_support*record.size()) {
                           frequentIS.add(temp.get(i));
                           haveFS=true;
                      }*/
                      }
                      if(count>=min_support*record.size()) {
                           frequentIS.add(temp.get(i));
                           haveFS=true;
                      }
                 }
                 if(!haveFS) {
                      endTag=true;
                 }
                 else times++;

            };
            System.out.println();


            AssociationRulesMining();//generate all the
association rules
```

```
          long end=System.currentTimeMillis();
          System.out.println("Executing time: "+(end-start)
+"ms");
     }

     private static List<List<String>> getrecord(String path){
          try {
               File dataset=new File(path);
               if(dataset.isFile()&&dataset.exists()) {
                    InputStreamReader read=new
InputStreamReader(new FileInputStream(dataset));
                    BufferedReader buffered_reader=new
BufferedReader(read);
                    String one_line;
                    while((one_line=buffered_reader.readLine())!
=null) {
                         String[] one_record=one_line.split(",
|:");
                         List<String> lineList=new
ArrayList<String>();
                         for(int i=1;i<one_record.length;i++) {
                              lineList.add(one_record[i]);
                         }
                         Collections.sort(lineList);
                         record.add(lineList);
                    }
                    read.close();
               }
               else if(dataset.isDirectory()&&dataset.exists())
{
                    System.out.println("Path is a directory!");
               }

               else System.out.println("Can't find the file");
          }
          catch(Exception e){
               e.printStackTrace();
          }
          System.out.println("The input data is following:");
          showData(record);
          System.out.println();
          return record;
     }

     private static List<Set<String>> brute_force(List<String>
itemset, int k){
```

```
          List<Set<String>> result=new ArrayList<>();
          getSubsets(itemset, k, 0, new HashSet<String>(),
result);
          return result;
     }

     private static void getSubsets(List<String> superSet, int
k, int idx, Set<String> current,List<Set<String>> solution) {
          //successful stop clause
          if (current.size() == k) {
               solution.add(new HashSet<>(current));
               return;
          }
          //unsuccessful stop clause
          if (idx == superSet.size()) return;
          String x = superSet.get(idx);
          current.add(x);
          //"guess" x is in the subset
          getSubsets(superSet, k, idx+1, current, solution);
          current.remove(x);
          //"guess" x is not in the subset
          getSubsets(superSet, k, idx+1, current, solution);
     }

     private static void AssociationRulesMining() {
          for(int i=0;i<frequentIS.size();i++) {
               List<String> temp=new ArrayList<String>();
               temp.addAll(frequentIS.get(i));
               if(temp.size()>1) {
                    List<String> tempClone=new
ArrayList<String>();
                    tempClone.addAll(temp);
                    List<List<String>>
allSubset=getSubsets(tempClone);

                    for(int j=0;j<allSubset.size();j++) {
                         List<String> s1=allSubset.get(j);
                         List<String> s2=getRest(temp,s1);
                         double
confidence=isAssociationRule(s1,s2,temp);
                         if(confidence>=min_confidence) {
                              double support=getCount(temp)*1.0/
record.size();
                              System.out.println("[Support=
"+support+", confidence= "+confidence+"]");
                         }
```

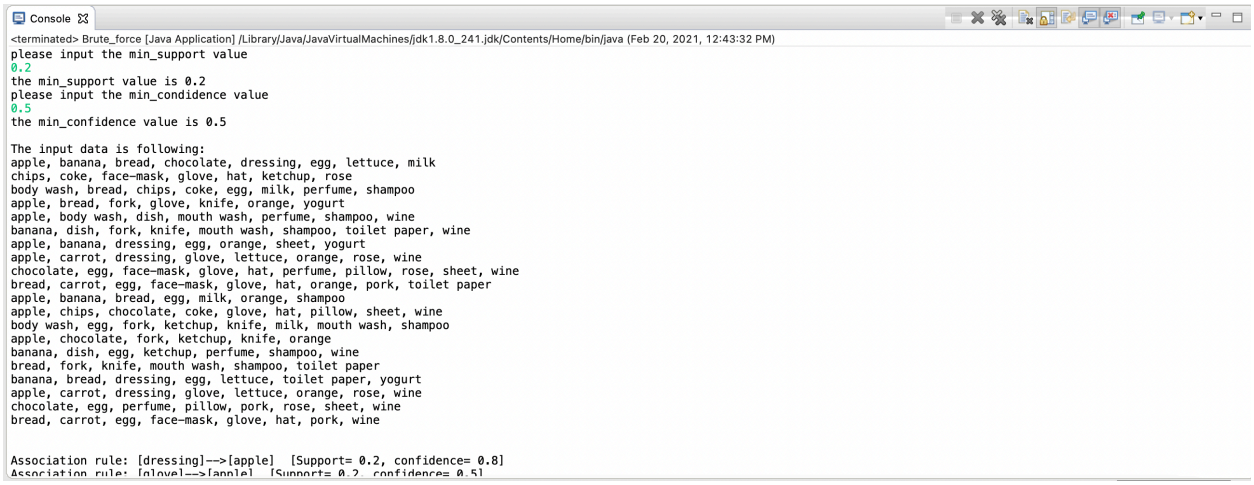
```
                    }
               }
          }

     }

     private static List<List<String>> getSubsets(List<String>
list) {
          List<List<String>> result=new
ArrayList<List<String>>();
          int length=list.size();
          int numberOfSubsets=length==0?0:1<<(length);

          for(int i=1;i<numberOfSubsets-1;i++) {//Starting from
1 because we don't need empty set
               List<String> subset=new ArrayList<String>();

               int index=i;
               for(int j=0;j<list.size();j++) {
                    if((index&1)==1) {
                         subset.add(list.get(j));
                    }
                    index=index>>1;
               }
               Collections.sort(subset);
               result.add(subset);

          }
          return result;
     }

     private static List<String> getRest(List<String> temp,
List<String> s1){
          List<String> result=new ArrayList<String>();

          for(int i=0;i<temp.size();i++) {
               String item=temp.get(i);
               if(!s1.contains(item))
                    result.add(item);
          }
          return result;
     }

     private static double isAssociationRule(List<String>
s1,List<String> s2,List<String> temp) {
          double confidence;
```

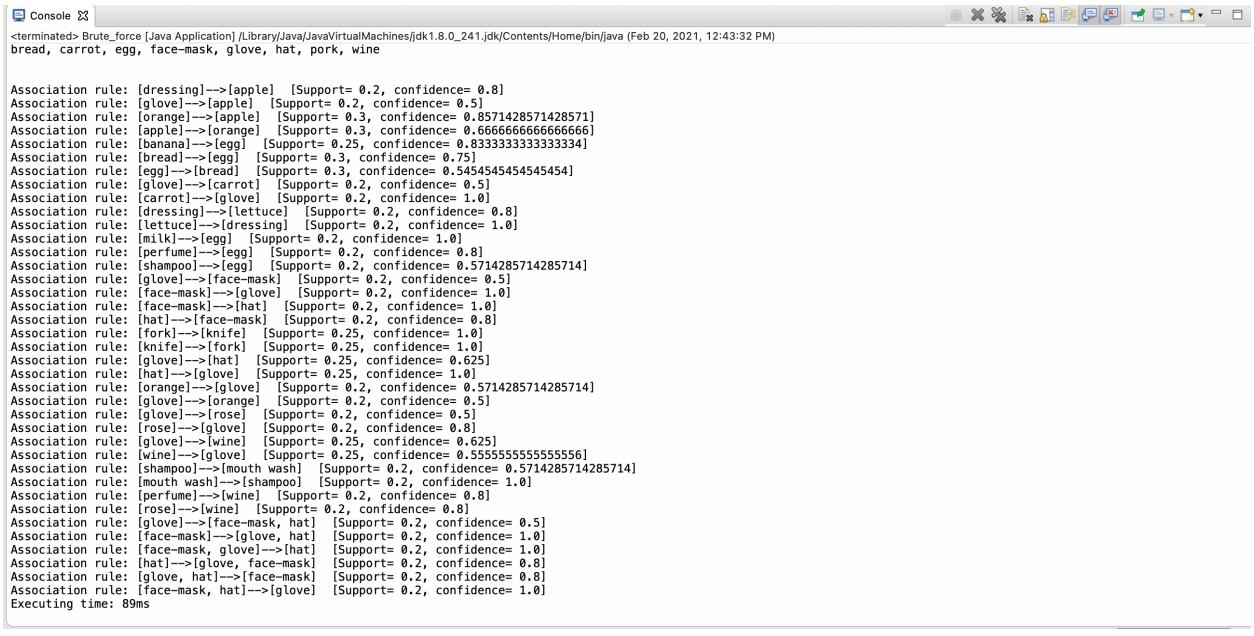
```
            int counts1;
            int counttemp;

            if(temp.size()!=0&&temp!=null) {
                counts1=getCount(s1);
                //System.out.println(counts1);
                counttemp=getCount(temp);
                //System.out.println(counttemp);
                confidence=counttemp*1.0/counts1;

                if(confidence>=min_confidence) {
                    System.out.print("Association rule:
"+s1.toString()+"-->"+s2.toString()+"  ");
                    return confidence;
                }
                else return 0;
            }
            else return 0;
        }

        private static int getCount(List<String> list) {
            int count=0;
            for(int i=0;i<record.size();i++) {
                boolean tag=true;
                for(int j=0;j<list.size();j++) {
                    String temp=list.get(j);
                    if(!record.get(i).contains(temp)) {
                        tag=false;
                        break;
                    }
                }
                if(tag==true) count++;
            }
            return count;
        }

        private static void showData(List<List<String>> list) {
            for(int i=0;i<list.size();i++) {
                int temp=list.get(i).size()-1;
                for(int j=0;j<temp;j++) {
                    System.out.print(list.get(i).get(j)+", ");
                }
                System.out.print(list.get(i).get(temp));
                System.out.println();
            }
        }
```

}

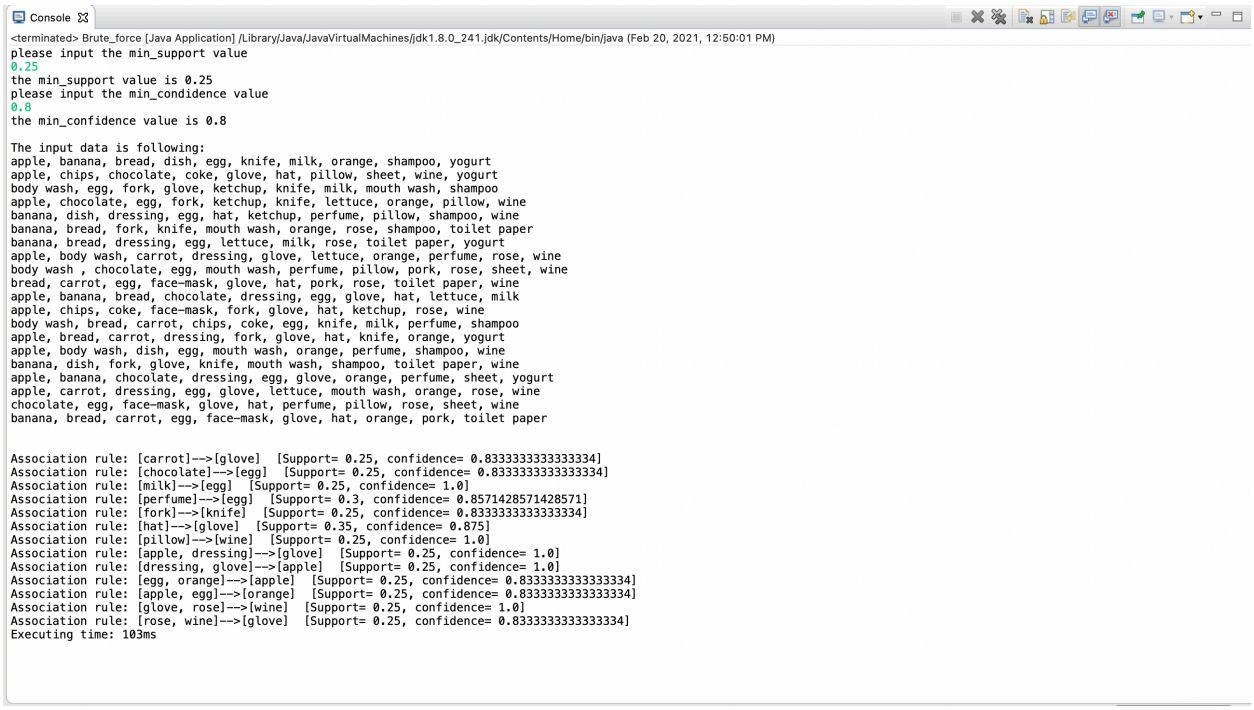
Screenshots:

Data set 01:

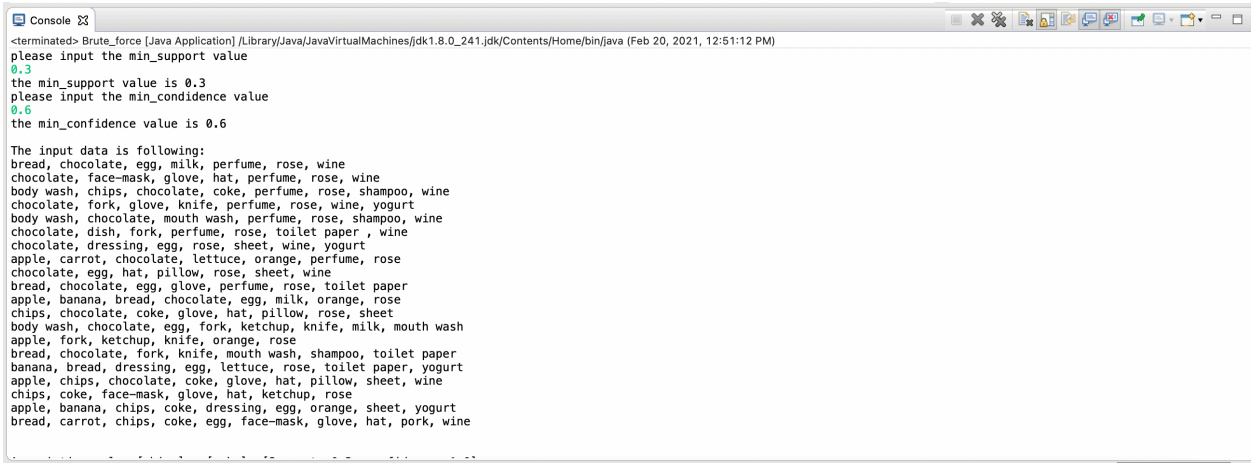




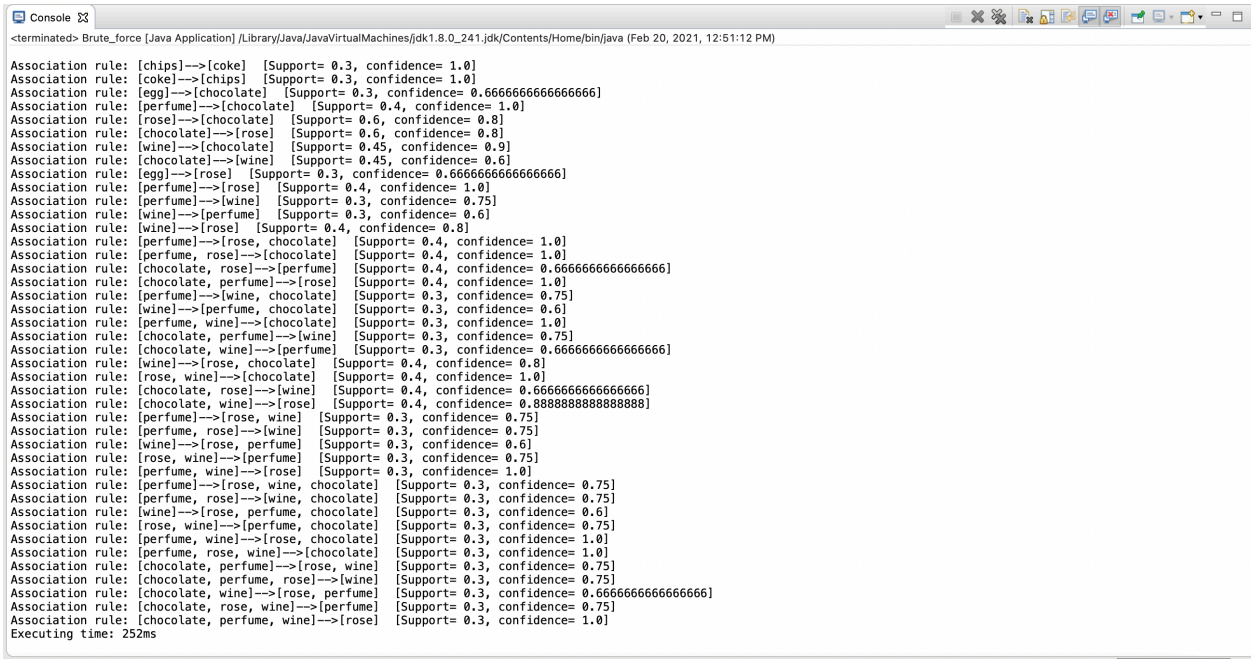
Data set 02:


Console
<terminated> Brute_force [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 12:50:01 PM)
please input the min_support value
0.25
the min_support value is 0.25
please input the min_condidence value
0.8
the min_confidence value is 0.8
The input data is following:
apple, banana, bread, dish, egg, knife, milk, orange, shampoo, yogurt
apple, chips, chocolate, coke, glove, hat, pillow, sheet, wine, yogurt
body wash, egg, fork, glove, ketchup, knife, milk, mouth wash, shampoo
apple, chocolate, egg, fork, ketchup, knife, lettuce, orange, pillow, wine
banana, dish, dressing, egg, hat, ketchup, perfume, pillow, shampoo, wine
banana, bread, fork, knife, mouth wash, orange, rose, shampoo, toilet paper
banana, bread, dressing, egg, lettuce, milk, rose, toilet paper, yogurt
apple, body wash, carrot, dressing, glove, lettuce, orange, perfume, rose, wine
body wash , chocolate, egg, mouth wash, perfume, pillow, pork, rose, sheet, wine
bread, carrot, egg, face-mask, glove, hat, pork, rose, toilet paper, wine
apple, banana, bread, chocolate, dressing, egg, glove, hat, lettuce, milk
apple, chips, coke, face-mask, fork, glove, hat, ketchup, rose, wine
body wash, bread, carrot, chips, coke, egg, knife, milk, perfume, shampoo
apple, bread, carrot, dressing, fork, glove, hat, knife, orange, yogurt
apple, body wash, dish, egg, mouth wash, orange, perfume, shampoo, wine
banana, dish, fork, glove, knife, mouth wash, shampoo, toilet paper, wine
apple, banana, chocolate, dressing, egg, glove, orange, perfume, sheet, yogurt
apple, carrot, dressing, egg, glove, lettuce, mouth wash, orange, rose, wine
chocolate, egg, face-mask, glove, hat, perfume, pillow, rose, sheet, wine
banana, bread, carrot, egg, face-mask, glove, hat, orange, pork, toilet paper
Association rule: [carrot]-->[glove] [Support= 0.25, confidence= 0.8333333333333334]
Association rule: [chocolate]-->[egg] [Support= 0.25, confidence= 0.8333333333333334]
Association rule: [milk]-->[egg] [Support= 0.25, confidence= 1.0]
Association rule: [perfume]-->[egg] [Support= 0.3, confidence= 0.8571428571428571]
Association rule: [fork]-->[knife] [Support= 0.25, confidence= 0.8333333333333334]
Association rule: [hat]-->[glove] [Support= 0.35, confidence= 0.875]
Association rule: [pillow]-->[wine] [Support= 0.25, confidence= 1.0]
Association rule: [apple, dressing]-->[glove] [Support= 0.25, confidence= 1.0]
Association rule: [dressing, glove]-->[apple] [Support= 0.25, confidence= 1.0]
Association rule: [egg, orange]-->[apple] [Support= 0.25, confidence= 0.8333333333333334]
Association rule: [apple, egg]-->[orange] [Support= 0.25, confidence= 0.8333333333333334]
Association rule: [glove, rose]-->[wine] [Support= 0.25, confidence= 1.0]
Association rule: [rose, wine]-->[glove] [Support= 0.25, confidence= 0.8333333333333334]
Executing time: 103ms

Data set 03:



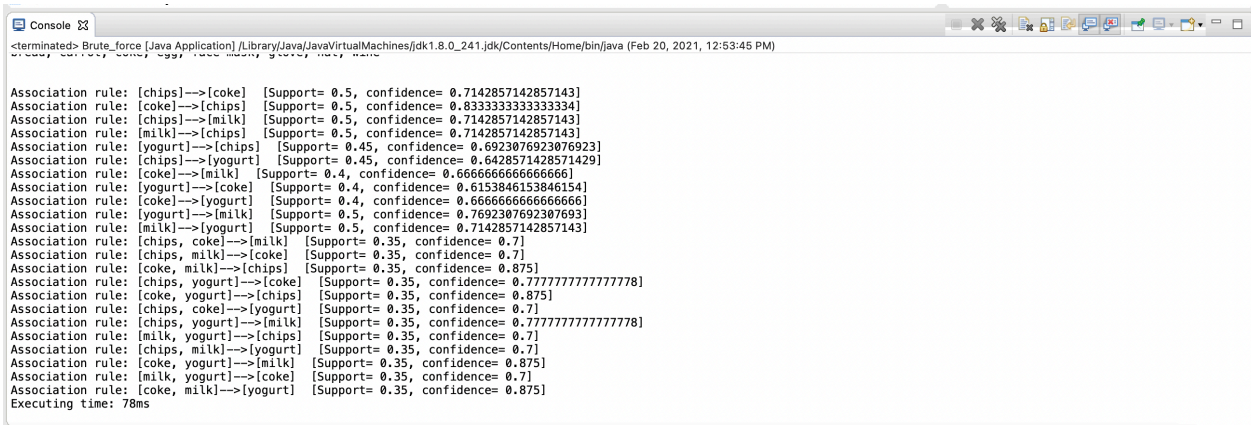
```
Console
<terminated> Brute_force [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 12:51:12 PM)
please input the min_support value
0.3
the min_support value is 0.3
please input the min_condidence value
0.6
the min_confidence value is 0.6

The input data is following:
bread, chocolate, egg, milk, perfume, rose, wine
chocolate, face-mask, glove, hat, perfume, rose, wine
body wash, chips, chocolate, coke, perfume, rose, shampoo, wine
chocolate, fork, glove, knife, perfume, rose, wine, yogurt
body wash, chocolate, mouth wash, perfume, rose, shampoo, wine
chocolate, dish, fork, perfume, rose, toilet paper , wine
chocolate, dressing, egg, rose, sheet, wine, yogurt
apple, carrot, chocolate, lettuce, orange, perfume, rose
chocolate, egg, hat, pillow, rose, sheet, wine
bread, chocolate, egg, glove, perfume, rose, toilet paper
apple, banana, bread, chocolate, egg, milk, orange, rose
chips, chocolate, coke, glove, hat, pillow, rose, sheet
body wash, chocolate, egg, fork, ketchup, knife, milk, mouth wash
apple, fork, ketchup, knife, orange, rose
bread, chocolate, fork, knife, mouth wash, shampoo, toilet paper
banana, bread, dressing, egg, lettuce, rose, toilet paper, yogurt
apple, chips, chocolate, coke, glove, hat, pillow, sheet, wine
chips, coke, face-mask, glove, hat, ketchup, rose
apple, banana, chips, coke, dressing, egg, orange, sheet, yogurt
bread, carrot, chips, coke, egg, face-mask, glove, hat, pork, wine
```



```
Console
<terminated> Brute_force [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 12:51:12 PM)

Association rule: [chips]-->[coke]  [Support= 0.3, confidence= 1.0]
Association rule: [coke]-->[chips]  [Support= 0.3, confidence= 1.0]
Association rule: [egg]-->[chocolate]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [perfume]-->[chocolate]  [Support= 0.4, confidence= 1.0]
Association rule: [rose]-->[chocolate]  [Support= 0.6, confidence= 0.8]
Association rule: [chocolate]-->[rose]  [Support= 0.6, confidence= 0.8]
Association rule: [wine]-->[chocolate]  [Support= 0.45, confidence= 0.9]
Association rule: [chocolate]-->[wine]  [Support= 0.45, confidence= 0.6]
Association rule: [egg]-->[rose]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [perfume]-->[rose]  [Support= 0.4, confidence= 1.0]
Association rule: [perfume]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[perfume]  [Support= 0.3, confidence= 0.6]
Association rule: [wine]-->[rose]  [Support= 0.4, confidence= 0.8]
Association rule: [perfume]-->[rose, chocolate]  [Support= 0.4, confidence= 1.0]
Association rule: [perfume, rose]-->[chocolate]  [Support= 0.4, confidence= 1.0]
Association rule: [chocolate, rose]-->[perfume]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [chocolate, perfume]-->[rose]  [Support= 0.4, confidence= 1.0]
Association rule: [perfume]-->[wine, chocolate]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[perfume, chocolate]  [Support= 0.3, confidence= 0.6]
Association rule: [perfume, wine]-->[chocolate]  [Support= 0.3, confidence= 1.0]
Association rule: [chocolate, perfume]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, wine]-->[perfume]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [wine]-->[rose, chocolate]  [Support= 0.4, confidence= 0.8]
Association rule: [rose, wine]-->[chocolate]  [Support= 0.4, confidence= 1.0]
Association rule: [chocolate, rose]-->[wine]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [chocolate, wine]-->[rose]  [Support= 0.4, confidence= 0.8888888888888888]
Association rule: [perfume]-->[rose, wine]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume, rose]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[rose, perfume]  [Support= 0.3, confidence= 0.6]
Association rule: [rose, wine]-->[perfume]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume, wine]-->[rose]  [Support= 0.3, confidence= 1.0]
Association rule: [perfume]-->[rose, wine, chocolate]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume, rose]-->[wine, chocolate]  [Support= 0.3, confidence= 0.75]
Association rule: [wine]-->[rose, perfume, chocolate]  [Support= 0.3, confidence= 0.6]
Association rule: [rose, wine]-->[perfume, chocolate]  [Support= 0.3, confidence= 0.75]
Association rule: [perfume, wine]-->[rose, chocolate]  [Support= 0.3, confidence= 1.0]
Association rule: [perfume, rose, wine]-->[chocolate]  [Support= 0.3, confidence= 1.0]
Association rule: [chocolate, perfume]-->[rose, wine]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, perfume, rose]-->[wine]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, wine]-->[rose, perfume]  [Support= 0.3, confidence= 0.6666666666666666]
Association rule: [chocolate, rose, wine]-->[perfume]  [Support= 0.3, confidence= 0.75]
Association rule: [chocolate, perfume, wine]-->[rose]  [Support= 0.3, confidence= 1.0]
Executing time: 252ms
```

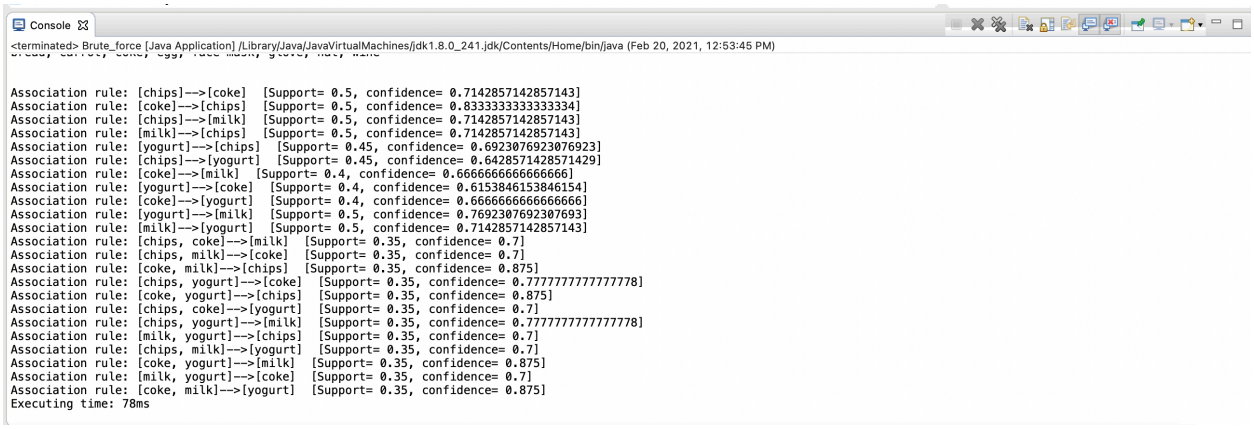
Data set 04:

```
Console
<terminated> Brute_force [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 12:53:45 PM)
please input the min_support value
0.35
the min_support value is 0.35
please input the min_condidence value
0.6
the min_confidence value is 0.6

The input data is following:
banana, chips, coke, dressing, lettuce, milk, yogurt
chips, coke, face-mask, glove, hat, milk, yogurt
body wash, chips, coke, milk, perfume, shampoo, yogurt
chips, coke, fork, glove, knife, milk, yogurt
apple, chips, coke, dish, milk, wine, yogurt
chips, coke, dish, fork, knife, milk, yogurt
apple, banana, chips, coke, orange, yogurt
apple, carrot, chips, coke, lettuce, milk, orange
chips, chocolate, egg, hat, milk, pillow, sheet, yogurt
bread, coke, egg, glove, milk, toilet paper, yogurt
apple, banana, chips, coke, orange, shampoo
chips, chocolate, coke, hat, pillow, sheet
chips, egg, fork, ketchup, knife, milk
apple, chips, chocolate, fork, ketchup, knife, milk, orange
banana, dish, egg, ketchup, milk, perfume, shampoo, wine, yogurt
knife, milk, mouth wash, shampoo, toilet paper, yogurt
banana, bread, chips, egg, rose, wine, yogurt
apple, glove, milk, orange, perfume, rose, wine
chocolate, egg, pillow, pork, sheet, yogurt
bread, carrot, coke, egg, face-mask, glove, hat, wine


Association rule: [chips]-->[coke]  [Support= 0.5, confidence= 0.7142857142857143]
```



```
Console
<terminated> Brute_force [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Feb 20, 2021, 12:53:45 PM)


Association rule: [chips]-->[coke]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [coke]-->[chips]  [Support= 0.5, confidence= 0.8333333333333334]
Association rule: [chips]-->[milk]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [milk]-->[chips]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [yogurt]-->[chips]  [Support= 0.45, confidence= 0.6923076923076923]
Association rule: [chips]-->[yogurt]  [Support= 0.45, confidence= 0.6428571428571429]
Association rule: [coke]-->[milk]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [yogurt]-->[coke]  [Support= 0.4, confidence= 0.6153846153846154]
Association rule: [coke]-->[yogurt]  [Support= 0.4, confidence= 0.6666666666666666]
Association rule: [yogurt]-->[milk]  [Support= 0.5, confidence= 0.7692307692307693]
Association rule: [milk]-->[yogurt]  [Support= 0.5, confidence= 0.7142857142857143]
Association rule: [chips, coke]-->[milk]  [Support= 0.35, confidence= 0.7]
Association rule: [chips, milk]-->[coke]  [Support= 0.35, confidence= 0.7]
Association rule: [coke, milk]-->[chips]  [Support= 0.35, confidence= 0.875]
Association rule: [chips, yogurt]-->[coke]  [Support= 0.35, confidence= 0.7777777777777778]
Association rule: [coke, yogurt]-->[chips]  [Support= 0.35, confidence= 0.875]
Association rule: [chips, coke]-->[yogurt]  [Support= 0.35, confidence= 0.7]
Association rule: [chips, yogurt]-->[milk]  [Support= 0.35, confidence= 0.7777777777777778]
Association rule: [milk, yogurt]-->[chips]  [Support= 0.35, confidence= 0.7]
Association rule: [chips, milk]-->[yogurt]  [Support= 0.35, confidence= 0.7]
Association rule: [coke, yogurt]-->[milk]  [Support= 0.35, confidence= 0.875]
Association rule: [milk, yogurt]-->[coke]  [Support= 0.35, confidence= 0.7]
Association rule: [coke, milk]-->[yogurt]  [Support= 0.35, confidence= 0.875]
Executing time: 78ms
```

Data set 05:



Comparison of executing time:

Data set 01:

T(Apriori)= 15ms, T(Brute-Force)=89ms

Data set 02:

T(Apriori)= 14ms, T(Brute-Force)=103ms

Data set 03:

T(Apriori)= 12ms, T(Brute-Force)=252ms

Data set 04:

T(Apriori)= 9ms, T(Brute-Force)=78ms

Data set 05:

T(Apriori)= 11ms, T(Brute-Force)=78ms

It’s obvious Apriori algorithm is faster than Brute-Force algorithm on each of the five dataset.