DATA MINING

CS634

MID TERM PROJECT

By

Kumaran Manangatti Dharman Lekha

km439

Contents

[APRIORI ALGORITHM 3](#_Toc445366261)

[APPLICATIONS OF THIS ALGORITHM: 4](#_Toc445366262)

[TERMS RELATED TO THIS ALGORITHM: 4](#_Toc445366263)

[STEPS INVOLVED IN APRIORI ALGORITHM PSEUDO-CODE: 4](#_Toc445366264)

[PSEUDO-CODE: 4](#_Toc445366265)

[SUPPORT: 5](#_Toc445366266)

[CONFIDENCE: 5](#_Toc445366267)

[AIM OF THE PROJECT: 6](#_Toc445366268)

[DATABASE 1 7](#_Toc445366269)

[DATABASE 2 8](#_Toc445366270)

[DATABASE 3 9](#_Toc445366271)

[DATABASE 4 10](#_Toc445366272)

[DATABASE 5 11](#_Toc445366273)

[EXECUTION STEPS: 12](#_Toc445366274)

[SOURCE CODE: 12](#_Toc445366275)

[SCREENSHOTS (Output copied from console into Notepad+): 25](#_Toc445366276)

[REFERENCES: 30](#_Toc445366277)

# APRIORI ALGORITHM

**Apriori** is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database

Association rule mining is a procedure which is meant to find frequent patterns, correlations, associations, or causal structures from data sets found in various kinds of databases such as relational databases, transactional databases, and other forms of data repositories.

Given a set of transactions, association rule mining aims to find the rules which enable us to predict the occurrence of a specific item based on the occurrences of the other items in the transaction.

Association rule mining is the data mining process of finding the rules that may govern associations and causal objects between sets of items.   
  
So in a given transaction with multiple items, it tries to find the rules that govern how or why such items are often bought together. For example, peanut butter and jelly are often bought together because a lot of people like to make Peanut Butter and Jelly sandwiches.

APPLICATIONS OF THIS ALGORITHM: **Basket Data Analysis** - is to analyze the association of purchased items in a single basket or single purchase as per the examples given above.

**Cross Marketing** - is to work with other businesses that complement your own, not competitors. For example, vehicle dealerships and manufacturers have cross marketing campaigns with oil and gas companies for obvious reasons.

**Catalog Design** - the selection of items in a business’ catalog are often designed to complement each other so that buying one item will lead to buying of another. So these items are often complements or very related.

# TERMS RELATED TO THIS ALGORITHM:

* **Frequent Itemsets:** The sets of item which has minimum support (denoted by Li for ith-Itemset).
* **Apriori Property:** Any subset of frequent itemset must be frequent.
* **Join Operation:** To find Lk, a set of candidate k-itemsets is generated by joining Lk-1 with itself.

# STEPS INVOLVED IN APRIORI ALGORITHM PSEUDO-CODE:

**JOIN STEP:** *Ck*is generated by joining Lk-1 with itself

**PRUNE STEP:** Any *(k-1)-*item set that is non frequent cannot be a subset of a frequent K-item set.

# PSEUDO-CODE:

*Ck* : Candidate item set of size k.

*Lk* : Frequent item set of size k

*L1*= (frequent items)

*For (k=1; Lk != 0;k++­­)do begin*

*Ck+1= candidates generated from Lk;*

*For each transaction t in database do increment the count of all candidates in Ck+1 that are contained in t.*

*Lk+1= candidates in Ck+1 with minimum support*

*End*

*Return kLk;*

# SUPPORT:

How to calculate support (support formula):

To calculate the support of A, calculate the number of transactions buying A divided by total number of transactions.

To calculate the support of XZ, calculate the support of [X, Z] i.e. the number of transactions buying X, Z divided by the total number of transactions.

***“Support of (X, Z) = no. of transactions buying X, Z/ total no. of transactions”***

***“Support of A = no. of transactions buying A/ total no. of transactions”***

# CONFIDENCE:

How to calculate confidence (confidence formula):

To calculate the confidence of X🡪Z, calculate the support of item set [X, Z] and calculate the support of item set.

***“Confidence of X🡪Z = sup(X, Z)/sup(X)”***

# AIM OF THE PROJECT:

Using Apriori Algorithm, we need to print out all the transactions i.e. frequent item sets and association rules satisfying both minimum support and confidence defined by the user.

The aim of the project is to create five unique tables of 20 transactions each containing 10 items and print the frequent item sets and association rules.

**Created Tables:**

Created 5 different transaction database of 10 unique items that are bought frequently in Amazon website. The brief description of each database will be displayed in the later chapters.

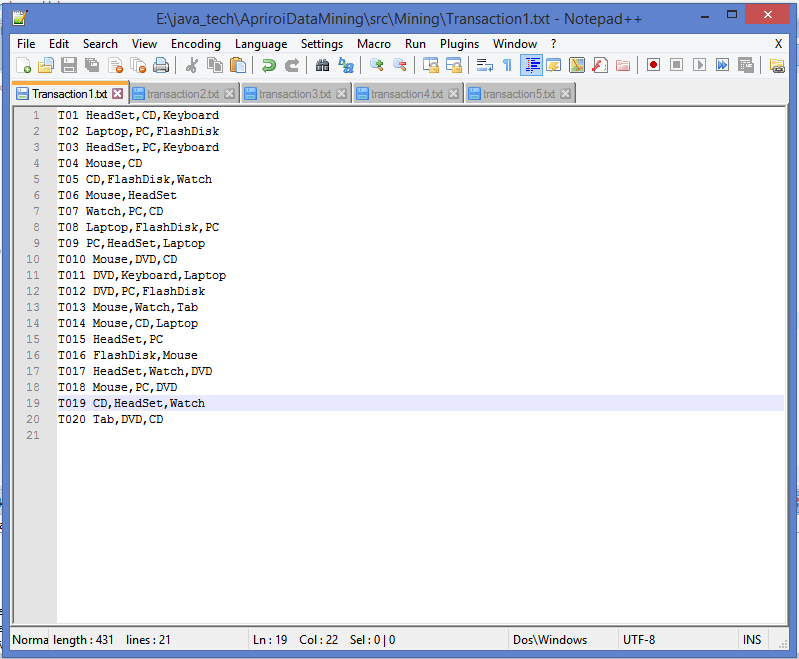
**Tools used:**

1. Operating system: Windows 8
2. Programming language: Java
3. Java Development Tool: Eclipse Java EE IDE , Mars Release version (4.5.1)
4. Database : Flat file (a text document where each line defines a new transaction)

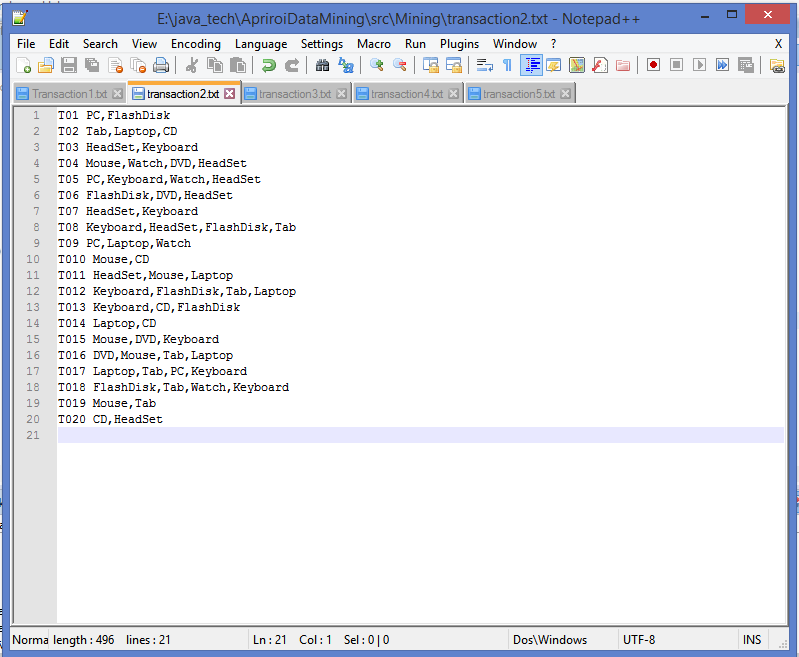
**10 unique items:**

1. Tab
2. CD
3. DVD
4. Laptop
5. PC
6. Mouse
7. Keyboard
8. Watch
9. FlashDisk
10. HeadSet

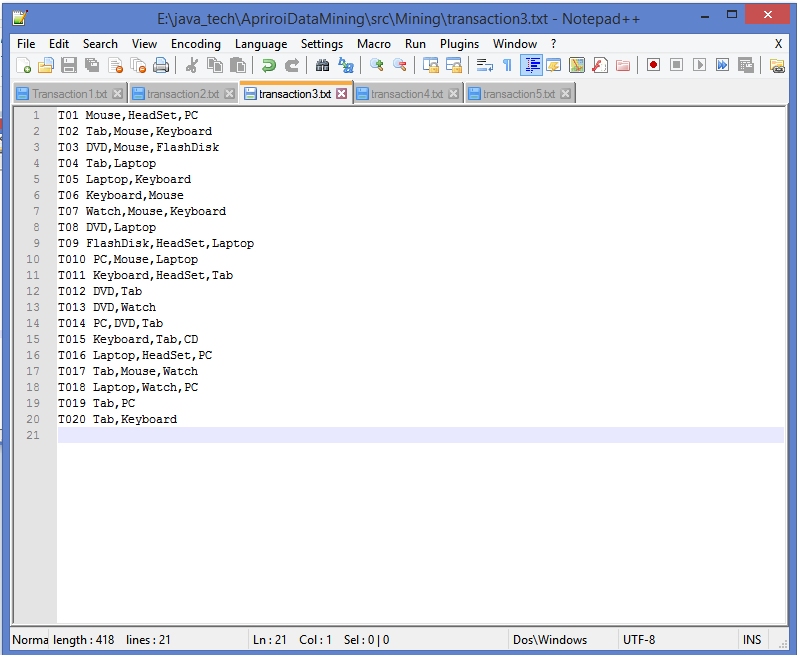
DATABASE 1: Transaction1 from Amazon stores database.



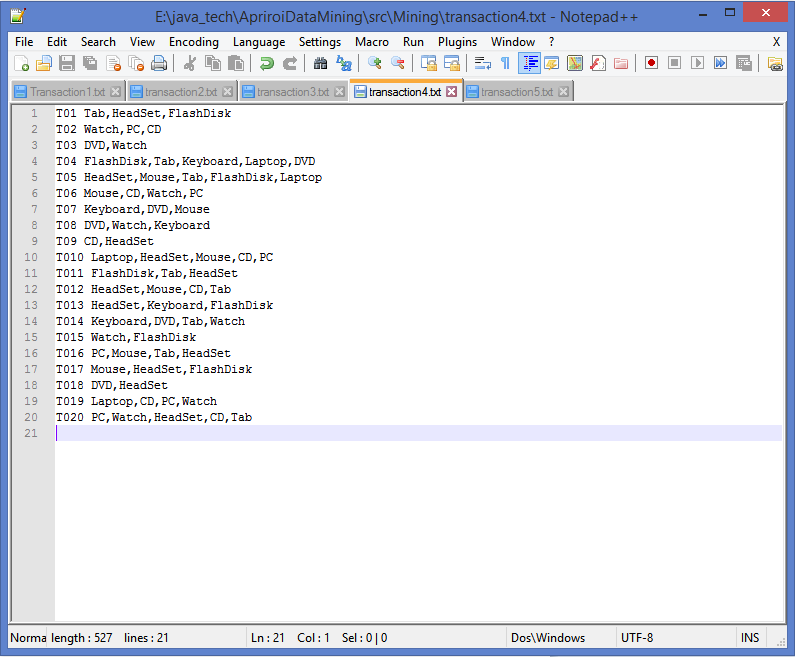
DATABASE 2**:** Transaction2 from Amazon Stores Database.



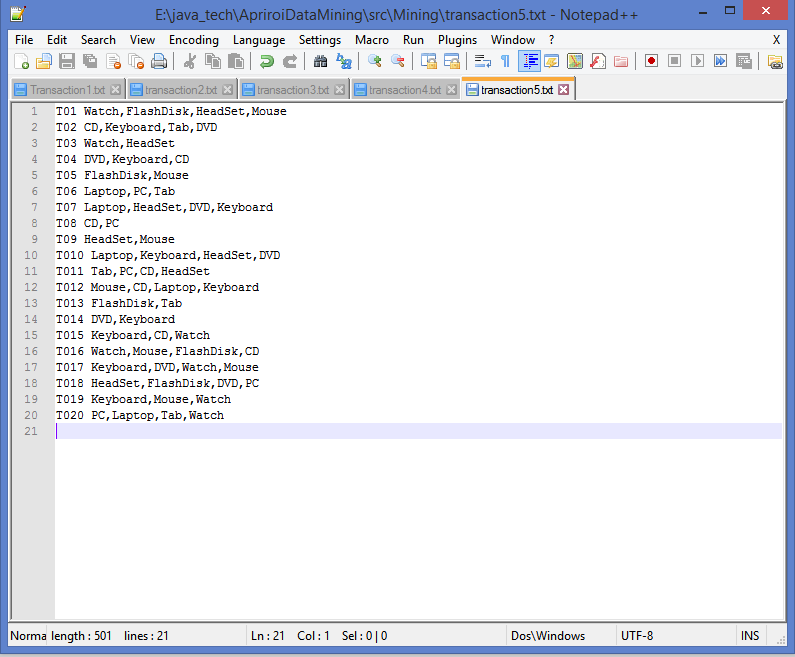
DATABASE 3: Transaction3 in Amazon Stores Database.



DATABASE 4: Transaction4 in Amazon Stores Database.



DATABASE 5: Transaction5 in Amazon Stores Database.



# EXECUTION STEPS:

1. We will have to run the main program in Eclipse IDE.

2. The user generates a random transaction Database of the 10 unique items.

3. The program is executed.

4. Enter the minimum support and confidence in the console

5. The Apriori Algorithm runs with the selected data set displaying list of data, frequent item sets, association rule mining and the total no. of confidence and support items.

# SOURCE CODE:

TransactionGeneration.java

|  |
| --- |
| package Mining**;**  **import** java**.**io**.**BufferedReader**;**  **import** java**.**io**.**BufferedWriter**;**  **import** java**.**io**.**File**;**  **import** java**.**io**.**FileReader**;**  **import** java**.**io**.**FileWriter**;**  **import** java**.**io**.**IOException**;**  **import** java**.**util**.**Arrays**;**  **import** java**.**util**.**Collections**;**  **import** java**.**util**.**HashMap**;**  **import** java**.**util**.**List**;**  **import** java**.**util**.**Random**;**  **import** java**.**util**.**Map**.**Entry**;**  /\* TransactionGeneration() generates transaction database and stores in a file\*/  public class TransactionGeneration **{**  String**[]** items **=** **{**"Tab"**,**"CD"**,**"DVD"**,**"Laptop"**,** "PC"**,** "Mouse"**,** "Keyboard"**,** "Watch"**,** "FlashDisk"**,** "HeadSet"**};**    public HashMap**<**String**,** String**>** nameItemSet**()**  **{**  String name**;**  HashMap**<**String**,** String**>** nameHashSet **=** **new** HashMap**<**String**,** String**>();**  **for(**int i**=**0**;** i**<**items**.**length**;**i**++)**  **{**  name **=** items**[**i**].**substring**(**0**,** 1**).**toString**();**  nameHashSet**.**put**(**name**,** items**[**i**]);**  **}**  **for** **(**Entry**<**String**,** String**>** entry **:** nameHashSet**.**entrySet**())**  **{**  String key **=** entry**.**getKey**();**  String value **=** entry**.**getValue**();**  **}**  **return** nameHashSet**;**  **}**    public void transGenerate**()**  **{**  StringBuffer stringBuffer **=** **new** StringBuffer**();**  List**<**String**>** names **=** Arrays**.**asList**(**items**);**  Random ran**=** **new** Random**();**  int transIDGenInt **=** 0**;**  int**[]** a **=** ran**.**ints**(**20**,** 2**,** 6**).**toArray**();**  String transaction = null;  try  {  File file = new File("E:\\java\_tech\\ApriroiDataMining\\Transaction\_Data.txt");  if (!file.exists()) {  file.createNewFile();  }  FileWriter fw = new FileWriter(file.getAbsoluteFile());  BufferedWriter bw = new BufferedWriter(fw);  for(int i=0; i < 20;i++)  {  transIDGenInt++;  Collections.shuffle(names);  transaction = "T0" + transIDGenInt+ " " +names.subList(0, a[i]).toString().replace("[", "").replace("]", "").replace(" ", "") + "\n";  stringBuffer.append(transaction);  bw.write(transaction);  }  bw.close();  System.out.println(stringBuffer);  }  catch(Exception e)  {  e.printStackTrace();  }  }  public HashMap<Integer, String> TransactionRead()  {  File file = new File("E:\\java\_tech\\ApriroiDataMining\\Transaction\_Data.txt");  String[] ProcessString, ProcessString1;  String line, collectiveData = "";  int i =1, j = 0;  try {  FileReader fileReader = new FileReader(file.getAbsoluteFile());  BufferedReader bufferedReader = new BufferedReader(fileReader);  StringBuffer stringBuffer = new StringBuffer();  HashMap<Integer, String> mainHashSet = new HashMap<Integer, String>();  HashMap<Integer, String> ProcessedDataHashSet = new HashMap<Integer, String>();  while ((line = bufferedReader.readLine()) != null)  {  ProcessString = line.split(" ");  mainHashSet.put(i, ProcessString[1]);  ProcessString1 = ProcessString[1].split(",");  for(j=0; j < ProcessString1.length; j++)  {  collectiveData = collectiveData+ProcessString1[j].substring(0,1);  }  ProcessedDataHashSet.put(i, collectiveData);  collectiveData ="";  stringBuffer.append(line);  stringBuffer.append("\n");  ++i;  }  fileReader.close();  return ProcessedDataHashSet;  }  catch (IOException e) {  e.printStackTrace();  }  return null;  }  /\*Returns Hashmap containing the transaction data  \*  \*/  public HashMap<Integer, String> generateHashMainItemSet()  {  HashMap<Integer, String> RequiredHashDataSet = TransactionRead();  return RequiredHashDataSet;  }  } |

AprioriAlgorithmImplementation.java

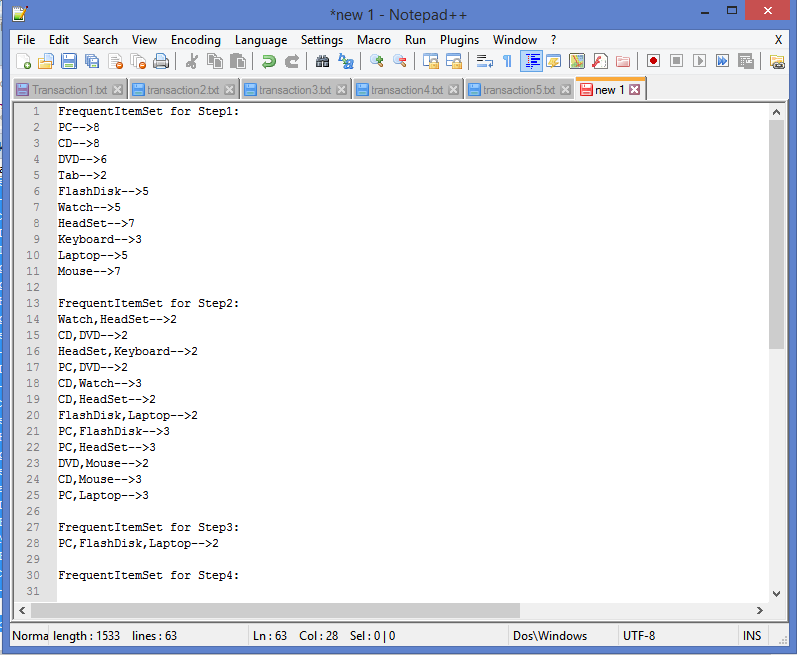
|  |
| --- |
| package Mining**;**  **import** java**.**util**.**ArrayList**;**  **import** java**.**util**.**Collections**;**  **import** java**.**util**.**HashMap**;**  **import** java**.**util**.**Map**.**Entry**;**  **import** java**.**util**.**Vector**;**  public class AprioriAlgorithmImplementation **{**  double support**;**  double confidence**;**  double totalTransaction**;**  HashMap**<**String**,** Integer**>** totaltransactionSet**;**  HashMap**<**String**,** Double**>** assocRuleHashTable**;**  public AprioriAlgorithmImplementation**()** **{**  **this.**support **=** DataAnalysis**.**support**;**  **this.**confidence **=** DataAnalysis**.**confidence**;**  **this.**totalTransaction **=** DataAnalysis**.**totalTransaction**;**  assocRuleHashTable **=** **new** HashMap**<>();**  **}**  /\*  Returns Hash map containing SingleItemSet  \*/  public HashMap**<**String**,** Integer**>** singleItemSet**()**  **{**  int i**,**count\_ProductName**;**  String collectiveKey**=**""**;**  TransactionGeneration T1 **=** **new** TransactionGeneration**();**  HashMap**<**String**,** String**>** nameHashSet **=** T1**.**nameItemSet**();**  HashMap**<**Integer**,** String**>** RequiredHashDataSet **=** T1**.**generateHashMainItemSet**();**  HashMap**<**String**,** Integer**>** singleItemDataSet **=** **new** HashMap**<**String**,** Integer**>();**  System**.**out**.**println**(**"FrequentItemSet for Step1:"**);**  **for** **(**Entry**<**Integer**,** String**>** entry **:** RequiredHashDataSet**.**entrySet**())**  **{**  String transRow **=** entry**.**getValue**();**  **for(**i **=**0**;** i **<** transRow**.**length**();** i**++)**  **{**  String productName **=** transRow**.**charAt**(**i**)+**""**;**  **if(**singleItemDataSet**.**containsKey**(**productName**))**  **{**  count\_ProductName **=** singleItemDataSet**.**get**(**productName**);**  singleItemDataSet**.**put**(**productName**,** **++**count\_ProductName**);**  **}**  **else**  **{**  singleItemDataSet.put(productName, 1);  }  }  }  return singleItemDataSet;  }  /\*  Returns Hash map containing SingleItemSet and its count  \*/  public HashMap<String, Integer> refinedSingleItemSet()  {  TransactionGeneration T1 = new TransactionGeneration();  HashMap<String, String> nameHashSet = T1.nameItemSet();  double minimumSupport;  String collectiveKey="";  HashMap<String, Integer> singleItemDataSet = singleItemSet();  HashMap<String, Integer> refinedsingleItemDataSet = new HashMap<String, Integer>();  for (Entry<String, Integer> entry : singleItemDataSet.entrySet())  {  String key = entry.getKey();  Integer value = entry.getValue();  minimumSupport = value / totalTransaction;  if(minimumSupport >= support)  {  refinedsingleItemDataSet.put(key, value);  }  }  for (Entry<String, Integer> entry1 : refinedsingleItemDataSet.entrySet())  {  String key = entry1.getKey();  Integer value = entry1.getValue();  for (Entry<String, String> entry2 : nameHashSet.entrySet())  {  String key1 = entry2.getKey();  String value1 = entry2.getValue();  if(key1.contains(key))  {  collectiveKey = collectiveKey + value1;  System.out.println(collectiveKey+"-->"+value);  }  }  collectiveKey = "";  }  totaltransactionSet = refinedsingleItemDataSet;  return refinedsingleItemDataSet;  }  /\*  Returns Hash map containing SubsequentKtemSet  \*/  public HashMap<String, Integer> kItemDataSet()  {  ArrayList<String> strConcatList = new ArrayList<String>();  TransactionGeneration T1 = new TransactionGeneration();;  HashMap<String, String> nameHashSet = T1.nameItemSet();  HashMap<Integer, String> RequiredHashDataSet = T1.generateHashMainItemSet();  int count\_check =0,count\_Item=0;  String strConcat;  String collectiveKey1="";  char[] str1arr;  double minimumSupport;  HashMap<String, Integer> refinedsingleItemDataSet = refinedSingleItemSet();  ArrayList<String> kItemDataList= new ArrayList<String>();  HashMap<String, Integer> kItemDataSet = new HashMap<String, Integer>();  HashMap<String, Integer> refinedkItemDataSet = new HashMap<String, Integer>();  System.out.println();  for (Entry<String, Integer> entry : refinedsingleItemDataSet.entrySet())  {  String key = entry.getKey();  strConcatList.add("" +key);  }  for(int i=0;i<strConcatList.size()-1;i++)  {  for(int j=i+1;j<strConcatList.size();j++)  {  strConcat = strConcatList.get(i) + strConcatList.get(j);  kItemDataList.add(strConcat);  }  }  Vector<Character> v = new Vector<Character>();  for (String item : kItemDataList)  {  for(int k = 0; k<item.length(); k++)  {  v.addElement(item.charAt(k));  }  for (Entry<Integer, String> entryDataSet : RequiredHashDataSet.entrySet())  {  String valueItem = entryDataSet.getValue();  for(int l=0;l<v.size(); l++)  {  if(!valueItem.contains(v.elementAt(l)+""))  {  break;  }  count\_check++;  if(count\_check == v.size())  {  if(kItemDataSet.containsKey(item))  {  count\_Item = kItemDataSet.get(item);  kItemDataSet.put(item, ++count\_Item);  }  else  {  kItemDataSet.put(item, 1);  }  }  }  count\_check = 0;  }  v.clear();  }  for (Entry<String, Integer> entry : kItemDataSet.entrySet())  {  String key = entry.getKey();  Integer value = entry.getValue();  minimumSupport = value / totalTransaction;  if(minimumSupport >= support)  {  refinedkItemDataSet.put(key, value);  }  }  System.out.println("FrequentItemSet for Step2:");  for (Entry<String, Integer> entry : refinedkItemDataSet.entrySet())  {  String key = entry.getKey();  str1arr = key.toCharArray();  Integer value = entry.getValue();  for(int h=0; h<str1arr.length; h++)  {  for(Entry<String, String> entry1 : nameHashSet.entrySet())  {  String key1 = entry1.getKey();  String value1 = entry1.getValue();  if(key1.contains(str1arr[h]+""))  {  collectiveKey1 = collectiveKey1 + value1;  if(h<str1arr.length-1)  {  collectiveKey1 = collectiveKey1 + ",";  }  }  }  }  System.out.println(collectiveKey1+"-->"+value);  collectiveKey1 = "";  }  totaltransactionSet.putAll(refinedkItemDataSet);  return refinedkItemDataSet;  }  /\*  Returns Hash map containing SubsequentKtemSet and its count  \*/  public HashMap<String, Integer> refinedIterateKitemSet(HashMap<String, Integer> kItemDataSetNeeded)  {  TransactionGeneration T1 = new TransactionGeneration();  HashMap<Integer, String> RequiredHashDataSet = T1.generateHashMainItemSet();  ArrayList<String> subStrList = new ArrayList<String>();  ArrayList<String> firstSubStrList = new ArrayList<String>();  ArrayList<String> kItemDataListNext = new ArrayList<String>();  String subStr,subStrComp, StrComp, subStrConcat, firstSubStr;  HashMap<String, String> nameHashSet = T1.nameItemSet();  int count\_Item, count\_check =0;  char[] str1arr;  String collectiveKey1 = "";  double minimumSupport;  HashMap<String, Integer> kItemDataSetIter = new HashMap<String, Integer>();  HashMap<String, Integer> refinedkItemDataSetIter = new HashMap<String, Integer>();  ArrayList<String> kItemDataArrayList = new ArrayList<String>();  for (Entry<String, Integer> entrySet : kItemDataSetNeeded.entrySet())  {  String key = entrySet.getKey();  firstSubStrList.add(key);  subStr = key.substring(0, key.length()-1);  subStrList.add(subStr);  kItemDataArrayList.add(key);  }  for(int i=0; i<subStrList.size(); i++)  {  subStr = subStrList.get(i);  firstSubStr = firstSubStrList.get(i);  for(int j =i+1; j<kItemDataArrayList.size(); j++ )  {  StrComp = kItemDataArrayList.get(j);  subStrComp = StrComp.substring(0, StrComp.length()-1);  if(subStr.contains(subStrComp))  {  subStrConcat = firstSubStr + StrComp.substring(subStr.length(), StrComp.length());  if(!kItemDataListNext.contains(subStrConcat))  {  kItemDataListNext.add(subStrConcat);  }  }  }  }  if(kItemDataListNext.size() ==0)  {  Collections.<String,String>emptyMap();  }  Vector<Character> v = new Vector<Character>();  for (String item : kItemDataListNext)  {  for(int k = 0; k<item.length(); k++)  {  v.addElement(item.charAt(k));  }  for (Entry<Integer, String> entryDataSet : RequiredHashDataSet.entrySet())  {  String valueItem = entryDataSet.getValue();  for(int l=0;l<v.size(); l++)  {  if(!valueItem.contains(v.elementAt(l)+""))  {  break;  }  count\_check++;  if(count\_check == v.size())  {  if(kItemDataSetIter.containsKey(item))  {  count\_Item = kItemDataSetIter.get(item);  kItemDataSetIter.put(item, ++count\_Item);  }  else  {  kItemDataSetIter.put(item, 1);  }  }  }  count\_check = 0;  }  v.clear();  }  for (Entry<String, Integer> entry : kItemDataSetIter.entrySet())  {  String key = entry.getKey();  Integer value = entry.getValue();  minimumSupport = value / totalTransaction;  if(minimumSupport >= support)  {  refinedkItemDataSetIter.put(key, value);  }  }  for (Entry<String, Integer> entry : refinedkItemDataSetIter.entrySet())  {  String key = entry.getKey();  str1arr = key.toCharArray();  Integer value = entry.getValue();  for(int h=0; h<str1arr.length; h++)  {  for(Entry<String, String> entry1 : nameHashSet.entrySet())  {  String key1 = entry1.getKey();  String value1 = entry1.getValue();  if(key1.contains(str1arr[h]+""))  {  collectiveKey1 = collectiveKey1 + value1;  if(h<str1arr.length-1)  {  collectiveKey1 = collectiveKey1 + ",";  }  }  }  }  System.out.println(collectiveKey1 + "-->" + value);  collectiveKey1 = "";  }  return refinedkItemDataSetIter;  }  public void iterateKitemSetFinal()  {  int k=3;  HashMap<String, Integer> refinedKitemSetFinal = new HashMap<>();  refinedKitemSetFinal = kItemDataSet();  while((refinedKitemSetFinal.size()!=0) )  {  System.out.println();  System.out.println("FrequentItemSet for Step"+k+":");  refinedKitemSetFinal = refinedIterateKitemSet(refinedKitemSetFinal);  totaltransactionSet.putAll(refinedKitemSetFinal);  k++;  }  System.out.println();  System.out.println("totaltransactionSet:"+totaltransactionSet);  }  public double keyValueFromHash(String searchValue)  {  double resultValue=0.0;  for (Entry<String, Integer> entry : totaltransactionSet.entrySet())  {  String key = entry.getKey();  if(searchValue.equals(key))  {  Integer Value = entry.getValue();  return Value;  }  }  return resultValue;  }  public char[] charSwap(char[] a,int i, int j)  {  Character temp=null;  temp = a[i];  a[i] = a[j];  a[j] = temp;  return a;  }  public HashMap<String, Double> perm(String s)  {  int i=0,k=0, l=0,m=s.length();  char[] str1arr,str2arr;  String collectiveKey = "", singleKey, collectiveKey1="",collectiveKey2="";  TransactionGeneration T = new TransactionGeneration();  HashMap<String, String> nameHashSet = T.nameItemSet();  double minimumConfidence, supportXY, supportX;  HashMap<String, Double> assocRuleHashtempTable = new HashMap<String, Double>();  String s\_cpy, str1, str2, str3, assocResult;  char c[];  s\_cpy = s;  while(i<=s.length()-1)  {  str1 = s.substring(0,i+1);  for(int j=i+1;j<m+1; j++)  {  str1 = s\_cpy.substring(0,i+1);  str1arr = str1.toCharArray();  if(str1==s)  {  break;  }  supportXY =keyValueFromHash(str1);  if(supportXY==0.0)  {  continue;  }  str2 = s\_cpy.substring(i+1);  str2arr = str2.toCharArray();  supportX=keyValueFromHash(str2);  if(supportX==0.0)  {  continue;  }  minimumConfidence = supportXY/supportX;  if(minimumConfidence>confidence)  {  assocResult = str1 + "-->" + str2;  for(int h=0; h<str1arr.length; h++)  {  for (Entry<String, String> entry : nameHashSet.entrySet())  {  String key = entry.getKey();  String value = entry.getValue();  if(key.contains(str1arr[h]+""))  {  collectiveKey1 = collectiveKey1 + value;  if(h<str1arr.length-1)  {  collectiveKey1 = collectiveKey1 + ",";  }  }  }  }  for(int h=0; h<str2arr.length; h++)  {  for (Entry<String, String> entry : nameHashSet.entrySet())  {  String key = entry.getKey();  String value = entry.getValue();  if(key.contains(str2arr[h]+""))  {  collectiveKey2 = collectiveKey2 + value;  if(h<str2arr.length-1)  {  collectiveKey2 = collectiveKey2 + ",";  }  }  }  System.out.println("collectivekey2:"+collectiveKey2);  }  collectiveKey=collectiveKey1 +"-->"+collectiveKey2;  assocRuleHashTable.put(collectiveKey, minimumConfidence);  collectiveKey ="";  collectiveKey1 = "";  collectiveKey2 = "";  System.out.println("supportXy:"+supportXY);  System.out.println("supportX:"+supportX);  // System.out.println("Association rule:" +str1 + "->" + str2 +(minimumConfidence) );  }  c = s\_cpy.toCharArray();  if(j<m)  c=charSwap(c, k, j);  str3 = String.valueOf(c);  s\_cpy = str3;  }  s\_cpy = s;  i++;  k++;  }  return assocRuleHashtempTable;  }  /\*  Returns Hash map containing Association Rules  \*/  public void generateAssociationRule()  {  int i;  String singleKey, collectiveKey ="";  TransactionGeneration T = new TransactionGeneration();  HashMap<String, String> nameHashSet = T.nameItemSet();  iterateKitemSetFinal();  for (Entry<String, Integer> entry : totaltransactionSet.entrySet())  {  String key = entry.getKey();  perm(key);  }  System.out.println("Association Rule: ==== Confidence");  for (Entry<String, Double> entry : assocRuleHashTable.entrySet())  {  String key = entry.getKey();  Double Value = entry.getValue();  System.out.println(key +" ==== "+ Value);  }  System.out.println();  System.out.println("size of Association rule:"+assocRuleHashTable.size());  }  } |

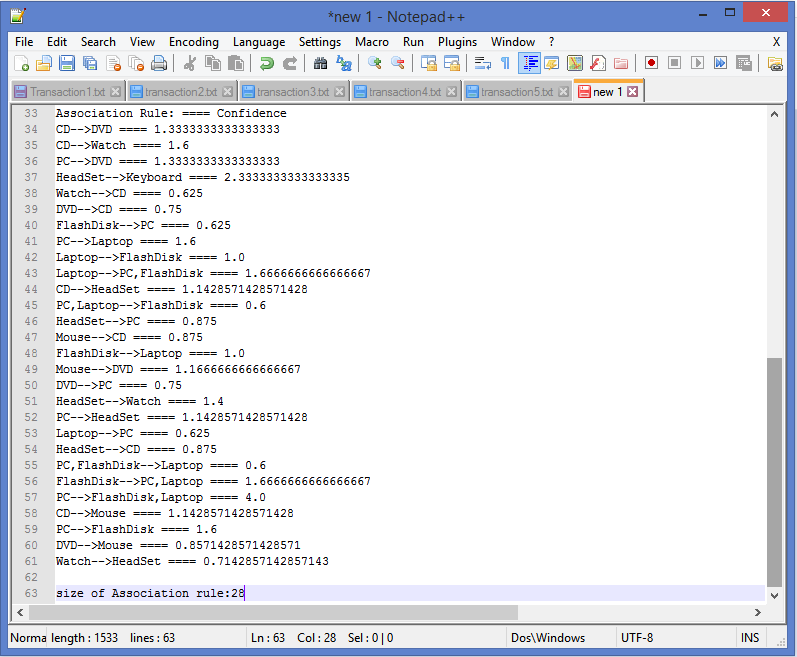
DataAnalysis.java

|  |
| --- |
| package Mining**;**  **import** java**.**util**.**Scanner**;**  public class DataAnalysis **{**  static public double confidence**;**  static public double support**;;**  static public double totalTransaction **=** 20**;**  public static void main**(**String**[]** args**)** **{**  // TODO Auto-generated method stub  Scanner Scn **=** **new** Scanner**(**System**.**in**);**  System**.**out**.**println**(**"Enter minimum support percentage(between 0 to 100):"**);**  support **=** Scn**.**nextDouble**()/**100**;**  System**.**out**.**println**(**"Enter minimum confidence percentage(between 0 to 100):"**);**  confidence **=** Scn**.**nextDouble**()/**100**;**  TransactionGeneration T **=** **new** TransactionGeneration**();**  AprioriAlgorithmImplementation A **=** **new** AprioriAlgorithmImplementation**();**  A**.**generateAssociationRule**();**  **}**  **}** |

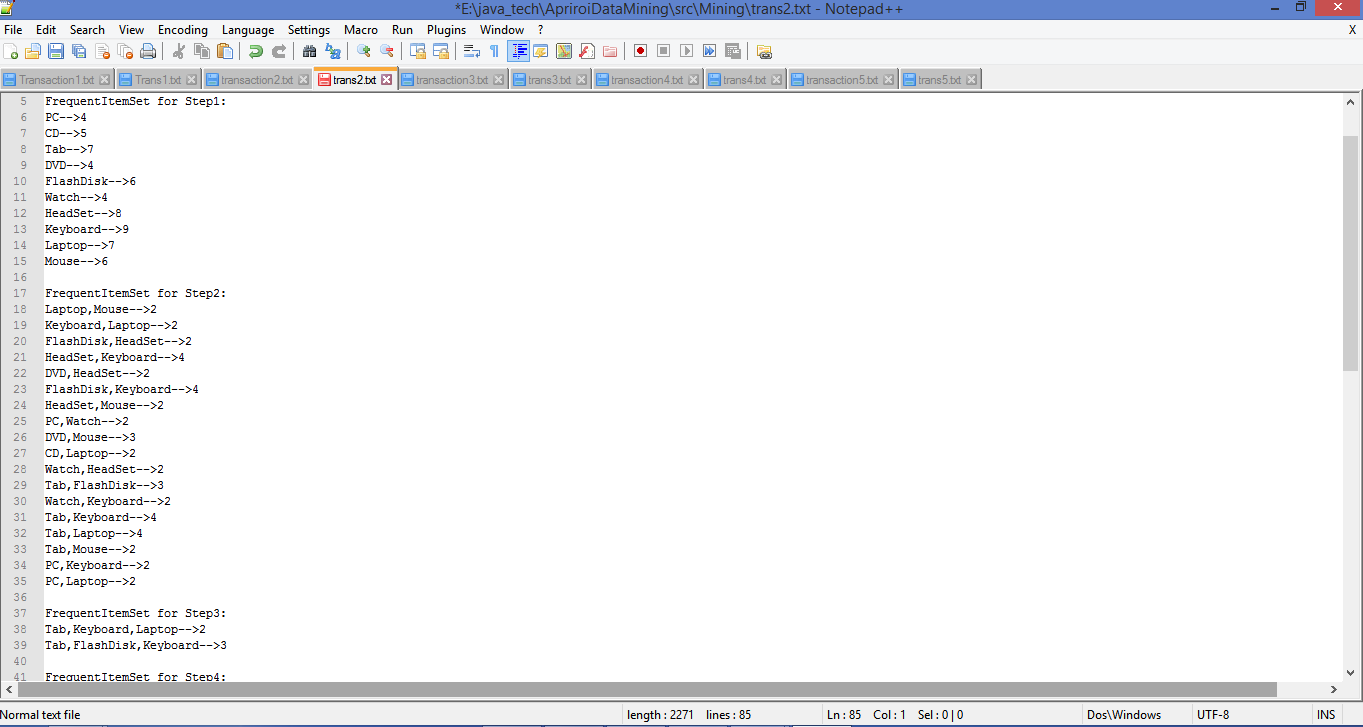
# SCREENSHOTS (Output copied from console into Notepad+):

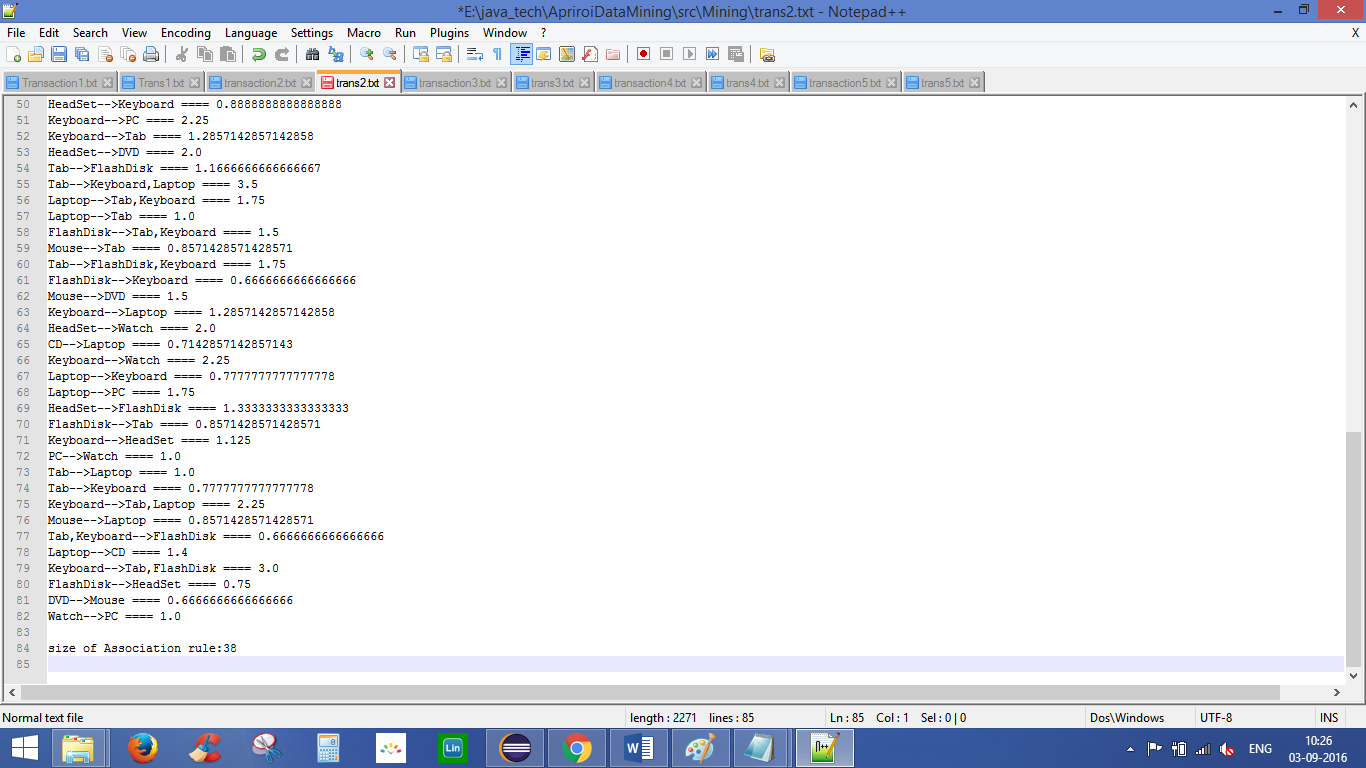
The Output for the **1st Transaction** Database with the Support values **10** and Confidence value **50.**



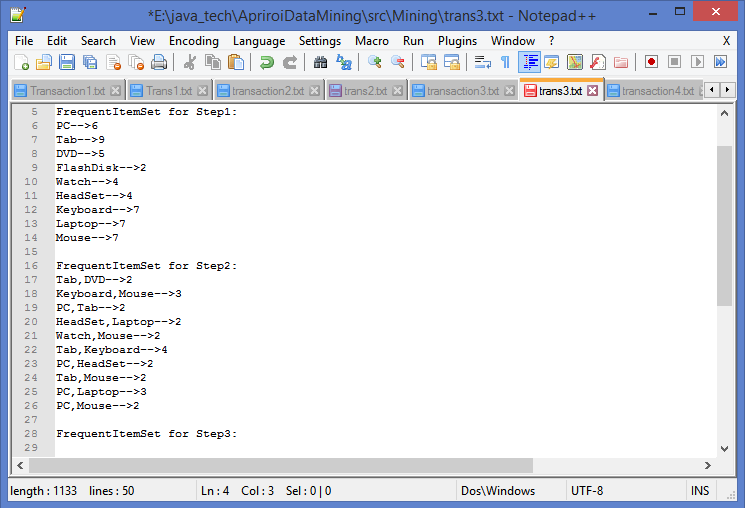


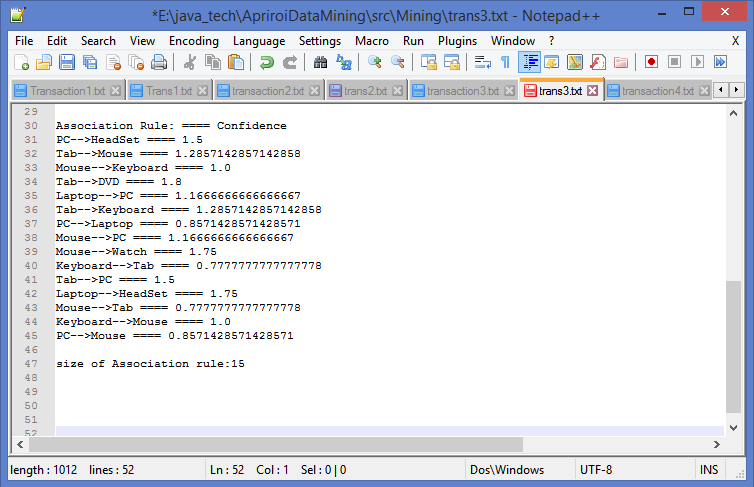
The Output for the **2nd Transaction** Database with the Support values **10** and Confidence value **60.**



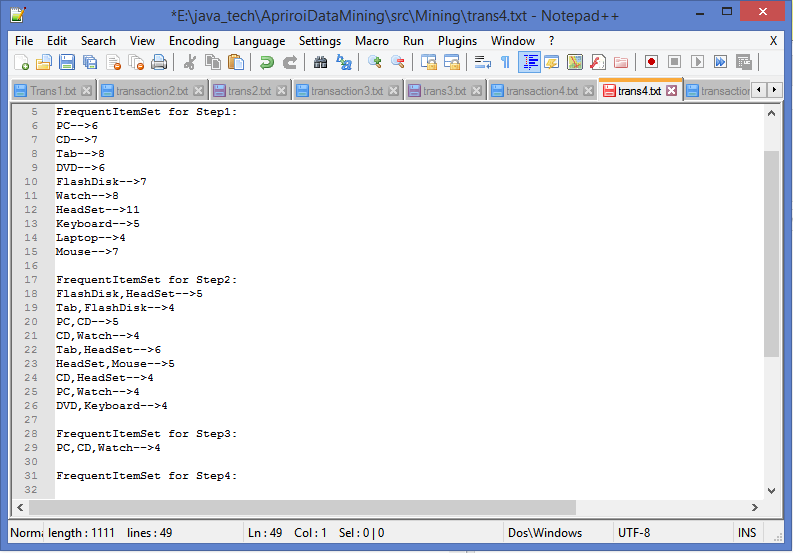


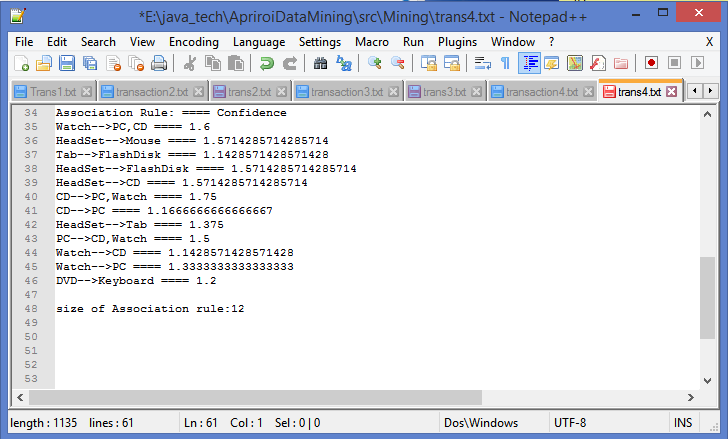
The Output for the **3rd Transaction** Database with the Support values **10** and Confidence value **70.**



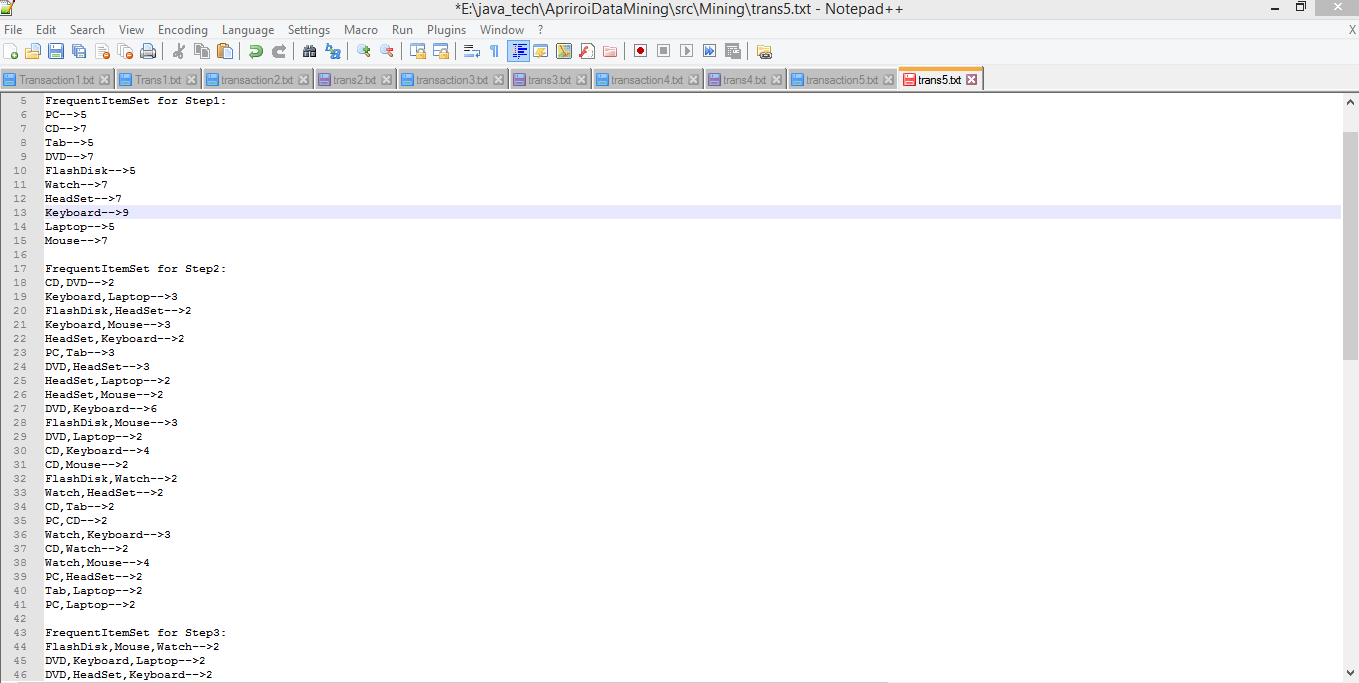


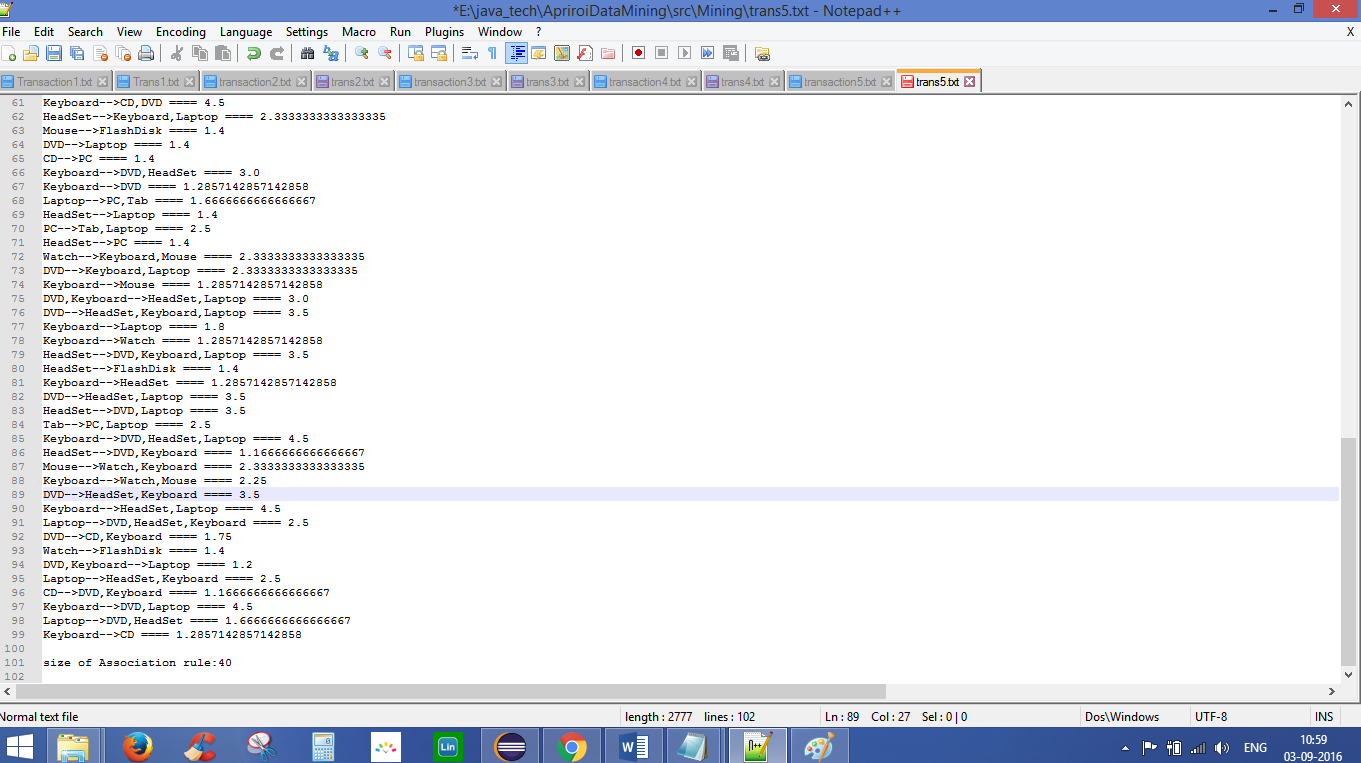
The Output for the **4th Transaction** Database with the Support values **20** and Confidence value **100.**





The Output for the **5th Transaction** Database with the Support values **10** and Confidence value **100.**





# REFERENCES:

[https://en.wikipedia.org/wiki/**Apriori**\_**algorithm**](https://en.wikipedia.org/wiki/Apriori_algorithm)

<http://www.codeproject.com/Articles/70371/Apriori-Algorithm>

<https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>