



## Relative Competence Centered Scrutiny and Implementation of Apriori, FP – Growth and Mapreduce Algorithms

Manpreet Kaur<sup>1</sup>, Prof.(Dr.)Vishal Goyal<sup>2</sup>

<sup>1</sup>Research Scholar (M.Phil.), Dept. of Comp. Science, Punjabi University, Patiala, Punjab, India.

<sup>2</sup>Professor, Dept. of Computer Science, Punjabi University, Patiala, Punjab, India.

**\*Corresponding Author:** Manpreet Kaur, Research Scholar (M.Phil.), Dept. of Comp. Science, Punjabi University, Patiala, Punjab, India.

**Abstract:** The major rise in data collection and storage has raised the necessity for much more powerful data analysis tools. The data collected in huge databases needs to be handled effectively and efficiently. The important and highly critical decisions are made not on the basis of information rich data stored in databases but instead on a decision maker's instinct merely because of the absence of the tools capable of extracting the valuable knowledge from vast amount of the data. Currently expert systems depends on users to manually input knowledge into knowledge bases. This process is often time consuming, expensive, and bias. The problem with data mining algorithms are their non-capability of dealing with non-static, and unbalanced data. There is a need for constantly updating the models to handle data velocity or new incoming data.

The objectives of the research paper is to implement the three popular data mining algorithms (Apriori algorithm, FP – Growth algorithm, and Map Reduce algorithm) using appropriate programming tool (preferably Java). The paper also perform comparative analysis of the three algorithms under study via measuring efficiency in terms of time. The paper also elaborates on analysis of all three algorithms on the basis of performance evaluation using accuracy metric.

**Keywords:** Apriori algorithm, data minng, FP – Growth algorithm, Map Reduce algorithm.

### 1. INTRODUCTION

Data mining deals with the kind of patterns that can be mined. On the basis of the kind of data to be mined, there are two categories of functions involved in data mining mentioned as under.

- Descriptive
- Classification and Prediction

#### 1.1. Descriptive Function

The descriptive function deals with the general properties of data in the database and are mentioned as under [1].

##### 1.1.1. Class/Concept Description

Class/Concept refers to the data to be associated with the classes or concepts. For example, in a company, the classes of items for sales include computer and printers, and concepts of customers include big spenders and budget spenders. Such descriptions of a class or a concept are called class/concept descriptions. These descriptions can be derived by the following two ways.

- Data Characterization – Data characterization refers to summarizing data of class under study. This class under study is called as Target Class.
- Data Discrimination – It refers to the mapping or classification of a class with some predefined group or class.

### 1.1.2. Mining of Frequent Patterns

Frequent patterns are those patterns that occur frequently in transactional data (FP - Growth). Here is the list of kind of frequent patterns.

- Frequent Item Set – It refers to a set of items that frequently appear together, for example, milk and bread.
- Frequent Subsequence – A sequence of patterns that occur frequently such as purchasing a camera is followed by memory card.
- Frequent Sub Structure – Substructure refers to different structural forms, such as graphs, trees, or lattices, which may be combined with item-sets or subsequences.

### 1.1.3. Mining of Association

Associations are used in retail sales to identify patterns that are frequently purchased together. This process refers to the process of uncovering the relationship among data and determining association rules.

For example, a retailer generates an association rule that shows that 70% of time milk is sold with bread and only 30% of times biscuits are sold with bread.

### 1.1.4. Mining of Correlations

It is a kind of additional analysis performed to uncover interesting statistical correlations between associated-attribute-value pairs or between two item sets to analyze that if they have positive, negative or no effect on each other.

### 1.1.5. Mining of Clusters

Cluster refers to a group of similar kind of objects. Cluster analysis refers to forming group of objects that are very similar to each other but are highly different from the objects in other clusters.

## 1.2. Classification and Prediction

Classification is the process of finding a model that describes the data classes or concepts [2]. The purpose is to be able to use this model to predict the class of objects whose class label is unknown. This derived model is based on the analysis of sets of training data. The derived model can be presented as classification (if-then) rules, decision trees, mathematical formulae, and neural networks [3].

The list of functions involved in these processes are mentioned as under.

### 1.2.1. Classification

It predicts the class of objects whose class label is unknown. Its objective is to find a derived model that describes and distinguishes data classes or concepts. The Derived Model is based on the analysis set of training data i.e. the data object whose class label is well known.

### 1.2.2. Prediction

It is used to predict missing or unavailable numerical data values rather than class labels. Regression Analysis is generally used for prediction. Prediction can also be used for identification of distribution trends based on available data [3].

### 1.2.3. Outlier Analysis

Outliers may be defined as the data objects that do not comply with the general behavior or model of the data available.

### 1.2.4. Evolution Analysis

Evolution analysis refers to the description and model regularities or trends for objects whose behavior changes over time.

## 2. DATA MINING ALGORITHMS

The research work covers the detailed study and implementation of three data mining algorithms: Apriori algorithm, FP-Growth algorithm, and MapReduce algorithm.

### 2.1. Apriori Algorithm

In data mining, Apriori algorithm [2, 3] is a traditional algorithm used for learning association rules. Association rules are referred to those statements which are used to find the relation between the data items of the database. Apriori algorithm is used to mine the frequent data items and corresponding association rule in the database of the transactions. Apriori algorithm is based upon the bottom up strategy in which the common subset of data items is expanded to add one more item at a time and then it is checked against the minimum support. Minimum support is the minimum value used to search frequent patterns that satisfy this restriction [4, 5]. The mining of association rule from huge amount of data assist the companies in taking important decisions regarding their business. This rule is used in many fields such as storage planning, analysis of customer shopping, good shelves design etc [4, 12].

#### Flowchart for Apriori algorithm

The working of Apriori algorithm is shown in the flowchart below depicted in Fig.1.

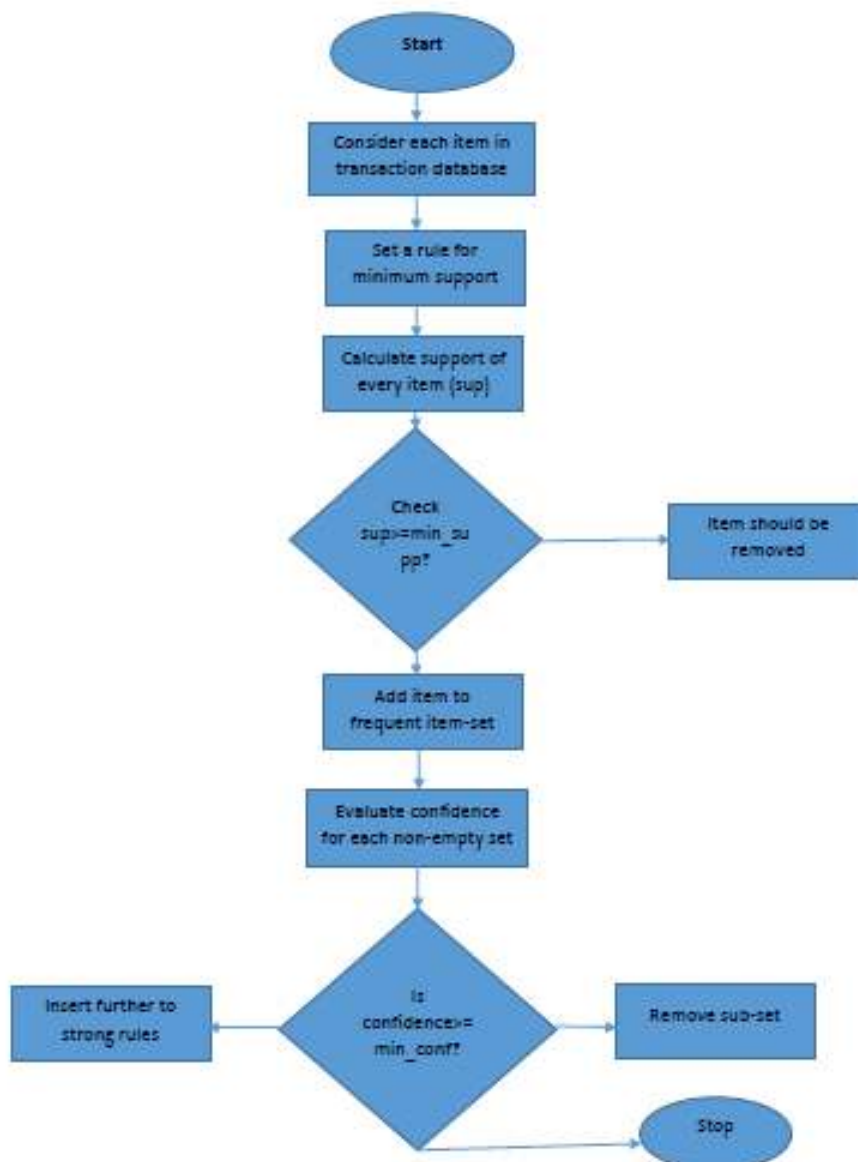


Fig1. The flowchart depicts the working of Apriori algorithm

**Example of Apriori Algorithm**

The data in the Table 1 is taken as input where “T.Id” refers to Transaction\_Id, “Items bought” shows the items bought together. The minimum support for example under study is set to 3.

**Table1.** Table shows the Input data

T.Id	Items bought
1	Cookies, tea, cake
2	Bread, tea, butter
3	Cookies, Bread, tea, butter
4	Bread, butter
5	pan cakes

Calculate the number of times each item appears in the table.

**Table2.** Table displays the items against frequency of its occurrence

Items bought	Support
Cookies	2
Bread	3
Tea	3
Butter	3
Pan cakes	1
Cake	1

Only items having occurrence equal to or greater than 3 are moved to next stage

**Table3.** Table shows the qualified items

Items bought	Support
Bread	3
Tea	3
Butter	3

Reassemble the three items with possible combinations.

**Table4.** Table shows the possible combinations of shortlisted items

Items bought
Bread, tea
Bread, butter
Tea, butter

Calculate the occurrences of combinations in Table 4

**Table5.** Table shows the occurrences of combinations

Items bought	Support
Bread, tea	2
Bread, butter	3
Tea, butter	2

Discard the products having minimum support less than 3.

**Table6.** Final output

Items bought	Support
Bread, butter	3

Only one item set with frequent item set is left with support 3.

**2.2. FP-Growth Algorithm**

Frequent patterns are item sets, subsequences, or substructures that appear in a data set with frequency no less than a user-specified threshold. For example, a set of items, such as milk and bread that appear frequently together in a transaction data set is a frequent item

set. Pattern mining can be applied on various types of data such as transaction databases, sequence databases, streams, strings, spatial data, graphs, etc. Frequent patterns are those patterns that occur frequently in transactional data [4, 5].

The most popular algorithm for pattern mining is the FP-Growth algorithm. The main idea of the algorithm is to use a divide and conquer strategy. Compress the database which provides the frequent sets; then divide this compressed database into a set of conditional databases, each associated with a frequent set and apply data mining on each database. It is designed to be applied on a transaction database to discover patterns in transactions made by customers in stores. But it can also be applied in several other applications [6, 11, 13].

### Flowchart for FP-Growth algorithm

The flowchart for FP-Growth algorithm is shown below in Fig. 2.

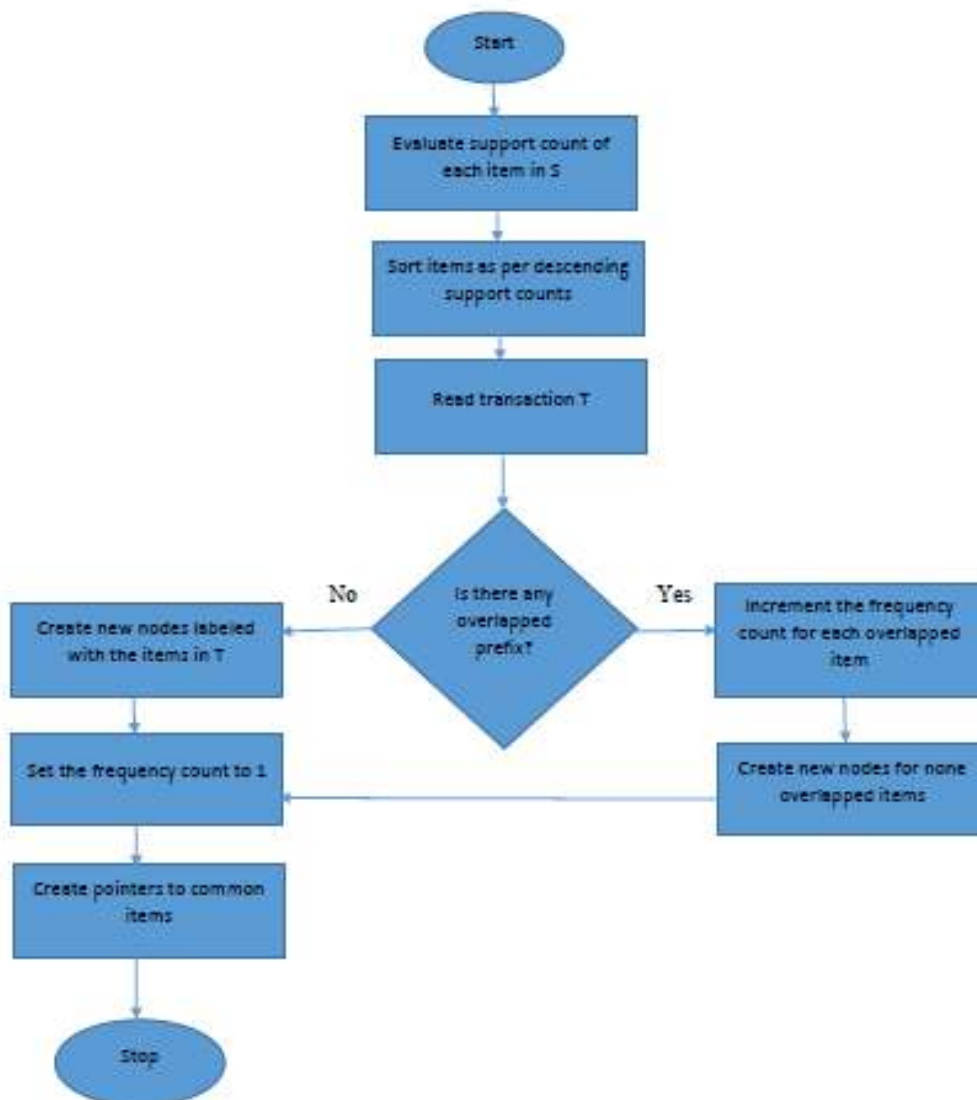


Fig2. The flowchart depicts the working of FP-Growth algorithm

### Pros and cons of FP-Growth

The pros and cons related to FP-Growth algorithm are mentioned as under [10].

#### Pros

- The major advantage of the FP-Growth algorithm is that it takes only two passes over the data set.
- The FP-Growth algorithm compresses the data set because of overlapping of paths.

- The candidate generation is not required.
- The working of the FP-Growth algorithm is much faster as compared to the Apriori algorithm.

*Cons*

- The FP-Growth algorithm may not fit into the memory.
- The FP-Growth algorithm is expensive to construct. It consumes time to build. But once it is done with construction, itemsets can be read off easily.
- Enormous time is wasted when support threshold is high as pruning can be practiced only on single items.
- The process of calculating the support can be carried out only after the entire data set is added to the FP-Tree.

### **2.3. Mapreduce Algorithm**

MapReduce is parallel programming paradigm that enables the distributed processing of massive data sets across the large cluster of commodity servers. The concept of MapReduce is easily understandable. The data which is given as input is usually very large in size and to complete it in specific time, it has to be distributed over the thousands of servers [6, 7, 9].

The Processing of MapReduce algorithm divides into six steps:

#### *2.3.1. Job Submission*

When the user writes a basic program for the creation of new JobClient, the JobClient send the request to JobTracker to get a new JobID. Then the JobClient will check whether the input and output directories are correct. After this, the JobClient will store the resources like the number of input data fragmentations, the configuration files and mapper/reducer JAR files to HDFS. Basically, JAR files will be keep as several backups. After all of this, the JobClient will submit a job request to JobTracker [8].

#### *2.3.2. Job Initialization*

JobTracker is the master of the system so it will take many JobClient requests. All the requests are placed in a queue which is managed by the job scheduler. Once the JobTracker starts to initialize, its job is to make a JobInProgress case to signify a job. The JobTracker must retrieve the input data from HDFS and to decide on the number of the map tasks. The reduce tasks and TaskInProgress are determined by the parameters in the configuration files.

#### *2.3.3. Task Allocation*

Firstly, the TaskTracker has to be launched which is responsible for the map and reduce tasks. The TaskTracker will send the message to the JobTracker for the completion of the task. When the job queue of the JobTracker is not empty, the TaskTracker will receive the tasks to do. Because of the shortage of TaskTracker computing capability, it can handle limited tasks. The TaskTracker basically have two task slots i.e. map task and reduce task. During task allocation, the JobTracker initially use the map task. Once the map task slot is empty it will receive another job task. When it is full, then the reduce task will receive the tasks to do.

#### *2.3.4. Map Tasks Execution*

In the map TaskTracker, there is a series of operations for the completion of the tasks. Initially, the map TaskTracker will make a TaskInProgress object to schedule and monitor the tasks. Secondly, the map TaskTracker will copy the JAR files and linked configuration files from HDFS to the local working directory. When all these things are completed, the TaskTracker will create a new TaskRunner to run the map task. The TaskRunner can launch a distinct JVM and will begin the map task within to execute map() function. During the execution, the map task can communicate with TaskTracker to report task progress until all the tasks are completed. At that point, all the computing results are stored within the local disk.



2.3.5. Reduce Tasks Execution

when the task execution of map tasks is completed, the JobTracker will follow the same procedure with reduce TaskTracker to allocate the tasks. The reduce TaskTracker also execute the reduce() function in separate JVM. At that point, the reduce task will download the results from map TaskTracker. When all the map tasks completed their execution, the JobTracker notify the reduce TaskTracker to start the execution. The same way, reduce task will communicate about the progress with TaskTracker until all the tasks are finished.

2.3.6. Job Completion

At each stage of reduce execution, all the results of reduce task will stored in the temporary file in HDFS. When the execution of all redcetasks is completed, all these temporary files are combined together into the final output file. The JobTracker received the message of completion and the JobClient notify the user and display the required information.

2.4. Algorithm For Mapreduce Algorithm

The step wise working of MapReduce algorithm in mentioned below [14].

- The incoming data can be alienated into n number of modules which depends upon the amount of input data and processing power of the individual unit.
- All these fragmented modules are then passed over to mapper function where these modules undergo simultaneous parallel processing.
- Thereafter, shuffling is conducted in order to gather similar looking patterns.
- Finally, reducer function is called which is responsible for getting the ultimate output in a reduced form.
- Moreover, this technique is scalable and depending upon increase in the data to be processed, the processing units can be further extended.

The working of MapReduce algorithm is shown in the flowchart depicted in Fig. 3 below.

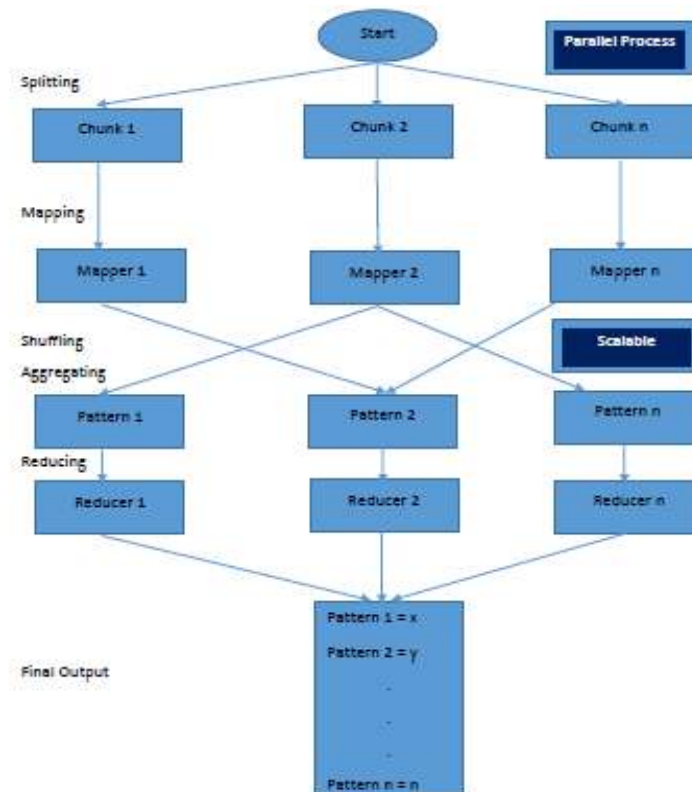


Fig3. The flowchart depicts the detailed working of MapReduce algorithm

### 3. CONTRIBUTION AND IMPLEMENTATION

A database titled “commondatabase.data” shown in Fig. 4 and Fig. 5 has been constructed which consists of 3196 rows and 37 columns i.e. each entry consists of 37 numbers.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	34	36	38	40	
2	1	3	5	7	9	12	13	15	17	19	21	23	25	27	29	31	34	36	38	40	
3	1	3	5	7	9	12	13	16	17	19	21	23	25	27	29	31	34	36	38	40	
4	1	3	5	7	9	11	13	15	17	20	21	23	25	27	29	31	34	36	38	40	
5	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	34	36	38	40	
6	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	34	36	38	40	
7	1	3	5	7	9	11	13	15	17	20	21	23	25	27	29	31	34	36	38	40	
8	1	3	5	7	9	12	13	15	17	19	21	24	25	27	29	31	34	36	38	40	
9	1	3	5	7	9	11	13	15	17	19	21	24	25	27	29	31	34	36	38	40	
10	1	3	5	7	9	11	13	16	17	19	21	24	25	27	29	31	34	36	38	40	
11	1	3	5	7	9	12	13	16	17	19	21	24	25	27	29	31	34	36	38	40	
12	1	3	5	7	9	11	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
13	1	3	5	7	9	11	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
14	1	3	5	7	9	11	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
15	1	3	5	7	9	12	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
16	1	3	5	7	9	12	13	15	17	19	21	24	25	27	29	31	34	36	38	40	
17	1	3	5	7	9	11	13	15	17	19	21	24	25	27	29	31	34	36	38	40	
18	1	3	5	7	9	12	13	16	17	19	21	24	25	27	29	31	34	36	38	40	
19	1	3	5	7	9	11	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
20	1	3	5	7	9	11	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
21	1	3	5	7	9	11	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
22	1	3	5	7	9	12	13	15	17	20	21	24	25	27	29	31	34	36	38	40	
23	1	3	5	7	9	11	13	15	17	19	21	24	25	27	29	32	34	36	38	40	

Fig4. The figure displays the snapshot of constructed database

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
3174	2	3	5	7	9	11	13	16	17	20	21	23	25	27	29	32	34	36	39	40	
3175	2	3	5	7	9	11	13	16	18	20	21	23	25	27	29	32	34	36	38	40	
3176	2	3	5	7	9	11	13	16	17	19	21	23	25	27	29	33	34	36	38	40	
3177	2	3	5	7	9	11	13	16	17	19	21	23	25	27	29	31	34	36	38	40	
3178	2	3	5	7	9	11	13	16	17	19	21	23	25	27	29	31	34	36	38	40	
3179	2	4	5	7	9	11	13	16	17	19	21	23	25	27	29	33	34	36	38	40	
3180	2	3	5	7	9	11	13	16	17	19	21	23	26	28	29	31	34	36	38	40	
3181	2	4	5	7	9	11	13	16	17	19	21	23	26	28	29	33	34	36	38	40	
3182	2	4	5	7	9	11	13	16	17	19	21	23	26	28	29	33	34	36	38	40	
3183	2	4	5	7	9	11	13	16	17	19	21	23	26	28	29	31	34	36	38	40	
3184	2	4	5	7	9	12	13	16	17	20	21	23	26	27	29	33	34	36	38	40	
3185	2	4	5	7	9	12	13	16	17	20	21	23	26	27	29	31	34	36	38	40	
3186	2	3	5	7	9	12	13	16	17	20	21	23	26	27	29	31	34	36	39	40	
3187	2	4	5	7	9	12	13	16	17	20	21	23	26	27	29	33	34	36	39	40	
3188	2	3	5	7	10	11	13	16	18	20	21	23	26	27	29	32	34	36	38	40	
3189	2	3	5	7	10	11	13	16	18	20	21	23	26	27	29	32	34	36	38	40	
3190	2	3	5	7	10	11	13	16	18	20	21	23	26	27	29	32	34	36	39	40	
3191	2	3	5	7	9	11	13	16	17	19	21	23	26	27	29	31	34	36	38	40	
3192	2	4	5	7	9	11	13	16	17	19	21	23	26	27	29	33	34	36	38	40	
3193	2	4	5	7	9	11	13	16	17	19	21	23	26	27	29	33	34	36	38	40	
3194	2	4	5	7	9	11	13	16	17	19	21	23	26	27	29	31	34	36	38	40	
3195	2	4	5	8	9	11	13	16	17	19	21	23	26	27	30	33	35	36	38	40	
3196	2	4	5	8	9	11	13	16	17	19	21	23	26	27	30	31	35	36	38	40	

Fig5. The figure displays the second snapshot of constructed database



**Evaluating working of Apriori algorithm and FP – Growth algorithm at minimum support of 80% (minsup>= 0.8)**

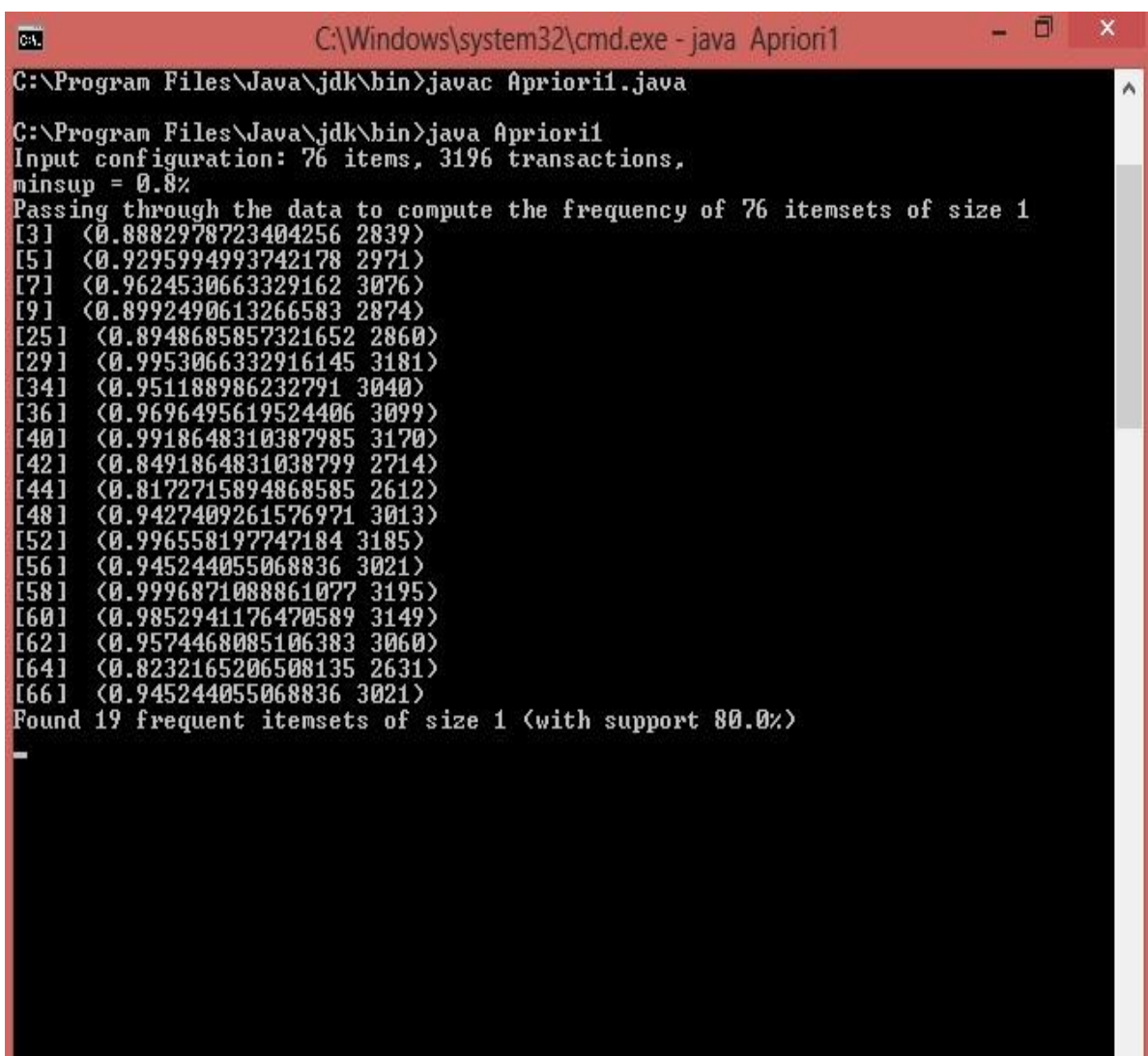
The constructed database is given as input to the Apriori algorithm program developed in Java to find out the frequent itemsets of sizes ranging from 1 to 14 with minimum support value of 80% (minsup=0.8%). As the total number of entries in the database is 3196, the 80% of this value is 2556.8 (3196 \* .80). So the extracted answer will contain only those itemsets whose support value occurrence is above 2556.8.

Fig. 6 shows the result obtained in accordance with itemsets of size 1. The first row extracted is as follows.

[3]<0.88829787234042562839>

Here [3] refers to the item been scanned.

0.8882978723404256 shows the support value of item [3] which is 88.82978% and is clearly above the minimum support value of 80% or is above minsup value of .80.



```
C:\Windows\system32\cmd.exe - java Apriori1
C:\Program Files\Java\jdk\bin>javac Apriori1.java
C:\Program Files\Java\jdk\bin>java Apriori1
Input configuration: 76 items, 3196 transactions,
minsup = 0.8%
Passing through the data to compute the frequency of 76 itemsets of size 1
[3] <0.8882978723404256 2839>
[5] <0.9295994993742178 2971>
[7] <0.9624530663329162 3076>
[9] <0.8992490613266583 2874>
[25] <0.8948685857321652 2860>
[29] <0.9953066332916145 3181>
[34] <0.951188986232791 3040>
[36] <0.9696495619524406 3099>
[40] <0.9918648310387985 3170>
[42] <0.8491864831038799 2714>
[44] <0.8172715894868585 2612>
[48] <0.9427409261576971 3013>
[52] <0.996558197747184 3185>
[56] <0.945244055068836 3021>
[58] <0.9996871088861077 3195>
[60] <0.9852941176470589 3149>
[62] <0.9574468085106383 3060>
[64] <0.8232165206508135 2631>
[66] <0.945244055068836 3021>
Found 19 frequent itemsets of size 1 (with support 80.0%)
```

**Fig6.** shows the result obtained in accordance with itemsets of size 1 having minsup value >=.80

Fig. 7 shows the result obtained in accordance with itemsets of size 2. The first row in the result extracted is as follows.

[9, 66] <0.8494993742177722 2715>

Here [9, 66] refers to the itemset been scanned.

0.8494993742177722 shows the support value of itemset [9, 66] which is 84.94993742% and is clearly above the minimum support value of 80% or is above minsup value of .80.

```
[9, 66] <0.8494993742177722 2715>
[7, 58] <0.9621401752190237 3075>
[29, 64] <0.818523153942428 2616>
[60, 62] <0.9430538172715894 3014>
[29, 44] <0.8125782227784731 2597>
[7, 9] <0.8620150187734669 2755>
[40, 64] <0.8229036295369212 2630>
[48, 62] <0.9001877346683355 2877>
[36, 60] <0.9549436795994993 3052>
[48, 56] <0.8879849812265331 2838>
[5, 60] <0.9148936170212766 2924>
[5, 7] <0.8945556946182729 2859>
[25, 60] <0.886107634543179 2832>
[52, 60] <0.9818523153942428 3138>
[5, 62] <0.8945556946182729 2859>
[9, 52] <0.8967459324155194 2866>
[44, 60] <0.8025657071339174 2565>
[56, 62] <0.9145807259073843 2923>
[7, 36] <0.9321026282853567 2979>
[52, 56] <0.9436795994993742 3016>
[3, 66] <0.8457446808510638 2703>
[3, 52] <0.8854818523153942 2830>
[60, 66] <0.945244055068836 3021>
[36, 52] <0.9662077596996246 3088>
[25, 66] <0.8494993742177722 2715>
[44, 58] <0.8169586983729662 2611>
[7, 60] <0.9483729662077597 3031>
[9, 60] <0.8870463078848561 2835>
[29, 34] <0.9499374217772215 3036>
[42, 56] <0.8041301627033792 2570>
[5, 34] <0.9039424280350438 2889>
[36, 42] <0.8216520650813517 2626>
[7, 56] <0.9133291614518148 2919>
[25, 56] <0.8438673341677096 2697>
[3, 48] <0.8360450563204005 2672>
[42, 62] <0.8128911138923655 2598>
[34, 62] <0.9114518147684606 2913>
[29, 62] <0.9527534418022529 3045>
[40, 66] <0.9377346683354193 2997>
[40, 48] <0.9346057571964956 2987>
[7, 25] <0.8582603254067585 2743>
[9, 58] <0.898936170212766 2873>
[29, 56] <0.9405506883604505 3006>
[9, 48] <0.8419899874843555 2691>
[29, 42] <0.8454317897371715 2702>
[48, 60] <0.9342928660826032 2986>
[36, 62] <0.9270963704630788 2963>
[58, 66] <0.9449311639549437 3020>
[34, 42] <0.814142678347935 2602>
[9, 40] <0.8923654568210263 2852>
[48, 58] <0.9424280350438048 3012>
[52, 62] <0.9540050062578223 3049>
[5, 9] <0.8394868585732165 2683>
[34, 56] <0.9023779724655819 2884>
Found 141 frequent itemsets of size 2 <with support 80.0%>
```

Fig7. shows the result obtained in accordance with itemsets of size 2

Fig. 8 shows the result obtained in accordance with itemsets of size 3. The first row in the result extracted is as follows.

```
[48, 52, 66] <0.895494367959952862>
```

Here [48, 52, 66] refers to the itemset been scanned.

0.89549436795995 shows the support value of itemset [48, 52, 66] which is 89.549436795995% and is clearly above the minimum support value of 80% or is above minsup value of .80.

```
[48, 52, 66] <0.89549436795995 2862>
[7, 58, 66] <0.9089486858573217 2905>
[58, 60, 62] <0.9430538172715894 3014>
[58, 60, 64] <0.80819774718398 2583>
[7, 25, 34] <0.8144555694618273 2603>
[7, 40, 48] <0.8970588235294118 2867>
[42, 52, 56] <0.8025657071339174 2565>
[3, 40, 48] <0.827909887359199 2646>
[7, 48, 60] <0.8973717146433041 2868>
[3, 40, 66] <0.8382352941176471 2679>
[5, 56, 60] <0.8623279098873592 2756>
[7, 36, 62] <0.889549436795995 2843>
[5, 29, 34] <0.9026908635794744 2885>
[25, 29, 58] <0.8908010012515645 2847>
[7, 29, 52] <0.9568210262828536 3058>
[7, 40, 66] <0.9017521902377973 2882>
[9, 40, 56] <0.8541927409261577 2730>
[7, 34, 48] <0.859511889862328 2747>
[3, 7, 48] <0.8128911138923655 2598>
[40, 58, 60] <0.9771589486858573 3123>
[48, 52, 60] <0.9308510638297872 2975>
[36, 48, 66] <0.8908010012515645 2847>
[42, 52, 58] <0.8469962453066333 2707>
[5, 7, 40] <0.8879849812265331 2838>
[58, 60, 66] <0.9449311639549437 3020>
[40, 58, 62] <0.9527534418022529 3045>
[7, 9, 34] <0.8272841051314143 2644>
[7, 48, 66] <0.8629536921151439 2758>
[7, 34, 66] <0.8688986232790988 2777>
[3, 52, 60] <0.8748435544430538 2796>
[3, 34, 58] <0.8466833541927409 2706>
[5, 36, 40] <0.8926783479349186 2853>
[5, 56, 66] <0.8335419274092616 2664>
[3, 9, 29] <0.8044430538172715 2571>
[25, 40, 62] <0.8520025031289111 2723>
[5, 36, 60] <0.8845431789737171 2827>
[48, 52, 62] <0.8967459324155194 2866>
[29, 34, 48] <0.8926783479349186 2853>
[3, 7, 29] <0.8651439299123905 2765>
[25, 29, 48] <0.8563829787234043 2737>
[34, 48, 56] <0.845118898623279 2701>
[48, 56, 58] <0.8876720901126408 2837>
[40, 58, 64] <0.8225907384230288 2629>
[7, 9, 36] <0.8316645807259074 2658>
[9, 56, 58] <0.8607634543178974 2751>
[7, 40, 62] <0.9152065081351689 2925>
[9, 29, 66] <0.8482478097622027 2711>
[3, 40, 62] <0.8476220275344181 2709>
[9, 48, 58] <0.8416770963704631 2690>
[7, 25, 58] <0.8579474342928661 2742>
Found 566 frequent itemsets of size 3 (with support 80.0%)
```

Fig8. shows the result obtained in accordance with itemsets of size 3

Fig. 9 shows the result obtained in accordance with itemsets of size 4. The first row in the result extracted is as follows.

[29, 48, 56, 66] <0.8416770963704631 2690>

Here [29, 48, 56, 66] refers to the itemset been scanned. 0.8416770963704631 shows the support value of itemset [29, 48, 56, 66] which is 84.16770963704631% and is clearly above the minimum support value of 80% or is above minsup value of .80.



```
[29, 48, 56, 66] <0.8416770963704631 2690>
[7, 9, 52, 60] <0.8479349186483104 2710>
[7, 40, 56, 62] <0.8782853566958698 2807>
[7, 56, 60, 66] <0.8601376720901126 2749>
[3, 52, 62, 66] <0.8078848560700876 2582>
[5, 36, 52, 56] <0.845118898623279 2701>
[7, 40, 48, 60] <0.889549436795995 2843>
[7, 25, 29, 56] <0.8110137672090113 2592>
[5, 9, 58, 60] <0.8269712140175219 2643>
[29, 34, 40, 58] <0.9424280350438048 3012>
[3, 7, 34, 52] <0.8213391739674594 2625>
[29, 34, 36, 60] <0.9055068836045056 2894>
[7, 36, 52, 60] <0.9145807259073843 2923>
[25, 40, 58, 66] <0.8416770963704631 2690>
[3, 5, 52, 60] <0.8122653316645807 2596>
[3, 29, 34, 66] <0.8085106382978723 2584>
[9, 40, 62, 66] <0.8088235294117647 2585>
[7, 9, 52, 66] <0.8113266583229036 2593>
[25, 29, 52, 56] <0.8388610763454318 2681>
[25, 34, 52, 56] <0.8022528160200251 2564>
[7, 36, 62, 66] <0.8385481852315394 2680>
[9, 29, 48, 62] <0.8006883604505632 2559>
[9, 29, 36, 60] <0.8548185231539425 2732>
[40, 52, 58, 66] <0.9339799749687109 2985>
[7, 40, 56, 60] <0.8917396745932415 2850>
[36, 42, 52, 58] <0.8194618272841051 2619>
[5, 58, 62, 66] <0.8526282853566959 2725>
[5, 40, 60, 62] <0.8760951188986232 2800>
[3, 9, 29, 52] <0.8025657071339174 2565>
[48, 58, 62, 66] <0.8585732165206508 2744>
[7, 29, 48, 52] <0.8995619524405507 2875>
[5, 34, 56, 62] <0.8310387984981227 2656>
[29, 34, 48, 66] <0.8554443053817271 2734>
[40, 48, 52, 56] <0.8785982478097623 2808>
[3, 7, 52, 62] <0.8263454317897372 2641>
[5, 48, 62, 66] <0.8063204005006258 2577>
[25, 29, 48, 66] <0.817584480600751 2613>
[7, 40, 42, 58] <0.8088235294117647 2585>
[7, 52, 56, 60] <0.8976846057571964 2869>
[29, 34, 36, 62] <0.8798498122653317 2812>
[25, 34, 52, 58] <0.8457446808510638 2703>
[3, 29, 36, 56] <0.8169586983729662 2611>
[29, 40, 56, 58] <0.932415519399249 2980>
[9, 29, 36, 62] <0.8275969962453066 2645>
[40, 52, 58, 64] <0.8222778473091364 2628>
[29, 34, 40, 56] <0.8939299123904881 2857>
[9, 29, 48, 60] <0.8341677096370463 2666>
[3, 29, 62, 66] <0.8107008760951189 2591>
[25, 48, 56, 60] <0.8035043804755945 2568>
[29, 36, 62, 66] <0.8720275344180225 2787>
Found 1383 frequent itemsets of size 4 (with support 80.0%)
```

Fig9. Shows the result obtained in accordance with itemsets of size 4

Similarly, the frequent itemsets of size 5 to 10 can be obtained.

Fig. 10 shows the result obtained in accordance with itemsets of size 10. The first row in the result extracted is as follows.

[7, 29, 36, 40, 48, 52, 58, 60, 62, 66] <0.8050688360450563 2573>

Here [7, 29, 36, 40, 48, 52, 58, 60, 62, 66] refers to the itemset been scanned.

0.8050688360450563 shows the support value of itemset [7, 29, 36, 40, 48, 52, 58, 60, 62, 66] which is 80.50688360450563 % and is clearly above the minimum support value of 80% or is above minsup value of .80. Fig. 10 also indicates that there are 2 unique itemsets of size 11 created from itemsets of size 10

The total time taken in the entire process is recorded as 103.432 seconds (103432 milliseconds) at minsup = .80.

```
[7, 29, 36, 40, 48, 52, 56, 58, 62] (0.8009136420525657 2586)
[7, 29, 36, 48, 52, 56, 58, 60, 66] (0.8025657071339174 2565)
[5, 29, 36, 40, 52, 56, 58, 60, 62] (0.80000625782227785 2557)
[7, 29, 36, 40, 52, 58, 60, 62, 66] (0.8291614518147684 2650)
[7, 29, 36, 40, 48, 52, 58, 62, 66] (0.8050688360450563 2573)
[29, 40, 48, 52, 56, 58, 60, 62, 66] (0.8072590738423029 2580)
[5, 29, 34, 40, 52, 56, 58, 60, 66] (0.804755944931164 2572)
[5, 7, 29, 36, 40, 48, 52, 58, 60] (0.8103879849812266 2590)
[7, 29, 34, 36, 40, 48, 58, 60, 66] (0.8069461827284106 2579)
[29, 34, 36, 48, 52, 58, 60, 62, 66] (0.8069461827284106 2579)
[7, 29, 36, 48, 52, 56, 58, 60, 62] (0.8060075093867334 2576)
[7, 29, 36, 40, 48, 52, 60, 62, 66] (0.8050688360450563 2573)
[29, 36, 40, 48, 52, 56, 58, 60, 62] (0.8310387984981227 2656)
[5, 7, 29, 34, 40, 52, 58, 60, 66] (0.8191489361702128 2618)
[7, 29, 36, 40, 48, 58, 60, 62, 66] (0.8085106382978723 2584)
[29, 36, 48, 52, 56, 58, 60, 62, 66] (0.8035043804755945 2568)
[7, 29, 36, 40, 52, 56, 58, 60, 62] (0.8304130162703379 2654)
[29, 34, 40, 52, 56, 58, 60, 62, 66] (0.8207133917396746 2623)
[7, 29, 36, 40, 48, 56, 58, 60, 62] (0.8031914893617021 2567)
[5, 29, 36, 40, 48, 52, 58, 60, 62] (0.8119524405506884 2595)
[29, 34, 36, 40, 52, 56, 58, 60, 66] (0.814142678347935 2602)
[7, 29, 36, 40, 48, 52, 58, 60, 62] (0.8366708385481852 2674)
[7, 34, 36, 40, 48, 52, 58, 60, 66] (0.804755944931164 2572)
[5, 29, 34, 40, 52, 56, 58, 60, 62] (0.8110137672090113 2592)
[29, 34, 36, 40, 48, 52, 58, 60, 66] (0.8376095118898623 2677)
[7, 29, 36, 40, 48, 52, 56, 58, 60] (0.8285356695869838 2648)
[7, 29, 40, 48, 52, 58, 60, 62, 66] (0.8132040050062578 2599)
[29, 34, 36, 40, 48, 52, 56, 58, 60] (0.8191489361702128 2618)
[7, 29, 36, 40, 52, 56, 58, 60, 66] (0.8194618272841051 2619)
[7, 36, 40, 48, 52, 56, 58, 60, 62] (0.8038172715894869 2569)
[34, 36, 40, 48, 52, 58, 60, 62, 66] (0.8044430538172715 2571)
[29, 34, 36, 40, 52, 56, 58, 60, 62] (0.8225907384230288 2629)
[29, 34, 36, 40, 48, 52, 56, 58, 62] (0.8013141426783479 2561)
[25, 29, 36, 40, 48, 52, 58, 60, 66] (0.8006883604505632 2559)
[5, 29, 40, 52, 56, 58, 60, 62, 66] (0.8031914893617021 2567)
[5, 29, 36, 40, 48, 52, 58, 60, 66] (0.8204005006257822 2622)
Found 85 frequent itemsets of size 9 (with support 80.0%)
Creating itemsets of size 10 based on 85 itemsets of size 9
Created 78 unique itemsets of size 10
Passing through the data to compute the frequency of 78 itemsets of size 10
[7, 29, 36, 40, 48, 52, 58, 60, 62, 66] (0.8050688360450563 2573)
[7, 29, 36, 40, 48, 52, 56, 58, 60, 62] (0.8016270337922403 2562)
[7, 29, 34, 36, 40, 48, 52, 58, 60, 66] (0.8041301627033792 2570)
[29, 34, 36, 40, 48, 52, 58, 60, 62, 66] (0.8031914893617021 2567)
Found 4 frequent itemsets of size 10 (with support 80.0%)
Creating itemsets of size 11 based on 4 itemsets of size 10
Created 2 unique itemsets of size 11
Passing through the data to compute the frequency of 2 itemsets of size 11
Execution time is: 103.432 seconds.
Found 8227 frequents sets for support 80.0% (absolute 2557)
Done
```

Fig10. Shows the result obtained in accordance with itemsets of size 10

The minsup value can be dynamically altered as desired and the operation can be conducted accordingly.

The same database shown in Fig. 4 and Fig. 5 is given as input to the source code of FP – Growth algorithm designed in Java platform. The minsup value has been set to .80%.

Fig. 11 shows the result obtained in accordance with itemsets of size 1. The first row in the result extracted is as follows.

```
[3]<0.88829787234042562839>
```

Here [3] refers to the itemset been scanned. 0.8882978723404256 shows the support value of itemset [3] which is 88.82978723404256 % and is clearly above the minimum support value of 80% or is above minsup value of .80. Fig. 5.13 also indicates that there are 76 itemsets of size 1 out of which 19 itemsets of size 1 qualified the set condition of minimum support.



```
C:\Program Files\Java\jdk\bin>java FPGrowth2
Input configuration: 76 items, 3196 transactions,
minsup = 0.8%
Passing through the data to compute the frequency of 76 itemsets of size 1
[3] <0.8882978723404256 2839>
[5] <0.9295994993742178 2971>
[7] <0.9624530663329162 3076>
[9] <0.8992490613266583 2874>
[25] <0.8948685857321652 2860>
[29] <0.9953066332916145 3181>
[34] <0.951188986232791 3040>
[36] <0.9696495619524406 3099>
[40] <0.9918648310387985 3170>
[42] <0.8491864831038799 2714>
[44] <0.8172715894868585 2612>
[48] <0.9427409261576971 3013>
[52] <0.996558197747184 3185>
[56] <0.945244055068836 3021>
[58] <0.9996871088861077 3195>
[60] <0.9852941176470589 3149>
[62] <0.9574468085106383 3060>
[64] <0.8232165206508135 2631>
[66] <0.945244055068836 3021>
Found 19 frequent itemsets of size 1 (with support 80.0%)
```

Fig11. Shows the result obtained in accordance with itemsets of size 1

Fig. 12 shows the result obtained in accordance with itemsets of size 2. The first row in the result extracted is as follows.

[29, 44] <0.812578222777847312597>

Here [29, 44] refers to the itemset been scanned.

0.81257822277784731 shows the support value of itemset [29, 44] which is 81.257822277784731 % and is clearly above the minimum support value of 80% or is above minsup value of .80. Fig. 12 also indicates 141 itemsets of size 2 qualified the set condition of minimum support i.e. equal to or greater than minsup. The itemset [29, 44] appeared frequently 2597 times in the database under study.

```
[29, 44] <0.81257822277784731 2597>
[7, 9] <0.8620150187734669 2755>
[40, 64] <0.8229036295369212 2630>
[48, 62] <0.9001877346683355 2877>
[36, 60] <0.9549436795994993 3052>
[48, 56] <0.8879849812265331 2838>
[5, 60] <0.9148936170212766 2924>
[5, 7] <0.8945556946182729 2859>
[25, 60] <0.886107634543179 2832>
[52, 60] <0.9818523153942428 3138>
[5, 62] <0.8945556946182729 2859>
[9, 52] <0.8967459324155194 2866>
[44, 60] <0.8025657071339174 2565>
[56, 62] <0.9145807259073843 2923>
[7, 36] <0.9321026282853567 2979>
[52, 56] <0.9436795994993742 3016>
[3, 66] <0.8457446808510638 2703>
[3, 52] <0.8854818523153942 2830>
[60, 66] <0.945244055068836 3021>
[36, 52] <0.9662077596996246 3088>
[25, 66] <0.8494993742177722 2715>
[44, 58] <0.8169586983729662 2611>
[7, 60] <0.9483729662077597 3031>
[9, 60] <0.8870463078848561 2835>
[29, 34] <0.9499374217772215 3036>
[42, 56] <0.8041301627033792 2570>
[5, 34] <0.9039424280350438 2889>
[36, 42] <0.8216520650813517 2626>
[7, 56] <0.9133291614518148 2919>
[25, 56] <0.8438673341677096 2697>
[3, 48] <0.8360450563204005 2672>
[42, 62] <0.8128911138923655 2598>
[34, 62] <0.9114518147680406 2913>
[29, 62] <0.9527534418022529 3045>
[40, 66] <0.9377346683354193 2997>
[40, 48] <0.9346057571964956 2987>
[7, 25] <0.8582603254067585 2743>
[9, 58] <0.898936170212766 2873>
[29, 56] <0.9405506883604505 3006>
[9, 48] <0.8419899874843555 2691>
[29, 42] <0.8454317897371715 2702>
[48, 60] <0.9342928660826032 2986>
[36, 62] <0.9270963704630788 2963>
[58, 66] <0.9449311639549437 3020>
[34, 42] <0.814142678347935 2602>
[9, 40] <0.8923654568210263 2852>
[48, 58] <0.9424280350438048 3012>
[52, 62] <0.9540050062578223 3049>
[5, 9] <0.8394868585732165 2683>
[34, 56] <0.9023779724655819 2884>
Found 141 frequent itemsets of size 2 (with support 80.0%)
```

Fig12. Shows the result obtained in accordance with itemsets of size 2



Fig. 13 shows the result obtained in accordance with itemsets of size 3. The first row in the result extracted is as follows.

[48, 52, 66] <0.84599436795995 2862>

Here [48, 52, 66] refers to the itemset been scanned.

0.84599436795995 shows the support value of itemset [48, 52, 66] which is 84.599436795995 % and is clearly above the minimum support value of 80% or is above minsup value of .80. Fig. 5.15 also indicates 566 itemsets of size 3 qualified the set condition of minimum support i.e. equal to or greater than minsup. The itemset [48, 52, 66] appeared frequently 2862 times in the database under study.

```
[48, 52, 66] (0.84599436795995 2862)
[7, 58, 66] (0.90894868858573217 2905)
[58, 60, 62] (0.9430538172715894 3014)
[58, 60, 64] (0.80819774718398 2583)
[7, 25, 34] (0.8144555694618273 2603)
[7, 40, 48] (0.8970588235294118 2867)
[42, 52, 56] (0.8025657071339174 2565)
[3, 40, 48] (0.827909887359199 2646)
[7, 48, 60] (0.8973717146433041 2868)
[3, 40, 66] (0.8382352941176471 2679)
[5, 56, 60] (0.8623279098873592 2756)
[7, 36, 62] (0.889549436795995 2843)
[5, 29, 34] (0.9026908635794744 2885)
[25, 29, 58] (0.8908010012515645 2847)
[7, 29, 52] (0.9568210262828536 3058)
[7, 40, 66] (0.9017521902377973 2882)
[9, 40, 56] (0.8541927409261577 2730)
[7, 34, 48] (0.859511889862328 2747)
[3, 7, 48] (0.8128911138923655 2598)
[40, 58, 60] (0.9771589486858573 3123)
[48, 52, 60] (0.9308510638297872 2975)
[36, 48, 66] (0.8908010012515645 2847)
[42, 52, 58] (0.8469962453066333 2707)
[5, 7, 40] (0.8879849812265331 2838)
[58, 60, 66] (0.9449311639549437 3020)
[40, 58, 62] (0.9527534418022529 3045)
[7, 9, 34] (0.8272841051314143 2644)
[7, 48, 66] (0.8629536921151439 2758)
[7, 34, 66] (0.8688986232790988 2777)
[3, 52, 60] (0.8748435544430538 2796)
[3, 34, 58] (0.8466833541927409 2706)
[5, 36, 40] (0.8926783479349186 2853)
[5, 56, 66] (0.8335419274092616 2664)
[3, 9, 29] (0.8044430538172715 2571)
[25, 40, 62] (0.8520025031289111 2723)
[5, 36, 60] (0.8845431789737171 2827)
[48, 52, 62] (0.8967459324155194 2866)
[29, 34, 48] (0.8926783479349186 2853)
[3, 7, 29] (0.8651439299123905 2765)
[25, 29, 48] (0.8563829787234043 2737)
[34, 48, 56] (0.845118898623279 2701)
[48, 56, 58] (0.8876720901126408 2837)
[40, 58, 64] (0.8225907384230288 2629)
[7, 9, 36] (0.8316645807259074 2658)
[9, 56, 58] (0.8607634543178974 2751)
[7, 40, 62] (0.9152065081351689 2925)
[9, 29, 66] (0.8482478097622027 2711)
[3, 40, 62] (0.8476220275344181 2709)
[9, 48, 58] (0.8416770963704631 2690)
[7, 25, 58] (0.8579474342928661 2742)
Found 566 frequent itemsets of size 3 (with support 80.0%)
```

Fig13. Shows the result obtained in accordance with itemsets of size 3

Similarly frequent itemsets can be obtained for size 4 to size 11.

Fig. 14 shows the result obtained in accordance with itemsets of size 10. The first row in the result extracted is as follows.

[7, 29, 36, 48, 52, 56, 58, 60, 66] <0.8025657071339174 2565>

Here [7, 29, 36, 48, 52, 56, 58, 60, 66] refers to the itemset been scanned.

0.8025657071339174 shows the support value of itemset [7, 29, 36, 48, 52, 56, 58, 60, 66] which is 80.25657071339174 % and is clearly above the minimum support value of 80% or is above minsup value of .80. Fig. 14 also indicates 78 itemsets of size 10 qualified the set condition of minimum support i.e. equal to or greater than minsup. The itemset [7, 29, 36, 48, 52, 56, 58, 60, 66] appeared frequently 2565 times in the database under study.

Fig. 14 also shows that 2 itemsets of size 11 also qualify the set condition of minimum support.

```
[7, 29, 36, 48, 52, 56, 58, 60, 66] <0.8025657071339174 2565>
[5, 29, 36, 40, 52, 56, 58, 60, 62] <0.8000625782227785 2557>
[7, 29, 36, 40, 52, 58, 60, 62, 66] <0.8291614518147684 2650>
[7, 29, 36, 40, 48, 52, 58, 62, 66] <0.8050688360450563 2573>
[29, 40, 48, 52, 56, 58, 60, 62, 66] <0.8072590738423029 2580>
[5, 29, 34, 40, 52, 56, 58, 60, 66] <0.804755944931164 2572>
[5, 7, 29, 36, 40, 48, 52, 58, 60] <0.8103879849812266 2590>
[7, 29, 34, 36, 40, 48, 58, 60, 66] <0.8069461827284106 2579>
[29, 34, 36, 48, 52, 58, 60, 62, 66] <0.8069461827284106 2579>
[7, 29, 36, 48, 52, 56, 58, 60, 62] <0.8060075093867334 2576>
[7, 29, 36, 40, 48, 52, 60, 62, 66] <0.8050688360450563 2573>
[29, 36, 40, 48, 52, 56, 58, 60, 62] <0.8310387984981227 2656>
[5, 7, 29, 34, 40, 52, 58, 60, 66] <0.8191489361702128 2618>
[7, 29, 36, 40, 48, 58, 60, 62, 66] <0.8085106382978723 2584>
[29, 36, 48, 52, 56, 58, 60, 62, 66] <0.8035043804755945 2568>
[7, 29, 36, 40, 52, 56, 58, 60, 62] <0.8304130162703379 2654>
[29, 34, 40, 52, 56, 58, 60, 62, 66] <0.8207133917396746 2623>
[7, 29, 36, 40, 48, 56, 58, 60, 62] <0.8031914893617021 2567>
[5, 29, 36, 40, 48, 52, 58, 60, 62] <0.8119524405506884 2595>
[29, 34, 36, 40, 52, 56, 58, 60, 66] <0.814142678347935 2602>
[7, 29, 36, 40, 48, 52, 58, 60, 62] <0.8366708385481852 2674>
[7, 34, 36, 40, 48, 52, 58, 60, 66] <0.804755944931164 2572>
[5, 29, 34, 40, 52, 56, 58, 60, 62] <0.8110137672090113 2592>
[29, 34, 36, 40, 48, 52, 58, 60, 66] <0.8376095118898623 2677>
[7, 29, 36, 40, 48, 52, 56, 58, 60] <0.8285356695869838 2648>
[7, 29, 40, 48, 52, 58, 60, 62, 66] <0.8132040050062578 2599>
[29, 34, 36, 40, 48, 52, 56, 58, 60] <0.8191489361702128 2618>
[7, 29, 36, 40, 52, 56, 58, 60, 66] <0.8194618272841051 2619>
[7, 36, 40, 48, 52, 56, 58, 60, 62] <0.8038172715894869 2569>
[34, 36, 40, 48, 52, 58, 60, 62, 66] <0.8044430538172715 2571>
[29, 34, 36, 40, 52, 56, 58, 60, 62] <0.8225907384230288 2629>
[29, 34, 36, 40, 48, 52, 56, 58, 62] <0.8013141426783479 2561>
[25, 29, 36, 40, 48, 52, 58, 60, 66] <0.8006883604505632 2559>
[5, 29, 40, 52, 56, 58, 60, 62, 66] <0.8031914893617021 2567>
[5, 29, 36, 40, 48, 52, 58, 60, 66] <0.8204005006257822 2622>
Found 85 frequent itemsets of size 9 (with support 80.0%)
Creating itemsets of size 10 based on 85 itemsets of size 9
Created 78 unique itemsets of size 10
Passing through the data to compute the frequency of 78 itemsets of size 10
[7, 29, 36, 40, 48, 52, 58, 60, 62, 66] <0.8050688360450563 2573>
[7, 29, 36, 40, 48, 52, 56, 58, 60, 62] <0.8016270337922403 2562>
[7, 29, 34, 36, 40, 48, 52, 58, 60, 66] <0.8041301627033792 2570>
[29, 34, 36, 40, 48, 52, 58, 60, 62, 66] <0.8031914893617021 2567>
Found 4 frequent itemsets of size 10 (with support 80.0%)
Creating itemsets of size 11 based on 4 itemsets of size 10
Created 2 unique itemsets of size 11
Passing through the data to compute the frequency of 2 itemsets of size 11
Execution time is: 103.107 seconds.
Found 8227 frequent sets for support 80.0% (absolute 2557)
Done
C:\Program Files\Java\jdk\bin>
```

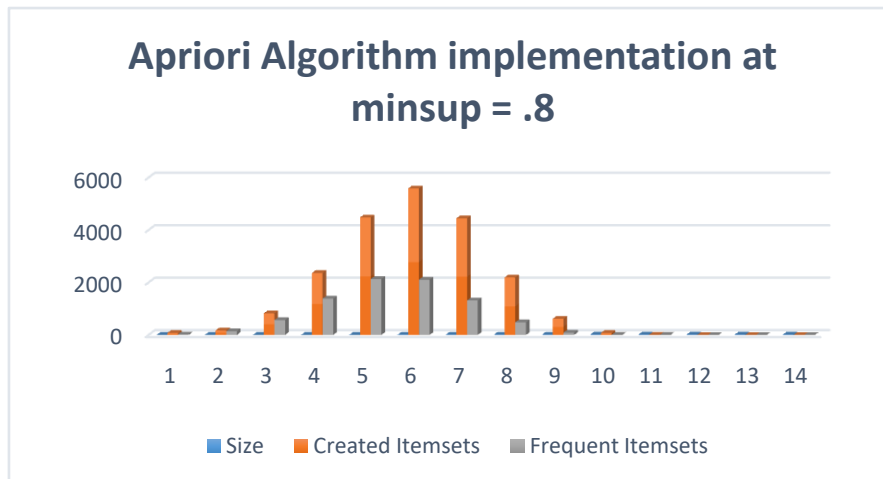
Fig14. Shows the result obtained in accordance with itemsets of size 10

The total time taken to conduct frequent mining using FP – Growth algorithm is 103.107 seconds (103107milliseconds).

Table 7 below show the figures obtained at minimum support of 80% (minsup=0.8) via running Apriori algorithm on database under study “commondatabase.dat”.

Table7. Displays the figures obtained on running Apriori algorithm at minsup= 0.8

Apriori algorithm at minsup=0.8		
Size	Created Itemsets	Frequent Itemsets
1	76	19
2	171	141
3	821	566
4	2360	1383
5	4478	2130
6	5583	2104
7	4445	1314
8	2189	481
9	617	85
10	78	4
11	4	2
12	0	0
13	0	0
14	0	0

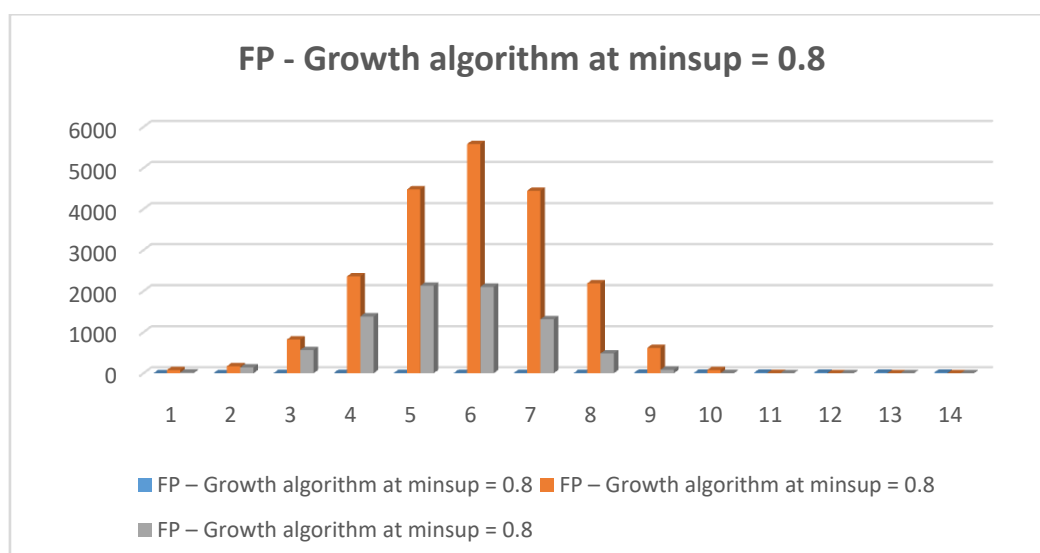


**Fig15.** Figure displays the figures obtained on running Apriori algorithm at minsup= 0.8

Table 8 below show the figures obtained at minimum support of 80% (minsup=0.8) via running FP - Growth algorithm on database under study “commondatabase.dat”.

**Table8.** Displays the figures obtained on running Apriori algorithm at minsup= 0.8

FP-Growth algorithm at minsup=0.8		
Size	Created Itemsets	Frequent Itemsets
1	76	19
2	171	141
3	821	566
4	2360	1383
5	4478	2130
6	5583	2104
7	4445	1314
8	2189	481
9	617	85
10	78	4
11	4	2
12	0	0
13	0	0
14	0	0



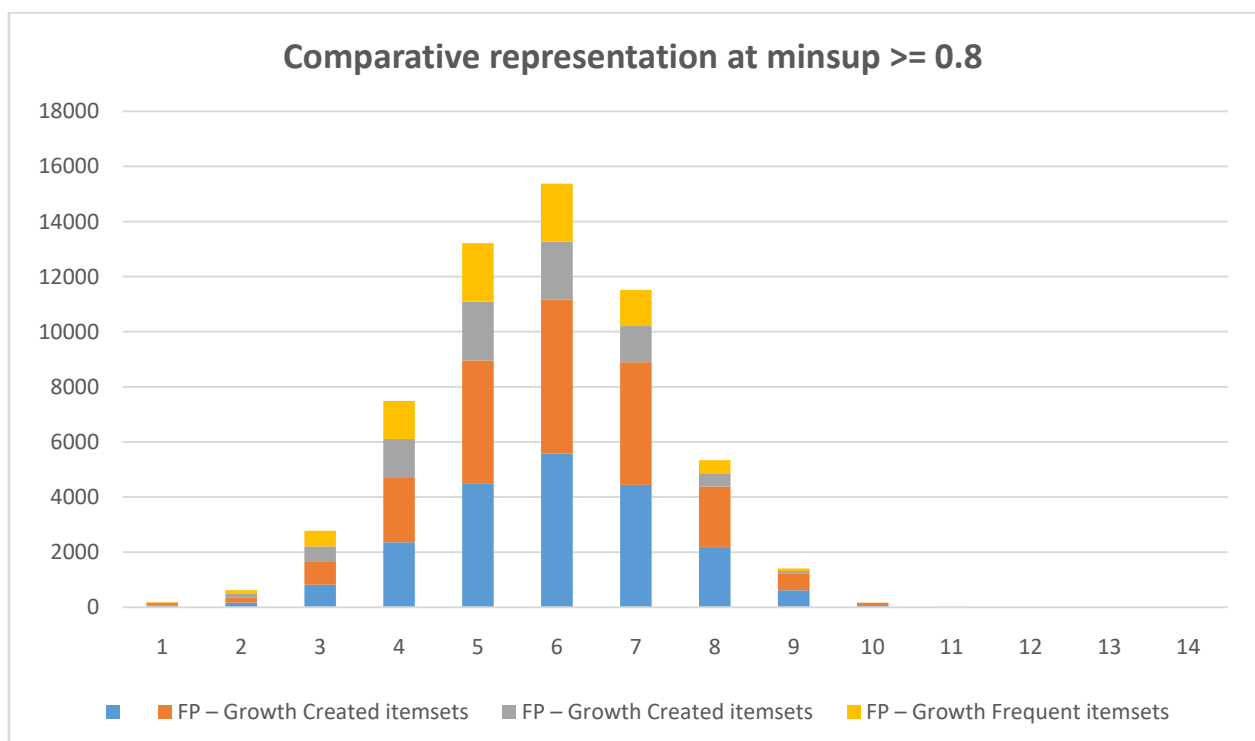
**Fig16.** Figure displays the figures obtained on running Apriori algorithm at minsup= 0.8

Table 9 shows the created itemsets and frequent itemsets values obtained at different sizes by both the algorithms.

**Table9.** Comparative table at minsup >= 0.8

Comparative table at minsup>=0.8 (Apriori algorithm Vs. FP-Growth algorithm)				
Size	Apriori Created itemsets	Apriori Frequent Itemsets	FP-Growth Created itemsets	Fp-Growth Frequent Itemsets
1	76	19	76	19
2	171	141	171	141
3	821	566	821	566
4	2360	1383	2360	1383
5	4478	2130	4478	2130
6	5583	2104	5583	2104
7	4445	1314	4445	1314
8	2189	481	2189	481
9	617	85	617	85
10	78	4	78	4
11	4	2	4	2
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0

Figure 17 shows the graphical representation of created itemsets and frequent itemsets values obtained at different sizes by both the algorithms.



**Fig17.** Shows the graphical representation of created itemsets and frequent itemsets values at different sizes of both the algorithms

#### 4. PERFORMANCE EVALUATION OF APRIORI ALGORITHM AND HADOOP BASED MAPREDUCE ALGORITHM

Among the two data mining algorithms, Apriori algorithm and FP – Growth algorithm, the Apriori algorithm dominates in performance when evaluated in terms of time taken. This section of the research paper compares the efficiency of Apriori algorithm with the MapReduce algorithm in terms of time consumed. The source code for Apriori algorithm has been constructed using C language. A small excel file titled “numbers.csv” shown in Fig. 18 is provided as input to both the algorithms.

	A	B	C	D	E	F
1	<b>A1</b>	<b>A2</b>	<b>A3</b>	<b>A4</b>	<b>A5</b>	<b>Sum</b>
2	1	5	2	0	0	8
3	2	3	0	1	0	6
4	3	4	0	0	0	7
5	2	1	3	0	0	6
6	1	2	3	0	0	6

Fig18. Input data file “numbers.csv”

The comparative graph of Apriori and Hadoop is shown below in Fig. 19. The result shows that MapReduce algorithm is much more speedy and efficient in mining as compared to Apriori algorithm.

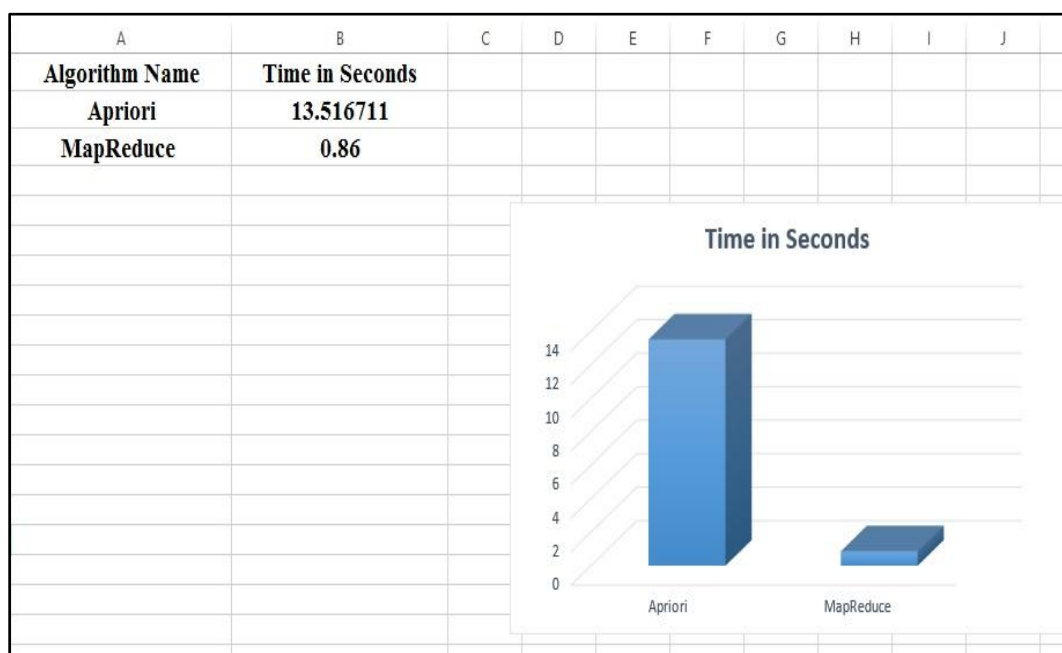


Fig19. Comparison of Apriori and MapReduce algorithm in terms of time in seconds

So, it can be concluded that the working of MapReduce algorithm is much better than Apriori algorithm.

## 5. CONCLUSION

The research work conducted has proved that among the two conventional data mining algorithms, Apriori algorithm and FP – Growth algorithm, the performance of Apriori algorithm is much better than FP – Growth algorithm when we talk about efficiency in terms of time taken. The test has been conducted on three minimum support values of 80%. The Apriori algorithm has proved its worth upon FP – Growth algorithm as evaluated and proved in section 3.

Thereafter, the comparison of Apriori algorithm is done with MapReduce algorithm to conduct the performance evaluation of both the algorithms. It is proved in the evaluation that MapReduce algorithm takes much less time in completing the operation as compared to Apriori algorithm.

Therefore, it can be concluded that out of three algorithms under study in this research paper, the MapReduce turns out to be the best in efficiency in terms to completing any particular operation relevant to mining.



## REFERENCES

- [1] Jagdev, G., Kaur, A.: Comparing Conventional Data Mining Algorithms with Hadoop based Map-Reduce Algorithm considering elections perspective. *International Journal of Innovative Research in Science and Engineering (IJIRSE)* 3(3), 57-68 (2017).
- [2] Basics of MapReduce Algorithm Explained with a Simple Example, <http://www.thegeekstuff.com/2014/05/Map-Reduce-algorithm/>. last accessed: 08/11/2018.
- [3] Jagdev, G., Kaur, S.: Analyzing Maneuver of Hadoop Framework and MapR Algorithm Proficient in supervising Big Data. *International Journal of Advanced Technology in Engineering and Science (IJATES)* 5(5), 505-515 (2017).
- [4] Tao, Y., Lin, W., Xiao, X.: Minimal MapReduce Algorithms. In: *Proceedings of SIGMOD'13*, New York (2013).
- [5] Gandomi, A., Haider, M.: Beyond the Hype: Big Data Concepts, methods and analytics. *International Journal of Information Management* 35 (2), 137-144 (2015).
- [6] Jagdev, G., Kaur, A., Kaur, A.: Excavating Big Data associated to Indian election scenario via Apache Hadoop. *International Journal of Advanced Research in Computer Science* 7 (6), 117-123 (2016).
- [7] Amanvir Kaur & Dr. Gagandeep Jagdev (2017). Analyzing Working of FP-Growth Algorithm for Frequent Pattern Mining, *International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)*, 4(4), pp.22-30, DOI: <http://dx.doi.org/10.20431/2349-4859.0404003>.
- [8] Faisal Mohammed Nafie Ali & Abdelmoneim Ali Mohamed Hamed; “Usage Apriori and clustering algorithms in WEKA tools to mining dataset of traffic accidents”, *Journal of Information and Telecommunication*, Taylor & Francis, 2018, VOL. 2, NO. 3, 231–245.
- [9] Dr. Gagandeep Jagdev et al., “A Comparative study of Conventional Data Mining Algorithms against Map-Reduce Algorithm”, in *International Journal of Advance Research in Science and Engineering (IJARSE)*, ISSN (O) – 2319-8354, ISSN (P) – 2319-8346, Volume – 06, Issue – 05, May 2017.
- [10] Salman Ahmed G, Sweta Bhattacharya; “Mining on Big Data Using Hadoop MapReduce Model”, *IOP Conf. Series: Materials Science and Engineering* 263 (2017) 042007 doi:10.1088/1757-899X/263/4/042007, pp. 1 – 12.
- [11] Dr. Gagandeep Jagdev et al., “A Study of Clustering and Classification Techniques involved in Data Mining”, in *International Journal of Advanced Technology in Engineering and Science (IJATES)*, ISSN – 2348-7550, Volume – 05, Issue – 05, May 2017.
- [12] Yutang Liu<sup>1</sup>, Qin Zhang, “Research on Association Rules Mining Algorithm Based on Large Data”, *Revista de la Facultad de Ingeniería U.C.V.*, Vol. 32, N°8, pp. 229-236, 2017.
- [13] JayshreeJha and Leena Ragha, “Educational Data Mining using Improved Apriori Algorithm”, *International Journal of Information and Computation Technology*. ISSN 0974-2239 Volume 3, Number 5 (2013), pp. 411-418.
- [14] Jiao Yabing, “Research of an Improved Apriori Algorithm in Data Mining Association Rules”, *International Journal of Computer and Communication Engineering*, Vol. 2, No. 1, January 2013.

**Citation:** Kaur, M & prof.(Dr).Vishal Goyal(2018). *Relative Competence Centered Scrutiny and Implementation of Apriori, FP – Growth and Mapreduce Algorithms. International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)*, 5(4), pp.49-68. DOI: <http://dx.doi.org/10.20431/2349-4859.0504007>

**Copyright:** © 2018 Authors. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.