UC RIVERSIDE UNIVERSITY OF CALIFORNIA, RIVERSIDE

Department of Computer Science and Engineering

Course Project
CS 236: Database Management Systems

Spring 2023

Team Members

Name	SID	E-mail id
Srushti Vikas Hembade	862395839	shemb001@ucr.edu
Aparna Mohan	862396080	amoha121@ucr.edu

Table of Contents

1. Introduction to the problem statement

The goal of our project is to find out which is the most profitable month by analyzing the revenue of the hotel in the years 2015-2018. Two datasets are considered for implementing this problem, they are initially loaded in the Hadoop File Distributed System and join operation is performed over it. Using MapReduce functions the popular month in a year of bookings is also found. Furthermore, we also explored the parallel execution of the MapReduce jobs to achieve faster execution time and reduce performance overhead. We also implemented functions to find the most popular payment method for each month. This analysis will provide useful insights into booking trends and income patterns, allowing for strategic decision-making and hotel operating optimization. The process of finding out the most profitable month by merging the two datasets exhibits the significant role of Database Management Systems in hotel management.

Extra credits:

1. **Achieved faster execution time and removed overheads.**

2. Contribution

Name	**Contribution**
Srushti Vikas Hembade	-TotalRevenue Mapper ,and Reducer functions. -Sort function for Reducer output -Report
Aparna Mohan	-TotalRevenue Driver -Implemented faster execution time and reduced overheads (Extra credit) -Improved efficiency of the jobs(Extra credit) -Report

3. Flow chart

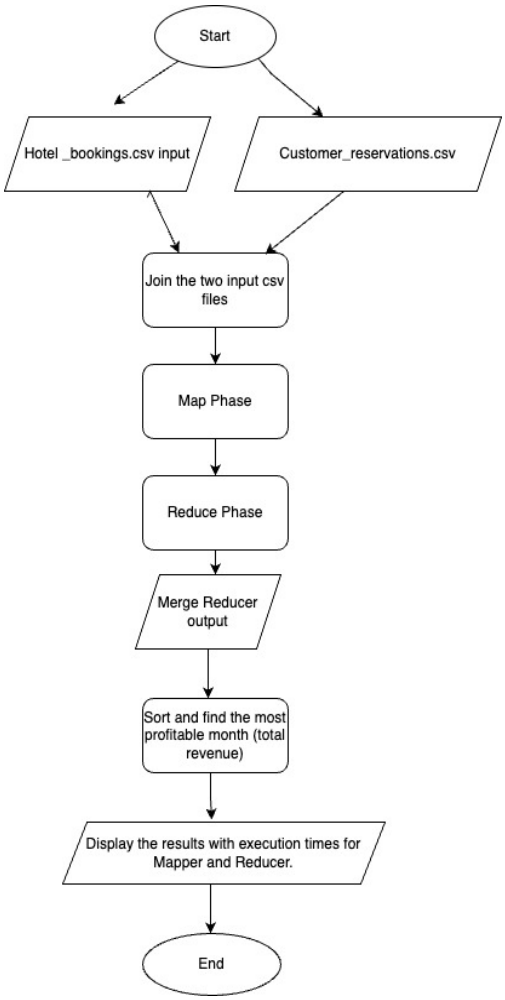


Figure 1. Basic flow chart for the project

4.1 Description of each Map Reduce

The job of Mapper in TotalRevenue

The TotalRevenue is a combination of the above implemented jobs. The job's aim is to handle hotel booking and customer reservation data and calculate total monthly revenue for various sorts of records.

1. A new condition has been added to the mapper class to handle customer reservation records. If the first field (hotel_booking_data[0]) contains "INN", the entry is a client reservation record.

2. The code checks if the booking status is "Not_Canceled" within the customer reservation record condition (bookingStatusStr.contains("Not_Canceled")). If this is the case, it will proceed with additional processing.
3. The required fields are taken from the customer reservation data (arrival month, average price per room, weekday stay, weekend stay), and the income is calculated as before.
4. To acquire the relevant month name, the arrival month is converted to an integer and supplied to the getMonthName method.
5. If the record does not have "INN" in the first field, it is classified as a hotel booking record.
6. The hotel booking record condition determines whether or not the booking status is 0 (bookingStatus == 0). If this is the case, it will proceed with additional processing.
7. Attributes like arrival month, average price per room, weekday stay, weekend stay is collected and the revenue is calculated as before.

The job of Reducer in TotalRevenue

The reducer is given key-value pairs with the key being of type Text and the value being of type DoubleWritable. It computes the total revenue by adding the values associated with each key and publishes the key and the related total revenue as the MapReduce job's output. The job can be viewed as:

1. Here, the Reducer interface overrides the reduce method, which is in charge of merging the values associated with the same key and determining the total income.
2. The value of the input key t_key is assigned to a variable key within the reduce method.
3. A variable totalRevenue is set to 0 to store the total revenue for a specific key.
4. Using the values iterator, the method iterates across the values associated with the input key.
5. The revenue is retrieved for each value using the get() method of DoubleWritable and added to the totalRevenue variable.
6. The key and the matching totalRevenue are emitted after iterating through all of the values.

4.2 Estimation of run time of the passes

From the output obtained in figure 1, we successfully obtained the run time passes for the Map Reduce jobs. With the given two datasets, there were 3 Map tasks launched along with one reduce job. From this, it can be observed that the total time for all map tasks is 9078

milliseconds, all reduce tasks take 3330 milliseconds. After improving the efficiency we were able to reduce it by 1090 milliseconds (mentioned in extra credits).

Driver class: Configurations and sort functions to sort the reducer output.

5. The chosen join and functions

The two datasets are analyzed and concatenated based on the index value. This helps for applying the MapReduce functions. Both the datasets have common attributes that are found and join operation is performed using python code as given in the snippet *dataset_combining.py*.

6. Code snippets

Mapper Class -

```
public class TotalRevenueMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {

    public void map(LongWritable key, Text value, OutputCollector<Text, DoubleWritable> output, Reporter reporter)
            throws IOException {

        String input_string = value.toString();
        String[] hotel_booking_data = input_string.split(regex:",");

        if (hotel_booking_data[0].contains(s:"INN")) {
            System.out.printf(format:"Entered customer reservation record", hotel_booking_data[0]);
            String bookingStatusStr = hotel_booking_data[9];

            if (bookingStatusStr.contains(s:"Not_Canceled")) {
                String arrivalMonth = hotel_booking_data[5];
                double avgPricePerRoom = Double.parseDouble(hotel_booking_data[8]);

                double stay_weekdays = Double.parseDouble(hotel_booking_data[2]);
                double stay_weekends = Double.parseDouble(hotel_booking_data[1]);

                double totalRevenue = avgPricePerRoom * (stay_weekdays + stay_weekends);

                int monthValue = Integer.parseInt(arrivalMonth);
                String month = getMonthName(monthValue);

                output.collect(new Text(month), new DoubleWritable(totalRevenue));
            }
        } else {
            System.out.printf(format:"Entered hotel booking record", hotel_booking_data[0]);
            String arrivalYearStr = hotel_booking_data[3];
            String bookingStatusStr = hotel_booking_data[1];

            if (!isNumeric(arrivalYearStr) || !isNumeric(bookingStatusStr)) {
                System.err.println("Invalid numeric value: arrivalYearStr = " + arrivalYearStr + ", bookingStatusStr = "
                        + bookingStatusStr);
                return;
            }
```

TotalRevenueMapper.java > TotalRevenueMapper > map(LongWritable, Text, OutputCollector<Text, DoubleWritable>, Reporter)

```
            // double arrivalYear = Double.parseDouble(arrivalYearStr);
            double bookingStatus = Double.parseDouble(bookingStatusStr);

            if (bookingStatus == 0) {
                String arrivalMonth = hotel_booking_data[4];
                double avgPricePerRoom = Double.parseDouble(hotel_booking_data[11]);

                double stay_weekdays = Double.parseDouble(hotel_booking_data[7]);
                double stay_weekends = Double.parseDouble(hotel_booking_data[8]);

                double totalRevenue = avgPricePerRoom * (stay_weekdays + stay_weekends);

                output.collect(new Text(arrivalMonth), new DoubleWritable(totalRevenue));
            }
        }
    }

    private boolean isNumeric(String str) {
        return str.matches(regex:"-?\\d+(\\.\\d+)?");
    }

    private String getMonthName(int monthValue) {
        String[] months = { "January", "February", "March", "April", "May", "June", "July", "August", "September",
                "October", "November", "December" };
        int array_index = monthValue - 1;

        if (array_index >= 0 && array_index < months.length) {
            return months[array_index];
        } else {
            return "Invalid Month";
        }
    }
}
```

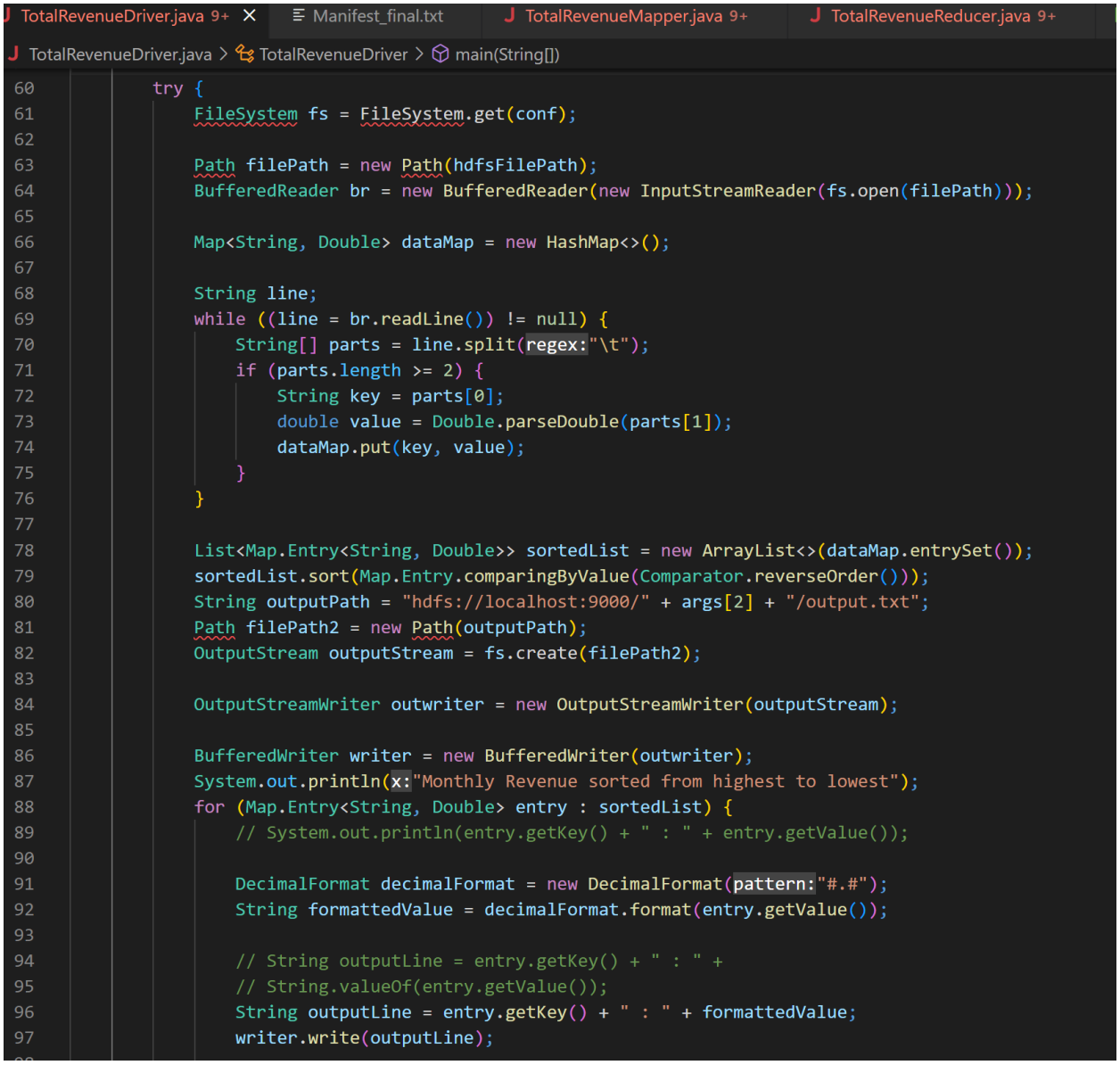
Reducer Class -

TotalRevenueReducer.java

```
package TotalRevenue;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.WritableComparator;

//import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class TotalRevenueReducer extends MapReduceBase implements Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    public void reduce(Text t_key, Iterator<DoubleWritable> values, OutputCollector<Text, DoubleWritable> output,
            Reporter reporter) throws IOException {
        Text key = t_key;
        double totalRevenue = 0;
        while (values.hasNext()) {
            totalRevenue += values.next().get();
        }
        output.collect(key, new DoubleWritable(totalRevenue));
    }
}
```

Driver Class -

```java
public class TotalRevenueDriver {
    public static class Config extends org.apache.hadoop.conf.Configuration {
    }

    Run | Debug
    public static void main(String[] args) {
        JobClient client_tr = new JobClient();

        JobConf job_conf_tr = new JobConf(TotalRevenueDriver.class);

        job_conf_tr.setJobName("TotalRevenue");

        job_conf_tr.setOutputKeyClass(Text.class);
        job_conf_tr.setOutputValueClass(DoubleWritable.class);

        job_conf_tr.setMapperClass(TotalRevenue.TotalRevenueMapper.class);
        job_conf_tr.setReducerClass(TotalRevenue.TotalRevenueReducer.class);

        job_conf_tr.setInputFormat(TextInputFormat.class);
        job_conf_tr.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(job_conf_tr, new Path(args[0]), new Path(args[1]));
        FileOutputFormat.setOutputPath(job_conf_tr, new Path(args[2]));

        client_tr.setConf(job_conf_tr);
        try {
            JobClient.runJob(job_conf_tr);
        } catch (Exception e) {
            e.printStackTrace();
        }

        Config conf = new Config();
        String hdfsFilePath = "hdfs://localhost:9000/" + args[2] + "/part-00000";
```



7. Extra credits

Improved efficiency and reduced the execution time of the TotalRevenue Mapper class.

Description

Here, we have reduced unnecessary string operations in the TotalRevenue Mapper class. The approach taken can be broken down as:

1. outputKey and outputValue are member variables of the kinds Text and DoubleWritable, respectively.
2. Instead of creating new instances for each output record, these variables are reused. This decreases the overhead of object creation and memory allocation, which results in faster execution.

3. In addition, for debugging purposes, we replaced the System.out.printf statements with simple System.out.println statements. This eliminates the formatting overhead involved with printf and speeds up execution.
4. The result obtained shows that the execution time for Mapper class to execute has reduced significantly by 1090 milliseconds.

```java
public class TotalRevenueMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {

    private Text outputKey = new Text();
    private DoubleWritable outputValue = new DoubleWritable();

    public void map(LongWritable key, Text value, OutputCollector<Text, DoubleWritable> output, Reporter reporter)
            throws IOException {

        String inputString = value.toString();
        String[] hotelBookingData = inputString.split(regex:",");

        if (hotelBookingData[0].contains(s:"INN")) {
            System.out.println("Entered customer reservation record: " + hotelBookingData[0]);
            String bookingStatusStr = hotelBookingData[9];

            if (bookingStatusStr.contains(s:"Not_Canceled")) {
                String arrivalMonth = hotelBookingData[5];
                double avgPricePerRoom = Double.parseDouble(hotelBookingData[8]);

                double stayWeekdays = Double.parseDouble(hotelBookingData[2]);
                double stayWeekends = Double.parseDouble(hotelBookingData[1]);

                double totalRevenue = avgPricePerRoom * (stayWeekdays + stayWeekends);

                int monthValue = Integer.parseInt(arrivalMonth);
                String month = getMonthName(monthValue);

                outputKey.set(month);
                outputValue.set(totalRevenue);
                output.collect(outputKey, outputValue);
            }
        } else {
            System.out.println("Entered hotel booking record: " + hotelBookingData[0]);
            String arrivalYearStr = hotelBookingData[3];
            String bookingStatusStr = hotelBookingData[1];

            if (!isNumeric(arrivalYearStr) || !isNumeric(bookingStatusStr)) {
                System.err.println("Invalid numeric value: arrivalYearStr = " + arrivalYearStr + ", bookingStatusStr = "
                        + bookingStatusStr);
                return;
            }
```

```
            double bookingStatus = Double.parseDouble(bookingStatusStr);

            if (bookingStatus == 0) {
                String arrivalMonth = hotelBookingData[4];
                double avgPricePerRoom = Double.parseDouble(hotelBookingData[11]);

                double stayWeekdays = Double.parseDouble(hotelBookingData[7]);
                double stayWeekends = Double.parseDouble(hotelBookingData[8]);

                double totalRevenue = avgPricePerRoom * (stayWeekdays + stayWeekends);

                outputKey.set(arrivalMonth);
                outputValue.set(totalRevenue);
                output.collect(outputKey, outputValue);
            }
        }
    }

    private boolean isNumeric(String str) {
        return str.matches(regex:"-?\\d+(\\.\\d+)?");
    }

    private String getMonthName(int monthValue) {
        String[] months = { "January", "February", "March", "April", "May", "June", "July", "August", "September",
                "October", "November", "December" };
        int arrayIndex = monthValue - 1;

        if (arrayIndex >= 0 && arrayIndex < months.length) {
            return months[arrayIndex];
        } else {
            return "Invalid Month";
        }
    }
}
```

8. Results and Conclusion

Hence, we were able to execute the Map Reduce jobs successfully for the hotel datasets and improve the execution time. We also observed that improving the execution time of the Total Revenue Mapper class, a significant amount of overhead was reduced. Figure 2 as explained in section 4.2, gives the estimated execution time for the Map reduce tasks. The file system counters which have details like the read operations, no of bytes read and written is also obtained. Map reduce framework further displays the mapped and reduced input/output records, combined input and output records.From figures 3, 4 and 5, it can be concluded that the highest revenue for the hotel was in the month of August with the hotel's revenue summing upto 3704215.7. The total revenue for each month is also sorted in the descending order in Figure 4. The second highest revenue was observed to be in the month of September with a revenue of 3293858.7

```
C:\CS236_MapReduce_Project>C:\hadoop-2.8.0\bin\hadoop jar TotalRevenue.jar /input/hotel-booking.csv /input/customer-reservations.csv /merged_revenue_months16
23/06/11 01:43:40 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/06/11 01:43:40 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/06/11 01:43:40 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your applicat
his.
23/06/11 01:43:41 INFO mapred.FileInputFormat: Total input files to process : 2
23/06/11 01:43:41 INFO mapreduce.JobSubmitter: number of splits:3
23/06/11 01:43:41 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1686465873787_0014
23/06/11 01:43:41 INFO impl.YarnClientImpl: Submitted application application_1686465873787_0014
23/06/11 01:43:41 INFO mapreduce.Job: The url to track the job: http://Artisan:8088/proxy/application_1686465873787_0014/
23/06/11 01:43:41 INFO mapreduce.Job: Running job: job_1686465873787_0014
23/06/11 01:43:47 INFO mapreduce.Job: Job job_1686465873787_0014 running in uber mode : false
23/06/11 01:43:47 INFO mapreduce.Job:  map 0% reduce 0%
23/06/11 01:43:54 INFO mapreduce.Job:  map 67% reduce 0%
23/06/11 01:43:55 INFO mapreduce.Job:  map 100% reduce 0%
23/06/11 01:44:00 INFO mapreduce.Job:  map 100% reduce 100%
23/06/11 01:44:00 INFO mapreduce.Job: Job job_1686465873787_0014 completed successfully
23/06/11 01:44:00 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=1296777
                FILE: Number of bytes written=3143031
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=8168965
                HDFS: Number of bytes written=305
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=3
                Launched reduce tasks=1
                Rack-local map tasks=3
                Total time spent by all maps in occupied slots (ms)=9078
                Total time spent by all reduces in occupied slots (ms)=3330
                Total time spent by all map tasks (ms)=9078
                Total time spent by all reduce tasks (ms)=3330
                Total vcore-milliseconds taken by all map tasks=9078
                Total vcore-milliseconds taken by all reduce tasks=3330
                Total megabyte-milliseconds taken by all map tasks=9295872
                Total megabyte-milliseconds taken by all reduce tasks=3409920
        Map-Reduce Framework
                Map input records=114980
                Map output records=74614
                Map output bytes=1147543
                Map output materialized bytes=1296789
                Input split bytes=299
                Combine input records=0
                Combine output records=0
                Reduce input groups=12
```

Figure 2. Execution time for each pass, file system counters and Map-Reduce framework

```
C:\CS236_MapReduce_Project>C:\hadoop-2.8.0\bin\hdfs dfs -cat /merged_revenue_months15/output.txt
August : 3704215.7299999935
September : 3293858.729999994
October : 2841535.190000005
July : 2839205.389999992
December : 1833451.8700000022
June : 1695072.229999998
May : 1593307.270000002
November : 1589000.009999996
April : 1404441.4000000027
March : 1206936.7200000014
February : 764238.900000002
January : 470901.7700000003
```

Figure 3. Revenue for each month

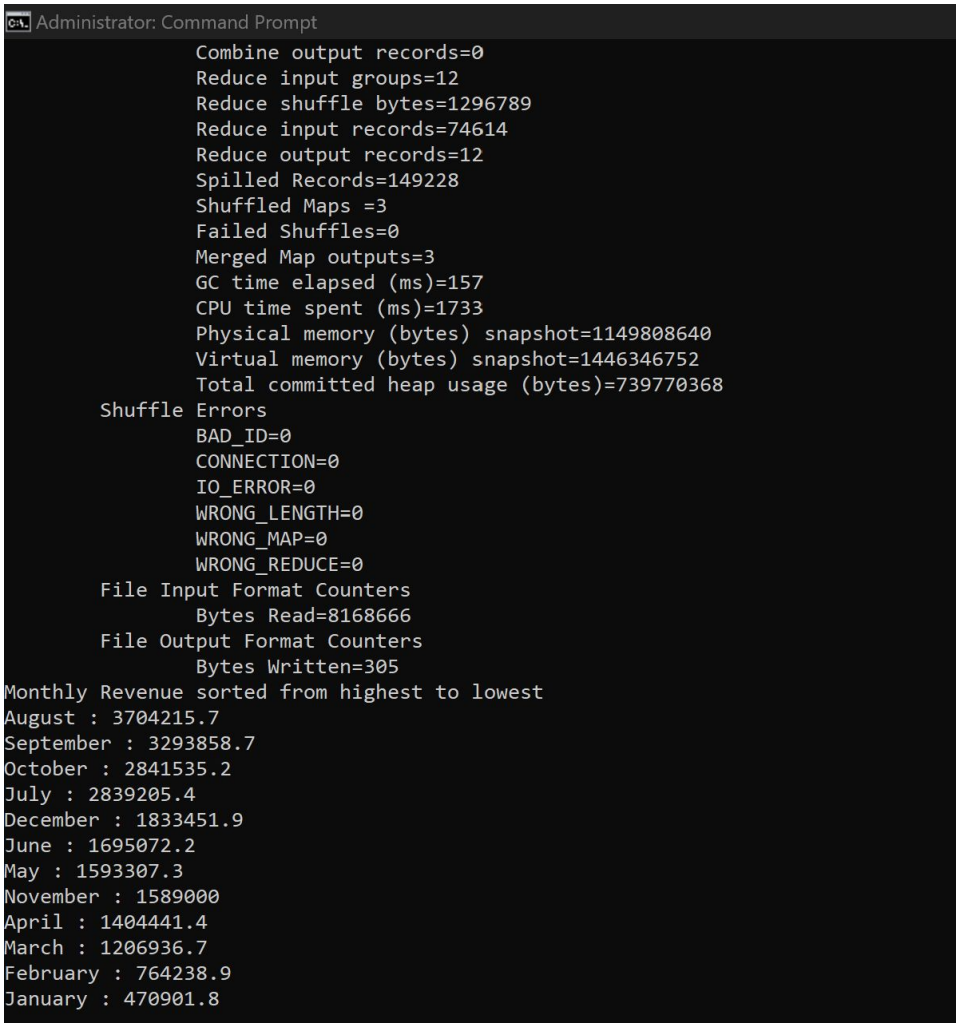


Figure 4. Sorted revenue for each month

```
C:\CS236_MapReduce_Project>C:\hadoop-2.8.0\bin\hdfs dfs -cat /merged_revenue_months19/part-00000
April   1404441.4000000027
August  3704215.7299999935
December        1833451.8700000022
February        764238.900000002
January 470901.7700000003
July    2839205.389999992
June    1695072.229999998
March   1206936.7200000014
May     1593307.270000002
November        1589000.009999996
October 2841535.190000005
September       3293858.729999994

C:\CS236_MapReduce_Project>C:\hadoop-2.8.0\bin\hdfs dfs -cat /merged_revenue_months19/output.txt
August : 3704215.7
September : 3293858.7
October : 2841535.2
July : 2839205.4
December : 1833451.9
June : 1695072.2
May : 1593307.3
November : 1589000
April : 1404441.4
March : 1206936.7
February : 764238.9
January : 470901.8

C:\CS236_MapReduce_Project>
```

Figure 5. Merged sorted revenue months output for each month

9. COMMANDS to Run (submitted zip folder):

1. Set you CLASSPATH for the Hadoop version installed in your system :
→
Set
CLASSPATH=C:\hadoop-2.8.0\share\hadoop\mapreduce\hadoop-mapreduce-client-core-2.8.0.jar;C:\hadoop-2.8.0\share\hadoop\mapreduce\hadoop-mapreduce-client-common-2.8.0.jar;C:\hadoop-2.8.0\share\hadoop\common\hadoop-common-2.8.0.jar;C:\CS236_MapReduce_Project\TotalRevenue*;C:\hadoop-2.8.0\lib*

2. Compile the java classes
→ javac -d . TotalRevenueMapper.java TotalRevenueReducer.java TotalRevenueDriver.java

3. Generate the Jar file to run on hdfs using the compiled java .class files for mapper , reducer and driver.
→ jar cfm TotalRevenue.jar Manifest_final.txt TotalRevenue/*.class

4. Start your hdfs cmds for dfs and yarn
→ C:\hadoop-2.8.0\sbin\start-all.cmd

5. Copy both the csv files in one folder named input from your local
→ C:\hadoop-2.8.0\bin\hdfs dfs -copyFromLocal input /

6. Check if the csv files were successfully uploaded to the hdfs
→ C:\hadoop-2.8.0\bin\hdfs dfs -ls /input

7. Run the jar file with hotel-booking.csv and customer-reservation.csv as input and an output folder on hdfs
→C:\hadoop-2.8.0\bin\hadoop jar TotalRevenue.jar /input/hotel-booking.csv /input/customer-reservations.csv /merged_revenue_months

8. Check for the output file generated by the reducer
→ C:\hadoop-2.8.0\bin\hdfs dfs -cat /merged_revenue_months/part-00000

9. Also can check the sorted output we write in a separate output.txt file in hdfs itself.
→ C:\hadoop-2.8.0\bin\hdfs dfs -cat /merged_revenue_months/output.txt