Project Report

Project Title: Identifying the most profitable month for hotel booking using MapReduce

Student ID:862339163
Email: hritvik7654@gmail.com
Name: Hritvik Gupta
Team Members and Contributions: Worked on project report,Overall structure and MapReduce 3.

Student ID: 862395274
Email: rsing116@ucr.edu
Name:Ritesh
Team Members and Contributions: Worked on setting up hadoop environment, MapReduce 1 and 2.

1. Introduction

In this project, our goal is to identify the most profitable month for hotel booking over the period of four years from 2015 to 2019. The given problem contains two different dataset, hotel bookings and customer reservations which contains the relevant information such as:

Booking_id,hotel,booking_status,lead_time,arrival_year,arrival_month,arrival_date_week_number,arrival_date_day_of_month,stays_in_weekend_nights,stays_in_week_nights,market_segment_type,country,avg_price_per_room,email.

To find the most profitable month. We used the month, weekday, and weekend nights, and price values were extracted from the line. Total revenue is calculated by multiplying the price by the sum of the weekday and weekend nights. The month and total revenue are then outputted as key-value pairs.This is done for both datasets and finally combines both the outputs on arrival month with corresponding monthly revenue. The last thing that is done is to sort and output the corresponding month with the highest revenue.

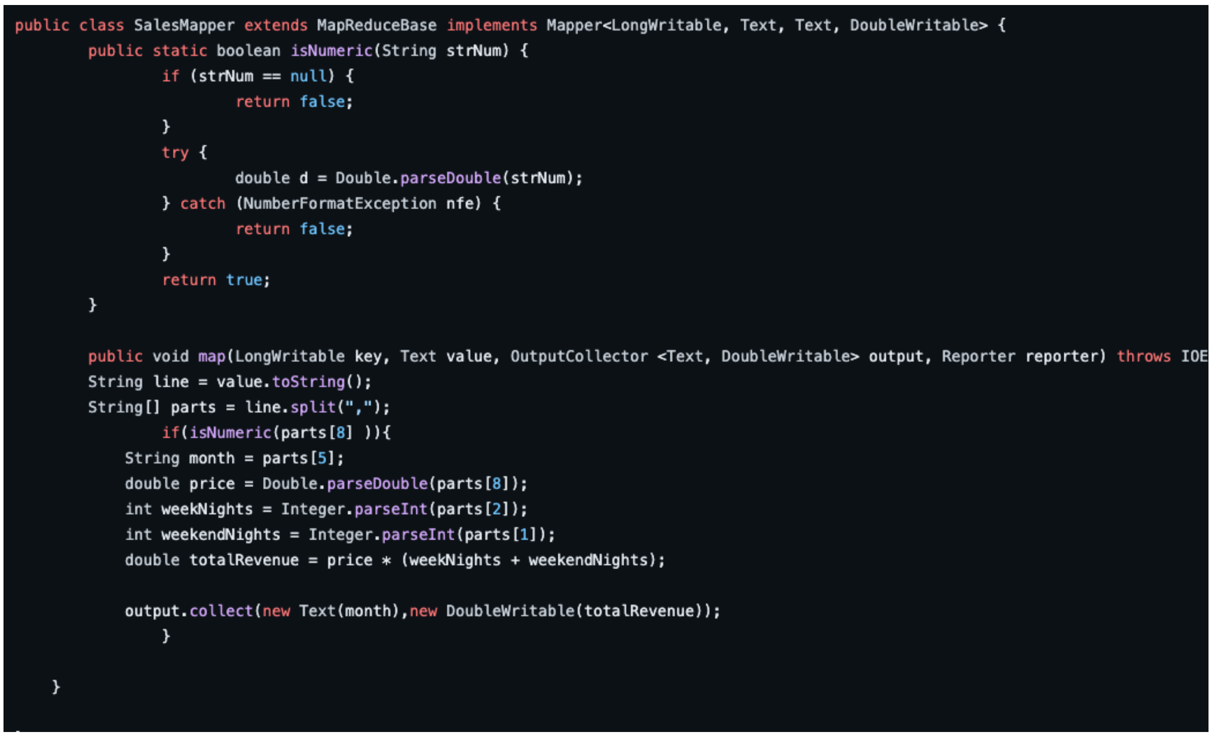
1. Methodology

The problem was divided into three separate MapReduce jobs. A single MapReduce job would have led to increased complexity and less efficiency. Each job served a distinct purpose and together they would deliver the final result.

A. MapReduce Job 1:

Mapper 1:

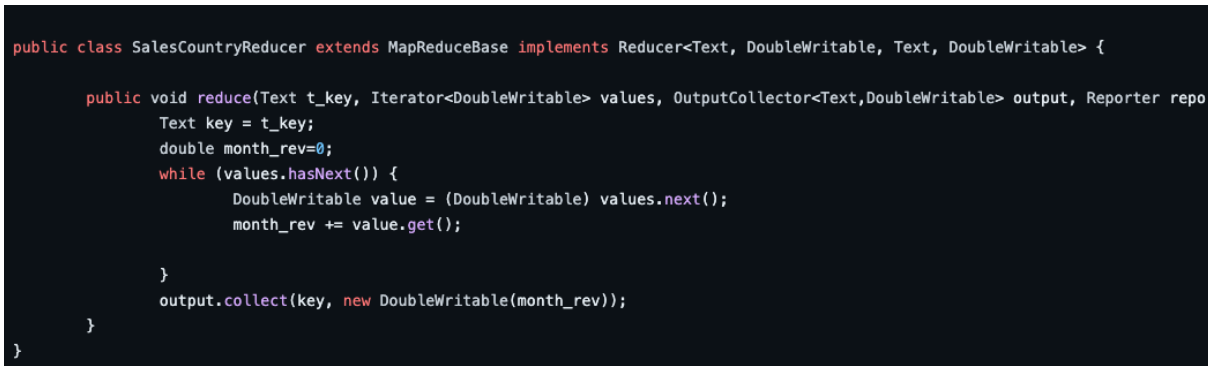
The job calculates the total revenue for each month from customer reservation dataset by calculating per booking by multiplying the price by the total nights, weekdays and weekends combined. The mapper extracts the booking date (month), number of weekday nights and weekend nights, and the price of each booking. It calculates the total revenue from each booking by multiplying the price by the total nights (weekdays and weekends combined) and emits this with the booking date as the key.



```
public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {
        public static boolean isNumeric(String strNum) {
                if (strNum == null) {
                        return false;
                }
                try {
                        double d = Double.parseDouble(strNum);
                } catch (NumberFormatException nfe) {
                        return false;
                }
                return true;
        }

        public void map(LongWritable key, Text value, OutputCollector <Text, DoubleWritable> output, Reporter reporter) throws IOE
        String line = value.toString();
        String[] parts = line.split(",");
                if(isNumeric(parts[8] )){
            String month = parts[5];
            double price = Double.parseDouble(parts[8]);
            int weekNights = Integer.parseInt(parts[2]);
            int weekendNights = Integer.parseInt(parts[1]);
            double totalRevenue = price * (weekNights + weekendNights);

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

    }
```

Reducer 1:

The reducer, receiving these, aggregates the revenues by month. Within this method, it iterates over all the revenues associated with a each month key. These revenues are then accumulated to compute the total revenue for the month.

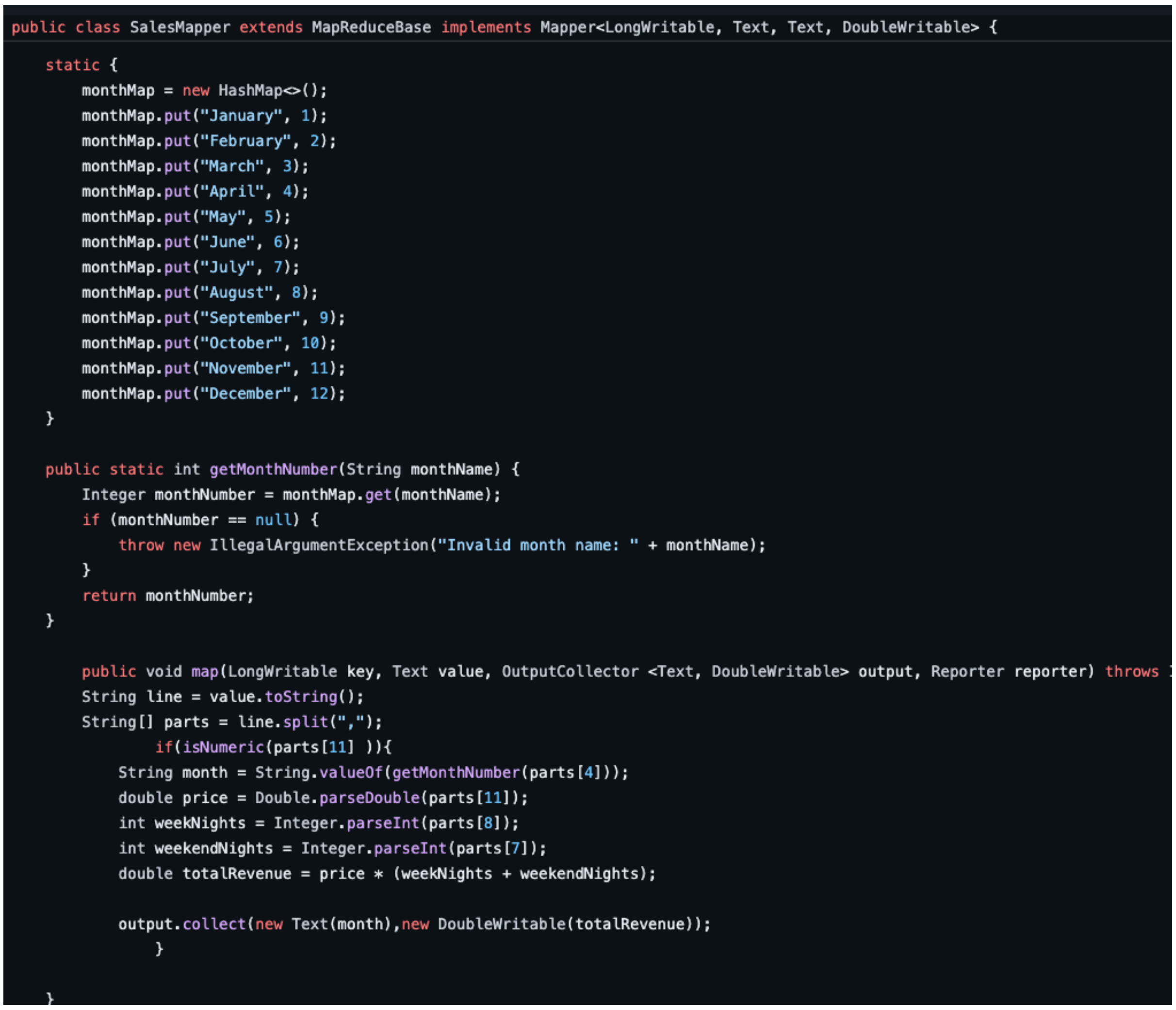


```
public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, DoubleWritable, Text, DoubleWritable> {

        public void reduce(Text t_key, Iterator<DoubleWritable> values, OutputCollector<Text,DoubleWritable> output, Reporter repo
                Text key = t_key;
                double month_rev=0;
                while (values.hasNext()) {
                        DoubleWritable value = (DoubleWritable) values.next();
                        month_rev += value.get();

                }
                output.collect(key, new DoubleWritable(month_rev));
        }
}
```

B. MapReduce Job 2:

Mapper 2:
We use the second mapper to process the second dataset which is the hotel reservation dataset, specifically targeting the extraction of month, price, weekday nights, and weekend nights data from each record. The total number of nights spent in the hotel regardless of weekdays or weekends and multiplies this by the room's

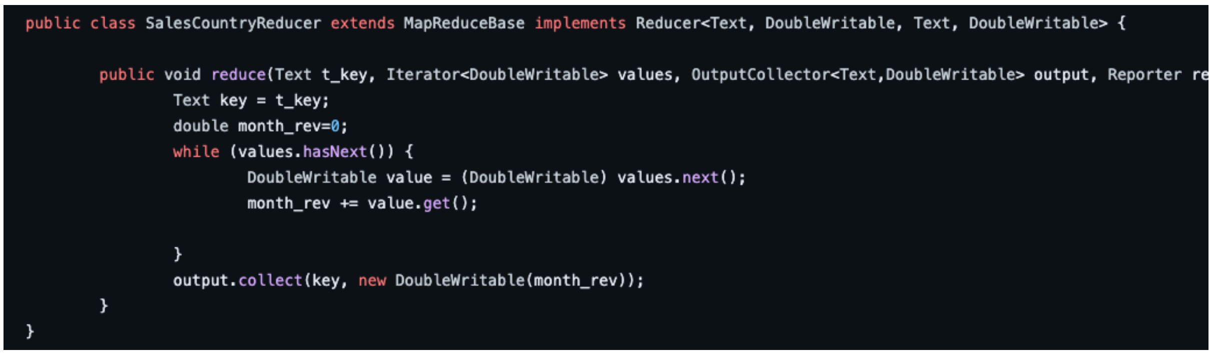
price. This gives us the total revenue from each hotel booking. This database had months in Alphabet format, so we have used a function getMonthNumber which uses a hashmap to convert it into an integer.



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

    static {
        monthMap = new HashMap<>();
        monthMap.put("January", 1);
        monthMap.put("February", 2);
        monthMap.put("March", 3);
        monthMap.put("April", 4);
        monthMap.put("May", 5);
        monthMap.put("June", 6);
        monthMap.put("July", 7);
        monthMap.put("August", 8);
        monthMap.put("September", 9);
        monthMap.put("October", 10);
        monthMap.put("November", 11);
        monthMap.put("December", 12);
    }

    public static int getMonthNumber(String monthName) {
        Integer monthNumber = monthMap.get(monthName);
        if (monthNumber == null) {
            throw new IllegalArgumentException("Invalid month name: " + monthName);
        }
        return monthNumber;
    }

        public void map(LongWritable key, Text value, OutputCollector <Text, DoubleWritable> output, Reporter reporter) throws
        String line = value.toString();
        String[] parts = line.split(",");
                if(isNumeric(parts[11] )){
            String month = String.valueOf(getMonthNumber(parts[4]));
            double price = Double.parseDouble(parts[11]);
            int weekNights = Integer.parseInt(parts[8]);
            int weekendNights = Integer.parseInt(parts[7]);
            double totalRevenue = price * (weekNights + weekendNights);

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

    }
```

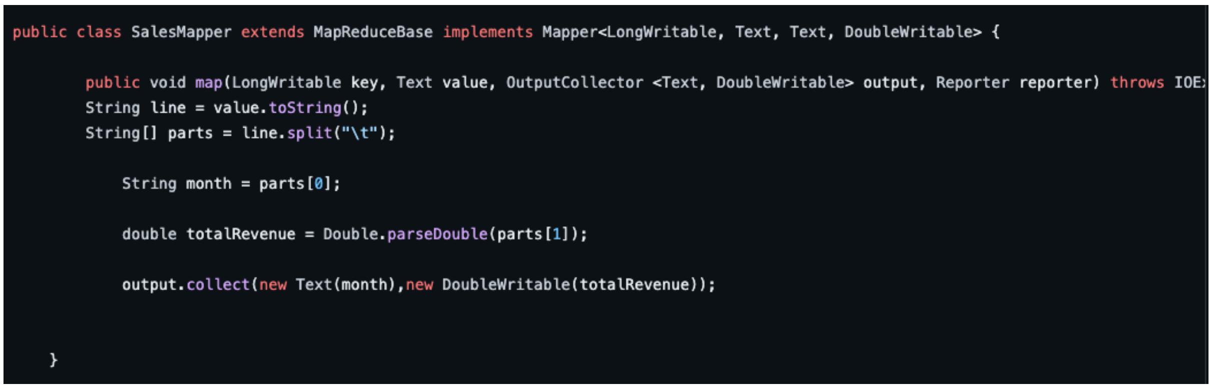
Reducer 2:

The reducer function does the task of aggregating the total monthly revenue from individual hotel bookings. This reducer takes a key-value pair from mapper where the key is the month and the value containing revenues associated with each booking. It loops through the revenue, adding them together to obtain the total revenue for each month. Finally,it outputs a key-value pair where the key is the month and the value is the total revenue for that month.

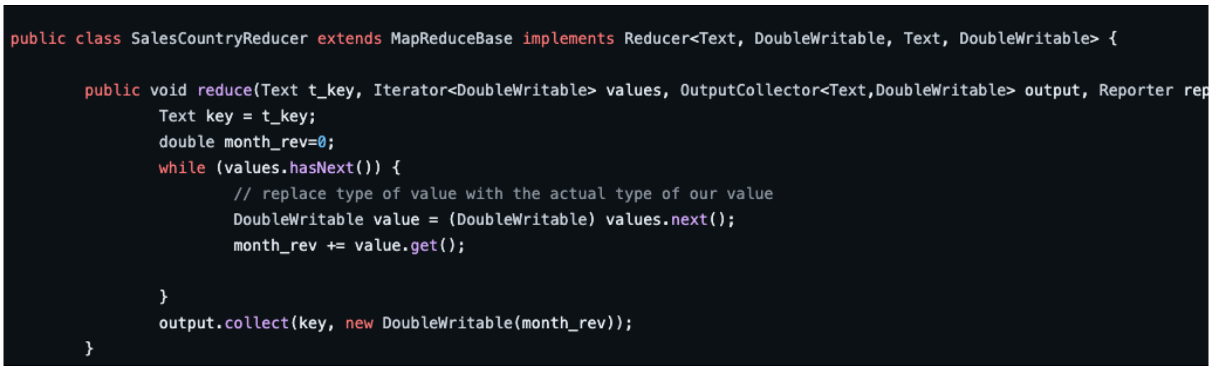

```
public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, DoubleWritable, Text, DoubleWritable> {

        public void reduce(Text t_key, Iterator<DoubleWritable> values, OutputCollector<Text,DoubleWritable> output, Reporter re
                Text key = t_key;
                double month_rev=0;
                while (values.hasNext()) {
                        DoubleWritable value = (DoubleWritable) values.next();
                        month_rev += value.get();

                }
                output.collect(key, new DoubleWritable(month_rev));
        }
}
```

C. MapReduce Job 3:

Mapper 3:
The third MapReduce job seems to aim to combine the output of the previous jobs. The month and total revenue are then extracted from their outputs and the total revenue is converted to a double data type. The function then maps these values into a new key-value pair where the key is the month and the value is the total revenue (. This pair is passed to the reducer function for the final calculation of the most profitable month.


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

        public void map(LongWritable key, Text value, OutputCollector <Text, DoubleWritable> output, Reporter reporter) throws IOE
        String line = value.toString();
        String[] parts = line.split("\t");

            String month = parts[0];

            double totalRevenue = Double.parseDouble(parts[1]);

            output.collect(new Text(month),new DoubleWritable(totalRevenue));


    }
```

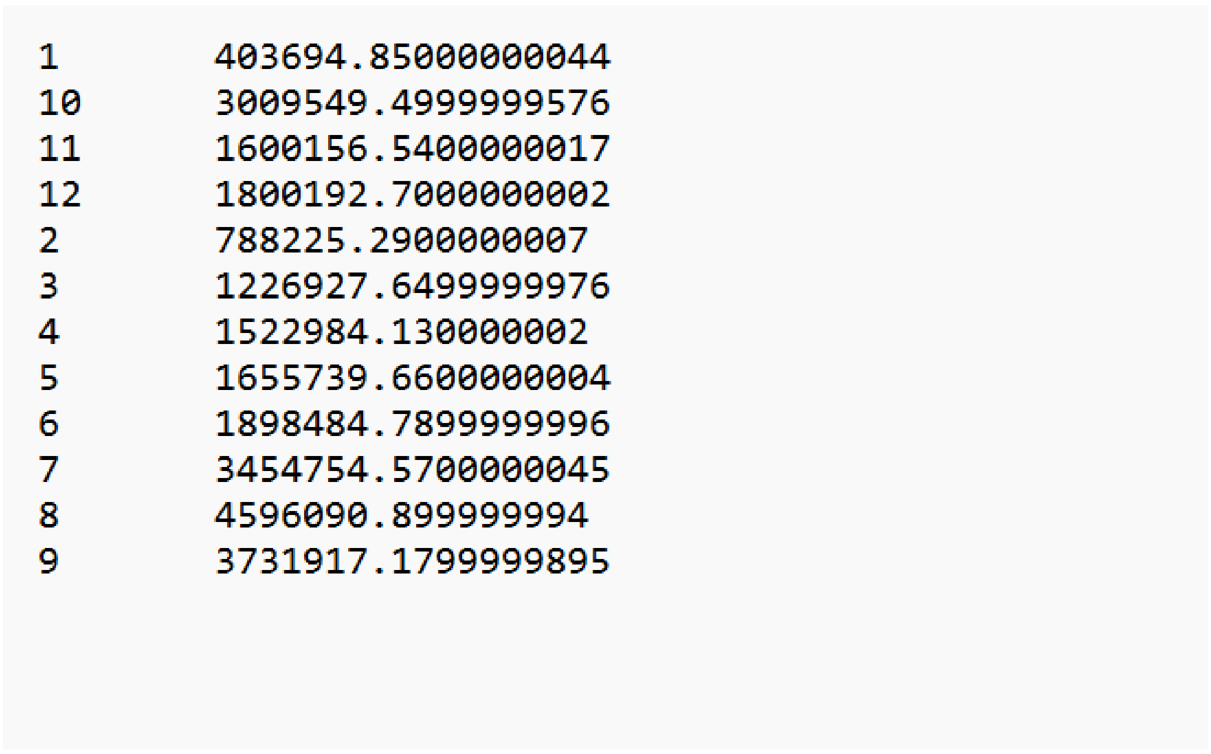
Reducer 3:
The Reducer functions by looping through all the values associated with a given key (i.e., month). It sums up these revenues to obtain the total revenue for the month. The reducer outputs a key-value pair where the key is the month and the value is the total revenue. These results represent the cumulative revenue for each month across the dataset, using which we will be able to identify the most profitable month for hotel and customer bookings.


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

        public void reduce(Text t_key, Iterator<DoubleWritable> values, OutputCollector<Text,DoubleWritable> output, Reporter rep
                Text key = t_key;
                double month_rev=0;
                while (values.hasNext()) {
                        // replace type of value with the actual type of our value
                        DoubleWritable value = (DoubleWritable) values.next();
                        month_rev += value.get();

                }
                output.collect(key, new DoubleWritable(month_rev));
        }
```

Choosing to do the join(s)

The decision to perform the join is done in a separate MapReduce job (Job 3) after the first two MapReduce jobs were primarily to extract the months and its corresponding revenue in order to keep the jobs simple and maintainable. By doing so, we can easily adjust the logic in any of these steps independently. This design also offers better scalability, as we can process large datasets in parallel in the first two jobs, and then reduce the size of the data to be processed in the third job for sorting and to reduce the time and space complexity for the problem.

We have merged the last two output from our hdfs and got a new output that contained the combined revenue for each month (for years 2015-2018).



```
1       403694.85000000044
10      3009549.4999999576
11      1600156.5400000017
12      1800192.7000000002
2       788225.2900000007
3       1226927.6499999976
4       1522984.130000002
5       1655739.6600000004
6       1898484.7899999996
7       3454754.5700000045
8       4596090.899999994
9       3731917.1799999895
```

We laster used a java class RevenueSorter to sort the file by using the second column as key for sorting. The final output can be observed below:

```
8       4596090.899999994
9       3731917.1799999895
7       3454754.5700000045
10      3009549.4999999576
6       1898484.7899999996
12      1800192.7000000002
5       1655739.6600000004
11      1600156.5400000017
4       1522984.130000002
3       1226927.6499999976
2       788225.2900000007
1       403694.85000000044
```

Based on our analysis of the revenue data over a four-year period, we have determined that the month with the highest revenue is August (8), while the month with the lowest revenue is January (1). This conclusion is based on sorting the data in ascending order according to the revenue generated in each month.

By presenting the revenue figures for all the months, not just the highest revenue month, we gain a comprehensive understanding of the revenue distribution throughout the four-year period. This format allows us to identify trends and patterns in revenue generation, providing valuable insights for decision-making and strategic planning.

Contribution:

Both of us have made significant contributions to this project, working together collaboratively to achieve the desired outcomes. Both team members actively participated in identifying and selecting the important columns for their respective datasets, as well as implementing the corresponding MapReduce classes using Hadoop.

Hritvik Gupta took the lead in analyzing the first dataset and successfully identified the crucial column that would serve as a valuable feature for the first database. With his expertise and diligent efforts, he developed a robust MapReduce class to process and extract relevant information from this dataset.

On the other hand, Ritesh Singh focused on the second dataset, conducting thorough analysis to determine the key column that would be instrumental for the second database. Ritesh skillfully implemented the MapReduce class for this dataset, ensuring the effective processing and utilization of the selected feature.

Throughout the project we maintained close collaboration, regularly engaging in discussions and planning sessions. This collaborative approach allowed us to leverage our individual strengths, exchange ideas, and address any challenges that arose during the implementation of the MapReduce jobs. Their collective efforts ensured that the project progressed smoothly and achieved the desired objectives.

Apart from this we have worked together for the installation and join problem of the project and whenever we had any issue we figured it out together.