Jaired Jawed

Jun 11, 2023

CS236 Project

Contributions

As I was the individual member of this group, I wrote all of the code and the report.

Prerequisites

- Install Java
- Install Hadoop

Usage

- Set alias to equal path to hadoop and hdfs

```
alias hadoop='/path-to-hadoop'
alias hdfs='/path-to-hdfs'
```

- Run run-project.sh, with the datasets, combined input folder, and output folder listed as arguments as shown below. The project will run all necessary files and list the month with the greatest revenue in the outputs directory.

```
sh run-project.sh ./datasets ./inputs ./outputs
```

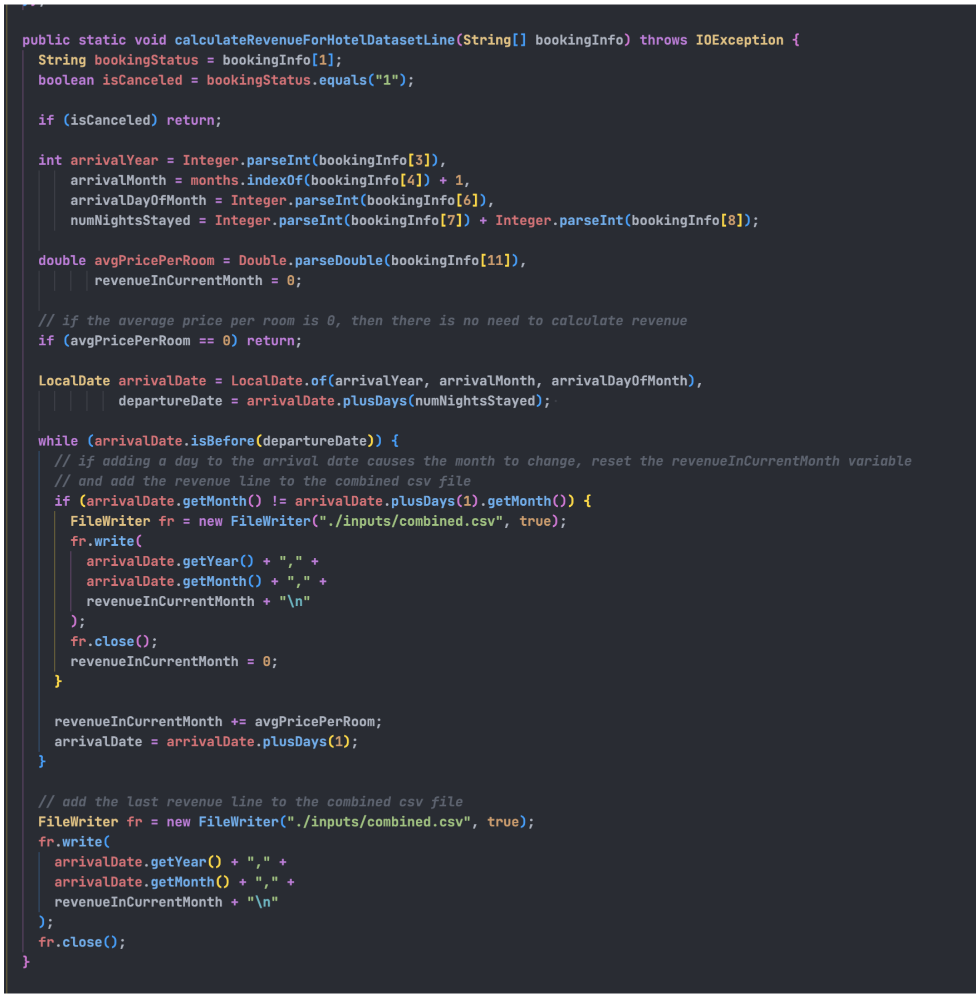
Description

Combining Datasets

The combining of `customer-reservations.csv` and `hotel-bookings.csv` occurs when running `JoinDatasets.java`. A directory named inputs is required in the folder this script is being run in because it adds all data provided by the user into a `combined.csv` file within this directory. Users will need to provide the path to the csv file they want to add to `combined.csv` as an argument of `JoinDatasets`, an example of this is listed below.

```
mkdir -p inputs
javac JoinDatasets.java -d classes
java -cp classes JoinDatasets customer-reservations.csv
```

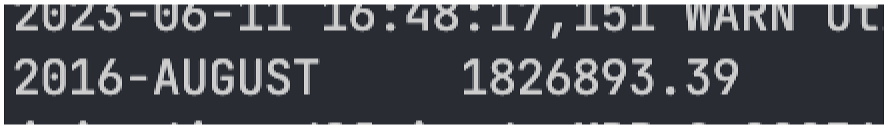
Therefore, this script needs to be run individually for both customer-reservations and hotel-bookings. JoinDatasets.java iterates through each reservation from both files and calculates the total revenue for that stay. If the stay overlaps between months, the stay is separated by month before being added into `combined.csv`. In one of the datasets, an invalid date was discovered (2018-02-29, 2018 was not a leap year). Reservations with this date were changed to 2018-02-28. Any canceled reservations were ignored because no revenue was generated from them. The month, year, and calculated revenue for each reservation were written to `combined.csv`, which was used for the MapReducer. A screenshot of the function used to add a reservation from the `hotel-reservations.csv` dataset can be seen below.



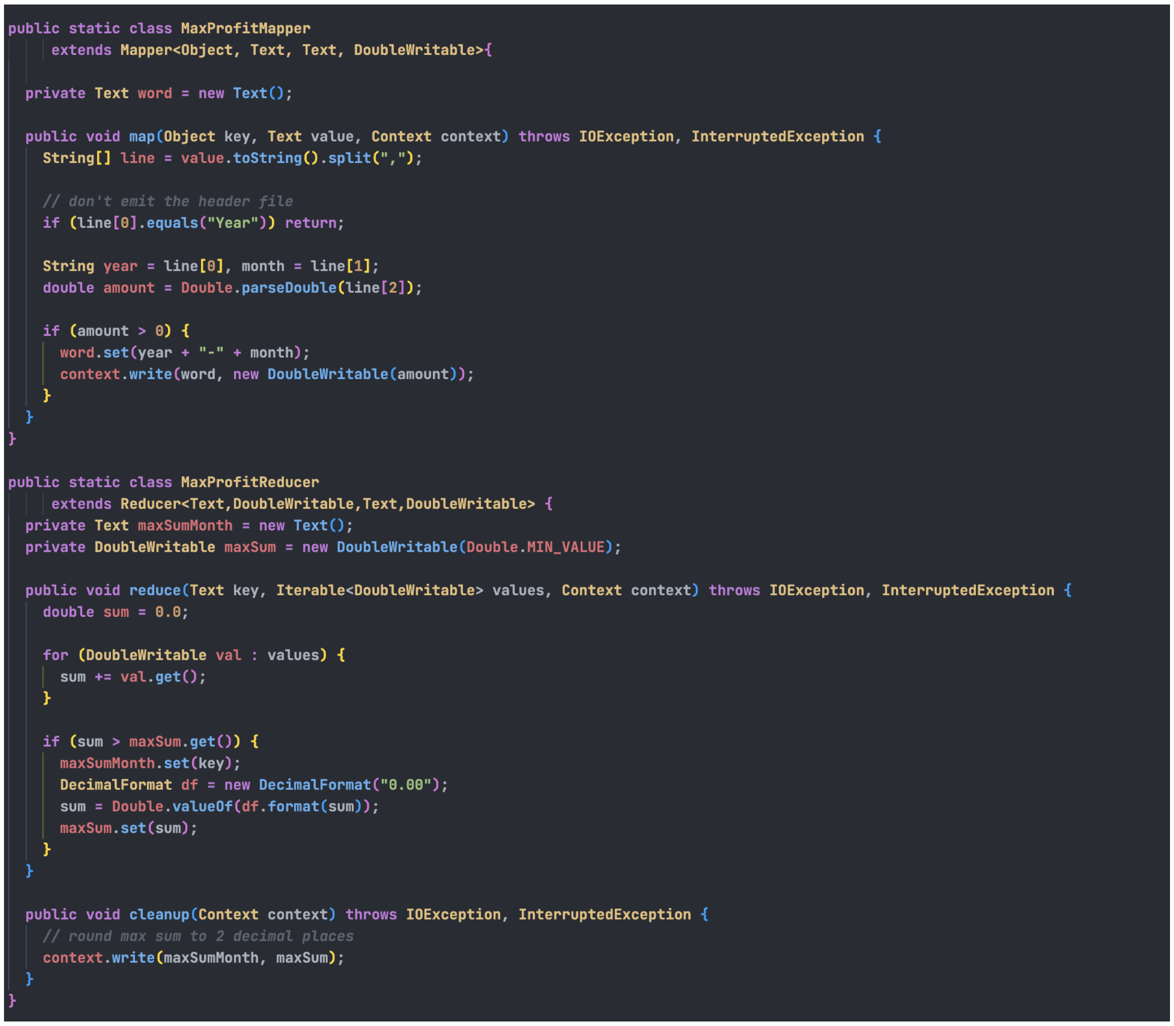
```java
public static void calculateRevenueForHotelDatasetLine(String[] bookingInfo) throws IOException {
  String bookingStatus = bookingInfo[1];
  boolean isCanceled = bookingStatus.equals("1");

  if (isCanceled) return;

  int arrivalYear = Integer.parseInt(bookingInfo[3]),
      arrivalMonth = months.indexOf(bookingInfo[4]) + 1,
      arrivalDayOfMonth = Integer.parseInt(bookingInfo[6]),
      numNightsStayed = Integer.parseInt(bookingInfo[7]) + Integer.parseInt(bookingInfo[8]);

  double avgPricePerRoom = Double.parseDouble(bookingInfo[11]),
         revenueInCurrentMonth = 0;

  // if the average price per room is 0, then there is no need to calculate revenue
  if (avgPricePerRoom == 0) return;

  LocalDate arrivalDate = LocalDate.of(arrivalYear, arrivalMonth, arrivalDayOfMonth),
            departureDate = arrivalDate.plusDays(numNightsStayed);

  while (arrivalDate.isBefore(departureDate)) {
    // if adding a day to the arrival date causes the month to change, reset the revenueInCurrentMonth variable
    // and add the revenue line to the combined csv file
    if (arrivalDate.getMonth() != arrivalDate.plusDays(1).getMonth()) {
      FileWriter fr = new FileWriter("./inputs/combined.csv", true);
      fr.write(
        arrivalDate.getYear() + "," +
        arrivalDate.getMonth() + "," +
        revenueInCurrentMonth + "\n"
      );
      fr.close();
      revenueInCurrentMonth = 0;
    }

    revenueInCurrentMonth += avgPricePerRoom;
    arrivalDate = arrivalDate.plusDays(1);
  }

  // add the last revenue line to the combined csv file
  FileWriter fr = new FileWriter("./inputs/combined.csv", true);
  fr.write(
    arrivalDate.getYear() + "," +
    arrivalDate.getMonth() + "," +
    revenueInCurrentMonth + "\n"
  );
  fr.close();
}
```

MapReducer

After combining the two datasets, the MapReducer could be used to find the month that generated the most revenue. `Combined.csv` is the only file used as input. The mapper goes through each line and emits the month and year combined as a key, along with the revenue for that reservation as the value. The reducer then combines each reservation from every month. If the revenue generated from a given month is greater than the revenue seen so far, the old month is replaced in the tracking. In `cleanup()`, the month that had the maximum revenue, and the maximum revenue are written to `/outputdata/part-r-00000` on the HDFS. Screenshots of the MapReducer, the month from which had the greatest revenue, and the revenue amount can be seen below. The month with the greatest revenue amount was August 2016, with $1826893.39. After running the MapReducer on several occasions, the typical runtime was ~25 seconds.



```
2023-06-11 16:48:17,151 WARN ut
2016-AUGUST      1826893.39
```



```java
public static class MaxProfitMapper
      extends Mapper<Object, Text, Text, DoubleWritable>{

  private Text word = new Text();

  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    String[] line = value.toString().split(",");

    // don't emit the header file
    if (line[0].equals("Year")) return;

    String year = line[0], month = line[1];
    double amount = Double.parseDouble(line[2]);

    if (amount > 0) {
      word.set(year + "-" + month);
      context.write(word, new DoubleWritable(amount));
    }
  }
}

public static class MaxProfitReducer
      extends Reducer<Text,DoubleWritable,Text,DoubleWritable> {
  private Text maxSumMonth = new Text();
  private DoubleWritable maxSum = new DoubleWritable(Double.MIN_VALUE);

  public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException {
    double sum = 0.0;

    for (DoubleWritable val : values) {
      sum += val.get();
    }

    if (sum > maxSum.get()) {
      maxSumMonth.set(key);
      DecimalFormat df = new DecimalFormat("0.00");
      sum = Double.valueOf(df.format(sum));
      maxSum.set(sum);
    }
  }

  public void cleanup(Context context) throws IOException, InterruptedException {
    // round max sum to 2 decimal places
    context.write(maxSumMonth, maxSum);
  }
}
```