Assignment 1
Research Topic in AI

Student Name(s) : Saurabh Shashikant Hebbalkar, Sampritha Hassan Manjunath
Student IS(s) : 19232697, 19232922

Introduction:

We have designed and implemented a Real-Time Feedback system for Taxi Drivers in New York City. The system streams the data, processes it and provides information to the Taxi Drivers regarding Peak Time and Busy Locations in the city. It also provides information regarding the accidents and generates a daily report of Peak Hour. We developed the application in Java and leveraged the Python Library (Pandas) for pre-processing the data.

Pipeline Design:

Python is widely known for its time efficiency and Java is known for its structure and object orientation. We decided to use the best of both the worlds in designing our Microservice Architecture. There are approximately 87 Lakh unsorted records in the Yellow Taxi csv file, sorting the file manually in Excel causes data loss as Excel cannot handle large dataset (more than 1048576 rows). We first tried to sort the file in Java, but it was time consuming; therefore, we decided to use Python to sort the CSV file as it was more time efficient.

Once the file is sorted and saved, we stream the data from this file using Java Stream. The data is streamed sequentially. To generate the desired report, we need only the Pickup Timestamp and the Location ID; thus, we filter these two columns and ignore the other columns from the csv file. There were irrelevant records present in the csv file dating back to the year 2002; we also filtered them out and just fetched January 2018 data. This data is Streamed by the Publisher service.

We are using ActiveMQ, an open source message broker to communicate between the micro services. ActiveMQ connection is initiated at the Publisher service. As we have multiple micro services who rely on this processed data; our architecture implements Publish & Subscribe semantics. We initialize Topic at the Publisher and when the data is published, it is received by all the Subscribers who have subscribed to the Topic.

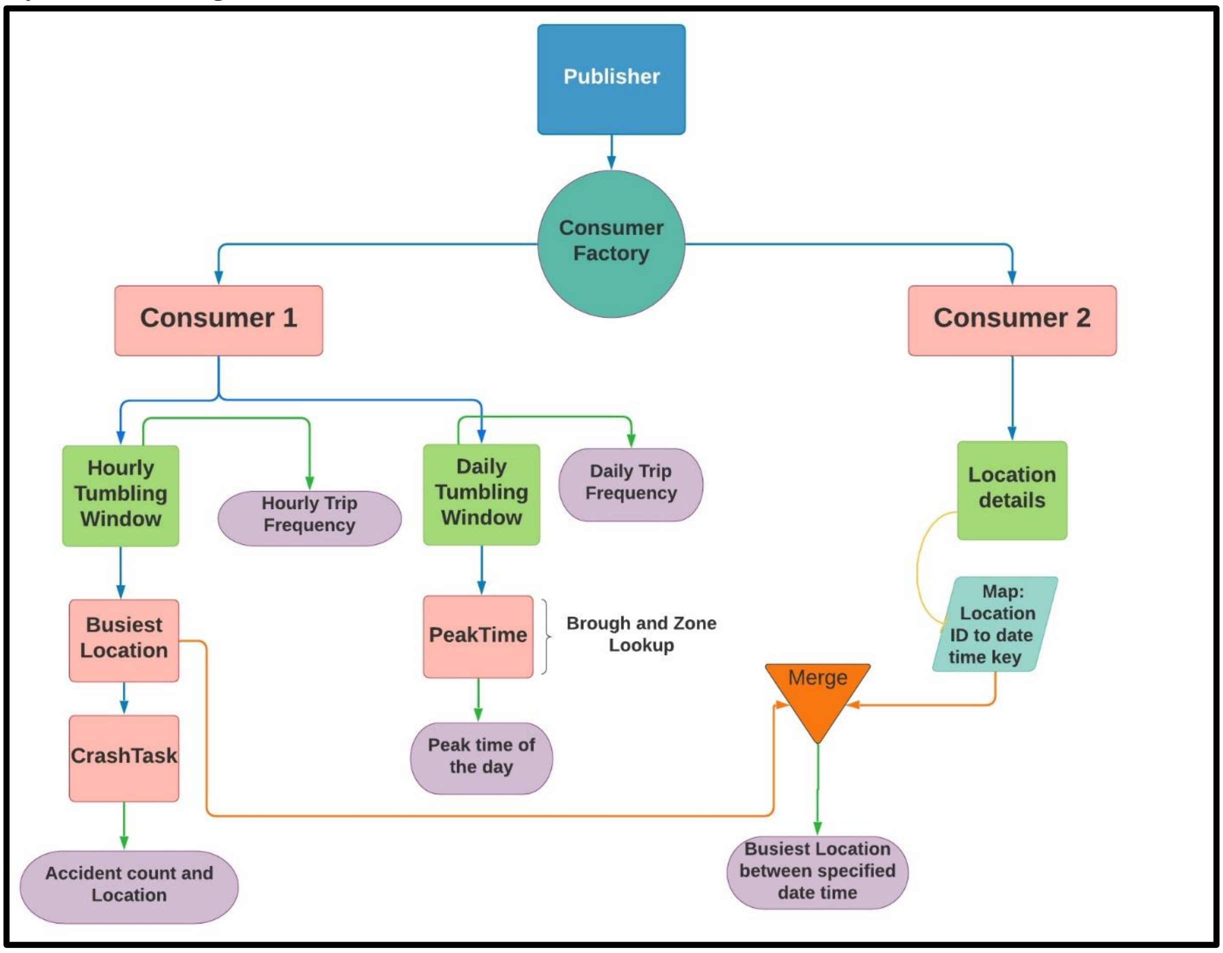
To have a reliable data communication between the Publisher & Subscriber, our inter-service communication checks for the data acknowledgement. When the message is received by the Subscriber, an acknowledgement is sent to the ActiveMQ and only then the next message is delivered to the subscriber. This process makes the system a bit slow, but avoiding it makes the system unreliable and affects the analytics. (Due to data and sequence loss)

By default, ActiveMQ creates a Message ID and Timestamp for each message which is being sent. As we have huge amount of data, creating ID and Timestamp for each message was consuming noticeable amount of time, hence we disabled the Message ID and Timestamp generation for each message. However, we are setting the delivery mode of the message to be Persistent which allows the messages to be written on the disk so that messages are not lost while broker restart.

Next, we have created a Consumer factory to initialize two consumers for the same Topic. In consumer factory we put the connection code which is commonly used to initiate both the consumers thus eliminating the redundancy. We have separated the processing of Location Mapping and Date & Time tumbling windows as two independent microservices as they both can be executed in parallel.

We are loading the crash data once independently and using it throughout the process to map the accidents, thus avoiding the need to stream the crash CSV every time to report the accident

Pipeline Flow Diagram:



Pipeline Output:

Consumer 1 Microservice: Creates two tumbling windows, one for each hour of the day and one for the day. The Daily window counts the frequency of the trips that occurred each day and the hourly window counts the hourly trip frequency. As the data is received in sequence, we calculate the trip frequency for each hour, starting from 0th Hour to 23rd Hour. The Date and time is then passed to BusiestLocation Microservice to identify the busiest location. In the Daily window, we summarize the hourly data to generate the total number of trips happened in the day. The date is then passed to PeakTime microservice to identify the peak time of the day.

BusiestLocation Microservice: Looks up the Mapping information generated by Consumer 2. This data is, merged with the date and time information received from Consumer 1 to lookup Borough and Zones from the Location CSV and report the busiest location of specified date and time. The Busiest location also sends the borough name along with date and time to the CrashTask microservice to generate report on accidents for the specified date and time.

CrashTask Microservice: Counts the total number of accidents occurred in the given borough on specified date time and reports the same. It also reports on the total number of accidents occurred across the city for specified date time along with the location details.

PeakTime Microservice: Compares the trip frequency of each hour of the day and returns the time which has recorded the maximum trip frequency.

Consumer 2 Microservice: Receives the pickup time and location data from the publisher. It creates a Multimap of the data with Date and Hour as the key and location ID as the Value. This mapping is used get the busiest location ID for the specified date time.

Design Strength &Weakness:
Strengths:
Pub-Sub Model (With Topic as Destination):
We are using a Pub-Sub model with One Publisher and Two Subscriber. The data is received by the subscribers at the same time and they can start processing independently. The output from these two microservices is merged and the final report is generated. This saves a lot of time as both the processes run on different cores parallelly.

Efficient Storage:
Since dataset is large, it contains data for the entire month. We only need to print the daily and hourly report once; therefore, we process the hourly and daily data and once the report is displayed, we clear the storage and is used for new day's information.

Object Oriented Approach:
- We are processing the data file using pure Java 8 streams. We have not used any framework and thus it keeps our code simple and easy to understand and modify if required.
- We are making use of static variable which helps to preserve the data which is stored in the variable. This data can be accessed by other microservices without the need of object instantiation
- We are using Multimap data structure provided by Google's guava jar. This helps in holding multiple values for a single key without the need of using map reduce approach.
- We are also running the microservices as independent thread. This makes the control from the calling function to return immediately and continue with processing next data without waiting for the result from the called function(thread).

Weakness:
Sorted Data:
We need the data to arrive in sequence; this system will work fine in real-time where the data arrives in sequence. If the data does not arrive in sequence, our system will still generate report but on the incomplete data.

Crash data Store:
We are currently loading the crash data only for January 2018 and storing it in a list. This list is used to generate a report on Accidents. We are not streaming the crash data and in future we can change the code to make it scalable.

Report stockpile:
We are currently not writing the report to any external source hence it limits us from referring to previous result. This can be avoided by implementing report logging.

Scalability:

- Whole taxi data is being streamed line by line without storing in memory. Even if the dataset is increased substantially, the system will be able the support it.
- We are using 1D array of size 24 to hold each day's trip count. On processing of each day's record, this array is nullified and used for next day. This approach will enable us to process any number of days without the need to provide extra storage.
  This approach is not only restricted to 1 month's data but, can also be used for multiple months.
- The system is reliable and generates expected results as each message is acknowledged and the message delivery is persistent. There is no issue of data loss or sequence loss in data communication.
- As we are using parallel processing and multi-threading, the resource allocation for the micro service is done efficiently.
- Try-Catch and Exception handling is done for all the classes there for as we scale, we will be able to handle the exceptions efficiently and solve new issues if they arise.
- We have created Consumer factory so that multiple subscribers can subscribe to specific Publisher topic at the same factory. If new subscribers want to subscribe to the same Publisher Topic, it just needs to be initiated at the factory and connection code can be reused.
- We can create many such Consumer Factories based on specific Topics and group them together eliminate redundancy.
- Free & Open source software are used to develop the system.
- Annotations available in Java provides the supplemental information about the program. For example, default run() method of the thread is overridden by the custom run method defined in each Thread class by using @Override
- Since the microservices are build using Java, a lot of libraries and resources are available and there are lot of cloud providers who can scale Java based microservices.

Reference:

1. Cugola, Gianpaolo, and Alessandro Margara. "Processing flows of information: From data stream to complex event processing." ACM Computing Surveys (CSUR) 44.3 (2012): 15
2. Jamshidi, Pooyan, et al. "Microservices: The journey so far and challenges ahead." IEEE Software, 2018.
3. Taibi, Davide, and Valentina Lenarduzzi. "On the definition of microservice bad smells." IEEE software, 2018
4. https://activemq.apache.org/components/artemis/documentation/1.5.0/perf-tuning.html
5. https://activemq.apache.org/slow-consumer-handling
6. https://activemq.apache.org/components/artemis/documentation/1.0.0/perf-tuning.html
7. https://activemq.apache.org/performance-tuning

Appendix:

BusiestLocation.java

```java
package yellowcab;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.AbstractMap;
import java.util.Collection;
import java.util.Map;
import java.util.function.Function;
import static java.util.stream.Collectors.counting;
import static java.util.stream.Collectors.groupingBy;
import java.util.stream.Stream;
import static yellowcab.CrashTask.getCrashCountOnBusiestLoc;
import static yellowcab.Consumer2.locationID;

public class BusiestLocation implements Runnable {

    String key;
    String fileName = "taxi+_zone_lookup.csv";

    // Set Key to the input key
    public BusiestLocation(String key) {
        this.key = key;
    }

    @Override
    public void run() {
        try {

            // Collection to store all the location ID for secified key
            Collection<Integer> busiestLoc = locationID.get(key);

            // Count the frequency of trips started at Pickup location
            Map<Integer, Long> collect = busiestLoc.stream().collect(groupingBy(Function.identity(), counting()));

            // get location ID where maximum trip booking are initiated
            int maxLoc = collect.entrySet().stream().max((entry1, entry2) -> entry1.getValue() > entry2.getValue() ? 1 : -1).get().getKey();

            // Borough and Zone Lookup for the identified busy location ID
            try ( Stream<String> data = Files.lines(Paths.get(fileName)).filter(s -> !s.isEmpty())) {
                data.skip(1).map(s -> new AbstractMap.SimpleEntry<>(s, s.split(",")))
                        .filter(s -> s.getValue()[0].equals(String.valueOf(maxLoc)))
                        .forEach(s -> {
                            System.out.println("Buisest Location : " + s.getValue()[1] + "," + s.getValue()[2]);
                            try {
                                // Get Accident count on the busiest Location
                                getCrashCountOnBusiestLoc(Integer.parseInt(key.split(",")[0]),
                                        Integer.parseInt(key.split(",")[1]), s.getValue()[1].replaceAll("^\"|\"$", ""));
                            } catch (IOException ex) {
                            }
                        });
            }
        } catch (IOException e) {
        }

    }
}
```

Consumer1.java

```java
package yellowcab;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import static yellowcab.YellowTaxi.thread;

public class Consumer1 implements MessageListener {

    static String key;

    // Initialize 1D array to hold counter for each record received
    static int[] dateTime = new int[24];

    @Override
    public void onMessage(Message message) {
        try {
            int dayFreq = 0, hourFreq = 0;

            if (message instanceof TextMessage) {
                TextMessage textMessage = (TextMessage) message;

                /*
                    When the message is not reached eof.
                        1. Enable Hourly Tumbling Window
                        2. Enable Daily Tumbling Window
                 */
                if (!(textMessage.getText().equals("eof"))) {

                    // Split the message into date and time
                    // Also split the date into dd, mm and yyyy to track the counter and array writing
                    // split the time to hh mm and ss
                    String day = textMessage.getText().split(",")[0].split(" ")[0];
                    int dd = Integer.parseInt(day.split("-")[2]);
                    int mm = Integer.parseInt(day.split("-")[1]);
                    int yyyy = Integer.parseInt(day.split("-")[0]);
                    String time = textMessage.getText().split(",")[0].split(" ")[1];
                    int hh = Integer.parseInt(time.split(":")[0]);

                    // Hourly Tumbling window
                    if (hh == 0 && dateTime[hh] == 0) {
                        dateTime[hh] += 1;
                    } else if (hh != 0 && dateTime[hh] == 0) {
                        dateTime[hh] += 1;
                        hourFreq = dateTime[hh - 1];
                        key = dd + "," + (hh - 1);
                        System.out.println("Total Trips between " + (hh - 1) + " to " + hh + " : " + hourFreq);
                        thread(new BusiestLocation(key), false);
```

```java
          }

          // Daily Tumbling window
          else if (hh == 0 && dateTime[23] != 0) {
            hourFreq = dateTime[23];
            key = (dd - 1) + "," + 23;
            System.out.println("Total Trips between " + 23 + " to " + 24 + " : " + hourFreq);
            // Identify and print busiest location
            thread(new BusiestLocation(key), false);

            // Summarise whole day's trip frequency
            for (int i = 0; i < 24; i++) {
              dayFreq += dateTime[i];
            }
            Thread.sleep(500);
            System.out.println("Total Trips made on " + (dd - 1) + "-" + mm + "-" + yyyy + " : " + dayFreq);
            // Identify and print peak time
            thread(new PeakTime(), false);
            Thread.sleep(500);
            dateTime = new int[24];
            dateTime[hh] += 1;
          } else {
            dateTime[hh] += 1;
          }
        }
        /*
          If the stream reaches the eof, print:
            1. 23rd to 24th hourly statistics
            2. Last day's peak time and total trip frequency
          This data is hardcoded as we need to process only for January 2018
        */
        else if (textMessage.getText().equals("eof")) {

          hourFreq = dateTime[23];
          key = 31 + "," + 23;
          System.out.println("Total Trips between " + 23 + " to " + 24 + " : " + hourFreq);
          // Identify and print busiest location
          thread(new BusiestLocation(key), false);

          // Summarise whole day's trip frequency
          for (int i = 0; i < 24; i++) {
            dayFreq += dateTime[i];
          }
          Thread.sleep(500);
          System.out.println("Total Trips made on " + 31 + "-" + 01 + "-" + 2018 + " : " + dayFreq);
          // Identify and print peak time
          thread(new PeakTime(), false);
        }
          }
        } catch (NumberFormatException | JMSException e) {
        } catch (InterruptedException ex) {
          Logger.getLogger(Consumer1.class.getName()).log(Level.SEVERE, null, ex);
        }
      }
    }
```

Consumer2.java

```java
package yellowcab;

import com.google.common.collect.ArrayListMultimap;
import com.google.common.collect.Multimap;
import javax.jms.Connection;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
import javax.jms.Session;
import javax.jms.TextMessage;

public class Consumer2 implements MessageListener {

    // Multimap to store all the Pickup Location at specified date time
    static Multimap<String, Integer> locationID = ArrayListMultimap.create();

    Connection connection = null;
    Session session = null;
    MessageConsumer consumer2 = null;

    // Constrcutor to set Connetion and session details
    public Consumer2(Connection connection, Session session, MessageConsumer consumer2) {

        this.connection = connection;
        this.session = session;
        this.consumer2 = consumer2;
    }

    @Override
    public void onMessage(Message message) {
        try {
            if (message instanceof TextMessage) {
                TextMessage textMessage = (TextMessage) message;

                // If eof receieved, end the consumer, session and connection
                if (textMessage.getText().equals("eof")) {
                    consumer2.close();
                    session.close();
                    connection.close();

                } else {

                    // Get Date Time and Pickup Location ID from the Message
                    int day = Integer.parseInt(textMessage.getText().split(",")[0].split(" ")[0].split("-")[2]);
                    int time = Integer.parseInt(textMessage.getText().split(",")[0].split(" ")[1].split(":")[0]);
                    int locID = Integer.parseInt(textMessage.getText().split(",")[1]);

                    // Generate Key as Date,Time
                    String key = day + "," + time;

                    // Add multiple Pickup location IDs to date,time key
                    locationID.put(key, locID);
                }
            }
        } catch (NumberFormatException | JMSException e) {
        }
    }
}
```

ConsumerFactory.java

```java
package yellowcab;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import org.apache.activemq.ActiveMQConnectionFactory;

public class ConsumerFactory implements Runnable {

    @Override
    public void run() {

        try {
            // Administrative object
            ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");

            Connection connection = connectionFactory.createConnection();

            // Create session object
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

            // Administrative object
            Destination dest = session.createTopic("myTopic");

            // Consumer1 subscribes to customerTopic
            MessageConsumer consumer1 = session.createConsumer(dest);
            consumer1.setMessageListener(new Consumer1());

            // Consumer2 subscribes to customerTopic
            MessageConsumer consumer2 = session.createConsumer(dest);
            consumer2.setMessageListener(new Consumer2(connection, session, consumer2));

            connection.start();

        } catch (JMSException e) {
        }
    }
}
```

CrashTask.java

```java
package yellowcab;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.AbstractMap;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Stream;
```

```java

public class CrashTask {

    // Data Strcuture to store Crash data
    static List<String> crashData = new ArrayList<>();
    static String fileName2 = "Crash.csv";
    static long count;

    public static void run() {
        // Stream Crash.csv
        try ( Stream<String> data = Files.lines(Paths.get(fileName2)).filter(s -> !s.isEmpty())) {
            data.map(s -> new AbstractMap.SimpleEntry<>(s, s.split(",")))
                    // Filter Jan 2018 from the csv
                    .filter(s -> (s.getValue()[0].split("/")[0].equals("01")
                    && s.getValue()[0].split("/")[2].equals("2018")))
                    .forEach(s -> {
                        // Handle empty values in the Borough field
                        if (s.getValue()[2].isEmpty()) {
                            s.getValue()[2] = "Location Not Available";
                        }
                        String crash = s.getValue()[0] + "," + s.getValue()[1] + ", " + s.getValue()[2];
                        // Add Date, Time and Borough deatils to the list
                        crashData.add(crash);

                    });
        } catch (IOException ex) {
        }
    }

    public static void getCrashCountOnBusiestLoc(int date, int time, String borough) throws IOException {
        // List to store crash details of specified date time
        List<String> temp = new ArrayList<>();
        crashData.parallelStream()
                .map(s -> new AbstractMap.SimpleEntry<>(s, s.split(",")))
                // Filter crashData on specified date time
                .filter(s -> Integer.parseInt(s.getValue()[0].split("/")[1]) == date
                && Integer.parseInt(s.getValue()[1].split(":")[0]) == time)
                .forEach(s -> {
                    String crash = s.getValue()[0] + "," + s.getValue()[1] + "," + s.getValue()[2];
                    // Add crash details of specified date time to a temperory list

                    temp.add(crash);
                });

        // Count the number of accidents occured in specified bororugh
        count = temp.parallelStream()
                .map(s -> new AbstractMap.SimpleEntry<>(s, s.split(",")))
                .filter(s -> s.getValue()[2].toLowerCase().contains(borough.toLowerCase()))
                .count();
        System.out.println("Number of accidents at " + borough + " : " + count);

        // Count the number of accidents occured overall city
        count = temp.parallelStream().count();
        System.out.println("Number of accidents across city : " + count);

        // Get the Location details of the accidents occured overall city
        List<String> t = new ArrayList<>();
        temp.stream()
                .map(s -> new AbstractMap.SimpleEntry<>(s, s.split(",")))
                .forEach(s -> t.add(s.getValue()[2]));

        Set<String> distinct = new HashSet<>(t);
        // Print the frequency in each location
        distinct.forEach((s) -> {
            System.out.println(s.toUpperCase() + ": " + Collections.frequency(t, s));
        });
        System.out.println();
    }
}
```

PeakTime.java

```java
package yellowcab;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

// Thread to calculate peak time
public class PeakTime implements Runnable {

    // Define start and end time of a day
    int startTime = 00;
    int endTime = 23;

    // List to hold frequency of each hour
    List<Integer> tripFreq = new ArrayList<>();

    // List to identify the time frame
    List<Integer> timeFrame = new ArrayList<>();

    @Override
    public void run() {

        // for each time, add the frequency to the list
        for (int i = startTime; i <= endTime; i++) {
            timeFrame.add(i);
            // fetch frequency from Consumer 1
            tripFreq.add(Consumer1.dateTime[i]);
        }

        // Calculate max frequency
        int maxVal = Collections.max(tripFreq);

        // Get the index of the max frequency
        // This gives the time frame of peak hour
        int maxIdx = tripFreq.indexOf(maxVal);
        System.out.println("Peak time : " + maxIdx + " to " + (maxIdx + 1));
        System.out.println("-------------------------------------------------\n");
        tripFreq.clear();
        timeFrame.clear();
    }
}
```

Publisher.java

```java
package yellowcab;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.AbstractMap;
import java.util.stream.Stream;

import javax.jms.Connection;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.jms.Session;

import org.apache.activemq.ActiveMQConnectionFactory;

public class Publisher implements Runnable {

    @Override
    public void run() {

        try {

            // Administrative object
            ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");

            Connection connection = connectionFactory.createConnection();
            connection.start();

            // Create session object
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

            // Administrative object
            Destination dest = session.createTopic("myTopic");

            // Producer object
            MessageProducer producer = session.createProducer(dest);

            // Avoid disk writing by setting Delivery Mode to NON_PERSISTENT
            producer.setDeliveryMode(DeliveryMode.PERSISTENT);

            /* Disable Message ID and Time Stamp creation for each message
                at producer end */
            producer.setDisableMessageID(true);
            producer.setDisableMessageTimestamp(true);

            String fileName = "SortedTaxiData.csv";

            // Stream Sorted CSV row by row
            try (Stream<String> data = Files.lines(Paths.get(fileName)).filter(s -> !s.isEmpty())) {
                data.skip(1).map(s -> new AbstractMap.SimpleEntry<>(s, s.split(",")))
                        // Filter by only Jan 2018 records
                        .filter(s -> (s.getValue()[1].split(" ")[0].split("-")[1].equals("01")
                        && s.getValue()[1].split(" ")[0].split("-")[0].equals("2018")))
                        .forEach(s -> {
                            try {
                                // Fetch Pickup DateTime and PickUp Location and create message
                                Message message = session.createTextMessage(s.getValue()[1] + "," + s.getValue()[7]);
                                // Send Message to ActiveMQ
                                producer.send(message);
                            } catch (JMSException e) {
                            }
                        });
                } catch (IOException ex) {
                }
                // Send End OF File Message to ActiveMQ
                // This will help to close the session once the file is processed
                producer.send(session.createTextMessage("eof"));
                connection.close();
            } catch (JMSException e) {
            }
    }
}
```

YellowTaxi.java

```
package yellowcab;

public class YellowTaxi {

    public static void main(String[] args) throws Exception {
        // load crash data
        CrashTask.run();

        // initialize publisher
        thread(new Publisher(), false);

        // Initialize gateway
        thread(new ConsumerFactory(), false);
    }

    // Run Threads
    public static void thread(Runnable runnable, boolean daemon) {
        Thread brokerThread = new Thread(runnable);
        brokerThread.setDaemon(daemon);
        brokerThread.start();
    }
}
```