





****

**Table of contents**

[**1. Introduction**](#_heading=h.30j0zll) **2**

[**2. Data exploration Activity**](#_heading=h.khh9mtw7wr8n) **3**

[**2.1. Tools/Softwares used for data exploration**](#_heading=h.xoklzgu34a34) **3**

[**2.2. Exploring the GTFS files**](#_heading=h.l36klp74phk2) **3**

[**2.3 Exploring the JSON files**](#_heading=h.tyjcwt) **6**

[**2.4. Exploring the Shapefiles files**](#_heading=h.3dy6vkm) **9**

[**3. Results**](#_heading=h.4d34og8) **12**

[**3.1 Distinguish the lines and periods by punctuality and by regularity.**](#_heading=h.2s8eyo1) **12**

[**3.2 Analyze the punctuality over the different lines, stops, and calendar.**](#_heading=h.17dp8vu) **21**

[**3.4 Analysis of problematic segments (bottlenecks).**](#_heading=h.26in1rg) **29**

[**3.5 Think your own of a valuable analysis on this data**](#_heading=h.lnxbz9) **31**

[**4. Conclusion**](#_heading=h.1ksv4uv) **31**

# **1. Introduction**

In Brussels, the company that is responsible for public transport is STIB-MIVB (which is an acronym for; Société des Transports Intercommunaux de Bruxelles - Maatschappij voor het Intercommunaal Vervoer (STIB-MIVB). STIB-MIVB transports over 1.2 million Brussels residents and commuters daily. However, the quality of the service provided to these passengers is very important as it has a direct impact on the daily activities of its passengers. The quality of the service depends on the satisfaction of the passengers. The latter is translated through the punctuality and regularity of Vehicle arrival times at the stop. Here, punctuality is when the vehicle arrives at the stop in the time which is scheduled, as in the GTFS file(A file that encodes schedule information that is printed on the papers that are displayed at stops) . On the other hand, regularity is a more important factor to measure the quality of service for frequent lines with headways lower than 12 minutes. For regularity the most important factor is to vary if the bus actually comes to the stop , the arrival time with respect to the GTFS file is not important.

In the context of this project, we were provided with data collected over a period of 3 weeks. These data included :

* JSON files : which contained the location of Vehicles for every +-30 seconds
* Esri Shape files that described the map(lines and stops ) of the STIB-MIVB network, on the 23rd of September.
* GTFS files that contained the offline plan/schedule covering the same period of the vehicle location data for the 3rd and the 23rd of September.

Given the above data, the aim of this project was for our team to analyze and provide quantitative

insights about the quality of service. This aim was further decomposed in to 5 sub points which included:

1. Distinguishing the lines/periods/day/timegroups for which the QoS is assessed by punctuality, and those assessed by regularity.
2. Analyzing the punctuality over the different lines, stops, and calendar days.
3. Analyzing the regularity over the different lines, stops, and time groups.
4. Identifying problematic lines/segments that cause large delays or irregularities which sometimes propagate to the rest of the line and also analyze to verify if there is a pattern for these irregularities.
5. Thinking our own valuable analysis on the data

To attain the objectives of this project we divided our report into 5 sections. A first part which that gave a brief introduction and context of the project, a second part that was focused on exploring the different datasets, a third part that presented the results of the different analysis that was done in order to answer all the questions with respect to the given objectives, and finally a fourth part that concluded our work by giving our own perspective/personal point of view based on the analysis we carried out.

# **2. Data exploration Activity**

## **2.1. Tools/Softwares used for data exploration**

This phase of the project was carried out using different tools/ softwares to explore the data that was provided to us for this project. Some of the softwares we used in this project included:

* **QGIS desktop application** that was used to load and extract the line information from the shape files into the CSV format.
* **Jupyter Lab** that was used to analyze both the Vehicule position files and the GTFS with the Python programming language, thanks to the great variety of python packages that eased the analysis and visualization tasks.
* **POWER BI** to visualize and explore the GTFS files.

## **2.2. Exploring the GTFS files**

As defined above the GTFS (General Transit Feed Specification (GTFS) file is a file that encodes schedule information that is printed on the papers that are displayed at stops. The folder of the GTFS files that were provided contained 09 files , 06 of which we considered important or mandatory to analyze and 03 of which we considered facultative. The following information is obtained from the Google API Documentation.

| No | File Name | Description | Required/Optional |
| --- | --- | --- | --- |
| 1 | Routes.txt | Transit routes  A route is a group of trips that are displayed  to riders as a single service. | Required |
| 2 | Trips.txt | Trips for each route. A trip is a sequence of two or more  stops that occur during a specific time period. | Required |
| 3 | Stop\_times.txt | Times that a vehicle arrives at and departs from each stop for  each trip. | Required |
| 4 | Stops.txt | Information about individual locations where vehicles pickup and drop passengers | Required |
| 5 | Calendar.txt | Service dates specified using a weekly schedule with start  and end dates. | Required |
| 6 | Agency.txt | Information about the transport agency | Required |
| 7 | Calendar\_dates.txt | Exceptions for the services defined in the calendar.txt. | Optional |
| 8 | shapes.txt | Provides rules for drawing lines on a map to represent a transit organization’s route. | Optional |
| 9 | Translations.txt | Provides fair translation information | Optional |

Fig. 01. GTFS files.

From the same source we found out that there was a connection between these files as represented below. These connections helped us to merge the different dataframes in order to create the data frame that will help us to answer the different questions of the analysis.

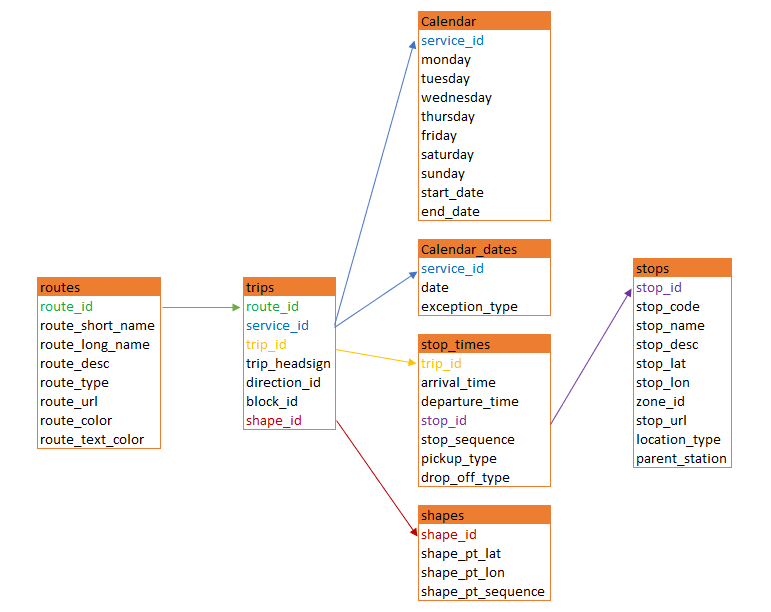


Fig. 02. General Transit Feed Specification (GTFS) table relations

During the study of the ‘stop\_times.txt’ file, dates exceeding 24 hours were identified.

Table

Description automatically generated

Fig. 03. Time at ‘stop\_times.txt.

To recalculate this time format, a code was written that brought the time data into a standard format. The *‘stop\_times.txt’* file has been replaced with *‘stop\_times\_new.txt’*.

## **2.3 Exploring the JSON files**

In addition to the GTFS files that make up our database, we have files in JSON (Javascript Object Notation) format. It is a popular data format used to represent structured data. Vehicle position files collect data about : real time vehicle position, the number of vehicles on a line that are running and the previous stop the vehicle passed through.

Indeed, the JSON files contain raw data but the JSON file format was not a suitable format for our data analysis given in the programming language we were using “Python”. So we used Python to convert the JSON files into CSV files.

We can see below the how the JSON data (“vehicle position”) structure looks :

*{'data':*

*[{'time': '1631270420474',*

*'Responses': [{'lines': [{'lineId': '1',*

*'vehiclePositions': [{'directionId': '8161',*

*'distanceFromPoint': 0,*

*'pointId': '8092'},*

*{'directionId': '8731', 'distanceFromPoint': 0, 'pointId': '8141'},*

*{'directionId': '8161', 'distanceFromPoint': 1, 'pointId': '8032'},*

*{'directionId': '8161', 'distanceFromPoint': 1, 'pointId': '8132'},*

*{'directionId': '8731', 'distanceFromPoint': 0, 'pointId': '8091'},*

*{'directionId': '8161', 'distanceFromPoint': 1, 'pointId': '8292'},*

*{'directionId': '8731', 'distanceFromPoint': 1, 'pointId': '8041'},*

*{'directionId': '8731', 'distanceFromPoint': 0, 'pointId': '8281'},*

*{'directionId': '8161', 'distanceFromPoint': 0, 'pointId': '8733'}]},*

*{'lineId': ...,*

*'vehiclePositions': [{'directionId': ...,*

*'distanceFromPoint': ...,*

*'pointId': ...}...]}*

Fig. 04. JSON data.

This file describes the following variables:

* '***data*'** is a JSON object.
* **'*time*'** is the time in milliseconds (unix epoch) at which the API was called (every 30 seconds).
* '***Responses***' is the array containing the result of the 9 API calls, each of these calls returns for the given line IDs all their vehicle positions.
* **'*vehiclePositions***' through different variables, this informs us about the positions of vehicles
* **'*directionId*'** is the identifier of the terminal stop.
* '*distanceFromPoint*' is the distance in meters between the vehicle and the last traversed stop.
* '***pointId*'** is the identifier of the last stop traversed by a vehicle.

Regarding data pre-processing, we performed a process of cleaning erroneous data (incorrect, incomplete, empty values ‘None’). Also, for data normalization, it is done by the json module. Importing into dataframe is done using the pandas module, which is a data analysis and manipulation tool.

The final merger of all 13 csv’s files was obtained using Pandas to combine all the JSON files to a single file.

Graphical user interface, text, application, email

Description automatically generatedFig. 05. JSON data merging.

The merged Data Frame combine the following data from GTFS files:

*'route\_id', 'service\_id', 'trip\_id', 'trip\_headsign','direction\_id', 'block\_id', 'shape\_id', 'arrival\_time',departure\_time', 'stop\_id', 'stop\_sequence', 'route\_short\_name', 'route\_long\_name', 'route\_type', 'monday', 'tuesday', 'wednesday','thursday', 'friday', 'saturday', 'sunday', 'start\_date', 'end\_date', 'start\_stop'*

Table

Description automatically generated

Fig. 06. Pandas dataframe for the file ‘vehiclePosition\_Final.csv’

Graphical user interface, application

Description automatically generated

Fig. 07. Power BI program was used at the start for general tasks of data exploration.

One of the problems that we faced working with JSON files is that 20 DirectionID, expressed by stop numbers in JSON, are missing in GTFS ‘stop.txt’ file.

Fig. 08. Stop numbers in JSON, are missing in GTFS

## **2.4. Exploring the Shapefiles files**

Geopositional information of stops and routes of lines was explored with the help of the QGIS software. The files ACTU\_LINES.shp and ACTU\_STOPS.shp contain the information about the spatial structure of the STIB/MIVB network: route of the lines and position of the stops. These files were loaded into the software and the map below was obtained, indicating all the geospatial coordinates of the stops in the different lines of the STIB\_MVIB network.

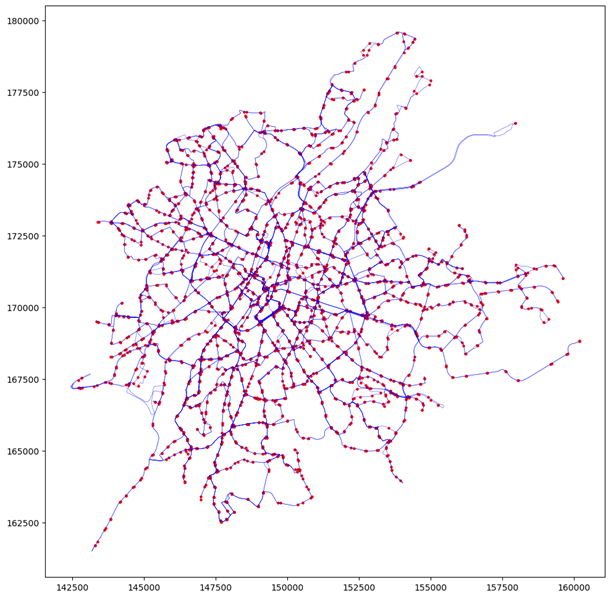


Fig. 09. Presentation of information of ACTU\_LINES.shp and ACTU\_STOPS combined in one graph with Matplotlib.

The table below is a representation of the information that was generated from the ACTU\_LINES shape file.

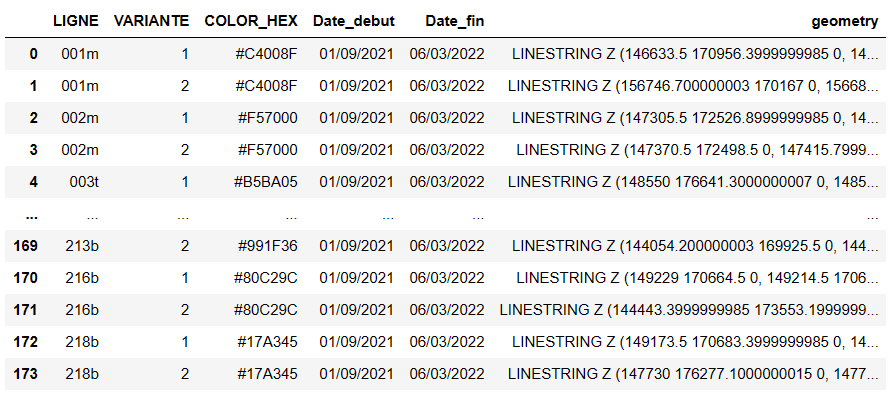


Fig. 10. Dataframe for the file ACTU\_LINE.shp

The attributes of the ACTU\_LINE.shp file are as follows and contain the following information:

* LIGNE: commercial line number (tram, bus or metro)
* VARIANTE: variant number of the line (1 (to) or 2 (from))
* COLOR\_HEX: official color code associated to the commercial line
* Date\_debut: start date of the represented network
* Date\_fin: end date of the represented network
* geometry: geometry field

Haven had an idea of the information found in the ACTU\_LINE we went further to explore the information found in the ACTU\_STOPS. Below is a view of the head of the data set generated from the attributes of the ACTU\_STOPS.

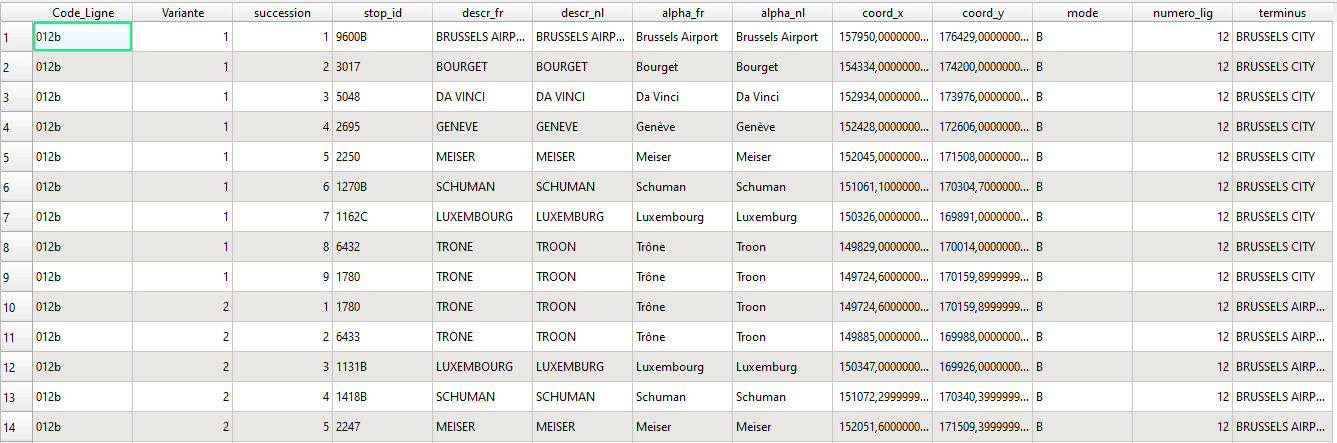


Fig. 11. Dataframe for the file ACTU\_STOPS.shp

The shapefile of the stops consists of all the stops for each line of the STIB/MIVB network covering both directions. Each stop is multiplied according to the number of lines serving this stop. The stop is graphically represented by a local point in terms of stop line position of the vehicle.

The following information is available for each attribute in the ACTU\_STOPS information table:

* code\_ligne: commercial line number (bus, tram or metro)
* VARIANTE: variant number of the line (1 (to) or 2 (from))
* succession: represents the stop position in the succession order of the stops along the itinerary
* stop\_id: stop number (internal identifier, 5 characters)
* descr\_fr: contains the functional name of the stop in capitals and in French
* descr\_nl: contains the functional name of the stop in capitals and in Dutch
* alpha\_fr: contains the official name of the stop in small letters and in French
* alpha\_nl: contains the official name of the stop in small letters and in Dutch
* coord\_x: coordinate X
* coord\_y: coordinate Y
* mode: represents the mode (B, T or M)
* geometry: geometry field
* numero\_lig: line number
* terminus: destination stop

# **3. Results**

## **3.1 Distinguish the lines and periods by punctuality and by regularity.**

**3.1.1 Methods and computations**

The first task that was assigned to us is to distinguish the lines/periods/day/timegroups for which the QoS is assessed by punctuality, and those assessed by regularity.

To achieve this goal, our first step was to create a common dataframe combining the following data from GTFS files:

*'route\_id', 'service\_id', 'trip\_id', 'trip\_headsign','direction\_id', 'block\_id', 'shape\_id', 'arrival\_time',departure\_time', 'stop\_id', 'stop\_sequence', 'route\_short\_name', 'route\_long\_name', 'route\_type', 'monday', 'tuesday', 'wednesday','thursday', 'friday', 'saturday', 'sunday', 'start\_date', 'end\_date', 'start\_stop'*

Table

Description automatically generated

Fig. 12. Merged GTFS data table.

Since the interval of movement is the difference in time between vehicles of the same route in the same direction sorted by time. So, we have applied a special kind of calculation in order to obtain data in the format as we see at every stop in real life.

A picture containing table

Description automatically generated

Fig. 13. STIB scheduled for route 95 at the “Grand-Place” stop.

Having this timetable for each line, it is possible to calculate the intervals for the whole day.

The traffic interval between vehicles is considered in terms of any one stop meaning that the vehicle interval is the same for all stops during a specific time.

For each individual line, we created a dataframe that includes all the trips that pass through a particular stop during the day. In other words, it is as if we are standing at the same stop all day and timing the intervals between vehicles of the same line.

To form this dataframe, the following parameters were introduced:

(result.stop\_sequence == 1) & (result.direction\_id == 1)

Graphical user interface, text, application

Description automatically generated

Fig. 14. Dataframe (result.stop\_sequence == 1) & (result.direction\_id == 1) .

Table

Description automatically generated

Fig. 15. Example of a generated Data Frame for route\_id = 1, stop\_sequence = 1 and direction\_id = 1.

The next step is to sort the time in ascending order. Now we have an ordered sequence of trips passing through one stop throughout the day. Using the function we calculate the difference between neighboring trips:

Chart, scatter chart

Description automatically generated

Fig. 16. Function to calculate difference between trips.

Table

Description automatically generated

Fig. 17. Time difference between trips of the bus line 52 “GARE CENTRALE - FOREST NATIONAL”.

Since the number of trips on one line during the day is large, this makes working with them not convenient. To avoid this problem, we divided the day into time windows and calculated the mean value of the movement interval for each time-window.

Text

Description automatically generated

Fig. 18. Divided the day into time windows

Table

Description automatically generated

Fig. 19. The frequency of transport on one tram line 5 “ERASME - HERRMANN-DEBROUX” during the tuesday.

**3.1.2 Results**

Now applying this methodology to calculate the frequency of transport, we have obtained data for buses, trams and metro for different days of the week. Assuming that on Monday, Tuesday, Thursday and Friday the timetable is the same.

Chart, bar chart

Description automatically generated

Fig. 20.

Graphic dash dotted line marks the border of 12 minutes. Everything above this line will be calculated according to the punctuality formula, and everything below will be calculated according to the regularity.

Chart

Description automatically generated

Fig. 21

Chart, line chart

Description automatically generated

Fig. 22

Chart

Description automatically generated

Fig. 23

Chart

Description automatically generated

Fig. 24

Chart

Description automatically generated

Fig. 25

Eventually we achieved a separation of transport modes such as tram, metro and bus and by time, in order to apply different calculation formulas for punctuality and regularity.

## **3.2 Analyze the punctuality over the different lines, stops, and calendar.**

Undoubtedly, for a complete and deep analysis it is necessary to have as much data as possible, also the data should be as accurate as possible. However, each task must be solved by the most efficient method.

Given the imperfection of the data contained in the JSON file and the absence of ID information for every vehicle, we apply the following approach. Since task 2 and task 3 require the analysis of the punctuality and regularity of the lines, therefore, stops were adopted by the coordinate system. More specifically, ‘*distanceFromPoint’, ‘pointID’* and *‘time’* information from the JSON file made it possible to reproduce the actual schedule for each stop. In other words, we recreated the picture of what really happened at each stop for each route.

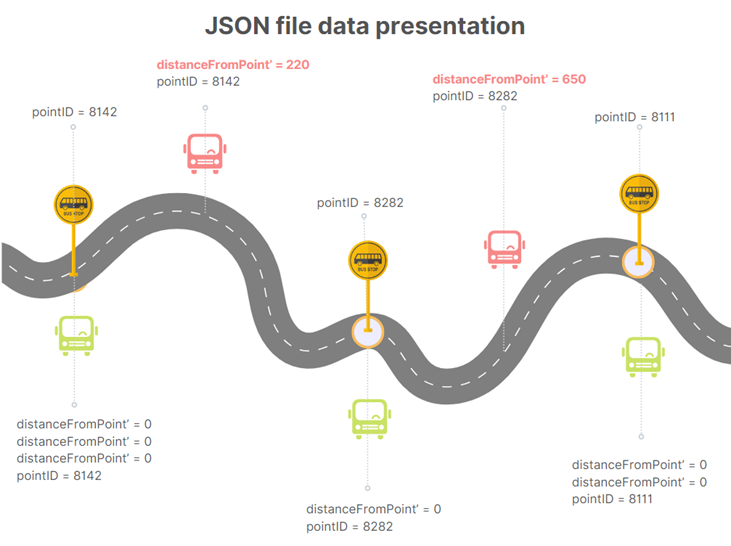


Fig. 26. JSON file data format

Each time when ‘*distanceFromPoint’* = 0 means that some bus of a certain route is at a stop. To define the line we use *‘directionID’,* to find a specific stop – *‘pointID’*. By time-ordering all points when ‘*distanceFromPoint’* = 0, and hence the bus was at the bus stop, we obtained a real schedule.

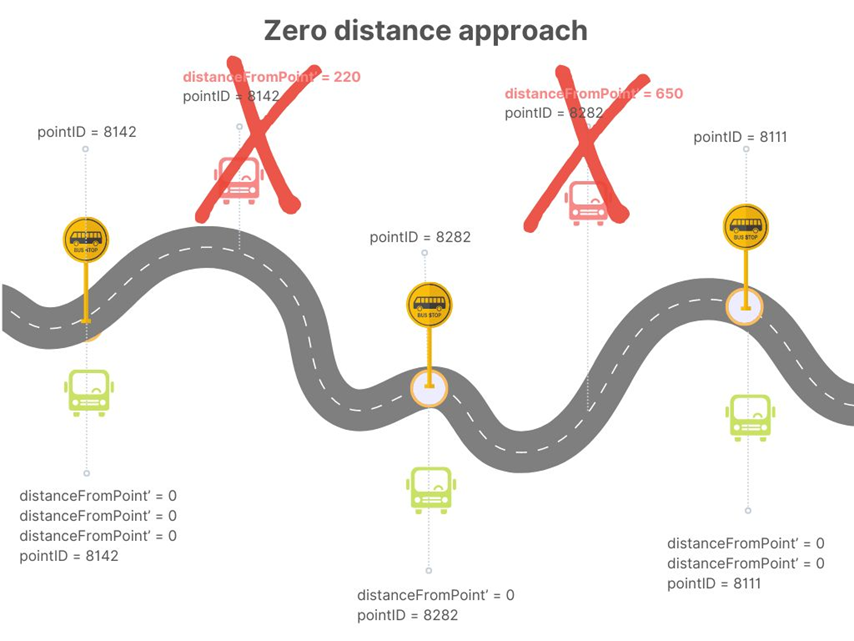


Fig. 27. JSON info only ‘distanceFromPoint’ = 0

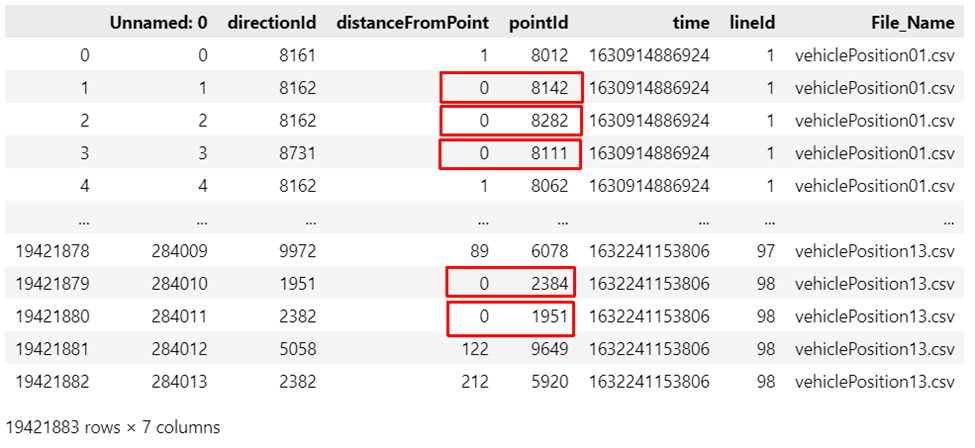


Fig. 28. JSON file

One of the features of this approach is the need to take into account the fact that the same vehicle that stands at a stop for more than 30 seconds generates a set of zero-marks (‘*distanceFromPoint’* = 0).

Exploring the Json file we see a lot of zero-marks especially at the final stops (terminuses). Which confirms the theory that the set of identical zero-marks indicate that it is the same bus standing at the bus stop, rather than that in thirty seconds one bus has left and another has arrived in its place.

To calculate the delays we joined 2 dataframes read from GTFS files and JSON files (we worked with the final version of the concatenated file including data from all 13 JSON files). The first dataframe has been named *df\_gtfs.* The second one has been named *df\_vp.*

|  | *df\_gtfs* | *df\_vp* |
| --- | --- | --- |
| Rows: | 1 906 448 | 19 421 883 |
| Columns: | 23 | 7 |

We applied *PySpark* from *Spark* in order to analyze the data since these 2 data frames have millions of rows. *Pandas* is performing very slowly on this amount of data and Big Data queries.

*Df\_gtfs* and *df\_vp* data frames have been converted to *RDD data frames* from *Spark* context. We named them *RDDdfvp* and *RDDgtfs* respectively.

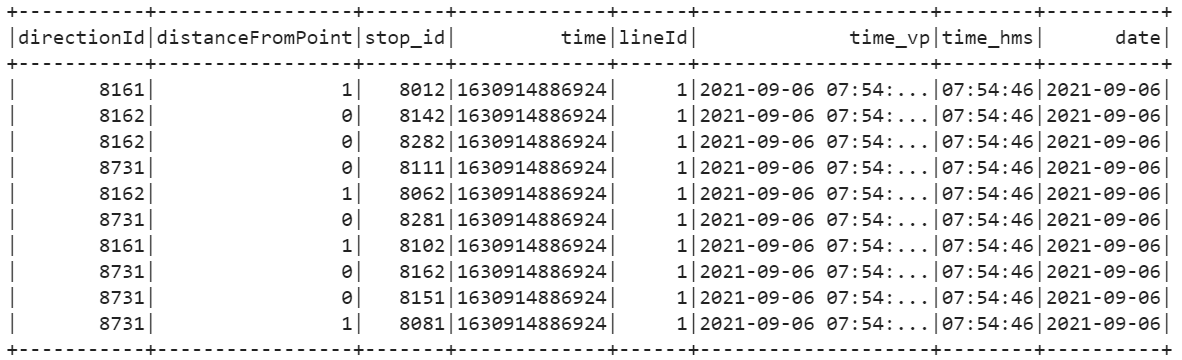


Fig. 29. *RDDdfvp* data frame representation.

As you can see from *RDDdfvp* data frame representation (see fig.29) we renamed *‘pointId’* to *‘stop\_id’* so that the features of 2 data frames were aligned. Plus, such features as *‘time\_hms’* and *‘date’* have been added based on the *‘time\_vp’* (*‘time\_vp’* has been obtained from the original feature ‘*time*’).

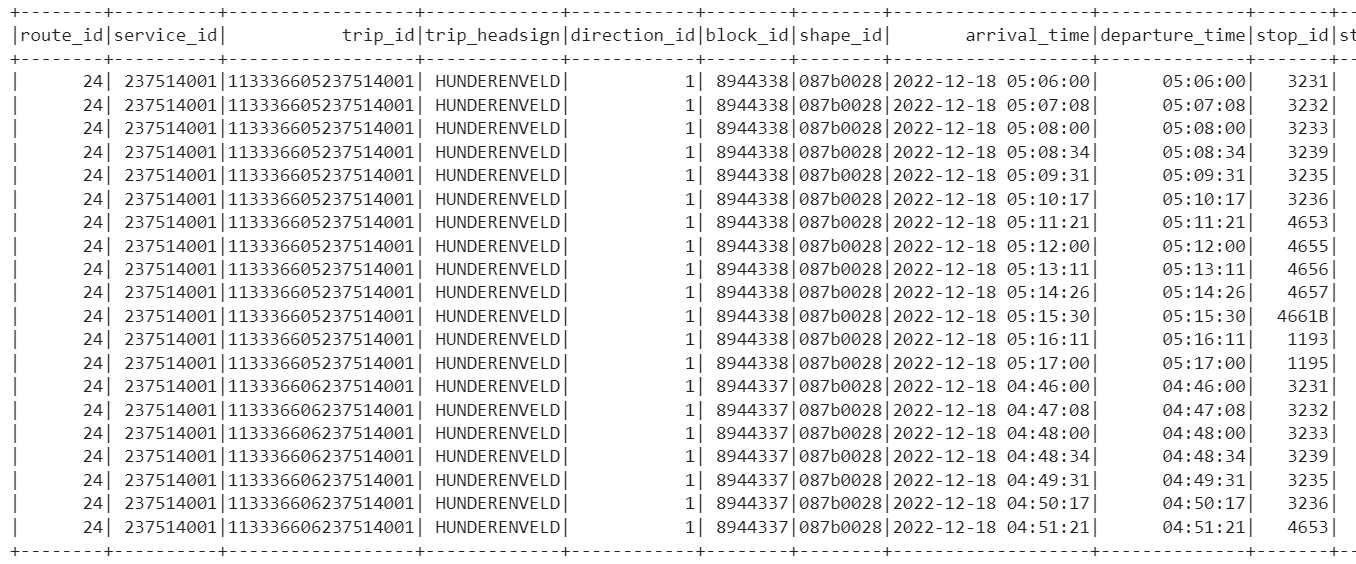


Fig. 30. *RDDgtfs* data frame representation

Bus 95 punctuality calculation.

We filtered both data frames as the following:

| *RDDdfvp* | *RDDgtfs* |
| --- | --- |
| *‘lineId’* = 95 | *‘route\_short\_name’* = 95 |
| *‘distanceFromPoint’* = 0 | ‘*direction\_id*’= 0 (optional) |
| *‘date’* = 06-07-08-09-10-11-12/09/2021 | *‘monday’ / ‘tuesday’/ ‘wednesday’/ ‘thursday’/ ‘friday’/ ‘saturday’/ ‘sunday’* = 1 |

Then we joined them on *‘stop\_id’.* The result is represented on figure 31.

The delay calculation approach has been chosen as the minimum absolute value of time difference between the real time vehicle position and scheduled time.

Here are the results (figure 32) and visualization (figure 33) of calculated delays:

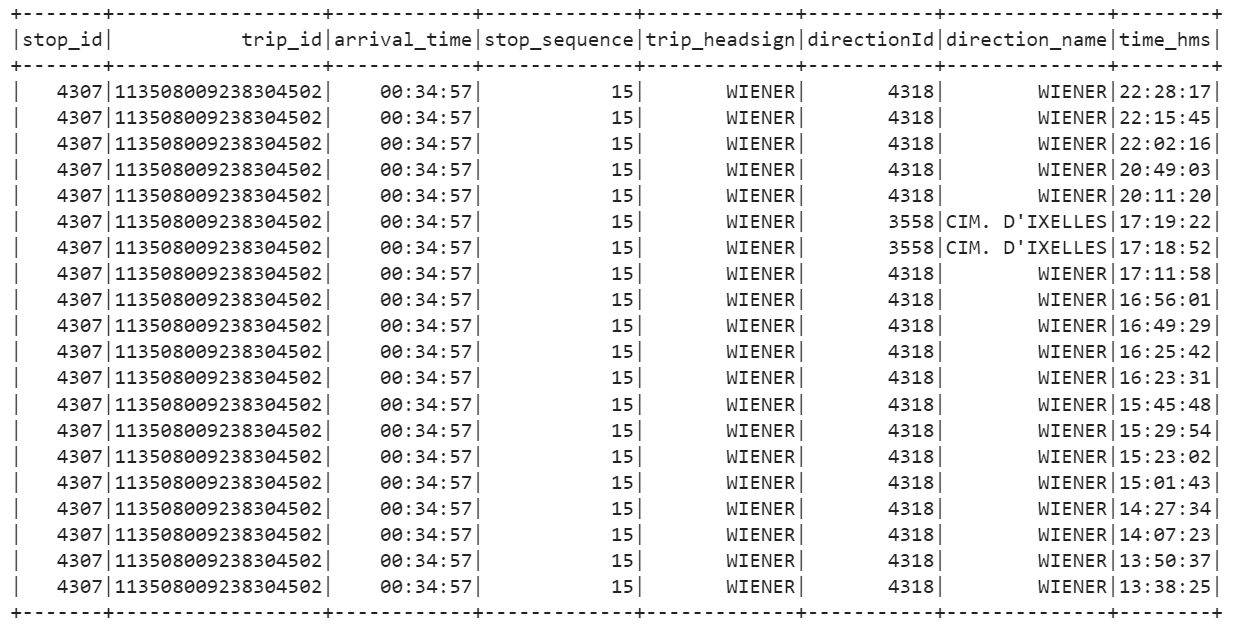


Fig.31.Joint *RDDvp* and *RDDgtfs* for line 95.

We group by ‘stop\_name’ (‘stop\_id’) and*‘arrival\_time’ and remember that ‘distanceFromPoint’* = 0 was implemented before.The data we received represent deviation from the schedule.

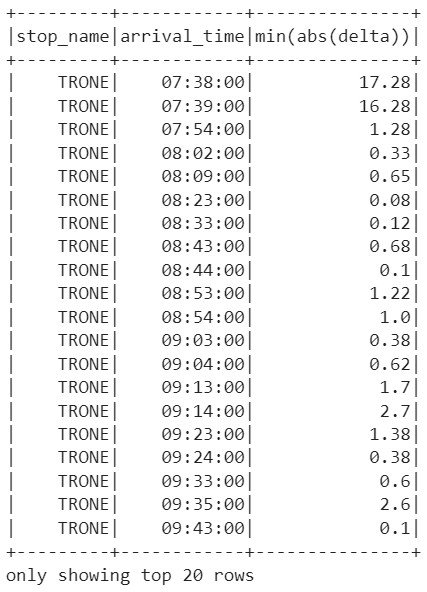


Fig.31a. Time difference between schedule and real time for line 95, stop ‘Trone’.

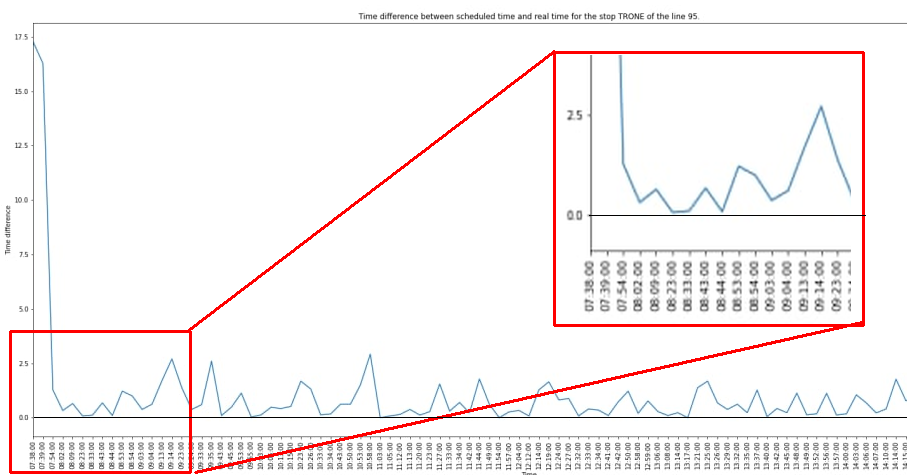


Fig.31b. Time difference between schedule and real time for line 95, stop ‘Trone’.

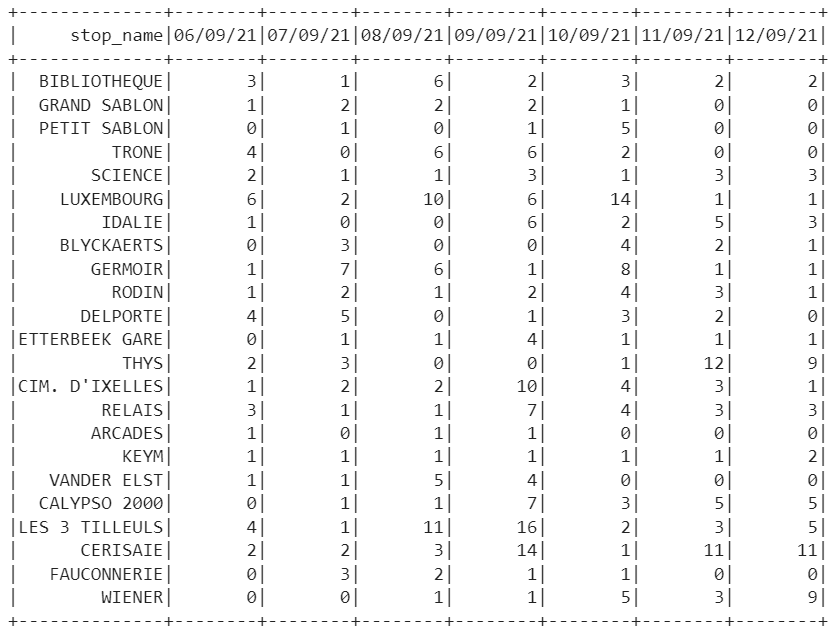


Fig.32. Delay calculation results for line 95.

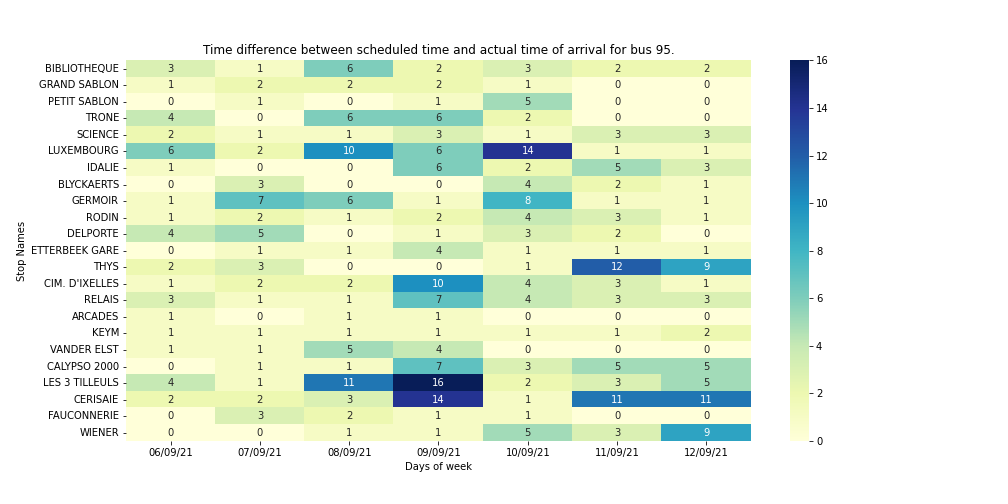


Fig.33. Delay visualization for line 95.

Same approach has been applied for metro line 2 and tramway line 3. We presented the results here below.

Metro 2 punctuality calculation.

| *RDDdfvp* | *RDDgtfs* |
| --- | --- |
| *‘lineId’* = 2 | *‘route\_short\_name’* = 2 |
| *‘distanceFromPoint’* = 0 | ‘*direction\_id*’= 0 (optional) |
| *‘date’* = 06/09/2021 - 12/09/2021 | *‘monday’ / ‘tuesday’/ ‘wednesday’/ ‘thursday’/ ‘friday’/ ‘saturday’/ ‘sunday’* = 1 |

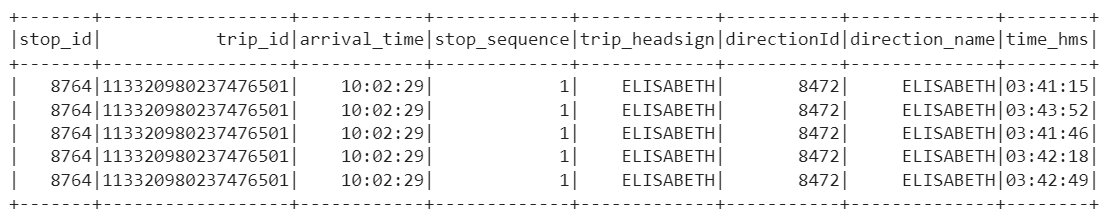


Fig.34. Joint *RDDvp* and *RDDgtfs* for line 2.

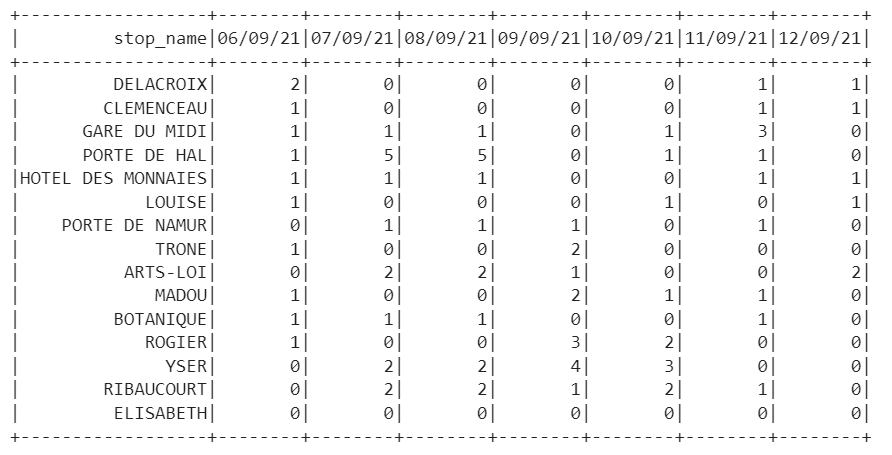


Fig. 35. Delay calculation results for line 2.

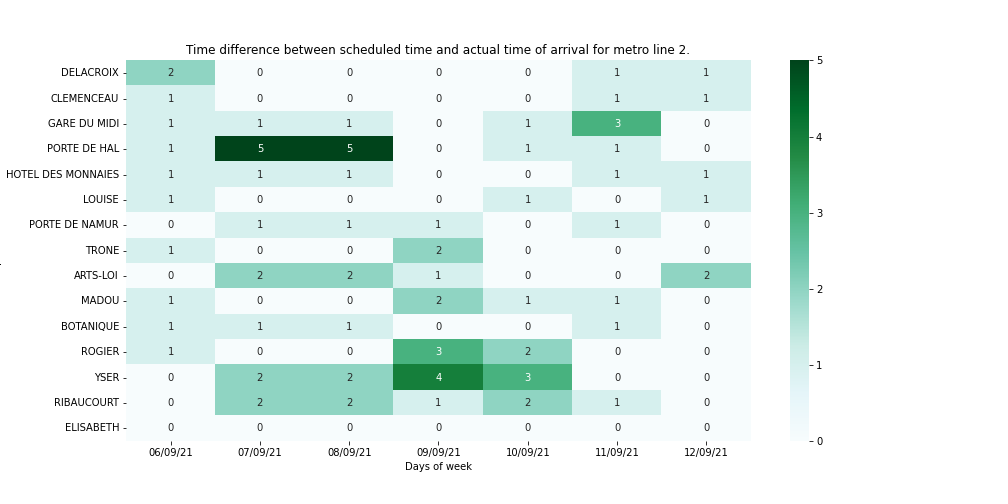


Fig.36. Delay visualization for line 2.

Tramway 3 punctuality calculation.

| *RDDdfvp* | *RDDgtfs* |
| --- | --- |
| *‘lineId’* = 3 | *‘route\_short\_name’* = 3 |
| *‘distanceFromPoint’* = 0 | ‘*direction\_id*’= 0 |
| *‘date’* = 06/09/2021 - 12/09/2021 | *‘monday’ / ‘tuesday’/ ‘wednesday’/ ‘thursday’/ ‘friday’/ ‘saturday’/ ‘sunday’* = 1 |

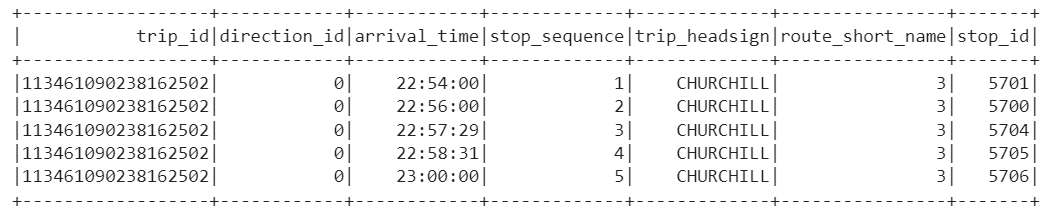


Fig.37. Joint *RDDvp* and *RDDgtfs* for line 3.

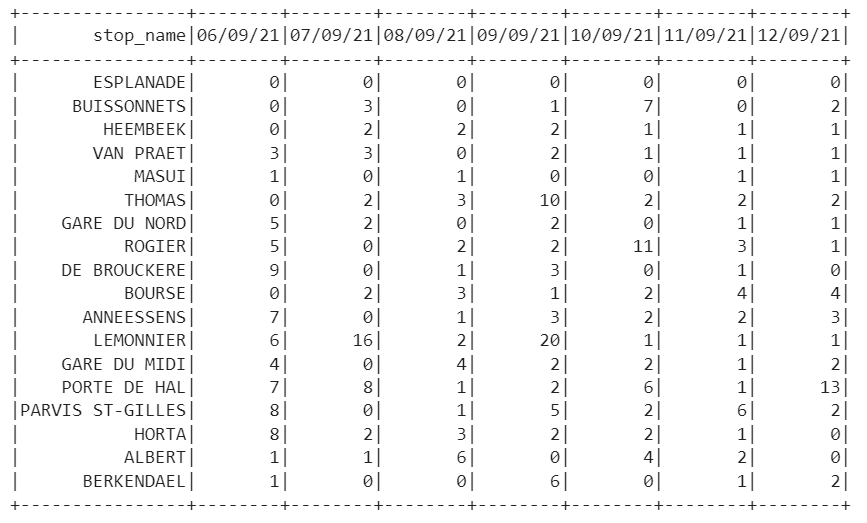


Fig. 38. Delay calculation results for line 3.

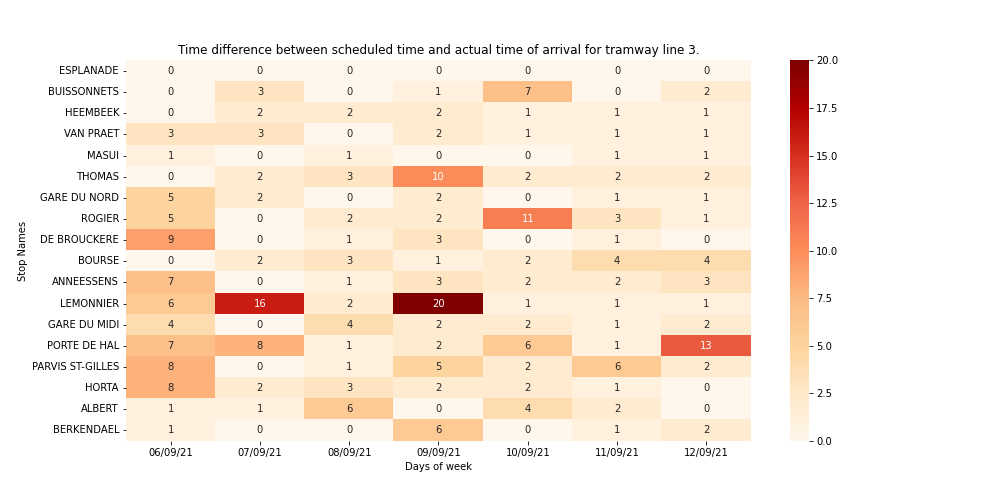


Fig.39. Delay visualization for line 3.

## **3.4 Analysis of problematic segments (bottlenecks).**

Traffic jams impact the movement of public transportation. This effect in particular slows down or stops at all traffic for some period of time. During some problems on the road, public transport waits until the bus/tram ahead can pass.

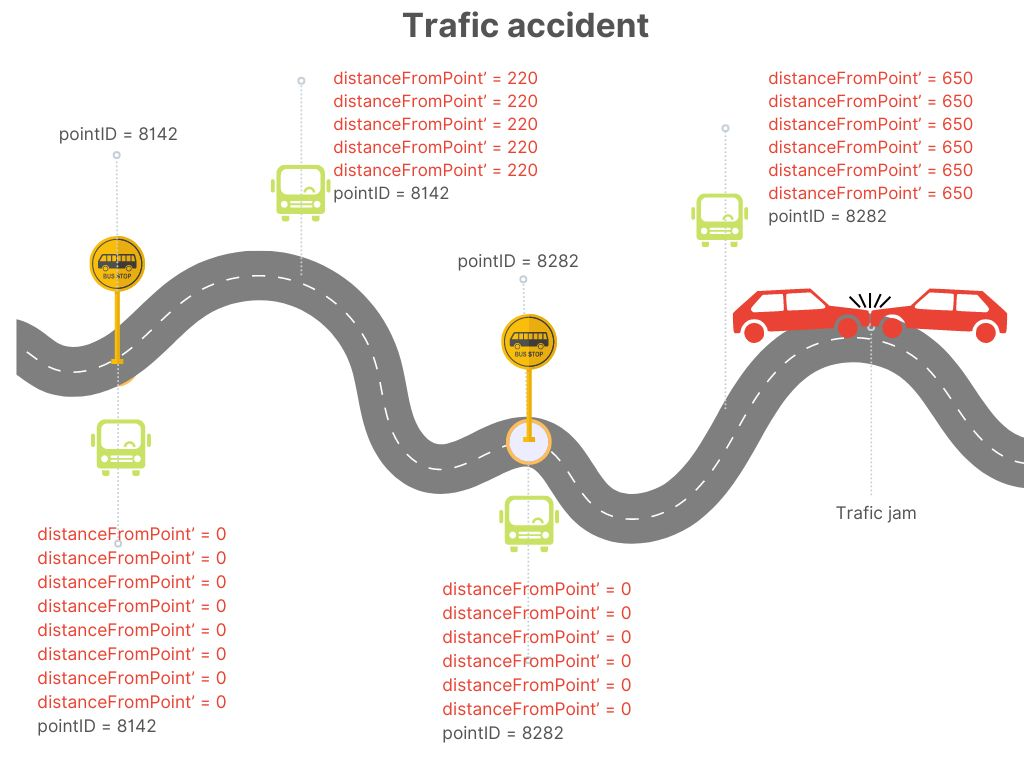


Fig.40. A traffic jam causes more (‘*distanceFromPoint’* = 0) to be generated.

Since the distance is counted from the stop, it is very convenient to observe a pair of following variables in conjunction:

*‘distanceFromPoint’+‘pointId’*

In general, when the same distance from the same stop is repeated over time in a variable *‘distanceFromPoint’+‘pointId’* - it tells us that the transport cannot move further (see Fig.40). Moreover, it’s possible to calculate the time of delays.And vice versa, when the transport situation is good we see a few identical *‘distanceFromPoint’+‘pointId’* repetition at the same place. Despite the fact that JSON is missing vehicle identification (personal ID), nevertheless it is still possible to analyze bottlenecks of routes.

## **3.5 Think your own of a valuable analysis on this data**

As part of the tasks of this project, we partitioned the lines based on regularity and punctuality. We calculated delays for each type of line. The result of these calculations was the identification of bottlenecks.

However, it is interesting to explore and build a model of the influence of factors on all the vehicles of the route. How much influence vehicles have on each other. For example, how much does the schedule change of one bus affect the others on the route. It is also interesting to analyze in deeper detail the influence of various reasons that lead to changes in the schedule, such as trivial peak hour traffic jams, weather conditions, accidents, and prolonged road construction.

Another issue that was left aside is the strict division of lines into regular and punctual.

As a result this information could be used to improve transport system management.

# **4. Conclusion.**

While GTFS data exploration we discovered some patterns on a daily basis, i.e. increase of frequency in rush hours and decrease of frequency in lunchtime and after midnight.

While JSON data exploration we used the “zero distance” method, i.e. we consider the vehicle being on the stop when the distance from the stop equals zero. Even though one vehicle can be on the zero distance from the stop and stay there for more than 30 seconds, this method detects systematic bottlenecks.

Undoubtedly, the identification of route vehicles and information about their speed could

open many new aspects for the project. For instance, speed info could have been used to manage the arrival time at each stop.

Challenges that we faced:

* JSON data has plenty of outliers, incorrect entries and incomplete information (no trip\_id and vehicle\_id).
* Departure and arrival time are equal, which is not possible in reality and causes errors in the schedule.

Despite that fact we applied methods that gave compelling results.