Hack My Ride

INFO-H423-Data Mining

Project Report

Tejaswini Dhupad
Khushnur Binte Jahangir
Himanshu Choudhary
Md Jamiur Rahman Rifat

December 20, 2021

TABLE OF CONTENT

Introduction

The Brussels Intercommunal Transport Company is the local public transport operator in Brussels, Belgium. It is usually referred to in English by the double acronym STIB-MIVB, or by its French acronym, STIB. It is responsible for the Brussels metro, Brussels trams and Brussels buses, linking with the De Lijn network in Flanders and the TEC network in Wallonia.[1]

The STIB network includes 4 metro lines, 18 tram lines and 52 bus lines (+ 11 Noctis night bus lines). Some 1,250 vehicles (tram, bus and metro) operate these lines and routes, 20/24h, 7 days a week.

STIB's lines/routes stretch over approximately 650 km, including

- 40 km of metro lines
- 147 km of tram lines
- 452 km of bus routes

Some 2,160 stops across the entire Brussels-Capital Region and its boroughs are served by STIB.

Since April 2007, STIB has been operating a night bus network: Noctis. On Fridays and Saturdays, 11 bus routes operate from twenty past midnight (00:20) until after 3 a.m. in the morning. They run from the centre of Brussels to the outer reaches of the Brussels-Capital Region.[2]

In this project, we have the STIB-MIVB data from it's open data portal and have tried to analyze the vehicle speed over different network segments and over time, analyze the vehicle delay at different stops and how it varies across stops and over time. Even tried the arrival time forecasting at a given stop in the route for a particular vehicle. Plus, we have even tried to infer the mode of transport (bus,tram,etc) for the GPS tracks data provided.

1. Data preprocessing

After receiving the data from the STIB-MIVB open-data portal, the data snapshot that we got was in JSON format. Hence, we did data cleaning and preprocessing. This section contains the information about the data used in the project along with data cleaning and processing operations.

1.1 Data Information

The data was collected for about 3 weeks from 6 Sept' 2021 to 21 Sept' 2021. It contains the following properties

- The location of all vehicles every +/- 30 seconds, encoded in JSON. The JSON format is described as -

Vehicle location JSON file format

```
{"data":[{
  "time":"1632409236387",
  "responses":[
      {"lines": [
      {"lineId": "1",
      "vehiclePositions": [
            {"directionId": "8161",
            "distanceFromPoint": 1,
            "pointId": "8122"}, ...]}, "lineId": ...]
      },{"lines":...}
  ],}, {...}, ... ]}
```

- GTFS files containing the offline plan/schedule covering the same period of the vehicle location data and two snapshots for 3 Sept' 2021 and 23 Sept' 2021 respectively.

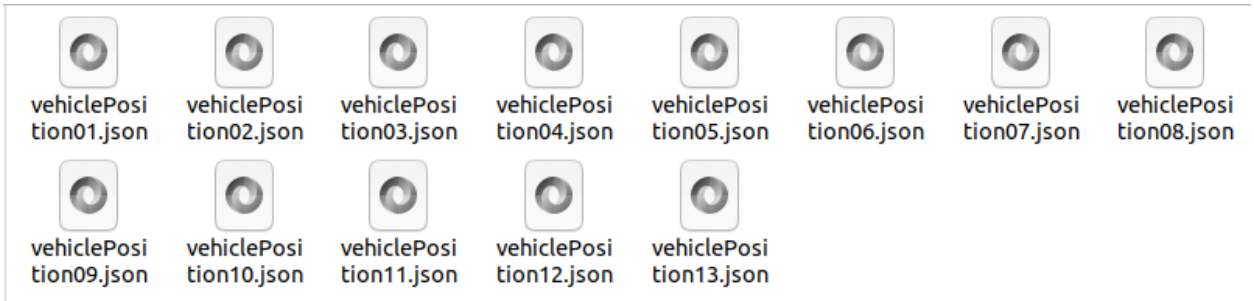
- Esri shape files describing the map (lines and stops) of STIB-MIVB network, two snapshots for 3 Sept' 2021 and 23 Sept' 2021.

- In addition we also got the data for 9 GPS tracks information.

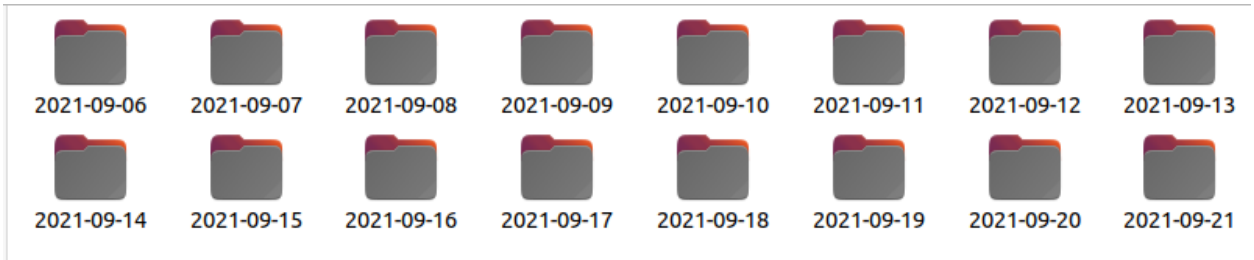
1.2 Data Flattening

The above mentioned JSON files contain the information of all the vehicles for the 3 weeks, which were converted to ".csv" file format along with the respective line ID, timestamp and direction Id and the point ID.

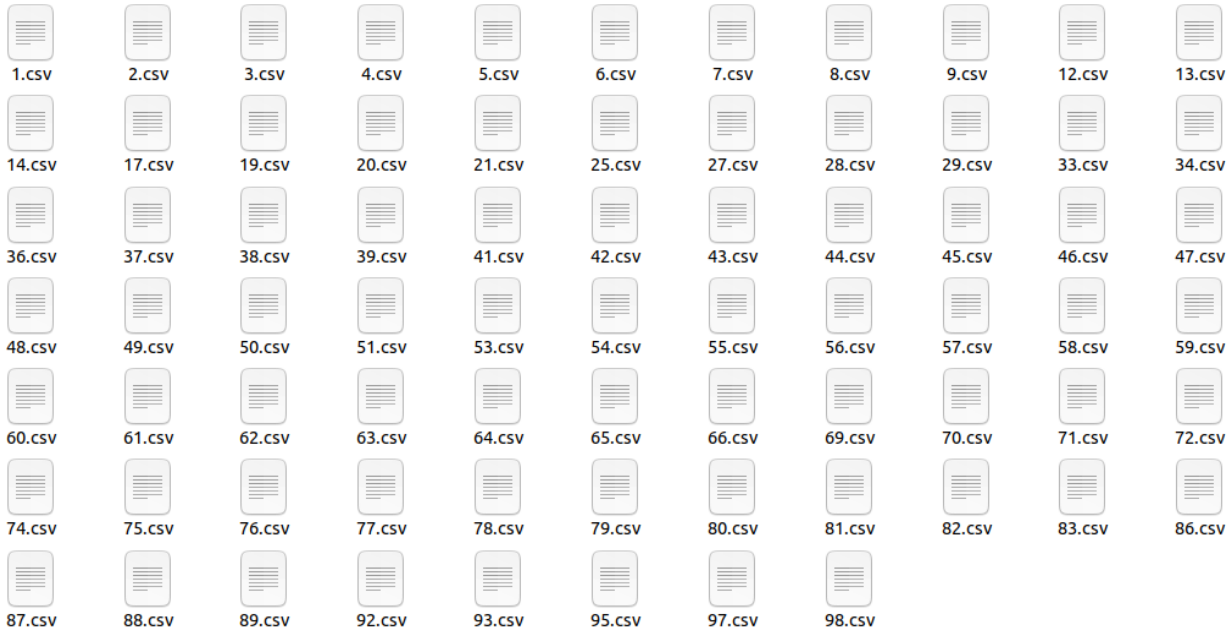
I. The original data consists of 13 json files with the above information.



II. The original data is converted into the ".csv" file formats and saved with their respective dates.



III. Where each data contains the information(".csv file") with the lineID. For example (6th September Data)

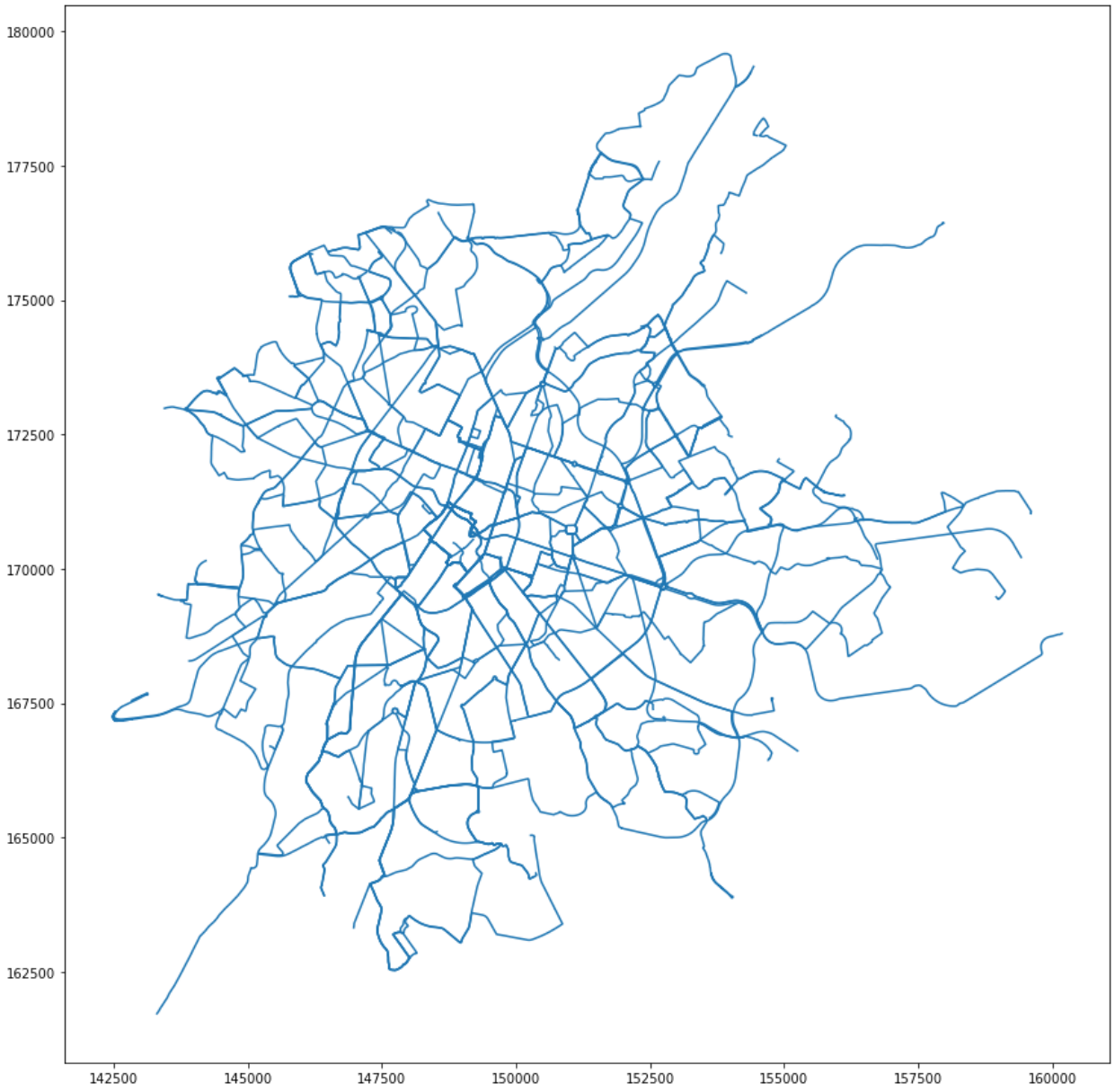


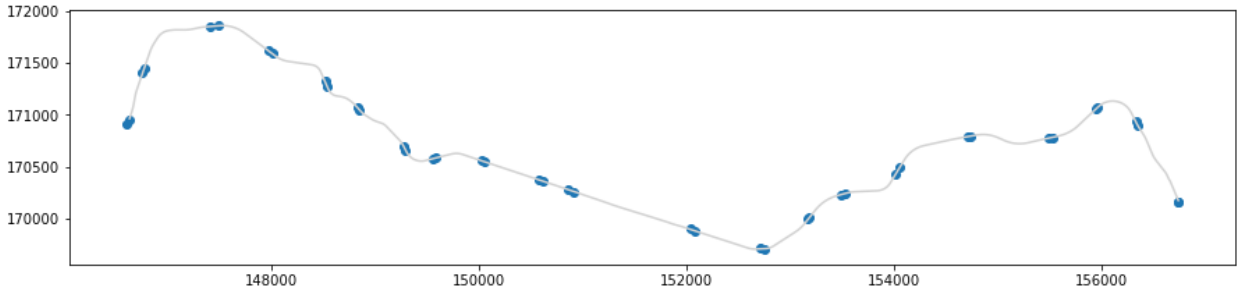
IV. And each csv file follows the format for each line_id, as shown below :

directionId	distanceFromPoint	pointId	line_id	timestamp	day	time
8161	1	8012	1	1630914886924	2021-09-06	07:54:46.924000
8162	0	8142	1	1630914886924	2021-09-06	07:54:46.924000
8162	0	8282	1	1630914886924	2021-09-06	07:54:46.924000
8731	0	8111	1	1630914886924	2021-09-06	07:54:46.924000
8162	1	8062	1	1630914886924	2021-09-06	07:54:46.924000
8731	0	8281	1	1630914886924	2021-09-06	07:54:46.924000
8161	1	8102	1	1630914886924	2021-09-06	07:54:46.924000
8731	0	8162	1	1630914886924	2021-09-06	07:54:46.924000
8731	0	8151	1	1630914886924	2021-09-06	07:54:46.924000
8731	1	8081	1	1630914886924	2021-09-06	07:54:46.924000
8161	0	8733	1	1630914886924	2021-09-06	07:54:46.924000
8731	0	8741	1	1630914886924	2021-09-06	07:54:46.924000
8731	0	8041	1	1630914886924	2021-09-06	07:54:46.924000
8161	0	8022	1	1630914917746	2021-09-06	07:55:17.746000
8162	0	8142	1	1630914917746	2021-09-06	07:55:17.746000
8162	1	8282	1	1630914917746	2021-09-06	07:55:17.746000
8731	0	8111	1	1630914917746	2021-09-06	07:55:17.746000
8162	0	8072	1	1630914917746	2021-09-06	07:55:17.746000
8731	0	8291	1	1630914917746	2021-09-06	07:55:17.746000
8161	0	8112	1	1630914917746	2021-09-06	07:55:17.746000
8731	0	8162	1	1630914917746	2021-09-06	07:55:17.746000
8731	1	8151	1	1630914917746	2021-09-06	07:55:17.746000
8731	0	8071	1	1630914917746	2021-09-06	07:55:17.746000
8161	0	8733	1	1630914917746	2021-09-06	07:55:17.746000
8731	1	8741	1	1630914917746	2021-09-06	07:55:17.746000
8731	0	8041	1	1630914917746	2021-09-06	07:55:17.746000
8161	0	8022	1	1630914948365	2021-09-06	07:55:48.365000
8162	1	8142	1	1630914948365	2021-09-06	07:55:48.365000
8162	1	8282	1	1630914948365	2021-09-06	07:55:48.365000
8731	1	8111	1	1630914948365	2021-09-06	07:55:48.365000
8162	0	8072	1	1630914948365	2021-09-06	07:55:48.365000

1.3 Distance Calculation

We used the “ACTU_LINES” and “ACTU_STOPS” shape files which contained the information about LINESTRING geometry for every Line ID and POINT geometry for each Point ID. To calculate the distance between two points we joined both of the files and for each line ID, for each direction(Variant) all the stops were projected on the line. After projecting all stop points the distance is calculated starting from the first stop to the last stop point(Direction ID) along the line. The final structured data can be seen in the below image.


180000
177500
175000
172500
170000
167500
165000
162500
142500
145000
147500
150000
152500
155000
157500
160000


172000
171500
171000
170500
170000
148000
150000
152000
154000
156000

line_id	variante	stop_id	succession	distance
1	1	8733	1	1.37210638552276E-09
1	1	8742	2	509.331152466031
1	1	8292	3	1451.5936615482
1	1	8282	4	2054.50465095434
1	1	8272	5	2720.6025411167
1	1	8012	6	3129.947653811
1	1	8022	7	3731.94444528701
1	1	8032	8	4075.83994676667
1	1	8042	9	4561.16450078185
1	1	8052	10	5152.63696899089
1	1	8062	11	5467.56915123496
1	1	8072	12	6686.4509977118
1	1	8082	13	7395.51925702507
1	1	8092	14	7930.49957045822
1	1	8102	15	8358.88228299996
1	1	8112	16	8992.07374229067
1	1	8122	17	9779.06442402476
1	1	8132	18	10589.6631048098
1	1	8142	19	11115.5672591105
1	1	8152	20	11640.6337448011
1	1	8161	21	12479.2241812173

1.4 Data Preparation

After flattening, the data was processed for vehicle separation. We used the “ACTU_LINES” and “ACTU_STOPS” shape files to get the information about routes on which vehicles are running. These files contain all the information about the geometry of lines as well as the sequence which the vehicle has to follow.

1.4.1 Data Cleaning

1. Since point_ids in provided data were in digits, the “ACTU_STOPS” file stop_ids were cleaned and converted to digits from string in order to match with the real world data.
2. Those point_ids which were not matching with the stop_ids of stop shape file, even after converting in digits were dropped.
3. Algorithm
    a. For each line divide data for each direction (Using direction ID)
    b. Save the sequence the vehicle will follow using the stop shape sequence file (ACTU_STOPS)
    c. Loop over the original data and follow these steps -

i. Start the vehicle journey from the point_id it starts and mark as visited
ii. Locate the point in the stop sequence, which we got for this direction_ID
iii. For each 45 (safer threshold value to include all 30 seconds closer values) seconds locate the vehicle either with the same point_id or the next point ID and mark visited true and save all information of point ID
iv. Break the journey of the vehicle if unable to locate it in the next 45 seconds.
v. Follow the same steps till the last stop
vi. Break when last stop is found

The whole process can be observed from the below images.

```
  DirId   Matched Points
  1780    9274
  9600    7894
  2247     642
  2250      41
  1162      28
```

```
1780: (1,
 'B',
 [9600, 3017, 5048, 2695, 2250, 1270, 1162, 6432, 1780],
 ['BRUSSELS AIRPORT',
  'BOURGET',
  'DA VINCI',
  'GENEVE',
  'MEISER',
  'SCHUMAN',
  'LUXEMBOURG',
  'TRONE',
  'TRONE'],
 ['POINT (157950 176429)',
  'POINT (154334 174200)',
  'POINT (152934 173976)',
  'POINT (152428 172606)',
  'POINT (152045 171508)',
  'POINT (151061.1 170304.7)',
  'POINT (150326 169891)',
  'POINT (149829 170014)',
  'POINT (149724.6 170159.9)']),
```

```
09:01:46.216000 -> 32 □ 31 □ 30 □ □ □ □
09:02:20.112000 -> 32 □ 31 □ 30 □ □ □ □
09:02:51.549000 -> 32 □ 31 □ 30 □ □ □ □
09:03:25.957000 -> 32 □ 31 □ 30 □ □ □ □
09:03:56.575000 -> 32 □ 31 □ 30 □ □ □ □
09:04:28.217000 -> 32 □ 31 □ 30 □ □ □ □
09:04:59.869000 -> 32 □ 31 □ 30 □ □ □ □
09:05:33.757000 -> 32 □ 31 □ 30 □ □ □ □
09:06:06.447000 -> 32 □ 31 □ 30 □ □ □ □
09:06:38.780000 -> 32 □ □ 31 30 □ □ □ □
09:07:14.110000 -> 32 □ □ 31 30 □ □ □ □
09:07:45.137000 -> 32 □ □ 31 30 □ □ □ □
09:08:18.350000 -> 32 □ □ 31 30 □ □ □ □
09:08:50.266000 -> □ 32 □ 31 □ 30 □ □ □
09:09:20.345000 -> □ 32 □ 31 □ 30 □ □ □
09:09:51.195000 -> □ 32 □ 31 □ 30 □ □ □
09:10:21.244000 -> □ 32 □ 31 □ 30 □ □ □
09:10:53.763000 -> □ 32 □ 31 □ 30 □ □ □
09:11:24.074000 -> □ 32 □ 31 □ 30 □ □ □
09:11:55.818000 -> □ 32 □ 31 □ 30 □ □ □
09:12:27.256000 -> □ 32 □ □ 31 30 □ □ □
09:12:58.489000 -> □ 32 □ □ 31 30 □ □ □
09:13:30.848000 -> □ □ 32 □ 31 □ 30 □ □
09:14:04.026000 -> □ □ 32 □ 31 □ 30 □ □
09:14:35.668000 -> □ □ 32 □ 31 □ 30 □ □
```

```
09:15:07.925000 -> □ □ 32 □ 31 □ 30 □ □
09:15:40.488000 -> □ □ 32 □ 31 □ 30 □ □
09:16:12.642000 -> □ □ 32 □ 31 □ 30 □ □
09:16:43.773000 -> □ □ 32 □ 31 □ 30 □ □
09:17:16.029000 -> □ □ 32 □ 31 □ 30 □ □
09:17:47.773000 -> □ □ 32 □ 31 □ □ 30 □
09:18:17.879000 -> □ □ 32 □ 31 □ □ □ 30
09:18:50.238000 -> □ □ □ 32 31 □ □ □ □
09:19:22.904000 -> □ □ □ 32 □ 31 □ □ □
09:19:55.263000 -> □ □ □ 32 □ 31 □ □ □
09:20:25.881000 -> □ □ □ 32 □ 31 □ □ □
09:20:59.878000 -> □ □ □ 32 □ 31 □ □ □
09:21:33.670000 -> □ □ □ 32 □ 31 □ □ □
09:22:03.981000 -> □ □ □ 32 □ 31 □ □ □
09:22:36.036000 -> □ □ □ 32 □ 31 □ □ □
09:23:08.289000 -> □ □ □ 32 □ 31 □ □ □
09:23:41.570000 -> □ □ □ □ 32 □ 31 □ □
09:24:13.314000 -> 33 □ □ □ 32 □ 31 □ □
09:24:44.547000 -> 33 □ □ □ 32 □ 31 □ □
09:25:16.600000 -> 33 □ □ □ 32 □ 31 □ □
09:25:48.957000 -> 33 □ □ □ 32 □ □ 31 □
09:26:20.242000 -> 33 □ □ □ 32 □ □ 31 □
```

The Final DataFrame can be seen from seen below.

day	directionId	line_id	veichle_id	mode	pointId	stop_name	next_point_id	stop_sequence	distanceFromPoint	prv_stop_time	time
2021-09-07	1780	12	19	B	9600	BRUSSELS AIRPORT	3017	1	0	07:33:30.982000	07:34:01.806000
2021-09-07	1780	12	19	B	9600	BRUSSELS AIRPORT	3017	1	565	07:34:01.806000	07:34:36.930000
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	391	07:34:36.930000	07:35:09.904000
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	732	07:35:09.904000	07:35:40.419000
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	902	07:35:40.419000	07:36:14.622000
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	1252	07:36:14.622000	07:36:45.240000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	0	07:36:45.240000	07:37:17.498000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	210	07:37:17.498000	07:37:51.699000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	324	07:37:51.699000	07:38:23.936000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	444	07:38:23.936000	07:38:54.266000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	661	07:38:54.266000	07:39:27.035000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	775	07:39:27.035000	07:39:57.653000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	1014	07:39:57.653000	07:40:30.728000
2021-09-07	1780	12	19	B	2695	GENEVE	2250	4	171	07:40:30.728000	07:41:00.937000
2021-09-07	1780	12	19	B	2250	MEISER	1270	5	1082	07:41:00.937000	07:41:34.115000
2021-09-07	1780	12	19	B	1270	SCHUMAN	1162	6	640	07:41:34.115000	07:42:07.805000
2021-09-07	1780	12	19	B	1162	LUXEMBOURG	6432	7	462	07:42:07.805000	07:42:38.423000

1.4.2 Speed and Stop Point Data Generation

Once we get the above data for each day of each line_id, we use our calculated distances to generate speed for each vehicle. Each row of the above data frame contains the information about the current and next stop of the vehicle along with the distance traversed by that vehicle (distance from pointID) and time taken.

day	directionId	line_id	veichle_id	mode	pointId	stop_name	next_point_id	stop_sequence	distanceFromPoint	time	speed
2021-09-07	1780	12	19	B	9600	BRUSSELS AIRPORT	3017	1	0	07:34:01.806000	0.000000
2021-09-07	1780	12	19	B	9600	BRUSSELS AIRPORT	3017	1	565	07:34:36.930000	16.142857
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	391	07:35:09.904000	23.803379
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	732	07:35:40.419000	11.000000
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	902	07:36:14.622000	5.000000
2021-09-07	1780	12	19	B	3017	BOURGET	5048	2	1252	07:36:45.240000	11.290323
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	0	07:37:17.498000	6.135942
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	210	07:37:51.699000	6.176471
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	324	07:38:23.936000	3.562500
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	444	07:38:54.266000	3.870968
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	661	07:39:27.035000	6.575758
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	775	07:39:57.653000	3.800000
2021-09-07	1780	12	19	B	5048	DA VINCI	2695	3	1014	07:40:30.728000	7.242424
2021-09-07	1780	12	19	B	2695	GENEVE	2250	4	171	07:41:00.937000	28.534359
2021-09-07	1780	12	19	B	2250	MEISER	1270	5	1082	07:41:34.115000	77.510094
2021-09-07	1780	12	19	B	1270	SCHUMAN	1162	6	640	07:42:07.805000	39.758604
2021-09-07	1780	12	19	B	1162	LUXEMBOURG	6432	7	462	07:42:38.423000	41.006906
2021-09-07	1780	12	19	B	1162	LUXEMBOURG	6432	7	462	07:43:11.704000	0.000000
2021-	1780	12	19	B	6432	TRONE	1780	8	0	07:43:45.599000	2.130067

The above data can be easily transformed to keep only those rows where the vehicle arrives for the first time. Once we get this data, after time adjustment, we can easily compare the times of this vehicle reaching that point with standard data defined for the vehicles.

2. Speed Analysis

In this section, we are going to explain our speed analysis for all the three modes of transport i.e., how the speed of metro, tram and bus are varying across different line_ID, different segments during weekdays and weekends. We are also going to show how speed is changing from morning to night to get the idea of average speed during peak hour and non-peak hours. We will demonstrate our findings by giving a graphical view (Barchart, line graph and heatmap) of the speed for these three transports.

Visualization Tool used:

- Tableau
- Kepler.gl

1. Weekday VS Weekend average speed of Metro, Tram and Bus (For all line id)

In the given data, we have found a total four-line id (1,2,5,6) for the metro. Among these line id, we try to find out which line id has the average highest speed for weekdays also for the. We have found out that line_Id-5 has the highest average speed(40.890 km/hr,41.118) for the metro [Fig-01,Fig-02]. In Fig-03, the average speed of the metro for all line IDs of weekdays and weekends are given together to compare the speed. We are showing the same analysis for bus but since bus has more line id than metro, the highest avg. The speed of the bus for all line_Id are being selected for an example. In [Fig-04,05] ,we can see that line ID-98 contains the highest speed (26.40 km/hr,29 km/hr) during weekdays and weekend respectively. The same analysis has been done for Tram in Fig-06,07 and we can see that line id-44 has the highest speed (29.6 km/hr, 29.39 km/hr) during weekdays and weekend respectively. In Fig 08, the comparison of speed between weekdays and weekend is given for more clarity.

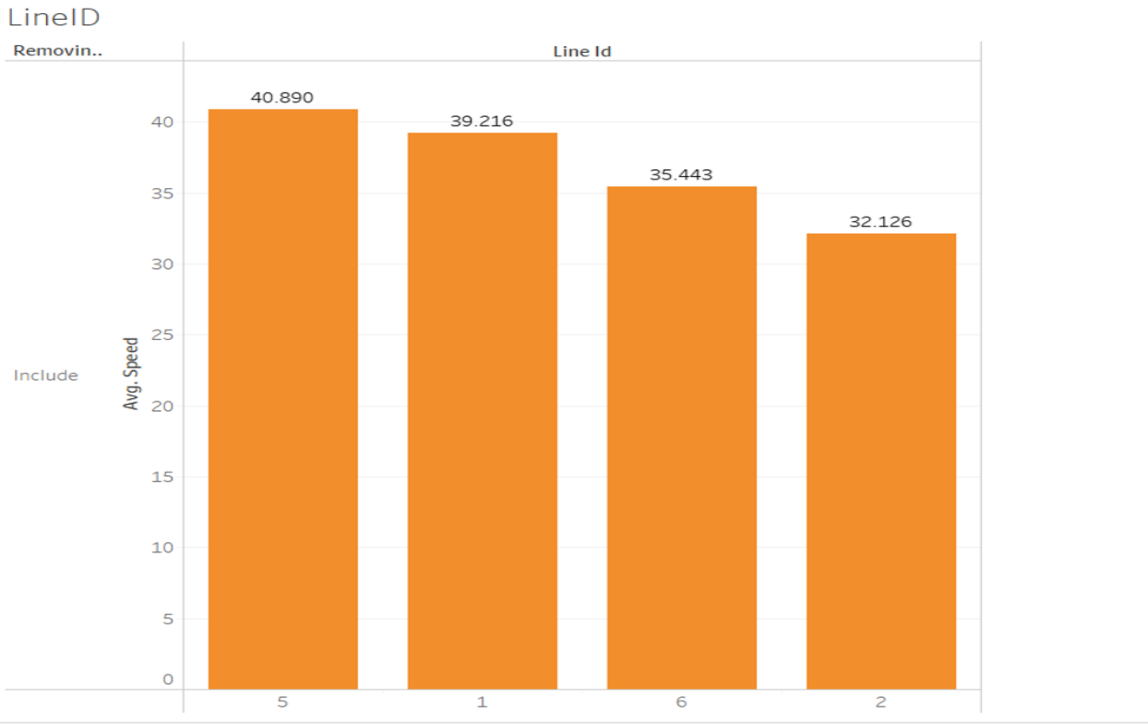


Fig-01 : Avg. Speed of metro for each line_ID (Weekdays)

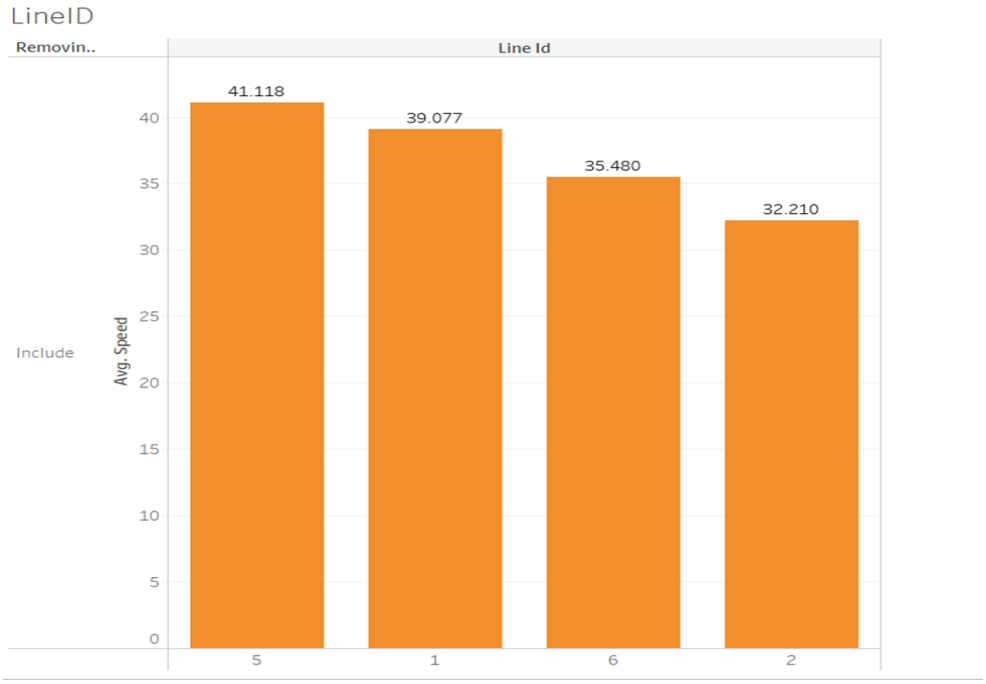


Fig-02 : Avg. speed of metro for each line_Id (Weekend)

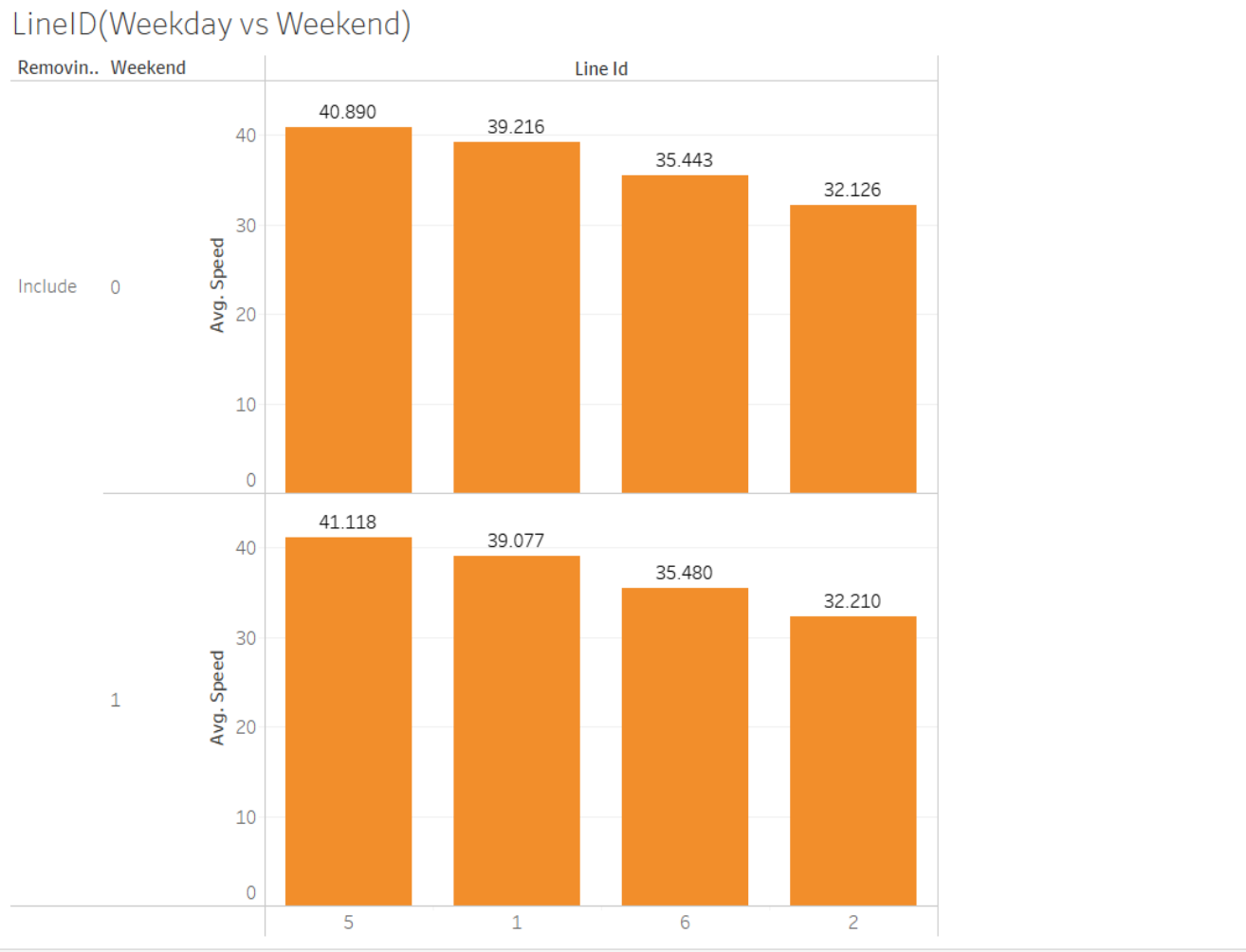


Fig-03 : Avg. speed of metro for each line id (Weekend-1,Weekday-0)

From this fig-03, it is shown that the average speed of the metro is slightly increased during weekends.

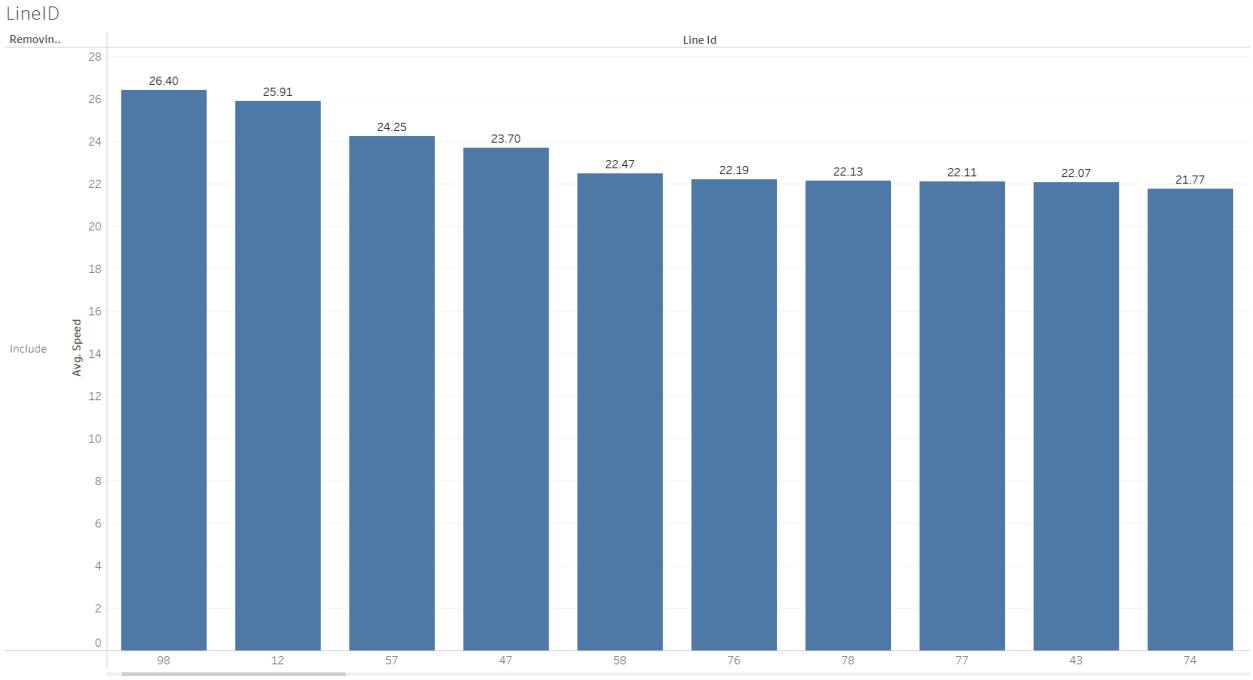


Fig-04 : Avg. Speed of bus for each line_ID (Weekdays)

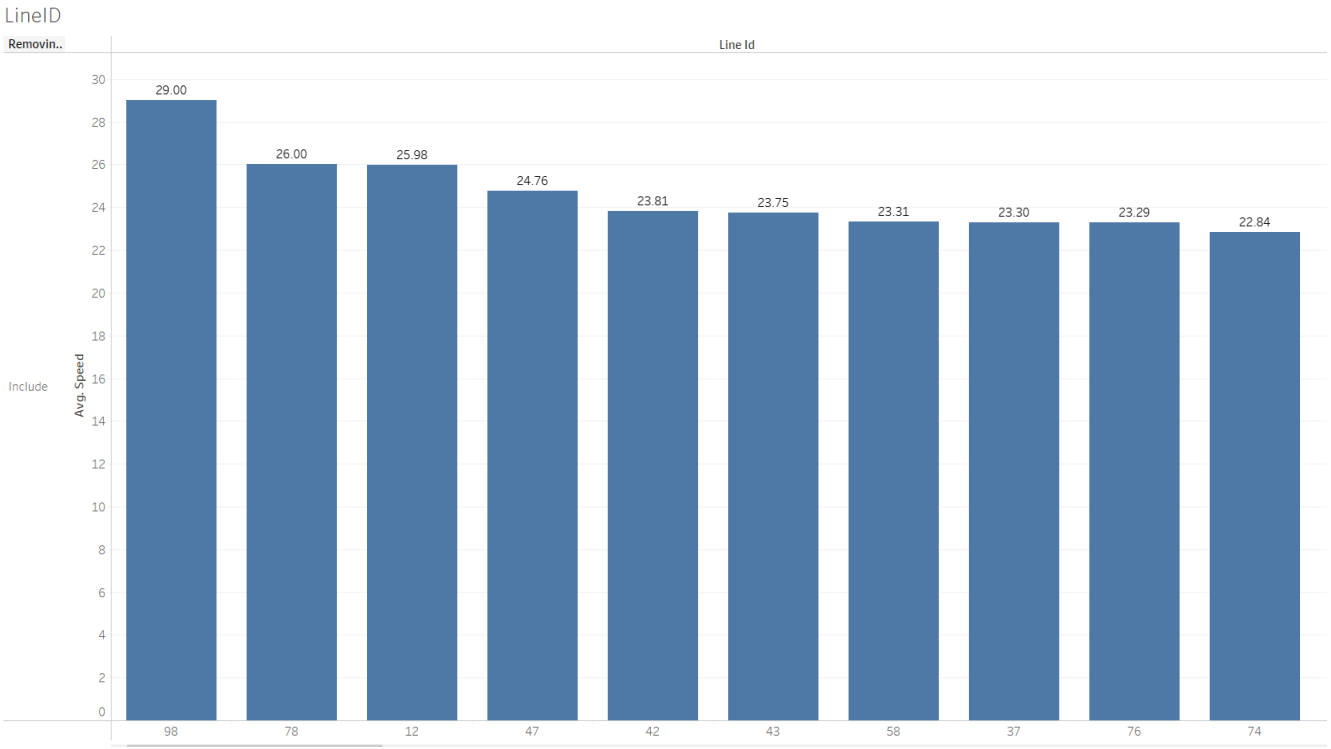


Fig-05 : Avg. Speed of bus for each line_ID (Weekend)

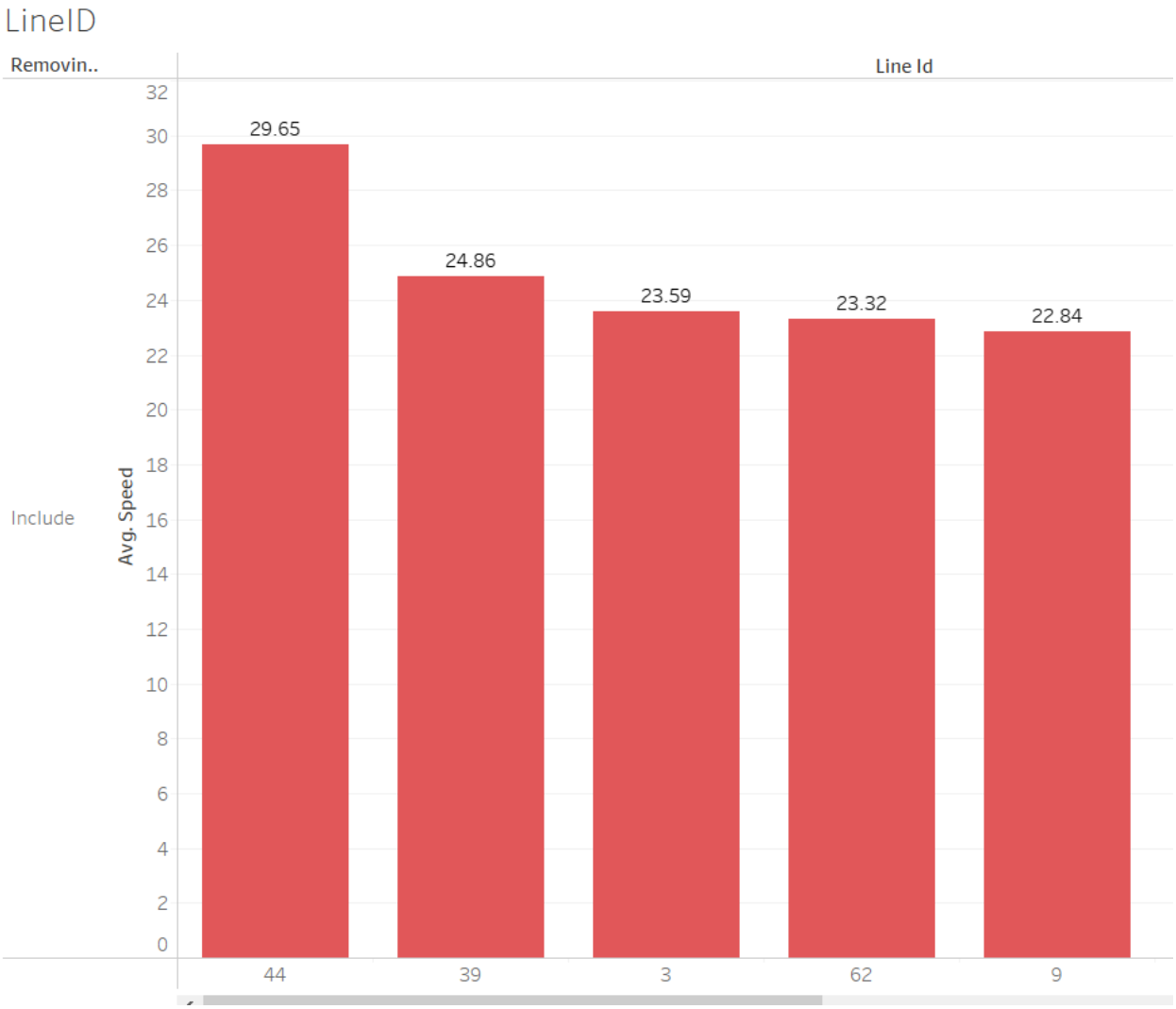


Fig-06 : Avg. Speed of tram for each line_ID (Weekdays)

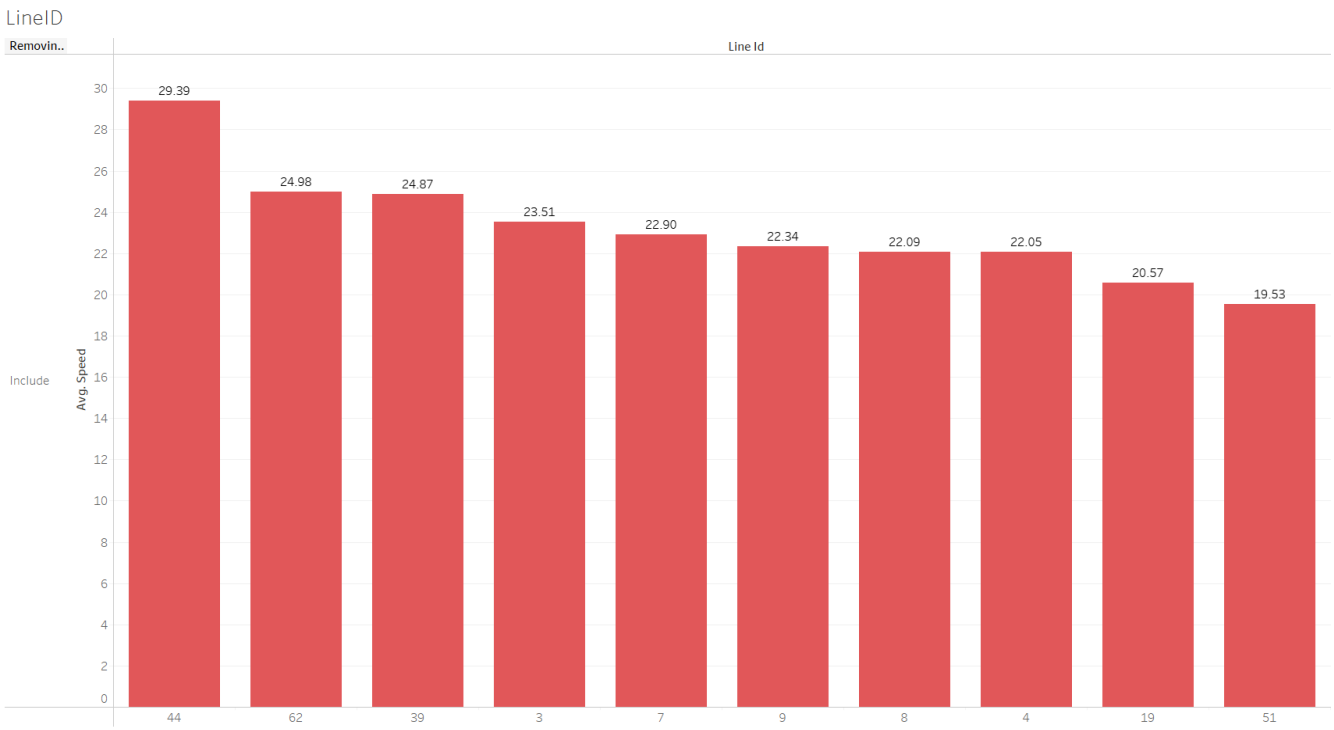


Fig-07 : Avg. Speed of tram for each line_ID (Weekend)

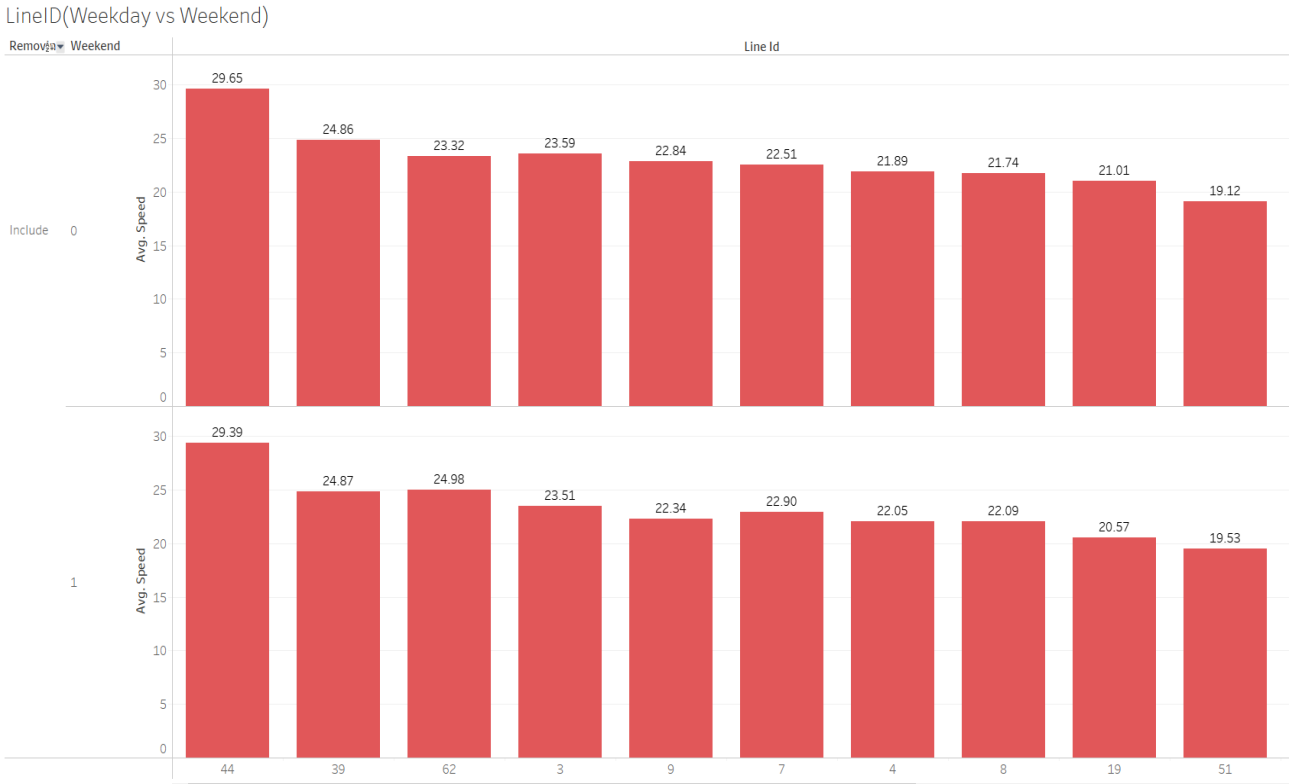


Fig-08 : Avg. Speed of tram for each line_ID (Weekdays-0,Weekend-1)

2. **Avg. Speed of all mode of transports across seven days**

After finding out the average speed of the metro, bus and tram during weekdays and weekends, we wanted to know which day the speed is highest and which day has the lowest speed. To get this information, we sort the speed in descending order for all seven days. And, we have found out that on Saturday the metro speed is the highest and on Wednesday, it has the lowest speed throughout the day [Fig-09]. For bus and tram, the speed is highest on Sunday.

heatmap (days/Highest Avg.speed)

Mode	Day Of Week	Removing 0 speed Include
M	Saturday	37.06
	Friday	37.00
	Thursday	36.96
	Tuesday	36.92
	Monday	36.90
	Sunday	36.88
	Wednesday	36.87
T	Sunday	21.05
	Wednesday	20.67
	Saturday	20.66
	Tuesday	20.66
	Thursday	20.65
	Friday	20.59
	Monday	20.54
B	Sunday	19.60
	Saturday	19.16
	Monday	18.85
	Wednesday	18.79
	Thursday	18.71
	Tuesday	18.64
	Friday	18.48

Fig-09 : Avg. Speed of all mode for all seven days

3. **Avg. Speed of transports (Metro, Tram and Bus) during different time of the day(Morning, Afternoon, Evening, Night and Late Night)**

To get more insight about the speed during time of the day, we try to plot the speed in a line chart to see how the speed varies from morning to late night for all three modes of transport. In fig-10, we can see that the speed of the metro starts low in the morning, and it picks the highest speed

during night. In fig-11,12 shows how tram and bus both have the highest speed during night and lowest speed during afternoon.

Fig-10 : Avg. Speed of metro for all session

Fig-11 : Avg. Speed of Tram for all session

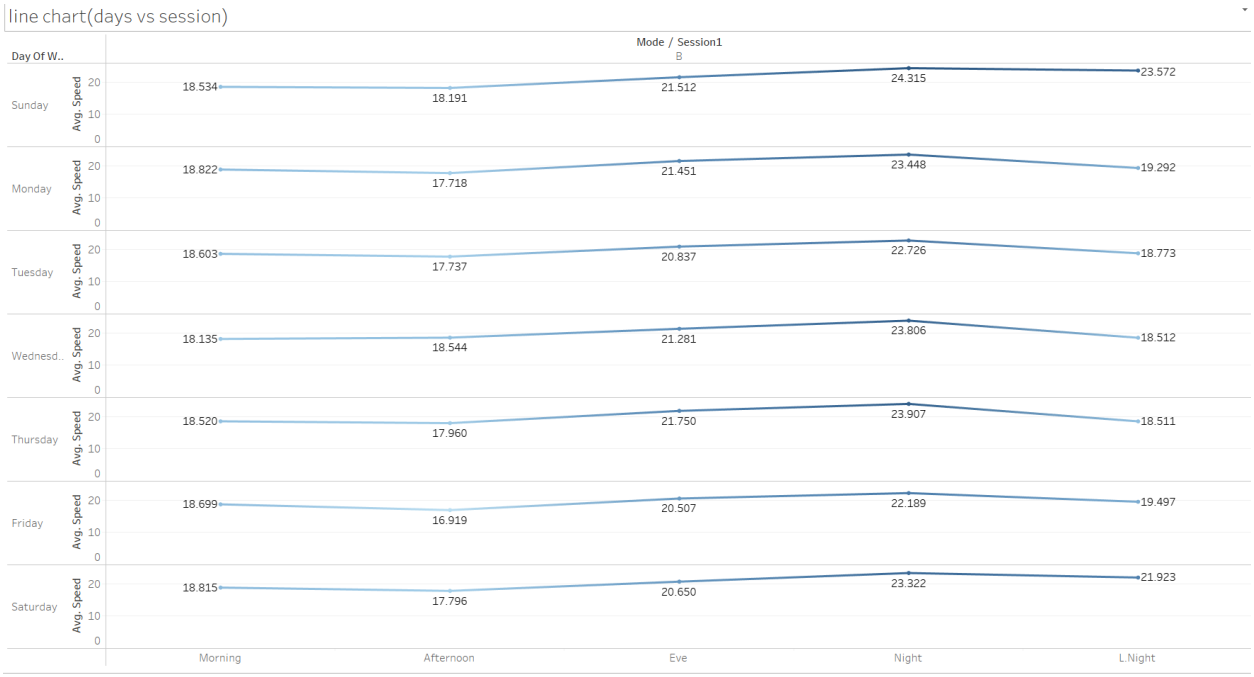


Fig-11 : Avg. Speed of bus for all session

4. Avg. speed for all three mode of transport during 24 hours

We have already found out which session of the day contains the highest, moderate and lowest speed for metro, bus and tram for all line_ID. To get the granularity of the data, we wanted to check how the speed is changing in 24 hours so that we can have an idea about which hour is the busiest for the transports and which hour is less busy. In fig-12, it shows how the speed is increasing during night and late night and mostly at moderate speed from morning to evening.

heatmap(hour vs avg.mode)

Time / Removing 0 speed

Mode	0 Include	1 Include	2 Include	3 Include	4 Include	5 Include	6 Include	7 Include	8 Include	9 Include	10 Include	11 Include	12 Include	13 Include	14 Include	15 Include	16 Include	17 Include	18 Include	19 Include	20 Include	21 Include	22 Include	23 Include	Avg. Al..
B	20.03	23.87	22.57	22.83	21.76	19.37	17.21	18.85	19.00	18.85	18.44	18.27	18.21	17.22	17.26	17.69	18.06	19.11	19.70	21.21	22.18	22.18	23.25	23.76	20.04
M			24.04	38.32	37.29	36.74	36.76	36.83	37.12	36.82	36.77	36.71	36.71	36.73	36.96	36.91	36.86	37.14	36.93	36.81	36.97	37.61	38.07	39.51	36.57
T			22.33	22.92	21.94	21.04	20.10	20.43	20.35	20.68	20.56	20.30	20.26	19.91	20.02	20.44	20.11	20.56	20.69	21.79	22.36	21.61	22.16	21.11	20.98
Avg.ALL Mode	20.03	23.87	22.98	28.02	27.00	25.72	24.69	25.37	25.49	25.45	25.26	25.09	25.06	24.62	24.75	25.01	25.01	25.61	25.77	26.60	27.17	27.13	27.83	28.12	25.69

Fig-12 : Avg. Speed of all mode for 24 hour

5. Avg. Speed for each segment across the line id (Metro, Bus, Tram)

Since we have already found out which line ID has the highest speed for each mode of transport, in this section we will demonstrate the top five segments for both directions in terms of highest avg. speed for line 5(metro), line 12 (bus), line id 44 (tram) in fig-13,14,15.

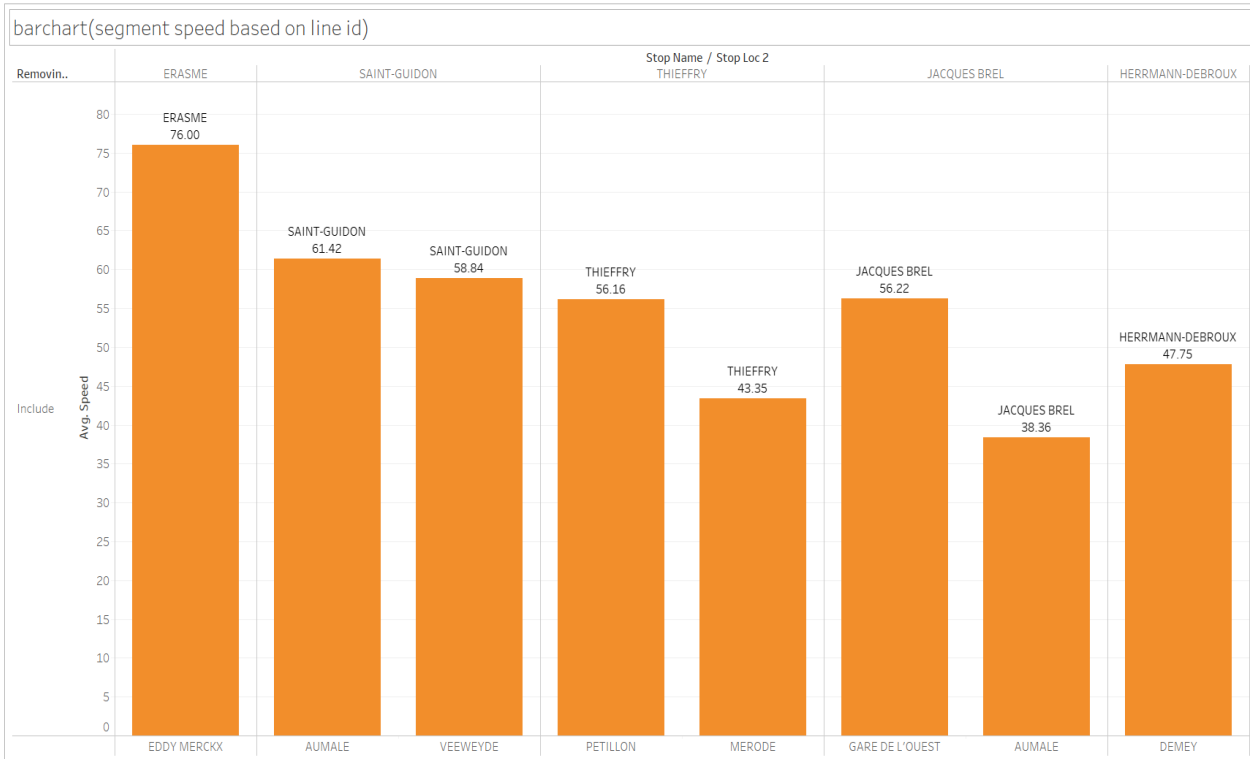


Fig-13 : Avg. Speed of top 5 segments for line_Id-05 (Metro)

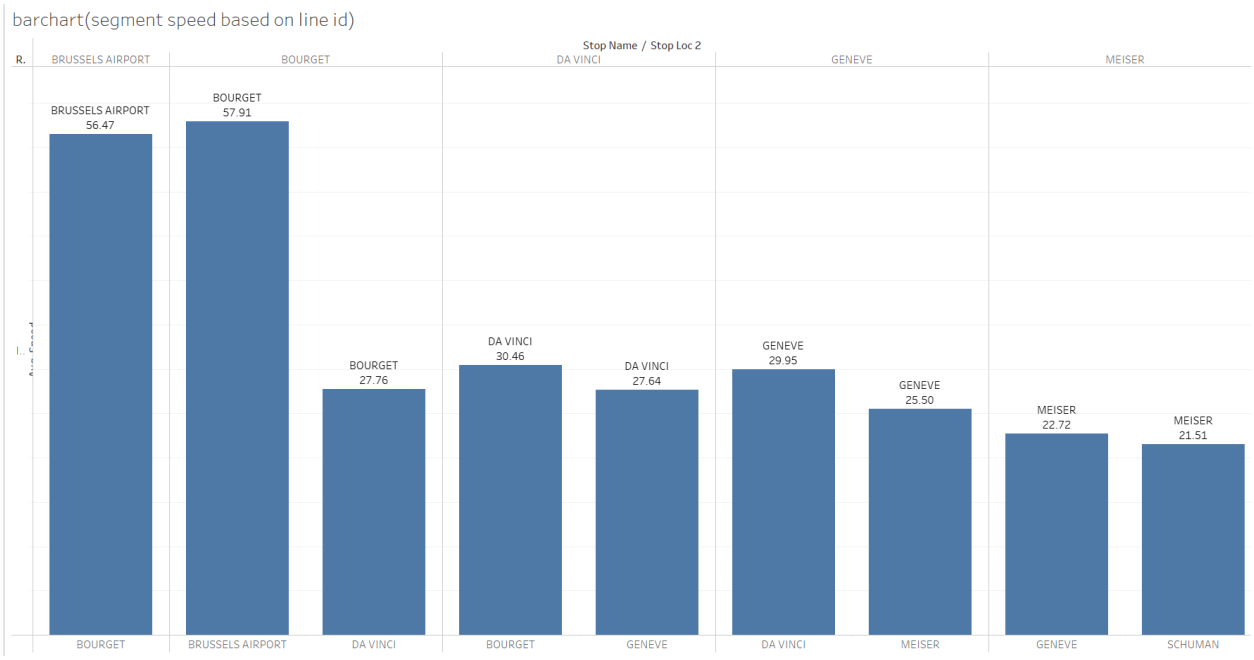


Fig-14 : Avg. Speed of top 5 segments for line_Id-12 (Bus)

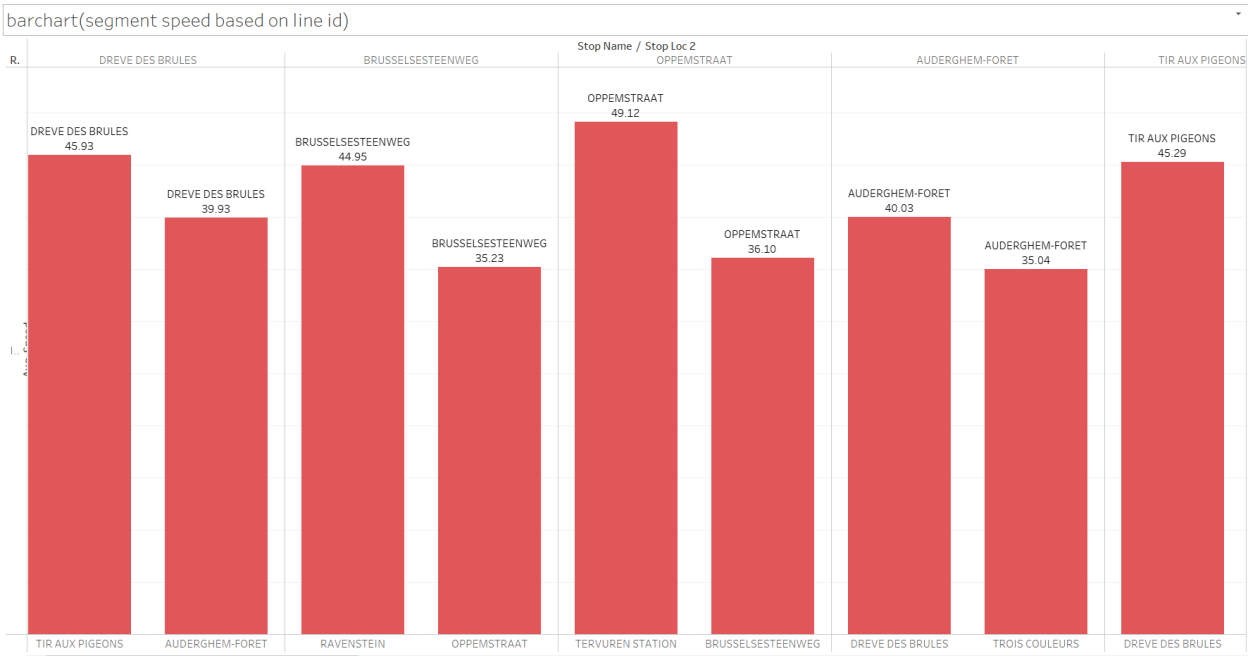


Fig-15 : Avg. Speed of top 5 segments for line_Id-44 (Tram)

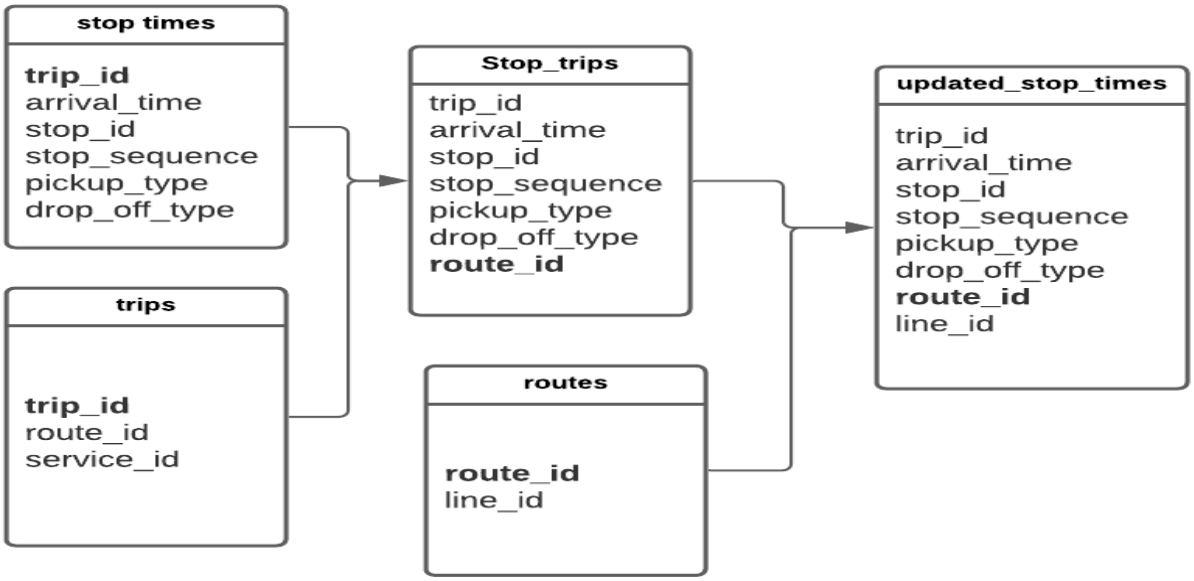
6. **Avg. Speed of all line id for each mode of transport**

We also use the Kepler tool to show the line id of every mode of transport for better clarity. Here in fig-16, we can see how the speed is varying across the lines for different transports. The color yellow denotes the bus lines, the pink color denotes the metro and the blue color denotes the tram.

Fig-16 : Avg. Speed of metro,tram and bus for all line id

3. Delay Analysis

3.1 Data Preparation



The standard schedule time in the provided stop_time.txt file only contains the trip_id, time and stop_id. But for the calculation we need to get the line_Id of each stop_id. Therefore we made a join operation among routes.txt and trip.txt with trip_id to get the route_Id for each trip. Then we will join with the route id to get all the required information in the stop_times.txt file. Initially the file looks like this.

```
trip_id,arrival_time,departure_time,stop_id,stop_sequence,pickup_type,drop_off_type
112387248235954071,21:07:00,21:07:00,4014,1,0,0
112387248235954071,21:09:00,21:09:00,3231,2,0,0
112387248235954071,21:10:08,21:10:08,3232,3,0,0
112387248235954071,21:11:00,21:11:00,3233,4,0,0
112387248235954071,21:11:43,21:11:43,3239,5,0,0
112387248235954071,21:12:54,21:12:54,3235,6,0,0
112387248235954071,21:13:51,21:13:51,3236,7,0,0
112387248235954071,21:15:11,21:15:11,4653,8,0,0
112387248235954071,21:16:00,21:16:00,4655,9,0,0
112387248235954071,21:17:11,21:17:11,4656,10,0,0
112387248235954071,21:18:26,21:18:26,4657,11,0,0
112387248235954071,21:19:30,21:19:30,4661B,12,0,0
112387248235954071,21:20:11,21:20:11,1193,13,0,0
112387248235954071,21:21:00,21:21:00,1195,14,0,0
112387248235954071,21:23:00,21:23:00,1196,15,0,0
112387248235954071,21:24:46,21:24:46,4059,16,0,0
112387248235954071,21:26:00,21:26:00,4010,17,0,0
112387248235954071,21:27:07,21:27:07,4062,18,0,0
112387248235954071,21:28:00,21:28:00,4101,19,0,0
```

But after the transformation it will become like the following :

service_id	arrival_time	stop_id	route_id	route_short_name
235954071	21:07:00	4014	24	87
235954071	21:09:00	3231	24	87
235954071	21:10:08	3232	24	87
235954071	21:11:00	3233	24	87
235954071	21:11:43	3239	24	87
235954071	21:12:54	3235	24	87
235954071	21:13:51	3236	24	87
235954071	21:15:11	4653	24	87
235954071	21:16:00	4655	24	87
235954071	21:17:11	4656	24	87
235954071	21:18:26	4657	24	87
235954071	21:19:30	4661	24	87
235954071	21:20:11	1193	24	87
235954071	21:21:00	1195	24	87
235954071	21:23:00	1196	24	87
235954071	21:24:46	4059	24	87
235954071	21:26:00	4010	24	87

Code snippet:

```
pd.read_csv("./Data/"+gtfs+"/stop_times.txt").filter(items=['trip_id', 'arrival_time','stop_id',
'stop_sequence'])

df_route=pd.read_csv("./Data/"+gtfs+"/routes.txt")

df_trip=pd.read_csv("./Data/"+gtfs+"/trips.txt")

df_time_trip=(pd.merge(df_time,df_trip,on='trip_id',how='inner'))
df_time_route=(pd.merge(df_time_trip,df_route,on='route_id',how='inner'))
df_time_route=df_time_route.filter(items=['trip_id','arrival_time','stop_id','route_id',
'route_short_name'])
df_time_route.info()

df_time_route['trip_id'] = df_time_route.astype({"trip_id":'str'})
df_time_route['trip_id'] = df_time_route['trip_id'].str[9:]
df_time_route['stop_id'] = df_time_route['stop_id'].str[:4]
df_time_route=df_time_route.dropna()
df_time_route = df_time_route.astype({"stop_id":'int64'})
df_time_route['trip_id'] = df_time_route.astype({"trip_id":'int64'})

df_time_route=df_time_route.rename(columns={"trip_id": "service_id"})

df_time_route.info(verbose=True, null_counts=True)
tm_lst=df_time_route['arrival_time'].tolist()
new_lst=[]
for i in tm_lst:
    splt=i.split(':')
    if splt[0]=='24' or splt[0]=='25':
        splt[0]='00'
    new_i=splt[0]+':'+splt[1]+':'+splt[2]
    new_lst.append(new_i)

df_time_route['arrival_time']=new_lst
df_time_route.to_csv('./Data/'+gtfs+'/updated_stop_times.csv',index=False)
```

3.1.1 Delay data calculation:

Algorithm :

1) First of all we will keep the data of only those service id those who operate on that day following the calendar.txt.
2) Then among the dates we will match with the line id and stop id of which we are going to calculate the delay time. We will get a lot of candidate dates.
3) Next we will try to match with the schedule which has the least difference.
4) As we are taking absolute differences in seconds it could be the case that both the bus could be late or early. To address this issue we will check the value of the reference one and the candidate ones and if it is less than 90 sec and the reference value is less than the candidate one it is early otherwise it is late.

Code snippet:

```
for stop,time_2,line,distance,speed in
 zip(stop_lst,arrive_time_lst,line_id_lst,distance_lst,speed_lst):
    least_time=10000000
    is_late=1
    schedule_list=((df_time.loc[(df_time['stop_id']==stop) & (df_time[
'route_short_name']==line)]).filter(items=['arrival_time']))['arrival_time'].tolist()

    if ((distance>20) & (speed!=0)):
        extra_time=distance/speed
        #print(extra_time," ",distance," ",speed,"new" )
        time_2=time_2-timedelta(seconds=extra_time)
    #print(stop,line,sequence)
    match_time=None
    #print(len(schedule_list))
    tmp_lst=[]
    for time_1 in schedule_list:

        if time_2>time_1:
            time_diff=(time_2-time_1).seconds
            tmp_is_late=1
        else:
            time_diff=(time_1-time_2).seconds

            tmp_is_late=0 #the vehicle is early
        if ((time_diff<least_time) | ((tmp_is_late==0) & (time_diff<60))):
            #print(time_diff)
            least_time=time_diff
            match_time=time_1
            is_late=tmp_is_late
    delay_time_lst.append(least_time)
    flag_lst.append(is_late)
    match_time_list.append(match_time)
```

3.2 Visualization

In this section, we are going to explain our delay analysis for all stops in Brussels for all the three modes of transport i.e. metro, tram and bus varying across different line_ID, during weekdays and weekends. We are also going to show how delay is changing from morning to night to get the idea of average delay during peak hour and non-peak hours. We will demonstrate our findings by giving a graphical view (Barchart, geospatial-view and heatmap) of the delay for these three modes of transport.

Visualization Tool used :

- ❖ Tableau
- ❖ Kepler.gl

1. Average Delay in seconds per LineID

From the given data, we analysed the delay for each LineID for all modes of transport - Bus, Tram and Metro.

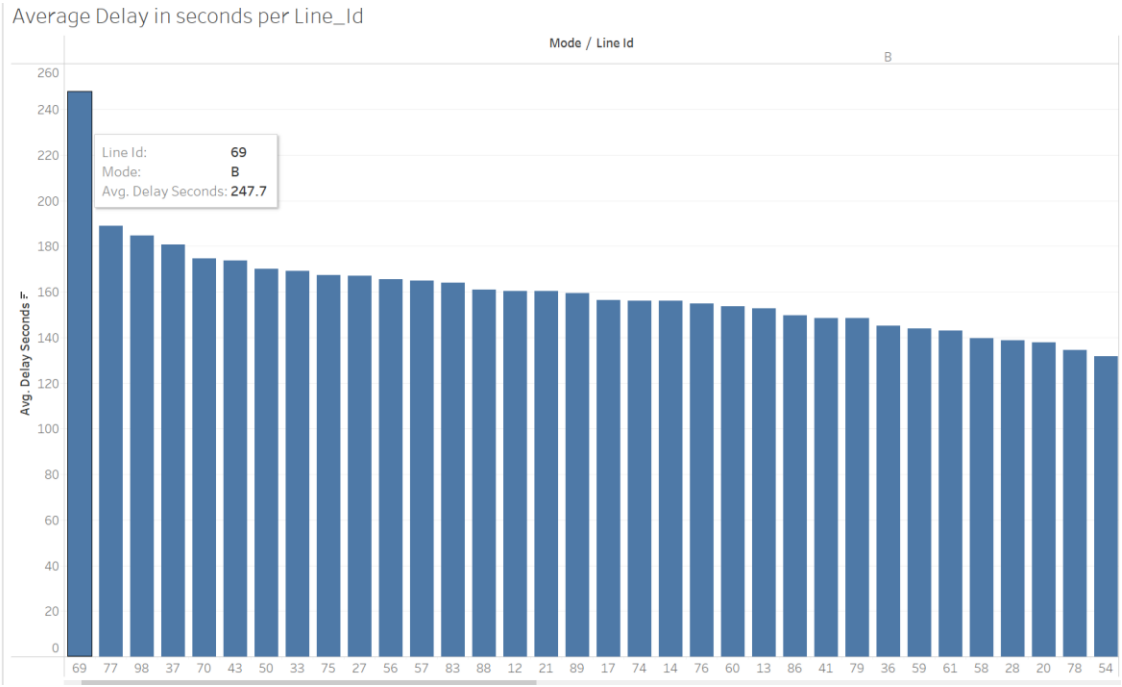


Fig.4.1. Average delay in seconds per LineId for Bus

The above figure displays the average delay for Bus on each LineId, and shows that the LineId - 69 has the highest delay of 247.7 seconds for Bus. The top five Line_Id which show the highest delays are - LineId 69, LineId 77, LineId 98, LineId 37, and LineId 70.

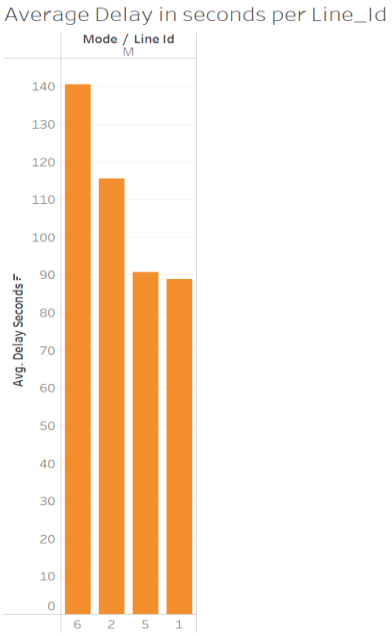


Fig.4.2. Average delay in seconds per LineId for Metro

The above figure displays the average delay for Metro on each LineId. From the above figure it can be inferred that the highest delay is found on LineId 6 which is 140.58 sec. Followed by Line Id 2, LineId 5 and LineId 1.

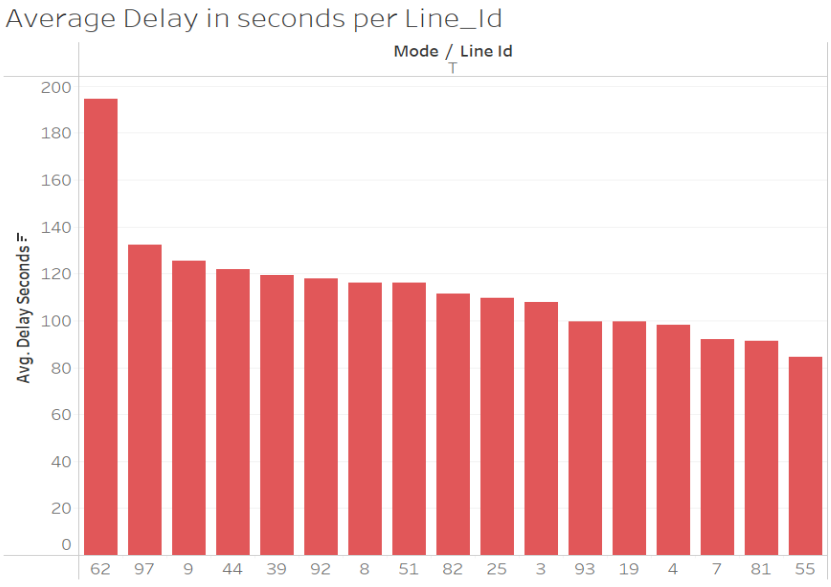


Fig.4.3. Average delay in seconds per LineId for Tram

The above figure displays the average delay for Trams on each LineId. From the above figure it can be inferred that the highest delay is found on LineId 62 which is 194.5 sec. Followed by Line Id 97, Line Id 9, LineId 44 and LineId 39.

Fig.4.4.Kepler.gl - Average delay on LineId

The above figure shows the average delay on each LineId for all modes of transport - Bus, Tram and Metro.

2. Average Delay per LineId w.r.t. Day of Week

From the given data, we analysed the delay for each LineID for all modes of transport - Bus, Tram and Metro for all days of the week.

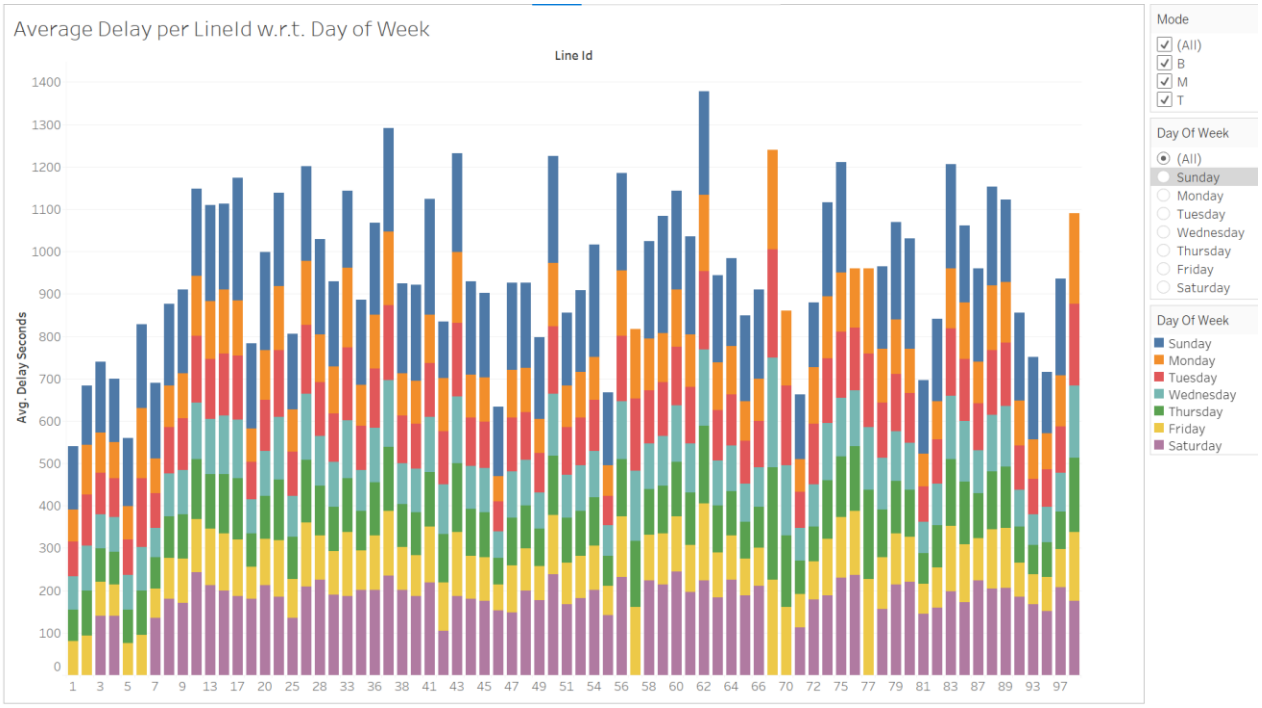


Fig.4.5. Average delay per LineId w.r.t. Day of Week

The above figure shows the average delay on each LineId on days of the week i.e. - Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday. It can be inferred from the above figure that LineId 62 shows the highest delay on Sunday. Plus it can be seen that the average delay on Sunday is highest followed by Monday, Tuesday, Wednesday, Thursday, Friday and Saturday.

3. Average Delay per Line_Id w.r.t. Day of Week - weekend/weekday

From the given data, we analysed the delay for each LineID for all modes of transport - Bus, Tram and Metro on weekdays and weekends.

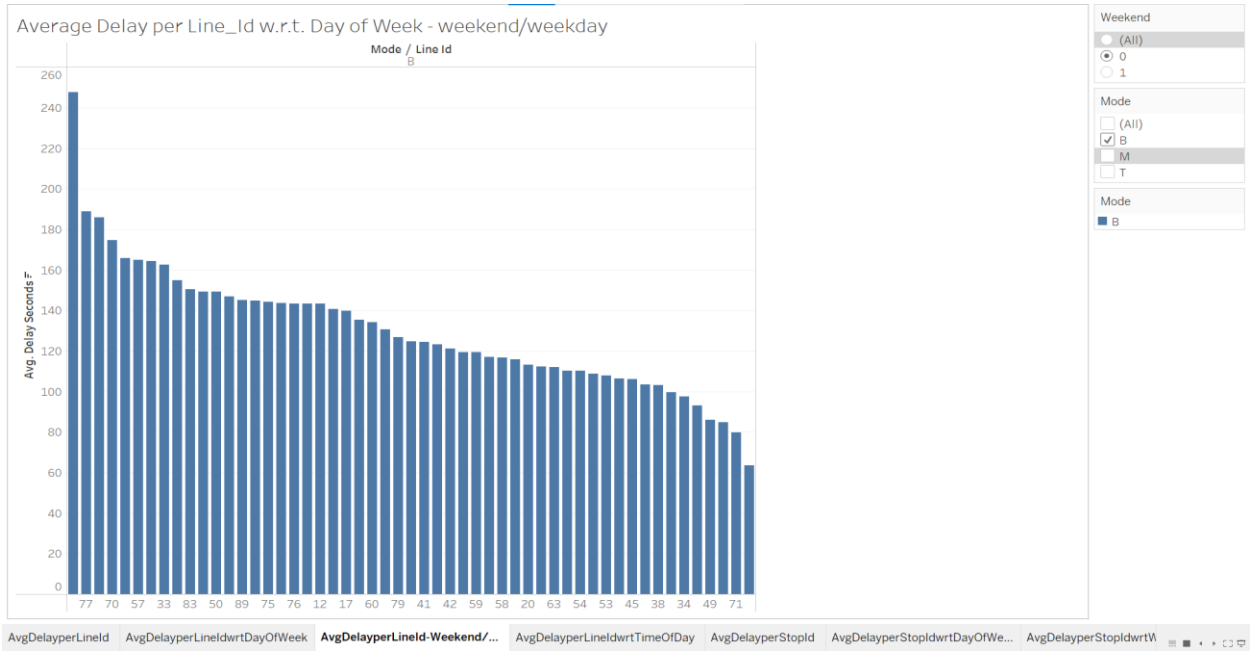


Fig.4.6.Average Delay per Line_Id w.r.t. Day of week - weekend/weekday for Bus on weekday

The above figure shows the average delay on weekdays and the highest delay of 247.7 sec is on LineId 69 for Bus.

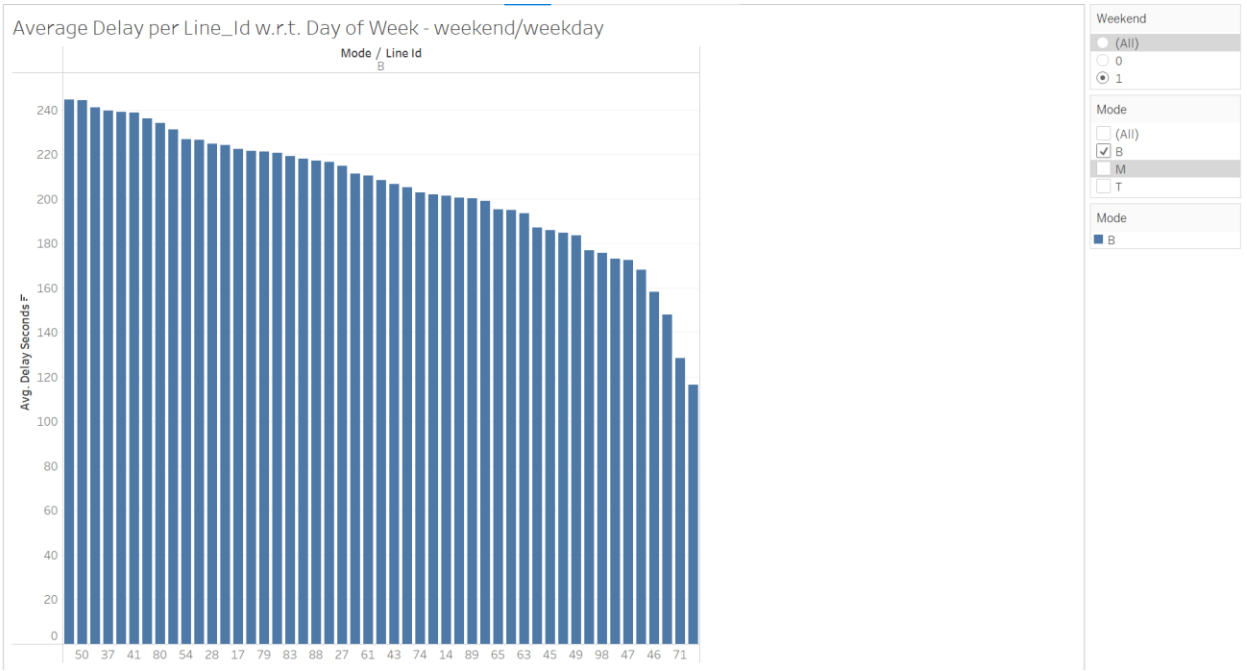


Fig.4.7.Average Delay per Line_Id w.r.t. Day of week - weekend/weekday for Bus on weekend

The above figure shows the average delay on weekends and the highest delay of 244.6 sec is on LineId 75 for Bus.
From the above two figures, we can see that the delay is higher for LineId's on weekends compared to weekdays.

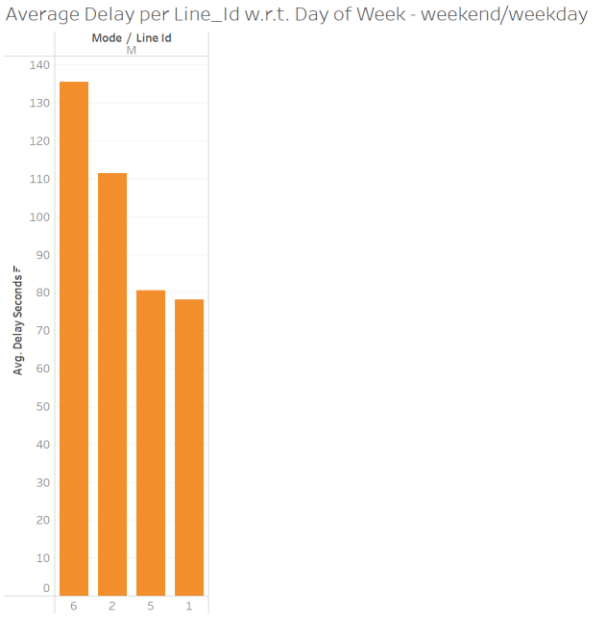


Fig.4.8.Average delay per LineId w.r.t. Day of week - weekend/weekday for Metro on weekday

The above figure shows the average delay on weekdays and the highest delay of 135.56 sec is on LineId 6 for Metro.

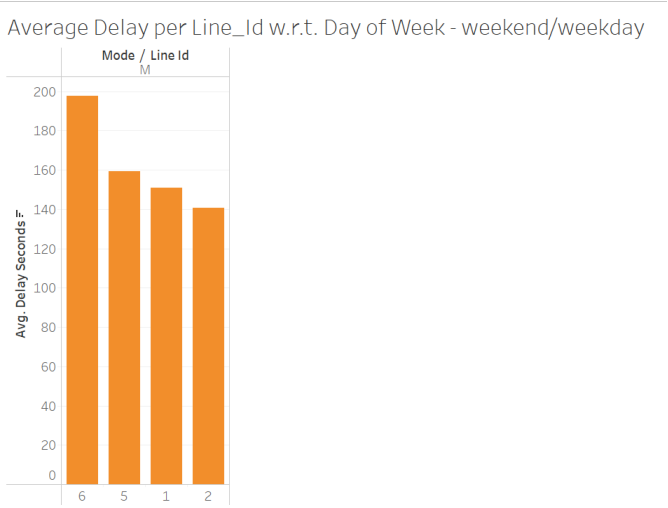


Fig.4.9.Average delay per LineId w.r.t. Day of week - weekend/weekday for Metro on weekend

The above figure shows the average delay on weekends and the highest delay of 197.70 sec is on LineId 6 for Metro.
From the above two figures, we can see that the delay is higher for LineId's on weekends compared to weekdays. Plus, LineId 6 is showing the highest delay on both weekdays and weekends.

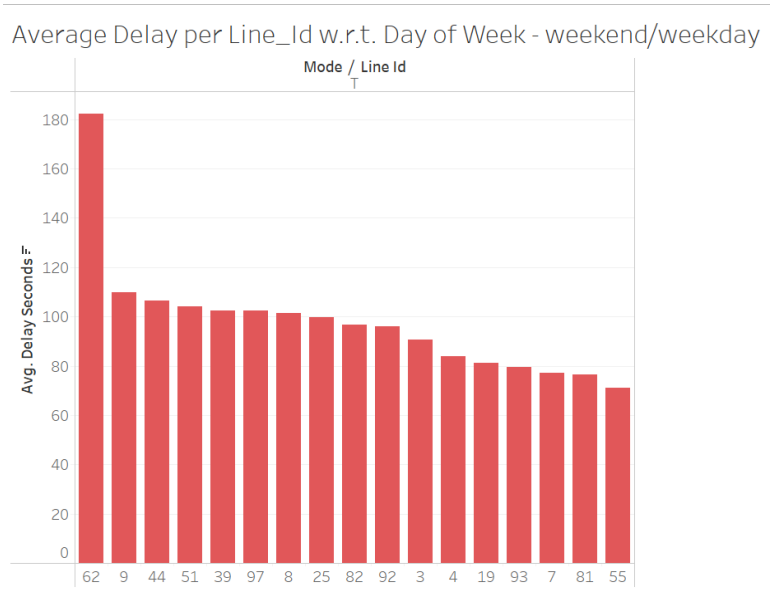


Fig.4.10.Average delay per LineId w.r.t. Day of week - weekend/weekday for Tram on weekday

The above figure shows the average delay on weekdays and the highest delay of 182.3 sec is on LineId 62 for Tram.

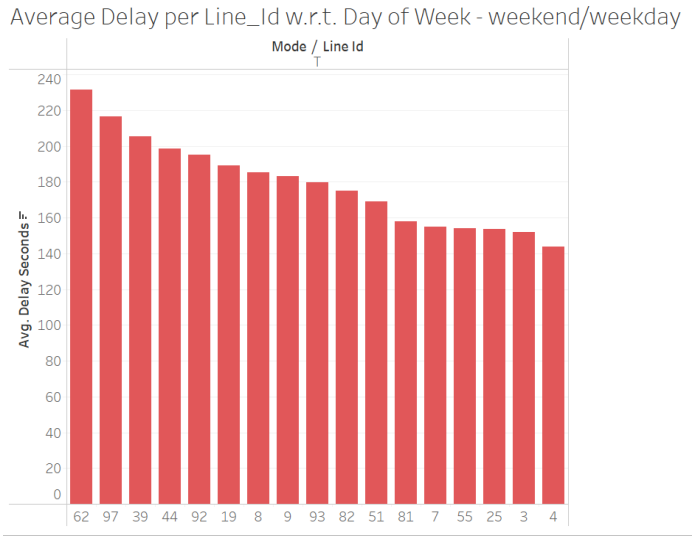


Fig.4.11.Average delay per LineId w.r.t. Day of week - weekend/weekday for Tram on weekend

The above figure shows the average delay on weekdays and the highest delay of 231.51 sec is on LineId 62 for Tram.
From the above two figures, we can see that the delay is higher for LineId's on weekends compared to weekdays. Plus, LineId 62 is showing the highest delay on both weekdays and weekends.

4. **Average Delay per LineId w.r.t. Time of the Day**

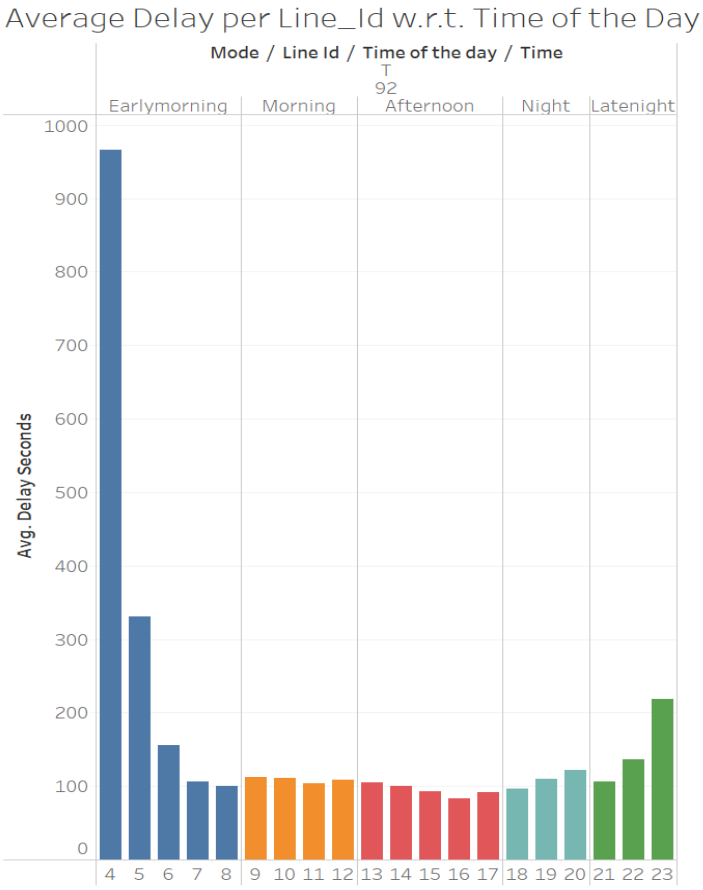


Fig.4.12.Average delay per LineId w.r.t. Time of Day

The above figure shows the average delay per LineId w.r.t. Time of the day. The time of the day is divided into - Early Morning, Morning, Afternoon, Evening, Night, Late Night and Midnight. Here we have displayed the LineId with the highest delay of 966 sec at 4AM in the early morning on LineId 92 for Tram. In this view, we come to know the trend of the delay on each LinId throughout the day from early morning to midnight.

5. Average Delay per Stop_Id

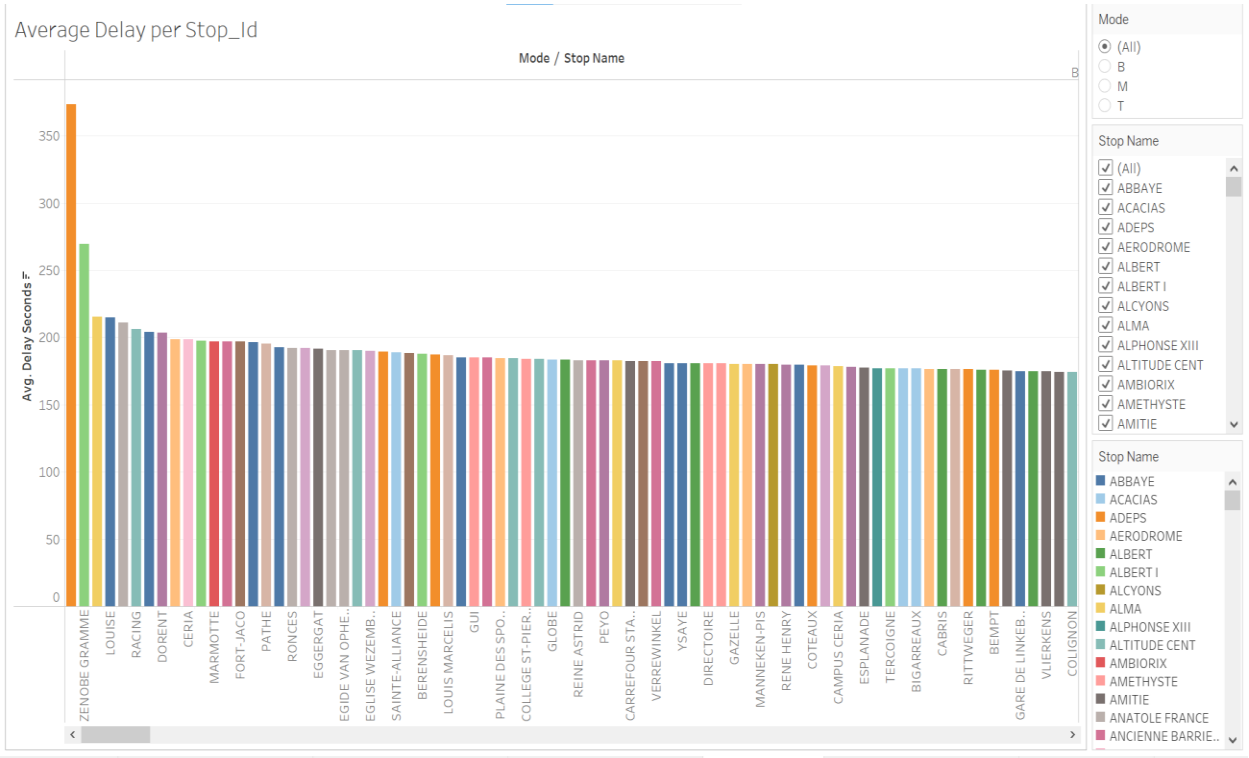


Fig.4.13.Average delay per Stop_Id

The above figure shows the average delay per StopId, we have sorted the data in descending order to find the stops with the highest delays. Stop name ‘Hoogveld’ has the highest delay of 373.1 sec for Bus.

As shown in the figure below, we even tried to plot the average delay at each stop in Kepler.gl, to represent it more visually. Here the average delay at each stop is show with the radius, as the delay increases so does the size of the radius. Hence, the below figure is quite intuitive and visually represents the delay at each stop.

Fig.4.14.Kepler.gl - Average delay at each StopId

6. Average delay per StopId w.r.t. Day of Week

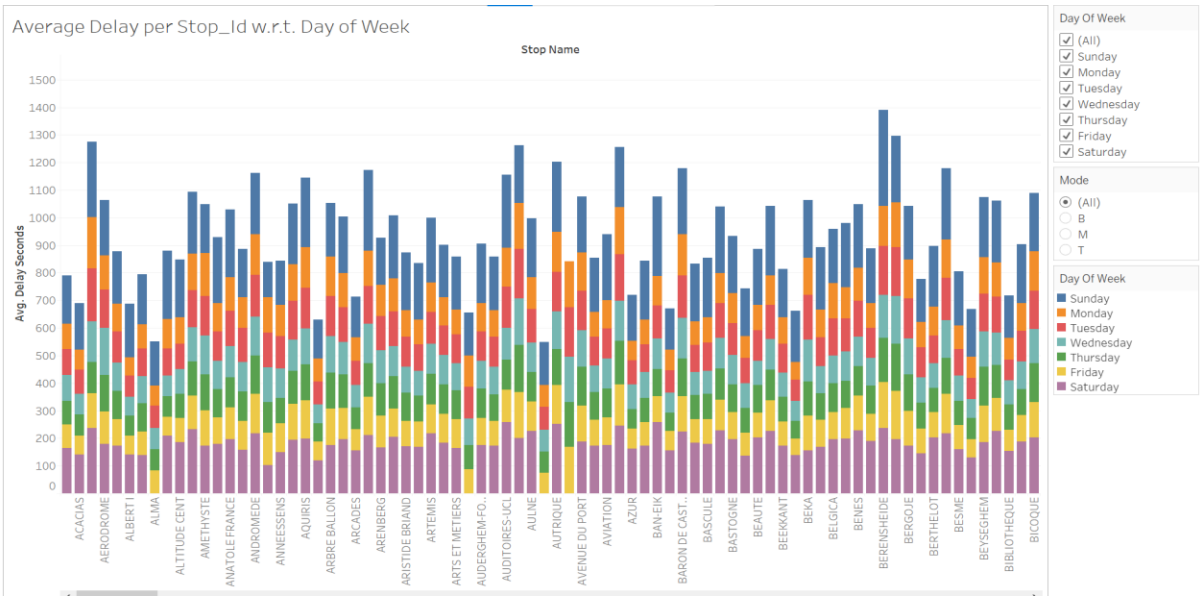


Fig.4.15.Average delay per StopId w.r.t. Day of week

The above figure shows the average delay for each StopId on days of the week i.e. - Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday. It can be inferred from the above

figure that the Stop Berenheide shows the highest delay on Sunday. Plus it can be seen that the average delay on Sunday is highest followed by Monday, Tuesday, Wednesday, Thursday, Friday and Saturday.

7. Average Delay per Stop_Id w.r.t. Day of Week - weekend/weekday

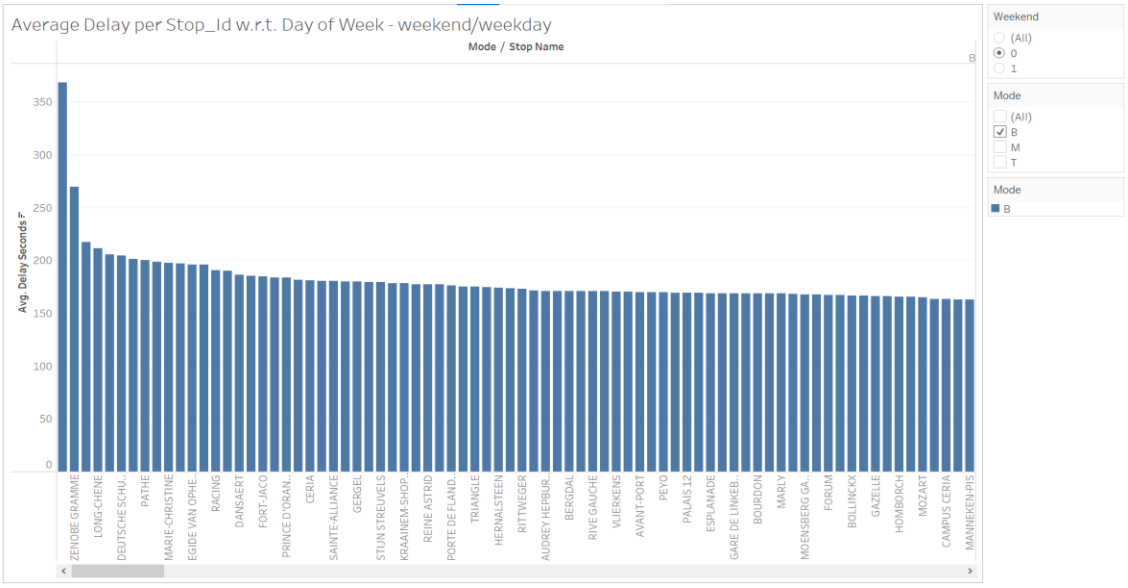


Fig.4.16.Average delay per Stop_Id w.r.t. Day of week- Bus on Weekday

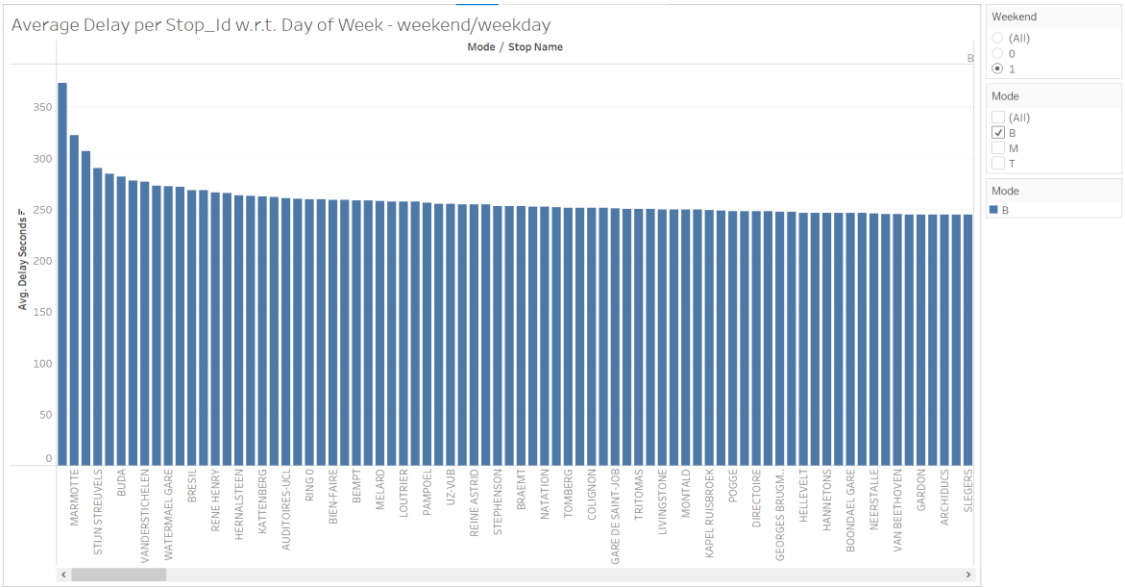


Fig.4.17.Average delay per Stop_Id w.r.t. Day of week- Bus on Weekend

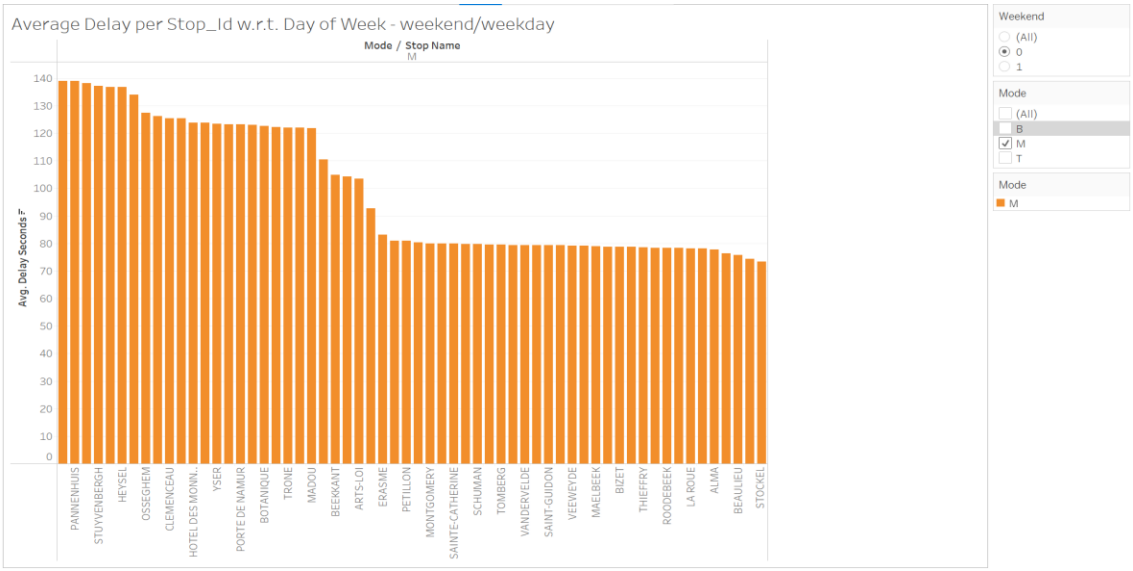


Fig.4.18.Average delay per Stop_Id w.r.t. Day of week- Metro on Weekday

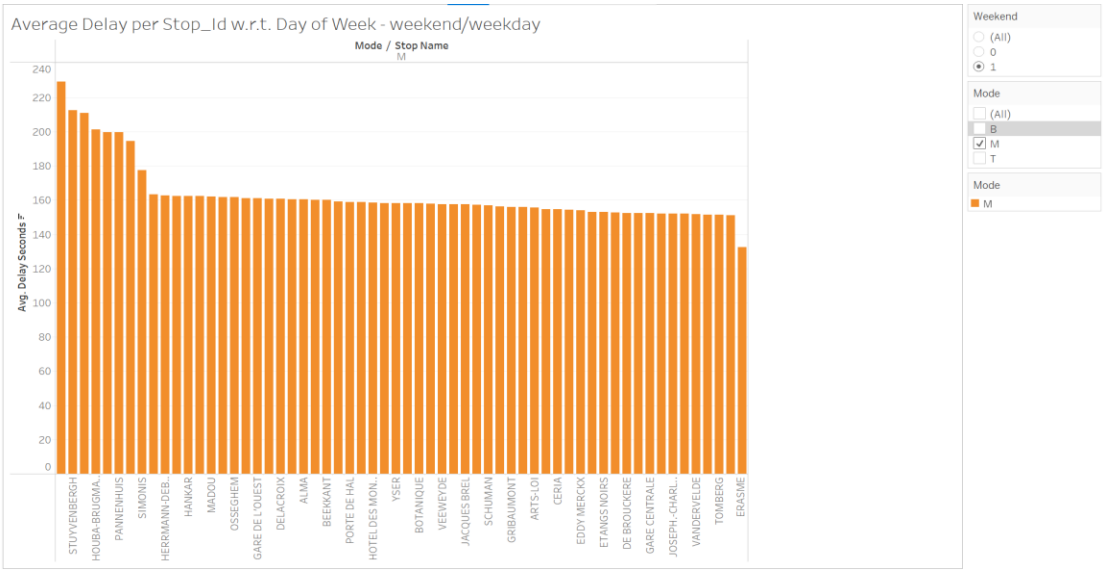


Fig.4.19.Average delay per Stop_Id w.r.t. Day of week- Metro on Weekend

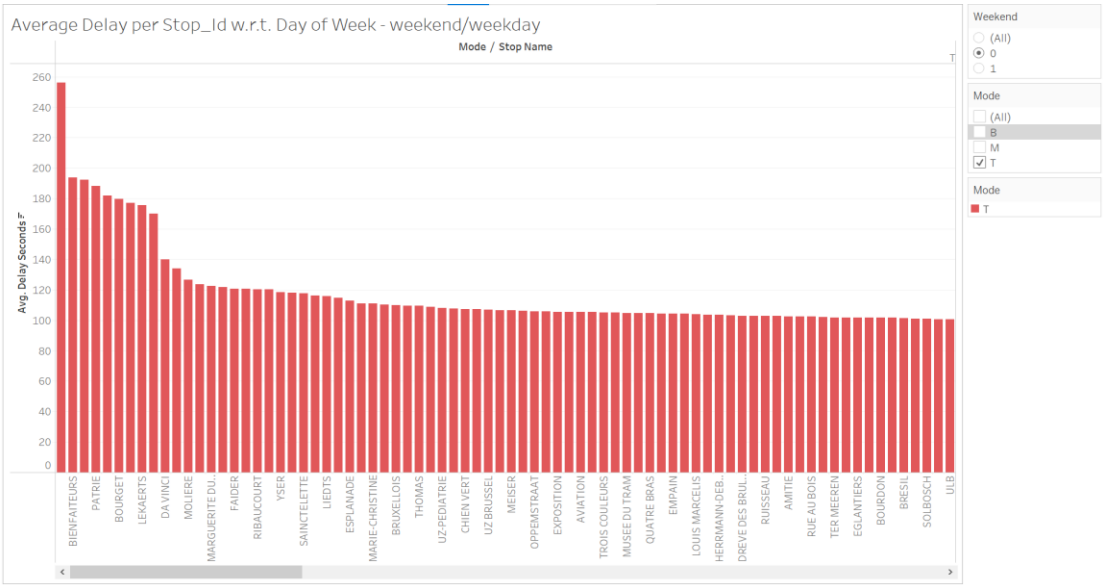


Fig.4.20.Average delay per Stop_Id w.r.t. Day of week- Tram on Weekday

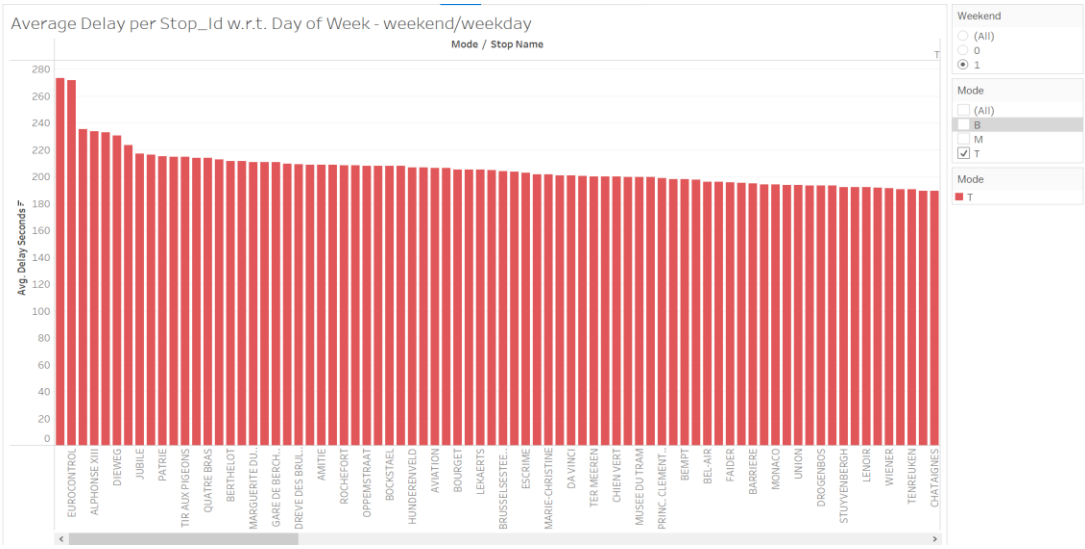


Fig.4.21.Average delay per Stop_Id w.r.t. Day of week- Tram on Weekend

8. Average Delay per StopId w.r.t. Time of the Day

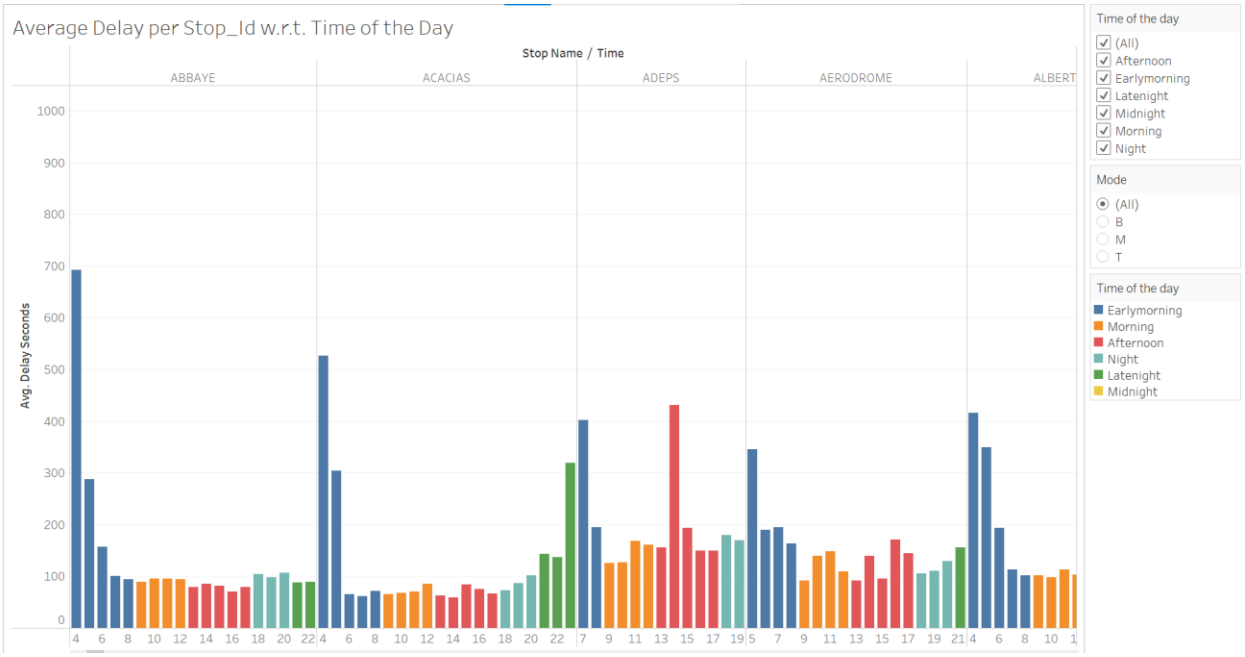


Fig.4.22.Average delay per StopId w.r.t. Time of the Day

The above figure shows the average delay at each stop across time, where time is divided into - Early morning, Morning, Afternoon, Night, Late Night and Midnight.

4. Arrival Time Prediction

4.1 Time Series

Time-series forecasting models are those models which are capable of predicting the future values which are based on previously observed values. Time-series forecasting is generally used for non-stationary data, where means and variance vary over time. It seems some events are repetitive over time. In time series it is captured in terms of seasonality and trend analysis. Time series analysis is the use of statistical methods to analyze time series data and extract meaningful statistics and characteristics about the data. Time series analysis helps identify trends, cycles, and seasonal variances to aid in the forecasting of a future event. Factors relevant to time series analysis include stationarity, seasonality and autocorrelation.

Time series analysis can be useful to see how a given variable changes over time (while time itself, in time series data, is often the independent variable). Time series analysis can also be used to examine how the changes associated with the chosen data point compare to shifts in other variables over the same time period.

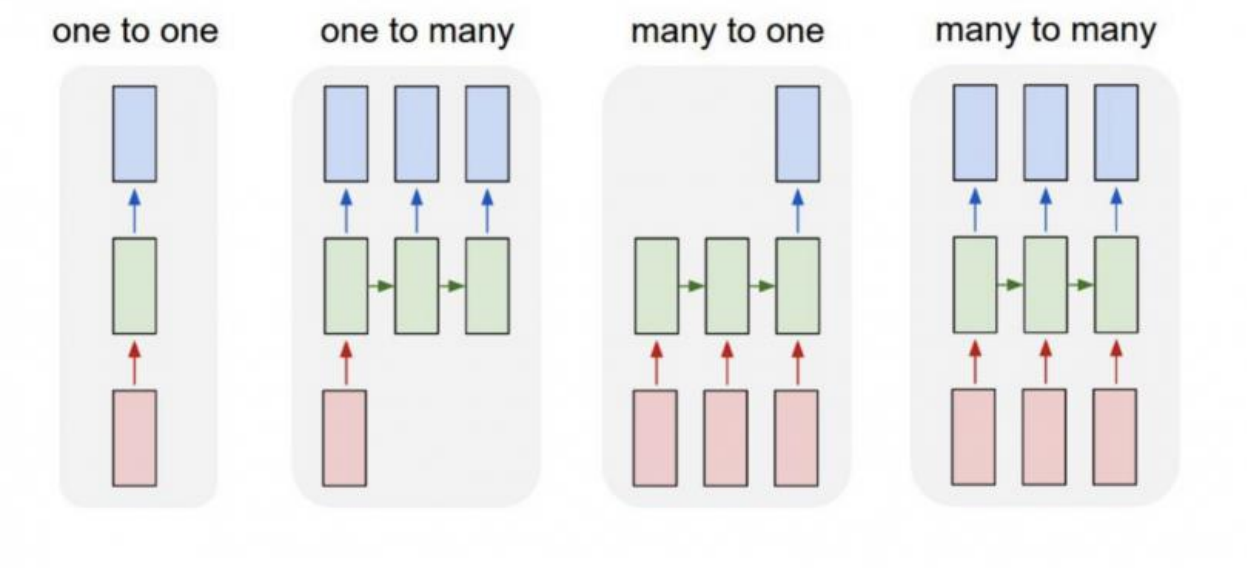
Why LSTM ?

1. Able to capture complex patterns from the Data
2. Only a Single Model is needed to predict arrival time for every segment, for each line ID.
3. Deployment friendly
4. Easy to retrain and extend once developed

4.2 Long Short Term Memory (LSTM)

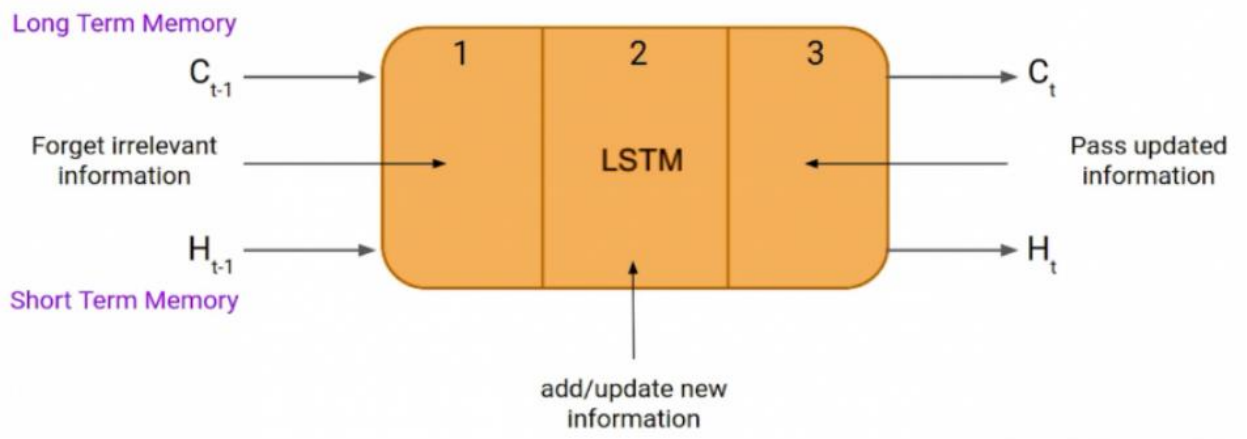
Sequence modeling is a task of predicting the next sequence based on the features provided to the data as well as the previous occurred sequence. So the output not only depends on the current input but also on the previous inputs as well.

Neural Network - Neural Networks are the combination of Nodes which comes from the same intuition as human brains, where the nodes are interconnected with each other to handle complex non-linear relationships. Based on the loss functions the model weights gets updated and it keeps updating till the model reaches the global minima.

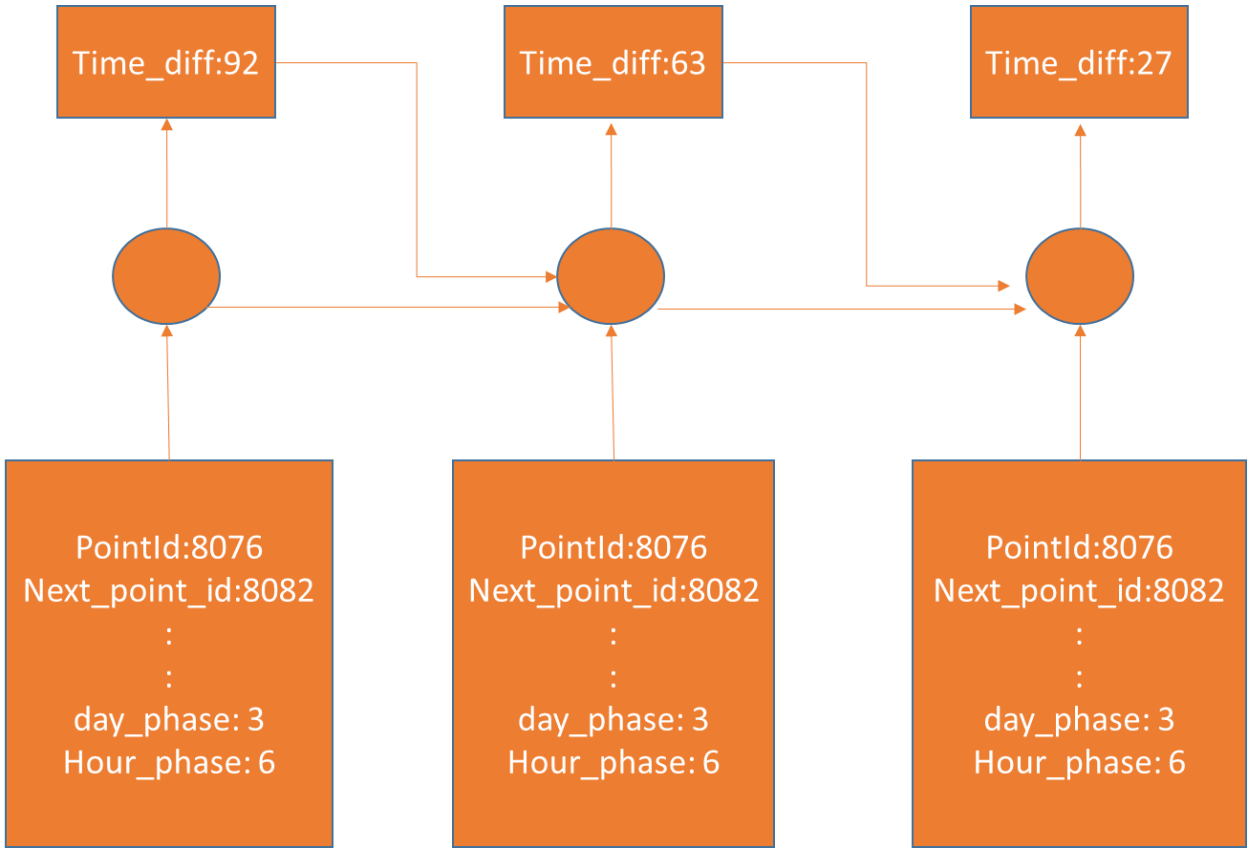


There are different Sequence models as defined in the above image. The basic sequence model is built using **Recurrent Neural Networks(RNN)**. RNNs are the type of neural network where the output from previous steps are fed to the next layers. LSTM networks are basically an extension of simple RNNs which was majorly developed in order to solve long term dependency problems, although nowadays it becomes a common practice to use LSTM Neural Networks instead of Simple RNN.

LSTM architecture :



There are mainly three gates in LSTM. The Forget gate is used to delete irrelevant information from the model memory whereas the update gate is used to add new information in the memory and finally the output gate is also used as a new input of the next layer. In our case we are using the predicted time value as an input to the next node.



Data Processing

As mentioned above, we need sequential data in order to learn the pattern of vehicles so we converted all the data in the desired format where each vehicle is passed as a one sequence and the time it took going from one segment to another.

I. From the above speed and delay analysis we can easily infer that the time taken for a vehicle to go from one stop to another majorly depends on the hour of the day. So the time of the day is an important feature for predicting the time. We also divided the whole time in

II. We know from our domain knowledge that each mode shares different properties, which are important features to learn for the model as well.

III. The distance between the segments is also a good feature for the model to learn, as it is correlated with the time taken between the segments.

Based on the above analysis and the domain knowledge we prepared the data as shown below, where each vehicle ID is representing one sequence and all the features on which it depends to predict the time difference.

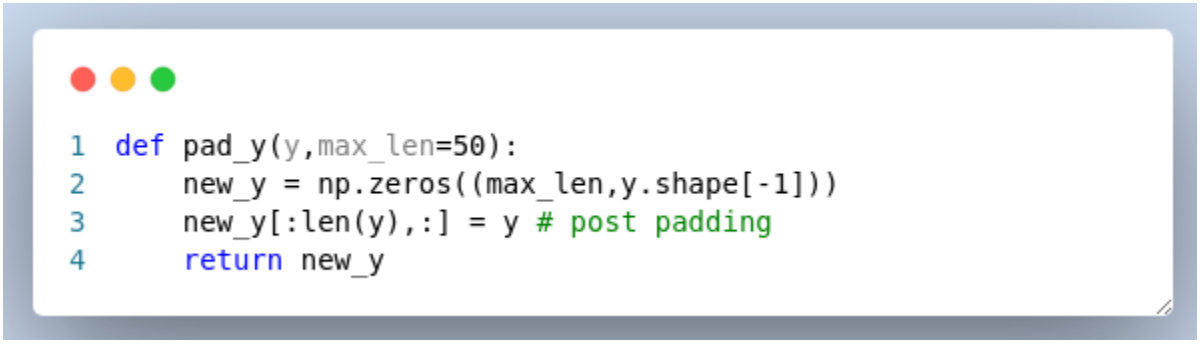
	day	veichle_id	pointId	next_point_id	stop_sequence	line_id	mode	day_of_week	min_time	day_phase	hour_phase	time_diff	distance	avg_time_diff
43106	2021-09-06	3	3458	3466	9	21	B	Monday	08:05:26.633000	3	8	71.439024	301.878802	82.135589
43107	2021-09-06	3	3466	2695	10	21	B	Monday	08:06:31.454000	3	8	62.000000	293.792810	79.551659
43108	2021-09-06	3	2695	3468	11	21	B	Monday	08:07:33.306000	3	8	85.301205	362.518091	67.826971
43109	2021-09-06	3	3468	4500	12	21	B	Monday	08:08:37.411000	3	8	94.000000	531.436600	118.666667
43110	2021-09-06	3	4500	4505	13	21	B	Monday	08:10:11.111000	3	8	128.000000	455.525960	119.333333
43111	2021-09-06	3	4505	3904	14	21	B	Monday	08:12:19.018000	3	8	139.382979	364.406643	101.114495
43112	2021-09-06	6	4505	3904	14	21	B	Monday	08:25:32.839000	3	8	63.000000	364.406643	101.114495
43113	2021-09-06	6	3904	3905	15	21	B	Monday	08:26:35.611000	3	8	96.666667	423.467359	90.777778
43114	2021-09-06	6	3905	3911	16	21	B	Monday	08:28:10.435000	3	8	67.609442	378.279415	82.217380
43115	2021-09-06	6	3911	6454	17	21	B	Monday	08:29:15.050000	3	8	111.413793	520.891061	113.471264
43116	2021-09-06	6	6454	1563	18	21	B	Monday	08:30:49.566000	3	8	31.000000	229.963892	57.044016
43117	2021-09-06	6	1563	3216	19	21	B	Monday	08:31:20.084000	3	8	66.000000	252.031500	94.890772
43118	2021-09-06	6	3216	3707	20	21	B	Monday	08:32:21.933000	3	8	69.000000	342.699199	77.600000
43119	2021-09-06	6	3707	1463	21	21	B	Monday	08:33:25.114000	3	8	130.000000	355.532987	134.007519

To give input in a LSTM we need a definite number of input nodes which model takes. In our data, since we took each sequence as a vehicle ID, the maximum sequence length is defined as the line id that contains the maximum no of sequence among all the line IDs. Here we defined **max len = 50 ,** because there is no sequence which has more than 50 segments.

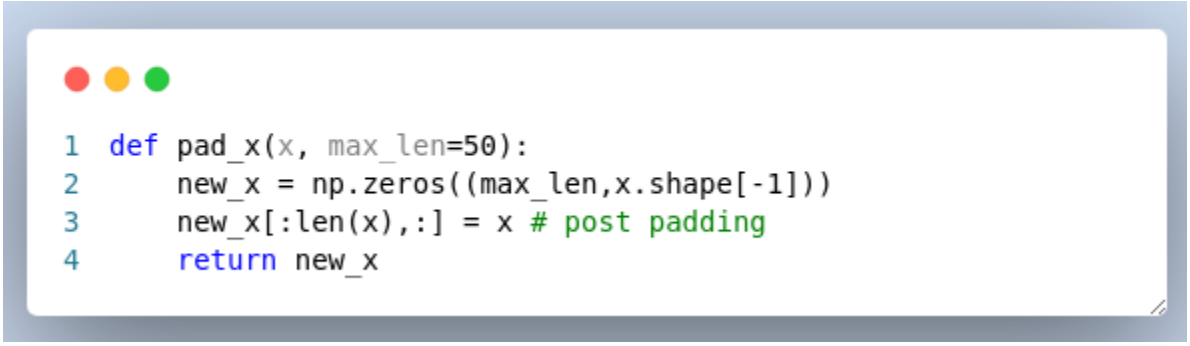
Since, model takes all values as integers, we need to map all the categorical values to integers, we mapped “day_of_week” and “mode” to the corresponding integer values as follows.

```
{'Monday': 1, 'Tuesday': 2, 'Wednesday': 3, 'Thursday': 4, 'Friday': 5,
'Saturday': 6, 'Sunday': 7}
{'M': 1, 'B': 2, 'T': 3}
```

As all line IDs contain different no of sequences, but we pass fixed size of input in model so we need to pad the data based on our max length. The way padding works is, we add zero vector to the end of feature vector and provide a unique identifier to the output vector.



```
def pad_y(y,max_len=50):
    new_y = np.zeros((max_len,y.shape[-1]))
    new_y[:len(y),:] = y # post padding
    return new_y
```



```
def pad_x(x, max_len=50):
    new_x = np.zeros((max_len,x.shape[-1]))
    new_x[:len(x),:] = x # post padding
    return new_x
```

Once the data is padded it is divided to train and test datasets. From 6th September 2021 to 19th September 2021 the data was given for training and the remaining two days were kept for the testing of the model.

4.2.1 Model Architecture

We used tensorflow and keras library which provides an easy interface to create complex neural network architecture. A many to many model is created where we defined 4 LSTM layers with

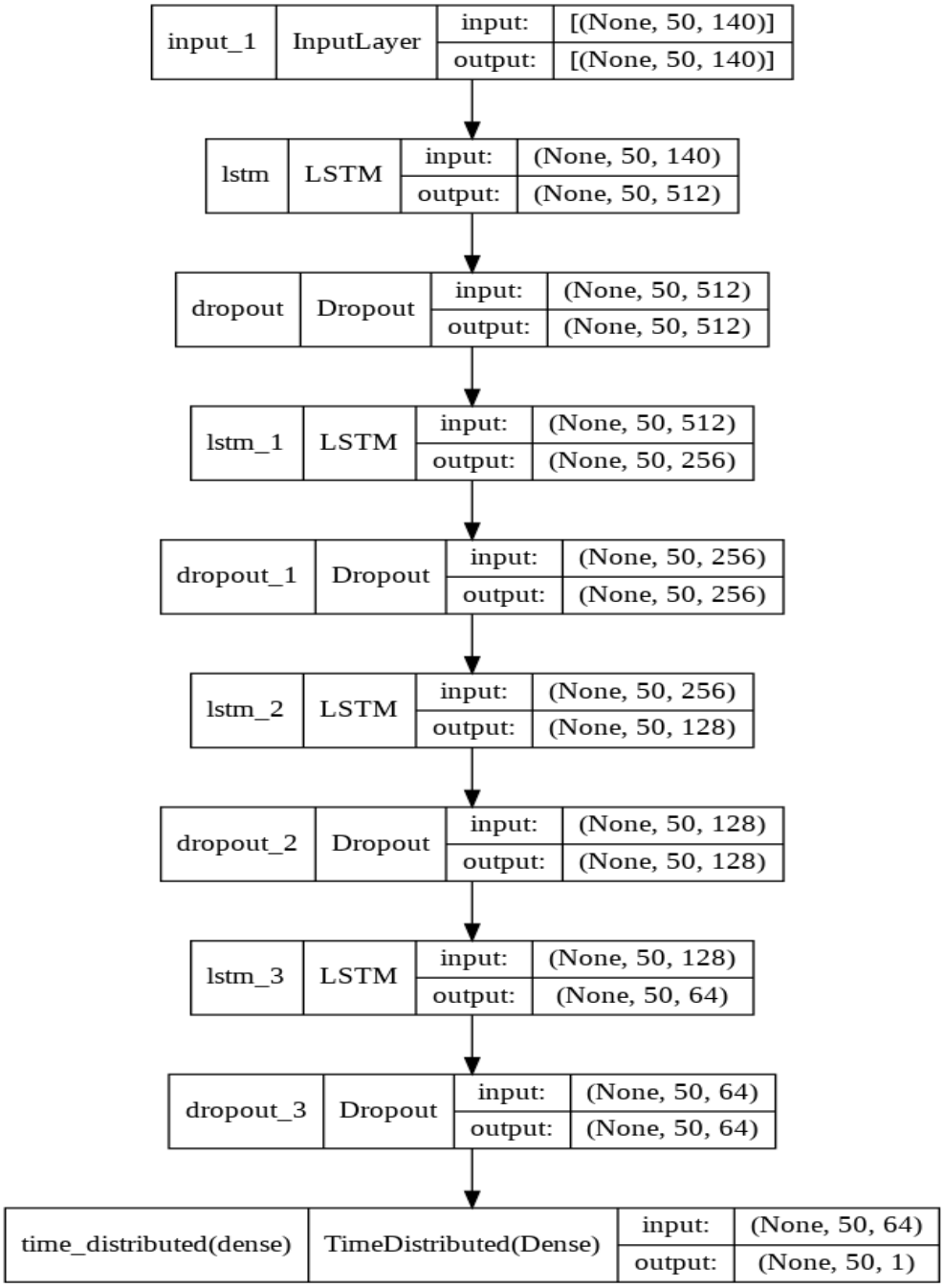
‘relu’ activation function. The dropout rate was set to 0.1 (To keep regularization effect). At the end a time-distributed layer with one node is added, as it predicts one value (time) for each input.

```
def many_to_many(timestamp, no_feature):
  model = Sequential()
  model.add(keras.Input(shape=(timestamp,no_feature,)))
  model.add(LSTM(512, activation='relu',return_sequences=True))
  model.add(Dropout(0.1))
  model.add(LSTM(256, activation='relu',return_sequences=True))
  model.add(Dropout(0.1))
  model.add(LSTM(128, activation='relu',return_sequences=True))
  model.add(Dropout(0.1))
  model.add(LSTM(64, activation='relu',return_sequences=True))
  model.add(Dropout(0.1))
  model.add(TimeDistributed(Dense(1)))
  return model
```

Where timestamp = 50 and no_features = 9. The detailed architecture of the following model can be seen from mode.summary() , and plot model functions provided by the keras.

```
model.summary()
```

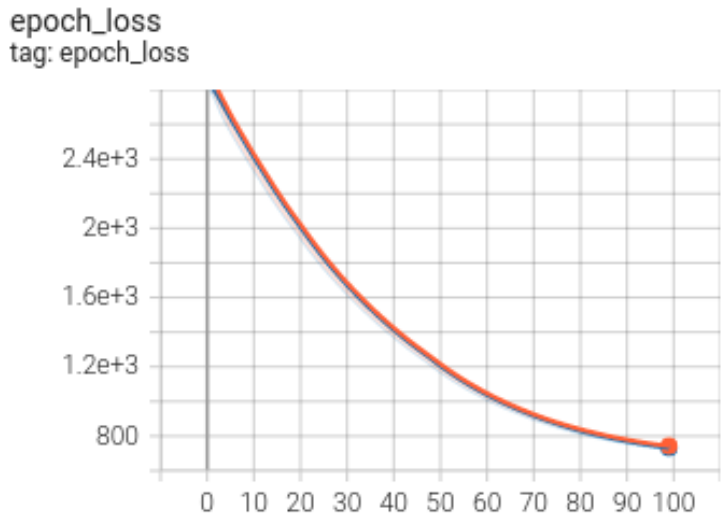
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 512)           1337344

 dropout (Dropout)           (None, 50, 512)           0

 lstm_1 (LSTM)               (None, 50, 256)           787456

 dropout_1 (Dropout)         (None, 50, 256)           0

 lstm_2 (LSTM)               (None, 50, 128)           197120

 dropout_2 (Dropout)         (None, 50, 128)           0

 lstm_3 (LSTM)               (None, 50, 64)            49408

 dropout_3 (Dropout)         (None, 50, 64)            0

 time_distributed (TimeDistr  (None, 50, 1)            65
 ibuted)

=================================================================
Total params: 2,371,393
Trainable params: 2,371,393
Non-trainable params: 0
_________________________________________________________________
```

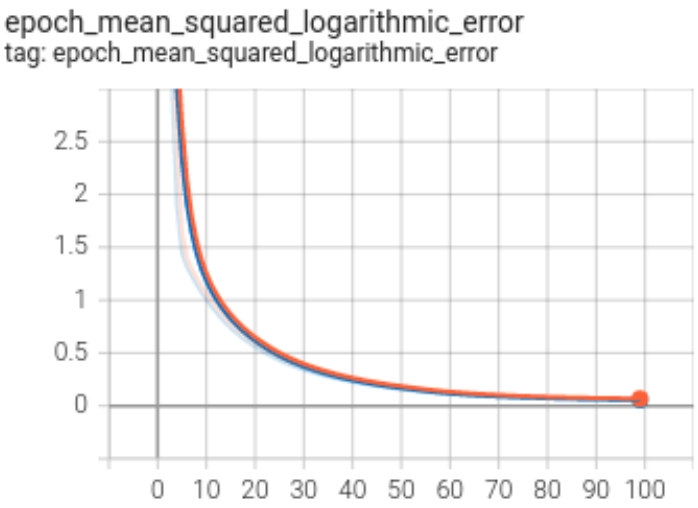


The above model architecture was trained on the data we generated earlier using ‘Adam’ optimizer and logarithmic mean squared error loss function. The model was trained with multiple 100 epochs and batch size 128 with the validation split of 20%.

```
opt = tf.keras.optimizers.Adam(clipvalue=0.25)
model.compile(optimizer=opt, loss='mse',metrics
 = [tf.keras.losses.MeanSquaredError()])
history = model.fit(X_train, y_train, epochs=100, validation_split=0.20, verbose
=1, batch_size=128,callbacks=[mc,tensorboard_callback])
```

Tensorflow also provides the **Tensorboard** function to save logs for the trained model. The loss and error curves can be seen below.





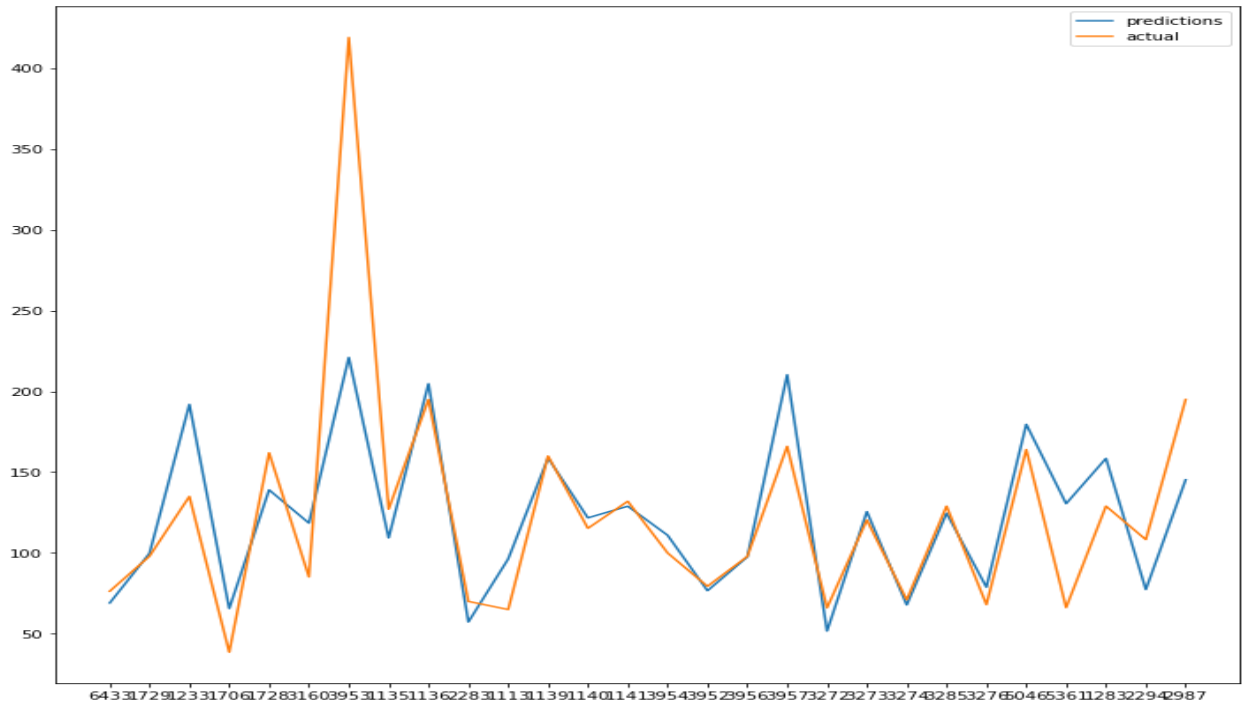
4.2.2 Results

Once the Model is trained we predict on our test dataset to check it's performance. The rmse error is indicated as below for the test data.

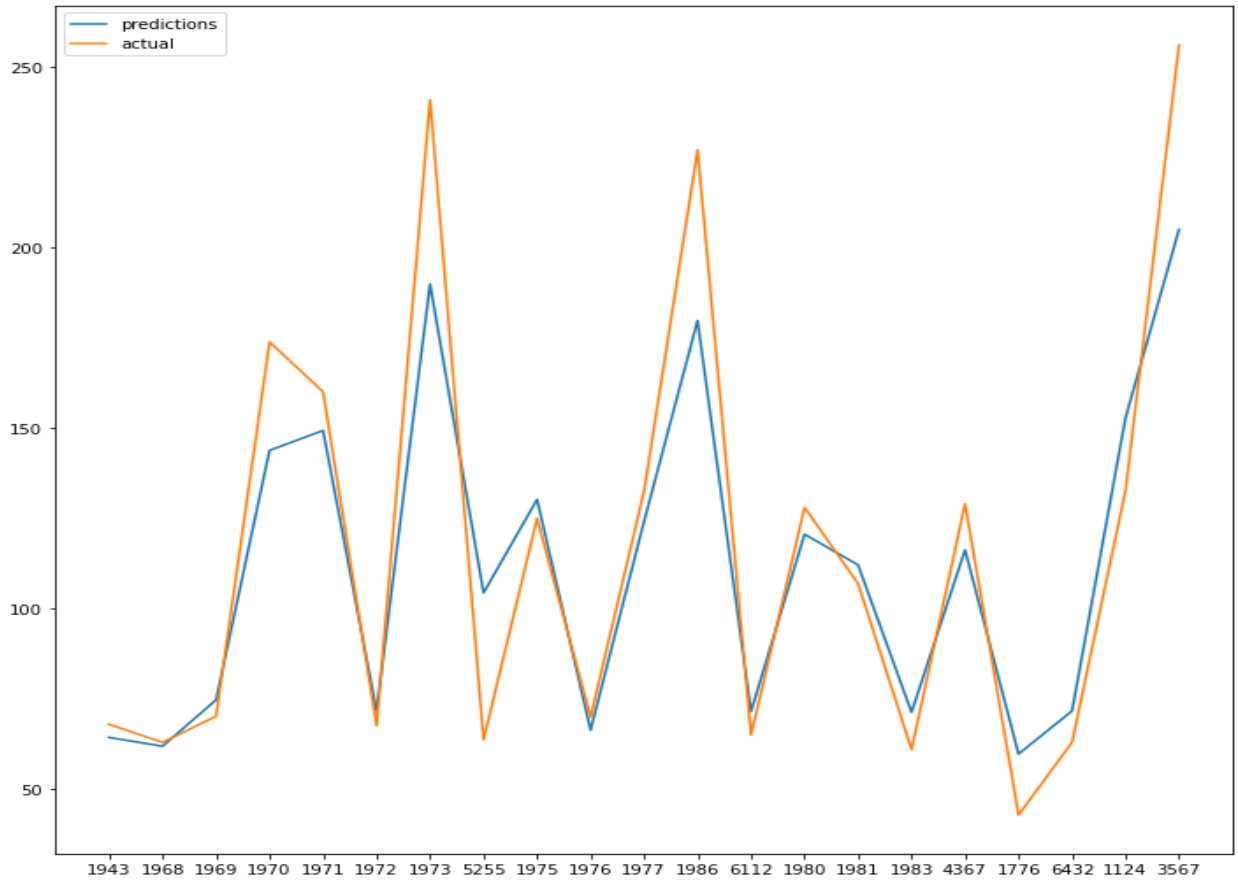
root mean squared error - 38.4304391625181

Since we are predicting for each vehicle we analyzed a few random outputs of the model for different line IDs. Here, as we can see the model is able to capture the trend of the time differences for the journey of the vehicle. The vehicle arrival time depends on many factor for that particular day, and model is trained on the multiple vehicles for multiple days so it is trained to predict average values between those segments based on different features.

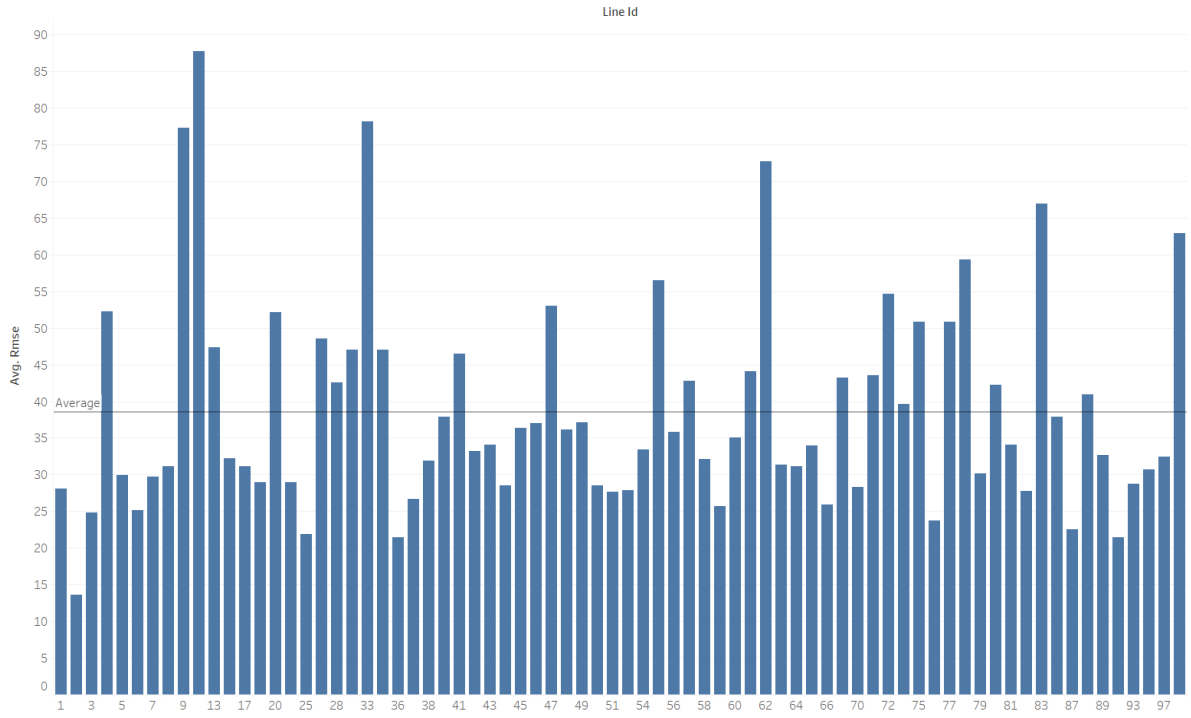
```
root mean squared error -  45.35972514288752
line  80
```




```
root mean squared error -  23.325972756578118
```


predictions
actual
250
200
150
100
50
1943
1968
1969
1970
1971
1972
1973
5255
1975
1976
1977
1986
6112
1980
1981
1983
4367
1776
6432
1124
3567

The below graph shows the average root mean squared error for all the lines. It can be seen that most of the values are below 40 which indicates that model is able to do the predictions correctly and is able to capture the trends as well.



5. Mode of Transportation

5.1 Data Preparation

We were provided the GPS tracks for 9 track Ids where we needed to predict the mode of transportation. The GPS tracks are from the city “Bruxelles” and consist of 4 classes - ‘Bus’, ‘Tramp’, ‘Metro’ and ‘Other’.

The given data can be seen below :

```
gps_track = pd.read_csv('/content/drive/MyDrive/data_mining/dataset/task4/GPStracks.csv')
gps_track.head()
```

	TrackId	lat	lon	time
0	1	50.851152	4.345326	2021-11-16T08:35:24Z
1	1	50.851235	4.345191	2021-11-16T08:35:45Z
2	1	50.851328	4.345069	2021-11-16T08:35:48Z
3	1	50.851403	4.344988	2021-11-16T08:35:51Z
4	1	50.851500	4.344920	2021-11-16T08:35:55Z

The data was in latitude and longitude format which was converted to point geometry and also since the GPS tracks were in “**EPSG:4326**” coordinate system, these were converted to belgium coordinate reference system (Belge 1972 / Belgian Lambert 72 - Belgium **EPSG:31370**). The processed data :

	TrackId	time	geometry
0	1	2021-11-16T08:35:24Z	POINT (148350.339 171195.050)
1	1	2021-11-16T08:35:45Z	POINT (148340.836 171204.286)
2	1	2021-11-16T08:35:48Z	POINT (148332.248 171214.634)
3	1	2021-11-16T08:35:51Z	POINT (148326.546 171222.979)
4	1	2021-11-16T08:35:55Z	POINT (148321.761 171233.771)

We also used the linestring data from the shape file for Algorithm.

	LIGNE	numero_lig	mode	VARIANTE	COLOR_HEX	Date_debut	Date_fin	geometry
0	001m	1	M	1	#C4008F	01/09/2021	06/03/2022	LINESTRING Z (146633.500 170956.400 0.000, 146...
1	002m	2	M	1	#F57000	01/09/2021	06/03/2022	LINESTRING Z (147305.500 172526.900 0.000, 147...
2	003t	3	T	1	#B5BA05	01/09/2021	06/03/2022	LINESTRING Z (148550.000 176641.300 0.000, 148...
3	004t	4	T	1	#F25482	01/09/2021	06/03/2022	LINESTRING Z (149360.500 172185.100 0.000, 149...
4	005m	5	M	1	#E6B012	01/09/2021	06/03/2022	LINESTRING Z (142852.600 167202.000 0.000, 142...

5.2 Data Modeling

To guess the mode of transportation we used two methods -

1. Finding nearest Neighbours (lines) of given GPS track points.

2. Classification Model using speed and acceleration.

5.2.1 Nearest Neighbour Algorithm

Since we already have the information of all the possible lines where the vehicle can traverse, we used that data in order to match with the GPS Points provided. As we also need to check if the trajectory is actually of some vehicle or it belongs to another class, we used the radius with the value 20 meters and if no line matches within this 20 meter radius, the point was classified as unmatched.

Algorithm

1. For each track do -
    a. Set threshold (20 meter in our case)
    b. Loop over all the track points of GPS trajectory
    c. For each Point find the distance with all the lines that are present
        i. If distance to closest line of the point > 20 meters, increase unmatched count
        ii. else keep track of the count of all line IDs that are matched within threshold value

After running the algorithms and taking top 5 matched values we provided the probability to each track Id based on it's distance from the line id and the number of points it matched with. Since the closer the distance is, the higher the chances of being that mode of line and higher the no. of point match to a particular line, higher the probability of that line.

5.2.2 Classification Model

Using the data that we got from the json file, we calculated the speed and acceleration on the basis of that and built the one vs rest classifier.

We calculated different stats for speed and acceleration using our speed analysis dataset as mentioned in section one. And used this data to train our model.

```
def stats_Calculator(data):
        mini = np.min(data)
        maxi = np.max(data)
        mean = np.mean(data)
        median = np.median(data)
        std = np.std(data)
        x_25 = np.percentile(data, 25)
        x_50 = np.percentile(data, 50)
        x_75 = np.percentile(data, 75)

        return [mini, maxi, mean, median,std,x_25,x_50,x_75]
```

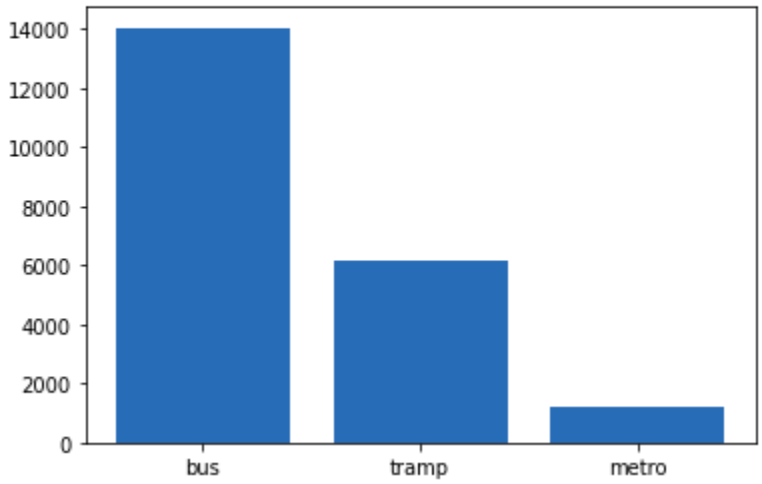
Here each row contains all the information about speed and acceleration for that particular mode.

mode	day_of_week	speed_min	speed_max	speed_mean	speed_median	speed_std	speed_per_25	speed_per_50	speed_per_75	acc_min	acc_max	acc_mean	acc_median	acc_std	acc_per_25	acc_per_50	acc_per_75
tramp	Tuesday	1.200000	74.671572	23.161010	18.109091	22.597119	10.238634	18.109091	23.834570	-1.767578	1.451982	-0.104369	-0.153011	0.962548	-0.606058	-0.153011	0.475071
bus	Tuesday	2.812500	34.861528	18.159022	17.121000	11.124225	7.575668	17.121000	26.803125	-0.794909	0.844588	-0.038620	-0.072271	0.459419	-0.380060	-0.072271	0.329106
bus	Tuesday	0.675000	30.515331	15.687666	18.848171	9.245746	6.968623	18.848171	23.415390	-0.667263	0.932510	0.106800	0.080130	0.451579	-0.164345	0.080130	0.500267
tramp	Tuesday	0.112500	41.998658	16.088545	11.043871	12.986266	6.991875	11.043871	28.837627	-2.187362	1.337926	0.079031	0.091289	0.754270	-0.172987	0.091289	0.626807
metro	Tuesday	0.105882	100.000000	39.823803	50.685995	36.511186	0.112500	50.685995	71.909506	-2.828242	3.026997	0.989091	1.532633	1.582044	0.003516	1.532633	2.319661

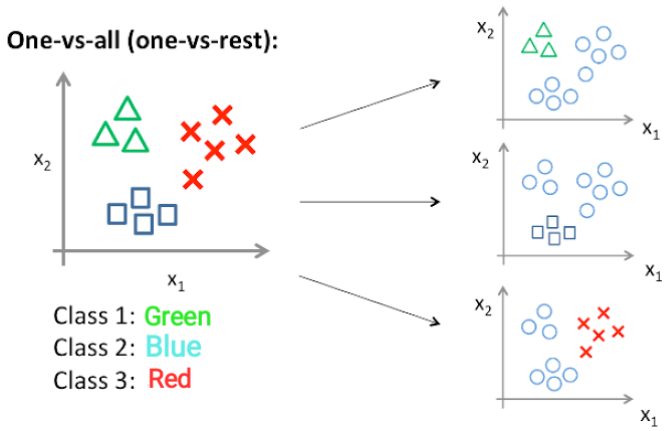
Data Distribution

```
plt.bar(['bus','tramp','metro'],temp)
```

```
<BarContainer object of 3 artists>
```

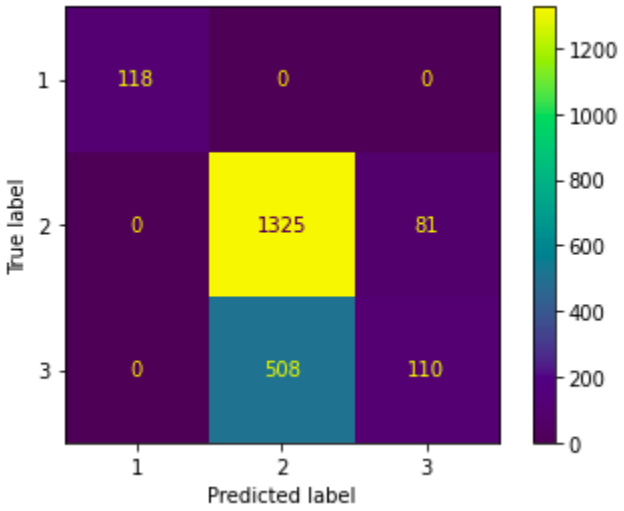


The data was splitted into train and test for the classifier model. We used one v/s rest classifier as our training model.



After training the mode we evaluated the results and it can be easily seen from the below pictures that the model was unable to distinguish between tram and metro based on speed and acceleration as both share the same features.

```
Test Set Accuracy : 72.50233426704014 %


Classification Report :

              precision    recall  f1-score   support

       metro       1.00      1.00      1.00       118
         bus       0.72      0.94      0.82      1406
       tramp       0.58      0.18      0.27       618

    accuracy                           0.73      2142
   macro avg       0.77      0.71      0.70      2142
weighted avg       0.70      0.73      0.67      2142
```



5.3 Results

Using our nearest neighbour algorithm we came up with good predictions and classified different tracks based on that. For example, In track ID 1, we got around 60% unmatch with the lines which were classified as other, whereas for trackID 4 we got highest match with line 50 for bus hence the mode of transportation classified as **BUS.**

Track id - 1
line unmatch percent 58.42696629213483

	track_id	line_id	Points matched No.	probability	class
0	1	other	0	1.000000	other
1	1	046b	48	0.593686	bus
2	1	218b	48	0.593686	bus
3	1	002m	15	0.250535	metro

Track id - 4
line unmatch percent 0.0

	track_id	line_id	Points matched No.	probability	class
0	4	050b	241	0.585064	bus
1	4	212b	125	0.409820	bus
2	4	097t	124	0.400773	tramp
3	4	032t	124	0.405666	tramp

Although since the classification model was unable to distinguish between tram and bus here the model predicted all the tracks as metro, the possible reason is that data is coming from different sources which is making the classification model incapable classify between different classes.

Conclusion

Here we have used the open data portal of STIB to analyze the vehicle patterns and we come up with different analyses. However, there were some limitations with this data that we resolved for better analytics. The Identification number for the vehicle was not provided, which made it complicated to track a single vehicle over its journey. Firstly we segregated the vehicles, calculated the distance between all the stops and prepared the data for speed and delay analysis.

In the Speed Analysis section we have shown how speed varies with different combinations including mode of transportations, time across the days, weekdays, weekends. The vehicles across all the line IDs tend to show higher speed in the night and lower in the day time. Metro runs with higher speed among all the transportation modes and bus and tram share almost the same speed.

In the Delay Analysis section, we have seen how the delay varies across different LineId and across StopId throughout the days of the week and time of the day. The analysis enabled us to understand more about the delays that happen for all three modes of transport - Bus, Tram and Metro in Brussels city at each stop. The average delay is highest for Bus when compared to other modes of transport, followed by Tram and least for Metro.

In the arrival time prediction we used one of the widely used neural network architectures (LSTM). LSTMs are designed for capturing the complex patterns from the sequence datasets. This can be observed from the predictions as well, where a single model is able to capture the details for each line id along with their different segments.

Given the analysis and predictions, we can conclude that - the analysis part helped us to understand the transport system better and it’s working whereas by applying the prediction model we were able to calculate the estimated time of the vehicle arrival. Hence, though the current model has it’s dependency on traffic, weather and time of the day. The model works quite optimal but it can be further improvised for good.

References

[1] https://en.wikipedia.org/wiki/Brussels_Intercommunal_Transport_Company

[2]https://www.stib-mivb.be/article.html?_guid=8086313c-3883-3410-f894-ec3da5b1280e&l=en#

[3] https://keras.io/api/layers/recurrent_layers/lstm/

[4] https://keras.io/api/metrics/regression_metrics/

[5] https://keras.io/api/losses/regression_losses/

[6] https://www.tensorflow.org/tensorboard

[7] https://stackabuse.com/solving-sequence-problems-with-lstm-in-keras-part-2/

[8] https://kepler.gl/

[9] https://www.tableau.com/