**A picture containing text

Description automatically generated**

**A01 – REPORT.**

# Description and Relevance

Bus routes are planned to take into consideration traffic patterns, patterns of movement etc. Over a period, these assumptions change. Traffic in one part of the city may increase while traffic in another part of the city may decrease. Also, the distribution of traffic throughout the day may change. All this implies that bus schedules need constant updating to reflect the current conditions.

One way to approach this problem is to consider every one-hour window and percentage the ratio of times buses has been late at every stop.

The routes and one-hour windows where we have the highest percentage of buses reporting late will be the prime candidates for replanning.

This dataset allows us to compute this information.

In this problem, we consider a bus late when it is late more than a threshold (set at 5 minutes by default).

Also, we only consider those routes where buses have been late by at least a minimum number in a month, currently set at 100. This is done to reduce noise.

Finally, we want to only display the routes for which the percentage of times they were late is bigger than a threshold, and sort them in descending order.

# Novelty

This problem is different from the four other problems in the assignment.

In the first problem, we take spatial rectangle based on the latitude and longitude. Then for every hour, within this rectangle, we calculate how often congestion is reported.

In the second problem, we want to calculate the timetable of a physical vehicle, as a pair of lineID and stationID, for a given day.

In the third problem, we compute the station that has the highest number of buses stopping at it so that we can use that station for best reach of advertisements.

In the fourth problem, we compute the distance travelled by individual vehicles so that we can find out when to send them for service.

This new problem is different from all others because we compute how many times a route has reported delays at stations, and then aggregate them by route and hour. This is not done for any of the given problems.

# Technical Discussion

To do this task, we must do the following steps:

1. Load the dataset
2. Filter by the month to reduce the dataset
3. Discard any rows where the bus is not stopped at a station
4. Calculate the number of times buses stops at a station for every route and hour
   1. We must exclude consecutive entries where a bus continues to stop at a station, and only take one of those entries
5. Calculate the number of times a bus stops at a station and is also late for every route and hour
   1. Again, we must exclude consecutive entries where the bus continues to stop at the same station and only take in one of those entries
   2. Discard any entries where the number of times a bus is late is less than the specified threshold
6. Perform an inner join of the results from Step 4 and 5 on the route and hour and calculate the percentage of times
7. Discard any rows where the percentage calculated in step 6 is less than the threshold
8. Sort in descending order and display

# Detailed Steps

Reading the dataset is performed by the following code block

inputDF = spark.read.format("csv") \

.option("delimiter", ",") \

.option("quote", "") \

.option("header", "false") \

.schema(my\_schema) \

.load(my\_dataset\_dir)

We then discard all columns where the bus is not at a stop. We add a new column called ‘time’ that is derived from date, a column called ‘isLate’ depending whether the delay is greater than the threshold or not. We also add a column for the hour, and although this is a repetition, it helps us to group things later, and also helps in debugging if we have to print the dataset in the intermediate states.

inputDF = inputDF\

.filter(month\_picked == f.substring(f.col('date'), 0, 7))\

.where(f.col('atStop') == 1)\

.withColumn('time', f.substring(f.col('date'), 12, 9))\

.withColumn('isLate', is\_late(f.col('delay')))\

.drop('congestion', 'longitude', 'latitude', 'busLinePatternumID')\

.withColumn('hour', f.substring(f.col('time'), 0, 2))

We next introduce a row number to the dataset, and use the lag to drop duplicate entries. Following this we group by busLineID and the hour, and take the counts. This is done for the instances where the bus is late.

# If a bus has been late at a stop, count it, but count every instance only once

# Do not count consecutive late instances

ws2 = Window.partitionBy('vehicleID', 'closerStopID', 'isLate').orderBy('vehicleId', 'time', 'closerStopID')

windowSpec = Window.partitionBy().orderBy('vehicleId', 'time')

uniqueLateDF = inputDF\

.where(f.col('isLate') == 1)\

.withColumn('rnum', f.row\_number().over(windowSpec))\

.withColumn('lag', f.lag('rnum', default=-1).over(ws2))\

.where(f.col('rnum') - f.col('lag') != 1)

aggregatedLateDF = uniqueLateDF\

.groupBy(['busLineID', 'hour'])\

.count()

We do a similar thing for all instances (where the bus is late and isn’t).

# If a bus has been at a top once, count it only once

ws2 = Window.partitionBy('vehicleID', 'closerStopID').orderBy('vehicleId', 'time')

windowSpec = Window.partitionBy().orderBy('vehicleId', 'time')

uniqueAtStopDF = inputDF.withColumn('rnum', f.row\_number().over(windowSpec))\

.withColumn('lag', f.lag('rnum', default=-1).over(ws2))\

.where(f.col('rnum') - f.col('lag') != 1)

aggregatedAtStopDF = uniqueAtStopDF\

.groupBy(['busLineID', 'hour'])\

.count()

We then create named tables for the two dataframes we obtained in the previous step and then join them to take the percentage of all late instances.

query = """

SELECT

ROUND(late\_instances.count / all\_instances.count \* 100, 2) AS percentage,

late\_instances.hour AS hour,

late\_instances.busLineID AS busLineID

FROM

late\_instances

INNER JOIN

all\_instances

ON

late\_instances.hour = all\_instances.hour

AND

late\_instances.busLineID = all\_instances.busLineID

AND

late\_instances.count > {}

AND

(late\_instances.count \* 100 / all\_instances.count) > {}

ORDER BY

percentage

DESC

""".format(late\_count\_threshold, late\_percentage\_threshold)

solutionDF = spark.sql(query)