Some useful stats about the data
no. of rows: 36010872
no. of distinct route_id: 558
no. of distinct vehicle IDs: 2276
max speed in the dataset: 4.39

query 2 logic and assumptions: We are reporting one vehicle which is fastest for a given route even if for a route there are multiple vehicles with same maximum speed to print less data.
query 3 logic and assumptions: We are reporting the latitude and longitude of a particular bus only once even if it is appearing multiple times on the route where the max speed or min speed vehicle is travelling as it will create confusion if we will add same bus with different lat,lng for a route.
query4 logic and assumptions: we are not finding distinct as some vehicles can have multiple speeds in that time
query5 logic and assumptions: Here we used the arc - distance formula to find the distance between lat and longitude which can be read from the link given below. In this query we reported the buses that are closest to IIITD's lat,lng which is :28.5459,77.2732
Reference: https://www.geeksforgeeks.org/program-distance-two-points-earth/
query6 logic and assumptions: In this query we found the vehicle id's whose speed>40 and printed those vehicles . In the results one vehicle can be reported multiple times as there is no mention of distinct vehicles to be reported So that is our assumption to print all vehicles even if same vehicle is exceeding threshold speed multiple times.
query7 logic and assumptions: In this query we are reporting the number of distinct vehicles on a given route (for a given time which will be provided by you which can be edited in the given_time7 variable in our program) and we are reporting the count for each route.

Kafka Primitives Used:

- We created two topics named testnum and testnum2 where we used testnum2 topic for sending filtered data to testnum2 topic for query 7 to 0th partition(following 0 based indexing).
- We used testnum topic with multiple partitions to send data for other queries where for query 6th we send the data by filtering at producer level of rows having speed>40 to 1st partition and for other queries we send the rows to 0th partition of the same topic.
- We also made 3 consumers one for query 7(consumer4_2.py), one for query 6(consumer4_1.py) and the remaining queries on 3rd consumer (consumer4_0.py)
- We also serialize data using json and compress using gzip at producer. After that, we stream data

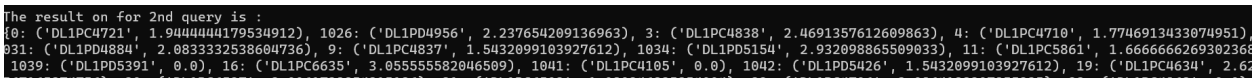
//(Producer filename:producer3.py).

Insights:

a) **We didn't used KSQL as it was taking a lot of time to find results for a relatively small amount of data like 21000 rows only and since we had almost 3.7 Crore rows So we dropped the idea of using KSQL.**

b) **Streaming of complete dataset takes 10 to 15mins(this is the best time we could achieve after many optimizations). Bottleneck is at producer as it does disk reads.**

2.



3.

select max(speed), min(speed) from vehicle_feed where timestamp = 1613984417;

	max(speed)	min(speed)
1	3.16358017921448	0.0

```
1   select vehicle_id from vehicle_feed where timestamp = '1613984417' and speed = 0.0;
```

	vehicle_id
1	DL1PD4287
2	DL1PD5267
3	DL1PD0899

```
1    select vehicle_id, speed from vehicle_feed where speed > 3.16 and timestamp ==1613984417 ;
```

	vehicle_id	speed
1	DL1PD4252	3.16358017921448
2	DL1PD4703	3.16358017921448

```
1    select DISTINCT route_id from vehicle_feed where (vehicle_id=='DL1PD4252' or vehicle_id=='DL1PD4287') and timestamp ==1613984417 ;
```

	route_id
1	1172.0
2	555.0

```
The result on for 3rd query is :
[3.1635801792144775, 555] [0.0, 1172]
[['DL1PD4384', '28.7964305877686', '76.9608993530273'], ['DL1PD4276', '28.7970523834229', '76.96067047119l4'], ['DL1PD4252', '28.8237113952637', '76.9914016
723633'], ['DL1PD4228', '28.7475891113281', '77.0900115966797'], ['DL1PD4185', '28.7486152648926', '77.0866928100586'], ['DL1PD4180', '28.800422668457', '77
.0352783203125'], ['DL1PD4556', '28.8178634643555', '76.9740447998047'], ['DL1PD4192', '28.8255615234375', '76.9908218383789'], ['DL1PD4265', '28.8005886077
881', '77.035247802734l']]
[['DL1PD4287', '28.680269241333', '77.1659240722656'], ['DL1PD4281', '28.6735877990723', '77.1694793701172'], ['DL1PD4814', '28.6736717224121', '77.16939544
67773'], ['DL1PD4371', '28.6814002990723', '77.0790863037109'], ['DL1PD4320', '28.6757125854492', '77.1254348754883']]
```

4.

```
The result on for 4th query is :
route avgspeed:0 1.0030864228804905
route avgspeed:1026 2.237654209136963
route avgspeed:3 0.848765417933464
route avgspeed:4 0.9259259303410848
route avgspeed:1031 1.6782407164573 67
route avgspeed:9 0.4552469253540039
route avgspeed:1034 0.5163817680799044
route avgspeed:11 0.6481481393178304
route avgspeed:1039 0.0
route avgspeed:16 1.0559964648314886
route avgspeed:1041 0.0
route avgspeed:1042 1.5432099103927612
route avgspeed:19 1.9855966567993164
route avgspeed:20 1.558642029762268
```

5.

```
{'DL1PC6792'}
```

```
['DL1PC6792', 28.5546226501465, 77.2713165283203]

    dist
✓ 0.1s

0.9872067639444058
```

6.nothing

7.

```
^CThe result on for 7th query is :
{0.0: 6, 1026.0: 1, 3.0: 6, 4.0: 3, 1031.0: 4, 9.0: 4, 1034.0: 13, 11.0: 3, 1039.0: 2, 16.0: 7, 1041.0: 1, 1042.0: 1, 19.0: 3,
.0: 3, 26.0: 6, 27.0: 13, 1053.0: 10, 1054.0: 1, 1055.0: 1, 757.0: 1, 34.0: 2, 35.0: 4, 36.0: 8, 37.0: 9, 1063.0: 2, 40.0: 4,
0: 1, 1073.0: 4, 50.0: 3, 1075.0: 9, 53.0: 4, 1078.0: 6, 55.0: 7, 1080.0: 2, 10.0: 4, 1087.0: 1, 64.0: 2, 65.0: 2, 1090.0: 2,
```

Queries

2).Select distinct route_id,vehicle_id,speed from vehicle_feed where (route_id,speed) in (Select distinct route_id,MAX(speed) as max_speed from vehicle_feed GROUP by route_id) ORDER BY route_id

3). (a) Select distinct vehicle_id,lat,lng,route_id from vehicle_feed where route_id in (Select distinct route_id from vehicle_feed where speed in (Select MAX(speed) as max_speed from (Select * from vehicle_feed where timestamp = 1613984417)))

//Query for max result

3). (b) Select distinct vehicle_id,lat,lng,route_id from vehicle_feed where route_id in (Select distinct route_id from vehicle_feed where speed in (Select MIN(speed) as min_speed from (Select * from vehicle_feed where timestamp = 1613984417)))

// Query for min result

I am assuming that the we need to report the buses that are the fastest and slowest at timestamp = 1613984417.

4)Select route_id,AVG(speed) as avg_speed from vehicle_feed where time >’12:00:00’ and time <’13:00:00’ GROUP BY route_id order by route_id
5)
ALTER TABLE vehicle_feed MODIFY lat float;
ALTER TABLE vehicle_feed MODIFY lng float;

Select distinct vehicle_id from vehicle_feed where time >’15:00:00’ and time < ‘16:00:00’ and (lat-28.5459)*(lat-28.5459)+(lng-77.2732)*(lng-77.2732) in (Select MIN((lat-28.5459)*(lat-28.5459)+(lng-77.2732)*(lng-77.2732)) as min_dist from vehicle_feed where time >’15:00:00’ and time < ’16:00:00’)
//In query3 we wrote the query for the euclidean distance formula i have taken (lat1-lat2)^2 + (lng1-lng)^2 but this need to be changed and as

the actual formula is quite tough to implement in sql as it is maths heavy so we implemented it in python . The logic remains same just the distance formula changes.

6). Select vehicle_id from vehicle_feed where speed>40

7) select route_id, Count(distinct vehicle_id) from vehicle_feed where time == '00:00:16' group by route_id;

Steps to run kafka in python(vagrant)

1.Start zookeeper(in vagrant it automatically turns on)
2.sudo -s
3. $ cd kafka_2.12-2.8.1
4. $ Now in one tab run bin/kafka-server-start.sh config/server.properties

5. Create topic with replication factor of 1 : bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic testnum2

6. Now in one tab Start the consumer
python3 consumer2.py

7. Start the producer
python3 producer2.py

Maven installation in eclipse:

Go to help-> Install new Software -> select add tab there add maven and in location -> http://download.eclipse.org/technology/m2e/releases/1.3 add this

then follow simple steps