# Data Engineering Project Assignment -2

**Team Name** - Data Foundry

**Team Members:**
Kiranmayi Thanikonda
Jayita Banerjee
Navya Sri Ambati

**Step-1:**

Construct a table showing each day for which your pipeline successfully, automatically processed one complete day's worth of sensor readings. The table should look like this:

| Date | Day of Week | # Sensor Readings | # Rows added to your database |
|---|---|---|---|
| 04/21/2025 | Monday | 662084 | 661678 |
| 04/22/2025 | Tuesday | 657487 | 657119 |
| 04/23/2025 | Wednesday | 686156 | 685815 |
| 04/24/2025 | Thursday | 682850 | 682375 |
| 04/25/2025 | Friday | 480013 | 479639 |
| 04/26/2025 | Saturday | 490978 | 490535 |
| 04/27/2025 | Sunday | 492362 | 491902 |
| 04/28/2025 | Monday | 512624 | 512365 |
| 04/29/2025 | Tuesday | 689640 | 689232 |
| 04/30/2025 | Wednesday | 706623 | 706241 |
| 05/01/2025 | Thursday | 625101 | 624564 |
| 05/02/2025 | Friday | 720166 | 719778 |

| | | | |
|---|---|---|---|
| 05/03/2025 | Saturday | 488878 | 488224 |
| 05/04/2025 | Sunday | 484105 | 483837 |
| 05/05/2025 | Monday | 505612 | 505357 |
| 05/06/2025 | Tuesday | 641745 | 641319 |
| 05/07/2025 | Wednesday | 699449 | 698771 |
| 05/08/2025 | Thursday | 678557 | 678027 |
| 05/09/2025 | Friday | 650559 | 650117 |

## Step-2: Data Validation

1. Every breadcrumb record must have a trip number. – Existence assertion
2. Each record must include a stop number, operation date, latitude, longitude, and meters. – Existence assertion
3. The ACT_TIME value (time of reading) must not be empty or missing. – Existence assertion
4. The operation date (OPD_DATE) must follow the format like “08May2025”. – Format check
5. The latitude value must be between 45.2 and 45.7 valid range near Portland. – Limit assertion
6. The longitude value must be between -124.0 and -122.0. – Limit assertion
7. The vehicle ID must be a positive number. – Limit assertion
8. The meter value distance must be zero or more; it can't be negative. – Limit assertion
9. If GPS satellite data is available, it should be a number between 0 and 12. – Limit assertion
10. No two breadcrumb records should have the same combination of vehicle ID, trip number, stop number, time, and meters. – Uniqueness constraint
11. The time value (ACT_TIME) must be between 0 and 86,400 seconds ( within 24 hours). – Limit assertion
12. Each record must include a vehicle ID. – Existence assertion
13. The GPS precision value (GPS_HDOP) should be between 0 and 26. – Limit assertion
14. The meter values for a trip should follow a realistic pattern normally or exponentially distributed. – Distribution assertion
15. When more satellites are used, the GPS precision (GPS_HDOP) should improve. – Statistical assertion
16. Within the same trip, distance (METERS) should increase as time (ACT_TIME) increases. – Inter-record assertion

17. The same trip number and stop number should not appear for different vehicles. – Inter-record assertion
18. In any trip, the stop number should always be greater than the trip number. – Inter-record assertion
19. If latitude is present, longitude must also be present—both are needed to pinpoint a location. – Intra-record assertion
20. Each trip ID must be linked to a valid route ID. – Referential integrity assertion

## Step-3: Documentation of Each of the Original Data Fields

| Column Name | Type | Documentation |
|---|---|---|
| EVENT_NO_TRIP | Numeric | A unique 9-digit integer that identifies a specific trip in the dataset. |
| EVENT_NO_STOP | Numeric | A 9-digit integer used to represent the stop location associated with a given trip. |
| OPD_DATE | Timestamp | Indicates the date when the breadcrumb record was collected. |
| VEHICLE_ID | Numeric | A unique 4-digit identifier assigned to each vehicle; up to 100 unique vehicle IDs exist in the dataset. |
| METERS | Numeric | Represents the cumulative odometer reading at a particular moment, measured in meters. |
| ACT_TIME | Numeric | Denotes the elapsed time from midnight of the operation date, measured in seconds, valid range from 0 to 86,340. |
| GPS_LONGITUDE | Numeric | Specifies the longitude coordinate of the vehicle at the given time; values range between -124.0 and -122.0. |
| GPS_LATITUDE | Numeric | Specifies the latitude coordinate of the vehicle at the given time; valid range is from 45.2 to 45.7. |
| GPS_SATELLITES | Numeric | Represents the number of GPS satellites used to determine the location; valid range is from 0 to 12. |

| | | |
|---|---|---|
| GPS_HDOP | Numeric | Horizontal Dilution of Precision a metric for GPS accuracy in the horizontal plane; valid values range from 0 to 25.5. |

## Step-4: Data Transformations

- A new timestamp column was created by combining OPD_DATE (as a date) with ACT_TIME (as seconds) to represent the exact time of each breadcrumb.
- If the value of ACT_TIME exceeded 86,340 seconds, the date was adjusted to the next day before adding the ACT_TIME value.
- Missing latitude and longitude values were filled by grouping the records by trip_id, sorting them by timestamp, and interpolating based on the subsequent values.
- The SPEED column was derived using the formula ΔMETERS / ΔTIME, where differences were calculated within each trip using groupby().diff().
- Missing SPEED values were filled using backward fill (.bfill()) to ensure no nulls during insertion.
- The dataset was sorted by trip_id and timestamp before calculating speed to maintain chronological order.
- The timestamp column was renamed to tstamp, and columns were reordered as [tstamp, latitude, longitude, speed, trip_id].
- Records failing validation (e.g., missing fields, invalid lat/lon, duplicate keys) were filtered out before transformation.
- Intermediate columns such as METERS, ACT_TIME, and OPD_DATE were dropped after computing necessary values.

## Step-5: Example Queries

Answer the following questions about the TriMet system using your vehicle group's sensor data in your database. In your submission document include your query code, number of rows in each query result (if applicable) and first five rows of the result (if applicable).

1. How many breadcrumb reading events occurred on January 1, 2023?

```
kthaniko@datafoundry-sub:~$ sudo -u postgres psql postgres
psql (15.12 (Debian 15.12-0+deb12u2))
Type "help" for help.

postgres=# SELECT *  FROM trip;
postgres=# select count(*) from breadcrumb WHERE DATE(tstamp) = '2023-01-01';
 count
--------
 483350
(1 row)

postgres=# select * from breadcrumb WHERE DATE(tstamp) = '2023-01-01' limit 5;
       tstamp        | latitude  |  longitude  |       speed        |  trip_id
---------------------+-----------+-------------+--------------------+-----------
 2023-01-01 00:00:04 | 45.526048 | -122.558123 |                6.2 | 229319827
 2023-01-01 00:00:49 | 45.526143 | -122.558123 | 0.2222222222222222 | 229319827
 2023-01-01 00:00:54 |  45.52639 |  -122.55812 |                5.8 | 229319827
 2023-01-01 00:00:59 | 45.526755 | -122.558112 |                8.4 | 229319827
 2023-01-01 00:01:04 | 45.527167 | -122.558103 |                  9 | 229319827
(5 rows)

postgres=#
```

2. How many breadcrumb reading events occurred on January 2, 2023?

```
postgres=# select count(*) from breadcrumb WHERE DATE(tstamp) = '2023-01-02';
 count
--------
 505935
(1 row)

postgres=# select * from breadcrumb WHERE DATE(tstamp) = '2023-01-02' limit 5;
       tstamp        | latitude  |  longitude  | speed |  trip_id
---------------------+-----------+-------------+-------+-----------
 2023-01-02 00:00:07 | 45.514732 | -122.680928 |     3 | 229709562
 2023-01-02 00:00:12 | 45.515032 | -122.680777 |     7 | 229709562
 2023-01-02 00:00:17 | 45.515328 | -122.680612 |     7 | 229709562
 2023-01-02 00:00:22 | 45.515635 | -122.680443 |   7.2 | 229709562
 2023-01-02 00:00:27 | 45.515938 | -122.680288 |   7.2 | 229709562
(5 rows)

postgres=#
```

3. On average, how many breadcrumb readings are collected on each day of the week?

select to_char(tstamp, 'Day') AS day_of_week, count(*) / count(distinct date(tstamp)) AS avg_readings FROM breadcrumb GROUP BY to_char(tstamp, 'Day'), EXTRACT(DOW FROM tstamp) ORDER BY EXTRACT(DOW FROM tstamp);

```
postgres=# select TO_CHAR(tstamp, 'Day') AS day_of_week, COUNT(*) / COUNT(DISTINCT DATE(tstamp)) AS avg_readings
 FROM breadcrumb GROUP BY TO_CHAR(tstamp, 'Day'), EXTRACT(DOW FROM tstamp) ORDER BY EXTRACT(DOW FROM tstamp);
 day_of_week | avg_readings
-------------+--------------
 Sunday      |       486268
 Monday      |       553453
 Tuesday     |       661070
 Wednesday   |       695747
 Thursday    |       661738
 Friday      |       615839
 Saturday    |       336937
(7 rows)

postgres=#
```

4. List the TriMet trips that traveled a section of I-205 between SE Division and SE Powell on January 1, 2023. To find this, search for all trips that have breadcrumb readings that occurred within a lat/long bounding box such as [(45.497805, -122.566576), (45.504025, -122.563187)].

   SELECT DISTINCT trip_id FROM breadcrumb WHERE DATE(tstamp) = '2023-01-01' AND latitude BETWEEN 45.497805 AND 45.504025 AND longitude BETWEEN -122.566576 AND -122.563187 limit 5;

```
postgres=# SELECT DISTINCT trip_id FROM breadcrumb WHERE DATE(tstamp) = '2023-01-01' AND latitude BETWEEN 45.497
805 AND 45.504025 AND longitude BETWEEN -122.566576 AND -122.563187 limit 5;
  trip_id
-----------
 229646175
 229730617
 229816196
 229816838
 229983840
(5 rows)

postgres=#
```

5. List all breadcrumb readings on a section of US-26 west side of the tunnel (bounding box: [(45.506022, -122.711662), (45.516636, -122.700316)]) during Mondays between 4pm and 6pm. Order the readings by tstamp. Then list readings for Sundays between 6am and 8am. How do these two time periods compare for this particular location?

   <u>Query for Mondays between 4pm and 6pm:</u>
   select b.* FROM BreadCrumb b

JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 1
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 16 AND 17
ORDER BY b.tstamp limit 5;

```
postgres=# select b.* FROM BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 1
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 16 AND 17
ORDER BY b.tstamp limit 5;
       tstamp        | latitude  |  longitude  | speed |  trip_id
---------------------+-----------+-------------+-------+-----------
 2022-12-19 16:01:13 | 45.508617 | -122.700788 |     9 | 222810467
 2022-12-19 16:01:18 |  45.50858 | -122.701477 |  10.6 | 222810467
 2022-12-19 16:01:23 | 45.508485 | -122.702253 |  12.2 | 222810467
 2022-12-19 16:01:28 | 45.508415 | -122.702902 |  10.2 | 222810467
 2022-12-19 16:01:33 | 45.508487 | -122.703395 |   7.8 | 222810467
(5 rows)

postgres=#
```

Query:
select count(*) FROM BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 1
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 16 AND 17;

```
postgres=# select count(*) FROM BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 1
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 16 AND 17;
 count
-------
   184
(1 row)

postgres=#
```

Query for Sundays between 6am and 8am:

```
select b.* from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 0
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 6 AND 7
ORDER BY b.tstamp limit 5;
```

```
postgres=# select b.* from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 0
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 6 AND 7
ORDER BY b.tstamp limit 5;
       tstamp        | latitude  |  longitude  | speed |  trip_id
---------------------+-----------+-------------+-------+-----------
 2022-12-25 06:03:50 | 45.506993 | -122.710868 |  19.4 | 225973992
 2022-12-25 06:03:55 | 45.507348 | -122.709732 |  19.4 | 225973992
 2022-12-25 06:04:00 | 45.507998 | -122.708832 |    20 | 225973992
 2022-12-25 06:04:05 | 45.508828 | -122.708207 |  20.8 | 225973992
 2022-12-25 06:04:10 | 45.509675 | -122.707637 |  20.6 | 225973992
(5 rows)

postgres=#
```

Query:

```
select count(*)  from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 0
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 6 AND 7;
```

```
postgres=# select count(*)  from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
WHERE b.latitude BETWEEN 45.506022 AND 45.516636
AND b.longitude BETWEEN -122.711662 AND -122.700316
AND EXTRACT(DOW FROM b.tstamp) = 0
AND EXTRACT(HOUR FROM b.tstamp) BETWEEN 6 AND 7;
 count
-------
    35
(1 row)
```

6. What is the maximum speed reached by any bus in the system?

```
postgres=# select max(speed) AS max_speed from BreadCrumb;
 max_speed
-----------
       236
(1 row)

postgres=#
```

7. List all speeds and give a count of the number of vehicles that move precisely at that speed during at least one trip. Sort the list by most frequent speed to least frequent.

```
select b.speed, count(distinct t.vehicle_id) AS vehicle_count
FROM BreadCrumb b JOIN Trip t ON b.trip_id = t.trip_id
GROUP BY b.speed ORDER BY vehicle_count DESC, b.speed ASC limit 5;
```

```
postgres=# select b.speed, count(distinct t.vehicle_id) AS vehicle_count
FROM BreadCrumb b JOIN Trip t ON b.trip_id = t.trip_id
GROUP BY b.speed ORDER BY vehicle_count DESC, b.speed ASC limit 5;
        speed        | vehicle_count
---------------------+---------------
                    0 |           189
                 0.05 |           189
                 0.08 |           189
                  0.1 |           189
 0.13333333333333333 |           189
(5 rows)

postgres=#
```

8. Which is the longest (in terms of time) trip of all trips in the data?

   select b.trip_id, max(b.tstamp) - min(b.tstamp) AS duration FROM BreadCrumb b GROUP BY b.trip_id ORDER BY duration DESC LIMIT 1;

```
postgres=# select b.trip_id, max(b.tstamp) - min(b.tstamp)
AS duration FROM BreadCrumb b GROUP BY b.trip_id
ORDER BY duration DESC LIMIT 1;
  trip_id  | duration
-----------+----------
 227814259 | 05:33:57
(1 row)

postgres=#
```

9. Are there differences in the number of breadcrumbs between a non-holiday Wednesday, a non-holiday Saturday, and a holiday? What can that tell us about TriMet's operations on those types of days?

   <u>For Holiday:</u>

   select count(*) AS count FROM BreadCrumb
   WHERE date(tstamp) = '2023-01-01';

```
postgres=# select count(*) AS count FROM BreadCrumb
WHERE date(tstamp) = '2023-01-01';
 count
--------
 483350
(1 row)
```

Non-holiday Wednesday:
select count(*) AS count FROM BreadCrumb
WHERE date(tstamp) = '2023-01-04';

```
postgres=# select count(*) AS count FROM BreadCrumb
WHERE date(tstamp) = '2023-01-04';
 count
--------
 696300
(1 row)

postgres=#
```

Non-holiday Saturday:
select count(*) AS count FROM BreadCrumb
WHERE date(tstamp) = '2023-01-07';

```
postgres=# select count(*) AS count FROM BreadCrumb
WHERE date(tstamp) = '2023-01-07';
 count
-------
 15685
(1 row)
```

10. Devise three new, interesting questions about the TriMet bus system that can be answered by your breadcrumb data. Show your questions, their answers, the SQL you used to get the answers and the results of running the SQL queries on your data (the number of result rows, and first five rows returned).

Question-1: What is the average speed of buses during peak hours (7 AM to 9 AM) on weekdays?

```
select avg(b.speed) AS average_speed from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
where extract(dow from b.tstamp)
between 1 and 5  and extract (hour from b.tstamp)
between 7 and 9  and b.speed IS NOT NULL;
```

```
postgres=# select avg(b.speed) AS average_speed from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
where extract(dow from b.tstamp)
between 1 and 5  and extract (hour from b.tstamp)
between 7 and 9  and b.speed IS NOT NULL;
   average_speed
-------------------
 8.773640499376487
(1 row)

postgres=#
```

Question-2: How many different buses were operating during a rainy day December 1, 2023?

```
select count(distinct t.trip_id) AS active_buses
from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
where date(b.tstamp) = '2022-12-26';
```

```
postgres=# select count(distinct t.trip_id) AS active_buses
from BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
where date(b.tstamp) = '2022-12-26';
 active_buses
--------------
         1281
(1 row)

postgres=#
```

Question-3: Find the Starting and Ending Times of a Trip?

```
select t.trip_id, min(b.tstamp) AS start_time,
max(b.tstamp) AS end_time FROM BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
GROUP BY t.trip_id
ORDER BY t.trip_id limit 5;
```

```
postgres=# select t.trip_id, min(b.tstamp) AS start_time,
max(b.tstamp) AS end_time FROM BreadCrumb b
JOIN Trip t ON b.trip_id = t.trip_id
GROUP BY t.trip_id
ORDER BY t.trip_id limit 5;
  trip_id  |     start_time      |      end_time
-----------+---------------------+---------------------
 222204052 | 2022-12-19 06:10:45 | 2022-12-19 06:50:05
 222204074 | 2022-12-19 06:50:10 | 2022-12-19 07:35:06
 222204140 | 2022-12-19 07:43:33 | 2022-12-19 08:29:26
 222204207 | 2022-12-19 08:59:53 | 2022-12-19 09:48:48
 222204269 | 2022-12-19 09:57:03 | 2022-12-19 10:45:08
(5 rows)

postgres=#
```

## **Step-6:** Perform These Queries on Your DB and Report the Results

Query #1:

```
select tstamp::date, count(*)
from breadcrumb
group by tstamp::date
order by tstamp::date ASC;
```

```
kthaniko@datafoundry-sub:~$ sudo -u postgres psql postgres
psql (15.12 (Debian 15.12-0+deb12u2))
Type "help" for help.

postgres=# select tstamp::date, count(*)
from breadcrumb
group by tstamp::date
order by tstamp::date ASC;
   tstamp   | count
------------+--------
 2022-12-19 | 642929
 2022-12-20 | 659921
 2022-12-21 | 685035
 2022-12-22 | 679675
 2022-12-23 | 479900
 2022-12-24 | 499567
 2022-12-25 | 489186
 2022-12-26 | 511496
 2022-12-27 | 685128
 2022-12-28 | 705908
 2022-12-29 | 628167
 2022-12-30 | 716869
 2022-12-31 | 495559
 2023-01-01 | 483350
 2023-01-02 | 505935
 2023-01-03 | 638163
 2023-01-04 | 696300
 2023-01-05 | 677374
 2023-01-06 | 650748
 2023-01-07 |  15685
(20 rows)

postgres=#
```

**GitHub Link:**

https://github.com/kthanikonda/DataEngineering/tree/main/Project/Data%20Validate%20Transform%20and%20Storage