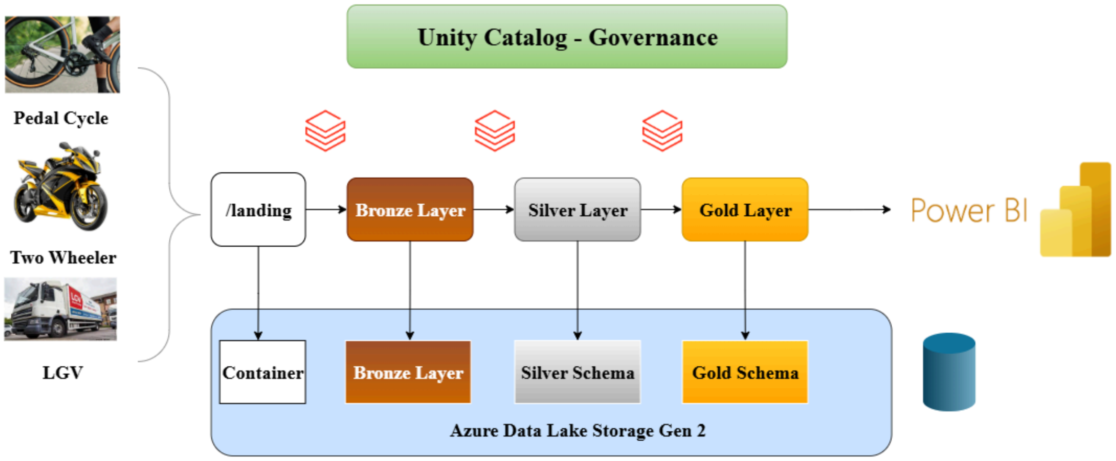


# End-to-End Azure Databricks Lakehouse – Traffic & Roads Analytics

| | |
|---|---|
| Tech Stack | ADLS Gen2 Azure Databricks Power BI Desktop & Service Python Unity Catalog |
| Brief Summary | **Built an end-to-end Azure Databricks lakehouse** on ADLS with **Landing→Bronze→Silver→Gold Delta tables**, using **incremental ingestion** and **new-record-only** transforms → **Result:** single source of truth & **Gold** ready for reporting/data science. |
| Link | https://github.com/khanhmdinh/khanhmdinh.github.io/tree/main/01_End-to-End%20Azure%20Databricks%20Lakehouse%20%E2%80%93%20Traffic%20%26%20Roads%20Analyt |

⭐ **Navigation bar**

# Summary

> **Scope of Work**

- **Datasets:** `traffic` , `roads` .
- **Storage:** Raw files staged in **ADLS Gen2 Landing** (manual drop to simulate ETL).
- **Processing: Databricks notebooks** (PySpark) implement **incremental ingestion** and transformations.
- **Medallion: Landing → Bronze → Silver → Gold** (all **Delta** tables).
- **Governance: Unity Catalog** for catalogs/schemas/tables and permissions.
- **Consumption: Power BI Service** report on Gold.
- **Mode:** Batch project that leverages **Structured Streaming/Auto Loader concepts** for **incremental** behavior (still run in batch for this course).

> **Deliverables**

- **Delta tables:** `bronze.*` , `silver.*` , `gold.*` for **traffic** and **roads**.
- **Databricks notebooks:** for incremental ingestion and Silver/Gold transformations.
- **Power BI report:** built on **Gold** datasets.
- **Unity Catalog artifacts:** catalog/schema/table definitions and permissions.

> **Key Behaviors & Principles**

- **Incremental by design:** Only new files/records move forward each run; **no full reloads**.
- **Idempotent transforms:** Silver logic re-runnable without duplication.
- **Separation of concerns:** Bronze = raw truth, Silver = cleaned/business logic, Gold = serve.

## Table of Contents

# Data Assessment

## Dataset Information

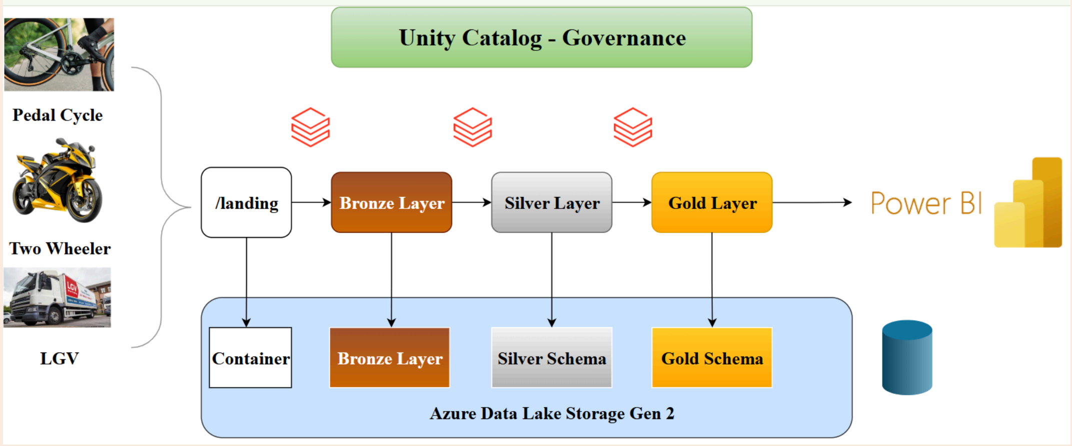
### Dataset 1: Raw Traffic counts dataset (count the types of vehicles that flowed past in a given period in day)

| Category | Column Name | Description |
|---|---|---|
| **Vehicle flow point** | Record ID | Uniquely identifies a record |
| | Count Point ID | A unique reference for the road link |
| | Direction of Travel | Diretion of travel |
| | Year | Year it happened |
| | Count Date | The day when the actual count took place |
| **Travel information of vehicle** | Hour | Hour 7 represents from 7 AM to 8 AM, and 17 tells from 5 PM to 6 PM |
| | Region ID | Website region identifier |
| | Region Name | The name of the Region that travel took place |
| | Local Authority Name | Local authority that region |
| | Road Name | This is the road name (for instance M25 or A3) |
| | Road Category ID | Uniquely identifies road ID |
| | Start junction Road Name | The road name of the start function of the link |
| | End junction Road Name | The road name of the end function of the link |
| | Latitude | Latitude of the Location |
| | Longtitude | Longtitude of the Location |
| | Total Link Length (km) | Total length of the nerwork road link |
| **Count of types of vehicles** | Pedal Cycles | Counts of pedal cycles |
| | Two Wheeler Motor Vehicles | Counts of two wheeled motor vehicles |
| | Car and Taxis | Counts of cars and taxis |
| | Buses and Coaches | Counts of buses and coaches |
| | LGV (Large Goods Vehicles) | Counts of LGV Type |
| | HGV (Heavy Good Vehicles) | Counts of HGV Type |
| | EV Car | Counts of EV Car |
| | EV Bike | Counts of EV Bikes |

### Dataset 2: Raw Roads dataset (the types of roads that got traveled)

| Column Name | Description |
|---|---|
| Road ID | Ordinal number |
| Road Category ID | Uniquely identifies road ID |
| Road Category | The type of road |
| Region ID | Website region identifier |
| Region Name | The name of the Region that travel took place |
| Total Link Length (km) | Total length of the nerwork road link (km) |
| Total Link Length (mile) | Total length of the nerwork road link (mile) |
| Total Motor Vehicles | Counts the total motor vehicles travelled in the particular road in a year |

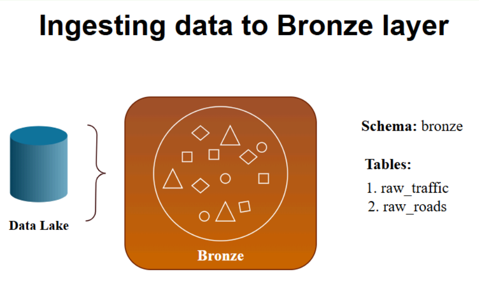
## I. Project Architecture



# II. Landing to Bronze Layer with Auto Loader

## 1. Overview

Implemented a **reliable landing-to-Bronze ingestion pattern** on **Azure Databricks** using **Auto Loader (cloudFiles)**. The pipeline reads CSV files from a **Landing** container, enforces an **explicit schema**, adds an `extract_time` lineage column, and writes to **Bronze Delta tables** with proper **checkpointing** and **batch-style triggers** (`availableNow`).



## 2. What I built

- **Read function** for `raw_traffic` **&** `raw_roads` (Auto Loader, CSV, explicit schema).
- **Write function** to append into `bronze_raw_traffic` & `bronze.raw_roads` (Delta) with `availableNow` trigger.
- Independent **schemaLocation** and **checkpointLocation** to avoid cross-contamination with traffic.
- Added **operational observability** (query names, checkpoints, and run messages).

## 3. Key Design Decisions

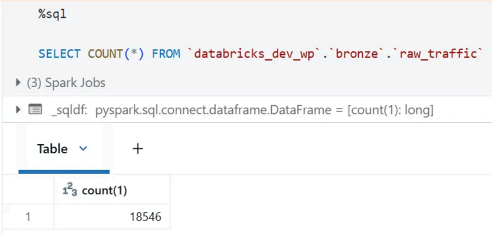
- **Auto Loader (cloudFiles)** for scalable file discovery and schema management.
- **Explicit schema** (StructType) instead of inference for stability and predictable evolution.
- **Separate state for each stream**:
  - Traffic: `<checkpoints>/raw_traffic_load/{schema_info|checkpoint}`
  - Roads: `<checkpoints>/raw_roads_load/{schema_info|checkpoint}`
- **Batch-style execution** with `trigger(availableNow=True)` so the stream **starts → ingests → stops**.
- **Lineage**: `extract_time = current_timestamp()` for auditing and validating incremental behavior.
- **No secrets / no hard-coding**: resolve storage URLs via **Unity Catalog External Locations** or widgets.
- **No extra derived columns** for roads (kept the schema minimal as per requirements).
- **Parameterization**: `env` (dev/test/prod) and UC **External Locations** for URLs (no hard-coding).

## 4. Implementation Sketch

02+Load+to+Bronze.ipynb

## 5. What I validated

- **Incremental behavior**: After uploading a new CSV into `landing/raw_traffic`, re-running the notebook (with `trigger(availableNow=True)`) **doubled** the Bronze row count from **18,546 → 37,092.**



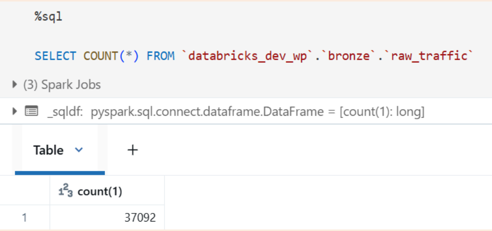
```
%sql

SELECT COUNT(*) FROM `databricks_dev_wp`.`bronze`.`raw_traffic`
```

▸ (3) Spark Jobs

▸ _sqldf: pyspark.sql.connect.dataframe.DataFrame = [count(1): long]

| | count(1) |
|---|---|
| 1 | 18546 |



```
%sql

SELECT COUNT(*) FROM `databricks_dev_wp`.`bronze`.`raw_traffic`
```

▸ (3) Spark Jobs

▸ _sqldf: pyspark.sql.connect.dataframe.DataFrame = [count(1): long]

| | count(1) |
|---|---|
| 1 | 37092 |

- **Lineage proof**: The `extract_time` column differs between pre-existing and newly ingested records, proving micro-batch separation.
- **No reprocessing**: Auto Loader consulted the **checkpoint** and skipped previously ingested files—no duplicates.

## 6. How it works

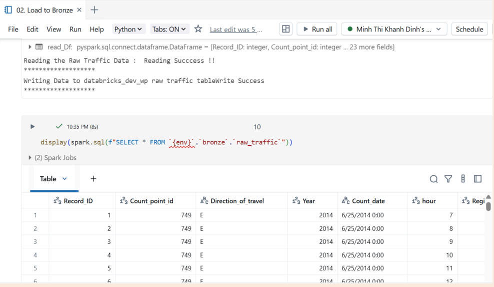
- **Discovery & schema:** `readStream.format("cloudFiles").option("cloudFiles.format","csv").option("cloudFiles.schemaLocation", ...)`
- **State management:** `checkpointLocation` stores progress (last processed file offset), enabling **exactly-once** semantics on retries.
- **Batch-style runs:** `trigger(availableNow=True)` reads the current backlog, **ingests, then stops**—ideal for scheduled daily/bi-daily loads.
- **Audit:** `extract_time = current_timestamp()` stamped at ingestion to verify run boundaries and support operational forensics.

## 7. Operationalization

- **Scheduling**: Use Databricks Jobs or an external orchestrator (e.g., ADF) to run at **08:00 / 20:00** or **file-arrival triggers**.
- **Monitoring**: Observe `queryName` in Spark UI; track row deltas and distinct `extract_time` values post-run.
- **Recovery**: On cluster restarts, **re-initialize** external-location variables; checkpoint ensures idempotent continuation.

## 8. Outputs

- **Both** Bronze tables populated: `bronze.raw_traffic` and `bronze.raw_roads`.



read_Df: pyspark.sql.connect.dataframe.DataFrame = [Record_ID: integer, Count_point_id: integer ... 23 more fields]

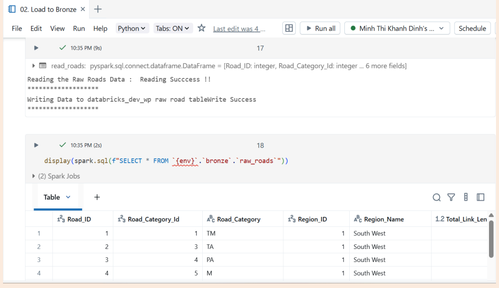
```
Reading the Raw Traffic Data :  Reading Succcess !!
******************
Writing Data to databricks_dev_wp raw traffic tableWrite Success
******************
```

10:35 PM (8s) 10

```
display(spark.sql(f"SELECT * FROM `{env}`.`bronze`.`raw_traffic`"))
```

(2) Spark Jobs

| | Record_ID | Count_point_id | Direction_of_travel | Year | Count_date | hour | Regi |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 749 | E | 2014 | 6/25/2014 0:00 | 7 | |
| 2 | 2 | 749 | E | 2014 | 6/25/2014 0:00 | 8 | |
| 3 | 3 | 749 | E | 2014 | 6/25/2014 0:00 | 9 | |
| 4 | 4 | 749 | E | 2014 | 6/25/2014 0:00 | 10 | |
| 5 | 5 | 749 | E | 2014 | 6/25/2014 0:00 | 11 | |
| 6 | 6 | 749 | E | 2014 | 6/25/2014 0:00 | 12 | |



10:35 PM (9s) 17

read_roads: pyspark.sql.connect.dataframe.DataFrame = [Road_ID: integer, Road_Category_Id: integer ... 6 more fields]

```
Reading the Raw Roads Data :  Reading Succcess !!
******************
Writing Data to databricks_dev_wp raw road tableWrite Success
******************
```

10:35 PM (2s) 18

```
display(spark.sql(f"SELECT * FROM `{env}`.`bronze`.`raw_roads`"))
```

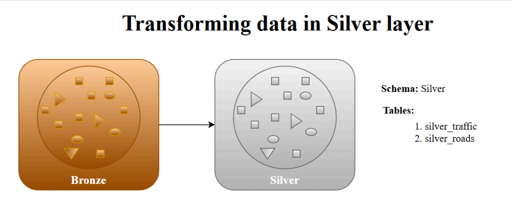
(2) Spark Jobs

| | Road_ID | Road_Category_Id | Road_Category | Region_ID | Region_Name | Total_Link_Len |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | TM | 1 | South West | |
| 2 | 2 | 3 | TA | 1 | South West | |
| 3 | 3 | 4 | PA | 1 | South West | |
| 4 | 4 | 5 | M | 1 | South West | |

- **Zero custom incremental logic**—Auto Loader + checkpoints delivered **reliable, incremental Bronze loads**.
- **Auditable runs** via `extract_time`; easy to prove what arrived when.
- Ready for **Bronze → Silver** quality rules and downstream modeling.

# III. Bronze to Silver Layer with Incremental DQ & Business Transforms

## 1. Objective



Promote **Bronze Delta** tables—`raw_traffic` and `raw_roads`—to curated **Silver** tables using **Structured Streaming over Delta**. Process **only newly arrived rows** per run, enforce **data quality**, and apply **dataset-specific business logic** that makes the data analysis-ready.

## 2. What I built

- Built the **Silver-Traffic** notebook and parameterized it by **environment** (dev/test/prod).
- Implemented **streaming reads from Delta** for incremental transforms (no reprocessing).
- Codified **duplicate removal**, **null handling**, and **dataset-specific features**.
- Wrote clean, append-only **writeStream → Silver** with checkpoints and run observability.

## 3. Design Highlights

- **Defense-in-depth DQ:**
  - **Duplicates:** `dropDuplicates()` on stream to prevent downstream cardinality issues.
  - **NULL:** Replace **string** nulls with `"Unknown"` and **numeric** nulls with `0` using `fillna`.
  - **Renaming Columns:** replacing with an underscore quoting if the dataset is coming without having any of the underscores
  - **Creating Electric Vehicles Count:** `EV_Bike` → `Electric_Vehicles_Count` / `Motor_Vehicles_Count`
- **Business features:**
  - `electric_vehicles_count = ev_car + ev_bike`
  - `motor_vehicles_count = cars + buses + lgv + hgv + electric_vehicles_count`
  - `transform_time = current_timestamp()` for lineage & auditability.

## 4. Implementation Sketch

03+Silver+Traffic+Transformations.ipynb

04. Common notebook.ipynb

05+Silver+Road+Transformations.ipynb

## 5. Data Quality

- **Duplicates:** Remove using `dropDuplicates()` to protect downstream keys/cardinality.
- **Nulls:**
  - String columns → replace with `"Unknown"`
  - Numeric columns → replace with `0`

- **Column hygiene:** Column names use underscores (handled during Bronze ingestion via explicit schemas).
- **Lineage:** Add `transform_time = current_timestamp()` in Silver for auditability.

## 6. Business Transformations

- **Traffic → Silver**
  - `electric_vehicles_count = ev_car + ev_bike`
  - `motor_vehicles_count = cars + buses + lgv + hgv + electric_vehicles_count`
- **Roads → Silver**
  - `road_category_name` from `road_category` code (e.g., `TO → "Class A Trunk Road"`, `TM → "Class A Trunk Motor"`, `PA → "Class A Principal Road"`, `PM → "Motorway"`, `M → "Class B Road"`, else `NA`)
  - `road_type` from `road_category_name` (`LIKE '%Class A%' → "Major"`, `LIKE '%Class B%' → "Minor"`, else `NA`)

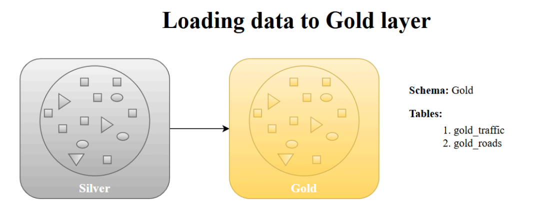
## 7. Outputs

- **Silver table created:**
  - **`silver.silver_traffic`**: cleaned, deduped, with `electric_vehicles_count`, `motor_vehicles_count`, `transform_time`.
  - **`silver.silver_roads`**: cleaned, deduped, with `road_category_name`, `road_type`, `transform_time`.
- **Incremental by design:** Only new Bronze rows are transformed per run (micro-batches + checkpoints).

# IV. Gold Layer: Transformations & Curated Tables

## 1. Objective

Read curated **Silver** tables, apply **light, business-oriented transforms**, and publish **Gold** tables for reporting. Keep processing **incremental** and **idempotent** using Structured Streaming over Delta with checkpointing.



## 2. Inputs

- `silver.silver_traffic`: already cleaned, includes `motor_vehicles_count`, `electric_vehicles_count`, `transform_time`.
- `silver.silver_roads`: already cleaned, includes `road_category_name`, `road_type`, `length_km`, `transform_time`.

## 3. Gold transforms (minimal, transcript-aligned)

- **Traffic:**
  - `vehicle_intensity` (a.k.a. density) = `motor_vehicles_count / length_km`.
  - `load_time = current_timestamp()` (lineage at Gold load).
- **Roads:**
  - No new business calc required here; add `load_time` for lineage.

## 4. Implementation Sketch

06+Final+Transformations.ipynb

## 5. Outputs

- **Created Gold tables:**
  - `${env}.gold.gold_traffic` with `vehicle_intensity` & `load_time`.
  - `${env}.gold.gold_roads` with `load_time`.
- **Operational posture:** Incremental, checkpointed, and environment-aware; ready to power semantic models and BI.

# V. Orchestration with Azure Databricks Workflows

## 1. Goal

Automate the **end-to-end pipeline** (Landing → Bronze → Silver → Gold) by chaining our notebooks in a **dependency-aware** job, parameterized by environment and backed by checkpoints for incremental, idempotent runs.

## 2. What I orchestrated

**Task graph (DAG)** — executed in order on success:

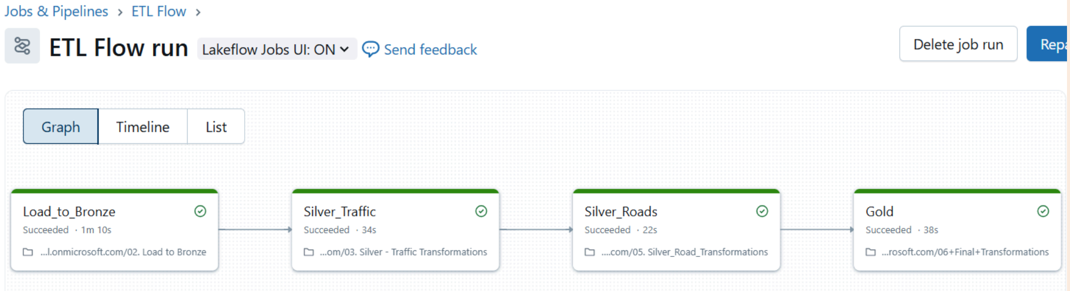
1. **Load to Bronze** — Auto Loader ingests *raw_traffic* and *raw_roads* from Landing → `bronze.*`
2. **Silver: Traffic** — incremental DQ + business transforms → `silver.silver_traffic`
3. **Silver: Roads** — incremental DQ + business transforms → `silver.silver_roads`
4. **Gold: Finalize** — light enrichments → `gold.gold_traffic`, `gold.gold_roads`

> Common / Config notebook is imported at the top of every task to centralize paths, helpers, and conventions:
>
> `%run /Workspace/<path>/common`

## 3. Key design choices

- **Single-source configuration:** `%run` (same Spark session) exposes **locations** (`landing`, `checkpoints`, `bronze`, `silver`, `gold`) and **utilities (**`remove_dupes`, `handle_nulls`).
- **Environment parameterization:** Each notebook reads `env` via widgets so the Workflow can pass `env=dev|test|prod`.
- **Incremental by default:** Bronze and Silver are **streaming over Delta** with **checkpointed** `writeStream` and `trigger(availableNow=True)`.
- **Failure isolation:** Downstream tasks run **only if predecessors succeed**; failures halt the DAG for fast feedback.

## 4. DAG Screenshot

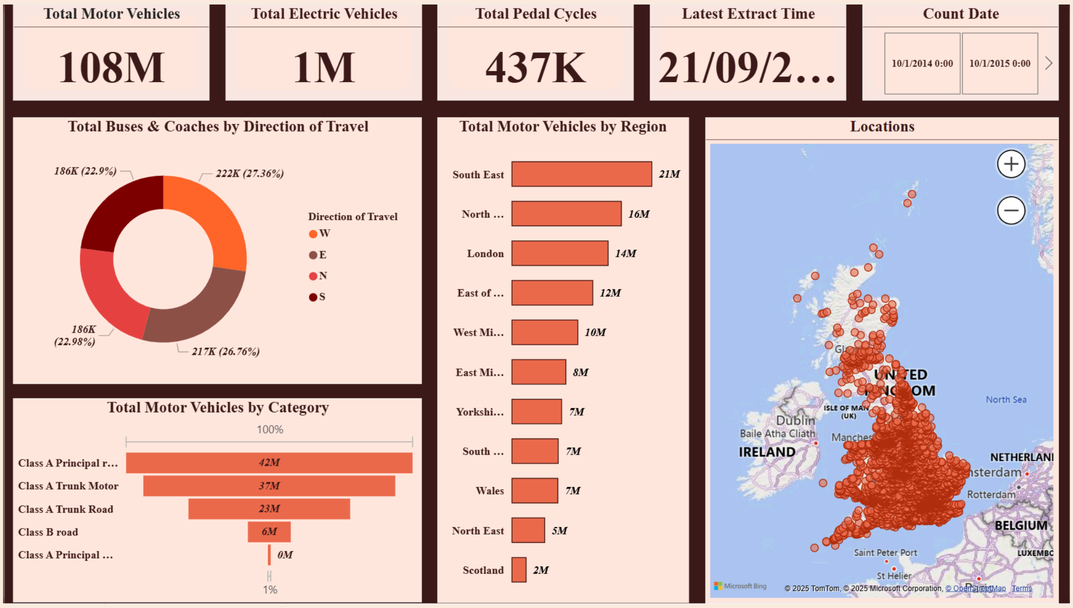

Jobs & Pipelines › ETL Flow ›
ETL Flow run
Lakeflow Jobs UI: ON
Send feedback
Delete job run
Rep
Graph
Timeline
List
Load_to_Bronze
Succeeded · 1m 10s
...l.onmicrosoft.com/02. Load to Bronze
Silver_Traffic
Succeeded · 34s
...om/03. Silver - Traffic Transformations
Silver_Roads
Succeeded · 22s
....com/05. Silver_Road_Transformations
Gold
Succeeded · 38s
...rosoft.com/06+Final+Transformations

## 5. Results

- One-click or scheduled workflows execute the entire **Landing → Bronze → Silver → Gold** flow **reliably**, in minutes on dev.
- After a test run (new files uploaded to Landing), **row counts increased across all layers** and Gold tables reflected new **lineage timestamps** (`extract_time`, `transform_time`, `load_time`), demonstrating incremental behavior end-to-end.

# Insights & Actions

**View the Live Dashboard:** https://app.powerbi.com/reportEmbed?reportId=3febcad3-e7d0-4354-b7b5-531a8d428863&autoAuth=true&ctid=dcca8464-f86f-44ac-a6dd-7032f818fe7b



## 1) Network Pressure → Hotspot Playbook

- **What we see:** Gold's `vehicle_intensity` highlights the links/corridors with the highest load per kilometre—the true pressure points to act on first.
- **Action:** Rank links by `vehicle_intensity`; treat the **top 5–10%** with a **Hotspot Playbook** (signal retiming, lane priority, incident-response rules).
- **Measure:** Track **p95/median** `vehicle_intensity` on treated links before/after using **Gold** snapshots (keyed by `load_time`).

## 2) Peak Windows → Time-Aligned Operations

- **What we see:** The traffic schema captures **hourly granularity** (e.g., hour 7 = 7–8 AM), enabling precise identification of peak windows by region/corridor.
- **Action:** Align staffing and **signal timing** to local peaks; schedule maintenance in **off-peak** windows per region.
- **Measure:** Reduction in **peak-window** `vehicle_intensity`; improved maintenance timeliness.

## 3) Investment Targeting → Spend Follows Pressure

- **What we see: Road categories/types** are derived in **Silver** (e.g., `road_type`), allowing intensity to be compared **per km** by class.
- **Action:** Rebalance **CapEx/Opex** toward categories with **persistently higher intensity/km**; defer lower-pressure segments.
- **Measure: Intensity/km** decreases in prioritized categories over successive Gold refreshes.

## 4) EV Overlay → Charger Placement Where It Matters

- **What we see:** `electric_vehicles_count` and `motor_vehicles_count` are standardized in **Silver**, enabling **EV share** views across corridors.
- **Action:** Pilot charger placement on **high-adoption, high-pressure** corridors to reduce cruising for charge and support growth.

- **Measure: EV share** uplift and **charger utilization** at pilot sites, without raising local intensity.

### 5) Proven, Auditable Incrementality → Operational Trust

- **What we proved:** A controlled test **doubled Bronze rows from 18,546 → 37,092** after a new CSV drop; distinct `extract_time` stamps and checkpointing prevented re-ingest/duplicates—validating **incremental, exactly-once** flow through Bronze→Silver→Gold.
- **Action:** Set refresh **SLAs** and add "delta since last load" tiles in the report to reassure stakeholders.
- **Measure:** Consistent **row deltas** and lineage timestamps (`extract_time` / `transform_time` / `load_time`) visible per run.

🌟 **Navigation bar**