Data Engineering Labs

Lab 1 & Lab 2 — Reflective Report

February 2026

Salma Benslimane
Zakaria Limi

1 Final Architecture

The final pipeline is composed of two distinct layers: a Python ingestion layer (Lab 1) and a dbt + DuckDB transformation and serving layer (Lab 2). The Python scraper collects raw app metadata and reviews from the Google Play Store API and persists them as immutable JSON and JSONL files. From there, dbt takes over and manages all transformations, data quality, and serving.

The dbt pipeline is organized into three logical layers:

- **Source models (src_*)**: expose raw JSON/JSONL files as queryable DuckDB relations using `read_json_auto()` and `read_ndjson_auto()` — no transformations applied.
- **Staging models (stg_*)**: rename columns to snake_case, cast data types, generate surrogate keys via `md5()`, filter null primary keys, and handle reserved SQL keywords.
- **Mart models (dims + facts)**: implement the Kimball star schema with `dim_apps`, `dim_categories`, `dim_developers`, `dim_date`, `fact_reviews` (incremental), and `fact_reviews_historical` (SCD2 point-in-time join).

1.1 Additional Features Implemented

Incremental Loading (fact_reviews)

`fact_reviews` is materialized as an incremental dbt model with:

```
{{ config(materialized='incremental', unique_key='review_sk') }}
```

On first run, the full dataset is loaded. On subsequent runs, only reviews with `reviewed_at` greater than the maximum already in the table are processed, preventing recomputation and duplicates.

SCD Type 2 (snap_apps + dim_apps_scd)

A dbt snapshot tracks historical changes using the check strategy on mutable attributes such as category, score, installs, and price. If a value changes, dbt closes the previous row and inserts a new version.

Historical Fact Table

`fact_reviews_historical` performs a point-in-time join between reviews and the SCD dimension to ensure historical correctness when app attributes change.

2 Implementation Details

2.1 Incremental Loading

Incremental loading is implemented using dbt's native incremental materialization. The filter:

```
{% if is_incremental() %}
where reviewed_at > (select max(reviewed_at) from {{ this }})
{% endif %}
```

This enables daily ingestion without reprocessing the full history.

2.2 SCD Type 2

Snapshots compare selected columns at each run. If differences are detected, dbt:

- Sets `dbt_valid_to` on the old record
- Inserts a new record with `dbt_valid_to = NULL`

The SCD dimension adds:

- A version-scoped surrogate key
- `valid_from`, `valid_to`
- `is_current` flag

2.3 Data Quality & Testing

Data quality is enforced using dbt generic tests across three layers.

Staging

- `not_null`, `unique`
- Referential integrity
- Accepted values (score 1–5)

Dimensions

- PK integrity
- Referential integrity
- Validity columns not null

Facts

- Unique review surrogate key
- Foreign key validation
- Accepted score range

In Lab 1, data cleaning was enforced manually in Python using explicit coercion for numeric values, timestamps, and null replacements.

3 Python-Only vs dbt-Based Pipeline Comparison

Dimension	**Python-Only (Lab 1)**	**dbt + DuckDB (Lab 2)**
Transformation Logic	Ad-hoc Python scripts	Declarative SQL models
Data Quality	Manual checks	Native dbt tests
Incremental Loading	Full rewrite	Incremental materialization
Schema Drift	Manual updates required	Handled at source model boundary
Historical Tracking	Not implemented	Snapshot-based SCD2
Dependency Management	Implicit ordering	Explicit DAG via ref()
Serving Layer	CSV + Streamlit	Direct DuckDB querying

The most significant structural difference is explicit dependency management through `ref()`, ensuring deterministic builds and enforced execution order.

4 Reflections

4.1 Most Fragile Part

The Python transformation layer in Lab 1 was the most fragile due to silent schema drift failures. Pandas allowed mismatched columns to propagate NaN values without raising exceptions.

In dbt, schema drift produces explicit compilation errors at the source model layer, preventing silent corruption.

4.2 Biggest Architectural Insight

The transformation layer must be the sole schema normalization boundary. Downstream layers should operate only on canonical internal schemas.

The difference between `fact_reviews` and `fact_reviews_historical` illustrates the importance of point-in-time joins for historical correctness.

4.3 One Design Decision to Change

Using `md5()` on single natural keys is simple but not robust for large-scale production systems. A stronger surrogate key strategy (e.g., dbt utilities or UUID-based keys) would improve collision resistance and historical tracking robustness.

5 Dashboard Screenshots

(On github repo too)

Screenshot 1 — App Performance

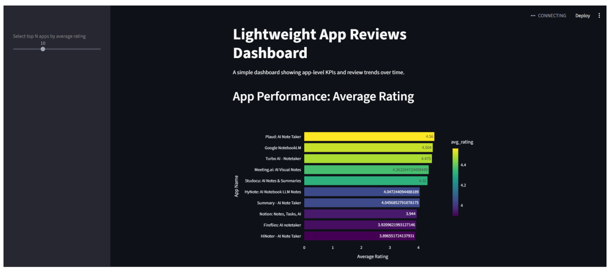
Top 10 AI note-taking apps ranked by average rating. This answers the business question: Which applications perform best or worst according to user reviews?

Screenshot 2 — Average Ratings Over Time

Daily average rating from 2023–2026. Early volatility reflects low review volume; later stability reflects increased sample size.

Screenshot 3 — Daily Review Volume

Daily review counts show recency bias due to newest-first scraping configuration.



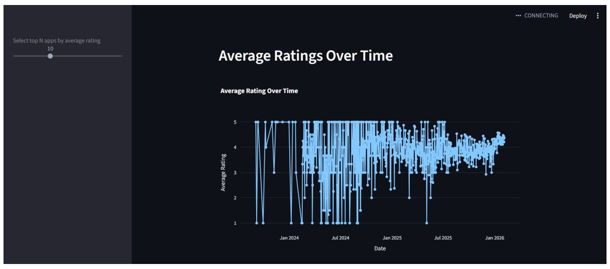
Lightweight App Reviews Dashboard

A simple dashboard showing app-level KPIs and review trends over time.

App Performance: Average Rating

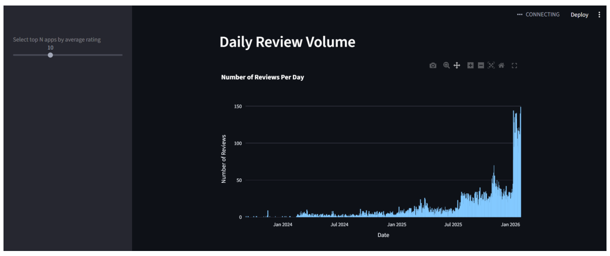
Select top N apps by average rating
10
CONNECTING
Deploy
Plaud: AI Note Taker 4.56
Google NotebookLM 4.504
Turbo AI - Notetaker 4.475
Meeting.ai: AI Visual Notes 4.362204724409449
Studocu: AI Notes & Summaries 4.32
HyNote: AI Notebook LLM Notes 4.047244094488189
Summary - AI Note Taker 4.0456852791878175
Notion: Notes, Tasks, AI 3.944
Fireflies: AI notetaker 3.9209621993127146
HiNoter - AI Note Taker 3.896551724137931
App Name
Average Rating
avg_rating
0
1
2
3
4
4.2
4.4

Figure 1: App Performance



Average Ratings Over Time

Figure 2: Average Ratings Over Time



Daily Review Volume

Figure 3: Daily Review Volume