Data Engineering Labs 1 & 2
Final Report

Adam KHALI , Ilyas DAHAOUI

February 2026

Contents

1 Introduction

This report presents the work completed in Lab 1 and Lab 2 of the Data Engineering course. Lab 2 focused on redesigning the transformation layer of the pipeline using dbt and DuckDB in order to improve modularity, testability, and analytical modeling practices.

The formal objectives of the second lab were:

- Install and configure the development environment (reusing the Python virtual environment created for Lab 1).
- Install DuckDB, dbt-core, and the dbt-duckdb adapter, and verify the installation (`dbt -version` should list DuckDB in the plugin section).
- Explore analytical data modeling, with emphasis on dimensional modeling and star schema design.
- Structure a transformation pipeline using dbt models and a layered architecture (staging and marts).
- Implement data quality testing using dbt and complement it with Python-based validation.
- Prepare a stable serving layer backed by DuckDB for downstream BI tools.

These objectives extend the Python-based pipeline developed in Lab 1, which ingested Google Play Store application and review data, performed cleaning operations, and produced analytics-ready outputs. Lab 2 refactors the transformation stage using dbt and DuckDB while retaining Python for extraction.

2 Lab 1: Python Data Pipeline

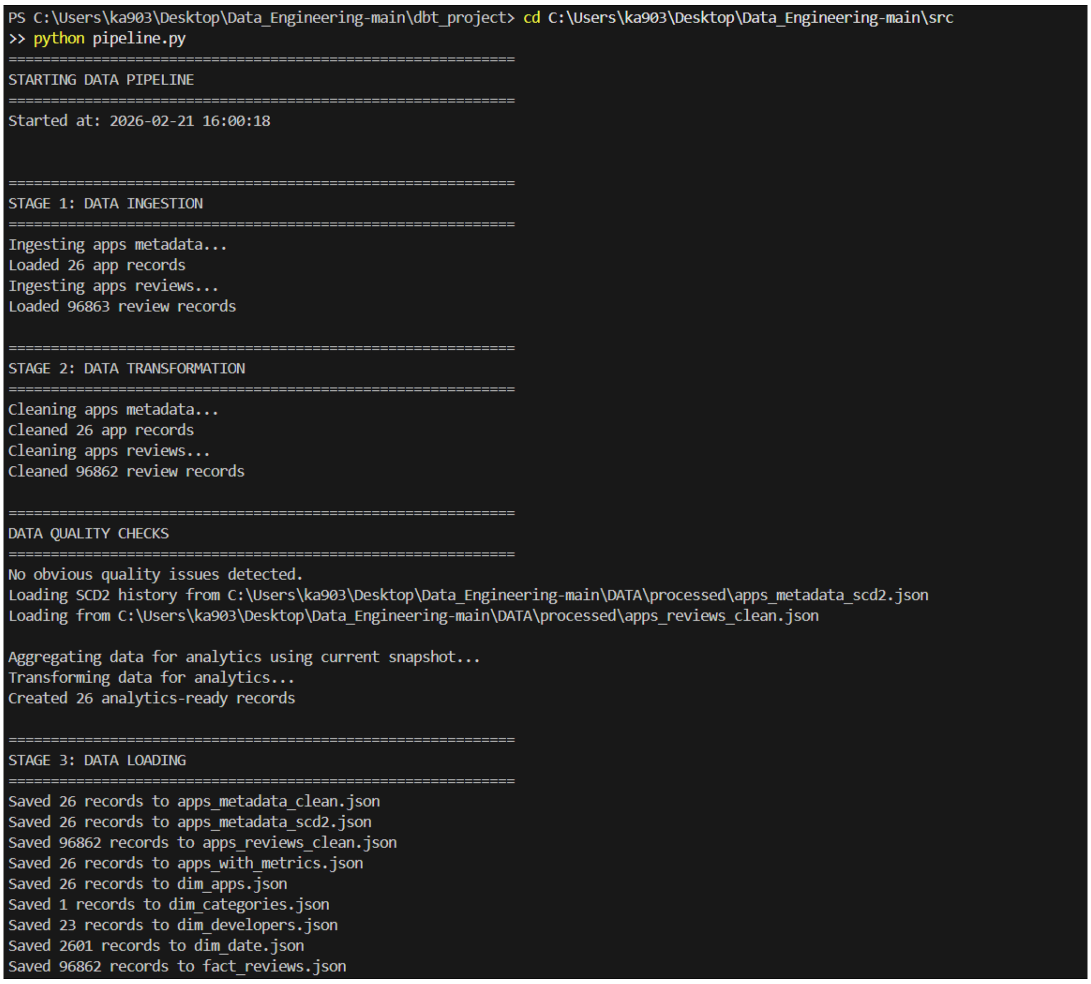
2.1 Architecture

The Lab 1 pipeline was implemented entirely in Python. It consisted of:

- Data ingestion from JSON files produced by a Google Play scraping tool.
- Cleaning and standardization of application metadata and reviews.
- Aggregation of review metrics and joins with application dimensions.
- Writing processed JSON outputs for downstream analysis.

Figure 1 illustrates the high-level architecture.

In Lab 2, the architecture evolved: Python continues to handle extraction and initial loading, while transformation and modeling are delegated to dbt. All data is stored in a DuckDB database file. dbt manages model dependencies, executes tests, and materializes tables. DuckDB serves as a unified analytical storage layer that can be queried directly by BI tools such as PowerBI or Metabase without additional ETL steps.



```
PS C:\Users\ka903\Desktop\Data_Engineering-main\dbt_project> cd C:\Users\ka903\Desktop\Data_Engineering-main\src
>> python pipeline.py
============================================================
STARTING DATA PIPELINE
============================================================
Started at: 2026-02-21 16:00:18


============================================================
STAGE 1: DATA INGESTION
============================================================
Ingesting apps metadata...
Loaded 26 app records
Ingesting apps reviews...
Loaded 96863 review records

============================================================
STAGE 2: DATA TRANSFORMATION
============================================================
Cleaning apps metadata...
Cleaned 26 app records
Cleaning apps reviews...
Cleaned 96862 review records

============================================================
DATA QUALITY CHECKS
============================================================
No obvious quality issues detected.
Loading SCD2 history from C:\Users\ka903\Desktop\Data_Engineering-main\DATA\processed\apps_metadata_scd2.json
Loading from C:\Users\ka903\Desktop\Data_Engineering-main\DATA\processed\apps_reviews_clean.json

Aggregating data for analytics using current snapshot...
Transforming data for analytics...
Created 26 analytics-ready records

============================================================
STAGE 3: DATA LOADING
============================================================
Saved 26 records to apps_metadata_clean.json
Saved 26 records to apps_metadata_scd2.json
Saved 96862 records to apps_reviews_clean.json
Saved 26 records to apps_with_metrics.json
Saved 26 records to dim_apps.json
Saved 1 records to dim_categories.json
Saved 23 records to dim_developers.json
Saved 2601 records to dim_date.json
Saved 96862 records to fact_reviews.json
```

Figure 1: Lab 1 Python pipeline architecture

2.2 Data Quality Analysis

An exploratory analysis in Lab 1 revealed several issues:

1. Missing or null fields (developer, updated, etc.).
2. Inconsistent data types (installs stored as strings, malformed timestamps).
3. Large JSONL review files causing memory pressure.
4. Duplicate and orphaned reviews were not detected.
5. Lack of validation leading to silent failures.

Midway through the lab, four CSV files were dropped into the raw folder: `note_taking_ai_reviews_` (a second review batch including deliberate duplicate `reviewIds`), `note_taking_ai_reviews_dirty.cs` (rows with the string "five" in the score column, negative values, invalid timestamps, and NULL markers), `note_taking_ai_reviews_schema_drift.csv` (alternate column names such as `appTitle` and `rating`), and `note_taking_ai_apps_updated.csv`, which provided updated application metadata with missing values and renamed fields.

These files were used to test deduplication, schema drift handling, and robust type parsing. The ingestion logic was extended to automatically scan the raw directory for

CSV files, read them generically, and normalize column names and values, demonstrating resilience to evolving data sources.

2.3 Pipeline Fragility

The most fragile component was the absence of deduplication logic for reviews. Re-running the extraction step appended duplicate records, corrupting aggregated metrics. Hard-coded field names also made the pipeline vulnerable to schema changes.

3 Enhancements Introduced in Lab 2

3.1 Incremental Loading

A `merge_reviews` function was implemented to merge new review records with existing processed data using `review_id` as a key. This enabled idempotent pipeline execution.

3.2 Slowly Changing Dimensions (SCD Type 2)

Application metadata changes are now tracked using SCD Type 2 logic. The history table includes:

- `start_date`
- `end_date`
- `current_flag`

This enables historical analysis of attribute changes over time.

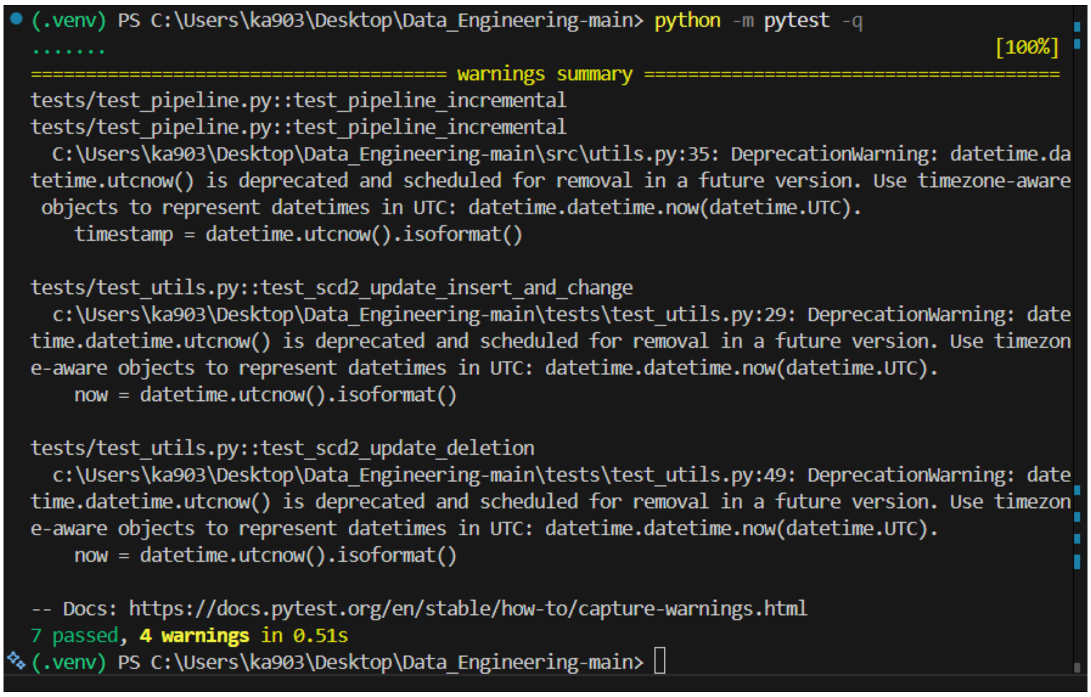
3.3 Data Quality and Testing

A new module `quality.py` performs validation checks:

- Missing identifiers
- Type mismatches
- Ratings outside the 1–5 range

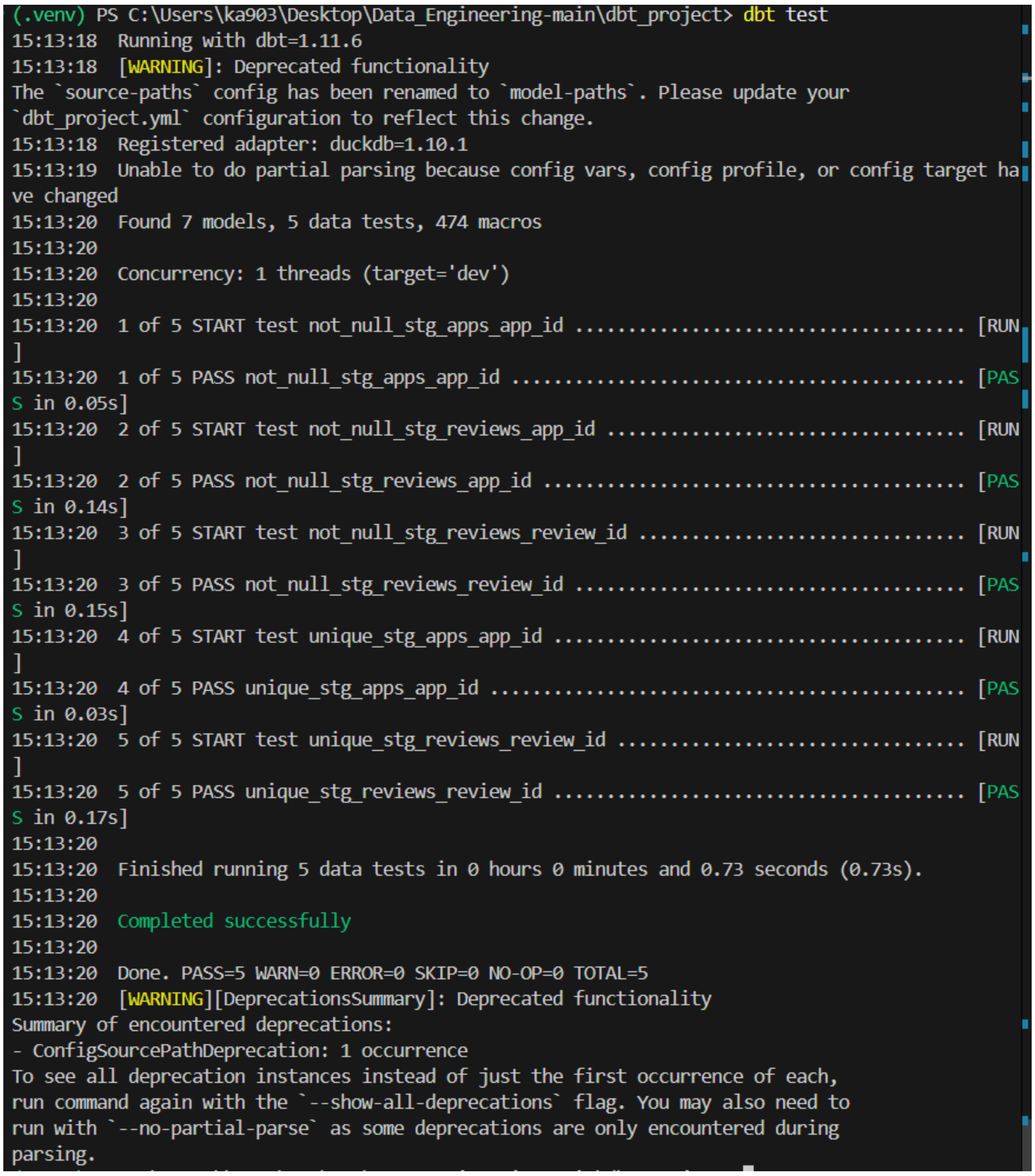
The pipeline halts on critical failures. PyTest covers utility functions, ingestion logic, and schema drift scenarios.

Figure 4 shows a typical execution of the Python pipeline after processing the supplementary CSV files, including data quality messages and record counts.



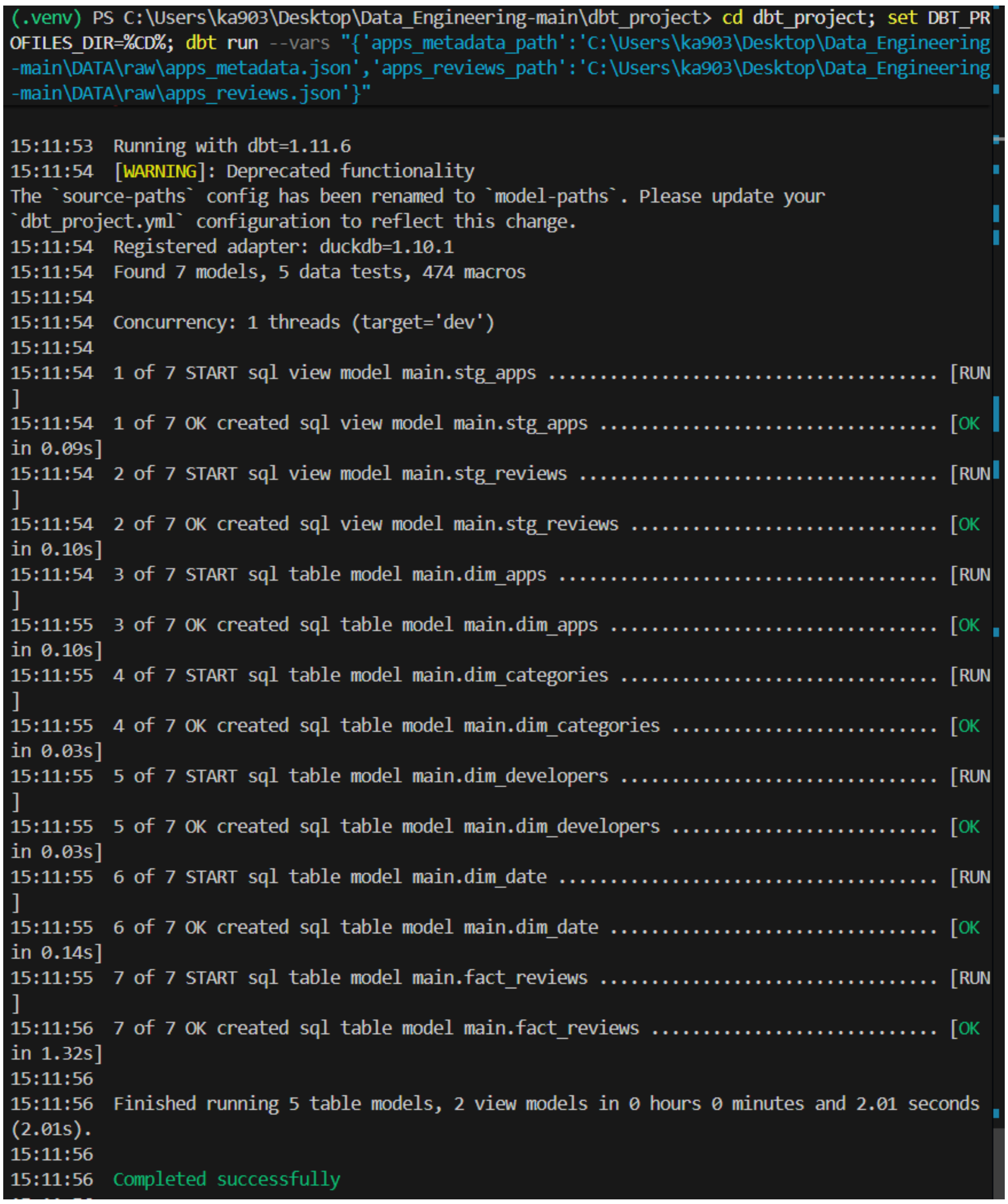
```
(.venv) PS C:\Users\ka903\Desktop\Data_Engineering-main> python -m pytest -q
.......                                                                  [100%]
====================================== warnings summary ======================================
tests/test_pipeline.py::test_pipeline_incremental
tests/test_pipeline.py::test_pipeline_incremental
  C:\Users\ka903\Desktop\Data_Engineering-main\src\utils.py:35: DeprecationWarning: datetime.da
tetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware
 objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    timestamp = datetime.utcnow().isoformat()

tests/test_utils.py::test_scd2_update_insert_and_change
  c:\Users\ka903\Desktop\Data_Engineering-main\tests\test_utils.py:29: DeprecationWarning: date
time.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezon
e-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    now = datetime.utcnow().isoformat()

tests/test_utils.py::test_scd2_update_deletion
  c:\Users\ka903\Desktop\Data_Engineering-main\tests\test_utils.py:49: DeprecationWarning: date
time.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezon
e-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    now = datetime.utcnow().isoformat()

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
7 passed, 4 warnings in 0.51s
(.venv) PS C:\Users\ka903\Desktop\Data_Engineering-main>
```

Figure 2: PyTest output demonstrating all unit and pipeline tests passing



```
(.venv) PS C:\Users\ka903\Desktop\Data_Engineering-main\dbt_project> dbt test
15:13:18  Running with dbt=1.11.6
15:13:18  [WARNING]: Deprecated functionality
The `source-paths` config has been renamed to `model-paths`. Please update your
`dbt_project.yml` configuration to reflect this change.
15:13:18  Registered adapter: duckdb=1.10.1
15:13:19  Unable to do partial parsing because config vars, config profile, or config target ha
ve changed
15:13:20  Found 7 models, 5 data tests, 474 macros
15:13:20
15:13:20  Concurrency: 1 threads (target='dev')
15:13:20
15:13:20  1 of 5 START test not_null_stg_apps_app_id ................................... [RUN
]
15:13:20  1 of 5 PASS not_null_stg_apps_app_id ......................................... [PAS
S in 0.05s]
15:13:20  2 of 5 START test not_null_stg_reviews_app_id ................................ [RUN
]
15:13:20  2 of 5 PASS not_null_stg_reviews_app_id ...................................... [PAS
S in 0.14s]
15:13:20  3 of 5 START test not_null_stg_reviews_review_id ............................. [RUN
]
15:13:20  3 of 5 PASS not_null_stg_reviews_review_id ................................... [PAS
S in 0.15s]
15:13:20  4 of 5 START test unique_stg_apps_app_id ..................................... [RUN
]
15:13:20  4 of 5 PASS unique_stg_apps_app_id ........................................... [PAS
S in 0.03s]
15:13:20  5 of 5 START test unique_stg_reviews_review_id ............................... [RUN
]
15:13:20  5 of 5 PASS unique_stg_reviews_review_id ..................................... [PAS
S in 0.17s]
15:13:20
15:13:20  Finished running 5 data tests in 0 hours 0 minutes and 0.73 seconds (0.73s).
15:13:20
15:13:20  Completed successfully
15:13:20
15:13:20  Done. PASS=5 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=5
15:13:20  [WARNING][DeprecationsSummary]: Deprecated functionality
Summary of encountered deprecations:
- ConfigSourcePathDeprecation: 1 occurrence
To see all deprecation instances instead of just the first occurrence of each,
run command again with the `--show-all-deprecations` flag. You may also need to
run with `--no-partial-parse` as some deprecations are only encountered during
parsing.
```

Figure 3: dbt test output showing data quality checks passing



```
(.venv) PS C:\Users\ka903\Desktop\Data_Engineering-main\dbt_project> cd dbt_project; set DBT_PR
OFILES_DIR=%CD%; dbt run --vars "{'apps_metadata_path':'C:\Users\ka903\Desktop\Data_Engineering
-main\DATA\raw\apps_metadata.json','apps_reviews_path':'C:\Users\ka903\Desktop\Data_Engineering
-main\DATA\raw\apps_reviews.json'}"

15:11:53  Running with dbt=1.11.6
15:11:54  [WARNING]: Deprecated functionality
The `source-paths` config has been renamed to `model-paths`. Please update your
`dbt_project.yml` configuration to reflect this change.
15:11:54  Registered adapter: duckdb=1.10.1
15:11:54  Found 7 models, 5 data tests, 474 macros
15:11:54
15:11:54  Concurrency: 1 threads (target='dev')
15:11:54
15:11:54  1 of 7 START sql view model main.stg_apps ................................... [RUN
]
15:11:54  1 of 7 OK created sql view model main.stg_apps .............................. [OK
in 0.09s]
15:11:54  2 of 7 START sql view model main.stg_reviews ................................ [RUN
]
15:11:54  2 of 7 OK created sql view model main.stg_reviews ........................... [OK
in 0.10s]
15:11:54  3 of 7 START sql table model main.dim_apps .................................. [RUN
]
15:11:55  3 of 7 OK created sql table model main.dim_apps ............................. [OK
in 0.10s]
15:11:55  4 of 7 START sql table model main.dim_categories ............................ [RUN
]
15:11:55  4 of 7 OK created sql table model main.dim_categories ....................... [OK
in 0.03s]
15:11:55  5 of 7 START sql table model main.dim_developers ............................ [RUN
]
15:11:55  5 of 7 OK created sql table model main.dim_developers ....................... [OK
in 0.03s]
15:11:55  6 of 7 START sql table model main.dim_date .................................. [RUN
]
15:11:55  6 of 7 OK created sql table model main.dim_date ............................. [OK
in 0.14s]
15:11:55  7 of 7 START sql table model main.fact_reviews .............................. [RUN
]
15:11:56  7 of 7 OK created sql table model main.fact_reviews ......................... [OK
in 1.32s]
15:11:56
15:11:56  Finished running 5 table models, 2 view models in 0 hours 0 minutes and 2.01 seconds
(2.01s).
15:11:56
15:11:56  Completed successfully
```

Figure 4: Log output from running the Python pipeline end-to-end

4 dbt & DuckDB-Based Pipeline (Lab 2)

4.1 Staging and Dimensional Modeling

The dbt project follows a layered structure with a `staging` folder for initial data ingestion and a `marts` folder for dimensional models. Staging models read raw JSON/CSV files using DuckDB's `read_json_auto` or `read_csv_auto` functions, rename columns, cast types, and filter malformed rows. These models are materialized as views and validated using schema tests (not null, unique, referential) defined in `schema.yml`.

The downstream mart layer implements the star schema. Each dimension selects from staging tables, generates surrogate keys, and ensures consistent definitions across the pipeline. A reusable date dimension is created via a recursive query. The fact table joins cleaned reviews to the dimensions using natural keys and filters out rows lacking valid foreign keys.

4.2 Incremental Models

The fact model was converted to an incremental model using:

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

The model uses an `is_incremental()` condition to filter rows based on the maximum existing review date in the target table, ensuring that only new records are processed during subsequent runs.

4.3 Snapshots for SCD2

dbt snapshots were used to track application metadata changes. Snapshots generate:

- `dbt_valid_from`
- `dbt_valid_to`

A downstream model derives an `is_current` flag for easier historical analysis.

5 Conclusion

These labs provided hands-on experience designing a complete data pipeline using Python and refactoring the transformation layer with dbt and DuckDB. The resulting architecture demonstrates improved modularity, reproducibility, scalability, and alignment with modern data engineering practices.