

Data Engineering Labs 1 & 2

Final Report

Student Name

February 2026

Contents

1	Introduction	2
2	Lab 1: Python Data Pipeline	2
2.1	Architecture	2
2.2	Data Quality Analysis	3
2.3	Pipeline Fragility	3
3	Enhancements Added in Lab 2	4
3.1	Incremental Loading	4
3.2	Slowly Changing Dimension (SCD2)	4
3.3	Data Quality and Testing	4
3.4	Star Schema Export	4
4	dbt & DuckDB-based Pipeline (Lab 2)	5
4.1	Environment Setup	5
4.2	Project Structure	6
4.3	Execution	8
4.4	Data Quality Tests	8
4.5	Serving Layer	8
5	Python-only vs dbt-based Comparison	9
6	Reflections	9
6.1	Most fragile element	9
6.2	Architectural insight	9
6.3	Design change I would make	9
7	Conclusion	9

1 Introduction

This document describes the work performed in Lab 1 and Lab 2 of the Data Engineering course. The objectives for Lab 2 were:

- Install and set up the environment (DuckDB, dbt-core, dbt-duckdb).
- Explore data modeling for analytics and construct a star schema.
- Structure a data pipeline using dbt models, separating raw data, transformation logic, and serving tables.
- Apply data quality tests using dbt and pytest.
- Prepare a stable serving layer for dashboards via DuckDB.

These goals build on Lab 1's Python pipeline, which ingested Google Play Store app and review data, performed cleaning, and produced analytics-ready outputs. Lab 2 extends that work by re-engineering the transformation stage using dbt and DuckDB.

2 Lab 1: Python Data Pipeline

2.1 Architecture

The first lab 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 app metadata and reviews.
- Aggregation of review metrics and joining with app dimension.
- Writing out processed JSON files for downstream analysis.

Figure 1 depicts the high-level architecture.

```

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 SCDF 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

During Lab 1, an exploratory script identified several issues:

1. Missing or null fields (developer, updated, etc.)
2. Inconsistent data types and formats (installs as strings, timestamps).
3. Reviews stored as a large JSONL file causing memory pressure.
4. Duplicate and orphaned reviews not detected.
5. Lack of validation resulting in silent failures.

These findings motivated enhancements in Lab 2.

2.3 Pipeline Fragility

The most fragile component was the lack of deduplication on review data. Re-running extraction appended the same reviews repeatedly, corrupting metrics. Additionally, hard-coded field names across modules made schema changes risky.

3 Enhancements Added in Lab 2

3.1 Incremental Loading

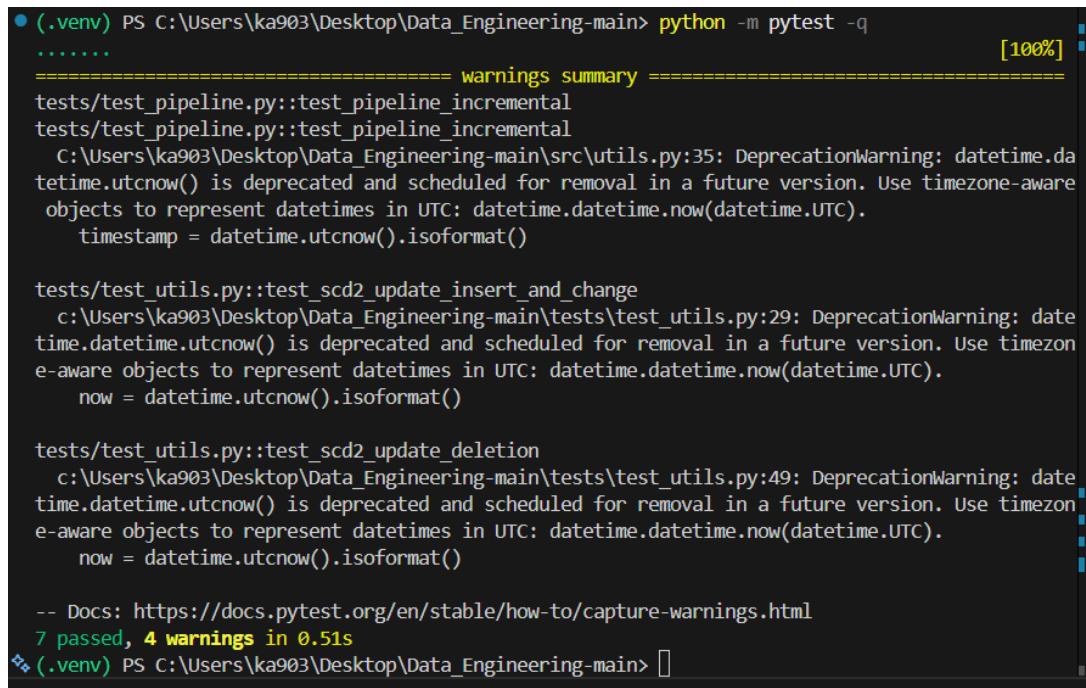
A utility function `merge_reviews` was written to merge new review records with existing processed data, keyed by `review_id`. This allowed the pipeline to run repeatedly without duplicating rows.

3.2 Slowly Changing Dimension (SCD2)

App metadata changes are now tracked using SCD Type 2 logic. The history table `apps_metadata_scd2.json` contains `start_date`, `end_date`, and `current_flag`. Python code performing the upsert is located in `src/utils.py`.

3.3 Data Quality and Testing

A new module `quality.py` implements basic checks that report missing IDs and type mismatches. The pipeline prints a summary before aggregation. PyTest tests were added for utilities, quality rules, and full pipeline runs. Screenshots of pytest output are shown in Figure 2.



```
● (.venv) PS C:\Users\ka903\Desktop\Data_Engineering-main> python -m pytest -q
.....
=====
===== 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.datetime.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: Automated tests passing

3.4 Star Schema Export

The pipeline can now emit a five-table star schema (`dimensions dim_apps, dim_categories, dim_developers, dim_date; fact_reviews`) as JSON.

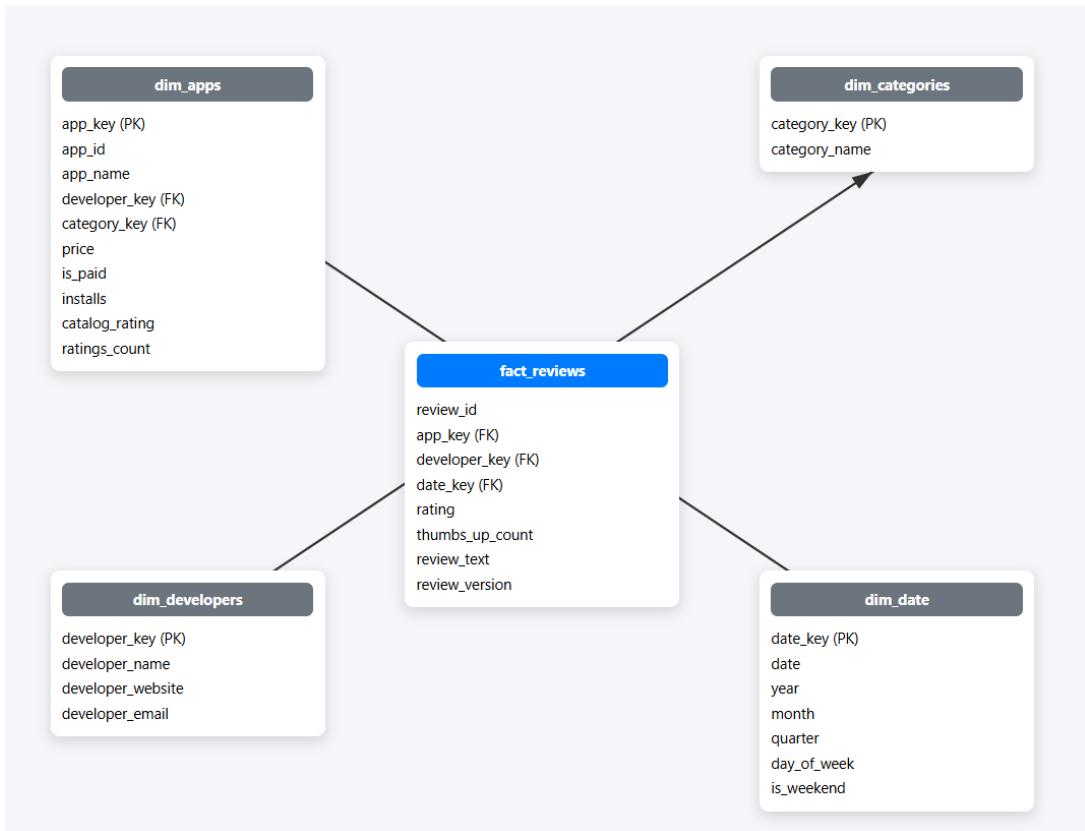


Figure 3: Final star schema generated by pipeline

4 dbt & DuckDB-based Pipeline (Lab 2)

4.1 Environment Setup

DuckDB and the dbt-duckdb adapter were installed in the same virtual environment used for Lab 1.

```
python -m pip install duckdb dbt-core dbt-duckdb
dbt --version
```

The last command produces output similar to Figure 4, which confirms that the installation was successful.

```
(.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 4: dbt test output demonstrating data quality checks

4.2 Project Structure

The dbt project resides in `dbt_project/`. Key files:

- `dbt_project.yml` – project configuration.
- `profiles.yml` – DuckDB connection pointing to `dbt.duckdb`.
- `models/staging/stg_apps.sql` and `stg_reviews.sql` – staging models reading raw JSON using DuckDB's `read_json_auto`.
- `models/marts/` – dimension and fact models implementing the star schema in SQL.
- `models/staging/schema.yml` – contains data quality tests.

```
version: 2
models:
  - name: stg_apps
```

```
columns:  
  - name: app_id  
    tests:  
      - not_null  
      - unique  
  - name: stg_reviews  
    columns:  
      - name: review_id  
        tests:  
          - not_null  
          - unique  
      - name: app_id  
        tests:  
          - not_null
```

4.3 Execution

```
(.venv) PS C:\Users\ka903\Desktop\Data_Engineering-main\dbt_project> cd dbt_project; set DBT_PROJECTFILES_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]
15:11:54  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 5: dbt run output with all models built

4.4 Data Quality Tests

After adding the tests, the command `dbt test` produced only PASS results, showing that identifiers are present and unique.

4.5 Serving Layer

The resulting DuckDB file (`dbt.duckdb`) contains the dimension and fact relations, suitable for consumption by BI tools.

5 Python-only vs dbt-based Comparison

Feature	Python pipeline	dbt pipeline
Transformation language	Python functions	SQL models
Incremental/SCD2 support	Manual (utils)	dbt snapshot/incremental
Testing	PyTest	dbt tests
Installation	pip	pip + dbt plugins
Execution	script run	<code>dbt run</code>
Warehouse requirement	None	DuckDB file

Table 1: Comparison of implementation approaches

6 Reflections

6.1 Most fragile element

Prior to enhancements, the review ingestion logic was extremely brittle; a repeated run produced thousands of duplicates. The tight coupling between transformation and analytics logic (everything in `transform_for_analytics`) was also problematic.

6.2 Architectural insight

Centralising configuration in a `config` module and deferring access to it at runtime allowed the code to be easily patched during tests.

6.3 Design change I would make

I would abstract schema definitions and field names so that changing a column requires editing a single location instead of multiple modules.

7 Conclusion

The labs provided hands-on experience building a complete data pipeline from scratch using Python and then refactoring the transformation layer to dbt running on DuckDB. Both implementations coexist in this repository and can be executed independently.