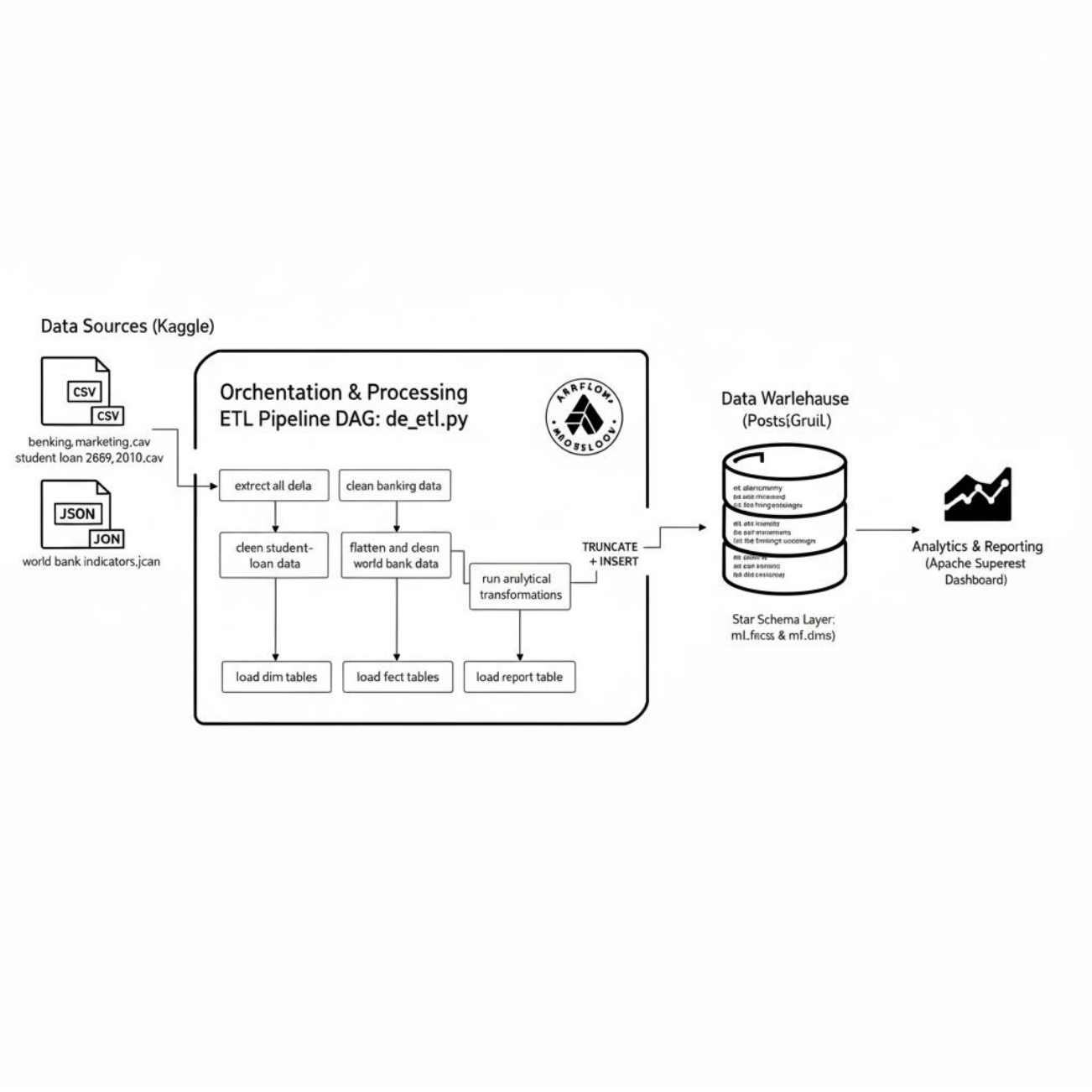
ETL Pipeline Analysis and Implementation



Contents

ETL Pipeline Analysis and Implementation Summary

This document summarizes the implementation and design decisions for the five key challenge areas, derived from the Airflow DAG code provided (de_etl.py).

1. Data Extraction (E-Stage)

The extraction layer successfully integrates three diverse data sources using Python/Pandas operators within the Airflow environment.

Source File	**Format**	**Loading Method**
banking_marketing.csv (data source = Kaggle)	CSV	pd.read_csv
student_loan_2009_2010.csv (data source = Kaggle)	CSV	pd.read_csv
world_bank_indicators.json (data source = Kaggle)	Nested JSON	Standard json.load (Python library)

Integrity and Validation

The code focuses on integrating integrity checks during the subsequent cleaning step:

- **Range & Domain Checks (Implicit):** Categorical fields (job, education, etc.) are checked for the invalid **'unknown'** value and normalized.
- **Cross-Field Validation:** The **contact_date** field is constructed by validating the compatibility of the day and month fields from the raw data using pd.to_datetime.

2. Data Cleaning & Transformation (T-Stage)

The transformation stage is modularized into three parallel tasks (**clean_banking_data, clean_student_loan_data, flatten_and_clean_world_bank_data**) and one aggregation task (**run_analytical_transformations**).

Standardization and Missing Value Strategy

Requirement	Implementation in Code	Strategy
Missing Values	Unknown categoricals $\rightarrow$ **'unknown_group'**. Missing dates $\rightarrow$ **'2024-01-01'**.	**Imputation with Sentinel Value:** Ensures all rows are usable while preserving the 'unknown' state for analysis.
Column Names	df.columns.str.lower().str.replace('.', '_')	**Snake Case Standardization:** Ensures compatibility with the PostgreSQL Data Warehouse (DW) environment.
Data Types	df['subscribed'] mapped to **True/False**.	**Boolean Conversion:** Aligns the target variable to an optimized DW type.
Date Formats	**df['contact_date'].dt.strftime('%Y-%m-%d')**	**ISO 8601 String Formatting:** This critical step resolves the common **psycopg2.errors.DatatypeMismatch** by sending a parseable date string instead of a large integer timestamp.

Analytical Transformations Implemented

1. **JSON Flattening & Relational Model:** The complex, nested World Bank JSON structure is successfully flattened into three relational tables: **dim_country, dim_indicator**, and **fact_indicator_value**.
2. **Marketing Funnel Metrics:** The run_analytical_transformations task groups the banking data by job and marital status to calculate the **conversion_rate** metric (mean of subscribed), producing the **report_marketing_metrics** table.

3. Data Loading (Warehouse Layer)

The data loading is managed by the **`WarehouseLoader`** class, which leverages the `PostgresHook` and is configured to run atomic, full-batch loads.

Warehouse Schema Design (Star Schema)

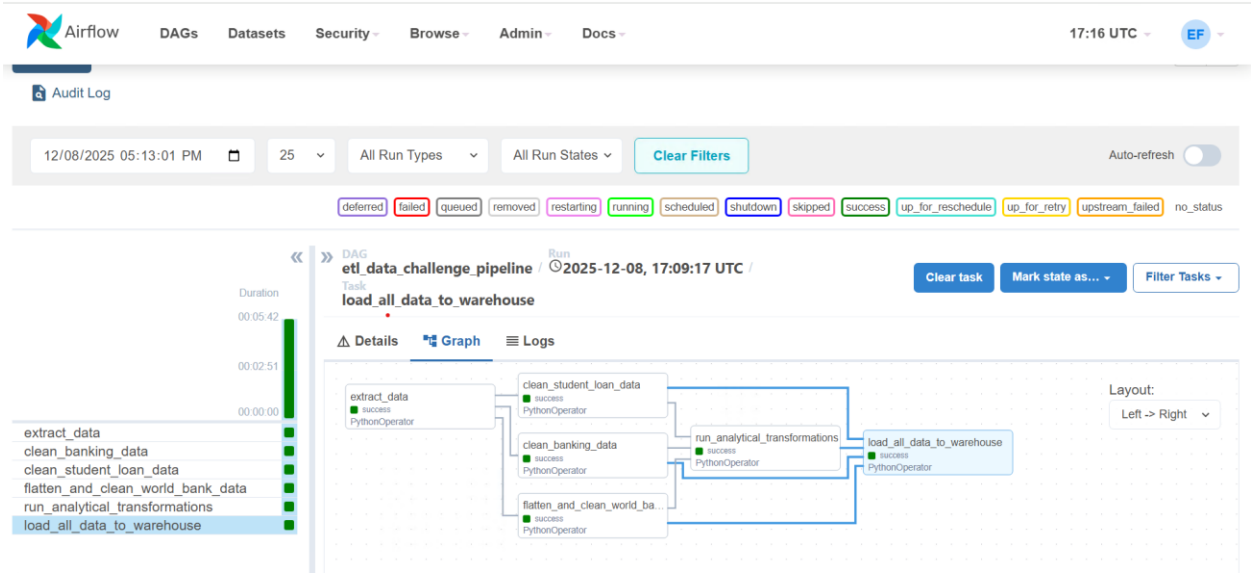
The design follows a **Star Schema** with a dedicated schema, `ml`:

Schema Object	Purpose	Primary Fields/Keys
Dimensions	ml.dim_country, ml.dim_indicator, ml.dim_school, ml.dim_banking_client	Natural Keys (country_code, indicator_code, ope_id) are used as joining keys.
Fact Tables	ml.fact_banking_campaign	Granular campaign records.
Fact Tables	ml.fact_indicator_value	Time-series data points for global indicators.
Fact Tables	ml.fact_student_loan	Aggregated quarterly loan metrics.

Loading Implementation Details

- **Technology: Python** (WarehouseLoader class) utilizing the **Airflow PostgresHook** to communicate with the PostgreSQL DW.
- **Methodology:** The _execute_insert_rows method uses the native **hook.insert_rows()** function for robust batch loading.
- **Commit Strategy:** Data is committed in batches of **1,000 rows** (commit_every=1000) to optimize transaction size and manage memory.
- **Write Disposition:** All data is loaded using a **TRUNCATE TABLE ... RESTART IDENTITY CASCADE** followed by INSERT, ensuring the DW state is fully replaced with the latest batch data.

3. Pipeline Automation & Scaling



Orchestration and DAG Design

- **Orchestrator: Apache Airflow** provides the orchestration layer.
- **DAG Structure:** The etl_data_challenge_pipeline follows a clear sequence: **Extraction >> Parallel Cleaning** >> **Analytical Aggregation** >> **Final Loading**.
- **Dependencies:** Explicit task dependencies (e.g., extract_task >> clean_banking_task) guarantee data freshness and prevent downstream tasks from running on incomplete data.

Reliability and Scaling Design

Concept	Implementation in Current DAG
Idempotency	Achieved via the TRUNCATE + INSERT logic in the WarehouseLoader. Rerunning any load task results in the exact same final state.
Incremental Updates	Currently uses full batch loads. To scale, the WarehouseLoader should be modified to implement an UPSERT logic (using PostgreSQL's ON CONFLICT DO UPDATE) to handle delta files and Change Data Capture (CDC).
Backfills	Handled natively by the Airflow scheduler, allowing the pipeline to be run retroactively for historical dates.
Schema Evolution	Requires an explicit strategy outside the ETL, such as using a tool like Alembic or Flyway to safely manage DDL changes before the loading task begins.
Monitoring	Airflow captures standard Python logging output (logger.info, logger.error), providing structured logs for tracing data flow and classifying errors.

5. Analytical Output Summary

The pipeline's primary analytical output is the **report_marketing_metrics** table, which facilitates data-driven decision-making.

	Visualization Recommendation (Superset)	Business Value
report_marketing_metrics	**Bar Chart And PIE (Conversion Rate)**	Answers: "Which Job/Marital segment converts best?" Directs marketing budget to the most efficient demographics.

Sperset Dashboard