Project Title: Loan ETL Data Pipeline

Presented by: Kritika Joshi

Presented to: Suresh Gautam

December 15, 2025

Abstract

This project involves the design and implementation of an automated ETL (Extract, Transform, Load) data pipeline for processing loan data using modern data engineering tools. The goal of the project is to build a reliable, scalable, and easy-to-maintain pipeline that can handle incremental CSV-based loan datasets while ensuring data quality and efficient data delivery. Apache Airflow is used as the orchestration tool to schedule, monitor, and manage the workflow, while PySpark is used to perform data cleaning, transformation, and aggregation operations. Raw loan data files are ingested and stored in MinIO, which acts as an S3-compatible data lake organized into Bronze (raw), Silver (cleaned), and Gold (curated) layers. The transformed data is stored in Parquet format for better performance and storage efficiency. Data quality checks are applied during the transformation phase to validate data accuracy. The final curated data is loaded into a PostgreSQL analytics database for reporting and analysis. Automated email notifications are sent after successful pipeline execution. The entire system is deployed using Docker, demonstrating a production-oriented ETL workflow.

Keywords

ETL Pipeline, Apache Airflow, PySpark, MinIO, PostgreSQL, Data Engineering, Data Quality, Docker, Parquet, Workflow Architecture

Table of Content

1. Introduction

1.1 Purpose

The purpose of this report is to document the design, implementation, and evaluation of an automated **Loan ETL (Extract, Transform, Load) Data Pipeline** developed as part of an internship project. This report aims to provide a clear understanding of how modern data engineering tools and best practices are applied to process loan data efficiently, reliably, and at scale. It serves as a technical and functional reference for supervisors, stakeholders, and future developers by detailing the system architecture, workflow design, data processing logic, and operational considerations of the ETL pipeline.

1.2 Overview

This report presents a comprehensive overview of an end-to-end ETL solution built to process incremental CSV-based loan datasets using a layered data lake architecture. The pipeline leverages **Apache Airflow** for orchestration, **PySpark** for distributed data transformation, **MinIO** as an S3-compatible object storage system, and **PostgreSQL** as the analytics database. The entire system is containerized using **Docker** to ensure consistency and portability across environments.

The report covers the project background, problem statement, scope, system architecture, component-level explanations, data flow, DAG design, transformation logic, and data quality mechanisms. Additionally, it discusses testing and validation approaches, challenges faced during development, and key considerations related to security,

scalability, and performance. The document concludes with recommendations for future enhancements, making it both descriptive and forward-looking.

1.3 Project Background

Loan data is typically generated from multiple operational systems and shared in the form of flat files such as CSVs, making manual data handling a common practice in many organizations. This manual approach often results in inconsistent data formats, frequent data quality issues such as missing values, duplicate records, and invalid entries, and a lack of proper auditability and data lineage. Additionally, manual processing increases the risk of errors and causes significant delays in reporting and analytical workflows. These challenges highlight the need for an automated, structured, and scalable data processing solution to ensure reliable, timely, and analytics-ready loan data.

2. WorkflowArchitecture

The ETL workflow architecture of this project is designed to provide an automated, scalable, and modular data processing pipeline using Apache Airflow, PySpark, MinIO, and PostgreSQL, all deployed in a containerized environment using Docker. The workflow begins with the availability of raw loan data files, primarily in CSV format, which act as the source system for the pipeline. These files represent incremental data loads and are treated as immutable raw inputs to ensure traceability and auditability throughout the pipeline lifecycle.

Apache Airflow acts as the central orchestration layer of the architecture. Within the Airflow environment, the Airflow Webserver provides a user interface for monitoring DAG executions, while the Scheduler is responsible for triggering tasks based on defined schedules or manual triggers. All execution metadata, task states, logs, and XCom messages are persisted in the Airflow Metadata Database (PostgreSQL). When a DAG is triggered, Airflow coordinates the execution of each stage of the pipeline in a controlled and fault-tolerant manner, ensuring retries, dependency management, and failure handling.

The ingestion phase of the workflow is initiated by Airflow through Python-based ingestion tasks. These tasks are responsible for reading the incoming CSV files and storing them in MinIO, which serves as an S3-compatible object storage layer. MinIO acts as the project's data lake and is logically organized into layered zones following Raw (Bronze), Staging/Silver, and Processed/Gold zones. Raw data is first stored in the Bronze layer in its original structure without modification, preserving data fidelity and allowing for reprocessing if needed.

Once the raw data is available in MinIO, Airflow triggers the PySpark ETL processing tasks. These Spark jobs read the raw data directly from MinIO and perform transformation operations such as null handling, data type conversion, deduplication, column standardization, and aggregation. The transformation logic is implemented using PySpark to efficiently handle larger datasets and support scalable processing. After applying cleaning and validation logic, the transformed datasets are written back to

MinIO in Parquet format, which improves performance, compression, and query efficiency. Cleaned datasets are stored in the Silver layer, while business-ready aggregated datasets are written to the Gold layer.

During the transformation phase, data quality checks are applied to ensure that the processed data meets predefined expectations, such as non-null constraints, valid ranges, and record count validation. Any inconsistencies or failures are logged through Airflow's logging mechanism, enabling easier troubleshooting and observability. After successful transformation, the curated datasets from the Gold layer are loaded into PostgreSQL analytics tables, which act as the serving layer for downstream consumption. These tables are optimized for reporting and analytical queries and can be accessed by business users or reporting tools. The PostgreSQL database also stores pipeline metadata, making it possible to track execution history and validate data completeness.

Finally, the workflow concludes with notification and reporting tasks managed by Airflow. Upon successful completion of the DAG, an automated summary email is sent via an SMTP service to business stakeholders, containing execution status and, where applicable, aggregated insights. This completes the end-to-end workflow from data ingestion to business consumption. Overall, the architecture ensures automation, reliability, scalability, and clear separation of concerns, making the ETL pipeline maintainable and suitable for real-world production use.

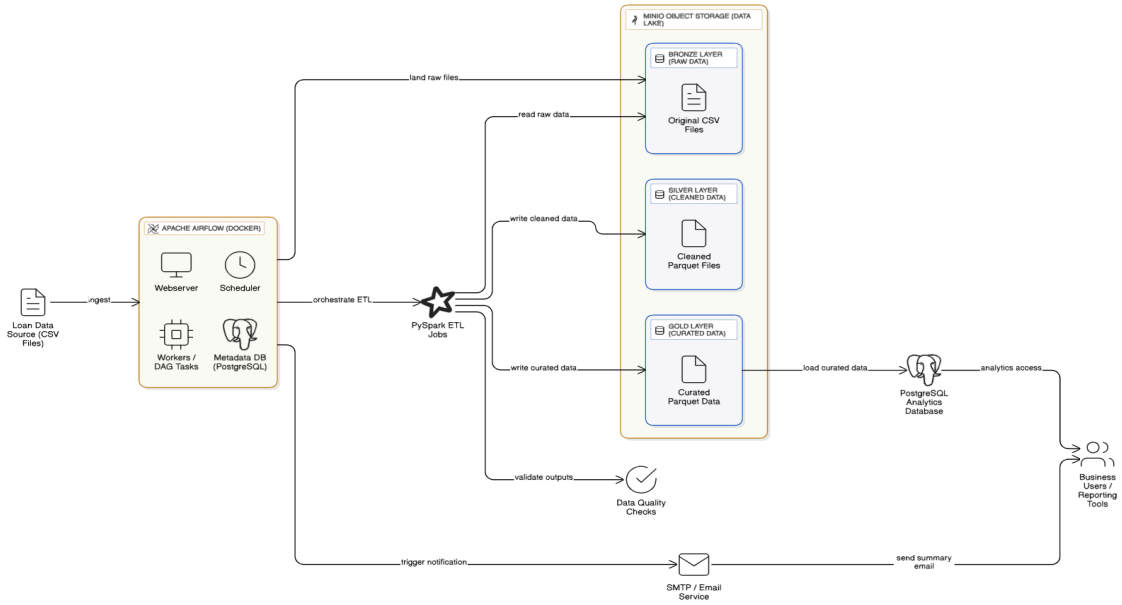

Fig 2.1: ETL Workflow Architecture

3. Components Used

a. Apache Airflow

Airflow is used as the orchestration engine. It schedules and monitors ETL workflows using Directed Acyclic Graphs (DAGs). It provides a web-based user interface for monitoring DAG runs, task statuses, and execution logs. Airflow also supports scheduling pipelines at regular intervals, ensuring automated daily processing of loan data. DAG represents a logical pipeline consisting of ingestion, validation, transformation, and load tasks. This orchestration layer improves reliability, observability, and operational control of the ETL workflow.

b. MinIO (S3-Compatible Storage)

MinIO serves as the object storage and data lake layer for the ETL pipeline. It is S3-compatible, making it suitable for scalable and cost-effective data storage. It is logically divided into three zones:

- Raw Zone: Original, unmodified CSV files
- Staging Zone: Validated and cleaned intermediate data
- Processed Zone: Final curated datasets

c. PySpark

PySpark is used as the primary data processing and transformation engine in the pipeline. It enables distributed processing of large datasets, improving performance and scalability. PySpark is especially used to handle data cleansing task , such as:

- Schema enforcement
- Data cleansing
- Deduplication
- Aggregations

d. PostgreSQL

PostgreSQL is used as the relational database for storing both Airflow metadata and final analytics tables. It maintains critical information related to DAG execution, task states, logs, and scheduling history. In addition, PostgreSQL acts

as the serving layer for the ETL pipeline by storing curated loan datasets optimized for reporting and analysis.

e. Docker

Docker is used to containerize all components of the ETL pipeline, including Airflow, PySpark, MinIO, and PostgreSQL. Containerization ensures consistent environments across development, testing, and deployment stages. It eliminates dependency conflicts and simplifies setup by packaging applications with their required libraries.

4. DAG Design & Workflow Scheduling

The ETL workflow is orchestrated using **two well-defined Apache Airflow DAGs**, each responsible for a specific stage of the data pipeline. This separation of concerns improves modularity, maintainability, and operational clarity.

The first DAG, **drive_watch_dag**, is designed to handle the ingestion phase. It begins with a GoogleDriveSensor (**wait_for_gdrive_file**) that continuously monitors the source location for the arrival of new loan data files. Once a file is detected, the workflow proceeds to the **download_and_compress** task, which downloads the file and prepares it for storage. The processed file is then uploaded to MinIO using the **upload_to_minio** task. Finally, a **send_email_notify** task sends an automated email notification confirming the successful ingestion of data. This DAG ensures that only newly available files are ingested and that the raw data layer remains immutable and traceable.

The second DAG, **spark_etl_dag**, is responsible for the transformation and processing stage of the pipeline. It consists of a **run_pyspark_etl** task implemented using a PythonOperator that triggers a PySpark job. This Spark job reads raw data from MinIO, applies data validation, cleaning, deduplication, and aggregation logic, and writes the transformed outputs back to MinIO in Parquet format. The DAG is designed to execute independently once raw data is available, enabling flexible scheduling and reprocessing if required.

Both DAGs are configured to support manual and scheduled execution, with built-in retry mechanisms, task-level logging, and execution status tracking through the Airflow UI. Dependencies within each DAG are strictly linear to ensure controlled progression from ingestion to completion. Together, these DAGs provide a reliable, observable, and production-oriented workflow that automates the end-to-end loan data processing lifecycle while allowing clear separation between ingestion and transformation responsibilities.

5. Transformations Applied

The transformation phase is a critical component of the ETL pipeline, responsible for converting raw loan data into clean, consistent, and analytics-ready datasets. All transformation logic is implemented using **PySpark,** enabling scalable and efficient processing. The transformations can be broadly categorized into validation rules, data cleaning logic, and transformations.

a. Validation Rules

Validation rules are applied to ensure the structural and logical correctness of the incoming loan data. Schema validation checks confirm that all expected columns are present with correct data types. Mandatory field checks are enforced on critical attributes such as loan ID, loan amount, and loan status. Data type enforcement converts fields into appropriate numeric, date, or string formats.

b. Data Cleaning Logic

Data cleaning focuses on improving accuracy and consistency of the dataset. Null values are handled by applying mean for numerical fields and mode for categorical fields. Duplicate records are identified and removed using primary key–based deduplication logic. Column names and formats are standardized to ensure consistency across datasets.

c. Transformations

Transformations convert cleaned data into analytics-ready datasets. Loan amount aggregations are performed to calculate totals, averages, and counts for reporting purposes. Status-based filtering is applied to categorize loans based on their current state. Date and time values are separated from timestamp fields to support time-based analysis. These transformations prepare the data for downstream reporting and decision-making.

6. Testing & QA Summary

Testing and Quality Assurance (QA) were conducted to ensure that the Loan ETL Data Pipeline functions correctly, reliably, and securely under expected operating conditions. A

structured test plan was followed to validate the functionality of Airflow DAGs, verify data accuracy at each processing stage, and ensure proper configuration and security practices. The testing activities focused on confirming that the pipeline meets both functional and non-functional requirements.

6.1 Test Plan

The test plan was designed to validate the end-to-end ETL workflow, from data ingestion to final data availability in the analytics database. Testing was carried out in a controlled Docker-based environment to closely simulate real-world execution. The following key testing categories were included:

a. **Functional Testing of DAGs**

   Functional testing was performed to verify that all Airflow DAGs execute successfully as per their defined workflows. This included validating task dependencies, execution order, retries, and failure handling. Each task within the DAGs such as file ingestion, Spark-based transformation, data loading, and email notification was tested individually. DAG executions were monitored through the Airflow UI to confirm successful completion and accurate status reporting.

b. **Data Validation Testing**

   Data validation tests were conducted to ensure data accuracy, completeness, and consistency throughout the pipeline. It checks for null handling, deduplication effectiveness, correct data type conversions, and accuracy of aggregated results loaded into PostgreSQL.

c. Security and Configuration Checks

Security and configuration testing focused on verifying that sensitive information such as database credentials and access keys were not hard-coded in DAGs or scripts. Environment variables and configuration files were reviewed to ensure secure credential handling. Docker configurations were validated to confirm correct service isolation and network communication. Logging and access controls were also reviewed to ensure that pipeline execution details could be monitored without exposing sensitive data.

6.2 Sample Test Cases and Results

The following table summarizes representative test cases executed during the QA phase. These test cases were selected from the complete test case document to demonstrate coverage across ingestion, transformation, security, and infrastructure validation.

Test Case ID	Description	Status
TC-001	Verifies that all DAGs load successfully in the Airflow UI without errors	Pass
TC-002	Confirms PySpark and required libraries are available in the runtime environment.	Pass
TC-003	Validates complete pipeline execution from ingestion to final data load.	Pass
TC-004	Ensures incoming data matches the expected schema and data types.	Pass
TC-005	Confirms re-running the pipeline does not create duplicate or inconsistent data.	Pass
TC-006	Verify task failures, trigger retries and proper failure handling.	Pass

TC-007	Ensures execution logs and task statuses are available for monitoring.	Pass
TC-008	Confirms sensitive credentials are securely managed and not hard-coded.	Fail
TC-009	Validates custom Airflow image includes PySpark and dependencies.	Pass
TC-010	Code quality & packaging	Partial Review Pass

7. Challenges Faced & Solutions Implemented

During the development of the Loan ETL Data Pipeline, several technical and operational challenges were encountered.

a. PySpark setup in Airflow

One of the major challenges was integrating PySpark with Apache Airflow within a containerized environment. The default Airflow Docker image does not include PySpark or its required dependencies, leading to import errors and runtime failures during Spark task execution.

Solution: This issue was resolved by building a **custom Airflow Docker image** that included PySpark, Java runtime, and required Python libraries. By extending the official Airflow image, the environment was standardized and Spark jobs were able to execute reliably within Airflow tasks.

b. DAG Dependency and Workflow Coordination Issues

Initially, the ingestion and transformation processes were implemented as separate DAGs, which caused coordination challenges. The transformation DAG could execute without

ensuring that ingestion had completed successfully, resulting in missed or incomplete processing.

Solution: Clear task dependencies were defined within each DAG, and execution order was enforced using Airflow's scheduling and triggering mechanisms. Logical separation was retained while ensuring proper orchestration, making the workflow more reliable and easier to manage.

c. Environment Configuration and Dependency Conflicts

Managing dependencies across multiple components such as Airflow, PySpark, MinIO, and PostgreSQL within a Dockerized environment posed configuration challenges. Version mismatches and missing libraries initially led to container startup failures and task execution errors.

Solution: The issue was resolved by standardizing dependencies through Docker and custom image builds, explicitly defining library versions in Dockerfiles and requirements files. This ensured consistent behavior across development and execution environments and improved overall system stability.

8. Future Improvements / Recommendations

While the current implementation meets project requirements, several enhancements can further improve robustness and enterprise readiness:

- Implement Parquet-Based Storage End-to-End: Extend Parquet usage across all layers, including staging and long-term storage, to maximize performance benefits.

- Add Data Lineage and Cataloging: Integrate tools for data lineage and metadata management to improve data discoverability, governance, and auditability.
- Introduce CI/CD for DAG Deployment: Implement continuous integration and deployment pipelines to automate testing, validation, and deployment of Airflow DAGs.
- Enable Real-Time Streaming Ingestion: Enhance the architecture to support streaming data sources (e.g., Kafka) for near real-time loan data processing.

9. Conclusion

The Loan ETL Data Pipeline successfully demonstrates a production-style data engineering solution built using modern tools and industry best practices. The project delivers a fully automated, scalable, and maintainable ETL workflow capable of processing incremental loan data with strong data quality controls. Through the use of Apache Airflow, PySpark, MinIO, PostgreSQL, and Docker, the pipeline ensures reliability, observability, and efficient data delivery. Overall, the solution aligns well with enterprise data architecture standards and serves as a strong foundation for real-world analytics and future enhancements.

Appendix structure

Appendix A: Data Ingestion Evidence

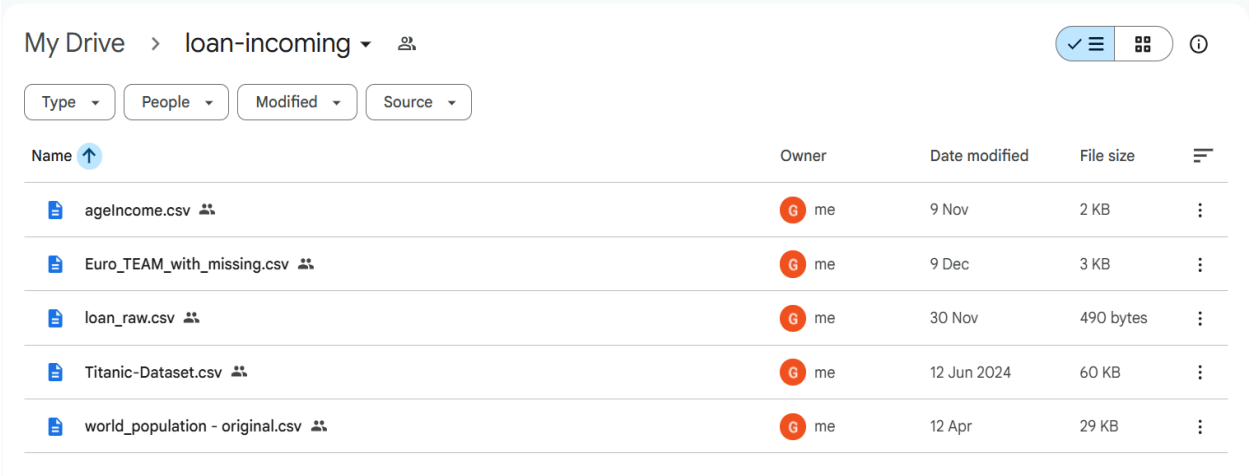

Fig. A.1: Google Drive Folder Containing Loan CSV Files

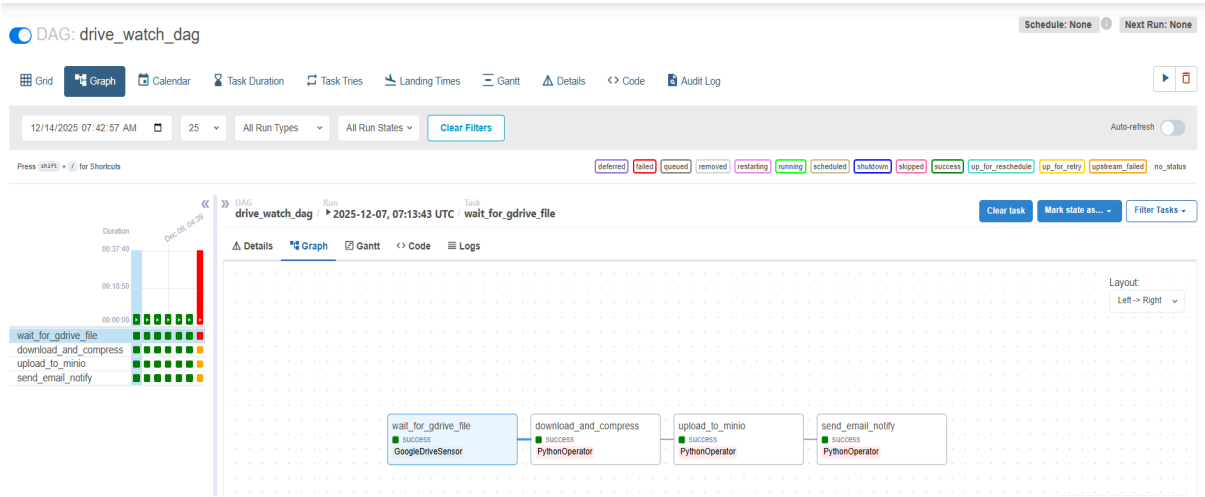

Fig. A.2: Airflow drive_watch_dag Execution Status

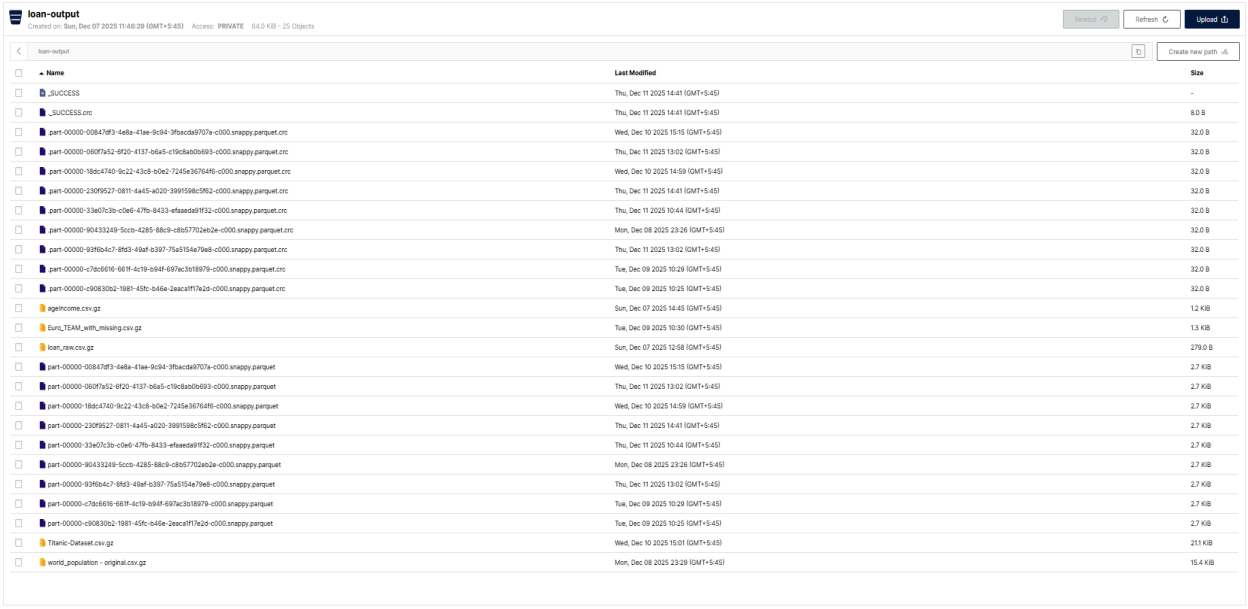

Fig. A.3: MinIO Bucket

Appendix B: Data Transformation Evidence

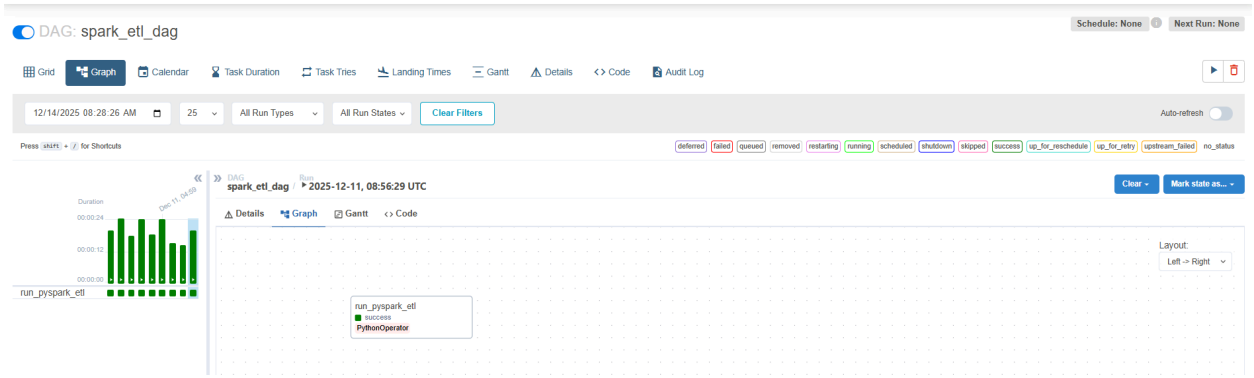

Fig. B.1: Airflow spark_etl_dag Graph View

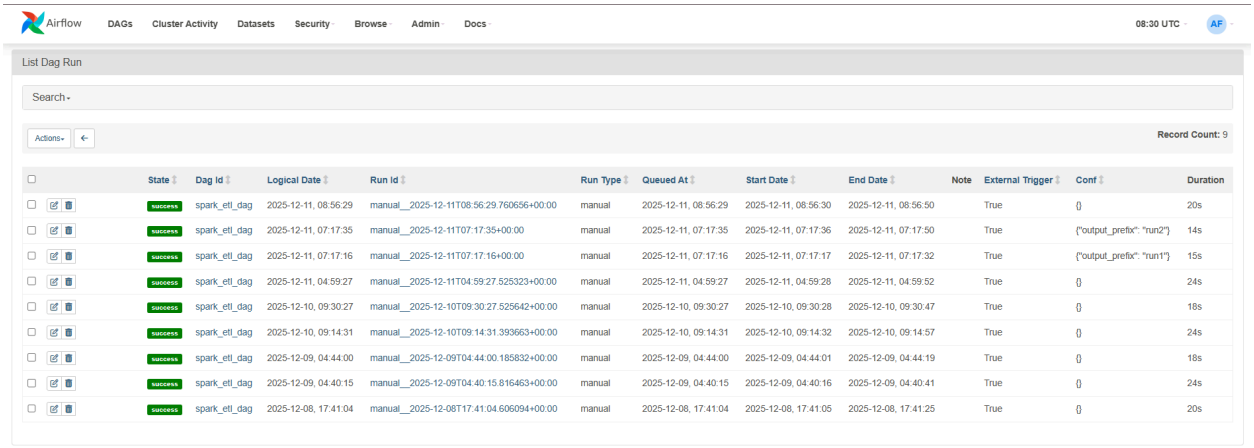

Fig. B.2: Spark ETL Task Logs Showing Successful Execution

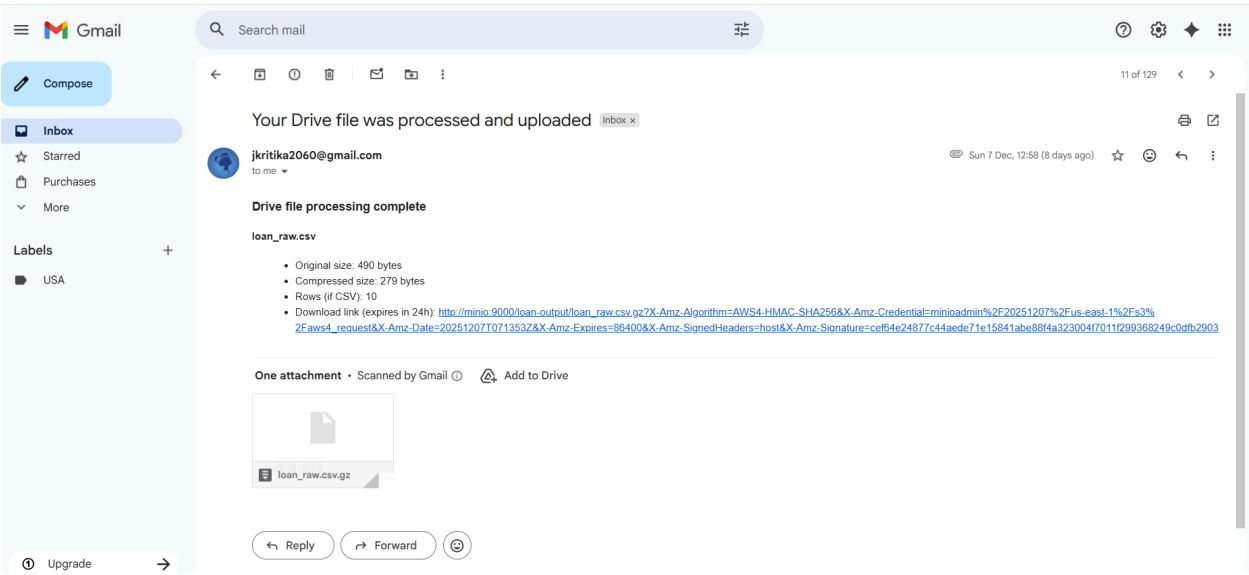

Fig. B.3: Proof of Automated Notification for Loan Data Processing