# ASSIGMENT REPORT

**🏗️ Architecture Summary**

This project implements a modular **lakehouse-style data pipeline** using **PySpark**, **Apache Airflow**, and **MinIO** to simulate an end-to-end emissions analytics workflow. The architecture is designed for flexibility, observability, and ease of migration to cloud platforms like AWS.

**1. Data Generation Layer**

A synthetic data generator (generate\_emissions\_data.py) creates 1,000+ rows of emissions data with realistic variability and intentional quality issues. This data mimics digital advertising metrics such as domain coverage, emissions types, country, device, and format, reflecting common patterns in ad tech analytics.

**2. Storage Layer**

Raw and processed data is stored in **MinIO**, which emulates Amazon S3. Data is written and read using the s3a://protocol from PySpark, allowing easy transition to AWS S3 if needed. All output artifacts (cleaned data, business logic outputs) are stored as Parquet files under distinct output prefixes.

**3. Orchestration Layer**

An **Apache Airflow DAG** (emissions\_pipeline\_dag.py) coordinates all steps in the pipeline:

* generate\_data: (optional) generates raw data
* ingest\_raw\_data: uploads CSV to MinIO
* ingest\_stage\_data: runs PySpark job to cleanse and validate data
* transform\_logic: executes PySpark aggregations and flags anomalies

Each task has built-in retry and error logging for robustness.

**4. Processing & Transformation Layer**

All core transformation logic is implemented in **PySpark**, making it scalable for large datasets. Business logic includes:

* Aggregations by country, domain, device, format
* Percentage breakdown of emissions types
* Anomaly detection on emission values
* Top-N domain analysis

The pipeline is modular, with independent scripts for each stage (clean\_data.py, transform\_logic.py), making it testable and maintainable.

**5. Model Layer (In-Progress)**

The project includes a placeholder for **dbt models**, which will support reusable, testable SQL logic built on the staged and curated layers. Once integrated, dbt will power semantic modeling, metrics standardization, and unit testing.

**6. Deployment & Environment**

All components run in **Docker containers** orchestrated via docker-compose. This provides a fully portable local development environment that closely resembles cloud-native infrastructure. Create dockerfile for airflow for better performance.

**📈 Data Flow Diagram (Paste into Word using Consolas or Courier New)**

┌────────────────────────────┐

│ Synthetic Data Generator │

│ (generate\_emissions\_data) │

└─────────────┬──────────────┘

│

▼

┌────────────────────────────┐

│ Raw CSV File (Local) │

└─────────────┬──────────────┘

│

▼

┌────────────────────────────────────────────┐

│ Ingest to MinIO Bucket (via Airflow) │

│ (s3a://raw\_data) │

└─────────────┬──────────────────────────────┘

│

▼

┌────────────────────────────┐

│ Clean Data with Spark │

│ (clean\_data.py task) │

└─────────────┬──────────────┘

│

▼

┌────────────────────────────┐

│ Staged Data (s3a://cleaned)│

└─────────────┬──────────────┘

│

▼

┌─────────────────────────────────────┐

│ Apply Business Logic using PySpark │

│ (process\_business\_logic.py) │

└─────────────┬───────────────────────┘

│

▼

┌────────────────────────────────────┐

│ Final Output (s3a://output/metrics)│

│ Format: Parquet │

└─────────────┬──────────────────────┘

│

▼

┌────────────────────────────┐

│ (Optional) dbt Models │

│ (on staged/output data) │

└────────────────────────────┘

A screenshot of a computer

AI-generated content may be incorrect.

MinIO

A screenshot of a computer

AI-generated content may be incorrect.

Airflow