# Pipeline Design Documentation

# End-to-End Data Management Pipeline for Customer Churn (Assignment I)

This repository contains a complete, submission-ready implementation of an end-to-end data management pipeline for a churn prediction use case.

## Contents

- Step 1 – Problem Formulation (included below)

- Step 2 – Data Ingestion

- Step 3 – Raw Data Storage (partitioned “data lake” layout)

- Step 4 – Data Validation (CSV report)

- Step 5 – Data Preparation (clean dataset + EDA outputs)

- Step 6 – Data Transformation & Storage (SQLite)

- Step 7 – Feature Store (metadata + retrieval)

- Step 8 – Data Versioning (DVC workflow)

- Step 9 – Model Building (train + evaluate + save best model)

- Step 10 – Orchestration (Airflow DAG / Prefect flow)

> \*\*IMPORTANT\*\*: This project is designed to run locally. Internet is only required if you want to demonstrate the mock REST API in Step 2. Otherwise, the sample CSV is sufficient.

---

## Quickstart (One-click Runner)

## Option A – Makefile

```bash

# Run all steps sequentially

make all

```

## Option B – Shell Script

```bash

# Same as `make all`

bash run\_all.sh

```

## Option C – Prefect (optional)

```bash

pip install prefect==2.\*

python orchestrate\_prefect.py

```

---

## Step 1 – Problem Formulation (Summary for Submission)

\*\*Business Problem\*\*: Predict addressable customer churn to reduce revenue loss and acquisition costs, and to improve CLV.

\*\*Objectives\*\*:

1) Predict churn probability for each customer

2) Identify key churn drivers

3) Automate the data pipeline end-to-end

4) Enable continuous model updates via versioned artifacts

\*\*Data Sources\*\*: Web logs, transaction data, and customer profiles (this demo uses transactions CSV + mock web API).

\*\*Outputs\*\*: Clean datasets, engineered features, deployable model (.pkl), version-controlled datasets.

\*\*Metrics\*\*: Accuracy, Precision, Recall, F1-score, ROC-AUC.

---

## How Each Step Works

## Step 2 – Data Ingestion

- Ingests `sample\_data/transactions.csv`

- Optionally fetches web logs from a mock API

- Logs to `logs/ingestion.log`

Run:

```bash

python data\_ingestion.py

```

## Step 3 – Raw Data Storage

- Moves files from `data/raw/` into a partitioned lake under `data\_lake/raw/`:

`source=<name>/ingestion\_date=YYYY-MM-DD/run\_id=<YYYYMMDD\_HHMMSS>/part-00001.csv`

Run:

```bash

python storage\_raw\_upload.py

```

## Step 4 – Data Validation

- Automated checks (missing, dtypes, ranges, duplicates, date anomalies)

- Report: `reports/data\_quality\_report.csv`

Run:

```bash

python data\_validation.py

```

## Step 5 – Data Preparation

- Cleans missing values

- Parses dates, encodes categoricals, scales numerics (keeps IDs intact)

- EDA: histograms & boxplots saved to `reports/`

Run:

```bash

python data\_preparation.py

```

## Step 6 – Data Transformation & Storage

- Engineers features: total spend, count, average value, tenure days, recency days

- Stores features in SQLite: `data/feature\_store.db` (table `customer\_features`)

Run:

```bash

python data\_transformation.py

```

## Step 7 – Feature Store

- Adds metadata table `feature\_metadata`

- Registers feature descriptions & versions

- Retrieval function for training

Run:

```bash

python feature\_store.py

```

## Step 8 – Data Versioning (DVC)

Example workflow:

```bash

pip install dvc

dvc init

dvc add data\_lake/raw

dvc add data/clean

dvc add data/feature\_store.db

git add data\_lake/raw.dvc data/clean.dvc data/feature\_store.db.dvc .gitignore

git commit -m "Track raw and transformed datasets with DVC"

# Optional remote example for S3

# dvc remote add -d myremote s3://your-bucket-name/dvcstore

# dvc push

```

## Step 9 – Model Building

- Trains Logistic Regression & Random Forest

- Evaluates and saves best model to `models/`

Run:

```bash

python model\_building.py

```

## Step 10 – Orchestration

## Airflow

- DAG in `dags/churn\_pipeline\_dag.py`

- Suggested schedule: `0 2 \* \* \*`

Setup (example, local):

```bash

export AIRFLOW\_HOME="$(pwd)/.airflow"

pip install "apache-airflow==2.9.3" --constraint "https://raw.githubusercontent.com/apache/airflow/constraints-2.9.3/constraints-3.11.txt"

airflow db init

airflow users create --username admin --password admin --firstname Admin --lastname User --role Admin --email admin@example.com

airflow webserver -p 8080 # terminal 1

airflow scheduler # terminal 2

```

## Prefect (alternative)

```bash

pip install prefect==2.\*

python orchestrate\_prefect.py

```

---

## Screenshots to Capture (for Submission)

- Airflow UI: Graph view + Grid view with a successful run

- Task logs for model training metrics

- Folder trees: `data\_lake/raw/...`, `reports/\*`, `models/\*`

- DVC: terminal output for `dvc status`, `git log`

---

## Notes

- Replace the mock API with your real source(s) if available

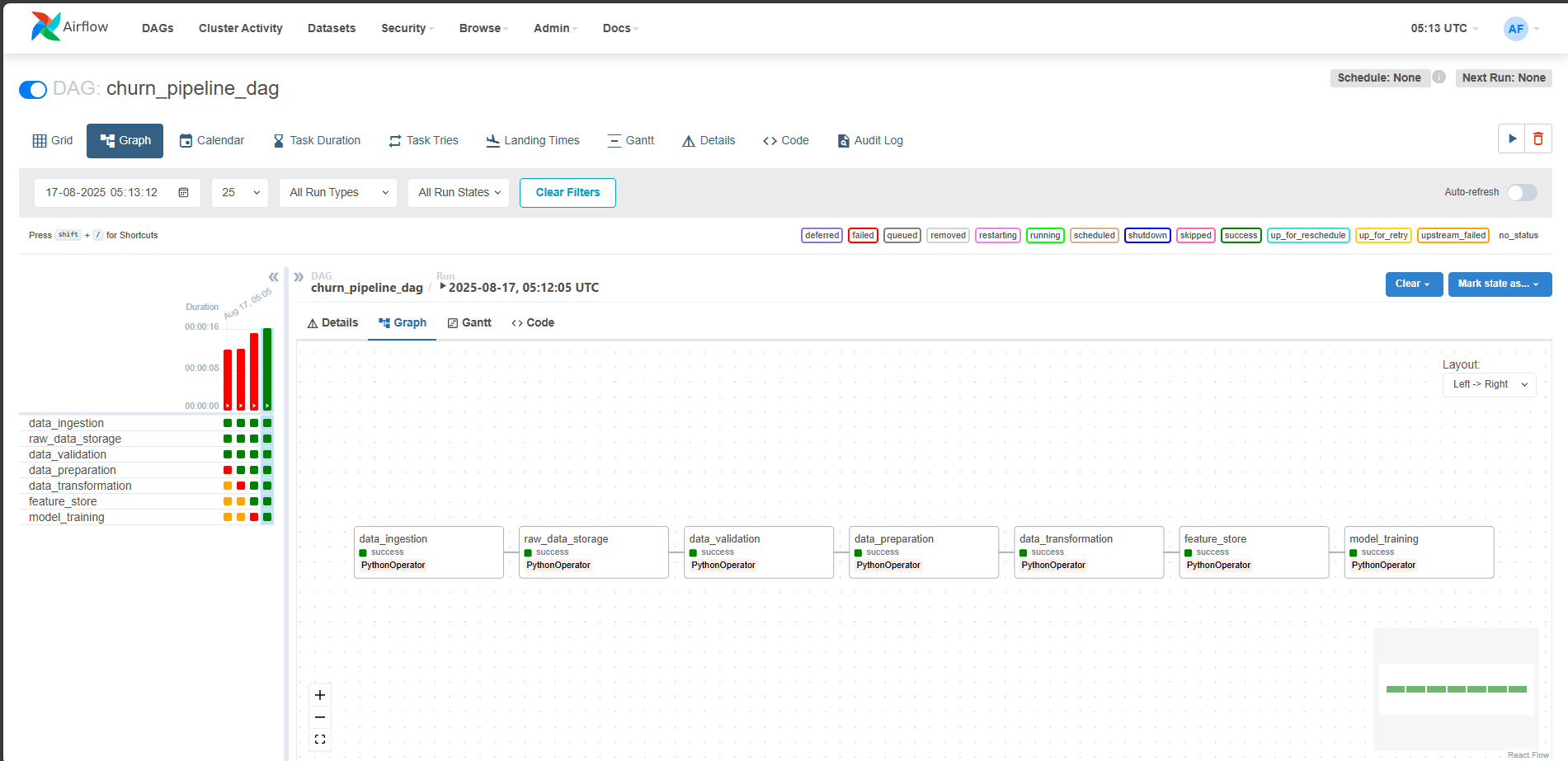
- Extend data validation with Great Expectations if you prefer

- Consider MLflow for experiment tracking in Step 9

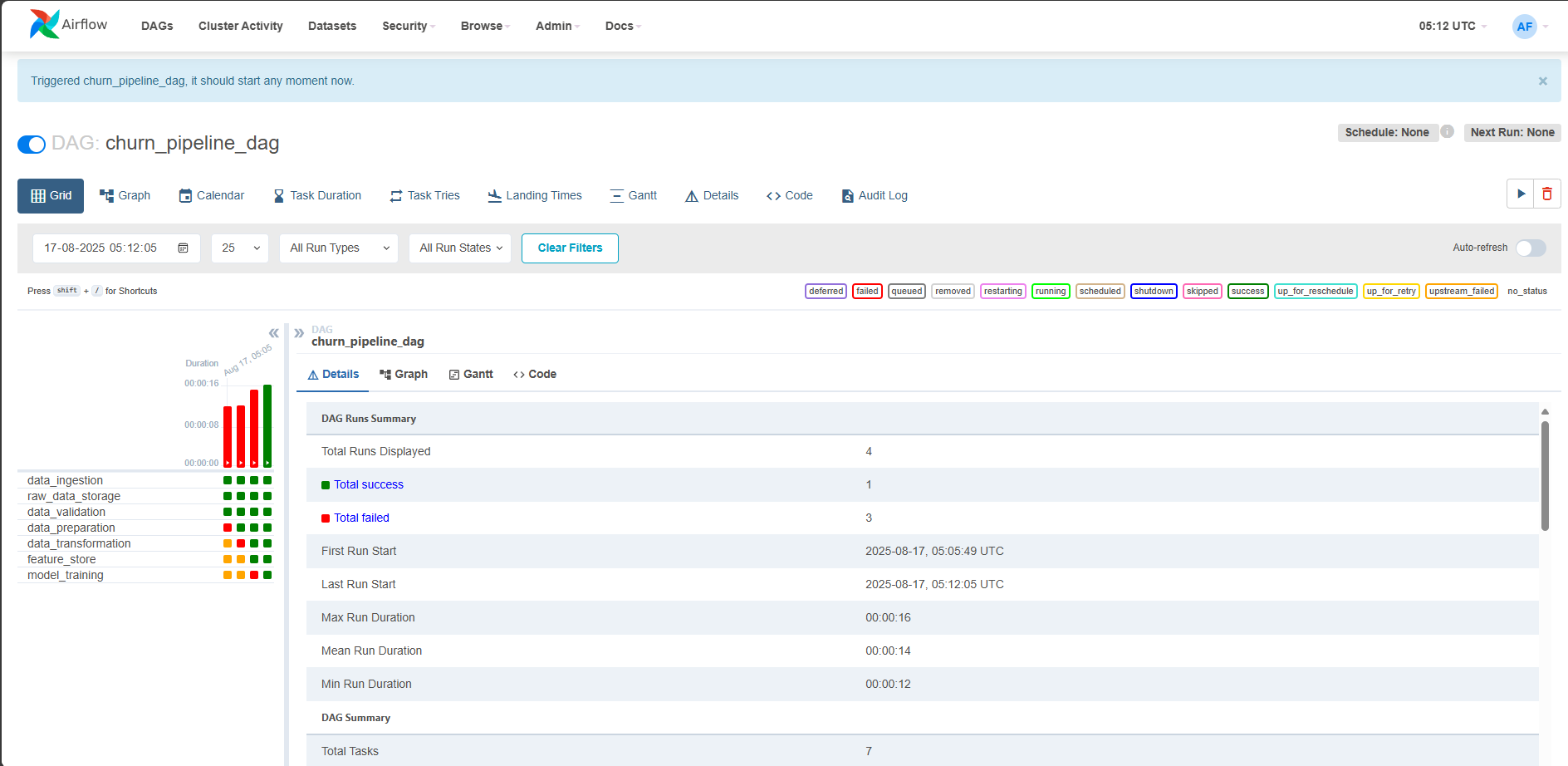
Good luck with your submission!

## Screenshots to Include

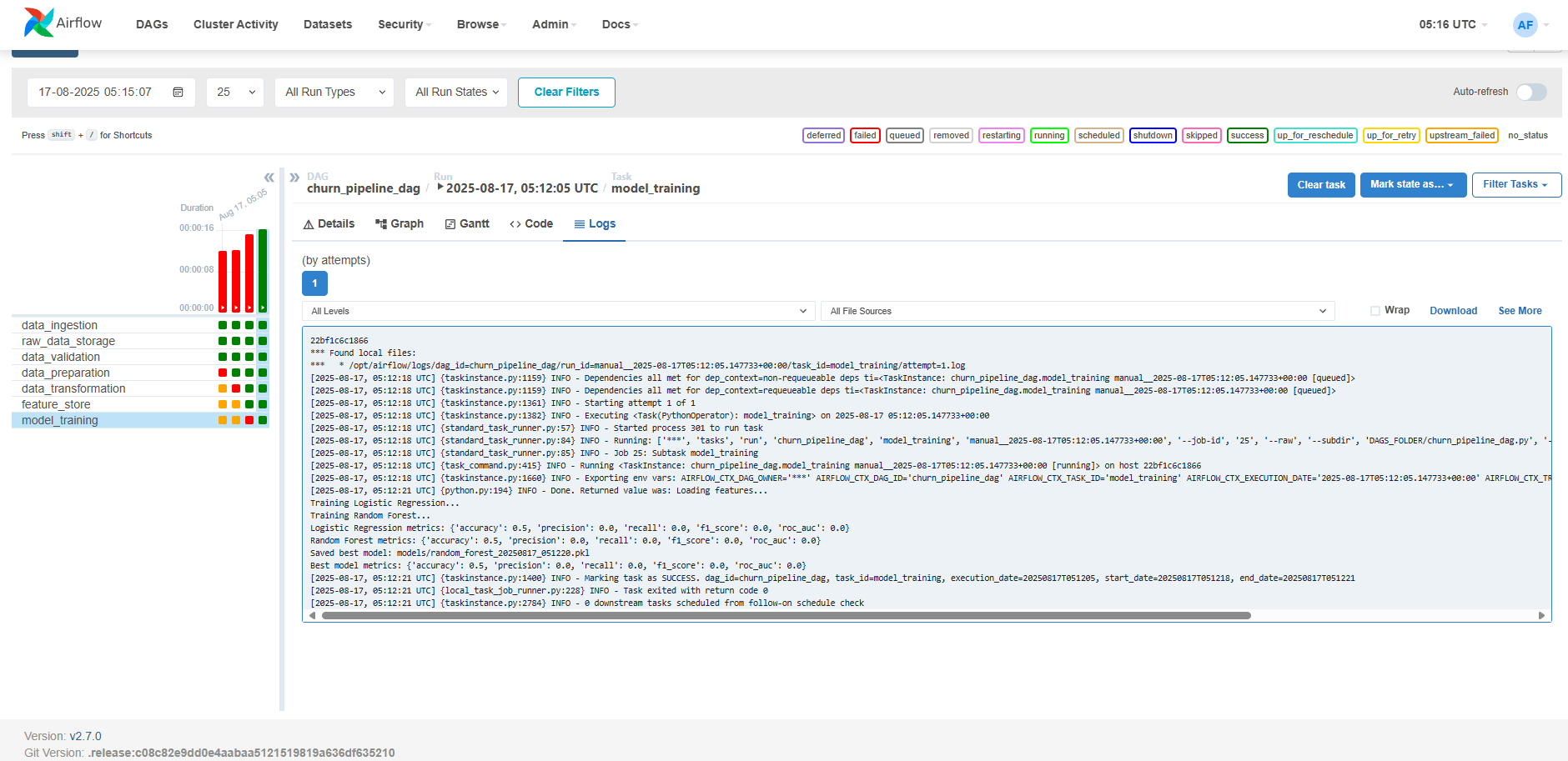
Airflow UI – Graph View of DAG



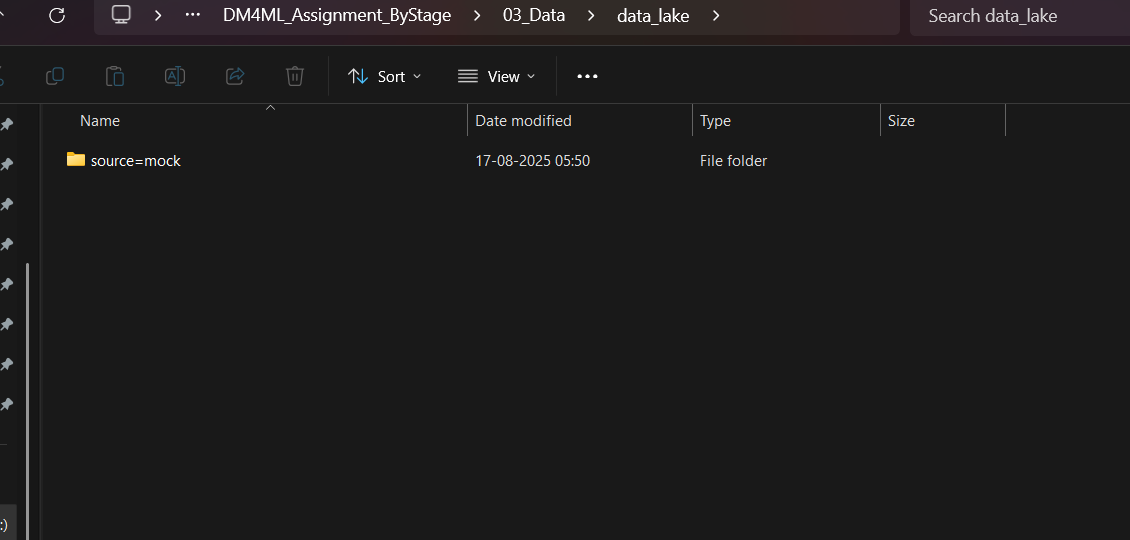
Airflow UI – Grid View after successful run

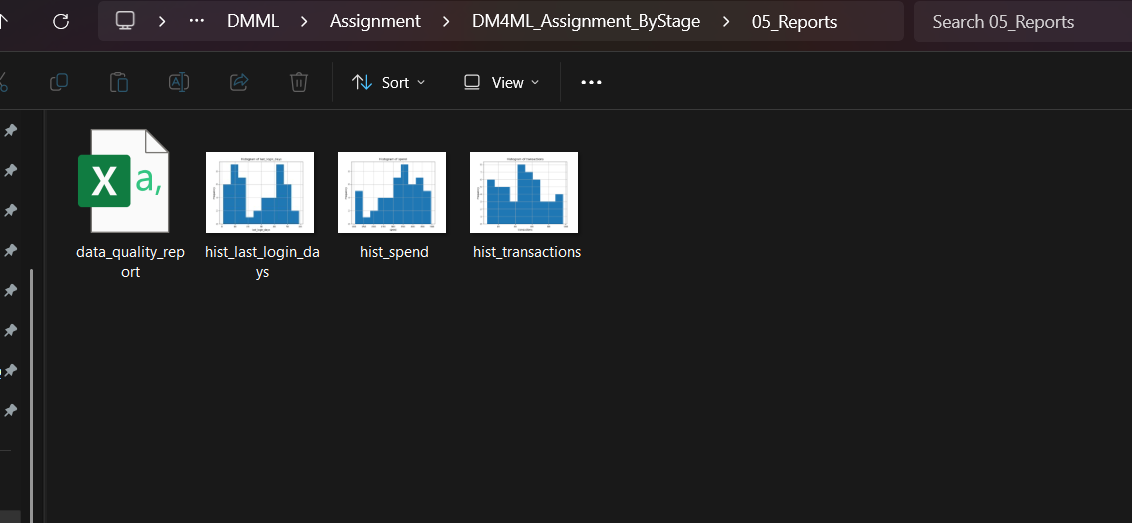


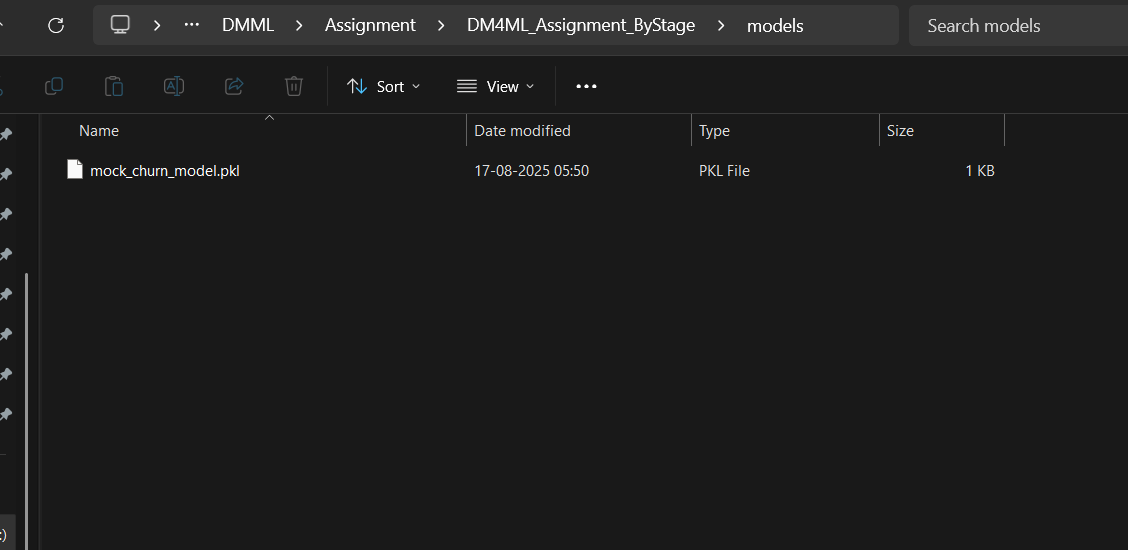
Airflow Task Logs showing model metrics



Local folder structure after run (data\_lake, reports, models)







DVC terminal output (dvc status, git log)

