Project Report: Automated Data Quality Pipeline with Apache Airflow

Introduction:

In modern data-driven environments, data quality is a critical factor for accurate analytics and decision-making. This project implements a Lakehouse architecture based on the Medallion model (Bronze → Silver → Gold), orchestrated using Apache Airflow.

The primary objective was to develop a fully automated, reproducible pipeline that ingests raw data, applies data quality checks, transforms it into analytics-ready formats, and generates aggregated summaries for insight generation.

Objectives

- Implement a modular Lakehouse pipeline using Airflow.
- Ensure data quality through automated cleaning and validation.
- Produce Gold layer outputs optimized for analytics and reporting.
- Maintain reproducibility and flexibility for future extensions.

Tools and Technologies

- Apache Airflow (2.7.2): Orchestration of workflow tasks.
- Python (pandas): Data cleaning, transformation, and validation.
- CSV datasets: Input and output data format.
- Docker: Containerized execution environment for Airflow.

Architecture Design

The project follows the Medallion Architecture:

Layer	Purpose
Bronze	Raw source data storage without modification.
Silver	Cleaned and validated datasets with missing, invalid, and inconsistent values handled.
Gold	Aggregated and summarized tables optimized for analysis, e.g., trip_summary.csv.

The workflow orchestrates these layers in a Directed Acyclic Graph (DAG) using Airflow, where each layer's tasks are sequentially dependent to guarantee data consistency.

Lakehouse Design

I designed the lakehouse architecture by following the Medallion Architecture pattern, which organizes data into three distinct layers: Bronze, Silver, and Gold.

Bronze Layer – Raw Data

- Stores the raw input exactly as received from the source (e.g., CSV or external dump)
- No transformations or cleaning

Silver Layer – Cleaned & Validated Data

- Applies data cleaning, type casting, and validation rules.
- Fixes missing values, invalid entries, incorrect formats, etc.

Gold Layer – Data of the highest quality

- Aggregated and summarized data for reporting, visualization, or business intelligence

Pipeline Implementation

DAG Overview

The pipeline is implemented as an Airflow DAG named **lakehouse_pipeline**, with the following PythonOperator tasks:

1. **clean_missing** – removes or imputes missing values.
2. **clean_invalid** – removes inconsistent or invalid records.
3. **clean_datetime** – standardizes datetime formats.
4. **cast_types** – ensures columns have appropriate data types.
5. **validate_data** – runs data validation checks and outputs a report.
6. **transform_gold** – aggregates data into analytics-ready tables.

DAG Code (lakehouse_pipeline.py)

```python
import sys
sys.path.insert(0, '/opt/airflow/scripts')

from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
from cleanMissing import run as clean_missing_run
from cleanInvalid import run as clean_invalid_run
from cleanDateTime import run as clean_datetime_run
from castTypes import run as cast_types_run
from validate import run as validate_run
from transform import run as transform_run

with DAG(
    dag_id='lakehouse_pipeline',
    start_date=datetime(2023, 1, 1),
    schedule_interval='@daily',
    catchup=False,
    tags=['lakehouse', 'data_quality']
) as dag:

    clean_missing_task = PythonOperator(
        task_id='clean_missing',
        python_callable=clean_missing_run
    )

    clean_invalid_task = PythonOperator(
        task_id='clean_invalid',
        python_callable=clean_invalid_run
    )

    clean_datetime_task = PythonOperator(
        task_id='clean_datetime',
        python_callable=clean_datetime_run
    )

    cast_types_task = PythonOperator(
        task_id='cast_types',
        python_callable=cast_types_run
    )

    validate_task = PythonOperator(
        task_id='validate_data',
        python_callable=validate_run
    )

    transform_task = PythonOperator(
        task_id='transform_gold',
        python_callable=transform_run
    )

    clean_missing_task >> clean_invalid_task >> clean_datetime_task >> cast_types_task >> validate_task
>> transform_task
```

Processing Scripts

cleanMissing.py

```
import pandas as pd

def run():
    df = pd.read_csv('/opt/airflow/data/bronze/yellow_tripdata.csv')
    df = df.dropna()
    df.to_csv('/opt/airflow/data/silver/yellow_tripdata_cleaned.csv', index=False)
    print("Cleaning missing values complete.")

```

cleanInvalid.py

```
import pandas as pd

def run():
    df = pd.read_csv('/opt/airflow/data/silver/yellow_tripdata_cleaned.csv')
    df = df[df['fare_amount'] >= 0]
    df.to_csv('/opt/airflow/data/silver/yellow_tripdata_valid.csv', index=False)
    print("Invalid data cleaning complete.")

```

cleanDateTime.py

```
import pandas as pd

def run():
    df = pd.read_csv('/opt/airflow/data/silver/yellow_tripdata_valid.csv')
    df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
    df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'])
    df.to_csv('/opt/airflow/data/silver/yellow_tripdata_datetime.csv', index=False)
    print("DateTime standardization complete.")

```

castTypes.py

```python
import pandas as pd

def run():
    df = pd.read_csv('/opt/airflow/data/silver/yellow_tripdata_datetime.csv')
    df['passenger_count'] = df['passenger_count'].astype(int)
    df['trip_distance'] = df['trip_distance'].astype(float)
    df.to_csv('/opt/airflow/data/silver/yellow_tripdata_casted.csv', index=False)
    print("Type casting complete.")

```

validate.py

```python
import pandas as pd

def run():
    df = pd.read_csv('/opt/airflow/data/silver/yellow_tripdata_casted.csv')
    validation_report = pd.DataFrame({
        "total_rows": [len(df)],
        "missing_values": [df.isna().sum().sum()],
        "negative_fares": [len(df[df['fare_amount'] < 0])]
    })
    validation_report.to_csv('/opt/airflow/data/silver/validation_report.csv', index=False)
    print("Validation complete.")

```

transform.py

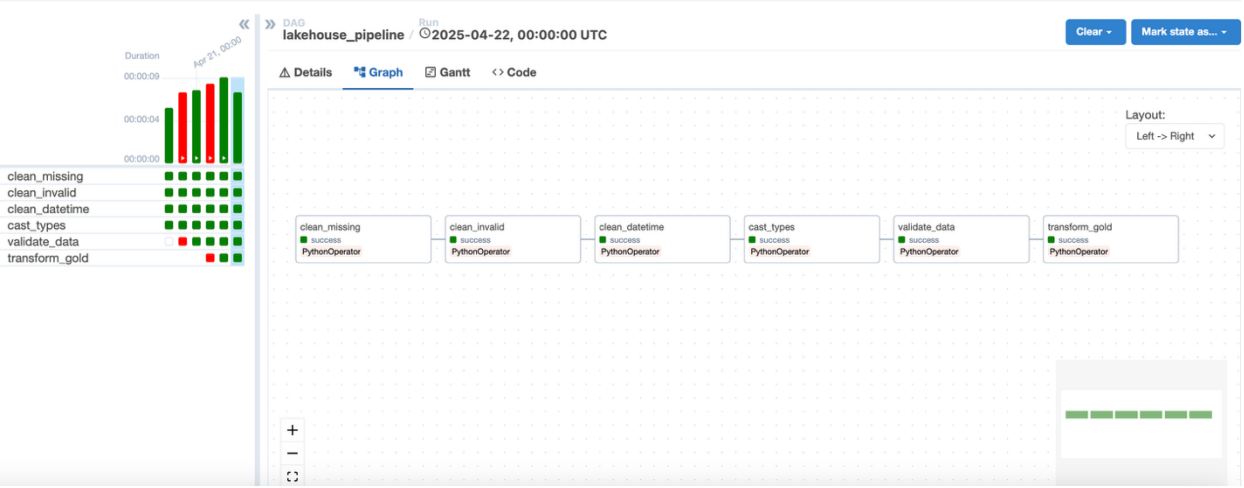
```python
import pandas as pd

def run():
    df = pd.read_csv('/opt/airflow/data/silver/yellow_tripdata_casted.csv')
    trip_summary = df.groupby('passenger_count').agg({
        'fare_amount': 'mean',
        'tip_amount': 'mean',
        'trip_distance': 'mean'
    }).reset_index()
    trip_summary.to_csv('/opt/airflow/data/gold/trip_summary.csv', index=False)
    print("Gold layer transformation complete.")

```

Orchestrating the Pipeline with Airflow DAG

- Each task in the DAG(Directed Acyclic Graph) represents a step in the lakehouse pipeline.
- Airflow ensures they run in the correct order with full traceability.

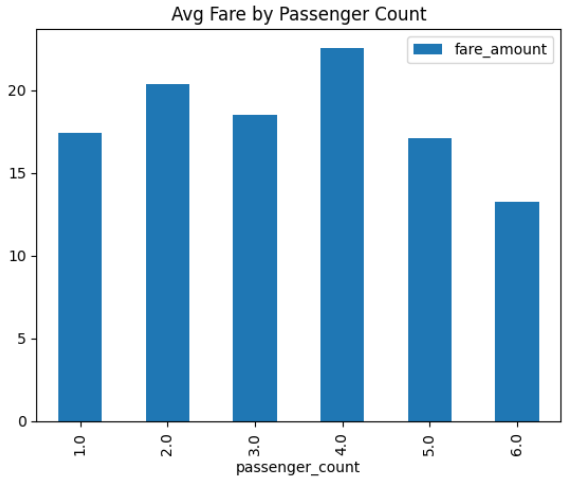


Gold Layer Output and Insight

- The Gold layer contains aggregated, analytics-ready data that was created after cleaning and validating the raw dataset through earlier pipeline stages.

passenger_count	fare_amount	tip_amount	trip_distance
1	17.386636307502314	3.4653102809509106	3.1078357517752395
2	20.348	3.8704864864864863	3.8523333333333336
3	18.490551181102365	3.4588976377952756	3.1948425196850394
4	22.548101265822783	3.733860759493671	4.012911392405063
5	17.096078431372547	3.491176470588235	3.1443137254901963
6	13.215625	2.83625	2.0993749999999998

- The Gold layer table (trip_summary.csv) is created in transform.py by grouping cleaned data and averaging key fields like fare amount, tip amount, and trip distance.



Results

- Automated end-to-end pipeline successfully executed via Airflow.
- Produced Silver layer cleaned and validated datasets with a validation report.
- Generated Gold layer aggregated summary table for analysis.
- DAG execution and logging provide full transparency and traceability for each task.

Key Learnings

- Hands-on experience with **Airflow DAG design** and task orchestration.
- Understanding of **data quality principles** and how they integrate with pipeline automation.
- Practical troubleshooting of **Dockerized environments**, path management, and dependencies.
- Modularity ensures the pipeline is **reproducible and extendable** for future datasets.

Conclusion

The project demonstrates the **implementation of a reliable Lakehouse pipeline using Apache Airflow**. The architecture ensures that datasets are cleaned, validated, and transformed into analytics-ready tables, supporting data-driven decision-making.

By modularizing the workflow and applying quality checks, the solution ensures that downstream analysis relies on **trusted and consistent datasets**.