ClimaSmart Data Pipeline Report

Platform: Google Cloud Platform (GCP)
Orchestration: Apache Airflow and Cloud Composer
Pipeline Objective: Automated weather data collection, processing, feature engineering, and quality validation for enhanced data-driven insights.

The ClimaSmart Data Pipeline is an end-to-end, automated data processing solution designed to streamline the handling of weather data, transforming raw data into validated, structured insights. The pipeline is built on Apache Airflow within Cloud Composer, leveraging GCP's cloud infrastructure for storage and processing scalability.

Integrating DVC with the ClimaSmart Data Pipeline enhances data tracking, version control, and reproducibility across all stages of the pipeline. By versioning raw, preprocessed, and feature-engineered datasets along with visualizations and schema statistics, DVC allows for efficient management and rollback of data changes, ensuring consistent and reproducible workflows. The use of remote storage (e.g., Google Cloud Storage) enables scalable storage and access, supporting the pipeline's modularity and robustness. This setup allows ClimaSmart to maintain data integrity, track lineage, and facilitate collaboration by managing large datasets and artifacts directly within a version-controlled environment.

Data Card - Daily

Variable Name	Role	Type	Description	Units	Missing Values
date	Feature	Date	Date of observation	-	No
temperature_2m_max	Feature	Float	Daily maximum temperature	°C	No
temperature_2m_min	Feature	Float	Daily minimum temperature	°C	No
daylight_duration	Feature	Float	Duration of daylight	Hours	Yes
sunshine_duration	Feature	Float	Duration of sunshine	Hours	Yes
precipitation_sum	Feature	Float	Total precipitation	mm	No
rain_sum	Feature	Float	Total rainfall	mm	No
showers_sum	Feature	Float	Total showers amount	mm	Yes
sunshine_ratio	Feature	Float	Ratio of sunshine hours to daylight hours	-	Yes
is_weekend	Feature	Binary	Whether the date falls on a weekend (0 = No, 1 = Yes)	-	No
is_hot_day	Feature	Binary	Whether it was a hot day (0 = No, 1 = Yes)	-	No
is_cold_day	Feature	Binary	Whether it was a cold day (0 = No, 1 = Yes)	-	No
is_rainy_day	Feature	Binary	Whether it was a rainy day (0 = No, 1 = Yes)	-	No
is_heavy_precipitation	Feature	Binary	Whether there was heavy precipitation (0 = No, 1 = Yes)	-	No

Data Card – Hourly

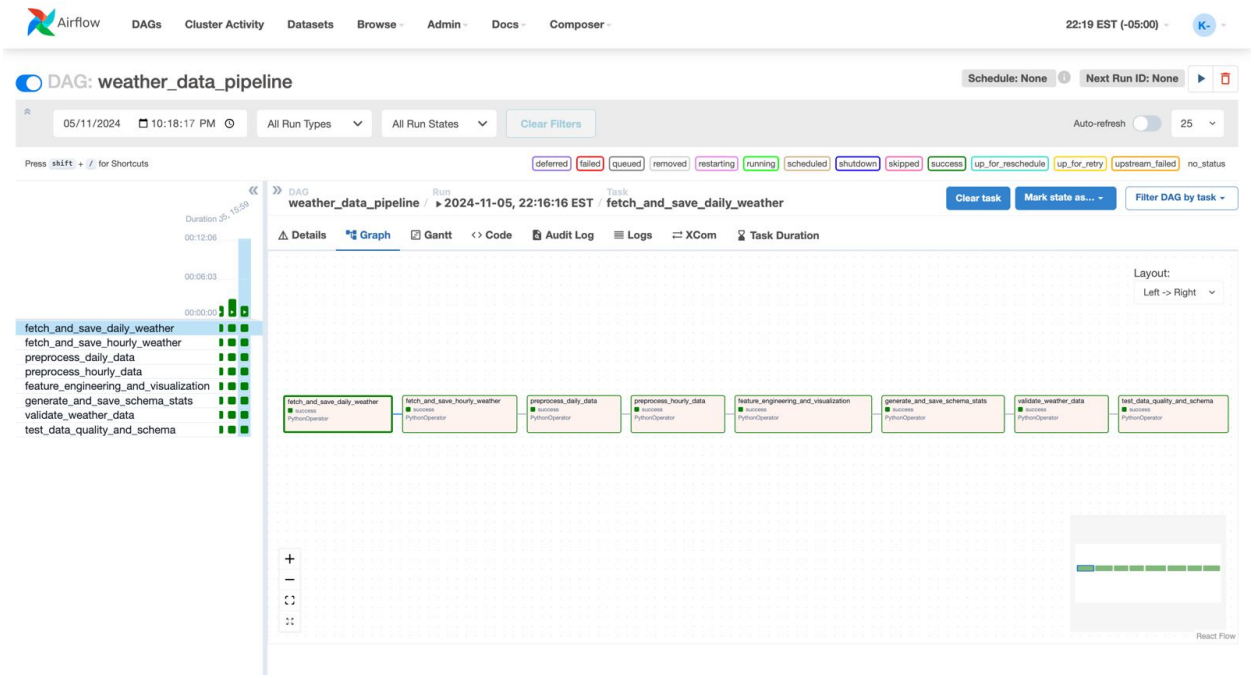
Variable Name	Role	Type	Description	Units	Missing Values
datetime	Feature	DateTime	Date and time of observation	-	No
temperature_2m	Feature	Float	Hourly temperature at 2 meters	°C	No
relative_humidity_2m	Feature	Float	Hourly relative humidity at 2 meters	%	No
dew_point_2m	Feature	Float	Hourly dew point temperature at 2 meters	°C	No
precipitation	Feature	Float	Hourly precipitation	mm	No
rain	Feature	Float	Hourly rainfall	mm	No
snowfall	Feature	Float	Hourly snowfall	mm	No
cloud_cover	Feature	Float	Cloud cover percentage	%	No
wind_speed_10m	Feature	Float	Wind speed at 10 meters	m/s	No
wind_direction_10m	Feature	Float	Wind direction at 10 meters	Degrees	No
is_day	Feature	Binary	Whether it is day (0 = No, 1 = Yes)	-	No
is_freezing	Feature	Binary	Whether it is freezing (0 = No, 1 = Yes)	-	No
is_raining	Feature	Binary	Whether it is raining (0 = No, 1 = Yes)	-	No
is_snowing	Feature	Binary	Whether it is snowing (0 = No, 1 = Yes)	-	No
temp_rolling_mean_24h	Feature	Float	24-hour rolling mean temperature	°C	Yes
precip_rolling_sum_24h	Feature	Float	24-hour rolling sum of precipitation	mm	Yes
wind_category	Feature	Categorical	Wind category based on speed	-	Yes

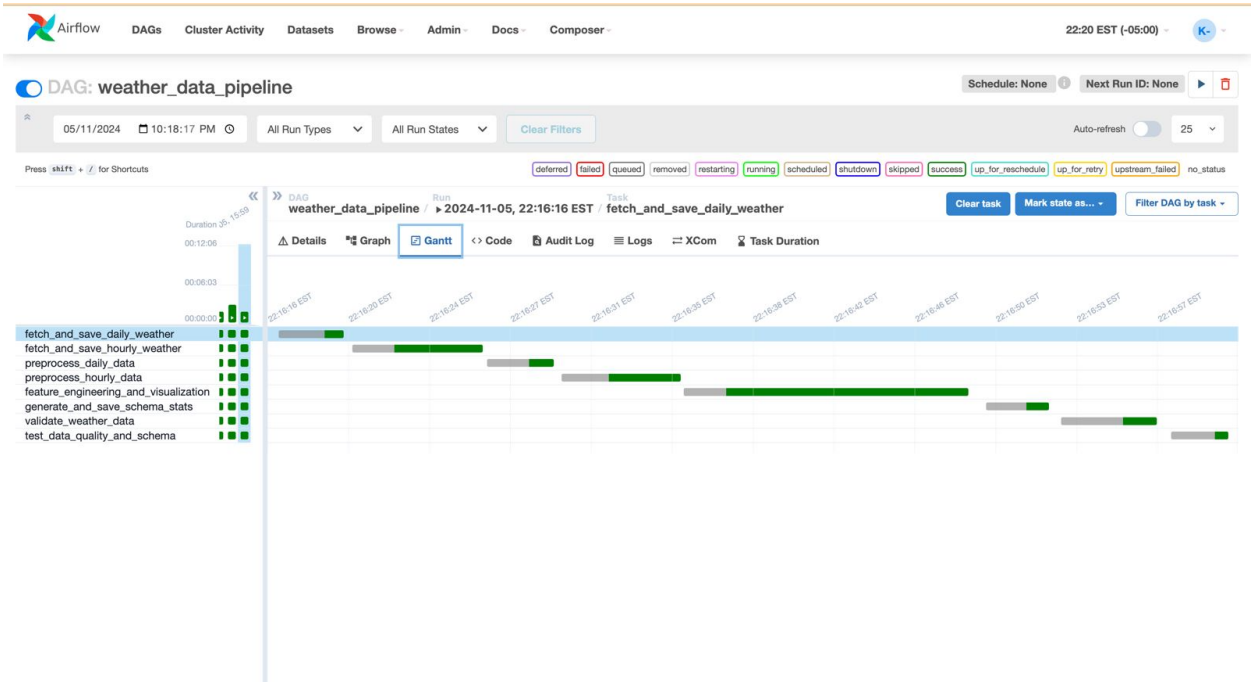
Overview of Pipeline Architecture

This pipeline consists of six core steps, each defined as a Directed Acyclic Graph (DAG) in Airflow, allowing for the efficient scheduling, monitoring, and logging of tasks. The DAGs are designed to automate both daily and hourly weather data processing, from initial data collection to final quality assurance.

DAG Overview and Status

Each stage of the pipeline is managed through individual DAGs in Apache Airflow, enabling modularity and error isolation. **All DAGs are confirmed to be running successfully**:





Detailed Workflow of Each Step

1. Fetch and Save Daily Weather Data

Objective: Automatically retrieve daily weather data from external APIs and store it in Google Cloud Storage (GCS) in a structured format.

Workflow:

- **Data Collection**:
  - API requests are made to weather data providers (e.g., Open-Meteo's Historical Weather API).
  - **requests** library is utilized for HTTP GET requests, with parameters specifying location, date, and other conditions.
- **Data Processing**:
  - The collected data is structured using **pandas**, ensuring a consistent schema, standardized time zones, and necessary transformations.
  - Columns are renamed and formatted as needed, with timestamps aligned to a standard format.
- **Storage**:
  - The processed data is saved as CSV files in **Google Cloud Storage (GCS)** for easy access by downstream applications.
- **DAG Status**: This DAG runs daily, triggered at specific intervals to ensure data freshness.

---

2. Fetch and Save Hourly Weather Data

Objective: Capture hourly weather data to support granular analysis and more responsive data-driven decisions.

Workflow:

- **Data Collection**:
  - Similar to daily data retrieval, hourly weather data is fetched through API calls using the `requests` library.
- **Data Processing**:
  - **pandas** is used to format and structure the hourly data, aligning it with the existing schema to ensure consistency.
  - Additional transformations may include aggregating data by hour or standardizing time zones.
- **Storage**:
  - The structured data is saved to **GCS** in CSV format, ready for downstream analysis and visualization tasks.

- **DAG Status**: This DAG runs every hour, capturing high-resolution data to allow for detailed and real-time weather insights.

---

3. Preprocess Daily and Hourly Data

The **weather_data_preprocessing.py** module ensures data quality by handling missing values, standardizing formats, and managing outliers.

Workflow:

- **Handling Missing Values**:
  - Missing data is managed through imputation or removal using **pandas** and **numpy** functions like `fillna` and `dropna`.
- **Data Transformation**:
  - The module standardizes temperature units, date formats, and other variables to meet analysis requirements.
  - Date and time fields are standardized, and new time-based columns are created for enhanced time series analysis.
- **Outlier Detection and Correction**:
  - Outliers are flagged and corrected using **scikit-learn**. Data is normalized and scaled where necessary to align with analysis or machine learning requirements.
- **DAG Status**: This DAG has been verified as running successfully, enabling clean and consistent data for further analysis.

---

4. Feature Engineering and Visualization

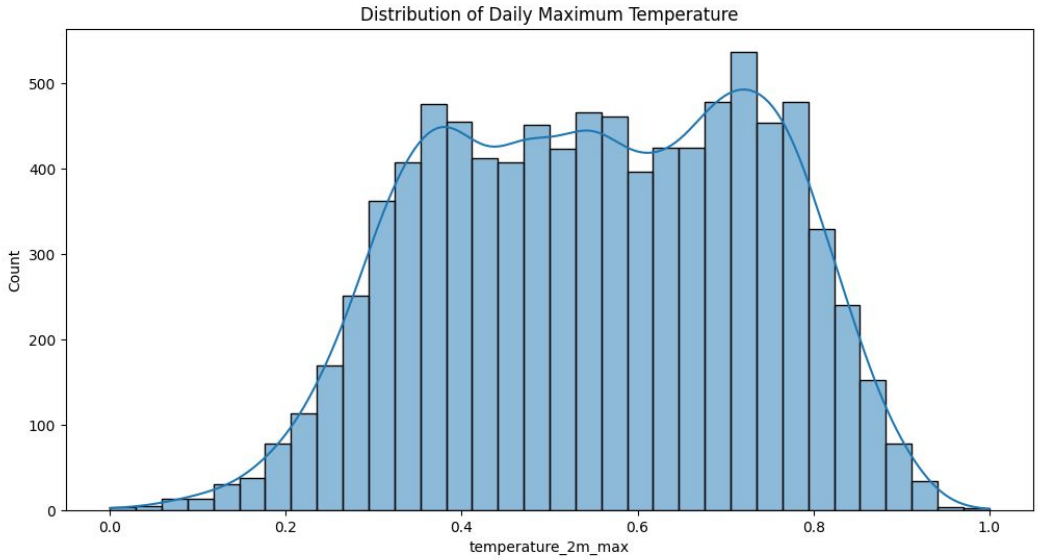
Feature engineering is a critical step to enrich the data for deeper insights, while the visualization module enables pattern detection and trend analysis.

Feature Engineering Workflow:

- **Transformations and Aggregations**:
  - The **feature_engineering.py** module creates new features, such as rolling averages, humidity indexes, or temperature differentials.
  - These engineered features provide valuable insights and enhance the data's usability for machine learning.
- **Storage**:
  - Feature-engineered data is saved to **GCS**, making it readily accessible for downstream applications.

Visualization Workflow:

- **Visualizations Created**:
  - **Time Series Plots**: Display changes in weather metrics over time.
  - **Heatmaps**: Identify high-intensity weather periods.
  - **Histograms and Scatter Plots**: Analyze data distributions and trends.
- **Libraries**:
  - **matplotlib** and **seaborn** create visuals for exploratory data analysis, which are saved as images in GCS or linked to dashboards.



- **DAG Status**: This DAG has successfully run, confirming that feature-engineered data and visualizations are generated without issues.

---

5. Generate and Save Schema Statistics

The schema statistics module provides summary metadata, such as data distributions, min/max values, and column data types, supporting quality checks and data validation.

Workflow:

- **Schema Generation**:
  - After feature engineering, schema statistics are calculated, including metrics like min/max, mean, and standard deviation values.
- **Data Storage**:
  - The schema statistics are saved to **GCS** for future reference and comparison with new data.
- **DAG Status**: This DAG runs successfully, ensuring schema statistics are readily available for validation purposes.

6. Validate Weather Data and Test Data Quality and Schema

The **weather_data_validation.py** module ensures the data meets schema standards and passes quality assurance checks.

Workflow:

- **Schema Validation**:
  - Ensures that collected data conforms to expected schema definitions in terms of column names, data types, and value ranges.
  - Uses **pandas** to check for deviations, missing columns, or inconsistent data types.
- **Quality Testing**:
  - Validates data quality with tests for duplicates, null values, and outlier detection.
  - Ensures data alignment across daily and hourly records to maintain integrity.
- **DAG Status**: This DAG runs successfully, validating data quality and schema adherence.

Technology Stack and Dependencies

The ClimaSmart pipeline relies on various tools and libraries:

- **Orchestration**: Apache Airflow in Cloud Composer manages the execution and scheduling of each DAG, with all DAGs confirmed as running successfully.
- **Data Storage**: **Google Cloud Storage (GCS)** is the primary storage for raw and processed data, schema statistics, and visualizations.
- **Python Libraries**:
  - **pandas** and **numpy**: Core libraries for data handling.
  - **requests**: Manages API interactions.
  - **matplotlib and seaborn**: Supports visualization.
  - **scikit-learn**: Used for data normalization, scaling, and outlier detection.

Orchestration and Scheduling with Apache Airflow

Apache Airflow's DAGs orchestrate each task, ensuring a well-coordinated and reliable data pipeline. Each step's dependencies are managed with Airflow, providing robust monitoring and logging for each DAG.

Scheduling Intervals:

- **Daily Pipeline**: Collects daily weather data each morning.
- **Hourly Pipeline**: Gathers hourly data on a recurring basis, enabling near real-time analysis.

All DAGs are set to trigger automatically and maintain detailed logs for auditability.

Conclusion

The ClimaSmart Data Pipeline is a robust, modular, and scalable solution that automates weather data handling. All DAGs have been confirmed to run successfully, enabling ClimaSmart to efficiently derive insights and maintain data quality with minimal manual oversight.

Summary of DAG Workflow and Status

DAG	Purpose	Status
Fetch and Save Daily Weather	Retrieves daily data	Successfully Running
Fetch and Save Hourly Weather	Retrieves hourly data	Successfully Running
Preprocess Data	Handles missing values, transformations	Successfully Running
Feature Engineering & Visualization	Creates new features and visuals	Successfully Running
Schema Statistics	Generates schema metadata	Successfully Running
Data Validation	Ensures schema and data quality	Successfully Running

This comprehensive pipeline provides ClimaSmart with a powerful data-driven platform for weather analysis, driving more effective decision-making across applications and use cases.