

Max Matteucci, Ethan Lebon, Ian Hedges, Caleb Brunton

Team Weather Trends & Prediction

Defining Project:

Section 1 (The Business Problem):

Many organizations rely on accurate, up-to-date weather information for logistics, operations, and planning, yet most tools either lack real-time updates or fail to combine live conditions with historical patterns. Our project solves this by building a real-time weather monitoring and forecasting pipeline that integrates a historical Kaggle weather dataset with continuously updated live weather data from the Open-Meteo API. By streaming new observations into BigQuery and analyzing them alongside historical data, the system will detect short-term trends, identify anomalies, and generate data-driven forecasts.

Section 2 (Data):

Kaggle:

https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data?utm_source=chatgpt.com

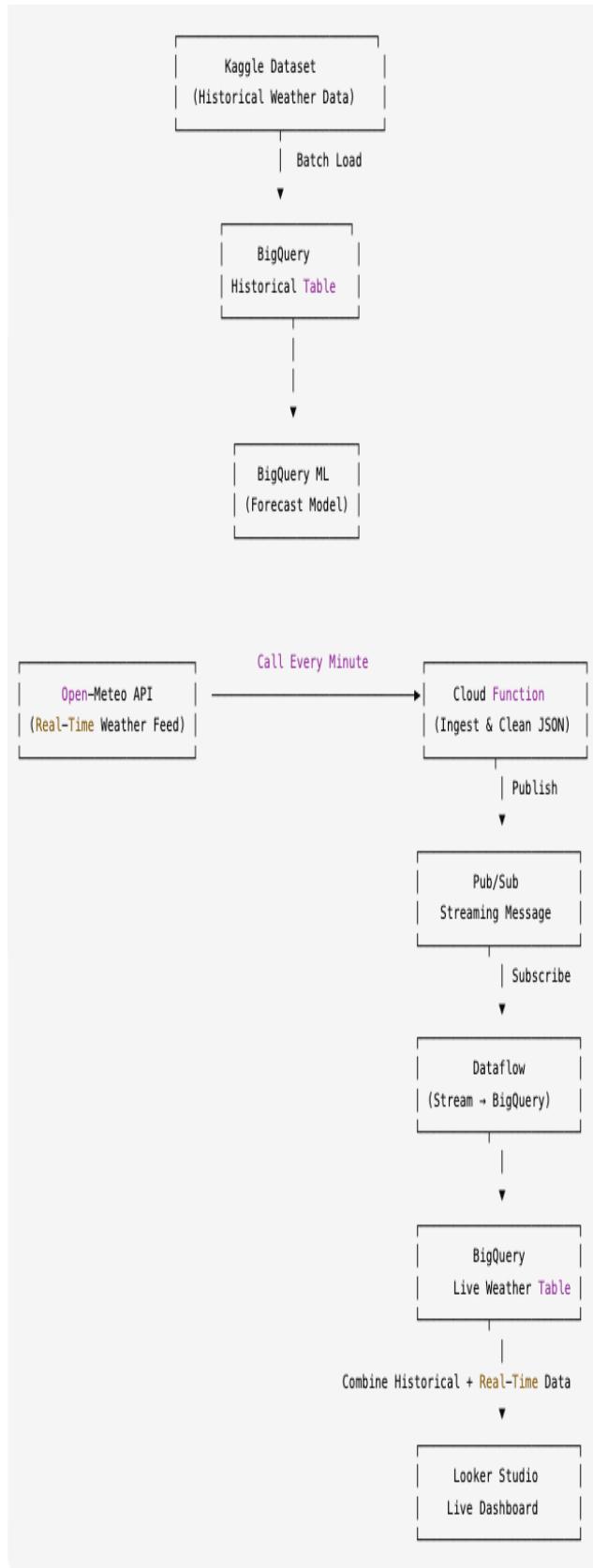
The Kaggle dataset has historical weather data for many cities in the United States and around the world. This dataset has attributes such as temperature, humidity, wind, as well as essentially any other weather attribute one can think of. For the scope of this project, this will help us develop our predictive analytics techniques, hopefully to successfully apply to forecasting weather and recognizing trends in our API.

Live API:

<https://open-meteo.com/en/docs>

This API provides live data of similar characteristics to the Kaggle Dataset. This API will allow us to have real world, modern insights, such as predicting future weather, or deriving insights from the data such as “dangerous travel conditions” or “frost warning”. Hybridizing these two sources of data will be crucial for the scope of this project.

Designing Cloud Architecture:



Our architecture combines a historical Kaggle dataset with real-time weather from the Open-Meteo API. A Cloud Function calls the Open-Meteo API every minute and pushes each response into a Pub/Sub topic. Dataflow continuously reads messages from Pub/Sub, transforms the JSON, and loads the results into a BigQuery streaming table.

The Kaggle dataset is batch-uploaded directly into a separate BigQuery historical table. BigQuery ML trains forecasting models using the combined data, and Looker Studio visualizes both real-time conditions and predictive outputs in a unified dashboard.

Plan the ML Component:

We will incorporate a machine learning component using BigQuery ML by training a regression model that predicts short-term future temperature based on historical and real-time weather features. The Kaggle dataset provides multi-year hourly measurements, which we will combine with recent observations streamed from the Open-Meteo API. Our model will use features such as current temperature, past temperature values, humidity, wind speed, wind direction, air pressure, cloud cover, and time-based variables like hour of day, day of week, and month. We will begin with a linear regression model in BigQuery ML to forecast temperature 1–3 hours ahead, and evaluate its accuracy by comparing the model's predictions against incoming live data from the API.

From there, we will take these insights and try to develop more sophisticated machine learning methods, such as predicting whether there will be a frost, or trying to find signals which predict even further out. Furthermore, we can use classification analysis to predict factors such as rain and snow. If we find these are not sufficient, we can branch out further and try more sophisticated machine learning methods.

Define the Final Dashboard:

Our Looker Studio dashboard will unify real-time weather observations, historical patterns, and machine-learning forecasts into a single interactive view. It will help users quickly understand current conditions, detect unusual weather events, and see short-term predictions generated by our BigQuery ML model. The dashboard will feature **three primary KPIs**, each aligned with our project's goals and data pipeline.

KPI 1 — Real-Time Temperature & Hourly Change

This KPI displays the current temperature pulled directly from the Open-Meteo API, along with its **hour-over-hour change** (rising, falling, or stable).

- **Purpose:** Helps users instantly assess current conditions and identify short-term shifts.
- **Data Source:** BigQuery Live Weather Table (streamed via Pub/Sub → Dataflow).
- **Calculation:**
 - `current_temp` = latest API reading
 - `temp_delta` = `current_temp` – temperature from 1 hour prior

KPI 2 — Forecasted Temperature (1–3 Hours Ahead)

This KPI presents the temperature prediction generated by our BigQuery ML regression model.

- **Purpose:** Allows users to anticipate near-term changes such as frost risk, heat spikes, or abrupt drops.
- **Data Source:** BigQuery ML model output table.
- **Calculation:**
 - Forecasted temperature using engineered features such as lag values, humidity, wind, cloud cover, and time-based variables.

This KPI directly reflects the core ML component of our project.

KPI 3 — Anomaly Detection Score (Unusual Weather Conditions)

This KPI highlights whether the current weather conditions deviate significantly from expected historical patterns.

- **Purpose:** Identifies unusual events such as sudden humidity spikes, abnormal wind speeds, or temperatures outside typical ranges.
- **Data Source:**
 - Kaggle Historical Table
 - BigQuery Live Weather Table
- **Calculation:**
 - Compute a z-score or percentile comparison using the historical distribution for the same **city**, **hour**, and **time of year**.

- Example:
 - `z = (current_temp - historical_avg) / historical_std`
- Threshold (e.g., $|z| > 2.0$) triggers an “anomaly” flag.

This KPI connects the historical data source with the real-time stream, which aligns precisely with the business problem.

Challenge: Construct Your Own Prompt:

Our Prompt

“Play devil’s advocate for our real-time weather monitoring and forecasting architecture. What is the most significant risk or potential point of failure in our design, specifically related to the ingestion of live Open-Meteo data and its integration with our BigQuery ML model? Explain why this risk is critical, and provide two or three strategies we can use to mitigate this before building.”

Gemini’s Response

Biggest Risk:

The highest-risk component of your architecture is the real-time ingestion pipeline that streams Open-Meteo data into BigQuery using Cloud Functions, Pub/Sub, and Dataflow. This part of the system has multiple external dependencies (API availability, network reliability, rate limits) and several internal moving parts that must all function correctly for your ML model to receive clean, timely data. If any single link in this chain fails—even temporarily—your BigQuery tables will have gaps, outdated timestamps, or missing values, which directly reduces the accuracy of your short-term temperature predictions.

Why This Matters:

*Your forecasting model relies on **continuously updated** observations to make accurate 1–3 hour predictions. Time-series ML models are extremely sensitive to missing or irregular intervals, so if API calls fail or the Cloud Function stops executing at the expected frequency, your model will be trained or evaluated on incomplete data. This could produce unreliable predictions, distort anomaly detection scores, and compromise the dashboard KPIs that depend on real-time values.*

Mitigation Strategies:

1. **Implement Robust Retry and Error Handling in Cloud Functions**
 - Add retry logic with exponential backoff when API calls fail or time out.
 - Log all failed requests and send them to a “retry” Pub/Sub topic so they are not lost.
This prevents small API disruptions from causing long-term data gaps.
2. **Add Data Quality Monitoring in BigQuery**

- *Create checks that detect missing timestamps or unusually long intervals without new data.*
- *Use Cloud Monitoring alerts to notify the team automatically when expected data does not arrive.*
*This ensures ingestion problems are detected **immediately** rather than discovered later during ML evaluation.*

3. **Use a Graceful Fallback Mode**

- *If live data is unavailable, temporarily substitute last-known values, interpolated values, or historical averages.*
- *This keeps the ML model and dashboard functional until the live stream resumes. It prevents the entire pipeline from breaking due to short outages.*