# Real-time IoT Data Pipeline — Proposal & Implementation Playbook (Predictive Analytics + Geospatial Dashboard)

## Project Title

Real-time IoT Data Pipeline with Predictive Analytics and Geospatial Dashboard

## Project Overview (1–2 lines)

Build an end-to-end IoT data pipeline that simulates sensor streams (temperature & humidity), performs batch and streaming processing, trains a predictive model to forecast future sensor readings and anomalies, and visualizes results on an interactive geospatial dashboard.

## Problem Statement

Many organizations collect sensor data but rely on basic threshold alerts or manual inspection. This project aims to move from reactive monitoring to proactive insights by forecasting short-term sensor behavior and showing spatial patterns on a map so operators can act before problems escalate.

## Objectives

* Simulate realistic IoT sensor streams and ingest them into a message queue.
* Implement batch ETL and a streaming analytics layer.
* Build a predictive model to forecast temperature/humidity for the next N minutes/hours and flag likely anomalies.
* Create an interactive geospatial dashboard (heatmap + time slider) showing live and forecasted values per sensor.
* Demonstrate deployment-ready components and document the pipeline.

## Scope & Success Criteria

**In scope:** simulation, ingestion, storage, preprocessing, predictive model, streaming alerts, geospatial visualization, documentation.

**Success criteria:** - Pipeline ingests simulated data every 5 seconds without backlog for the test scale (e.g., 100 sensors). - Prediction model achieves acceptable error on synthetic holdout set (e.g., MAE < chosen threshold) and detects injected anomalies. - Dashboard updates live and shows forecast overlay and heatmap of sensor values.

## Data Simulation — Schema & Realism Guidelines

**Schema (CSV / JSON events):** - timestamp (ISO8601) - sensor\_id (string) - latitude, longitude (float) - temperature\_c (float) - humidity\_pct (float) - battery\_v (float) - status (string: OK / ERROR)

**Simulation parameters & techniques:** - Assign each sensor a base temperature/humidity (depends on location). Use a daily sinusoidal pattern + random Gaussian noise to mimic diurnal variation. - Add long-term trend or drift for some sensors to simulate sensor degradation. - Inject anomalies deliberately (spikes, dropouts, stuck-at-value, unrealistic jumps) at controlled rates (e.g., 0.5–2% of events) so anomaly detection can be validated. - Add packet loss / missing timestamps and out-of-order events to reflect real ingestion issues. - Randomize transmission jitter (not exactly every 5s for all sensors — add ±1–3 seconds) to simulate network jitter.

**Why realistic simulation matters:** - Models trained only on perfectly clean synthetic data often fail on real-world data. Introducing noise, anomalies, and distributional variability reduces overfitting to ideal conditions.

## Ingestion & Storage

**Options:** MQTT -> Kafka or direct Kafka producer (Python). For simplicity: Python generator -> Kafka topic (or file-based logging for first milestone).

**Storage:** - Raw events: append to object storage (e.g., Azure Blob / S3) or HDFS for batch. - Processed/batch tables: SQL database or Delta Lake / Parquet files partitioned by date. - Time-series DB (optional): InfluxDB or TimescaleDB for efficient queries.

## ETL & Streaming

* **Batch ETL:** Use Python scripts or Azure Data Factory to read raw files, clean, aggregate (5-minute, hourly), compute rolling stats, and store final tables.
* **Streaming:** Use Kafka + Spark Streaming / Flink / Azure Stream Analytics to compute near-real-time metrics and apply alert logic.
* **Alerting:** Two-layer alerts: (1) threshold-based (simple), (2) model-based predictions that estimate probability of crossing threshold within time window.

## Predictive Model — Design & Implementation

**Task goal:** forecast next T steps of temperature/humidity for each sensor (e.g., 15 min / 1 hour) and predict probability of breach.

**Candidate algorithms:** - Classical: Facebook Prophet (good for seasonality), ARIMA (baseline). - ML tree-based: XGBoost / LightGBM on windowed features. - Deep learning: LSTM/GRU for sequence modeling, Temporal Fusion Transformer (if time permits).

**Feature ideas:** - Lag features: t-1, t-2, … t-k - Rolling statistics: mean, std, min, max over last N readings - Time features: hour-of-day, day-of-week, sine/cosine for daily cycle - Spatial features: neighboring sensors’ average (if available) - Meta-features: battery level, transmission interval

**Training plan:** - Split by time: train on earliest 70%, validate 15%, test last 15% to maintain temporal integrity. - Cross-validate across multiple simulated scenarios (different noise seeds) to ensure robustness.

**Evaluation metrics:** - Regression: MAE, RMSE, MAPE (be careful with near-zero denominators) - For anomaly/breach detection: Precision, Recall, F1, ROC-AUC

**Model serving:** - Expose prediction service as a REST endpoint (e.g., FastAPI) that the streaming layer or dashboard can call. - Optionally run model inference within streaming job (Spark UDF or Flink operator) for low-latency predictions.

## Geospatial Dashboard — UI Design & Tools

**Key features:** - Map with sensors as points; circle size or color denotes current value. - Heatmap layer showing aggregated temperature/humidity across area. - Time slider to move back and see historical snapshots. - Forecast overlay: show predicted temperature for each sensor for next N steps (e.g., small trend line on hover). - Alert layer: highlight sensors likely to breach soon.

**Tools:** - Web: Leaflet.js / Mapbox + a small React or Streamlit app. - BI: Power BI Map visuals or Grafana Worldmap plugin. - For prototyping: Streamlit + pydeck or folium works fast.

## Deployment & Observability

* Use Docker for components (producer, stream processor, model server, dashboard).
* Basic monitoring: logs, a simple Prometheus + Grafana stack (optional) to monitor pipeline health.
* CI: GitHub Actions to run lints and tests; pipeline deployment scripts (docker-compose for demo; Kubernetes manifests if advanced).

## Milestones & Timeline (Suggested 12-week plan)

**Week 1–2 — Proposal & Data Simulation** - Deliverables: Project Proposal, team roles, Python data generator + sample logs.

**Week 3–4 — Ingestion & Storage** - Deliverables: Kafka/topic setup or file ingestion pipeline; raw storage with sample partitions.

**Week 5–6 — Batch ETL & Preprocessing** - Deliverables: ETL scripts, cleaned datasets, feature store or preprocessed Parquet files.

**Week 7–8 — Predictive Modeling** - Deliverables: Trained model(s), evaluation report, saved model artifact.

**Week 9 — Streaming & Alerts Integration** - Deliverables: Streaming job with model inference or calls to model server; alert output logs.

**Week 10 — Geospatial Dashboard** - Deliverables: Interactive dashboard (live demo or screenshots), heatmap + forecast overlay.

**Week 11 — Hardening & Documentation** - Deliverables: README, architecture diagram, runbook, code comments.

**Week 12 — Final Presentation & Submission** - Deliverables: PowerPoint (DEPI template), final report (PDF), GitHub repo, live demo or recorded demo.

## Team Roles Template (Example)

* Team leader / PM: coordination, mentor communication, final presentation
* Data Engineer: ingestion, storage, streaming
* ML Engineer: modeling, evaluation, model serving
* Frontend / Dashboard: dashboard frontend and UX
* QA / Documentation: testing, report writing, deployment scripts

## Risks & Mitigations

* **Risk:** Synthetic data not matching real-world distribution → **Mitigation:** Introduce noise, multiple simulation scenarios, and try to locate at least one small public real dataset to fine-tune the model.
* **Risk:** Pipeline bottlenecks under higher scale → **Mitigation:** test locally with increasing load and optimize partitions, parallelism.
* **Risk:** Time constraints → **Mitigation:** prioritize core features (ingestion, model prototype, dashboard) and mark advanced features as stretch goals.

## Deliverables (Checklist)

* Python data generator + sample logs
* Ingestion scripts / Kafka setup (or file-based ingestion)
* Batch ETL scripts and cleaned datasets
* Trained model and evaluation report
* Streaming job with alert outputs
* Geospatial dashboard (live or screenshots)
* Final Presentation (PowerPoint) + Final Report (PDF)
* GitHub repository (fixed link)

## Quick Proposal Text (copy-paste-ready)

**Title:** Real-time IoT Data Pipeline with Predictive Analytics and Geospatial Dashboard

**Summary:** This project builds an end-to-end IoT pipeline that simulates sensor streams, processes data in batch and streaming modes, trains a predictive model to forecast near-term sensor values and possible threshold breaches, and visualizes results on an interactive geospatial dashboard. The pipeline demonstrates proactive monitoring, model-driven alerts, and spatial analytics.

**Deliverables:** Data generator, ETL & streaming pipeline, trained forecasting model, alert logic, interactive map dashboard, final report and presentation.

## Immediate Next Steps (what to do now)

1. Confirm team members and assign roles.
2. Finalize the number of sensors and spatial area for simulation (e.g., 50 sensors across city X).
3. Implement the Python simulator and produce the first 24 hours of sample logs.
4. Prepare the GitHub repo and initial README.

If you want, I can also: - Provide the Python simulator script (ready-to-run) as a separate file. - Provide a starter Jupyter notebook for training a baseline forecasting model (Prophet or LSTM). - Generate a simple Streamlit dashboard prototype that maps sensor points and shows live values.

*Prepared for DEPI Graduation Project — AI & Data Science Track (Round 2)*