



Prediction of air quality: Project Scoping

Machine Learning Operations IE7374

09.30.2024

Team members: Donya Dabiri, Srilakshmi Kanagala, Sparsh Marwah,
Anirudha Raghava Sarma Kuchibhotla, Vachan Srinivasagowda,
Swapna Vippaturi

Advisor: Dr. Ramin Mohammadi

1. Introduction	3
2. Dataset Information	3
Dataset Introduction	3
Data Card	3
Data Sources	4
Data Quality and Challenges	4
Data Rights and Privacy	4
3. Data Planning and Split	4
Handling Missing Values	4
Data Splitting Strategy	5
4. Github Repository	5
5. Project Scope	7
Problem	7
Current Solution	7
Proposed Solution	7
6. Current Approach Flow Chart and Bottleneck Detection	7
Current Approach	7
Bottleneck Detection	10
7. Metrics, Objectives, and Business Goals	10
8. Failure Analysis	10
Potential risks to the project can be categorized into 4 categories	10
Data-related risks	10
Model-related risks	11
Pipeline-related risks	11
Post deployment risks	11
9. Deployment Infrastructure	11
Model Packaging	11
Model deployment	12
CI/CD pipeline	12
Monitoring and logging	12
10. Monitoring Plan	13
Airflow Monitoring	13
MLflow Monitoring	13
Joint Airflow-MLflow Monitoring	13
Alerts and Notifications	14
11. Success and Acceptance Criteria	14
Model Performance and Evaluation	14
Reproducibility with MLflow	15
Automation and Workflow Management with Airflow	15

Containerization with Docker.....	16
Scalability and Deployment on Google Cloud Platform (GCP).....	16
12. Timeline Planning	17
Phase 1: MLflow, Data Processing, and Modeling (Week 1).....	17
Phase 2: Airflow and GCP Setup (Week 2-5).....	17
Phase 3: Docker Setup (Week 6).....	18
Phase 4: Continuous GitHub Integration (Throughout the Project).....	18

1. Introduction

Project Scoping Submission: Air Quality Prediction for PM2.5 & PM10 Using OpenAQ Data

Air pollution remains a significant global health and environmental challenge. It has been proven that pollutants in air lead to severe respiratory and cardiovascular diseases. This project aims to predict PM2.5 & PM10 (particulate matter with a diameter of less than 2.5 micrometers & 10 micrometers) concentrations using data aggregated by OpenAQ, an environmental tech nonprofit that collects air quality data from various sources globally. The dataset includes air quality measurements from reference-grade government monitors and air sensors. The project seeks to leverage this data to develop predictive models, enabling better forecasting of PM2.5 & PM10 levels, facilitating proactive measures, and informing policy-making to ensure healthier living environments.

By leveraging OpenAQ's extensive and openly accessible air quality data, this project aims to create a robust framework for understanding air pollution trends and identifying potential high-risk areas. The resulting model will serve as a tool for researchers, policymakers, and environmentalists to address air pollution and advocate for cleaner air.

2. Dataset Information

Dataset Introduction

The OpenAQ dataset consists of air quality data collected from various sources worldwide, including government and research institutions. The dataset includes key pollutants such as PM2.5, PM10, CO, NO2, SO2, and O3, along with metadata like location, timestamp, and data source information. The data is continuously updated, ensuring up-to-date information for monitoring and analysis.

Data Card

Size: Varies based on the region and selected data range.

Format: JSON/CSV

Features: Temperature, Humidity

Location: City, country, and geographic coordinates.

Date and Time: Timestamps of each measurement.

Pollutants: PM2.5, PM10, CO, NO2, NO, O3 concentrations.

Source: Information on the data source, including reference-grade monitors or low-cost sensors.

Units: Measurements are reported in standardized units (e.g., $\mu\text{g}/\text{m}^3$ for PM2.5), Fahrenheit.

Data Sources

The dataset is sourced from [OpenAQ's centralized platform](#) and the [air quality dataset](#) from the UCI website, which aggregates data from a range of official sources, including governmental and research bodies. The data is harmonized to ensure consistency and usability across regions and time periods.

Data Quality and Challenges

- Missing Data: Missing or inconsistent values may exist due to data availability or sensor issues.
- Data Discrepancies: Variations in pollutant measurement units and data collection frequencies require careful preprocessing to ensure uniformity. Different cities have different sets of features as it is collected from different sources.

Data Rights and Privacy

The dataset is publicly available through OpenAQ's open-access platform and can be used for research and analysis purposes. There are no privacy concerns as the data does not contain personally identifiable information (PII).

3. Data Planning and Split

Handling Missing Values

Missing values in the dataset, often due to sensor malfunctions or data collection inconsistencies, will be handled using various techniques:

- Imputation: Employing mean or median imputation to fill missing values.
- Forward/Backward Filling: Using forward and backward filling for time-series data to maintain the temporal sequence.

Data Cleaning and Preprocessing

- Outlier Detection: Outliers will be detected using statistical methods like IQR and Z-score. Extreme values will be transformed or capped to minimize their impact.
- Unit Standardization: All pollutant concentrations will be converted to a common unit (e.g., $\mu\text{g}/\text{m}^3$) to ensure consistency, and all the units for the temperature column will be converted to a common unit (e.g., F).

Data Splitting Strategy

To ensure model generalization and robust evaluation, the data will be divided into three distinct subsets:

1. Training Set (70%)* Used to train the model and learn patterns.
2. Validation Set (15%): Used to fine-tune the model and avoid overfitting.
3. Testing Set (15%): Used for final evaluation to measure the model's performance on unseen data.

This approach ensures that the model can effectively predict PM_{2.5} & PM₁₀ levels while minimizing overfitting and enhancing its ability to generalize across different geographical regions and time periods.

4. Github Repository

```

project-root/
|
|— LICENSE          <- Project license information.
|— README.md        <- Overview, setup, and usage details.
|
|— data/
|   |— raw/          <- Original, immutable data dump.
|   |— interim/      <- Intermediate data, after transformations.
|   |— processed/    <- Final data, ready for modeling.
|
|— docs/            <- Documentation files and manuals.
```

```

|
|— models/          <- Trained models, predictions, and model summaries.
|
|— notebooks/       <- Jupyter notebooks for analysis and modeling.
|
|— references/       <- Data dictionaries, research papers, and manuals.
|
|— reports/         <- Generated analysis in HTML, PDF, LaTeX formats.
|   |— figures/      <- Graphics and figures for reports.
|
|— requirements.txt  <- Required dependencies for the project.
|
|— setup.py         <- Setup script to install the project as a package.
|
|— src/             <- Source code for use in this project.
|   |— __init__.py   <- Makes src a Python module.
|   |— data/         <- Scripts for data download and generation.
|       |— make_dataset.py
|   |— features/     <- Scripts for feature engineering.
|       |— build_features.py
|   |— models/       <- Scripts for training and using models.
|       |— predict_model.py
|       |— train_model.py
|   |— visualization/ <- Scripts for visualizations.
|       |— visualize.py

```

Github Link : <https://github.com/KARSarma/Air-Quality>

5. Project Scope

Problem

Climate change intensifies air pollution, increasing the levels of harmful particulate matter (PM2.5, PM10) in the atmosphere. These pollutants pose significant health risks as they penetrate deep into the lungs and bloodstream, causing respiratory and cardiovascular diseases. Accurately predicting PM levels is challenging due to the complex interactions between climate factors and emissions.

Current Solution

The widely-used Autoregressive Integrated Moving Average (ARIMA) model provides baseline predictions of pollutant concentrations based on historical data. While it captures linear trends and patterns in pollution data, ARIMA is limited in handling non-linear relationships, seasonality, and the complex interactions between various climate factors and pollutants.

Proposed Solution

To address these limitations, the project proposes using advanced machine learning models, particularly Long Short-Term Memory (LSTM) networks, XGBoost, and other models, to predict PM2.5 and PM10 levels.

6. Current Approach Flow Chart and Bottleneck Detection

Current Approach

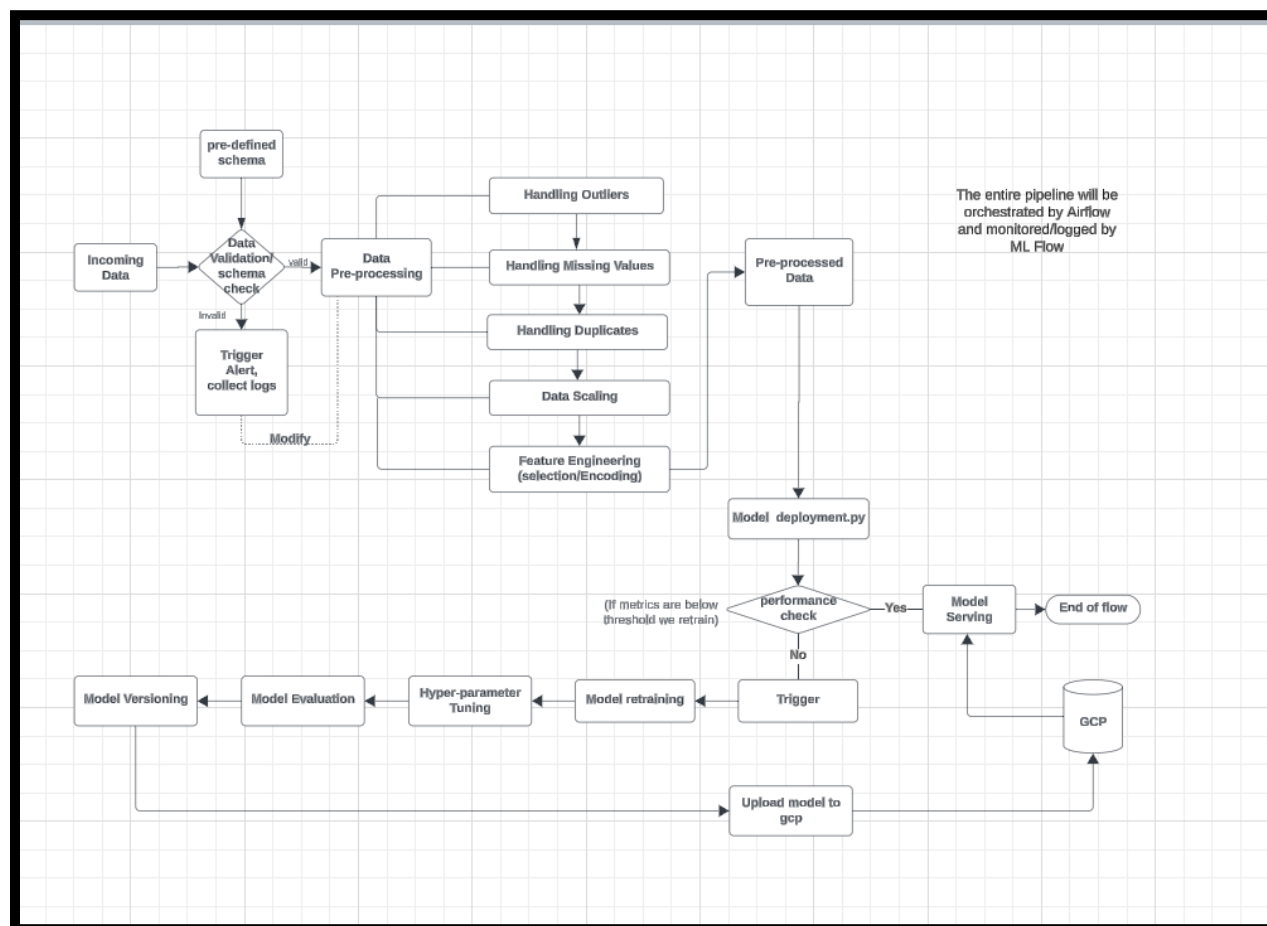
1. Data Ingestion

Continuously collect new data from various sources like APIs, databases. This data might include new measurements or environmental changes relevant to the model's predictions. Store the new data in a data storage system (e.g., cloud storage, databases) to be used for preprocessing and model retraining.

2. Data Validation and Quality Checks

Before using new data for predictions or model retraining, ensure its quality by performing checks like those done during the initial pipeline (e.g., schema validation, handling missing values, checking for outliers).

FlowChart:




3. Data Preprocessing

Apply the same preprocessing steps used during model training on the new data. This includes normalization, feature engineering, encoding, and handling missing values. The preprocessed data is stored in a format ready for inference and, if required, for use in retraining the model.

4. Batch Inference

Use the deployed model to make predictions on new data in either batch. The data pipeline periodically collects new data (e.g., daily, weekly) and processes it in bulk to generate predictions. Apache Airflow to schedule jobs for batch processing. Save predictions to a database or a data warehouse for downstream applications or user access.

5. Monitoring the Deployed Model



Continuously monitor the model's performance on new data to detect potential issues like data drift, concept drift, and performance degradation. Track metrics such as accuracy, RMSE, data distribution changes, prediction confidence, and latency. Set up alerts to notify the team if there is a significant drop in performance or if data drift is detected.

6. Triggering Retraining (if needed)

If monitoring detects significant changes (e.g., data drift, performance drop), the pipeline triggers model retraining using the new data.

Data Collection: Aggregate the new data with historical data for model retraining.

Data Splitting: Split the aggregated dataset into training, validation, and test sets.

Retraining: Run the model training script using the latest dataset.

Hyperparameter Tuning: Optionally tune hyperparameters during retraining for optimal performance. Apache Airflow (scheduling retraining jobs), MLFlow for tracking experiments, custom Python scripts for retraining logic.

7. Model Evaluation and Validation

Evaluate the newly retrained model on a validation set to ensure it meets predefined performance metrics (e.g., accuracy, RMSE) before replacing the deployed model.

8. Model Versioning and Deployment


If the retrained model passes the evaluation phase, deploy it to production while versioning the previous model for rollback if necessary. Register the new model version using MLFlow Model Registry. Deploy the new model, either by replacing the existing model endpoint or by using A/B testing for a gradual rollout.

9. Continuous Monitoring and Feedback Loop

After deploying the new model, continue to monitor its performance on the incoming data to maintain an ongoing feedback loop for further improvements using cloud monitoring services (GCP Monitoring), logging tools (Google Cloud Logging).

10. Airflow and ML flow monitoring and logging

Airflow will monitor DAG runs, task failures, and resource usage with its UI and by integrating Prometheus and Grafana for performance dashboards and alerts. MLflow will track model performance, data/model drift, and manage model versions through its Model Registry, with predictions and metrics logged after each batch run. Together, Airflow and MLflow will allow for seamless monitoring of batch predictions by adding Airflow tasks to log results in



MLflow and setting up automated retraining workflows based on model performance. Alerts for task failures, SLA breaches, or performance drops will be set up via email or Slack from both systems.

Bottleneck Detection

- Lack of knowledge regarding the tools related to a ML Ops project.
- Inconsistency in the data. e.g. different cities have different sets of features.
- Different systems lead to different behavior of all the tools.

7. Metrics, Objectives, and Business Goals

The accuracy of the machine learning model is evaluated using root mean squared error (RMSE), mean absolute error (MAE), and R^2 . The effectiveness of the pipeline should also be tracked. Some of the metrics for pipeline evaluation are model deployment time and retraining frequency. These two metrics are subjected to change during the process due to their dependency on the data. For the time being, model deployment time is targeted to be less than an hour and the retraining frequency would be monthly.

The project aligns with broader societal goals that address public health challenges. By predicting air quality and providing early warning, this project helps in minimizing health risks associated with air pollution. It contributes to better health outcomes and lowers healthcare costs. It also helps cities make more effective decisions about pollution control and come up with more fruitful guidelines. Moreover, informing citizens of the air pollution prediction can help raise awareness of air quality issues and can foster a culture of environmental awareness.

8. Failure Analysis

Potential risks to the project can be categorized into 4 categories:

Data-related risks

Collected data from sensors and weather stations might be noisy, incomplete or have outliers due to hardware failure. Poor data can lead to inaccurate predictions. To alleviate this challenge, robust data cleaning and outlier detection procedures are implemented to handle missing values or outliers. There might also be data drift, changing the weather and pollution patterns over time due to the season change or global warming. To avoid this issue, the model can be retrained regularly with new data as part of the pipeline. Moreover, certain time periods of conditions

might have fewer data points which affect accuracy. Hence, data augmentation techniques can be used to handle such situations.

Model-related risks

Firstly, the model might be overfitting, which leads to poor performance for new data. To avoid overfitting, regularization methods can be used during model training.

Pipeline-related risks

Pipeline components could fail due to infrastructure issues or software bugs, resulting in missed predictions or inability to retrain or redeploy the model. To bypass this issue, Airflow can be used for monitoring pipeline tasks.

Continuous integration (CI) and continuous deployment (CD) might fail to deploy new models due to code errors or dependency issues, causing delays in updating models. Github actions are used to automate testing and deployment to avoid this issue.

Post deployment risks

Model's performance is subject to degradation over time due to changes in climate, pollution patterns, industry of the city and even lifestyle of the citizens. Hence, by the passage of time, the predictions will be inaccurate. A solution to this issue is implementing Airflow to retrain the model with new data automatically. In other words, Airflow comes to help in automating the retraining process.

9. Deployment Infrastructure

The steps involved in the deployment process are:

- Model Packaging
- Model Deployment
- CI/CD pipeline
- Monitoring and logging

Tools stack: Jupyter notebook, Docker, GitHub, Google Cloud Services (Google Data Lake, Google Data Warehouse), Airflow, ML flow

Model Packaging

- After the final model is selected, the model will be saved in a pickle file using Scikit-learn (.pkl file).

- Post this, the dependencies of the model will be defined using requirement.txt.
- Finally, the model will be containerized using the docker platform.

Model deployment

- The Airflow platform will be used to periodically run the model on new batches of data.
- The new batches of data will be stored in **GCP or Google cloud platform** and will be fed to the model using batch data processing.
- Model predictions will then be sent back to the GCP after each batch job done through airflow pipelining.

CI/CD pipeline

- Automate the deployment and retraining pipeline for batch models using Airflow for orchestration.
- For versioning of both code and model artifacts, a GitHub **repository** will be used. Furthermore, the whole ML process will be divided into three parts: features, testing, and production to manage changes effectively.
- Airflow will be integrated with the version control system to automate the whole process using DAGs (Directed Acyclic Graphs) and will be divided into three steps as building, testing, and deploying the batch model.

Build stage: Compilation of code and dependency check.

Test stage: Batch processing tests to ensure they work correctly with new data.

Deploy stage: After successful testing, Trigger the batch job DAG in airflow to deploy new model version

- These jobs will be scheduled on a time interval / successful runs of previous jobs basis.
- Output will be an automated batch prediction pipeline managed by Airflow, with control for code and models, ensuring a structured and repeatable deployment process.

Monitoring and logging

ML flow: Model-centric concerns will be monitored using ML flow such as tracking model performance, model registry, batch-job logging, and model drift.

Airflow: Airflow will be used for scheduling batch jobs based on time intervals / successful runs of previous jobs.

10. Monitoring Plan

Airflow Monitoring

DAG and Task Monitoring

Monitor: DAG runs, task failures, task duration.

How: Use Airflow UI for real-time tracking and set up email/Slack alerts for failures.

Tools: Airflow logs, built-in alerts.

Resource and Performance Monitoring

Monitor: CPU, memory usage, and task performance.

How: Integrate Airflow with Prometheus and Grafana to visualize resource usage.

Tools: Prometheus + Grafana for dashboards and alerts.

MLflow Monitoring

Model Performance Monitoring

Monitor: Model metrics (accuracy, RMSE, etc.) and data/model drift.

How: Log predictions and performance metrics after each batch run to MLflow.

Tools: MLflow UI for metrics visualization, Evidently AI for drift detection.

Model Version Control

Monitor: Track and compare different model versions.

How: Use MLFlow's Model Registry for version control.

Tools: MLflow for Model Registry and version management.

Joint Airflow-MLflow Monitoring

Logging Predictions in MLflow from Airflow

Monitor: Ensure batch predictions are logged to MLflow.

How: Add an Airflow task to log batch results (predictions) into MLflow.

Tools: Airflow tasks + MLflow API.

Automated Retraining

Monitor: Trigger retraining when performance drops.

How: Use Airflow to periodically check model performance in MLflow and schedule retraining if needed.

Tools: Airflow DAG with MLflow performance monitoring.

Alerts and Notifications

Airflow Alerts: Task failures or SLA breaches.

MLflow Alerts: Custom scripts for performance drops.

Tools: Email/Slack alerts from both Airflow and MLflow.

11. Success and Acceptance Criteria


The air quality detection project leverages a powerful pipeline integrating MLflow, Airflow, Docker, and Google Cloud Platform (GCP), specifically utilizing GCP Data Warehouse for efficient data storage and management. The project aims to deliver air quality forecasts using time series analysis, while ensuring automation, scalability, and cost-efficiency.

Criteria for Success

- The pipeline should provide air quality predictions with a Mean Absolute Error (MAE) target of X units.
- The system should be fully automated and scalable across different environments.

Criteria for Acceptance

- The system must handle scaling to manage increasing data loads while maintaining prediction accuracy.
- Comprehensive documentation should enable reproducibility.



Model performance is a key success factor, with Mean Absolute Error (MAE) used as the primary metric for prediction accuracy. Supporting metrics, such as Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE), provide additional insights into how well the model handles variances in air quality levels. RMSE is also used as a threshold parameter to trigger the model retraining for maintaining the model health for the business goal.

Criteria for Success

- The model should maintain MAE between X units.
- RMSE and MAPE should show that the model performs well across varying datasets.
- Performance should improve over baseline models by at least 5-10%.

Criteria for Acceptance

- The model must consistently meet the targeted MAE and perform well in live testing environments.

Reproducibility with MLflow

To ensure full reproducibility, MLflow tracks all experiments, including hyperparameters, metrics, and models. This enables consistent model reproduction across different environments.

Criteria for Success

- All experiments should be logged with MLflow, including key parameters and evaluation metrics.
- Models must be reproducible with minimal variation in results across different platforms.

Criteria for Acceptance

- The project will be accepted if model experiments can be recreated with consistent performance (within a 1-2% variation) across environments.
- Successful deployment of the model on at least two platforms is required without any major configuration issues.

Automation and Workflow Management with Airflow

Automation is critical for managing tasks such as data ingestion, preprocessing, model training, and evaluation. Airflow automates these workflows, ensuring efficient execution and retraining when new data is available.

Criteria for Success

- The pipeline must be fully automated using Airflow, with a target uptime of 90-95%.
- Model retraining should occur automatically when significant data updates are available.

Criteria for Acceptance

- The project will be accepted if Airflow reliably handles data ingestion, model retraining, and evaluation with minimal manual intervention.
- Task failure rates should remain below 2-3%, with reliable task retries.

Containerization with Docker

Docker is used to containerize the entire system, ensuring portability and consistency across different environments. This allows for easy scaling and deployment of the system on GCP.

Criteria for Success

- Docker containers must encapsulate all dependencies, making the system environment-agnostic.
- Containers should be lightweight and efficiently handle both development and production workloads.

Criteria for Acceptance

- The system will be accepted if Docker containers run successfully in both local and cloud environments, without dependency conflicts.
- The containers should handle moderate scaling with no significant performance bottlenecks.

Scalability and Deployment on Google Cloud Platform (GCP)

The system is deployed on GCP, utilizing GCP Data Warehouse for managing data at scale. The system should dynamically scale to handle air quality data streams, ensuring efficient storage and processing.

Criteria for Success

- The system should provide predictions with response times of under 1 second for most cases.

- GCP must scale dynamically to accommodate data growth without performance degradation.

Criteria for Acceptance

- The project will be accepted if GCP can handle a 50-75% increase in data load while maintaining response times below 5 seconds for predictions. pollution patterns, industry of the city, and even lifestyle of the citizens. Hence, by the passage of time, the predictions will be inaccurate. A solution to this issue is implementing Airflow to retrain the model with new data automatically. In other words, Airflow comes to help in automating the retraining process.

12. Timeline Planning

Phase 1: MLflow, Data Processing, and Modeling (Week 1)

- Day 1-2: MLflow Setup and Experiment Tracking

Install and configure MLflow for tracking experiments and models. Set up experiment logging and ensure reproducibility for all modeling activities.

- Day 3-5: Data Collection, Cleaning, and Preprocessing

Ingest, clean, and preprocess the air quality data, addressing any missing values or anomalies. Engineer relevant features and prepare the data for time series analysis.

- Day 6-7: Model Development

Build baseline and advanced models (e.g., ARIMA, LSTM, XGBoost) for air quality prediction. Evaluate models based on metrics such as MAE, RMSE, and MAPE, and track all results in MLflow.

Phase 2: Airflow and GCP Setup (Week 2-5)

- Week 2-3: Airflow Setup and Workflow Automation

Install and configure Airflow for automating data ingestion, preprocessing, and model training workflows. Design and implement Airflow DAGs to manage the entire workflow, including scheduled retraining.

- Week 4-5: GCP Setup and Integration

Set up GCP Data Warehouse for efficient storage and querying of air quality data. Deploy the models on GCP for predictions. Test GCP autoscaling to ensure the system can handle increasing data volumes.

Phase 3: Docker Setup (Week 6)

- Week 6: Docker Containerization

Containerize the full project, including models, workflows, and data pipelines, using Docker. Ensure the Docker containers are portable and scalable, capable of running in both local and cloud environments. Perform testing to ensure the containers function correctly across environments, including GCP.

Phase 4: Continuous GitHub Integration (Throughout the Project)

- Ongoing: Version Control and Collaboration

Use GitHub throughout the project to manage code, track changes, and collaborate efficiently. Ensure all components, including MLflow, Airflow, GCP, and Docker configurations, are versioned in GitHub. Implement CI/CD pipelines for automated testing and smooth deployment.

Post-Deployment: Monitoring and Maintenance (Ongoing)

- Ongoing: Monitoring and Retraining

Monitor system performance and model accuracy continuously, with automated alerts for any issues. Set up automated retraining workflows using Airflow and GCP to incorporate new data and maintain accuracy over time.

13. Cost and Resource Estimation

Service Display Name	Total Cost (USD)/month	Total Usage/month
Instances(Compute Engine)	\$88.85	9,520 units
Editions (BigQuery)	\$29.95	7 units
Cloud Storage	\$19.90	1,050 units
Artifact Registry	\$10.05	110 units
Cloud Logging(Cloud Operations)	\$3.00	350 units
Cloud Run Functions	\$2.77 each	153,700,900,000 units
Data Transfer (Networking)	\$0.10	6 units

Total estimated cost :- \$154/month