Project Statement: Air Quality Analysis and Prediction Model (Python)

1. Executive Summary

This document outlines the requirements and scope for a Python-based machine learning model designed to analyze historical air quality data, identify pollution trends, and provide short-term forecasts for key pollutants and the overall Air Quality Index (AQI). The primary goal is to provide data-driven insights to inform public health initiatives and environmental policy decisions.

2. Project Objectives

1. **Trend Analysis:** Analyze time-series data to detect significant historical trends, seasonality, and long-term changes in air quality metrics.
2. **Forecasting:** Develop an accurate prediction model for future concentrations of key pollutants (e.g., PM2.5, NO_2, O_3) over a 24- to 72-hour horizon.
3. **Feature Importance:** Identify and quantify the influence of various meteorological factors (temperature, humidity, wind speed) and temporal variables (time of day, day of week) on air pollution levels.
4. **Reporting:** Generate comprehensive reports and visualizations of current air quality status, prediction accuracy, and key feature impacts.

3. Scope and Key Features

Feature	Description	Output/Metric
Data Ingestion	Automated script to fetch and clean raw data from specified APIs or CSV sources.	Cleaned Pandas DataFrame.
Exploratory Data Analysis (EDA)	Visualization of pollutant distributions, correlation matrices, and time-series decomposition.	Statistical summaries, Histograms, Box plots.
Model Training	Implementation of machine learning or time-series algorithms for forecasting.	Trained model artifact (e.g., .pkl file).

Prediction & Evaluation	Generation of forecasts and evaluation of model performance against ground truth data.	RMSE, MAE, R-squared, Prediction vs. Actual charts.
Feature Engineering	Creation of lag features, moving averages, and temporal identifiers (e.g., hour, month).	Engineered features for model input.

4. Methodology and Technical Stack

The model employs a multi-stage process leveraging core Python libraries for data science and machine learning.

4.1. Data Preprocessing and Analysis

- **Language:** Python 3.9+
- **Core Libraries:**
  - **Pandas:** Handling and manipulation of time-series data structures.
  - **NumPy:** Numerical operations and array handling.
  - **Matplotlib / Seaborn:** Generation of static visualizations (trends, correlations).

4.2. Modeling Approaches

The system will evaluate and utilize the most effective algorithm based on data characteristics.

1. **Time Series Analysis (Baseline):**
   - **Techniques:** ARIMA/SARIMA models for seasonal and non-seasonal time-series forecasting.
   - **Library:** statsmodels.
2. **Regression-Based Forecasting:**
   - **Techniques:** Gradient Boosting Regressor (e.g., XGBoost, LightGBM) or Random Forest for improved non-linear prediction accuracy by integrating meteorological features.
   - **Library:** scikit-learn / XGBoost.

4.3. Data Sources

The model requires historical data points with high granularity (hourly or daily), including:

Data Type	Example Fields	Source Requirement

Pollutants	PM2.5, PM10, O_3, NO_2, SO_2, CO, AQI	Historical hourly readings.
Meteorological	Temperature, Humidity, Wind Speed/Direction, Pressure, Rainfall	Synchronized with pollutant readings.
Temporal	Timestamp, Hour of Day, Day of Week, Is_Weekend	Generated during feature engineering.

5. Deliverables

- **model.py:** Contains the primary functions for data processing, model training, prediction, and evaluation.
- **analysis_report.ipynb (Jupyter Notebook):** Detailed EDA, model selection process, hyperparameter tuning, and final performance metrics.
- **requirements.txt:** A list of all necessary Python dependencies and their versions.
- **trained_model.pkl:** The serialized, trained model object ready for deployment or batch prediction.

6. Future Enhancements

1. **Real-Time Dashboard:** Integrate with a lightweight web framework (e.g., Streamlit or Flask) to visualize live data and rolling 72-hour forecasts.
2. **Ensemble Modeling:** Combine forecasts from multiple models (e.g., ARIMA + XGBoost) to further reduce prediction error.
3. **Anomaly Detection:** Implement isolation forest or similar techniques to flag unusual pollutant spikes for root cause analysis.