**Project Title:** Predicting Air Quality for Sustainable Urban Living

**Team Members**

1. Dawit Teklebrehan
2. Shimels Tilahun
3. Robel Ermiyas
4. Robel Roba
5. Tewodros Gebretsadk
6. Negassa Retta

**Date of Submisson: April 29, 2025**

**Data Preparation, Feature Engineering, and Model Exploration**

**1. Data Preparation**

**1.1 Data Collection**

The data for this project will be sourced from multiple reliable open data platforms:

* **OpenAQ API**: Provides hourly air quality measurements for pollutants like PM2.5, PM10, CO, SO2, NO2, and O3.
* **World Air Quality Index (WAQI)**: Offers historical air quality index data and real-time readings.
* **NASA Earth Observation (NEO)**: Provides satellite data on temperature, vegetation, and atmospheric conditions.
* **Local Weather APIs**: Includes temperature, humidity, wind speed, and precipitation data.

**1.2 Data Cleaning and Integration**

* Remove duplicate and irrelevant rows.
* Handle missing values using mean imputation or forward-fill techniques.
* Convert all timestamps to a consistent format (UTC).
* Merge air quality and weather datasets based on time and location.
* Filter out extreme outliers using IQR or z-score methods.

import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

# Load dataset

df = pd.read\_csv("air\_quality\_data.csv")

# Handle missing values

imputer = SimpleImputer(strategy='mean')

df[['PM2\_5', 'PM10', 'NO2', 'CO']] = imputer.fit\_transform(df[['PM2\_5','PM10', 'NO2', 'CO']])

# Convert timestamp

df['datetime'] = pd.to\_datetime(df['datetime'])

df.set\_index('datetime', inplace=True)

# Drop duplicates

df.drop\_duplicates(inplace=True

**1.3 Data Formatting**

* Ensure consistent column naming conventions (e.g., PM2.5 → PM2\_5).
* Normalize or scale features as needed using MinMaxScaler or StandardScaler.
* Convert categorical values (e.g., weather condition) using one-hot encoding if required.

# Normalize numeric features

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(df[['PM2\_5', 'PM10', 'NO2', 'CO', 'temperature', 'humidity']])

scaled\_df = pd.DataFrame(scaled\_features, columns=['PM2\_5', 'PM10', 'NO2', 'CO', 'temperature', 'humidity'])

**2. Feature Engineering**

**2.1 Temporal Features**

* Extract **hour**, **day of week**, **month**, and **season** from timestamps.
* Encode cyclical time features using sine and cosine transformations.

import numpy as np

df['hour'] = df.index.hour

df['dayofweek'] = df.index.dayofweek

df['month'] = df.index.month

# Encode cyclical time

df['hour\_sin'] = np.sin(2 \* np.pi \* df['hour'] / 24)

df['hour\_cos'] = np.cos(2 \* np.pi \* df['hour'] / 24)

**2.2 Meteorological Features**

* Aggregate data like **temperature, humidity, wind speed**, and **precipitation**.
* Calculate **moving averages** (3-hour, 6-hour, and 24-hour) for pollutant levels.

**2.3 Spatial Features (Optional)**

* If geolocation is available, derive features like **distance from major roads or factories**.
* Include latitude and longitude if using spatial models.

**2.4 Lag Features**

* Create lagged versions of air pollutant concentrations to capture temporal dependencies (e.g., PM2\_5\_lag1, lag2, ...).

# Create lag features for PM2.5

for i in range(1, 4):

df[f'PM2\_5\_lag{i}'] = df['PM2\_5'].shift(i)

**2.5 Interaction Features**

* Combine meteorological and time features (e.g., humidity × hour) to capture conditional relationships.

**3. Model Exploration**

**3.1 Baseline Models**

To set benchmarks and compare performance, the following models will be used:

* **Linear Regression**: For baseline comparison.
* **Decision Tree Regressor**: Easy to interpret, good for initial testing.

**3.2 Advanced Models**

* **Random Forest Regressor**: Robust, reduces variance.
* **Gradient Boosting Regressor (XGBoost or LightGBM)**: Handles non-linearity and performs well with structured data.
* **Support Vector Regressor (SVR)**: Effective in high-dimensional space, sensitive to feature scaling.

**3.3 Model Evaluation Strategy**

* Split dataset into **training (70%)**, **validation (15%)**, and **test (15%)** sets.
* Use **k-fold cross-validation** for model tuning and validation.
* Performance metrics:
  + **Mean Absolute Error (MAE)**
  + **Root Mean Squared Error (RMSE)**
  + **R-squared (R²)**

**3.4 Model Interpretability**

* Use **SHAP (SHapley Additive exPlanations)** to explain feature importance.
* Visualize feature contributions to individual predictions.