AeroVision

AI-Powered Air Quality Index Prediction & Monitoring System

Code Overview & Learning Guide

File: air_pol.py

Dataset: Kaggle - Air Quality Data in India

AeroVision - Air Quality Code Overview

What This Project Does

This is an Air Quality Index (AQI) Prediction System. It takes pollution data from Indian cities and trains Machine Learning models to predict the AQI value. The pipeline follows the classic ML workflow: Load -> Explore -> Clean -> Engineer Features -> Train -> Evaluate -> Interpret.

The ML Pipeline (Step by Step)

Step 1: Data Loading (line 106)

```
df = pd.read_csv('/city_aqi_day.csv', parse_dates=['Date'])
```

Loads a CSV dataset containing daily air quality readings from various Indian cities. The parse_dates parameter converts the 'Date' column into proper datetime objects instead of plain strings, enabling time-based operations later.

Step 2: Exploratory Data Analysis - EDA (lines 113-190)

Multiple visualizations are created to understand the data:

- Missing value check - calculates what % of each column is null
- Pie chart & bar chart - shows distribution of AQI categories (Good, Moderate, Poor, Severe)
- Histplot by city - shows which cities have which AQI categories
- Correlation heatmap - shows how strongly each pollutant relates to others
- Box plots - visualizes the spread and outliers in each pollutant
- Pair plot - scatterplots between every pair of features to spot relationships

Step 3: Handling Missing Values (lines 155-161)

```
# Numbers -> fill with median (robust to outliers)
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

# Categories -> fill with most frequent value (mode)
df[col] = df[col].fillna(df[col].mode()[0])
```

Median is used for numeric columns because it is robust to outliers (unlike mean). Mode (most frequent value) is used for categorical/text columns.

Step 4: Outlier Removal (lines 199-207)

Uses the IQR (Interquartile Range) method:

```
IQR = Q3 - Q1
Lower Bound = Q1 - 1.5 * IQR
Upper Bound = Q3 + 1.5 * IQR
Keep only rows where value is within bounds
```

Any data point outside this range is considered an outlier and removed. This is the standard statistical approach for outlier detection. The 1.5 multiplier is a widely accepted convention in statistics.

Step 5: Feature Engineering (lines 290-333)

Three key transformations are applied:

- Label Encoding - Converts categorical text (city names, AQI buckets) into numbers so ML models can process them.

Example: 'Delhi' -> 0, 'Mumbai' -> 1

- Rolling Average - Computes a 7-day moving average of AQI to smooth out daily noise. Note: it is created then immediately dropped, so it doesn't contribute to the final model.
- StandardScaler - Normalizes all numeric features to have mean=0 and std=1. This ensures models treat all features equally regardless of their original scale.

Step 6: Train-Test Split (lines 381-386)

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

80% of the data is used for training and 20% for testing. random_state=42 ensures the split is reproducible every time you run the code. X contains the features (pollutants), y contains the target (AQI value).

Step 7: Model Training (lines 388-396)

Two models are trained:

Model	Description
Random Forest (200 trees)	Ensemble of decision trees that vote together.
XGBoost (200 trees, lr=0.1)	Gradient-boosted trees; each corrects prior errors.

Random Forest is robust and hard to overfit. XGBoost builds trees sequentially, where each new tree corrects the mistakes of previous trees, often achieving higher accuracy.

Step 8: Model Evaluation (lines 399-421)

Two metrics are used to compare models:

- R2 Score - How much variance the model explains. 1.0 = perfect, 0 = useless.
- RMSE (Root Mean Squared Error) - Average prediction error in original units. Lower = better.

Results are visualized as bar charts for easy comparison.

Step 9: Model Interpretation with SHAP (lines 426-448)

SHAP (SHapley Additive exPlanations) answers the question: 'Which features contributed most to each prediction?'

- Bar plots - Show global feature importance across all predictions
- KDE plots - Show the distribution of each feature's impact across all test samples

Key Findings

- PM2.5 is the strongest predictor of AQI
- CO, SO2, PM10, NO2 are key secondary drivers
- O3, Benzene, Toluene contribute less overall
- Both Random Forest and XGBoost agree on feature importance, confirming consistency
- XGBoost shows sharper, more confident SHAP distributions compared to Random Forest

AeroVision - Air Quality Code Overview

Dataset Features Reference

- City - Where the pollution data was recorded
- Date - When the measurement was taken
- PM2.5 & PM10 - Particulate matter (tiny vs bigger dust particles that harm lungs)
- NO, NO2, NOx - Gases from vehicles/industries, cause smog & breathing issues
- NH3 (Ammonia) - From fertilizers/livestock, forms harmful particles
- CO (Carbon Monoxide) - From incomplete fuel burning, toxic at high levels
- SO2 (Sulfur Dioxide) - From coal/oil burning, causes acid rain & breathing problems
- O3 (Ozone) - Good in upper atmosphere, harmful near ground (smog)
- Benzene, Toluene, Xylene - Chemicals from exhaust/paints, harmful to health
- AQI - Air Quality Index, a score from 0-500 showing overall air quality
- AQI_Bucket - Category: Good, Satisfactory, Moderate, Poor, Very Poor, Severe

Notes for Learners

- The code has some redundancy - encoding and scaling is done twice (lines 291-308 and lines 357-378). This is common when converting a Colab notebook to a .py file.
- StandardScaler is imported at line 355 but used earlier at line 307. This would cause a NameError if run top-to-bottom as a script.
- The rolling average is created then immediately dropped (lines 301-304), so it doesn't actually contribute to the model.
- The overall pipeline (Load -> EDA -> Clean -> Feature Engineer -> Train -> Evaluate -> Interpret) is the industry-standard approach for any ML project.