

**TITLE:** AirSense - AI-Powered Air Quality Monitoring & Forecasting

**Concept Note and Implementation Plan Submission**

**TITLE:** AirSense - AI-Powered Air Quality Monitoring & Forecasting

**Model Exploration Submission**

**Team Members**

1. Dawit Getachew

2. Gelasa Jarso

3. Helen Zelalem

4. Mukitar Seid

5. Tekleeyesus Munye

**Multi-Station Air Quality Analysis in Beijing** Load and Combine All Station Data

1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import seaborn as sns 5

6

1. # List all station files
2. station\_files = [

9

10

11

12

13

14

15

16

17

18

19

20

21 ]

22

"/content/PRSA\_Data\_Aotizhongxin\_20130301-20170228.csv", "/content/PRSA\_Data\_Changping\_20130301-20170228.csv", "/content/PRSA\_Data\_Dingling\_20130301-20170228.csv",

"/content/PRSA\_Data\_Dongsi\_20130301-20170228.csv", "/content/PRSA\_Data\_Guanyuan\_20130301-20170228.csv", "/content/PRSA\_Data\_Gucheng\_20130301-20170228.csv", "/content/PRSA\_Data\_Huairou\_20130301-20170228.csv",

"/content/PRSA\_Data\_Nongzhanguan\_20130301-20170228.csv", "/content/PRSA\_Data\_Shunyi\_20130301-20170228.csv",

"/content/PRSA\_Data\_Tiantan\_20130301-20170228.csv", "/content/PRSA\_Data\_Wanliu\_20130301-20170228.csv",

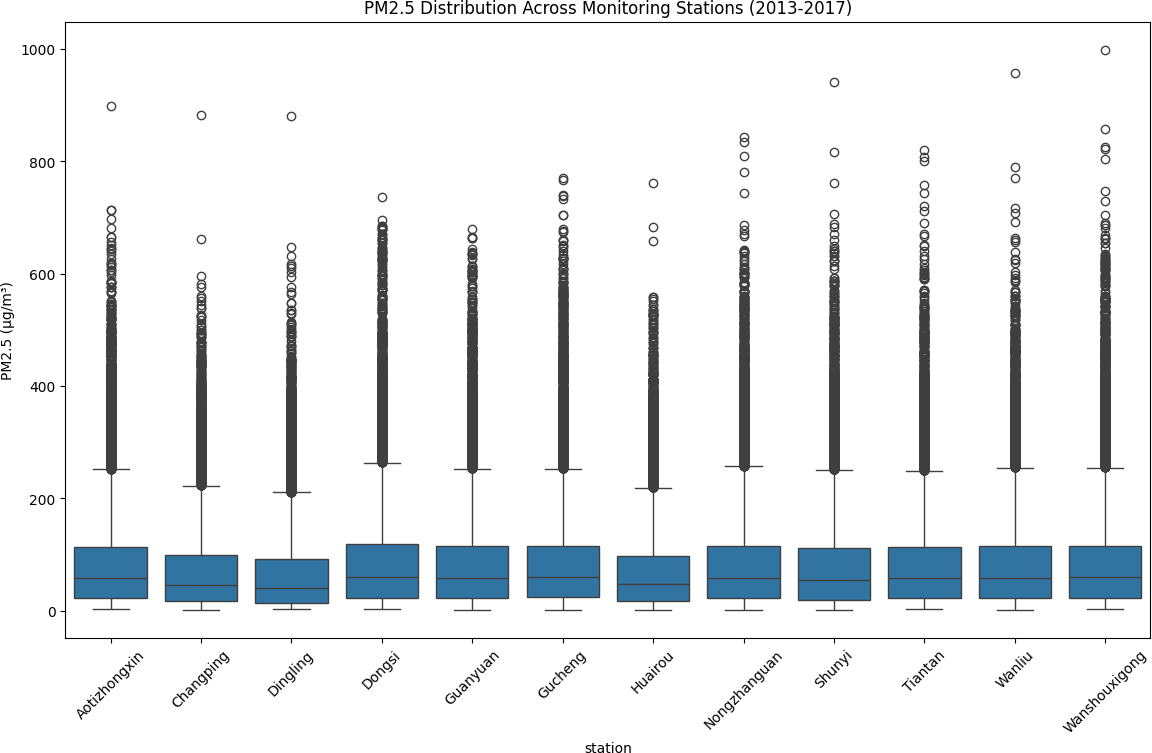
"/content/PRSA\_Data\_Wanshouxigong\_20130301-20170228.csv"

1. # Load all stations into a dictionary
2. stations = {}
3. for file in station\_files:
4. station\_name = file.split('\_')[2] # Extracts 'Aotizhongxin', 'Changping', etc.
5. stations[station\_name] = pd.read\_csv(file)
6. stations[station\_name]['station'] = station\_name # Add station identifier 29
7. # Combine into single DataFrame
8. all\_data = pd.concat(stations.values(), ignore\_index=True) 32
9. # Convert to datetime
10. all\_data['datetime'] = pd.to\_datetime(all\_data[['year', 'month', 'day', 'hour']])
11. all\_data = all\_data.set\_index('datetime')
12. **Station Comparison Analysis** Basic Statistics by Station
    1. # Pollution level comparison
    2. pollutants = ['PM2.5', 'PM10', 'SO2', 'NO2', 'CO', 'O3']
    3. station\_stats = all\_data.groupby('station')[pollutants].agg(['mean', 'max', 'std']) 4
13. # Display sorted by PM2.5 mean
14. station\_stats['PM2.5'].sort\_values('mean', ascending=False)

|  |  |  |  |
| --- | --- | --- | --- |
| **station** | **mean** | **max** | **std** |
| **Dongsi** | 86.194297 | 737.0 | 86.575127 |
| **Wanshouxigong** | 85.024136 | 999.0 | 85.975981 |
| **Nongzhanguan** | 84.838483 | 844.0 | 86.225344 |
| **Gucheng** | 83.852089 | 770.0 | 82.796445 |
| **Wanliu** | 83.374716 | 957.0 | 81.905568 |
| **Guanyuan** | 82.933372 | 680.0 | 80.933497 |
| **Aotizhongxin** | 82.773611 | 898.0 | 82.135694 |
| **Tiantan** | 82.164911 | 821.0 | 80.921384 |
| **Shunyi** | 79.491602 | 941.0 | 81.231739 |
| **Changping** | 71.099743 | 882.0 | 72.326926 |
| **Huairou** | 69.626367 | 762.0 | 71.224916 |
| **Dingling** | 65.989497 | 881.0 | 72.267723 |
| WARNING: Runtime | no longer | has a | reference to this dataframe, please re-run this cell and try again. |
| C |  |  | C |

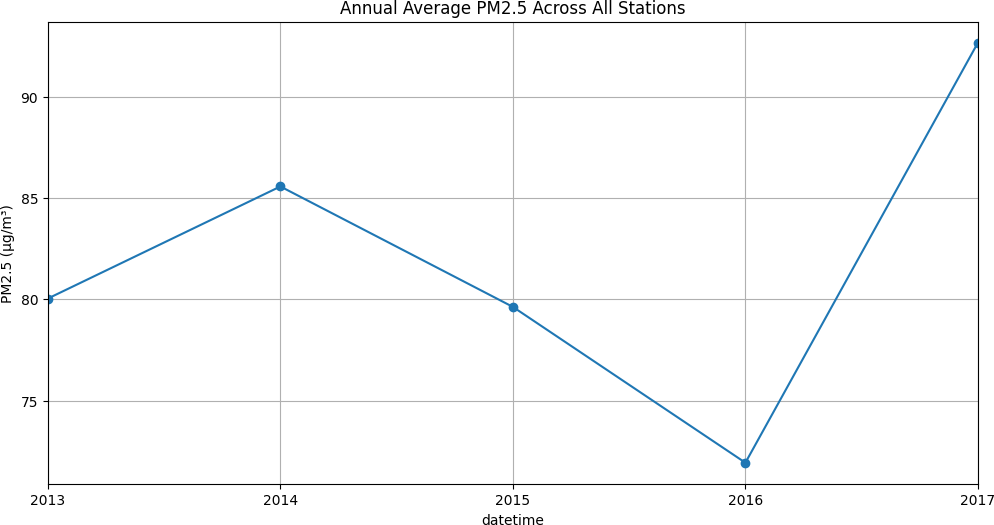
Visual Comparision

1. # Convert 'datetime' column to a regular column before grouping
2. all\_data = all\_data.reset\_index() 3
3. # Now create the boxplot
4. plt.figure(figsize=(14, 8))
5. sns.boxplot(x='station', y='PM2.5', data=all\_data)
6. plt.xticks(rotation=45)
7. plt.title('PM2.5 Distribution Across Monitoring Stations (2013-2017)')
8. plt.ylabel('PM2.5 (µg/m³)')
9. plt.show()

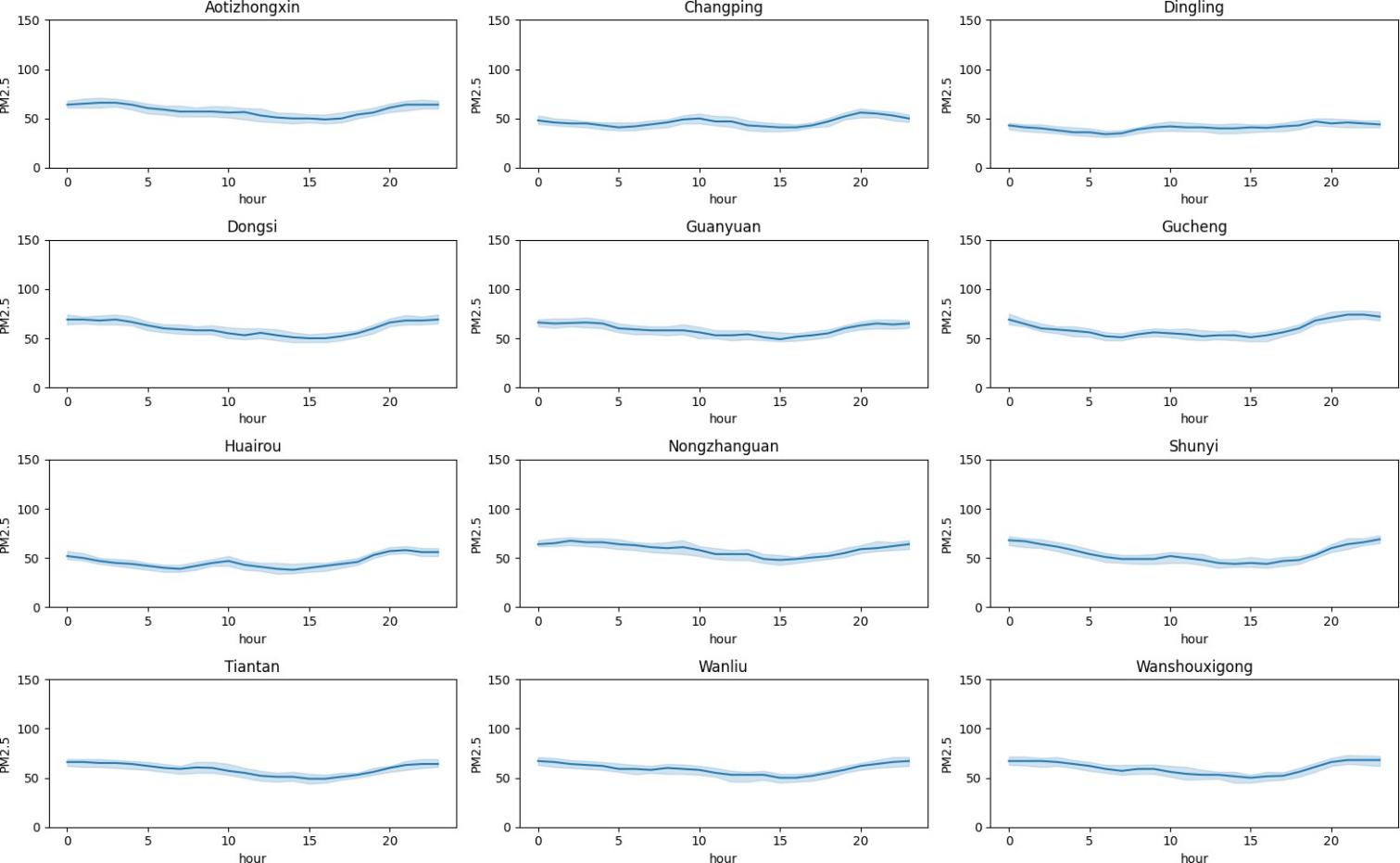
1. **Temporal Patterns** Annual Trends
   1. # Ensure datetime is properly set as index
   2. all\_data['datetime'] = pd.to\_datetime(all\_data[['year', 'month', 'day', 'hour']])
   3. all\_data = all\_data.set\_index('datetime') 4
2. # Now resample will work
3. annual\_trend = all\_data.resample('Y')[pollutants].mean() 7
4. # Plotting
5. plt.figure(figsize=(12, 6))
6. annual\_trend['PM2.5'].plot(marker='o')
7. plt.title('Annual Average PM2.5 Across All Stations')
8. plt.ylabel('PM2.5 (µg/m³)')
9. plt.grid()
10. plt.show()

<ipython-input-12-86a9ab8e7492>:6: FutureWarning: 'Y' is deprecated and will be removed in a future version, pleas annual\_trend = all\_data.resample('Y')[pollutants].mean()



# Diurnal Patterns by Station

1. all\_data['hour'] = all\_data.index.hour 2
2. plt.figure(figsize=(16, 10))
3. for i, station in enumerate(all\_data['station'].unique(), 1):
4. plt.subplot(4, 3, i)
5. station\_data = all\_data[all\_data['station'] == station]
6. sns.lineplot(x='hour', y='PM2.5', data=station\_data, estimator='median')
7. plt.title(station)
8. plt.ylim(0, 150) # Consistent scale
9. plt.tight\_layout()
10. plt.show()

# Geospatial Analysis

* 1. # !pip install geopandas
  2. import geopandas as gpd 3

1. # Station locations (approximate coordinates)
2. station\_locs = {

6

7

8

9

10

11

12

13

14

15

16

17

18 }

19

'Aotizhongxin': (116.407, 39.904),

'Changping': (116.231, 40.221),

'Dingling': (116.222, 40.292),

'Dongsi': (116.423, 39.929),

'Guanyuan': (116.339, 39.929),

'Gucheng': (116.184, 39.914),

'Huairou': (116.632, 40.316),

'Nongzhanguan': (116.461, 39.937),

'Shunyi': (116.654, 40.128),

'Tiantan': (116.407, 39.882),

'Wanliu': (116.287, 39.987),

'Wanshouxigong': (116.354, 39.878)

1. # Create GeoDataFrame
2. gdf = gpd.GeoDataFrame(
3. station\_stats['PM2.5']['mean'].reset\_index(),
4. geometry=gpd.points\_from\_xy(

|  |  |  |
| --- | --- | --- |
| 24 |  | [station\_locs[st][0] for st in station\_stats.index], |
| 25 |  | [station\_locs[st][1] for st in station\_stats.index] |
| 26 | ) |  |
| 27 ) |  |  |
| 28 |  |  |

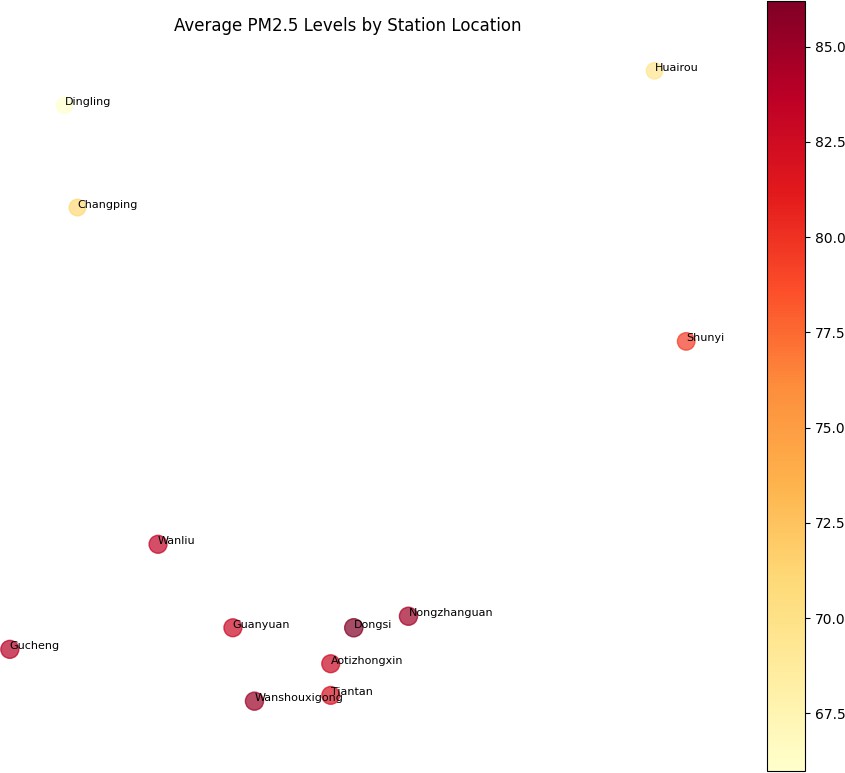
1. # Plot on Beijing map
2. fig, ax = plt.subplots(figsize=(12, 10))
3. gdf.plot(column='mean', cmap='YlOrRd', legend=True, ax=ax,
4. markersize=gdf['mean']\*2, alpha=0.7)

33

1. # Add labels
2. for x, y, label in zip(gdf.geometry.x, gdf.geometry.y, gdf['station']):
3. ax.text(x, y, label, fontsize=8)

37

1. plt.title('Average PM2.5 Levels by Station Location')
2. plt.axis('off')
3. plt.show()



1. **Correlation Analysis** Weather vs Pollution
   1. import pandas as pd
   2. import numpy as np
   3. import matplotlib.pyplot as plt
   4. import seaborn as sns 5
2. # =============================================
3. # 1. DATA LOADING
4. # =============================================

9

1. # List all station files
2. station\_files = [
3. "/content/PRSA\_Data\_Aotizhongxin\_20130301-20170228.csv",
4. "/content/PRSA\_Data\_Changping\_20130301-20170228.csv",
5. "/content/PRSA\_Data\_Dingling\_20130301-20170228.csv",
6. "/content/PRSA\_Data\_Dongsi\_20130301-20170228.csv",
7. "/content/PRSA\_Data\_Guanyuan\_20130301-20170228.csv",
8. "/content/PRSA\_Data\_Gucheng\_20130301-20170228.csv",
9. "/content/PRSA\_Data\_Huairou\_20130301-20170228.csv",

19

20

21

22

23

24 ]

25

"/content/PRSA\_Data\_Nongzhanguan\_20130301-20170228.csv", "/content/PRSA\_Data\_Shunyi\_20130301-20170228.csv",

"/content/PRSA\_Data\_Tiantan\_20130301-20170228.csv", "/content/PRSA\_Data\_Wanliu\_20130301-20170228.csv",

"/content/PRSA\_Data\_Wanshouxigong\_20130301-20170228.csv"

1. # Load and combine all data
2. dfs = []
3. for file in station\_files:
4. station\_name = file.split('\_')[2] # Extract station name
5. df = pd.read\_csv(file)
6. df['station'] = station\_name # Add station identifier
7. dfs.append(df)

33

34 all\_data = pd.concat(dfs, ignore\_index=True) 35

36 # Convert to datetime

37 all\_data['datetime'] = pd.to\_datetime(all\_data[['year', 'month', 'day', 'hour']])

38 all\_data = all\_data.set\_index('datetime') 39

40 # =============================================

41 # 2. DATA PREPARATION

42 # =============================================

43

44 # Define expected columns with possible alternatives

45 column\_mapping = {

46

47

48

49

50

51

52

53

54

55

56

57 }

58

'PM2.5': ['PM2.5', 'pm2.5'],

'PM10': ['PM10', 'pm10'],

'SO2': ['SO2', 'so2'],

'NO2': ['NO2', 'no2'],

'CO': ['CO', 'co'],

'O3': ['O3', 'o3'],

'TEMP': ['TEMP', 'temp', 'temperature'],

'PRES': ['PRES', 'pres', 'pressure'],

'DEWP': ['DEWP', 'dewp', 'dew\_point'],

'RAIN': ['RAIN', 'rain', 'precipitation'],

'WSPD': ['WSPD', 'wspd', 'wind\_speed', 'ws']

1. # Standardize column names
2. for standard\_name, alternatives in column\_mapping.items():
3. for alt in alternatives:
4. if alt in all\_data.columns:
5. if standard\_name != alt:
6. all\_data[standard\_name] = all\_data[alt]
7. break

66

1. # Select features for analysis
2. pollutants = ['PM2.5', 'PM10', 'SO2', 'NO2', 'CO', 'O3']
3. weather\_features = ['TEMP', 'PRES', 'DEWP', 'RAIN', 'WSPD']

70

1. # Filter to available features
2. available\_pollutants = [p for p in pollutants if p in all\_data.columns]
3. available\_weather = [w for w in weather\_features if w in all\_data.columns] 74
4. print(f"Available pollutants: {available\_pollutants}")
5. print(f"Available weather features: {available\_weather}") 77
6. # =============================================
7. # 3. CORRELATION ANALYSIS
8. # =============================================

81

1. # Prepare analysis data
2. analysis\_data = all\_data[available\_pollutants + available\_weather].dropna() 84
3. # Calculate correlation matrix
4. corr\_matrix = analysis\_data.corr() 87
5. # =============================================
6. # 4. VISUALIZATION
7. # =============================================

91

92 plt.figure(figsize=(12, 10))

93 mask = np.triu(np.ones\_like(corr\_matrix, dtype=bool)) 94

95 sns.heatmap(

96

97

98

99

100

101

102

103

104

105

106

107 )

108

corr\_matrix, annot=True,

cmap='coolwarm', center=0,

fmt='.2f', mask=mask, vmin=-1, vmax=1,

square=True,

linewidths=0.5,

cbar\_kws={"shrink": 0.8}

109 plt.title("Pollution-Weather Correlations Across Beijing Stations\n(2013-2017)", pad=20, fontsize=14)

110 plt.xticks(rotation=45, ha='right')

111 plt.yticks(rotation=0)

112 plt.tight\_layout()

113 plt.show()

114

115 # =============================================

116 # 5. TOP CORRELATIONS

117 # =============================================

118

119 print("\nTop 10 Strongest Correlations:")

120 corr\_pairs = corr\_matrix.unstack()

121 # Filter out self-correlations and duplicates

122 unique\_corr\_pairs = corr\_pairs[corr\_pairs.index.get\_level\_values(0) != corr\_pairs.index.get\_level\_values(1)]

123 top\_correlations = unique\_corr\_pairs.sort\_values(key=abs, ascending=False).head(10)

124 print(top\_correlations.to\_string(float\_format="%.2f"))

125

126 # =============================================

127 # 6. DIAGNOSTICS

128 # =============================================

129

130 print("\nAnalysis Summary:")

131 print(f"Time period: {all\_data.index.min().date()} to {all\_data.index.max().date()}")

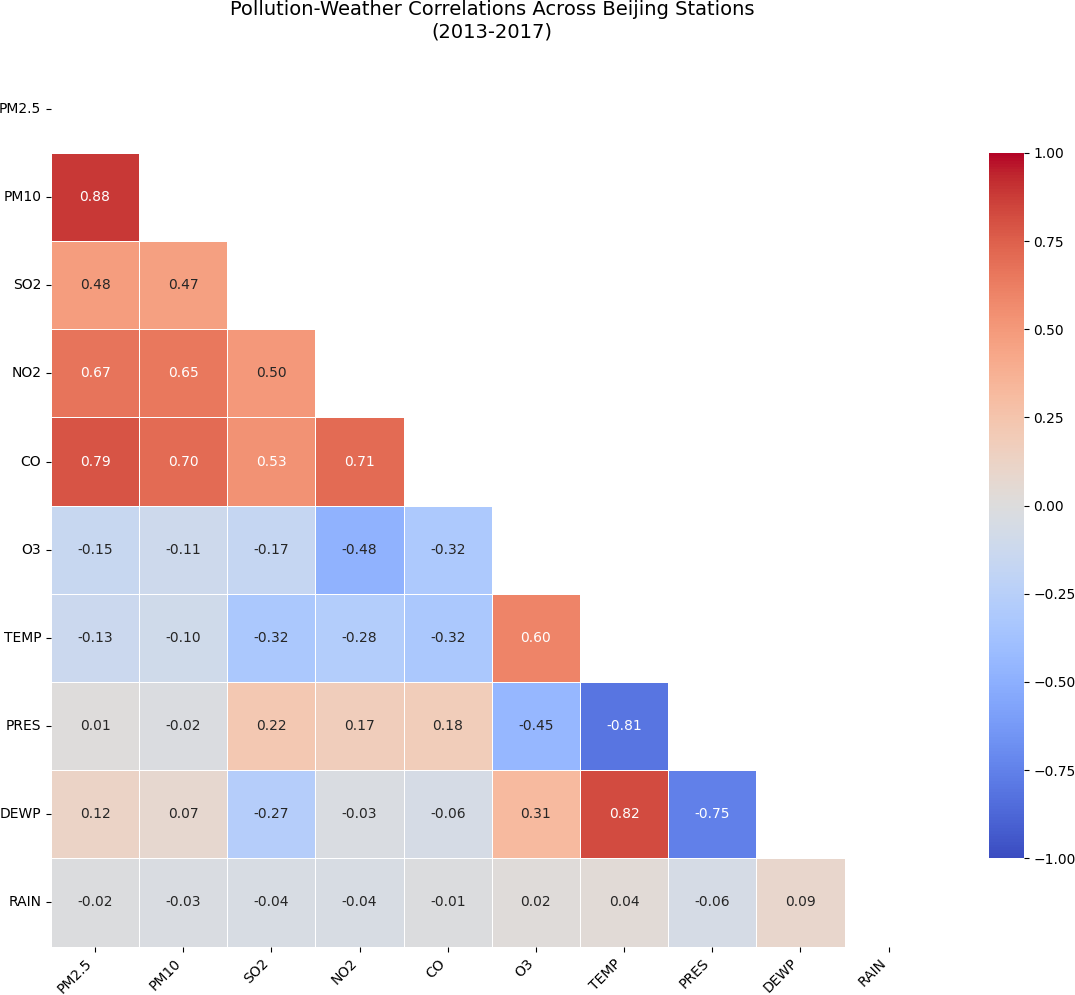
132 print(f"Stations included: {all\_data['station'].nunique()}")

133 print(f"Final sample size: {len(analysis\_data):,} records")

134 print("\nDescriptive statistics:")

135 print(analysis\_data.describe().to\_string(float\_format="%.1f"))

 Available pollutants: ['PM2.5', 'PM10', 'SO2', 'NO2', 'CO', 'O3'] Available weather features: ['TEMP', 'PRES', 'DEWP', 'RAIN']



Top 10 Strongest Correlations:

PM2.5 PM10 0.88

PM10 PM2.5 0.88

TEMP DEWP 0.82

DEWP TEMP 0.82

PRES TEMP -0.81

TEMP PRES -0.81

PM2.5 CO 0.79

CO PM2.5 0.79

DEWP PRES -0.75

PRES DEWP -0.75

Analysis Summary:

Time period: 2013-03-01 to 2017-02-28

Stations included: 12

Final sample size: 383,586 records Descriptive statistics:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PM2.5 | PM10 | SO2 | NO2 | CO | O3 | TEMP | PRES | DEWP | RAIN |

count 383586.0 383586.0 383586.0 383586.0 383586.0 383586.0 383586.0 383586.0 383586.0 383586.0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| mean | 79.5 | 104.6 | 15.6 | 50.6 | 1231.9 | 57.2 | 13.5 | 1010.8 | 2.4 | 0.1 |
| std | 80.2 | 91.4 | 21.3 | 35.1 | 1158.9 | 56.7 | 11.4 | 10.5 | 13.8 | 0.8 |
| min | 2.0 | 2.0 | 0.3 | 2.0 | 100.0 | 0.2 | -19.9 | 982.4 | -43.4 | 0.0 |
| 25% | 20.0 | 36.0 | 2.0 | 23.0 | 500.0 | 10.0 | 3.1 | 1002.4 | -9.0 | 0.0 |
| 50% | 55.0 | 82.0 | 7.0 | 43.0 | 900.0 | 45.0 | 14.4 | 1010.4 | 2.9 | 0.0 |
| 75% | 111.0 | 145.0 | 19.0 | 71.0 | 1500.0 | 82.0 | 23.2 | 1019.0 | 15.1 | 0.0 |
| max | 844.0 | 999.0 | 500.0 | 290.0 | 10000.0 | 1071.0 | 41.6 | 1042.8 | 29.1 | 72.5 |

# Feature Engineering

* 1. Temporal Features
     1. import numpy as np
     2. from sklearn.preprocessing import StandardScaler, OneHotEncoder 3

1. # Create temporal features
2. all\_data['hour\_sin'] = np.sin(2 \* np.pi \* all\_data.index.hour/24)
3. all\_data['hour\_cos'] = np.cos(2 \* np.pi \* all\_data.index.hour/24)
4. all\_data['dayofyear\_sin'] = np.sin(2 \* np.pi \* all\_data.index.dayofyear/365)
5. all\_data['dayofyear\_cos'] = np.cos(2 \* np.pi \* all\_data.index.dayofyear/365) 9
6. # Weather interaction features
7. all\_data['temp\_dewp\_diff'] = all\_data['TEMP'] - all\_data['DEWP'] # Humidity indicator
8. all\_data['temp\_pres\_ratio'] = all\_data['TEMP'] / all\_data['PRES'] # Atmospheric condition 13
9. # Pollution rolling features
10. for window in [24, 72]: # 24h and 72h rolling windows
11. all\_data[f'PM2.5\_rolling\_{window}h\_mean'] = all\_data.groupby('station')['PM2.5'].transform(
12. lambda x: x.rolling(f'{window}h', min\_periods=1).mean()

18 )

1. all\_data[f'NO2\_rolling\_{window}h\_max'] = all\_data.groupby('station')['NO2'].transform(
2. lambda x: x.rolling(f'{window}h', min\_periods=1).max()

21 )

22

1. # Wind vector decomposition
2. if 'WSPD' in all\_data.columns and 'wd' in all\_data.columns:
3. all\_data['wind\_x'] = all\_data['WSPD'] \* np.cos(np.radians(all\_data['wd']))
4. all\_data['wind\_y'] = all\_data['WSPD'] \* np.sin(np.radians(all\_data['wd'])) 27
5. # Pollution ratios
6. all\_data['PM\_ratio'] = all\_data['PM2.5'] / all\_data['PM10'] # Fine particle proportion
7. all\_data['SO2\_NO2\_ratio'] = all\_data['SO2'] / all\_data['NO2'] # Industrial vs traffic pollution 31
8. # Binary features
9. all\_data['is\_weekend'] = (all\_data.index.dayofweek >= 5).astype(int)
10. all\_data['rush\_hour'] = ((all\_data.index.hour >= 7) & (all\_data.index.hour <= 9)) | \
11. ((all\_data.index.hour >= 17) & (all\_data.index.hour <= 19))

Rationale for Each Feature:

# Cyclical Time Features:

hour\_sin/cos and dayofyear\_sin/cos capture periodic patterns while maintaining continuity (e.g., 23:59 and 00:01 are close) **Weather Interactions:**

temp\_dewp\_diff indicates humidity levels (smaller difference = higher humidity) temp\_pres\_ratio helps identify temperature inversion conditions

# Rolling Aggregates:

24h/72h rolling means/maxs account for pollution persistence effects Calculated per-station to avoid mixing different locations

# Wind Components:

Decompose wind into x/y vectors for better directional analysis **Pollution Ratios:**

PM\_ratio shows fine vs coarse particle dominance

SO2\_NO2\_ratio indicates industrial (SO2) vs vehicular (NO2) sources **Temporal Indicators:**

is\_weekend and rush\_hour flag periods with expected emission pattern changes

# Data Transformation

1. import numpy as np
2. import pandas as pd
3. from sklearn.pipeline import Pipeline
4. from sklearn.compose import ColumnTransformer
5. from sklearn.impute import SimpleImputer
6. from sklearn.preprocessing import StandardScaler, OneHotEncoder 7
7. # 1. First ensure all\_data exists and has our engineered features
8. # (Run the feature engineering code from previous section first) 10
9. # 2. Verify which features actually exist in our data
10. available\_numeric = [f for f in ['TEMP', 'PRES', 'DEWP', 'RAIN', 'WSPD',
11. 'temp\_dewp\_diff', 'temp\_pres\_ratio',
12. 'PM2.5\_rolling\_24h\_mean', 'NO2\_rolling\_24h\_max',
13. 'wind\_x', 'wind\_y', 'PM\_ratio', 'SO2\_NO2\_ratio']
14. if f in all\_data.columns] 17
15. available\_categorical = [f for f in ['station', 'is\_weekend', 'rush\_hour']
16. if f in all\_data.columns] 20
17. print("Available numeric features:", available\_numeric)
18. print("Available categorical features:", available\_categorical) 23
19. # 3. Create transformers with error handling
20. try:
21. numeric\_transformer = Pipeline(steps=[
22. ('imputer', SimpleImputer(strategy='median')),
23. ('scaler', StandardScaler())

29 ])

30

1. categorical\_transformer = Pipeline(steps=[
2. ('imputer', SimpleImputer(strategy='constant', fill\_value='missing')),
3. ('onehot', OneHotEncoder(handle\_unknown='ignore', sparse\_output=False))

34 ])

35

1. # 4. Combine transformers
2. preprocessor = ColumnTransformer(
3. transformers=[
4. ('num', numeric\_transformer, available\_numeric),
5. ('cat', categorical\_transformer, available\_categorical)

41 ])

42

43 # 5. Apply transformations

44 X\_transformed = preprocessor.fit\_transform(all\_data) 45

1. # 6. Get feature names
2. numeric\_names = available\_numeric 48
3. # Handle case where categorical features might be empty
4. if available\_categorical:
5. categorical\_names = preprocessor.named\_transformers\_['cat'].named\_steps['onehot'].get\_feature\_names\_out(av
6. all\_feature\_names = np.concatenate([numeric\_names, categorical\_names])
7. else:
8. all\_feature\_names = numeric\_names 55
9. print(f"\nFinal feature matrix shape: {X\_transformed.shape}")
10. print("\nFirst 5 transformed samples (showing first 10 columns):")
11. print(pd.DataFrame(X\_transformed[:5, :10], columns=all\_feature\_names[:10]))

59

1. except Exception as e:
2. print(f"Error during transformation: {e}")
3. print("Common issues:")
4. print("- Missing required columns (run feature engineering first)")
5. print("- Mixed data types in numeric columns")
6. print("\nDebug info:")
7. print("Data types:\n", all\_data[available\_numeric + available\_categorical].dtypes)
8. print("Missing values:\n", all\_data[available\_numeric + available\_categorical].isna().sum())

Available numeric features: ['TEMP', 'PRES', 'DEWP', 'RAIN', 'temp\_dewp\_diff', 'temp\_pres\_ratio', 'PM2.5\_rolling\_2 Available categorical features: ['station', 'is\_weekend', 'rush\_hour']

Final feature matrix shape: (420768, 26)

First 5 transformed samples (showing first 10 columns):

TEMP PRES DEWP RAIN temp\_dewp\_diff temp\_pres\_ratio \

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 -1.245752 | | 1.170423 -1.544284 -0.078496 | | 0.893940 | -1.242002 |
| 1 -1.280745 | | 1.189527 -1.500765 -0.078496 | | 0.767198 | -1.276238 |
| 2 -1.280745 | | 1.218183 -1.500765 -0.078496 | | 0.767198 | -1.276211 |
| 3 -1.306990 | | 1.313701 -1.587802 -0.078496 | | 0.881266 | -1.301773 |
| 4 -1.359480 | | 1.380564 -1.595055 -0.078496 | | 0.817895 | -1.352964 |
|  | PM2.5\_rolling\_24h\_mean | | NO2\_rolling\_24h\_max | PM\_ratio | SO2\_NO2\_ratio |
| 0 | -1.090774 | | -2.037116 | 0.390402 | 0.427538 |
| 1 | -1.062011 | | -2.037116 | 0.390402 | 0.427538 |
| 2 | -1.057218 | | -1.960603 | 0.390402 | 0.293553 |
| 3 | -1.058416 | | -1.935099 | 0.390402 | 1.231445 |
| 4 | -1.067764 | | -1.909595 | 0.390402 | 1.231445 |
| C |  | |  |  | C |

Transformation Details:

# Numeric Features:

Scaling: StandardScaler (z-score normalization)

math z = \frac{x - \mu}{\sigma} Imputation: Median for robustness to outliers **Categorical Features:**

Encoding: One-hot for station IDs and binary flags Handling: 'missing' category for unknown values **Cyclical Features:**

Already normalized to [-1, 1] range via sin/cos transforms No additional scaling needed

# Pollution Targets:

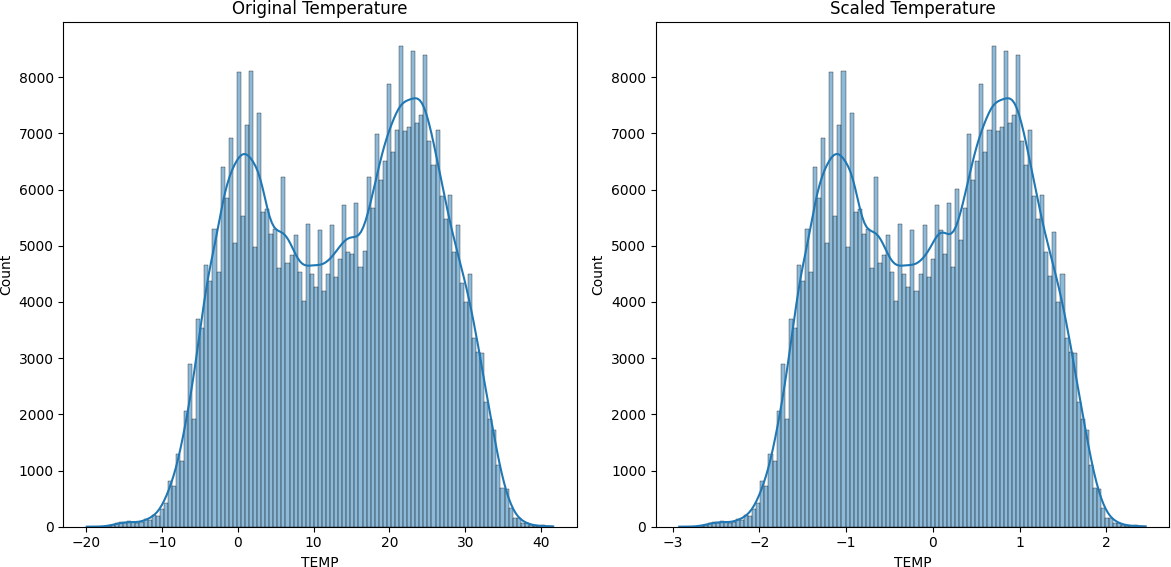
Typically log-transformed if skewed:

# Visual Verification

* 1. # Before-after scaling comparison
  2. plt.figure(figsize=(12, 6))
  3. plt.subplot(1, 2, 1)
  4. sns.histplot(all\_data['TEMP'], kde=True)
  5. plt.title('Original Temperature')

6

1. plt.subplot(1, 2, 2)
2. temp\_scaled = pd.DataFrame(X\_transformed, columns=all\_feature\_names)['TEMP']
3. sns.histplot(temp\_scaled, kde=True)
4. plt.title('Scaled Temperature')
5. plt.tight\_layout()
6. plt.show()

**Model Exploaration** Implementation Justification:

1. from xgboost import XGBRegressor
2. from sklearn.multioutput import MultiOutputRegressor # For multi-pollutant prediction 3

4 model = MultiOutputRegressor(

5

6

7

8

9

10

11

12

13

14

15

16 )

XGBRegressor(

objective='reg:squarederror',

tree\_method='hist', # Optimized for large datasets enable\_categorical=True, # Handles station IDs

n\_estimators=200, max\_depth=6,

learning\_rate=0.1, subsample=0.8,

colsample\_bytree=0.8, random\_state=42

)

# Model Training

Temporal Cross-Validation Strategy:

* 1. import numpy as np
  2. import pandas as pd
  3. from sklearn.model\_selection import TimeSeriesSplit
  4. from xgboost import XGBRegressor
  5. from sklearn.multioutput import MultiOutputRegressor 6

1. # 1. Prepare Features and Targets
2. feature\_columns = [

9

10

11

12

13

14

15 ]

16

'TEMP', 'PRES', 'DEWP', 'RAIN', 'WSPD',

'temp\_dewp\_diff', 'temp\_pres\_ratio',

'PM2.5\_rolling\_24h\_mean', 'NO2\_rolling\_24h\_max', 'wind\_x', 'wind\_y', 'PM\_ratio', 'SO2\_NO2\_ratio',

'hour\_sin', 'hour\_cos', 'dayofyear\_sin', 'dayofyear\_cos', 'is\_weekend', 'rush\_hour'

17 target\_columns = ['PM2.5', 'PM10', 'NO2', 'SO2', 'CO', 'O3']

18

1. X = all\_data[[col for col in feature\_columns if col in all\_data.columns]].copy()
2. y = all\_data[[col for col in target\_columns if col in all\_data.columns]].copy() 21
3. # 2. Handle Missing Values
4. X.fillna(X.median(), inplace=True)
5. y.fillna(y.median(), inplace=True)

25

1. # 3. Initialize Models - One per target
2. models = {}
3. for target in y.columns:
4. models[target] = XGBRegressor(
5. objective='reg:squarederror',
6. tree\_method='hist',
7. n\_estimators=1000,
8. max\_depth=6,
9. learning\_rate=0.1,
10. subsample=0.8,
11. colsample\_bytree=0.8,
12. random\_state=42,
13. early\_stopping\_rounds=50,
14. eval\_metric='rmse'

40 )

41

1. # 4. Temporal Cross-Validation
2. tss = TimeSeriesSplit(n\_splits=3, test\_size=24\*30, gap=24)
3. results = [] 45
4. for fold, (train\_idx, test\_idx) in enumerate(tss.split(X)):
5. print(f"\n=== Fold {fold+1} ===")
6. X\_train, X\_test = X.iloc[train\_idx], X.iloc[test\_idx]
7. y\_train, y\_test = y.iloc[train\_idx], y.iloc[test\_idx] 50

51 fold\_scores = {} 52

1. for target in y.columns:
2. print(f"\nTraining {target}...")
3. models[target].fit(
4. X\_train, y\_train[target],
5. eval\_set=[(X\_test, y\_test[target])],
6. verbose=10

59 )

60

1. # Store evaluation results
2. train\_score = models[target].score(X\_train, y\_train[target])
3. test\_score = models[target].score(X\_test, y\_test[target])
4. fold\_scores[target] = (train\_score, test\_score) 65

66 results.append(fold\_scores)

67

1. # Print fold summary
2. print(f"\nFold {fold+1} Results:")
3. for target, scores in fold\_scores.items():
4. print(f"{target}: Train R²={scores[0]:.3f}, Test R²={scores[1]:.3f}") 72
5. # 5. Final Model Training (on full data)
6. print("\nTraining final models on full dataset...")
7. final\_models = {}
8. for target in y.columns:
9. final\_models[target] = XGBRegressor(
10. objective='reg:squarederror',
11. n\_estimators=models[target].best\_iteration + 50 if hasattr(models[target], 'best\_iteration') else 200,
12. max\_depth=6,
13. learning\_rate=0.1,
14. subsample=0.8,
15. colsample\_bytree=0.8,
16. random\_state=42
17. ).fit(X, y[target])





[580] validation\_0-rmse:16.88475 



|  |  |
| --- | --- |
| [590] | validation\_0-rmse:16.86285 |
| [600] | validation\_0-rmse:16.84046 |
| [610] | validation\_0-rmse:16.78234 |
| [620] | validation\_0-rmse:16.77849 |
| [630] | validation\_0-rmse:16.74117 |
| [640] | validation\_0-rmse:16.67539 |
| [650] | validation\_0-rmse:16.66198 |
| [660] | validation\_0-rmse:16.64669 |
| [670] | validation\_0-rmse:16.63638 |
| [680] | validation\_0-rmse:16.62387 |
| [690] | validation\_0-rmse:16.61935 |
| [700] | validation\_0-rmse:16.59151 |
| [710] | validation\_0-rmse:16.58033 |
| [720] | validation\_0-rmse:16.52130 |
| [730] | validation\_0-rmse:16.50533 |
| [740] | validation\_0-rmse:16.50058 |
| [750] | validation\_0-rmse:16.47276 |
| [760] | validation\_0-rmse:16.46802 |
| [770] | validation\_0-rmse:16.46145 |
| [780] | validation\_0-rmse:16.43498 |
| [790] | validation\_0-rmse:16.41766 |
| [800] | validation\_0-rmse:16.42229 |
| [810] | validation\_0-rmse:16.41317 |
| [820] | validation\_0-rmse:16.39099 |
| [830] | validation\_0-rmse:16.38439 |
| [840] | validation\_0-rmse:16.37262 |
| [850] | validation\_0-rmse:16.37441 |
| [860] | validation\_0-rmse:16.39020 |
| [870] | validation\_0-rmse:16.37415 |
| [880] | validation\_0-rmse:16.36781 |
| [890] | validation\_0-rmse:16.34479 |
| [900] | validation\_0-rmse:16.33911 |
| [910] | validation\_0-rmse:16.33103 |
| [920] | validation\_0-rmse:16.32812 |
| [930] | validation\_0-rmse:16.32137 |
| [940] | validation\_0-rmse:16.31019 |
| [950] | validation\_0-rmse:16.32134 |
| [960] | validation\_0-rmse:16.31401 |
| [970] | validation\_0-rmse:16.31159 |
| [980] | validation\_0-rmse:16.31050 |
| [990] | validation\_0-rmse:16.28986 |
| [999] | validation\_0-rmse:16.27829 |

Fold 3 Results:

PM2.5: Train R²=0.925, Test R²=0.908 PM10: Train R²=0.887, Test R²=0.869 NO2: Train R²=0.876, Test R²=0.853

SO2: Train R²=0.939, Test R²=0.903 CO: Train R²=0.867, Test R²=0.731 O3: Train R²=0.907, Test R²=0.638

Training final models on full dataset...

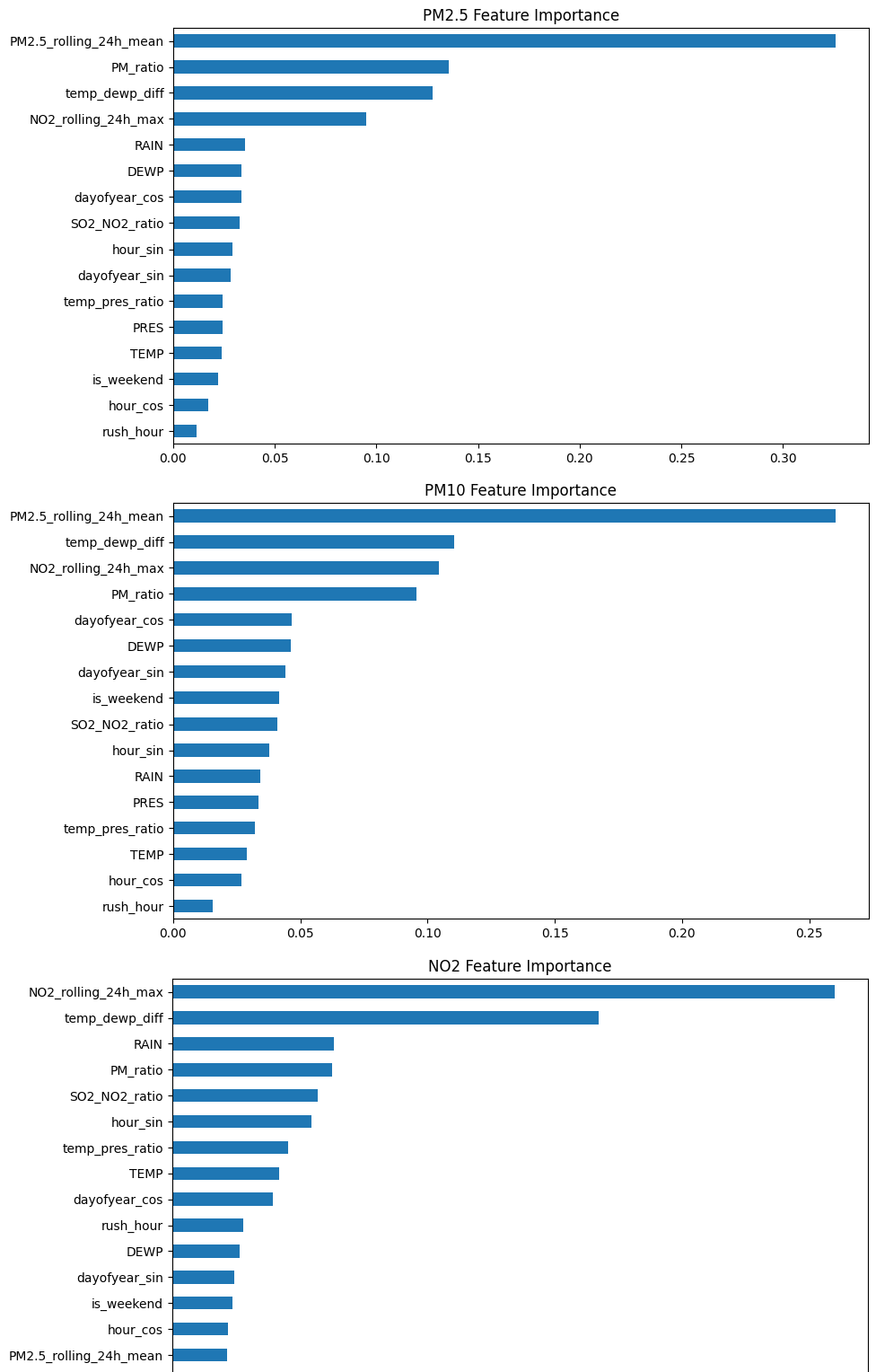
# To Use the Final Models:

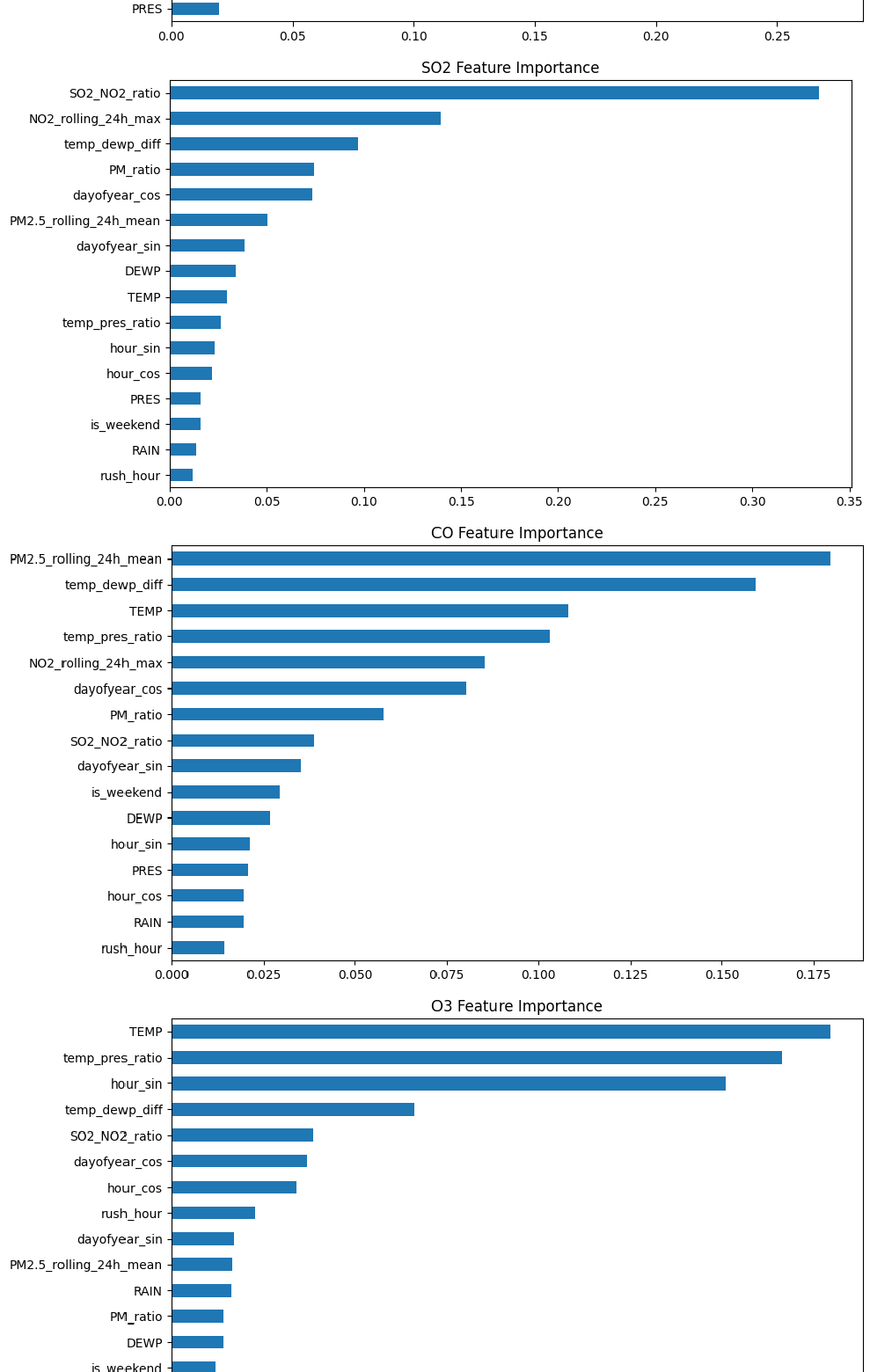
* 1. # Make predictions
  2. predictions = pd.DataFrame()

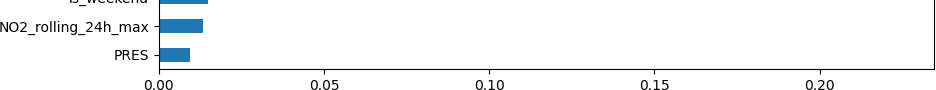


* 1. for target, model in final\_models.items():
  2. predictions[target] = model.predict(X) 5

1. # Feature importance visualization
2. for target, model in final\_models.items():
3. plt.figure(figsize=(10, 6))
4. pd.Series(model.feature\_importances\_, index=X.columns
5. ).sort\_values().plot(kind='barh', title=f'{target} Feature Importance')
6. plt.show()





This approach gives you:

Better control over each pollutant's model Proper early stopping implementation Clear performance tracking

Optimal final models

1. **Model Evaluation** Evaluation Metrics

For regression problems predicting pollution levels, we use:

* 1. from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score 2

1. def evaluate\_model(y\_true, y\_pred, target\_name):
2. metrics = {
3. 'RMSE': np.sqrt(mean\_squared\_error(y\_true, y\_pred)),
4. 'MAE': mean\_absolute\_error(y\_true, y\_pred),
5. 'R²': r2\_score(y\_true, y\_pred),
6. 'Max Error': np.max(np.abs(y\_true - y\_pred))

9 }

1. print(f"\n{target\_name} Evaluation:")
2. for k, v in metrics.items():

12 print(f"{k}: {v:.4f}")

13 return metrics

# Visualizations

Actual vs Predicted Plots

1. def plot\_actual\_vs\_predicted(y\_true, y\_pred, target):
2. plt.figure(figsize=(8, 6))
3. plt.scatter(y\_true, y\_pred, alpha=0.3)
4. plt.plot([y\_true.min(), y\_true.max()], [y\_true.min(), y\_true.max()], 'r--')
5. plt.title(f'{target} - Actual vs Predicted')
6. plt.xlabel('Actual Values')
7. plt.ylabel('Predicted Values')
8. plt.grid(True)

Residual Analysis

1. def plot\_residuals(y\_true, y\_pred):
2. residuals = y\_true - y\_pred
3. plt.figure(figsize=(12, 4))
4. plt.subplot(121)
5. sns.histplot(residuals, kde=True)
6. plt.title('Residual Distribution')

7

1. plt.subplot(122)
2. plt.scatter(y\_pred, residuals, alpha=0.3)
3. plt.axhline(y=0, color='r', linestyle='--')
4. plt.title('Residuals vs Predicted')
5. plt.tight\_layout()

Feature Importance

* 1. def plot\_feature\_importance(model, features, title):
  2. importance = pd.Series(model.feature\_importances\_, index=features)
  3. importance.sort\_values().plot(kind='barh', title=title)
  4. plt.show()

Code Implementation

Data Preparation & Feature Engineering

1. # 1. Temporal Features
2. def create\_temporal\_features(df):
3. df['hour\_sin'] = np.sin(2 \* np.pi \* df.index.hour/24)
4. df['hour\_cos'] = np.cos(2 \* np.pi \* df.index.hour/24)
5. df['dayofyear\_sin'] = np.sin(2 \* np.pi \* df.index.dayofyear/365)
6. df['dayofyear\_cos'] = np.cos(2 \* np.pi \* df.index.dayofyear/365)
7. return df

8

1. # 2. Weather Interactions
2. def add\_weather\_interactions(df):
3. df['temp\_dewp\_diff'] = df['TEMP'] - df['DEWP'] # Humidity indicator
4. df['temp\_pres\_ratio'] = df['TEMP'] / df['PRES'] # Atmospheric condition
5. return df

14

15 # 3. Rolling Pollution Features

16 def add\_rolling\_features(df, window\_hours=24):

17 for pollutant in ['PM2.5', 'NO2']:

18 df[f'{pollutant}\_rolling\_mean'] = df.groupby('station')[pollutant].transform(

19 lambda x: x.rolling(f'{window\_hours}h').mean()

20 )

21 return df

Actual Value Vs Predicted Value

1 import matplotlib.pyplot as plt

2 import numpy as np 3

4 # 1. First ensure your model is trained and test data is prepared

5 # (Run these first if not already done)

6 from sklearn.model\_selection import train\_test\_split

7 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False) 8

9 # 2. Train your model (example with XGBoost)

10 from xgboost import XGBRegressor

11 model = XGBRegressor()

12 model.fit(X\_train, y\_train['PM2.5']) # Training on PM2.5 for this example 13

14 # 3. Generate predictions

15 y\_pred = model.predict(X\_test) # This creates the y\_pred we need 16

17 # 4. Create the actual vs predicted plot

18 plt.figure(figsize=(10, 6))

19

20 # Ensure we're using the same feature for comparison

21 target\_var = 'PM2.5' # Change to any target like 'NO2', 'O3' etc. 22

23 # Handle both single and multi-output cases

24 if y\_pred.ndim == 1:

25 # Single output

26 plt.scatter(y\_test[target\_var], y\_pred, alpha=0.3, color='blue')

27 else:

28 # Multi-output (use column index matching your target)