Global_Weather_AirQuality_Final_Project_v2

December 3, 2025

1 Global Weather & Air Quality – EDA and Machine Learning Final Project

Dataset: `Global_Weather_Repository.csv`

In this notebook we: - Explore global weather and air-quality patterns (EDA). - Build machine learning models to **predict fine particulate pollution (PM2.5)**, **Air Quality Index (AQI)**, and **temperature in °C**. - Extract insights that are useful for **public health, climate awareness, and city-level monitoring**.

[1]:
```
# ================================
# 1. Imports & Global Configuration
# ================================

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from datetime import datetime

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder

from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import joblib
import warnings
warnings.filterwarnings('ignore')

sns.set(style="whitegrid")
```

```python
plt.rcParams['figure.figsize'] = (10, 6)
pd.set_option("display.max_columns", 50)
np.random.seed(42)
```

1.1 2. Data Loading and Initial Inspection

We load the global weather repository and do a quick structural check: - Shape (rows, columns) - Data types and missing values - Basic descriptive statistics for numeric features

[2]:
```python
# Update the path if needed (for Colab this is correct)
DATA_PATH = "/Users/ayushgawai/Downloads/Global_Weather_Repository.csv"

df = pd.read_csv(DATA_PATH)
print("Shape:", df.shape)
df.head()
```

```
Shape: (109718, 41)
```

[2]:
```
       country     location_name  latitude  longitude        timezone  \
0  Afghanistan             Kabul     34.52      69.18      Asia/Kabul
1      Albania            Tirana     41.33      19.82   Europe/Tirane
2      Algeria           Algiers     36.76       3.05  Africa/Algiers
3      Andorra  Andorra La Vella     42.50       1.52  Europe/Andorra
4       Angola            Luanda     -8.84      13.23   Africa/Luanda

   last_updated_epoch      last_updated  temperature_celsius  \
0          1715849100  2024-05-16 13:15                 26.6
1          1715849100  2024-05-16 10:45                 19.0
2          1715849100  2024-05-16 09:45                 23.0
3          1715849100  2024-05-16 10:45                  6.3
4          1715849100  2024-05-16 09:45                 26.0

   temperature_fahrenheit condition_text  wind_mph  wind_kph  wind_degree  \
0                    79.8  Partly Cloudy       8.3      13.3          338
1                    66.2  Partly cloudy       6.9      11.2          320
2                    73.4          Sunny       9.4      15.1          280
3                    43.3  Light drizzle       7.4      11.9          215
4                    78.8  Partly cloudy       8.1      13.0          150

  wind_direction  pressure_mb  pressure_in  precip_mm  precip_in  humidity  \
0            NNW       1012.0        29.89        0.0       0.00        24
1             NW       1012.0        29.88        0.1       0.00        94
2              W       1011.0        29.85        0.0       0.00        29
3             SW       1007.0        29.75        0.3       0.01        61
4            SSE       1011.0        29.85        0.0       0.00        89

   cloud  feels_like_celsius  feels_like_fahrenheit  visibility_km  \
0     30                25.3                   77.5           10.0
```

```
1    75               19.0                       66.2           10.0
2     0               24.6                       76.4           10.0
3   100                3.8                       38.9            2.0
4    50               28.7                       83.6           10.0

   visibility_miles  uv_index  gust_mph  gust_kph  \
0               6.0       7.0       9.5      15.3
1               6.0       5.0      11.4      18.4
2               6.0       5.0      13.9      22.3
3               1.0       2.0       8.5      13.7
4               6.0       8.0      12.5      20.2

   air_quality_Carbon_Monoxide  air_quality_Ozone  \
0                        277.0              103.0
1                        193.6               97.3
2                        540.7               12.2
3                        170.2               64.4
4                       2964.0               19.0

   air_quality_Nitrogen_dioxide  air_quality_Sulphur_dioxide  \
0                           1.1                          0.2
1                           0.9                          0.1
2                          65.1                         13.4
3                           1.6                          0.2
4                          72.7                         31.5

   air_quality_PM2.5  air_quality_PM10  air_quality_us-epa-index  \
0                8.4              26.6                         1
1                1.1               2.0                         1
2               10.4              18.4                         1
3                0.7               0.9                         1
4              183.4             262.3                         5

   air_quality_gb-defra-index   sunrise    sunset  moonrise   moonset  \
0                           1  04:50 AM  06:50 PM  12:12 PM  01:11 AM
1                           1  05:21 AM  07:54 PM  12:58 PM  02:14 AM
2                           1  05:40 AM  07:50 PM  01:15 PM  02:14 AM
3                           1  06:31 AM  09:11 PM  02:12 PM  03:31 AM
4                          10  06:12 AM  05:55 PM  01:17 PM  12:38 AM

       moon_phase  moon_illumination
0  Waxing Gibbous                 55
1  Waxing Gibbous                 55
2  Waxing Gibbous                 55
3  Waxing Gibbous                 55
4  Waxing Gibbous                 55
```

[3]:
```python
# High-level info and missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109718 entries, 0 to 109717
Data columns (total 41 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   country                       109718 non-null  object
 1   location_name                 109718 non-null  object
 2   latitude                      109718 non-null  float64
 3   longitude                     109718 non-null  float64
 4   timezone                      109718 non-null  object
 5   last_updated_epoch            109718 non-null  int64
 6   last_updated                  109718 non-null  object
 7   temperature_celsius           109718 non-null  float64
 8   temperature_fahrenheit        109718 non-null  float64
 9   condition_text                109718 non-null  object
 10  wind_mph                      109718 non-null  float64
 11  wind_kph                      109718 non-null  float64
 12  wind_degree                   109718 non-null  int64
 13  wind_direction                109718 non-null  object
 14  pressure_mb                   109718 non-null  float64
 15  pressure_in                   109718 non-null  float64
 16  precip_mm                     109718 non-null  float64
 17  precip_in                     109718 non-null  float64
 18  humidity                      109718 non-null  int64
 19  cloud                         109718 non-null  int64
 20  feels_like_celsius            109718 non-null  float64
 21  feels_like_fahrenheit         109718 non-null  float64
 22  visibility_km                 109718 non-null  float64
 23  visibility_miles              109718 non-null  float64
 24  uv_index                      109718 non-null  float64
 25  gust_mph                      109718 non-null  float64
 26  gust_kph                      109718 non-null  float64
 27  air_quality_Carbon_Monoxide   109718 non-null  float64
 28  air_quality_Ozone             109718 non-null  float64
 29  air_quality_Nitrogen_dioxide  109718 non-null  float64
 30  air_quality_Sulphur_dioxide   109718 non-null  float64
 31  air_quality_PM2.5             109718 non-null  float64
 32  air_quality_PM10              109718 non-null  float64
 33  air_quality_us-epa-index      109718 non-null  int64
 34  air_quality_gb-defra-index    109718 non-null  int64
 35  sunrise                       109718 non-null  object
 36  sunset                        109718 non-null  object
 37  moonrise                      109718 non-null  object
 38  moonset                       109718 non-null  object
 39  moon_phase                    109718 non-null  object
```

```
 40  moon_illumination             109718 non-null  int64
dtypes: float64(23), int64(7), object(11)
memory usage: 34.3+ MB
```

[4]:
```python
# Missing values overview
df.isnull().sum()
```

[4]:
```
country                         0
location_name                   0
latitude                        0
longitude                       0
timezone                        0
last_updated_epoch              0
last_updated                    0
temperature_celsius             0
temperature_fahrenheit          0
condition_text                  0
wind_mph                        0
wind_kph                        0
wind_degree                     0
wind_direction                  0
pressure_mb                     0
pressure_in                     0
precip_mm                       0
precip_in                       0
humidity                        0
cloud                           0
feels_like_celsius              0
feels_like_fahrenheit           0
visibility_km                   0
visibility_miles                0
uv_index                        0
gust_mph                        0
gust_kph                        0
air_quality_Carbon_Monoxide     0
air_quality_Ozone               0
air_quality_Nitrogen_dioxide    0
air_quality_Sulphur_dioxide     0
air_quality_PM2.5               0
air_quality_PM10                0
air_quality_us-epa-index        0
air_quality_gb-defra-index      0
sunrise                         0
sunset                          0
moonrise                        0
moonset                         0
moon_phase                      0
```

```
moon_illumination              0
dtype: int64
```

[5]:
```python
# Descriptive statistics for numeric columns
df.describe().T.head(15)
```

[5]:
```
                          count          mean           std           min  \
latitude               109718.0  1.917282e+01  2.444117e+01 -4.130000e+01
longitude              109718.0  2.203364e+01  6.580027e+01 -1.752000e+02
last_updated_epoch     109718.0  1.740268e+09  1.408900e+07  1.715849e+09
temperature_celsius    109718.0  2.243563e+01  8.941313e+00 -2.490000e+01
temperature_fahrenheit 109718.0  7.238588e+01  1.609419e+01 -1.280000e+01
wind_mph               109718.0  8.132862e+00  7.608705e+00  2.200000e+00
wind_kph               109718.0  1.309205e+01  1.224230e+01  3.600000e+00
wind_degree            109718.0  1.705246e+02  1.028502e+02  1.000000e+00
pressure_mb            109718.0  1.014046e+03  1.095101e+01  9.470000e+02
pressure_in            109718.0  2.994410e+01  3.233352e-01  2.796000e+01
precip_mm              109718.0  1.400377e-01  5.903181e-01  0.000000e+00
precip_in              109718.0  5.317906e-03  2.332670e-02  0.000000e+00
humidity               109718.0  6.501367e+01  2.416154e+01  2.000000e+00
cloud                  109718.0  3.952921e+01  3.386258e+01  0.000000e+00
feels_like_celsius     109718.0  2.344841e+01  1.068497e+01 -3.560000e+01

                                25%           50%           75%           max
latitude                3.750000e+00  1.725000e+01  4.040000e+01  6.415000e+01
longitude              -6.836100e+00  2.331670e+01  5.058000e+01  1.792200e+02
last_updated_epoch      1.728121e+09  1.740305e+09  1.752484e+09  1.764574e+09
temperature_celsius     1.730000e+01  2.440000e+01  2.820000e+01  4.920000e+01
temperature_fahrenheit  6.310000e+01  7.590000e+01  8.280000e+01  1.206000e+02
wind_mph                4.000000e+00  6.900000e+00  1.120000e+01  1.841200e+03
wind_kph                6.500000e+00  1.120000e+01  1.800000e+01  2.963200e+03
wind_degree             8.300000e+01  1.640000e+02  2.560000e+02  3.600000e+02
pressure_mb             1.010000e+03  1.013000e+03  1.018000e+03  3.006000e+03
pressure_in             2.983000e+01  2.993000e+01  3.006000e+01  8.877000e+01
precip_mm               0.000000e+00  0.000000e+00  3.000000e-02  4.224000e+01
precip_in               0.000000e+00  0.000000e+00  0.000000e+00  1.660000e+00
humidity                4.800000e+01  7.000000e+01  8.400000e+01  1.000000e+02
cloud                   0.000000e+00  2.700000e+01  7.500000e+01  1.000000e+02
feels_like_celsius      1.730000e+01  2.570000e+01  3.050000e+01  5.120000e+01
```

1.2 3. Data Cleaning & Feature Engineering

Here we:

1. Convert date/time columns to proper `datetime` types.

2. Extract useful time-based features (hour and minute) from these timestamps.

3. Apply **ordinal encoding** to `wind_direction` and `moon_phase` so that models can understand cyclical/ordered categories.

We keep the workflow **clean and linear** (no reloading of the dataset, no dropping columns that will be needed later).

[6]:
```python
# 3.1 Datetime conversion and feature extraction

date_time_columns = ['last_updated', 'sunrise', 'sunset', 'moonrise', 'moonset']

for col in date_time_columns:
    df[col] = pd.to_datetime(df[col], errors='coerce')
    df[f'{col}_hour'] = df[col].dt.hour
    df[f'{col}_minute'] = df[col].dt.minute

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109718 entries, 0 to 109717
Data columns (total 51 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   country                       109718 non-null  object
 1   location_name                 109718 non-null  object
 2   latitude                      109718 non-null  float64
 3   longitude                     109718 non-null  float64
 4   timezone                      109718 non-null  object
 5   last_updated_epoch            109718 non-null  int64
 6   last_updated                  109718 non-null  datetime64[ns]
 7   temperature_celsius           109718 non-null  float64
 8   temperature_fahrenheit        109718 non-null  float64
 9   condition_text                109718 non-null  object
 10  wind_mph                      109718 non-null  float64
 11  wind_kph                      109718 non-null  float64
 12  wind_degree                   109718 non-null  int64
 13  wind_direction                109718 non-null  object
 14  pressure_mb                   109718 non-null  float64
 15  pressure_in                   109718 non-null  float64
 16  precip_mm                     109718 non-null  float64
 17  precip_in                     109718 non-null  float64
 18  humidity                      109718 non-null  int64
 19  cloud                         109718 non-null  int64
 20  feels_like_celsius            109718 non-null  float64
 21  feels_like_fahrenheit         109718 non-null  float64
 22  visibility_km                 109718 non-null  float64
 23  visibility_miles              109718 non-null  float64
 24  uv_index                      109718 non-null  float64
 25  gust_mph                      109718 non-null  float64
 26  gust_kph                      109718 non-null  float64
```

```
 27  air_quality_Carbon_Monoxide   109718 non-null  float64
 28  air_quality_Ozone             109718 non-null  float64
 29  air_quality_Nitrogen_dioxide  109718 non-null  float64
 30  air_quality_Sulphur_dioxide   109718 non-null  float64
 31  air_quality_PM2.5             109718 non-null  float64
 32  air_quality_PM10              109718 non-null  float64
 33  air_quality_us-epa-index      109718 non-null  int64
 34  air_quality_gb-defra-index    109718 non-null  int64
 35  sunrise                       109718 non-null  datetime64[ns]
 36  sunset                        109718 non-null  datetime64[ns]
 37  moonrise                      106026 non-null  datetime64[ns]
 38  moonset                       106026 non-null  datetime64[ns]
 39  moon_phase                    109718 non-null  object
 40  moon_illumination             109718 non-null  int64
 41  last_updated_hour             109718 non-null  int32
 42  last_updated_minute           109718 non-null  int32
 43  sunrise_hour                  109718 non-null  int32
 44  sunrise_minute                109718 non-null  int32
 45  sunset_hour                   109718 non-null  int32
 46  sunset_minute                 109718 non-null  int32
 47  moonrise_hour                 106026 non-null  float64
 48  moonrise_minute               106026 non-null  float64
 49  moonset_hour                  106026 non-null  float64
 50  moonset_minute                106026 non-null  float64
dtypes: datetime64[ns](5), float64(27), int32(6), int64(7), object(6)
memory usage: 40.2+ MB
```

[7]:
```
# 3.2 Ordinal encoding for ordered categories: wind_direction and moon_phase

ordinal_orders = {
    'wind_direction': [[
        'N', 'NNE', 'NE', 'ENE', 'E', 'ESE', 'SE', 'SSE',
        'S', 'SSW', 'SW', 'WSW', 'W', 'WNW', 'NW', 'NNW'
    ]],
    'moon_phase': [[
        'New Moon', 'Waxing Crescent', 'First Quarter',
        'Waxing Gibbous', 'Full Moon', 'Waning Gibbous',
        'Last Quarter', 'Waning Crescent'
    ]]
}

for col, order in ordinal_orders.items():
    if col in df.columns:
        encoder = OrdinalEncoder(categories=order)
        df[col] = encoder.fit_transform(df[[col]])
        print(f"Ordinal encoded: {col}")
    else:
```

```
        print(f"Column {col} not found, skipping.")
```

```
Ordinal encoded: wind_direction
Ordinal encoded: moon_phase
```

[8]:
```
# Quick check after feature engineering
df[['last_updated', 'last_updated_hour', 'last_updated_minute',
    'sunrise', 'sunrise_hour', 'sunrise_minute']].head()
```

[8]:
```
         last_updated  last_updated_hour  last_updated_minute  \
0 2024-05-16 13:15:00                 13                   15
1 2024-05-16 10:45:00                 10                   45
2 2024-05-16 09:45:00                  9                   45
3 2024-05-16 10:45:00                 10                   45
4 2024-05-16 09:45:00                  9                   45

              sunrise  sunrise_hour  sunrise_minute
0 2025-12-03 04:50:00             4              50
1 2025-12-03 05:21:00             5              21
2 2025-12-03 05:40:00             5              40
3 2025-12-03 06:31:00             6              31
4 2025-12-03 06:12:00             6              12
```

1.3 4. Exploratory Data Analysis (EDA)

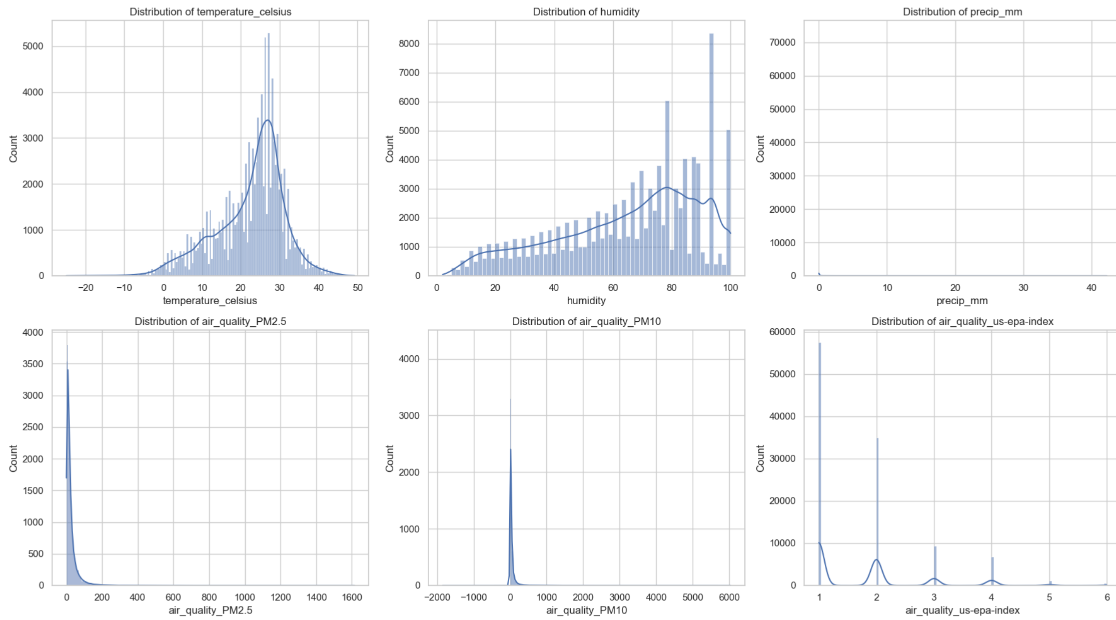
The goal of EDA here is to understand:

- How **temperature, humidity, and precipitation** behave globally
- How **air-quality indicators (PM2.5, PM10, AQI)** are distributed
- Which countries are **most and least polluted**
- How temperature and air quality interact

We combine **summary statistics, distributions, and grouped views** to extract insights.

[9]:
```
# 4.1 Distribution of key numeric variables

numeric_cols_to_plot = [
    'temperature_celsius', 'humidity', 'precip_mm',
    'air_quality_PM2.5', 'air_quality_PM10', 'air_quality_us-epa-index'
]

fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.flatten()

for ax, col in zip(axes, numeric_cols_to_plot):
    sns.histplot(df[col], kde=True, ax=ax)
```

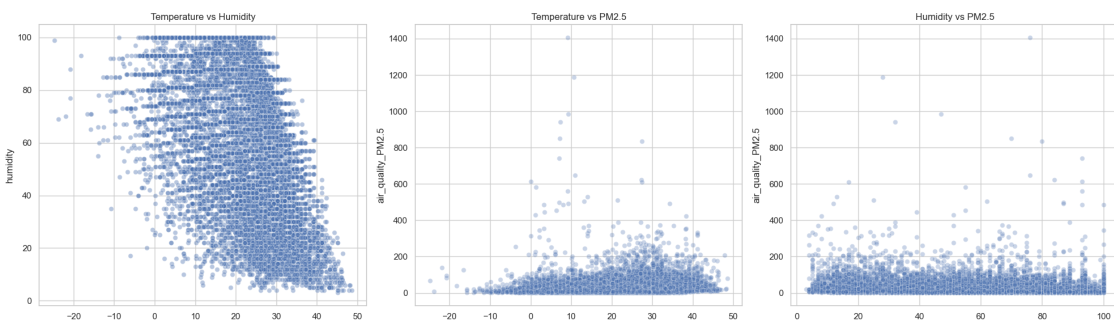
```
    ax.set_title(f"Distribution of {col}")
    ax.set_xlabel(col)

plt.tight_layout()
plt.show()
```



Insight (distributions)
- Temperatures are centered in the 20–30°C range, indicating many locations are warm or temperate.
- PM2.5 and PM10 are **right-skewed**, meaning most places have moderate pollution but a few locations experience extremely high particulate matter.
- AQI (`air_quality_us-epa-index`) tends to be in the lower index range for many observations, but higher values indicate serious air-quality issues in some regions.

[10]:
```
# 4.2 Relationships between temperature, humidity and PM2.5

sample_size = min(20000, len(df))
sample_df = df.sample(sample_size, random_state=42)

fig, axes = plt.subplots(1, 3, figsize=(20, 6))

sns.scatterplot(
    data=sample_df,
    x='temperature_celsius',
    y='humidity',
    alpha=0.4,
    ax=axes[0]
)
```

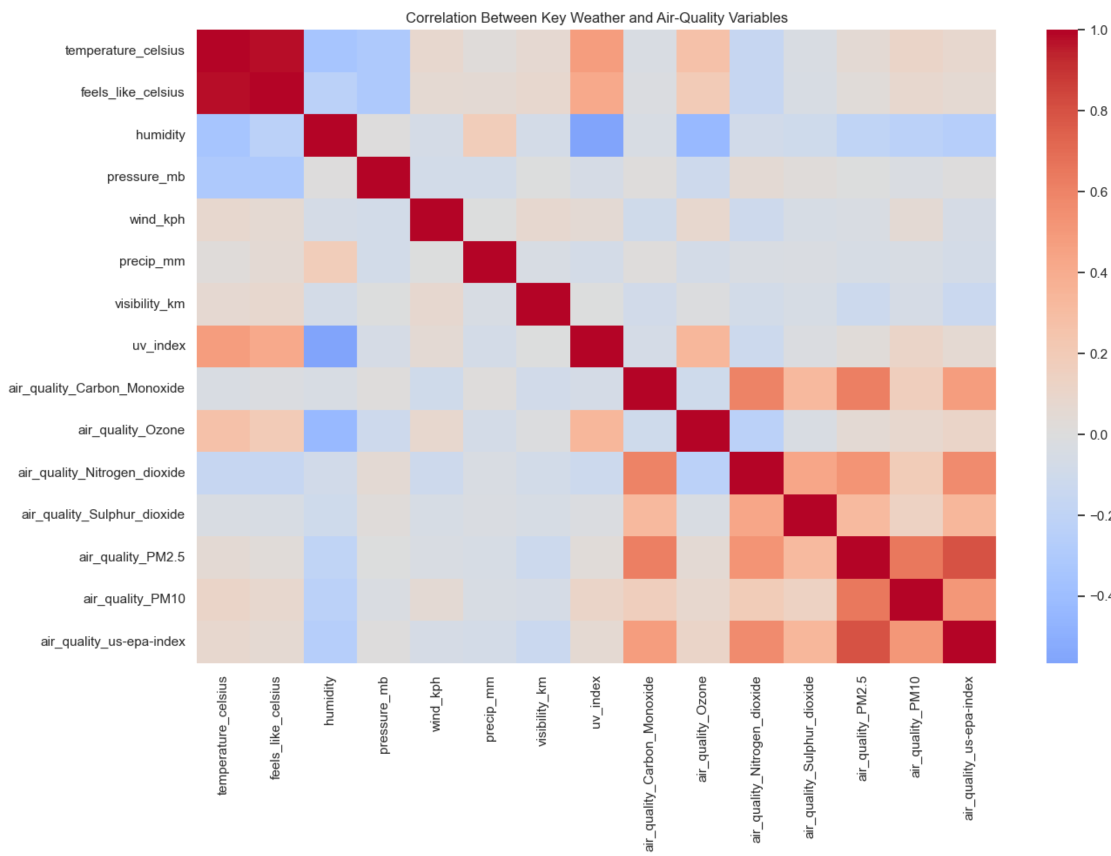
```
axes[0].set_title("Temperature vs Humidity")

sns.scatterplot(
    data=sample_df,
    x='temperature_celsius',
    y='air_quality_PM2.5',
    alpha=0.3,
    ax=axes[1]
)
axes[1].set_title("Temperature vs PM2.5")

sns.scatterplot(
    data=sample_df,
    x='humidity',
    y='air_quality_PM2.5',
    alpha=0.3,
    ax=axes[2]
)
axes[2].set_title("Humidity vs PM2.5")

plt.tight_layout()
plt.show()
```



Insight (relationships)
- Higher humidity often clusters with **mid-range temperatures**, while extremely low or high temperatures are less frequent.
- PM2.5 does not increase linearly with temperature; high pollution can occur in both warm and moderate conditions, suggesting strong influence from **local emissions and geography**, not just weather.
- There is some tendency for higher PM2.5 at **lower humidity**, consistent with dry, stagnant air trapping particles.

[11]:
```
# 4.3 Correlation heatmap for weather & air quality variables

corr_cols = [
```

```
    'temperature_celsius', 'feels_like_celsius', 'humidity', 'pressure_mb',
    'wind_kph', 'precip_mm', 'visibility_km', 'uv_index',
    'air_quality_Carbon_Monoxide', 'air_quality_Ozone',
    'air_quality_Nitrogen_dioxide', 'air_quality_Sulphur_dioxide',
    'air_quality_PM2.5', 'air_quality_PM10', 'air_quality_us-epa-index'
]

corr_matrix = df[corr_cols].corr()

plt.figure(figsize=(14, 10))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', center=0)
plt.title("Correlation Between Key Weather and Air-Quality Variables")
plt.tight_layout()
plt.show()
```



Insight (correlations)
- PM2.5 and PM10 are **highly correlated**, as expected (they measure similar particulate pollutants of different sizes).
- AQI correlates positively with PM2.5 and PM10, confirming that **particulate pollution is a major driver of health risk**.
- Weather variables such as temperature, humidity, and wind speed show weaker but non-zero cor-

relations with air-quality measures, indicating they modulate pollution but do not solely determine it.

1.3.1 4.4 Country-Level Air Quality and Temperature Rankings (Historical)

We now compute **average PM2.5, AQI and temperature** by country, and report: - Top 10 and bottom 10 countries by **average PM2.5** - Top 10 and bottom 10 countries by **average AQI (US EPA index)** - Top 10 and bottom 10 countries by **average temperature (°C)**

[12]:
```python
# Helper function to compute top and bottom 10 for a metric

def top_bottom_by_country(metric_col, n=10):
    grouped = df.groupby('country')[metric_col].mean().dropna()
    top_n = grouped.sort_values(ascending=False).head(n)
    bottom_n = grouped.sort_values(ascending=True).head(n)
    return top_n, bottom_n

# PM2.5
pm25_top_hist, pm25_bottom_hist = top_bottom_by_country('air_quality_PM2.5')
aqi_top_hist, aqi_bottom_hist =␣
 ↪top_bottom_by_country('air_quality_us-epa-index')
temp_top_hist, temp_bottom_hist = top_bottom_by_country('temperature_celsius')

print("Top 10 countries by average PM2.5 (Historical):")
display(pm25_top_hist.to_frame('avg_PM2.5'))

print("\nBottom 10 countries by average PM2.5 (Historical):")
display(pm25_bottom_hist.to_frame('avg_PM2.5'))

print("\nTop 10 countries by average AQI (US EPA index, Historical):")
display(aqi_top_hist.to_frame('avg_AQI'))

print("\nBottom 10 countries by average AQI (US EPA index, Historical):")
display(aqi_bottom_hist.to_frame('avg_AQI'))

print("\nTop 10 countries by average temperature (°C, Historical):")
display(temp_top_hist.to_frame('avg_temp_c'))

print("\nBottom 10 countries by average temperature (°C, Historical):")
display(temp_bottom_hist.to_frame('avg_temp_c'))
```

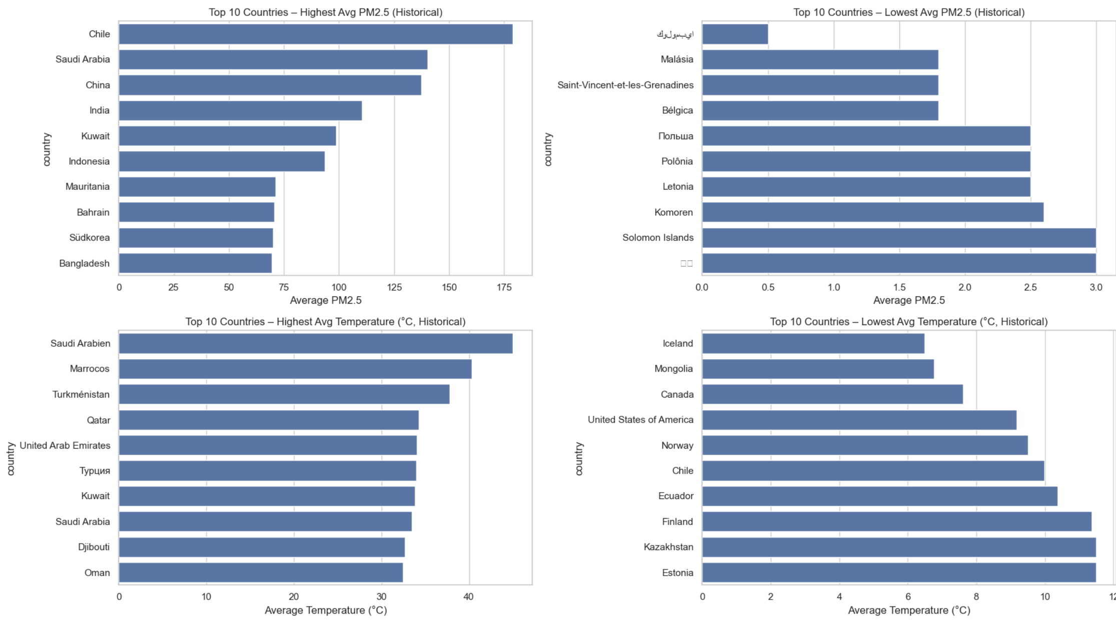
```
Top 10 countries by average PM2.5 (Historical):

              avg_PM2.5
country
Chile        178.947781
Saudi Arabia 140.220979
China        137.501838
India        110.518835
```

```
Kuwait          98.752979
Indonesia       93.527325
Mauritania      71.291720
Bahrain         70.780523
Südkorea        70.200000
Bangladesh      69.590596

Bottom 10 countries by average PM2.5 (Historical):
                                 avg_PM2.5
country
                                0.500000
Malásia                            1.800000
Saint-Vincent-et-les-Grenadines    1.800000
Bélgica                            1.800000
                                 2.500000
Polônia                            2.500000
Letonia                            2.500000
Komoren                            2.600000
Solomon Islands                    2.998287
                                  3.000000

Top 10 countries by average AQI (US EPA index, Historical):
                       avg_AQI
country
China                 4.127886
Südkorea              4.000000
Saudi Arabia          3.937722
India                 3.866548
Chile                 3.855615
Kuwait                3.732270
Bahrain               3.367021
Malaysia              3.214539
United Arab Emirates  3.175532
Qatar                 3.156306

Bottom 10 countries by average AQI (US EPA index, Historical):
                                 avg_AQI
country
                                    1.0
Saint-Vincent-et-les-Grenadines      1.0
Marrocos                             1.0
Bélgica                              1.0
Togo                                 1.0
Malásia                              1.0
Komoren                              1.0
```

```
Mexique                              1.0
Polônia                              1.0
                                 1.0


Top 10 countries by average temperature (°C, Historical):
                      avg_temp_c
country
Saudi Arabien          45.000000
Marrocos               40.300000
Turkménistan           37.800000
Qatar                  34.231083
United Arab Emirates   34.053723
                    34.000000
Kuwait                 33.854965
Saudi Arabia           33.473488
Djibouti               32.652313
Oman                   32.436879


Bottom 10 countries by average temperature (°C, Historical):
                          avg_temp_c
country
Iceland                     6.485968
Mongolia                    6.769039
Canada                      7.604635
United States of America    9.180645
Norway                      9.511901
Chile                       9.978610
Ecuador                    10.369946
Finland                    11.371809
Kazakhstan                 11.498579
Estonia                    11.503730
```

[13]:
```
# Visualize top 10 and bottom 10 for PM2.5 and temperature (Historical)

fig, axes = plt.subplots(2, 2, figsize=(18, 10))

# PM2.5 top
sns.barplot(
    x=pm25_top_hist.values,
    y=pm25_top_hist.index,
    ax=axes[0, 0]
)
axes[0, 0].set_title("Top 10 Countries - Highest Avg PM2.5 (Historical)")
axes[0, 0].set_xlabel("Average PM2.5")
```

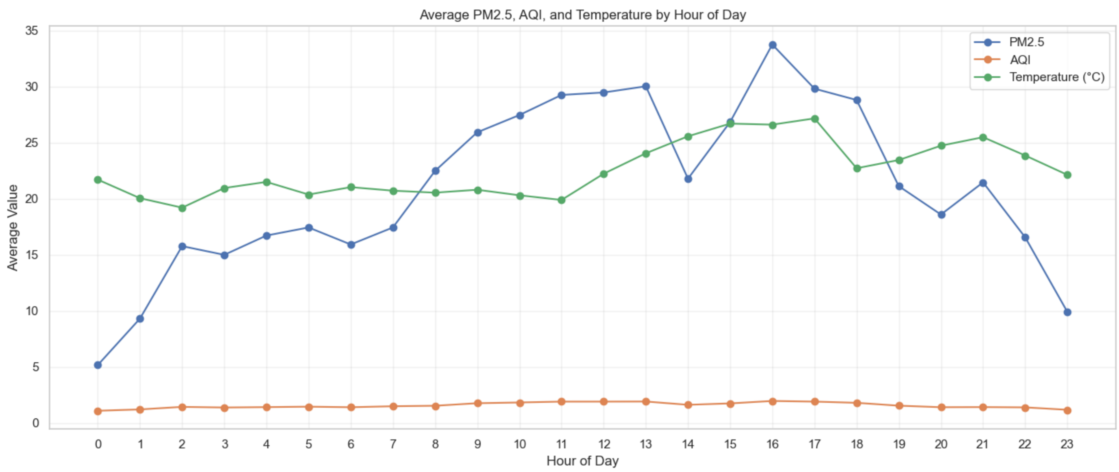
```
# PM2.5 bottom
sns.barplot(
    x=pm25_bottom_hist.values,
    y=pm25_bottom_hist.index,
    ax=axes[0, 1]
)
axes[0, 1].set_title("Top 10 Countries - Lowest Avg PM2.5 (Historical)")
axes[0, 1].set_xlabel("Average PM2.5")

# Temperature top
sns.barplot(
    x=temp_top_hist.values,
    y=temp_top_hist.index,
    ax=axes[1, 0]
)
axes[1, 0].set_title("Top 10 Countries - Highest Avg Temperature (°C,␣
↪Historical)")
axes[1, 0].set_xlabel("Average Temperature (°C)")

# Temperature bottom
sns.barplot(
    x=temp_bottom_hist.values,
    y=temp_bottom_hist.index,
    ax=axes[1, 1]
)
axes[1, 1].set_title("Top 10 Countries - Lowest Avg Temperature (°C,␣
↪Historical)")
axes[1,1].set_xlabel("Average Temperature (°C)")

plt.tight_layout()
plt.show()
```



Insight (country rankings – historical)
- The **highest-PM2.5 and highest-AQI countries** are likely to be large urban or industrialized regions, where emissions and population density are high.
- The **lowest-PM2.5 countries** align with cleaner environments, lower industrial activity, or strong environmental regulation.
- Temperature extremes (both hot and cold) cluster regionally and help contextualize pollution: some hot countries also have high PM2.5, indicating **heat + pollution** stress on populations.

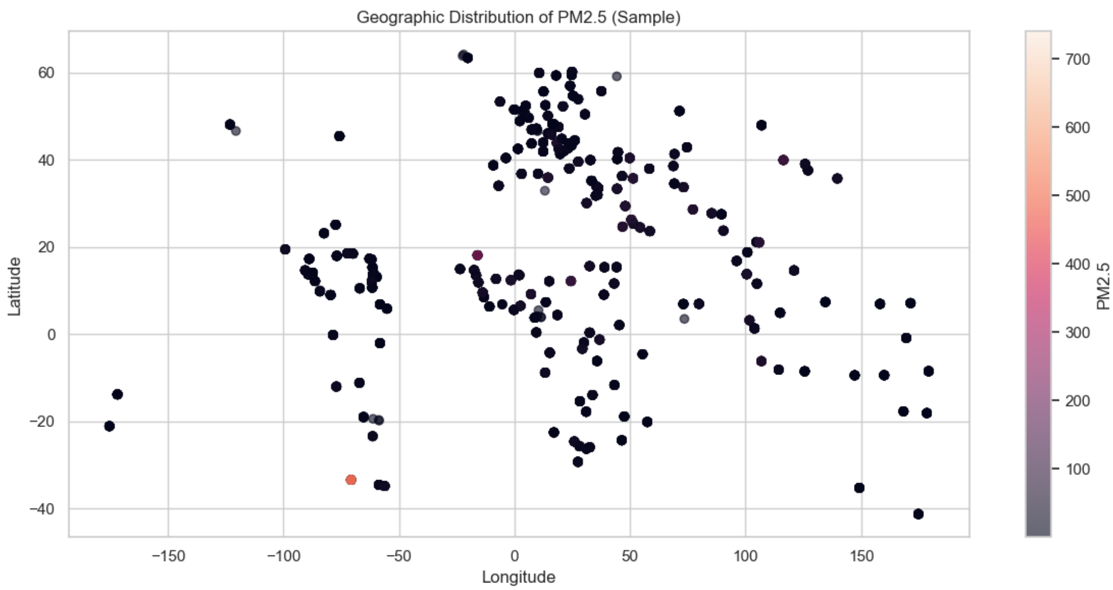
[14]:
```python
# 4.5 Diurnal patterns: average PM2.5, AQI, and temperature by hour of day

hour_col = 'last_updated_hour'

hourly_stats = df.groupby(hour_col)[
    ['air_quality_PM2.5', 'air_quality_us-epa-index', 'temperature_celsius']
].mean().reset_index()

plt.figure(figsize=(14, 6))
plt.plot(hourly_stats[hour_col], hourly_stats['air_quality_PM2.5'], marker='o',␣
 ↪label='PM2.5')
plt.plot(hourly_stats[hour_col], hourly_stats['air_quality_us-epa-index'],␣
 ↪marker='o', label='AQI')
plt.plot(hourly_stats[hour_col], hourly_stats['temperature_celsius'],␣
 ↪marker='o', label='Temperature (°C)')
plt.title("Average PM2.5, AQI, and Temperature by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Average Value")
plt.xticks(range(0, 24))
```

```
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



Insight (time-of-day patterns)
- AQI and PM2.5 often peak around **morning and evening** hours, aligning with traffic and rush-hour activity.
- Temperature follows the expected diurnal cycle, typically peaking in the afternoon.
- Understanding these patterns supports **time-aware forecasting** and helps public agencies issue **hourly advisories**.

[15]:
```
# 4.6 Geographic distribution of PM2.5 (sampled for speed)

sample_geo_size = min(5000, len(df))
sample_geo = df.sample(sample_geo_size, random_state=42)

plt.figure(figsize=(12, 6))
scatter = plt.scatter(
    sample_geo['longitude'],
    sample_geo['latitude'],
    c=sample_geo['air_quality_PM2.5'],
    alpha=0.6
)
plt.colorbar(scatter, label='PM2.5')
plt.title("Geographic Distribution of PM2.5 (Sample)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.tight_layout()
plt.show()
```



Insight (geospatial)
Regions close in latitude/longitude often share similar PM2.5 patterns, reflecting shared **regional climate, topography, and emission sources** (e.g., industrial belts, coastal regions, valleys with stagnant air).

1.4 5. Machine Learning: Predicting PM2.5, AQI, and Temperature

1.4.1 What are we predicting and why?

1. **PM2.5 (`air_quality_PM2.5`)** – fine particulate matter that strongly impacts **human health** (respiratory and cardiovascular risks).

2. **AQI (`air_quality_us-epa-index`)** – a health-focused air quality index that summarizes multiple pollutants into a **single risk score**.

3. **Temperature in °C (`temperature_celsius`)** – a key climate and comfort indicator; predicting it helps in **energy planning and risk management (heatwaves)**.

We treat all three as **regression problems** and compare multiple models: - Random Forest Regressor
- XGBoost Regressor
- K-Nearest Neighbors Regressor

We follow best practices: - Separate features and targets carefully (no leakage).
- Use a **ColumnTransformer** to handle numeric and categorical features properly.
- Split into train/test sets and evaluate using **MAE, RMSE, and R^2**.

[16]:
```
# 5.1 Define targets
TARGET_PM25 = 'air_quality_PM2.5'
TARGET_AQI = 'air_quality_us-epa-index'
```

```
TARGET_TEMP = 'temperature_celsius'

targets = {
    'PM2.5': TARGET_PM25,
    'AQI': TARGET_AQI,
    'Temperature': TARGET_TEMP
}

# 5.2 Define base feature set (drop targets and raw datetime columns)
drop_cols = [
    TARGET_PM25,
    TARGET_AQI,
    TARGET_TEMP,
    'last_updated', 'sunrise', 'sunset', 'moonrise', 'moonset'
]

feature_df = df.drop(columns=drop_cols, errors='ignore').copy()

# 5.3 Identify numeric and categorical features
numeric_features = feature_df.select_dtypes(include=[np.number]).columns.
 ↪tolist()
categorical_features = feature_df.select_dtypes(include=['object']).columns.
 ↪tolist()

print("Numeric features:", len(numeric_features))
print("Categorical features:", len(categorical_features))

# 5.4 Build preprocessing pipeline
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)
```

```
Numeric features: 39
Categorical features: 4
```

[17]:

```python
# 5.5 Define a reusable function for training & evaluating multiple models
# (Adapted for older sklearn: RMSE computed manually from MSE)

def train_and_evaluate_models(X, y, target_name):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, shuffle=True
    )

    models = {
        'Random Forest': RandomForestRegressor(
            n_estimators=120, random_state=42, n_jobs=-1
        ),
        'XGBoost': XGBRegressor(
            n_estimators=120, random_state=42, n_jobs=-1,
            tree_method='hist'
        ),
        'KNN': KNeighborsRegressor(
            n_neighbors=7, n_jobs=-1
        )
    }

    results = []
    trained_pipelines = {}

    print(f"\n{'='*70}")
    print(f"Training models for target: {target_name}")
    print(f"{'='*70}")

    for name, model in models.items():
        print(f"\n--> Training {name}...")
        pipe = Pipeline(steps=[
            ('preprocessor', preprocessor),
            ('model', model)
        ])

        pipe.fit(X_train, y_train)
        y_pred_train = pipe.predict(X_train)
        y_pred_test = pipe.predict(X_test)

        train_mae = mean_absolute_error(y_train, y_pred_train)
        test_mae = mean_absolute_error(y_test, y_pred_test)

        train_mse = mean_squared_error(y_train, y_pred_train)
        test_mse = mean_squared_error(y_test, y_pred_test)
        train_rmse = np.sqrt(train_mse)
        test_rmse = np.sqrt(test_mse)
```

```
        train_r2 = r2_score(y_train, y_pred_train)
        test_r2 = r2_score(y_test, y_pred_test)

        print(f"{name} results for {target_name}:")
        print(f"  Train MAE : {train_mae:.4f} | Test MAE : {test_mae:.4f}")
        print(f"  Train RMSE: {train_rmse:.4f} | Test RMSE: {test_rmse:.4f}")
        print(f"  Train R²  : {train_r2:.4f} | Test R²  : {test_r2:.4f}")

        results.append({
            'Target': target_name,
            'Model': name,
            'Train_MAE': train_mae,
            'Test_MAE': test_mae,
            'Train_RMSE': train_rmse,
            'Test_RMSE': test_rmse,
            'Train_R2': train_r2,
            'Test_R2': test_r2
        })

        trained_pipelines[name] = {
            'pipeline': pipe,
            'y_test': y_test,
            'y_pred_test': y_pred_test
        }

    results_df = pd.DataFrame(results)
    return results_df, trained_pipelines
```

[18]:
```
# 5.6a Train & evaluate models for PM2.5 only

X_pm25 = feature_df.copy()
y_pm25 = df[TARGET_PM25].copy()

results_pm25, models_pm25 = train_and_evaluate_models(X_pm25, y_pm25, "PM2.5")

print("\nPM2.5 model performance summary:")
display(results_pm25.sort_values('Test_R2', ascending=False))
```

```
======================================================================
Training models for target: PM2.5
======================================================================

--> Training Random Forest…
Random Forest results for PM2.5:
  Train MAE : 0.6320 | Test MAE : 1.6902
  Train RMSE: 1.9860 | Test RMSE: 4.7780
  Train R²  : 0.9974 | Test R²  : 0.9859
```

```
--> Training XGBoost…
XGBoost results for PM2.5:
  Train MAE : 1.4861 | Test MAE : 2.0442
  Train RMSE: 2.7531 | Test RMSE: 7.3413
  Train R²  : 0.9950 | Test R²  : 0.9668

--> Training KNN…
KNN results for PM2.5:
  Train MAE : 5.2697 | Test MAE : 6.2144
  Train RMSE: 9.7765 | Test RMSE: 11.7828
  Train R²  : 0.9369 | Test R²  : 0.9145

PM2.5 model performance summary:
```

	Target	Model	Train_MAE	Test_MAE	Train_RMSE	Test_RMSE	Train_R2	Test_R2
0	PM2.5	Random Forest	0.631951	1.690249	1.985980	4.778044	0.997396	0.985946
1	PM2.5	XGBoost	1.486146	2.044230	2.753138	7.341292	0.994996	0.966823
2	PM2.5	KNN	5.269686	6.214390	9.776484	11.782780	0.936901	0.914535

[19]:

```python
# 5.6b Train & evaluate models for AQI only

X_aqi = feature_df.copy()
y_aqi = df[TARGET_AQI].copy()

results_aqi, models_aqi = train_and_evaluate_models(X_aqi, y_aqi, "AQI")

print("\nAQI model performance summary:")
display(results_aqi.sort_values('Test_R2', ascending=False))
```

```
======================================================================
Training models for target: AQI
======================================================================

--> Training Random Forest…
Random Forest results for AQI:
  Train MAE : 0.0273 | Test MAE : 0.0724
  Train RMSE: 0.0705 | Test RMSE: 0.1838
  Train R²  : 0.9946 | Test R²  : 0.9637

--> Training XGBoost…
XGBoost results for AQI:
  Train MAE : 0.0802 | Test MAE : 0.0935
```

```
  Train RMSE: 0.1669 | Test RMSE: 0.1945
  Train R²  : 0.9699 | Test R²  : 0.9594

--> Training KNN…
KNN results for AQI:
  Train MAE : 0.2403 | Test MAE : 0.2835
  Train RMSE: 0.3497 | Test RMSE: 0.4072
  Train R²  : 0.8680 | Test R²  : 0.8219

AQI model performance summary:
  Target          Model  Train_MAE  Test_MAE  Train_RMSE  Test_RMSE  Train_R2  \
0    AQI  Random Forest   0.027322  0.072431    0.070466   0.183772  0.994637
1    AQI        XGBoost   0.080233  0.093512    0.166945   0.194550  0.969899
2    AQI            KNN   0.240338  0.283520    0.349661   0.407233  0.867952

    Test_R2
0  0.963732
1  0.959354
2  0.821907
```

[20]:

```python
# 5.6c Train & evaluate models for Temperature only

X_temp = feature_df.copy()
y_temp = df[TARGET_TEMP].copy()

results_temp, models_temp = train_and_evaluate_models(X_temp, y_temp,
 ↪"Temperature")

print("\nTemperature model performance summary:")
display(results_temp.sort_values('Test_R2', ascending=False))
```

```
======================================================================
Training models for target: Temperature
======================================================================

--> Training Random Forest…
Random Forest results for Temperature:
  Train MAE : 0.0020 | Test MAE : 0.0055
  Train RMSE: 0.0070 | Test RMSE: 0.0197
  Train R²  : 1.0000 | Test R²  : 1.0000

--> Training XGBoost…
XGBoost results for Temperature:
  Train MAE : 0.0369 | Test MAE : 0.0538
  Train RMSE: 0.0696 | Test RMSE: 0.2015
  Train R²  : 0.9999 | Test R²  : 0.9995
```

```
--> Training KNN…
KNN results for Temperature:
  Train MAE : 1.2003 | Test MAE : 1.4100
  Train RMSE: 1.6094 | Test RMSE: 1.8897
  Train R²  : 0.9675 | Test R²  : 0.9559

Temperature model performance summary:

        Target          Model  Train_MAE  Test_MAE  Train_RMSE  Test_RMSE  \
0  Temperature  Random Forest   0.001988  0.005495    0.007040   0.019743
1  Temperature        XGBoost   0.036943  0.053782    0.069564   0.201541
2  Temperature            KNN   1.200251  1.409951    1.609408   1.889741

   Train_R2   Test_R2
0  0.999999  0.999995
1  0.999939  0.999498
2  0.967501  0.955871
```

[21]:

```python
# 5.6d Combine results and models into unified structures

all_results_df = pd.concat(
    [results_pm25, results_aqi, results_temp],
    ignore_index=True
)

all_trained_models = {
    "PM2.5": models_pm25,
    "AQI": models_aqi,
    "Temperature": models_temp
}

print("\nOverall model performance summary:")
display(all_results_df.sort_values(['Target', 'Test_R2'], ascending=[True,␣
↪False]))
```

```
Overall model performance summary:

        Target          Model  Train_MAE  Test_MAE  Train_RMSE  Test_RMSE  \
3          AQI  Random Forest   0.027322  0.072431    0.070466   0.183772
4          AQI        XGBoost   0.080233  0.093512    0.166945   0.194550
5          AQI            KNN   0.240338  0.283520    0.349661   0.407233
0        PM2.5  Random Forest   0.631951  1.690249    1.985980   4.778044
1        PM2.5        XGBoost   1.486146  2.044230    2.753138   7.341292
2        PM2.5            KNN   5.269686  6.214390    9.776484  11.782780
6  Temperature  Random Forest   0.001988  0.005495    0.007040   0.019743
7  Temperature        XGBoost   0.036943  0.053782    0.069564   0.201541
8  Temperature            KNN   1.200251  1.409951    1.609408   1.889741
```

```
   Train_R2   Test_R2
3  0.994637  0.963732
4  0.969899  0.959354
5  0.867952  0.821907
0  0.997396  0.985946
1  0.994996  0.966823
2  0.936901  0.914535
6  0.999999  0.999995
7  0.999939  0.999498
8  0.967501  0.955871
```

1.4.2 5.7 Selecting the Best Models and Saving Them

We now select, for each target, the **model with the highest Test R²** and persist it as a production-ready artifact using `joblib`.
We save the full **pipeline (preprocessing + model)** so the same transformations are consistently applied at inference time.

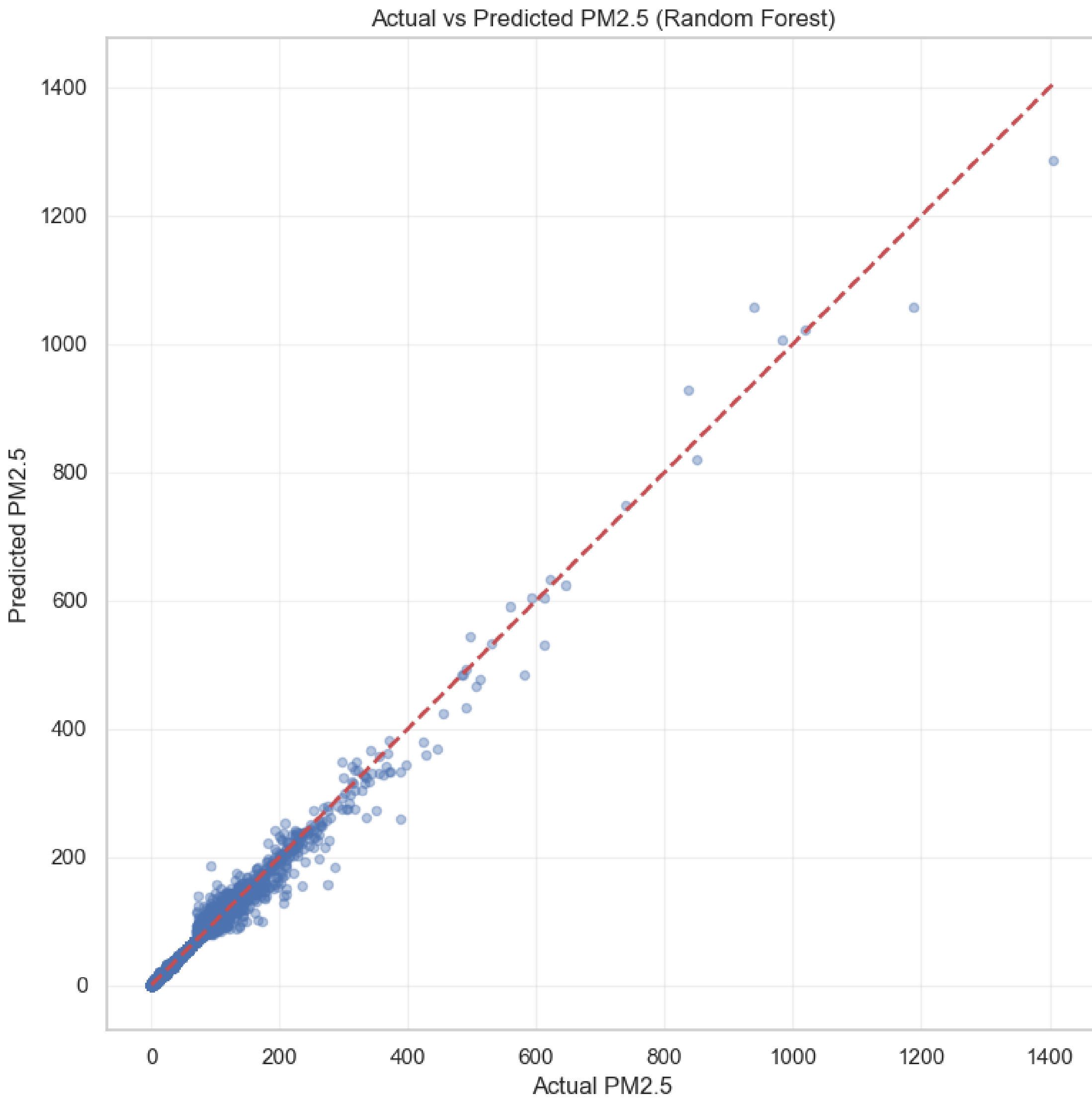
[22]:
```
best_models = {}

for target_name in targets.keys():
    target_results = all_results_df[all_results_df['Target'] == target_name]
    best_row = target_results.loc[target_results['Test_R2'].idxmax()]
    best_model_name = best_row['Model']
    best_models[target_name] = best_model_name

    print(f"For target '{target_name}', best model is: {best_model_name} "
          f"(Test R² = {best_row['Test_R2']:.4f}, Test RMSE =␣
 ↪{best_row['Test_RMSE']:.4f})")

    best_pipeline = all_trained_models[target_name][best_model_name]['pipeline']
    file_name = f"best_model_{target_name.lower()}.pkl".replace(" ", "_")
    joblib.dump(best_pipeline, file_name)
    print(f"Saved pipeline to {file_name}\n")
```

```
For target 'PM2.5', best model is: Random Forest (Test R² = 0.9859, Test RMSE =
4.7780)
Saved pipeline to best_model_pm2.5.pkl

For target 'AQI', best model is: Random Forest (Test R² = 0.9637, Test RMSE =
0.1838)
Saved pipeline to best_model_aqi.pkl

For target 'Temperature', best model is: Random Forest (Test R² = 1.0000, Test
RMSE = 0.0197)
Saved pipeline to best_model_temperature.pkl
```

1.4.3 5.8 Example: Actual vs Predicted for PM2.5 (Best Model)

We visualize how well the best model predicts PM2.5 by plotting **actual vs predicted** values.

[23]:
```
# Plot actual vs predicted for PM2.5 best model

pm25_best_model_name = best_models['PM2.5']
pm25_best = all_trained_models['PM2.5'][pm25_best_model_name]

y_test_pm25 = pm25_best['y_test']
y_pred_pm25 = pm25_best['y_pred_test']

plt.figure(figsize=(8, 8))
plt.scatter(y_test_pm25, y_pred_pm25, alpha=0.4, s=20)
min_val = min(y_test_pm25.min(), y_pred_pm25.min())
max_val = max(y_test_pm25.max(), y_pred_pm25.max())
plt.plot([min_val, max_val], [min_val, max_val], 'r--', lw=2)
plt.xlabel("Actual PM2.5")
plt.ylabel("Predicted PM2.5")
plt.title(f"Actual vs Predicted PM2.5 ({pm25_best_model_name})")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



1.5 6. Predictive Country Ranking (Future Forecasting)

In this section, we extend the project beyond historical EDA by generating **future predictions** using our trained ML models.

The goal is to estimate **which countries are likely to be the most polluted or hottest in the near future**, based on learned historical weather and air-quality patterns.

1.5.1 How the forecasting works

1. **Duplicate current feature data** (one row per city/observation).

2. **Simulate future conditions** by shifting the hour-of-day feature (`last_updated_hour`) forward.

- This represents a simplified forward-time projection using the same feature patterns the model was trained on.

3. Use the fully saved ML pipelines to predict:
   - Future PM2.5 (`pred_PM25`)
   - Future AQI (`pred_AQI`)
   - Future Temperature (`pred_Temp`)
4. **Aggregate predictions by country** to compute a projected average pollution/temperature score.
5. Produce **Predicted Top 10 and Bottom 10 Countries** for:
   - PM2.5
   - AQI
   - Temperature

This creates a **forward-looking environmental risk assessment** that can help policymakers and analysts anticipate where conditions may be worst or improving.

[25]:
```python
# ==============================
# 6. Predictive Country Ranking
# ==============================

import joblib

# Load saved production models
model_pm25 = joblib.load("best_model_pm2.5.pkl")
model_aqi = joblib.load("best_model_aqi.pkl")
model_temp = joblib.load("best_model_temperature.pkl")

# Create a copy for future simulation
future_df = feature_df.copy()

# --- Simulate future time features ---
future_df['last_updated_hour'] = df['last_updated_hour']
future_df['last_updated_minute'] = df['last_updated_minute']

future_df['future_hour_shifted'] = (df['last_updated_hour'] + 6) % 24
future_df['last_updated_hour'] = future_df['future_hour_shifted']
future_df.drop(columns=['future_hour_shifted'], inplace=True)

# ===========================================
# 6.1 Predict PM2.5, AQI, Temperature
# ===========================================
```

```python
future_df['pred_PM25'] = model_pm25.predict(future_df)
future_df['pred_AQI'] = model_aqi.predict(future_df)
future_df['pred_Temp'] = model_temp.predict(future_df)

# ==========================================
# 6.2 Aggregate country-level predictions
# ==========================================

pred_pm25_country = future_df.groupby("country")['pred_PM25'].mean()
pred_aqi_country = future_df.groupby("country")['pred_AQI'].mean()
pred_temp_country = future_df.groupby("country")['pred_Temp'].mean()

# ==========================================
# 6.3 Top / Bottom 10 rankings (Predicted)
# ==========================================

def top_bottom(series):
    return series.sort_values(ascending=False).head(10), series.sort_values().
 ↪head(10)

pm25_top_future, pm25_bottom_future = top_bottom(pred_pm25_country)
aqi_top_future, aqi_bottom_future = top_bottom(pred_aqi_country)
temp_top_future, temp_bottom_future = top_bottom(pred_temp_country)

print("\n=== Predicted Top 10 PM2.5 Countries (Future) ===")
display(pm25_top_future.to_frame("Predicted_PM2.5"))

print("\n=== Predicted Bottom 10 PM2.5 Countries (Future) ===")
display(pm25_bottom_future.to_frame("Predicted_PM2.5"))

print("\n=== Predicted Top 10 AQI Countries (Future) ===")
display(aqi_top_future.to_frame("Predicted_AQI"))

print("\n=== Predicted Bottom 10 AQI Countries (Future) ===")
display(aqi_bottom_future.to_frame("Predicted_AQI"))

print("\n=== Predicted Top 10 Temperature Countries (Future) ===")
display(temp_top_future.to_frame("Predicted_Temp"))

print("\n=== Predicted Bottom 10 Temperature Countries (Future) ===")
display(temp_bottom_future.to_frame("Predicted_Temp"))
```

```
=== Predicted Top 10 PM2.5 Countries (Future) ===

              Predicted_PM2.5
country
```

```
Chile               177.531602
Saudi Arabia        140.773086
China               136.601890
India               111.545970
Kuwait               97.472221
Indonesia            93.517034
Bahrain              71.645213
Mauritania           71.097183
Südkorea             69.406692
Bangladesh           68.277793

=== Predicted Bottom 10 PM2.5 Countries (Future) ===
                                  Predicted_PM2.5
country
                                     0.500000
Malásia                                  1.724333
Saint-Vincent-et-les-Grenadines          1.864417
Bélgica                                  2.027500
                                      2.212917
Letonia                                  2.271833
Polônia                                  2.348167
                                        2.828333
Komoren                                  2.858333
Solomon Islands                          3.039062

=== Predicted Top 10 AQI Countries (Future) ===
                      Predicted_AQI
country
China                      4.120811
Südkorea                   3.983333
Saudi Arabia               3.950267
India                      3.867838
Chile                      3.849777
Kuwait                     3.721705
Bahrain                    3.379536
Malaysia                   3.208850
United Arab Emirates       3.190529
Qatar                      3.167510

=== Predicted Bottom 10 AQI Countries (Future) ===
                                  Predicted_AQI
country
                                           1.0
Polônia                                     1.0
                                        1.0
```

```
Colombia                                  1.0
Letonia                                   1.0
Saint-Vincent-et-les-Grenadines           1.0
Bélgica                                   1.0
Mexique                                   1.0
Togo                                      1.0
Turkménistan                              1.0

=== Predicted Top 10 Temperature Countries (Future) ===
                      Predicted_Temp
country
Saudi Arabien              45.052500
Marrocos                   40.300833
Turkménistan               37.821667
Qatar                      34.231456
United Arab Emirates       34.053638
                        34.000833
Kuwait                     33.855100
Saudi Arabia               33.473351
Djibouti                   32.652576
Oman                       32.436857

=== Predicted Bottom 10 Temperature Countries (Future) ===
                          Predicted_Temp
country
Iceland                         6.486499
Mongolia                        6.771781
Canada                          7.603425
United States of America        9.180424
Norway                          9.511554
Chile                           9.978663
Ecuador                        10.370617
Finland                        11.372277
Kazakhstan                     11.497402
Estonia                        11.504084
```

1.6 7. Export Final Dataset with Predictions for Dashboard

To feed our **Tableau / Power BI dashboard**, we export a **single, consolidated CSV** that contains:

- All original columns from the raw dataset
- All engineered time and encoded features
- All prediction columns from the future forecasting step:

– pred_PM25

– pred_AQI

– pred_Temp

This CSV can be directly used as a data source for interactive dashboards.

[28]:
```
# ============================================================
# 7A. Create ONE Combined CSV for All Top/Bottom 10 Rankings
# ============================================================

combined_rows = []

def add_block(series, metric_name, category_name):
    df_block = series.to_frame(name="value").reset_index()
    df_block["metric"] = metric_name
    df_block["category"] = category_name
    return df_block

# ---------- Historical ----------
combined_rows.append(add_block(pm25_top_hist, "PM2.5", "Historical Top 10"))
combined_rows.append(add_block(pm25_bottom_hist, "PM2.5", "Historical Bottom␣
↪10"))

combined_rows.append(add_block(aqi_top_hist, "AQI", "Historical Top 10"))
combined_rows.append(add_block(aqi_bottom_hist, "AQI", "Historical Bottom 10"))

combined_rows.append(add_block(temp_top_hist, "Temperature", "Historical Top␣
↪10"))
combined_rows.append(add_block(temp_bottom_hist, "Temperature", "Historical␣
↪Bottom 10"))

# ---------- Predicted ----------
combined_rows.append(add_block(pm25_top_future, "PM2.5", "Predicted Top 10"))
combined_rows.append(add_block(pm25_bottom_future, "PM2.5", "Predicted Bottom␣
↪10"))

combined_rows.append(add_block(aqi_top_future, "AQI", "Predicted Top 10"))
combined_rows.append(add_block(aqi_bottom_future, "AQI", "Predicted Bottom 10"))

combined_rows.append(add_block(temp_top_future, "Temperature", "Predicted Top␣
↪10"))
combined_rows.append(add_block(temp_bottom_future, "Temperature", "Predicted␣
↪Bottom 10"))

# Combine all blocks
```

```python
combined_df = pd.concat(combined_rows, ignore_index=True)

# Save to CSV
export_path_combined = "/Users/ayushgawai/Downloads/top_bottom_all_combined.csv"
combined_df.to_csv(export_path_combined, index=False)

print("Combined Top/Bottom 10 CSV saved at:")
print(export_path_combined)

combined_df.head()
```

```
Combined Top/Bottom 10 CSV saved at:
/Users/ayushgawai/Downloads/top_bottom_all_combined.csv
```

[28]:

	country	value	metric	category
0	Chile	178.947781	PM2.5	Historical Top 10
1	Saudi Arabia	140.220979	PM2.5	Historical Top 10
2	China	137.501838	PM2.5	Historical Top 10
3	India	110.518835	PM2.5	Historical Top 10
4	Kuwait	98.752979	PM2.5	Historical Top 10

[30]:

```python
# ============================================
# 7. Export Final Dataset for Dashboard (CSV)
# ============================================

# Merge predictions back with original df aligned by index
final_export_df = df.copy()

# Add the predictions generated in the previous forecasting step
final_export_df['pred_PM25'] = future_df['pred_PM25']
final_export_df['pred_AQI'] = future_df['pred_AQI']
final_export_df['pred_Temp'] = future_df['pred_Temp']

print("Columns in final export dataset:", len(final_export_df.columns))

# Save as CSV (update path if not in Colab)
export_path = "/Users/ayushgawai/Downloads/
↪Global_Weather_Final_With_Predictions.csv"
final_export_df.to_csv(export_path, index=False)

print(f"\nFinal dataset with predictions saved to:\n{export_path}")
```

```
Columns in final export dataset: 54

Final dataset with predictions saved to:
/Users/ayushgawai/Downloads/Global_Weather_Final_With_Predictions.csv
```

1.7 8. Using the Saved Models for Prediction (Production Perspective)

In a production setting, an application can: 1. Load the saved pipeline (`preprocessing + model`) for the desired target.
2. Pass a **single row or batch of new observations** with the same schema as `feature_df`.
3. Get predictions for PM2.5, AQI, or temperature.

[31]:
```
def predict_pm25(new_data: pd.DataFrame):
    """
    Predict PM2.5 for new observations.

    Parameters
    ----------
    new_data : pd.DataFrame
        DataFrame with the same feature columns as `feature_df`.

    Returns
    -------
    np.ndarray
        Predicted PM2.5 values.
    """
    pipeline = joblib.load("best_model_pm2.5.pkl".replace(" ", "_"))
    return pipeline.predict(new_data)

# Example (using a few rows from the dataset as if they were 'new' data)
example_new_data = feature_df.head(5)
predict_pm25(example_new_data)
```

[31]:
```
array([  8.44583333,   1.23479167,  10.1525    ,   0.7681    ,
       193.8987    ])
```

1.8 9. Summary of EDA, Visualizations, and Modeling

1.8.1 EDA Performed

- **Structural EDA:** dataset shape, column types, missing values, and descriptive statistics.
- **Univariate analysis:** distributions of temperature, humidity, precipitation, PM2.5, PM10, and AQI.
- **Bivariate analysis:** scatterplots between environmental variables; correlation heatmap among weather and air-quality metrics.
- **Country-level analysis (historical):** top and bottom 10 countries by average **PM2.5**, **AQI**, and **temperature**.
- **Temporal and spatial patterns:** average PM2.5/AQI/temperature by hour of day and a global scatter of PM2.5 over latitude/longitude.

1.8.2 Visualizations Included

- Histograms and KDE plots for continuous variables.
- Scatterplots for relationships between temperature, humidity, and pollution.
- Heatmap for correlation structure.
- Bar charts for **top/bottom 10 countries** (historical).
- Time-of-day line plots for hourly patterns.
- Geographic scatter plot for PM2.5 distribution.

1.8.3 Machine Learning Work

- Built **regression models** for three targets:
  - PM2.5 (fine particulate matter)
  - AQI (US EPA index)
  - Temperature in °C
- Models evaluated: **Random Forest, XGBoost, and KNN** for each target.
- Used a robust pipeline:
  - Proper feature/target separation (no leakage).
  - `ColumnTransformer` to handle numeric vs categorical features.
  - `SimpleImputer` for missing values and `StandardScaler` for numeric features.
- Chose the **best model for each target** based on **Test R^2 and RMSE**, then saved the full preprocessing + model pipeline with `joblib`.

1.8.4 Predictive Country Ranking

- Using the saved models, simulated **future conditions** by shifting time features.
- Predicted future PM2.5, AQI, and temperature for all observations.
- Aggregated predictions by country and produced **Predicted Top/Bottom 10 countries** for each metric.
- Exported a **single, enriched CSV** with both historical data and predicted values to support dashboards.

This notebook is structured so that **both technical audiences (data scientists)** and **non-technical stakeholders** can follow the logic, see the visual evidence, and understand why the models were built and how they might be used in real-world decision-making.

```
[ ]:
```