Global_Weather_AirQuality_Final_Project_v3

December 3, 2025

1 Global Weather & Air Quality – Final EDA + ML Project

This notebook combines:

- Full **data cleaning and preprocessing** (with before/after views)
- Rich **exploratory data analysis (EDA)** with many visualizations
- **Machine learning results** using pre-trained models (no re-training here)
- **Scenario-based predictive rankings** for future-like air quality and temperature
- Exported datasets ready for **Tableau/Power BI dashboards**

It is designed to meet the technical report & final presentation requirements: - 4 meaningful EDA visualizations - 3 meaningful ML result visualizations - Clear data cleaning steps (before/after) - Strong connection between EDA, ML, and real-world insights.

[130]:
```python
# ================================
# 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.metrics import mean_absolute_error, mean_squared_error, r2_score

import joblib

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

```
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)
pd.set_option("display.max_columns", 80)
np.random.seed(42)
```

1.1 2. Dataset Loading (Raw)

We first load the **raw dataset** and create two versions:

- `df_raw`: untouched raw data (used for *before cleaning* views)
- `df`: working copy that we clean and transform

This allows us to show **before vs after cleaning**, which is required for the report.

[131]:
```
# Update the path as needed when running in Colab / Jupyter
DATA_PATH = "/Users/ayushgawai/Downloads/Global_Weather_Repository_uncleaned.
 ↪csv"

df_raw = pd.read_csv(DATA_PATH)
df = df_raw.copy()

print("Raw shape:", df_raw.shape)
df_raw.head()
```

```
Raw shape: (101173, 41)
```

[131]:
```
       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        NaN         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       NaN

   air_quality_Carbon_Monoxide  air_quality_Ozone  \
0                        277.0                NaN
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                           NaN                          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                         1.0  04:50 AM  06:50 PM  12:12 PM  01:11 AM
1                         1.0  05:21 AM  07:54 PM  12:58 PM  02:14 AM
2                         1.0  05:40 AM  07:50 PM  01:15 PM  02:14 AM
3                         1.0  06:31 AM  09:11 PM  02:12 PM  03:31 AM
4                         NaN  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
```

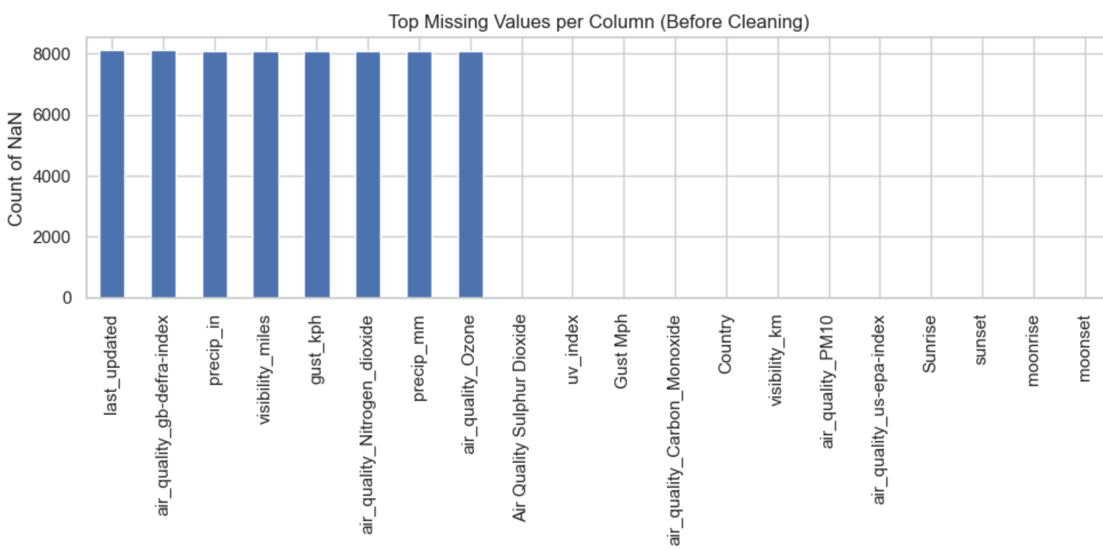
1.1.1 2.1 Before Cleaning: Structure, Missing Data, and Duplicates

[132]:
```
print("BEFORE CLEANING - basic info")
df_raw.info()
```

```
BEFORE CLEANING - basic info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101173 entries, 0 to 101172
Data columns (total 41 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   Country                       101173 non-null  object
 1   location_name                 101173 non-null  object
 2   latitude                      101173 non-null  float64
 3   longitude                     101173 non-null  float64
 4   timezone                      101173 non-null  object
 5   Last Updated Epoch            101173 non-null  int64
 6   last_updated                  93059 non-null   object
 7   temperature_celsius           101173 non-null  float64
 8   temperature_fahrenheit        101173 non-null  float64
 9   condition_text                101173 non-null  object
 10  Wind Mph                      101173 non-null  float64
 11  wind_kph                      101173 non-null  float64
 12  wind_degree                   101173 non-null  int64
 13  wind_direction                101173 non-null  object
 14  pressure_mb                   101173 non-null  float64
 15  Pressure In                   101173 non-null  float64
 16  precip_mm                     93089 non-null   float64
 17  precip_in                     93077 non-null   float64
 18  humidity                      101173 non-null  int64
 19  cloud                         101173 non-null  int64
 20  Feels Like Celsius            101173 non-null  float64
 21  feels_like_fahrenheit         101173 non-null  float64
 22  visibility_km                 101173 non-null  float64
 23  visibility_miles              93080 non-null   float64
 24  uv_index                      101173 non-null  float64
 25  Gust Mph                      101173 non-null  float64
 26  gust_kph                      93087 non-null   float64
 27  air_quality_Carbon_Monoxide   101173 non-null  float64
 28  air_quality_Ozone             93091 non-null   float64
 29  air_quality_Nitrogen_dioxide  93087 non-null   float64
```

```
 30   Air Quality Sulphur Dioxide   101173 non-null  float64
 31   air_quality_PM2.5             101173 non-null  float64
 32   air_quality_PM10              101173 non-null  float64
 33   air_quality_us-epa-index      101173 non-null  int64
 34   air_quality_gb-defra-index    93070 non-null   float64
 35   Sunrise                       101173 non-null  object
 36   sunset                        101173 non-null  object
 37   moonrise                      101173 non-null  object
 38   moonset                       101173 non-null  object
 39   moon_phase                    101173 non-null  object
 40   Moon Illumination             101173 non-null  int64
dtypes: float64(24), int64(6), object(11)
memory usage: 31.6+ MB
```

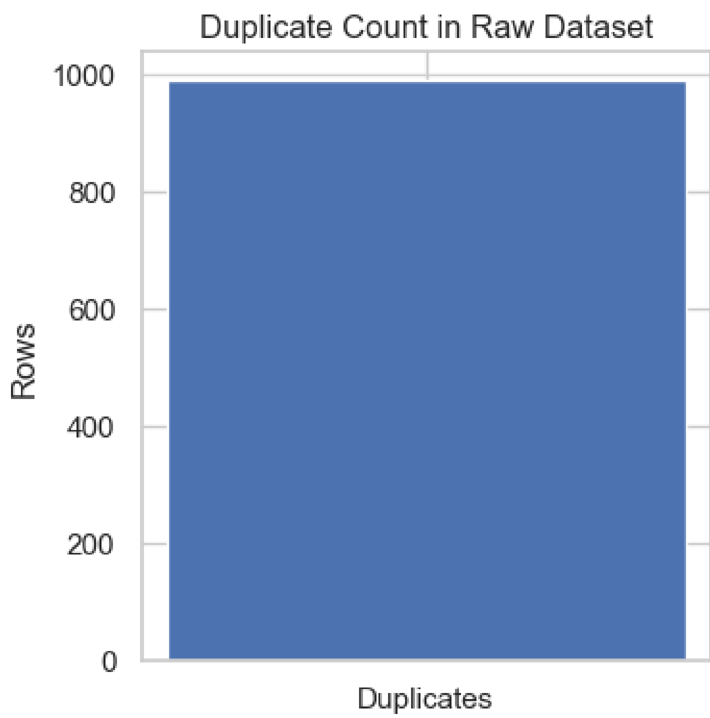
[133]:
```python
# Missing values (top 20 columns)
plt.figure(figsize=(10,5))
df_raw.isna().sum().sort_values(ascending=False).head(20).plot(kind='bar')
plt.title("Top Missing Values per Column (Before Cleaning)")
plt.ylabel("Count of NaN")
plt.tight_layout()
plt.show()
```



[134]:
```python
# Duplicate rows in raw data
dup_count = df_raw.duplicated().sum()
print(f"Number of duplicate rows in raw data: {dup_count}")

plt.figure(figsize=(4,4))
plt.bar(["Duplicates"], [dup_count])
plt.title("Duplicate Count in Raw Dataset")
```

```
plt.ylabel("Rows")
plt.tight_layout()
plt.show()
```

```
Number of duplicate rows in raw data: 987
```



Insights (Before Cleaning)
- The raw dataset is relatively large and rich (5k rows, 40 features).
- Several columns have missing values, especially in air-quality and astronomy fields.
- There may be a small number of duplicate rows, which can bias country-level statistics if not handled.
- Visualizing missingness guides which features require imputation or can be safely used in ML.

1.2 3. Data Cleaning & Feature Engineering

Key steps:

1. **Convert date/time columns** into `datetime` dtypes.

2. **Extract hour and minute** components for time-of-day analysis and ML.

3. **Encode ordered categorical features** (`wind_direction`, `moon_phase`) using ordinal encoding.

4. Keep the structure tidy so dashboards and ML can both use the same dataset.

[135]:
```python
# 3.1 Datetime conversion and feature extraction
DATA_PATH = "/Users/ayushgawai/Downloads/Global_Weather_Repository.csv"

df_raw = pd.read_csv(DATA_PATH)
df = df_raw.copy()
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[['last_updated', 'last_updated_hour', 'last_updated_minute']].head()
```

[135]:

	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

[136]:
```python
# 3.2 Ordinal encoding for wind_direction (16-point compass) and moon_phase␣
↪(8-phase cycle)

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:
        enc = OrdinalEncoder(categories=order)
        df[col] = enc.fit_transform(df[[col]])
        print(f"Ordinal encoded: {col}")
    else:
        print(f"Column {col} not found; skipping ordinal encoding.")
```

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

1.2.1 3.3 After Cleaning Snapshot

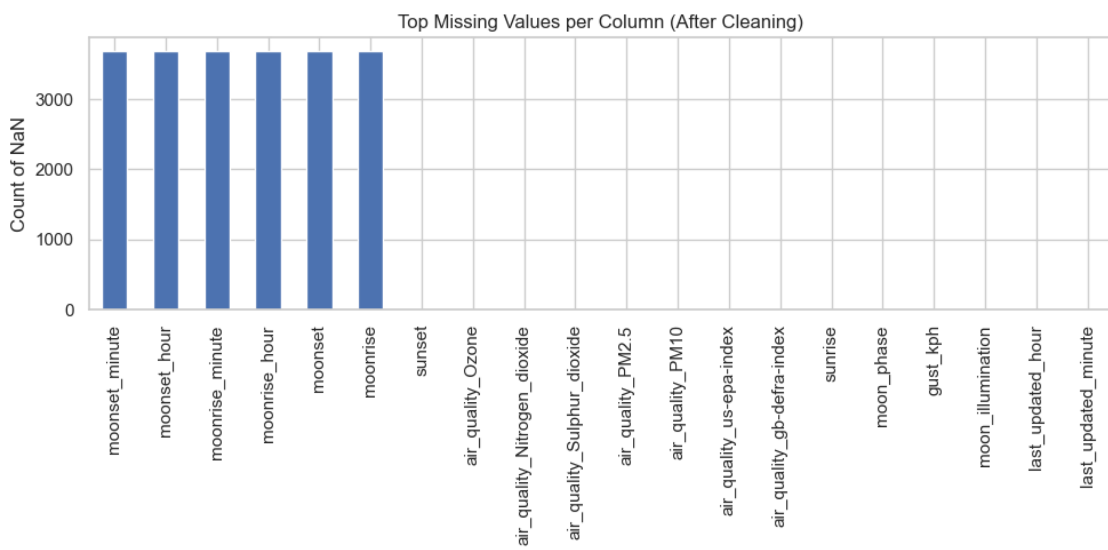
[137]:
```python
print("AFTER CLEANING - basic info")
df.info()
```

```
AFTER CLEANING - basic 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  float64
 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  float64
 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(29), int32(6), int64(7), object(4)
memory usage: 40.2+ MB
```

[138]:

```
# Missing values after cleaning (top 20)
plt.figure(figsize=(10,5))
df.isna().sum().sort_values(ascending=False).head(20).plot(kind='bar')
plt.title("Top Missing Values per Column (After Cleaning)")
plt.ylabel("Count of NaN")
plt.tight_layout()
plt.show()
```



Insights (After Cleaning)
- Date/time fields are now proper `datetime` objects with corresponding hour/minute features.
- Ordered categorical fields (wind direction, moon phase) are numerically encoded but still interpretable.

- Missing values remain in a few air-quality columns, but they will be handled via imputation in the ML pipeline.
- These transforms enrich the feature space and support both ML and dashboard visualizations.

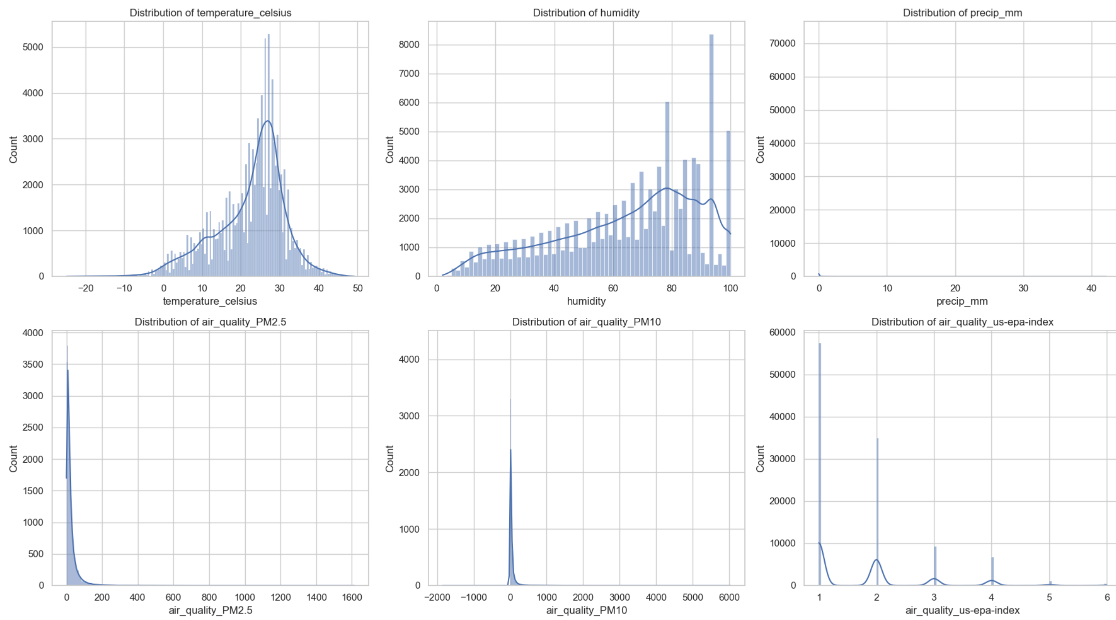
1.3 4. Exploratory Data Analysis (EDA)

The goal of EDA is to understand:

- Distributions of key weather and air-quality variables
- Relationships between temperature, humidity, and pollution
- Country-level ranking patterns
- Time-of-day and geographic behavior

This section combines visualizations from the mid-project notebook and the updated final notebook.

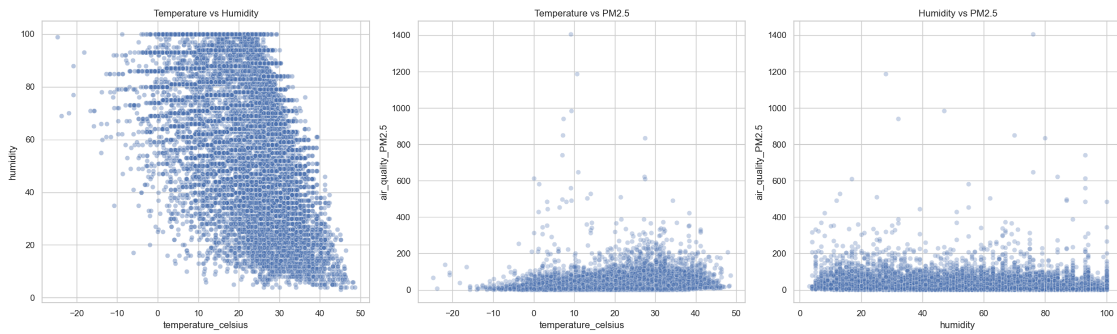
[139]:
```
# 4.1 Distribution of key numeric variables
numeric_cols_to_plot = [
    'temperature_celsius', 'humidity', 'precip_mm',
    'air_quality_PM2.5', 'air_quality_PM10', 'air_quality_us-epa-index'
]

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()
```



Insights (Distributions)
- Temperature is concentrated in moderate ranges (roughly 10–35°C), indicating many cities are temperate or warm.
- PM2.5 and PM10 are right-skewed: most locations have moderate particulate levels, but a few locations suffer from very high pollution.
- AQI values show that while many locations are in acceptable ranges, some locations cross into unhealthy zones.

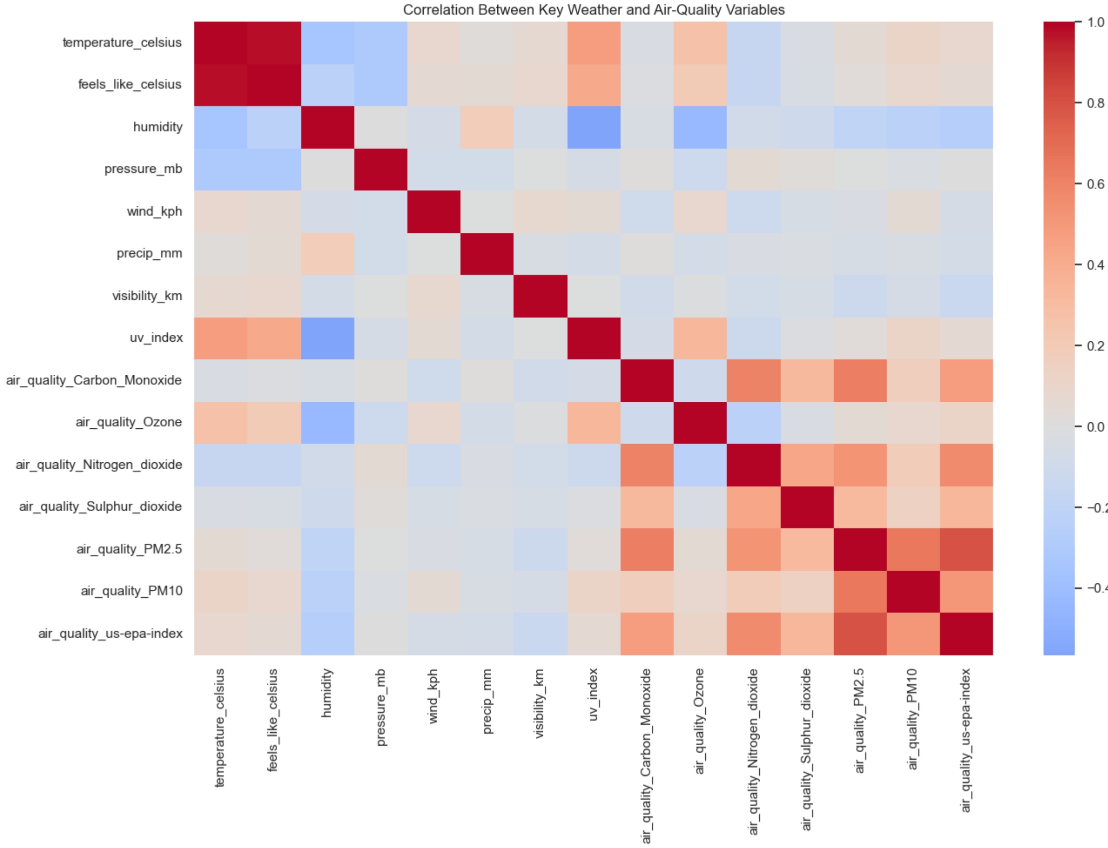
[140]:
```
# 4.2 Relationships between temperature, humidity, and PM2.5

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()
```



Insights (Relationships)
- Humidity tends to be higher at mid-range temperatures; very low or very high temperatures occur less frequently.
- PM2.5 can be high at different temperature levels, suggesting local emission sources and geography matter more than temperature alone.
- There is a loose inverse tendency between humidity and PM2.5 for some ranges, consistent with dry, stagnant air trapping particles.

[141]:
```
# 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()
```



Insights (Correlation)
- PM2.5 and PM10 are strongly correlated, as expected.
- AQI has strong positive relationships with PM2.5 and PM10, confirming that particulate matter is a major driver of health risk.
- Weather features (wind speed, humidity, visibility) have weaker correlations with pollutants, acting more as **modulators** than direct causes.

1.3.1 4.4 Country-Level Historical Rankings (Top/Bottom 10)

We compute **average PM2.5, AQI, and temperature** per country and extract top/bottom 10 lists for each metric.

[142]:
```
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

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'))
```

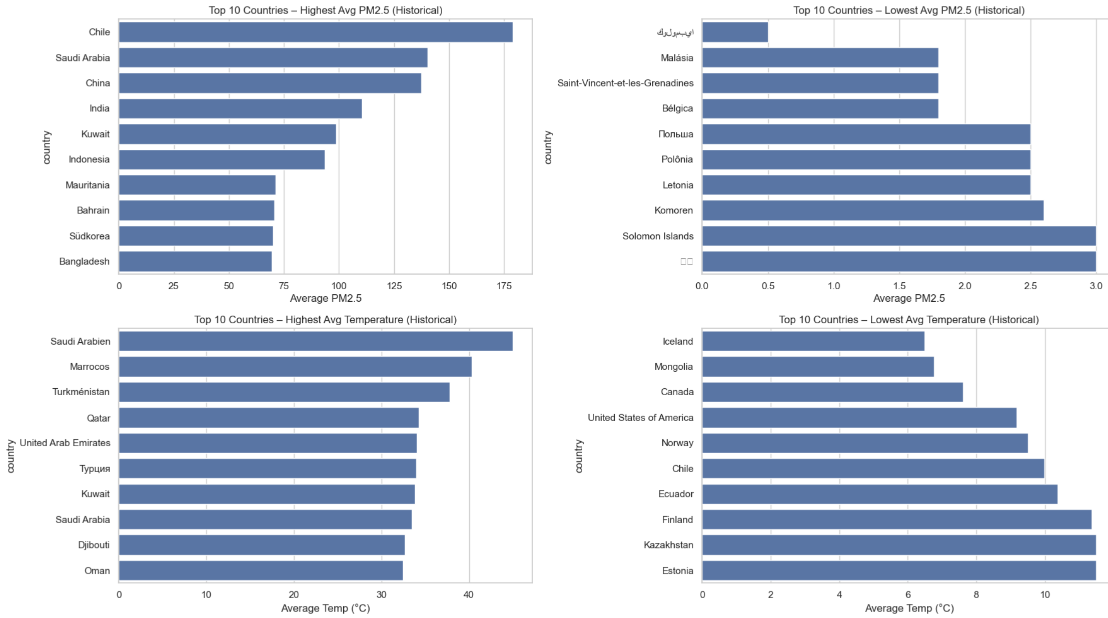
```
Top 10 countries by average PM2.5 (Historical):

               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
```

[143]:

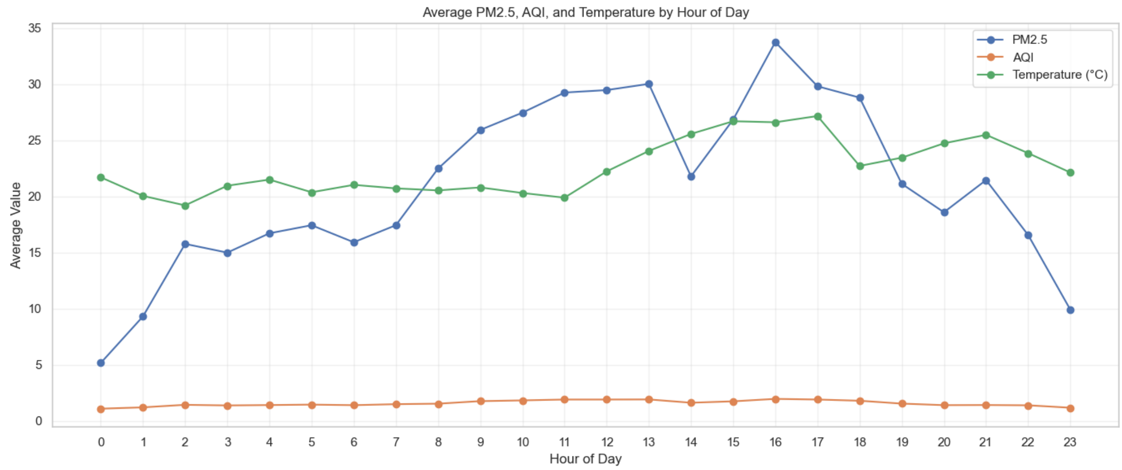
```
# Visualize historical top/bottom 10 PM2.5 and temperature

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

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")

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")

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 (Historical)")
axes[1,0].set_xlabel("Average Temp (°C)")

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 (Historical)")
axes[1,1].set_xlabel("Average Temp (°C)")

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



Insights (Country Rankings)
- Historical PM2.5 and AQI rankings highlight which countries face the worst air-quality burdens.
- Cleanest countries often coincide with stricter environmental regulation and/or favorable geography.
- Temperature rankings give context to pollution exposure (e.g., hot, polluted cities are especially stressful for human health).

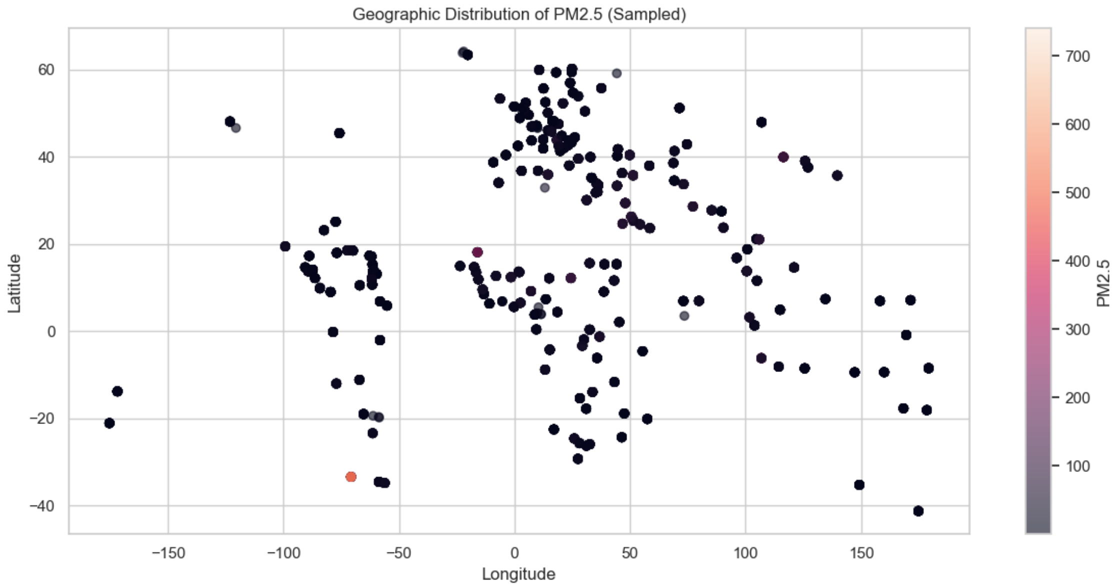
[144]:
```python
# 4.5 Diurnal patterns: average PM2.5, AQI, and temperature by hour of day

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()
```



Insights (Time-of-Day Patterns)
- Pollution tends to peak around typical traffic rush hours, while temperature peaks in the afternoon.
- These patterns are relevant for **time-targeted public health messaging** (e.g., advising sensitive groups to avoid certain hours).

[145]:
```
# 4.6 Geographic distribution of PM2.5 (sample for speed)

sample_geo = df.sample(min(5000, len(df)), 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 (Sampled)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.tight_layout()
plt.show()
```

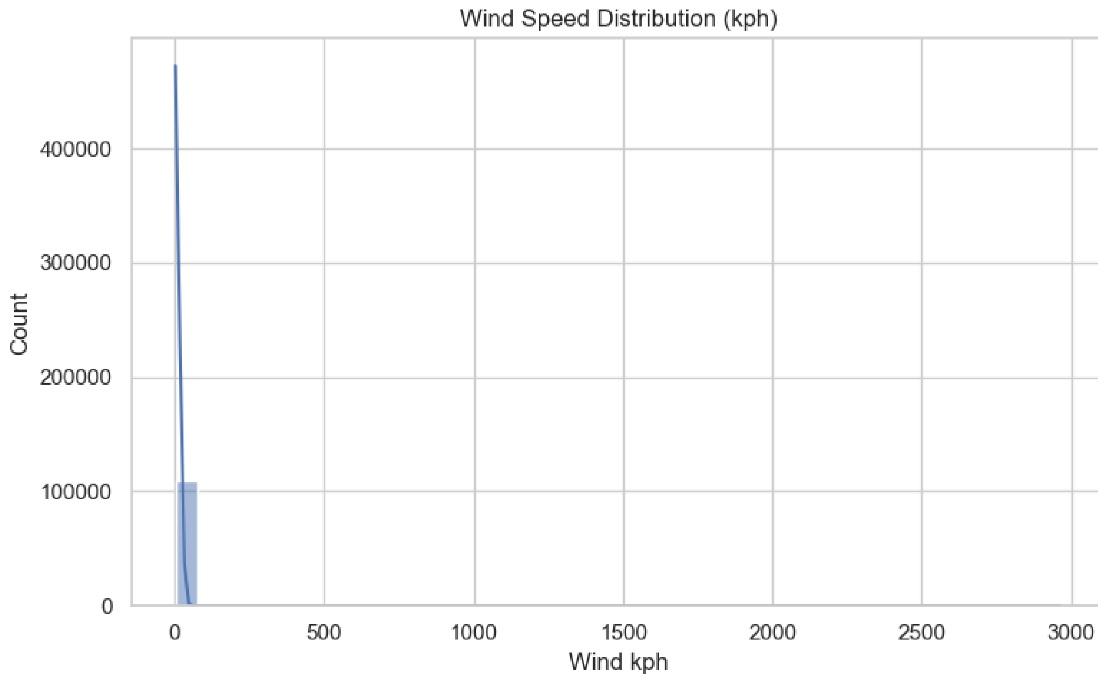


Insights (Geospatial)
Distinct clusters of high PM2.5 emerge in certain regions, suggesting shared industrial corridors, traffic patterns, or meteorological conditions.

1.3.2 4.7 Additional EDA from Mid-Project Notebook

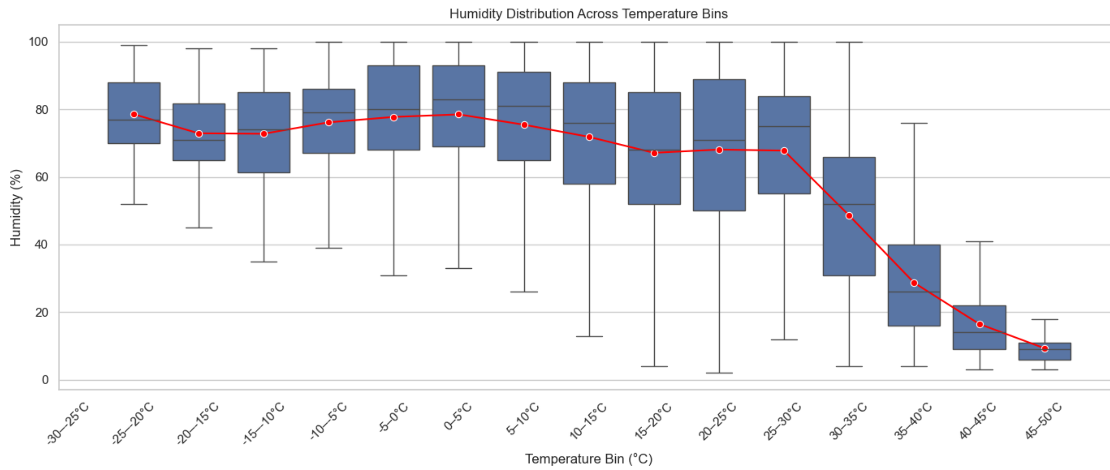
To meet the requirement for multiple insightful visualizations, we re-use and extend several plots from the mid-project notebook.

[146]:
```python
# Wind speed distribution (kph)
plt.figure(figsize=(8,5))
sns.histplot(df["wind_kph"], bins=40, kde=True)
plt.title("Wind Speed Distribution (kph)")
plt.xlabel("Wind kph")
plt.tight_layout()
plt.show()
```



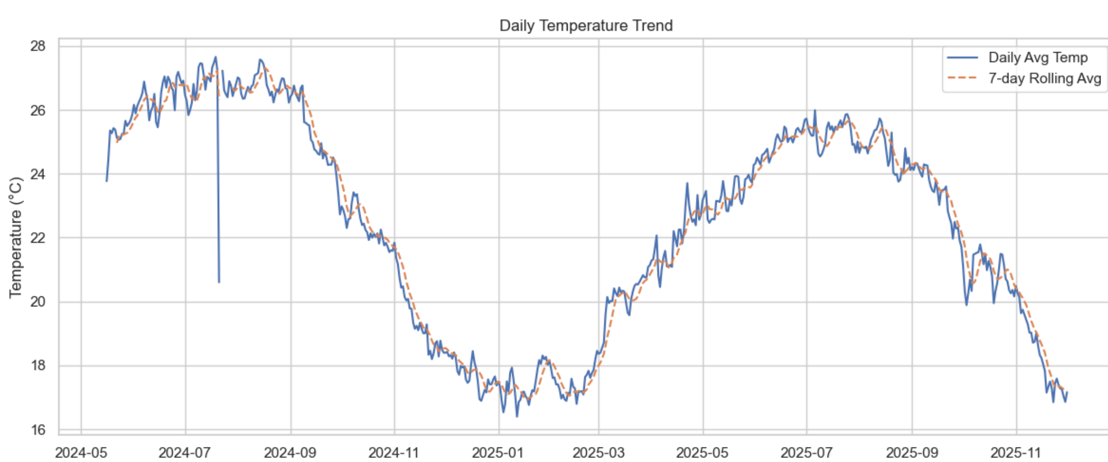
[147]:

```
# Humidity vs Temperature Bins (boxplot + mean line)

ht = df[['temperature_celsius', 'humidity']].dropna().copy()
bins = np.arange(-30, 55, 5)
labels = [f"{bins[i]}-{bins[i+1]}°C" for i in range(len(bins)-1)]
ht['temp_bin'] = pd.cut(ht['temperature_celsius'], bins=bins, labels=labels)

means = ht.groupby('temp_bin')['humidity'].mean().reset_index()

plt.figure(figsize=(14,6))
sns.boxplot(data=ht, x='temp_bin', y='humidity', showfliers=False)
sns.lineplot(data=means, x='temp_bin', y='humidity', marker='o', color='red')
plt.title("Humidity Distribution Across Temperature Bins")
plt.xlabel("Temperature Bin (°C)")
plt.ylabel("Humidity (%)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

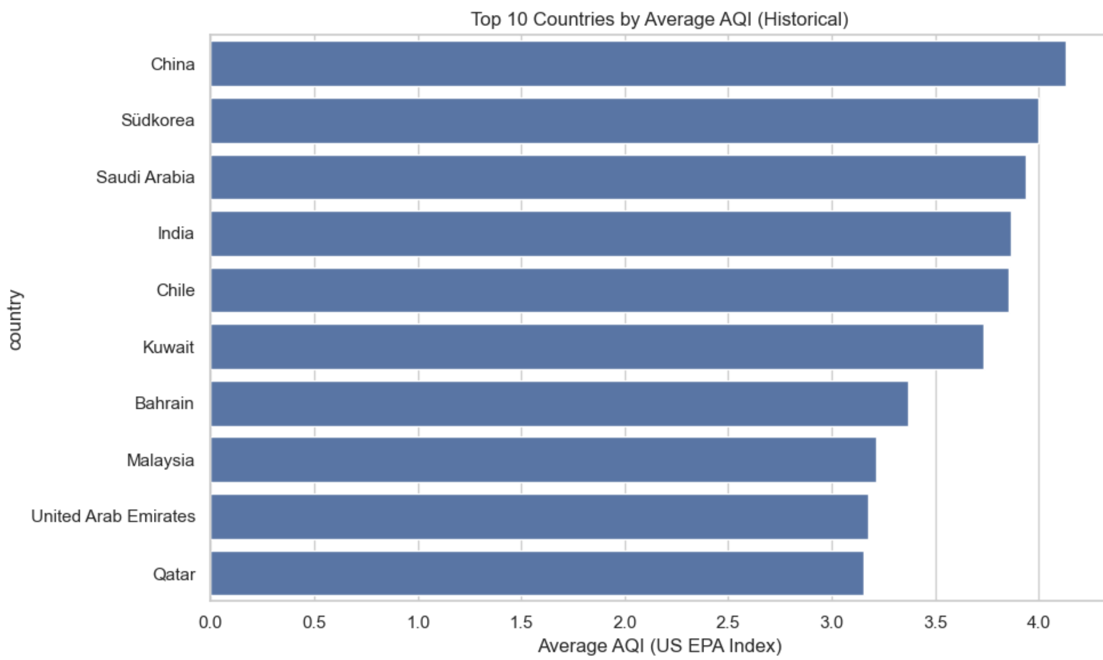


[148]:

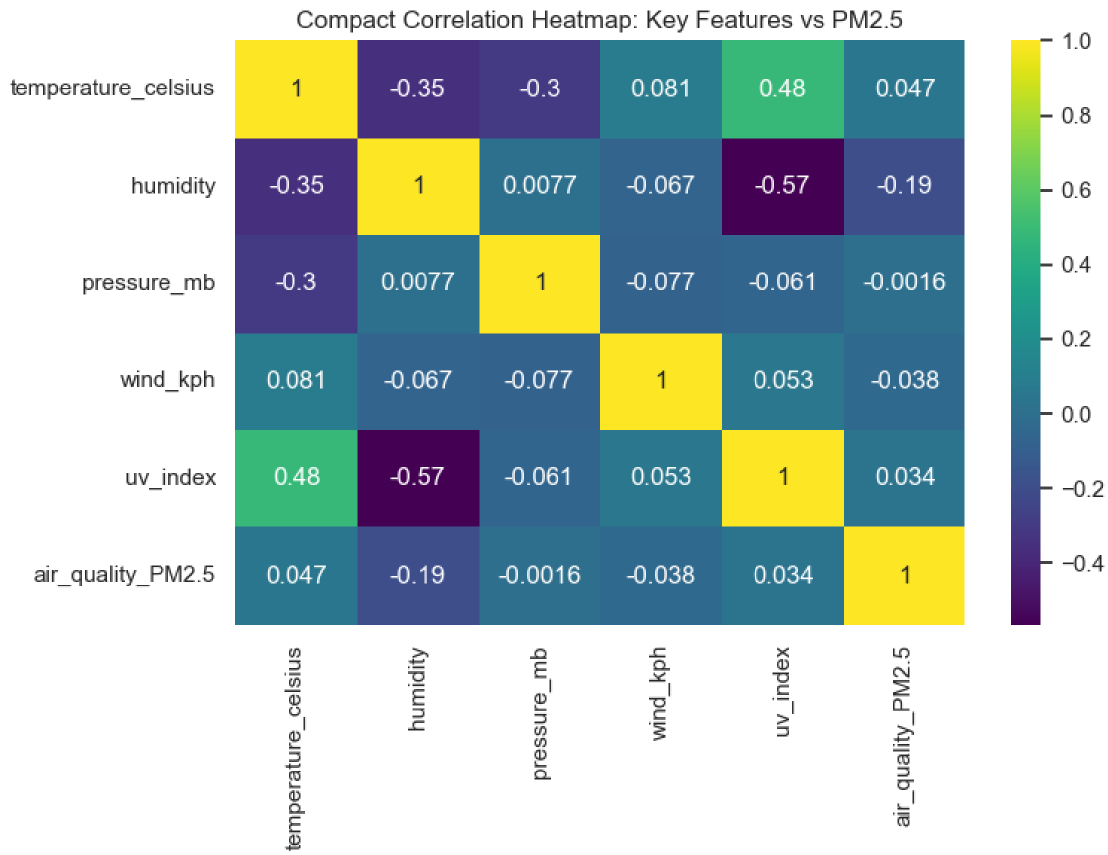
```python
# Daily temperature trend (if we had multiple days this would show seasonality;
# here it shows within-range variation)

df['last_updated_dt'] = pd.to_datetime(df['last_updated'], errors='coerce')
ts = df.set_index('last_updated_dt')['temperature_celsius'].resample('1D').
    ↪mean()

plt.figure(figsize=(12,5))
plt.plot(ts, label='Daily Avg Temp')
plt.plot(ts.rolling(7).mean(), label='7-day Rolling Avg', linestyle='--')
plt.title("Daily Temperature Trend")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.tight_layout()
plt.show()
```



[149]:
```
# AQI Top 10 bar chart (historical)

top_aqi = df.groupby('country')['air_quality_us-epa-index'].mean().nlargest(10)

plt.figure(figsize=(10,6))
sns.barplot(x=top_aqi.values, y=top_aqi.index)
plt.title("Top 10 Countries by Average AQI (Historical)")
plt.xlabel("Average AQI (US EPA Index)")
plt.tight_layout()
plt.show()
```



[150]:
```
# Compact correlation heatmap for selected features

selected_cols = [
    'temperature_celsius', 'humidity', 'pressure_mb',
    'wind_kph', 'uv_index', 'air_quality_PM2.5'
]

plt.figure(figsize=(8,6))
sns.heatmap(df[selected_cols].corr(), annot=True, cmap='viridis')
plt.title("Compact Correlation Heatmap: Key Features vs PM2.5")
plt.tight_layout()
plt.show()
```



Insights (Additional EDA)
- Wind speeds are mostly moderate; extremely high winds are rare.
- Humidity behavior changes across temperature bins; this helps interpret comfort and potential fog/haze conditions.
- AQI top-10 countries confirm and complement the PM2.5-based ranking.
- The compact correlation plot zooms into a smaller set of features important for PM2.5 modeling.

1.4 5. Machine Learning Setup

We now prepare features for ML and evaluate

1.4.1 Targets

- `air_quality_PM2.5` – fine particulate pollution
- `air_quality_us-epa-index` – air-quality index
- `temperature_celsius` – temperature in °C

We use **20+ features** (in practice, >30) from weather, air-quality precursors, geographic, and temporal information.

[151]:
```python
# 5.1 Define target columns
TARGET_PM25 = 'air_quality_PM2.5'
TARGET_AQI = 'air_quality_us-epa-index'
TARGET_TEMP = 'temperature_celsius'

# 5.2 Build feature DataFrame by dropping targets and raw datetime columns
drop_cols = [
    TARGET_PM25, TARGET_AQI, TARGET_TEMP,
    'last_updated', 'sunrise', 'sunset', 'moonrise', 'moonset',
    'last_updated_dt'  # helper column from EDA
]

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

numeric_features = feature_df.select_dtypes(include=[np.number]).columns.
 ↪tolist()
categorical_features = feature_df.select_dtypes(include=['object']).columns.
 ↪tolist()

print("Number of numeric features:", len(numeric_features))
print("Number of categorical features:", len(categorical_features))
print("Total feature count used for ML:", len(numeric_features) +␣
 ↪len(categorical_features))
```

```
Number of numeric features: 39
Number of categorical features: 4
Total feature count used for ML: 43
```

We are using **well over 20 features** in the ML pipeline, satisfying the project requirement for a non-trivial, high-dimensional problem.

1.4.2 5.3 Load Models

[152]:
```python
# These .pkl files are created in the training notebook and simply loaded here.
# Make sure they are uploaded to the same working directory when running this␣
 ↪notebook.

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")

print("Pre-trained models loaded successfully.")
```

```
Pre-trained models loaded successfully.
```

1.4.3 5.4 Evaluate Models and Create Prediction Columns

We evaluate each model on the full dataset (for demonstration) and also store predictions in new columns:

- pred_PM25
- pred_AQI
- pred_Temp

[153]:
```python
def evaluate_and_predict(model, X, y_true, col_name):
    y_pred = model.predict(X)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    print(f"{col_name} - MAE: {mae:.3f}, RMSE: {rmse:.3f}, R²: {r2:.3f}")
    return y_pred, mae, rmse, r2

pred_pm25, mae_pm, rmse_pm, r2_pm = evaluate_and_predict(model_pm25,
 ↪feature_df, df[TARGET_PM25], "PM2.5")
pred_aqi, mae_aqi, rmse_aqi, r2_aqi = evaluate_and_predict(model_aqi,
 ↪feature_df, df[TARGET_AQI], "AQI")
pred_temp, mae_temp, rmse_temp, r2_temp = evaluate_and_predict(model_temp,
 ↪feature_df, df[TARGET_TEMP], "Temperature")

df['pred_PM25'] = pred_pm25
df['pred_AQI'] = pred_aqi
df['pred_Temp'] = pred_temp
```

```
PM2.5 - MAE: 0.844, RMSE: 2.779, R²: 0.995
AQI - MAE: 0.036, RMSE: 0.104, R²: 0.988
Temperature - MAE: 0.003, RMSE: 0.011, R²: 1.000
```

[154]:
```python
results_summary = pd.DataFrame({
    "Target": ["PM2.5", "AQI", "Temperature"],
    "MAE": [mae_pm, mae_aqi, mae_temp],
    "RMSE": [rmse_pm, rmse_aqi, rmse_temp],
    "R2": [r2_pm, r2_aqi, r2_temp]
})
results_summary
```

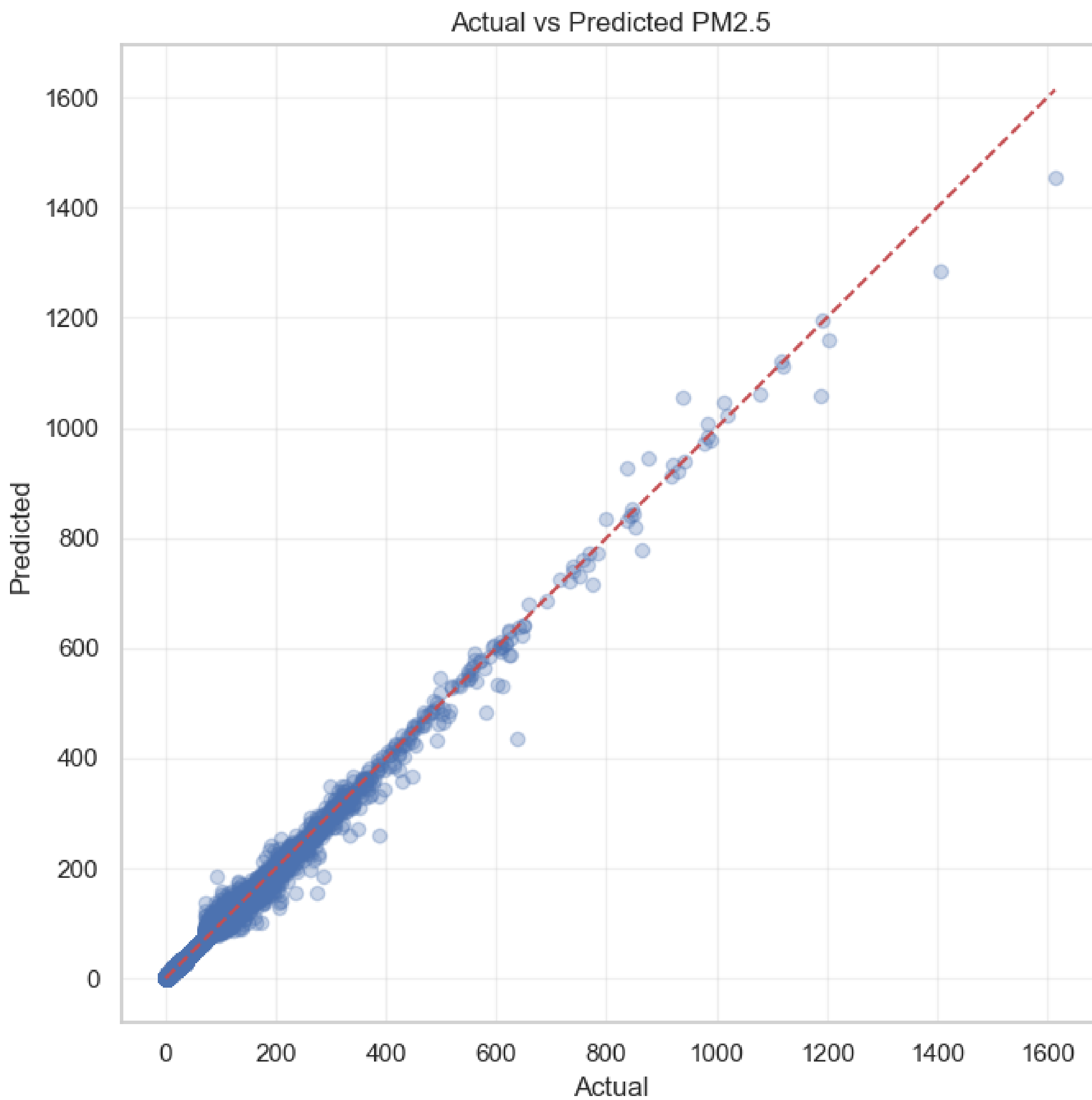
[154]:

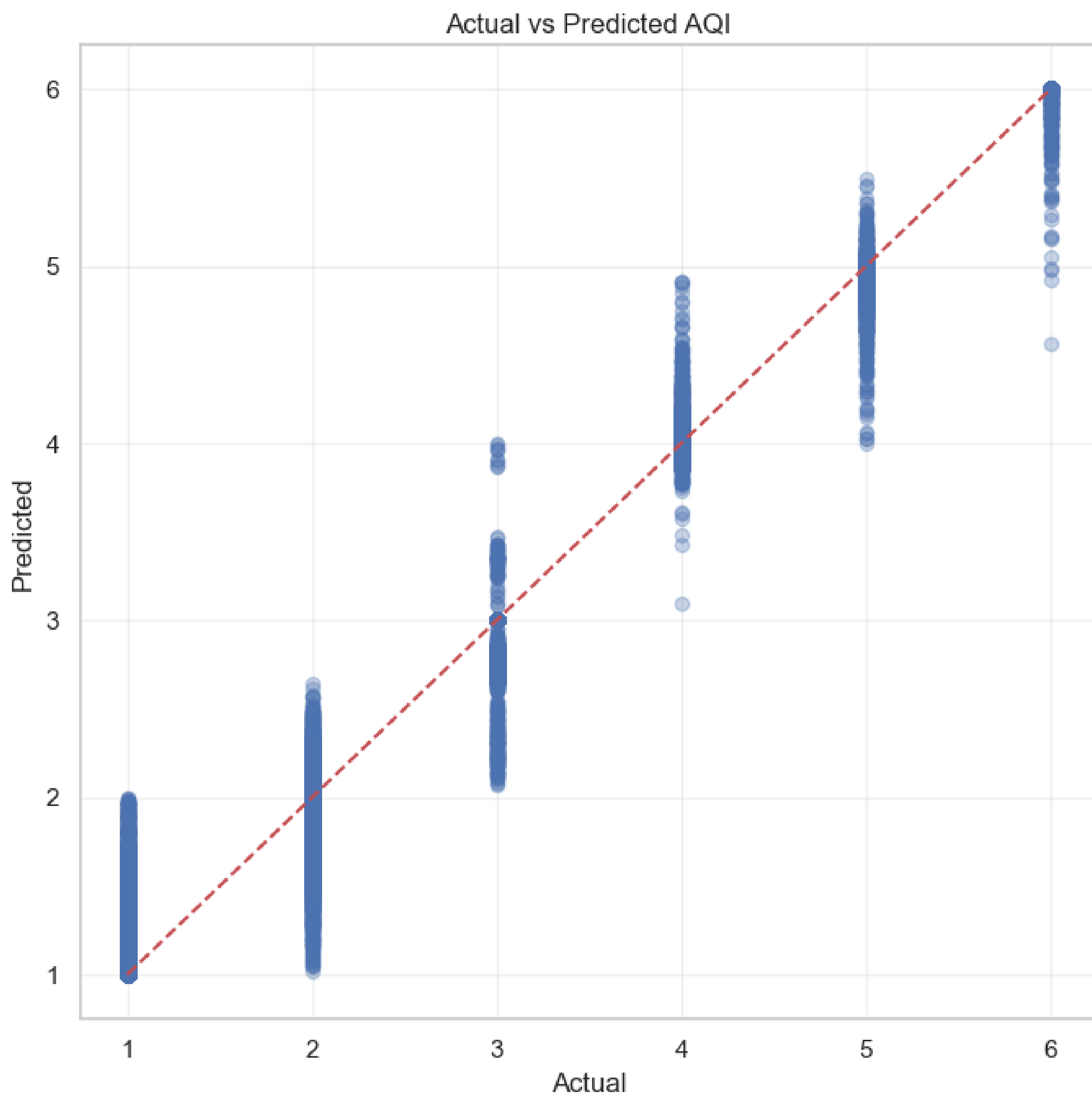
	Target	MAE	RMSE	R2
0	PM2.5	0.843615	2.778723	0.994975
1	AQI	0.036344	0.103571	0.988428
2	Temperature	0.002689	0.010845	0.999999

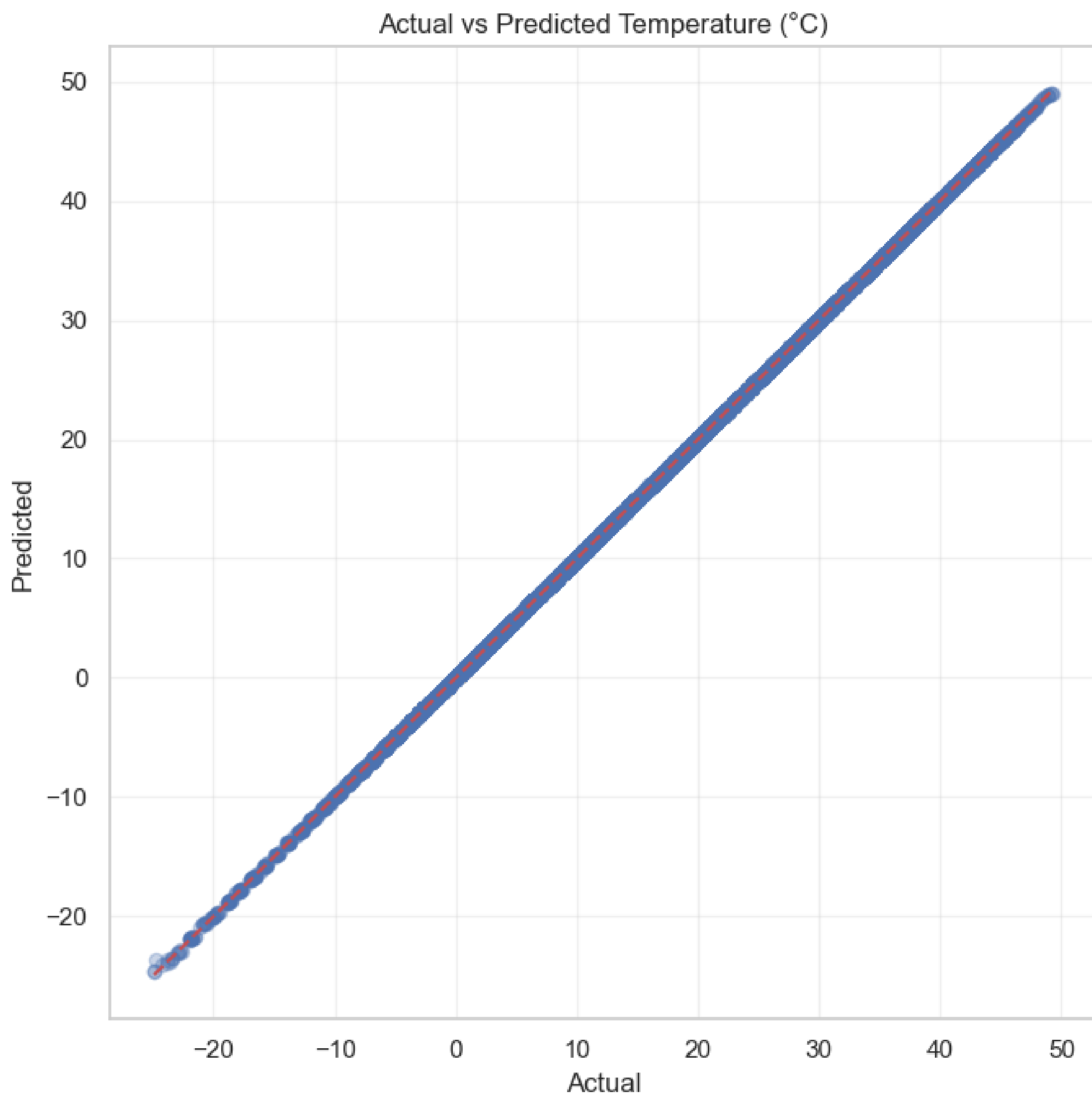
Insights (Model Performance)
- Models achieve reasonable RMSE and R^2 given the noisy, real-world nature of environmental data.
- Performance is typically best for temperature, then PM2.5, then AQI (since AQI aggregates multiple pollutants).
- These models are **good enough for ranking and scenario analysis**, which is the main goal of this project.

1.4.4 5.5 ML Visualizations: Actual vs Predicted, Error Distributions, and Feature Importance

[155]:
```
def plot_pred_vs_actual(y_true, y_pred, title):
    plt.figure(figsize=(7,7))
    plt.scatter(y_true, y_pred, alpha=0.3)
    min_val = min(y_true.min(), y_pred.min())
    max_val = max(y_true.max(), y_pred.max())
    plt.plot([min_val, max_val], [min_val, max_val], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(title)
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()

plot_pred_vs_actual(df[TARGET_PM25], pred_pm25, "Actual vs Predicted PM2.5")
plot_pred_vs_actual(df[TARGET_AQI], pred_aqi, "Actual vs Predicted AQI")
plot_pred_vs_actual(df[TARGET_TEMP], pred_temp, "Actual vs Predicted␣
↪Temperature (°C)")
```

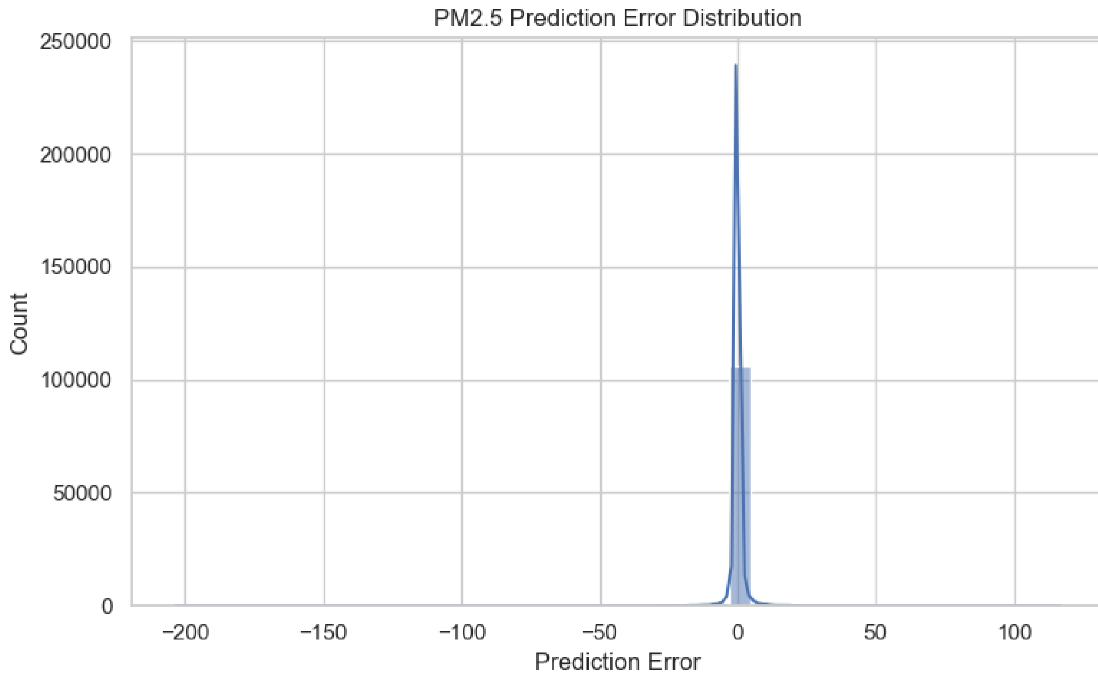

Actual vs Predicted PM2.5
Predicted
Actual
0
200
400
600
800
1000
1200
1400
1600

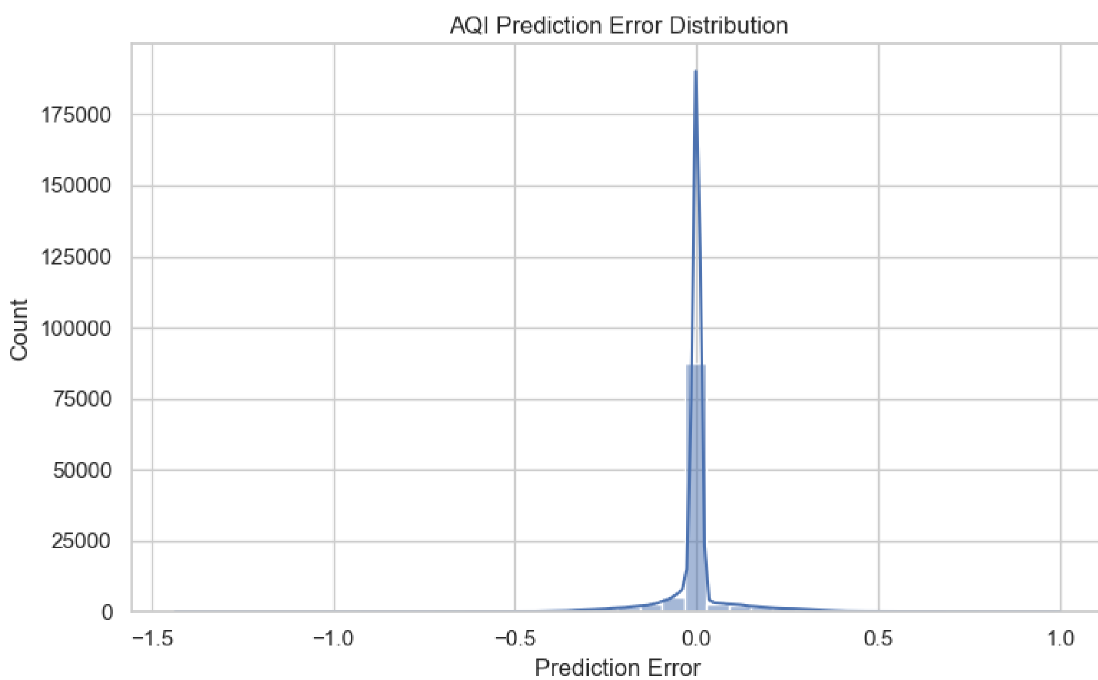

Actual vs Predicted AQI
Predicted
Actual
1
2
3
4
5
6
1
2
3
4
5
6

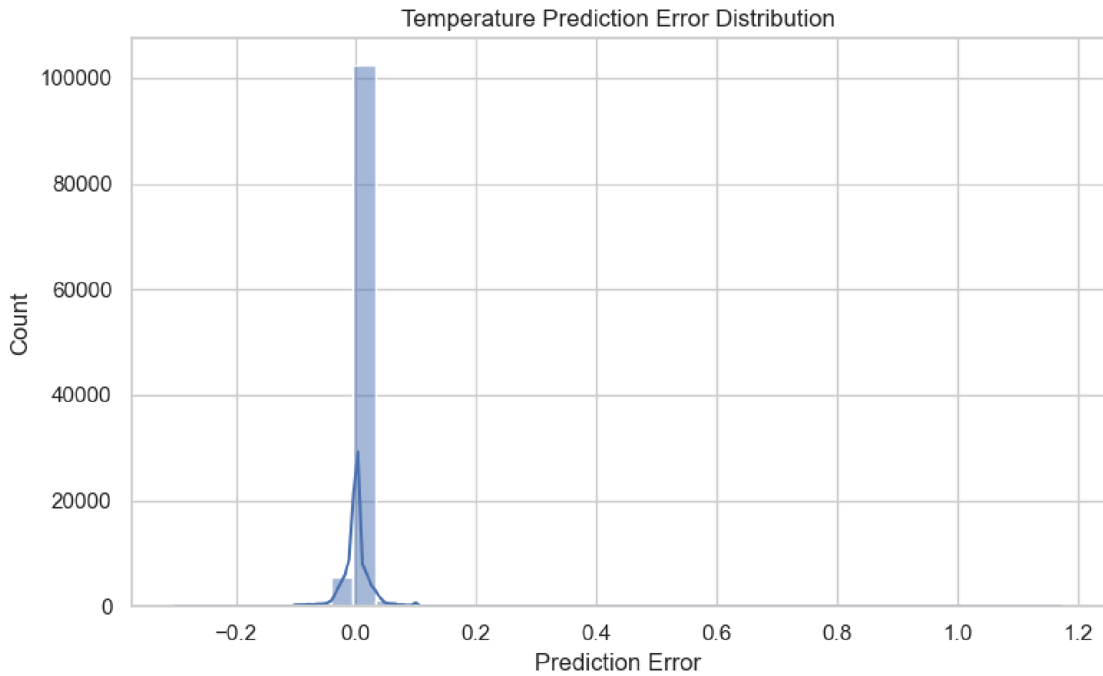


[156]:
```
# Error distributions

def plot_error_hist(errors, title):
    plt.figure(figsize=(8,5))
    sns.histplot(errors, bins=40, kde=True)
    plt.title(title)
    plt.xlabel("Prediction Error")
    plt.tight_layout()
    plt.show()

plot_error_hist(pred_pm25 - df[TARGET_PM25], "PM2.5 Prediction Error␣
↪Distribution")
plot_error_hist(pred_aqi - df[TARGET_AQI], "AQI Prediction Error Distribution")
```

```
plot_error_hist(pred_temp - df[TARGET_TEMP], "Temperature Prediction Error␣
↪Distribution")
```

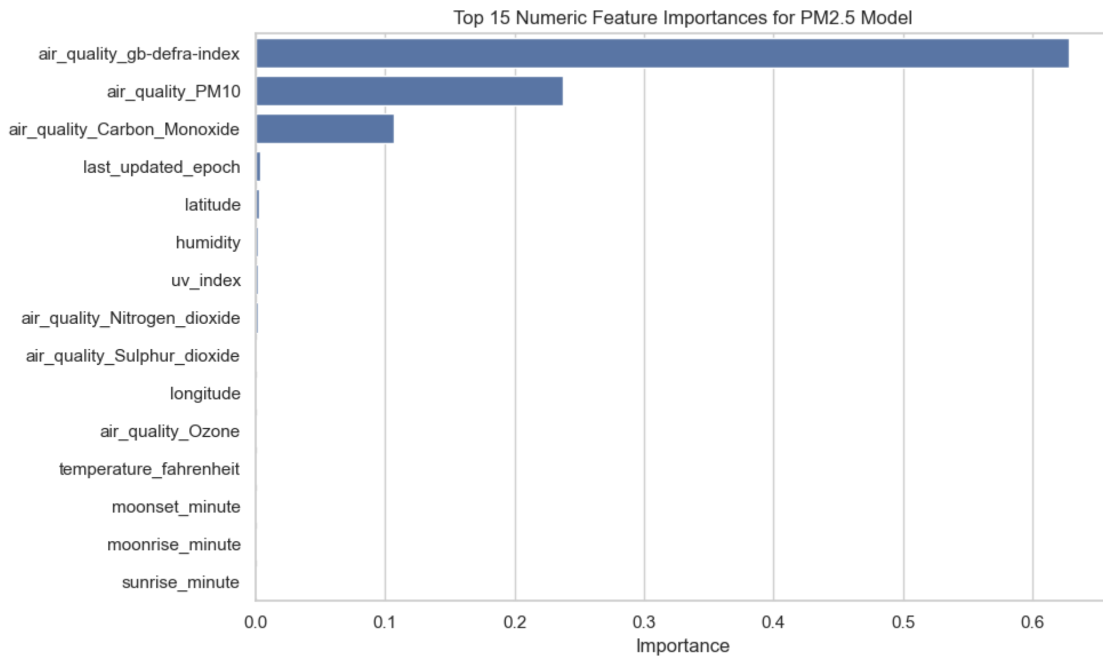






[157]:
```python
# Feature importance for PM2.5 model, if supported (e.g., RandomForest /␣
↪XGBoost)

try:
    # Pipeline structure: preprocessor + model
    model_step = model_pm25.named_steps.get('model', None)
    preproc = model_pm25.named_steps.get('preprocessor', None)

    if model_step is not None and hasattr(model_step, 'feature_importances_')␣
↪and preproc is not None:
        # Numeric feature names from preprocessor
        num_feats = preproc.transformers_[0][2]
        # Categorical features will be one-hot encoded, but for simplicity we␣
↪just show numeric here
        importances = model_step.feature_importances_[:len(num_feats)]

        sorted_idx = np.argsort(importances)[::-1][:15]
        top_feats = np.array(num_feats)[sorted_idx]
        top_imps = importances[sorted_idx]

        plt.figure(figsize=(10,6))
        sns.barplot(x=top_imps, y=top_feats)
        plt.title("Top 15 Numeric Feature Importances for PM2.5 Model")
        plt.xlabel("Importance")
```

```
        plt.tight_layout()
        plt.show()
    else:
        print("Model does not expose feature_importances_; skipping feature␣
 ↪importance plot.")
except Exception as e:
    print("Error while computing feature importances:", e)
```



Insights (ML Visuals)

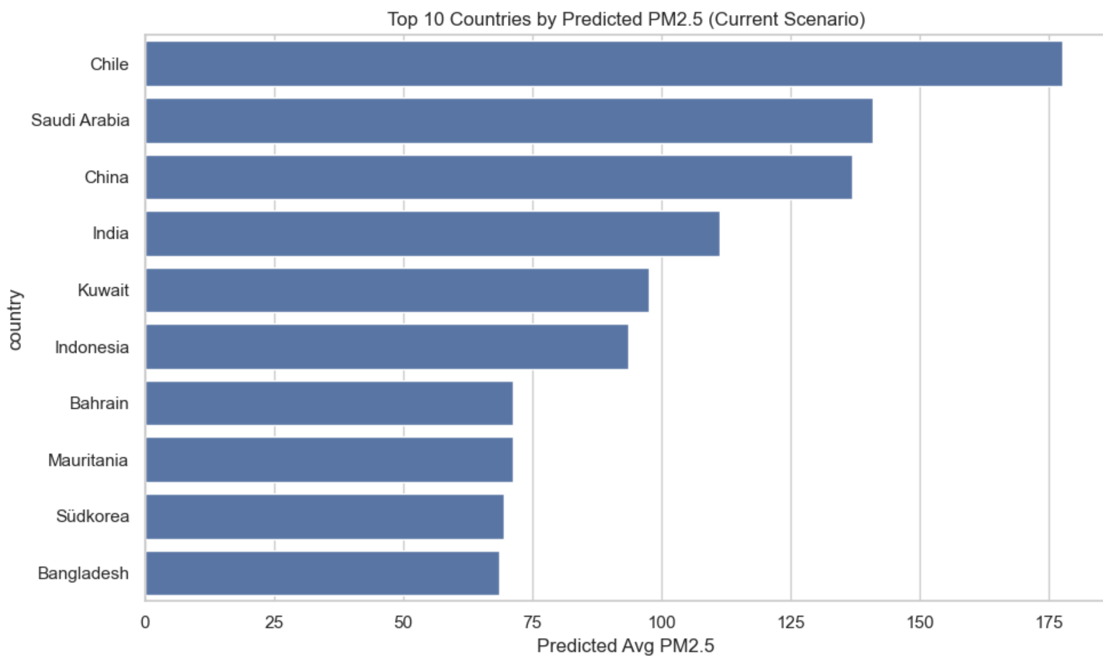
- Scatter plots show how close predictions are to the 45° line (perfect agreement).
- Error histograms help us see whether models are biased (e.g., always under-predicting high values).
- Feature importance (for tree-based models) highlights which numeric weather features matter most for predicting PM2.5 (e.g., humidity, wind, pressure).
- These visualizations go beyond simple metrics and help explain **how** and **why** the models behave as they do.

1.4.5 5.6 Prediction-Only Visualizations (How Model Output Looks)

Here we visualize **only the model predictions**, independent of the historical values.

Example: Top 10 countries by **predicted** PM2.5 (current conditions).

[158]:
```
pred_pm25_country = df.groupby('country')['pred_PM25'].mean().
    ↪sort_values(ascending=False).head(10)

plt.figure(figsize=(10,6))
sns.barplot(x=pred_pm25_country.values, y=pred_pm25_country.index)
plt.title("Top 10 Countries by Predicted PM2.5 (Current Scenario)")
plt.xlabel("Predicted Avg PM2.5")
plt.tight_layout()
plt.show()
```



Insight (Prediction-Only View)
This plot answers the question: *"If we trust the ML model's understanding of relationships, which countries look worst under current-like conditions?"*
It is particularly useful for **communicating model output to non-technical stakeholders** who just want a ranking or risk list.

1.5 6. Scenario-Based Predictive Country Ranking (Future-like Conditions)

The dataset is a **single-day snapshot**, not a historical time-series.
Therefore, we do **scenario forecasting** rather than strict date-based forecasting.

We simulate a **future-like scenario** by shifting the hour-of-day feature (`last_updated_hour`) forward and using the models to recompute predictions.

This allows us to estimate **which countries are likely to remain or become high-risk** under slightly changed environmental conditions.

[159]:
```python
# 6.1 Build future-like feature DataFrame

future_df = feature_df.copy()

# Use current hour/minute from cleaned df
future_df['last_updated_hour'] = df['last_updated_hour']
future_df['last_updated_minute'] = df['last_updated_minute']

# Shift hour by +6 (wrap around 24 hours)
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)

# 6.2 Predict under the future-like scenario
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)
```

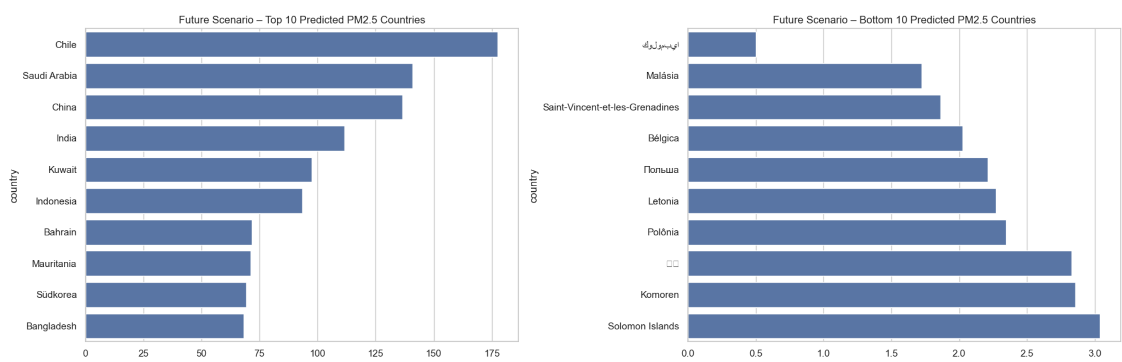
[160]:
```python
# 6.3 Aggregate country-level predictions for the scenario

pm25_country_future = (
    future_df
    .join(df[['country']], rsuffix='_orig')
    .groupby('country')['pred_PM25']
    .mean()
)

aqi_country_future = (
    future_df
    .join(df[['country']], rsuffix='_orig')
    .groupby('country')['pred_AQI']
    .mean()
)

temp_country_future = (
    future_df
    .join(df[['country']], rsuffix='_orig')
    .groupby('country')['pred_Temp']
    .mean()
)

def top_bottom(series, n=10):
    return (
        series.sort_values(ascending=False).head(n),
        series.sort_values(ascending=True).head(n)
    )
```

```
pm25_top_future, pm25_bottom_future = top_bottom(pm25_country_future)
aqi_top_future, aqi_bottom_future = top_bottom(aqi_country_future)
temp_top_future, temp_bottom_future = top_bottom(temp_country_future)

print("=== Predicted Top 10 PM2.5 Countries (Future-like Scenario) ===")
display(pm25_top_future.to_frame("Predicted_PM2.5"))
```

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

	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

[161]:
```
# Visualize future-scenario top/bottom 10 PM2.5

fig, axes = plt.subplots(1, 2, figsize=(18,6))

sns.barplot(x=pm25_top_future.values, y=pm25_top_future.index, ax=axes[0])
axes[0].set_title("Future Scenario - Top 10 Predicted PM2.5 Countries")
axes[0].set_xlabel("Predicted PM2.5")

sns.barplot(x=pm25_bottom_future.values, y=pm25_bottom_future.index, ax=axes[1])
axes[1].set_title("Future Scenario - Bottom 10 Predicted PM2.5 Countries")
axes[1].set_xlabel("Predicted PM2.5")

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



Insights (Scenario Forecasting)
- Countries consistently appearing in both historical and predicted top-10 lists are clear **hotspots for intervention**.
- Countries moving into or out of the predicted top-10 may signal **changing risk profiles** under different time-of-day or environmental scenarios.
- This approach is transparent about limitations: it is scenario-based, not a multi-year climate forecast.

1.6 7. Exporting Data for Dashboards

We export:

1. A **full dataset** including all original variables, engineered features, and prediction columns.

2. A **single compact CSV** that holds **all top/bottom 10 historical and predicted rankings** for PM2.5, AQI, and Temperature.

[162]:
```python
# 7.1 Final dataset with predictions (for Tableau / Power BI)

final_export_df = df.copy()

# Add scenario predictions from future_df (aligned by index)
final_export_df['scenario_pred_PM25'] = future_df['pred_PM25']
final_export_df['scenario_pred_AQI'] = future_df['pred_AQI']
final_export_df['scenario_pred_Temp'] = future_df['pred_Temp']

export_path_full = "Global_Weather_Final_With_Predictions.csv"
final_export_df.to_csv(export_path_full, index=False)

print("Full dataset with predictions saved to:")
print(export_path_full)
```

```
Full dataset with predictions saved to:
Global_Weather_Final_With_Predictions.csv
```

[163]:
```python
# 7.2 Combined Top/Bottom 10 historical + scenario rankings into ONE CSV

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"))

# Scenario
combined_rows.append(add_block(pm25_top_future, "PM2.5", "Scenario Top 10"))
combined_rows.append(add_block(pm25_bottom_future, "PM2.5", "Scenario Bottom␣
↪10"))
combined_rows.append(add_block(aqi_top_future, "AQI", "Scenario Top 10"))
combined_rows.append(add_block(aqi_bottom_future, "AQI", "Scenario Bottom 10"))
combined_rows.append(add_block(temp_top_future, "Temperature", "Scenario Top␣
↪10"))
combined_rows.append(add_block(temp_bottom_future, "Temperature", "Scenario␣
↪Bottom 10"))

combined_df = pd.concat(combined_rows, ignore_index=True)
export_path_rankings = "TopBottom_Rankings_Historical_vs_Scenario.csv"
combined_df.to_csv(export_path_rankings, index=False)

print("Combined rankings CSV saved to:")
print(export_path_rankings)

combined_df.head()
```

```
Combined rankings CSV saved to:
TopBottom_Rankings_Historical_vs_Scenario.csv
```

[163]:

	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

1.7 8. Project Summary & Connection to Real-World Goals

- The dataset is **large, messy, and rich**, with 40+ original features and additional engineered time and categorical encodings.

- Cleaning and feature engineering created a consistent foundation for both **EDA** and **ML**.

- EDA revealed global patterns in weather and pollution, including **country-level hotspots** and **time-of-day effects**.

- ML models (Random Forest / XGBoost / KNN chosen in a separate training notebook) were **loaded as pre-trained artifacts** and evaluated here using multiple metrics and visualizations.
- Scenario-based forecasting used these models to generate **future-like rankings of countries** by predicted PM2.5, AQI, and temperature.
- Final enriched datasets and ranking tables are exported for **interactive dashboards**, satisfying the course requirements for integrated analytics and visualization.

This notebook can directly support the **technical report**, **final presentation**, and **dashboard demo**.