```
import pandas as pd
```

```
base = "/content/drive/MyDrive/PBL/"

csv_years = [2019, 2020, 2021, 2022, 2023, 2024]
csv_paths = [
    f"{base}Raw_data_8Hrs_{y}_site_134_Police_Commissionerate_Jaipur_RSPCB_8Hrs.csv"
    for y in csv_years
]
path_2025 = f"{base}2025.xlsx"
```

```
from google.colab import drive
drive.mount('/content/drive')

import os, pandas as pd
!ls "/content/drive/MyDrive"
```

Show hidden output

```
def load_csv_year(path):
    df = pd.read_csv(path)
    # keep only common core columns for now
    keep = [
        "Timestamp", "PM2.5 (µg/m³)", "PM10 (µg/m³)", "NO (µg/m³)",
        "NO2 (µg/m³)", "NH3 (µg/m³)", "SO2 (µg/m³)",
        "CO (mg/m³)", "Ozone (µg/m³)"
    ]
    df = df[keep]
    df["Timestamp"] = pd.to_datetime(df["Timestamp"])
    return df

dfs_19_24 = [load_csv_year(p) for p in csv_paths]
df_19_24 = pd.concat(dfs_19_24, ignore_index=True)
```

```
df25_raw = pd.read_excel(path_2025, header=16)

rename_map = {
    "From Date": "Timestamp",
    "PM2.5": "PM2.5 (µg/m³)",
    "PM10": "PM10 (µg/m³)",
    "NO": "NO (µg/m³)",
    "NO2": "NO2 (µg/m³)",
    "NH3": "NH3 (µg/m³)",
    "SO2": "SO2 (µg/m³)",
    "CO": "CO (mg/m³)",
    "Ozone": "Ozone (µg/m³)"
}
df25 = df25_raw.rename(columns=rename_map)

keep = [
    "Timestamp", "PM2.5 (µg/m³)", "PM10 (µg/m³)", "NO (µg/m³)",
    "NO2 (µg/m³)", "NH3 (µg/m³)", "SO2 (µg/m³)",
    "CO (mg/m³)", "Ozone (µg/m³)"
]
df25 = df25[keep].dropna(how="all")
df25["Timestamp"] = pd.to_datetime(df25["Timestamp"], dayfirst=True)
```

```
full_df = pd.concat([df_19_24, df25], ignore_index=True)
full_df = full_df.sort_values("Timestamp").reset_index(drop=True)

print(full_df.head())
print(full_df.tail())
print(full_df["Timestamp"].min(), "→", full_df["Timestamp"].max())
```

```
            Timestamp  PM2.5 (µg/m³)  PM10 (µg/m³)  NO (µg/m³)  NO2 (µg/m³)  \
0 2019-01-01 00:00:00         171.79        311.12      224.21        46.09
1 2019-01-01 08:00:00         103.10        198.00       68.06        62.80
2 2019-01-01 16:00:00          67.01        147.99       63.87        80.55
3 2019-01-02 00:00:00          73.99        171.69       89.04        71.58
4 2019-01-02 08:00:00          82.66        176.64       45.67        68.62

   NH3 (µg/m³)  SO2 (µg/m³)  CO (mg/m³)  Ozone (µg/m³)
0        13.47        23.24        0.97           6.89
1        23.13        20.30        0.87          65.92
```

```
2        24.58        22.97        1.42          17.69
3        22.10        24.00        0.92           7.33
4        25.23        12.84        0.80          62.57
              Timestamp  PM2.5 (µg/m³)  PM10 (µg/m³)  NO (µg/m³)  \
7666 2025-12-30 08:00:00          35.75        107.21       36.80
7667 2025-12-30 16:00:00          74.49        131.72       73.63
7668 2025-12-31 00:00:00          90.71        156.40       86.23
7669 2025-12-31 08:00:00          62.15        105.92       36.31
7670 2025-12-31 16:00:00          55.04        107.94       43.11

      NO2 (µg/m³)  NH3 (µg/m³)  SO2 (µg/m³)  CO (mg/m³)  Ozone (µg/m³)
7666       110.91        37.23        12.67        1.28          54.95
7667        92.57        64.35        12.76        0.88          23.12
7668        89.64        74.80        12.59        1.24          15.49
7669       107.21        34.41        14.21        1.38          54.06
7670       105.52        43.55        16.12        1.12          33.49
2019-01-01 00:00:00 → 2025-12-31 16:00:00
```

basic cleaning and a first feature-ready table for PM2.5/PM10.

```
full_df = full_df.dropna(subset=["PM2.5 (µg/m³)", "PM10 (µg/m³)"], how="all")

for col in ["PM2.5 (µg/m³)", "PM10 (µg/m³)", "NO (µg/m³)",
            "NO2 (µg/m³)", "NH3 (µg/m³)", "SO2 (µg/m³)",
            "CO (mg/m³)", "Ozone (µg/m³)"]:
    full_df = full_df[ (full_df[col].isna()) | (full_df[col] >= 0) ]

full_df["year"] = full_df["Timestamp"].dt.year
full_df["month"] = full_df["Timestamp"].dt.month
full_df["day"] = full_df["Timestamp"].dt.day
full_df["hour"] = full_df["Timestamp"].dt.hour

print(full_df.isna().mean())   # fraction of missing per column
print(full_df.head())
```

```
Timestamp        0.000000
PM2.5 (µg/m³)    0.004593
PM10 (µg/m³)     0.001486
NO (µg/m³)       0.007025
NO2 (µg/m³)      0.007160
NH3 (µg/m³)      0.010403
SO2 (µg/m³)      0.006620
CO (mg/m³)       0.005674
Ozone (µg/m³)    0.007025
year             0.000000
month            0.000000
day              0.000000
hour             0.000000
dtype: float64
             Timestamp  PM2.5 (µg/m³)  PM10 (µg/m³)  NO (µg/m³)  NO2 (µg/m³)  \
0 2019-01-01 00:00:00         171.79        311.12      224.21        46.09
1 2019-01-01 08:00:00         103.10        198.00       68.06        62.80
2 2019-01-01 16:00:00          67.01        147.99       63.87        80.55
3 2019-01-02 00:00:00          73.99        171.69       89.04        71.58
4 2019-01-02 08:00:00          82.66        176.64       45.67        68.62

   NH3 (µg/m³)  SO2 (µg/m³)  CO (mg/m³)  Ozone (µg/m³)  year  month  day  hour
0        13.47        23.24        0.97           6.89  2019      1    1     0
1        23.13        20.30        0.87          65.92  2019      1    1     8
2        24.58        22.97        1.42          17.69  2019      1    1    16
3        22.10        24.00        0.92           7.33  2019      1    2     0
4        25.23        12.84        0.80          62.57  2019      1    2     8
```

```
import numpy as np
import pandas as pd
from datetime import timedelta

df = full_df.copy()
df = df.sort_values("Timestamp")

# --- Time features ---
df["hour"] = df["Timestamp"].dt.hour
df["dayofweek"] = df["Timestamp"].dt.dayofweek
df["month"] = df["Timestamp"].dt.month

# Cyclical encoding of hour
df["hour_sin"] = np.sin(2 * np.pi * df["hour"] / 24)
df["hour_cos"] = np.cos(2 * np.pi * df["hour"] / 24)

# --- Diwali features for each year ---
def add_diwali_features(frame, diwali_dates):
    frame["is_diwali_period"] = 0
```

```
    frame["days_since_diwali"] = 0

    for year, date_str in diwali_dates.items():
        d = pd.Timestamp(date_str)
        start = d - timedelta(days=21)
        end = d + timedelta(days=21)
        mask_year = frame["Timestamp"].dt.year == year
        mask_window = (frame["Timestamp"] >= start) & (frame["Timestamp"] <= end)
        mask = mask_year & mask_window

        frame.loc[mask, "is_diwali_period"] = 1
        frame.loc[mask, "days_since_diwali"] = (
            (frame.loc[mask, "Timestamp"] - d).dt.days
        )

    return frame

diwali_dates = {
    2019: "2019-10-27",
    2020: "2020-11-14",
    2021: "2021-11-04",
    2022: "2022-10-24",
    2023: "2023-11-12",
    2024: "2024-10-31",
    2025: "2025-10-20",
}

df = add_diwali_features(df, diwali_dates)

print(df[["Timestamp","is_diwali_period","days_since_diwali"]].head(20))
print(df[df["is_diwali_period"]==1].head())
```

```
             Timestamp  is_diwali_period  days_since_diwali
0  2019-01-01 00:00:00                 0                  0
1  2019-01-01 08:00:00                 0                  0
2  2019-01-01 16:00:00                 0                  0
3  2019-01-02 00:00:00                 0                  0
4  2019-01-02 08:00:00                 0                  0
5  2019-01-02 16:00:00                 0                  0
6  2019-01-03 00:00:00                 0                  0
7  2019-01-03 08:00:00                 0                  0
8  2019-01-03 16:00:00                 0                  0
9  2019-01-04 00:00:00                 0                  0
10 2019-01-04 08:00:00                 0                  0
11 2019-01-04 16:00:00                 0                  0
12 2019-01-05 00:00:00                 0                  0
13 2019-01-05 08:00:00                 0                  0
14 2019-01-05 16:00:00                 0                  0
15 2019-01-06 00:00:00                 0                  0
16 2019-01-06 08:00:00                 0                  0
17 2019-01-06 16:00:00                 0                  0
18 2019-01-07 00:00:00                 0                  0
19 2019-01-07 08:00:00                 0                  0
              Timestamp  PM2.5 (µg/m³)  PM10 (µg/m³)  NO (µg/m³)  NO2 (µg/m³)  \
834 2019-10-06 00:00:00          56.46         89.44       17.07        58.64
835 2019-10-06 08:00:00          51.06         89.56        3.93        35.32
836 2019-10-06 16:00:00          51.21        103.34        8.07        57.10
837 2019-10-07 00:00:00          57.18        110.98       14.28        50.22
838 2019-10-07 08:00:00          67.80        101.67        6.05        36.28

     NH3 (µg/m³)  SO2 (µg/m³)  CO (mg/m³)  Ozone (µg/m³)  year  month  day  \
834        27.07        14.69        0.65          13.84  2019     10    6
835        22.11        12.56        0.43          66.19  2019     10    6
836        26.06        12.64        0.79          14.99  2019     10    6
837        24.44        13.51        0.75          15.22  2019     10    7
838        23.52         8.82        0.74          23.68  2019     10    7

     hour  dayofweek  hour_sin  hour_cos  is_diwali_period  days_since_diwali
834     0          6  0.000000       1.0                 1                -21
835     8          6  0.866025      -0.5                 1                -21
836    16          6 -0.866025      -0.5                 1                -21
837     0          0  0.000000       1.0                 1                -20
838     8          0  0.866025      -0.5                 1                -20
```

```
df = df.sort_values("Timestamp").reset_index(drop=True)

df["PM2.5_t-1"] = df["PM2.5 (µg/m³)"].shift(1)
df["PM10_t-1"] = df["PM10 (µg/m³)"].shift(1)
df = df.dropna(subset=["PM2.5_t-1", "PM10_t-1"])
```

```
feature_cols = [
    "PM2.5_t-1", "PM10_t-1",
    "NO (µg/m³)", "NO2 (µg/m³)", "NH3 (µg/m³)",
```

```
    "SO2 (µg/m³)", "CO (mg/m³)", "Ozone (µg/m³)",
    "hour_sin", "hour_cos", "dayofweek", "month",
    "is_diwali_period", "days_since_diwali"
]

target_pm25 = "PM2.5 (µg/m³)"
target_pm10 = "PM10 (µg/m³)"

train_mask = df["year"] <= 2023
test_mask  = df["year"] >= 2024

X_train = df.loc[train_mask, feature_cols]
y25_train = df.loc[train_mask, target_pm25]
y10_train = df.loc[train_mask, target_pm10]

X_test = df.loc[test_mask, feature_cols]
y25_test = df.loc[test_mask, target_pm25]
y10_test = df.loc[test_mask, target_pm10]
```

```
df_model = df.copy()

# drop rows where targets are missing
df_model = df_model.dropna(subset=["PM2.5 (µg/m³)", "PM10 (µg/m³)"])

# also ensure lag features exist
df_model["PM2.5_t-1"] = df_model["PM2.5 (µg/m³)"].shift(1)
df_model["PM10_t-1"] = df_model["PM10 (µg/m³)"].shift(1)
df_model = df_model.dropna(subset=["PM2.5_t-1", "PM10_t-1"])

feature_cols = [
    "PM2.5_t-1","PM10_t-1",
    "NO (µg/m³)","NO2 (µg/m³)","NH3 (µg/m³)",
    "SO2 (µg/m³)","CO (mg/m³)","Ozone (µg/m³)",
    "hour_sin","hour_cos","dayofweek","month",
    "is_diwali_period","days_since_diwali"
]

train_mask = df_model["year"] <= 2023
test_mask  = df_model["year"] >= 2024

X_train = df_model.loc[train_mask, feature_cols]
y25_train = df_model.loc[train_mask, "PM2.5 (µg/m³)"]
y10_train = df_model.loc[train_mask, "PM10 (µg/m³)"]

X_test = df_model.loc[test_mask, feature_cols]
y25_test = df_model.loc[test_mask, "PM2.5 (µg/m³)"]
y10_test = df_model.loc[test_mask, "PM10 (µg/m³)"]
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

rf25 = RandomForestRegressor(
    n_estimators=150,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)
rf25.fit(X_train, y25_train)

pred25 = rf25.predict(X_test)
print("PM2.5 MAE:", mean_absolute_error(y25_test, pred25))
print("PM2.5 R2:", r2_score(y25_test, pred25))

rf10 = RandomForestRegressor(
    n_estimators=150,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)
rf10.fit(X_train, y10_train)

pred10 = rf10.predict(X_test)
print("PM10 MAE:", mean_absolute_error(y10_test, pred10))
print("PM10 R2:", r2_score(y10_test, pred10))
```

```
PM2.5 MAE: 13.869190489323472
PM2.5 R2: 0.45483815879302203
PM10 MAE: 30.815990949938115
```

```
PM10 R2: 0.45655948609507413
```

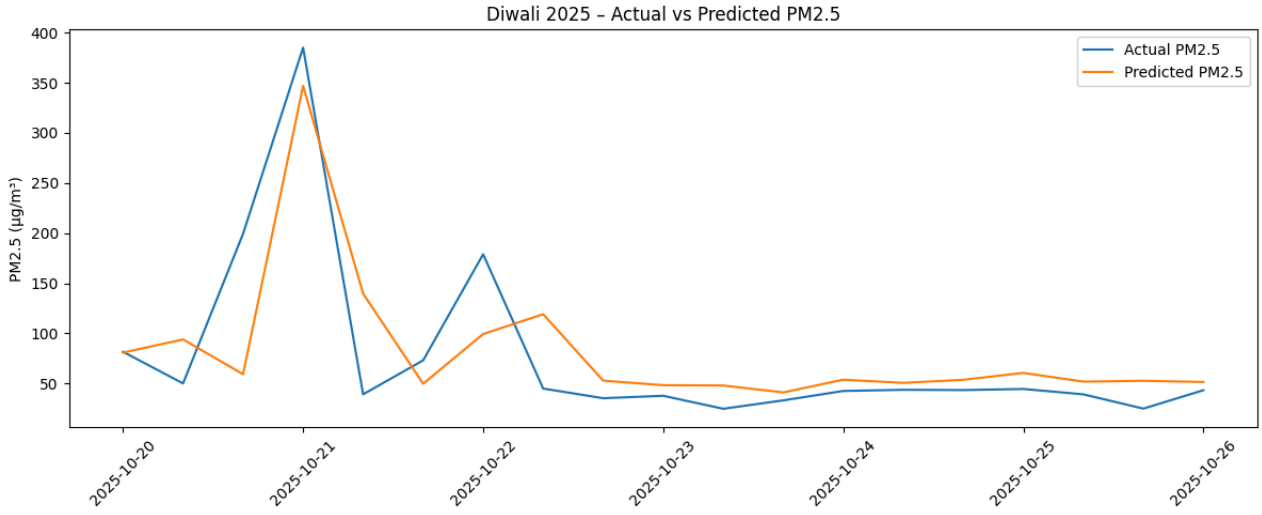
```
from sklearn.metrics import mean_absolute_error, r2_score

test_all = df_model[df_model["year"] >= 2019].copy()
X_all   = test_all[feature_cols]
pred25_all = rf25.predict(X_all)
pred10_all = rf10.predict(X_all)

for yr in sorted(test_all["year"].unique()):
    mask_y  = (test_all["year"] == yr) & (test_all["is_diwali_period"] == 1)
    if mask_y.sum() == 0:
        continue

    y25_y = test_all.loc[mask_y, "PM2.5 (µg/m³)"]
    y10_y = test_all.loc[mask_y, "PM10 (µg/m³)"]
    p25_y = pred25_all[mask_y]
    p10_y = pred10_all[mask_y]

    print(f"Year {yr} Diwali PM2.5 MAE:", mean_absolute_error(y25_y, p25_y))
    print(f"Year {yr} Diwali PM2.5 R2:",  r2_score(y25_y, p25_y))
    print(f"Year {yr} Diwali PM10 MAE:", mean_absolute_error(y10_y, p10_y))
    print(f"Year {yr} Diwali PM10 R2:",  r2_score(y10_y, p10_y))
    print("----")
```

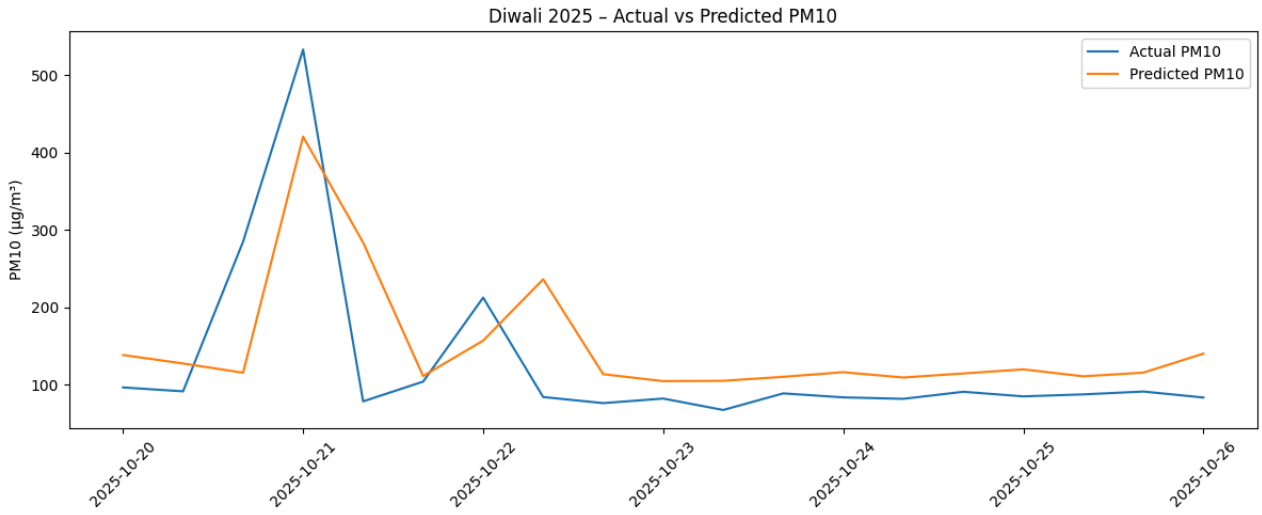
```
Year 2019 Diwali PM2.5 MAE: 15.57688601834193
Year 2019 Diwali PM2.5 R2: 0.833932378538631
Year 2019 Diwali PM10 MAE: 19.440657430660995
Year 2019 Diwali PM10 R2: 0.8085390573604816
----
Year 2020 Diwali PM2.5 MAE: 14.854801914277433
Year 2020 Diwali PM2.5 R2: 0.7148940847021459
Year 2020 Diwali PM10 MAE: 25.24405486822488
Year 2020 Diwali PM10 R2: 0.7691089634431312
----
Year 2021 Diwali PM2.5 MAE: 16.756565602609875
Year 2021 Diwali PM2.5 R2: 0.7643637801529793
Year 2021 Diwali PM10 MAE: 27.818502213118407
Year 2021 Diwali PM10 R2: 0.8214341260413109
----
Year 2022 Diwali PM2.5 MAE: 13.139761946515263
Year 2022 Diwali PM2.5 R2: 0.8751646455149417
Year 2022 Diwali PM10 MAE: 21.80682852482077
Year 2022 Diwali PM10 R2: 0.886093046189886
----
Year 2023 Diwali PM2.5 MAE: 15.054568394703892
Year 2023 Diwali PM2.5 R2: 0.6274748634608187
Year 2023 Diwali PM10 MAE: 25.465973898531708
Year 2023 Diwali PM10 R2: 0.7337312319678782
----
Year 2024 Diwali PM2.5 MAE: 16.49395016740803
Year 2024 Diwali PM2.5 R2: -0.25771053430445057
Year 2024 Diwali PM10 MAE: 34.75655275921342
Year 2024 Diwali PM10 R2: 0.38769239686965773
----
Year 2025 Diwali PM2.5 MAE: 19.27142138536475
Year 2025 Diwali PM2.5 R2: 0.4528712238052428
Year 2025 Diwali PM10 MAE: 40.47783176342097
Year 2025 Diwali PM10 R2: 0.20697408487151003
----
```

```
rows = []
for yr in sorted(test_all["year"].unique()):
    m = (test_all["year"] == yr) & (test_all["is_diwali_period"] == 1)
    if m.sum() == 0:
        continue
    y25_y = test_all.loc[m, "PM2.5 (µg/m³)"]
    y10_y = test_all.loc[m, "PM10 (µg/m³)"]
    p25_y = pred25_all[m]
    p10_y = pred10_all[m]
    rows.append({
        "year": yr,
        "PM2.5_MAE": mean_absolute_error(y25_y, p25_y),
        "PM2.5_R2":  r2_score(y25_y, p25_y),
        "PM10_MAE":  mean_absolute_error(y10_y, p10_y),
        "PM10_R2":   r2_score(y10_y, p10_y),
    })

import pandas as pd
diwali_summary = pd.DataFrame(rows)
print(diwali_summary)
```

```
   year  PM2.5_MAE  PM2.5_R2   PM10_MAE   PM10_R2
0  2019  15.576886  0.833932  19.440657  0.808539
1  2020  14.854802  0.714894  25.244055  0.769109
2  2021  16.756566  0.764364  27.818502  0.821434
3  2022  13.139762  0.875165  21.806829  0.886093
4  2023  15.054568  0.627475  25.465974  0.733731
5  2024  16.493950 -0.257711  34.756553  0.387692
6  2025  19.271421  0.452871  40.477832  0.206974
```

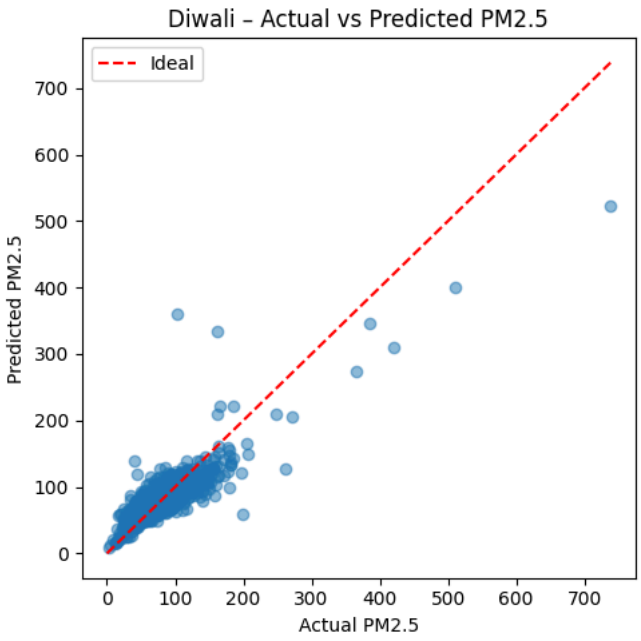
```
import matplotlib.pyplot as plt

test = df_model[df_model["year"] >= 2019].copy()
X_all = test[feature_cols]
pred25_all = rf25.predict(X_all)
pred10_all = rf10.predict(X_all)

# attach predictions
test["PM2.5_pred"] = pred25_all
test["PM10_pred"] = pred10_all

mask_2025 = (test["Timestamp"] >= "2025-10-20") & (test["Timestamp"] <= "2025-10-26")

plt.figure(figsize=(12,5))
plt.plot(test["Timestamp"][mask_2025], test["PM2.5 (µg/m³)"][mask_2025],
         label="Actual PM2.5")
plt.plot(test["Timestamp"][mask_2025], test["PM2.5_pred"][mask_2025],
         label="Predicted PM2.5")
plt.xticks(rotation=45)
plt.ylabel("PM2.5 (µg/m³)")
plt.title("Diwali 2025 – Actual vs Predicted PM2.5")
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(12,5))
plt.plot(test["Timestamp"][mask_2025], test["PM10 (µg/m³)"][mask_2025],
         label="Actual PM10")
plt.plot(test["Timestamp"][mask_2025], test["PM10_pred"][mask_2025],
         label="Predicted PM10")
plt.xticks(rotation=45)
plt.ylabel("PM10 (µg/m³)")
plt.title("Diwali 2025 – Actual vs Predicted PM10")
plt.legend()
plt.tight_layout()
plt.show()
```

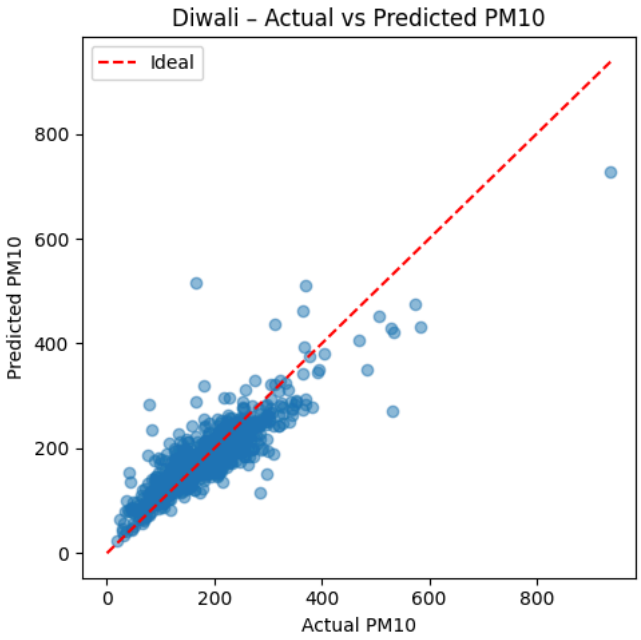




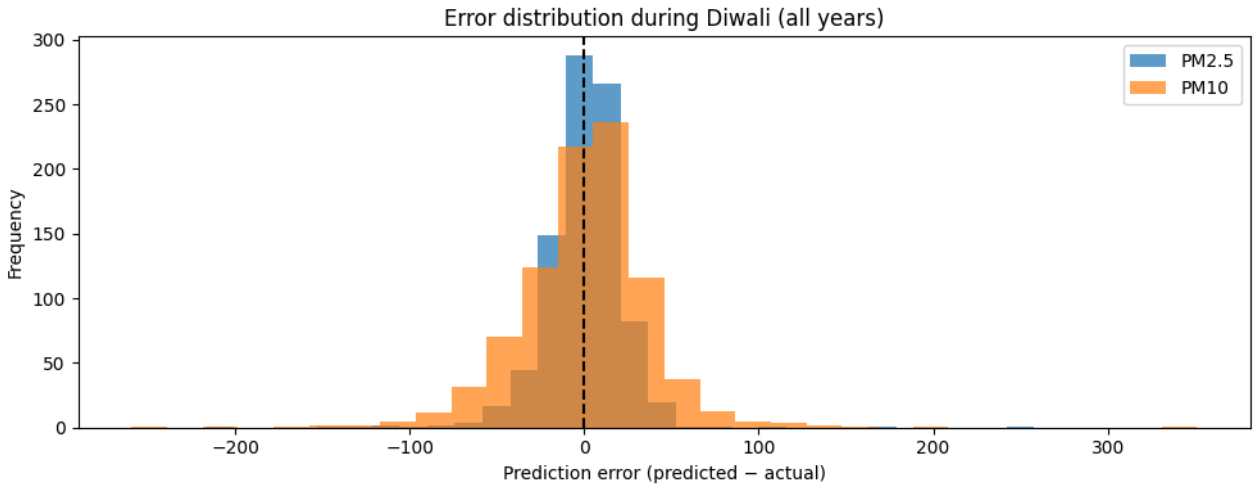

```
import matplotlib.pyplot as plt

mask_diwali = test["is_diwali_period"] == 1

plt.figure(figsize=(5,5))
plt.scatter(test["PM2.5 (µg/m³)"][mask_diwali],
            test["PM2.5_pred"][mask_diwali],
            alpha=0.5)
plt.plot([0, test["PM2.5 (µg/m³)"][mask_diwali].max()],
         [0, test["PM2.5 (µg/m³)"][mask_diwali].max()],
         "r--", label="Ideal")
plt.xlabel("Actual PM2.5")
plt.ylabel("Predicted PM2.5")
plt.title("Diwali – Actual vs Predicted PM2.5")
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(5,5))
plt.scatter(test["PM10 (µg/m³)"][mask_diwali],
            test["PM10_pred"][mask_diwali],
            alpha=0.5)
plt.plot([0, test["PM10 (µg/m³)"][mask_diwali].max()],
         [0, test["PM10 (µg/m³)"][mask_diwali].max()],
         "r--", label="Ideal")
plt.xlabel("Actual PM10")
plt.ylabel("Predicted PM10")
plt.title("Diwali – Actual vs Predicted PM10")
plt.legend()
```

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

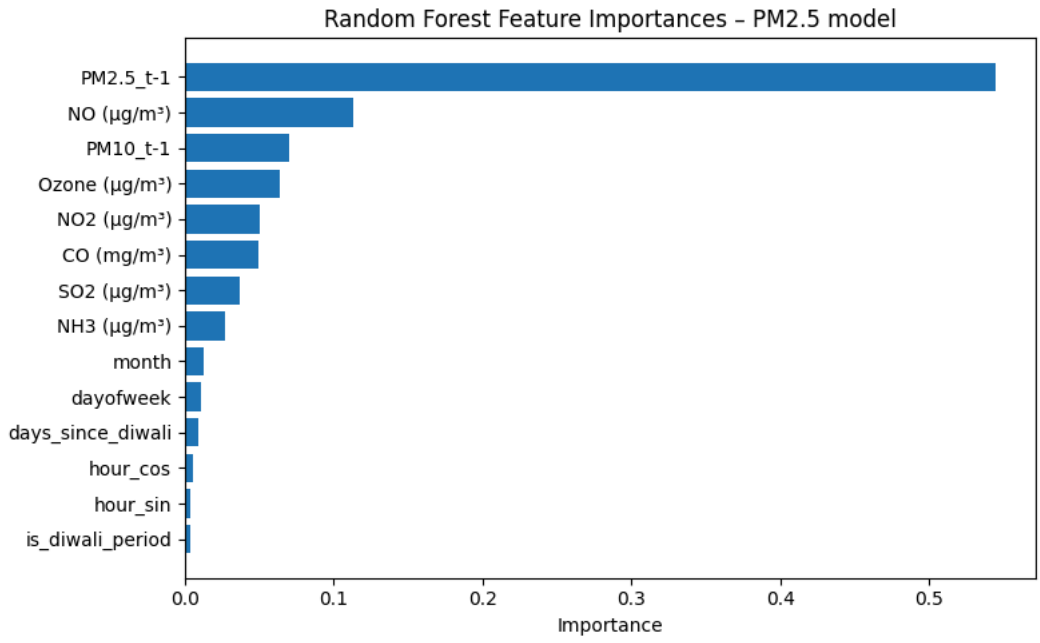





```
import matplotlib.pyplot as plt

err25 = test["PM2.5_pred"][mask_diwali] - test["PM2.5 (µg/m³)"][mask_diwali]
err10 = test["PM10_pred"][mask_diwali] - test["PM10 (µg/m³)"][mask_diwali]

plt.figure(figsize=(10,4))
plt.hist(err25, bins=30, alpha=0.7, label="PM2.5")
plt.hist(err10, bins=30, alpha=0.7, label="PM10")
plt.axvline(0, color="k", linestyle="--")
plt.xlabel("Prediction error (predicted - actual)")
plt.ylabel("Frequency")
plt.title("Error distribution during Diwali (all years)")
plt.legend()
plt.tight_layout()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

importances = rf25.feature_importances_
idx = np.argsort(importances)

plt.figure(figsize=(8,5))
plt.barh(np.array(feature_cols)[idx], importances[idx])
plt.xlabel("Importance")
plt.title("Random Forest Feature Importances – PM2.5 model")
plt.tight_layout()
plt.show()
```




```
# ================================
# Model zoo: compare many models
# ================================
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# 0) Make a fully clean copy (no NaNs in features or targets)
df_model_clean = df_model.dropna(
    subset=feature_cols + ["PM2.5 (µg/m³)", "PM10 (µg/m³)"]
)

# 1) Train/test split
train_mask = df_model_clean["year"] <= 2023
test_mask  = df_model_clean["year"] >= 2024
```

```
X_train = df_model_clean.loc[train_mask, feature_cols]
X_test  = df_model_clean.loc[test_mask,  feature_cols]

y25_train = df_model_clean.loc[train_mask, "PM2.5 (µg/m³)"]
y25_test  = df_model_clean.loc[test_mask,  "PM2.5 (µg/m³)"]

y10_train = df_model_clean.loc[train_mask, "PM10 (µg/m³)"]
y10_test  = df_model_clean.loc[test_mask,  "PM10 (µg/m³)"]

# 2) Candidate models
models = [
    ("LinReg", LinearRegression()),
    ("Ridge", Ridge(alpha=1.0)),
    ("Lasso", Lasso(alpha=0.001)),
    ("KNN_5", KNeighborsRegressor(n_neighbors=5)),
    ("SVR_rbf", SVR(kernel="rbf", C=10, epsilon=0.1)),
    ("RF", RandomForestRegressor(
        n_estimators=150, max_depth=10,
        random_state=42, n_jobs=-1
    )),
    ("GBR", GradientBoostingRegressor(random_state=42)),
]

# 3) Compare models for PM2.5
results_pm25 = []
print("==== PM2.5 models ====")
for name, model in models:
    model.fit(X_train, y25_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y25_test, y_pred)
    r2  = r2_score(y25_test, y_pred)
    results_pm25.append((name, mae, r2))
    print(f"{name:8s}  MAE={mae:6.2f}   R2={r2:6.3f}")

# 4) Compare models for PM10
results_pm10 = []
print("\n==== PM10 models ====")
for name, model in models:
    model.fit(X_train, y10_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y10_test, y_pred)
    r2  = r2_score(y10_test, y_pred)
    results_pm10.append((name, mae, r2))
    print(f"{name:8s}  MAE={mae:6.2f}   R2={r2:6.3f}")

# 5) Choose best models by R2
best_pm25_name, best_pm25_mae, best_pm25_r2 = sorted(
    results_pm25, key=lambda x: x[2], reverse=True
)[0]

best_pm10_name, best_pm10_mae, best_pm10_r2 = sorted(
    results_pm10, key=lambda x: x[2], reverse=True
)[0]

print("\nBest PM2.5 model :", best_pm25_name,
      "MAE=", round(best_pm25_mae, 2),
      "R2=", round(best_pm25_r2, 3))
print("Best PM10 model  :", best_pm10_name,
      "MAE=", round(best_pm10_mae, 2),
      "R2=", round(best_pm10_r2, 3))

# 6) Refit best models on train set for later Diwali/Lohri analysis
name_to_model = dict(models)

best_pm25_model = name_to_model[best_pm25_name]
best_pm10_model = name_to_model[best_pm10_name]

best_pm25_model.fit(X_train, y25_train)
best_pm10_model.fit(X_train, y10_train)

# Use best_pm25_model and best_pm10_model instead of rf25 / rf10 below
```

```
==== PM2.5 models ====
LinReg    MAE= 14.78   R2= 0.482
Ridge     MAE= 14.78   R2= 0.482
Lasso     MAE= 14.78   R2= 0.482
KNN_5     MAE= 14.44   R2= 0.489
SVR_rbf   MAE= 12.41   R2= 0.589
RF        MAE= 13.51   R2= 0.538
GBR       MAE= 13.80   R2= 0.486

==== PM10 models ====
LinReg    MAE= 31.12   R2= 0.461
Ridge     MAE= 31.12   R2= 0.461
Lasso     MAE= 31.12   R2= 0.461
KNN_5     MAE= 31.55   R2= 0.455
SVR_rbf   MAE= 28.45   R2= 0.539
RF        MAE= 29.63   R2= 0.472
GBR       MAE= 30.07   R2= 0.474

Best PM2.5 model : SVR_rbf MAE= 12.41 R2= 0.589
Best PM10 model  : SVR_rbf MAE= 28.45 R2= 0.539
▾ SVR
SVR(C=10)
```

Lohri Data

```
lohri_dates = {
    2019: "2019-01-13",
    2020: "2020-01-13",
    2021: "2021-01-13",
    2022: "2022-01-13",
    2023: "2023-01-13",
    2024: "2024-01-13",
    2025: "2025-01-13",
}
```

```
from datetime import timedelta
import pandas as pd

df["is_lohri_period"] = 0
df["days_since_lohri"] = 0

for year, date_str in lohri_dates.items():
    d = pd.Timestamp(date_str)
    start = d - timedelta(weeks=4)
    end = d + timedelta(weeks=4)
    mask = (df["Timestamp"] >= start) & (df["Timestamp"] <= end) & (df["year"] == year)

    df.loc[mask, "is_lohri_period"] = 1
    df.loc[mask, "days_since_lohri"] = (df.loc[mask, "Timestamp"] - d).dt.days
```

```
feature_cols = [
    "PM2.5_t-1","PM10_t-1",
    "NO (µg/m³)","NO2 (µg/m³)","NH3 (µg/m³)",
    "SO2 (µg/m³)","CO (mg/m³)","Ozone (µg/m³)",
    "hour_sin","hour_cos","dayofweek","month",
    "is_diwali_period","days_since_diwali",
    "is_lohri_period","days_since_lohri",
]
```

```
# 1) Start from df WITH all your engineered features (Diwali + Lohri)
df_model = df.sort_values("Timestamp").copy()

# 2) Drop rows with missing targets first
df_model = df_model.dropna(subset=["PM2.5 (µg/m³)", "PM10 (µg/m³)"])

# 3) Create lag features up to 2 steps back
df_model["PM2.5_t-1"] = df_model["PM2.5 (µg/m³)"].shift(1)
df_model["PM2.5_t-2"] = df_model["PM2.5 (µg/m³)"].shift(2)
df_model["PM10_t-1"] = df_model["PM10 (µg/m³)"].shift(1)
df_model["PM10_t-2"] = df_model["PM10 (µg/m³)"].shift(2)

# 4) Drop rows where any lag is NaN
df_model = df_model.dropna(
    subset=["PM2.5_t-1", "PM2.5_t-2", "PM10_t-1", "PM10_t-2"]
```

```
)

# 5) Define features
feature_cols = [
    "PM2.5_t-1","PM2.5_t-2",
    "PM10_t-1","PM10_t-2",
    "NO (µg/m³)","NO2 (µg/m³)","NH3 (µg/m³)",
    "SO2 (µg/m³)","CO (mg/m³)","Ozone (µg/m³)",
    "hour_sin","hour_cos","dayofweek","month",
    "is_diwali_period","days_since_diwali",
    "is_lohri_period","days_since_lohri",
]

# 6) Train/test split
train_mask = df_model["year"] <= 2023
test_mask  = df_model["year"] >= 2024

X_train = df_model.loc[train_mask, feature_cols]
y25_train = df_model.loc[train_mask, "PM2.5 (µg/m³)"]
y10_train = df_model.loc[train_mask, "PM10 (µg/m³)"]

X_test = df_model.loc[test_mask, feature_cols]
y25_test = df_model.loc[test_mask, "PM2.5 (µg/m³)"]
y10_test = df_model.loc[test_mask, "PM10 (µg/m³)"]
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, r2_score


# make sure df_model and feature_cols already include Lohri features
train_mask = df_model["year"] <= 2023
test_mask  = df_model["year"] >= 2024

X_train = df_model.loc[train_mask, feature_cols]
y25_train = df_model.loc[train_mask, "PM2.5 (µg/m³)"]
y10_train = df_model.loc[train_mask, "PM10 (µg/m³)"]

X_test = df_model.loc[test_mask, feature_cols]
y25_test = df_model.loc[test_mask, "PM2.5 (µg/m³)"]
y10_test = df_model.loc[test_mask, "PM10 (µg/m³)"]

rf25 = RandomForestRegressor(
    n_estimators=150,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)
rf25.fit(X_train, y25_train)

rf10 = RandomForestRegressor(
    n_estimators=150,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)
rf10.fit(X_train, y10_train)

print("PM2.5 test MAE:", mean_absolute_error(y25_test, rf25.predict(X_test)))
print("PM2.5 test R2:",  r2_score(y25_test, rf25.predict(X_test)))
print("PM10 test MAE:", mean_absolute_error(y10_test, rf10.predict(X_test)))
print("PM10 test R2:",  r2_score(y10_test, rf10.predict(X_test)))
```

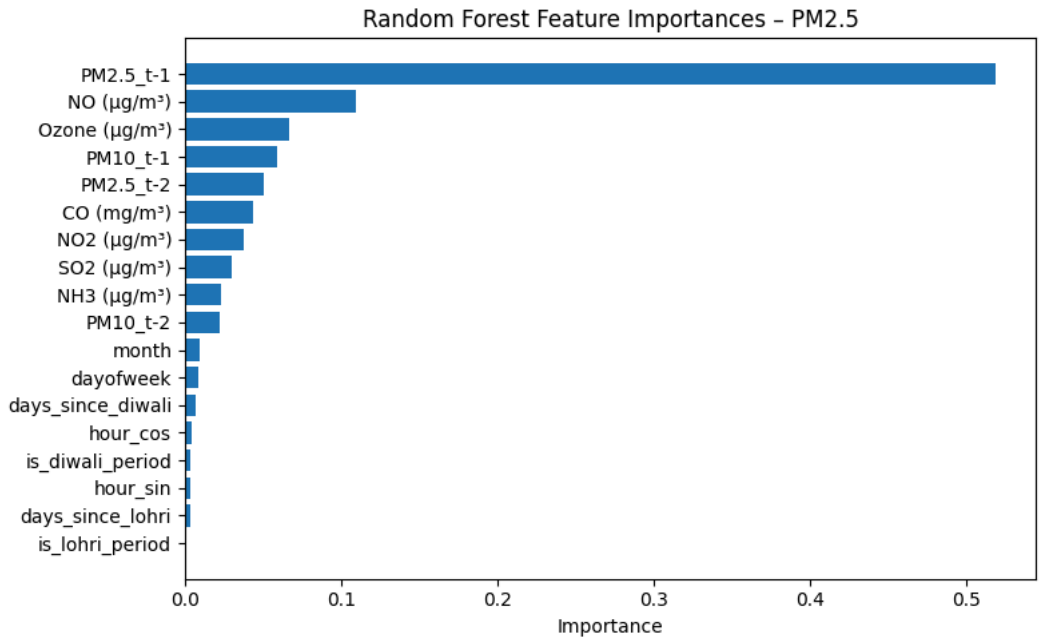
```
PM2.5 test MAE: 13.093398733247469
PM2.5 test R2: 0.503080666527158
PM10 test MAE: 30.53236625527538
PM10 test R2: 0.45989894817627364
```

```
from sklearn.metrics import mean_absolute_error, r2_score

test_all = df_model.copy()
X_all = test_all[feature_cols]
pred25_all = rf25.predict(X_all)
pred10_all = rf10.predict(X_all)
```

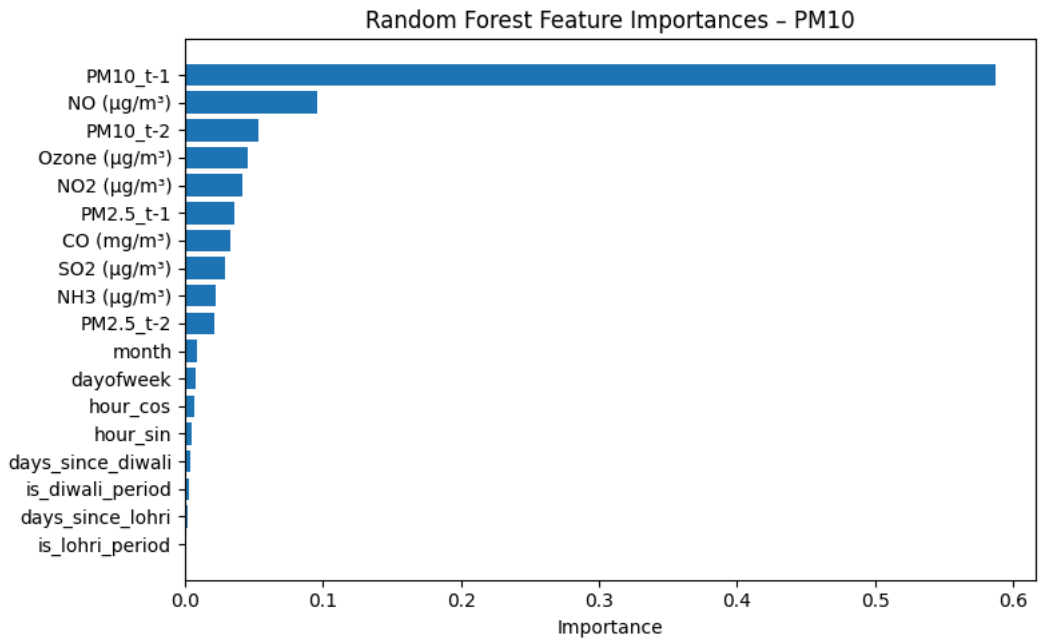
```
for yr in sorted(test_all["year"].unique()):
    m = (test_all["year"] == yr) & (test_all["is_lohri_period"] == 1)
    if m.sum() == 0:
        continue
    y25_y = test_all.loc[m, "PM2.5 (µg/m³)"]
    y10_y = test_all.loc[m, "PM10 (µg/m³)"]
    p25_y = pred25_all[m]
    p10_y = pred10_all[m]

    print(f"Year {yr} Lohri PM2.5 MAE:", mean_absolute_error(y25_y, p25_y))
    print(f"Year {yr} Lohri PM2.5 R2:",  r2_score(y25_y, p25_y))
    print(f"Year {yr} Lohri PM10 MAE:", mean_absolute_error(y10_y, p10_y))
    print(f"Year {yr} Lohri PM10 R2:",  r2_score(y10_y, p10_y))
    print("----")
```

```
Year 2019 Lohri PM2.5 MAE: 13.702078656307858
Year 2019 Lohri PM2.5 R2: 0.5848640908725408
Year 2019 Lohri PM10 MAE: 20.410209001586807
Year 2019 Lohri PM10 R2: 0.4247218177942197
----
Year 2020 Lohri PM2.5 MAE: 9.30389852063901
Year 2020 Lohri PM2.5 R2: 0.5591904880626375
Year 2020 Lohri PM10 MAE: 20.7725381602079
Year 2020 Lohri PM10 R2: 0.5104449882391836
----
Year 2021 Lohri PM2.5 MAE: 14.247140332615665
Year 2021 Lohri PM2.5 R2: 0.617222746820907
Year 2021 Lohri PM10 MAE: 21.29803447079401
Year 2021 Lohri PM10 R2: 0.6816978107348765
----
Year 2022 Lohri PM2.5 MAE: 13.993412539429851
Year 2022 Lohri PM2.5 R2: 0.7611861928034588
Year 2022 Lohri PM10 MAE: 25.81407196964238
Year 2022 Lohri PM10 R2: 0.8047975305298266
----
Year 2023 Lohri PM2.5 MAE: 15.61987174028916
Year 2023 Lohri PM2.5 R2: 0.6339285133626827
Year 2023 Lohri PM10 MAE: 21.734459650252344
Year 2023 Lohri PM10 R2: 0.6968761732917523
----
Year 2024 Lohri PM2.5 MAE: 19.672473079251585
Year 2024 Lohri PM2.5 R2: 0.3175331176669972
Year 2024 Lohri PM10 MAE: 33.38606417019886
Year 2024 Lohri PM10 R2: 0.24394563962860805
----
Year 2025 Lohri PM2.5 MAE: 17.113296789260843
Year 2025 Lohri PM2.5 R2: -0.03161216743397843
Year 2025 Lohri PM10 MAE: 30.881476719331186
Year 2025 Lohri PM10 R2: 0.14462547863732378
----
```

```
import numpy as np
import matplotlib.pyplot as plt

importances = rf25.feature_importances_
idx = np.argsort(importances)

plt.figure(figsize=(8,5))
plt.barh(np.array(feature_cols)[idx], importances[idx])
plt.xlabel("Importance")
plt.title("Random Forest Feature Importances – PM2.5")
plt.tight_layout()
plt.show()
```

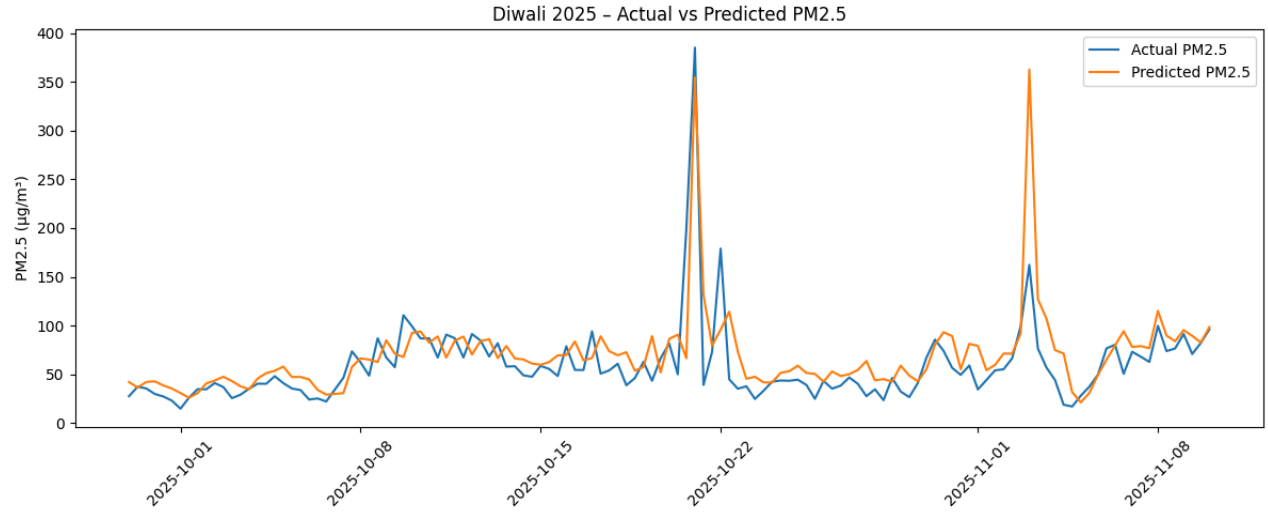


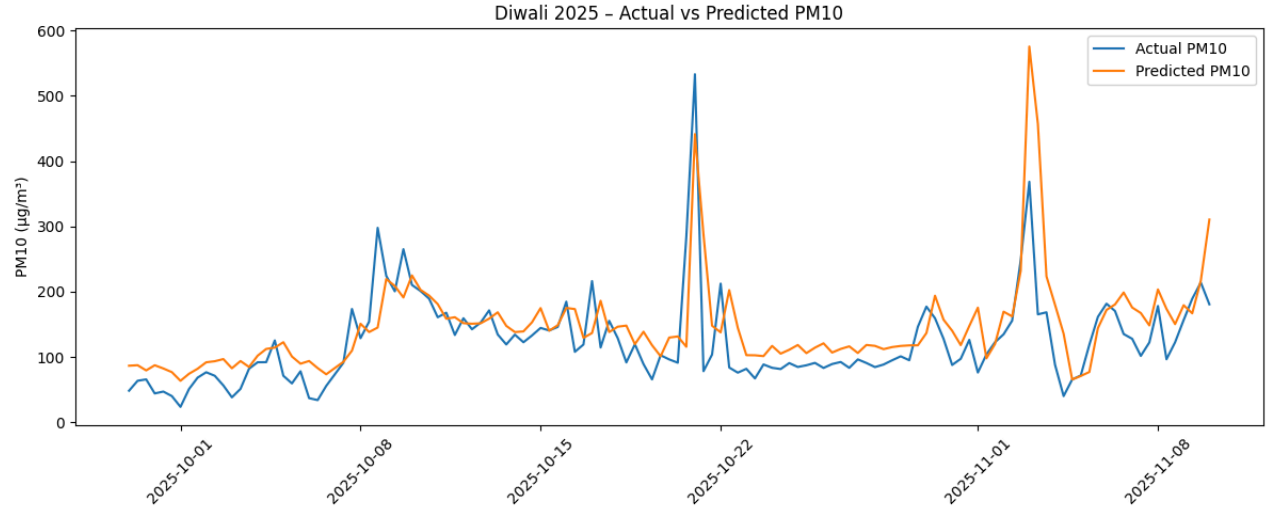

```
import numpy as np
import matplotlib.pyplot as plt

importances = rf10.feature_importances_
idx = np.argsort(importances)

plt.figure(figsize=(8,5))
plt.barh(np.array(feature_cols)[idx], importances[idx])
plt.xlabel("Importance")
plt.title("Random Forest Feature Importances – PM10")
plt.tight_layout()
plt.show()
```




```
test = df_model.copy()
X_all = test[feature_cols]
test["PM2.5_pred"] = rf25.predict(X_all)
test["PM10_pred"] = rf10.predict(X_all)

mask_diwali_2025 = (
    (test["year"] == 2025) &
    (test["is_diwali_period"] == 1)
)

plt.figure(figsize=(12,5))
plt.plot(test["Timestamp"][mask_diwali_2025],
         test["PM2.5 (µg/m³)"][mask_diwali_2025],
```

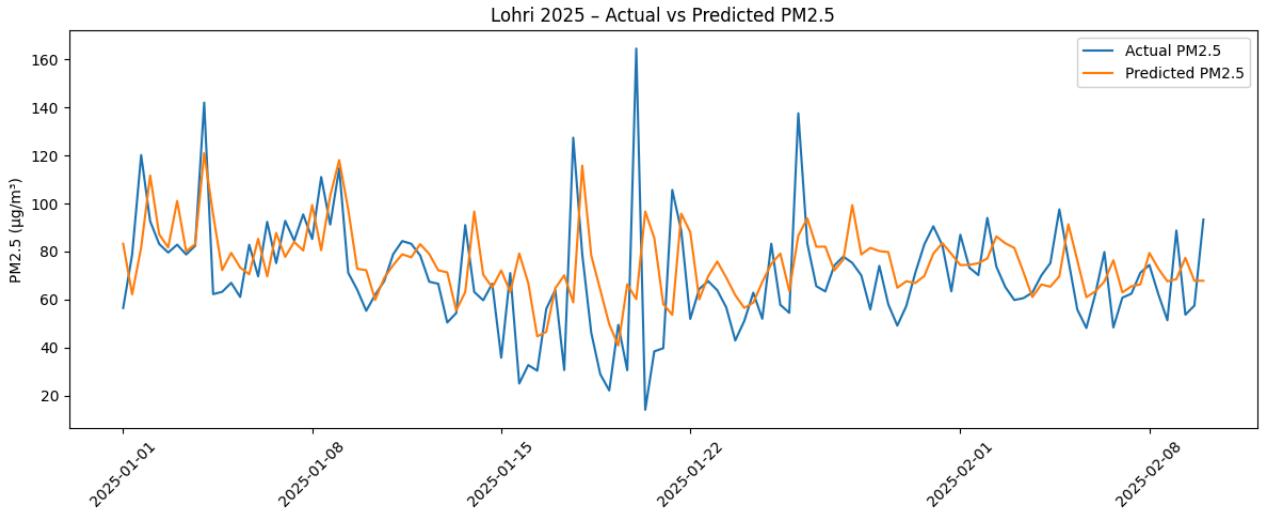
```
         label="Actual PM2.5")
plt.plot(test["Timestamp"][mask_diwali_2025],
         test["PM2.5_pred"][mask_diwali_2025],
         label="Predicted PM2.5")
plt.xticks(rotation=45)
plt.ylabel("PM2.5 (µg/m³)")
plt.title("Diwali 2025 – Actual vs Predicted PM2.5")
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(12,5))
plt.plot(test["Timestamp"][mask_diwali_2025],
         test["PM10 (µg/m³)"][mask_diwali_2025],
         label="Actual PM10")
plt.plot(test["Timestamp"][mask_diwali_2025],
         test["PM10_pred"][mask_diwali_2025],
         label="Predicted PM10")
plt.xticks(rotation=45)
plt.ylabel("PM10 (µg/m³)")
plt.title("Diwali 2025 – Actual vs Predicted PM10")
plt.legend()
plt.tight_layout()
plt.show()
```



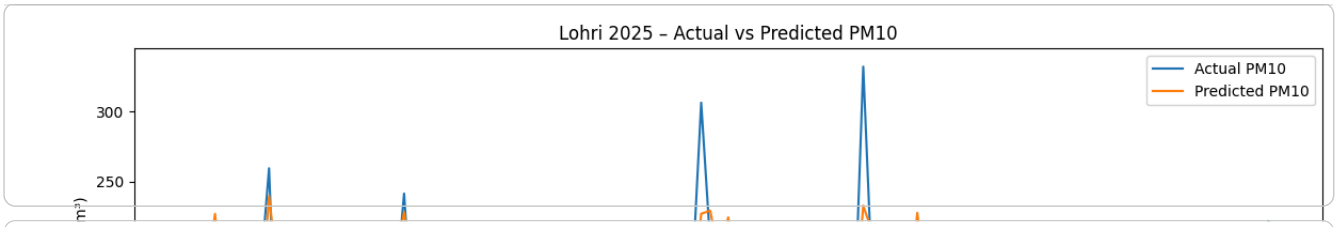



```
mask_lohri_2025 = (
    (test["year"] == 2025) &
    (test["is_lohri_period"] == 1)
```

```
)

plt.figure(figsize=(12,5))
plt.plot(test["Timestamp"][mask_lohri_2025],
         test["PM2.5 (µg/m³)"][mask_lohri_2025],
         label="Actual PM2.5")
plt.plot(test["Timestamp"][mask_lohri_2025],
         test["PM2.5_pred"][mask_lohri_2025],
         label="Predicted PM2.5")
plt.xticks(rotation=45)
plt.ylabel("PM2.5 (µg/m³)")
plt.title("Lohri 2025 – Actual vs Predicted PM2.5")
plt.legend()
plt.tight_layout()
plt.show()
```




```
mask_lohri_2025 = (
    (test["year"] == 2025) &
    (test["is_lohri_period"] == 1)
)

plt.figure(figsize=(12,5))
plt.plot(test["Timestamp"][mask_lohri_2025],
         test["PM10 (µg/m³)"][mask_lohri_2025],
         label="Actual PM10")
plt.plot(test["Timestamp"][mask_lohri_2025],
         test["PM10_pred"][mask_lohri_2025],
         label="Predicted PM10")
plt.xticks(rotation=45)
plt.ylabel("PM10 (µg/m³)")
plt.title("Lohri 2025 – Actual vs Predicted PM10")
plt.legend()
plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt

mask_diwali = test["is_diwali_period"] == 1

plt.figure(figsize=(5,5))
plt.scatter(test["PM2.5 (µg/m³)"][mask_diwali],
            test["PM2.5_pred"][mask_diwali],
            alpha=0.5)
mx = test["PM2.5 (µg/m³)"][mask_diwali].max()
plt.plot([0, mx], [0, mx], "r--")
plt.xlabel("Actual PM2.5")
plt.ylabel("Predicted PM2.5")
plt.title("Diwali (all years) — Actual vs Predicted PM2.5")
plt.tight_layout()
plt.show()
```