Analysis 1

In [ ]:

```
# ------------------------------------------------------------
# STEP 1: Import libraries
# ------------------------------------------------------------
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pathlib import Path

# ------------------------------------------------------------
# STEP 2: Load dataset
# ------------------------------------------------------------
df = pd.read_csv("Dataset.csv")

# ------------------------------------------------------------
# STEP 3: Create filtered copy of DataFrame
# ------------------------------------------------------------
# Columns to drop (case-insensitive match for safety)
cols_to_drop = ["Week Number", "TMAX (°F, weekly mean)", "TMIN (°F, weekly mean)", "Max

# Drop if present
df_filtered = df.copy()
df_filtered = df_filtered.drop(
    columns=[col for col in df_filtered.columns if col.strip().lower() in [c.lower() for
    errors="ignore"
)

#print("Filtered shape:", df_filtered.shape)
print("Dropped columns:", cols_to_drop)

# ------------------------------------------------------------
# STEP 4: Recreate correlation matrix (numeric variables only)
# ------------------------------------------------------------
numeric_df = df_filtered.select_dtypes(include=[np.number])
corr_matrix = numeric_df.corr(method="pearson")

# Save correlation matrix for reference
#corr_matrix.to_csv("Filtered_Correlation_Matrix.csv", index=True)

# ------------------------------------------------------------
# STEP 5: Plot updated heatmap
# ------------------------------------------------------------
plt.figure(figsize=(12, 10))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    center=0,
    square=True,
    linewidths=0.5,
    cbar_kws={"label": "Pearson Correlation", "shrink": 0.8}
)
plt.title("Filtered Correlation Heatmap (Excluding TMAX, TMIN, Max Wind)", fontsize=14,
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

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

In [ ]:

```
# ------------------------------------------------------------
# STEP 1: Import libraries
# ------------------------------------------------------------
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# ------------------------------------------------------------
# STEP 2: Prepare data for VIF
# ------------------------------------------------------------
# Select only numeric predictors (excluding your target variable)
# Replace "Respiratory_ED_Visits" with your actual target column name
target_col = "Respiratory_Weekly_ER_Admissions"

# Drop target variable if present
X = df_filtered.select_dtypes(include=[np.number]).drop(columns=[target_col], errors="ig

# Drop rows with NaN values to avoid errors in VIF computation
X = X.dropna()

# Add constant for intercept (required by VIF calculation)
X_const = add_constant(X)

# ------------------------------------------------------------
# STEP 3: Compute VIF for each feature
# ------------------------------------------------------------
vif_data = pd.DataFrame()
vif_data["Feature"] = X_const.columns
vif_data["VIF"] = [variance_inflation_factor(X_const.values, i)
                   for i in range(X_const.shape[1])]

# Remove the intercept row
vif_data = vif_data[vif_data["Feature"] != "const"].reset_index(drop=True)

# ------------------------------------------------------------
# STEP 4: Identify high-VIF features
# ------------------------------------------------------------
threshold = 5  # Common cutoff; can change to 10 for stricter filtering
high_vif = vif_data[vif_data["VIF"] >= threshold]

# Display results
print("✅ VIF Results:")
print(vif_data.sort_values("VIF", ascending=False))

if not high_vif.empty:
    print("\n⚠ Features with High Multicollinearity (VIF ≥ 5):")
    print(high_vif)
else:
    print("\n✅ No variables exceed the VIF threshold of", threshold)
```

In [ ]:

```
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
import pandas as pd
import numpy as np

# --- Data prep ---
target_col = "Respiratory_Weekly_ER_Admissions"
X = df_filtered.select_dtypes(include=[np.number]).drop(columns=[target_col], errors="ig
y = df_filtered[target_col].loc[X.index]

mask = X.dropna().index
X = X.loc[mask]
y = y.loc[mask]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# --- Train Random Forest ---
rf = RandomForestRegressor(n_estimators=1000, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)

# --- Permutation Importance ---
r = permutation_importance(
    rf, X_test, y_test, n_repeats=10, random_state=42, scoring="r2"
)

# Create importance DataFrame
importance_df = pd.DataFrame({
    "Feature": X.columns,
    "Importance": r.importances_mean,
    "Std": r.importances_std
}).sort_values("Importance", ascending=False)

print("\nPermutation Feature Importance (R² Drop):")
print(importance_df.head(15))

# Optional: visualize top features
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.barh(importance_df["Feature"][:15][::-1], importance_df["Importance"][:15][::-1])
plt.xlabel("Mean R² Decrease (Higher = More Important)")
plt.title("Permutation Feature Importance (Random Forest)")
plt.tight_layout()
plt.show()
```

In [ ]:

```
# ================================================================
# Lag analysis with robust column resolution (includes PM2.5 & Week Number)
# ================================================================
import pandas as pd
import numpy as np
import re
import unicodedata

# ---------- 0) Load ----------
df0 = pd.read_csv("Dataset.csv")
print("Original shape:", df0.shape)

# ---------- 1) Config ----------
TARGET_COL = "Respiratory_Weekly_ER_Admissions"

# Drop only unwanted vars
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
drop_list = [
    "PRCP (inches, weekly sum)",
    "SNOW (inches, weekly sum)",
    "TMAX (°F, weekly mean)",
    "TMIN (°F, weekly mean)",
    "Max Wind Speed (mph, weekly mean)",
    "Avg PM10 (ug/m3 SC)",
    "# Heart disease ER visits (weekly total)",
    "# stroke hospital discharges (weekly total)",
    "# COPD cases (weekly total)"
]

# Desired groups (names are “intents”; we’ll resolve to actual headers)
want_health = ["ILI_Weekly_ER_Admissions", "Asthma_weekly_ER_Admissions"]
want_pollutants = [
    "Avg 8h O3 (ppm)",
    "Avg 1h NO2 (ppb)",
    "Avg PM2.5 (ug/m3 LC)",   # <-- robustly resolved below
    "Avg 1h SO2 (ppb)",
    "8h avg CO (ppm)"
]
want_weather = ["TAVG (°F, weekly mean)", "Avg_WND (mph, weekly mean)"]
want_seasonal = ["Week Number"]  # we will include as lag0 in outputs

# Lag plan targets
LAGS_HEALTH = [0, 1, 2]
LAGS_POLLUTANTS = [0, 1, 2, 3]
LAGS_WEATHER = [0, 1]
LAGS_SEASONAL = [0]  # include Week Number in outputs as lag0

# ---------- 2) Helpers: normalization + robust resolution ----------
def norm(s: str) -> str:
    s0 = unicodedata.normalize("NFKC", str(s))
    s0 = s0.replace("µ", "u").replace("°", "").replace("³", "3")
    s0 = re.sub(r"m²|m\^2", "m2", s0, flags=re.I)
    s0 = re.sub(r"m³|m\^3", "m3", s0, flags=re.I)
    s0 = re.sub(r"\s+", " ", s0.strip()).lower()
    return s0

norm_map = {norm(c): c for c in df0.columns}

def resolve_exact_or_fuzzy(intended: str, fuzzy_patterns=None):
    """Try normalized exact match; if not, try regex patterns against normalized headers
    key = norm(intended)
    if key in norm_map:
        return norm_map[key], "exact"
    if fuzzy_patterns:
        for pat in fuzzy_patterns:
            rx = re.compile(pat)
            for nk, orig in norm_map.items():
                if rx.search(nk):
                    return orig, f"fuzzy:{pat}"
    return None, None

def drop_ci(df, names):
    lowers = {norm(n) for n in names}
    cols_to_drop = [c for c in df.columns if norm(c) in lowers]
    return df.drop(columns=cols_to_drop, errors="ignore")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
# ---------- 3) Drop unwanted ----------
df1 = drop_ci(df0, drop_list)

# ---------- 4) Resolve desired columns robustly ----------
resolved = []

# Health (exact or case-insensitive)
for name in want_health:
    col, how = resolve_exact_or_fuzzy(name)
    if col: resolved.append(col)
    else: print("[Missing health]", name)

# Pollutants (handle PM2.5 fuzzily)
for name in want_pollutants:
    if "pm2.5" in name.lower() or "pm25" in name.lower():
        # match any like: "pm2.5" or "pm25" + optionally spaces + "(ug/m3" or "(µg/m³" e
        fuzzy = [r"pm\s*2\.?5|pm25"]  # first ensure PM2.5 tokens present
        col, how = resolve_exact_or_fuzzy(name, fuzzy_patterns=fuzzy)
        if not col:
            # broader fallback: must contain 'pm' and 'm3'
            col, how = resolve_exact_or_fuzzy(name, fuzzy_patterns=[r"pm.*m3"])
    else:
        col, how = resolve_exact_or_fuzzy(name)
    if col:
        resolved.append(col)
    else:
        print("[Missing pollutant]", name, "— check header spelling/units")

# Weather
for name in want_weather:
    col, how = resolve_exact_or_fuzzy(name)
    if col: resolved.append(col)
    else: print("[Missing weather]", name)

# Seasonal (Week Number)
for name in want_seasonal:
    col, how = resolve_exact_or_fuzzy(name, fuzzy_patterns=[r"\bweek\b"])
    if col: resolved.append(col)
    else: print("[Missing seasonal]", name, "— check exact header (e.g., 'Week Number')"

# Target must exist
tcol, _ = resolve_exact_or_fuzzy(TARGET_COL)
if not tcol:
    raise ValueError(f"Target column '{TARGET_COL}' not found. Check exact name.")
resolved = list(dict.fromkeys([tcol] + resolved))  # de-dup, keep order

df_base = df1[resolved].copy()
print("Base (kept) columns:", df_base.columns.tolist())

# ---------- 5) Coerce numerics (leave Week Number intact) ----------
for c in df_base.columns:
    if c == tcol:
        continue
    if df_base[c].dtype == "object":
        df_base[c] = pd.to_numeric(df_base[c].astype(str).str.replace(",", ""), errors="

# ---------- 6) Build lag plan from actual resolved names ----------
# Identify actual PM2.5 column
pm25_actual = None
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
for c in df_base.columns:
    if re.search(r"pm\s*2\.?5|pm25", norm(c)):
        pm25_actual = c
        break

# Partition lists from actual columns
health_cols   = [c for c in df_base.columns if norm(c) in [norm(x) for x in want_health]
weather_cols  = [c for c in df_base.columns if norm(c) in [norm(x) for x in want_weather
seasonal_cols = [c for c in df_base.columns if norm(c) in [norm(x) for x in want_seasona
pollutant_cols = []
for name in want_pollutants:
    if "pm2.5" in name.lower() or "pm25" in name.lower():
        if pm25_actual and pm25_actual not in pollutant_cols:
            pollutant_cols.append(pm25_actual)
    else:
        # add if present
        hit, _ = resolve_exact_or_fuzzy(name)
        if hit and hit not in pollutant_cols:
            pollutant_cols.append(hit)

# Build lag plan
lag_plan = {}
for c in health_cols:   lag_plan[c] = LAGS_HEALTH
for c in pollutant_cols: lag_plan[c] = LAGS_POLLUTANTS
for c in weather_cols:  lag_plan[c] = LAGS_WEATHER
for c in seasonal_cols: lag_plan[c] = LAGS_SEASONAL  # include Week Number in outputs as

# ---------- 7) Add lags and drop NaNs from lagging ----------
order_key = seasonal_cols[0] if seasonal_cols else None
def add_lags(df_in, lag_plan, order_col=None):
    df = df_in.copy()
    if order_col and order_col in df.columns:
        df = df.sort_values(order_col).reset_index(drop=True)
    for col, lags in lag_plan.items():
        for k in lags:
            df[f"{col}_lag{k}"] = df[col] if k == 0 else df[col].shift(k)
    return df

df_lag = add_lags(df_base, lag_plan, order_col=order_key)
df_lag = df_lag.dropna().reset_index(drop=True)
print("Shape after lagging & dropping NaNs:", df_lag.shape)

# ---------- 8) Correlation tables ----------
records = []
for base_col, lags in lag_plan.items():
    for k in lags:
        col = f"{base_col}_lag{k}"
        if col in df_lag.columns and np.issubdtype(df_lag[col].dtype, np.number):
            corr = df_lag[tcol].corr(df_lag[col])
            records.append({
                "variable_base": base_col,
                "lag": k,
                "corr_with_target": corr,
                "abs_corr": abs(corr)
            })

lag_corr = pd.DataFrame(records).sort_values(["variable_base", "abs_corr"], ascending=[T
lag_corr.to_csv("lag_correlation_long.csv", index=False)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
best_lag = (lag_corr
            .sort_values(["variable_base", "abs_corr"], ascending=[True, False])
            .groupby("variable_base", as_index=False)
            .first()
            .sort_values("abs_corr", ascending=False))
best_lag.to_csv("best_lag_summary.csv", index=False)

print("\nVariables included in lag analysis:", sorted(set(lag_corr["variable_base"])))
print("\nTop 12 strongest (by |corr|) lags:")
print(best_lag.head(12).to_string(index=False))
print("\n✅ Saved:")
print(" - lag_correlation_long.csv  (ALL lags for EVERY kept variable, incl. Week Number
print(" - best_lag_summary.csv      (best lag per variable)")
```

In [ ]:

```
# ============================================================
# Permutation Feature Importance using fuzzy column matching
# ============================================================
import pandas as pd
import numpy as np
import re
import unicodedata
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# ---------- 1) Load dataset ----------
df_full = pd.read_csv("Dataset.csv")
print("Original shape:", df_full.shape)

# ---------- 2) Normalization helper ----------
def norm(s: str) -> str:
    s0 = unicodedata.normalize("NFKC", str(s))
    s0 = s0.replace("µ", "u").replace("°", "").replace("³", "3")
    s0 = re.sub(r"m²|m\^2", "m2", s0, flags=re.I)
    s0 = re.sub(r"m³|m\^3", "m3", s0, flags=re.I)
    s0 = re.sub(r"\s+", " ", s0.strip()).lower()
    return s0

norm_map = {norm(c): c for c in df_full.columns}

def resolve_exact_or_fuzzy(intended: str, fuzzy_patterns=None):
    """Try normalized exact match first, else fuzzy regex search."""
    key = norm(intended)
    if key in norm_map:
        return norm_map[key]
    if fuzzy_patterns:
        for pat in fuzzy_patterns:
            rx = re.compile(pat)
            for nk, orig in norm_map.items():
                if rx.search(nk):
                    return orig
    return None

# ---------- 3) Define best lags ----------
best_lags = {
    "ILI_Weekly_ER_Admissions": 0,
    "Asthma_weekly_ER_Admissions": 0,
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    "8h avg CO (ppm)": 2,
    "Avg 8h O3 (ppm)": 3,
    "Avg 1h NO2 (ppb)": 2,
    "Avg PM2.5 (ug/m3 LC)": 1,
    "Avg 1h SO2 (ppb)": 0,
    "TAVG (°F, weekly mean)": 0,
    "Avg_WND (mph, weekly mean)": 1,
    "Week Number": 0
}
TARGET = "Respiratory_Weekly_ER_Admissions"

# ---------- 4) Resolve all best-lag columns robustly ----------
resolved_lags = {}
for name, lag in best_lags.items():
    # Add fuzzy fallback patterns
    fuzzy = []
    if "pm2.5" in name.lower() or "pm25" in name.lower():
        fuzzy = [r"pm\s*2\.?5|pm25"]
    elif "week" in name.lower():
        fuzzy = [r"week"]
    elif "no2" in name.lower():
        fuzzy = [r"no2"]
    elif "so2" in name.lower():
        fuzzy = [r"so2"]
    elif "co" in name.lower():
        fuzzy = [r"\bco\b"]
    elif "o3" in name.lower():
        fuzzy = [r"o3"]
    elif "tavg" in name.lower():
        fuzzy = [r"tavg"]
    elif "wind" in name.lower() or "wnd" in name.lower():
        fuzzy = [r"wnd|wind"]

    resolved = resolve_exact_or_fuzzy(name, fuzzy)
    if resolved:
        resolved_lags[resolved] = lag
    else:
        print(f"[Warning] Could not match: {name}")

# Resolve target
tcol = resolve_exact_or_fuzzy(TARGET, [r"respiratory"])
if not tcol:
    raise ValueError(f"Target column '{TARGET}' not found!")

print("\nResolved lag columns:")
for k, v in resolved_lags.items():
    print(f" - {k}  (lag {v})")
print(f"\nTarget column: {tcol}")

# ---------- 5) Add lags ----------
def add_lags(df_in, lag_map, order_col=None):
    df = df_in.copy()
    if order_col and order_col in df.columns:
        df = df.sort_values(order_col).reset_index(drop=True)
    for col, lag in lag_map.items():
        new_col = f"{col}_lag{lag}"
        if lag == 0:
            df[new_col] = df[col]
        else:
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
            df[new_col] = df[col].shift(lag)
    return df.dropna().reset_index(drop=True)

# Determine sorting column
order_key = resolve_exact_or_fuzzy("Week Number", [r"week"])
df_lagged = add_lags(df_full, resolved_lags, order_col=order_key)

# ---------- 6) Prepare features ----------
feature_cols = [f"{col}_lag{lag}" for col, lag in resolved_lags.items()]
X = df_lagged[feature_cols]
y = df_lagged[tcol]

# ---------- 7) Train-test split ----------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# ---------- 8) Train Random Forest ----------
rf = RandomForestRegressor(
    n_estimators=500,
    random_state=42,
    min_samples_split=5,
    min_samples_leaf=2,
    n_jobs=-1
)
rf.fit(X_train, y_train)

# ---------- 9) Permutation Importance ----------
result = permutation_importance(
    rf, X_test, y_test, n_repeats=30, random_state=42, scoring="r2"
)
pfi_df = pd.DataFrame({
    "Feature": X.columns,
    "Mean_R2_Drop": result.importances_mean,
    "Std": result.importances_std
}).sort_values("Mean_R2_Drop", ascending=False)

print("\nPermutation Feature Importance (R² drop):")
print(pfi_df)

# ---------- 10) Save + Plot ----------
pfi_df.to_csv("PFI_best_lags.csv", index=False)
print("\n✅ Saved: PFI_best_lags.csv")

plt.figure(figsize=(8,5))
plt.barh(pfi_df["Feature"], pfi_df["Mean_R2_Drop"], xerr=pfi_df["Std"], color="skyblue")
plt.xlabel("Mean R² Drop (Importance)")
plt.ylabel("Feature")
plt.title("Permutation Feature Importance (Best-Lag Variables, Fuzzy Matched)")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

In [ ]:

```
# ============================================================
# Create lagged dataset with final structure (Weather 0–2, AQ 0–3, Health 0–2)
# ============================================================
import pandas as pd
from pathlib import Path

# ---------- 1) Load ----------
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
df = pd.read_csv("Dataset.csv")
df.columns = df.columns.str.strip()

# ---------- 2) Define variables ----------
TARGET = "Respiratory_Weekly_ER_Admissions"

# Health variables (0–2)
'''
health_vars = [
    "ILI_Weekly_ER_Admissions",
    "Asthma_weekly_ER_Admissions"
]
'''

# Air quality variables (0–3)
air_vars = [
    "8h avg CO (ppm)",
    "Avg 8h O3 (ppm)",
    "Avg 1h NO2 (ppb)",
    "Avg PM2.5 (ug/m3 LC)",
    "Avg 1h SO2 (ppb)"
]

# Weather variables (0–2)
weather_vars = [
    "TAVG (°F, weekly mean)",
    "Avg_WND (mph, weekly mean)"
]

time_col = "Week Number"

# ---------- 3) Lag configuration ----------
#LAGS_HEALTH = [0, 1, 2]
LAGS_AIR = [0, 1, 2, 3]
LAGS_WEATHER = [0, 1, 2]

# ---------- 4) Helper to add lags ----------
def add_lags(df, col_list, lags):
    for col in col_list:
        if col not in df.columns:
            print(f"⚠ Skipping missing column: {col}")
            continue
        for k in lags:
            df[f"{col}_lag{k}"] = df[col].shift(k)
    return df

# ---------- 5) Sort, apply lags ----------
df = df.sort_values(time_col).reset_index(drop=True)
#df = add_lags(df, health_vars, LAGS_HEALTH)
df = add_lags(df, air_vars, LAGS_AIR)
df = add_lags(df, weather_vars, LAGS_WEATHER)

# ---------- 6) Drop NaN rows from lagging ----------
df_lagged = df.dropna().reset_index(drop=True)

# ---------- 7) Save output ----------
out_csv = Path("Dataset_with_lags_v2.csv")
df_lagged.to_csv(out_csv, index=False)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
print("✅ Saved:", out_csv)
print("Shape after lagging:", df_lagged.shape)
print("Columns generated:", len(df_lagged.columns))
```

In [ ]:

```
# ============================================================
# Gaussian GAM (LinearGAM) — AR(1,2), strong λ, robust NaN/Inf handling
# Train 1..(N-last_n), Test last_n weeks
# ============================================================
import pandas as pd, numpy as np, re
from pygam import LinearGAM, s
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# ---------- params ----------
LAST_N   = 40
ROLL_WIN = 2        # 1 = no smoothing
LAM_GRID = [10, 30, 100, 300, 1000]

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

def normalize_cols(df):
    ren = {}
    for c in df.columns:
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
            ren[c] = re.sub(r"pm\s*2\.?5|pm25","PM2.5",c,flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df

df = normalize_cols(df)
TARGET = "Respiratory_Weekly_ER_Admissions"

# ---------- add AR(1), AR(2) ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- optional light denoise of target ----------
if ROLL_WIN > 1:
    before = len(df)
    df[TARGET] = df[TARGET].rolling(ROLL_WIN, center=True, min_periods=ROLL_WIN).mean()
    print(f"Rolled target (win={ROLL_WIN}). Edge rows added NaNs: {before - len(df.dropn

# ---------- keep predictors ----------
predictors = [
    "Avg 8h O3 (ppm)_lag3",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h NO2 (ppb)_lag2",
    "8h avg CO (ppm)_lag2",
    "Avg 1h SO2 (ppb)_lag0",
    "TAVG (°F, weekly mean)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "Week Number",
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    "Resp_lag1",
    "Resp_lag2",
]
missing = [c for c in predictors if c not in df.columns]
if missing:
    raise KeyError(f"Missing predictors: {missing}")

# ---------- build clean modeling frame (drop NaN/Inf) ----------
model_cols = predictors + [TARGET]
dfm = df[model_cols].copy()

# Replace ±Inf -> NaN, then drop any NaN rows
dfm = dfm.replace([np.inf, -np.inf], np.nan)
nan_counts = dfm.isna().sum().sort_values(ascending=False)
if nan_counts.max() > 0:
    print("NaN count per column before dropna():")
    print(nan_counts[nan_counts > 0])

dfm = dfm.dropna().reset_index(drop=True)

# Sort by Week Number (safe now because NaNs removed)
if "Week Number" not in dfm.columns:
    raise KeyError("Expected 'Week Number' in dataset.")
dfm = dfm.sort_values("Week Number").reset_index(drop=True)

# ---------- split ----------
N = len(dfm); train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows after cleaning. Got N={N}, LAST_N={LAST_N}")

X = dfm[predictors].copy()
y = dfm[TARGET].copy()
wk = dfm["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")

# Final NaN/Inf safety check
assert np.isfinite(X_train.values).all() and np.isfinite(X_test.values).all(), "X contai
assert np.isfinite(y_train.values).all() and np.isfinite(y_test.values).all(), "y contai

# ---------- scale X ----------
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.values)
X_test_s  = scaler.transform(X_test.values)

# ---------- fit LinearGAM on log1p(y) ----------
y_train_log = np.log1p(y_train.values)

terms = s(0)
for i in range(1, X_train_s.shape[1]):
    terms += s(i)

gam = LinearGAM(terms).gridsearch(X_train_s, y_train_log, lam=LAM_GRID)
print("Chosen λ:", gam.lam)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
# ---------- predict & evaluate ----------
y_pred = np.expm1(gam.predict(X_test_s))
r2   = r2_score(y_test.values, y_pred)
rmse = np.sqrt(mean_squared_error(y_test.values, y_pred))
mae  = mean_absolute_error(y_test.values, y_pred)

print("\n📊 Gaussian GAM — AR(1,2); robust cleaning")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- plot ----------
plt.figure(figsize=(9,5))
plt.plot(wk_test.values, y_test.values, label=f"Actual (last {LAST_N} weeks)", marker="o
plt.plot(wk_test.values, y_pred,       label="Predicted (LinearGAM)", marker="s", color=
plt.title(f"Gaussian GAM — Last {LAST_N} Weeks\n(AR(1,2), X standardized, strong smoothi
plt.xlabel("Week Number"); plt.ylabel("Weekly ER Visits"); plt.legend(); plt.tight_layou

# ---------- save ----------
pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_GaussianGAM": y_pred
}).to_csv(f"GaussianGAM_holdout_last{LAST_N}_AR_clean.csv", index=False)
print(f"\n✅ Saved: GaussianGAM_holdout_last{LAST_N}_AR_clean.csv")
```

In [ ]:

```
# ============================================================
# Gaussian GAM (LinearGAM) — AR(1,2), strong λ, robust NaN/Inf handling
# Train 1..(N-last_n), Test last_n weeks
# Full-series plot with COVID highlight
# ============================================================
import pandas as pd, numpy as np, re
from pygam import LinearGAM, s
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# ---------- params ----------
LAST_N   = 40
ROLL_WIN = 2        # 1 = no smoothing
LAM_GRID = [10, 30, 100, 300, 1000]

# Week-number mapping (1 = first week of 2015)
# 2015: 1–52, 2016: 53–104, 2017: 105–156, 2018: 157–208, 2019: 209–260
# 2020: 261–312, 2021: 313–364, 2022: 365–416, 2023: 417–468, 2024: 469–520
# Approx COVID main period: mid-March 2020 to end of 2021
COVID_START_WEEK = 271   # ~ mid-March 2020
COVID_END_WEEK   = 364   # end of 2021

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

def normalize_cols(df):
    ren = {}
    for c in df.columns:
        cl = c.lower()
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
        if "pm2.5" in cl or "pm25" in cl:
            ren[c] = re.sub(r"pm\s*2\.?5|pm25","PM2.5",c,flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df

df = normalize_cols(df)
TARGET = "Respiratory_Weekly_ER_Admissions"

# ---------- add AR(1), AR(2) ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- optional light denoise of target ----------
if ROLL_WIN > 1:
    before = len(df)
    df[TARGET] = df[TARGET].rolling(ROLL_WIN, center=True, min_periods=ROLL_WIN).mean()
    print(f"Rolled target (win={ROLL_WIN}). Edge rows added NaNs: {before - len(df.dropn

# ---------- keep predictors ----------
predictors = [
    "Avg 8h O3 (ppm)_lag3",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h NO2 (ppb)_lag2",
    "8h avg CO (ppm)_lag2",
    "Avg 1h SO2 (ppb)_lag0",
    "TAVG (°F, weekly mean)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "Week Number",
    "Resp_lag1",
    "Resp_lag2",
]
missing = [c for c in predictors if c not in df.columns]
if missing:
    raise KeyError(f"Missing predictors: {missing}")

# ---------- build clean modeling frame (drop NaN/Inf) ----------
model_cols = predictors + [TARGET]
dfm = df[model_cols].copy()

# Replace ±Inf -> NaN, then drop any NaN rows
dfm = dfm.replace([np.inf, -np.inf], np.nan)
nan_counts = dfm.isna().sum().sort_values(ascending=False)
if nan_counts.max() > 0:
    print("NaN count per column before dropna():")
    print(nan_counts[nan_counts > 0])

dfm = dfm.dropna().reset_index(drop=True)

# Sort by Week Number (safe now because NaNs removed)
if "Week Number" not in dfm.columns:
    raise KeyError("Expected 'Week Number' in dataset.")
dfm = dfm.sort_values("Week Number").reset_index(drop=True)

# ---------- split ----------
N = len(dfm)
train_end = N - LAST_N
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
if train_end <= 0:
    raise ValueError(f"Not enough rows after cleaning. Got N={N}, LAST_N={LAST_N}")

X  = dfm[predictors].copy()
y  = dfm[TARGET].copy()
wk = dfm["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")

# Final NaN/Inf safety check
assert np.isfinite(X_train.values).all() and np.isfinite(X_test.values).all(), "X contai
assert np.isfinite(y_train.values).all() and np.isfinite(y_test.values).all(), "y contai

# ---------- scale X ----------
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.values)
X_test_s  = scaler.transform(X_test.values)

# ---------- fit LinearGAM on log1p(y) ----------
y_train_log = np.log1p(y_train.values)

terms = s(0)
for i in range(1, X_train_s.shape[1]):
    terms += s(i)

gam = LinearGAM(terms).gridsearch(X_train_s, y_train_log, lam=LAM_GRID)
print("Chosen λ:", gam.lam)

# ---------- predict & evaluate ----------
y_pred = np.expm1(gam.predict(X_test_s))
r2   = r2_score(y_test.values, y_pred)
rmse = np.sqrt(mean_squared_error(y_test.values, y_pred))
mae  = mean_absolute_error(y_test.values, y_pred)

print("\n📊 Gaussian GAM — AR(1,2); robust cleaning")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- FULL SERIES PLOT with COVID Highlight ----------
plt.figure(figsize=(11,6))
ax = plt.gca()

# COVID shading (in Week Number units)
ax.axvspan(COVID_START_WEEK, COVID_END_WEEK,
           alpha=0.15, color="red", label="COVID period", zorder=0)

# All actual ER visits, 2015–2024
ax.plot(wk.values, y.values,
        color="black", linewidth=1.3, marker="o", markersize=3,
        label="Actual (2015–2024)", zorder=2)

# Predicted vs actual for the last LAST_N weeks
ax.plot(wk_test.values, y_test.values,
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
        color="tab:blue", linewidth=1.8, marker="o", markersize=5,
        label=f"Actual (last {LAST_N} weeks)", zorder=3)

ax.plot(wk_test.values, y_pred,
        color="tab:orange", linewidth=1.8, marker="s", markersize=5,
        label=f"Predicted (LinearGAM, last {LAST_N} weeks)", zorder=4)

ax.set_title("Gaussian GAM — Full Series (2015–2024)\nAR(1,2), Standardized Predictors,
ax.set_xlabel("Week Number")
ax.set_ylabel("Weekly ER Visits")
ax.legend()
plt.tight_layout()
plt.show()

# ---------- save ----------
pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_GaussianGAM": y_pred
}).to_csv(f"GaussianGAM_holdout_last{LAST_N}_AR_clean.csv", index=False)
print(f"\n✅ Saved: GaussianGAM_holdout_last{LAST_N}_AR_clean.csv")
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# --- Compute TRAIN residuals (original scale) ---
y_train_pred_log = gam.predict(X_train_s)
y_train_pred = np.expm1(y_train_pred_log)
res_train = y_train.values - y_train_pred   # residuals on original scale

# --- Settings ---
nlags = 30
N = len(res_train)
crit = 1.96 / np.sqrt(N)  # 95% bounds for alpha = 0.05

# ========================
# ACF plot (Train Residuals)
# ========================
plt.figure(figsize=(8,4))
plot_acf(res_train, lags=nlags, alpha=None)
plt.hlines([crit, -crit], xmin=-0.5, xmax=nlags+0.5, linestyles="dashed", colors="red")
plt.hlines(0, xmin=-0.5, xmax=nlags+0.5, colors="black", linewidth=1)
plt.title("ACF — Training Residuals (Original Scale)")
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.tight_layout()
plt.show()

# ========================
# PACF plot (Train Residuals)
# ========================
plt.figure(figsize=(8,4))
plot_pacf(res_train, lags=nlags, method="ywm", alpha=None)
plt.hlines([crit, -crit], xmin=-0.5, xmax=nlags+0.5, linestyles="dashed", colors="red")
plt.hlines(0, xmin=-0.5, xmax=nlags+0.5, colors="black", linewidth=1)
plt.title("PACF — Training Residuals (Original Scale)")
plt.xlabel("Lag")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
plt.ylabel("Partial Autocorrelation")
plt.tight_layout()
plt.show()
```

In [ ]:

```
# ============================================================
# Gaussian GAM (LinearGAM) — AR(1,2), strong λ, robust NaN/Inf handling
# Train 1..(N-last_n), Test last_n weeks
# ============================================================
import pandas as pd, numpy as np, re
from pygam import LinearGAM, s
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# ---------- params ----------
LAST_N   = 40
ROLL_WIN = 1        # 1 = no smoothing
LAM_GRID = [10, 30, 100, 300, 1000]

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

def normalize_cols(df):
    ren = {}
    for c in df.columns:
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
            ren[c] = re.sub(r"pm\s*2\.?5|pm25","PM2.5",c,flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df

df = normalize_cols(df)
TARGET = "Respiratory_Weekly_ER_Admissions"

# ---------- add AR(1), AR(2) ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- optional light denoise of target ----------
if ROLL_WIN > 1:
    before = len(df)
    df[TARGET] = df[TARGET].rolling(ROLL_WIN, center=True, min_periods=ROLL_WIN).mean()
    print(f"Rolled target (win={ROLL_WIN}). Edge rows added NaNs: {before - len(df.dropn

# ---------- keep predictors ----------
predictors = [
    "Avg 8h O3 (ppm)_lag3",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h NO2 (ppb)_lag2",
    "8h avg CO (ppm)_lag2",
    "Avg 1h SO2 (ppb)_lag0",
    "TAVG (°F, weekly mean)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "Week Number",
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    "Resp_lag1",
    "Resp_lag2",
]
missing = [c for c in predictors if c not in df.columns]
if missing:
    raise KeyError(f"Missing predictors: {missing}")

# ---------- build clean modeling frame (drop NaN/Inf) ----------
model_cols = predictors + [TARGET]
dfm = df[model_cols].copy()

# Replace ±Inf -> NaN, then drop any NaN rows
dfm = dfm.replace([np.inf, -np.inf], np.nan)
nan_counts = dfm.isna().sum().sort_values(ascending=False)
if nan_counts.max() > 0:
    print("NaN count per column before dropna():")
    print(nan_counts[nan_counts > 0])

dfm = dfm.dropna().reset_index(drop=True)

# Sort by Week Number (safe now because NaNs removed)
if "Week Number" not in dfm.columns:
    raise KeyError("Expected 'Week Number' in dataset.")
dfm = dfm.sort_values("Week Number").reset_index(drop=True)

# ---------- split ----------
N = len(dfm); train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows after cleaning. Got N={N}, LAST_N={LAST_N}")

X = dfm[predictors].copy()
y = dfm[TARGET].copy()
wk = dfm["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")

# Final NaN/Inf safety check
assert np.isfinite(X_train.values).all() and np.isfinite(X_test.values).all(), "X contai
assert np.isfinite(y_train.values).all() and np.isfinite(y_test.values).all(), "y contai

# ---------- scale X ----------
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.values)
X_test_s  = scaler.transform(X_test.values)

# ---------- fit LinearGAM on log1p(y) ----------
y_train_log = np.log1p(y_train.values)

terms = s(0)
for i in range(1, X_train_s.shape[1]):
    terms += s(i)

gam = LinearGAM(terms).gridsearch(X_train_s, y_train_log, lam=LAM_GRID)
print("Chosen λ:", gam.lam)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
# ---------- predict & evaluate ----------
y_pred = np.expm1(gam.predict(X_test_s))
r2   = r2_score(y_test.values, y_pred)
rmse = np.sqrt(mean_squared_error(y_test.values, y_pred))
mae  = mean_absolute_error(y_test.values, y_pred)

print("\n📊 Gaussian GAM — AR(1,2); robust cleaning")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- plot ----------
plt.figure(figsize=(9,5))
plt.plot(wk_test.values, y_test.values, label=f"Actual (last {LAST_N} weeks)", marker="o
plt.plot(wk_test.values, y_pred,       label="Predicted (LinearGAM)", marker="s", color=
plt.title(f"Gaussian GAM — Last {LAST_N} Weeks\n(AR(1,2), X standardized, strong smoothi
plt.xlabel("Week Number"); plt.ylabel("Weekly ER Visits"); plt.legend(); plt.tight_layou

# ---------- save ----------
pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_GaussianGAM": y_pred
}).to_csv(f"GaussianGAM_holdout_last{LAST_N}_AR_clean.csv", index=False)
print(f"\n✅ Saved: GaussianGAM_holdout_last{LAST_N}_AR_clean.csv")
```

In [ ]:

```
# ============================================================
# Gaussian GAM (LinearGAM) — AR(1,2), strong λ, robust NaN/Inf handling
# Train 1..(N-last_n), Test last_n weeks
# Full-series plot with COVID highlight
# ============================================================
import pandas as pd, numpy as np, re
from pygam import LinearGAM, s
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# ---------- params ----------
LAST_N   = 40
ROLL_WIN = 1        # 1 = no smoothing
LAM_GRID = [10, 30, 100, 300, 1000]

# COVID period in "Week Number" units (1 = first week of 2015)
# 2015: 1–52, 2016: 53–104, 2017: 105–156, 2018: 157–208, 2019: 209–260
# 2020: 261–312, 2021: 313–364, 2022: 365–416, 2023: 417–468, 2024: 469–520
COVID_START_WEEK = 271   # ~ mid-March 2020
COVID_END_WEEK   = 364   # end of 2021

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

def normalize_cols(df):
    ren = {}
    for c in df.columns:
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
            ren[c] = re.sub(r"pm\s*2\.?5|pm25","PM2.5",c,flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df

df = normalize_cols(df)
TARGET = "Respiratory_Weekly_ER_Admissions"

# ---------- add AR(1), AR(2) ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- optional light denoise of target ----------
if ROLL_WIN > 1:
    before = len(df)
    df[TARGET] = df[TARGET].rolling(ROLL_WIN, center=True, min_periods=ROLL_WIN).mean()
    print(f"Rolled target (win={ROLL_WIN}). Edge rows added NaNs: {before - len(df.dropn

# ---------- keep predictors ----------
predictors = [
    "Avg 8h O3 (ppm)_lag3",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h NO2 (ppb)_lag2",
    "8h avg CO (ppm)_lag2",
    "Avg 1h SO2 (ppb)_lag0",
    "TAVG (°F, weekly mean)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "Week Number",
    "Resp_lag1",
    "Resp_lag2",
]
missing = [c for c in predictors if c not in df.columns]
if missing:
    raise KeyError(f"Missing predictors: {missing}")

# ---------- build clean modeling frame (drop NaN/Inf) ----------
model_cols = predictors + [TARGET]
dfm = df[model_cols].copy()

# Replace ±Inf -> NaN, then drop any NaN rows
dfm = dfm.replace([np.inf, -np.inf], np.nan)
nan_counts = dfm.isna().sum().sort_values(ascending=False)
if nan_counts.max() > 0:
    print("NaN count per column before dropna():")
    print(nan_counts[nan_counts > 0])

dfm = dfm.dropna().reset_index(drop=True)

# Sort by Week Number (safe now because NaNs removed)
if "Week Number" not in dfm.columns:
    raise KeyError("Expected 'Week Number' in dataset.")
dfm = dfm.sort_values("Week Number").reset_index(drop=True)

# ---------- split ----------
N = len(dfm); train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows after cleaning. Got N={N}, LAST_N={LAST_N}")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
X = dfm[predictors].copy()
y = dfm[TARGET].copy()
wk = dfm["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")

# Final NaN/Inf safety check
assert np.isfinite(X_train.values).all() and np.isfinite(X_test.values).all(), "X contai
assert np.isfinite(y_train.values).all() and np.isfinite(y_test.values).all(), "y contai

# ---------- scale X ----------
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.values)
X_test_s  = scaler.transform(X_test.values)

# ---------- fit LinearGAM on log1p(y) ----------
y_train_log = np.log1p(y_train.values)

terms = s(0)
for i in range(1, X_train_s.shape[1]):
    terms += s(i)

gam = LinearGAM(terms).gridsearch(X_train_s, y_train_log, lam=LAM_GRID)
print("Chosen λ:", gam.lam)

# ---------- predict & evaluate ----------
y_pred = np.expm1(gam.predict(X_test_s))
r2   = r2_score(y_test.values, y_pred)
rmse = np.sqrt(mean_squared_error(y_test.values, y_pred))
mae  = mean_absolute_error(y_test.values, y_pred)

print("\n📊 Gaussian GAM — AR(1,2); robust cleaning")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- FULL-SERIES PLOT with COVID highlight ----------
plt.figure(figsize=(11,6))
ax = plt.gca()

# Shade COVID period on the full x-axis
ax.axvspan(COVID_START_WEEK, COVID_END_WEEK,
           alpha=0.15, color="red", label="COVID period", zorder=0)

# Plot full actual series (2015–2024)
ax.plot(wk.values, y.values,
        color="black", linewidth=1.3, marker="o", markersize=3,
        label="Actual (2015–2024)", zorder=2)

# Overlay last LAST_N weeks: actual vs predicted
ax.plot(wk_test.values, y_test.values,
        color="tab:blue", linewidth=1.8, marker="o", markersize=5,
        label=f"Actual (last {LAST_N} weeks)", zorder=3)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
ax.plot(wk_test.values, y_pred,
        color="tab:orange", linewidth=1.8, marker="s", markersize=5,
        label=f"Predicted (last {LAST_N} weeks)", zorder=4)

ax.set_title("Gaussian GAM — Full Series (2015–2024)\nAR(1,2), Standardized Predictors,
ax.set_xlabel("Week Number")
ax.set_ylabel("Weekly ER Visits")
ax.legend()
plt.tight_layout()
plt.show()

# ---------- save ----------
pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_GaussianGAM": y_pred
}).to_csv(f"GaussianGAM_holdout_last{LAST_N}_AR_clean.csv", index=False)
print(f"\n✅ Saved: GaussianGAM_holdout_last{LAST_N}_AR_clean.csv")
```

In [ ]:

```
# ============================================================
# CatBoost baseline — no grid search, no AR terms, no MA(y)
# Temporal split: train 1..(N-LAST_N), test last LAST_N weeks
# ============================================================
import pandas as pd, numpy as np, re
from catboost import CatBoostRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# ---------- config ----------
LAST_N = 40              # change to 40 or 50 if you want
TARGET = "Respiratory_Weekly_ER_Admissions"
RANDOM_STATE = 42

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

# normalize common tricky column names (no transformations)
def normalize_cols(df):
    ren = {}
    for c in df.columns:
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
            ren[c] = re.sub(r"pm\s*2\.?5|pm25", "PM2.5", c, flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            # keep any lag suffix if present
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df

df = normalize_cols(df)

# ---------- clean: replace Inf, drop NaNs from lag creation ----------
df = df.replace([np.inf, -np.inf], np.nan).dropna().reset_index(drop=True)

# must have Week Number to keep temporal order (ok to use as a feature)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
if "Week Number" not in df.columns:
    raise KeyError("Expected 'Week Number' in dataset.")

df = df.sort_values("Week Number").reset_index(drop=True)

# ---------- build feature set ----------
# "Pure" setup: use all numeric columns EXCEPT:
#   - the target
#   - any explicit AR features of the target (Resp_lag*)
# (We will keep Week Number as a simple temporal feature.)
drop_exact = {TARGET}
drop_prefixes = ("Resp_lag",)  # exclude any AR terms if present

numeric_cols = [c for c in df.columns if np.issubdtype(df[c].dtype, np.number)]
feature_cols = []
for c in numeric_cols:
    if c in drop_exact:
        continue
    if any(c.startswith(p) for p in drop_prefixes):
        continue
    feature_cols.append(c)

# sanity
if TARGET not in df.columns:
    raise KeyError(f"Target '{TARGET}' not found in the dataset.")
if len(feature_cols) == 0:
    raise ValueError("No features left after filtering. Check column names.")

# ---------- temporal split ----------
N = len(df)
train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows for temporal split. N={N}, LAST_N={LAST_N}")

X = df[feature_cols].copy()
y = df[TARGET].copy()
wk = df["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")
print(f"Using {len(feature_cols)} features.")

# ---------- CatBoost (simple baseline, no tuning) ----------
# Notes:
#  - iterations & early_stopping help avoid severe overfit without grid search
#  - no categorical handling (we're numeric-only here)
cb = CatBoostRegressor(
    loss_function="RMSE",
    iterations=1000,
    learning_rate=0.05,
    depth=6,
    random_seed=RANDOM_STATE,
    early_stopping_rounds=50,
    verbose=False
)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
cb.fit(X_train, y_train, eval_set=(X_test, y_test), use_best_model=True)

# ---------- evaluate ----------
y_pred = cb.predict(X_test)
r2   = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)

print("\n📊 CatBoost Baseline — no AR, no MA(y), no tuning")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- plot ----------
plt.figure(figsize=(10,5))
plt.plot(wk_test.values, y_test.values, label=f"Actual (last {LAST_N} weeks)", marker="o
plt.plot(wk_test.values, y_pred,       label="Predicted (CatBoost baseline)", marker="s"
plt.title(f"CatBoost Baseline — Last {LAST_N} Weeks\n(no AR terms, no MA target, no tuni
plt.xlabel("Week Number"); plt.ylabel("Weekly ER Visits"); plt.legend(); plt.tight_layou

# ---------- save ----------
out = pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_CatBoost": y_pred
})
out.to_csv(f"CatBoost_baseline_last{LAST_N}.csv", index=False)
print(f"\n✅ Saved: CatBoost_baseline_last{LAST_N}.csv")
```

In [ ]:

```
# ============================================================
# CatBoost baseline — no grid search, no AR terms, no MA(y)
# Temporal split: train 1..(N-LAST_N), test last LAST_N weeks
# Full-series plot with COVID highlight
# ============================================================
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from catboost import CatBoostRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# ---------- config ----------
LAST_N       = 40   # number of weeks in holdout set
TARGET       = "Respiratory_Weekly_ER_Admissions"
RANDOM_STATE = 42

# COVID period in "Week Number" units (1 = first week of 2015)
# 2015: 1–52, 2016: 53–104, 2017: 105–156, 2018: 157–208, 2019: 209–260
# 2020: 261–312, 2021: 313–364, 2022: 365–416, 2023: 417–468, 2024: 469–520
COVID_START_WEEK = 271   # ~ mid-March 2020
COVID_END_WEEK   = 364   # end of 2021

# ---------- helper: normalize column names (no value transforms) ----------
def normalize_cols(df: pd.DataFrame) -> pd.DataFrame:
    """
    Normalize some common tricky column names:
    - PM2.5 / pm25 -> 'PM2.5'
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    - avg_wnd / avg wind -> 'Avg_WND (mph, weekly mean)[_lagX]'
    """
    rename_dict = {}
    for col in df.columns:
        cl = col.lower()
        # unify PM2.5 naming
        if "pm2.5" in cl or "pm25" in cl:
            rename_dict[col] = re.sub(r"pm\s*2\.?5|pm25", "PM2.5", col, flags=re.I)

        # unify wind naming, preserving lag suffix if any
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", col, flags=re.I)
            suffix = m.group(1) if m else ""
            rename_dict[col] = f"Avg_WND (mph, weekly mean){suffix}"

    return df.rename(columns=rename_dict) if rename_dict else df


# ---------- load & basic cleaning ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()
df = normalize_cols(df)

# Replace ±Inf with NaN, then drop any NaN rows (from lags etc.)
df = df.replace([np.inf, -np.inf], np.nan).dropna().reset_index(drop=True)

# Need Week Number for temporal order
if "Week Number" not in df.columns:
    raise KeyError("Expected 'Week Number' in dataset.")

# Sort chronologically
df = df.sort_values("Week Number").reset_index(drop=True)

# ---------- feature construction ----------
# Use all numeric columns except:
#   - the target
#   - any explicit AR terms of the target (Resp_lag*)
numeric_cols = [
    c for c in df.columns
    if np.issubdtype(df[c].dtype, np.number)
]

drop_exact    = {TARGET}
drop_prefixes = ("Resp_lag",)

feature_cols = []
for col in numeric_cols:
    if col in drop_exact:
        continue
    if any(col.startswith(pref) for pref in drop_prefixes):
        continue
    feature_cols.append(col)

if TARGET not in df.columns:
    raise KeyError(f"Target '{TARGET}' not found in dataset.")
if not feature_cols:
    raise ValueError("No features left after filtering. Check column names.")

# ---------- temporal split ----------
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
N = len(df)
train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows for temporal split. N={N}, LAST_N={LAST_N}")

X  = df[feature_cols].copy()
y  = df[TARGET].copy()
wk = df["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test         = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")
print(f"Using {len(feature_cols)} features.")

# ---------- CatBoost baseline model ----------
cb = CatBoostRegressor(
    loss_function="RMSE",
    iterations=1000,
    learning_rate=0.05,
    depth=6,
    random_seed=RANDOM_STATE,
    early_stopping_rounds=50,
    verbose=False
)

cb.fit(X_train, y_train, eval_set=(X_test, y_test), use_best_model=True)

# ---------- evaluation ----------
y_pred = cb.predict(X_test)

r2   = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)

print("\n📊 CatBoost Baseline — no AR, no MA(y), no tuning")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- FULL-SERIES PLOT with COVID highlight ----------
plt.figure(figsize=(11, 6))
ax = plt.gca()

# Shade COVID period on full x-axis
ax.axvspan(COVID_START_WEEK, COVID_END_WEEK,
           alpha=0.15, color="red", label="COVID period", zorder=0)

# Plot full actual series (2015–2024)
ax.plot(
    wk.values, y.values,
    color="black", linewidth=1.3, marker="o", markersize=3,
    label="Actual (2015–2024)", zorder=2
)

# Overlay last LAST_N weeks: actual vs predicted
ax.plot(
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    wk_test.values, y_test.values,
    color="tab:blue", linewidth=1.8, marker="o", markersize=5,
    label=f"Actual (last {LAST_N} weeks)", zorder=3
)

ax.plot(
    wk_test.values, y_pred,
    color="tab:orange", linewidth=1.8, marker="s", markersize=5,
    label=f"Predicted (last {LAST_N} weeks)", zorder=4
)

ax.set_title(
    "CatBoost Baseline — Full Series (2015–2024)\n"
    "No AR terms, no MA target, no tuning"
)
ax.set_xlabel("Week Number")
ax.set_ylabel("Weekly ER Visits")
ax.legend()
plt.tight_layout()
plt.show()

# ---------- save ----------
out = pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_CatBoost": y_pred
})
out.to_csv(f"CatBoost_baseline_last{LAST_N}.csv", index=False)
print(f"\n✅ Saved: CatBoost_baseline_last{LAST_N}.csv")
```

In [ ]:

```
# ================================================================
# CatBoost — Best Lags + AR(1,2) + deeper trees + scaled X
# ================================================================
import pandas as pd, numpy as np, re
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

LAST_N = 40
TARGET = "Respiratory_Weekly_ER_Admissions"
RANDOM_STATE = 42

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

# ---------- normalize tricky names ----------
def normalize_cols(df):
    ren = {}
    for c in df.columns:
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
            ren[c] = re.sub(r"pm\s*2\.?5|pm25","PM2.5",c,flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I); suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
df = normalize_cols(df)

# ---------- add short-term autoregressive features ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- clean and order ----------
df = df.replace([np.inf, -np.inf], np.nan).dropna().reset_index(drop=True)
df = df.sort_values("Week Number").reset_index(drop=True)

# ---------- best-lag features ----------
best_lag_feats = [
    "8h avg CO (ppm)_lag2",
    "Avg 8h O3 (ppm)_lag3",
    "Avg 1h NO2 (ppb)_lag2",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h SO2 (ppb)_lag0",
    "TAVG (°F, weekly mean)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "Week Number",
    "Resp_lag1",
    "Resp_lag2"
]

missing = [c for c in best_lag_feats if c not in df.columns]
if missing:
    raise KeyError(f"Missing expected columns: {missing}")

# ---------- temporal split ----------
N = len(df)
train_end = N - LAST_N
X = df[best_lag_feats].copy()
y = df[TARGET].copy()
wk = df["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")

# ---------- scale predictors ----------
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s  = scaler.transform(X_test)

# ---------- CatBoost (slightly deeper, slower learning) ----------
cb = CatBoostRegressor(
    loss_function="RMSE",
    iterations=1500,
    learning_rate=0.03,
    depth=8,
    l2_leaf_reg=3.0,
    subsample=0.9,
    random_seed=RANDOM_STATE,
    early_stopping_rounds=100,
    verbose=False
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
)
cb.fit(X_train_s, y_train, eval_set=(X_test_s, y_test), use_best_model=True)

# ---------- evaluate ----------
y_pred = cb.predict(X_test_s)
r2   = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)

print("\n📊 CatBoost — Best Lags + AR(1,2) + tuned structure (no MA)")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- plot ----------
plt.figure(figsize=(10,5))
plt.plot(wk_test.values, y_test.values, label=f"Actual (last {LAST_N} weeks)", marker="o
plt.plot(wk_test.values, y_pred, label="Predicted (CatBoost tuned)", marker="s", color="
plt.title(f"CatBoost — Best Lags + AR(1,2), Scaled X, deeper trees\nHoldout: last {LAST_
plt.xlabel("Week Number"); plt.ylabel("Weekly ER Visits")
plt.legend(); plt.tight_layout(); plt.show()

# ---------- save ----------
out = pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_CatBoost_tuned": y_pred
})
out.to_csv(f"CatBoost_bestlags_AR12_scaled_last{LAST_N}.csv", index=False)
print(f"\n✅ Saved: CatBoost_bestlags_AR12_scaled_last{LAST_N}.csv")
```

In [ ]:

```
# ============================================================
# CatBoost — Best Lags + AR(1,2) + deeper trees + scaled X
# Full-series plot with COVID highlight
# ============================================================
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# ---------- config ----------
LAST_N       = 40
TARGET       = "Respiratory_Weekly_ER_Admissions"
RANDOM_STATE = 42

# COVID period in "Week Number" units (1 = first week of 2015)
# 2015: 1–52, 2016: 53–104, 2017: 105–156, 2018: 157–208, 2019: 209–260
# 2020: 261–312, 2021: 313–364, 2022: 365–416, 2023: 417–468, 2024: 469–520
COVID_START_WEEK = 271   # ~ mid-March 2020
COVID_END_WEEK   = 364   # end of 2021

# ---------- normalize tricky names ----------
def normalize_cols(df: pd.DataFrame) -> pd.DataFrame:
    rename_dict = {}
    for c in df.columns:
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
            rename_dict[c] = re.sub(r"pm\s*2\.?5|pm25", "PM2.5", c, flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            rename_dict[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=rename_dict) if rename_dict else df

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()
df = normalize_cols(df)

# ---------- add short-term autoregressive features ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- clean and order ----------
df = df.replace([np.inf, -np.inf], np.nan).dropna().reset_index(drop=True)

if "Week Number" not in df.columns:
    raise KeyError("Expected 'Week Number' in dataset.")

df = df.sort_values("Week Number").reset_index(drop=True)

# ---------- best-lag features ----------
best_lag_feats = [
    "8h avg CO (ppm)_lag2",
    "Avg 8h O3 (ppm)_lag3",
    "Avg 1h NO2 (ppb)_lag2",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h SO2 (ppb)_lag0",
    "TAVG (°F, weekly mean)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "Week Number",
    "Resp_lag1",
    "Resp_lag2",
]

missing = [c for c in best_lag_feats if c not in df.columns]
if missing:
    raise KeyError(f"Missing expected columns: {missing}")

# ---------- temporal split ----------
N = len(df)
train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows for temporal split. N={N}, LAST_N={LAST_N}")

X  = df[best_lag_feats].copy()
y  = df[TARGET].copy()
wk = df["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test         = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")

# ---------- scale predictors ----------
scaler   = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s  = scaler.transform(X_test)

# ---------- CatBoost (slightly deeper, slower learning) ----------
cb = CatBoostRegressor(
    loss_function="RMSE",
    iterations=1500,
    learning_rate=0.03,
    depth=8,
    l2_leaf_reg=3.0,
    subsample=0.9,
    random_seed=RANDOM_STATE,
    early_stopping_rounds=100,
    verbose=False
)

cb.fit(X_train_s, y_train, eval_set=(X_test_s, y_test), use_best_model=True)

# ---------- evaluate ----------
y_pred = cb.predict(X_test_s)
r2   = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)

print("\n📊 CatBoost — Best Lags + AR(1,2) + tuned structure (no MA)")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- FULL-SERIES PLOT with COVID highlight ----------
plt.figure(figsize=(11, 6))
ax = plt.gca()

# Shade COVID period on full x-axis
ax.axvspan(COVID_START_WEEK, COVID_END_WEEK,
           alpha=0.15, color="red", label="COVID period", zorder=0)

# Plot full actual series (2015–2024)
ax.plot(
    wk.values, y.values,
    color="black", linewidth=1.3, marker="o", markersize=3,
    label="Actual (2015–2024)", zorder=2
)

# Overlay last LAST_N weeks: actual vs predicted
ax.plot(
    wk_test.values, y_test.values,
    color="tab:blue", linewidth=1.8, marker="o", markersize=5,
    label=f"Actual (last {LAST_N} weeks)", zorder=3
)

ax.plot(
    wk_test.values, y_pred,
    color="tab:orange", linewidth=1.8, marker="s", markersize=5,
    label=f"Predicted (last {LAST_N} weeks)", zorder=4
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
)

ax.set_title(
    "CatBoost — Best Lags + AR(1,2), Scaled X, Deeper Trees\n"
    "Full Series (2015–2024) with COVID Highlight"
)
ax.set_xlabel("Week Number")
ax.set_ylabel("Weekly ER Visits")
ax.legend()
plt.tight_layout()
plt.show()

# ---------- save ----------
out = pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_CatBoost_tuned": y_pred
})
out.to_csv(f"CatBoost_bestlags_AR12_scaled_last{LAST_N}.csv", index=False)
print(f"\n✅ Saved: CatBoost_bestlags_AR12_scaled_last{LAST_N}.csv")
```

In [ ]:

```
# ============================================================
# CatBoost — GAM-like setup with optional MA(y), AR(1,2), log1p(y)
# Temporal holdout: last LAST_N weeks
# ============================================================
import pandas as pd, numpy as np, re
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# ---------- knobs ----------
LAST_N     = 40        # size of test window (weeks)
ROLL_WIN   = 2         # 1 = no MA; set 2 for light target smoothing (parity with GAM)
SCALE_X    = False     # trees don't need scaling; set True if you want it
RANDOM_SEED = 42

# CatBoost structure (mirrors "smoothness" control)
CB_KW = dict(
    loss_function="RMSE",
    iterations=1500,
    learning_rate=0.03,
    depth=8,
    l2_leaf_reg=3.0,
    subsample=0.9,
    random_seed=RANDOM_SEED,
    early_stopping_rounds=100,
    verbose=False,
)

TARGET = "Respiratory_Weekly_ER_Admissions"

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

# ---------- normalize tricky names ----------
def normalize_cols(df):
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    ren = {}
    for c in df.columns:
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
            ren[c] = re.sub(r"pm\s*2\.?5|pm25","PM2.5", c, flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df

df = normalize_cols(df)

# ---------- AR(1), AR(2) from ORIGINAL target (no smoothing in features) ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- OPTIONAL: light MA on TARGET only (after AR lags, to avoid leakage into fea
if ROLL_WIN > 1:
    before = len(df)
    df[TARGET] = df[TARGET].rolling(ROLL_WIN, center=True, min_periods=ROLL_WIN).mean()
    # We'll drop NaNs later in a single pass
    print(f"[Info] Applied centered MA to target with window={ROLL_WIN}.")

# ---------- choose best-lag feature set (plus AR terms) ----------
feat_cols = [
    "8h avg CO (ppm)_lag2",
    "Avg 8h O3 (ppm)_lag3",
    "Avg 1h NO2 (ppb)_lag2",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h SO2 (ppb)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "TAVG (°F, weekly mean)_lag0",
    "Week Number",
    "Resp_lag1",
    "Resp_lag2",
]

missing = [c for c in feat_cols if c not in df.columns]
if missing:
    raise KeyError(f"Missing expected columns: {missing}")

# ---------- clean: drop NaN/Inf; sort by time ----------
df = (df[feat_cols + [TARGET]]
      .replace([np.inf, -np.inf], np.nan)
      .dropna()
      .sort_values("Week Number")
      .reset_index(drop=True))

# ---------- temporal split ----------
N = len(df)
train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows after cleaning. N={N}, LAST_N={LAST_N}")

X = df[feat_cols].copy()
y = df[TARGET].copy()
wk = df["Week Number"].copy()
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")
print(f"Using {len(feat_cols)} features.")
print(f"ROLL_WIN={ROLL_WIN} (target MA), SCALE_X={SCALE_X}")

# ---------- optional scaling ----------
if SCALE_X:
    scaler = StandardScaler()
    X_train_use = scaler.fit_transform(X_train)
    X_test_use  = scaler.transform(X_test)
else:
    X_train_use, X_test_use = X_train, X_test

# ---------- train on log1p(y), predict expm1 back ----------
y_train_log = np.log1p(y_train)

cb = CatBoostRegressor(**CB_KW)
cb.fit(X_train_use, y_train_log, eval_set=(X_test_use, np.log1p(y_test)), use_best_model

y_pred = np.expm1(cb.predict(X_test_use))

# ---------- evaluate ----------
r2   = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)

print("\n📊 CatBoost — GAM-like: log1p(y), AR(1,2), optional MA(y)")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- plot ----------
plt.figure(figsize=(10,5))
plt.plot(wk_test.values, y_test.values, label=f"Actual (last {LAST_N} weeks)", marker="o
plt.plot(wk_test.values, y_pred,       label="Predicted (CatBoost)", marker="s", color="
ma_note = " (MA on y)" if ROLL_WIN > 1 else ""
plt.title(f"CatBoost — Last {LAST_N} Weeks\n(AR(1,2), log1p(y){ma_note})")
plt.xlabel("Week Number"); plt.ylabel("Weekly ER Visits")
plt.legend(); plt.tight_layout(); plt.show()

# ---------- save ----------
pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_CatBoost": y_pred
}).to_csv(f"CatBoost_holdout_last{LAST_N}_AR12_logy_MA{ROLL_WIN}.csv", index=False)
print(f"\n✅ Saved: CatBoost_holdout_last{LAST_N}_AR12_logy_MA{ROLL_WIN}.csv")
```

In [ ]:

```
# ============================================================
# CatBoost — GAM-like setup with optional MA(y), AR(1,2), log1p(y)
# Temporal holdout: last LAST_N weeks
# Full-series plot with COVID highlight
# ============================================================
import pandas as pd, numpy as np, re
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# ---------- knobs ----------
LAST_N      = 40        # size of test window (weeks)
ROLL_WIN    = 2         # 1 = no MA; set 2 for light target smoothing (parity with GAM)
SCALE_X     = False     # trees don't need scaling; set True if you want it
RANDOM_SEED = 42

# CatBoost structure (mirrors "smoothness" control)
CB_KW = dict(
    loss_function="RMSE",
    iterations=1500,
    learning_rate=0.03,
    depth=8,
    l2_leaf_reg=3.0,
    subsample=0.9,
    random_seed=RANDOM_SEED,
    early_stopping_rounds=100,
    verbose=False,
)

TARGET = "Respiratory_Weekly_ER_Admissions"

# COVID period in "Week Number" units (1 = first week of 2015)
# 2015: 1–52, 2016: 53–104, 2017: 105–156, 2018: 157–208, 2019: 209–260
# 2020: 261–312, 2021: 313–364, 2022: 365–416, 2023: 417–468, 2024: 469–520
COVID_START_WEEK = 271   # ~ mid-March 2020
COVID_END_WEEK   = 364   # end of 2021

# ---------- load ----------
df = pd.read_csv("Dataset_with_lags_v2.csv")
df.columns = df.columns.str.strip()

# ---------- normalize tricky names ----------
def normalize_cols(df):
    ren = {}
    for c in df.columns:
        cl = c.lower()
        if "pm2.5" in cl or "pm25" in cl:
            ren[c] = re.sub(r"pm\s*2\.?5|pm25","PM2.5", c, flags=re.I)
        if "avg_wnd" in cl or "avg wind" in cl:
            m = re.search(r"(_lag\d+)$", c, flags=re.I)
            suf = m.group(1) if m else ""
            ren[c] = f"Avg_WND (mph, weekly mean){suf}"
    return df.rename(columns=ren) if ren else df

df = normalize_cols(df)

# ---------- AR(1), AR(2) from ORIGINAL target (no smoothing in features) ----------
df["Resp_lag1"] = df[TARGET].shift(1)
df["Resp_lag2"] = df[TARGET].shift(2)

# ---------- OPTIONAL: light MA on TARGET only (after AR lags, to avoid leakage into fea
if ROLL_WIN > 1:
    df[TARGET] = df[TARGET].rolling(ROLL_WIN, center=True, min_periods=ROLL_WIN).mean()
    print(f"[Info] Applied centered MA to target with window={ROLL_WIN}.")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
# ---------- choose best-lag feature set (plus AR terms) ----------
feat_cols = [
    "8h avg CO (ppm)_lag2",
    "Avg 8h O3 (ppm)_lag3",
    "Avg 1h NO2 (ppb)_lag2",
    "Avg PM2.5 (ug/m3 LC)_lag1",
    "Avg 1h SO2 (ppb)_lag0",
    "Avg_WND (mph, weekly mean)_lag1",
    "TAVG (°F, weekly mean)_lag0",
    "Week Number",
    "Resp_lag1",
    "Resp_lag2",
]

missing = [c for c in feat_cols if c not in df.columns]
if missing:
    raise KeyError(f"Missing expected columns: {missing}")

# ---------- clean: drop NaN/Inf; sort by time ----------
df = (
    df[feat_cols + [TARGET]]
    .replace([np.inf, -np.inf], np.nan)
    .dropna()
    .sort_values("Week Number")
    .reset_index(drop=True)
)

# ---------- temporal split ----------
N = len(df)
train_end = N - LAST_N
if train_end <= 0:
    raise ValueError(f"Not enough rows after cleaning. N={N}, LAST_N={LAST_N}")

X  = df[feat_cols].copy()
y  = df[TARGET].copy()
wk = df["Week Number"].copy()

X_train, X_test = X.iloc[:train_end], X.iloc[train_end:]
y_train, y_test = y.iloc[:train_end], y.iloc[train_end:]
wk_test         = wk.iloc[train_end:]

print(f"Train size: {len(X_train)} | Test size: {len(X_test)} (last {LAST_N} weeks)")
print(f"Test weeks: {wk_test.iloc[0]} .. {wk_test.iloc[-1]}")
print(f"Using {len(feat_cols)} features.")
print(f"ROLL_WIN={ROLL_WIN} (target MA), SCALE_X={SCALE_X}")

# ---------- optional scaling ----------
if SCALE_X:
    scaler      = StandardScaler()
    X_train_use = scaler.fit_transform(X_train)
    X_test_use  = scaler.transform(X_test)
else:
    X_train_use, X_test_use = X_train, X_test

# ---------- train on log1p(y), predict expm1 back ----------
y_train_log = np.log1p(y_train)
y_test_log  = np.log1p(y_test)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
cb = CatBoostRegressor(**CB_KW)
cb.fit(X_train_use, y_train_log, eval_set=(X_test_use, y_test_log), use_best_model=True)

y_pred = np.expm1(cb.predict(X_test_use))

# ---------- evaluate ----------
r2   = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)

print("\n📊 CatBoost — GAM-like: log1p(y), AR(1,2), optional MA(y)")
print(f"R²:   {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE:  {mae:.3f}")

# ---------- FULL-SERIES PLOT with COVID highlight ----------
plt.figure(figsize=(11, 6))
ax = plt.gca()

# Shade COVID period across the full x-axis
ax.axvspan(COVID_START_WEEK, COVID_END_WEEK,
           alpha=0.15, color="red", label="COVID period", zorder=0)

# Full actual series (2015–2024 after cleaning)
ax.plot(
    wk.values, y.values,
    color="black", linewidth=1.3, marker="o", markersize=3,
    label="Actual (2015–2024)", zorder=2
)

# Overlay last LAST_N weeks actual vs predicted
ax.plot(
    wk_test.values, y_test.values,
    color="tab:blue", linewidth=1.8, marker="o", markersize=5,
    label=f"Actual (last {LAST_N} weeks)", zorder=3
)

ax.plot(
    wk_test.values, y_pred,
    color="tab:orange", linewidth=1.8, marker="s", markersize=5,
    label=f"Predicted (last {LAST_N} weeks)", zorder=4
)

ma_note = " (MA on y)" if ROLL_WIN > 1 else ""
ax.set_title(
    "CatBoost — Full Series (2015–2024)\n"
    f"GAM-like (AR(1,2), log1p(y){ma_note}) with COVID Highlight"
)
ax.set_xlabel("Week Number")
ax.set_ylabel("Weekly ER Visits")
ax.legend()
plt.tight_layout()
plt.show()

# ---------- save ----------
pd.DataFrame({
    "Week_Number": wk_test.values,
    "Actual_ER_Visits": y_test.values,
    "Predicted_CatBoost": y_pred
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
}).to_csv(f"CatBoost_holdout_last{LAST_N}_AR12_logy_MA{ROLL_WIN}.csv", index=False)
print(f"\n✅ Saved: CatBoost_holdout_last{LAST_N}_AR12_logy_MA{ROLL_WIN}.csv")
```

In [ ]:

```
# ----------------------------------------
# Residual ACF & PACF for TRAINING SET
# ----------------------------------------
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt
import numpy as np

# Compute training predictions (on log scale, then back-transform)
y_train_pred_log = cb.predict(X_train_use)          # log1p scale
y_train_pred = np.expm1(y_train_pred_log)           # original scale

# Compute residuals (original scale)
res_train = y_train.values - y_train_pred

# Number of lags & critical bounds for alpha = 0.05
nlags = 30
N = len(res_train)
crit = 1.96 / np.sqrt(N)   # ~95% CI bounds (two-sided)

# ------------------- ACF -------------------
plt.figure(figsize=(8,4))
plot_acf(res_train, lags=nlags, alpha=None)
plt.hlines([crit, -crit], xmin=-0.5, xmax=nlags+0.5, colors="red", linestyles="dashed")
plt.hlines(0, xmin=-0.5, xmax=nlags+0.5, colors="black")
plt.title("ACF – Training Residuals (Original Scale)")
plt.xlabel("Lag"); plt.ylabel("Autocorrelation")
plt.tight_layout()
plt.show()

# ------------------- PACF -------------------
plt.figure(figsize=(8,4))
plot_pacf(res_train, lags=nlags, method="ywm", alpha=None)
plt.hlines([crit, -crit], xmin=-0.5, xmax=nlags+0.5, colors="red", linestyles="dashed")
plt.hlines(0, xmin=-0.5, xmax=nlags+0.5, colors="black")
plt.title("PACF – Training Residuals (Original Scale)")
plt.xlabel("Lag"); plt.ylabel("Partial Autocorrelation")
plt.tight_layout()
plt.show()
```

Analysis 2

In [ ]:

```
# ============================
# Asthma MLR: Pre/Post Wildfire Slope & Structural-Break Analysis
# ============================

# --- Imports ---
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
from scipy.stats import f as f_dist

# --- CONFIG ---
DATA_PATH = Path("Dataset.csv")  # <-- change to your CSV
# Wildfire window (weeks inclusive) to EXCLUDE from pre/post fits but still plot
WILDFIRE_START, WILDFIRE_END = 470, 475

# Columns expected in the file
COL_WEEK = "Week Number"
COL_Y = "Asthma_weekly_ER_Admissions"
FEATURES = [" Avg PM2.5 (ug/m3 LC)", "Avg 8h O3 (ppm)", "Avg 1h NO2 (ppb)",
"TAVG (°F, weekly mean)", "Avg_WND (mph, weekly mean)", "PRCP (inches, weekly sum)"]  #

# --- 1) Load & basic prep ---
df = pd.read_csv(DATA_PATH)

# Ensure numeric + drop rows with missing essentials
cols_needed = [COL_WEEK, COL_Y] + FEATURES
for c in cols_needed:
    df[c] = pd.to_numeric(df[c], errors="coerce")
df = df.dropna(subset=cols_needed).copy()

# Keep only valid week range if present (e.g., 1..521)
if df[COL_WEEK].min() < 1 or df[COL_WEEK].max() > 10000:
    print("Warning: Week looks unusual; check 'Week' column.")
df = df.sort_values(COL_WEEK).reset_index(drop=True)

# Create masks for pre/post (omit wildfire weeks for estimation)
pre_mask  = df[COL_WEEK] < WILDFIRE_START
post_mask = df[COL_WEEK] > WILDFIRE_END
wild_mask = df[COL_WEEK].between(WILDFIRE_START, WILDFIRE_END)

# --- Helper: build X with intercept ---
def add_const_X(frame, feature_names):
    X = frame[feature_names].copy()
    X = sm.add_constant(X)  # adds intercept column named 'const'
    return X

# --- 2) VIF check (pooled sample excluding wildfire weeks) ---
X_vif = add_const_X(df.loc[pre_mask | post_mask, FEATURES], FEATURES)
# Compute VIF only for predictors (exclude the intercept at index 0)
vif_table = pd.DataFrame({
    "feature": ["const"] + FEATURES,
    "VIF": [np.nan] + [
        variance_inflation_factor(X_vif.values, i)
        for i in range(1, X_vif.shape[1])
    ]
})
print("\n=== Variance Inflation Factors (pooled, excl. wildfire) ===")
print(vif_table.to_string(index=False))

# --- 3) Fit OLS models: pooled (no wildfire weeks), pre-only, post-only ---
y_pooled = df.loc[pre_mask | post_mask, COL_Y]
X_pooled = add_const_X(df.loc[pre_mask | post_mask, FEATURES], FEATURES)
model_pooled = sm.OLS(y_pooled, X_pooled).fit()

y_pre = df.loc[pre_mask, COL_Y]
X_pre = add_const_X(df.loc[pre_mask, FEATURES], FEATURES)
model_pre = sm.OLS(y_pre, X_pre).fit()
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
y_post = df.loc[post_mask, COL_Y]
X_post = add_const_X(df.loc[post_mask, FEATURES], FEATURES)
model_post = sm.OLS(y_post, X_post).fit()

print("\n=== Pooled model (excl. wildfire weeks) summary ===")
print(model_pooled.summary())

# --- 4) Compare slopes (coefficients) pre vs post ---
coef_pre = model_pre.params.reindex(["const"] + FEATURES)
coef_post = model_post.params.reindex(["const"] + FEATURES)
coef_compare = pd.DataFrame({"Pre": coef_pre, "Post": coef_post})
coef_compare["Post-Pre"] = coef_compare["Post"] - coef_compare["Pre"]

print("\n=== Coefficient comparison (Pre vs Post) ===")
print(coef_compare.to_string())

# --- 5) Manual Chow Test at the wildfire break (using pooled vs separate fits) ---
# Formula:
# F = ((SSR_pooled - (SSR_pre + SSR_post)) / k) / ((SSR_pre + SSR_post) / (n1 + n2 - 2k)
# where k = number of parameters (incl. intercept), n1=pre obs, n2=post obs
def chow_test(model_pooled, model_pre, model_post, n1, n2, k):
    SSR_pooled = np.sum(model_pooled.resid**2)
    SSR_pre = np.sum(model_pre.resid**2)
    SSR_post = np.sum(model_post.resid**2)
    num = (SSR_pooled - (SSR_pre + SSR_post)) / k
    den = (SSR_pre + SSR_post) / (n1 + n2 - 2*k)
    F_stat = num / den
    p_value = f_dist.sf(F_stat, k, (n1 + n2 - 2*k))
    return F_stat, p_value, SSR_pooled, SSR_pre, SSR_post

n1 = X_pre.shape[0]
n2 = X_post.shape[0]
k = X_pre.shape[1]  # includes intercept

F_stat, p_val, SSR_pooled, SSR_pre, SSR_post = chow_test(model_pooled, model_pre, model_
print("\n=== Chow Test (break between weeks {} and {}) ===".format(WILDFIRE_START-1, WIL
print(f"F({k}, {n1 + n2 - 2*k}) = {F_stat:0.3f},  p = {p_val:0.4f}")
print(f"SSR pooled: {SSR_pooled:0.1f} | SSR pre: {SSR_pre:0.1f} | SSR post: {SSR_post:0.

# --- 6) Visualizations ---
plt.figure(figsize=(10, 5))
# Preds using pooled model for ALL weeks (including wildfire) to show deviation
X_all_for_pred = add_const_X(df[FEATURES], FEATURES)
df["Pred_pooled"] = model_pooled.predict(X_all_for_pred)

plt.plot(df[COL_WEEK], df[COL_Y], label="Actual Asthma Visits")
plt.plot(df[COL_WEEK], df["Pred_pooled"], label="Predicted (Pooled, trained excl. wildfi

# Shade wildfire weeks
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, label="Wildfire window")

plt.title("Asthma ED Visits: Actual vs. Pooled OLS Predictions")
plt.xlabel("Week")
plt.ylabel("Visits")
plt.legend()
plt.tight_layout()
plt.show()
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
# Residuals (pooled fit predictions applied to all weeks)
df["Resid_all"] = df[COL_Y] - df["Pred_pooled"]
plt.figure(figsize=(10, 4))
plt.plot(df[COL_WEEK], df["Resid_all"])
plt.axhline(0, linestyle="--", linewidth=1)
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2)
plt.title("Residuals over Time (Pooled model applied to all weeks)")
plt.xlabel("Week")
plt.ylabel("Residual (Actual - Predicted)")
plt.tight_layout()
plt.show()

# Coefficient comparison bar chart (Pre vs Post) for all predictors
coef_plot = coef_compare.loc[FEATURES].copy()  # exclude intercept from bars for clarity
ax = coef_plot[["Pre", "Post"]].plot(kind="bar", figsize=(10, 5))
plt.title("Pre vs Post Coefficients (Asthma MLR)")
plt.ylabel("Slope (Δ visits per unit)")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()

# Optional: print simple diagnostics
print("\n=== Quick diagnostics ===")
print(f"Pre sample size (n1): {n1} | Post sample size (n2): {n2}")
print(f"Adj. R^2 Pre:  {model_pre.rsquared_adj:0.3f}")
print(f"Adj. R^2 Post: {model_post.rsquared_adj:0.3f}")
print(f"Adj. R^2 Pooled (excl. wildfire weeks): {model_pooled.rsquared_adj:0.3f}")

# Notes for interpretation:
# - If the Chow p-value < 0.05, there is evidence the regression relationship changed af
# - Inspect which coefficients changed the most in 'coef_compare' (e.g., PM25).
# - Check VIFs: if > ~5-10, consider collinearity; for this hypothesis test, keep the se
```

In [ ]:

```
# ============================
# ILI MLR: Pre/Post Wildfire Slope & Structural-Break Analysis
# ============================
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

# --- CONFIG ---
DATA_PATH = Path("Dataset.csv")                      # <-- your CSV
WILDFIRE_START, WILDFIRE_END = 470, 475              # excluded from estimation
COL_WEEK = "Week Number"
COL_Y = "ILI_Weekly_ER_Admissions"                   # <-- change if your ILI name diffe
FEATURES = [
    " Avg PM2.5 (ug/m3 LC)", "Avg 8h O3 (ppm)", "Avg 1h NO2 (ppb)",
    "TAVG (°F, weekly mean)", "Avg_WND (mph, weekly mean)", "PRCP (inches, weekly sum)"
]

# --- Load & prep ---
df = pd.read_csv(DATA_PATH)
for c in [COL_WEEK, COL_Y] + FEATURES:
    df[c] = pd.to_numeric(df[c], errors="coerce")
df = df.dropna(subset=[COL_WEEK, COL_Y] + FEATURES).sort_values(COL_WEEK).reset_index(dr
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
pre_mask  = df[COL_WEEK] <  WILDFIRE_START
post_mask = df[COL_WEEK] >  WILDFIRE_END

def add_const_X(frame, feature_names):
    X = sm.add_constant(frame[feature_names].copy())
    return X

# --- VIF (pooled excl. wildfire) ---
X_vif = add_const_X(df.loc[pre_mask | post_mask, FEATURES], FEATURES)
vif_table = pd.DataFrame({
    "feature": ["const"] + FEATURES,
    "VIF": [np.nan] + [variance_inflation_factor(X_vif.values, i) for i in range(1, X_vi
})
print("\n=== VIF (ILI, pooled excl. wildfire) ===")
print(vif_table.to_string(index=False))

# --- Fit pooled / pre / post ---
y_pooled = df.loc[pre_mask | post_mask, COL_Y]; X_pooled = add_const_X(df.loc[pre_mask |
y_pre    = df.loc[pre_mask, COL_Y];             X_pre    = add_const_X(df.loc[pre_mask,
y_post   = df.loc[post_mask, COL_Y];            X_post   = add_const_X(df.loc[post_mask,

model_pooled = sm.OLS(y_pooled, X_pooled).fit()
model_pre    = sm.OLS(y_pre,    X_pre).fit()
model_post   = sm.OLS(y_post,   X_post).fit()

print("\n=== Pooled model summary (ILI) ===")
print(model_pooled.summary())

# --- Coefficient comparison ---
coef_pre = model_pre.params.reindex(["const"] + FEATURES)
coef_post = model_post.params.reindex(["const"] + FEATURES)
coef_compare = pd.DataFrame({"Pre": coef_pre, "Post": coef_post})
coef_compare["Post-Pre"] = coef_compare["Post"] - coef_compare["Pre"]
print("\n=== Coefficient comparison (ILI) ===")
print(coef_compare.to_string())

# --- Chow test ---
def chow_test(model_pooled, model_pre, model_post, n1, n2, k):
    SSR_pooled = np.sum(model_pooled.resid**2)
    SSR_pre    = np.sum(model_pre.resid**2)
    SSR_post   = np.sum(model_post.resid**2)
    num = (SSR_pooled - (SSR_pre + SSR_post)) / k
    den = (SSR_pre + SSR_post) / (n1 + n2 - 2*k)
    F_stat = num / den
    p_val = f_dist.sf(F_stat, k, (n1 + n2 - 2*k))
    return F_stat, p_val, SSR_pooled, SSR_pre, SSR_post

n1, n2, k = X_pre.shape[0], X_post.shape[0], X_pre.shape[1]
F_stat, p_val, SSRp, SSR1, SSR2 = chow_test(model_pooled, model_pre, model_post, n1, n2,
print(f"\n=== Chow Test (ILI, break {WILDFIRE_START-1}/{WILDFIRE_END+1}) ===")
print(f"F({k}, {n1+n2-2*k}) = {F_stat:.3f},  p = {p_val:.4f}")
print(f"SSR pooled: {SSRp:.1f} | SSR pre: {SSR1:.1f} | SSR post: {SSR2:.1f}")

# --- Visuals ---
plt.figure(figsize=(10,5))
X_all = add_const_X(df[FEATURES], FEATURES)
df["Pred_pooled"] = model_pooled.predict(X_all)
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual ILI Visits")
plt.plot(df[COL_WEEK], df["Pred_pooled"], label="Predicted (Pooled excl. wildfire)", lin
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, label="Wildfire window")
plt.title("ILI: Actual vs. Pooled OLS Predictions")
plt.xlabel("Week"); plt.ylabel("Visits"); plt.legend(); plt.tight_layout(); plt.show()

df["Resid_all"] = df[COL_Y] - df["Pred_pooled"]
plt.figure(figsize=(10,4))
plt.plot(df[COL_WEEK], df["Resid_all"]); plt.axhline(0, ls="--", lw=1)
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2)
plt.title("ILI Residuals over Time (pooled model on all weeks)")
plt.xlabel("Week"); plt.ylabel("Residual"); plt.tight_layout(); plt.show()

coef_compare.loc[FEATURES][["Pre","Post"]].plot(kind="bar", figsize=(10,5))
plt.title("ILI: Pre vs Post Coefficients"); plt.ylabel("Slope (Δ visits/unit)")
plt.xticks(rotation=0); plt.tight_layout(); plt.show()

print("\n=== Quick diagnostics (ILI) ===")
print(f"n_pre={n1}, n_post={n2} | Adj.R2 pre={model_pre.rsquared_adj:.3f} | post={model_
```

In [ ]:

```
# ============================
# Respiratory (non-asthma) MLR: Pre/Post Wildfire Slope & Structural-Break
# ============================
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

# --- CONFIG ---
DATA_PATH = Path("Dataset.csv")                          # <-- your CSV
WILDFIRE_START, WILDFIRE_END = 470, 475
COL_WEEK = "Week Number"
COL_Y = "Respiratory_Weekly_ER_Admissions"              # <-- change if your column diffe
FEATURES = [
    " Avg PM2.5 (ug/m3 LC)", "Avg 8h O3 (ppm)", "Avg 1h NO2 (ppb)",
    "TAVG (°F, weekly mean)", "Avg_WND (mph, weekly mean)", "PRCP (inches, weekly sum)"
]

# --- Load & prep ---
df = pd.read_csv(DATA_PATH)
for c in [COL_WEEK, COL_Y] + FEATURES:
    df[c] = pd.to_numeric(df[c], errors="coerce")
df = df.dropna(subset=[COL_WEEK, COL_Y] + FEATURES).sort_values(COL_WEEK).reset_index(dr

pre_mask  = df[COL_WEEK] <  WILDFIRE_START
post_mask = df[COL_WEEK] >  WILDFIRE_END

def add_const_X(frame, feature_names):
    return sm.add_constant(frame[feature_names].copy())

# --- VIF (pooled excl. wildfire) ---
X_vif = add_const_X(df.loc[pre_mask | post_mask, FEATURES], FEATURES)
vif_table = pd.DataFrame({
    "feature": ["const"] + FEATURES,
    "VIF": [np.nan] + [variance_inflation_factor(X_vif.values, i) for i in range(1, X_vi
})
print("\n=== VIF (Respiratory, pooled excl. wildfire) ===")
print(vif_table.to_string(index=False))
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
# --- Fit pooled / pre / post ---
y_pooled = df.loc[pre_mask | post_mask, COL_Y]; X_pooled = add_const_X(df.loc[pre_mask |
y_pre    = df.loc[pre_mask, COL_Y];             X_pre    = add_const_X(df.loc[pre_mask,
y_post   = df.loc[post_mask, COL_Y];            X_post   = add_const_X(df.loc[post_mask,

model_pooled = sm.OLS(y_pooled, X_pooled).fit()
model_pre    = sm.OLS(y_pre,    X_pre).fit()
model_post   = sm.OLS(y_post,   X_post).fit()

print("\n=== Pooled model summary (Respiratory) ===")
print(model_pooled.summary())

# --- Coefficient comparison ---
coef_pre = model_pre.params.reindex(["const"] + FEATURES)
coef_post = model_post.params.reindex(["const"] + FEATURES)
coef_compare = pd.DataFrame({"Pre": coef_pre, "Post": coef_post})
coef_compare["Post-Pre"] = coef_compare["Post"] - coef_compare["Pre"]
print("\n=== Coefficient comparison (Respiratory) ===")
print(coef_compare.to_string())

# --- Chow test ---
def chow_test(model_pooled, model_pre, model_post, n1, n2, k):
    SSR_pooled = np.sum(model_pooled.resid**2)
    SSR_pre    = np.sum(model_pre.resid**2)
    SSR_post   = np.sum(model_post.resid**2)
    num = (SSR_pooled - (SSR_pre + SSR_post)) / k
    den = (SSR_pre + SSR_post) / (n1 + n2 - 2*k)
    F_stat = num / den
    p_val = f_dist.sf(F_stat, k, (n1 + n2 - 2*k))
    return F_stat, p_val, SSR_pooled, SSR_pre, SSR_post

n1, n2, k = X_pre.shape[0], X_post.shape[0], X_pre.shape[1]
F_stat, p_val, SSRp, SSR1, SSR2 = chow_test(model_pooled, model_pre, model_post, n1, n2,
print(f"\n=== Chow Test (Respiratory, break {WILDFIRE_START-1}/{WILDFIRE_END+1}) ===")
print(f"F({k}, {n1+n2-2*k}) = {F_stat:.3f},  p = {p_val:.4f}")
print(f"SSR pooled: {SSRp:.1f} | SSR pre: {SSR1:.1f} | SSR post: {SSR2:.1f}")

# --- Visuals ---
plt.figure(figsize=(10,5))
X_all = add_const_X(df[FEATURES], FEATURES)
df["Pred_pooled"] = model_pooled.predict(X_all)
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual Respiratory Visits")
plt.plot(df[COL_WEEK], df["Pred_pooled"], label="Predicted (Pooled excl. wildfire)", lin
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, label="Wildfire window")
plt.title("Respiratory: Actual vs. Pooled OLS Predictions")
plt.xlabel("Week"); plt.ylabel("Visits"); plt.legend(); plt.tight_layout(); plt.show()

df["Resid_all"] = df[COL_Y] - df["Pred_pooled"]
plt.figure(figsize=(10,4))
plt.plot(df[COL_WEEK], df["Resid_all"]); plt.axhline(0, ls="--", lw=1)
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2)
plt.title("Respiratory Residuals over Time (pooled model on all weeks)")
plt.xlabel("Week"); plt.ylabel("Residual"); plt.tight_layout(); plt.show()

coef_compare.loc[FEATURES][["Pre","Post"]].plot(kind="bar", figsize=(10,5))
plt.title("Respiratory: Pre vs Post Coefficients"); plt.ylabel("Slope (Δ visits/unit)")
plt.xticks(rotation=0); plt.tight_layout(); plt.show()
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
print("\n=== Quick diagnostics (Respiratory) ===")
print(f"n_pre={n1}, n_post={n2} | Adj.R2 pre={model_pre.rsquared_adj:.3f} | post={model_
```

In [ ]:

```
# ============================
# COPD MLR: Pre/Post Wildfire Slope & Structural-Break Analysis
# ============================
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

DATA_PATH = Path("Dataset.csv")
WILDFIRE_START, WILDFIRE_END = 470, 475
COL_WEEK = "Week Number"
COL_Y = "# COPD cases (weekly total)"  # <-- change if your column differs
FEATURES = [
    " Avg PM2.5 (ug/m3 LC)", "Avg 8h O3 (ppm)", "Avg 1h NO2 (ppb)",
    "TAVG (°F, weekly mean)", "Avg_WND (mph, weekly mean)", "PRCP (inches, weekly sum)"
]

df = pd.read_csv(DATA_PATH)
for c in [COL_WEEK, COL_Y] + FEATURES: df[c] = pd.to_numeric(df[c], errors="coerce")
df = df.dropna(subset=[COL_WEEK, COL_Y] + FEATURES).sort_values(COL_WEEK).reset_index(dr

pre_mask  = df[COL_WEEK] <  WILDFIRE_START
post_mask = df[COL_WEEK] >  WILDFIRE_END

addc = lambda frame: sm.add_constant(frame[FEATURES].copy())

# VIF (pooled, excl. wildfire)
X_vif = addc(df.loc[pre_mask | post_mask]);
vif = pd.DataFrame({"feature":["const"]+FEATURES,
                    "VIF":[np.nan]+[variance_inflation_factor(X_vif.values,i) for i in r
print("\n=== VIF (COPD) ==="); print(vif.to_string(index=False))

# Fit pooled / pre / post
y_pooled, X_pooled = df.loc[pre_mask|post_mask, COL_Y], addc(df.loc[pre_mask|post_mask])
y_pre,    X_pre    = df.loc[pre_mask, COL_Y],           addc(df.loc[pre_mask])
y_post,   X_post   = df.loc[post_mask, COL_Y],          addc(df.loc[post_mask])
model_pooled = sm.OLS(y_pooled, X_pooled).fit()
model_pre, model_post = sm.OLS(y_pre, X_pre).fit(), sm.OLS(y_post, X_post).fit()
print("\n=== Pooled model summary (COPD) ==="); print(model_pooled.summary())

# Coefficient comparison
coef_pre  = model_pre.params.reindex(["const"]+FEATURES)
coef_post = model_post.params.reindex(["const"]+FEATURES)
coef_cmp  = pd.DataFrame({"Pre":coef_pre,"Post":coef_post}); coef_cmp["Post-Pre"]=coef_c
print("\n=== Coefficient comparison (COPD) ==="); print(coef_cmp.to_string())

# Chow test
def chow(m_pool,m1,m2,n1,n2,k):
    SSRp, SSR1, SSR2 = np.sum(m_pool.resid**2), np.sum(m1.resid**2), np.sum(m2.resid**2)
    F = ((SSRp-(SSR1+SSR2))/k)/((SSR1+SSR2)/(n1+n2-2*k))
    p = f_dist.sf(F, k, (n1+n2-2*k)); return F,p,SSRp,SSR1,SSR2
n1,n2,k = X_pre.shape[0], X_post.shape[0], X_pre.shape[1]
F,p,SSRp,SSR1,SSR2 = chow(model_pooled, model_pre, model_post, n1,n2,k)
print(f"\n=== Chow Test (COPD, break {WILDFIRE_START-1}/{WILDFIRE_END+1}) ===")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
print(f"F({k}, {n1+n2-2*k}) = {F:.3f}, p = {p:.4f}")
print(f"SSR pooled: {SSRp:.1f} | SSR pre: {SSR1:.1f} | SSR post: {SSR2:.1f}")

# Visuals
plt.figure(figsize=(10,5))
X_all = addc(df); df["Pred_pooled"] = model_pooled.predict(X_all)
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual COPD Visits")
plt.plot(df[COL_WEEK], df["Pred_pooled"], label="Predicted (pooled excl. wildfire)", lw=
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, label="Wildfire window")
plt.title("COPD: Actual vs. Pooled OLS Predictions"); plt.xlabel("Week"); plt.ylabel("Vi
plt.legend(); plt.tight_layout(); plt.show()

df["Resid_all"] = df[COL_Y] - df["Pred_pooled"]
plt.figure(figsize=(10,4)); plt.plot(df[COL_WEEK], df["Resid_all"]); plt.axhline(0, ls="
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2)
plt.title("COPD Residuals over Time (pooled model on all weeks)"); plt.xlabel("Week"); p
plt.tight_layout(); plt.show()

coef_cmp.loc[FEATURES][["Pre","Post"]].plot(kind="bar", figsize=(10,5))
plt.title("COPD: Pre vs Post Coefficients"); plt.ylabel("Slope (Δ visits/unit)")
plt.xticks(rotation=0); plt.tight_layout(); plt.show()

print("\n=== Quick diagnostics (COPD) ===")
print(f"n_pre={n1}, n_post={n2} | Adj.R2 pre={model_pre.rsquared_adj:.3f} | post={model_
```

In [ ]:

```
# ============================
# Heart Disease MLR: Pre/Post Wildfire Slope & Structural-Break Analysis
# ============================
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

DATA_PATH = Path("Dataset.csv")
WILDFIRE_START, WILDFIRE_END = 470, 475
COL_WEEK = "Week Number"
COL_Y = "# Heart disease ER visits (weekly total)"  # <-- change if needed
FEATURES = [
    " Avg PM2.5 (ug/m3 LC)", "Avg 8h O3 (ppm)", "Avg 1h NO2 (ppb)",
    "TAVG (°F, weekly mean)", "Avg_WND (mph, weekly mean)", "PRCP (inches, weekly sum)"
]

df = pd.read_csv(DATA_PATH)
for c in [COL_WEEK, COL_Y] + FEATURES: df[c] = pd.to_numeric(df[c], errors="coerce")
df = df.dropna(subset=[COL_WEEK, COL_Y] + FEATURES).sort_values(COL_WEEK).reset_index(dr

pre_mask  = df[COL_WEEK] <  WILDFIRE_START
post_mask = df[COL_WEEK] >  WILDFIRE_END

addc = lambda frame: sm.add_constant(frame[FEATURES].copy())

X_vif = addc(df.loc[pre_mask | post_mask]);
vif = pd.DataFrame({"feature":["const"]+FEATURES,
                    "VIF":[np.nan]+[variance_inflation_factor(X_vif.values,i) for i in r
print("\n=== VIF (Heart Disease) ==="); print(vif.to_string(index=False))

y_pooled, X_pooled = df.loc[pre_mask|post_mask, COL_Y], addc(df.loc[pre_mask|post_mask])
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
y_pre,    X_pre    = df.loc[pre_mask, COL_Y],           addc(df.loc[pre_mask])
y_post,   X_post   = df.loc[post_mask, COL_Y],          addc(df.loc[post_mask])

model_pooled = sm.OLS(y_pooled, X_pooled).fit()
model_pre, model_post = sm.OLS(y_pre, X_pre).fit(), sm.OLS(y_post, X_post).fit()
print("\n=== Pooled model summary (Heart Disease) ==="); print(model_pooled.summary())

coef_pre  = model_pre.params.reindex(["const"]+FEATURES)
coef_post = model_post.params.reindex(["const"]+FEATURES)
coef_cmp  = pd.DataFrame({"Pre":coef_pre,"Post":coef_post}); coef_cmp["Post-Pre"]=coef_c
print("\n=== Coefficient comparison (Heart Disease) ==="); print(coef_cmp.to_string())

def chow(m_pool,m1,m2,n1,n2,k):
    SSRp, SSR1, SSR2 = np.sum(m_pool.resid**2), np.sum(m1.resid**2), np.sum(m2.resid**2)
    F = ((SSRp-(SSR1+SSR2))/k)/((SSR1+SSR2)/(n1+n2-2*k))
    p = f_dist.sf(F, k, (n1+n2-2*k)); return F,p,SSRp,SSR1,SSR2
n1,n2,k = X_pre.shape[0], X_post.shape[0], X_pre.shape[1]
F,p,SSRp,SSR1,SSR2 = chow(model_pooled, model_pre, model_post, n1,n2,k)
print(f"\n=== Chow Test (Heart Disease, break {WILDFIRE_START-1}/{WILDFIRE_END+1}) ===")
print(f"F({k}, {n1+n2-2*k}) = {F:.3f}, p = {p:.4f}")
print(f"SSR pooled: {SSRp:.1f} | SSR pre: {SSR1:.1f} | SSR post: {SSR2:.1f}")

plt.figure(figsize=(10,5))
X_all = addc(df); df["Pred_pooled"] = model_pooled.predict(X_all)
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual Heart Disease Visits")
plt.plot(df[COL_WEEK], df["Pred_pooled"], label="Predicted (pooled excl. wildfire)", lw=
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, label="Wildfire window")
plt.title("Heart Disease: Actual vs. Pooled OLS Predictions")
plt.xlabel("Week"); plt.ylabel("Visits"); plt.legend(); plt.tight_layout(); plt.show()

df["Resid_all"] = df[COL_Y] - df["Pred_pooled"]
plt.figure(figsize=(10,4)); plt.plot(df[COL_WEEK], df["Resid_all"]); plt.axhline(0, ls="
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2)
plt.title("Heart Disease Residuals over Time (pooled model on all weeks)")
plt.xlabel("Week"); plt.ylabel("Residual"); plt.tight_layout(); plt.show()

coef_cmp.loc[FEATURES][["Pre","Post"]].plot(kind="bar", figsize=(10,5))
plt.title("Heart Disease: Pre vs Post Coefficients"); plt.ylabel("Slope (Δ visits/unit)"
plt.xticks(rotation=0); plt.tight_layout(); plt.show()

print("\n=== Quick diagnostics (Heart Disease) ===")
print(f"n_pre={n1}, n_post={n2} | Adj.R2 pre={model_pre.rsquared_adj:.3f} | post={model_
```

In [ ]:

```
# ============================
# Stroke MLR: Pre/Post Wildfire Slope & Structural-Break Analysis
# ============================
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

DATA_PATH = Path("Dataset.csv")
WILDFIRE_START, WILDFIRE_END = 470, 475
COL_WEEK = "Week Number"
COL_Y = "# stroke hospital discharges (weekly total)"  # <-- change if needed
FEATURES = [
    " Avg PM2.5 (ug/m3 LC)", "Avg 8h O3 (ppm)", "Avg 1h NO2 (ppb)",
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    "TAVG (°F, weekly mean)", "Avg_WND (mph, weekly mean)", "PRCP (inches, weekly sum)"
]

df = pd.read_csv(DATA_PATH)
for c in [COL_WEEK, COL_Y] + FEATURES: df[c] = pd.to_numeric(df[c], errors="coerce")
df = df.dropna(subset=[COL_WEEK, COL_Y] + FEATURES).sort_values(COL_WEEK).reset_index(dr

pre_mask  = df[COL_WEEK] <  WILDFIRE_START
post_mask = df[COL_WEEK] >  WILDFIRE_END

addc = lambda frame: sm.add_constant(frame[FEATURES].copy())

X_vif = addc(df.loc[pre_mask | post_mask]);
vif = pd.DataFrame({"feature":["const"]+FEATURES,
                    "VIF":[np.nan]+[variance_inflation_factor(X_vif.values,i) for i in r
print("\n=== VIF (Stroke) ==="); print(vif.to_string(index=False))

y_pooled, X_pooled = df.loc[pre_mask|post_mask, COL_Y], addc(df.loc[pre_mask|post_mask])
y_pre,    X_pre    = df.loc[pre_mask, COL_Y],           addc(df.loc[pre_mask])
y_post,   X_post   = df.loc[post_mask, COL_Y],          addc(df.loc[post_mask])

model_pooled = sm.OLS(y_pooled, X_pooled).fit()
model_pre, model_post = sm.OLS(y_pre, X_pre).fit(), sm.OLS(y_post, X_post).fit()
print("\n=== Pooled model summary (Stroke) ==="); print(model_pooled.summary())

coef_pre  = model_pre.params.reindex(["const"]+FEATURES)
coef_post = model_post.params.reindex(["const"]+FEATURES)
coef_cmp  = pd.DataFrame({"Pre":coef_pre,"Post":coef_post}); coef_cmp["Post-Pre"]=coef_c
print("\n=== Coefficient comparison (Stroke) ==="); print(coef_cmp.to_string())

def chow(m_pool,m1,m2,n1,n2,k):
    SSRp, SSR1, SSR2 = np.sum(m_pool.resid**2), np.sum(m1.resid**2), np.sum(m2.resid**2)
    F = ((SSRp-(SSR1+SSR2))/k)/((SSR1+SSR2)/(n1+n2-2*k))
    p = f_dist.sf(F, k, (n1+n2-2*k)); return F,p,SSRp,SSR1,SSR2
n1,n2,k = X_pre.shape[0], X_post.shape[0], X_pre.shape[1]
F,p,SSRp,SSR1,SSR2 = chow(model_pooled, model_pre, model_post, n1,n2,k)
print(f"\n=== Chow Test (Stroke, break {WILDFIRE_START-1}/{WILDFIRE_END+1}) ===")
print(f"F({k}, {n1+n2-2*k}) = {F:.3f}, p = {p:.4f}")
print(f"SSR pooled: {SSRp:.1f} | SSR pre: {SSR1:.1f} | SSR post: {SSR2:.1f}")

plt.figure(figsize=(10,5))
X_all = addc(df); df["Pred_pooled"] = model_pooled.predict(X_all)
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual Stroke Visits")
plt.plot(df[COL_WEEK], df["Pred_pooled"], label="Predicted (pooled excl. wildfire)", lw=
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, label="Wildfire window")
plt.title("Stroke: Actual vs. Pooled OLS Predictions")
plt.xlabel("Week"); plt.ylabel("Visits"); plt.legend(); plt.tight_layout(); plt.show()

df["Resid_all"] = df[COL_Y] - df["Pred_pooled"]
plt.figure(figsize=(10,4)); plt.plot(df[COL_WEEK], df["Resid_all"]); plt.axhline(0, ls="
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2)
plt.title("Stroke Residuals over Time (pooled model on all weeks)")
plt.xlabel("Week"); plt.ylabel("Residual"); plt.tight_layout(); plt.show()

coef_cmp.loc[FEATURES][["Pre","Post"]].plot(kind="bar", figsize=(10,5))
plt.title("Stroke: Pre vs Post Coefficients"); plt.ylabel("Slope (Δ visits/unit)")
plt.xticks(rotation=0); plt.tight_layout(); plt.show()
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
print("\n=== Quick diagnostics (Stroke) ===")
print(f"n_pre={n1}, n_post={n2} | Adj.R2 pre={model_pre.rsquared_adj:.3f} | post={model_
```

Analysis 3

In [ ]:

```
# ============================================================
# UNIVERSAL SCRIPT: Interaction MLR + Pre/Post Slopes + Chow Test
# Works for: ILI, Respiratory, Stroke (just change COL_Y)
# ============================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

# ============================================================
# CONFIG — CHANGE ONLY THE OUTCOME VARIABLE HERE
# ============================================================
DATA_PATH = Path("Dataset.csv")

COL_WEEK = "Week Number"

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# CHANGE THIS LINE FOR EACH ANALYSIS:
# ILI:        COL_Y = "ILI_weekly_ER_Admissions"
# Respiratory: COL_Y = "Respiratory_weekly_ER_Admissions"
# Stroke:      COL_Y = "Stroke_weekly_ER_Admissions"
COL_Y = "ILI_Weekly_ER_Admissions"
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

FEATURES = [
    " Avg PM2.5 (ug/m3 LC)",
    "Avg 8h O3 (ppm)",
    "Avg 1h NO2 (ppb)",
    "TAVG (°F, weekly mean)",
    "Avg_WND (mph, weekly mean)",
    "PRCP (inches, weekly sum)"
]

# Wildfire event window
WILDFIRE_START, WILDFIRE_END = 470, 475

# ============================================================
# LOAD & CLEAN
# ============================================================
df = pd.read_csv(DATA_PATH)

cols_needed = [COL_WEEK, COL_Y] + FEATURES
for c in cols_needed:
    df[c] = pd.to_numeric(df[c], errors="coerce")

df = df.dropna(subset=cols_needed).copy()
df = df.sort_values(COL_WEEK).reset_index(drop=True)

pre_mask = df[COL_WEEK] < WILDFIRE_START
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
post_mask = df[COL_WEEK] > WILDFIRE_END
wild_mask = df[COL_WEEK].between(WILDFIRE_START, WILDFIRE_END)

# ============================================================
# CREATE INTERACTIONS (6 interactions)
# ============================================================
df["PMxTemp"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["TAVG (°F, weekly mean)"]
df["PMxO3"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg 8h O3 (ppm)"]
df["PMxNO2"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg 1h NO2 (ppb)"]
df["TempxO3"] = df["TAVG (°F, weekly mean)"] * df["Avg 8h O3 (ppm)"]
df["TempxWind"] = df["TAVG (°F, weekly mean)"] * df["Avg_WND (mph, weekly mean)"]
df["PMxWind"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg_WND (mph, weekly mean)"]

INTERACTIONS = ["PMxTemp", "PMxO3", "PMxNO2", "TempxO3", "TempxWind", "PMxWind"]

# ============================================================
# HELPER
# ============================================================
def add_const_X(frame, feature_names):
    X = frame[feature_names].copy()
    X = sm.add_constant(X)
    return X

# ============================================================
# MODEL 1 — POOLED (WITHOUT INTERACTIONS)
# ============================================================
X_base_pooled = add_const_X(df.loc[pre_mask | post_mask, FEATURES], FEATURES)
y_pooled = df.loc[pre_mask | post_mask, COL_Y]

model_base = sm.OLS(y_pooled, X_base_pooled).fit()

print("\n==============================")
print(f"POOLED BASE MODEL (NO INTERACTIONS) — {COL_Y}")
print("==============================")
print(model_base.summary())

# ============================================================
# MODEL 2 — POOLED (WITH INTERACTIONS)
# ============================================================
ALL_FEATURES = FEATURES + INTERACTIONS

X_int_pooled = add_const_X(df.loc[pre_mask | post_mask, ALL_FEATURES], ALL_FEATURES)
model_int = sm.OLS(y_pooled, X_int_pooled).fit()

print("\n==============================")
print(f"POOLED MODEL WITH INTERACTIONS — {COL_Y}")
print("==============================")
print(model_int.summary())

# Compare performance
print("\n=== PERFORMANCE COMPARISON ===")
print(f"Adj R² (No Interactions): {model_base.rsquared_adj:.4f}")
print(f"Adj R² (With Interactions): {model_int.rsquared_adj:.4f}")
print(f"AIC Base: {model_base.aic:.2f} | AIC Int: {model_int.aic:.2f}")
print(f"BIC Base: {model_base.bic:.2f} | BIC Int: {model_int.bic:.2f}")

# ============================================================
# MODEL 3 — PRE-EVENT AND POST-EVENT (WITH INTERACTIONS)
# ============================================================
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
y_pre = df.loc[pre_mask, COL_Y]
X_pre = add_const_X(df.loc[pre_mask, ALL_FEATURES], ALL_FEATURES)
model_pre = sm.OLS(y_pre, X_pre).fit()

y_post = df.loc[post_mask, COL_Y]
X_post = add_const_X(df.loc[post_mask, ALL_FEATURES], ALL_FEATURES)
model_post = sm.OLS(y_post, X_post).fit()

# ============================================================
# SLOPE COMPARISON — INTERACTION TERMS ONLY
# ============================================================
coef_pre = model_pre.params[INTERACTIONS]
coef_post = model_post.params[INTERACTIONS]

coef_compare = pd.DataFrame({
    "Pre": coef_pre,
    "Post": coef_post,
    "Post-Pre": coef_post - coef_pre
})

print("\n==============================")
print(f"INTERACTION SLOPE COMPARISON — {COL_Y}")
print("==============================")
print(coef_compare)

# ============================================================
# CHOW TEST (WITH INTERACTIONS)
# ============================================================
def chow_test(model_pooled, model_pre, model_post, n1, n2, k):
    SSR_pooled = np.sum(model_pooled.resid**2)
    SSR_pre = np.sum(model_pre.resid**2)
    SSR_post = np.sum(model_post.resid**2)
    num = (SSR_pooled - (SSR_pre + SSR_post)) / k
    den = (SSR_pre + SSR_post) / (n1 + n2 - 2*k)
    F_stat = num / den
    p_value = f_dist.sf(F_stat, k, (n1+n2-2*k))
    return F_stat, p_value

n1 = X_pre.shape[0]
n2 = X_post.shape[0]
k = X_pre.shape[1]

F_stat, p_val = chow_test(model_int, model_pre, model_post, n1, n2, k)

print("\n==============================")
print(f"CHOW TEST WITH INTERACTIONS — {COL_Y}")
print("==============================")
print(f"F-statistic: {F_stat:.4f}, p-value: {p_val:.6f}")

# ============================================================
# VISUALIZATION — ACTUAL VS PREDICTED (WITH INTERACTIONS)
# ============================================================
df["Pred_Int"] = model_int.predict(add_const_X(df[ALL_FEATURES], ALL_FEATURES))

plt.figure(figsize=(10,5))
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual")
plt.plot(df[COL_WEEK], df["Pred_Int"], label="Predicted (with interactions)")
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, color='orange')
plt.title(f"{COL_Y}: Actual vs Predicted (With Interactions)")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
plt.xlabel("Week")
plt.ylabel("Weekly ER Visits")
plt.legend()
plt.tight_layout()
plt.show()

# ============================================================
# VISUALIZATION — RESIDUALS (FROM INTERACTION MODEL)
# ============================================================
df["Residuals_Int"] = df[COL_Y] - df["Pred_Int"]

plt.figure(figsize=(10,4))
plt.plot(df[COL_WEEK], df["Residuals_Int"], label="Residuals")
plt.axhline(0, linestyle="--", color="black", linewidth=1)

# Highlight wildfire window
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, color='orange', label="Wildfire")

plt.title(f"{COL_Y}: Residuals Over Time (Interaction Model)")
plt.xlabel("Week")
plt.ylabel("Residual (Actual - Predicted)")
plt.legend()
plt.tight_layout()
plt.show()
```

In [ ]:

```
# ============================================================
# UNIVERSAL SCRIPT: Interaction MLR + Pre/Post Slopes + Chow Test
# Works for: ILI, Respiratory, Stroke (just change COL_Y)
# ============================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

# ============================================================
# CONFIG — CHANGE ONLY THE OUTCOME VARIABLE HERE
# ============================================================
DATA_PATH = Path("Dataset.csv")

COL_WEEK = "Week Number"

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# CHANGE THIS LINE FOR EACH ANALYSIS:
# ILI:        COL_Y = "ILI_weekly_ER_Admissions"
# Respiratory: COL_Y = "Respiratory_weekly_ER_Admissions"
# Stroke:      COL_Y = "Stroke_weekly_ER_Admissions"
COL_Y = "Respiratory_Weekly_ER_Admissions"
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

FEATURES = [
    " Avg PM2.5 (ug/m3 LC)",
    "Avg 8h O3 (ppm)",
    "Avg 1h NO2 (ppb)",
    "TAVG (°F, weekly mean)",
    "Avg_WND (mph, weekly mean)",
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    "PRCP (inches, weekly sum)"
]

# Wildfire event window
WILDFIRE_START, WILDFIRE_END = 470, 475

# ============================================================
# LOAD & CLEAN
# ============================================================
df = pd.read_csv(DATA_PATH)

cols_needed = [COL_WEEK, COL_Y] + FEATURES
for c in cols_needed:
    df[c] = pd.to_numeric(df[c], errors="coerce")

df = df.dropna(subset=cols_needed).copy()
df = df.sort_values(COL_WEEK).reset_index(drop=True)

pre_mask = df[COL_WEEK] < WILDFIRE_START
post_mask = df[COL_WEEK] > WILDFIRE_END
wild_mask = df[COL_WEEK].between(WILDFIRE_START, WILDFIRE_END)

# ============================================================
# CREATE INTERACTIONS (6 interactions)
# ============================================================
df["PMxTemp"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["TAVG (°F, weekly mean)"]
df["PMxO3"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg 8h O3 (ppm)"]
df["PMxNO2"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg 1h NO2 (ppb)"]
df["TempxO3"] = df["TAVG (°F, weekly mean)"] * df["Avg 8h O3 (ppm)"]
df["TempxWind"] = df["TAVG (°F, weekly mean)"] * df["Avg_WND (mph, weekly mean)"]
df["PMxWind"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg_WND (mph, weekly mean)"]

INTERACTIONS = ["PMxTemp", "PMxO3", "PMxNO2", "TempxO3", "TempxWind", "PMxWind"]

# ============================================================
# HELPER
# ============================================================
def add_const_X(frame, feature_names):
    X = frame[feature_names].copy()
    X = sm.add_constant(X)
    return X

# ============================================================
# MODEL 1 — POOLED (WITHOUT INTERACTIONS)
# ============================================================
X_base_pooled = add_const_X(df.loc[pre_mask | post_mask, FEATURES], FEATURES)
y_pooled = df.loc[pre_mask | post_mask, COL_Y]

model_base = sm.OLS(y_pooled, X_base_pooled).fit()

print("\n==============================")
print(f"POOLED BASE MODEL (NO INTERACTIONS) — {COL_Y}")
print("==============================")
print(model_base.summary())

# ============================================================
# MODEL 2 — POOLED (WITH INTERACTIONS)
# ============================================================
ALL_FEATURES = FEATURES + INTERACTIONS
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
X_int_pooled = add_const_X(df.loc[pre_mask | post_mask, ALL_FEATURES], ALL_FEATURES)
model_int = sm.OLS(y_pooled, X_int_pooled).fit()

print("\n==============================")
print(f"POOLED MODEL WITH INTERACTIONS — {COL_Y}")
print("==============================")
print(model_int.summary())

# Compare performance
print("\n=== PERFORMANCE COMPARISON ===")
print(f"Adj R² (No Interactions): {model_base.rsquared_adj:.4f}")
print(f"Adj R² (With Interactions): {model_int.rsquared_adj:.4f}")
print(f"AIC Base: {model_base.aic:.2f} | AIC Int: {model_int.aic:.2f}")
print(f"BIC Base: {model_base.bic:.2f} | BIC Int: {model_int.bic:.2f}")

# ================================================================
# MODEL 3 — PRE-EVENT AND POST-EVENT (WITH INTERACTIONS)
# ================================================================
y_pre = df.loc[pre_mask, COL_Y]
X_pre = add_const_X(df.loc[pre_mask, ALL_FEATURES], ALL_FEATURES)
model_pre = sm.OLS(y_pre, X_pre).fit()

y_post = df.loc[post_mask, COL_Y]
X_post = add_const_X(df.loc[post_mask, ALL_FEATURES], ALL_FEATURES)
model_post = sm.OLS(y_post, X_post).fit()

# ================================================================
# SLOPE COMPARISON — INTERACTION TERMS ONLY
# ================================================================
coef_pre = model_pre.params[INTERACTIONS]
coef_post = model_post.params[INTERACTIONS]

coef_compare = pd.DataFrame({
    "Pre": coef_pre,
    "Post": coef_post,
    "Post-Pre": coef_post - coef_pre
})

print("\n==============================")
print(f"INTERACTION SLOPE COMPARISON — {COL_Y}")
print("==============================")
print(coef_compare)

# ================================================================
# CHOW TEST (WITH INTERACTIONS)
# ================================================================
def chow_test(model_pooled, model_pre, model_post, n1, n2, k):
    SSR_pooled = np.sum(model_pooled.resid**2)
    SSR_pre = np.sum(model_pre.resid**2)
    SSR_post = np.sum(model_post.resid**2)
    num = (SSR_pooled - (SSR_pre + SSR_post)) / k
    den = (SSR_pre + SSR_post) / (n1 + n2 - 2*k)
    F_stat = num / den
    p_value = f_dist.sf(F_stat, k, (n1+n2-2*k))
    return F_stat, p_value

n1 = X_pre.shape[0]
n2 = X_post.shape[0]
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
k = X_pre.shape[1]

F_stat, p_val = chow_test(model_int, model_pre, model_post, n1, n2, k)

print("\n==============================")
print(f"CHOW TEST WITH INTERACTIONS — {COL_Y}")
print("==============================")
print(f"F-statistic: {F_stat:.4f}, p-value: {p_val:.6f}")

# ================================================================
# VISUALIZATION — ACTUAL VS PREDICTED (WITH INTERACTIONS)
# ================================================================
df["Pred_Int"] = model_int.predict(add_const_X(df[ALL_FEATURES], ALL_FEATURES))

plt.figure(figsize=(10,5))
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual")
plt.plot(df[COL_WEEK], df["Pred_Int"], label="Predicted (with interactions)")
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, color='orange')
plt.title(f"{COL_Y}: Actual vs Predicted (With Interactions)")
plt.xlabel("Week")
plt.ylabel("Weekly ER Visits")
plt.legend()
plt.tight_layout()
plt.show()

# ================================================================
# VISUALIZATION — RESIDUALS (FROM INTERACTION MODEL)
# ================================================================
df["Residuals_Int"] = df[COL_Y] - df["Pred_Int"]

plt.figure(figsize=(10,4))
plt.plot(df[COL_WEEK], df["Residuals_Int"], label="Residuals")
plt.axhline(0, linestyle="--", color="black", linewidth=1)

# Highlight wildfire window
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, color='orange', label="Wildfire")

plt.title(f"{COL_Y}: Residuals Over Time (Interaction Model)")
plt.xlabel("Week")
plt.ylabel("Residual (Actual - Predicted)")
plt.legend()
plt.tight_layout()
plt.show()
```

In [ ]:

```
# ================================================================
# UNIVERSAL SCRIPT: Interaction MLR + Pre/Post Slopes + Chow Test
# Works for: ILI, Respiratory, Stroke (just change COL_Y)
# ================================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import f as f_dist

# ================================================================
# CONFIG — CHANGE ONLY THE OUTCOME VARIABLE HERE
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```python
# ============================================================
DATA_PATH = Path("Dataset.csv")

COL_WEEK = "Week Number"

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# CHANGE THIS LINE FOR EACH ANALYSIS:
# ILI:       COL_Y = "ILI_weekly_ER_Admissions"
# Respiratory: COL_Y = "Respiratory_weekly_ER_Admissions"
# Stroke:      COL_Y = "Stroke_weekly_ER_Admissions"
COL_Y = "# stroke hospital discharges (weekly total)"
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

FEATURES = [
    " Avg PM2.5 (ug/m3 LC)",
    "Avg 8h O3 (ppm)",
    "Avg 1h NO2 (ppb)",
    "TAVG (°F, weekly mean)",
    "Avg_WND (mph, weekly mean)",
    "PRCP (inches, weekly sum)"
]

# Wildfire event window
WILDFIRE_START, WILDFIRE_END = 470, 475

# ============================================================
# LOAD & CLEAN
# ============================================================
df = pd.read_csv(DATA_PATH)

cols_needed = [COL_WEEK, COL_Y] + FEATURES
for c in cols_needed:
    df[c] = pd.to_numeric(df[c], errors="coerce")

df = df.dropna(subset=cols_needed).copy()
df = df.sort_values(COL_WEEK).reset_index(drop=True)

pre_mask = df[COL_WEEK] < WILDFIRE_START
post_mask = df[COL_WEEK] > WILDFIRE_END
wild_mask = df[COL_WEEK].between(WILDFIRE_START, WILDFIRE_END)

# ============================================================
# CREATE INTERACTIONS (6 interactions)
# ============================================================
df["PMxTemp"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["TAVG (°F, weekly mean)"]
df["PMxO3"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg 8h O3 (ppm)"]
df["PMxNO2"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg 1h NO2 (ppb)"]
df["TempxO3"] = df["TAVG (°F, weekly mean)"] * df["Avg 8h O3 (ppm)"]
df["TempxWind"] = df["TAVG (°F, weekly mean)"] * df["Avg_WND (mph, weekly mean)"]
df["PMxWind"] = df[" Avg PM2.5 (ug/m3 LC)"] * df["Avg_WND (mph, weekly mean)"]

INTERACTIONS = ["PMxTemp", "PMxO3", "PMxNO2", "TempxO3", "TempxWind", "PMxWind"]

# ============================================================
# HELPER
# ============================================================
def add_const_X(frame, feature_names):
    X = frame[feature_names].copy()
    X = sm.add_constant(X)
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
    return X

# ============================================================
# MODEL 1 — POOLED (WITHOUT INTERACTIONS)
# ============================================================
X_base_pooled = add_const_X(df.loc[pre_mask | post_mask, FEATURES], FEATURES)
y_pooled = df.loc[pre_mask | post_mask, COL_Y]

model_base = sm.OLS(y_pooled, X_base_pooled).fit()

print("\n==============================")
print(f"POOLED BASE MODEL (NO INTERACTIONS) — {COL_Y}")
print("==============================")
print(model_base.summary())

# ============================================================
# MODEL 2 — POOLED (WITH INTERACTIONS)
# ============================================================
ALL_FEATURES = FEATURES + INTERACTIONS

X_int_pooled = add_const_X(df.loc[pre_mask | post_mask, ALL_FEATURES], ALL_FEATURES)
model_int = sm.OLS(y_pooled, X_int_pooled).fit()

print("\n==============================")
print(f"POOLED MODEL WITH INTERACTIONS — {COL_Y}")
print("==============================")
print(model_int.summary())

# Compare performance
print("\n=== PERFORMANCE COMPARISON ===")
print(f"Adj R² (No Interactions): {model_base.rsquared_adj:.4f}")
print(f"Adj R² (With Interactions): {model_int.rsquared_adj:.4f}")
print(f"AIC Base: {model_base.aic:.2f} | AIC Int: {model_int.aic:.2f}")
print(f"BIC Base: {model_base.bic:.2f} | BIC Int: {model_int.bic:.2f}")

# ============================================================
# MODEL 3 — PRE-EVENT AND POST-EVENT (WITH INTERACTIONS)
# ============================================================
y_pre = df.loc[pre_mask, COL_Y]
X_pre = add_const_X(df.loc[pre_mask, ALL_FEATURES], ALL_FEATURES)
model_pre = sm.OLS(y_pre, X_pre).fit()

y_post = df.loc[post_mask, COL_Y]
X_post = add_const_X(df.loc[post_mask, ALL_FEATURES], ALL_FEATURES)
model_post = sm.OLS(y_post, X_post).fit()

# ============================================================
# SLOPE COMPARISON — INTERACTION TERMS ONLY
# ============================================================
coef_pre = model_pre.params[INTERACTIONS]
coef_post = model_post.params[INTERACTIONS]

coef_compare = pd.DataFrame({
    "Pre": coef_pre,
    "Post": coef_post,
    "Post-Pre": coef_post - coef_pre
})

print("\n==============================")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
print(f"INTERACTION SLOPE COMPARISON — {COL_Y}")
print("==============================")
print(coef_compare)

# ============================================================
# CHOW TEST (WITH INTERACTIONS)
# ============================================================
def chow_test(model_pooled, model_pre, model_post, n1, n2, k):
    SSR_pooled = np.sum(model_pooled.resid**2)
    SSR_pre = np.sum(model_pre.resid**2)
    SSR_post = np.sum(model_post.resid**2)
    num = (SSR_pooled - (SSR_pre + SSR_post)) / k
    den = (SSR_pre + SSR_post) / (n1 + n2 - 2*k)
    F_stat = num / den
    p_value = f_dist.sf(F_stat, k, (n1+n2-2*k))
    return F_stat, p_value

n1 = X_pre.shape[0]
n2 = X_post.shape[0]
k = X_pre.shape[1]

F_stat, p_val = chow_test(model_int, model_pre, model_post, n1, n2, k)

print("\n==============================")
print(f"CHOW TEST WITH INTERACTIONS — {COL_Y}")
print("==============================")
print(f"F-statistic: {F_stat:.4f}, p-value: {p_val:.6f}")

# ============================================================
# VISUALIZATION — ACTUAL VS PREDICTED (WITH INTERACTIONS)
# ============================================================
df["Pred_Int"] = model_int.predict(add_const_X(df[ALL_FEATURES], ALL_FEATURES))

plt.figure(figsize=(10,5))
plt.plot(df[COL_WEEK], df[COL_Y], label="Actual")
plt.plot(df[COL_WEEK], df["Pred_Int"], label="Predicted (with interactions)")
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, color='orange')
plt.title(f"{COL_Y}: Actual vs Predicted (With Interactions)")
plt.xlabel("Week")
plt.ylabel("Weekly ER Visits")
plt.legend()
plt.tight_layout()
plt.show()

# ============================================================
# VISUALIZATION — RESIDUALS (FROM INTERACTION MODEL)
# ============================================================
df["Residuals_Int"] = df[COL_Y] - df["Pred_Int"]

plt.figure(figsize=(10,4))
plt.plot(df[COL_WEEK], df["Residuals_Int"], label="Residuals")
plt.axhline(0, linestyle="--", color="black", linewidth=1)

# Highlight wildfire window
plt.axvspan(WILDFIRE_START, WILDFIRE_END, alpha=0.2, color='orange', label="Wildfire")

plt.title(f"{COL_Y}: Residuals Over Time (Interaction Model)")
plt.xlabel("Week")
plt.ylabel("Residual (Actual - Predicted)")
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

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

In [ ]:

```

```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF