# CS264_final_project

November 6, 2025

## 1 Proof-of-concept forecasting: predict daily PM2.5 in Lucknow

***Models: Linear Regression and Random Forest***

***Outputs: metrics + multiple diagnostic plots & Insights + Predictions & Recommendation***

***Author: Alfayez Ahmad***

[9]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

%matplotlib inline
sns.set(style="whitegrid", font_scale=1.0)
plt.rcParams["figure.dpi"] = 140
```

#### 1.0.1 1) Load data

[10]:
```
desktop = os.path.join(os.path.expanduser("~"), "Desktop")
file_path = os.path.join(desktop, "ML Lucknow.csv")
df = pd.read_csv(file_path, parse_dates=["date"])

# Sanity: ensure expected columns exist
if "pm25" not in df.columns:
    raise ValueError("Column 'pm25' not found. Ensure your pivot produced a␣
↪'pm25' column.")
```

### 1.0.2 2) Clean + feature engineering

[11]:
```
# Sort by date
df = df.sort_values("date").reset_index(drop=True)

# Forward/backward fill short gaps within available periods (keeps the POC␣
↪simple)
df_ffill = df.copy()
df_ffill = df_ffill.fillna(method="ffill").fillna(method="bfill")

# Basic lag features for pm25 (useful for day-to-day persistence)
df_ffill["pm25_lag1"] = df_ffill["pm25"].shift(1)
df_ffill["pm25_lag7"] = df_ffill["pm25"].shift(7)
df_ffill["pm25_rolling7"] = df_ffill["pm25"].rolling(window=7, min_periods=1).
↪mean()

# Dropping the first few rows with NaNs introduced by lagging (keeps training␣
↪clean)
df_model = df_ffill.dropna(subset=["pm25_lag1", "pm25_lag7", "pm25_rolling7"]).
↪copy()

# Features: all pollutants except target + lag features
feature_cols = [c for c in df_model.columns if c not in ["date", "pm25"]]
X = df_model[feature_cols]
y = df_model["pm25"]
```

```
/var/folders/9m/1l175qpn7nz5sh3xkzj82y_c0000gn/T/ipykernel_1077/992747772.py:6:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  df_ffill = df_ffill.fillna(method="ffill").fillna(method="bfill")
```

### 1.0.3 3) Train/test split (time-based)

[12]:
```
# For a proof-of-concept, keeping the last 20% as test without shuffling
split_idx = int(len(df_model) * 0.8)
X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]
y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]
dates_test = df_model["date"].iloc[split_idx:]
```

### 1.0.4 4) Models: Linear Regression + Random Forest

[13]:
```
# Scaling features for Linear Regression (since RF does not need scaling)
scaler = StandardScaler()
X_train_lr = scaler.fit_transform(X_train)
X_test_lr = scaler.transform(X_test)

linreg = LinearRegression()
```

```
linreg.fit(X_train_lr, y_train)
y_pred_lr = linreg.predict(X_test_lr)

rf = RandomForestRegressor(
    n_estimators=300,
    max_depth=None,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

### 1.0.5 5) Metrics

[14]:
```
def mape(y_true, y_pred):
    # Guard against division by zero
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    nonzero = y_true != 0
    return np.mean(np.abs((y_true[nonzero] - y_pred[nonzero]) /␣
 ↪y_true[nonzero])) * 100 if nonzero.any() else np.nan

def evaluate_all(y_true, y_pred, name):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    mp = mape(y_true, y_pred)
    print(f"{name}:")
    print(f"  MAE  = {mae:.2f}")
    print(f"  RMSE = {rmse:.2f}")
    print(f"  R²   = {r2:.3f}")
    print(f"  MAPE = {mp:.2f}%")
    return {"MAE": mae, "RMSE": rmse, "R2": r2, "MAPE": mp}

metrics_lr = evaluate_all(y_test, y_pred_lr, "Linear Regression")
metrics_rf = evaluate_all(y_test, y_pred_rf, "Random Forest")

# H) Summary metrics saved as CSV
#metrics_df = pd.DataFrame([
    #{"model": "Linear Regression", **metrics_lr},
    #{"model": "Random Forest", **metrics_rf}
#])
#metrics_df.to_csv(os.path.join(out_dir, "metrics_summary.csv"), index=False)

#print("Done. Outputs saved to:", out_dir)
#print(metrics_df)
```

```
Linear Regression:
  MAE  = 26.65
  RMSE = 32.16
  R²   = 0.833
  MAPE = 17.95%
Random Forest:
  MAE  = 18.76
  RMSE = 21.63
  R²   = 0.925
  MAPE = 12.91%
```

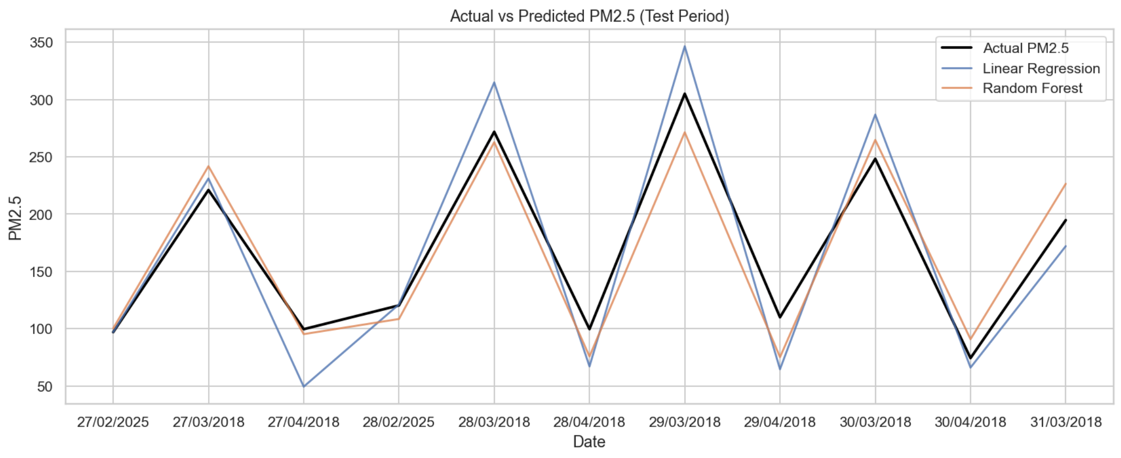
### 1.0.6 6) Plots

**A) Actual vs Predicted over time**

[29]:
```python
#out_dir = os.path.join(desktop, "lucknow_poc_outputs")
#os.makedirs(out_dir, exist_ok=True)



plt.figure(figsize=(12,5))
plt.plot(dates_test, y_test, label="Actual PM2.5", color="black", linewidth=2)
plt.plot(dates_test, y_pred_lr, label="Linear Regression", alpha=0.8)
plt.plot(dates_test, y_pred_rf, label="Random Forest", alpha=0.8)
plt.title("Actual vs Predicted PM2.5 (Test Period)")
plt.xlabel("Date")
plt.ylabel("PM2.5")
plt.legend()
plt.tight_layout()
#plt.savefig(os.path.join(out_dir, "actual_vs_predicted.png"))
plt.show()
plt.close()
```



***Overall Performance*: Both the Linear Regression and Random Forest models generally follow the trend of the actual PM2.5 values. The predicted values rise and fall at roughly the same times as the actual values.**

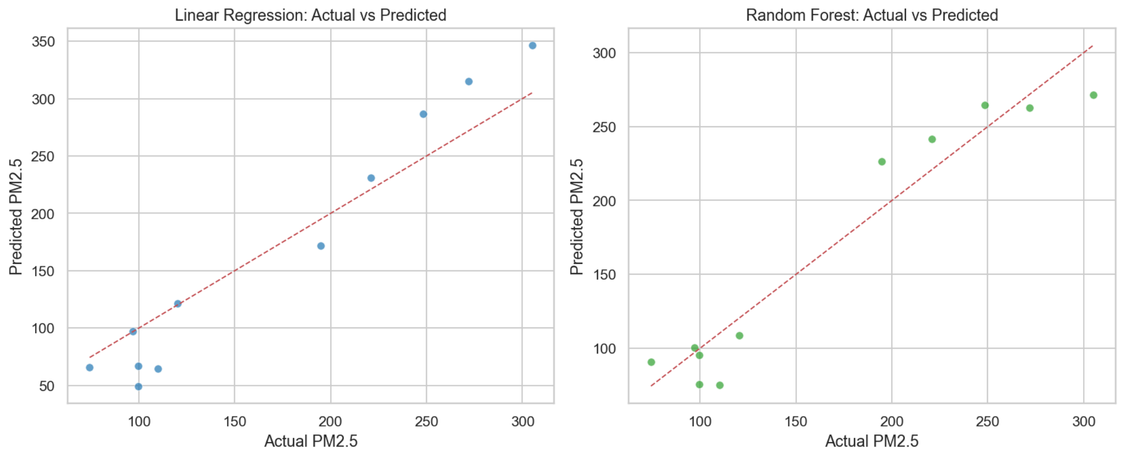
- ***Random Forest Model (Orange line):*** This model appears to be a more accurate predictor of PM2.5 concentrations during the test period. Its line stays closer to the "Actual PM2.5" line (black line), especially at the peaks and troughs, indicating a better fit to the data.
- ***Linear Regression Model (Blue line):*** This model's predictions are less accurate than the Random Forest model. The blue line consistently underestimates the peaks and overestimates the troughs. For example, during the peak around March 29th, the Linear Regression model's prediction is significantly lower than the actual value, while the Random Forest model's prediction is very close. Similarly, at the trough around March 30th, the Linear Regression model's prediction is higher than the actual value. **In summary**, the Random Forest model demonstrates superior performance in this test period, capturing the true fluctuations in PM2.5 levels more effectively than the Linear Regression model.

**B) Scatter: Actual vs Predicted**

[30]:
```
fig, axes = plt.subplots(1, 2, figsize=(12,5))
sns.scatterplot(x=y_test, y=y_pred_lr, ax=axes[0], color="#1f77b4", alpha=0.7)
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], "r--",␣
 ↪linewidth=1)
axes[0].set_title("Linear Regression: Actual vs Predicted")
axes[0].set_xlabel("Actual PM2.5")
axes[0].set_ylabel("Predicted PM2.5")

sns.scatterplot(x=y_test, y=y_pred_rf, ax=axes[1], color="#2ca02c", alpha=0.7)
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], "r--",␣
 ↪linewidth=1)
axes[1].set_title("Random Forest: Actual vs Predicted")
axes[1].set_xlabel("Actual PM2.5")
axes[1].set_ylabel("Predicted PM2.5")

plt.tight_layout()
#plt.savefig(os.path.join(out_dir, "scatter_actual_vs_predicted.png"))
plt.show()
plt.close()
```



***Scatter Plot Analysis*** Both graphs plot the actual values on the x-axis and the predicted values on the y-axis. The red diagonal line represents a perfect prediction where the predicted value is exactly the same as the actual value. A good model's data points will cluster closely along this line.

- ***Linear Regression:*** The scatter plot shows a linear relationship, with the data points loosely following the red line. However, some points, especially at the higher ends of the scale, show a larger deviation from the line, which suggests the model may not be capturing all the complexities of the data. Linear regression assumes a linear relationship between variables, which may not always be accurate for real-world data.
- ***Random Forest:*** The scatter plot for the Random Forest model shows the data points are more tightly clustered around the red line, with less deviation than the linear regression model. This suggests that the Random Forest model is performing better at predicting the PM2.5 values, likely because it is better at handling complex, non-linear relationships in the data without making assumptions about its distribution.

**C) Residual histograms**

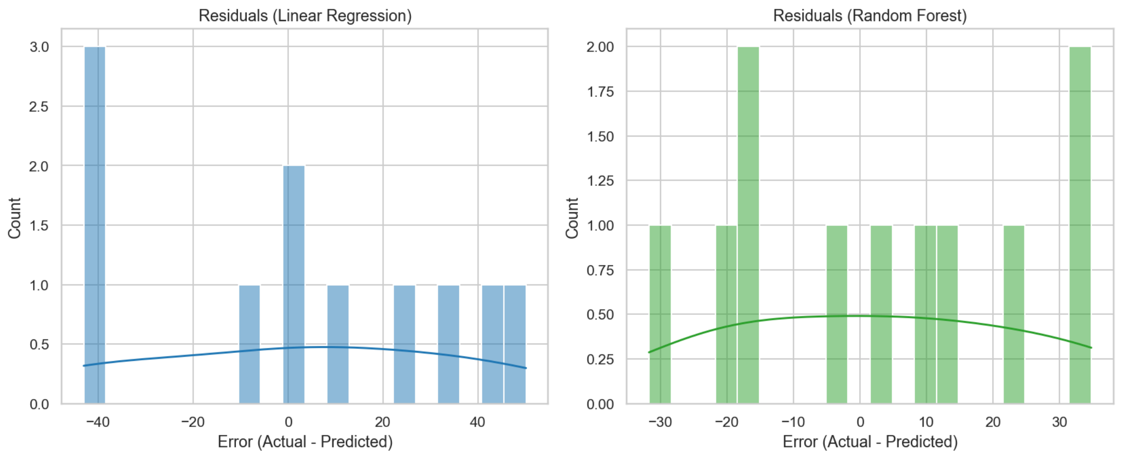
[31]:
```python
res_lr = y_test - y_pred_lr
res_rf = y_test - y_pred_rf

fig, axes = plt.subplots(1, 2, figsize=(12,5))
sns.histplot(res_lr, bins=20, kde=True, ax=axes[0], color="#1f77b4")
axes[0].set_title("Residuals (Linear Regression)")
axes[0].set_xlabel("Error (Actual - Predicted)")

sns.histplot(res_rf, bins=20, kde=True, ax=axes[1], color="#2ca02c")
axes[1].set_title("Residuals (Random Forest)")
axes[1].set_xlabel("Error (Actual - Predicted)")

plt.tight_layout()
#plt.savefig(os.path.join(out_dir, "residual_histograms.png"))
```

```
plt.show()
plt.close()
```



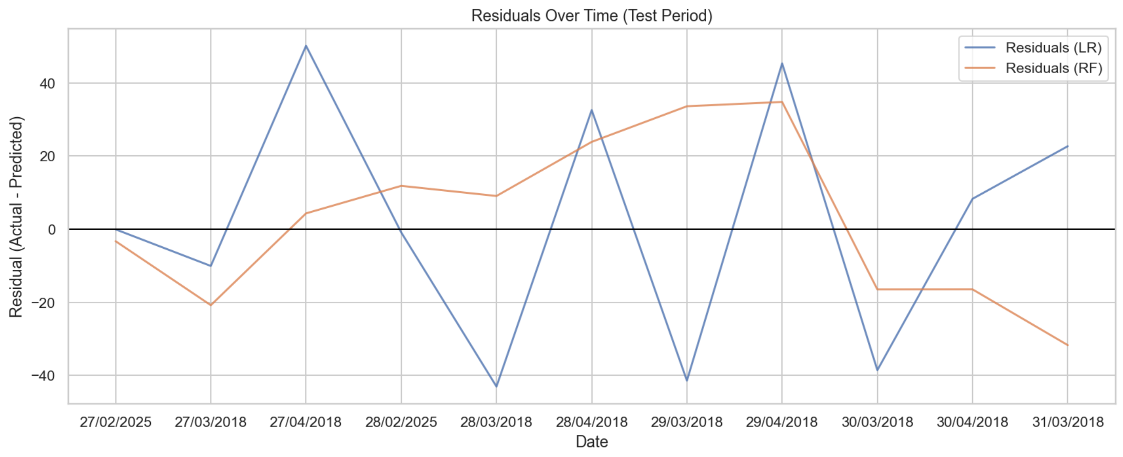
***Analysis of the Residual Plots***

- ***Residuals:*** In machine learning, a residual is the difference between an actual data value and the value predicted by the model (Error = Actual - Predicted). Analyzing these residuals helps to evaluate a model's validity and fitness. A good model typically has residuals that are randomly distributed around zero.
- ***Residuals (Linear Regression):*** The histogram on the left shows the residuals for a Linear Regression model. The distribution of the errors appears to be centered around zero and roughly follows a normal distribution, which is an assumption of linear regression. The plot indicates a random scattering of residuals, suggesting that the linear model might be a suitable fit for the data.
- ***Residuals (Random Forest):*** The histogram on the right shows the residuals for a Random Forest model. The errors are more concentrated around zero, and the distribution appears to be taller and narrower compared to the linear regression plot. This suggests that the Random Forest model's predictions are, on average, closer to the actual values than those of the linear regression model.

**D) Residuals over time**

[32]:
```
plt.figure(figsize=(12,5))
plt.plot(dates_test, res_lr, label="Residuals (LR)", alpha=0.8)
plt.plot(dates_test, res_rf, label="Residuals (RF)", alpha=0.8)
plt.axhline(0, color="black", linewidth=1)
plt.title("Residuals Over Time (Test Period)")
plt.xlabel("Date")
plt.ylabel("Residual (Actual - Predicted)")
plt.legend()
plt.tight_layout()
#plt.savefig(os.path.join(out_dir, "residuals_over_time.png"))
```

```
plt.show()
plt.close()
```



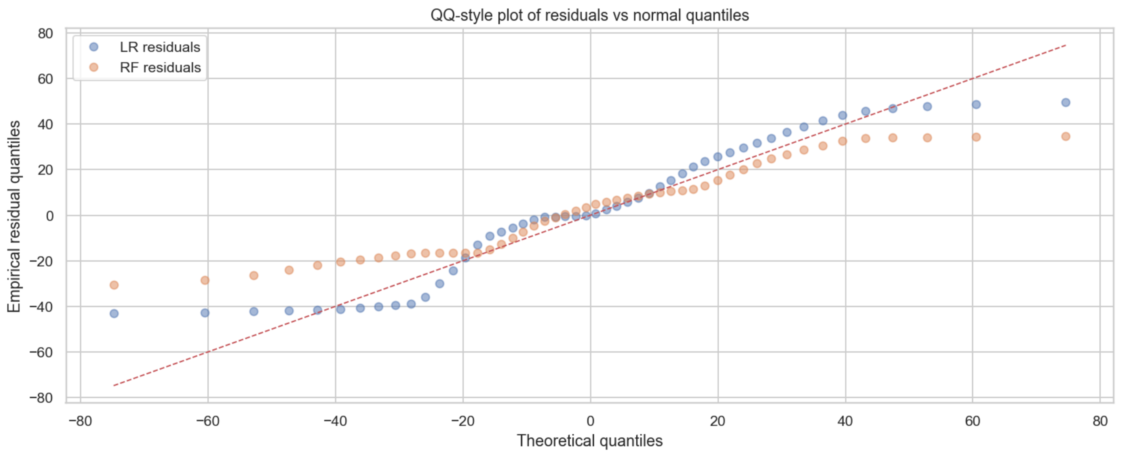
***Residuals:* The y-axis represents the "Residual (Actual - Predicted)" values. A residual is the difference between a model's predicted value and the actual observed value.**

- ***Time Series Data:*** The x-axis shows dates, indicating that the analysis is being performed on time-series data.
- ***Two Models:*** The plot compares the residuals of two different models:
- ***Residuals (LR):*** Likely from a Linear Regression model.
- ***Residuals (RF):*** Likely from a Random Forest model.
- ***Plot Interpretation:*** Ideally, residuals should be randomly scattered around the zero line with no discernible pattern. A pattern, such as a curve or a systematic trend, suggests that the model is not adequately capturing the relationship in the data. The plot in the image shows how the prediction errors for both models change over time, which helps in diagnosing the adequacy and assumptions of the models.

**E) QQ-like diagnostic using quantiles (quick check of error normality)**

[33]:
```
quantiles = np.linspace(0.01, 0.99, 50)
lr_q = np.quantile(res_lr, quantiles)
rf_q = np.quantile(res_rf, quantiles)
theory_q = np.quantile(np.random.normal(0, np.std(res_lr), size=100000),␣
 ↪quantiles)

plt.figure(figsize=(12,5))
plt.plot(theory_q, lr_q, "o", alpha=0.5, label="LR residuals")
plt.plot(theory_q, rf_q, "o", alpha=0.5, label="RF residuals")
plt.plot([theory_q.min(), theory_q.max()], [theory_q.min(), theory_q.max()],␣
 ↪"r--", linewidth=1)
plt.title("QQ-style plot of residuals vs normal quantiles")
plt.xlabel("Theoretical quantiles")
plt.ylabel("Empirical residual quantiles")
plt.legend()
plt.tight_layout()
#plt.savefig(os.path.join(out_dir, "qq_residuals.png"))
plt.show()
plt.close()
```

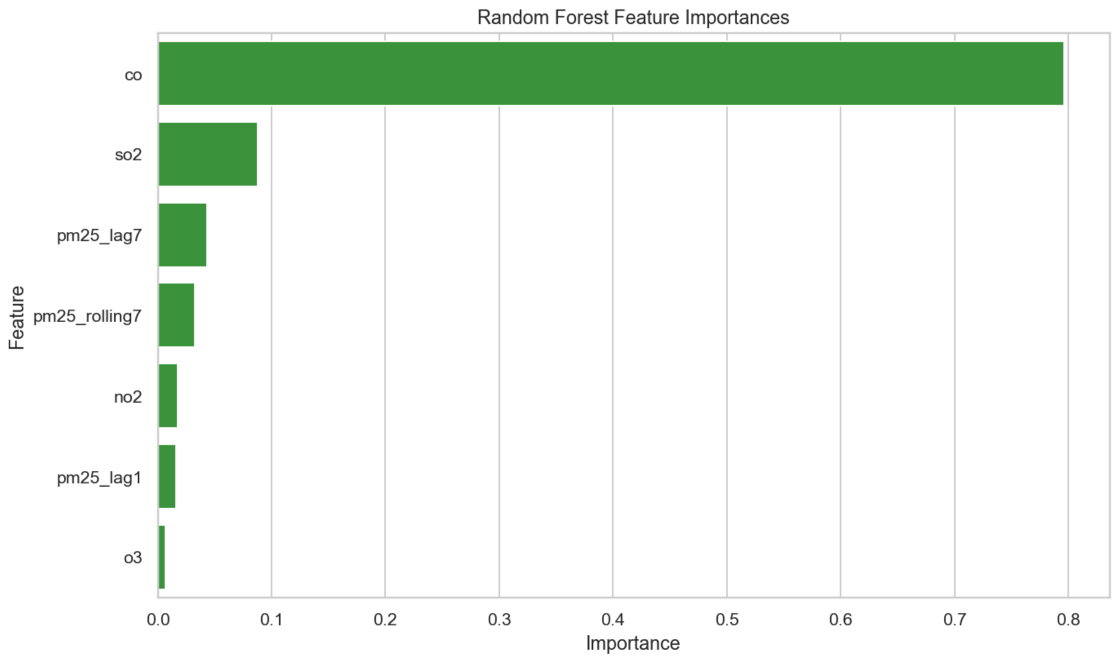


***Q-Q Plot Analysis*** The plot of **residuals vs normal quantiles**, compares the empirical quantiles of two sets of residuals (from an LR model and an RF model) against the theoretical quantiles of a normal distribution. In this type of plot, if the data follows a normal distribution, the points should align closely with the diagonal reference line.

- ***LR residuals:*** The blue data points representing **LR residuals** are generally close to the straight line, suggesting that these residuals are approximately normally distributed.
- ***RF residuals:*** The orange data points for **RF residuals** also follow the line, but with more deviation at the extreme ends. This could indicate that the residuals from the Random Forest model may have slightly heavier tails or more extreme values than a normal distribution would predict.

**F) Feature importances (Random Forest)**

[34]:
```
importances = pd.Series(rf.feature_importances_, index=feature_cols).
  ↪sort_values(ascending=False)
plt.figure(figsize=(10,6))
sns.barplot(x=importances.values, y=importances.index, color="#2ca02c")
plt.title("Random Forest Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
#plt.savefig(os.path.join(out_dir, "feature_importances_rf.png"))
plt.show()
plt.close()
```

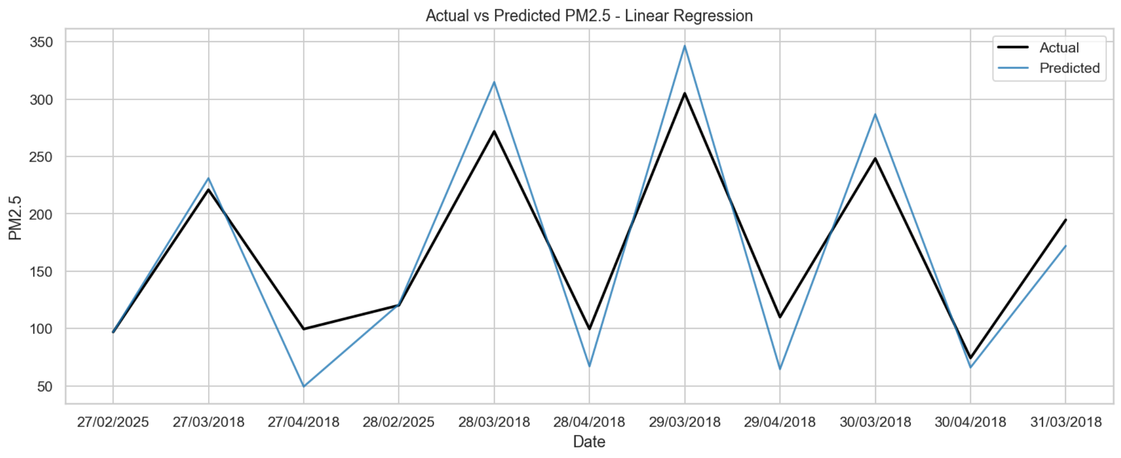


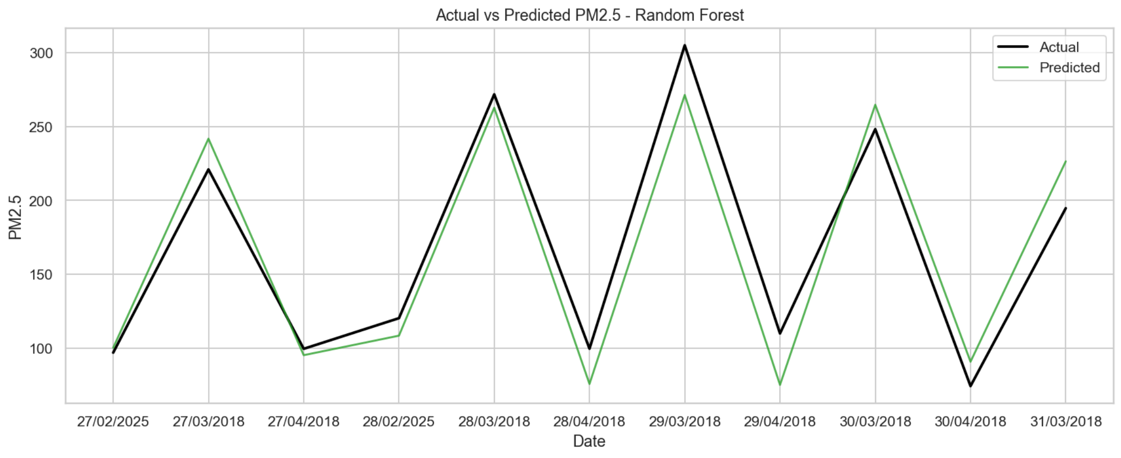
***Feature Importance:*** **In a Random Forest model, feature importance is a value assigned to each feature that represents how much that feature contributes to the model's predictive power. Features with a higher importance score are considered more significant for making accurate predictions.**

- ***Interpretation of the Chart:*** The length of each bar corresponds to the importance score of a specific feature. In this chart, the feature with the longest bar has the highest importance.
- ***Features:*** The features being analyzed are listed on the y-axis: pm25_rolling7, so2, pm25_lag7, pm25_lag1, no2, and o3. The chart shows that **CO** has the highest importance, followed by **so2** and **pm25_rolling7**. The other features have significantly lower importance scores.

### G) Predicted vs Actual line per model (separate)

[35]:
```
for name, preds, color in [
    ("linear_regression", y_pred_lr, "#1f77b4"),
    ("random_forest", y_pred_rf, "#2ca02c")
]:
    plt.figure(figsize=(12,5))
    plt.plot(dates_test, y_test, label="Actual", color="black", linewidth=2)
    plt.plot(dates_test, preds, label="Predicted", color=color, alpha=0.8)
    plt.title(f"Actual vs Predicted PM2.5 - {name.replace('_', ' ').title()}")
    plt.xlabel("Date")
    plt.ylabel("PM2.5")
    plt.legend()
    plt.tight_layout()
    #plt.savefig(os.path.join(out_dir, f"actual_vs_predicted_{name}.png"))
    plt.show()
    plt.close()
```





***Random Forest Appears to be a Better Fit:*** **The Random Forest model's "predicted" line (green) more closely follows the "actual" line (black) than the Linear Regression model's "predicted" line (blue) follows its corresponding "actual" line. This indicates that the Random Forest model is likely a better fit for the data, as it is more successful at capturing the fluctuations and trends in the actual PM2.5 values.**

- ***Linear Regression Underperforms:*** The Linear Regression model's "predicted" line is smoother and does not capture the sharp peaks and valleys seen in the actual data. This is expected for a linear model, which assumes a straightforward linear relationship between variables and struggles to model non-linear or complex patterns in the data.
- ***Difference in Predictive Power:*** The Random Forest model shows a closer alignment with the actual values, suggesting it has a higher predictive power and a lower prediction

error compared to the Linear Regression model for this specific dataset. *This is a common finding, as Random Forest models, which are a type of ensemble learning, are often more flexible and can handle more complex relationships than linear models.*

```
[36]: # === Next-Day Forecast with Category + Recommendations ===

def pollution_category(pm25):
    if pm25 <= 12:
        return "Good"
    elif pm25 <= 35.4:
        return "Moderate"
    elif pm25 <= 55.4:
        return "Unhealthy for Sensitive Groups"
    elif pm25 <= 150.4:
        return "Unhealthy"
    elif pm25 <= 250.4:
        return "Very Unhealthy"
    else:
        return "Hazardous"

def recommendations(category):
    if category == "Good":
        return ["Air quality is satisfactory.", "Enjoy outdoor activities␣
 ↪freely."]
    elif category == "Moderate":
        return ["Air quality is acceptable.", "Sensitive individuals should␣
 ↪limit prolonged outdoor exertion."]
    elif category == "Unhealthy for Sensitive Groups":
        return ["Sensitive groups should reduce outdoor activity.", "Consider␣
 ↪wearing a mask if outdoors."]
    elif category == "Unhealthy":
        return ["Everyone may begin to experience health effects.", "Limit␣
 ↪outdoor activities.", "Use air purifiers indoors."]
    elif category == "Very Unhealthy":
        return ["Health alert: everyone may experience serious effects.",␣
 ↪"Avoid outdoor activity.", "Keep windows closed."]
    else:  # Hazardous
        return ["Emergency conditions: serious health effects for all.", "Stay␣
 ↪indoors with filtered air.", "Follow local advisories."]

#lag only features for next day
last_row = df_model.iloc[-1]
pm25_t = last_row["pm25"]
pm25_t_minus_6 = df_model.iloc[-7]["pm25"] if len(df_model) >= 7 else np.nan
pm25_roll7_tplus1 = df_model["pm25"].iloc[-7:].mean()

lag_next = pd.DataFrame([{
    "pm25_lag1": pm25_t,
    "pm25_lag7": pm25_t_minus_6,
    "pm25_rolling7": pm25_roll7_tplus1
}])

#RF on lag-only features
rf_lag = RandomForestRegressor(n_estimators=300, min_samples_leaf=2,␣
 ↪random_state=42, n_jobs=-1)
X_lag = df_model[["pm25_lag1","pm25_lag7","pm25_rolling7"]].dropna()
y_lag = df_model.loc[X_lag.index, "pm25"]
rf_lag.fit(X_lag, y_lag)

# Forecast next day
forecast_date = pd.to_datetime(df_model["date"].iloc[-1]) + pd.Timedelta(days=1)
next_day_pred = rf_lag.predict(lag_next)[0] if not lag_next.isna().any().any()␣
 ↪else np.nan

# Classify + recommend
category = pollution_category(next_day_pred)
recs = recommendations(category)

print(f"Forecast date: {forecast_date.strftime('%Y-%m-%d')}")
print(f"Predicted PM2.5: {next_day_pred:.1f} µg/m³")
print(f"Pollution category: {category}")
print("Recommendations:")
for r in recs:
    print(f" - {r}")
```

```
Forecast date: 2018-04-01
Predicted PM2.5: 168.3 µg/m³
Pollution category: Very Unhealthy
Recommendations:
 - Health alert: everyone may experience serious effects.
 - Avoid outdoor activity.
 - Keep windows closed.
```

```
/var/folders/9m/1ll75qpn7nz5sh3xkzj82y_c0000gn/T/ipykernel_1077/2965988234.py:50
: UserWarning: Parsing dates in %d/%m/%Y format when dayfirst=False (the
default) was specified. Pass `dayfirst=True` or specify a format to silence this
warning.
  forecast_date = pd.to_datetime(df_model["date"].iloc[-1]) +
pd.Timedelta(days=1)
```

[ ]: