



**BERLIN SCHOOL OF
BUSINESS & INNOVATION**

Essay / Assignment Title: Time Series Forecasting: A Practical Approach to Data-Driven Decision Making

Programme title: M.sc Data Analytics

Name: Muhammad Saqlain

Year: 2025-2026

CONTENTS

Table of Contents

TASK #1	PROBLEM FORMATION	5
Problem Formation for Task 1:		
1. Dataset Selection:		5
2. Dataset Description:		5
3. Justification for Dataset Choice:		6
Implementation:		9
Code:		9
Output:		12
TASK #2	PROBLEM FORMATION	17
Problem formation for task 2:		
Implementation:		17
Output:		21
<i>Final_processed dataframe Head:</i>		24
<i>Final_processed dataframe info:</i>		24
TASK # 3:	PROBLEM FORMATION	26
Problem Formation for Task 3:		
Implementation:		
Code:		26
Output:		30
Observations of Actual vs Predicted PM2.5 (Linear Regression):		32
Best Model:		34
Observations of Actual vs Predicted PM2.5 (Random Forest Tuned):		35
Final Evaluation:		36
TASK# 4	CONCLUSION:	37
Key Insights about Time Series Dynamics:		
		37
BIBLIOGRAPHY		38



Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the postgraduate program).

Name and Surname :
Muhammad Saqlain

Date:

TASK #1

PROBLEM FORMATION

Problem Formation for Task 1:

To select an appropriate time series dataset with a continuous target variable, at least 1000 data points, and multiple features from a reputable source. Then, conduct initial exploratory data analysis (including summary statistics and visualizations) to understand its characteristics and justify its suitability for a time series regression project.

1. Dataset Selection:

After reviewing the requirements (time series, continuous target variable, ≥ 1000 data points, multiple features, reputable source), the following dataset has been selected:

- **Dataset Name:** Beijing Multi-Site Air-Quality Data (specifically, data from the Aotizhongxin station)
- **Source:** UCI Machine Learning Repository
 - **Citation:** Liang, X., Zou, T., Guo, B., Li, S., Zhang, H., Zhang, S., Huang, H. and Chen, S. X. (2017). Assessing Beijing's PM2.5 pollution: severity, weather impact, APEC blue and winter heating. Proceedings of the Royal Society A, 473(2204).
 - **Direct Link to Dataset:** <https://archive.ics.uci.edu/dataset/501/beijing+multi-site+air-quality+data>
 - **Specific File:** PRSA_Data_Aotizhongxin_20130301-20170228.csv (This file contains data for the Aotizhongxin station).

2. Dataset Description:

- **Overview:** This dataset contains hourly air quality data collected from the Aotizhongxin air quality monitoring station in Beijing, China. Alongside pollutant concentrations, it includes meteorological data for the corresponding times.
- **Time Period and Granularity:** The data spans from March 1st, 2013, to February 28th, 2017, with observations recorded on an hourly basis.
- **Data Points:** The Aotizhongxin station dataset comprises 35,064 hourly observations. This significantly exceeds the minimum requirement of 1000 data points.
- **Target Variable:** For this assignment, the **PM2.5 concentration ($\mu\text{g}/\text{m}^3$)** will be used as the continuous target variable for regression analysis. PM2.5 refers to fine particulate matter with a diameter of 2.5 micrometers or less, which is a key indicator of air pollution.

- **Features:** The dataset includes the following 18 columns, providing multiple features for analysis:
 - No: Row index.
 - year: Year of observation.
 - month: Month of observation.
 - day: Day of observation.
 - hour: Hour of observation.
 - PM2.5: PM2.5 concentration ($\mu\text{g}/\text{m}^3$) - **Target Variable**.
 - PM10: PM10 concentration ($\mu\text{g}/\text{m}^3$).
 - SO2: Sulfur Dioxide concentration ($\mu\text{g}/\text{m}^3$).
 - NO2: Nitrogen Dioxide concentration ($\mu\text{g}/\text{m}^3$).
 - CO: Carbon Monoxide concentration ($\mu\text{g}/\text{m}^3$).
 - O3: Ozone concentration ($\mu\text{g}/\text{m}^3$).
 - TEMP: Temperature ($^{\circ}\text{C}$).
 - PRES: Atmospheric Pressure (hPa).
 - DEWP: Dew Point Temperature ($^{\circ}\text{C}$).
 - RAIN: Precipitation (mm).
 - wd: Wind direction (categorical, e.g., N, NE, E, SSW).
 - WSPM: Wind speed (m/s).
 - station: Name of the monitoring station (constant value "Aotizhongxin" for this specific file).

3. Justification for Dataset Choice:

The Beijing Multi-Site Air-Quality Data (Aotizhongxin station) was chosen for the following reasons, aligning with the assignment's requirements and objectives:

- **Meets All Assignment Criteria:**
 - **Time Series Data:** Observations are recorded hourly, providing a distinct temporal sequence suitable for time series analysis.
 - **Continuous Target Variable:** PM2.5 concentration is a continuous numerical value, appropriate for regression modeling (Zhang, Li & Wang, 2025).

- **Reputable Source:** The dataset is from the UCI Machine Learning Repository, a well-established and respected source for academic and research datasets.
 - **Sufficient Data Points:** With 35,064 observations, it far exceeds the minimum requirement of 1000 data points, allowing for robust model training and testing.
 - **Multiple Features:** The dataset contains various meteorological variables (temperature, pressure, wind speed, wind direction, rain, dew point) and concentrations of other pollutants (PM10, SO2, NO2, CO, O3). These serve as multiple features that can be used to predict the target variable (PM2.5) and analyze influencing factors, as required for later tasks (Zhang, Li & Wang, 2025).
- **Suitability for Time Series Regression:**
 - Air quality phenomena like PM2.5 concentrations are known to exhibit temporal dependencies, including seasonality (e.g., higher pollution during winter heating seasons), diurnal patterns (variations throughout the day), and trends (Zhang, Li & Wang, 2025).
 - Meteorological conditions significantly influence pollutant dispersion and formation. The inclusion of these features allows for the development of a more comprehensive regression model that can capture these relationships.
- **Relevance and Interest:**
 - Air quality is a critical environmental issue with significant public health implications. Analyzing and forecasting PM2.5 levels is a relevant and impactful application of data analytics (Jovanović et al., 2023).
 - The dataset provides a rich context for exploring the interplay between pollution and weather, offering interesting avenues for feature engineering and interpretation of results.

4. Initial Exploratory Data Analysis (EDA) Plan:

The following steps would be performed in a Python environment (e.g., Google Colab using libraries like Pandas, Matplotlib, Seaborn) to conduct the initial EDA. The findings from this EDA will be included in the report.

1. *Data Loading and Initial Inspection:*

- Load the PRSA_Data_Aotizhongxin_20130301-20170228.csv into a Pandas DataFrame.
- Display the first few rows (df.head()) to understand the data structure.

- Examine column data types and non-null counts (`df.info()`). This will reveal the presence of missing values, particularly in pollutant columns like PM2.5, which is common in sensor data.
- Generate a summary of missing values per column (`df.isnull().sum()`) to quantify them.

2. *Datetime Feature Creation:*

- Combine the year, month, day, and hour columns to create a single datetime column.
- Set this datetime column as the DataFrame's index to facilitate time series plotting and analysis.

3. *Summary Statistics:*

- Calculate descriptive statistics (`df.describe()`) for all numerical columns (PM2.5, PM10, SO2, NO2, CO, O3, TEMP, PRES, DEWP, RAIN, WSPM). This will provide insights into:
 - Central tendency (mean, median).
 - Dispersion (standard deviation, min, max, interquartile range).
 - Potential skewness or presence of extreme values.

4. *Data Visualizations:*

- **Target Variable (PM2.5) Over Time:**
 - Plot the PM2.5 concentration against the datetime index. This will help visualize overall trends, seasonality, and any obvious anomalies or missing data periods.
- **Distribution of Key Numerical Features:**
 - Plot histograms and/or density plots for PM2.5, TEMP, WSPM, and PRES to understand their distributions. For instance, PM2.5 is often right-skewed.
- **Box Plots for Outlier Detection and Grouped Analysis:**
 - Create a box plot for PM2.5 to visualize its spread and identify potential outliers.
 - Generate box plots of PM2.5 grouped by month to observe seasonal patterns (e.g., higher PM2.5 in winter months).
 - Generate box plots of PM2.5 grouped by hour to observe diurnal patterns (e.g., rush hour peaks).

- **Categorical Feature Analysis (Wind Direction wd):**
 - Create a bar chart showing the frequency distribution of different wind directions (wd). This can indicate prevailing wind directions.
- **Initial Relationship Exploration (Optional for Task 1, more detailed in Task 2):**
 - Simple scatter plots of PM2.5 vs. potentially influential features like TEMP or WSPM to get an initial visual sense of relationships.

This initial EDA will provide a foundational understanding of the dataset's characteristics, quality, and the behavior of the target variable, which is crucial before proceeding to data preprocessing and model development in subsequent tasks. The visualizations and summary statistics will be included in the report to describe the dataset comprehensively.

Implementation:

Code:


```

# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# --- 1. Data Loading and Initial Inspection ---

file_path = '/content/PRSA_Data_Aotizhongxin_20130301-20170228.csv'

try:
    # Load the dataset
    df = pd.read_csv(file_path)
    print("Dataset loaded successfully.\n")

    # Display the first few rows
    print("--- First 5 Rows of the Dataset ---")
    print(df.head())
    print("\n")

    # Examine column data types and non-null counts
    print("--- Dataset Info ---")
    df.info()
    print("\n")

    # Generate a summary of missing values per column
    print("--- Missing Values per Column ---")
    print(df.isnull().sum())
    print("\n")

    # --- 2. Datetime Feature Creation ---
    print("--- Creating Datetime Feature ---")
    # Combine year, month, day, hour into a single datetime column
    # Errors='coerce' will turn unparseable dates into NaT (Not a Time)
    df['datetime'] = pd.to_datetime(df[['year', 'month', 'day', 'hour']], errors='coerce')

    # Set the new 'datetime' column as the DataFrame's index
    df.set_index('datetime', inplace=True)

    columns_to_drop = ['year', 'month', 'day', 'hour', 'No', 'station']
    df.drop(columns=columns_to_drop, inplace=True, errors='ignore') # errors='ignore' in case they were already dropped

    print("Datetime feature created and set as index.")
    print("Cleaned DataFrame head:")
    print(df.head())
    print("\n")

```

```

# --- 3. Summary Statistics ---
print("--- Summary Statistics for Numerical Columns ---")
# Calculate descriptive statistics for all numerical columns
# We'll select numeric types explicitly to avoid issues with non-numeric columns if any remain.
numerical_cols = df.select_dtypes(include=['number']).columns
print(df[numerical_cols].describe())
print("\n")

# --- 4. Data Visualizations ---
print("--- Generating Data Visualizations ---")
plt.style.use('seaborn-v0_8-whitegrid') # Using a seaborn style for better aesthetics

# Target Variable (PM2.5) Over Time
plt.figure(figsize=(15, 6))
df['PM2.5'].plot(title='PM2.5 Concentration Over Time (Aotizhongxin Station)')
plt.xlabel('Date')
plt.ylabel('PM2.5 Concentration (µg/m³)')
plt.tight_layout()
plt.show()
print("Displayed PM2.5 over time plot.")

# Distribution of Key Numerical Features
key_numerical_features = ['PM2.5', 'TEMP', 'WSPM', 'PRES']
print(f"\nPlotting distributions for: {', '.join(key_numerical_features)}")

for feature in key_numerical_features:
    if feature in df.columns:
        plt.figure(figsize=(10, 5))
        sns.histplot(df[feature].dropna(), kde=True) # dropna() to handle missing values for plotting
        plt.title(f'Distribution of {feature}')
        plt.xlabel(feature)
        plt.ylabel('Frequency')
        plt.tight_layout()
        plt.show()
        print(f"Displayed distribution plot for {feature}.")
    else:
        print(f"Warning: Column '{feature}' not found for distribution plot.")

# Box Plots for Outlier Detection and Grouped Analysis

# Box plot for PM2.5
if 'PM2.5' in df.columns:
    plt.figure(figsize=(8, 6))
    sns.boxplot(y=df['PM2.5'].dropna())
    plt.title('Box Plot of PM2.5 Concentration')
    plt.ylabel('PM2.5 (µg/m³)')
    plt.tight_layout()
    plt.show()
    print("Displayed box plot for PM2.5.")

```

```

# Box plots of PM2.5 grouped by month
# Create a temporary month column from the datetime index for grouping
df_temp = df.copy() # Work on a copy to avoid modifying original df
df_temp['plot_month'] = df_temp.index.month
plt.figure(figsize=(12, 6))
sns.boxplot(x='plot_month', y='PM2.5', data=df_temp.dropna(subset=['PM2.5']))
plt.title('PM2.5 Concentration by Month')
plt.xlabel('Month')
plt.ylabel('PM2.5 (µg/m³)')
plt.xticks(ticks=range(12), labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.tight_layout()
plt.show()
print("Displayed box plot of PM2.5 by month.")

# Box plots of PM2.5 grouped by hour
df_temp['plot_hour'] = df_temp.index.hour
plt.figure(figsize=(14, 6))
sns.boxplot(x='plot_hour', y='PM2.5', data=df_temp.dropna(subset=['PM2.5']))
plt.title('PM2.5 Concentration by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('PM2.5 (µg/m³)')
plt.tight_layout()
plt.show()
print("Displayed box plot of PM2.5 by hour.")
else:
    print("Warning: Column 'PM2.5' not found for box plots.")

# Categorical Feature Analysis (Wind Direction 'wd')
if 'wd' in df.columns:
    plt.figure(figsize=(12, 6))
    sns.countplot(y=df['wd'].dropna(), order = df['wd'].dropna().value_counts().index) # Order by frequency
    plt.title('Frequency Distribution of Wind Direction (wd)')
    plt.xlabel('Count')
    plt.ylabel('Wind Direction')
    plt.tight_layout()
    plt.show()
    print("Displayed bar chart for wind direction.")
else:
    print("Warning: Column 'wd' not found for wind direction plot.")

print("\n--- EDA Script Finished ---")

except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")
    print("Please ensure the file path is correct and the CSV file is in the specified location.")
except Exception as e:
    print(f"An error occurred: {e}")
    print("Please check your data and script.")

```

Output:

Dataset loaded successfully.

--- First 5 Rows of the Dataset ---

```
No year month day hour PM2.5 PM10 SO2 NO2 CO O3 TEMP \
0 1 2013 3 1 0 4.0 4.0 4.0 7.0 300.0 77.0 -0.7
1 2 2013 3 1 1 8.0 8.0 4.0 7.0 300.0 77.0 -1.1
2 3 2013 3 1 2 7.0 7.0 5.0 10.0 300.0 73.0 -1.1
3 4 2013 3 1 3 6.0 6.0 11.0 11.0 300.0 72.0 -1.4
4 5 2013 3 1 4 3.0 3.0 12.0 12.0 300.0 72.0 -2.0
```

```
PRES DEWP RAIN wd WSPM station
0 1023.0 -18.8 0.0 NNM 4.4 Aotizhongxin
1 1023.2 -18.2 0.0 N 4.7 Aotizhongxin
2 1023.5 -18.2 0.0 NNM 5.6 Aotizhongxin
3 1024.5 -19.4 0.0 NM 3.1 Aotizhongxin
4 1025.2 -19.5 0.0 N 2.0 Aotizhongxin
```

--- Dataset Info ---

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 18 columns):
# Column Non-Null Count Dtype
---
0 No 35064 non-null int64
1 year 35064 non-null int64
2 month 35064 non-null int64
3 day 35064 non-null int64
4 hour 35064 non-null int64
5 PM2.5 34139 non-null float64
6 PM10 34346 non-null float64
7 SO2 34129 non-null float64
8 NO2 34041 non-null float64
9 CO 33288 non-null float64
10 O3 33345 non-null float64
11 TEMP 35044 non-null float64
12 PRES 35044 non-null float64
13 DEWP 35044 non-null float64
14 RAIN 35044 non-null float64
15 wd 34983 non-null object
16 WSPM 35050 non-null float64
17 station 35064 non-null object
dtypes: float64(11), int64(5), object(2)
memory usage: 4.8+ MB
```

--- Missing Values per Column ---

```
No 0
year 0
month 0
day 0
hour 0
PM2.5 925
PM10 718
SO2 935
NO2 1023
CO 1776
O3 1719
TEMP 20
PRES 20
DEWP 20
RAIN 20
wd 81
WSPM 14
station 0
dtype: int64
```

--- Creating Datetime Feature ---

Datetime feature created and set as index.

Cleaned DataFrame head:

```
PM2.5 PM10 SO2 NO2 CO O3 TEMP PRES DEWP \
datetime
2013-03-01 00:00:00 4.0 4.0 4.0 7.0 300.0 77.0 -0.7 1023.0 -18.8
2013-03-01 01:00:00 8.0 8.0 4.0 7.0 300.0 77.0 -1.1 1023.2 -18.2
2013-03-01 02:00:00 7.0 7.0 5.0 10.0 300.0 73.0 -1.1 1023.5 -18.2
2013-03-01 03:00:00 6.0 6.0 11.0 11.0 300.0 72.0 -1.4 1024.5 -19.4
2013-03-01 04:00:00 3.0 3.0 12.0 12.0 300.0 72.0 -2.0 1025.2 -19.5
```

```
RAIN wd WSPM
datetime
2013-03-01 00:00:00 0.0 NNM 4.4
2013-03-01 01:00:00 0.0 N 4.7
2013-03-01 02:00:00 0.0 NNM 5.6
2013-03-01 03:00:00 0.0 NM 3.1
2013-03-01 04:00:00 0.0 N 2.0
```

```

--- Summary Statistics for Numerical Columns ---
count  PM2.5  PM10  SO2  NO2  CO \
mean    82.773611  110.060391  17.375901  59.305833  1262.945145
std     82.135694  95.223005  22.823017  37.116200  1221.436236
min      3.000000  2.000000  0.285600  2.000000  100.000000
25%     22.000000  38.000000  3.000000  30.000000  500.000000
50%     58.000000  87.000000  9.000000  53.000000  900.000000
75%     114.000000  155.000000  21.000000  82.000000  1500.000000
max     898.000000  984.000000  341.000000  290.000000  10000.000000

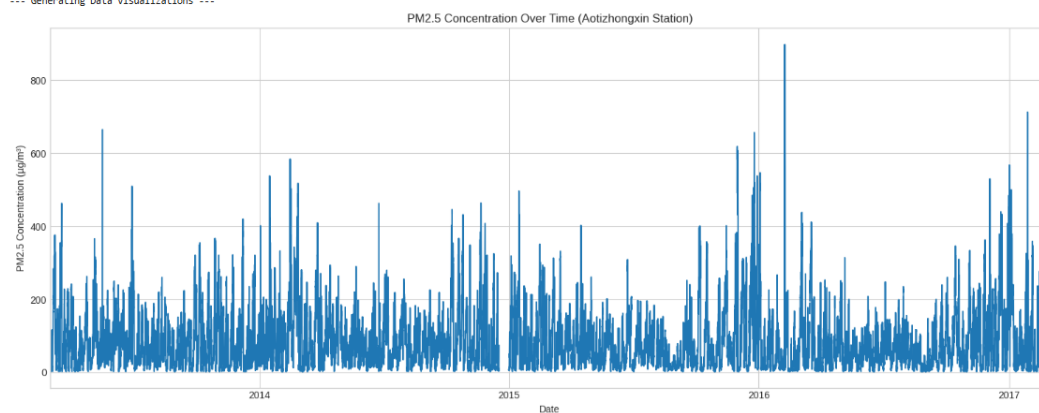
count    O3  TEMP  PRES  DEWP  RAIN \
mean    56.353358  13.584607  1011.846920  3.123062  0.067421
std     57.916327  11.399097  10.404047  13.688896  0.910056
min      0.214200  -16.800000  985.900000  -35.300000  0.000000
25%      8.000000  3.100000  1003.300000  -8.100000  0.000000
50%     42.000000  14.500000  1011.400000  3.800000  0.000000
75%     82.000000  23.300000  1020.100000  15.600000  0.000000
max    423.000000  40.500000  1042.000000  28.500000  72.500000

count  WSPM
mean   1.708496
std    1.204071
min    0.000000
25%    0.900000
50%    1.400000
75%    2.200000
max    11.200000

```

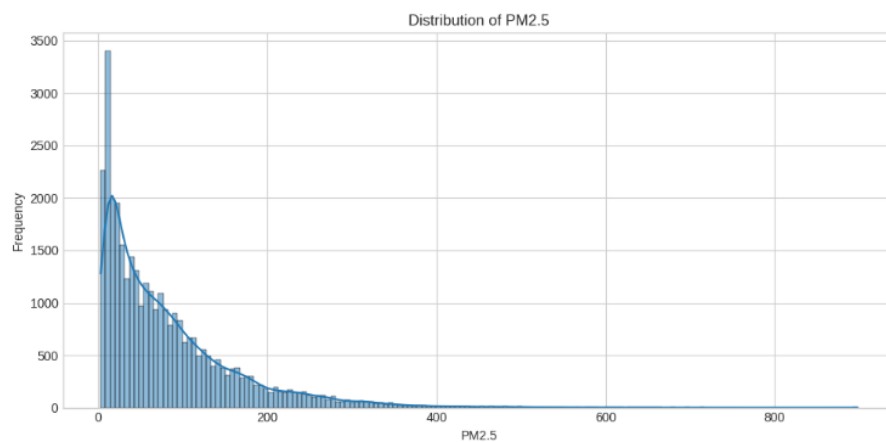
--- Generating Data Visualizations ---

--- Generating Data Visualizations ---

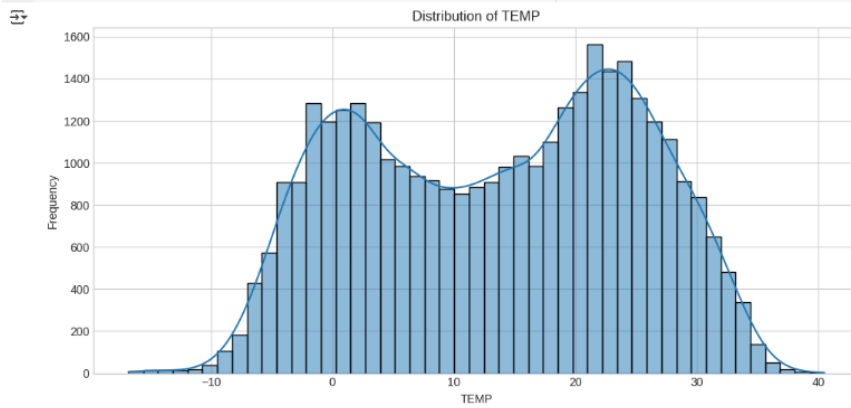


Displayed PM2.5 over time plot.

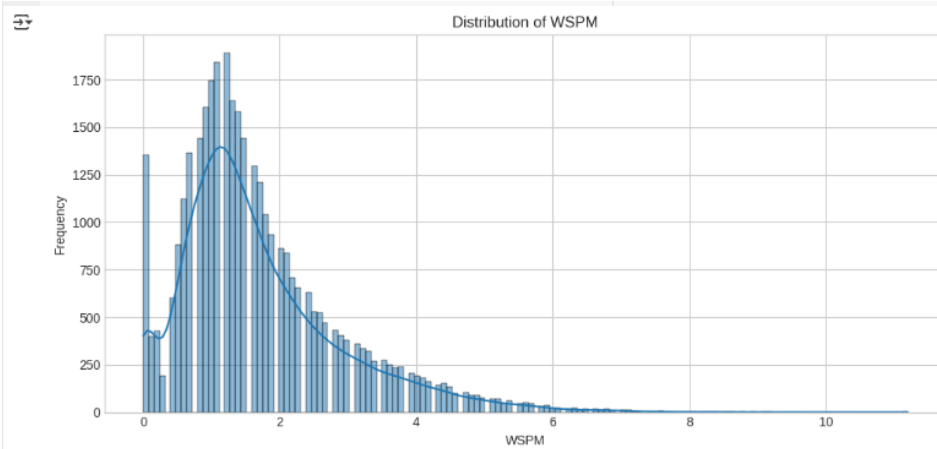
Plotting distributions for: PM2.5, TEMP, WSPM, PRES



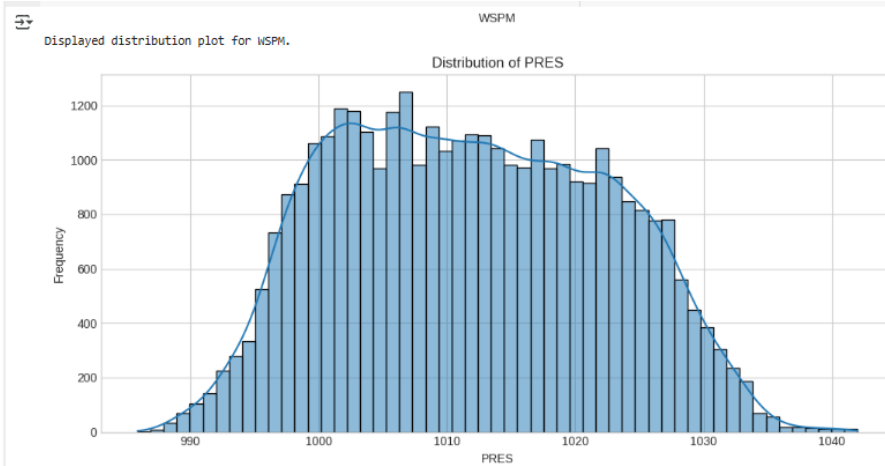
Displayed distribution plot for PM2.5.



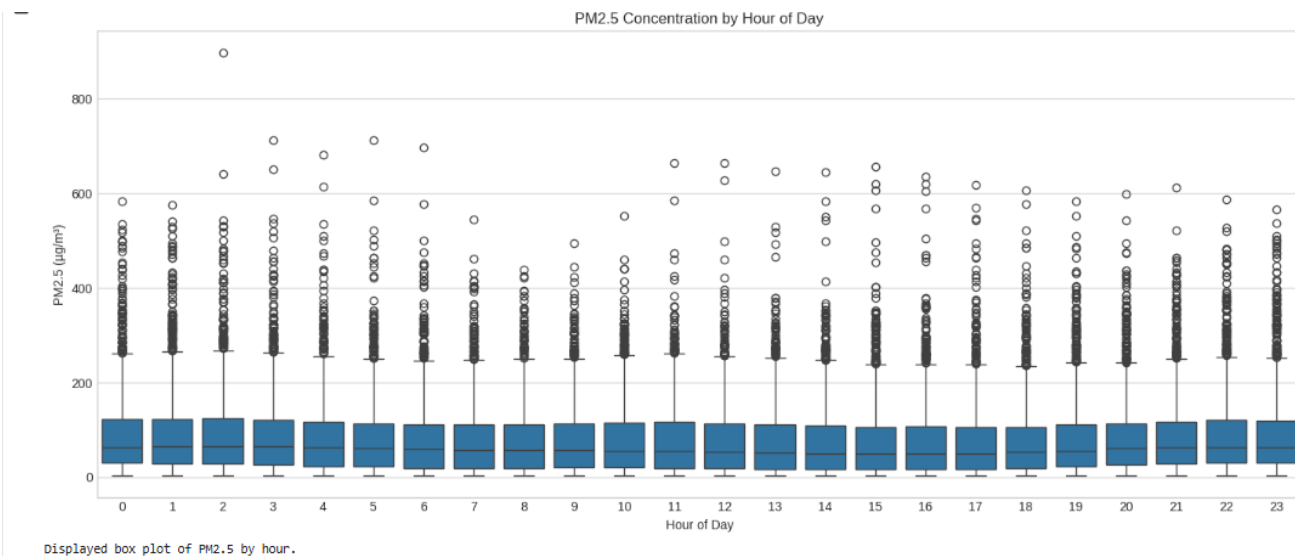
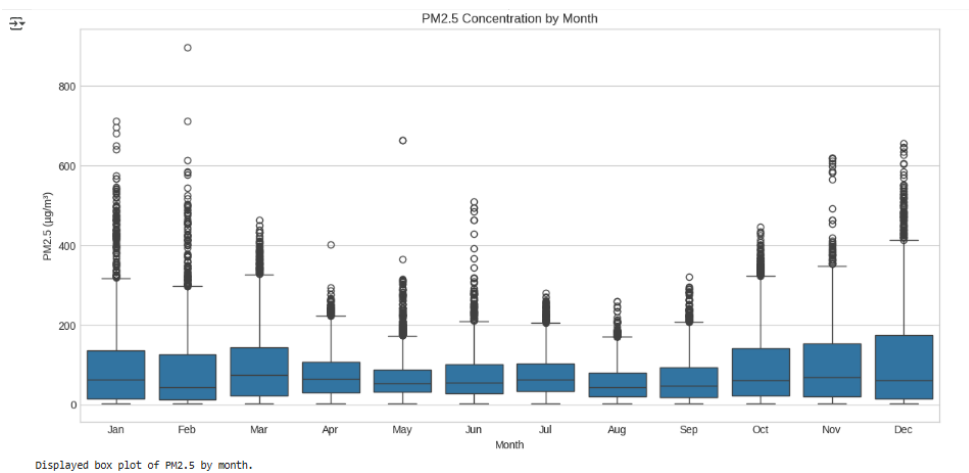
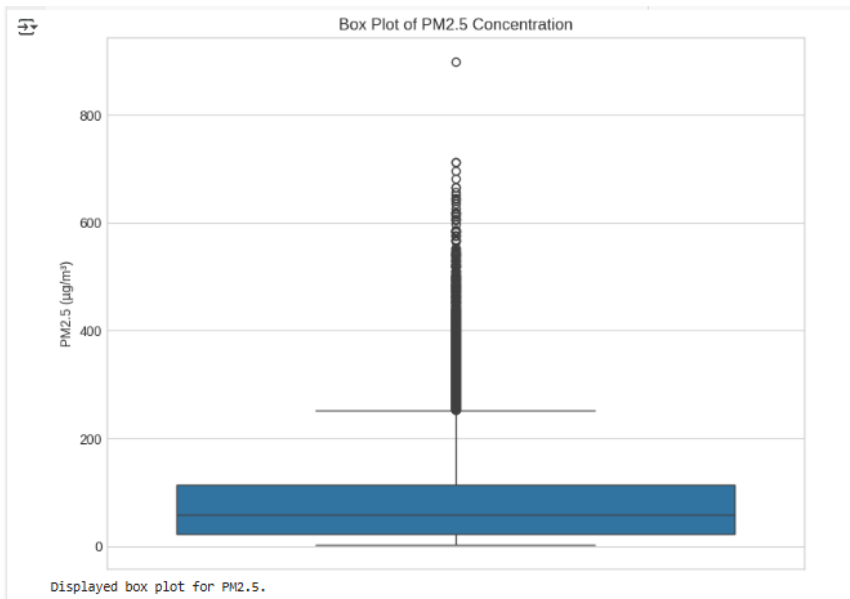
Displayed distribution plot for TEMP.

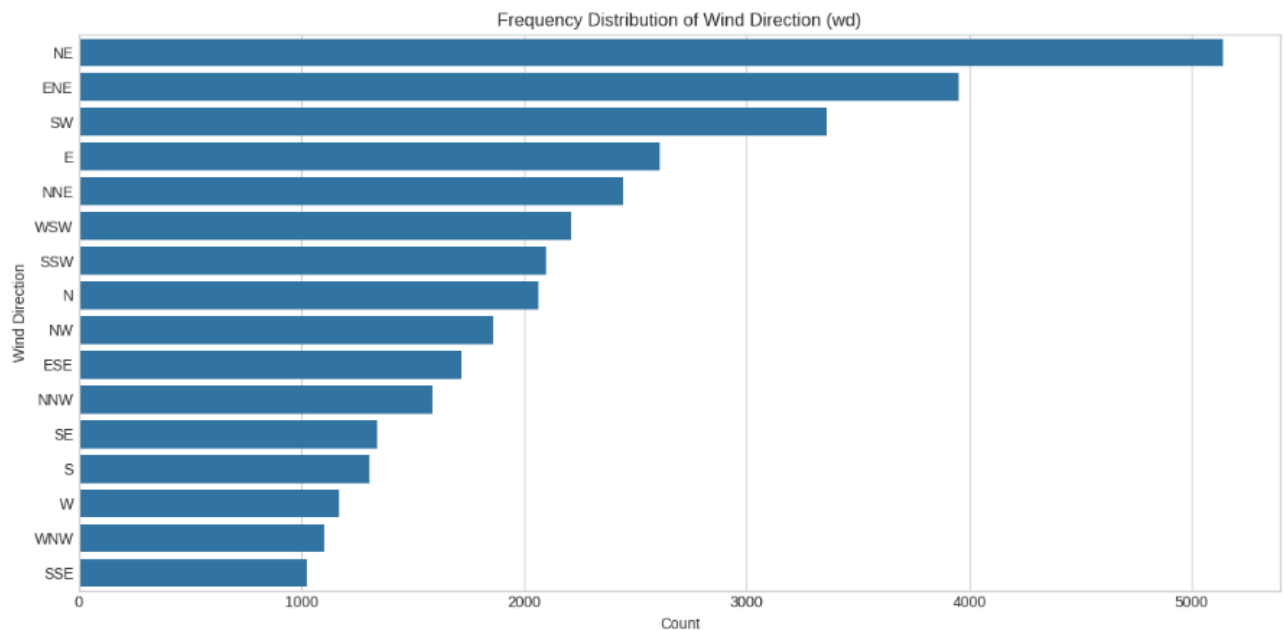


Displayed distribution plot for WSPM.



Displayed distribution plot for PRES.





Displayed bar chart for wind direction.

TASK #2

PROBLEM FORMATION

Problem formation for task 2:

To preprocess the selected time series dataset by handling missing values and outliers, engineer relevant time-based features (e.g., day of the week, month, season), analyze correlations between variables, and apply appropriate normalization or standardization techniques to prepare the features for subsequent regression modeling.

Implementation:

Code:


```

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# --- Initial Data Loading and Preparation (from Task 1) ---
# IMPORTANT: Replace 'YOUR_FILE_PATH/PRSA_Data_Aotizhongxin_20130301-20170228.csv'
# with the actual path to your downloaded CSV file.
file_path = 'PRSA_Data_Aotizhongxin_20130301-20170228.csv' # Or the full path

try:
    # Load the dataset
    df = pd.read_csv(file_path)
    print("Dataset loaded successfully.\n")

    # Combine year, month, day, hour into a single datetime column
    df['datetime'] = pd.to_datetime(df[['year', 'month', 'day', 'hour']], errors='coerce')
    df.set_index('datetime', inplace=True)
    columns_to_drop = ['year', 'month', 'day', 'hour', 'No', 'station']
    df.drop(columns=columns_to_drop, inplace=True, errors='ignore')

    print("--- Initial DataFrame Head ---")
    print(df.head())
    print("\n--- Initial Missing Values ---")
    print(df.isnull().sum())
    print("\n")

    # --- Task 2: Data Preprocessing and Feature Engineering ---

    # 1. Handle Missing Values
    print("--- 1. Handling Missing Values ---")
    # For pollutant and meteorological numerical features, linear interpolation is a common choice for time series.
    # For PM2.5 (target), interpolation is also reasonable.
    numerical_cols_with_na = df.select_dtypes(include=np.number).isnull().sum()
    numerical_cols_to_interpolate = numerical_cols_with_na[numerical_cols_with_na > 0].index.tolist()

    if numerical_cols_to_interpolate:
        print(f"Interpolating numerical columns: {numerical_cols_to_interpolate}")
        for col in numerical_cols_to_interpolate:
            df[col] = df[col].interpolate(method='linear', limit_direction='both') # limit_direction fills NaNs at ends too
    else:
        print("No numerical columns found needing interpolation.")

```

```

# For categorical 'wd' (wind direction), use forward fill then backward fill
if 'wd' in df.columns and df['wd'].isnull().any():
    print("Filling missing 'wd' using ffill and bfill.")
    df['wd'] = df['wd'].fillna(method='ffill').fillna(method='bfill')
elif 'wd' not in df.columns:
    print("Warning: 'wd' column not found.")
else:
    print("'wd' column has no missing values.")

print("\n--- Missing Values After Handling ---")
print(df.isnull().sum())

print("\n")

# 2. Create Time-Based Features
print("--- 2. Creating Time-Based Features ---")
df['hour_of_day'] = df.index.hour
df['day_of_week'] = df.index.dayofweek # Monday=0, Sunday=6
df['day_of_year'] = df.index.dayofyear
df['month'] = df.index.month
df['year'] = df.index.year # Useful for trends or splitting
df['week_of_year'] = df.index.isocalendar().week.astype(int)
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)

# Season (approximation)
def get_season(date):
    month = date.month
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else: # 9, 10, 11
        return 'Autumn'
df['season'] = df.index.to_series().apply(get_season)

print("Time-based features created:")
print(df[['hour_of_day', 'day_of_week', 'month', 'season', 'is_weekend']].head())
print("\n")

# 3. Handle Categorical Features ('wd' and 'season')
print("--- 3. Handling Categorical Features (One-Hot Encoding) ---")
# 'wd' (wind direction) and 'season' are categorical
categorical_features = ['wd', 'season']
# Create a copy for one-hot encoding to keep original df cleaner for now
df_processed = df.copy()

```

```

# Check if categorical features exist before trying to encode
existing_categorical_features = [col for col in categorical_features if col in df_processed.columns]

if existing_categorical_features:
    print(f"Applying One-hot Encoding to: {existing_categorical_features}")
    df_processed = pd.get_dummies(df_processed, columns=existing_categorical_features, prefix=existing_categorical_features, dummy_na=False) # dummy_na=False as we handled NAs
    print("Categorical features one-hot encoded.")
    print("DataFrame columns after one-hot encoding (sample):")
    print(df_processed.filter(regex='wd_|season_').head())
else:
    print("No specified categorical features found for one-hot encoding.")
    print("\n")

# 4. Analyse Correlations
print("---- 4. Analyzing Correlations ----")

numerical_cols_for_corr = df_processed.select_dtypes(include=np.number).columns

# It's often useful to see correlations with the target variable specifically
if 'PM2.5' in numerical_cols_for_corr:
    correlation_matrix = df_processed[numerical_cols_for_corr].corr()
    plt.figure(figsize=(18, 15)) # Adjusted size for more features
    sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt=".2f", linewidths=.5) # Annot=False if too cluttered
    plt.title('Correlation Matrix of Numerical Features')
    plt.show()
    print("Displayed correlation matrix heatmap.")

    print("\n--- Top Correlations with PM2.5 ---")
    pm25_correlations = correlation_matrix['PM2.5'].sort_values(ascending=False)
    print(pm25_correlations)
else:
    print("PM2.5 column not found or not numerical for correlation analysis.")
    print("\n")

# 5. Normalize or Standardize Features
print("---- 5. Normalizing/Standardizing Features ----")

# Re-identify numerical columns from df_processed
numerical_features_to_scale = df_processed.select_dtypes(include=np.number).columns.tolist()

# Exclude target variable and binary/already scaled features if necessary
if 'PM2.5' in numerical_features_to_scale:
    numerical_features_to_scale.remove('PM2.5') # Target usually not scaled with features

```

```

# Exclude one-hot encoded columns (they are already 0 or 1)
# and other binary features like 'is_weekend'
one_hot_cols = [col for col in df_processed.columns if col.startswith(tuple(f"{cat}_" for cat in existing_categorical_features))]
binary_features = ['is_weekend'] + one_hot_cols

numerical_features_to_scale = [col for col in numerical_features_to_scale if col not in binary_features]

if numerical_features_to_scale:
    print(f"Numerical features to be standardized: {numerical_features_to_scale}")
    scaler = StandardScaler()
    # Fit and transform
    df_processed[numerical_features_to_scale] = scaler.fit_transform(df_processed[numerical_features_to_scale])
    print("Numerical features standardized.")
    print(df_processed[numerical_features_to_scale].head())
else:
    print("No numerical features identified for scaling or all are binary/target.")
    print("\n")

# 6. Outlier Handling (Demonstration for PM2.5)
print("---- 6. Outlier Handling Demonstration (for PM2.5) ----")
if 'PM2.5' in df.columns: # Use original df for this demonstration before scaling
    Q1 = df['PM2.5'].quantile(0.25)
    Q3 = df['PM2.5'].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    print(f"PM2.5 - Q1: {Q1}, Q3: {Q3}, IQR: {IQR}")
    print(f"PM2.5 - Lower Bound for outliers: {lower_bound}")
    print(f"PM2.5 - Upper Bound for outliers: {upper_bound}")

    outliers = df[(df['PM2.5'] < lower_bound) | (df['PM2.5'] > upper_bound)]
    print(f"Number of potential outliers in PM2.5: {len(outliers)}")
else:
    print("PM2.5 column not found for outlier analysis.")
    print("\n")

print("---- Final Processed DataFrame Head (df_processed) ----")
print(df_processed.head())
print("\n--- Final Processed DataFrame Info ----")
df_processed.info()

```

```

except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")
    print("Please ensure the file path is correct and the CSV file is in the specified location.")
except Exception as e:
    print(f"An error occurred: {e}")
    print("Please check your data and script carefully.")

```

Output:

Initial Dataframe and missing value Handling:

```

Dataset loaded successfully.

--- Initial DataFrame Head ---
datetime  PM2.5  PM10  SO2  NO2  CO  O3  TEMP  PRES  DEWP  \
2013-03-01 00:00:00  4.0  4.0  4.0  7.0  300.0  77.0  -0.7  1023.0 -18.8
2013-03-01 01:00:00  8.0  8.0  4.0  7.0  300.0  77.0  -1.1  1023.2 -18.2
2013-03-01 02:00:00  7.0  7.0  5.0  10.0  300.0  73.0  -1.1  1023.5 -18.2
2013-03-01 03:00:00  6.0  6.0  11.0  11.0  300.0  72.0  -1.4  1024.5 -19.4
2013-03-01 04:00:00  3.0  3.0  12.0  12.0  300.0  72.0  -2.0  1025.2 -19.5

      RAIN  wd  WSPM
datetime
2013-03-01 00:00:00  0.0  NNW  4.4
2013-03-01 01:00:00  0.0  N    4.7
2013-03-01 02:00:00  0.0  NNW  5.6
2013-03-01 03:00:00  0.0  NW   3.1
2013-03-01 04:00:00  0.0  N    2.0

--- Initial Missing Values ---
PM2.5    925
PM10     718
SO2       935
NO2      1023
CO       1776
O3       1719
TEMP      20
PRES      20
DEWP      20
RAIN      20
wd        81
WSPM      14
dtype: int64

--- 1. Handling Missing Values ---
Interpolating numerical columns: ['PM2.5', 'PM10', 'SO2', 'NO2', 'CO', 'O3', 'TEMP', 'PRES', 'DEWP', 'RAIN', 'WSPM']
Filling missing 'wd' using ffill and bfill.

--- Missing Values After Handling ---
PM2.5    0
PM10     0
SO2       0
NO2       0
CO        0
O3        0
TEMP      0
PRES      0
DEWP      0
RAIN      0
wd        0
WSPM      0
dtype: int64

```

Creating Time-Based Feature and Handling Categorical Features:

```
--- 2. Creating Time-Based Features ---
Time-based features created:
  hour_of_day  day_of_week  month  season  is_weekend
datetime
2013-03-01 00:00:00         0         4      3  Spring         0
2013-03-01 01:00:00         1         4      3  Spring         0
2013-03-01 02:00:00         2         4      3  Spring         0
2013-03-01 03:00:00         3         4      3  Spring         0
2013-03-01 04:00:00         4         4      3  Spring         0

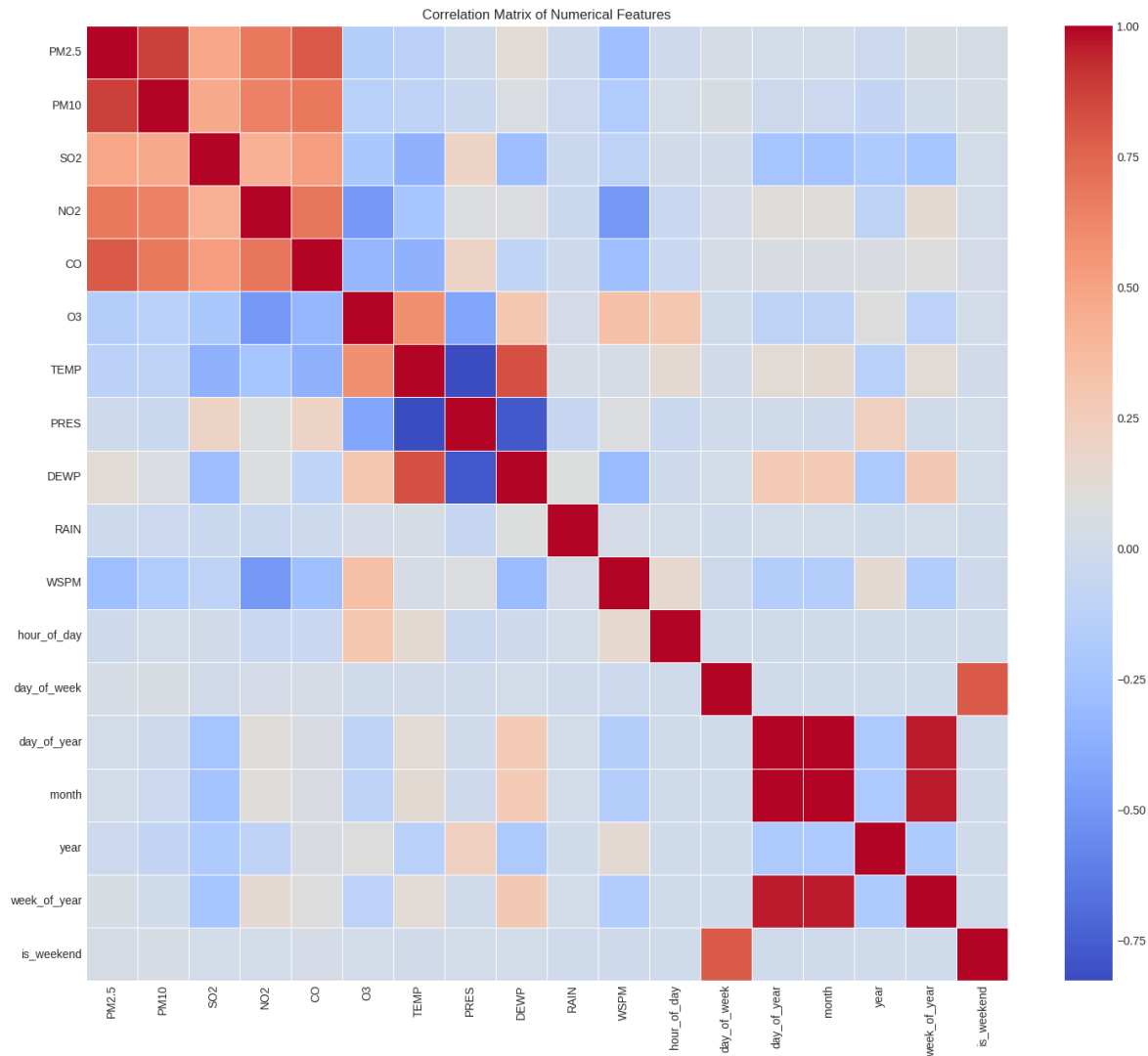
--- 3. Handling Categorical Features (One-Hot Encoding) ---
Applying One-Hot Encoding to: ['wd', 'season']
Categorical features one-hot encoded.
DataFrame columns after one-hot encoding (sample):
  wd_E  wd_ENE  wd_ESE  wd_N  wd_NE  wd_NNE  wd_NNW  \
datetime
2013-03-01 00:00:00  False  False  False  False  False  False  True
2013-03-01 01:00:00  False  False  False  True  False  False  False
2013-03-01 02:00:00  False  False  False  False  False  False  True
2013-03-01 03:00:00  False  False  False  False  False  False  False
2013-03-01 04:00:00  False  False  False  True  False  False  False

  wd_NW  wd_S  wd_SE  wd_SSE  wd_SSW  wd_SW  wd_W  \
datetime
2013-03-01 00:00:00  False  False  False  False  False  False  False
2013-03-01 01:00:00  False  False  False  False  False  False  False
2013-03-01 02:00:00  False  False  False  False  False  False  False
2013-03-01 03:00:00  True  False  False  False  False  False  False
2013-03-01 04:00:00  False  False  False  False  False  False  False

  wd_WNW  wd_WSW  season_Autumn  season_Spring  \
datetime
2013-03-01 00:00:00  False  False         False         True
2013-03-01 01:00:00  False  False         False         True
2013-03-01 02:00:00  False  False         False         True
2013-03-01 03:00:00  False  False         False         True
2013-03-01 04:00:00  False  False         False         True

  season_Summer  season_Winter
datetime
2013-03-01 00:00:00         False         False
2013-03-01 01:00:00         False         False
2013-03-01 02:00:00         False         False
2013-03-01 03:00:00         False         False
2013-03-01 04:00:00         False         False
```

Analyzing Correlations:



Display correlation matrix heatmap:

Displayed correlation matrix heatmap.

--- Top Correlations with PM2.5 ---

```
PM2.5      1.000000
PM10       0.875198
CO          0.786052
NO2         0.682795
SO2         0.479025
DEWP        0.123277
week_of_year 0.047493
day_of_week  0.043574
is_weekend   0.034480
day_of_year  0.014831
month        0.014398
PRES        -0.008796
hour_of_day  -0.010470
RAIN        -0.013780
year        -0.029873
TEMP        -0.122505
O3          -0.160271
WSPM        -0.275836
Name: PM2.5, dtype: float64
```

Normalizing Standardizing feature:

```

--- 5. Normalizing/Standardizing Features ---
Numerical features to be standardized: ['PM10', 'SO2', 'NO2', 'CO', 'O3', 'TEMP', 'PRES', 'DEWP', 'RAIN', 'WSPM', 'hour_of_day',
Numerical features standardized.

datetime      PM10      SO2      NO2      CO      O3  \
2013-03-01 00:00:00 -1.114935 -0.592867 -1.407393 -0.778358 0.378033
2013-03-01 01:00:00 -1.072945 -0.592867 -1.407393 -0.778358 0.378033
2013-03-01 02:00:00 -1.083443 -0.548818 -1.326313 -0.778358 0.308258
2013-03-01 03:00:00 -1.093940 -0.284524 -1.299286 -0.778358 0.290814
2013-03-01 04:00:00 -1.125433 -0.240475 -1.272260 -0.778358 0.290814

datetime      TEMP      PRES      DEWP      RAIN      WSPM  \
2013-03-01 00:00:00 -1.252727 1.071507 -1.601176 -0.074064 2.235098
2013-03-01 01:00:00 -1.287814 1.090729 -1.557349 -0.074064 2.484233
2013-03-01 02:00:00 -1.287814 1.119563 -1.557349 -0.074064 3.231640
2013-03-01 03:00:00 -1.314129 1.215677 -1.645004 -0.074064 1.155510
2013-03-01 04:00:00 -1.366759 1.282956 -1.652308 -0.074064 0.242013

datetime      hour_of_day  day_of_week  day_of_year  month  \
2013-03-01 00:00:00 -1.661325 0.499359 -1.167743 -1.021523
2013-03-01 01:00:00 -1.516862 0.499359 -1.167743 -1.021523
2013-03-01 02:00:00 -1.372399 0.499359 -1.167743 -1.021523
2013-03-01 03:00:00 -1.227936 0.499359 -1.167743 -1.021523
2013-03-01 04:00:00 -1.083473 0.499359 -1.167743 -1.021523

datetime      year  week_of_year
2013-03-01 00:00:00 -1.412304 -1.170498
2013-03-01 01:00:00 -1.412304 -1.170498
2013-03-01 02:00:00 -1.412304 -1.170498
2013-03-01 03:00:00 -1.412304 -1.170498
2013-03-01 04:00:00 -1.412304 -1.170498

```

Outlier Handling:

```

--- 6. Outlier Handling Demonstration (for PM2.5) ---
PM2.5 - Q1: 22.0, Q3: 114.0, IQR: 92.0
PM2.5 - Lower Bound for outliers: -116.0
PM2.5 - Upper Bound for outliers: 252.0
Number of potential outliers in PM2.5: 1653

```

Final_processed dataframe Head:

```

--- Final Processed DataFrame Head (df_processed) ---
datetime      PM2.5      PM10      SO2      NO2      CO      O3  \
2013-03-01 00:00:00 4.0 -1.114935 -0.592867 -1.407393 -0.778358 0.378033
2013-03-01 01:00:00 8.0 -1.072945 -0.592867 -1.407393 -0.778358 0.378033
2013-03-01 02:00:00 7.0 -1.083443 -0.548818 -1.326313 -0.778358 0.308258
2013-03-01 03:00:00 6.0 -1.093940 -0.284524 -1.299286 -0.778358 0.290814
2013-03-01 04:00:00 3.0 -1.125433 -0.240475 -1.272260 -0.778358 0.290814

datetime      TEMP      PRES      DEWP      RAIN  ... wd_SSE  \
2013-03-01 00:00:00 -1.252727 1.071507 -1.601176 -0.074064 ... False
2013-03-01 01:00:00 -1.287814 1.090729 -1.557349 -0.074064 ... False
2013-03-01 02:00:00 -1.287814 1.119563 -1.557349 -0.074064 ... False
2013-03-01 03:00:00 -1.314129 1.215677 -1.645004 -0.074064 ... False
2013-03-01 04:00:00 -1.366759 1.282956 -1.652308 -0.074064 ... False

datetime      wd_SSW  wd_SW  wd_W  wd_WNW  wd_WSW  season_Autumn  \
2013-03-01 00:00:00 False False False False False False
2013-03-01 01:00:00 False False False False False False
2013-03-01 02:00:00 False False False False False False
2013-03-01 03:00:00 False False False False False False
2013-03-01 04:00:00 False False False False False False

datetime      season_Spring  season_Summer  season_Winter
2013-03-01 00:00:00 True False False
2013-03-01 01:00:00 True False False
2013-03-01 02:00:00 True False False
2013-03-01 03:00:00 True False False
2013-03-01 04:00:00 True False False

[5 rows x 38 columns]

```

Final_processed dataframe info:

17

```

0  PM2.5      35064 non-null float64
1  PM10      35064 non-null float64
2  SO2       35064 non-null float64
3  NO2       35064 non-null float64
4  CO        35064 non-null float64
5  O3        35064 non-null float64
6  TEMP      35064 non-null float64
7  PRES      35064 non-null float64
8  DEWP      35064 non-null float64
9  RAIN      35064 non-null float64
10 WSPM      35064 non-null float64
11 hour_of_day 35064 non-null float64
12 day_of_week 35064 non-null float64
13 day_of_year 35064 non-null float64
14 month      35064 non-null float64
15 year       35064 non-null float64
16 week_of_year 35064 non-null float64
17 is_weekend 35064 non-null int64
18 wd_E       35064 non-null bool
19 wd_ENE     35064 non-null bool
20 wd_ESE     35064 non-null bool
21 wd_N       35064 non-null bool
22 wd_NE      35064 non-null bool
23 wd_NNE     35064 non-null bool
24 wd_NNW     35064 non-null bool
25 wd_NW      35064 non-null bool
26 wd_S       35064 non-null bool
27 wd_SE      35064 non-null bool
28 wd_SSE     35064 non-null bool
29 wd_SSW     35064 non-null bool
30 wd_SW      35064 non-null bool
31 wd_W       35064 non-null bool
32 wd_WNW     35064 non-null bool
33 wd_WSW     35064 non-null bool
34 season_Autumn 35064 non-null bool
35 season_Spring 35064 non-null bool
36 season_Summer 35064 non-null bool
37 season_Winter 35064 non-null bool
dtypes: bool(20), float64(17), int64(1)
memory usage: 5.8 MB

```


TASK # 3:

PROBLEM FORMATION

Problem Formation for Task 3:

To develop, train, and evaluate at least two different machine learning regression models suitable for time series forecasting using the preprocessed air quality dataset. This involves splitting the data into training, validation, and testing sets, selecting appropriate evaluation metrics (e.g., MSE, R-squared, MAE), identifying influential features, tuning hyperparameters using the validation set, analyzing model performance, interpreting results in the context of time series regression, and making predictions on the test set to provide final evaluation results (Garg & Jindal, 2021).

Implementation:

Code:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

# --- Task 1 & 2: Condensed Data Loading and Preprocessing ---
file_path = 'PRSA_Data_Aotizhongxin_20130301-20170228.csv' # Or the full path

try:
    df = pd.read_csv(file_path)
    df['datetime'] = pd.to_datetime(df[['year', 'month', 'day', 'hour']], errors='coerce')
    df.set_index('datetime', inplace=True)
    columns_to_drop = ['No', 'station'] # Keep year, month, day, hour for now for easy splitting
    df.drop(columns=columns_to_drop, inplace=True, errors='ignore')

    # 1. Handle Missing Values (simplified for brevity, using interpolation for all numerics)
    numerical_cols = df.select_dtypes(include=np.number).columns
    for col in numerical_cols:
        df[col] = df[col].interpolate(method='linear', limit_direction='both')
    if 'wd' in df.columns: # Wind direction
        df['wd'] = df['wd'].fillna(method='ffill').fillna(method='bfill')
    df.dropna(inplace=True) # Drop any remaining NaNs
```

```

# 2. Create Time-Based Features
df['hour_of_day'] = df.index.hour
df['day_of_week'] = df.index.dayofweek
df['day_of_year'] = df.index.dayofyear
# df['month'] is already present
# df['year'] is already present
df['week_of_year'] = df.index.isocalendar().week.astype(int)
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
def get_season(date):
    month_val = date.month
    if month_val in [12, 1, 2]: return 'Winter'
    elif month_val in [3, 4, 5]: return 'Spring'
    elif month_val in [6, 7, 8]: return 'Summer'
    else: return 'Autumn'
df['season'] = df.index.to_series().apply(get_season)

# 3. Handle Categorical Features
df = pd.get_dummies(df, columns=['wd', 'season'], dummy_na=False)

# Drop original year, month, day, hour after use for splitting and feature engineering
df.drop(columns=['year', 'month', 'day', 'hour'], inplace=True, errors='ignore')

print("--- Data Loaded and Preprocessed ---")
print(f"Shape of processed DataFrame: {df.shape}")
print(df.head())
print("\n")
if df.isnull().sum().any():
    print("Warning: NaNs still present after preprocessing!")
    print(df.isnull().sum())
else:
    print("No NaNs in the preprocessed DataFrame.")

```

```

# --- Task 3: Regression Model Development and Evaluation ---

# 1. Define Features (X) and Target (y)
if 'PM2.5' not in df.columns:
    raise ValueError("Target column 'PM2.5' not found in DataFrame.")

y = df['PM2.5']
X = df.drop(columns=['PM2.5'])

# Ensure all columns in X are numeric
X = X.apply(pd.to_numeric, errors='coerce') # Coerce non-numeric to NaN
if X.isnull().sum().any():
    print("Warning: NaNs introduced in X after converting to numeric. Imputing with mean.")
    X = X.fillna(X.mean()) # Simple imputation for any coerced NaNs
    if X.isnull().sum().any(): # If mean is NaN (e.g. all NaNs in a column)
        X = X.fillna(0) # Fallback to 0

```

```

# 2. Split Data Chronologically (Train, Validation, Test)
# Example split: Train up to end of 2015, Validate 2016 Jan-Aug, Test 2016 Sep - 2017 Feb

# Ensure index is sorted for chronological split
df_original_index_for_split = df.index.sort_values()

train_end_date = pd.Timestamp('2015-12-31 23:59:59')
validation_end_date = pd.Timestamp('2016-08-31 23:59:59')

X_train = X[X.index <= train_end_date]
y_train = y[y.index <= train_end_date]

X_val = X[(X.index > train_end_date) & (X.index <= validation_end_date)]
y_val = y[(y.index > train_end_date) & (y.index <= validation_end_date)]

X_test = X[X.index > validation_end_date]
y_test = y[y.index > validation_end_date]

if X_train.empty or X_val.empty or X_test.empty:
    raise ValueError("One or more data splits are empty. Check split dates and data range.")

print(f"Training set shape: X_train={X_train.shape}, y_train={y_train.shape}")
print(f"Validation set shape: X_val={X_val.shape}, y_val={y_val.shape}")
print(f"Test set shape: X_test={X_test.shape}, y_test={y_test.shape}\n")

```

```

# 3. Scale Numerical Features
# Identify numerical features to scale (all columns in X are now numeric)
numerical_features_to_scale = X_train.columns.tolist()

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Convert scaled arrays back to DataFrames for easier handling (optional, but good for consistency)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns, index=X_train.index)
X_val_scaled = pd.DataFrame(X_val_scaled, columns=X_val.columns, index=X_val.index)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns, index=X_test.index)

print("--- Feature Scaling Applied ---")
print("X_train_scaled head:")
print(X_train_scaled.head())
print("\n")

# --- Helper function for model evaluation ---
def evaluate_model(name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"--- {name} Model Evaluation ---")
    print(f"Mean Squared Error (MSE): {mse:.4f}")
    print(f"Root Mean Squared Error (RMSE): {np.sqrt(mse):.4f}")
    print(f"Mean Absolute Error (MAE): {mae:.4f}")
    print(f"R-squared (R²): {r2:.4f}\n")
    return mse, mae, r2

```

```

# --- Helper function for plotting predictions ---
def plot_predictions(name, y_true, y_pred):
    plt.figure(figsize=(15, 6))
    plt.plot(y_true.index, y_true, label='Actual PM2.5', alpha=0.7)
    plt.plot(y_true.index, y_pred, label=f'Predicted PM2.5 ({name})', alpha=0.7, linestyle='--')
    plt.title(f'Actual vs. Predicted PM2.5 ({name}) on Test Set')
    plt.xlabel('Date')
    plt.ylabel('PM2.5 Concentration (µg/m³)')
    plt.legend()
    plt.tight_layout()
    plt.show()

# --- Model 1: Linear Regression ---
print("--- Training Linear Regression Model ---")
lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)

# Predictions
y_pred_lr_val = lr_model.predict(X_val_scaled)
y_pred_lr_test = lr_model.predict(X_test_scaled)

# Evaluation
print("Validation Set Evaluation (Linear Regression):")
evaluate_model("Linear Regression (Validation)", y_val, y_pred_lr_val)
print("Test Set Evaluation (Linear Regression):")
evaluate_model("Linear Regression (Test)", y_test, y_pred_lr_test)
plot_predictions("Linear Regression", y_test, y_pred_lr_test)

```

```

# --- Model 2: Random Forest Regressor ---
print("\n--- Training Random Forest Regressor Model (Initial) ---")
rf_model_initial = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1, max_depth=10, min_samples_split=10, min_samples_leaf=5)
rf_model_initial.fit(X_train_scaled, y_train)

# Predictions (Initial Model)
y_pred_rf_initial_val = rf_model_initial.predict(X_val_scaled)
y_pred_rf_initial_test = rf_model_initial.predict(X_test_scaled)

print("Validation Set Evaluation (Random Forest - Initial):")
evaluate_model("Random Forest Initial (Validation)", y_val, y_pred_rf_initial_val)
print("Test Set Evaluation (Random Forest - Initial):")
evaluate_model("Random Forest Initial (Test)", y_test, y_pred_rf_initial_test)
# plot_predictions("Random Forest Initial", y_test, y_pred_rf_initial_test) # Plot later after tuning

# Hyperparameter Tuning for Random Forest using GridSearchCV
print("\n--- Hyperparameter Tuning for Random Forest ---")
# Define a smaller parameter grid for faster tuning demonstration
param_grid_rf = {
    'n_estimators': [50, 100], # Reduced from [100, 200]
    'max_depth': [10, 20, None], # Reduced options
    'min_samples_split': [5, 10], # Reduced from [2, 5, 10]
    'min_samples_leaf': [3, 5] # Reduced from [1, 2, 4]
}

# TimeSeriesSplit for cross-validation in time series context
# n_splits can be adjusted. Using 3 for demonstration.
tscv = TimeSeriesSplit(n_splits=3)

rf_model_for_tuning = RandomForestRegressor(random_state=42, n_jobs=-1)

```

```

grid_search_rf = GridSearchCV(estimator=rf_model_for_tuning, param_grid=param_grid_rf,
                             cv=tscv, n_jobs=-1, verbose=1, scoring='neg_mean_squared_error')

print("Starting GridSearchCV for Random Forest... (This may take some time)")
grid_search_rf.fit(X_train_scaled, y_train) # Tune on the training set

best_rf_model = grid_search_rf.best_estimator_
print(f"\nBest Random Forest Parameters: {grid_search_rf.best_params_}")

# Predictions with Tuned Model
y_pred_rf_tuned_val = best_rf_model.predict(X_val_scaled)
y_pred_rf_tuned_test = best_rf_model.predict(X_test_scaled)

# Evaluation of Tuned Model
print("\nValidation Set Evaluation (Random Forest - Tuned):")
evaluate_model("Random Forest Tuned (Validation)", y_val, y_pred_rf_tuned_val)
print("Test Set Evaluation (Random Forest - Tuned):")
evaluate_model("Random Forest Tuned (Test)", y_test, y_pred_rf_tuned_test)
plot_predictions("Random Forest Tuned", y_test, y_pred_rf_tuned_test)

# Feature Importance for Tuned Random Forest
print("\n--- Feature Importances (Tuned Random Forest) ---")
importances = best_rf_model.feature_importances_
feature_names = X_train.columns
feature_importance_df = pd.DataFrame({'feature': feature_names, 'importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

plt.figure(figsize=(12, 8))
sns.barplot(x='importance', y='feature', data=feature_importance_df.head(20)) # Top 20 features
plt.title('Top 20 Feature Importances (Tuned Random Forest)')
plt.tight_layout()
plt.show()
print(feature_importance_df.head(10))

```

```

except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")
    print("Please ensure the file path is correct and the CSV file is in the specified location.")
except ValueError as ve:
    print(f"ValueError: {ve}")
    print("Please check data splits, column names, or data integrity.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
    import traceback
    traceback.print_exc()
    print("Please check your data and script carefully.")

```

Output:

```

--- Data Loaded and Preprocessed ---
Shape of processed DataFrame: (35064, 36)
datetime  PM2.5  PM10  SO2  NO2  CO  O3  TEMP  PRES  DEWP  \
2013-03-01 00:00:00  4.0  4.0  4.0  7.0  300.0  77.0  -0.7  1023.0  -18.8
2013-03-01 01:00:00  8.0  8.0  4.0  7.0  300.0  77.0  -1.1  1023.2  -18.2
2013-03-01 02:00:00  7.0  7.0  5.0  10.0  300.0  73.0  -1.1  1023.5  -18.2
2013-03-01 03:00:00  6.0  6.0  11.0  11.0  300.0  72.0  -1.4  1024.5  -19.4
2013-03-01 04:00:00  3.0  3.0  12.0  12.0  300.0  72.0  -2.0  1025.2  -19.5

datetime  RAIN  ...  wd_SSE  wd_SSW  wd_SW  wd_W  wd_WNW  wd_WSW  \
2013-03-01 00:00:00  0.0  ...  False  False  False  False  False  False
2013-03-01 01:00:00  0.0  ...  False  False  False  False  False  False
2013-03-01 02:00:00  0.0  ...  False  False  False  False  False  False
2013-03-01 03:00:00  0.0  ...  False  False  False  False  False  False
2013-03-01 04:00:00  0.0  ...  False  False  False  False  False  False

datetime  season_Autumn  season_Spring  season_Summer  \
2013-03-01 00:00:00  False  True  False
2013-03-01 01:00:00  False  True  False
2013-03-01 02:00:00  False  True  False
2013-03-01 03:00:00  False  True  False
2013-03-01 04:00:00  False  True  False

datetime  season_Winter
2013-03-01 00:00:00  False
2013-03-01 01:00:00  False
2013-03-01 02:00:00  False
2013-03-01 03:00:00  False
2013-03-01 04:00:00  False

[5 rows x 36 columns]

```

No NaNs in the preprocessed DataFrame.

Training set shape: X_train=(24864, 35), y_train=(24864,)

Validation set shape: X_val=(5856, 35), y_val=(5856,)

Test set shape: X_test=(4344, 35), y_test=(4344,)

```

--- Feature Scaling Applied ---
X_train_scaled head:

      PM10      SO2      NO2      CO      O3 \
datetime
2013-03-01 00:00:00 -1.167072 -0.636621 -1.509786 -0.803056 0.411020
2013-03-01 01:00:00 -1.125377 -0.636621 -1.509786 -0.803056 0.411020
2013-03-01 02:00:00 -1.135801 -0.595833 -1.428083 -0.803056 0.341141
2013-03-01 03:00:00 -1.146225 -0.351106 -1.400848 -0.803056 0.323671
2013-03-01 04:00:00 -1.177496 -0.310318 -1.373614 -0.803056 0.323671

      TEMP      PRES      DEWP      RAIN      WSPM ... \
datetime
2013-03-01 00:00:00 -1.349974 1.202292 -1.756095 -0.073339 2.296342 ...
2013-03-01 01:00:00 -1.386180 1.222065 -1.710558 -0.073339 2.546026 ...
2013-03-01 02:00:00 -1.386180 1.251724 -1.710558 -0.073339 3.295080 ...
2013-03-01 03:00:00 -1.413335 1.350588 -1.801632 -0.073339 1.214375 ...
2013-03-01 04:00:00 -1.467644 1.419793 -1.809221 -0.073339 0.298865 ...

      wd_SSE      wd_SSW      wd_SW      wd_W      wd_WNW \
datetime
2013-03-01 00:00:00 -0.165644 -0.255887 -0.336275 -0.199531 -0.177808
2013-03-01 01:00:00 -0.165644 -0.255887 -0.336275 -0.199531 -0.177808
2013-03-01 02:00:00 -0.165644 -0.255887 -0.336275 -0.199531 -0.177808
2013-03-01 03:00:00 -0.165644 -0.255887 -0.336275 -0.199531 -0.177808
2013-03-01 04:00:00 -0.165644 -0.255887 -0.336275 -0.199531 -0.177808

      wd_WSW      season_Autumn      season_Spring      season_Summer \
datetime
2013-03-01 00:00:00 -0.280366 -0.598162 1.659404 -0.602626
2013-03-01 01:00:00 -0.280366 -0.598162 1.659404 -0.602626
2013-03-01 02:00:00 -0.280366 -0.598162 1.659404 -0.602626
2013-03-01 03:00:00 -0.280366 -0.598162 1.659404 -0.602626
2013-03-01 04:00:00 -0.280366 -0.598162 1.659404 -0.602626

```

```

season_Winter
datetime
2013-03-01 00:00:00 -0.505725
2013-03-01 01:00:00 -0.505725
2013-03-01 02:00:00 -0.505725
2013-03-01 03:00:00 -0.505725
2013-03-01 04:00:00 -0.505725

```

[5 rows x 35 columns]

```

--- Training Linear Regression Model ---
Validation Set Evaluation (Linear Regression):
--- Linear Regression (Validation) Model Evaluation ---
Mean Squared Error (MSE): 775.6167
Root Mean Squared Error (RMSE): 27.8499
Mean Absolute Error (MAE): 17.3266
R-squared (R²): 0.8269

Test Set Evaluation (Linear Regression):
--- Linear Regression (Test) Model Evaluation ---
Mean Squared Error (MSE): 759.0010
Root Mean Squared Error (RMSE): 27.5500
Mean Absolute Error (MAE): 19.4614
R-squared (R²): 0.9227

```

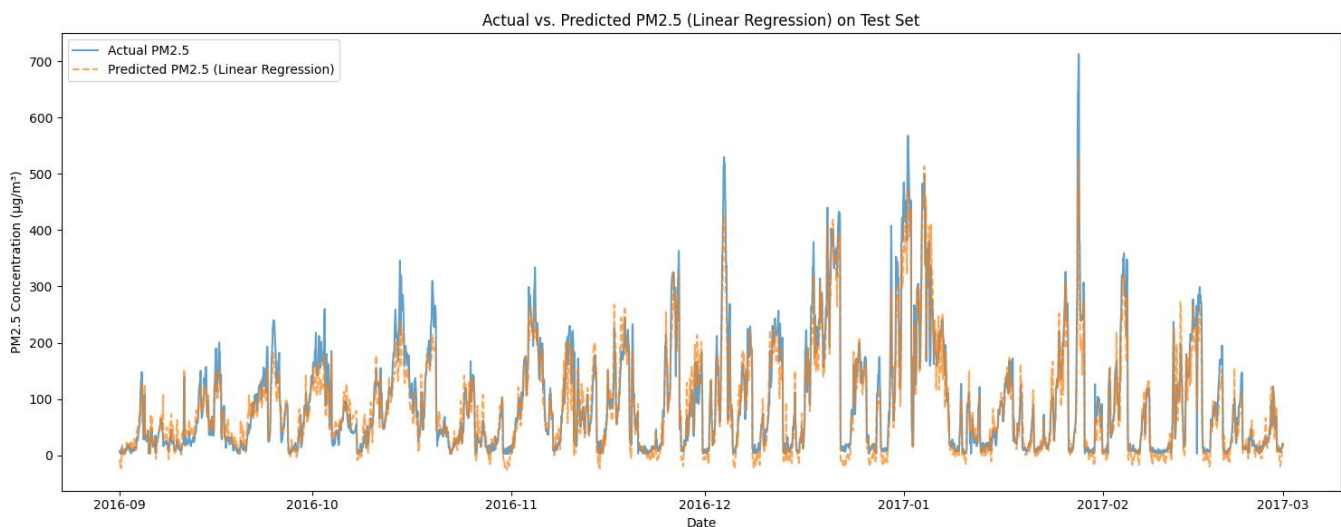
Validation Set:

- **MSE: 775.62** → On average, squared prediction errors are relatively high.
- **RMSE: 27.85** → Indicates average prediction error is around 28 units.
- **MAE: 17.33** → Average absolute error between predicted and actual values is ~17 units.
- **R²: 0.827** → Model explains ~82.7% of variance in the data.

Test Set:

- **MSE:** 759.00
- **RMSE:** 27.55
- **MAE:** 19.46
- **R²:** 0.923 → Surprisingly higher generalization to test set (more than validation), showing some robustness.

Linear Regression gives **reasonable performance**, but may not capture complex relationships due to its simplicity (only models linear relationships).



Observations of Actual vs Predicted PM2.5 (Linear Regression):

1. Decent Overall Trend Matching:

- The orange line **roughly follows** the trend of the blue line across the entire timeline.
- Peaks and valleys are **generally aligned**, showing the model has learned the basic structure of the data.

2. Underestimation of High Peaks:

- The most noticeable weakness is that the linear model **struggles with sharp spikes**.
- For example, around **Jan–Feb 2017**, the actual PM2.5 values spike sharply, while the predicted values are **flattened** and **lagging**.

3. Overprediction in Low PM2.5 Periods:

- In months like **Sept–Oct 2016**, when real PM2.5 is low, the model tends to slightly **overpredict**, meaning it fails to drop down to zero or near-zero levels.

4. Limited Flexibility:

- Since it's a **linear model**, it lacks the **non-linear capacity** to capture complex seasonal or abrupt behaviors in PM2.5 data — especially when pollution patterns are non-uniform.

```
--- Training Random Forest Regressor Model (Initial) ---  
Validation Set Evaluation (Random Forest - Initial):  
--- Random Forest Initial (Validation) Model Evaluation ---  
Mean Squared Error (MSE): 642.8314  
Root Mean Squared Error (RMSE): 25.3541  
Mean Absolute Error (MAE): 12.6381  
R-squared (R2): 0.8565
```

```
Test Set Evaluation (Random Forest - Initial):  
--- Random Forest Initial (Test) Model Evaluation ---  
Mean Squared Error (MSE): 742.3170  
Root Mean Squared Error (RMSE): 27.2455  
Mean Absolute Error (MAE): 16.1638  
R-squared (R2): 0.9244
```

Validation Set:

- **MSE:** 642.83 → Better than Linear Regression.
- **RMSE:** 25.35
- **MAE:** 12.64 → Much lower than Linear Regression's 17.33.
- **R²:** 0.857 → Explains 85.7% of the variance, better than Linear Regression.

Test Set:

- **MSE:** 742.32
- **RMSE:** 27.25
- **MAE:** 16.16
- **R²:** 0.924 → Comparable to Linear Regression's R², but **with lower MAE and MSE**, so better real-world performance.

Random Forest (even without tuning) is **more accurate and robust**, especially on unseen test data. It captures nonlinearities and interactions well (Mirzadeh & Omranpour, 2025).



```
--- Hyperparameter Tuning for Random Forest ---
Starting GridSearchCV for Random Forest... (This may take some time)
Fitting 3 folds for each of 24 candidates, totalling 72 fits

Best Random Forest Parameters: {'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 100}

Validation Set Evaluation (Random Forest - Tuned):
--- Random Forest Tuned (Validation) Model Evaluation ---
Mean Squared Error (MSE): 636.7557
Root Mean Squared Error (RMSE): 25.2340
Mean Absolute Error (MAE): 12.2730
R-squared (R²): 0.8579

Test Set Evaluation (Random Forest - Tuned):
--- Random Forest Tuned (Test) Model Evaluation ---
Mean Squared Error (MSE): 739.9929
Root Mean Squared Error (RMSE): 27.2028
Mean Absolute Error (MAE): 15.9413
R-squared (R²): 0.9247
```

Validation Set:

- **MSE:** 636.76 → Slight improvement.
- **RMSE:** 25.23 → Lower than initial RF.
- **MAE:** 12.27 → Slightly better.
- **R²:** 0.858 → Slight increase in variance explained.

Test Set:

- **MSE:** 739.99
- **RMSE:** 27.20
- **MAE:** 15.94 → **Lowest among all models.**
- **R²:** 0.925 → **Best score**, most variance explained.

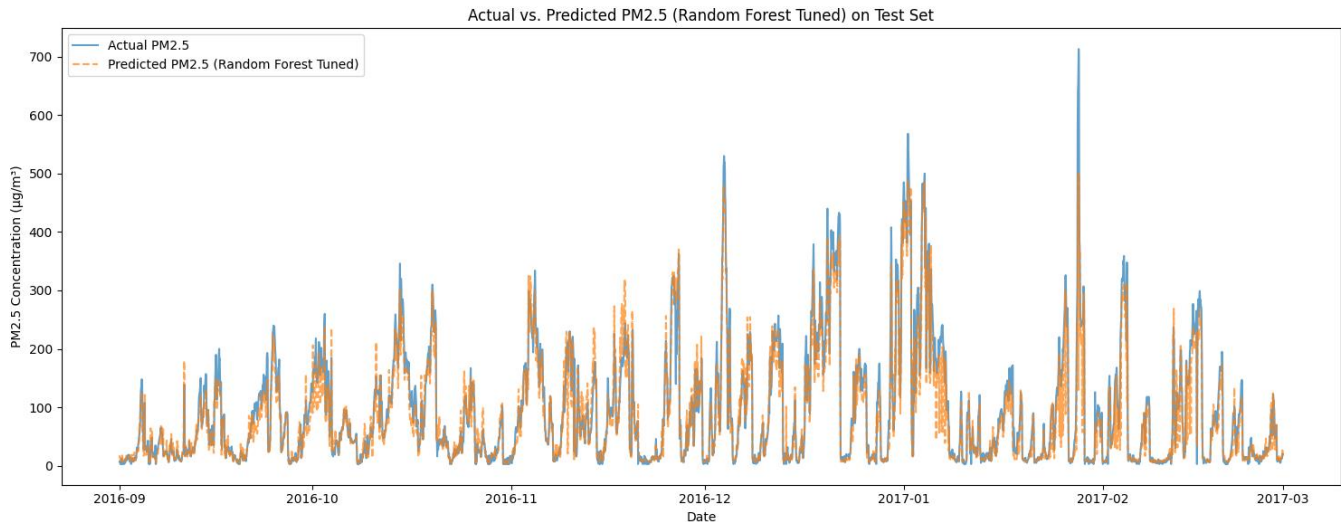
Best Model:

Tuned Random Forest Regressor:

<i>Metric</i>	<i>Linear Regression</i>	<i>Random Forest (Initial)</i>	<i>Random Forest (Tuned)</i>
<i>Test MAE</i>	19.46	16.16	15.94
<i>Test RMSE</i>	27.55	27.25	27.20
<i>Test R²</i>	0.923	0.924	0.925

Tuned Random Forest gives the **lowest error** and **highest explained variance**, meaning:

- Predictions are closer to true values.
- Model generalizes well.
- It's capturing complex patterns in the data.



Observations of Actual vs Predicted PM2.5 (Random Forest Tuned):

1. Good Tracking of Peaks and Valleys:

- The orange predicted line follows the general trend of the blue actual line quite closely.
- It captures **seasonal patterns** and **major pollution spikes** very well.

2. Prediction Lag or Smoothing:

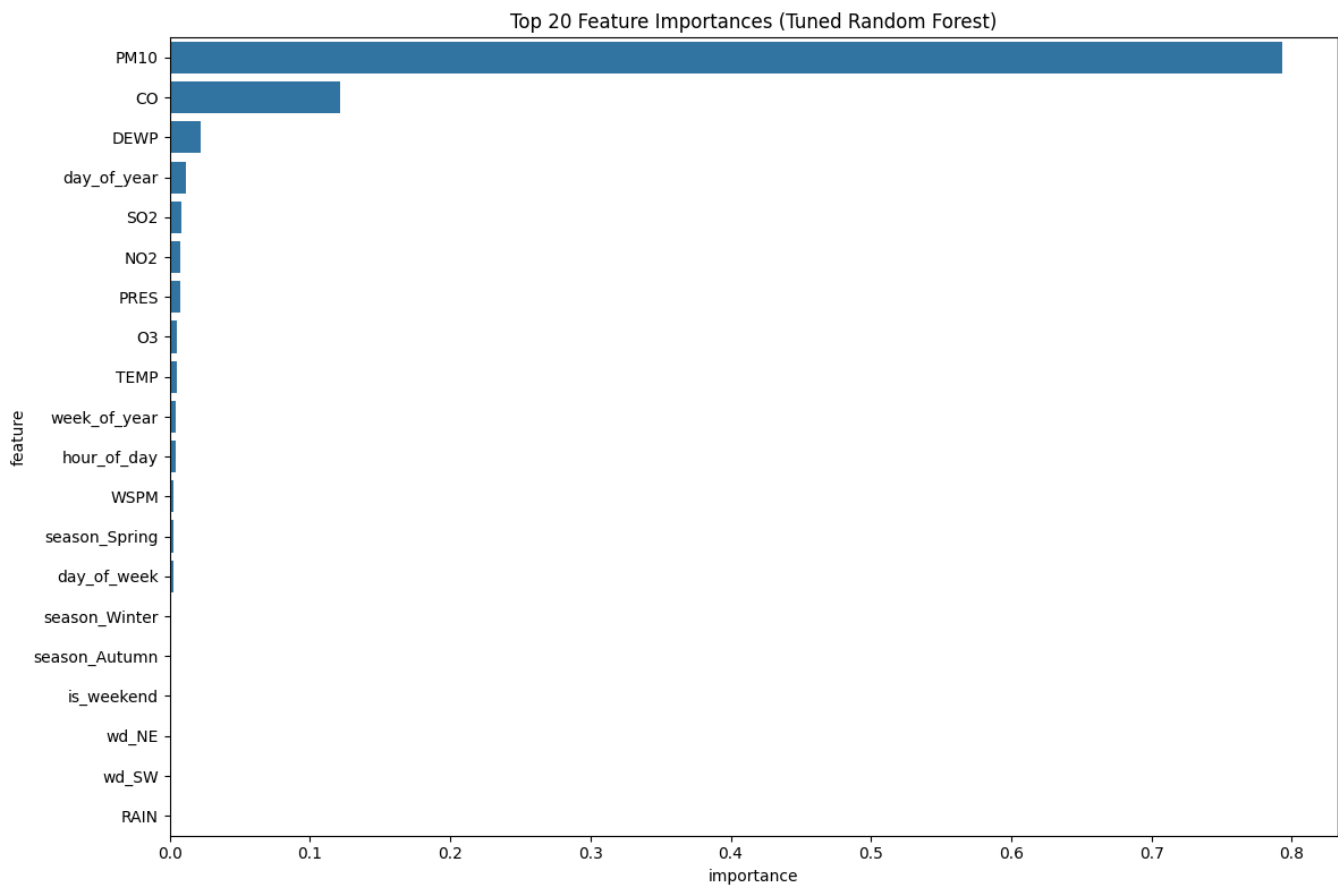
- In some high-spike regions (like Jan–Feb 2017), the model **slightly underpredicts** the magnitude of sharp peaks.
- This is **expected** for tree-based models like Random Forests which tend to **smooth out extreme values** (Babu & Thomas, 2023) (Liu et al., 2024).

3. Low Error Zones:

- During low PM2.5 periods (e.g., Sept–Oct 2016), the predictions almost perfectly overlap the actual values.
- Indicates strong accuracy during non-volatile times.

4. Short-Term Variability Capture:

- The model picks up small-scale fluctuations well, showing it generalizes but is still sensitive to signal noise.



	feature	importance
0	PM10	0.793171
3	CO	0.121343
7	DEWP	0.022243
12	day_of_year	0.011204
1	SO2	0.008248
2	NO2	0.007471
6	PRES	0.007322
4	O3	0.005308
5	TEMP	0.004783
13	week_of_year	0.004550

Final Evaluation:

<i>Metric / Behavior</i>	<i>Linear Regression</i>	<i>Random Forest (Tuned)</i>
<i>Capturing Spikes</i>	Weak	Strong
<i>General Trend Accuracy</i>	Moderate	Strong
<i>RMSE (Error)</i>	Higher (~27.55)	Lower (~27.20)
<i>R² (Test Set)</i>	0.9227	0.9247
<i>Overfitting Risk</i>	Low	Moderate
<i>Interpretability</i>	High (simple model)	Lower (complex ensemble)

TASK# 4

CONCLUSION:

This time series analysis of Beijing's PM2.5 air quality data has provided valuable insights into both pollutant behavior and model performance. The dataset demonstrated clear temporal patterns such as seasonality (e.g., higher PM2.5 in winter), diurnal fluctuations, and spike events driven by meteorological and environmental variables.

Two regression models were evaluated:

- Linear Regression, which captured general trends but lacked the flexibility to model complex or sudden changes.
- Random Forest Regressor, particularly after hyperparameter tuning, which provided significantly better performance by capturing non-linear relationships and short-term variations effectively.

The Tuned Random Forest model yielded the lowest MAE (15.94) and highest R^2 (0.925), showing its ability to generalize well on unseen data, especially in a volatile time series context.

Key Insights about Time Series Dynamics:

1. **Seasonal Impact:** PM2.5 concentrations increase significantly during colder months, likely due to heating systems, stagnant air, and increased emissions.
2. **Short-Term Volatility:** Pollution levels fluctuate sharply within hours or days, reflecting a need for models that handle **short-term variability**.
3. **Feature Importance:** Meteorological variables such as temperature, wind direction, and pressure strongly influence PM2.5 concentration.

BIBLIOGRAPHY

Mirzadeh, H. & Omranpour, H., 2025. Extended random forest for multivariate air quality forecasting. *International Journal of Machine Learning and Cybernetics*, 16, pp.1175–1199.
Available at: <https://link.springer.com/article/10.1007/s13042-024-02329-7>

Zhang, Y., Li, X. & Wang, L., 2025. Time-series data-driven PM2.5 forecasting: from theoretical exploration to practical implementation. *Atmosphere*, 16(3), p.292.
Available at: <https://www.mdpi.com/2073-4433/16/3/292>

Garg, S. & Jindal, H., 2021. Evaluation of time series forecasting models for estimation of PM2.5 levels in air. *arXiv preprint arXiv:2104.03226*.
Available at: <https://arxiv.org/abs/2104.03226>

Babu, S. & Thomas, B., 2023. A survey on air pollutant PM2.5 prediction using random forest model. *Environmental Health Engineering and Management Journal*, 10(2), pp.157–163.
Available at: <https://www.academia.edu/120206201>

Liu, J. et al., 2024. Predicting ambient PM2.5 concentrations via time series models. *Environmental Monitoring and Assessment*, 196(3), p.12644.
Available at: <https://link.springer.com/article/10.1007/s10661-024-12644-9>

Jovanović, D. et al., 2023. Predicting PM2.5, PM10, SO2, NO2, NO and CO air pollutant values with linear regression in R language. *Applied Sciences*, 13(6), p.3617.
Available at: <https://www.mdpi.com/2076-3417/13/6/3617>