LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CU6051NI Artificial Intelligence

25% Individual Coursework

Submission: Final Submission

Academic Semester: Autumn Semester 2025

Credit: 15 credit semester long module

Student Name: Shivam Kumar Thakur

London Met ID: 23050396

College ID: NP01CP4A230301

Assignment Due Date: 21/01/2026

Assignment Submission Date: 21/01/2026

Submitted To: Er. Roshan Shrestha

GitHub Link	https://github.com/dashivam06/kathmandu-air-quality-prediction

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will b e awarded.

Table of Content

Acronyms and Abbreviations Used in This Study

Air Quality & Health Acronyms	
PM2.5	Particulate Matter 2.5 micrometers or smaller
O3	Ozone
AQI	Air Quality Index
WHO	World Health Organization

Technical & Statistical Acronyms	
AI	Artificial Intelligence
ML	Machine Learning
RNN	Recurrent Neural Network
LSTM	Long Short- Term Memory
CV	Cross Validation
NaN	Not a Number
UTC	Coordinated Universal Time

Model Evaluation Metrics	
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
MSE	Mean Squared Error
R2	R- squared (Coefficient of Determination)

Table of Tables

Table of Figures

1. Title of the Project

Short-Term Air Quality Prediction Using Time Series Analysis and Machine Learning Models

2. Introduction

2.1. Introduction to the topic

Air pollution has come out as one of the most critical environmental and public health challenges, particularly in developing cities like Kathmandu, which are experiencing rapid urbanization and industrial growth. The concentration of harmful pollutants in the air, including particulate matter (PM2.5) and ground-level ozone (O_3), has risen significantly over the past decades, posing serious threats to human health. PM2.5 particles are extremely fine and can penetrate deep into the respiratory tract and bloodstream, leading to respiratory diseases, cardiovascular problems, and other long-term health complications. Ground-level ozone, on the other hand, can exacerbate lung inflammation and reduce lung function, especially in vulnerable populations such as children, the elderly, and individuals with pre-existing health conditions. Beyond direct health impacts, poor air quality also affects overall quality of life, reduces productivity, and increases healthcare costs, making it a pressing socio-economic concern.

In cities like Kathmandu, geographical factors such as valley topography further worsen air pollution by trapping pollutants and limiting natural dispersion. Seasonal factors, including dry winters and temperature inversions, intensify pollution levels for prolonged periods. In such contexts, monitoring air pollution and understanding its patterns is not only vital for public health but also for urban planning, environmental policy-making, and sustainable development strategies. Reliable air quality information forms the foundation for mitigation planning and long-term environmental management.

2.2. Introduction to the chosen problem domain

The primary focus of this project is urban air quality monitoring and predictive modeling, with a specific emphasis on short-term forecasting of PM2.5 concentrations in Kathmandu. Urban air pollution is highly dynamic in nature, with pollutant levels changing hourly due to a combination of human activities and environmental factors. Vehicular emissions, industrial operations, construction activities, and residential fuel usage contribute continuously to pollution levels, while meteorological conditions influence pollutant dispersion and accumulation. These interacting factors create complex temporal patterns that are difficult to model using traditional analytical approaches.

Short-term prediction of PM2.5 concentrations is particularly important because even brief exposure to elevated pollution levels can have serious health impacts. Timely forecasting allows health authorities to issue early warnings and enables individuals to reduce outdoor exposure during high-risk periods. Urban administrators can also use predictive information to implement temporary mitigation strategies such as traffic regulation and industrial control, supporting proactive environmental management and pollution mitigation.

The chosen problem domain further involves analyzing large-scale historical air quality data, which often contain missing values, sensor errors, and temporal irregularities. PM2.5 data exhibit strong temporal dependencies and occasional extreme spikes, making them well-suited for time-series modeling. By focusing on PM2.5 prediction in Kathmandu, this project addresses a locally relevant and socially significant problem while demonstrating how data-driven methods can support air quality monitoring, public health protection, and evidence-based environmental policymaking in rapidly urbanizing regions.

2.3. AI Concepts Used

a) Regression (Machine Learning Regression Models)

Regression in machine learning is a supervised learning technique used to predict continuous numerical values by learning relationships between input variables (features) and an output variable (target). (GeeksForGeeks, 2026)

b) Recurrent Neural Network (LSTM)

A recurrent neural network or RNN is a deep neural network trained on sequential or time series data to create a machine learning (ML) model that can make sequential predictions or conclusions based on sequential inputs. (IBM, 2026)

c) Time-Series Feature Engineering

Time-series feature engineering involves transforming raw temporal data into meaningful input features for machine learning models. In this project, lag-based features representing PM2.5 values from previous hours are created to capture short-term trends in air pollution levels.

d) Data Preprocessing and Cleaning

Data preprocessing is a critical step to ensure the reliability of machine learning models. In this project, invalid PM2.5 sensor readings are identified using threshold-based rules and treated as missing values. Missing values is then handled using forward filling and backward filling technique.

e) Model Evaluation Metrics

Model evaluation metrics are used to assess the accuracy and reliability of the prediction models. Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are used to measure prediction error, while the R^2 score is implemented in our models to ensure and explain variations in PM2.5 concentrations.

3. Background of the Study

3.1. Problem Overview

3.1.1. Air Pollution Crisis in the Kathmandu Valley

The rise in air pollution has put Nepal on the crisis of public health and environmental issues. There are various polluters like hasty urbanization, traffic congestion, industrial activities and other daily sources of emission which have surpassed the acceptable range of air quality deterioration in Kathmandu valley. In the major city areas of the valley, such as Kathmandu, Lalitpur, Bhaktapur etc. face similar types of emission sources and urban infrastructure that end up in producing similar problem of air quality deterioration.

3.1.2. Trends in Air Quality Index (AQI)

According to recent air quality monitoring, the AQI level in the Kathmandu Valley is regularly between "Poor" to "Very Unhealthy, " putting the residents of the Kathmandu Valley at serious risk for illness. The Kathmandu Valley consistently measures PM2.5 concentrations far above international safety thresholds, the most dangerous type of particulate pollution, which can enter the bloodstream and deeply penetrate the lungs.

Figure 1 : AQI Trend Over the Year 2025

3.1.3. PM2.5 and Its Health Impacts

PM2.5 are tiny particles that can be breathed in. They are dangerous because they are so small that they can get deep into the lungs and into the bloodstream. Vehicle exhaust, brick kilns, road dust, industrial emissions, and burning fuel in homes are all major sources of PM2.5 in Kathmandu.

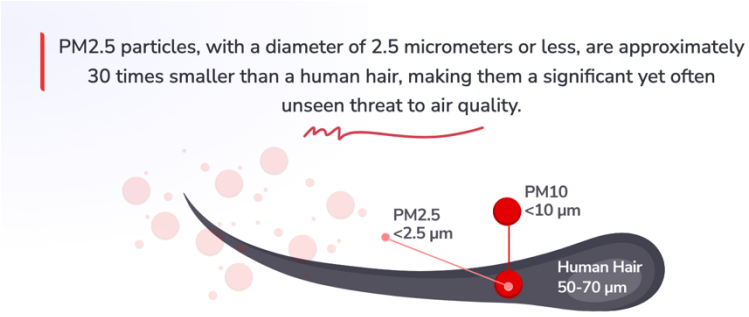


Figure 2 : PM 2.5 comparison to human hair

Figure 3 : PM25 Concentration over the past month in Kathmandu

According to international organization reports. The WHO recommends that the concentration of annual mean pm2.5 should be 5µg/m³, yet in Kathmandu it is recorded at 7 times higher than that defined figure.

3.1.4. Mortality and Disease Burden Linked to Air Pollution

The effects of chronic exposure to these pollutants are disastrous. In 2019, Nepal reported roughly 59 deaths per 100, 000 people attributable to diseases linked to high levels of pm2.5, with a national mean pm2.5 concentration of about 83 μg/m³. Air pollution is the primary cause of premature deaths in the nation.

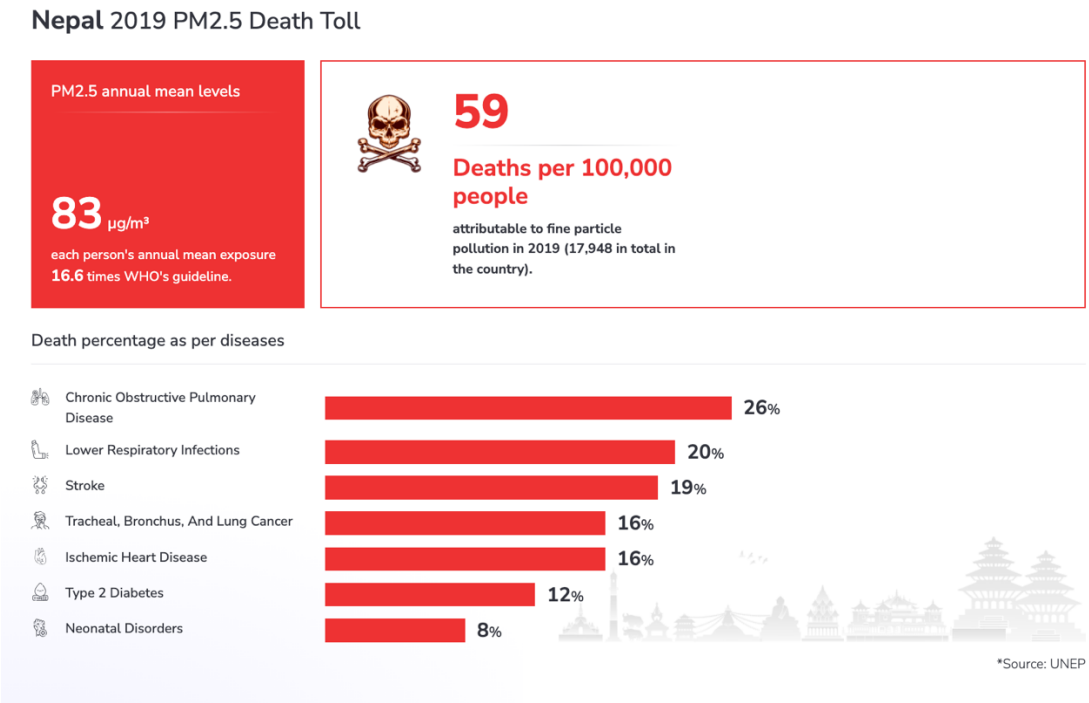


Figure 4 : 2019 PM2.5 Death Toll

3.1.5. PM2.5 Exposure Expressed as Cigarette Equivalents

To better inform people about the health impacts of PM2.5 exposure, air pollution levels can be explained by using cigarette-equivalent metrics. As per the known exposure conversion, the current pollution levels in the Kathmandu Valley are equal to the following approximate inhalation equivalents:

- Per day: ~ 2.7 cigarettes
- Per week: ~18.9 cigarettes
- Per month: ~81 cigarettes

This means, the person in Kathmandu who does not smoke is involuntarily exposed to air pollution equivalent to smoking nearly three cigarettes every day, solely through breathing ambient air.

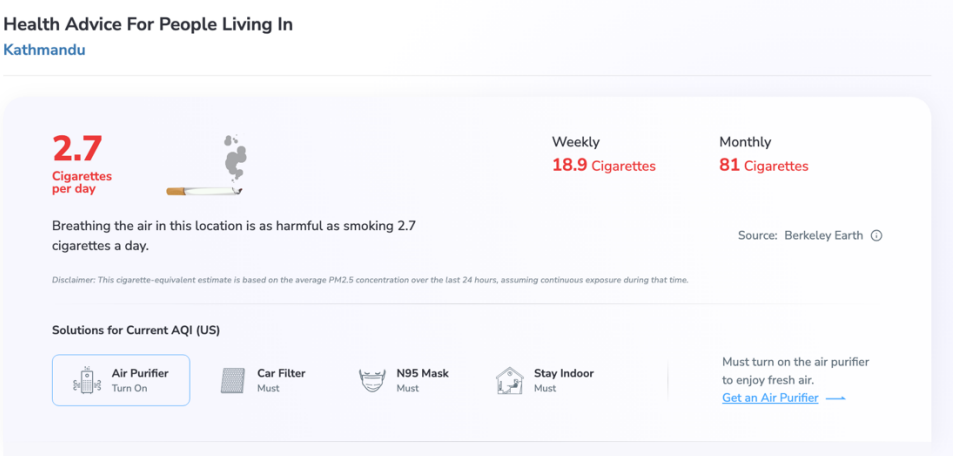


Figure 5 : PM2.5 Exposure Expressed as Cigarette Equivalent

3.1.6. Comparative Air Quality Analysis with Global Cities

Air quality in Kathmandu Valley is becoming a cause for concern for people worldwide. Comparisons between the air quality levels in Kathmandu and West Vancouver (Canada) reveal severe differences in AQI ratings, despite both urban areas having similarly developed vehicles operating on their streets..

Figure 6 : Lalitpur AQI compared to Vancouver Canada

3.1.7. Media Attention and Public Awareness

Public awareness of Kathmandu's air quality crisis has increased significantly, as seen in extensive media coverage. Major international and regional media outlets and content creators have repeatedly highlighted Kathmandu as one of the most polluted cities in the world.

3.2. Research work done in coursework 1

In Coursework 1, the theme of the project was changed from predicting NEPSE stock price to predicting PM2.5 air pollution. This was because predicting stock price was not of any major social concern. The data collected for this research was collected from Open Data Nepal. They provide free air quality sensor data for the Kathmandu Valley.

First, the dataset needed intensive preprocessing. The time stamp columns recording times in both local time and UTC converted to the standard datetime format. On closer observation, the quality problems in the dataset included missing values, null values, and inaccurate PM2.5 readings of 999, which are not feasible values for the amount of pollution. The inaccurate values in the dataset have been treated as missing.

For the treatment of missing data while ensuring temporal continuity, both forward fill (ffill) and backward fill (bfill) functions were used to fill the missing entries to a maximum of two consecutive entries to not significantly alter the original patterns of air pollution. Missing entries not handled through this strategic imputation were retained as NaN. After this step, features based on lag were developed to identify temporal relationships for PM2.5 concentrations. Data rows with NaN entries for the developed lag features were deleted.

Then, exploratory data analysis and visualization techniques were conducted to better understand the data. A bar chart was used to analyze sensor downtime and determine when air quality sensors were not active. Furthermore, scatter plots were used to display PM2.5 values to ensure the effectiveness of data preprocessing.

Based on the nature of the problem and the characteristics of the data at hand, several machine learning models were identified for experimentation. These models included Linear Regression, Random Forest Regression, and LSTM networks. Linear Regression was identified for experimentation based on its theoretical foundations for which an elaborate explanation had been proposed within the coursework document. Random Forest models were identified based on their ability to handle non-linear relationships within the data. LSTM networks were identified based on their ability to handle temporal relationships within time-series data. Suitable metrics for evaluation of each identified model had been determined.

3.3. Related Works

The related research works on air quality prediction and PM2.5 modeling are summarized in different tables. A separate table has been dedicated to each study, representing the authors, problem studied, dataset, methodology, findings, and takeaways. This arrangement provides clarity and a well-structured outlook of previous studies and their approaches to clear up any confusion regarding the methodology of this project.

Related Works: 1

Author(s)	Basanta Khadka, Nabin G.C, Umesh Sapkota, Utsav Dhungana (2020)
Year Published	2020
Problem Addressed	Air quality prediction in Kathmandu
Method Used	Multiple Linear Regression, Random Forest, Neural Network, LSTM
Dataset	Air quality data from Kathmandu (PM2.5, O3, etc.)
Results	Neural Network provided best prediction for hourly, daily, and current AQI; ML models effectively captured temporal trends
Observations	▪ Dataset limited to specific monitoring stations ▪ Some sensor downtime ▪ PM2.5 -999 values required preprocessing
Link	https://www.scribd.com/document/510300340/Aqi-to-Print

Table 1 : Related Work (I)

Related Works (I) - Reference Image:

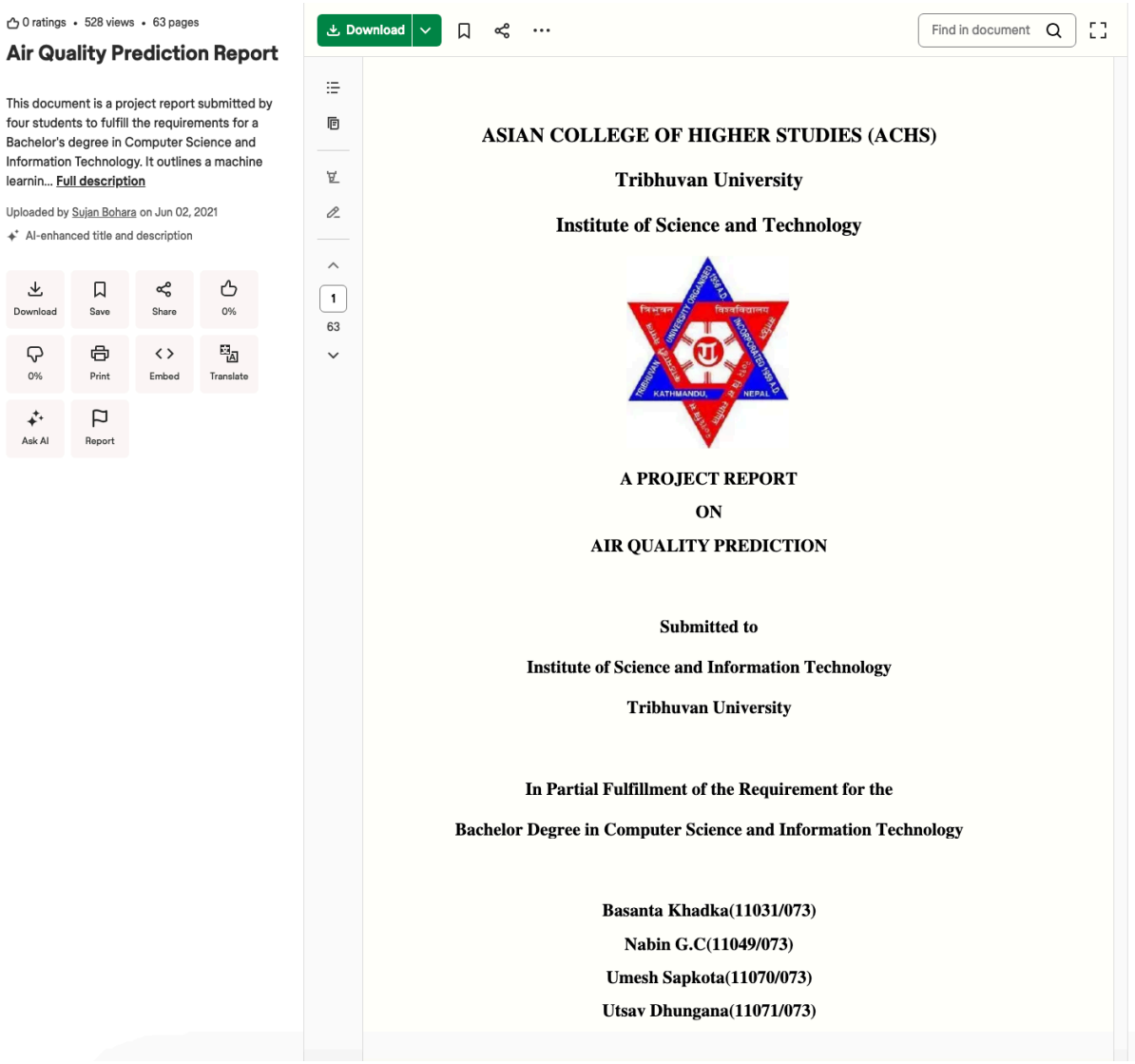


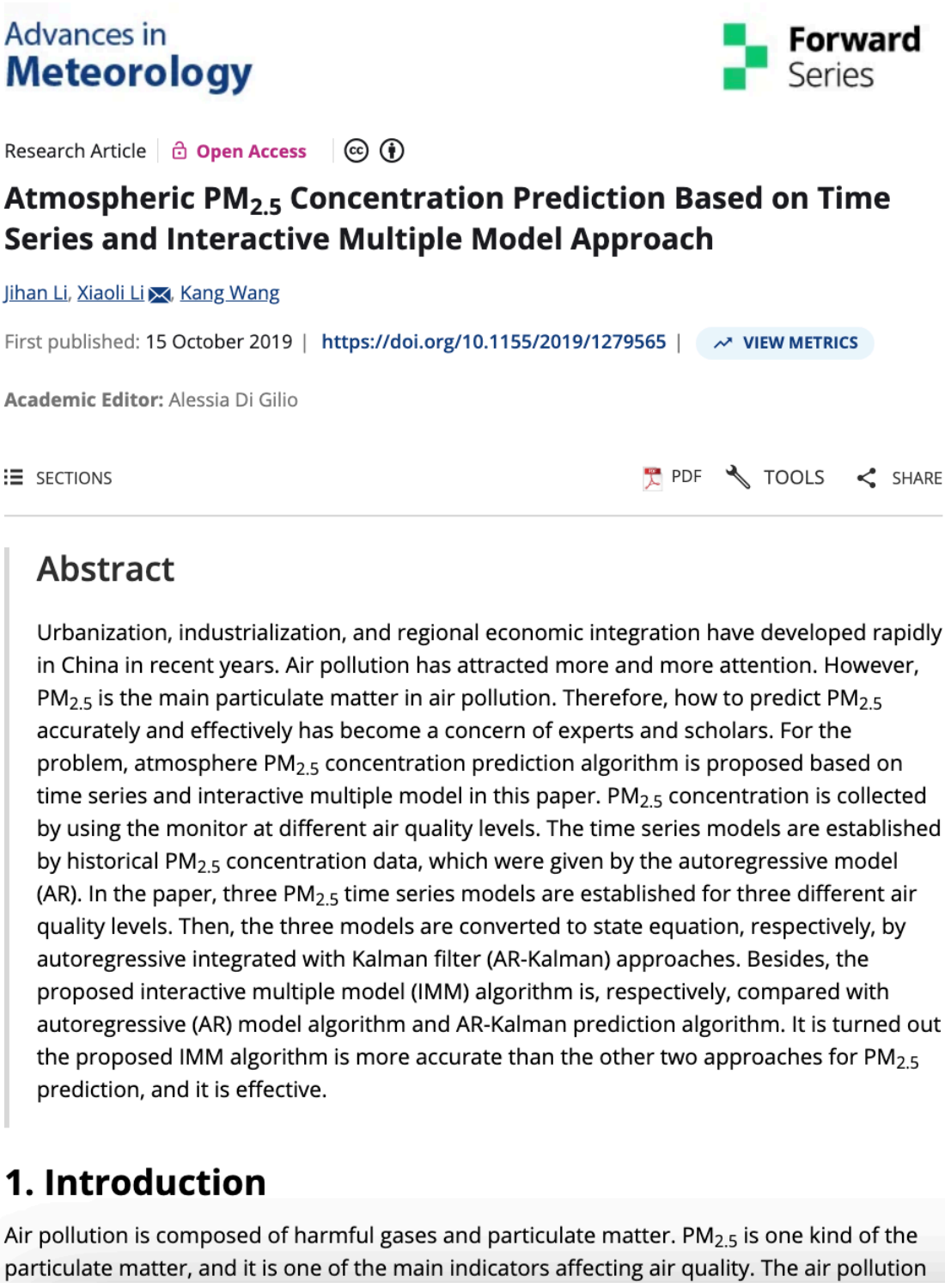
Figure 7 : Related Work (I) - Reference Image

Related Works: 2

Author(s)	Jihan Li, Xiaoli Li, Kang Wang
Year Published	2019
Problem Addressed	PM2.5 time series prediction
Method Used	Machine Learning, Regression
Dataset	Air quality dataset
Results	Demonstrated ML methods can predict short-term PM2.5 fluctuations
Observations	▪ Limited feature explanation ▪ Regional dataset only
Link	https://onlinelibrary.wiley.com/doi/10.1155/2019/1279565

Table 2 : Related Work (II)

Related Works (II) - Reference Image:



Advances in Meteorology

Forward Series

Research Article | Open Access

Atmospheric $PM_{2.5}$ Concentration Prediction Based on Time Series and Interactive Multiple Model Approach

Jihan Li, Xiaoli Li, Kang Wang

First published: 15 October 2019 | https://doi.org/10.1155/2019/1279565 | VIEW METRICS

Academic Editor: Alessia Di Gilio

SECTIONS PDF TOOLS SHARE

Abstract

Urbanization, industrialization, and regional economic integration have developed rapidly in China in recent years. Air pollution has attracted more and more attention. However, $PM_{2.5}$ is the main particulate matter in air pollution. Therefore, how to predict $PM_{2.5}$ accurately and effectively has become a concern of experts and scholars. For the problem, atmosphere $PM_{2.5}$ concentration prediction algorithm is proposed based on time series and interactive multiple model in this paper. $PM_{2.5}$ concentration is collected by using the monitor at different air quality levels. The time series models are established by historical $PM_{2.5}$ concentration data, which were given by the autoregressive model (AR). In the paper, three $PM_{2.5}$ time series models are established for three different air quality levels. Then, the three models are converted to state equation, respectively, by autoregressive integrated with Kalman filter (AR-Kalman) approaches. Besides, the proposed interactive multiple model (IMM) algorithm is, respectively, compared with autoregressive (AR) model algorithm and AR-Kalman prediction algorithm. It is turned out the proposed IMM algorithm is more accurate than the other two approaches for $PM_{2.5}$ prediction, and it is effective.

1. Introduction

Air pollution is composed of harmful gases and particulate matter. $PM_{2.5}$ is one kind of the particulate matter, and it is one of the main indicators affecting air quality. The air pollution

Figure 8 : Related Works (II) - Reference Image

Related Works: 3

Author(s)	Surendra Bhatta, Yuekui Yang
Year Published	2023
Problem Addressed	Reconstructing PM2.5 data over Kathmandu Valley from 1980–present
Method Used	XGBoost Regression, Machine Learning
Dataset	PM2.5 data from US Embassy Phora Durbar, meteorological data from MERRA2
Results	10-fold CV score ~83.4%, R^2 ~84%, RMSE 15.82 μg/m³, MAE 10.27 μg/m³; cross-test R^2 56–67% for 2018–2020; model captured seasonal variations and anti-correlation with humidity
Observations	▪ Limited to historical reconstruction ▪ Model dependent on MERRA2 input ▪ Extreme events may be underrepresented ▪ Mostly focused on long-term reconstruction ▪ Not short-term hourly prediction
Link	https://www.mdpi.com/2073-4433/14/7/1073

Table 3 : Related Work (III)

Related Works (III) - Reference Image:

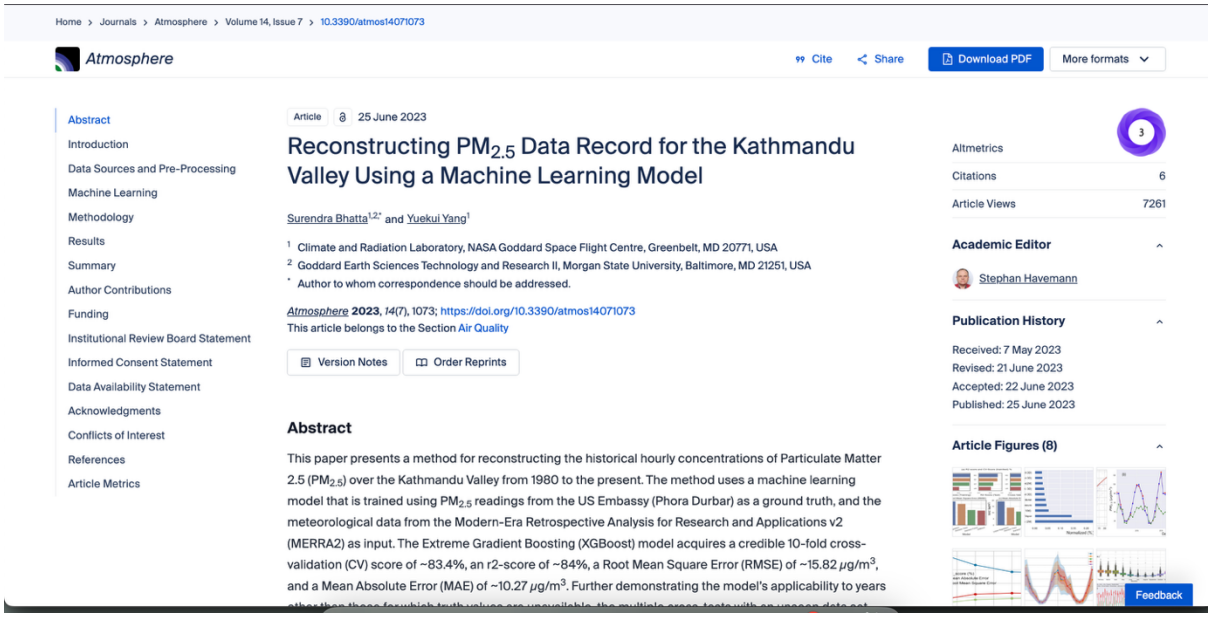


Figure 9 : Related Works (III) - Reference Image

Related Works: 4

Author(s)	Yusuf Sawsany
Year Published	2020
Problem Addressed	Time series air quality prediction in Dar es Salaam
Method Used	Linear Regression
Dataset	Hourly PM2.5 and other pollutants
Results	Linear regression captured basic trends but underperformed during high pollution peaks
Observations	▪ Focused on linear models only ▪ Did not explore ensemble or deep learning models ▪ Limited feature engineering
Link	https://medium.com/@sawsanyusuf/time-series-with-python-case-study-air-quality-in-dar-es-salaam-2-linear-regression-model-7e3fa165dff1

Table 4 : Related Work (V)

Related Works (IV) - Reference Image:

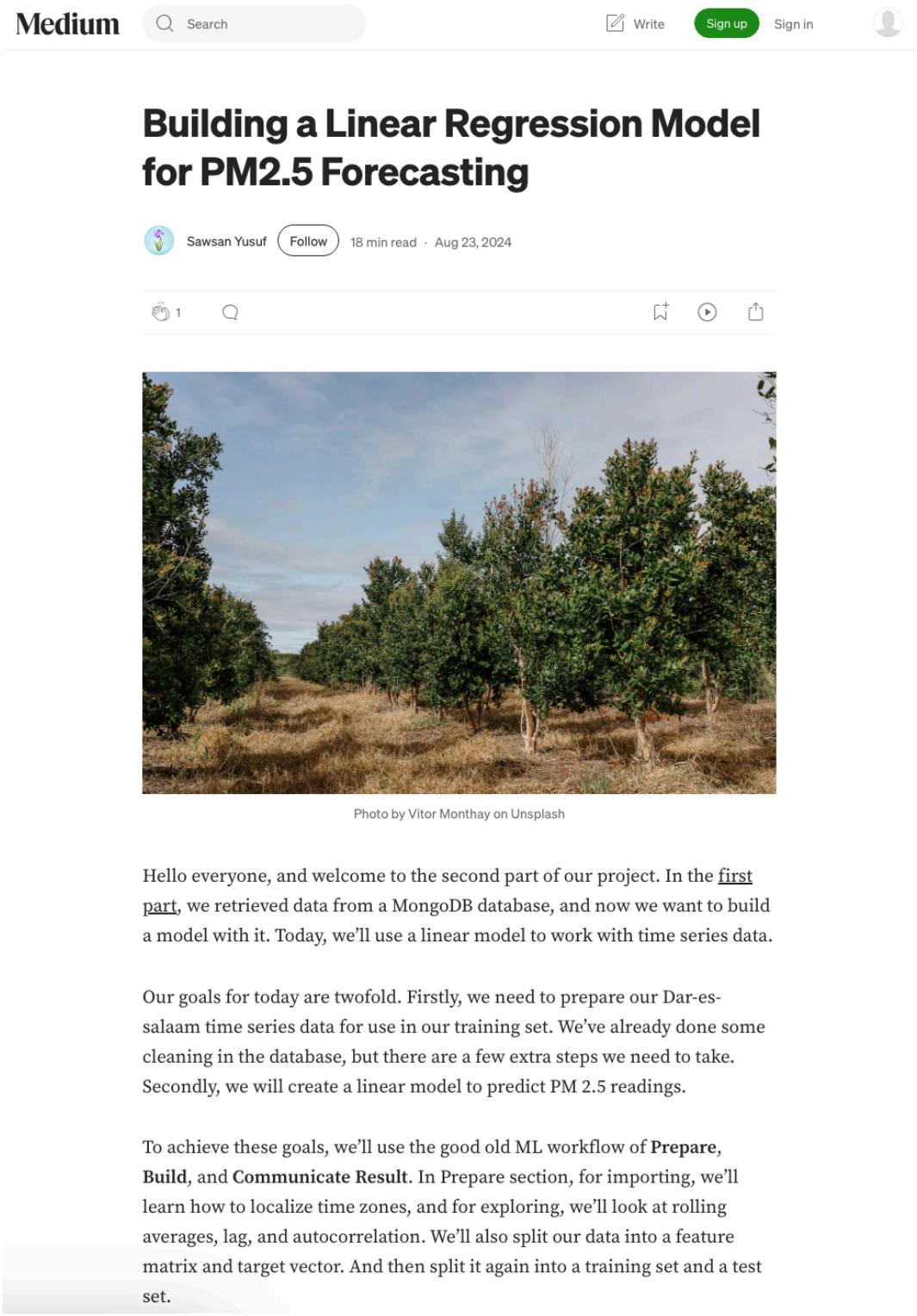


Figure 10 : Related Works (IV) - Reference Image

4. Solution

4.1. Used ML Algorithms

a) Linear Regression

Linear Regression is a fundamental statistical method used to define linear relationship between a dependent and one or more independent variables. It helps predict outcomes by fitting a straight line to observed data points, making it easy to interpret and apply (Analytics Vidhya, 2026).

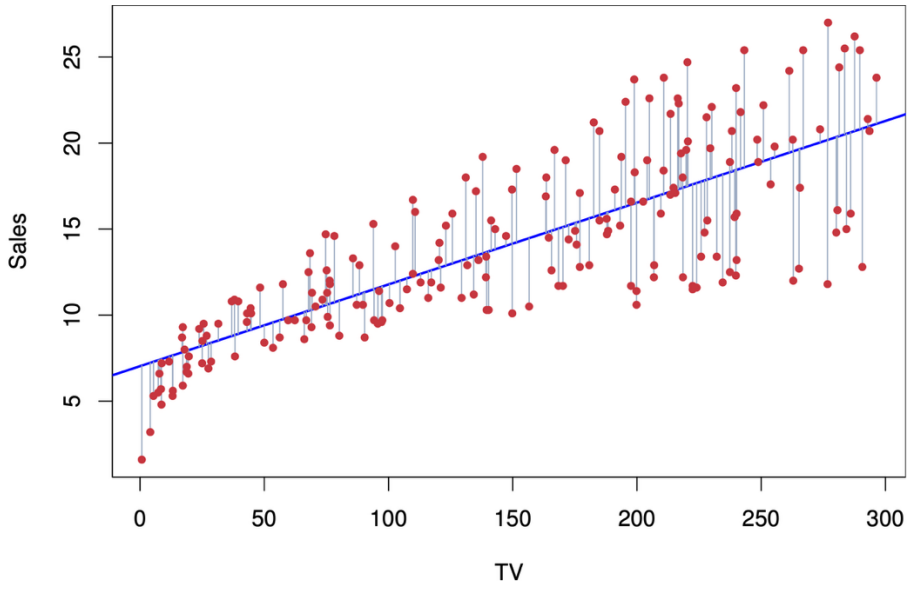


Figure 11 : Linear Regression Reference Image

Reason for Selection:

It provides a simple baseline model to predict next-hour PM2.5 values from previous 3–4 hours. It is useful for understanding linear trends and short-term dependencies in air quality data.

b) Random Forest Regression

Random forest is a commonly used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems. (IBM, 2026)

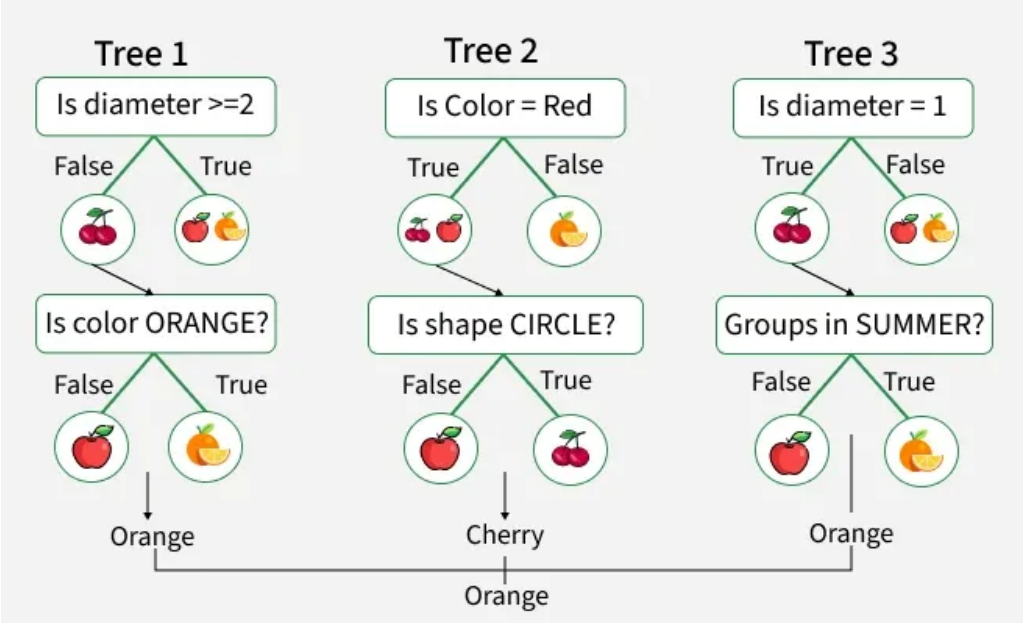


Figure 12 : Random Forest Regression Reference Image

Reason for Selection:

It captures non-linear relationships between past PM2.5 values, time-of-day, and other temporal features. Handles feature importance and is robust to outliers or unusual pollution spikes.

c) LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) is an enhanced version of the Recurrent Neural Network (RNN) designed by Hochreiter and Schmidhuber. LSTMs can capture long-term dependencies in sequential data making them ideal for tasks like language translation, speech recognition and time series forecasting. Unlike traditional RNNs which use a single hidden state passed through time LSTMs introduce a memory cell that holds information over extended periods addressing the challenge of learning long-term dependencies. (GeeksForGeeks, 2026)

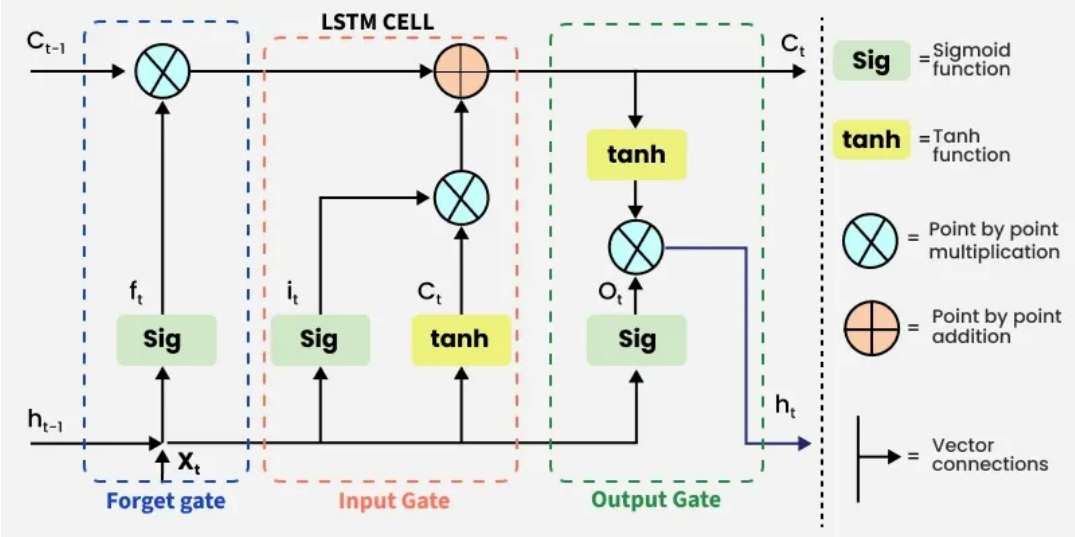


Figure 13 : LSTM Reference Image

Reason for Selection:

It is specialized for sequential data and time series. It can learn long-term dependencies, diurnal and weekly patterns, and temporal correlations in PM2.5 measurements for more accurate next-hour prediction.

4.2. Used Evaluation Metrics

Since this project involves regression to predict the next-hour PM2.5 concentration, the following metrics will be used:

a) Root Mean Squared Error (RMSE)

The root mean square error (RMSE) measures the average difference between a statistical model's predicted values and the actual values. Mathematically, it is the standard deviation of the residuals. Residuals represent the distance between the regression line and the data points. (StatisticsByJim, 2026)

$$RMSE = \sqrt{\frac{\Sigma (y_i - \hat{y}_i)^2}{N - P}}$$

Figure 14 : RMSE Formula

Reason for selecting:

Measures the square root of the average squared differences between predicted and actual PM2.5 values, emphasizing larger errors.

b) **Mean Absolute Error (MAE)**

Mean Absolute Error (MAE) is a statistical measure that evaluates the accuracy of a predictive or forecasting model by calculating the average of the absolute differences between predicted and actual values.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} | x_i - x |$$

Figure 15 : MAE Formula

MAE is expressed in the same units as the data and provides a straightforward interpretation of the average size of prediction errors. Analysts use the mean absolute error widely in forecasting, regression analysis, and machine learning to assess how closely model predictions align with observed outcomes. (Statistics By Jim, 2026)

Reason for selecting:

Calculates the average absolute difference between predicted and actual values, providing a straightforward measure of prediction accuracy.

c) **R-squared (R^2)**

R-squared is the percentage of the response variable variation that is explained by a linear model. It is always between 0 and 100%. R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. (Statistics By Jim, 2026)

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Figure 16 : R2 Squared Formula

Reason for selecting:

Indicates the proportion of variance in PM2.5 values explained by the model, assessing overall goodness of fit.

d) **Mean Squared Error (MSE)**

Mean squared error (MSE) measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values. When a model has no error, the MSE equals zero. As model error increases, its value increases. The mean squared error is also known as the mean squared deviation (MSD). (Statistics By Jim, 2026)

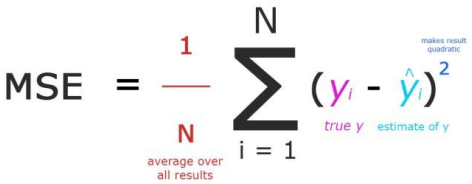


Figure 17 : MSE Formula

Reason for selecting:

Computes the average squared difference between predicted and observed values, useful for model optimization and comparison.

4.3. Tools and Technologies Used

a) Python

Serves as the primary programming language due to its simplicity, extensive libraries, and strong community support for data analysis, machine learning, and time-series modeling.

Figure 18 : Python Logo

b) Pandas

Used for data manipulation, cleaning, preprocessing, and handling structured datasets efficiently.

Figure 19 : Pandas Logo

c) NumPy

Provides support for numerical computations, array operations, and mathematical functions required for data analysis.

Figure 20 : Numpy Logo

d) Scikit-learn

Utilized for implementing traditional machine learning models such as Linear Regression and Random Forest, as well as for model evaluation and feature preprocessing.

Figure 21 : Scikit Learn Logo

e) Matplotlib

Used for creating basic plots and visualizations to understand trends and patterns in data.

Figure 22 : Matplotlib Logo

f) Seaborn

Built on top of Matplotlib, used for advanced statistical visualizations and exploratory data analysis.

Figure 23 : Seaborn Logo

g) Jupyter Notebook

Provides an interactive environment for writing code, visualizing outputs, documenting experiments, and presenting results in a structured manner.

Figure 24 : Jupyter Logo

h) Git and GitHub

Used for tracking changes in code, maintaining version history, and managing project development efficiently.

Figure 25 : Version Control

4.4. Solution Design (Pseducode and Flowchart)

4.4.1 Logistic Regression–Based Solution Design

4.4.1.1 Pseudocode for Logistic Regression Model

```
START

LOAD kathmandu_pm25_dataset.csv into data_table

KEEP ONLY rows where parameter = "pm25"

CONVERT local and utc columns to datetime format

REMOVE unnecessary columns:
   - location
   - city
   - latitude
   - longitude
   - country

REPLACE invalid PM2.5 values:
   IF value <= 3 OR value >= 102 THEN
      SET value to NaN (empty)
   END IF

FILL empty values using:
   First: Look backward up to 3 hours
   Then: Look forward up to 3 hours

CREATE lag features:
   lag1 = previous hour PM2.5 value
   lag2 = 2 hours ago PM2.5 value
   lag3 = 3 hours ago PM2.5 value

REMOVE rows with missing lag values

DIVIDE data into two parts:
   TRAINING = data before Jan 1, 2020
   TESTING = data after Jan 1, 2020
```

```
PREPARE training data:
   X_train = lag1, lag2, lag3 from TRAINING
   y_train = actual PM2.5 from TRAINING

PREPARE testing data:
   X_test = lag1, lag2, lag3 from TESTING
   y_test = actual PM2.5 from TESTING

CREATE Linear Regression model

TRAIN model using:
   Input: X_train
   Target: y_train

USE trained model to PREDICT:
   predicted_pm25 = model.predict(X_test)

CALCULATE performance metrics:
   RMSE = sqrt(average((predicted - actual)²))
   MSE = average((predicted - actual)²)
   MAE = average(|predicted - actual|)
   R² = how well predictions match actual (0 to 1)

DISPLAY metrics in formatted table

CREATE comparison plot:
   Plot actual PM2.5 values
   Plot predicted PM2.5 values
   Show both on same time chart

END
```

4.4.1.2 Flowchart of Logistic Regression Model

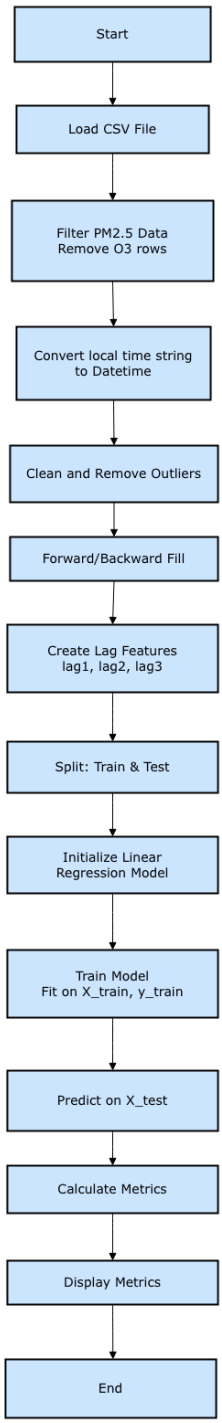


Figure 26 : Flowchart - Linear Regression

4.4.2 Random Forest–Based Solution Design

4.4.2.1 Pseudocode for Random Forest Model

```
START

LOAD kathmandu_pm25_dataset.csv into data_table

KEEP ONLY rows where parameter = "pm25"

CONVERT local and utc columns to datetime format

REMOVE unnecessary columns:
   - location
   - city
   - latitude
   - longitude
   - country

REPLACE invalid PM2.5 values:
   IF value <= 3 OR value >= 102 THEN
      SET value to NaN (empty)
   END IF

FILL empty values using:
   First: Look backward up to 3 hours
   Then: Look forward up to 3 hours

CREATE lag features:
   lag1 = previous hour PM2.5 value
   lag2 = 2 hours ago PM2.5 value
   lag3 = 3 hours ago PM2.5 value

REMOVE rows with missing lag values

DIVIDE data into two parts:
   TRAINING = data before Jan 1, 2020
   TESTING = data after Jan 1, 2020

PREPARE training data:
   X_train = lag1, lag2, lag3 from TRAINING
   y_train = actual PM2.5 from TRAINING

PREPARE testing data:
```

```
    X_test = lag1, lag2, lag3 from TESTING
    y_test = actual PM2.5 from TESTING

CREATE Random Forest model with settings:
    Number of trees = 300
    Maximum depth per tree = 12
    Minimum samples per leaf = 5
    Random seed = 42
    Use all processor cores

TRAIN Random Forest using:
    Input: X_train
    Target: y_train
    Each tree learns different patterns

USE trained forest to PREDICT:
    predicted_pm25 = average of all 300 tree predictions

CALCULATE performance metrics:
    RMSE = sqrt(average((predicted - actual)²))
    MSE = average((predicted - actual)²)
    MAE = average(|predicted - actual|)
    R² = how well predictions match actual (0 to 1)

DISPLAY metrics in formatted table

CREATE comparison plot:
    Plot actual PM2.5 values
    Plot predicted PM2.5 values
    Show both on same time chart

END
```

4.4.2.2 Flowchart of Random Forest Model

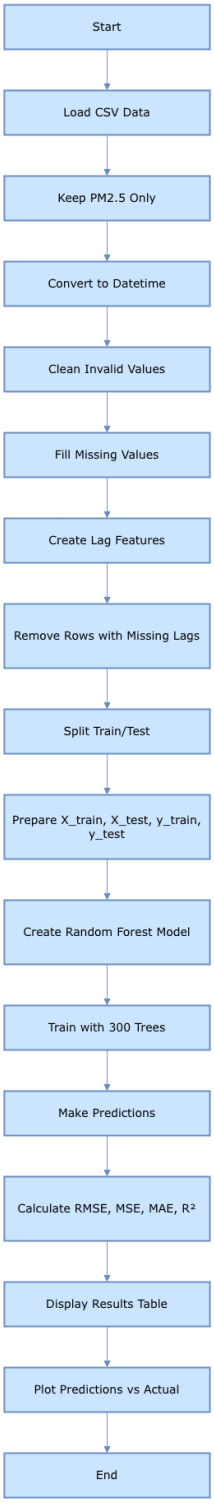


Figure 27 : Flowchart – Random Forest Regression

4.4.3 LSTM-Based Solution Design

4.4.3.1 Pseudocode for LSTM Model

```
START

LOAD kathmandu_pm25_dataset.csv into data_table

KEEP ONLY rows where parameter = "pm25"

CONVERT local and utc columns to datetime format

REMOVE unnecessary columns:
   - location
   - city
   - latitude
   - longitude
   - country

REPLACE invalid PM2.5 values:
   IF value <= 3 OR value >= 102 THEN
      SET value to NaN (empty)
   END IF

FILL empty values using:
   First: Look backward up to 3 hours
   Then: Look forward up to 3 hours

CREATE lag features:
   lag1 = previous hour PM2.5 value
   lag2 = 2 hours ago PM2.5 value
   lag3 = 3 hours ago PM2.5 value

REMOVE rows with missing lag values

DIVIDE data into two parts:
   TRAINING = data before Jan 1, 2020
   TESTING = data after Jan 1, 2020

PREPARE training data:
   X_train = lag1, lag2, lag3 from TRAINING
   y_train = actual PM2.5 from TRAINING
```

```
PREPARE testing data:
   X_test = lag1, lag2, lag3 from TESTING
   y_test = actual PM2.5 from TESTING

SCALE data between 0 and 1:
   Scale X_train values
   Scale X_test values
   Scale y_train values
   Scale y_test values

RESHAPE data for LSTM:
   Convert each row to 3 time steps
   Format as: samples × time steps × features

CREATE LSTM neural network:
   Add LSTM layer with 64 memory cells
   Add output layer with 1 neuron
   Set optimizer = Adam
   Set loss function = MSE

TRAIN LSTM model:
   Run 7 training cycles (epochs)
   Batch size = 32 samples
   Keep 20% data for validation

USE trained LSTM to PREDICT:
   predicted_scaled = model.predict(X_test_reshaped)
   Convert predictions back to original scale

CALCULATE performance metrics:
   RMSE = sqrt(average((predicted - actual)²))
   MSE = average((predicted - actual)²)
   MAE = average(|predicted - actual|)
   R² = how well predictions match actual (0 to 1)

DISPLAY metrics in formatted table

CREATE comparison plot:
   Plot actual PM2.5 values
   Plot predicted PM2.5 values
   Show both on same chart

END
```

4.4.3.2 Flowchart of LSTM Model

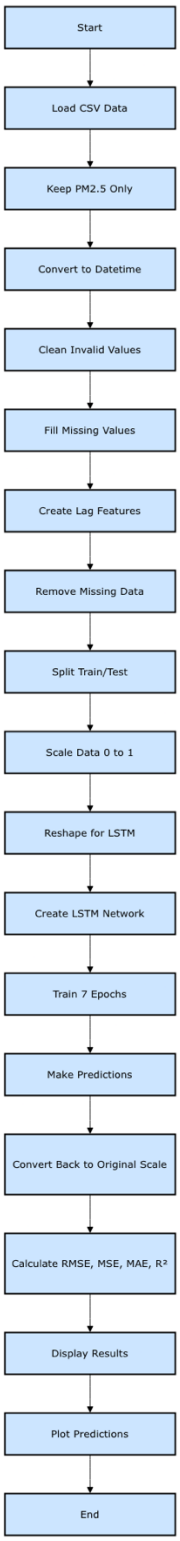


Figure 28 : Flowchart – LSTM

4.5. Explanation of the development process

Step 1: Importing Required Libraries

The necessary libraries were imported for data manipulation, numerical computation, data visualization and machine learning problem of Python. Pandas and NumPy libraries were used for handling data; Matplotlib and Seaborn for visualization; Scikit-learn and TensorFlow/Keras for modeling and evaluating.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

Figure 29 : DP - Importing Required Libraries

Step 2: Loading Dataset

The Open Data Nepal provided the PM2.5 air quality data set to the working space. The Kathmandu Valley dataset featured pollution measurements captured in timestamps and collected through air sensors.

```
# Read the CSV file containing Kathmandu air quality measurements into a DataFrame
df = pd.read_csv("kathmandu_pm25_dataset.csv")

print("Length of the dataset: ", len(df), "\n\n")
df.head()
```

```
Length of the dataset:  61976
```

	locationId	location	city	country	utc	local	parameter	value	unit	latitude	longitude
0	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T18:15:00+00:00	2021-03-13T00:00:00+05:45	o3	0.051	ppm	27.712463	85.315704
1	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T18:15:00+00:00	2021-03-13T00:00:00+05:45	pm25	69.000	µg/m³	27.712463	85.315704
2	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T17:15:00+00:00	2021-03-12T23:00:00+05:45	pm25	69.000	µg/m³	27.712463	85.315704
3	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T17:15:00+00:00	2021-03-12T23:00:00+05:45	o3	0.030	ppm	27.712463	85.315704
4	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T16:15:00+00:00	2021-03-12T22:00:00+05:45	o3	0.030	ppm	27.712463	85.315704

Figure 30 : DP - Loading Dataset

Step 3: Selection of Relevant Pollutant Variable

The only variable retained from the dataset of air quality variables was PM2.5. All other pollutants such as O_3 were removed. The focus of the study is to examine PM2.5 as the major pollutant of Kathmandu Valley which has documented health impacts. Through eliminating unrelated variables noise was removed and modeling was confirmed to be on-target and computationally efficient.

```
# Keep only the rows where the 'parameter' column is 'pm25'
# This removes other pollutants if they exist (e.g., o3, etc.)
pm25_df = df[df["parameter"] == "pm25"].copy()

print("Length of the dataset: ", len(pm25_df), "\n\n")
pm25_df.head()
```

```
Length of the dataset:  31832
```

	locationId	location	city	country	utc	local	parameter	value	unit	latitude	longitude
1	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T18:15:00+00:00	2021-03-13T00:00:00+05:45	pm25	69.0	µg/m³	27.712463	85.315704
2	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T17:15:00+00:00	2021-03-12T23:00:00+05:45	pm25	69.0	µg/m³	27.712463	85.315704
5	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T16:15:00+00:00	2021-03-12T22:00:00+05:45	pm25	60.0	µg/m³	27.712463	85.315704
6	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T15:15:00+00:00	2021-03-12T21:00:00+05:45	pm25	58.0	µg/m³	27.712463	85.315704
9	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12T14:15:00+00:00	2021-03-12T20:00:00+05:45	pm25	58.0	µg/m³	27.712463	85.315704

Figure 31 : DP - Selection of Relevant Pollutant Variable

Step 4: Time Conversion and Data Formatting

The fields showing local time and the UTC time stamp were converted into a common datetime format. This ensured correct timing order and helped time series analysis.

```
# Convert 'local' and 'utc' columns from string to datetime objects
# This allows proper time-based operations (resampling, filtering, etc.)
pm25_df["local"] = pd.to_datetime(pm25_df["local"])
pm25_df["utc"] = pd.to_datetime(pm25_df["utc"])
pm25_df.head()
```

Figure 32 : DP -Time Conversion and Data Formatting

Step 5: Column Pruning

Irrelevant and non-contributory columns were removed to simplify the dataset and focus the analysis on essential features.

Before Pruning:

	locationId	location	city	country	utc	local	parameter	value	unit	latitude	longitude
1	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12 18:15:00+00:00	2021-03-13 00:00:00+05:45	pm25	69.0	µg/m³	27.712463	85.315704
2	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12 17:15:00+00:00	2021-03-12 23:00:00+05:45	pm25	69.0	µg/m³	27.712463	85.315704
5	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12 16:15:00+00:00	2021-03-12 22:00:00+05:45	pm25	60.0	µg/m³	27.712463	85.315704
6	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12 15:15:00+00:00	2021-03-12 21:00:00+05:45	pm25	58.0	µg/m³	27.712463	85.315704
9	3460	US Diplomatic Post: Phora Durbar Kathmandu	Kathmandu	NP	2021-03-12 14:15:00+00:00	2021-03-12 20:00:00+05:45	pm25	58.0	µg/m³	27.712463	85.315704

Figure 33 : DP - Before Pruning

After Pruning :

```
# Remove columns that are not needed for this analysis to simplify the dataset
pm25_df = pm25_df.drop(columns = ["location","city","latitude","longitude","country"])
pm25_df.head()
```

	locationId	utc	local	parameter	value	unit
1	3460	2021-03-12 18:15:00+00:00	2021-03-13 00:00:00+05:45	pm25	69.0	µg/m³
2	3460	2021-03-12 17:15:00+00:00	2021-03-12 23:00:00+05:45	pm25	69.0	µg/m³
5	3460	2021-03-12 16:15:00+00:00	2021-03-12 22:00:00+05:45	pm25	60.0	µg/m³
6	3460	2021-03-12 15:15:00+00:00	2021-03-12 21:00:00+05:45	pm25	58.0	µg/m³
9	3460	2021-03-12 14:15:00+00:00	2021-03-12 20:00:00+05:45	pm25	58.0	µg/m³

Figure 34 : DP - After Pruning

Step 6: Exploratory Data Analysis Using Visualization

The PM2.5 concentration values are plotted using a scatter plot for exploratory data analysis. The visualization shows PM2.5 data is facing issues like negative values, highly unrealistic high values, and an abundance of outliers. The anomalies or errors detected in the sensor showed that there were errors in the linear relationship and in the data. Further cleaning and behaviour profile development will be performed in future instances.

```
# Create a large scatter plot of PM2.5 values over local time for the full time series
plt.figure(figsize=(50, 20))
plt.scatter(pm25_df["local"],pm25_df["value"])
plt.title("Spread of pm25 over the time")
plt.xlabel("Local Time")
plt.ylabel("pm25 value")
plt.show()
```

Figure 35 : DP - Exploratory Data Analysis Using Visualization CODE

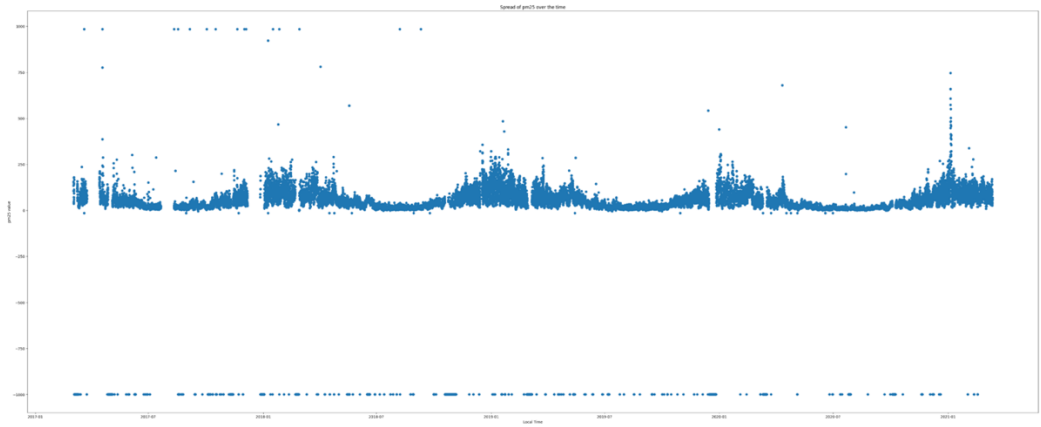


Figure 36 : Exploratory Data Analysis Using Visualization

Step 7: Counting Invalid Values

Negative and unusually high PM2.5 readings were counted to assess the number of outliers and guide the cleaning process.

```
# In reality, PM2.5 levels never drop to zero.
# Furthermore, there is no historical record of concentrations exceeding $370 µg/m³$ in Kathmandu.
# Values outside this range are likely sensor errors; therefore, we will treat them as outliers
# and use backfill (bfill) followed by forward fill (ffill) to impute the missing data.

# Count's how many values are abnormally high (>= 102)
count = (pm25_df["value"] >=102).sum()

# Why 102 and why not 100 ?
# Selected 102 as the upper threshold based on model tuning. While 100 is standard,
# 102 optimized the r^2 score, preserving valid high-pollution peaks while removing true sensor noise.

print("No of outliers/invalid data in pm25:" ,count)

# Count's how many values are zero or negative (physically impossible for concentration)
negative_pm25_values = (pm25_df["value"] <= 0).sum()
print("No of negative/invalid data in pm25:" ,negative_pm25_values)
```

```
No of outliers/invalid data in pm25: 3910
No of negative/invalid data in pm25: 2477
```

Figure 37 : DP - Counting Invalid Values

Step 8: Handling Invalid and Extreme Values

To have good quality data all physically impossible or extreme PM2.5 values were tagged as missing. Data values that fell below a minimum value or exceeded a maximum value were likely caused by sensor errors, so these were converted to NaN. This was effective as it enabled controlled imputation in later steps, preventing extreme values from impacting model training and predictions.

```
# Total invalid values before replacing them with NaN
print("Total Invalid value count before replacing:  " + str((negative_pm25_values + count)))

# Create a new column that will store cleaned PM2.5 values
pm25_df["clean_value"] = pm25_df["value"].copy()

# Replace clearly invalid values (<= 3 or >= 102) with NaN in 'clean_value'
# These NaNs will later be filled using forward/backward fill
pm25_df.loc[(pm25_df["value"] <= 3) | (pm25_df["value"] >= 102),"clean_value"] = pd.NA

# Count how many NaNs are in the cleaned column
invalid_count = pm25_df["clean_value"].isna().sum()

print("Total count of Nan value in dataframe before removing the nan:" , invalid_count)

print(pm25_df[["local","clean_value","value","parameter"]].head())
```

```
Total Invalid value count before replacing:  6387
Total count of Nan value in dataframe before removing the nan: 6692
                       local  clean_value  value parameter
1 2021-03-13 00:00:00+05:45         69.0   69.0      pm25
2 2021-03-12 23:00:00+05:45         69.0   69.0      pm25
5 2021-03-12 22:00:00+05:45         60.0   60.0      pm25
6 2021-03-12 21:00:00+05:45         58.0   58.0      pm25
9 2021-03-12 20:00:00+05:45         58.0   58.0      pm25
```

Figure 38 : DP - Handling Invalid and Extreme Values

Step 9: Creation of Sensor Downtime Attribute

To gain insights into the reliability of the PM2.5 measurements, a binary attribute named sensor_down was created. This attribute indicates whether a sensor reading was invalid or missing, with a value of 1 representing a sensor downtime (NaN or erroneous value) and 0 indicating normal operation. This feature allowed the analysis of sensor reliability over time and helped in understanding periods where data quality might affect model performance.

```
# Create a binary flag column where 1 means the sensor is "down" (invalid/NaN clean_value) and 0 means the sensor is workin
pm25_df["sensor_down"] = pm25_df["clean_value"].isna().astype(int)

print(pm25_df[["local","clean_value","value","parameter","sensor_down"]].sort_values(by="value", ascending=False).head())
```

```
                          local  clean_value  value parameter  sensor_down
55115 2017-09-05 11:00:00+05:45          NaN  985.0      pm25            1
49978 2018-01-26 12:00:00+05:45          NaN  985.0      pm25            1
48840 2018-02-27 15:00:00+05:45          NaN  985.0      pm25            1
51369 2017-12-04 10:00:00+05:45          NaN  985.0      pm25            1
51482 2017-12-01 12:00:00+05:45          NaN  985.0      pm25            1
```

Figure 39 : DP - Creation of Sensor Downtime Attribute

Step 10: Analysis and Visualization of Sensor Downtime

Summation of the sensor_down flags was taken on a monthly basis to give sensor down hours, i.e., total number of hours the sensor was down. This aggregation immediately revealed the temporal pattern of the functioning of sensors. In order to visualize the downtime, bar plots depicting the monthly downtime over the data collection period were generated and subsequently, a heatmap to visualize the intensity of downtime over years and months was generated. The visualizations were useful for identifying periods of high sensor inactivity for subsequent cleaning and modeling.

Line Plot Visualization:

Visualization Code: [Figure 66 : DP - Sensor Down Time Analysis (Line Plot Code)]

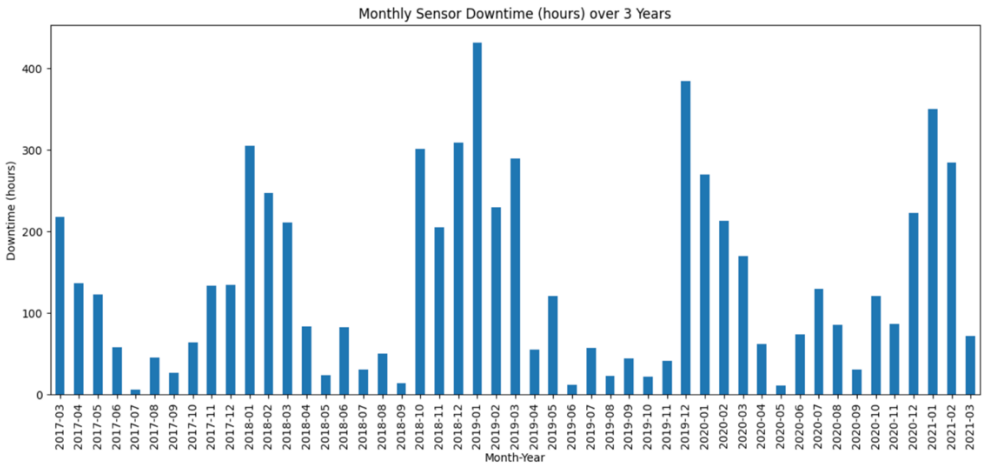


Figure 40 : DP - Analysis and Visualization of Sensor Downtime (Lineplot)

Heatmap Visualization:

Visualization Code: [Figure 65 : DP - Sensor Down Time Analysis (Heatmap Code)]

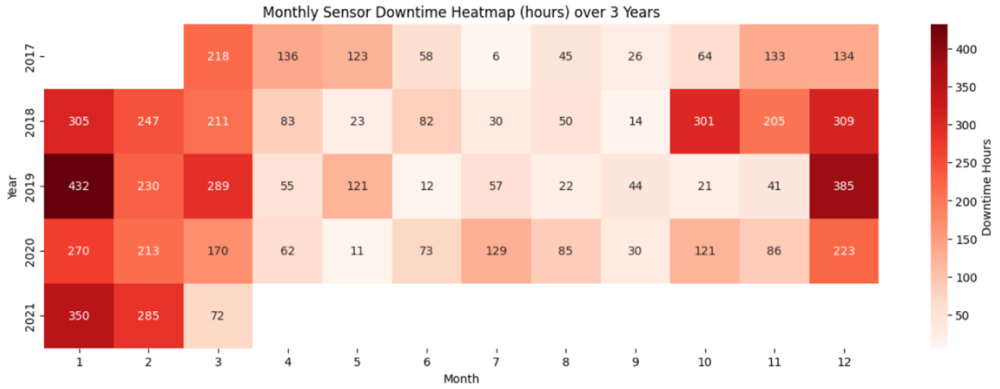


Figure 41 : DP - Analysis and Visualization of Sensor Downtime (HeatMap)

Step 11: Handling Missing Values with Forward and Backward Filling

To handle missing PM2.5 values or invalid readings, imputation of values were carried out in a controlled manner. We used backward fill (bfill) and forward fill (ffill) for missing data that lasted three hours. This method preserved the time continuity of the dataset while limiting the alteration of actual pollution patterns Values for which no technique was applicable were retained as NaN for future treatment. The values that were cleaned and imputed were stored in a new column value_filled, which was taken as a primary input for time series modelling.

```
# Fill NaNs in clean_value using backward fill (bfill) up to 3 hours, then forward fill (ffill) up to 3 hours.
pm25_df["value_filled"] = pm25_df["clean_value"].bfill(limit=3).ffill(limit=3)

# Count remaining NaNs after filling
invalid_count = pm25_df["value_filled"].isna().sum()

print("Total count of Nan value in dataframe after filling:" , invalid_count)

print(pm25_df[["local","clean_value","value","parameter","value_filled"]].head())
```

```
Total count of Nan value in dataframe after filling: 2614
                      local  clean_value  value parameter  value_filled
1 2021-03-13 00:00:00+05:45         69.0   69.0      pm25          69.0
2 2021-03-12 23:00:00+05:45         69.0   69.0      pm25          69.0
5 2021-03-12 22:00:00+05:45         60.0   60.0      pm25          60.0
6 2021-03-12 21:00:00+05:45         58.0   58.0      pm25          58.0
9 2021-03-12 20:00:00+05:45         58.0   58.0      pm25          58.0
```

Figure 42 : DP - Handling Missing Values with Forward and Backward Filling

Step 12: Time-Series Feature Engineering

The dataset was examined for consecutive timestamps separated by exactly one hour, indicating whether a gap existed in the time series. After assuring the temporal integrity, lag features were created out of PM2.5 values from the last 1, 2, and 3 hours. These lag-based features captured temporal dependencies and enabled models to learn how prior pollution levels affected future concentrations. Excluded from the study were the entries with missing lag values as well as those sequences which did not have continuous hourly measurements.

```
# Create lag features: PM2.5 from the previous 1, 2, 3, and (duplicated) 3 hours
# These will be used as inputs to prediction models
pm25_df['lag1'] = pm25_df['value_filled'].shift(1)
pm25_df['lag2'] = pm25_df['value_filled'].shift(2)
pm25_df['lag3'] = pm25_df['value_filled'].shift(3)
pm25_df['lag4'] = pm25_df['value_filled'].shift(3)


print("Before dropping the NaN from the lag features: \n")
print(pm25_df[["local","value_filled","lag1","lag2","lag3"]].head(10))


# Remove rows where lag features or target are NaN (these occur at the start of the series)
print("\n\nAfter dropping the NaN from the lag features: \n")
pm25_df = pm25_df.dropna(subset=["lag1","lag2","lag3","value_filled"])

# Keep only rows where the last 3 lags are truly 1, 2, and 3 hours apart
pm25_df = pm25_df[
    (pm25_df['local'] - pm25_df['local'].shift(1) == pd.Timedelta(hours=1)) &
    (pm25_df['local'] - pm25_df['local'].shift(2) == pd.Timedelta(hours=2)) &
    (pm25_df['local'] - pm25_df['local'].shift(3) == pd.Timedelta(hours=3))
]
```

Figure 43 : DP - Time-Series Feature Engineering

```
Before dropping the NaN from the lag features:

                       local  value_filled  lag1  lag2  lag3
0 2017-03-03 05:00:00+05:45           NaN   NaN   NaN   NaN
1 2017-03-03 06:00:00+05:45           NaN   NaN   NaN   NaN
2 2017-03-03 07:00:00+05:45           NaN   NaN   NaN   NaN
3 2017-03-03 08:00:00+05:45           NaN   NaN   NaN   NaN
4 2017-03-03 09:00:00+05:45          72.1   NaN   NaN   NaN
5 2017-03-03 10:00:00+05:45          72.1  72.1   NaN   NaN
6 2017-03-03 11:00:00+05:45          72.1  72.1  72.1   NaN
7 2017-03-03 12:00:00+05:45          72.1  72.1  72.1  72.1
8 2017-03-03 13:00:00+05:45          48.0  72.1  72.1  72.1
9 2017-03-03 14:00:00+05:45          40.0  48.0  72.1  72.1


After dropping the NaN from the lag features:

                       local  value_filled  lag1  lag2  lag3
10 2017-03-03 15:00:00+05:45          36.0  40.0  48.0  72.1
11 2017-03-03 16:00:00+05:45          48.5  36.0  40.0  48.0
12 2017-03-03 17:00:00+05:45          41.7  48.5  36.0  40.0
13 2017-03-03 18:00:00+05:45          42.3  41.7  48.5  36.0
14 2017-03-03 19:00:00+05:45          59.6  42.3  41.7  48.5
15 2017-03-03 20:00:00+05:45          70.3  59.6  42.3  41.7
16 2017-03-03 21:00:00+05:45          70.3  70.3  59.6  42.3
17 2017-03-03 22:00:00+05:45          70.3  70.3  70.3  59.6
18 2017-03-03 23:00:00+05:45          70.3  70.3  70.3  70.3
19 2017-03-04 00:00:00+05:45          92.8  70.3  70.3  70.3
```

Figure 44 : DP - Time-Series Feature Engineering (II)

Step 13: Visualization of Time-Series Patterns

A scatter plot of the cleaned PM2.5 time series was prepared for visual inspection. Using a line chart enabled us to verify the timing structure of the data after preprocessing. Plus, it gave some early insights into short-term variations and seasonal patterns.

```
# Plot the time series using the filled values instead of raw values
plt.figure(figsize=(50,20))
plt.scatter(pm25_df["local"],pm25_df["value_filled"])
plt.title("Spread of pm25 over the time")
plt.xlabel("Local Time")
plt.ylabel("pm25 value")
plt.show()
```

Figure 45 : DP - Visualization of Time-Series Patterns Code

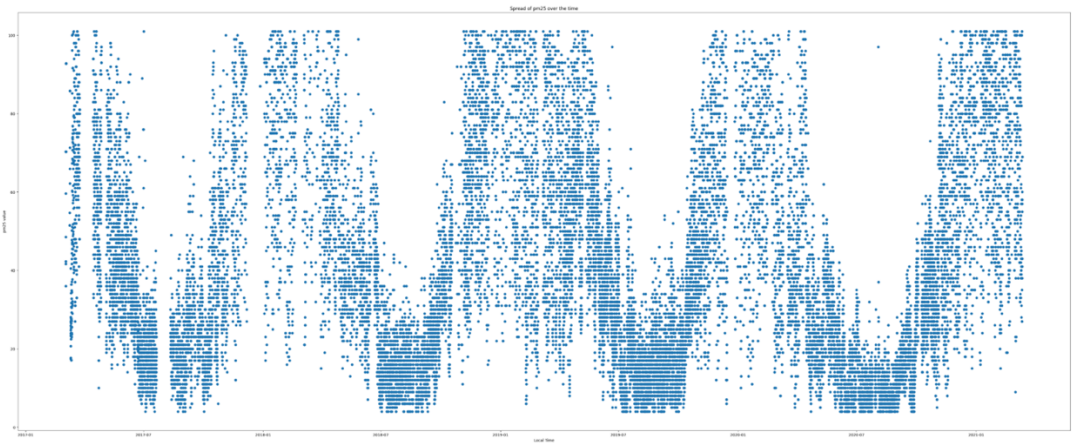


Figure 46 : DP - Visualization of Time-Series Patterns

Step 14: Dataset Splitting

The cleaned dataset was divided into training and testing sets to enable unbiased model evaluation. The training dataset contained the pm25 reading from 2017 to 2020 while the test dataset contain data value from 2020 to 2021.

```
# Split data into training (before 2020) and testing (after 2020-01-01)
train = pm25_df[pm25_df["local"] < "2020-01-01"]
test = pm25_df[pm25_df["local"] > "2020-01-01"]

# Features: lagged values; Target: current PM2.5
X_train = train[['lag1', 'lag2', 'lag3']]
y_train = train['value_filled']

X_test = test[['lag1', 'lag2', 'lag3']]
y_test = test['value_filled']
```

Figure 47 : DP - Dataset Splitting

Step 15: Model Training

We implemented several machine learning models. We utilized Linear Regression as our baseline model, Random Forest Regression to capture non-linear relationships, and Long Short-Term Memory (LSTM) networks for sequential temporal essence.

Each model was trained using historical PM2.5 data and optimized to learn relevant patterns from the training dataset.

For Linear Regression:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
import math

# Initialize a simple linear regression model
linear_model = LinearRegression()

# Fit the model using lag features as predictors
linear_model.fit(X_train,y_train)

# Predict PM2.5 values on the test set
lg__y_pred = linear_model.predict(X_test)

# Calculate evaluation metrics: MSE, MAE, R², and RMSE
lg_mse = mean_squared_error(y_test, lg__y_pred)
lg_mae = mean_absolute_error(y_test, lg__y_pred)
lg_r2 = r2_score(y_test, lg__y_pred)
lg_rmse = math.sqrt(lg_mse)

print("\n" + "╔═══════════════════════════════════════════╗")
print( "║   LINEAR REGRESSION EVALUATION METRICS    ║")
print( "╠═══════════════════════════════════════════╣")
print(f"║   RMSE: {lg_rmse:>30.6f}    ║")
print( "╠═══════════════════════════════════════════╣")
print(f"║   MSE:  {lg_mse:>30.6f}    ║")
print( "╠═══════════════════════════════════════════╣")
print(f"║   MAE:  {lg_mae:>30.6f}    ║")
print( "╠═══════════════════════════════════════════╣")
print(f"║   R²:   {lg_r2:>30.6f}    ║")
print( "╚═══════════════════════════════════════════╝")
```

Figure 48 : DP - Model Training (Linear Regression)

For Random Forest Regression:

```
from sklearn.ensemble import RandomForestRegressor

# Configure a Random Forest for regression
rf = RandomForestRegressor(
    n_estimators=300,
    max_depth=12,
    min_samples_leaf=5,
    random_state=42,
    n_jobs=-1
)
# Train the Random Forest model
rf.fit(X_train, y_train)

# Predict on the test set
rf__y_pred = rf.predict(X_test)

# Compute metrics for the Random Forest model
rf_mse = mean_squared_error(y_test, rf__y_pred)
rf_rmse = np.sqrt(rf_mse)
rf_r2 = r2_score(y_test, rf__y_pred)
rf_mae = mean_absolute_error(y_test, rf__y_pred)


print("\n" + "╔═══════════════════════════════════════════════╗")
print("║     RANDOM FOREST EVALUATION METRICS     ║")
print("╠═══════════════════════════════════════════════╣")
print(f"║  RMSE: {rf_rmse:>30.6f}    ║")
print("╠═══════════════════════════════════════════════╣")
print(f"║  MSE:  {rf_mse:>30.6f}    ║")
print("╠═══════════════════════════════════════════════╣")
print(f"║  MAE:  {rf_mae:>30.6f}    ║")
print("╠═══════════════════════════════════════════════╣")
print(f"║  R²:   {rf_r2:>30.6f}    ║")
print("╚═══════════════════════════════════════════════╝")
```

Figure 49 : DP - Model Training (Random Forest Regression)

For Long Term Short Memory (LSTM):

To implement the LSTM model, we needed to scale the data in which we used the minmaxscaler method from sklearn.

Scaling using MinMaxScaler :

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))

X_train_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], 3, 1))
X_test_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], 3, 1))
```

Figure 50 : DP - Model Training - Scaling using MinMaxScaler (LSTM)

Implementing Model:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([
    LSTM(64, input_shape=(3, 1)),
    Dense(1)
])

model.compile(
    optimizer="adam",
    loss="mse"
)

model.summary()
```

Figure 51 : DP - Model Training - Implementing Model (LSTM)

```
/Users/dashivam06/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urlli
  warnings.warn(
/Users/dashivam06/Library/Python/3.9/lib/python/site-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do
  super().__init__(**kwargs)

Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
dense (Dense)	(None, 1)	65

```
Total params: 16,961 (66.25 KB)

Trainable params: 16,961 (66.25 KB)

Non-trainable params: 0 (0.00 B)
```

Figure 52 : DP - Model Training - Model Running (LSTM)

LSTM Model Training :

```python
history = model.fit(
    X_train_lstm,
    y_train_scaled,
    epochs=7,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)

y_pred_scaled = model.predict(X_test_lstm)

# Convert back to original scale
lstm__y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_test_original = scaler_y.inverse_transform(y_test_scaled)


lstm_mse = mean_squared_error(y_test_original, lstm__y_pred)
lstm_rmse = np.sqrt(lstm_mse)
lstm_mae = mean_absolute_error(y_test_original, lstm__y_pred)
lstm_r2 = r2_score(y_test_original, lstm__y_pred)


print("\n" + "╔═══════════════════════════════════════╗")
print("║      LSTM MODEL EVALUATION METRICS    ║")
print("╠═══════════════════════════════════════╣")
print(f"║  RMSE: {lstm_rmse:>30.6f}    ║")
print("╠═══════════════════════════════════════╣")
print(f"║  MSE:  {lstm_mse:>30.6f}    ║")
print("╠═══════════════════════════════════════╣")
print(f"║  MAE:  {lstm_mae:>30.6f}    ║")
print("╠═══════════════════════════════════════╣")
print(f"║  R²:   {lstm_r2:>30.6f}    ║")
print("╚═══════════════════════════════════════╝")
```

Figure 53 : DP - Model Training (LSTM)

```
Epoch 1/7
423/423 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0119 - val_loss: 0.0090
Epoch 2/7
423/423 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0118 - val_loss: 0.0092
Epoch 3/7
423/423 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0119 - val_loss: 0.0089
Epoch 4/7
423/423 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0121 - val_loss: 0.0089
Epoch 5/7
423/423 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0123 - val_loss: 0.0089
Epoch 6/7
423/423 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0118 - val_loss: 0.0089
Epoch 7/7
423/423 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0115 - val_loss: 0.0089
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 463us/step
```

Figure 54 : DP - Model Training - Model Calculating (LSTM)

Step 18: Model Evaluation

Model performance was evaluated using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the R^2 score to assess prediction accuracy and reliability.

4.6. Achieved Results

4.6.1. Model Performance Evaluation

This section evaluates the performance of three predictive models .i.e. Linear Regression, Random Forest Regression, and LSTM for forecasting PM2.5 concentrations one hour ahead using the previous three hours of data.

- Visualization Code : [: Appendix - AR - Model Performance Evaluation Code]

- Visualization :

Figure 55 : AR - Model Performance Evaluation

Explanation of the Figure :

This figure presents a **model performance comparison** for predicting PM2.5 levels, using the past three hours of data to forecast the next hour. The metrics used for the evaluation of different models are RMSE, MAE and R2 Score.

RMSE Analysis :

RMSE measures the average magnitude of prediction errors, with lower values indicating better accuracy. The **LSTM** model performs best with an RMSE of 9.561, followed by Linear Regression (9.667) and Random Forest (9.757).

MAE Analysis :

MAE represents the average absolute difference between predicted and actual values. Again, **LSTM** shows the smallest average error (5.938), with Linear Regression (5.955) and Random Forest (6.258) slightly higher.

R^2 Score Analysis :

R^2 Score indicates how well the model explains the variance in the data, with values closer to 1 being better. **LSTM** leads with an R^2 of 0.894, followed by Linear Regression (0.892) and Random Forest (0.890).

Overall Interpretation:

Across all evaluation metrics, the LSTM model demonstrates marginally superior performance. However, the relatively small differences among the models indicate that all three approaches are reasonably effective for short-term PM2.5 prediction.

A detailed, model-wise breakdown of all evaluation metric results is provided in the Appendix. Link: [: Appendix - Every Model Evaluation Metrics *Results*]

4.6.2. Actual vs Predicted Value Analysis

This section presents segments of approximately 400 consecutive time steps where the actual PM2.5 measurements are compared against the predicted values from different models (e.g., Linear Regression and Random Forest) for Kathmandu.

4.6.2.1. Linear Regression

- Visualization Code: [: Appendix - Actual vs Predicted Value Analysis - Linear Regression *Code*]

- Visualization:

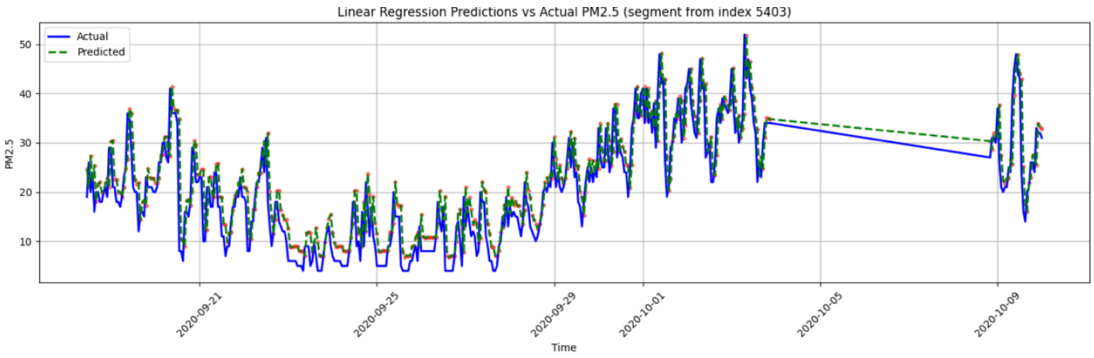


Figure 56 : Actual vs Predicted Value Analysis - Linear Regression

- Explanation of the Figure:
  This chart shows time ordered PM2.5 readings in Kathmandu, with blue representing actual values and green dashed representing linear regression predictions. The close overlap of the two lines shows the model tracks the real PM2.5 pattern well, with some divergence toward the right where predictions and actual values start to differ more noticeably.

4.6.2.2. Random Forest Regression

- Visualization Code: [: Appendix - Actual vs Predicted Value Analysis - Random Forest *Code*]

- Visualization:

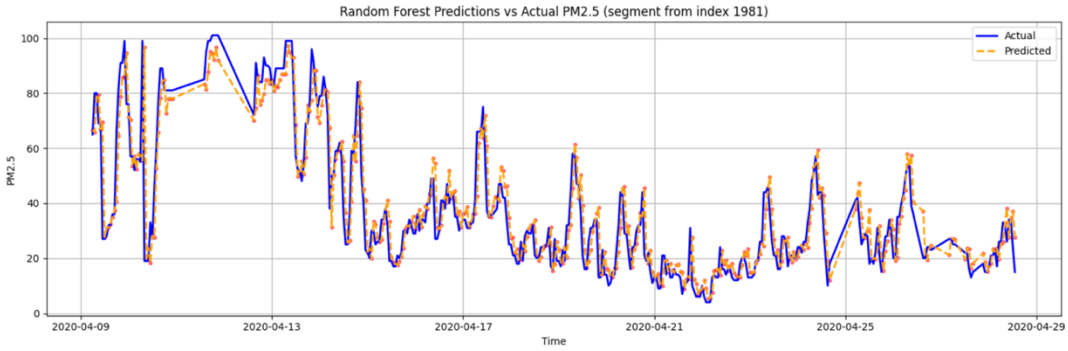


Figure 57 : Actual vs Predicted Value Analysis - Random Forest

- Explanation of the Figure:

  This chart shows time ordered PM2.5 readings, comparing actual values (solid blue line) with Random Forest model predictions (dashed orange line with markers). The two lines overlap very closely, indicating that the Random Forest model predicts Kathmandu's PM2.5 levels with high accuracy over this period.

4.6.2.3. LSTM Model

- Visualization Code: [*:* Appendix - Actual vs Predicted Value Analysis - LSTM Code]

- Visualization:

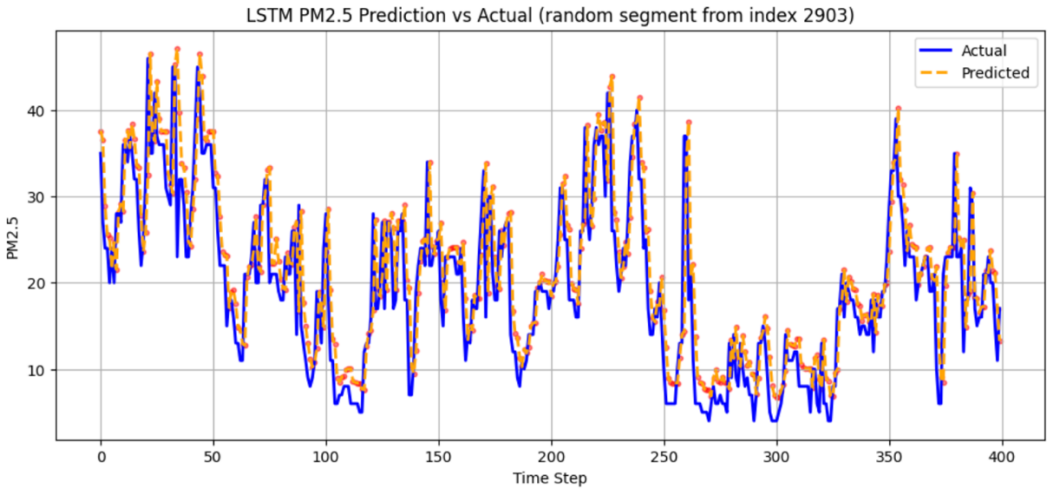


Figure 58 : Actual vs Predicted Value Analysis - LSTM

- Explanation of the Figure:

  This plot shows the real PM2.5 data (solid blue) relative to the LSTM prediction (dashed orange with markers) at a continuous series of time steps. The heavy overlap between the two curves indicates that, the LSTM records the peaks and troughs in Kathmandu PM 2.5 concentration with high fidelity.

4.6.3. Exploratory Visualization and Temporal Dynamics of PM2.5

4.6.3.1 Hour-wise Variability and Prediction Uncertainty

This subsection analyzes the variability of PM2.5 concentrations across different hours of the day to identify time periods that are more challenging for accurate prediction

- Visualization Code: [: Appendix - Exploratory Visualization - Hour wise Variability and Prediction Uncertainty]

- Visualization :

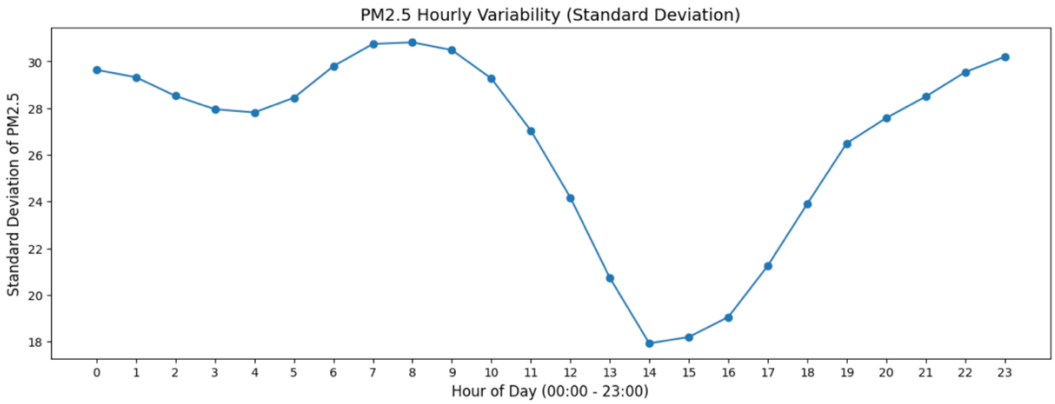


Figure 59 : Exploratory Visualization - Hour wise Variability and Prediction Uncertainty

- Explanation of the Figure :

It was found that the variability of PM2.5 is minimal at the time of the day when it is 14:00 (2 PM) which means that pollution rates are the same and can be predicted better. On the contrary, the variability increases in the evening and early-morning hours, specifically at approximately 00:00, 09:00, and 23:00. These seasons have higher swings in the PM2.5 levels and the short time prediction has been made more challenging.

4.6.3.2. Long-Term Temporal Trends in PM2.5 Concentration

This analysis examines historical PM2.5 concentration patterns across multiple years to identify long-term trends and seasonal behavior.

- Visualization Code: [: Appendix - Exploratory Visualization - Long-Term Temporal Trends in PM2.5 Concentration]

- Visualization:

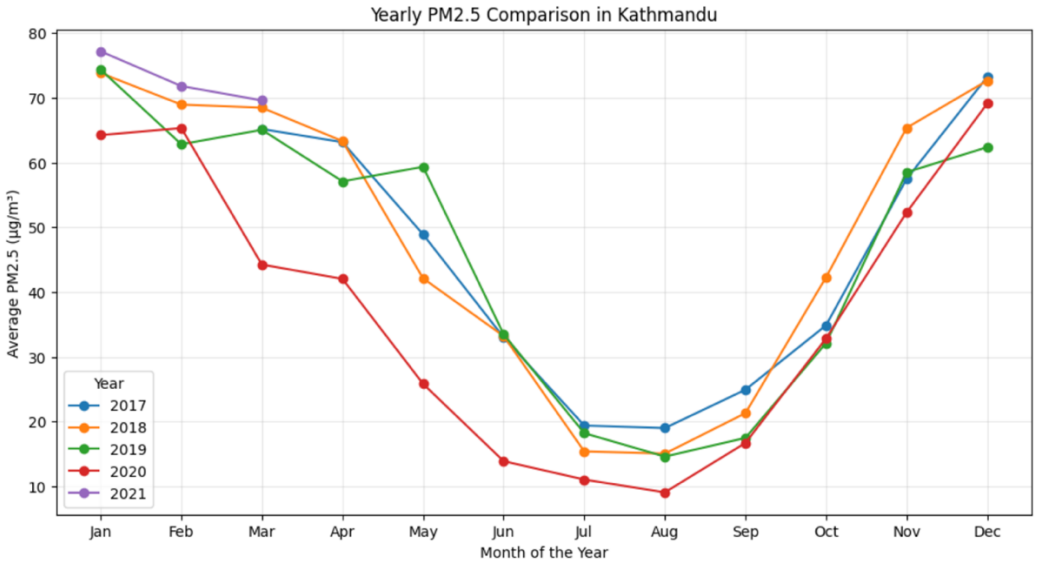


Figure 60 : Exploratory Visualization - Long-Term Temporal Trends in PM2.5 Concentration

- Explanation of the Figure:

This figure compares the average monthly PM2.5 levels in Kathmandu from 2017 to 2021, showing a clear seasonal cycle where pollution peaks in the winter months (November–January) and drops to its lowest levels during the monsoon season (June–August). During the rainy months, frequent precipitation helps wash out particulate matter from the atmosphere, leading to noticeably cleaner air, whereas the dry, stable winter conditions allow pollutants to accumulate, reflecting persistent air quality challenges across all years.

4.6.4. Myth Buster Section: Common Assumptions about Air Pollution

4.6.4.1 Was There a Significant Decline in Air Pollution in Kathmandu During COVID-19 period?

Answer:

The analysis clearly shows that PM2.5 concentrations during the COVID-era year were significantly lower compared to non-COVID years. This reduction reflects the impact of decreased vehicular movement, industrial activity, and overall human mobility during lockdown periods.

- Visualization Code: [: Appendix - Exploratory Visualization - Long-Term Temporal Trends in PM2.5 Concentration]
- Visualization:

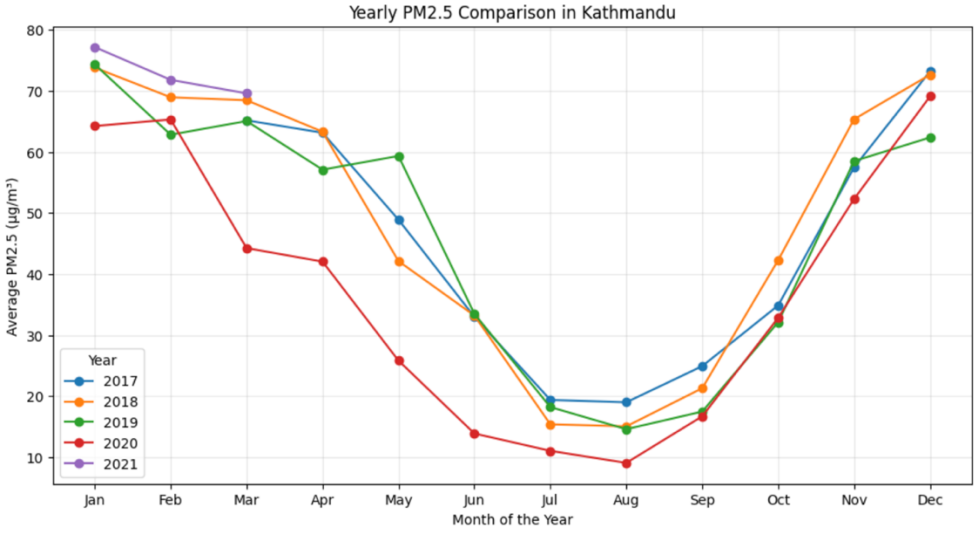


Figure 61 : Exploratory Visualization - PM25 Concentrations over the Covid Period

Conclusion:

The COVID-era year exhibits significantly lower PM2.5 concentrations, highlighting the substantial influence of reduced human activity on air quality.

4.6.4.2 Is PM2.5 Significantly Lower on Weekends Compared to Weekdays?

Answer:

Contrary to common belief, the difference between weekday and weekend PM2.5 concentrations is minimal. The mean PM2.5 level during weekdays is 44.91, while weekends exhibit a comparable value of 44.37. The box plot visualization further confirms substantial overlap between the two distributions, indicating no statistically meaningful separation between weekday and weekend pollution levels.

- Visualization Code: [: Appendix - Exploratory Visualization – Weekend vs Weekday Daily PM2.5]
- Visualization:

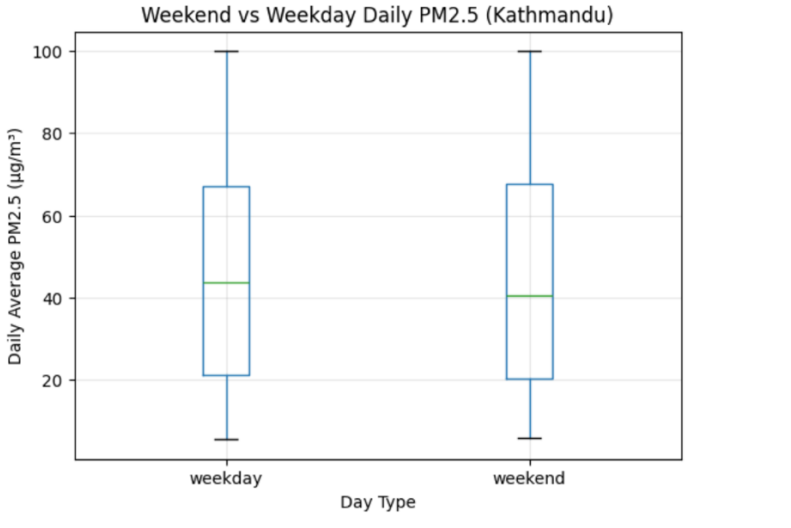


Figure 62 : Exploratory Visualization – Weekend vs Weekday Daily PM2.5

Conclusion:

This analysis disproves the common assumption that weekend pollution levels are substantially lower in Kathmandu. One possible explanation is that PM2.5 particles persist in the atmosphere for extended periods and do not dissipate immediately with short-term reductions in emission sources. As a result, pollution accumulated over preceding days may continue to influence air quality during weekends, leading to similar concentration levels across the week.

5. Conclusion

5.1. Analysis of the Work Done

This project focused on short-term PM2.5 prediction in Kathmandu using historical air quality data and machine learning techniques. These considered filtering of PM2.5 observations, converting time stamps, dropping redundant columns, finding sensor errors, and controlled forward filling and backward filling of missing reading while not breaking up the hourly continuity of observations.

Scatter plots were used to explore pollutant behavior, including time-series analysis, outlier detection, and sensor failure detection. Bar charts were used to compare model performances. Further exploratory visualizations were used to investigate model performance, including hourly PM2.5 variation, long-term PM2.5 trajectories, and prediction errors. Myth-buster analyzes were also conducted to determine whether air quality decreased during the COVID-19 pandemic period and whether the weekend PM2.5 concentrations were statistically considerably lower than those on weekdays.

The target was predicted based on the last three hours of lag features and three different models, Linear Regression, Random Forest and LSTM have been trained. The models were evaluated with RMSE, MAE and R^2 score and the overview of the results are visualized by bar charts. The models' performance was qualitatively assessed with a plot of measured versus predicted PM2.5 concentrations for a random sample of 400 data points.

Overall, the LSTM model performed best with the lowest RMSE of 9.561, MAE of 5.938, and R^2 of 0.894 in total. Linear Regression and Random Forest resulted in similar values. Thus, all three machine learning models can be used for the short-term prediction of PM2.5 concentrations. Different evaluation metrics are shown in detail in the Appendix.

5.2. How the Solution Addresses Real-World Problems

The proposed solution addresses real problems related to tracking air pollution and protecting public health. By providing individuals with accurate short-term forecasts of PM2.5 levels, the approach allows them to implement preventive actions and reduce health risks during periods of high pollution. These forecasts can aid policymakers and health authorities in preparing emergency responses, issuing advisories, and implementing temporary measures such as traffic control and industrial regulation during phases of severe pollution.

Additionally, this study demonstrates the use of machine learning to noisy, incomplete, and irregular environmental data from the actual world. This strategy strengthens urban administration's capacity to control air quality and promotes evidence-based decision making. Since air pollution is increasing quickly in Kathmandu, these prediction methods can help with long-term environmental planning and public awareness.

5.3. Further Work

The application can be further enhanced with the integration of meteorological data such as rainfall, windspeed, temperature humidity that strongly affect air pollution dispersion and patterns of seasonal air quality and incorporating weather variables that would help the model to acclimatize to seasonal change and sudden pollution episodes.

It can also be enhanced by not just concentrating on PM2.5 but also on other pollutants that are harmful like nitrogen dioxide and sulfur dioxide and also by extending the prediction from 24hrs to 72 hrs but concentrating on the forecast now that will not only enhance the usefulness but would also help the hospitals, schools and even organizers as well as the city authorities to plan better during the peak periods of the year.

We can create a convenient application which can be hosted in either mobile or web that can provide the population with real time air quality forecasts, health warnings and alerts. The automated alert system to government agencies can also be implemented by us that would assist in supporting emergency pollution control and also include traffic data including industrial activity logs and construction information to determine the source of pollution to institute mitigation measures.

Furthermore, we can use modern techniques such as transfer learning could be explored
and adapt the trained models to other Nepali cities like Pokhara or Biratnagar which would need minimum retraining. We can develop it further by using continuous validation using new sensor data, collaboration with local institutions and integration with modern tools which can contribute to sustainable development and a healthy city.

6. References

GeeksForGeeks, 2026. *Regression in machine learning - GeeksForGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/machine-learning/regression-in-machine-learning/

IBM, 2026. *What is a recurrent neural network (RNN)? | IBM.* [Online]
Available at: https://www.ibm.com/think/topics/recurrent-neural-networks

IBM, 2026. *What is random forest? | IBM.* [Online]
Available at: https://www.ibm.com/think/topics/random-forest

Analytics Vidya , 2026. *Linear Regression in Machine Learning.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/

GeeksForGeeks, 2026. *What is LSTM - Long Short Term Memory? | GeeksForGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/

Analytics Vidhya, 2026. *Linear Regression in Machine Learning.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/

StatisticsByJim, 2026. *Root Mean Square Error (RMSE) - StatisticsByJim.* [Online]
Available at: https://statisticsbyjim.com/regression/root-mean-square-error-rmse/

Statistics By Jim, 2026. *Mean Absolute Error [MAE] Statistics By Jim.* [Online]
Available at: https://statisticsbyjim.com/glossary/mean-absolute-error/

Statistics By Jim, 2026. *R-squared - Statistics By Jim.* [Online]
Available at: https://statisticsbyjim.com/glossary/r-squared/

Statistics By Jim, 2026. *Mean Squared Error (MSE) - Statistics By Jim.* [Online]
Available at: https://statisticsbyjim.com/regression/mean-squared-error-mse/

7. Appendix

7.1. Achieved Results Section Images:

a) Every Model Evaluation Metrics Results

LINEAR REGRESSION EVALUATION METRICS	
RMSE:	9.666691
MSE:	93.444918
MAE:	5.954900
R^2:	0.891906

RANDOM FOREST EVALUATION METRICS	
RMSE:	9.756542
MSE:	95.190104
MAE:	6.257682
R^2:	0.889887

LSTM MODEL EVALUATION METRICS	
RMSE:	9.593418
MSE:	92.033663
MAE:	6.181459
R^2:	0.893538

Figure 63 : Appendix - Every Model Evaluation Metrics Results

b) AR - Model Performance Evaluation Code

```
models = ["Linear", "Random Forest", "LSTM"]

rmse = [lg_rmse, rf_rmse, lstm_rmse]
mae  = [lg_mae, rf_mae, lstm_mae]
r2   = [lg_r2, rf_r2, lstm_r2]

fig, axes = plt.subplots(1, 3, figsize=(16,5))

def annotate_bars(ax, bars, decimals=3):
    """Add value labels on top of each bar"""
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2,
                height,
                f"{height:.{decimals}f}",
                ha='center', va='bottom', fontsize=10)

# RMSE
bars_rmse = axes[0].bar(models, rmse, color='#1f77b4')  # blue
axes[0].set_title("RMSE")
axes[0].set_ylabel("Error")
annotate_bars(axes[0], bars_rmse)

# MAE
bars_mae = axes[1].bar(models, mae, color='#ff7f0e')  # orange
axes[1].set_title("MAE")
annotate_bars(axes[1], bars_mae)

# R2
bars_r2 = axes[2].bar(models, r2, color='#2ca02c')  # green
axes[2].set_title("R² Score")
axes[2].set_ylim(0.8, 1.0)
annotate_bars(axes[2], bars_r2)

fig.suptitle("Model Performance Comparison", fontsize=14)
plt.tight_layout()
plt.show()
```

Figure 64 : Appendix - AR - Model Performance Evaluation Code

7.2. Explanation of the development Process

a) Sensor Down Time Analysis (Heatmap Code)

```
# Convert PeriodIndex back to a normal column for further processing
monthly_downtime = monthly_downtime.reset_index()
monthly_downtime["Year"] = monthly_downtime["local"].dt.year
monthly_downtime["Month"] = monthly_downtime["local"].dt.month

# Create a pivot table: rows = Year, columns = Month, values = downtime hours
heatmap_data = monthly_downtime.pivot(index="Year", columns="Month", values="sensor_down")


# Plot a heatmap to visualize downtime intensity by year and month
plt.figure(figsize=(16,5))
sns.heatmap(heatmap_data, annot=True, fmt=".0f", cmap="Reds", cbar_kws={'label': 'Downtime Hours'})
plt.title("Monthly Sensor Downtime Heatmap (hours) over 3 Years")
plt.xlabel("Month")
plt.ylabel("Year")
plt.show()
```

Figure 65 : DP - Sensor Down Time Analysis (Heatmap Code)

b) Sensor Down Time Analysis (Line Plot Code)

```
# Group by month (using the 'local' datetime) and sum the sensor_down flags
# Each row represents one hour, so the sum is the total hours of downtime per month
monthly_downtime = pm25_df.groupby(pm25_df["local"].dt.to_period("M"))["sensor_down"].sum()

monthly_downtime.head()

# Plot monthly downtime as a bar chart
plt.figure(figsize=(15,6))
monthly_downtime.plot(kind="bar")
plt.title("Monthly Sensor Downtime (hours) over 3 Years")
plt.xlabel("Month-Year")
plt.ylabel("Downtime (hours)")
plt.show()
```

Figure 66 : DP - Sensor Down Time Analysis (Line Plot Code)

7.3. Actual vs Predicted Value Analysis

a) Actual vs Predicted Value Analysis – Linear Regression Code

```
# Compare actual vs predicted PM2.5 over time for the test period

import matplotlib.pyplot as plt
import numpy as np

# Number of points to visualize
n_points = 400  # adjust as needed

# Random starting index
start_idx = np.random.randint(0, len(y_test) - n_points)

# Select segment
time_segment = test['local'].iloc[start_idx:start_idx + n_points]
y_true_segment = y_test[start_idx:start_idx + n_points]
lg_pred_segment = lg__y_pred[start_idx:start_idx + n_points]

plt.figure(figsize=(15,5))
plt.plot(time_segment, y_true_segment, label='Actual', color='blue', linewidth=2)
plt.plot(time_segment, lg_pred_segment, label='Predicted', color='green', linestyle='--', linewidth=2)
plt.scatter(time_segment, lg_pred_segment, color='red', s=10, alpha=0.5)  # optional scatter
plt.xlabel("Time")
plt.ylabel("PM2.5")
plt.title(f"Linear Regression Predictions vs Actual PM2.5 (segment from index {start_idx})")
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 67 : Appendix - Actual vs Predicted Value Analysis - Linear Regression Code

b) Actual vs Predicted Value Analysis - Random Forest Code

```
# Visual comparison of Random Forest predictions vs actual PM2.5 on the test period
import matplotlib.pyplot as plt
import numpy as np

# Number of points to visualize
n_points = 400

# Random starting index
start_idx = np.random.randint(0, len(y_test) - n_points)

# Select segment
time_segment = test['local'].iloc[start_idx:start_idx + n_points]
y_true_segment = y_test[start_idx:start_idx + n_points]
rf_pred_segment = rf__y_pred[start_idx:start_idx + n_points]

plt.figure(figsize=(15,5))
plt.plot(time_segment, y_true_segment, label='Actual', color='blue', linewidth=2)
plt.plot(time_segment, rf_pred_segment, label='Predicted', color='orange', linestyle='--', linewidth=2)
plt.scatter(time_segment, rf_pred_segment, color='red', s=10, alpha=0.5)  # optional scatter
plt.xlabel("Time")
plt.ylabel("PM2.5")
plt.title(f"Random Forest Predictions vs Actual PM2.5 (segment from index {start_idx})")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Figure 68 : Appendix - Actual vs Predicted Value Analysis - Random Forest Code

c) Actual vs Predicted Value Analysis - LSTM Code

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

# Number of points to visualize
n_points = 400  # length of segment to plot

# Random starting index
start_idx = np.random.randint(0, len(y_test_original) - n_points)

y_true = y_test_original[start_idx:start_idx + n_points]
y_pred = lstm__y_pred[start_idx:start_idx + n_points]

plt.figure(figsize=(12,5))
plt.plot(y_true, label='Actual', color='blue', linewidth=2)
plt.plot(y_pred, label='Predicted', color='orange', linestyle='--', linewidth=2)
plt.scatter(range(n_points), y_pred, color='red', s=10, alpha=0.5)  # scatter to highlight predictions
plt.title(f"LSTM PM2.5 Prediction vs Actual (random segment from index {start_idx})")
plt.xlabel("Time Step")
plt.ylabel("PM2.5")
plt.legend()
plt.grid(True)
plt.show()
```

Figure 69 : Appendix - Actual vs Predicted Value Analysis - LSTM Code

7.4. Exploratory Visualization and Temporal Dynamics of PM2.5

a) Hour wise Variability and Prediction Uncertainty

```
# Question : Which hours are most unpredictable?

pm25_df_hourly_var = pm25_df.groupby(pm25_df["local"].dt.hour)["value_filled"].std()
ax =  pm25_df_hourly_var.plot(kind='line', marker='o', figsize=(15, 5))
ax.set_title("PM2.5 Hourly Variability (Standard Deviation)", fontsize=14)
ax.set_xlabel("Hour of Day (00:00 - 23:00)", fontsize=12)
ax.set_ylabel("Standard Deviation of PM2.5", fontsize=12)
ax.set_xticks(range(0, 24))
plt.show()
```

Figure 70 : Appendix - Exploratory Visualization - Hour wise Variability and Prediction Uncertainty Code

7.5. Myth Buster Section: Common Assumptions about Air Pollution

a) Long-Term Temporal Trends in PM2.5 Concentration

```
# Add year and month columns for longer-term trend analysis
pm25_df['year'] = pm25_df['local'].dt.year
pm25_df['month'] = pm25_df['local'].dt.month

# Compute monthly average PM2.5 for each year
yearly_trend = pm25_df.groupby(['year', 'month'])['value_filled'].mean().unstack(level=0)

# Plot yearly monthly curves to compare seasons across years
plt.figure(figsize=(12, 6))
yearly_trend.plot(ax=plt.gca(), marker='o')
plt.title("Yearly PM2.5 Comparison in Kathmandu")
plt.xlabel("Month of the Year")
plt.ylabel("Average PM2.5 (µg/m³)")
plt.xticks(
    range(1, 13),
    ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
)
plt.legend(title="Year")
plt.grid(True, alpha=0.3)
plt.show()
```

Figure 71 : Appendix - Exploratory Visualization - Long-Term Temporal Trends in PM2.5 Concentration Code

b) Weekend vs Weekday Daily PM2.5

```
# Myth Buster : Is PM2.5 in Kathmandu significantly lower on weekends (Saturday) compared to weekdays?
pm25_df["day_type"] = pm25_df["local"].dt.dayofweek.apply(
    lambda x: "weekend" if x == 6 else "weekday"
)
daily_pm25 = (
    pm25_df
    .groupby([pm25_df["local"].dt.date, "day_type"])["value_filled"]
    .mean()
    .reset_index()
)

print("--- Daily PM2.5 Means ---")
print(daily_pm25.groupby("day_type")["value_filled"].mean())
print("---------------------------\n")


plt.figure(figsize=(6,5))
daily_pm25.boxplot(column="value_filled", by="day_type")

plt.title("Weekend vs Weekday Daily PM2.5 (Kathmandu)")
plt.suptitle("")
plt.xlabel("Day Type")
plt.ylabel("Daily Average PM2.5 (µg/m³)")
plt.grid(alpha=0.3)
plt.show()
```

```
--- Daily PM2.5 Means ---
day_type
weekday    44.910679
weekend    44.376522
Name: value_filled, dtype: float64
---------------------------
```

Figure 72 : Appendix - Exploratory Visualization – Weekend vs Weekday Daily PM2.5