



**SRI VENKATESWARA INTERNSHIP PROGRAM
FOR RESEARCH IN ACADEMICS
(SRI-VIPRA)**



SRI-VIPRA

Project Report of 2024: SVP-2452

**“Air Pollution Prediction with Artificial Intelligence(AI) &
Machine Learning (ML):**

A case study of Indian cities”

IQAC

Sri Venkateswara College



University of Delhi

Benito Juarez Road, Dhaula Kuan, New Delhi


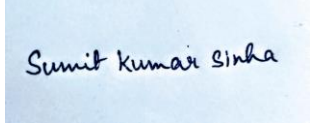

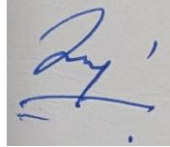


New Delhi -110021

SRIVIPRA PROJECT 2024

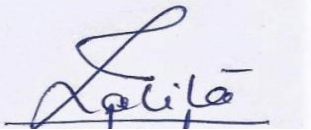
Title : ...Air Pollution Prediction with AI & ML:A case study of Indian cities..

Details of Mentors	
Name of Mentor: Dr. Tarakeswara Rao K Name of Department: Electronics Designation: Assistant Professor	Name of Co-Mentor: Dr. Lalita Josyula Name of Department: Electronics Designation: Professor
	

List of students under the SRIVIPRA Project

S.No	Photo	Name of the student	Roll number	Course	Signature
1		Sumit Kumar Sinha	1622049	B.Sc. (Hons.) Electronics	
2		Oikyam Jyotish Chetia	1122053	B.Sc.(P) Life Sciences	
3		Abhinav Mallick	1722043	B.Sc. (H) Mathematics	


Signature of Mentor


Signature of Co-Mentor

Certificate of Originality

This is to certify that the aforementioned students from Sri Venkateswara College have participated in the summer project SVP-2452 titled “**Air Pollution Prediction with AI & ML: A case study of Indian cities**”. The participants have carried out the research project work under our guidance and supervision from 1st July, 2024 to 30th September 2024. The work carried out is original and carried out in an online/offline/hybrid mode.



Signature of Mentor



Signature of Co-Mentor

Acknowledgements

We would like to express our gratitude and thanks to our Professors and Mentors, Dr. Tarakeswara Rao Kaviti & Prof. Lalita Josyula for guiding us in this project. They provided us with their valuable suggestions throughout the process. They encouraged us to undertake this project from time to time.

We would also like to give special thanks to Sri Venkateswara College for providing all necessary facilities to carry out this Project smoothly. We will keep on trusting the College facilities for my future endeavours in the college.

We would like to express our sincere thanks to the Principal of Sri Venkateswara College, University of Delhi for organizing the SRIVIPRA internship Program, 2024. This program has helped us to put our scientific and computational understanding to application.

We would also like to thank the Coordinators of SRIVIPRA program 2024 for providing us the opportunity to be a part of this SRIVIPRA PROJECT 2024.

We would also like to thank our family and friends for supporting us throughout this project.

TABLE OF CONTENTS

S.No	Topic	Page No.
1.	Introduction	
2.	Objectives & Methodology	
3.	Dataset Overview	
4.	Tools and Technologies	
5.	Implementation	
6.	Result and Discussion	
7.	Conclusion and Future Development	
8.	References	
9.	Appendix1: Code and Output	

1. Introduction

1.1 Background

Air pollution is one of the most pressing environmental issues faced by modern society. With rapid industrialization, urbanization, and increasing vehicular emissions, the levels of harmful pollutants in the atmosphere have significantly risen. Pollutants such as particulate matter (PM_{2.5}, PM₁₀), nitrogen oxides (NO, NO₂, NO_x), carbon monoxide (CO), sulfur dioxide (SO₂), and volatile organic compounds (VOCs) like benzene and toluene have detrimental effects on air quality. Poor air quality not only degrades the environment but also poses serious health risks to the population, particularly those with pre-existing respiratory conditions like asthma or chronic bronchitis. This is why the **Air Quality Index (AQI)**, a metric used to assess and report daily air quality levels, plays a critical role in public health management and environmental policy-making.

Air pollution refers to the contamination of the atmosphere by harmful substances including gases, particulate matter, chemicals, and biological molecules. These pollutants can come from both natural sources, such as wildfires and volcanic eruptions, and human activities like vehicle emissions, industrial processes, and the burning of fossil fuels. [1]

Air pollution is one of the serious issues all around the globe and can cause a variety of health problems such as skin infections, eye diseases, throat infections, lung cancer, bronchitis diseases, respiratory diseases, cardiovascular issues, exacerbation of asthma, etc. According to WHO around 7 million people got affected with numerous diseases and reports of 1.3 million deaths due to air pollution. Long-term exposure of air pollutants may increase the chances of premature mortalities. Children might face developmental issues such as physiological and cognitive. Pregnant women face developmental issues which includes low birth weight, premature births, etc. In addition to that it also contributes to environmental degradation, including acid rain, depletion of the ozone layer, global warming, introduces a serious threat to plant, and also impacts our economy seriously. Huge financial investments are required to mitigate the air pollution which leads to economic losses for the government and organizations. [2]

India regularly sees AQI values exceeding 300 in many major cities, which indicates a "hazardous" level of pollution. For example, Delhi is often in the spotlight for its alarmingly high AQI during the winter months, driven by a combination of vehicular emissions, industrial discharges, and crop-burning activities in nearby states. In some instances, AQI readings in Delhi have even surpassed 500, leading to public health emergencies. Studies suggest that long-term exposure to high PM_{2.5} levels can reduce life expectancy, contribute to chronic illnesses such as asthma and bronchitis, and increase the risk of heart attacks. In terms of specific pollutant data, the average annual PM_{2.5} concentration in several Indian cities frequently exceeds the World Health Organization (WHO) guideline of 10 µg/m³ by several multiples. Cities like Kolkata, Mumbai, and Chennai also experience high concentrations of other harmful pollutants such as NO₂ and CO due to traffic congestion and industrial emissions. Furthermore, there is an upward trend in ground-level ozone pollution, which exacerbates respiratory problems, particularly in vulnerable populations like children and the elderly. [3]

AQI calculation

The Air Quality Index (AQI) is a numerical value that reflects the quality of air in a specific area, providing a measure of pollution levels and their potential impact on human health. The AQI is calculated based on the concentrations of key air pollutants such as particulate matter (PM10 and PM2.5), ozone (O₃), nitrogen dioxide (NO₂), sulfur dioxide (SO₂), carbon monoxide (CO), and ammonia (NH₃). Each of these pollutants has its own individual sub-index value that contributes to the overall AQI. The AQI calculation for each pollutant follows a linear interpolation method, using breakpoints that correspond to different concentration ranges for each pollutant. These breakpoints are linked to AQI categories that represent different health impact levels: "Good," "Satisfactory," "Moderate," "Poor," "Very Poor," and "Severe." The formula used for calculating the AQI for a given pollutant is:

$$AQI_p = ((I_{HI} - I_{LO}) / (C_{HI} - C_{LO})) \times (C_p - C_{LO}) + I_{LO}$$

Where, AQI_p is the AQI for pollutant p, C_p is the actual concentration of pollutant p, C_{HI} and C_{LO} are the upper and lower concentration limits for the AQI category in which C_p falls, I_{HI} and I_{LO} are the corresponding AQI values for the upper and lower concentration breakpoints.

The final AQI value is determined by taking the maximum AQI among all pollutants being monitored for a given time period. This ensures that the pollutant with the highest sub-index, which poses the greatest health risk, dominates the overall AQI value for that location. For example, if the AQI for PM2.5 is the highest among all pollutants in a city on a particular day, this will dictate the overall AQI level and the associated health advisory. The index helps communicate air quality to the public in a simple and standardized way, offering clear guidance on whether the air is safe to breathe and what actions individuals should take to protect their health.

The AQI includes six color-coded categories, each corresponding to a range of index values. The higher the AQI value, the greater the level of air pollution and the greater the health concern. For example, an AQI value of 50 or below represents good air quality, while an AQI value over 300 represents hazardous air quality.

For each pollutant an AQI value of 100 generally corresponds to an ambient air concentration that equals the level of the short-term national ambient air quality standard for protection of public health. AQI values at or below 100 are generally thought of as satisfactory. When AQI values are above 100, air quality is unhealthy: at first for certain sensitive groups of people, then for everyone as AQI values get higher.

The AQI is divided into six categories. Each category corresponds to a different level of health concern. Each category also has a specific color. The color makes it easy for people to quickly determine whether air quality is reaching unhealthy levels in their communities. [4]

AQI Category, Pollutants and Health Breakpoints								
AQI Category (Range)	PM ₁₀ 24-hr	PM _{2.5} 24-hr	NO ₂ 24-hr	O ₃ 8-hr	CO 8-hr (mg/m ³)	SO ₂ 24-hr	NH ₃ 24-hr	Pb 24-hr
Good (0-50)	0-50	0-30	0-40	0-50	0-1.0	0-40	0-200	0-0.5
Satisfactory (51-100)	51-100	31-60	41-80	51-100	1.1-2.0	41-80	201-400	0.5 – 1.0
Moderately polluted (101-200)	101-250	61-90	81-180	101-168	2.1- 10	81-380	401-800	1.1-2.0
Poor (201-300)	251-350	91-120	181-280	169-208	10-17	381-800	801-1200	2.1-3.0
Very poor (301-400)	351-430	121-250	281-400	209-748*	17-34	801-1600	1200-1800	3.1-3.5
Severe (401-500)	430+	250+	400+	748+*	34+	1600+	1800+	3.5+

Fig 1.1. AQI categories for different pollutants

1.2 Problem Statement

Predicting AQI based on various pollutants such as PM_{2.5}, PM₁₀, CO, NO, NO₂, and ozone is a complex challenge due to the dynamic interaction between these pollutants, weather conditions, and geographical factors. Accurate AQI prediction allows authorities to take proactive steps to manage pollution levels and protect public health. The main goal of this project is to develop machine learning models that can accurately predict AQI based on the concentrations of various pollutants, thereby providing timely and actionable information.

1.3 Importance

Accurate AQI prediction holds immense importance in several domains:

- **Public Health:** Timely AQI predictions enable individuals to take preventive measures, such as avoiding outdoor activities during periods of high pollution. It also helps healthcare providers to identify and address health issues related to air pollution exposure.
- **Policy Making:** Governments can use AQI predictions to develop and implement effective air pollution control policies, such as vehicle emission standards, industrial regulations, and urban planning strategies.
- **Environmental Awareness:** AQI predictions can raise public awareness about air pollution and its impacts, encouraging individuals to adopt sustainable practices and support environmentally friendly initiatives.

2.Objective

The main objective of this project is to build a machine learning model capable of accurately predicting the **Air Quality Index (AQI)** based on the concentration of various pollutants, including PM2.5, PM10, NO, NO2, NOx, CO, Ozone, SO2, and other factors such as wind speed and direction. By doing so, the model can provide real-time or near real-time AQI forecasts, which can be used to inform public health policies, individual precautions, and environmental actions.

The project also aims to:

- Identify the most influential pollutants contributing to AQI levels.
- Optimize different machine learning models, such as Decision Trees, Random Forests, Gradient Boosting, and others, to achieve the best prediction accuracy.
- Develop visualization tools to analyze and communicate model results and pollutant trends.

2.1 Scope

The scope of this project is defined as follows:

- **Geographic Area:** The model focuses on air quality data collected from specific regions or cities. For this project, the dataset might cover a specific city (e.g., Kolkata) or a broader area depending on available data.
- **Time Period:** The dataset used includes air quality and pollutant data from **January 1, 2015, to December 31, 2020**. The model is trained on this time period to make predictions for the following months.
- **Predictors:** The main predictors (features) used in the model include various pollutants (PM2.5, PM10, NO, NO2, NOx, CO, etc.), meteorological variables like wind speed and direction, and time-related factors such as month or season.

2.2 Challenges

Several challenges were encountered during the development of this project:

1. **Data Quality and Missing Values:** Air pollution data often contains missing values due to sensor malfunctions, downtime, or calibration errors. Imputing these missing values effectively was critical for ensuring the accuracy of the model.
2. **Multicollinearity:** Some pollutants are strongly correlated with each other (e.g., NO, NO2, and NOx), which can cause issues in model performance. It was necessary to handle multicollinearity through feature engineering or by selecting the most relevant features.
3. **Seasonality and Temporal Trends:** Air pollution levels can fluctuate significantly based on seasonal changes, holidays, or local activities. Capturing these temporal trends and seasonality factors is essential for making accurate predictions.
4. **Complexity of Pollutant Interaction:** Various pollutants interact with each other in complex ways, influenced by weather conditions, geographical factors, and human activity. Modeling these relationships requires careful selection of machine learning algorithms.
5. **Model Generalization:** While the model can perform well on historical data, ensuring that it generalizes to future or unseen data is a challenge, particularly when considering evolving environmental policies and changes in pollutant sources.

2.3 Methodology

2.3.1. Experimental setup

The dataset utilized for the proposed model evaluation is the publicly available Air Quality Data in India (2015-2020) from the Kaggle repository. It comprises air quality and air quality index (AQI) data recorded at hourly and daily intervals across various stations in 26 major cities in India. The dataset includes attributes such as date, month, year, PM_{2.5}, PM₁₀, NO, NO₂, NO_x, NH₃, CO, SO₂, O₃, Benzene, Toluene, AQI, and AQI_Bucket. The AQI_Bucket classifies AQI into six categories: good, satisfactory, moderate, poor, very poor, and severe.

For the study, Delhi, Guwahati, and Thiruvananthapuram are selected from different geographic regions of India. A daily timeframe was chosen for model experimentation. Separate CSV files are being created for each of these cities. Most parameters are common across the selected cities; however, in cases of discrepancies, certain parameters are entirely omitted from the datasets. Additionally, AQI_Bucket is excluded from the new datasets, as the model experimentation focused on a regression-based approach.

1. Preprocessing

The dataset used for the proposed model experimentation contained null values and outliers, necessitating data cleaning. First, the null values are identified and analyzed, then replaced with the mean values of their respective columns. However, columns with a significant number of null values are entirely removed from the dataset. Outliers are first detected and addressed using a quartile-based method that scaled the extreme values. After that the processed data was once again analyzed

2. Feature selection, Independent and Dependent variables

The individual parameters within each dataset are correlated with one another through various methods such as multiple plots, a correlation table, and a heat map generated from the correlation table. Emphasis is placed on parameters that exhibited a certain desired level of correlation, while those that did not meet this threshold are being removed from their respective datasets. Each city's dataset displayed its own unique correlation patterns, so there is no strict requirement to select the same parameters across all datasets. However, it is essential to note that parameters like PM_{2.5} and PM₁₀ are mandatory for all datasets used in the prediction model.

Following this, based on feature selection, the independent variables were determined according to the specific dataset of each city, while the dependent variable remained the same across all datasets, i.e., 'AQI'.

The final parameters are once again compared after then once again compared with accordance to their importance before splitting the test and train data. The difference here is that, all the required parameters for predicting AQI, are now stored as independent variables and are correlated against AQI in the form of horizontal bar-graph.

3.) Spiting train and test data

The training and testing datasets are splited using `sklearn` libraries, with 70% of the data allocated for training.

4. Working with the model

The models used, are then imported from sklearn, Linear Regression, Lasso Regression, Ridge Regression, Decision Tree (as regression model), Random Forest (as regression model), KNN (as regression model), are being used as prediction model. Every model is first trained with the train data and then the test data is implemented, each test and train data are scored with R²

scores. Mean score was used for scoring the prediction on the created variables in a cross validated method, again using R^2 scores. Predicted AQI values are then plotted as a histogram, also a scatter plot is plotted comparing predicted values and actual values, and lastly line plot is plotted, again for comparison purpose.

Each model is hypertuned or optimised, irrespective of any display of overfitting, so to get better results. Linear Regression is optimised with Lasso and Ridge Regression, Decision Tree and Random Forest is hypertuned with Grid Search CV and Randomized Search CV, respectively. While KNN is hypertuned with different K-values.

5. Comparing Scores

R^2 , MAE, MSE, RMSE scores for all of the models (before and after hypertuning) are then compared to each other, topic, shows the comparison below.

6. AQI calculator

A function is created taking the basis of the entire models used for prediction, which inputs all the selected parameters and outputs the AQI value for each of the model, while the tagging the model along with it.

```
def predict_aqi(modelz, model_names, input_data):
    """
    Predict AQI using multiple trained regression models and print predictions alongside model names.

    Parameters:
    - modelz: A list of trained regression models.
    - model_names: A list of model names corresponding to each regression model.
    - input_data: A numpy array or list of features for prediction.

    Returns:
    - predictions: A list of tuples containing the model name and predicted AQI value.
    """
    input_array = np.array(input_data).reshape(1, -1)
    preds = []

    for model, name in zip(modelz, model_names):
        try:
            pred = model.predict(input_array)
            preds.append((name, pred[0]))
            # Store model name with prediction
        except Exception as e:
            print(f"Error with model {name}: {e}")
            preds.append((name, None))
            # Append None for models that fail

    # Print model names with their predictions
    for name, predicts in preds:
        print(f"Model: {name}, Predicted AQI: {predicts}")

    return preds
```

Fig.2.1. displays the function that inputs variables and calculate the AQI levels using the models that have been trained with the datasets. This function holds true for all the datasets.

3. Dataset Overview

3.1 Dataset Source

The dataset used for this project likely originates from official government agencies, environmental organizations, or air quality monitoring stations that continuously track pollution levels across various cities. Common sources for such datasets include the Central Pollution Control Board (CPCB) in India. These agencies regularly publish air quality data based on readings from automated monitoring stations located across cities.

3.2 Features

The dataset consists of several important features, which include various air pollutants and meteorological data that influence AQI levels. The key features are:

1. **PM2.5:** Fine particulate matter smaller than 2.5 micrometers, a major component of air pollution that can penetrate the lungs and affect respiratory health.
2. **PM10:** Particulate matter smaller than 10 micrometers, which can cause throat irritation and exacerbate lung conditions.
3. **NO₂ (Nitrogen Dioxide):** A harmful gas primarily produced by vehicle emissions and industrial activities.
4. **NO:** Nitric oxide, often produced during combustion, which can react to form NO₂.
5. **CO (Carbon Monoxide):** A colorless, odorless gas that can be dangerous in high concentrations, often emitted from cars and other combustion sources.
6. **Ozone (O₃):** A reactive gas present in both the Earth's upper atmosphere (good ozone) and at ground level (bad ozone), which can cause respiratory problems.
7. **Total NO_x:** A combined measure of nitrogen oxides (NO, NO₂, NO_x) that affect air quality.

3.3 Target Variable

The target variable in this dataset is the **AQI (Air Quality Index)**. The AQI provides a simplified numerical measure to convey the overall quality of the air. Higher AQI values indicate worse air quality, with values over 100 typically considered unhealthy.

3.4 Data Preprocessing

Several preprocessing steps were performed to prepare the dataset for machine learning:

1. **Handling Missing Values:** The dataset contained some missing values, likely due to sensor malfunctions or data collection gaps. These missing values were imputed using median values to avoid introducing bias into the model. Median imputation was chosen as it is robust against outliers.
2. **Feature Engineering:** Related pollutant variables such as **NO**, **NO₂**, and **NO_x** were combined into a new feature called **Total NO_x** to reduce multicollinearity and better represent the overall impact of nitrogen oxides.
3. **Dropping Irrelevant Features:** Some features, such as **Xylene**, were dropped because they had no or insufficient data across the dataset.
4. **Scaling:** Since different pollutants are measured in different units, it was important to scale the data to ensure all features contribute equally to the machine learning model. Standardization (subtracting the mean and dividing by the standard deviation) or MinMax scaling was applied.
5. **Date and Time Processing:** The date column was converted to a suitable datetime format, and additional time-related features like the **month** or **season** were extracted to capture temporal trends in air pollution.

4. Tools and Technologies

4.1 Programming Languages

- **Python:** The primary programming language used for this project is Python due to its extensive libraries and frameworks for data analysis, machine learning, and visualization. Python's simplicity and readability make it an ideal choice for both beginners and experienced data scientists.

4.2 Libraries/Frameworks

- **Pandas:** A powerful library used for data manipulation and analysis. It provides data structures such as DataFrames, which make it easy to handle and analyze structured data efficiently.
- **NumPy:** This library is used for numerical computations and provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- **Scikit-learn:** A key library for machine learning in Python. It provides a variety of algorithms and tools for classification, regression, clustering, and model evaluation. Scikit-learn is used for implementing machine learning models, including decision trees, random forests, and gradient boosting.
- **Matplotlib:** A widely used plotting library in Python that provides a flexible way to create static, animated, and interactive visualizations. It is often used for creating basic plots and figures.
- **Seaborn:** Built on top of Matplotlib, Seaborn provides a high-level interface for drawing attractive and informative statistical graphics. It simplifies the process of creating complex visualizations, such as heatmaps and pair plots.
- **XGBoost:** An optimized gradient boosting library designed to be highly efficient and scalable. It is particularly useful for structured/tabular data and is known for its performance in machine learning competitions.

4.3 ML Algorithms

- **Decision Trees:** A simple yet powerful algorithm that uses a tree-like model of decisions. Decision trees are interpretable and can handle both numerical and categorical data.
- **Random Forests:** An ensemble method that combines multiple decision trees to improve prediction accuracy and control overfitting. Random forests are robust and effective for a wide range of datasets.
- **Gradient Boosting:** An iterative ensemble technique that builds trees one at a time, with each new tree correcting errors made by the previous ones. It is highly effective for regression and classification problems.
- **XGBoost:** An advanced gradient boosting algorithm that optimizes the model performance through techniques like regularization, which helps prevent overfitting.

4.4 Development Tools

- **Jupyter Notebooks:** An open-source web application that allows for the creation and sharing of documents that contain live code, equations, visualizations, and narrative text. Jupyter Notebooks are particularly useful for data exploration, visualization, and documentation of the analysis process.

5.implementation

5.1 Data Preprocessing

1. Handling Missing Values:

- The dataset contained missing values due to sensor malfunctions or gaps in data collection. To handle these missing values, the median imputation method was employed. This approach was chosen because the median is robust to outliers and provides a central tendency measure that minimizes bias.

2. Outlier Detection:

- Outliers in the dataset were identified using visualizations such as box plots and scatter plots. If necessary, outliers could be addressed by either removing them or applying transformations (like logarithmic) to reduce their impact on the model.

3. Feature Engineering:

- Combining pollutant features: The individual pollutant features—**NO**, **NO2**, and **NOX**—were combined into a single feature called **Total NOx**. This aggregation helps reduce multicollinearity and simplifies the model by focusing on the overall contribution of nitrogen oxides to air quality.

4. Scaling:

- Since the features were measured on different scales, feature scaling was applied to standardize the data. Techniques such as Min-Max scaling or Standardization (Z-score normalization) were used to bring all features to a similar range, enhancing model performance.

5.2 Train-Test Split

- The dataset was divided into training and testing sets using an 80-20 split. This means that 80% of the data was used for training the model, while the remaining 20% was reserved for testing. This approach is beneficial because it provides enough data for the model to learn effectively while still having a separate dataset to evaluate its performance. The test set serves as a safeguard against overfitting, ensuring that the model generalizes well to unseen data.

5.3 Model Building

Multiple machine learning models were implemented to predict AQI based on pollutant levels:

1. Decision Tree:

- A straightforward algorithm that splits the data into branches based on feature values. It's interpretable but can easily overfit the training data.

2. Random Forest:

- An ensemble method that uses multiple decision trees to improve prediction accuracy and robustness. Random Forests reduce overfitting by averaging predictions from different trees.

3. Gradient Boosting:

- An ensemble technique that builds trees sequentially, where each tree corrects errors from the previous one. This model tends to perform well for complex datasets.

4. XGBoost:

- A powerful implementation of gradient boosting that optimizes for speed and performance. It includes additional features like regularization, which helps mitigate overfitting.

5.4 Parameter Tuning

- Hyperparameter optimization was performed using techniques like **GridSearchCV** and **RandomizedSearchCV**.

- **GridSearchCV** involves exhaustively searching through a specified subset of hyperparameters to find the best combination.
- **RandomizedSearchCV** samples a given number of candidates from a parameter space, allowing for faster execution when compared to GridSearch. This tuning process helps to improve model performance by identifying the most suitable hyperparameters.

5.5 Cross-Validation

- K-Fold cross-validation was employed to assess model performance. This method involves dividing the training dataset into K subsets (or folds). The model is trained on K-1 folds and tested on the remaining fold. This process is repeated K times, with each fold serving as the test set once. Cross-validation helps ensure that the model's performance is consistent across different subsets of the data, reducing the likelihood of overfitting and providing a more accurate estimate of model generalization to unseen data.

6. Results and Discussion

6.1 Results

6.1.1 Fold Wise Best Model Performances across cities in comparison to Lagged Variables

Table 4. Delhi Fold Wise Model Performance for Linear Regression

Fold	Without Lagged Variables				With Lagged Variables			
	RMSE	MSE	MAE	R ² Score	RMSE	MSE	MAE	R ² Score
1	51.86	2689	40.64	77.53	38.89	1512	29.56	89.95
2	73.54	5409	60.81	76.80	40.40	1632	30.24	88.42
3	38.11	1452	28.10	88.73	35.77	1279	27.23	88.97
4	36.98	1367	30.18	89.06	44.60	1989	30.98	86.29
5	38.71	1498	32.48	89.66	35.31	1247	26.56	91.91
Average	47.84	2483	38.44	84.36	39.00	1532	28.91	89.11

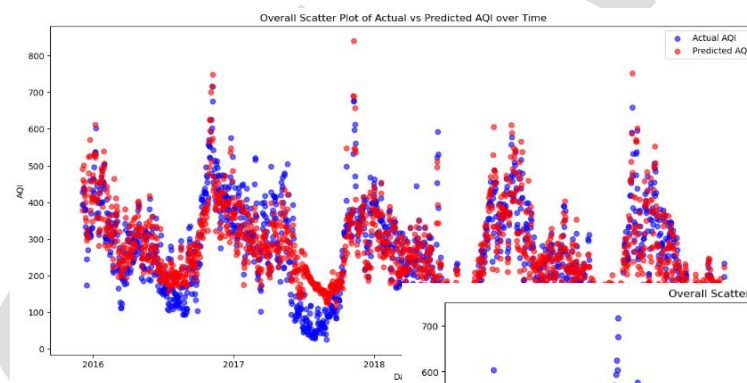


Figure 6: Scatter Plot of Actual vs Predicted AQI by Gradient Boosting Regressor

Table 5. Delhi Fold Wise Model Performance for Gradient Boosting Regressor

Fold	Without Lagged Variables				With Lagged Variables			
	RMSE	MSE	MAE	R ² Score	RMSE	MSE	MAE	R ² Score
1	50.95	2595.60	39.60	78.32%	29.96	897.73	22.17	94.04%

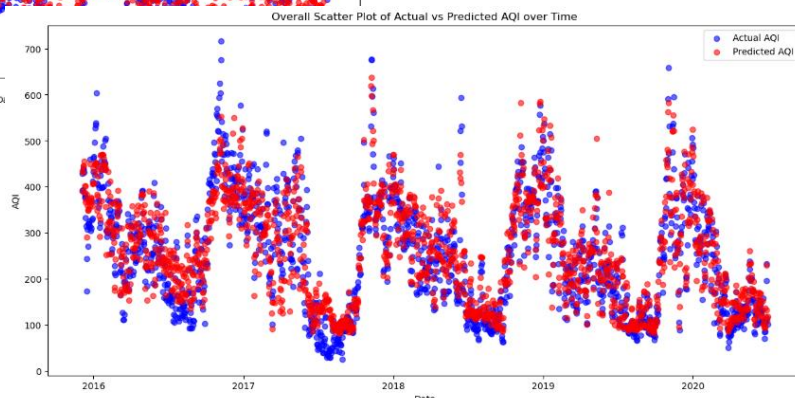


Figure 5: Scatter Plot of Actual vs Predicted AQI by Linear Regression.

2	61.84	3824.13	47.77	83.60%	28.61	818.80	20.11	94.20%
3	37.42	1400.20	28.59	89.14%	28.48	811.01	20.01	93.02%
4	32.18	1035.67	23.90	91.72%	29.76	885.63	20.71	93.90%
5	26.89	723.26	20.70	95.01%	24.75	612.44	17.20	96.03%
Average	41.86	1915.77	32.11	87.56%	28.31	805.12	20.04	94.24%

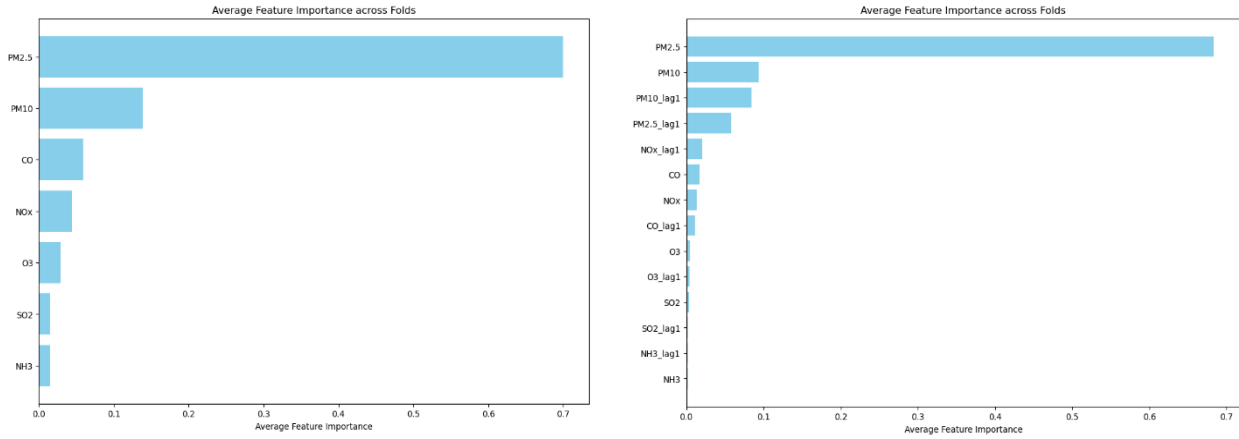


Figure 7: (Left) Feature Importance from GBR without Lagged Variables. (Right) Feature Importance from GBR with Lagged Variables

Table 6. Guwahati Fold Wise Model Performance for GradientBoosting Regressor

Fold	Without Lagged Variables				With Lagged Variables			
	RMSE	MSE	MAE	R ² Score	RMSE	MSE	MAE	R ² Score
1	123.10	15154.40	38.64	36.80%	23.87	570.01	18.31	63.33%
2	104.85	10993.90	32.34	22.08%	24.24	587.69	19.89	43.85%
3	108.78	11833.46	55.41	52.29%	48.00	2304.45	28.63	62.24%
4	16.14	260.57	11.43	-0.73%	9.16	83.96	6.67	67.54%
5	48.27	2330.48	27.92	52.62%	36.47	1329.79	19.82	72.97%
6	50.11	2511.03	40.09	55.90%	31.65	1001.98	25.39	82.40%
7	36.86	1358.44	27.59	62.17%	31.20	973.16	21.76	72.90%
8	38.31	1467.90	30.71	73.42%	33.06	1092.66	24.31	80.21%
9	28.14	791.61	18.51	82.87%	17.82	317.73	12.51	93.12%
10	9.70	94.02	6.86	50.71%	8.25	68.07	6.51	64.31%
Average	56.43	4679.58	28.95	48.81%	26.37	832.95	18.38	70.29%

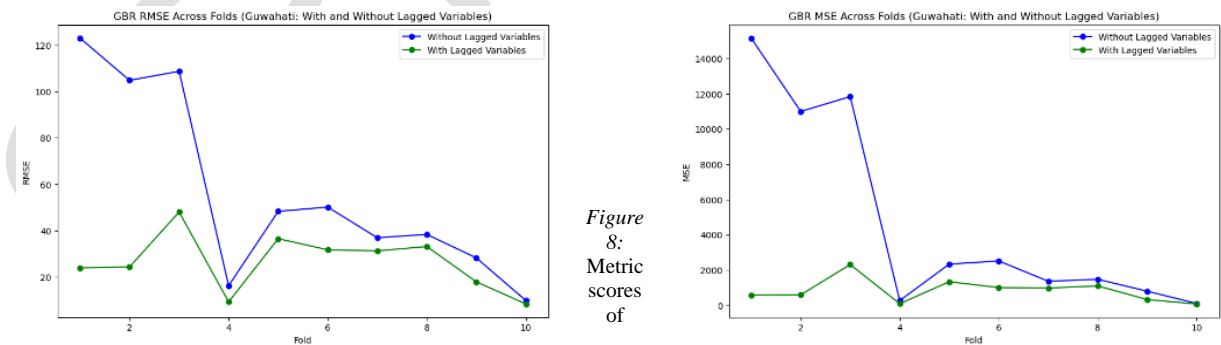


Figure 8: Metric scores of Guwahati visualized across folds

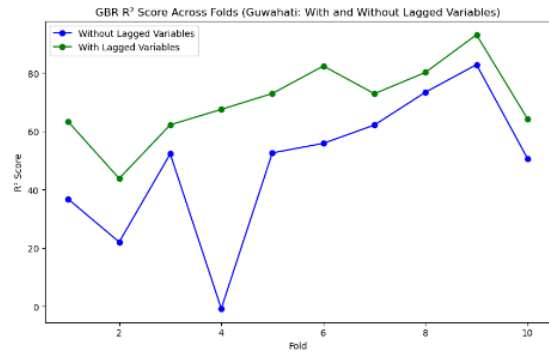
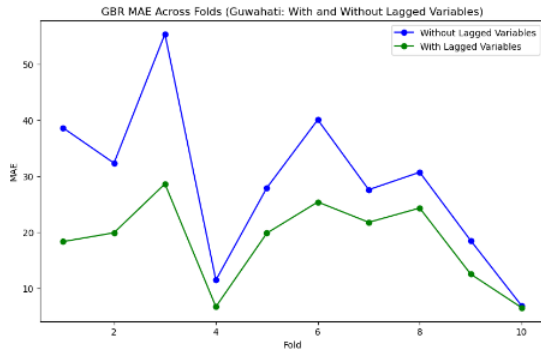


Table 7.

Hyderabad Fold Wise Model Performance for GradientBoosting Regressor

Fold	Without Lagged Variables				With Lagged Variables			
	RMSE	MSE	MAE	R ² Score	RMSE	MSE	MAE	R ² Score
1	58.01	3365.64	42.12	22.14%	31.83	1013.30	19.92	76.56%
2	26.94	725.99	20.37	56.32%	27.65	764.39	21.19	54.01%
3	17.90	320.43	14.27	72.98%	13.98	195.50	11.16	83.52%
4	12.24	149.83	9.59	87.14%	8.35	69.66	6.67	94.02%
5	14.24	202.78	12.43	84.15%	8.83	78.00	7.00	93.90%
Average	25.87	952.93	19.76	64.55%	18.13	424.17	13.19	80.40%

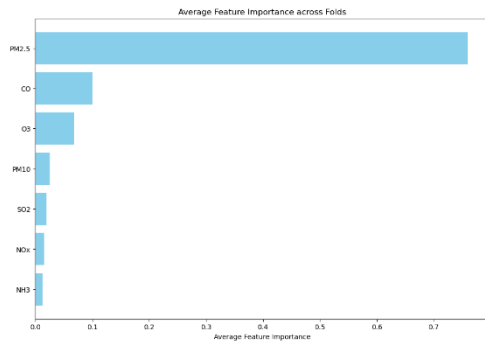


Figure 9: (Left) Feature Importance from GBR without Lagged Variables. (Right) Feature Importance from GBR with Lagged Variables

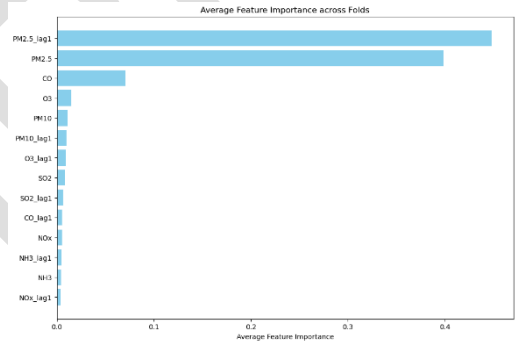


Table 8. Kolkata Fold Wise Model

Performance for Ridge Regression

Fold	Without Lagged Variables				With Lagged Variables			
	RMSE	MSE	MAE	R ² Score	RMSE	MSE	MAE	R ² Score
1	13.92	193.68	9.85	89.51%	13.31	177.06	9.60	90.41%
2	88.34	7803.25	75.85	19.18%	81.50	6642.39	68.16	31.21%
3	36.23	1312.88	30.26	83.59%	27.39	750.07	22.53	90.62%
4	26.58	706.72	22.73	65.16%	21.68	470.12	18.23	76.82%
5	14.74	217.17	10.00	24.40%	10.18	103.56	8.01	63.95%
6	8.00	64.01	6.16	57.93%	5.33	28.40	4.28	81.34%
7	26.84	720.23	20.06	86.84%	24.27	589.16	15.72	89.24%
8	25.84	667.72	19.97	77.73%	21.62	467.21	16.99	84.42%
9	15.92	253.30	11.50	91.36%	11.69	136.62	8.23	95.34%
10	5.18	26.80	4.12	77.65%	6.74	45.45	5.18	62.11%
Average	26.16	1196.58	21.05	67.34%	22.37	941.00	17.69	76.54%

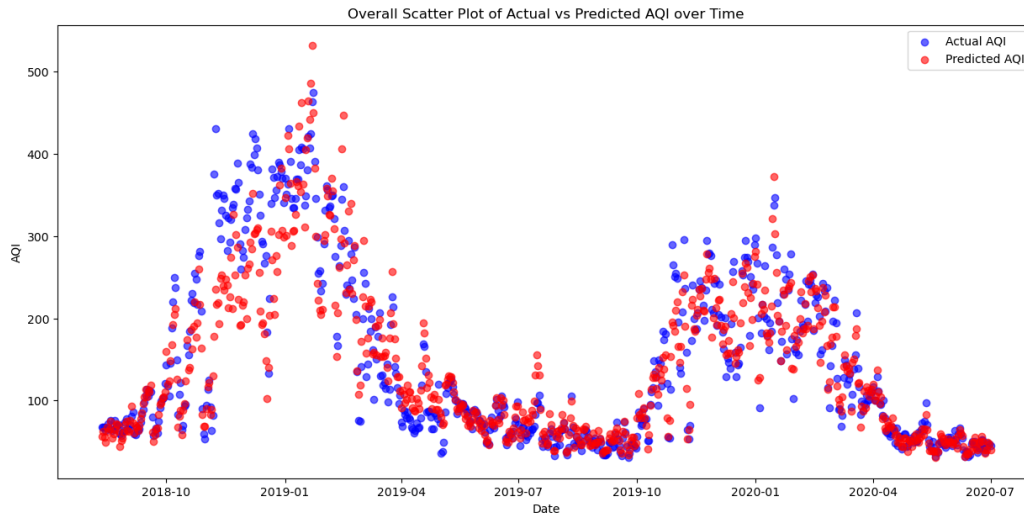


Figure 10: Scatter Plot of Actual vs Predicted AQI by Ridge Regression

Table 9. Visakhapatnam Fold Wise Model Performance for GradientBoosting Regressor

Fold	Without Lagged Variables				With Lagged Variables			
	RMSE	MSE	MAE	R ² Score	RMSE	MSE	MAE	R ² Score
1	41.84	1750.45	30.16	50.38%	16.95	287.31	11.35	92.54%
2	17.07	291.45	12.82	57.48%	19.03	362.13	11.88	88.43%
3	25.94	672.99	17.90	88.90%	15.57	242.32	10.47	89.69%
4	24.05	578.18	14.76	67.51%	14.03	196.93	10.37	93.60%
5	16.90	285.45	12.14	81.65%	19.89	395.45	11.85	83.16%
Average	25.16	715.70	17.56	69.18%	17.09	296.83	11.19	89.49%

6.1.2 Overall model performances across cities

Table 10. R² scores across all models and all cities

Model	Without Lagged Variables					With Lagged Variables				
	Delhi	Guwahati	Hyderabad	Visakhapatnam	Kolkata	Delhi	Guwahati	Hyderabad	Visakhapatnam	Kolkata
Linear	84.36%	40.11%	71.52%	69.36%	67.25%	89.11%	39.83%	74.18%	86.02%	76.09%
Ridge	84.36%	44.67%	71.46%	69.86%	67.34%	89.10%	28.36%	74.66%	86.03%	76.54%
Lasso	84.36%	41.23%	71.35%	69.53%	67.12%	89.11%	32.69%	74.34%	86.11%	76.26%
DTR	82.45%	37.80%	59.21%	58.42%	55.05%	88.36%	42.59%	70.26%	81.90%	53.93%
XGB	85.71%	35.50%	69.91%	67.56%	58.85%	94.01%	64.51%	79.10%	89.44%	69.52%
GBR	87.56%	48.81%	64.55%	69.18%	62.08%	94.24%	70.29%	80.40%	89.49%	69.50%
Average	84.80%	41.35%	68.00%	67.32%	62.95%	90.66%	46.38%	75.49%	86.50%	70.31%

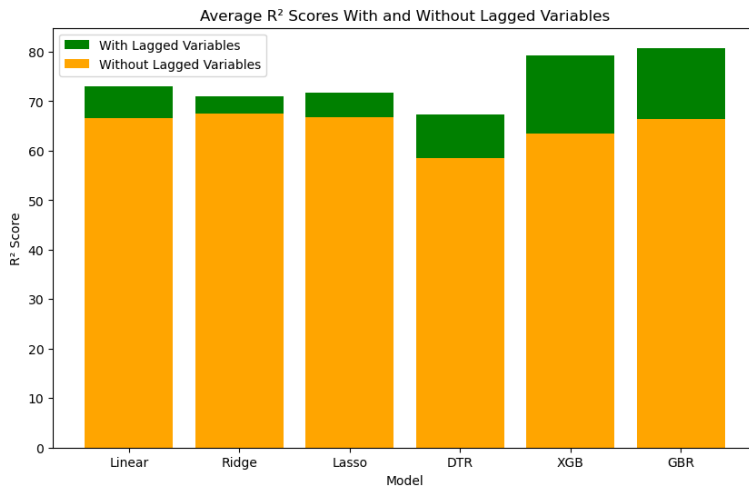
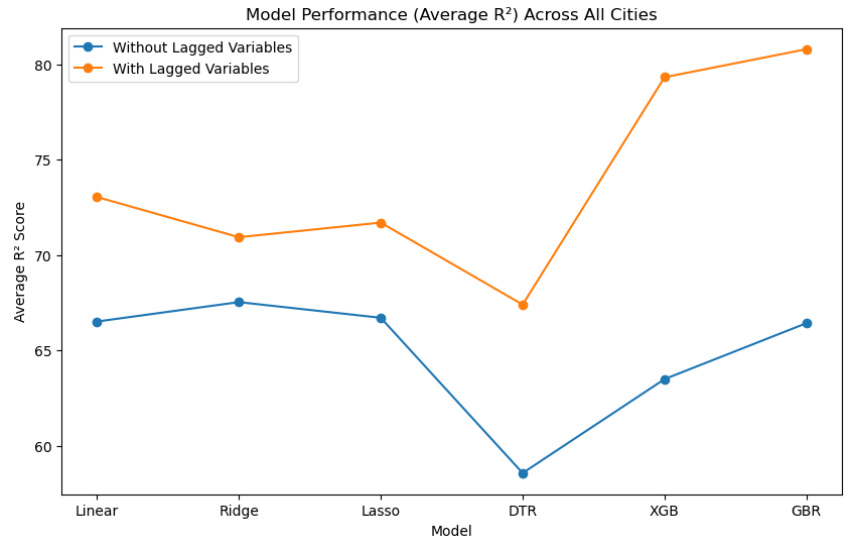


Figure 11. Increase in R² score due to Lagged Variables across all models

Figure 12. R² score visualized across Models with and without Lagged Variables



6.2 Discussions

6.2.1 Delhi

The Linear Regression model improved significantly with the inclusion of lagged variables. The average R² score increased from 84.36% (without lagged) to 89.11% (with lagged). The RMSE and MAE metrics also improved, showing better predictive performance with lagged variables. The GradientBoosting Regressor (GBR) achieved an impressive improvement, with the R² increasing from 87.56% (without lagged) to 94.24% (with lagged). *This shows that GBR effectively learned from the lagged features and performed robustly in Delhi.*

6.2.2 Guwahati

The overall performance across models was lower without lagged variables, but lagged inputs provided substantial improvements. For GBR, the R² increased from 48.81% to 70.29% when lagged variables were included. This highlights the importance of considering temporal features in AQI prediction for Guwahati, which greatly boosted the model's predictive capability.

6.2.3 Hyderabad

Similar improvements were observed in Hyderabad, with GBR showing a marked increase in R² from 64.55% (without lagged) to 80.40% (with lagged). Lagged features clearly helped the models better capture air quality variations in this city.

6.2.4 Kolkata

There was an overall improvement in model performance with lagged variables for all models, with GBR showing an R² increase from 62.08% to 69.50%. This suggests that including lagged features helps models account for the temporal dependencies in AQI better in Kolkata.

6.2.5 Visakhapatnam

GBR exhibited improved performance here as well, with an R² increasing from 69.18% to 89.49% due to lagged variables, making it the best-performing model for this city. The addition of lagged variables in Visakhapatnam resulted in more accurate predictions across all models, especially for GBR and XGB.

6.2.6 R² Score Comparison

Across all models and cities, the inclusion of lagged variables consistently improved the R² scores. The Gradient Boosting Regressor (GBR) emerged as the best-performing model across most cities. For example, in Delhi, it achieved an R² score of 94.24% with lagged variables. XGBoost (XGB) also showed good performance, particularly in Hyderabad and Visakhapatnam, where its R² scores increased to 79.10% and 89.44%, respectively, when lagged variables were included.

6.2.7 Model Average Performance

The average R² score across all cities for all models improved from 84.80% (without lagged) to 90.66% (with lagged), showing that the inclusion of lagged variables led to a significant overall improvement in AQI prediction. GBR consistently performed better than other models, indicating its effectiveness in capturing temporal dependencies and non-linear relationships in the data.

6.2.8 City-Specific Observations

Delhi and Visakhapatnam had the highest gains in R² score with the inclusion of lagged variables, particularly for advanced models like GBR and XGB. Guwahati and Hyderabad saw notable improvements, though their R² scores remained lower than Delhi and Visakhapatnam, suggesting the models may have struggled with more complex or noisier data in these cities. However, lagged variables did contribute to improved predictions across all cities.

7. Conclusion and Future Work

This research presents an alternative and robust approach to Air Quality Index (AQI) prediction by incorporating temporal dependencies through the inclusion of lagged pollutant variables. Unlike traditional models that often ignore the time sequence in AQI data, our study emphasizes the importance of respecting the temporal nature of air quality data. Through the use of a Nested Cross-Validation framework, this paper ensures rigorous and unbiased evaluation of different models, preventing data leakage and overfitting, which are common issues in time-series forecasting.

The introduction of lagged variables (e.g., PM2.5_lag1, PM10_lag1) significantly improved the predictive power of the models across all cities, as demonstrated by the consistent increase in R² scores. Notably, models like Gradient Boosting Regressor (GBR) and XGBoost (XGB) capitalized on the temporal information, yielding better performance, with GBR achieving the highest R² score of 94.24% for Delhi. The results underscore that incorporating lagged variables better captures the dynamic nature of pollutants and their effect on AQI, making the predictions more reflective of real-world conditions.

Moreover, the study utilized a TimeSeriesSplit method within the cross-validation framework, ensuring that the models trained on past data were evaluated on future unseen data, replicating real-world applications more effectively. The nested hyperparameter tuning, carried out within this time-series structure, further enhanced model performance by optimizing key parameters in a systematic and unbiased manner.

In comparison to conventional approaches that often jumble rows or ignore temporal structures, our model is more rigorous and applicable to real-world AQI forecasting, where time dependencies are critical. The integration of lagged features and time-respecting model evaluations ensures that this method not only yields higher predictive accuracy but also enhances the model's generalizability for real-world applications in urban air quality monitoring and policymaking.

Future work can extend this approach by incorporating additional temporal features, such as weather data or long-term seasonal effects, to further improve model performance and reliability. The field can also explore how exogenous factors—such as economic, social, and quality-of-life indicators—might influence air quality. The aim could be to investigate any correlations that exist which could lead to deeper insights and recommendations for policy interventions. Alongside this exploring Hybrid Models that combine traditional forecasting methods with non-linear techniques such as decision trees and neural networks could also be employed with the goal of capturing any unexplained variance in the residuals that may not have been addressed by the initial forecasting models. The use of advanced ensemble methods or deep learning architectures in a similar framework could also push the boundaries of AQI prediction accuracy. Hence, the author believes there remains a lot of work to be discovered in this sub-field that is at the intersection of Air Quality and Machine Learning.

References

1. Seinfeld, J. H., & Pandis, S. N. (2006). *Atmospheric Chemistry and Physics: From Air Pollution to Climate Change* (2nd ed.). Wiley-Interscience.
2. Natarajan, S.K., Shanmurthy, P., Arockiam, D. *et al.* Optimized machine learning model for air quality index prediction in major cities in India. *Sci Rep* 14, 6795 (2024).
3. Guttikunda et al., 2019: "Air pollution in Indian cities: Short- and long-term exposure health impacts".
4. CPCB (Central Pollution Control Board), 2014: "National Air Quality Index".
5. Prediction of Air Quality Index Using Machine Learning Techniques: A Comparative Analysis; <https://doi.org/10.1155/2023/4916267>
6. <https://www.kaggle.com/code/rohanrao/calculating-aqi-air-quality-index-tutorial>
7. <https://www.kaggle.com/datasets/rohanrao/air-quality-data-in-india>
8. Machine learning-based prediction of air quality index and air quality grade: a comparative analysis; <https://doi.org/10.1007/s13762-023-05016-2>
9. Optimized machine learning model for air quality index prediction in major cities in India; <https://doi.org/10.1038/s41598-024-54807-1>
10. Time series forecasting using a hybrid ARIMA and neural network model by G. Peter Zhang.

11. <https://clip.cpcb.gov.in/index.php/faq/>

9.Apendex1

Code and Output:

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

[ ]: df = pd.read_csv("C:\Users\skuma\.ipynb_checkpoints\city_day.csv")
df.info()

[ ]: # Remove duplicate rows
df_cleaned = df.drop_duplicates()
df_cleaned

[ ]: # Remove rows with any null values
df_cleaned = df.dropna()
df_cleaned

[ ]: df.columns

[ ]: df.head()
df.sort_index()

[ ]: df.describe().T

[ ]: # Show null values in the DataFrame
null_values = df.isnull()
print(null_values)

[1180]: # Count the number of null values in each column
null_counts = df.isnull().sum()
print(null_counts)

...
```

```

import numpy as np # Linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from warnings import filterwarnings
filterwarnings('ignore')

```

Input data files are available in the read-only "../input/" directory
For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

```

import os
for dirname, _, filenames in os.walk(r"C:\Users\skuma\.ipynb_checkpoints\city_day.csv"):
    for filename in filenames:
        print(os.path.join(dirname, filename))
df_city_day = pd.read_csv(r"C:\Users\skuma\.ipynb_checkpoints\city_day.csv")

```

```
df_city_day.head()
```

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI	AQI_Bucket
0	Ahmedabad	01-01-2018	84.46	NaN	7.58	87.62	48.40	NaN	7.58	102.36	69.02	14.48	45.60	6.33	278.0	Poor
1	Ahmedabad	02-01-2018	76.51	NaN	9.26	76.69	44.32	NaN	9.26	103.46	93.51	14.29	61.83	7.48	256.0	Poor
2	Ahmedabad	03-01-2018	63.88	NaN	13.87	69.32	44.16	NaN	13.87	70.30	105.33	12.77	60.99	9.50	300.0	Poor
3	Ahmedabad	04-01-2018	81.10	NaN	19.42	109.07	67.15	NaN	19.42	93.30	65.69	19.56	72.46	11.29	532.0	Severe
4	Ahmedabad	05-01-2018	73.61	NaN	25.96	173.08	103.80	NaN	25.96	93.86	36.51	17.69	61.48	9.93	534.0	Severe

```
df_city_day.shape
```

(4755, 16)

```
[1188]: df_city_day.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4755 entries, 0 to 4754
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   City        4755 non-null  object
 1   Date        4755 non-null  object
 2   PM2.5       4648 non-null  float64
 3   PM10        3533 non-null  float64
 4   NO          4681 non-null  float64
 5   NO2         4688 non-null  float64
 6   NOx         4718 non-null  float64
 7   NH3         3784 non-null  float64
 8   CO          4786 non-null  float64
 9   SO2         4611 non-null  float64
10  O3          4586 non-null  float64
11  Benzene     4397 non-null  float64
12  Toluene     4397 non-null  float64
13  Xylene      2153 non-null  float64
14  AQI         4588 non-null  float64
15  AQI_Bucket  4588 non-null  object
dtypes: float64(13), object(3)
memory usage: 594.5+ KB

```

```
[1190]: df_city_day.describe().T
```

	count	mean	std	min	25%	50%	75%	max
PM2.5	4648.0	53.392136	35.850812	2.00	29.8200	45.475	66.9700	311.35
PM10	3533.0	111.223170	64.841549	0.21	65.0900	101.540	142.0600	586.27
NO	4681.0	16.861431	20.471042	0.40	6.0100	10.040	18.3100	197.73
NO2	4680.0	39.939248	32.811022	0.51	18.3800	30.620	48.7825	292.02
NOx	4710.0	39.774573	33.723387	0.00	19.5100	29.280	46.5700	293.10
NH3	3784.0	26.104313	23.449026	0.15	11.4100	19.400	31.8100	219.26
CO	4706.0	5.569768	13.528905	0.00	0.6500	0.870	1.2700	134.85
SO2	4611.0	20.676276	27.904089	0.97	7.0900	10.890	17.2750	186.08
O3	4586.0	39.512610	21.043433	0.38	23.9025	36.755	51.5900	162.43
Benzene	4397.0	4.494287	6.825684	0.00	0.7400	2.460	4.9000	115.14
Toluene	4397.0	14.407769	19.097708	0.00	3.0400	8.190	18.9700	371.65
Xylene	2153.0	3.554617	5.250683	0.00	1.1300	2.400	4.2800	109.23
AQI	4588.0	195.156059	216.783199	23.00	82.0000	116.000	202.2500	2049.00

```
[1194]: # This function takes a DataFrame as a parameter and returns a table showing the number of null values in this DataFrame and the percentage of these null values in the total values.def bos_deger_goster(df):
def show_null_value(df):
    mis_val = df.isnull().sum()
    mis_val_percent = 100 * df.isna().sum() / len(df) # Calculates the percentage of blank values in each column out of the total values. We multiply by 100 to express it as a percentage.
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1) # Concatenates the number and percentage of null values into a new DataFrame.
    mis_val_table_ren_columns = mis_val_table.rename(columns = {0 : 'Missing Values', 1 : '% of Total Values'})
    return mis_val_table_ren_columns # DataFrame'ı döndürür.

show_null_value(df_city_day)
```

```
[1192]:
```

	Missing Values	% of Total Values
City	0	0.000000
Date	0	0.000000
PM2.5	107	2.250263
PM10	1222	25.699264
NO	74	1.556257
NO2	75	1.577287
NOx	45	0.946372
NH3	971	20.420610
CO	49	1.030494
SO2	144	3.028391
O3	169	3.554154
Benzene	358	7.528917
Toluene	358	7.528917
Xylene	2602	54.721346
AQI	167	3.512093
AQI_Bucket	167	3.512093

```
[1204]: df_city_day['Date'] = pd.to_datetime(df_city_day['Date'], format='%d-%m-%Y')
df_city_day = df_city_day.sort_values(by = 'Date')
```

```
[1206]: df_city_day['Date'].min(), df_city_day['Date'].max()
```

```
[1206]: (Timestamp('2018-01-01 00:00:00'), Timestamp('2020-07-01 00:00:00'))
```

```
[1208]: df_city_day.columns
```

```
[1208]: Index(['City', 'Date', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket'],
dtypes='object')
```

```
[1225]: # This function takes a DataFrame as a parameter and identifies outliers for numeric columns in the DataFrame. It replaces these outliers with the corresponding quartile values (Q1 or Q3). Outliers are identified using the interquartile range (IQR).
def replace_outliers_with_quartiles(df):
    for column in df_city_day.select_dtypes(include='number').columns: # used to cycle through all numeric columns in the DataFrame.
        Q1 = df_city_day[column].quantile(0.25)
        Q3 = df_city_day[column].quantile(0.75)
        IQR = Q3 - Q1
        # To identify outliers, lower and upper limits are calculated and values outside these limits are considered outliers.
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # For each column, we identify outliers and replace them with Q1 or Q3. We do this using a Lambda function. If the value is less than the lower bound, it is replaced with Q1. If it is greater than the upper bound, it is replaced with Q3. In the last case, the value is not changed and
        df_city_day[column] = df_city_day[column].apply(
            lambda x: Q1 if x < lower_bound else Q3 if x > upper_bound else x
        )
    return df_city_day
df_city_day = replace_outliers_with_quartiles(df_city_day)
```

```
[1227]: df_city_day.describe().T
```

```
[1227]:
```

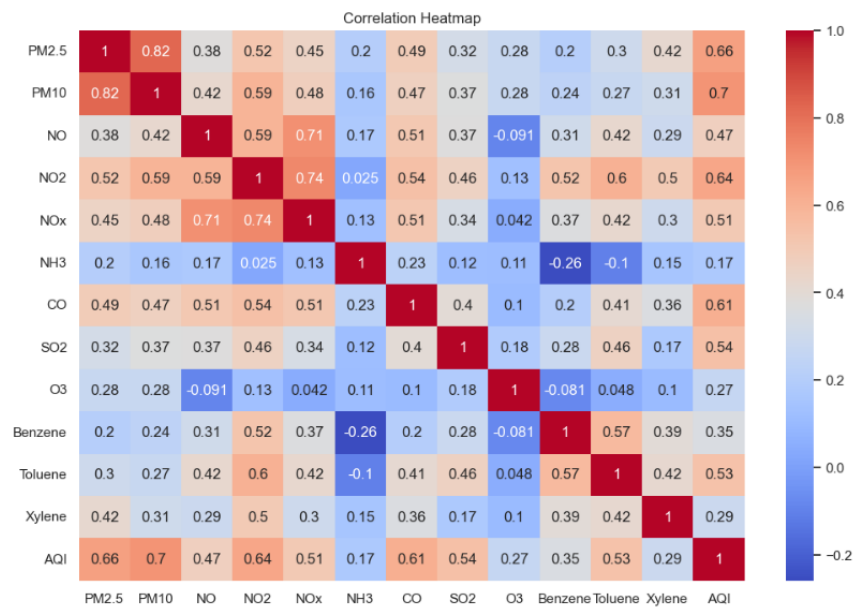
	count	mean	min	25%	50%	75%	max	std
Date	4755	2019-04-29 10:28:41.640378368	2018-01-01 00:00:00	2018-09-15 00:00:00	2019-05-11 00:00:00	2019-12-16 00:00:00	2020-07-01 00:00:00	NAN
PM2.5	4648.0	49.180962	2.0	29.82	45.475	66.97	122.43	25.09113
PM10	3533.0	105.447832	0.21	65.09	101.54	142.06	257.04	51.708117
NO	4681.0	12.026753	0.4	6.01	10.04	18.31	36.74	7.795707
NO2	4680.0	34.17444	0.51	18.38	30.62	48.773125	94.35	19.27415
NOx	4710.0	32.809849	0.0	19.51	29.28	46.5475	87.15	16.868484
NH3	3784.0	21.899897	0.15	11.41	19.4	31.78	62.2	13.142773
CO	4706.0	0.896793	0.0	0.65	0.87	1.27	2.2	0.358583
SO2	4611.0	11.624363	0.97	7.09	10.89	17.2725	32.39	5.811861
O3	4586.0	38.520961	0.38	23.9025	36.755	51.59	93.05	18.800295
Benzene	4397.0	2.873806	0.0	0.74	2.46	4.9	11.14	2.350867
Toluene	4397.0	11.368185	0.0	3.04	8.19	18.97	42.71	10.396442
Xylene	2153.0	2.867032	0.0	1.13	2.4	4.28	8.79	2.085384
AQI	4588.0	137.864701	23.0	82.0	116.0	202.0625	382.0	72.280916


```
[1237]: df=df_city_day.drop(columns=['City'])

[1239]: import matplotlib.pyplot as plt
import seaborn as sns

# Select only numeric columns from the dataframe
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Plot the heatmap for correlations between numeric columns
plt.figure(figsize=(12, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



```
[1241]: #The most important variables affecting the AQI value appear to be PM2.5, PM10, CO and NOx.
#We will make predictions based on data above 0.25
```

```
df_city_day=df_city_day.drop(columns=['Xylene', 'Benzene','O3'])
#We remove the Xylene, Benzene, O3 columns from df_city_day.
```

```
[1243]: df_city_day.head(2)
```

```
[1243]:
```

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	Toluene	AQI
0	Ahmedabad	2018-01-01	84.46	NaN	7.58	87.62	48.40	NaN	1.27	17.275	18.97	278.0
3842	Visakhapatnam	2018-01-01	59.73	99.04	1.73	23.69	13.89	11.68	0.95	5.750	3.67	131.0

```
[1245]: df_full=df_city_day[df_city_day['AQI'].notna()]
```

```
[1247]: import pandas as pd

# Convert the 'Date' column to datetime format
df_full['Date'] = pd.to_datetime(df_full['Date'])

# Now you can use the .dt accessor to extract the year
df_full['Year'] = df_full['Date'].dt.year

# Display the first few rows to check
df_full.head()
```

```
[1247]:
```

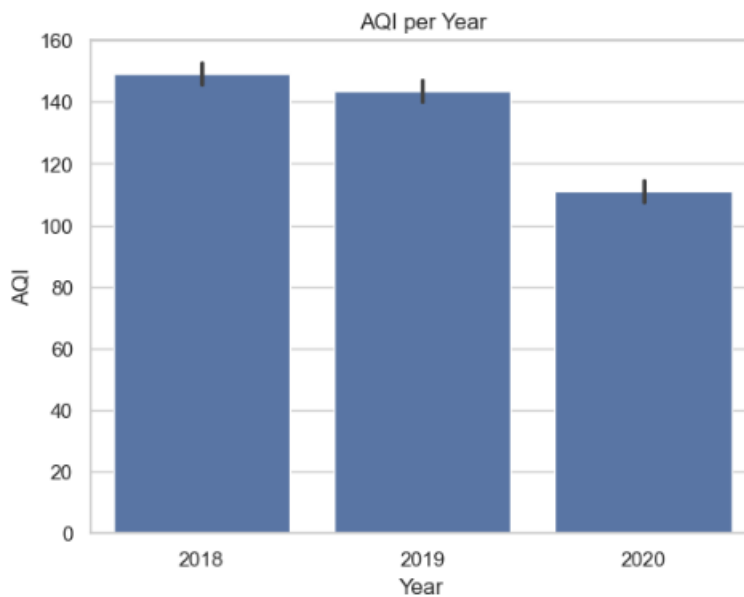
	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	Toluene	AQI	Year
0	Ahmedabad	2018-01-01	84.46	NaN	7.58	87.62	48.40	NaN	1.27	17.275	18.97	278.0	2018
3842	Visakhapatnam	2018-01-01	59.73	99.04	1.73	23.69	13.89	11.68	0.95	5.750	3.67	131.0	2018
2115	Jaipur	2018-01-01	82.33	178.65	9.33	43.34	30.68	23.53	0.00	9.850	4.52	221.0	2018
1202	Chennai	2018-01-01	51.16	NaN	8.39	13.63	11.72	31.81	0.03	6.000	2.97	89.0	2018
3843	Visakhapatnam	2018-01-02	64.22	106.57	1.69	28.36	16.24	11.86	1.02	9.840	3.60	125.0	2018

```
[1249]: df_full.head(2)
```

```
[1249]:
```

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	Toluene	AQI	Year
0	Ahmedabad	2018-01-01	84.46	NaN	7.58	87.62	48.40	NaN	1.27	17.275	18.97	278.0	2018
3842	Visakhapatnam	2018-01-01	59.73	99.04	1.73	23.69	13.89	11.68	0.95	5.750	3.67	131.0	2018

```
[1460]: plt.title('AQI per Year')
sns.barplot(x='Year',y='AQI',data=df_full);
```



```
[1463]: import pandas as pd
import matplotlib.pyplot as plt

# Convert the 'Date' column to datetime format
df_city_day['Date'] = pd.to_datetime(df_city_day['Date'], errors='coerce')

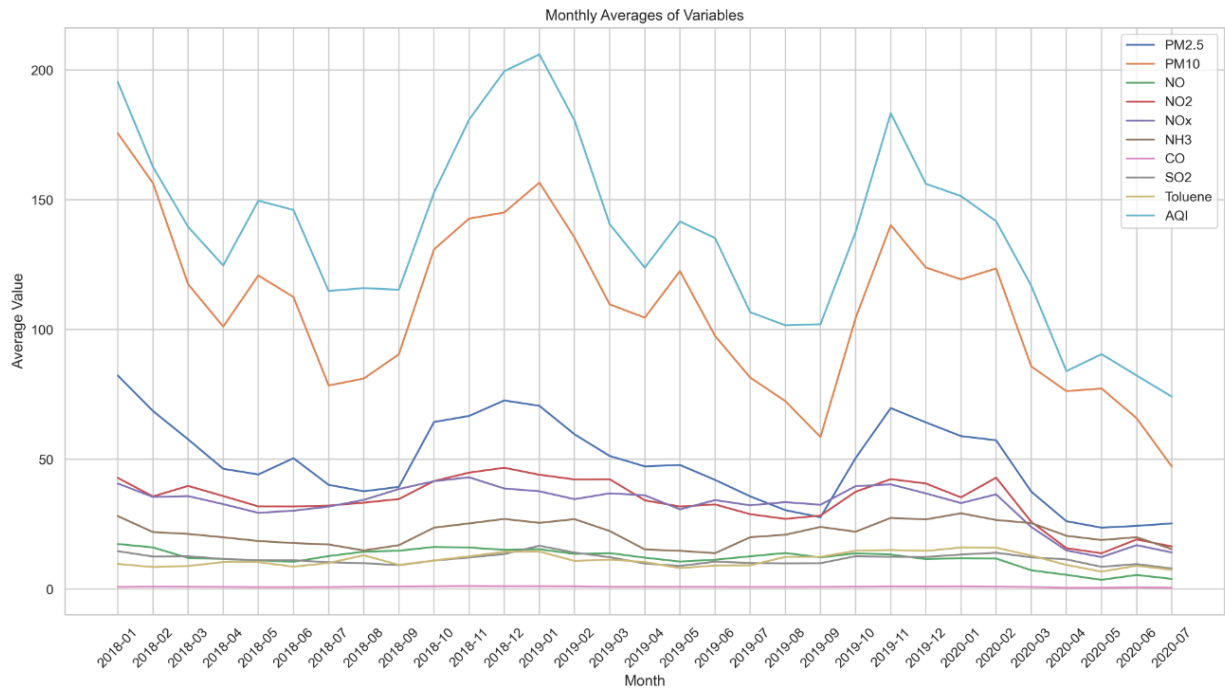
# Extract the month (convert to period format for monthly aggregation)
df_city_day['Month'] = df_city_day['Date'].dt.to_period('M')

# Get the List of numerical columns
numerical_cols = df_city_day.select_dtypes(include='number').columns.tolist()

# Group by 'Month' and calculate the mean for each numerical column
monthly_data = df_city_day.groupby('Month')[numerical_cols].mean()

# Plot the data
plt.figure(figsize=(14, 8), dpi=300)
for col in numerical_cols:
    plt.plot(monthly_data.index.astype(str), monthly_data[col], label=col)

plt.xlabel('Month') # "Months"
plt.ylabel('Average Value') # "Average Value"
plt.title('Monthly Averages of Variables') # "Monthly Averages of Variables"
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```
[1466]: df_city_day['Month'] = df_city_day['Date'].dt.to_period('M')
df_city_day['Year'] = df_city_day['Date'].dt.year

monthly_aqi = df_city_day.groupby('Month')['AQI'].mean()

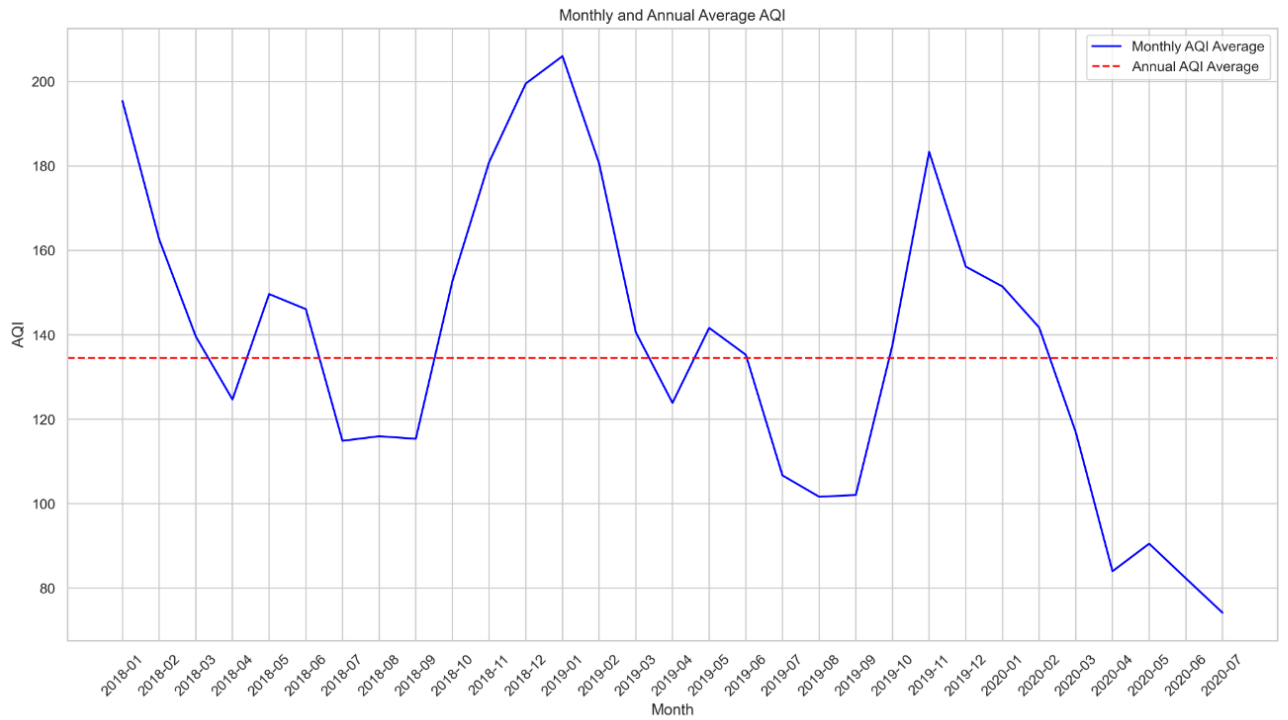
annual_aqi_mean = df_city_day.groupby('Year')['AQI'].mean()

plt.figure(figsize=(14, 8), dpi=300)

plt.plot(monthly_aqi.index.astype(str), monthly_aqi, label='Monthly AQI Average', color='blue')

plt.axhline(y=annual_aqi_mean.mean(), color='red', linestyle='--', label='Annual AQI Average')

plt.xlabel('Month')
plt.ylabel('AQI')
plt.title('Monthly and Annual Average AQI')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```
[1469]: cols = ['AQI', 'PM2.5', 'PM10', 'CO', 'NO', 'NO2']

cmap = plt.get_cmap('Spectral')
color = [cmap(i) for i in np.linspace(0, 1, 8)]
explode = [0.2, 0, 0, 0, 0, 0]

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 8), dpi=300)
axes = axes.flatten()

for ax, col in zip(axes, cols):
    # Group the cities and select the 8 cities with the highest total
    x = df_city_day.groupby('City')[col].sum().sort_values(ascending=False)
    x = x.reset_index('City')

    top_cities = x[:8]
    sizes = top_cities[col].values
    labels = top_cities['City'].tolist()

    # Pasta grafiği oluştur
    wedges, texts, autotexts = ax.pie(sizes, shadow=True, autopct='%1.1f%%',
                                     colors=color, explode=explode,
                                     wedgeprops={'edgecolor': 'black', 'linewidth': 0.3},
                                     labels=labels)

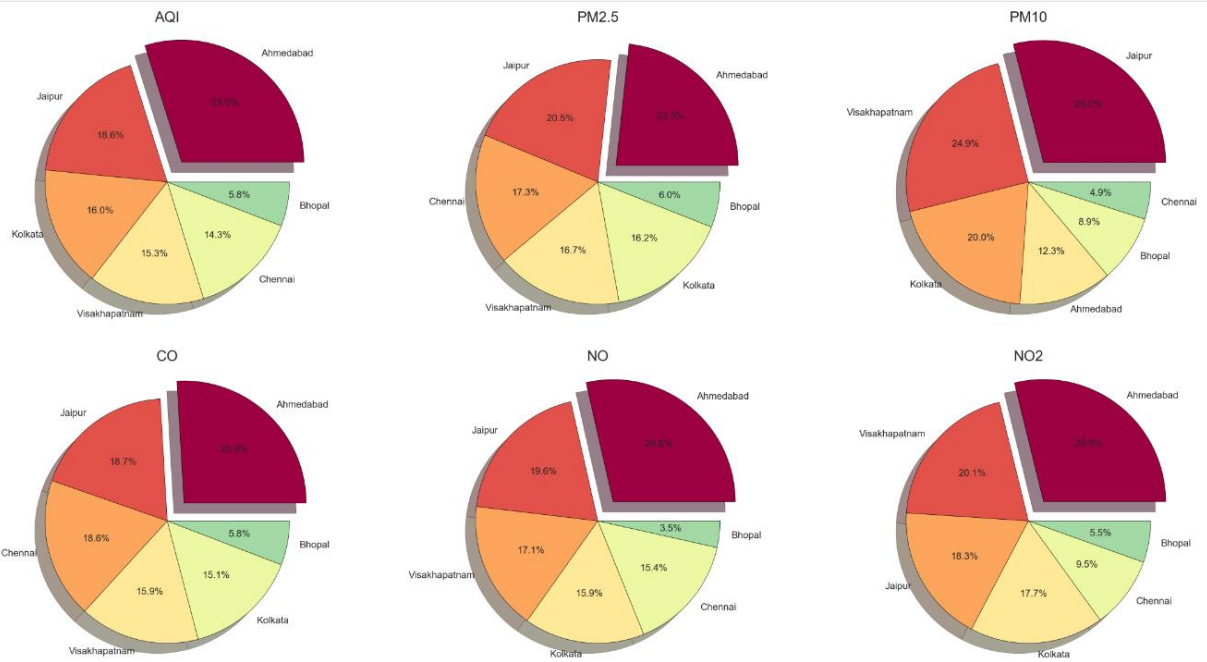
    for text in texts:
        text.set_fontsize(8)

    for autotext in autotexts:
        autotext.set_fontsize(8)

    ax.set_title(f'{col}')

for i in range(len(cols), len(axes)):
    fig.delaxes(axes[i])

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



```
[1472]: fig, axes = plt.subplots(1, 3, figsize=(15, 10), sharey=True)

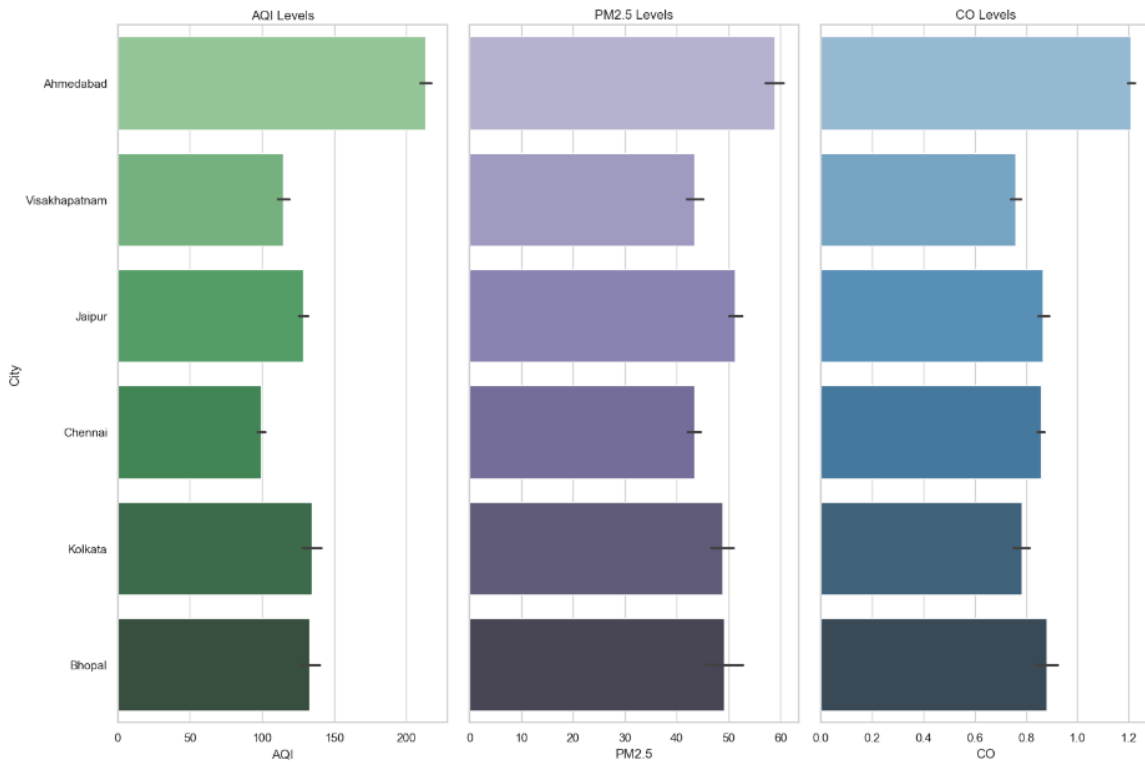
sns.barplot(ax=axes[0], y='City', x='AQI', data=df_city_day, palette='Greens_d')
axes[0].set_title('AQI Levels')

sns.barplot(ax=axes[1], y='City', x='PM2.5', data=df_city_day, palette='Purples_d')
axes[1].set_title('PM2.5 Levels')

sns.barplot(ax=axes[2], y='City', x='CO', data=df_city_day, palette='Blues_d')
axes[2].set_title('CO Levels')

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

SRI



DATA MODELING

```
[1529]: from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet, SGDRegressor
from sklearn.metrics import r2_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import RandomForestRegressor

[1531]: # We combined the numeric and one-hot encoded columns of the training, validation and test datasets.
X_train = train_inputs[numeric_cols + encoded_cols]
X_val = val_inputs[numeric_cols + encoded_cols]
X_test = test_inputs[numeric_cols + encoded_cols]

[1533]: # We made a prediction and filled the predictions with the average of the AQI column
def guess_mean(inputs):
    return np.full(len(inputs), df_full.AQI.mean())

[1535]: guess_mean_train_r2_score=r2_score(guess_mean(X_train), train_target)
print("The error was calculated by assigning average values to the prediction train_r2score: ", guess_mean_train_r2_score)
guess_mean_val_r2_score=r2_score(guess_mean(X_val), val_target)
print("The error was calculated by assigning average values to the prediction val_r2score: ", guess_mean_val_r2_score)
guess_mean_test_r2_score=r2_score(guess_mean(X_test), test_target)
print("The error was calculated by assigning average values to the prediction test_r2score: ", guess_mean_test_r2_score)
The error was calculated by assigning average values to the prediction train_r2score: -6.49373864302651e+30
The error was calculated by assigning average values to the prediction val_r2score: -6.356336594915273e+30
The error was calculated by assigning average values to the prediction test_r2score: -6.35551982704852e+30

[1537]: # We made a prediction and filled the predictions with random value between the lowest and highest values of AQI column.
def guess_random(inputs):
    lo, hi = df_full.AQI.min(), df_full.AQI.max()
    return np.random.random(len(inputs)) * (hi - lo) + lo

[1539]: guess_random_train_r2_score=r2_score(guess_random(X_train), train_target)
print("The error was calculated by assigning random values to the prediction train_r2score: ", guess_random_train_r2_score)
guess_random_val_r2_score=r2_score(guess_random(X_val), val_target)
print("The error was calculated by assigning random values to the prediction val_r2_score: ", guess_random_val_r2_score)
guess_random_test_r2_score=r2_score(guess_random(X_test), test_target)
print("The error was calculated by assigning random values to the prediction test_r2_score : ", guess_random_test_r2_score)
The error was calculated by assigning random values to the prediction train_r2_score: -0.8596218770725828
The error was calculated by assigning random values to the prediction val_r2_score : -0.9744822492948388
The error was calculated by assigning random values to the prediction test_r2_score : -0.8477264614118693
```

```
[1537]: # We made a prediction and filled the predictions with random value between the lowest and highest values of AQI column.
def guess_random(inputs):
    lo, hi = df_full.AQI.min(), df_full.AQI.max()
    return np.random.random(len(inputs)) * (hi - lo) + lo

[1539]: guess_random_train_r2_score=r2_score(guess_random(X_train), train_target)
print("The error was calculated by assigning random values to the prediction train_r2_score: ", guess_random_train_r2_score)
guess_random_val_r2_score=r2_score(guess_random(X_val), val_target)
print("The error was calculated by assigning random values to the prediction val_r2_score: ", guess_random_val_r2_score)
guess_random_test_r2_score=r2_score(guess_random(X_test), test_target)
print("The error was calculated by assigning random values to the prediction test_r2_score: ", guess_random_test_r2_score)

The error was calculated by assigning random values to the prediction train_r2_score: -0.8596218770725828
The error was calculated by assigning random values to the prediction val_r2_score: -0.9744822492948388
The error was calculated by assigning random values to the prediction test_r2_score: -0.8477264614110693
```

```
[1541]: results = []
#we created a dict where we create objects from different models
models = {
    'LinearRegression': LinearRegression(),
    'Ridge': Ridge(),
    'SGDRegressor': SGDRegressor(),
    'ElasticNet':ElasticNet(),
    'Lasso': Lasso(),
    'SVR':SVR(kernel='linear'),
    'GradientBoostingRegressor': GradientBoostingRegressor(random_state=42)
}
```

```
[1543]: def try_model(model,name):
    model.fit(X_train, train_target)
    train_preds = model.predict(X_train)
    val_preds = model.predict(X_val)
    test_preds=model.predict(X_test)

    # The R^2 score indicates how well the model predicted. A value close to 1 indicates that the model predicted perfectly.
    train_r2_score = r2_score(train_target, train_preds)
    val_r2_score= r2_score(val_target, val_preds)
    test_r2_score = r2_score(test_target, test_preds)

    print(f"{name} Model:")
    print("Train r2_score : ", train_r2_score)
    print("Validation r2_score : ", val_r2_score)
    print("Test r2_score : ", test_r2_score)
    print("-" * 40)
    # We add the name of each model and the scores of that model to the result list.
    results.append({'Model': name, 'Train R2 Score': train_r2_score, 'Validation R2 Score': val_r2_score, 'Test R2 Score': test_r2_score})
```

```
[1545]: # Train and test each model
for name, model in models.items():
    try_model(model,name)

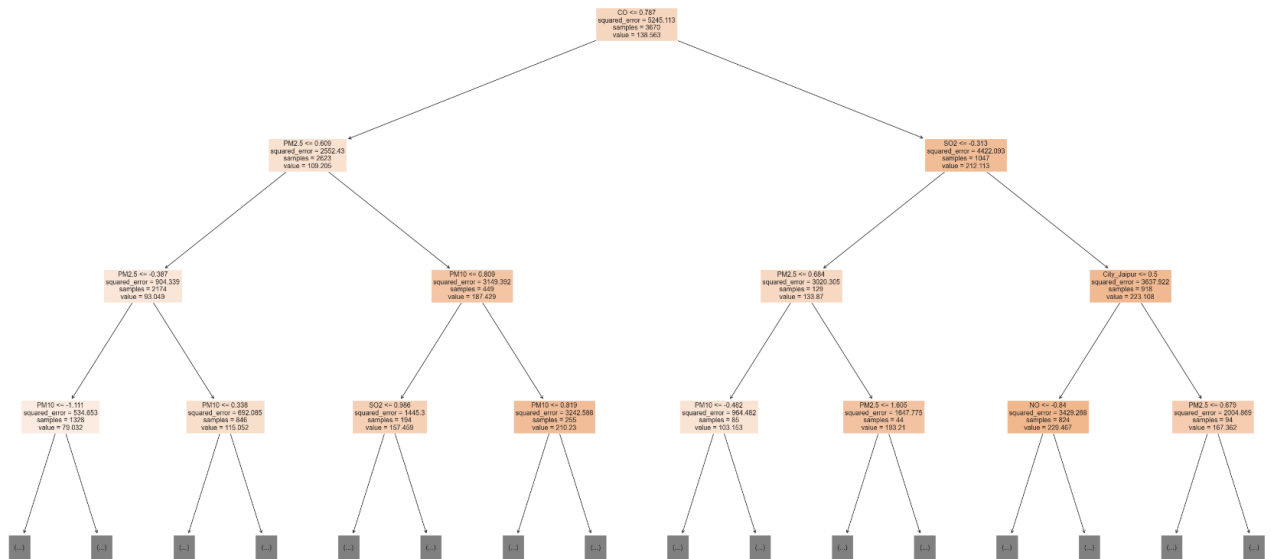
LinearRegression Model:
Train r2_score : 0.7144024257214825
Validation r2_score : 0.6991517366800604
Test r2_score : 0.701951508120177
-----
Ridge Model:
Train r2_score : 0.7144017681711486
Validation r2_score : 0.6992243138431155
Test r2_score : 0.7019839516286439
-----
SGDRegressor Model:
Train r2_score : 0.7125869880377176
Validation r2_score : 0.6988046730971249
Test r2_score : 0.7022241844713835
-----
ElasticNet Model:
Train r2_score : 0.6670505477725734
Validation r2_score : 0.6688643520722499
Test r2_score : 0.6560407301351258
-----
Lasso Model:
Train r2_score : 0.705382958723212
Validation r2_score : 0.6974695103684444
Test r2_score : 0.6944735351166755
-----
SVR Model:
Train r2_score : 0.6867631030121301
Validation r2_score : 0.6784261399590048
Test r2_score : 0.6762780526675916
-----
GradientBoostingRegressor Model:
Train r2_score : 0.865599281465258
Validation r2_score : 0.8436015568383904
Test r2_score : 0.8580094005388339
-----
```

```
[1546]: %time
from sklearn.tree import DecisionTreeRegressor, plot_tree
tree = DecisionTreeRegressor(random_state=42)
try_model(tree, name="DecisionTreeRegressor")
```

DecisionTreeRegressor Model:
 Train r2_score : 0.9995576369004648
 Validation r2_score : 0.6862123401180087
 Test r2_score : 0.9411874370390598

 CPU times: total: 46.9 ms
 Wall time: 54.4 ms

```
[1547]: plt.figure(figsize=(40, 20))
# The max_depth=3 parameter specifies 3 depth levels, filled=True makes the tree nodes colored, the feature_names=numeric_cols+encoded_cols parameter specifies the feature names.
plot_tree(tree, max_depth=3, filled=True, feature_names=numeric_cols+encoded_cols);
```




```
1548]: %%time
# Makes predictions using a forest of decision trees. The n_jobs=-1 parameter specifies the number of cores to be used in training the model. -1 ensures that all cores are used and the model training can be completed in a shorter time.
rf = RandomForestRegressor(random_state=42, n_jobs=-1)
try_model(rf,name="RandomForestRegressor")

RandomForestRegressor Model:
Train r2_score : 0.9785947729440941
Validation r2_score : 0.8598979282706836
Test r2_score : 0.9558670972898635
.....
CPU times: total: 5.42 s
Wall time: 704 ms
```

```
1551]: rf.feature_importances_
```

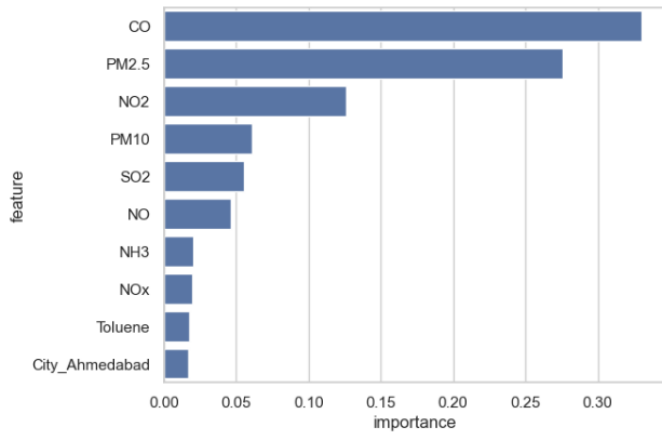
```
1551]: array([0.27610583, 0.06094242, 0.04606549, 0.125725 , 0.01961694,
0.02042506, 0.33051102, 0.05512271, 0.01785045, 0.00469566,
0.01685505, 0.00184024, 0.00207961, 0.01183282, 0.00820607,
0.00212563])
```

```
1552]: importance_df = pd.DataFrame({
'feature': numeric_cols+encoded_cols,
'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)
importance_df.head(10)
```

```
1552]:
```

	feature	importance
6	CO	0.330511
0	PM2.5	0.276106
3	NO2	0.125725
1	PM10	0.060942
7	SO2	0.055123
2	NO	0.046065
5	NH3	0.020425
4	NOx	0.019617
8	Toluene	0.017850
10	City_Ahmedabad	0.016855

```
[1554]: sns.barplot(data=importance_df.head(10), x='importance', y='feature');
```



We found the GradientBoostingRegressor model to be suitable for our data set. We tried to find the best model by optimizing the model hyperparameters using RandomizedSearchCV.

```
[1560]: GBR = GradientBoostingRegressor()
param_distributions = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.001, 0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 0.9, 1.0]
}
```

```
[1562]: # We used RandomizedSearchCV to optimize the hyperparameters of the GradientBoostingRegressor model
from sklearn.model_selection import RandomizedSearchCV
randomized_search = RandomizedSearchCV(
    estimator=GBR,
    param_distributions=param_distributions,
    n_iter=10,
    cv=5,
    verbose=1,
    n_jobs=-1,
    random_state=42
)

randomized_search.fit(X_train, train_target)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

```
[1562]: > RandomizedSearchCV
> estimator: GradientBoostingRegressor
  + GradientBoostingRegressor
  GradientBoostingRegressor()
```

```
[1563]: print("Best Hyperparameters :", randomized_search.best_params_)
print("Best Score :", randomized_search.best_score_)
```

```
Best Hyperparameters : {'subsample': 1.0, 'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': 5, 'learning_rate': 0.1}
Best Score : 0.841011084919575
```

```
[1564]: best_model = randomized_search.best_estimator_
best_model.fit(X_train, train_target)
train_preds = best_model.predict(X_train)
val_preds = best_model.predict(X_val)
test_preds=best_model.predict(X_test)

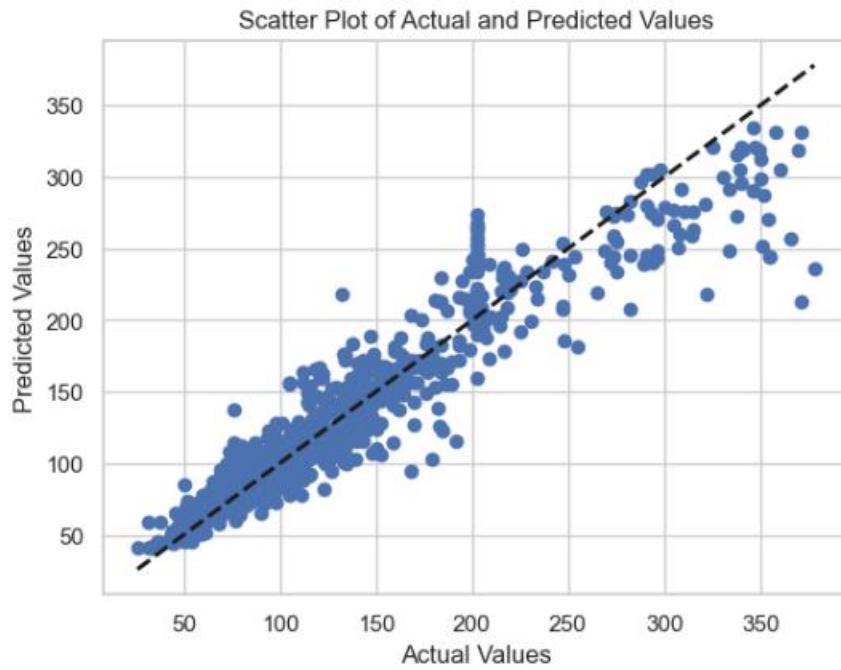
train_r2_score = r2_score(train_target, train_preds)
val_r2_score= r2_score(val_target, val_preds)
test_r2_score = r2_score(test_target, test_preds)

print("Train r2_score : ", train_r2_score)
print("Validation r2_score : ", val_r2_score)
print("Test r2_score : ", test_r2_score)

Train r2_score : 0.9097352043870428
Validation r2_score : 0.856035154076472
Test r2_score : 0.8966827312611687
```

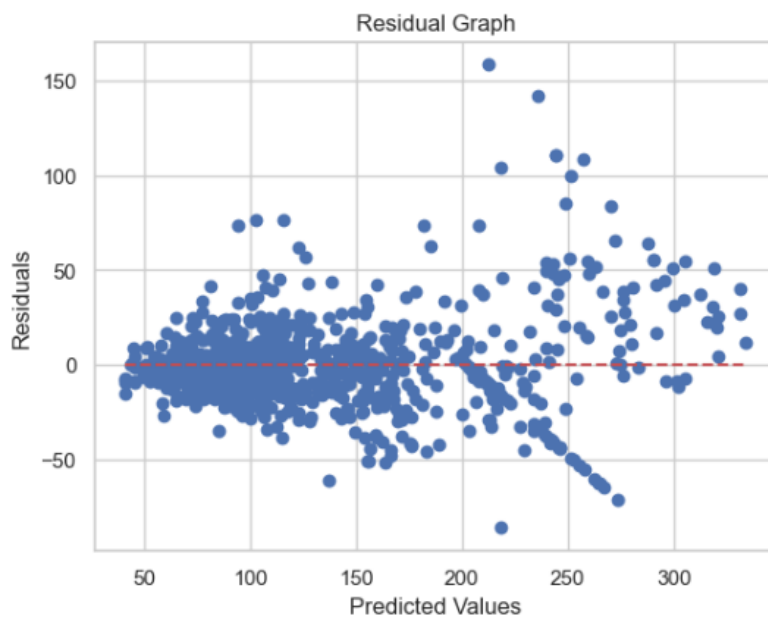
```
[1565]: def plot_actual_predict_graph(test_target,test_preds):
    # Scatter plot
    plt.scatter(test_target, test_preds)
    plt.plot([min(test_target), max(test_target)], [min(test_target), max(test_target)], 'k--', lw=2)
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title('Scatter Plot of Actual and Predicted Values')
    plt.show()

plot_actual_predict_graph(test_target,test_preds)
```



```
[1566]: def plot_residual_graph(test_target, test_preds):
#We assign the difference between the target and the prediction to the residuals variable
residuals = test_target - test_preds
#each point represents (x=predicted value, y=residual of that value).
plt.scatter(test_preds, residuals)
plt.hlines(0, min(test_preds), max(test_preds), colors='r', linestyle='dashed') # we drew y= 0 line
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Graph')
plt.show()

plot_residual_graph(test_target, test_preds)
```



```
[1567]: #We wrote a function to predict when external data is entered.
def predict_input(single_input):
    input_df = pd.DataFrame([single_input])
    input_df['Date'] = pd.to_datetime(input_df['Date'], format='%d-%m-%Y')
    input_df['Year'] = input_df['Date'].dt.year
    input_df[numeric_cols] = imputer.transform(input_df[numeric_cols])
    input_df[numeric_cols] = scaler.transform(input_df[numeric_cols])
    input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
    X_input = input_df[numeric_cols + encoded_cols]
    pred = best_model.predict(X_input)[0]
    return pred
```

```
[1568]: #We created an input as a single data entry.
```

```
new_input = {'City': 'Delhi',
             'PM2.5': 23.2,
             'PM10': 33.2,
             'NO': 10.2,
             'NO2': 4.2,
             'NOx': 10.4,
             'NH3': 52.0,
             'CO': 13.0,
             'SO2': 20.0,
             'Toluene': 89.0,
             'Date': '01-01-2019',
            }

predict_input(new_input)
```

```
[1568]: 159.47089573245458
```

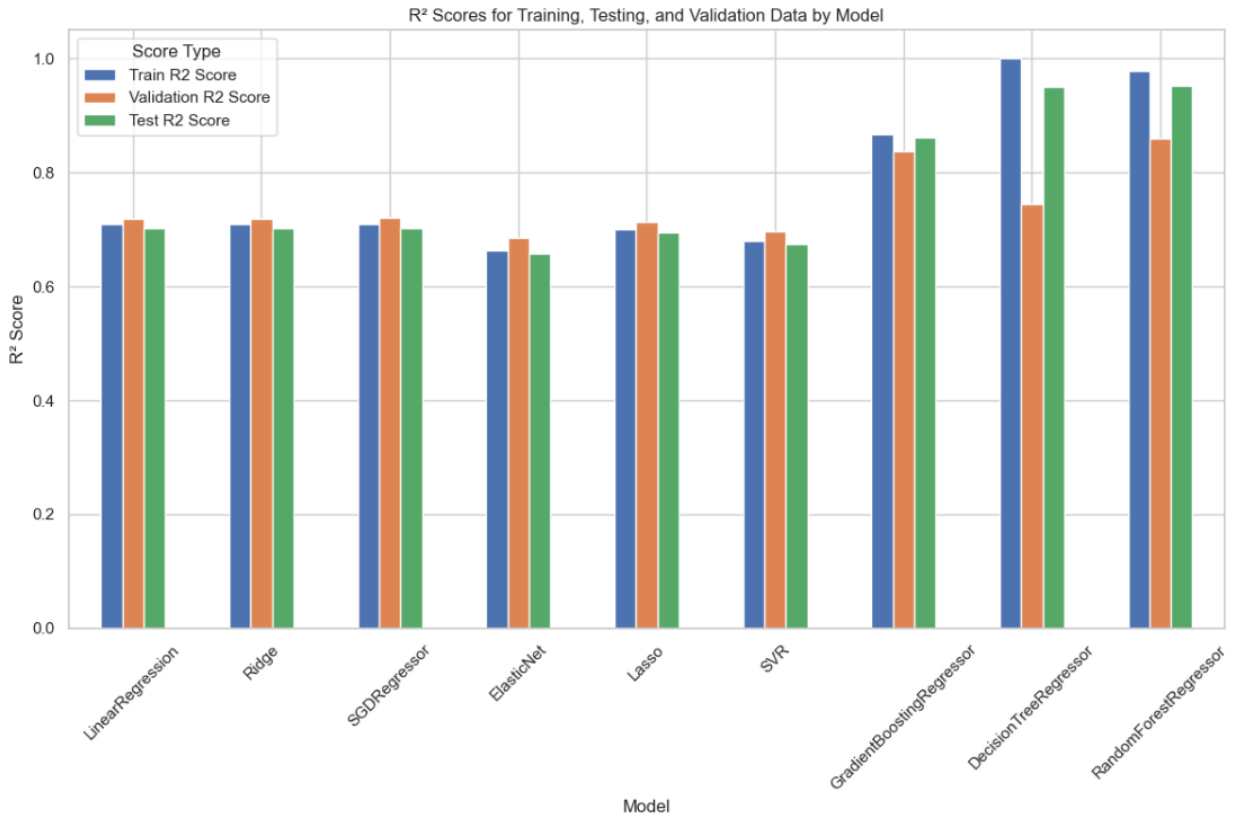
```
[1569]: results_df = pd.DataFrame(results)
```

```
[1570]: results_df
```

```
[1570]:
```

	Model	Train R2 Score	Validation R2 Score	Test R2 Score
0	LinearRegression	0.714402	0.699152	0.701952
1	Ridge	0.714402	0.699224	0.701984
2	SGDRegressor	0.712587	0.698805	0.702224
3	ElasticNet	0.667051	0.668864	0.656041
4	Lasso	0.705383	0.697470	0.694474
5	SVR	0.686763	0.678426	0.676278
6	GradientBoostingRegressor	0.865599	0.843602	0.858009
7	DecisionTreeRegressor	0.999558	0.686212	0.941187
8	RandomForestRegressor	0.978595	0.850898	0.955867

```
[1366]: # Visualizing the results
results_df.set_index('Model').plot(kind='bar', figsize=(12, 8))
plt.title('R2 Scores for Training, Testing, and Validation Data by Model')
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.legend(title='Score Type')
plt.xticks(rotation=45)
plt.tight_layout() # Ensures that all elements (axis labels, titles, subtitles, etc.) are properly placed within the figure area.
plt.show()
```



SRI-VI