Air Quality Prognosis: The Comparative Analysis Using Machine Learning Techniques

A PROJECT REPORT

Submitted by

ABDULLA NADEEM N (2116210701008)
BRUCELIN PRAISE W.S (2116210701044)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY, CHENNAI

MAY 2024

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Thesis titled **"Air Quality Prognosis: The Comparative Analysis Using Machine Learning Techniques**" is the bonafide work of "**ABDULLA NADEEM N (2116210701008),BRUCELIN PRAISE W.S(210701044)"** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does notform part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. S. Vinod Kumar., MTech., Ph.D.

AP(SG)

PROJECT COORDINATOR

Professor

Department of Computer Science and EngineeringRajalakshmi Engineering College

Chennai - 602 105

Submitted to Project Viva-Voce Examination held on_____

Internal Examiner **External Examine**r

ABSTRACT

Since environmental pollution directly affects public health, there has been an increased focus on its prevention and management in recent years. This paper aims to enhance air quality assessment systems to effectively combat air pollution. The study uses the air quality index as the determining criterion and PM 2.5, NO, NO2, NOx, NH3, CO, SO2, O3, Benzene and Toluene as defining parameters. Predictive models are created using a variety of machine learning methods, such as Regression model like Linear Regression, Ensemble methods like Boosting and Bagging using XG Boost and Random Forest algorithms respectively, and Artificial Neural Network. The study evaluates these algorithms' precision and generalizability. The identification of trends and the creation of effective pollution mitigation methods are facilitated by the use of machine learning for AQI prediction. Furthermore, the study incorporates data from multiple geographic regions to ensure robustness and relevance across different environmental contexts. It also examines the impact of seasonal variations on pollutant levels and their subsequent effect on the accuracy of the predictive models. Additionally, the research explores the integration of real-time sensor data to enhance the responsiveness of air quality monitoring systems. By leveraging feature importance techniques, the study identifies key pollutants that significantly influence the AQI, thereby providing targeted insights for policymakers. The results demonstrate that ensemble methods, particularly XG Boost and Linear Regression model outperform other models in terms of accuracy and computational efficiency. The findings underscore the potential of advanced machine learning techniques in developing proactive and adaptive air quality management strategies**.**

ACKNOWLEDGMENT

First, we thank the almighty god for the successful completion of the project. Our sincere thanks to our chairman **Mr. S. Meganathan B.E., F.I.E.,** for his sincere endeavor in educating us in his premier institution. We would like to express our deepgratitude to our beloved Chairperson **Dr. Thangam Meganathan Ph.D.,** for her enthusiastic motivation which inspired us a lot in completing this project and Vice Chairman **Mr. Abhay Shankar Meganathan B.E., M.S.,** for providing us with the requisite infrastructure.

We also express our sincere gratitude to our college Principal,

Dr. S. N. Murugesan M.E., PhD., and **Dr. P. KUMAR M.E., PhD, Director computing and information science, and Head Of Department of Computer Science and Engineering** and our project coordinator **Dr. S. Vinod Kumar., MTech., Ph.D.** for her encouragement and guiding us throughout the project towards successful completion of this project and to our parents, friends, all faculty members and supporting staffs for their direct and indirect involvement in successful completionof the project for their encouragement and support.

ABDULLA NADEEM N

BRUCELIN PRAISE W.S

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Quality of Air is a critical environmental factor that profoundly influences public health and wellbeing. In recent years, heightened concerns surrounding environmental pollution, particularly air pollution, have underscored the urgent need for effective prevention and control measures. The detrimental impacts of poor air quality, exacerbated by phenomena such as air pollutants, are increasingly evident, posing significant challenges to individuals' health and everyday activities. As a result, there is an urgent need for sophisticated approaches that can precisely measure and forecast air quality situations so that prompt actions may be taken to reduce pollution and protect human health.Traditional methods of air quality evaluation often rely on simplistic indices that may not capture the complexities of pollutant interactions and their impacts on human health comprehensively. As such, there is a growing emphasis on optimizing air quality evaluation systems through the integration of advanced technologies, particularly machine learning (ML) techniques. ML offers a promising avenue for enhancing the accuracy and predictive capabilities of air quality assessment models by leveraging large datasets encompassing diverse pollutant parameters. By discerning intricate patterns and relationships within these datasets, ML algorithms can yield insights that facilitate more robust air quality predictions, thus empowering policymakers and environmental agencies to formulate targeted interventions and regulatory measures.This research endeavors to contribute to the ongoing efforts aimed at advancing air quality assessment methodologies through the application of ML techniques. By employing a diverse array of ML algorithms, including Regression models, Ensemble methods, and Artificial Neural Networks, this study seeks to establish predictive models capable of accurately forecasting Air Quality Index (AQI) values.

1.1 PROBLEM STATEMENT

The increasing threat posed by air pollution to public health and the environment necessitates the development of advanced air quality assessment systems for effective pollution management. This study aims to address this pressing concern by focusing on enhancing the predictive capabilities of air quality assessment models. The problem statement revolves around the need to develop accurate and reliable predictive models for forecasting air quality index (AQI) levels. This entails leveraging machine learning techniques to analyze historical air quality data and various pollutant parameters, including PM 2.5, NO, NO2, NOx, NH3, CO, SO2, O3, Benzene, and Toluene. The challenge lies in identifying the most effective machine learning algorithms and methodologies for AQI prediction, considering factors such as computational efficiency, model interpretability, and generalizability across different geographic regions and seasonal variations. Additionally, the study aims to explore the integration of real-time sensor data and feature importance analysis to enhance the responsiveness and accuracy of air quality monitoring systems. By addressing these challenges, the research seeks to contribute to the development of proactive and adaptive strategies for mitigating air pollution and safeguarding public health.

1.1 SCOPE OF THE WORK

The scope of work outlined in the provided description encompasses several essential components to achieve the objectives of enhancing air quality assessment systems and combating air pollution effectively. The project's first phase involves comprehensive data collection, focusing on gathering air quality index (AQI) data along with defining parameters such as PM 2.5, NO, NO2, NOx, NH3, CO, SO2, O3, Benzene, and Toluene from various geographic regions.The next phase entails the development and implementation of predictive models using a range of machine learning techniques, including Regression models like Linear Regression, Ensemble methods such as Boosting and Bagging utilizing XG Boost and Random Forest algorithms, and ANN.

1.2 AIM AND OBJECTIVES OF THE PROJECT

The aim of this study is to enhance air quality assessment systems through the application of advanced machine learning techniques, with a primary focus on predicting the air quality index (AQI) levels. The objectives include developing predictive models using regression, ensemble methods, and artificial neural networks, evaluating their precision and generalizability, assessing the impact of seasonal variations on pollutant levels, integrating real-time sensor data for improved monitoring, and identifying key pollutants influencing AQI. By achieving these objectives, the study aims to provide valuable insights for policymakers and stakeholders to develop proactive strategies for mitigating air pollution and safeguarding public health and the environment.

1.2 RESOURCES

The resources required for this study include access to comprehensive datasets containing historical air quality measurements, meteorological data, and pollutant parameters such as PM 2.5, NO, NO2, NOx, NH3, CO, SO2, O3, Benzene, and Toluene. Additionally, computational resources are essential for implementing and training machine learning models, including regression algorithms, ensemble methods such as boosting and bagging, and artificial neural networks. Access to geographic information systems (GIS) software may be necessary for spatial analysis, while expertise in data preprocessing and statistical analysis is crucial for ensuring data quality and reliability.

1.3 MOTIVATION

The motivation behind this study stems from the critical need to address the escalating threat of air pollution to public health and the environment. With air quality deterioration becoming a pressing global concern, there is a clear imperative to enhance existing assessment systems for more effective pollution management.

CHAPTER 2
LITRETURE SURVEY

The literature survey for this study encompasses a comprehensive review of existing research on air quality assessment, machine learning applications, and pollution mitigation strategies. Numerous studies have explored the use of machine learning techniques for air quality prediction, highlighting the efficacy of regression models, ensemble methods, and artificial neural networks in forecasting pollutant concentrations and air quality indices. Additionally, research has examined the influence of various factors such as meteorological conditions, geographical features, and emission sources on air quality dynamics. Studies focusing on the integration of real-time sensor data and satellite imagery have provided valuable insights into improving the accuracy and responsiveness of air quality monitoring systems. Furthermore, the literature review extends to pollution mitigation strategies, including regulatory measures, technological innovations, and community-based initiatives aimed at reducing pollutant emissions and mitigating the adverse impacts of air pollution on public health and the environment. By synthesizing findings from diverse literature sources, this study aims to build upon existing knowledge and contribute novel insights to the field of air quality assessment and management.

"A Survey on Machine Learning Techniques for Air Pollution Prediction and Control" by Wilson et al.This survey paper by Wilson et al. provides a comprehensive overview of machine learning techniques used for air pollution prediction and control, with a specific focus on air quality index (AQI) prediction. The authors review the application of supervised and unsupervised learning methods, as well as ensemble techniques, in modeling air pollution dynamics. They discuss the integration of satellite data, ground-based measurements, and atmospheric modeling in developing accurate AQI prediction models. Additionally, the paper examines the challenges and future research directions in leveraging machine learning for air quality management and environmental policy-making.

CHAPTER 3

SYSTEM DESIGN

3.1 GENERAL

In general, air quality assessment is crucial for safeguarding public health and the environment. By accurately measuring pollutant concentrations and predicting air quality levels, stakeholders can implement effective pollution mitigation strategies. Machine learning techniques offer promising tools for enhancing air quality assessment systems, enabling more accurate predictions and timely interventions. Integrating real-time sensor data and considering factors such as seasonal variations and geographical influences further improves the reliability of air quality forecasts. Ultimately, advancements in air quality assessment contribute to healthier environments and better quality of life for communities worldwide.

3.2 METHODOLOGY

DATA COLLECTION

This study uses data on air quality in Chennai. The daily historical data on Chennai's air quality, which verify its reliability and authority, originates from the Central Control Room for Air Quality Management. The webpage of the CPCB, situated at https://cpcb.nic.in, provides the information to the general public.

DATA PREPROCESSING:

In the data preprocessing phase, the columns with missing values are identified and appropriate imputation technique is applied. For numeric features representing air pollutants such as PM2.5, NO, NO2, and others, the missing values are imputed using simple imputation methods like median or mean or mode and also the sophisticated imputation techniques

such as KNN imputation, ensuring that the imputed values preserve the distribution of the data. The data points with absence of AQI (dependent variable) is dropped to improve accuracy of the model. Table 1 displays the subset of the dataset.

Date	PM2.5	NO	NO_2	NO_x	NH_3	CO	SO_2	O_3	Benzene	Toluene	AQI
27-06-2020	26.42	7.25	12.96	19.59	33.2	1.1	7.29	68.51	0.1	0.07	95
28-06-2020	25.93	7.81	10	16.39	35.98	0.76	6.48	77.45	0.09	0	98
29-06-2020	21.3	7.65	9.69	16.74	34.07	0.96	6.62	62.57	0.09	0.01	104
30-06-2020	24.14	8.42	12.38	20.29	34.17	1.05	7.5	68.75	0.17	0.16	110
01-07-2020	15.95	6.22	10.72	16.44	33.52	1.02	9.23	48.37	0.09	0	92

Table 1 – Air Quality Dataset

CORRELATION MATRIX:

A correlation matrix is essential for comprehending the connections between different contaminants and their effects on air quality in air quality prediction regression tasks. It assists in determining which pollutants have a substantial impact on air quality by looking at correlations between independent factors (like pollutant concentrations) and the dependent variable (like the AQI). Finding significant connections can help direct the feature selection process, guaranteeing that the model contains pertinent predictors.

It also helps identify multicollinearity, which is a prevalent problem in air quality datasets including highly linked contaminants. This makes it possible to create regression models that are more precise and understandable.

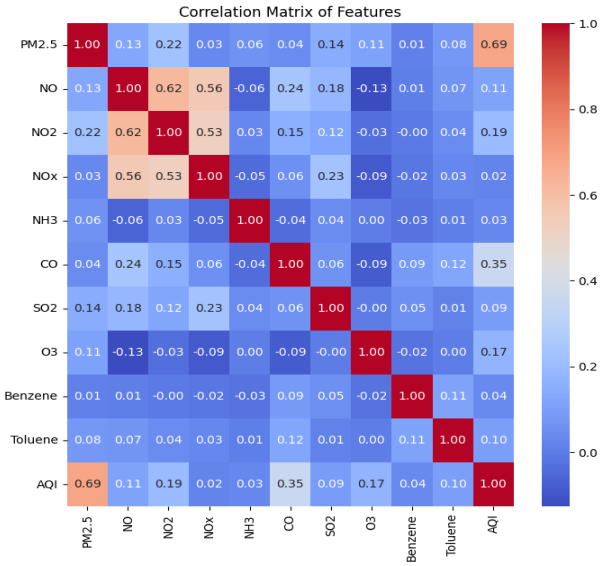

Fig 1: Heatmap representing correlation between features

MODEL SELECTION:

Based on the various prediction contents, machine learning models can be classified as regression models or classification models. Whereas the regression model is mostly used to predict continuous values, the classification model primarily predicts discrete labels. Since the air quality index is a continuous number, it is modelled using a regression approach. In this study, four algorithms have been used: Linear Regression, XG Boost, Random Forest Regression, and ANN. Each of the aforementioned algorithms are applied to match the AQI in this paper. When comparing the predictions of each model on the AQI, daily average concentration of PM 2.5, NO, NO2, NOx, NH3, CO, SO2, O3, Benzene, and Toluene in the data table is used as the input feature of the model, and the value of AQI is used as the label.

LINEAR REGRESSION:

In Linear Regression, the core task revolves around modelling the relationship between predictor variables (e.g., pollutant concentrations) and the target variable (AQI). Linear Regression seeks to fit a linear equation to the data, where each predictor variable is multiplied by a coefficient and summed to predict the AQI. During model training, the algorithm learns these coefficients by minimizing the difference between the predicted and actual values of AQI. Once trained, the model can predict the AQI for new input data by applying the learned coefficients to the corresponding predictor variables.

The following is the linear equation used by a Linear Regression model:

$$y = c + m_1x_1+m_2x_2+m_3x_3+\ldots+m_nx_n$$

y is the predicted AQI value.

c is the Y intercept.

m_1, m_2, m_3,…, m_n - The coefficients corresponding to the independent variables of x_1, x_2, x_3, x_n respectively.

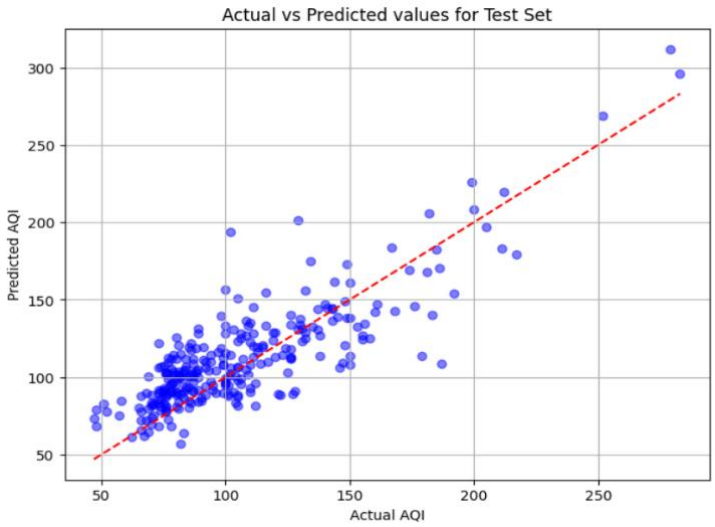

Fig 2: Linear Regression

RANDOM FOREST REGRESSION:

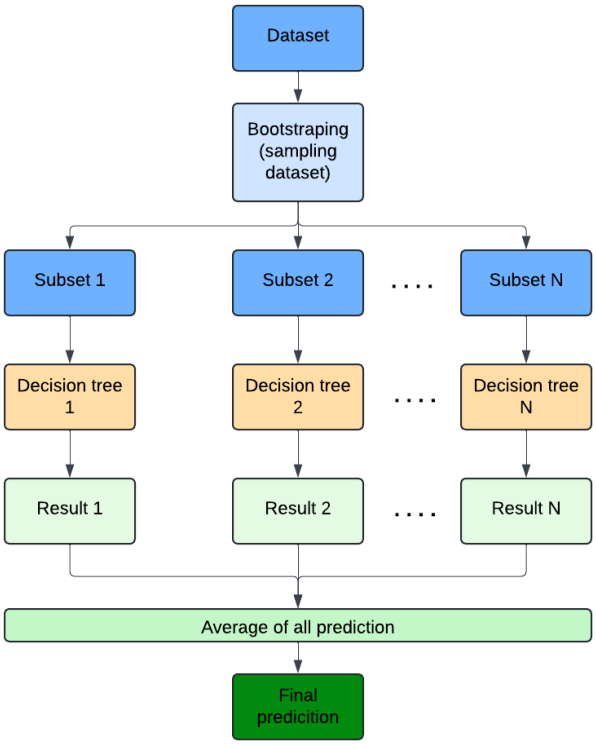

Fig 3. Random Forest Regression model architecture

Using an ensemble of decision trees to generate precise predictions is the fundamental function of a Random Forest Regression model in forecasting the Air Quality Index (AQI). Individual predictions are made by each decision tree in the ensemble once it has separately learned a portion of the features. Using random feature selection and bootstrap samples of the input, the model builds a large number of decision trees during training. In order to arrive at a final prediction, the model combines the predictions of each individual tree. Randomness in data sampling and feature selection reduces overfitting and increases the resilience of the model. Random Forest Regression does not rely on a particular mathematical equation, in contrast to conventional linear models. Rather, it generates a more precise and reliable estimate of the AQI by aggregating the predictions of several trees. Random Forest Regression (RFR) is a potent technique for predicting AQI because of its ability to capture intricate nonlinear correlations.

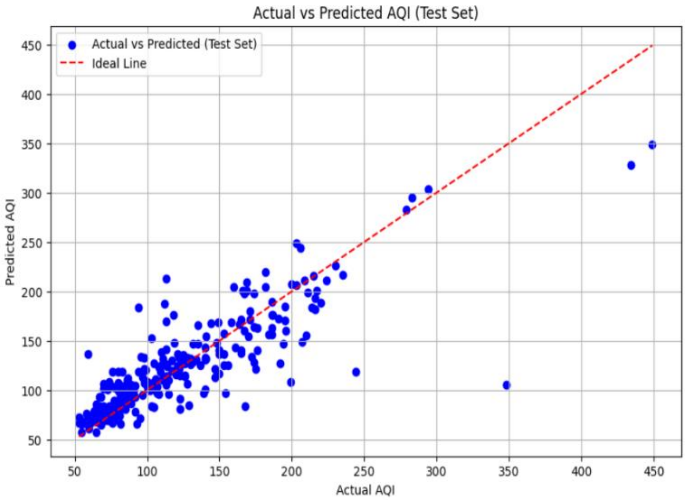

Fig 4: Random Forest Regression prediction

XG BOOST REGRESSION:

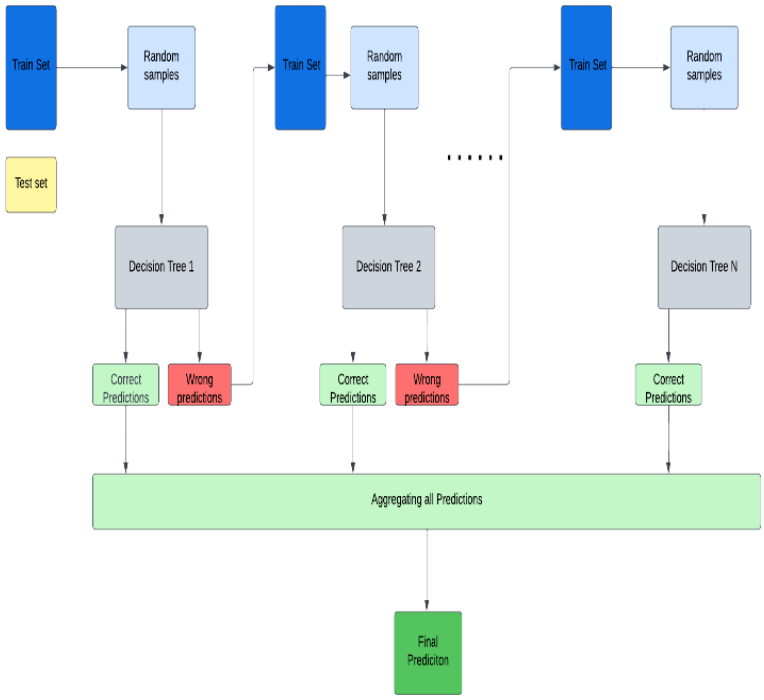

Fig 5. XG Boost Regression model architecture

The air quality index (AQI) is predicted by the ensemble learning technique XG Boost regression by combining the predictions of several decision trees. It creates a series of decision trees iteratively, with each new tree fixing the mistakes of the preceding ones. XG Boost reduces a loss function during training by optimizing the sum of the gradients of the loss function with respect to the predictions. To ensure robustness and avoid overfitting, the weighting of each tree's contribution to the final prediction is determined by its performance and regularization parameters. To increase model performance iteratively, XG Boost combines tree-pruning and additive training techniques. The predictive equation generates the final AQI prediction by integrating the predictions from each individual tree and adjusting for learning rate. This method is an effective tool for AQI prediction problems because it can handle complex nonlinear interactions between input features and target variables.

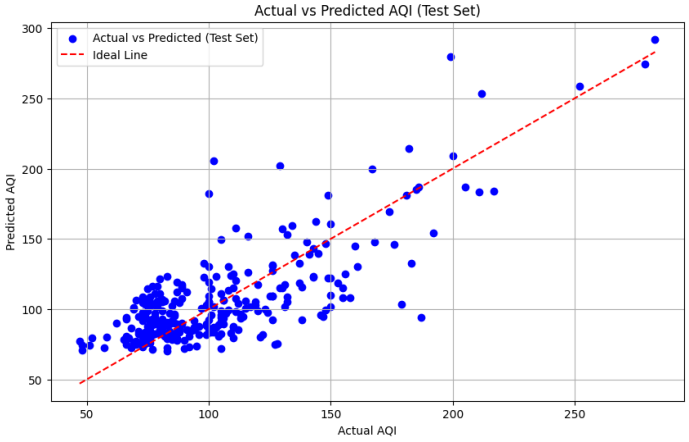

Fig 6: XG Boost Regressor prediction

ARTIFICIAL NEURAL NETWORK (ANN):

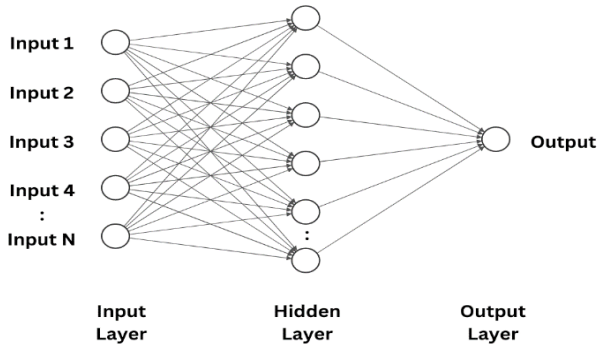

Fig 7. ANN architecture

In an ANN, input data is processed by several interconnected layers of neurons, which then provide output predictions. Features like pollution concentrations are fed into the input layer for AQI prediction, and then intricate patterns are extracted by hidden layers that follow. Inputs are subjected to weights and biases by individual neurons, and non-linearity is introduced by activation functions. Backpropagation modifies weights during training in order to reduce prediction errors.

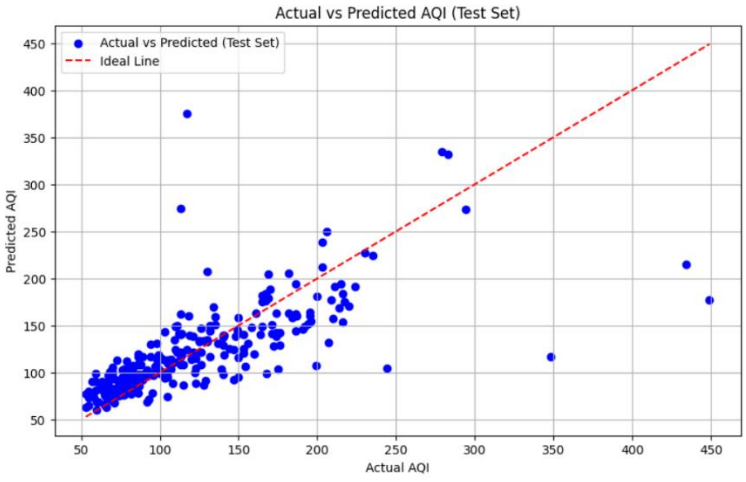

Fig 8. ANN prediction

MODEL TRAINING

The dataset underwent meticulous partitioning into training, validation, and testing sets to facilitate AQI prediction with precision**.** Originally, the dataset was divided into two categories: a temporary dataset, which had the remaining 30% of cases, and training data, which included 70% of all cases. Subsequently, test and validation sets, each comprising 15% of the original dataset, were equally divided from the temporary dataset. This stratified splitting technique lessened bias and enhanced the model's generalizability by guaranteeing that each subset had a representative distribution of data points. The test set made it easier to assess performance on untested data, while the training set was used to train the model's parameters. Finally, the model's robustness was evaluated and its hyperparameters were adjusted using the validation set.

MODEL EVALUATION:

The degree of fit of relevant information and the accuracy of predicted values are the two metrics used to assess the effectiveness of the regression method. To evaluate the effectiveness of AQI prediction models, evaluation measures including Mean Absolute Error, Mean Squared Error, Root Mean Squared Error, and are essential. In the MSE computation, which finds the average of squared differences between actual and expected AQI values, larger errors are given more weight. RMSE, or the square root of MSE, which is derived from MSE, represents the average magnitude of errors in the same units as the AQI.The mean absolute difference (MAE) between the actual and predicted AQI values gives a more intuitive understanding of prediction accuracy. Lower values of MSE, RMSE, and MAE imply better model accuracy in accurately anticipating AQI values. Effective management of air quality and decision-making processes depend on this. The R-squared (R2) score, also known as the coefficient of determination, is the proportion of the variance in the dependent variable (AQI) that can be predicted from the independent variables (features) in the model. Its range is 0 to 1, where 1 indicates a perfect fit, meaning that all variations in the dependent variable are explained by the independent factors.

Since a higher R2 score in AQI prediction indicates that the model captures a larger percentage of the variance in AQI values, it suggests a better fit and more reliable predictions. By assessing the model's overall goodness-of-fit and ability to explain the variability in AQI data, the R2 score aids in the selection and interpretation of models used in air quality studies.

Mean Squared Error:

$$\text{MSE}= \frac{1}{N}\sum_{i=1}^{N}(yi-\hat{y}i)\text{^}2$$

Root Mean Squared Error:

$$\text{RMSE} = \sqrt{MSE}$$

Mean Absolute Error:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|yi-\hat{y}i|$$

R-Squared Error:

$$\text{R-Squared} = 1-\frac{\sum_{i=1}^{n}(yi-\hat{y}i)\text{^}2}{\sum_{i=1}^{n}(yi-\bar{y})\text{^}2}$$

n - Total number of samples.

y_i - actual value of AQI (dependent variable for a AQI prediction)

$\hat{y}_i$ - predicted value of the AQI (dependent variable for AQI prediction)

$\bar{y}$ – average of the actual values.

The above evaluation metrics are most commonly used for evaluating the performance of regression models.

MODEL SCORES

Regression models	Test set score				Validation set score			
	MSE	RMSE	MAE	R-Squared	MSE	RMSE	MAE	R-Squared
Linear Regression	**474.52**	**21.78**	**16.98**	0.65	511.88	22.62	18.07	0.65
Random Forest Regression	898	29.96	18.64	**0.72**	682.21	26.11	17.47	0.67
XG Boost Regression	599.51	24.48	18.66	0.56	**404.88**	**20.12**	**14.93**	**0.72**
ANN	1590.9	39.88	22.29	0.51	869.67	29.49	19.81	0.58

Table 2 – Score of the models

3.2 DEVELOPMENTAL ENVIRONMENT

3.3.1 HARDWARE REQUIREMENTS

The hardware requirements for this project primarily include a standard computer system with sufficient processing power and memory to handle data preprocessing, model training, and evaluation tasks. A modern multicore processor and ample RAM are recommended to support the computational demands of machine learning algorithms.

Table 3.1 Hardware Requirements

COMPONENTS	SPECIFICATION
PROCESSOR	Intel Core i5
RAM	8 GB RAM
GPU	NVIDIA GeForce GTX 1650
MONITOR	15” COLOR
HARD DISK	512 GB
PROCESSOR SPEED	MINIMUM 1.1 GHz

3.3.2 SOFTWARE REQUIREMENTS

The software requirements for this project encompass a range of tools and libraries for data preprocessing, model development, and analysis. Python programming language serves as the primary platform for implementing machine learning algorithms, utilizing libraries such as pandas and NumPy for data manipulation, scikit-learn for model building, and TensorFlow or PyTorch for deep learning tasks. Jupyter Notebook or similar integrated development environments facilitate interactive coding and result visualization. Geographic information system (GIS) software may be utilized for spatial analysis, while version control systems like Git ensure collaborative development and reproducibility.

CHAPTER 4

PROJECT DESCRIPTION

4.1 METHODOLODGY

The methodology for this study involves several key steps aimed at enhancing air quality assessment through machine learning techniques. Firstly, comprehensive datasets containing historical air quality measurements and relevant pollutant parameters are collected and preprocessed to ensure data quality. Next, various machine learning algorithms, including regression models, ensemble methods, and artificial neural networks, are implemented and trained on the prepared dataset. The performance of these models is evaluated using appropriate metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

4.2 MODULE DESCRIPTION

The module description for this project involves several interconnected components to facilitate the development and evaluation of machine learning models for air quality assessment. The data preprocessing module focuses on collecting, cleaning, and standardizing datasets containing historical air quality measurements and pollutant parameters. The model development module encompasses the implementation of various machine learning algorithms, including regression models, ensemble methods, and neural networks, to predict air quality index (AQI) levels. The model evaluation module assesses the performance of these models using metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), considering factors like seasonal variations and geographical influences.

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 OUTPUT

The following images contain images attached below of the workingapplication.

MODEL SCORES

Regression models	Test set score				Validation set score			
	MSE	RMSE	MAE	R-Squared	MSE	RMSE	MAE	R-Squared
Linear Regression	**474.52**	**21.78**	**16.98**	0.65	511.88	22.62	18.07	0.65
Random Forest Regression	898	29.96	18.64	**0.72**	682.21	26.11	17.47	0.67
XG Boost Regression	599.51	24.48	18.66	0.56	**404.88**	**20.12**	**14.93**	**0.72**
ANN	1590.9	39.88	22.29	0.51	869.67	29.49	19.81	0.58

Table 2 – Score of the models

5.2 RESULT

From the Table 2, Based on the test set score, It is found that the Linear Regression (LR) has lower MSE, RMSE, and MAE ensuring generalization ability compared to the other models. This suggests that LR is performing better in terms of prediction accuracy on the test data compared to the other models (Random Forest Regression, XG Boost Regression, and ANN) for this particular dataset. If R-Squared is chosen as the evaluation metric then RFR suits best for AQI Prediction.

Similarly, XG Boost appears to have the lowest MSE, RMSE, and MAE based on the validation set score, suggesting greater performance in terms of prediction accuracy. Additionally, it has the highest R-squared score, suggesting that it explains the highest proportion of the variance in the test set compared to the other models. So, XG Boost has good fitting ability.There's indeed a trade-off between LR, RFR, and XG Boost in the test set scores. LR performs consistently well across all metrics, while RFR exhibits higher R-squared but with higher error metrics compared to LR. XG Boost strikes a balance between LR and RFR, showing relatively good performance but with slightly higher error metrics than LR. Meanwhile, ANN shows the highest error metrics and lower R-squared compared to LR, RFR, and XG Boost, indicating less effective performance in this context.So, if R-Squared is chosen as evaluation metric in both test set and evaluation set score the ensemble models such as XG Boost Regression and Random Forest Regression suits best for AQI prediction..

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

It is clear from the thorough examination of the many regression models used to predict AQI that each model has advantages and disadvantages in distinct areas. XG Boost Regression, Random Forest Regression and Linear Regression produce competitive results, especially when it comes to generalization ability, suggesting that they have the ability to make correct predictions. But when compared to other models, the Artificial Neural Network (ANN) performs worse, indicating the need for additional improvement or different strategies. Because of its balanced performance and simplicity, XG Boost and Linear Regression are the best appropriate model for AQI prediction when taking into account both the fitting ability and generalization capacity.

6.2 FUTURE ENHANCEMENT

Future enhancements for AQI prediction using machine learning models can significantly improve their accuracy and reliability. Incorporating additional data sources such as satellite imagery, traffic data, and weather conditions can provide a more comprehensive view of factors affecting AQI. Robustness to missing data through advanced imputation techniques is essential. Ensuring model generalization by using cross-validation and regularization techniques can prevent overfitting. Advanced feature engineering techniques, including polynomial features and interaction terms, can capture complex relationships in the data. Temporal dependencies can be better modeled using Long Short-Term Memory (LSTM) networks or Temporal Convolutional Networks (TCN). Extensive hyperparameter optimization with Grid Search, Random Search, or Bayesian Optimization can further enhance model performance. Exploring advanced ensemble methods like stacking, blending, and bagging can leverage the strengths of different models.

APPENDIX

SOURCE CODE:

ANN

```
# -*- coding: utf-8 -*-
"""ANN.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1tH2bIVJ6W_ksP8gAASRfyyP3RqjY5cAd
"""

import numpy as np
import pandas as pd

df = pd.read_excel('/content/CHN_AQI.xlsx')

df.tail()

df.info()

df.isnull().sum()

## df = df.drop(df.columns[[0, 4, 6]], axis=1)
df = df.drop(columns=['City'])

df.info()

from sklearn.impute import SimpleImputer, KNNImputer

# Load your dataset (assuming it's already loaded into a DataFrame named 'df')

# Drop the 'Xylene' column as it has no non-null values
df.drop(columns=['Xylene','PM10'], inplace=True)

# Define columns for statistical imputation and KNN imputation
statistical_impute_cols = ['PM2.5', 'NO', 'NO2', 'NOx', 'CO', 'SO2', 'O3']
knn_impute_cols = ['NH3', 'Benzene', 'Toluene']

# Impute missing values using statistical measures (mean, median, or mode)
statistical_imputer = SimpleImputer(strategy='mean')  # You can change the strategy if needed
```

```
df[statistical_impute_cols] = statistical_imputer.fit_transform(df[statistical_impute_cols])

# Impute missing values using K-nearest neighbors (KNN) imputation
knn_imputer = KNNImputer(n_neighbors=5)  # You can adjust the number of neighbors as needed
df[knn_impute_cols] = knn_imputer.fit_transform(df[knn_impute_cols])

# Drop rows with missing AQI values
df.dropna(subset=['AQI', 'AQI_Bucket'], inplace=True)

# Now your dataset should have missing values imputed and rows with missing AQI values dropped

df.info()

df.head()

df.tail()

# Maximum value in the AQI column
max_aqi = df['AQI'].max()
print("Maximum AQI value:", max_aqi)

# Unique values in the AQI_Bucket column
unique_aqi_buckets = df['AQI_Bucket'].unique()
print("Unique AQI buckets:", unique_aqi_buckets)

df.info()

df = df.set_index('Date')
df.head()

# Define mapping dictionary for AQI buckets
aqi_bucket_mapping = {
   'Good': 0,
   'Satisfactory': 1,
   'Moderate': 2,
   'Poor': 3,
   'Very Poor': 4,
   'Severe': 5
}

# Map AQI buckets to numerical values
df['Encoded_AQI_Bucket'] = df['AQI_Bucket'].map(aqi_bucket_mapping)
df.head()

df = df.drop(columns=['AQI_Bucket', 'Encoded_AQI_Bucket'])
```

```
df.tail()

df.info()

# Assuming your dataset is named df
X = df.drop(columns=['AQI'])  # Independent variables (features)
y = df['AQI']  # Dependent variable (target)

# Check the shapes of X and y
print("Shape of X:", X)
print("Shape of y:", y)

import seaborn as sns
import matplotlib.pyplot as plt
# Calculate the correlation matrix
corr_matrix = df.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", annot_kws={"size": 10})
plt.title('Correlation Matrix of Features')
plt.show()

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Split the dataset into training, test, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Standardize the features (optional but recommended for neural networks)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_val_scaled = scaler.transform(X_val)

# Build the neural network model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(32, activation='relu'),
```

```
    Dense(1)  # Output layer with single neuron for regression task
])

# Compile the model
opt = Adam(learning_rate=0.001)
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_data=(X_val_scaled,
y_val))

# Predict AQI values on the test set
y_pred_test = model.predict(X_test_scaled)

# Predict AQI values on the validation set
y_pred_val = model.predict(X_val_scaled)

# Evaluate the model
mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
r2_test = r2_score(y_test, y_pred_test)

# Evaluate the model on the validation set
mse_val = mean_squared_error(y_val, y_pred_val)
mae_val = mean_absolute_error(y_val, y_pred_val)
rmse_val = np.sqrt(mean_squared_error(y_val, y_pred_val))
r2_val = r2_score(y_val, y_pred_val)



# Print evaluation metrics
print("Test Set:")
print("Mean Squared Error (MSE):", mse_test)
print("Mean Absolute Error (MAE):", mae_test)
print("Root Mean Squared Error (RMSE) on the test set:", rmse_test)
print("R-squared (R2) Score:", r2_test)

# Print evaluation metrics for validation set
print("Validation Set:")
print("Mean Squared Error (MSE):", mse_val)
print("Mean Absolute Error (MAE):", mae_val)
print("Root Mean Squared Error (RMSE) on the validation set:", rmse_val)
print("R-squared (R2) Score:", r2_val)

# import numpy as np
# import tensorflow as tf
```

```
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Dense
# from tensorflow.keras.optimizers import Adam
# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import StandardScaler
# from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# # Assuming X and y are your feature matrix and target variable, respectively

# # Split the dataset into training, test, and validation sets
# X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
# X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# # Standardize the features (optional but recommended for neural networks)
# scaler = StandardScaler()
# X_train_scaled = scaler.fit_transform(X_train)
# X_test_scaled = scaler.transform(X_test)
# X_val_scaled = scaler.transform(X_val)

# # Build the neural network model with adjusted hyperparameters
# model = Sequential([
#     Dense(32, activation='relu', input_shape=(X_train_scaled.shape[1],)),
#     Dense(16, activation='relu'),
#     Dense(1)  # Output layer with single neuron for regression task
# ])

# # Compile the model with adjusted learning rate
# model.compile(optimizer='adam', loss='mean_squared_error')

# # Train the model with reduced number of epochs
# history = model.fit(X_train_scaled, y_train, epochs=30, batch_size=16, validation_data=(X_val_scaled,
y_val))

# # Predict AQI values on the test set
# y_pred_test = model.predict(X_test_scaled)

# # Predict AQI values on the validation set
# y_pred_val = model.predict(X_val_scaled)

# # Evaluate the model on the test set
# mse_test = mean_squared_error(y_test, y_pred_test)
# mae_test = mean_absolute_error(y_test, y_pred_test)
# rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
# r2_test = r2_score(y_test, y_pred_test)

# # Evaluate the model on the validation set
```

```
# mse_val = mean_squared_error(y_val, y_pred_val)
# mae_val = mean_absolute_error(y_val, y_pred_val)
# rmse_val = np.sqrt(mean_squared_error(y_val, y_pred_val))
# r2_val = r2_score(y_val, y_pred_val)


# # Print evaluation metrics
# print("Test Set:")
# print("Mean Squared Error (MSE):", mse_test)
# print("Mean Absolute Error (MAE):", mae_test)
# print("Root Mean Squared Error (RMSE) on the test set:", rmse_test)
# print("R-squared (R2) Score:", r2_test)

# # Print evaluation metrics for validation set
# print("Validation Set:")
# print("Mean Squared Error (MSE):", mse_val)
# print("Mean Absolute Error (MAE):", mae_val)
# print("Root Mean Squared Error (RMSE) on the validation set:", rmse_val)
# print("R-squared (R2) Score:", r2_val)

import matplotlib.pyplot as plt

# Scatter plot for Test Set
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', label='Actual vs Predicted (Test Set)')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Ideal Line')
plt.xlabel('Actual AQI')
plt.ylabel('Predicted AQI')
plt.title('Actual vs Predicted AQI (Test Set)')
plt.legend()
plt.grid(True)
plt.show()
```

Linear Regression

```
# -*- coding: utf-8 -*-
"""LR.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1rI3nghEFMkddiNNydMXDK98jyS-J1FTC
"""
```

```
import numpy as np
import pandas as pd

df = pd.read_excel('/content/CHN_AQI.xlsx')

df.tail()

df.info()

df.isnull().sum()

## df = df.drop(df.columns[[0, 4, 6]], axis=1)
df = df.drop(columns=['City'])

df.info()

from sklearn.impute import SimpleImputer, KNNImputer

# Load your dataset (assuming it's already loaded into a DataFrame named 'df')

# Drop the 'Xylene' column as it has no non-null values
df.drop(columns=['Xylene','PM10'], inplace=True)

# Define columns for statistical imputation and KNN imputation
statistical_impute_cols = ['PM2.5', 'NO', 'NO2', 'NOx', 'CO', 'SO2', 'O3']
knn_impute_cols = ['NH3', 'Benzene', 'Toluene']

# Impute missing values using statistical measures (mean, median, or mode)
statistical_imputer = SimpleImputer(strategy='mean')  # You can change the strategy if needed
df[statistical_impute_cols] = statistical_imputer.fit_transform(df[statistical_impute_cols])

# Impute missing values using K-nearest neighbors (KNN) imputation
knn_imputer = KNNImputer(n_neighbors=5)  # You can adjust the number of neighbors as needed
df[knn_impute_cols] = knn_imputer.fit_transform(df[knn_impute_cols])

# Drop rows with missing AQI values
df.dropna(subset=['AQI', 'AQI_Bucket'], inplace=True)

# Now your dataset should have missing values imputed and rows with missing AQI values dropped

df.info()

df.head()

df.tail()

# Maximum value in the AQI column
max_aqi = df['AQI'].max()
```

```
print("Maximum AQI value:", max_aqi)

# Unique values in the AQI_Bucket column
unique_aqi_buckets = df['AQI_Bucket'].unique()
print("Unique AQI buckets:", unique_aqi_buckets)

df.info()

df = df.set_index('Date')
df.head()

# Define mapping dictionary for AQI buckets
aqi_bucket_mapping = {
    'Good': 0,
    'Satisfactory': 1,
    'Moderate': 2,
    'Poor': 3,
    'Very Poor': 4,
    'Severe': 5
}

# Map AQI buckets to numerical values
df['Encoded_AQI_Bucket'] = df['AQI_Bucket'].map(aqi_bucket_mapping)
df.head()

df = df.drop(columns=['AQI_Bucket', 'Encoded_AQI_Bucket'])

df.tail()

df.info()

# Assuming your dataset is named df
X = df.drop(columns=['AQI'])  # Independent variables (features)
y = df['AQI']  # Dependent variable (target)

# Check the shapes of X and y
print("Shape of X:", X)
print("Shape of y:", y)

import seaborn as sns

# Calculate the correlation matrix
corr_matrix = df.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", annot_kws={"size": 10})
plt.title('Correlation Matrix of Features')
```

```
plt.show()

from sklearn.model_selection import train_test_split

# Split the dataset into training (70%) and temporary (30%)
X_train_temp, X_temp, y_train_temp, y_temp = train_test_split(X, y, test_size=0.3, shuffle=False)

# Further split the temporary dataset (30%) into test (50%) and validation (50%)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, shuffle=False)

# Check the shapes of the datasets
print("Training set shape:", X_train_temp.shape, y_train_temp.shape)
print("Test set shape:", X_test.shape, y_test.shape)
print("Validation set shape:", X_val.shape, y_val.shape)

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

# Initialize the linear regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train_temp, y_train_temp)

# Make predictions on the test data
y_pred_test = model.predict(X_test)

# Evaluate the model on the test set
mse_test = mean_squared_error(y_test, y_pred_test)
print("Mean Squared Error on the test set:", mse_test)

# Calculate Mean Absolute Error (MAE) for test set
mae_test = mean_absolute_error(y_test, y_pred_test)
print("Mean Absolute Error (MAE) on the test set:", mae_test)

# Calculate Root Mean Squared Error (RMSE) for test set
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
print("Root Mean Squared Error (RMSE) on the test set:", rmse_test)

# Calculate R-squared for test set
r2_test = r2_score(y_test, y_pred_test)
print("R-squared on the test set:", r2_test)

# Make predictions on the validation data
y_pred_val = model.predict(X_val)

# Evaluate the model on the validation set
mse_val = mean_squared_error(y_val, y_pred_val)
```

```
print("Mean Squared Error on the validation set:", mse_val)

# Calculate Mean Absolute Error (MAE) for validation set
mae_val = mean_absolute_error(y_val, y_pred_val)
print("Mean Absolute Error (MAE) on the validation set:", mae_val)

# Calculate Root Mean Squared Error (RMSE) for validation set
rmse_val = np.sqrt(mean_squared_error(y_val, y_pred_val))
print("Root Mean Squared Error (RMSE) on the validation set:", rmse_val)

# Calculate R-squared for validation set
r2_val = r2_score(y_val, y_pred_val)
print("R-squared on the validation set:", r2_val)

import matplotlib.pyplot as plt

# Plotting the actual vs predicted values for the test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_test, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.title('Actual vs Predicted values for Test Set')
plt.xlabel('Actual AQI')
plt.ylabel('Predicted AQI')
plt.grid(True)
plt.show()

import numpy as np

# Generate indices for each data point
indices = np.arange(len(y_test))

# Plotting the actual and predicted values as a bar chart
plt.figure(figsize=(10, 6))
bar_width = 0.35

plt.bar(indices, y_test, bar_width, label='Actual AQI', color='blue')
plt.bar(indices + bar_width, y_pred_test, bar_width, label='Predicted AQI', color='orange')

plt.xlabel('Data Point Index')
plt.ylabel('AQI')
plt.title('Actual vs Predicted values for Test Set')
plt.xticks(indices + bar_width / 2, indices)
plt.legend()
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
```

```
# Plotting the actual and predicted values along with dates
plt.figure(figsize=(12, 6))
plt.plot(X_test.index, y_test, label='Actual AQI', marker='o', color='blue')
plt.plot(X_test.index, y_pred_test, label='Predicted AQI', marker='o', color='orange')

plt.title('Actual vs Predicted AQI')
plt.xlabel('Date')
plt.ylabel('AQI')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Resample data to a lower frequency (e.g., monthly)
X_test_resampled = X_test.resample('M').mean()
y_test_resampled = y_test.resample('M').mean()
y_pred_test_resampled = pd.Series(y_pred_test, index=X_test.index).resample('M').mean()

# Plotting the actual and predicted values along with resampled dates
plt.figure(figsize=(12, 6))
plt.plot(X_test_resampled.index, y_test_resampled, label='Actual AQI', marker='o', color='blue')
plt.plot(X_test_resampled.index, y_pred_test_resampled, label='Predicted AQI', marker='o',
color='orange')

plt.title('Actual vs Predicted AQI (Monthly)')
plt.xlabel('Date')
plt.ylabel('AQI')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt

# Test set scores
test_mse = [474.52, 898, 599.51, 1590.9]
test_r_squared = [0.65, 0.72, 0.56, 0.51]

# Validation set scores
val_mse = [511.88, 682.21, 404.88, 869.67]
val_r_squared = [0.65, 0.67, 0.72, 0.58]

# Model names
models = ['Linear Regression', 'Random Forest Regression', 'XG Boost Regression', 'ANN']

# Plotting fitting ability (Validation set)
```

```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.barh(models, val_r_squared, color='skyblue')
plt.xlabel('R-squared Score')
plt.title('Fitting Ability (Validation Set)')
plt.xlim(0, 1)

# Plotting generalization ability (Test set)
plt.subplot(1, 2, 2)
plt.barh(models, test_r_squared, color='lightgreen')
plt.xlabel('R-squared Score')
plt.title('Generalization Ability (Test Set)')
plt.xlim(0, 1)

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

Random Forest Regression

```
# -*- coding: utf-8 -*-
"""RFR.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1-2k7wne1fkBexaC6VYclpPFcLw13dpgd
"""

import numpy as np
import pandas as pd

df = pd.read_excel('/content/CHN_AQI.xlsx')

df.tail()

df.info()

df.isnull().sum()

## df = df.drop(df.columns[[0, 4, 6]], axis=1)
df = df.drop(columns=['City'])

df.info()

from sklearn.impute import SimpleImputer, KNNImputer

# Load your dataset (assuming it's already loaded into a DataFrame named 'df')
```

```
# Drop the 'Xylene' column as it has no non-null values
df.drop(columns=['Xylene','PM10'], inplace=True)

# Define columns for statistical imputation and KNN imputation
statistical_impute_cols = ['PM2.5', 'NO', 'NO2', 'NOx', 'CO', 'SO2', 'O3']
knn_impute_cols = ['NH3', 'Benzene', 'Toluene']

# Impute missing values using statistical measures (mean, median, or mode)
statistical_imputer = SimpleImputer(strategy='mean')  # You can change the strategy if needed
df[statistical_impute_cols] = statistical_imputer.fit_transform(df[statistical_impute_cols])

# Impute missing values using K-nearest neighbors (KNN) imputation
knn_imputer = KNNImputer(n_neighbors=5)  # You can adjust the number of neighbors as needed
df[knn_impute_cols] = knn_imputer.fit_transform(df[knn_impute_cols])

# Drop rows with missing AQI values
df.dropna(subset=['AQI', 'AQI_Bucket'], inplace=True)

# Now your dataset should have missing values imputed and rows with missing AQI values dropped

df.info()

df.head()

df.tail()

# Maximum value in the AQI column
max_aqi = df['AQI'].max()
print("Maximum AQI value:", max_aqi)

# Unique values in the AQI_Bucket column
unique_aqi_buckets = df['AQI_Bucket'].unique()
print("Unique AQI buckets:", unique_aqi_buckets)

df.info()

df = df.set_index('Date')
df.head()

# Define mapping dictionary for AQI buckets
aqi_bucket_mapping = {
    'Good': 0,
    'Satisfactory': 1,
    'Moderate': 2,
    'Poor': 3,
    'Very Poor': 4,
    'Severe': 5
```

```
}

# Map AQI buckets to numerical values
df['Encoded_AQI_Bucket'] = df['AQI_Bucket'].map(aqi_bucket_mapping)
df.head()

df = df.drop(columns=['AQI_Bucket', 'Encoded_AQI_Bucket'])

df.tail()

df.info()

# Assuming your dataset is named df
X = df.drop(columns=['AQI'])  # Independent variables (features)
y = df['AQI']  # Dependent variable (target)

# Check the shapes of X and y
print("Shape of X:", X)
print("Shape of y:", y)

import seaborn as sns
import matplotlib.pyplot as plt
# Calculate the correlation matrix
corr_matrix = df.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", annot_kws={"size": 10})
plt.title('Correlation Matrix of Features')
plt.show()

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split

# Assuming X and y are your features and target variable
# Split the data into training, test, and validation sets (70%, 15%, 15%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Initialize the Random Forest Regression model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

#rf_model = RandomForestRegressor(n_estimators= 250,  # Increase number of trees
                                 # max_depth=15,      # Limit tree depth to prevent overfitting
                                 # min_samples_split=5,  # Increase minimum samples per split
                                 # min_samples_leaf=2,   # Increase minimum samples per leaf
                                 # max_features='sqrt',  # Use square root of features
```

```
                    # bootstrap=True,       # Enable bootstrapping
                    # random_state=42)      # For reproducibility


# Train the model
rf_model.fit(X_train, y_train)

# Make predictions on the test and validation sets
y_pred_test = rf_model.predict(X_test)
y_pred_val = rf_model.predict(X_val)

# Evaluate model performance
mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
r2_test = r2_score(y_test, y_pred_test)

mse_val = mean_squared_error(y_val, y_pred_val)
mae_val = mean_absolute_error(y_val, y_pred_val)
rmse_val = np.sqrt(mean_squared_error(y_val, y_pred_val))
r2_val = r2_score(y_val, y_pred_val)

# Print evaluation metrics
print("Test Set:")
print("Mean Squared Error (MSE):", mse_test)
print("Mean Absolute Error (MAE):", mae_test)
print("Root Mean Squared Error (RMSE) on the test set:", rmse_test)
print("R-squared (R2) Score:", r2_test)

print("\nValidation Set:")
print("Mean Squared Error (MSE):", mse_val)
print("Mean Absolute Error (MAE):", mae_val)
print("Root Mean Squared Error (RMSE) on the validation set:", rmse_val)
print("R-squared (R2) Score:", r2_val)

import matplotlib.pyplot as plt

# Scatter plot for Test Set
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', label='Actual vs Predicted (Test Set)')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Ideal Line')
plt.xlabel('Actual AQI')
plt.ylabel('Predicted AQI')
plt.title('Actual vs Predicted AQI (Test Set)')
plt.legend()
plt.grid(True)
plt.show()
```

```
# Scatter plot for Validation Set
plt.figure(figsize=(10, 6))
plt.scatter(y_val, y_pred_val, color='green', label='Actual vs Predicted (Validation Set)')
plt.plot([min(y_val), max(y_val)], [min(y_val), max(y_val)], color='red', linestyle='--', label='Ideal Line')
plt.xlabel('Actual AQI')
plt.ylabel('Predicted AQI')
plt.title('Actual vs Predicted AQI (Validation Set)')
plt.legend()
plt.grid(True)
plt.show()

# Resample data to a lower frequency (e.g., monthly)
X_test_resampled = X_test.resample('M').mean()
y_test_resampled = y_test.resample('M').mean()
y_pred_test_resampled = pd.Series(y_pred_test, index=X_test.index).resample('M').mean()

# Plotting the actual and predicted values along with resampled dates
plt.figure(figsize=(12, 6))
plt.plot(X_test_resampled.index, y_test_resampled, label='Actual AQI', marker='o', color='blue')
plt.plot(X_test_resampled.index, y_pred_test_resampled, label='Predicted AQI', marker='o',
color='orange')

plt.title('Actual vs Predicted AQI (Monthly)')
plt.xlabel('Date')
plt.ylabel('AQI')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

XG BOOST

```
# -*- coding: utf-8 -*-
"""XBR.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1SuT6Y0LpC6b8EV1COI8IgStQyYHUMiAM
"""

import numpy as np
import pandas as pd

df = pd.read_excel('/content/CHN_AQI.xlsx')
```

```
df.tail()

df.info()

df.isnull().sum()

## df = df.drop(df.columns[[0, 4, 6]], axis=1)
df = df.drop(columns=['City'])

df.info()

from sklearn.impute import SimpleImputer, KNNImputer

# Load your dataset (assuming it's already loaded into a DataFrame named 'df')

# Drop the 'Xylene' column as it has no non-null values
df.drop(columns=['Xylene','PM10'], inplace=True)

# Define columns for statistical imputation and KNN imputation
statistical_impute_cols = ['PM2.5', 'NO', 'NO2', 'NOx', 'CO', 'SO2', 'O3']
knn_impute_cols = ['NH3', 'Benzene', 'Toluene']

# Impute missing values using statistical measures (mean, median, or mode)
statistical_imputer = SimpleImputer(strategy='mean')  # You can change the strategy if needed
df[statistical_impute_cols] = statistical_imputer.fit_transform(df[statistical_impute_cols])

# Impute missing values using K-nearest neighbors (KNN) imputation
knn_imputer = KNNImputer(n_neighbors=5)  # You can adjust the number of neighbors as needed
df[knn_impute_cols] = knn_imputer.fit_transform(df[knn_impute_cols])

# Drop rows with missing AQI values
df.dropna(subset=['AQI', 'AQI_Bucket'], inplace=True)

# Now your dataset should have missing values imputed and rows with missing AQI values dropped

df.info()

df.head()

df.tail()

# Maximum value in the AQI column
max_aqi = df['AQI'].max()
print("Maximum AQI value:", max_aqi)

# Unique values in the AQI_Bucket column
unique_aqi_buckets = df['AQI_Bucket'].unique()
print("Unique AQI buckets:", unique_aqi_buckets)
```

```
df.info()

df = df.set_index('Date')
df.head()

# Define mapping dictionary for AQI buckets
aqi_bucket_mapping = {
    'Good': 0,
    'Satisfactory': 1,
    'Moderate': 2,
    'Poor': 3,
    'Very Poor': 4,
    'Severe': 5
}

# Map AQI buckets to numerical values
df['Encoded_AQI_Bucket'] = df['AQI_Bucket'].map(aqi_bucket_mapping)
df.head()

df = df.drop(columns=['AQI_Bucket', 'Encoded_AQI_Bucket'])

df.tail()

df.info()

# Assuming your dataset is named df
X = df.drop(columns=['AQI'])  # Independent variables (features)
y = df['AQI']  # Dependent variable (target)

# Check the shapes of X and y
print("Shape of X:", X)
print("Shape of y:", y)

import seaborn as sns
import matplotlib.pyplot as plt
# Calculate the correlation matrix
corr_matrix = df.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", annot_kws={"size": 10})
plt.title('Correlation Matrix of Features')
plt.show()

import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
```

```
# Split the dataset into training (70%) and temporary (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, shuffle=False)

# Further split the temporary dataset (30%) into test (50%) and validation (50%)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, shuffle=False)


# Initialize the XGBRegressor model with hyperparameters
# xgb_model = xgb.XGBRegressor(n_estimators=100,  # Number of trees (default is 100)
#                              max_depth=6,       # Maximum depth of each tree (default is 3)
#                              learning_rate=0.3, # Learning rate (default is 0.1)
#                              colsample_bytree=1.0,  # Percentage of features used per tree (default is 1.0)
#                              subsample=1.0,     # Percentage of samples used per tree (default is 1.0)
#                              random_state=None)   # Random state for reproducibility

xgb_model = xgb.XGBRegressor(n_estimators=100,  # Number of trees (default is 100)
                             max_depth=3,       # Maximum depth of each tree (default is 3)
                             learning_rate=0.1, # Learning rate (default is 0.1)
                             colsample_bytree=0.8,  # Percentage of features used per tree (default is 1.0)
                             subsample=0.8,     # Percentage of samples used per tree (default is 1.0)
                             random_state=42)   # Random state for reproducibility


# Train the model on the training data
xgb_model.fit(X_train, y_train)

# Predict AQI values on the test set
y_pred_test = xgb_model.predict(X_test)

# Predict AQI values on the validation set
y_pred_val = xgb_model.predict(X_val)

# Evaluate the model
mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
r2_test = r2_score(y_test, y_pred_test)

mse_val = mean_squared_error(y_val, y_pred_val)
mae_val = mean_absolute_error(y_val, y_pred_val)
rmse_val = np.sqrt(mean_squared_error(y_val, y_pred_val))
r2_val = r2_score(y_val, y_pred_val)

# Print evaluation metrics
print("Test Set:")
print("Mean Squared Error (MSE):", mse_test)
print("Mean Absolute Error (MAE):", mae_test)
```

```
print("Root Mean Squared Error (RMSE) on the test set:", rmse_test)
print("R-squared (R2) Score:", r2_test)
print("\nValidation Set:")
print("Mean Squared Error (MSE):", mse_val)
print("Mean Absolute Error (MAE):", mae_val)
print("Root Mean Squared Error (RMSE) on the validation set:", rmse_val)
print("R-squared (R2) Score:", r2_val)

import matplotlib.pyplot as plt

# Scatter plot for Test Set
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', label='Actual vs Predicted (Test Set)')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Ideal Line')
plt.xlabel('Actual AQI')
plt.ylabel('Predicted AQI')
plt.title('Actual vs Predicted AQI (Test Set)')
plt.legend()
plt.grid(True)
plt.show()

import matplotlib.pyplot as plt

# Plotting the actual and predicted values along with dates
plt.figure(figsize=(12, 6))
plt.plot(X_test.index, y_test, label='Actual AQI', marker='o', color='blue')
plt.plot(X_test.index, y_pred_test, label='Predicted AQI', marker='o', color='orange')

plt.title('Actual vs Predicted AQI')
plt.xlabel('Date')
plt.ylabel('AQI')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Resample data to a lower frequency (e.g., monthly)
X_test_resampled = X_test.resample('M').mean()
y_test_resampled = y_test.resample('M').mean()
y_pred_test_resampled = pd.Series(y_pred_test, index=X_test.index).resample('M').mean()

# Plotting the actual and predicted values along with resampled dates
plt.figure(figsize=(12, 6))
plt.plot(X_test_resampled.index, y_test_resampled, label='Actual AQI', marker='o', color='blue')
plt.plot(X_test_resampled.index, y_pred_test_resampled, label='Predicted AQI', marker='o',
color='orange')
```

```
plt.title('Actual vs Predicted AQI (Monthly)')
plt.xlabel('Date')
plt.ylabel('AQI')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

REFERENCES

1.Zhang, K., & Batterman, S. (2019). Air pollution and health risks due to vehicle traffic. Science of the Total Environment, 627, 515-527.

2.Tripathi, P., De A., & Basu, K. (2020). Air quality index prediction: a comparative study using machine learning techniques. Journal of Cleaner Production, 266, 121985.

3.Wei, Y., Chen, C., & Fu, K. (2021). Machine learning-based air quality prediction model considering meteorological factors in Xi'an, China. Environmental Monitoring and Assessment, 193(7), 463.

4.Yadav, S., Singh, N., & Kumar, R. (2020). A machine learning approach for air quality prediction using meteorological factors. Environmental Modeling & Assessment, 25(4), 377-392.

5.Cervone, G., Franch, G., & Ault, T. (2017). Use of machine learning algorithms for PM2.5 forecasting. Environmental Forensics, 18(1), 22-29.

6.Hagen-Zanker, A., Nicholls, R. J., & Richards, J. A. (2021). Data fusion of satellite-based estimates and machine learning for high-resolution air quality mapping. Remote Sensing of Environment, 253, 112209.