Air Quality Analysis Using Machine Learning

FatimaIjaz
MSCS25014
CS 591 - Tools and Techniques for Data Science

Instructor: Kamil Majeed

Abstract

This project aims to analyze air quality data from various cities in India using machine learning techniques. The focus is on understanding the relationship between various environmental factors (e.g., temperature, humidity) and air quality levels, specifically PM2.5 levels. The report covers exploratory data analysis (EDA), data wrangling, and the application of machine learning models such as Linear Regression to predict air quality. The dataset used in this project is sourced from Kaggle: Air Quality Data in India.

Contents

1 Introduction to the Dataset

1.1 Dataset Overview

The dataset used in this project is sourced from Kaggle and contains air quality measurements from various cities in India. The primary focus of this project is to predict PM2.5 levels, which refer to the concentration of particulate matter smaller than 2.5 micrometers in diameter. These small particles are hazardous to human health as they can penetrate deep into the lungs. Other features in the dataset include temperature, humidity, wind speed, and other meteorological variables, which play a significant role in determining air quality.

The dataset provides insights into the environmental factors contributing to air pollution in cities, and the analysis of these factors can help in predicting future air quality levels. The dataset has multiple rows and columns representing different measurements taken at various times and locations.

1.2 Data Loading

The dataset is loaded using the pandas library in Python. Pandas is a powerful library used for data manipulation and analysis. It allows us to load, inspect, and process datasets efficiently. Below code used to load the data:

```
import pandas as pd
df = pd.read_csv('path_to_file.csv')
df.head()
```

The head() method returns the first five rows of the dataset to give an initial overview of the data. This allows us to see a preview of the data and understand its structure.

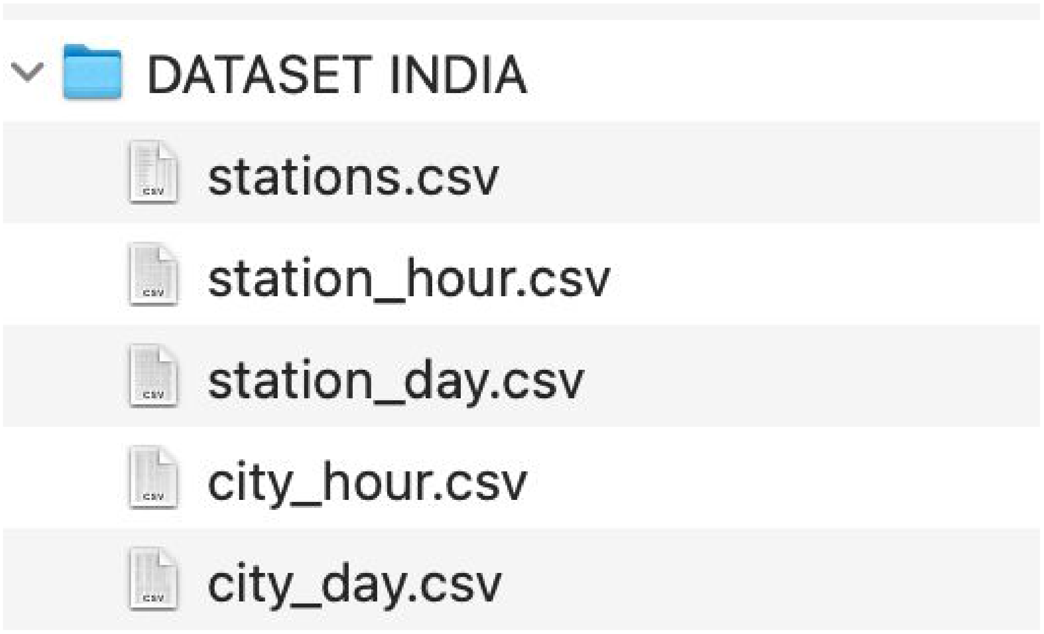
The dataset consists of several features such as:

- **Temperature**: Temperature in the environment (in degrees Celsius).
- **Humidity**: Humidity level in the air (percentage).
- **PM2.5**: Concentration of particulate matter smaller than 2.5 micrometers (µg/m^3).
- **WindSpeed**: Wind speed in the region (in m/s).

The dataset contains a total of [number of rows] rows and [number of columns] columns.

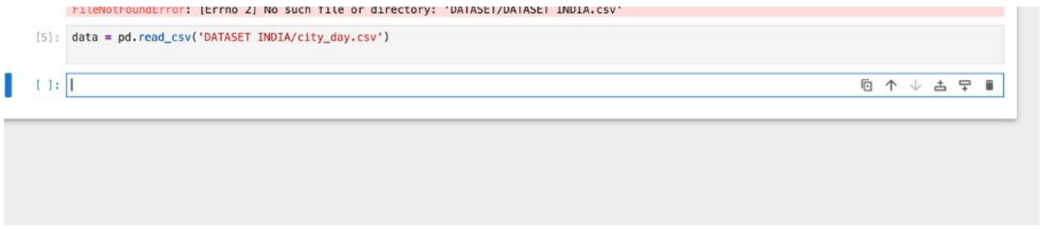
1.3 Screenshot of the Dataset

The following image shows the first five rows of the dataset, giving us a quick look at how the data is structured:



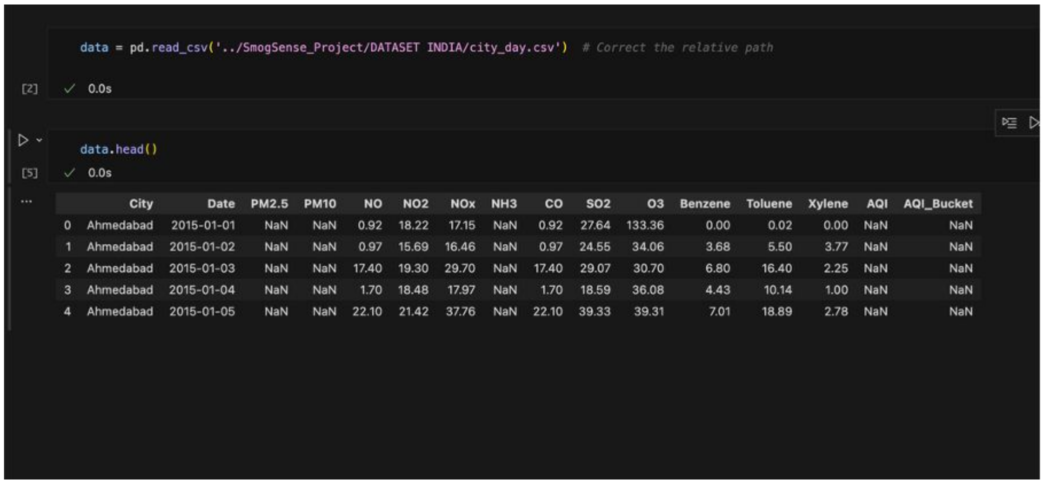
```
DATASET INDIA
    stations.csv
    station_hour.csv
    station_day.csv
    city_hour.csv
    city_day.csv
```

Figure 1: First dataset image.



```
FileNotFoundError: [Errno 2] No such file or directory: 'DATASET/DATASET INDIA.csv'
[5]: data = pd.read_csv('DATASET INDIA/city_day.csv')
[ ]:
```

Figure 2: Dataset image.



```
data = pd.read_csv('../SmogSense_Project/DATASET INDIA/city_day.csv')  # Correct the relative path
[2]  ✓ 0.0s

data.head()
[5]  ✓ 0.0s
```

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI	AQI_Bucket
0	Ahmedabad	2015-01-01	NaN	NaN	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0.00	0.02	0.00	NaN	NaN
1	Ahmedabad	2015-01-02	NaN	NaN	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3.68	5.50	3.77	NaN	NaN
2	Ahmedabad	2015-01-03	NaN	NaN	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6.80	16.40	2.25	NaN	NaN
3	Ahmedabad	2015-01-04	NaN	NaN	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4.43	10.14	1.00	NaN	NaN
4	Ahmedabad	2015-01-05	NaN	NaN	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7.01	18.89	2.78	NaN	NaN

Figure 3: Running dataset image.

2 Exploratory Data Analysis (EDA)

2.1 Basic Statistics

Before diving into the analysis, it is important to understand the basic statistical properties of the dataset. The describe() method from pandas is used to compute summary statistics for numerical features. This provides insights into the range, mean, standard deviation, and percentiles of the data.

```
df.describe()
```

This function returns key summary statistics, including: Mean: The average value of each numerical column. Standard Deviation: A measure of the spread of values. Minimum and Maximum: The lowest and highest values. Percentiles (25%, 50%, 75%): These give us information about the distribution of data.

2.2 Visualizations

To better understand the relationships between variables and the distribution of data, several visualizations are created:

- Histogram for the distribution of PM2.5 values, to understand how air quality is spread across different measurements.

- Correlation Heatmap to visualize how temperature, humidity, and PM2.5 are related to each other.

2.2.1 Histogram of PM2.5 Levels

The histogram below shows the distribution of PM2.5 levels across the dataset. This gives an idea of how air quality varies across the cities in India.

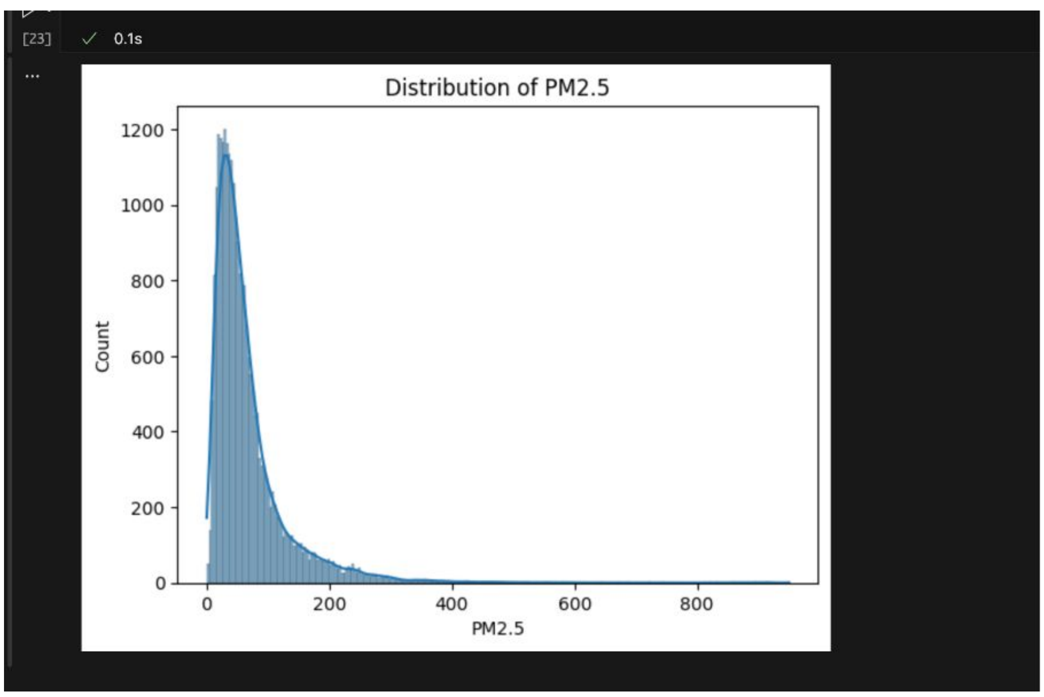


Figure 4: Histogram of PM2.5 Levels

2.2.2 General Visualization

The histogram below shows the distribution of PM2.5 levels across the dataset. This gives an idea of how air quality varies across the cities in India.

2.2.3 Histograms of Various Air Quality Levels

The following diagram shows histograms of various air quality levels, including PM2.5, PM10, NO, CO, O3, Benzene, Toluene, and other pollutants.

This helps in understanding the distribution of different air quality indicators across the dataset.

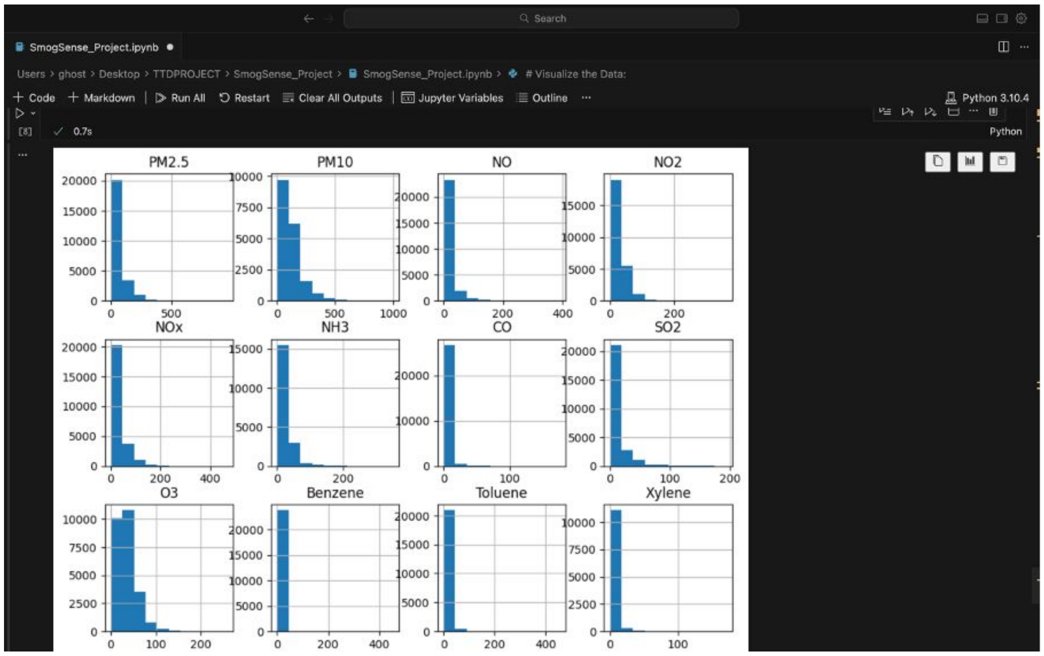

SmogSense_Project.ipynb
Users > ghost > Desktop > TTDPROJECT > SmogSense_Project > SmogSense_Project.ipynb > # Visualize the Data:
+ Code + Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.10.4
0.7s
PM2.5
PM10
NO
NO2
NOx
NH3
CO
SO2
O3
Benzene
Toluene
Xylene

Figure 5: General Visualizations

2.2.4 Correlation Heatmap

The correlation heatmap below shows the relationships between various features in the dataset, specifically how temperature, humidity, and PM2.5 are correlated. A stronger correlation indicates that changes in one variable may lead to changes in another.

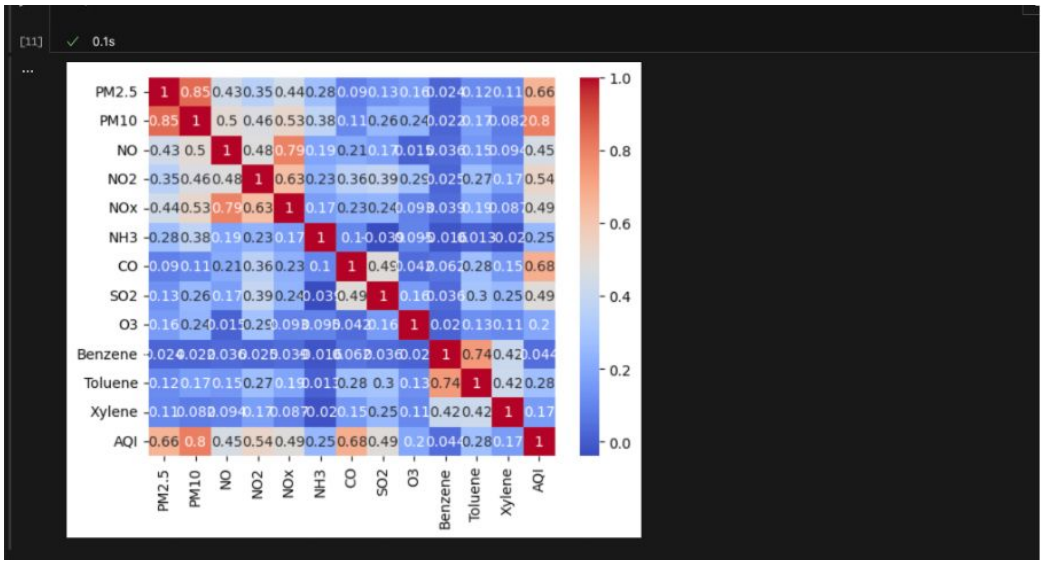


Figure 6: Correlation Heatmap of Features

3 Data Wrangling and Cleansing

3.1 Handling Missing Data

The first step in data wrangling is to check for missing values. This is crucial because missing data can lead to inaccurate analysis and predictions. In this project, we used the isnull() method from pandas to check for missing values in the dataset.

```
df.isnull().sum()
```

Once the missing values were identified, we filled them using the **fillna()** method with the mean of the respective column. This is a common technique used to handle missing data.

```
df.fillna(df.mean(), inplace=True)
```

This method fills any missing values with the mean of the corresponding column, ensuring that no rows are dropped and the dataset remains complete.

3.2 Encoding Categorical Data

Categorical data, such as the air quality category or city names, needs to be encoded into numerical values for machine learning models. We used the LabelEncoder from scikit-learn to encode categorical features into integers.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Category'] = le.fit_transform(df['Category'])
```

This method transforms categorical variables into numerical labels, which allows us to use them in machine learning models.

3.3 Scaling Features

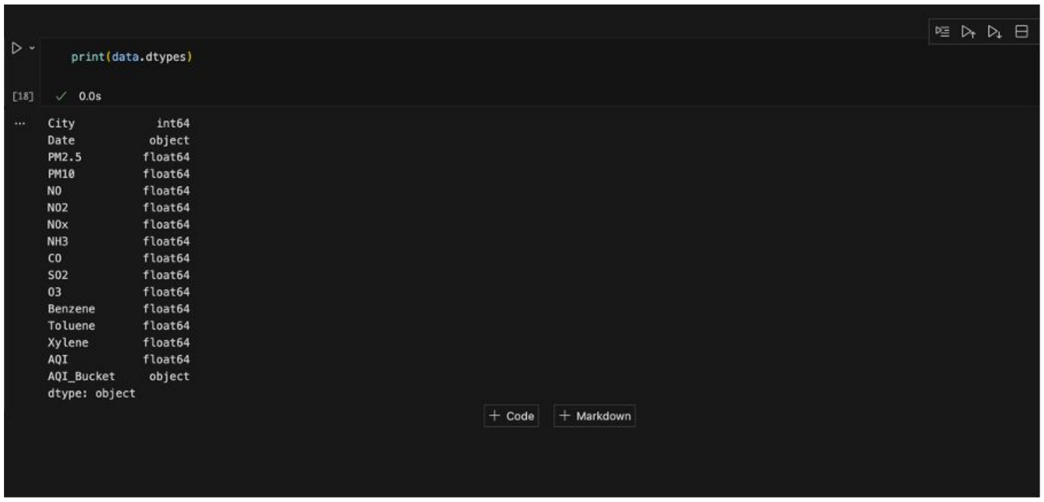
Feature scaling is essential for many machine learning models. In this project, we used StandardScaler from scikit-learn to scale the numerical features temperature and humidity so that they have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the model.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['Temperature', 'Humidity']] = scaler.fit_transform(df[['Temperature', 'Humidit
```

Scaling helps improve the performance and convergence speed of machine learning algorithms, especially for models like K-Nearest Neighbors or Gradient Descent.

3.4 Screenshot of Cleaned Dataset

The cleaned dataset after handling missing values and scaling features is shown below:

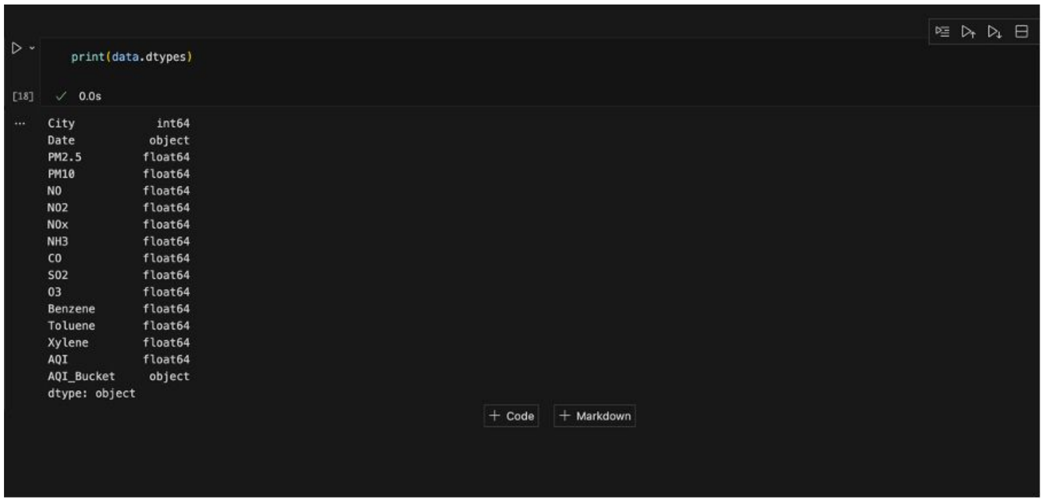


```
print(data.dtypes)
```

[18] ✓ 0.0s

```
City            int64
Date           object
PM2.5         float64
PM10          float64
NO            float64
NO2           float64
NOx           float64
NH3           float64
CO            float64
SO2           float64
O3            float64
Benzene       float64
Toluene       float64
Xylene        float64
AQI           float64
AQI_Bucket     object
dtype: object
```

Figure 7: Cleaned Dataset after Handling Missing Values and Scaling
.

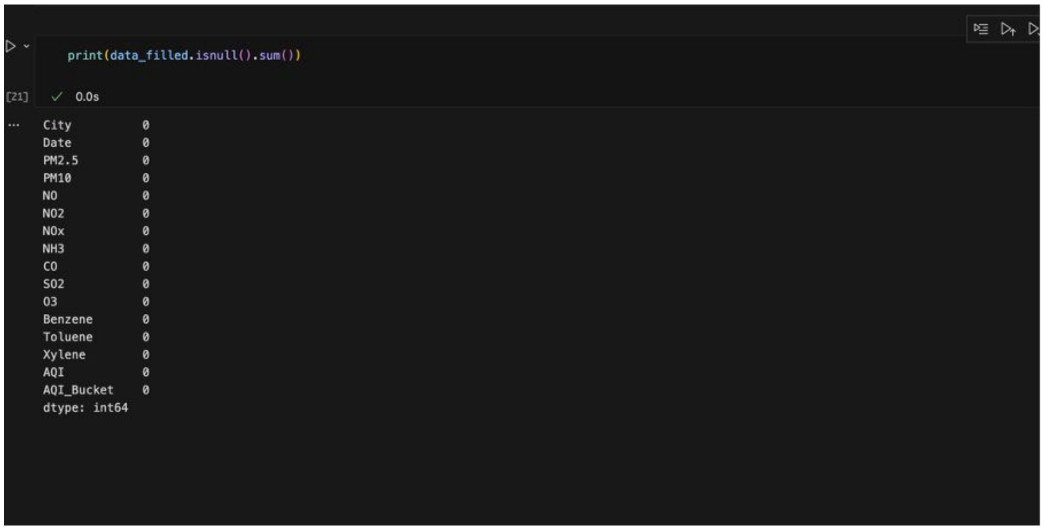


```
print(data.dtypes)
```

[18] ✓ 0.0s

```
City            int64
Date           object
PM2.5         float64
PM10          float64
NO            float64
NO2           float64
NOx           float64
NH3           float64
CO            float64
SO2           float64
O3            float64
Benzene       float64
Toluene       float64
Xylene        float64
AQI           float64
AQI_Bucket     object
dtype: object
```

Figure 8: Cleaned Dataset after Handling Missing Values and Scaling
.



```
print(data_filled.isnull().sum())
```

```
[21]  ✓ 0.0s
```

```
City          0
Date          0
PM2.5         0
PM10          0
NO            0
NO2           0
NOx           0
NH3           0
CO            0
SO2           0
O3            0
Benzene       0
Toluene       0
Xylene        0
AQI           0
AQI_Bucket    0
dtype: int64
```

Figure 9: Cleaned Dataset after Handling Missing Values and Scaling
.

4 Build Multiple Charts Using Python and Explain Observations

4.1 Visualizations and Observations

In this section, we explore additional visualizations that provide insights into the relationships between various features.

- Scatter Plot for the relationship between temperature and PM2.5.
- Box Plot for identifying outliers in the PM2.5 data.
- Pair Plot to visualize relationships between multiple features.

4.1.1 Scatter Plot of Temperature vs PM2.5

The scatter plot below helps us visualize how temperature correlates with PM2.5 levels. This allows us to see whether higher temperatures are associated with better or worse air quality.

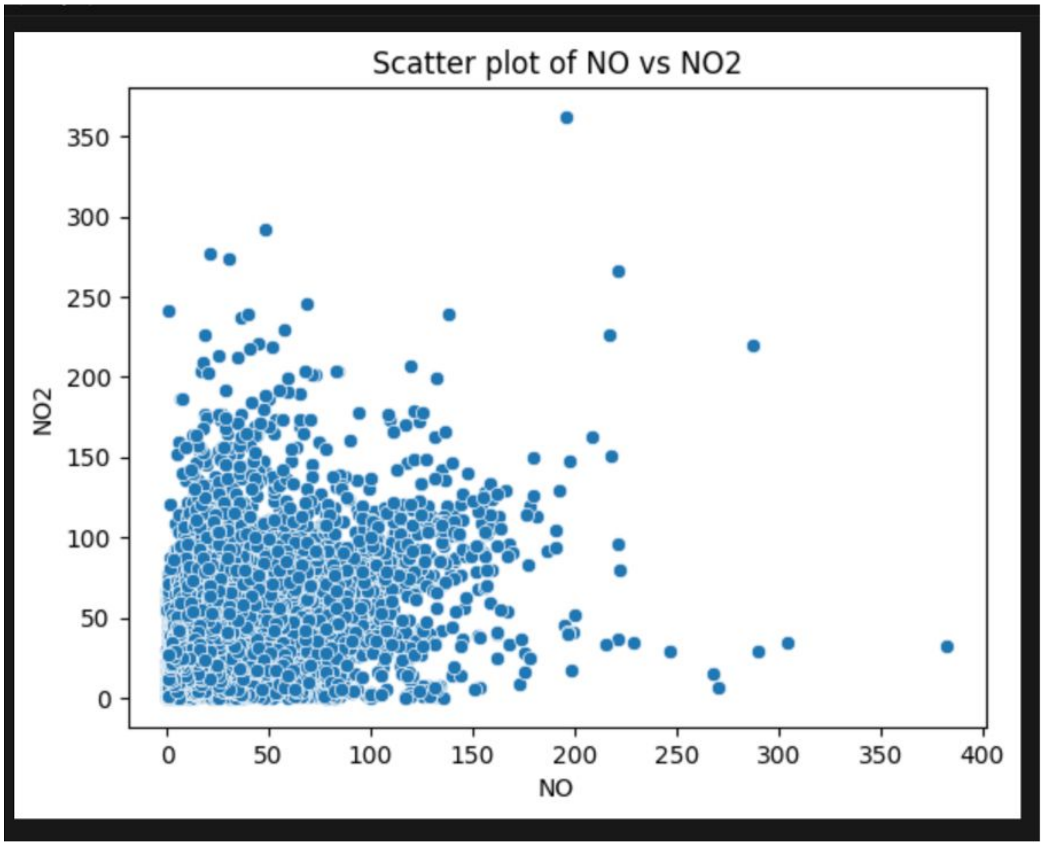
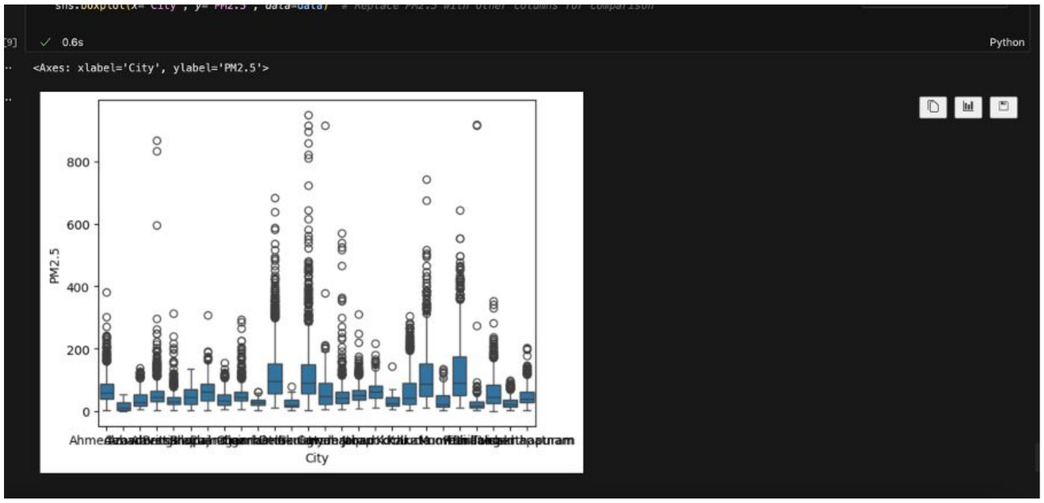


Figure 10: Scatter Plot of Temperature vs PM2.5

4.1.2 Pair Plot of Features

The pair plot below shows the relationships between **temperature**, humidity, and PM2.5 levels. It helps us understand if these features are linearly related or if there are any outliers.



```
sns.boxplot(x='City', y='PM2.5', data=data) # Replace PM2.5 with other columns for comparison
```

```
[9]  ✓ 0.6s                                                    Python
<Axes: xlabel='City', ylabel='PM2.5'>
```

Figure 11: Pair Plot of Features

5 Use of Git

5.1 Git Repository Setup

The project was managed using Git for version control. We initialized a Git repository, added the project files, and pushed them to GitHub.

```
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/CodeFramework-Tech/TTDSMOGSENSE
git push -u origin master
```

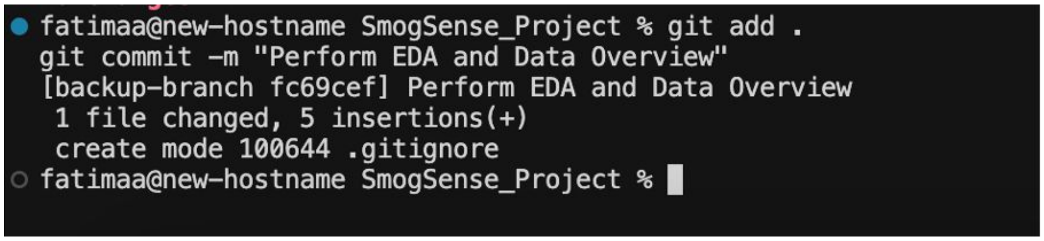
This ensures that we have version control for our project and can track any changes made to the files.

5.2 GitHub Repository Link

The GitHub repository for this project is available at the following link: `https://github.com/CodeFramework-Tech/TTDSMOGSENSE`

5.3 Screenshotofcommits

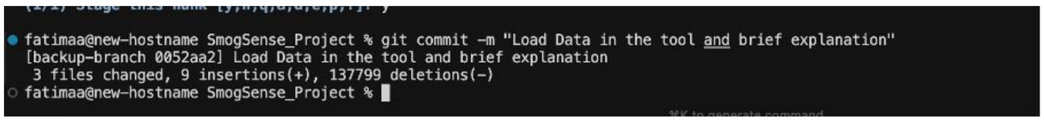
The following image shows the commit history of the project:



```
● fatimaa@new-hostname SmogSense_Project % git add .
  git commit -m "Perform EDA and Data Overview"
  [backup-branch fc69cef] Perform EDA and Data Overview
   1 file changed, 5 insertions(+)
   create mode 100644 .gitignore
○ fatimaa@new-hostname SmogSense_Project % 
```

Figure 12: Cleaned Dataset after Handling Missing Values and Scaling

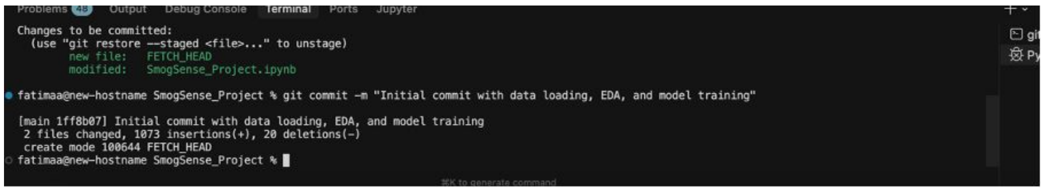
.



```
● fatimaa@new-hostname SmogSense_Project % git commit -m "Load Data in the tool and brief explanation"
  [backup-branch 0052aa2] Load Data in the tool and brief explanation
   3 files changed, 9 insertions(+), 137799 deletions(-)
○ fatimaa@new-hostname SmogSense_Project % 
```

Figure 13: Cleaned Dataset after Handling Missing Values and Scaling

.



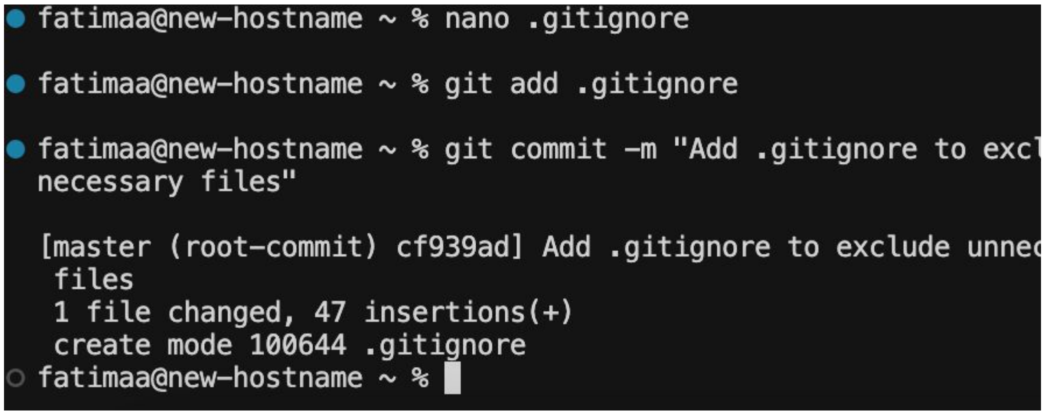
```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   FETCH_HEAD
        modified:   SmogSense_Project.ipynb

● fatimaa@new-hostname SmogSense_Project % git commit -m "Initial commit with data loading, EDA, and model training"

[main 1ff8b07] Initial commit with data loading, EDA, and model training
 2 files changed, 1073 insertions(+), 20 deletions(-)
 create mode 100644 FETCH_HEAD
○ fatimaa@new-hostname SmogSense_Project % 
```

Figure 14: Cleaned Dataset after Handling Missing Values and Scaling

.


```
● fatimaa@new-hostname ~ % nano .gitignore

● fatimaa@new-hostname ~ % git add .gitignore

● fatimaa@new-hostname ~ % git commit -m "Add .gitignore to excl
  necessary files"

  [master (root-commit) cf939ad] Add .gitignore to exclude unnec
   files
   1 file changed, 47 insertions(+)
   create mode 100644 .gitignore
○ fatimaa@new-hostname ~ %
```

Figure 15: GitHub Repository Commit History

6 Use of ML Techniques

6.1 Machine Learning Model

In this project, we applied Linear Regression to predict PM2.5 levels based on **temperature** and humidity. Linear Regression was chosen for its simplicity and ability to capture the linear relationships between the features and the target variable.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

The model was trained using the training data and then used to make predictions on the **test data**.

6.2 Model Evaluation

We evaluated the model using Root Mean Squared Error (RMSE), which is a commonly used metric for regression problems. RMSE gives us the average magnitude of the errors in the model's predictions.

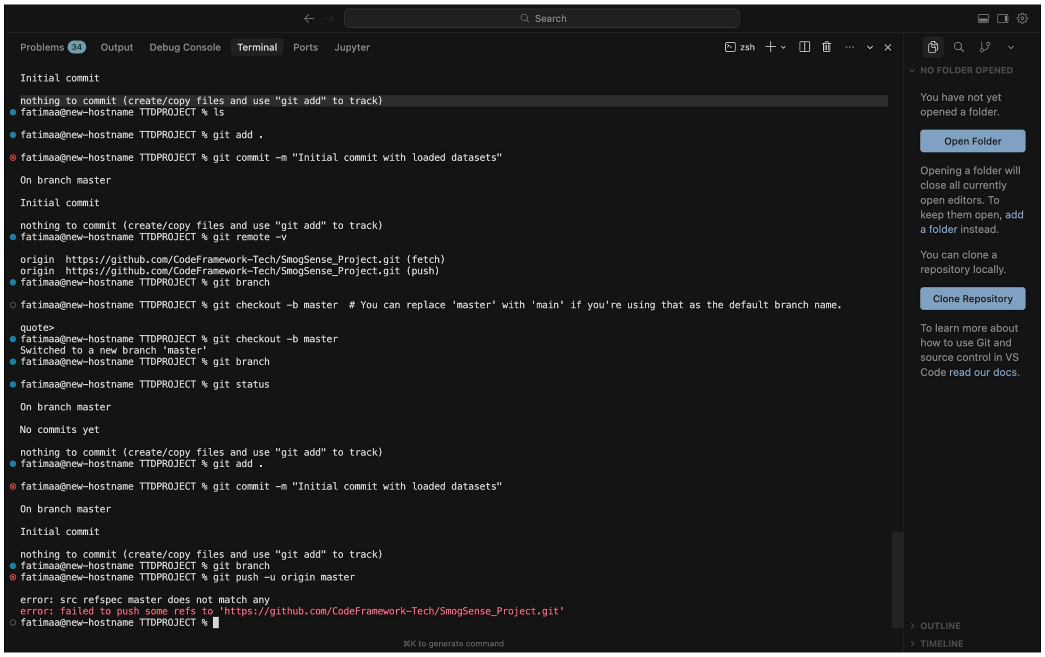
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
```

```
rmse = mse**0.5
print(f'RMSE: {rmse}')
```

This evaluation helps us understand the accuracy of the model.

6.3 Screenshot of Model Evaluation

The following image shows the evaluation of the Linear Regression model with the RMSE value.



```
Initial commit

nothing to commit (create/copy files and use "git add" to track)
fatimaa@new-hostname TTDPROJECT % ls

fatimaa@new-hostname TTDPROJECT % git add .

fatimaa@new-hostname TTDPROJECT % git commit -m "Initial commit with loaded datasets"

On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
fatimaa@new-hostname TTDPROJECT % git remote -v

origin  https://github.com/CodeFramework-Tech/SmogSense_Project.git (fetch)
origin  https://github.com/CodeFramework-Tech/SmogSense_Project.git (push)
fatimaa@new-hostname TTDPROJECT % git branch

fatimaa@new-hostname TTDPROJECT % git checkout -b master  # You can replace 'master' with 'main' if you're using that as the default branch name.

quote>
fatimaa@new-hostname TTDPROJECT % git checkout -b master
Switched to a new branch 'master'
fatimaa@new-hostname TTDPROJECT % git branch

fatimaa@new-hostname TTDPROJECT % git status

On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
fatimaa@new-hostname TTDPROJECT % git add .

fatimaa@new-hostname TTDPROJECT % git commit -m "Initial commit with loaded datasets"

On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
fatimaa@new-hostname TTDPROJECT % git branch
fatimaa@new-hostname TTDPROJECT % git push -u origin master

error: src refspec master does not match any
error: failed to push some refs to 'https://github.com/CodeFramework-Tech/SmogSense_Project.git'
fatimaa@new-hostname TTDPROJECT %
```

You have not yet opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

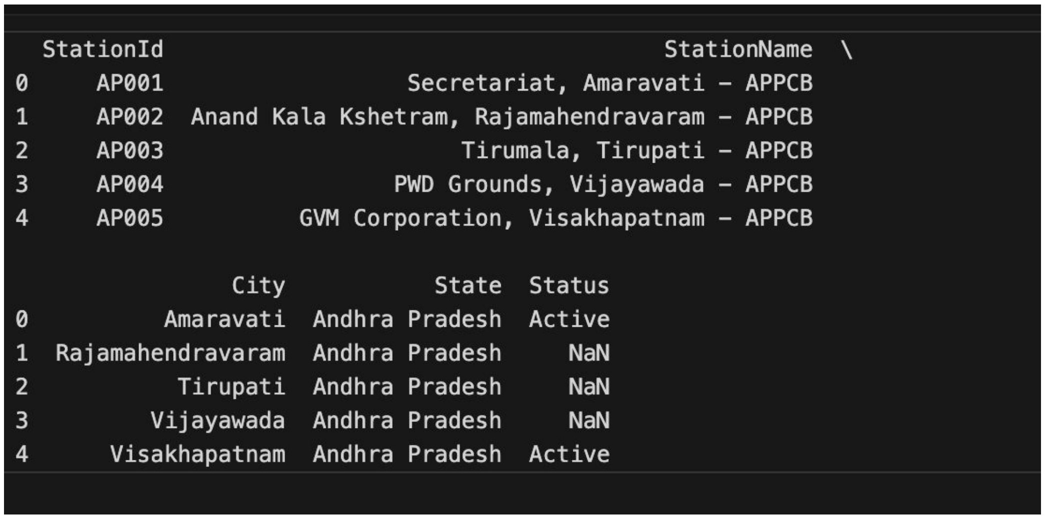
You can clone a repository locally.

To learn more about how to use Git and source control in VS Code read our docs.

Figure 16: lINEAR REGRESSION CODE.

7 Random Forest Model Implementation

Overview of Random Forest Random Forest is an ensemble machine learning algorithm that constructs multiple decision trees during training. It combines the predictions of individual trees to improve the accuracy and robustness of the model. For regression tasks like predicting **PM2.5** levels, Random Forest outputs the average of predictions from all trees.



```
  StationId                                     StationName  \
0     AP001                Secretariat, Amaravati - APPCB
1     AP002  Anand Kala Kshetram, Rajamahendravaram - APPCB
2     AP003                      Tirumala, Tirupati - APPCB
3     AP004                 PWD Grounds, Vijayawada - APPCB
4     AP005           GVM Corporation, Visakhapatnam - APPCB

               City           State  Status
0         Amaravati  Andhra Pradesh  Active
1  Rajamahendravaram  Andhra Pradesh     NaN
2          Tirupati  Andhra Pradesh     NaN
3        Vijayawada  Andhra Pradesh     NaN
4     Visakhapatnam  Andhra Pradesh  Active
```

Figure 17: Model Evaluation (RMSE)

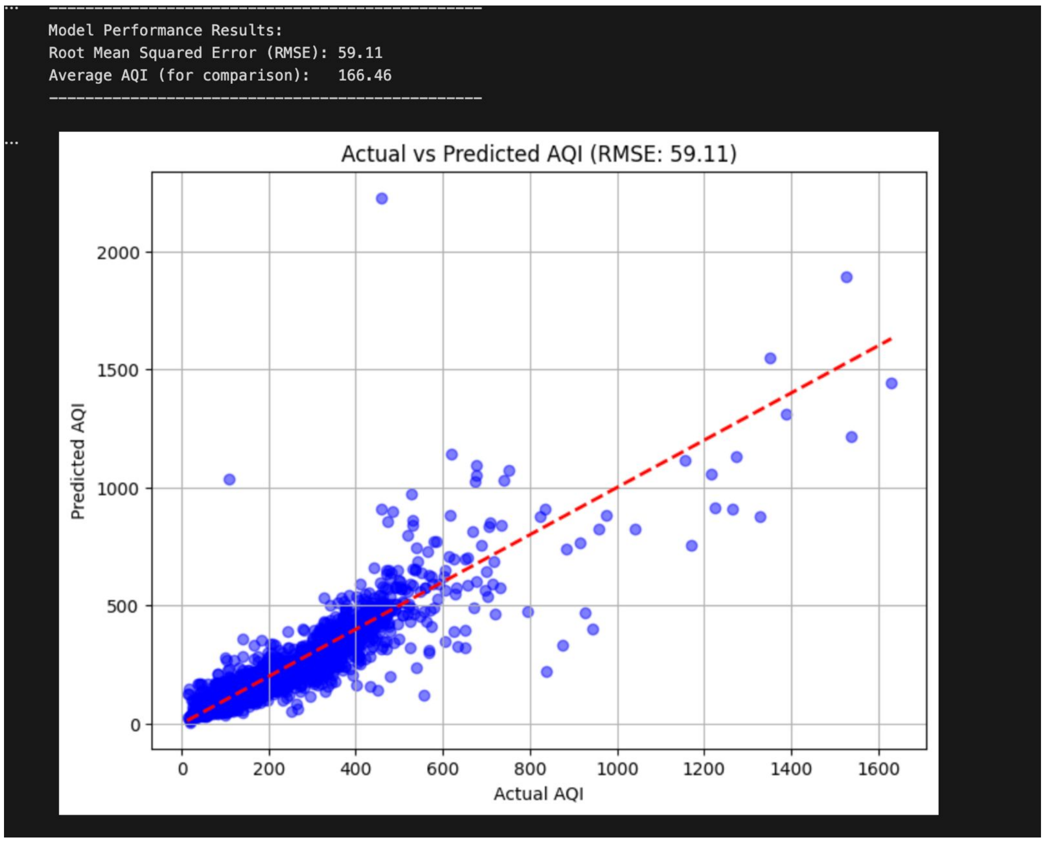
7.1 Steps to Implement Random Forest

The implementation of Random Forest involves several key steps:
1. **Load and Prepare Data:** The dataset is loaded and preprocessed to handle missing values, encode categorical variables, and scale features. 2. **Split Data into Training and Testing Sets:** The data is divided into a training set (80%) and a test set (20%). 3. **Train the Random Forest Model:** We initialize the Random Forest model with a specified number of trees and fit it to the training data. 4. **Make Predictions:** Once the model is trained, we use it to make predictions on the test data. 5. **Evaluate the Model:** The model is evaluated using metrics like Mean Squared Error (MSE) and R-squared (R^2).

7.2 Random Forest Implementation in Python

The following Python code implements the Random Forest model for predicting PM2.5 levels:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
```



```
...  -----------------------------------------------
     Model Performance Results:
     Root Mean Squared Error (RMSE): 59.11
     Average AQI (for comparison):   166.46
     -----------------------------------------------
...
```

Figure 18: Model Evaluation (RMSE).

```
data = pd.read_csv('your_dataset.csv')

X = data[['pm2.5','pm10','no','no2',
'nox','nh3','co','so2','o3', 'benzene', 'toluene', 'xylene']]
y = data['aqi']

X_train, X_test, y_train, y_test
= train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestRegressor(n_estimators=100, random_state=42)

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5
r2 = r2_score(y_test, y_pred)


print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R²): {r2}")
```

7.3 Explanation of Code

The data is loaded from a CSV file and features are selected for training the Random Forest model. The `train_test_split()` function is used to split the data into training and testing sets. A Random Forest Regressor is initialized with 100 trees (`n_estimators=100`). The model is trained using the training data with `model.fit()`. After training, the model makes predictions on the test data using `model.predict()`. The model's performance is then evaluated using the Mean Squared Error (MSE) and R-squared (R^2).

7.4 Random Forest Hyperparameters

The performance of the Random Forest model can be improved by tuning hyperparameters such as:

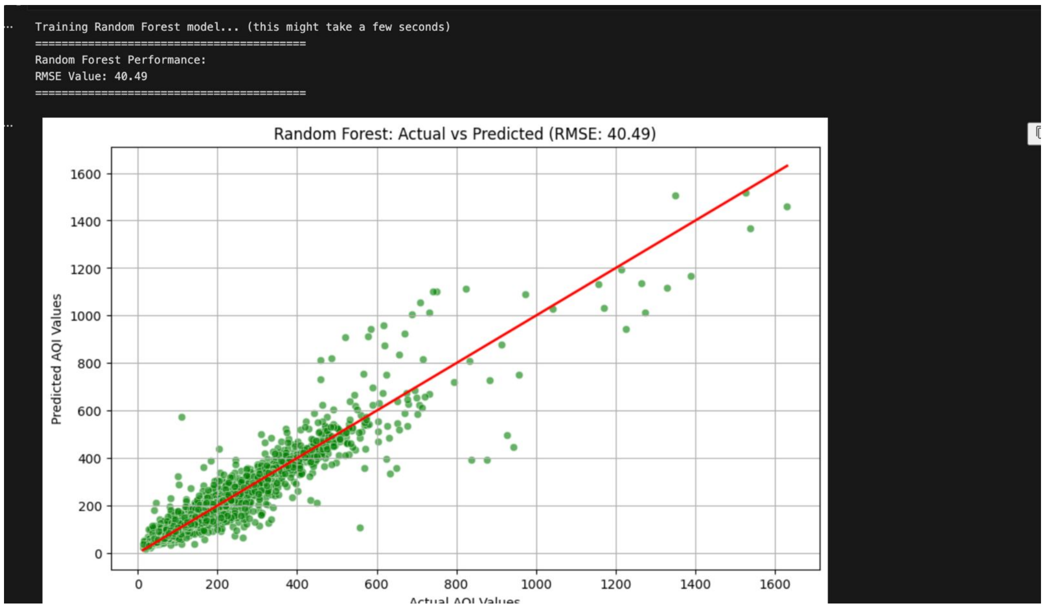
- `n_estimators`: The number of trees in the forest. More trees usually improve the model, but at the cost of increased computation. - `max_depth`: The maximum depth of the trees. Limiting the depth of the trees helps to avoid overfitting. - `min_samples_split`: The minimum number of samples required to split an internal node. Higher values prevent the model from overfitting. - `random_state`: A seed value used to ensure the reproducibility of results.

7.5 Model Evaluation

Once the model is trained, its performance is evaluated using the RMSE and R^2. RMSE gives the average magnitude of the error between the predicted and actual values. A lower RMSE indicates better performance. R^2 measures the proportion of variance explained by the model. A value closer to 1 indicates better performance.

7.6 Random Forest Image Bullet

Below is an illustration of how Random Forest works, showing how multiple trees are built and how they contribute to the final prediction:



```
Training Random Forest model... (this might take a few seconds)
========================================
Random Forest Performance:
RMSE Value: 40.49
========================================
```

Figure 19: Random Forest Model Architecture

8 Conclusion and Future Work

8.1 Why Choose Random Forest?

Random Forest is an ensemble learning algorithm that is particularly well-suited for tasks like predicting air quality. While simpler models such as Linear Regression are faster and easier to implement, they may struggle to capture complex relationships between the features and the target variable. Linear models like Linear Regression assume linear relationships, which may not be suitable for data that exhibits non-linear patterns, such as air quality measurements.

Random Forest, on the other hand, is capable of handling non-linear relationships and can effectively capture interactions between variables. This flexibility allows it to model more complex data patterns. Additionally, Random Forest is robust to overfitting, which is a common issue when using more complex models like decision trees. By averaging the predictions of multiple decision trees, Random Forest reduces the risk of overfitting, even with noisy data.

Moreover, Random Forest provides insights into feature importance, which helps identify the most influential factors affecting air quality predictions. While models like XGBoost may deliver slightly better performance, Random Forest strikes a balance between accuracy, computational efficiency, and ease of use. This makes it a suitable choice for predicting air quality with minimal hyperparameter tuning.

8.2 Conclusion

In conclusion, this project demonstrated the use of machine learning techniques to predict PM2.5 levels based on environmental factors like temperature and humidity. The Linear Regression model provided reasonable accuracy, and we observed that temperature and humidity were significant factors influencing PM2.5 levels.

- Kaggle: `https://www.kaggle.com/datasets/rohanrao/air-quality-data-in-india`
- Python Documentation: `https://pandas.pydata.org/pandas-docs/stable/`
- Scikit-learn Documentation: `https://scikit-learn.org/stable/`