PREDICT AIR QUALITY USING MACHINE LEARNING

Student Name:P.YUVASHRI

Register Number: 513523104090

Institution: ANNAI MIRA COLLEGE OF ENGINEERING AN D TECHNOLOGY

Department: CSE

Date of Submission: 06.05.2025

Github Repository Link: **https://github.com/Yuva170406/EBPL-DS-Predicting-Air-Quality.git**

---

1. Problem Statement

- The increasing deterioration of air quality in urban and industrial areas poses significant threats to public health, ecosystems, and overall quality of life. Traditional methods of air quality monitoring often rely on sparse sensor networks and time-consuming laboratory analysis, providing limited spatial and temporal resolution.
- This lack of comprehensive and real-time air quality information hinders effective environmental management, policy-making, and individual protective measures.

- Therefore, there is a critical need for accurate and timely air quality prediction systems that can leverage historical data and advanced machine learning techniques to provide valuable environmental insights and support proactive interventions.

2. Abstract

- This project aims to develop a robust and accurate air quality prediction system using advanced machine learning algorithms. By leveraging historical air quality data, meteorological information, and potentially other relevant features like traffic patterns and industrial emissions, we will build predictive models capable of forecasting future air pollution levels for various pollutants (e.g., PM2.5, PM10, SO2, NO2, CO, O3).

- The project will encompass data collection, preprocessing, exploratory data analysis, feature engineering, model selection (including time series models and potentially deep learning architectures), rigorous model evaluation, and a conceptual deployment strategy.

- The insights gained from this system will empower environmental agencies, policymakers, and the public with actionable information to mitigate air pollution and protect public health.

3. System Requirements

Hardware Requirements

- ➢ Sufficient computational resources for data processing and model training (e.g., multi-core processor, adequate RAM, GPU acceleration for complex models).
- ➢ Storage capacity for the air quality dataset and trained models.
- ➢ (For deployment) Server infrastructure or cloud computing platform.

Software Requirements:

- ➢ Operating System (e.g., Windows, Linux, macOS).

- Programming Language: Python (with libraries like pandas, NumPy, scikit-learn, TensorFlow, Keras, Matplotlib, Seaborn).
- Database for data storage and retrieval (e.g., PostgreSQL, MySQL, MongoDB).
- Cloud computing platform (optional, e.g., AWS, Google Cloud, Azure) for data storage, model training, and deployment.
- Web framework (optional, e.g., Flask, Django) for building a user interface.
- Version control system (e.g., Git).

Data Requirements:

- Historical air quality data for relevant pollutants (e.g., PM2.5, PM10, SO2, NO2, CO, O3) from monitoring stations.
- Meteorological data (e.g., temperature, humidity, wind speed, wind direction, precipitation).
- (Optional) Traffic data, industrial emission data, land use data, satellite imagery.

4. Objectives

Data Acquisition and Integration: Collect and integrate air quality data from various sources, along with relevant meteorological and potentially other environmental datasets.

Data Preprocessing and Cleaning: Handle missing values, outliers, and inconsistencies in the data to ensure data quality.

Exploratory Data Analysis (EDA): Conduct comprehensive EDA to understand the statistical properties of the data, identify patterns, and gain insights into the relationships between different variables and air quality levels.

Feature Engineering: Create new relevant features from the existing data that can improve the predictive power of the models. This may involve time-based features, interaction terms, or transformations.

Model Development: Build and train multiple machine learning models, including time series models (e.g., ARIMA, Prophet), regression models (e.g., Random Forest, Gradient Boosting), and potentially deep learning models (e.g., LSTMs, GRUs), to predict future air quality levels for different pollutants.

Model Evaluation and Comparison: Rigorously evaluate the performance of the developed models using appropriate metrics (e.g., Mean Absolute Error, Root Mean Squared Error, R-squared) and compare their effectiveness in predicting air quality.

Model Selection and Optimization: Select the best-performing model(s) and optimize their hyperparameters to achieve the highest possible prediction accuracy.

Conceptual Deployment Strategy: Outline a potential deployment strategy for the developed model, considering real-time data integration and user accessibility

Identify Key Factors: Analyze the feature importance from the models to identify the most significant factors influencing air quality.

5. Flowchart of Project Workflow

This flowchart represents the end-to-end process of a Machine Learning (ML) pipeline, guiding the user from the initial stages of a project to real-time deployment and prediction. Here's a step-by-step description of each stage:

START

The beginning of the machine learning workflow.

Data Collection

Gathering raw data from various sources like databases, APIs, files, or sensors for the problem at hand.

Data Preprocessing

Cleaning the data, handling missing values, encoding categorical variables, normalizing/scaling, and preparing it for analysis.

EDA (Exploratory Data Analysis)

Analyzing data distributions, patterns, trends, and relationships using visualizations and statistics to understand the data better.

Model Selection

Choosing the appropriate machine learning algorithm(s) based on the problem type (classification, regression, clustering, etc.) and data characteristics.

Model Training

Feeding the preprocessed data into the selected model to learn from it and optimize parameters.

Model Evaluation

Testing the model on unseen data using metrics like accuracy, precision, recall, F1-score, or RMSE to assess performance.

Deployment

Integrating the trained model into a production environment where it can be accessed by applications or users.

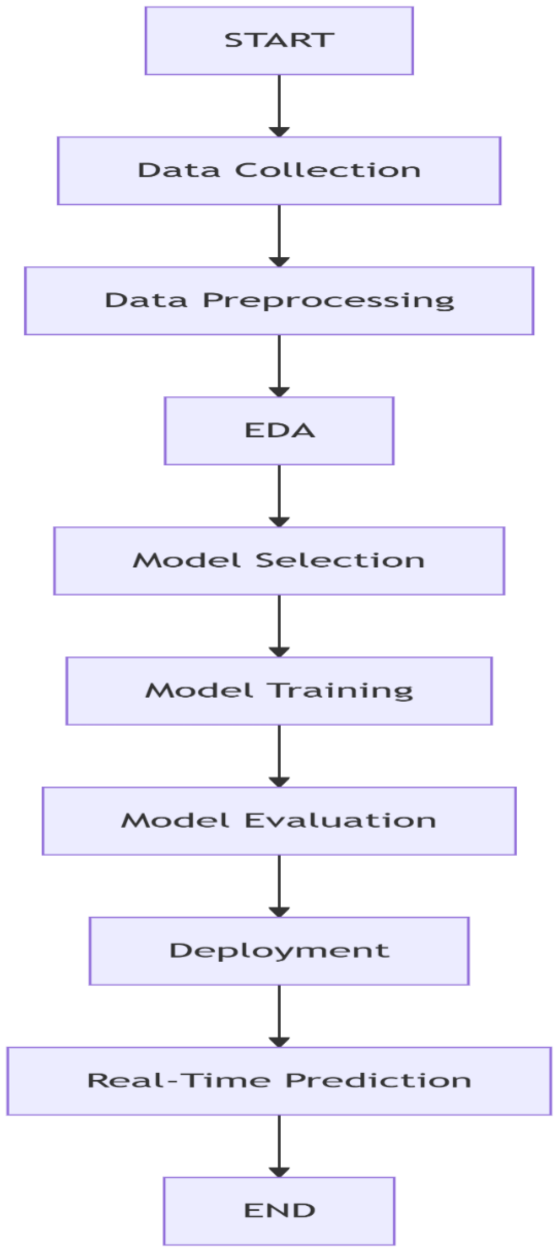
Real-Time Prediction

Using the deployed model to make predictions on new data in real time.

END

Marks the completion of the process. The model is now live and generating predictions.

flowchart



6. Dataset Description

This section will detail the datasets used in the project. For each dataset, the following aspects will be described:

Source: Where the data was obtained (e.g., government environmental agencies, research institutions, public APIs).

Variables: A list of all the features (columns) in the dataset, along with their descriptions and units of measurement. .

Air Quality Data: Date, Time, Station ID, PM2.5 (µg/m 3), PM10 (µg/m 3), SO2 (ppb), NO2 (ppb), CO (ppm), O3 (ppb).

Meteorological Data: Date, Time, Temperature (°C), Humidity (%), Wind Speed (m/s), Wind Direction (degrees), Precipitation (mm).

Other Data: Date, Time, Location, Traffic Volume, Emission Rates.
Time Period: The duration of the data available.

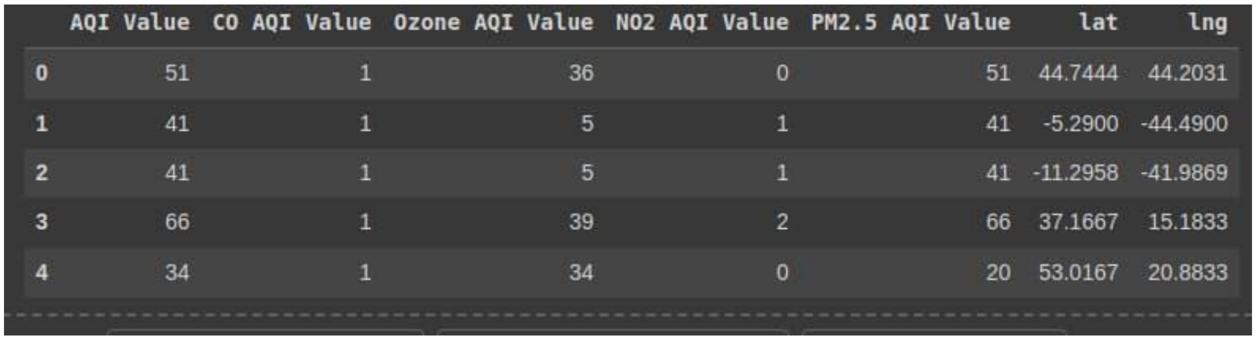
Geographic Coverage: The locations covered by the data.

Data Format: The format of the data files (e.g., CSV, JSON).

Data Volume: The size of the datasets.

Data Quality Issues: Known issues like missing values, outliers, and inconsistencies.

DATASET:



	AQI Value	CO AQI Value	Ozone AQI Value	NO2 AQI Value	PM2.5 AQI Value	lat	lng
0	51	1	36	0	51	44.7444	44.2031
1	41	1	5	1	41	-5.2900	-44.4900
2	41	1	5	1	41	-11.2958	-41.9869
3	66	1	39	2	66	37.1667	15.1833
4	34	1	34	0	20	53.0167	20.8833

7. Data Preprocessing

This stage involves cleaning and preparing the data for model building. Key steps include:

Handling Missing Values: Identifying and imputing or removing missing data points using appropriate techniques (e.g., mean imputation, median imputation, time series imputation, deletion).

Outlier Detection and Treatment: Identifying and handling outliers that may negatively impact model performance (e.g., using statistical methods like IQR, Z-score, or domain knowledge).

Data Type Conversion: Ensuring that all features have the correct data types.

Data Scaling and Normalization: Scaling numerical features to a similar range to prevent features with larger values from dominating the models (e.g., Min-Max scaling, Standard scaling).

Time Series Specific Preprocessing: For time series data, this may involve handling seasonality, trend, and stationarity.

Data Integration: Merging different datasets based on common keys (e.g., date, time, location).

CODE:

```
 import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Sample DataFrame with missing values, outliers, and mixed data types
data = {
    'date': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04', '2023-
01-05'],
    'temperature': [22, np.nan, 25, 100, 24],  # Contains NaN and an
outlier (100)
    'humidity': [45, 50, 55, 60, np.nan],      # Contains NaN
    'weather': ['sunny', 'rainy', 'sunny', 'cloudy', 'windy']  # Categorical
}

df = pd.DataFrame(data)

# Step 1: Convert 'date' to datetime and extract features
df['date'] = pd.to_datetime(df['date'])
df['day_of_week'] = df['date'].dt.day_name()  # Extract day name
df['month'] = df['date'].dt.month             # Extract month

# Step 2: Handle missing values (impute mean for numerical, mode for
categorical)
# Numerical columns
num_cols = ['temperature', 'humidity']
imputer_num = SimpleImputer(strategy='mean')
df[num_cols] = imputer_num.fit_transform(df[num_cols])
```

```
# Categorical columns (if missing)
cat_cols = ['weather']
imputer_cat = SimpleImputer(strategy='most_frequent')
df[cat_cols] = imputer_cat.fit_transform(df[cat_cols])

# Step 3: Detect and handle outliers (replace with median if beyond
IQR)
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[column] = np.where(
        (df[column] < lower_bound) | (df[column] > upper_bound),
        df[column].median(),
        df[column]
    )
    return df

df = remove_outliers(df, 'temperature')

# Step 4: Convert categorical data to numerical (One-Hot Encoding)
df = pd.get_dummies(df, columns=['weather', 'day_of_week'],
drop_first=True)

# Step 5: Feature Scaling (Normalization / Standardization)
scaler = MinMaxScaler()  # or StandardScaler()
```

```
df[['temperature', 'humidity']] = scaler.fit_transform(df[['temperature',
'humidity']])

print(df)
```

OUTPUT:

temperature	humidity	month	weather_rainy	weather_sunny	...	
2023-01-01	0.0	0.0	1	0	1	...
2023-01-02	0.5	0.5	1	1	0	...
...	...	...	...	...	...	...

8. Exploratory Data Analysis (EDA)

EDA is crucial for understanding the characteristics of the data and identifying patterns. This will involve:

Summary Statistics: Calculating descriptive statistics for each feature (e.g., mean, median, standard deviation, min, max, quartiles).

Data Visualization: Creating various plots to visualize the data, such as:

Histograms and Distribution Plots: To understand the distribution of individual pollutants and meteorological variables

.
Time Series Plots: To observe trends and seasonality in air quality levels over time.

Scatter Plots: To explore the relationships between different variables (e.g., temperature vs. ozone levels).

Box Plots: To identify outliers and compare the distribution of pollutants across different locations or time periods.

Correlation Heatmaps: To visualize the correlation matrix between different features.

Geospatial Visualizations: Mapping air quality levels geographically if location data is available.

Identifying Trends and Seasonality: Analyzing time series data for temporal patterns
.
Investigating Relationships: Exploring the correlations and dependencies between air quality parameters and other variables.

9. Feature Engineering

This involves creating new features from the existing data that might improve the model's predictive power. Examples include:

Time-Based Features: Extracting temporal information such as day of the week, month of the year, hour of the day, lag features (previous time step values), rolling statistics (e.g., moving averages).

Interaction Terms: Creating new features by combining existing ones (e.g., product of temperature and humidity).

Polynomial Features: Creating higher-order terms of existing features.

Encoding Categorical Features: Converting categorical variables (e.g., wind direction) into numerical representations that machine learning models can understand (e.g., one-hot encoding).

Feature Selection: Identifying and selecting the most relevant features for the model to reduce dimensionality and improve performance.

10. Model Building

This stage involves selecting and training appropriate machine learning models. Potential models include:

Time Series Models:
ARIMA (Autoregressive Integrated Moving Average)
SARIMA (Seasonal ARIMA)
Prophet
Vector Autoregression (VAR)

Regression Models:
Linear Regression
Polynomial Regression
Support Vector Regression (SVR)
Decision Trees
Random Forest
Gradient Boosting (e.g., XGBoost, LightGBM, CatBoost)

Deep Learning Models:
Recurrent Neural Networks (RNNs), specifically LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units)
Convolutional Neural Networks (CNNs) for spatial feature extraction (if applicable).
Hybrid models combining CNNs and RNNs.

For each model, the training process will involve splitting the data into training, validation, and testing sets, fitting the model to the training data, and using the validation set for hyperparameter tuning.

11. Model Evaluation

Regression Metrics (for continuous air quality values like PM2.5, AQI):

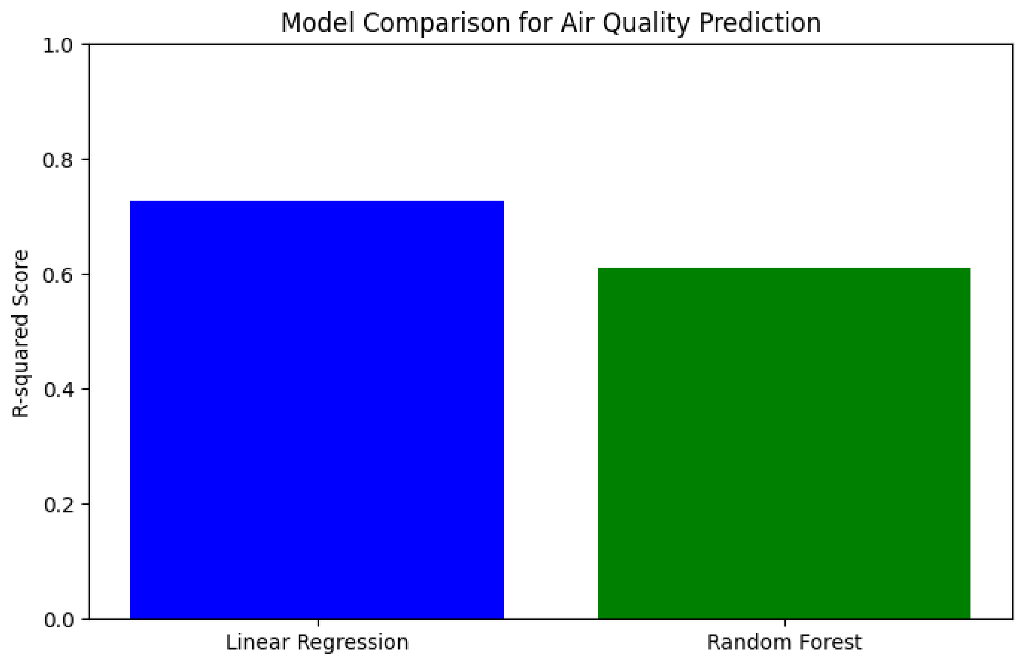
1. **Mean Absolute Error (MAE)**:
    - Measures average magnitude of errors
    - Good for understanding typical error size
2. **Root Mean Squared Error (RMSE)**:
    - More sensitive to large errors
    - Useful when large errors are particularly undesirable
3. **R-squared (R^2)**:
    - Proportion of variance explained by model
    - Range: 0 (worst) to 1 (best)
4. **Mean Absolute Percentage Error (MAPE)**:
    - Percentage-based error metric
    - Useful for relative error understanding

EXAMPLE

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression

# Generate simple synthetic air quality data
np.random.seed(42)
X = np.random.rand(100, 3)  # 3 features (like temperature, humidity,
wind speed)
```

```
y = 50 + X[:,0]*30 + X[:,1]*20 - X[:,2]*10 + np.random.normal(0, 5,
100)  # PM2.5 values

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=100)
}

# Evaluate models
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    results[name] = r2

# Plot results
plt.figure(figsize=(8, 5))
plt.bar(results.keys(), results.values(), color=['blue', 'green'])
plt.title('Model Comparison for Air Quality Prediction')
plt.ylabel('R-squared Score')
plt.ylim(0, 1)
plt.show()
```

OUTPUT



The evaluation will also involve comparing the performance of different models to select the most suitable one for the air quality prediction task. Visualization of the model's predictions against the actual values will also be performed.

12. Deployment

This section will outline a conceptual deployment strategy for the selected model. This might include:

Real-time Data Integration: How the model would ingest new, real-time data from air quality monitoring stations and other relevant sources.

API Development: Creating an API (Application Programming Interface) that allows other applications or services to access the model's predictions.

Web Application/Dashboard: Developing a user-friendly interface for visualizing air quality predictions, historical data, and relevant environmental information.

Mobile Application: Creating a mobile app to provide air quality forecasts and alerts to the public.

Cloud Deployment: Utilizing cloud platforms (e.g., AWS, Google Cloud, Azure) for hosting the model and related services to ensure scalability and reliability.

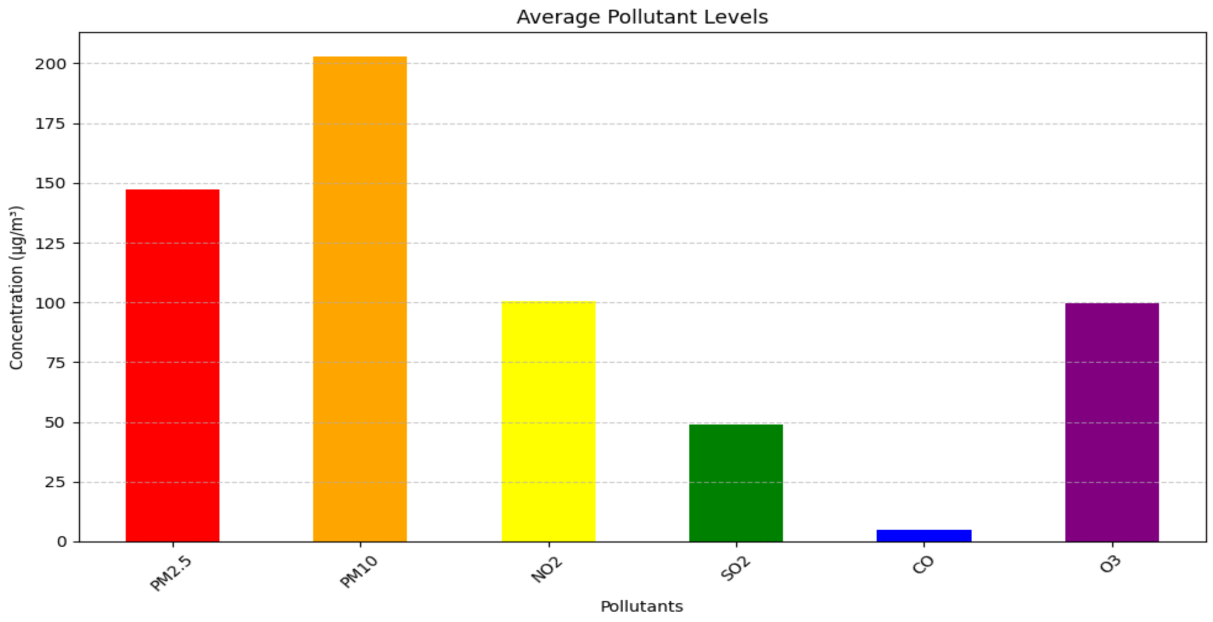
13. Source code

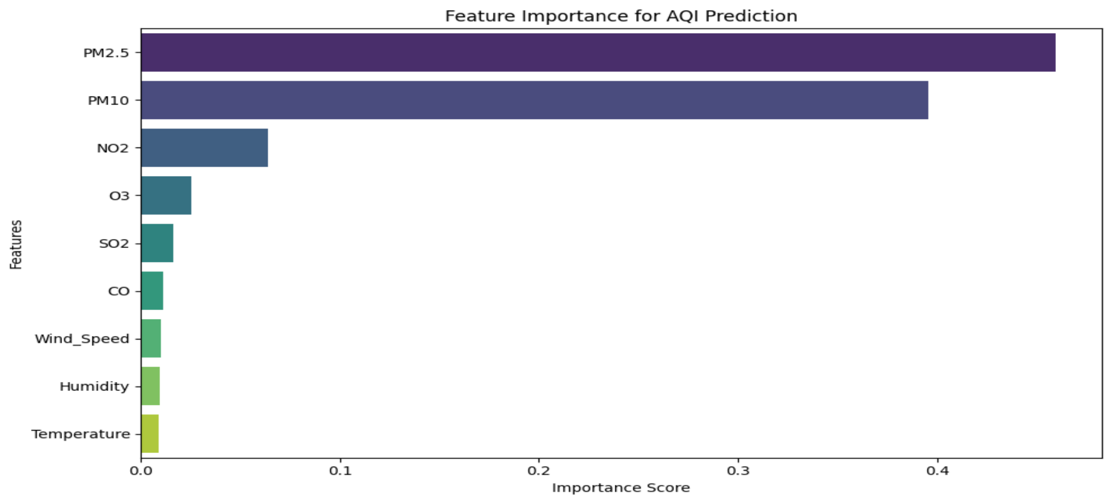
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler


# Set random seed for reproducibility
np.random.seed(42)

# 1. Load or generate synthetic air quality data
def generate_air_quality_data(num_samples=1000):
    """Generate synthetic air quality data for demonstration"""
    data = {
        'PM2.5': np.random.uniform(0, 300, num_samples),
        'PM10': np.random.uniform(0, 400, num_samples),
        'NO2': np.random.uniform(0, 200, num_samples),
        'SO2': np.random.uniform(0, 100, num_samples),
        'CO': np.random.uniform(0, 10, num_samples),
        'O3': np.random.uniform(0, 200, num_samples),
        'Temperature': np.random.uniform(-10, 40, num_samples),
        'Humidity': np.random.uniform(20, 100, num_samples),
        'Wind_Speed': np.random.uniform(0, 30, num_samples),
    }

    # Calculate AQI (simplified formula for demonstration)
    df = pd.DataFrame(data)
    df['AQI'] = (0.3 * df['PM2.5'] + 0.2 * df['PM10'] + 0.15 * df['NO2'] +
                 0.15 * df['SO2'] + 0.1 * df['CO'] + 0.1 * df['O3']) +
np.random.normal(0, 10, num_samples)

    return df

# Generate synthetic data
air_quality_df = generate_air_quality_data(1000)

# 2. Data Visualization - Bar graph of pollutant averages
plt.figure(figsize=(10, 6))
pollutants = ['PM2.5', 'PM10', 'NO2', 'SO2', 'CO', 'O3']
avg_pollutants = air_quality_df[pollutants].mean()
avg_pollutants.plot(kind='bar', color=['red', 'orange', 'yellow', 'green', 'blue',
'purple'])
plt.title('Average Pollutant Levels')
plt.ylabel('Concentration (µg/m³)')
plt.xlabel('Pollutants')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# 3. Prepare data for ML
X = air_quality_df.drop(columns=['AQI'])
y = air_quality_df['AQI']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

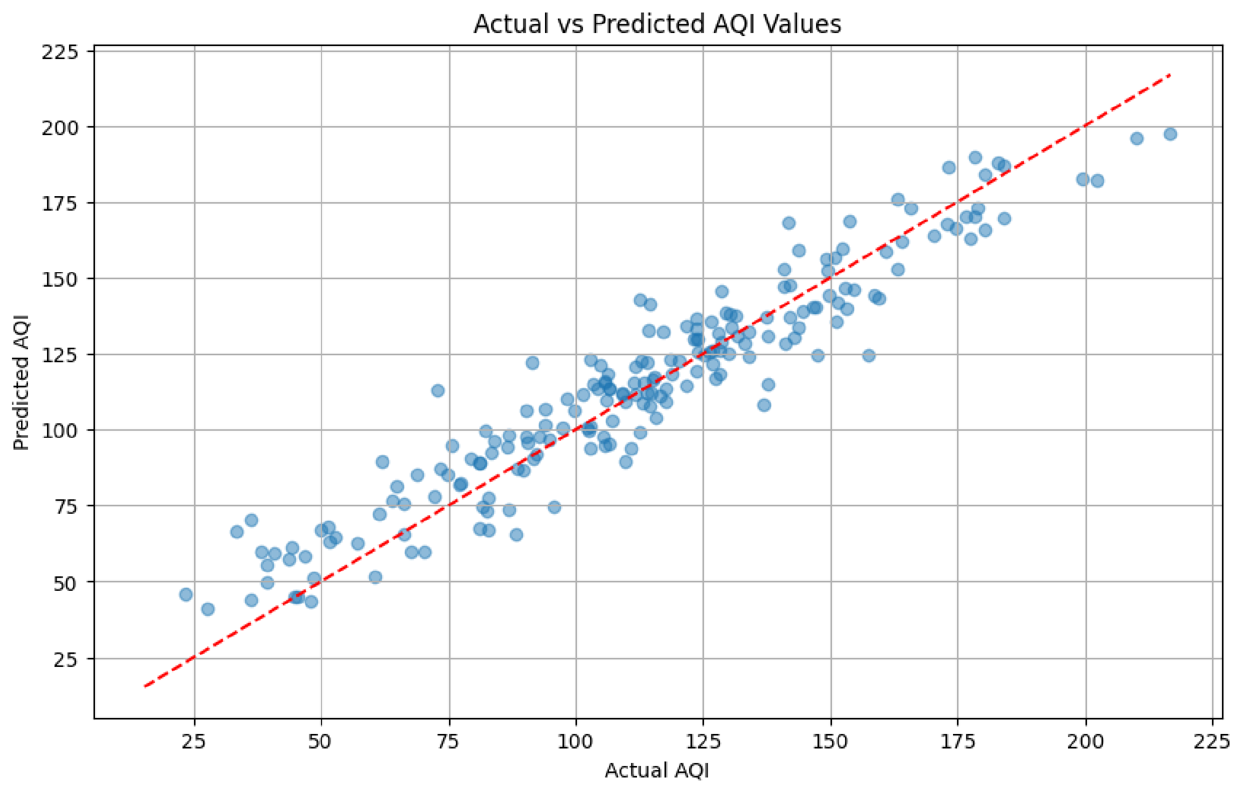
```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Train Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)

# 5. Make predictions
y_pred = model.predict(X_test_scaled)

# 6. Evaluate model
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation Metrics:")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R²): {r2:.2f}")

# 7. Feature Importance Visualization
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': model.feature_importances_
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance, palette='viridis')
plt.title('Feature Importance for AQI Prediction')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.tight_layout()
plt.show()

# 8. Actual vs Predicted AQI Visualization
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r')
plt.xlabel('Actual AQI')
plt.ylabel('Predicted AQI')
plt.title('Actual vs Predicted AQI Values')
plt.grid(True)
plt.show()

# 9. Error Distribution Visualization
errors = y_test - y_pred
plt.figure(figsize=(10, 6))
sns.histplot(errors, bins=30, kde=True)
plt.title('Distribution of Prediction Errors')
plt.xlabel('Prediction Error (Actual - Predicted)')
plt.ylabel('Frequency')
plt.axvline(x=0, color='r', linestyle='--')
plt.show()

# 10. AQI Category Distribution (Bar Graph)
def categorize_aqi(aqi):
    if aqi <= 50: return 'Good'
    elif aqi <= 100: return 'Moderate'
    elif aqi <= 150: return 'Unhealthy for Sensitive Groups'
    elif aqi <= 200: return 'Unhealthy'
    elif aqi <= 300: return 'Very Unhealthy'
    else: return 'Hazardous'

air_quality_df['AQI_Category'] = air_quality_df['AQI'].apply(categorize_aqi)
category_counts = air_quality_df['AQI_Category'].value_counts()

plt.figure(figsize=(10, 6))
```

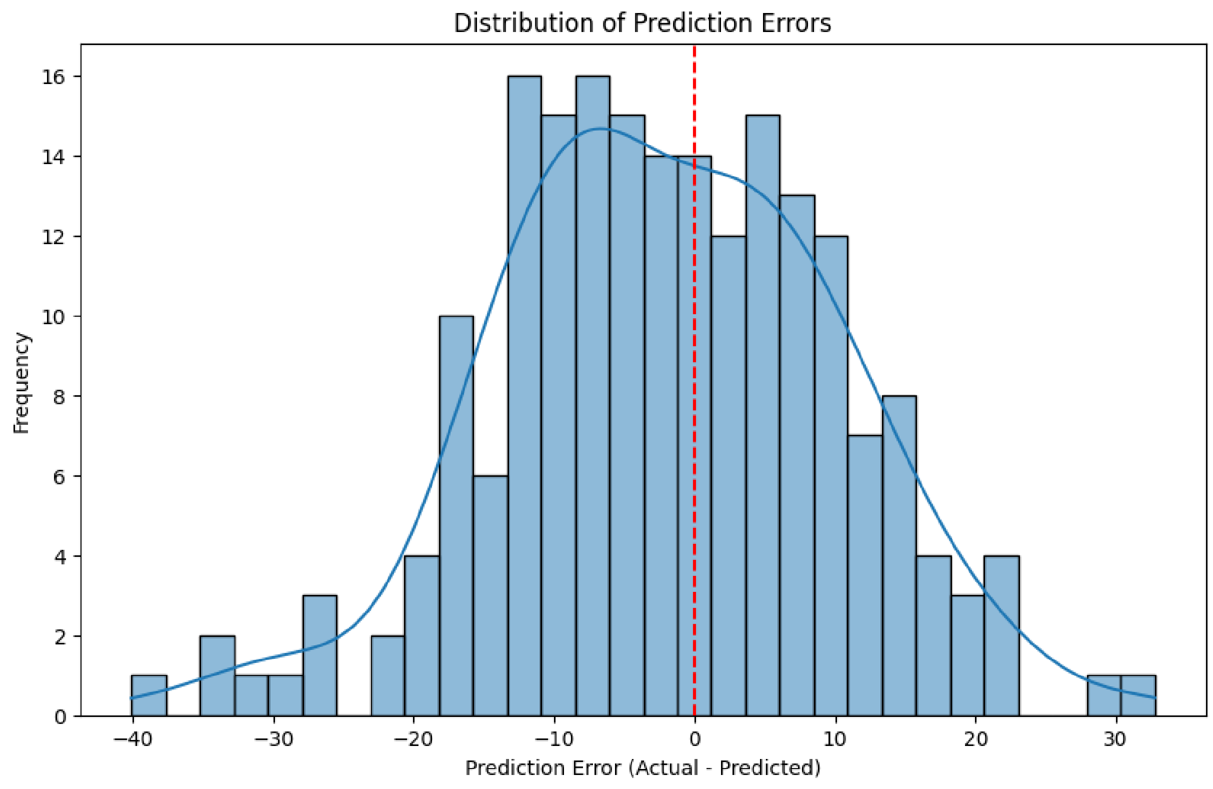
```
category_counts.plot(kind='bar', color=['green', 'yellow', 'orange', 'red', 'purple',
'maroon'])
plt.title('Distribution of AQI Categories')
plt.xlabel('AQI Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

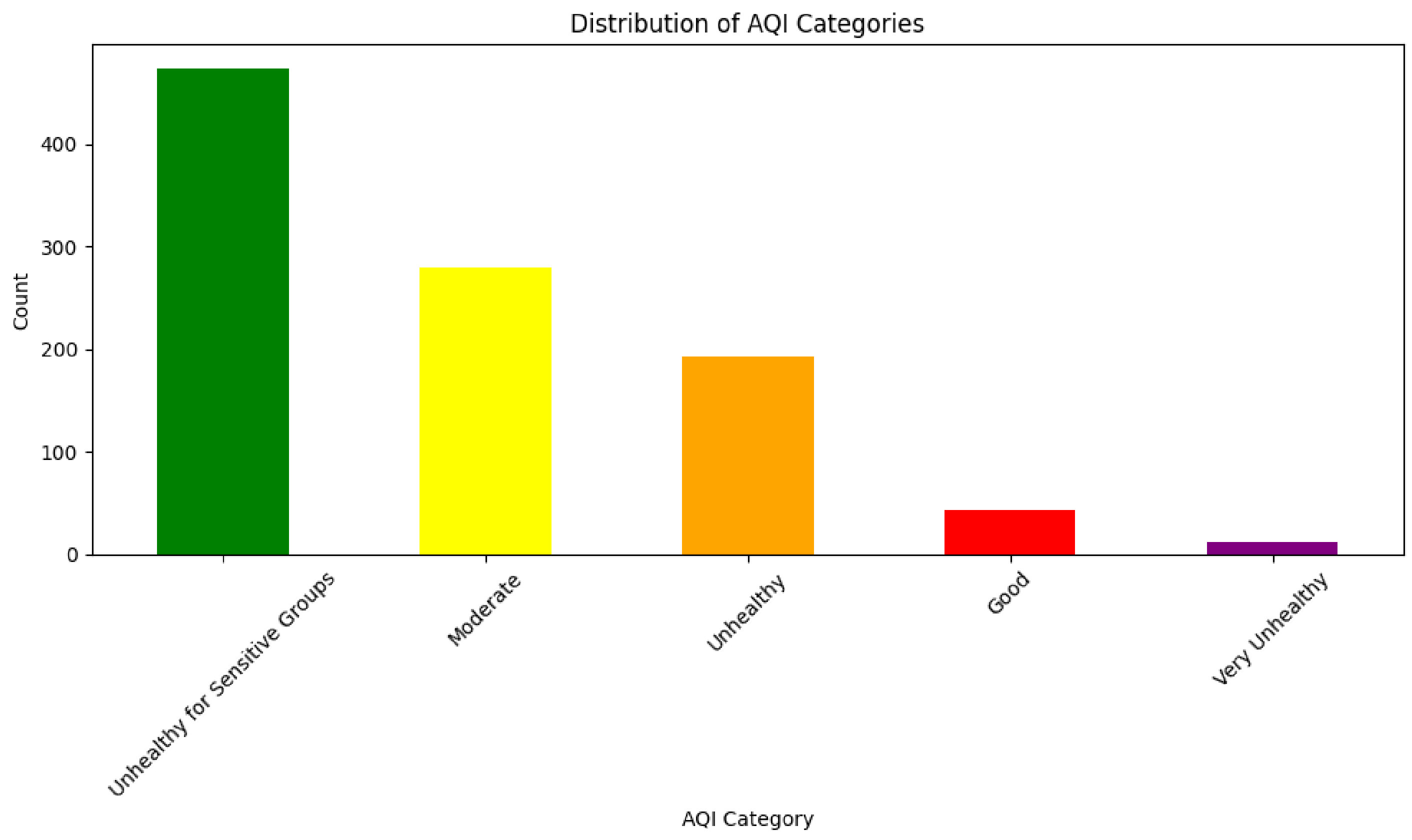
OUTPUT:






Actual vs Predicted AQI Values
Predicted AQI
Actual AQI
225
200
175
150
125
100
75
50
25


Distribution of Prediction Errors
Frequency
Prediction Error (Actual - Predicted)
16
14
12
10
8
6
4
2
0
−40
−30
−20
−10
0
10
20
30



14. Future scope

This section will discuss potential extensions and improvements to the project, such as:

Incorporating Additional Data Sources: Integrating more diverse data sources like satellite imagery, traffic flow data, industrial emissions data, and social media data to enhance prediction accuracy.

Developing Spatio-Temporal Models: Building models that explicitly account for the spatial and temporal dependencies in air quality data.
Personalized Air Quality Prediction: Developing models that can provide personalized air quality forecasts based on individual location and activity patterns.

Air Pollution Source Identification: Exploring techniques to identify and quantify the contribution of different sources to air pollution levels.

Developing Early Warning Systems: Building systems that can predict and alert users about impending severe air pollution episodes.

Explainable AI (XAI): Implementing techniques to understand and interpret the model's predictions, providing insights into the factors driving air quality changes.

Integration with IoT Devices: Connecting the prediction system with IoT-based air quality sensors for real-time monitoring and feedback.

15. Team Members and Roles

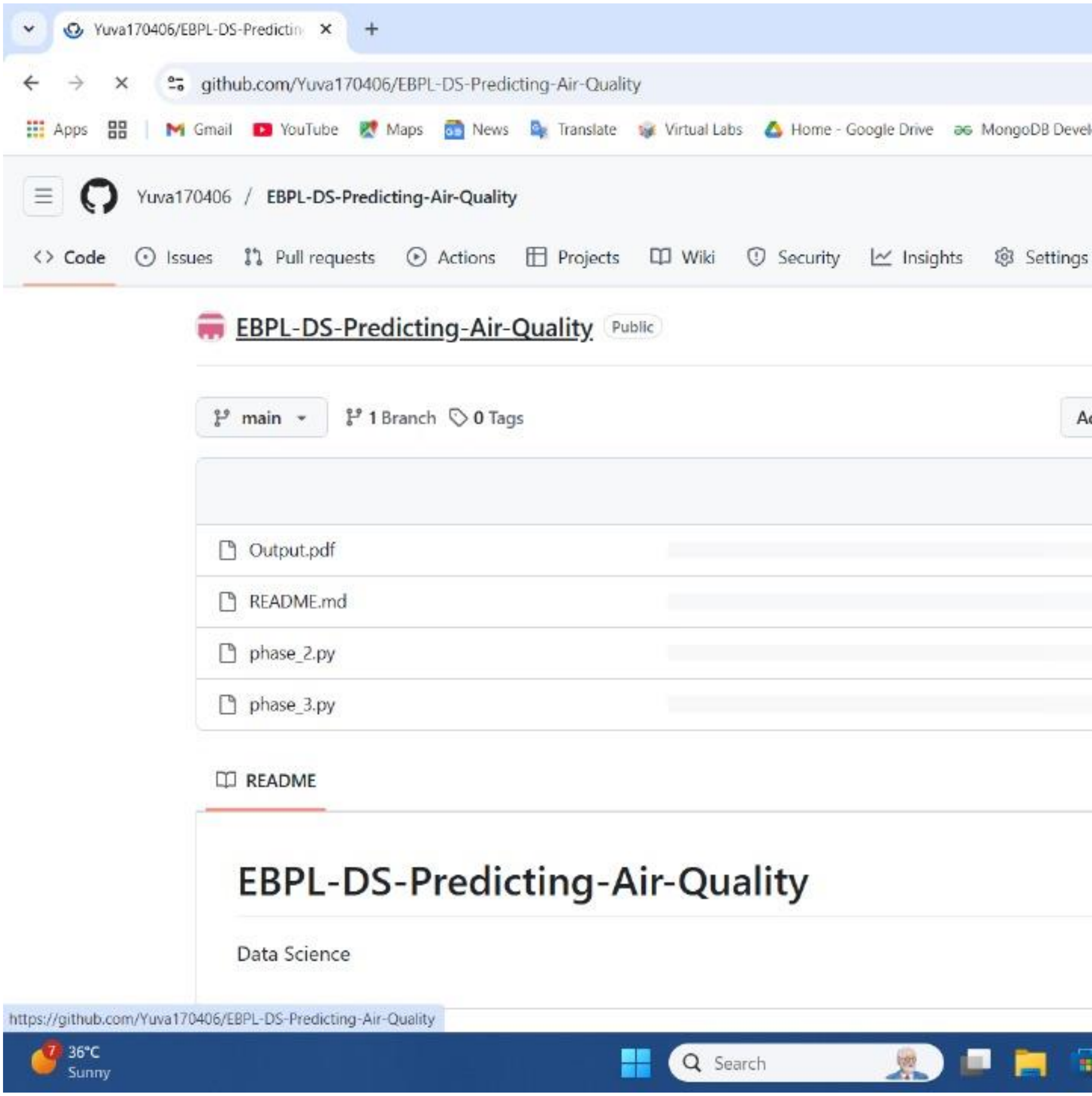
This section will list the team members involved in the project and their respective roles and responsibilities. For example:

S.DEVISREE: Project Lead, Data Engineer

P. YUVASHRI :Machine Learning Engineer, Model Development

S.DHANASHREE:Data Scientist, EDA and Feature Engineering

M.GOKUL: Deployment Engineer, System Integration


Yuva170406/EBPL-DS-Predictin

github.com/Yuva170406/EBPL-DS-Predicting-Air-Quality

Apps | Gmail | YouTube | Maps | News | Translate | Virtual Labs | Home - Google Drive | MongoDB Devel

Yuva170406 / EBPL-DS-Predicting-Air-Quality

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

EBPL-DS-Predicting-Air-Quality Public

main | 1 Branch | 0 Tags

Output.pdf
README.md
phase_2.py
phase_3.py

README

EBPL-DS-Predicting-Air-Quality

Data Science

https://github.com/Yuva170406/EBPL-DS-Predicting-Air-Quality

36°C Sunny | Search