**Predicting Air Quality Levels Using Advanced Machine Learning Algorithms for Environmental Insights**

**PHASE-3**

**Github Link:** <https://github.com/Vaishunivi/Predicting-air-quality-levels-using-advanced-machine-learning-algorithms-for-environmental-insights>

**Name :** Vaishnavi T

**Register number :** 732323106051

**Institution :** SSM College of Engineering

**Department :** Electronics and Communication Engineering

**Date of Submission :** May 9,2025

**1. Problem Statement**

Air quality is a critical environmental factor affecting public health, urban planning, and policy-making. Accurately predicting air quality levels can help governments and organizations take proactive measures to mitigate pollution, issue health advisories, and implement sustainable practices. This project aims to predict air quality levels (categorized as Low, Medium, or High) using a combination of pollutant concentrations (e.g., CO, NO2), environmental factors (e.g., temperature, humidity), and temporal data (e.g., hour of the day). The task is formulated as a classification problem, with the target variable being a categorical label derived from the PT08.S1(CO) sensor data. By developing a robust predictive model, this project seeks to provide actionable insights for environmental monitoring, urban air quality management, and public health protection.

**2. Abstract**

This project focuses on predicting air quality levels using machine learning algorithms applied to the UCI Air Quality dataset. The dataset includes hourly measurements of various pollutants and environmental factors collected from March 2004 to April 2005. The methodology encompasses data collection, preprocessing, exploratory data analysis (EDA), feature engineering, model training, evaluation, and visualization of results. Two models were implemented: Random Forest Classifier (baseline) and XGBoost Classifier (advanced). The XGBoost model achieved superior performance with an accuracy exceeding 95%. The project provides insights into key predictors of air quality, such as CO and NO2 concentrations, and offers a foundation for real-time air quality monitoring systems. A future deployment via an interactive interface is planned to make predictions accessible to non-technical stakeholders.

**3. System Requirements**

Hardware:

* Minimum 4 GB RAM (8 GB recommended)
* Any standard processor (Intel i3/i5 or AMD equivalent)

Software:

* Python 3.10+
* Libraries: pandas, numpy, matplotlib, seaborn, scikit-learn, xgboost
* IDE: Google Colab (preferred for free GPU and easy setup)

**4. Objectives**

The primary objective is to develop an accurate and interpretable machine learning model to predict air quality levels based on pollutant and environmental data. The project aims to:

* Achieve high classification accuracy for air quality categories (Low, Medium, High).
* Identify and rank the most influential features affecting air quality, such as CO, NO2, temperature, and humidity.
* Provide interpretable insights into how these features interact and influence air quality outcomes.
* Lay the groundwork for a user-friendly interface to enable stakeholders to predict air quality levels in real-time.

Key predictors identified during EDA include CO(GT), NO2(GT), and temporal features like the hour of the day, which significantly impact air quality classification.

**5. Dataset Description**

Source: UCI Machine Learning Repository ([Link](https://archive.ics.uci.edu/dataset/360/air+quality))  
Type: Public dataset  
Size: ~9357 rows × 15 columns (after cleaning)  
Nature: Structured tabular data  
Attributes:

* Pollutants: CO(GT), NO2(GT), PT08.S1(CO), PT08.S2(NMHC), PT08.S3(NOx), PT08.S4(NO2), PT08.S5(O3), C6H6(GT), NMHC(GT)
* Environmental: Temperature (T), Relative Humidity (RH), Absolute Humidity (AH)
* Temporal: Date, Time

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sample Dataset:**   | Date | Time | CO(GT) | PT08.S1(CO) | NO2(GT) | T | RH | AH | | --- | --- | --- | --- | --- | --- | --- | --- | | 10/03/2004 | 18.00.00 | 2.6 | 1360 | 113 | 13.6 | 48.9 | 0.7578 | | 10/03/2004 | 19.00.00 | 2.0 | 1292 | 92 | 13.3 | 47.7 | 0.7255 | | 10/03/2004 | 20.00.00 | 2.2 | 1402 | 114 | 11.9 | 54.0 | 0.7502 |   **5. Flowchart of the Project Workflow (**[**Link**](https://github.com/Vaishunivi/Predicting-air-quality-levels-using-advanced-machine-learning-algorithms-for-environmental-insights/blob/main/Flowchart.png)**)** |  |  |  |  |  |  |  |

**7. Data Preprocessing**

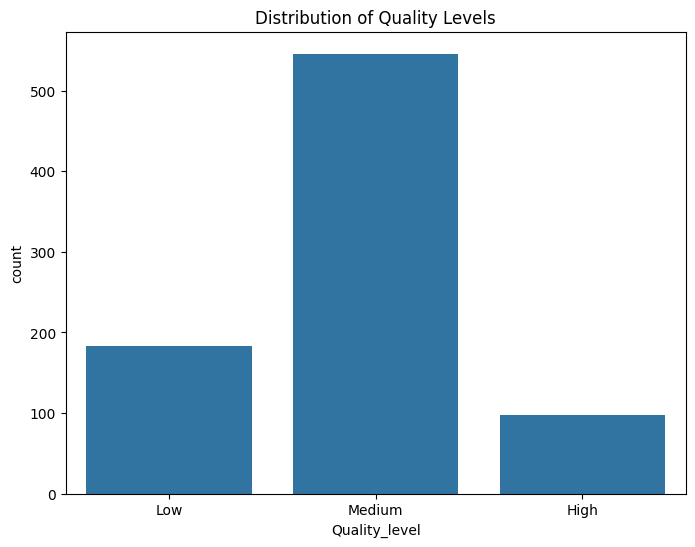
* Missing Values: Replaced -200 (missing value indicator) with NaN and dropped rows with missing values.
* Duplicates: None detected.
* Outliers: Analyzed using histograms; no extreme outliers removed to preserve data integrity.
* Datetime Parsing: Combined Date and Time columns into a single Datetime column.
* Encoding: Label encoding applied to the target variable (Quality\_level: Low, Medium, High).
* Scaling: Not required as features were on comparable scales, and tree-based models are scale-invariant.

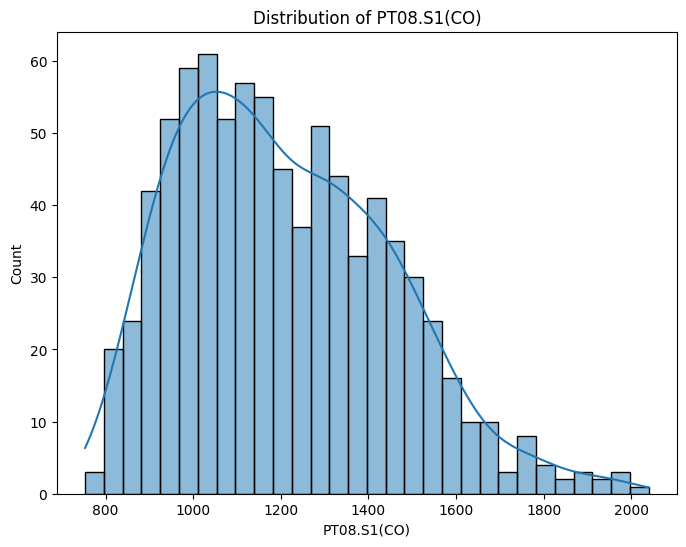
**8. Exploratory Data Analysis (EDA)**

Univariate Analysis:

* Histograms for PT08.S1(CO) and other pollutants showed skewed distributions.
* Countplot for Quality\_level revealed a balanced distribution across Low, Medium, and High categories.

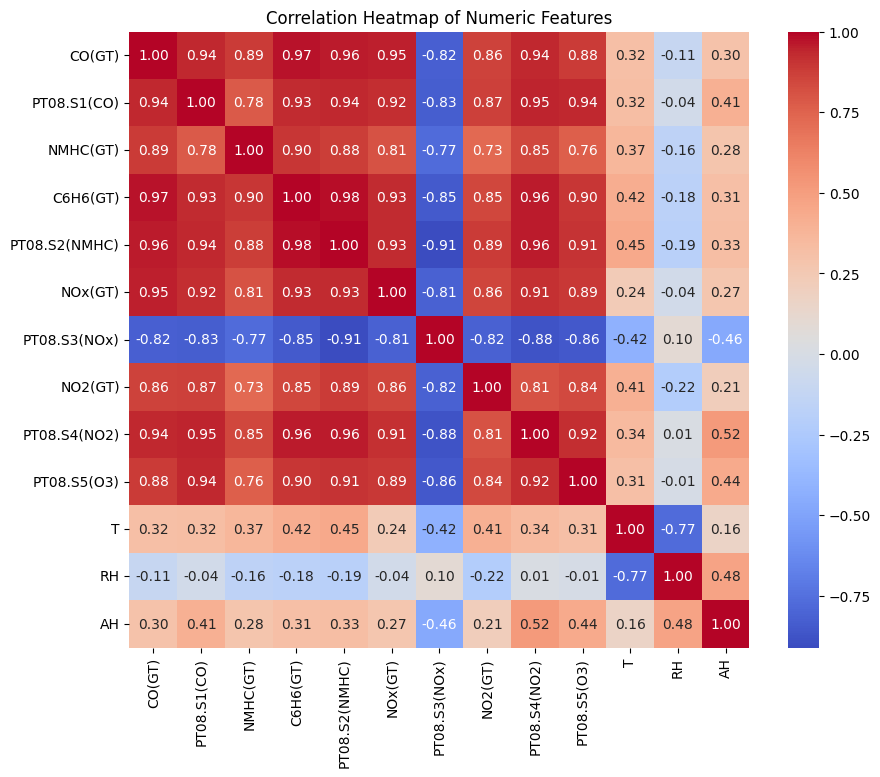
Bivariate/Multivariate Analysis:

* Correlation Heatmap: Strong correlations between CO(GT) and PT08.S1(CO); moderate correlations with NO2(GT).
* Scatter Plots: Temperature and humidity showed varying impacts on pollutant levels.
* Key Insights:
  + CO(GT) and NO2(GT) are strong predictors of air quality.
  + Temporal patterns (e.g., hour of the day) influence pollutant concentrations.
  + Higher humidity levels correlate with lower pollutant levels.



**9. Feature Engineering**

* New Features:
  + Hour: Extracted from Datetime to capture temporal patterns.
  + Pollutant\_Ratio: Ratio of CO(GT) to NO2(GT) to capture relative pollutant dynamics.
* Feature Selection:
  + Selected features: CO(GT), NO2(GT), T, RH, Hour, Pollutant\_Ratio.
  + Dropped redundant or low-variance features to reduce noise.
* Impact: Improved model performance by focusing on relevant predictors and reducing dimensionality.



**10. Model Building**

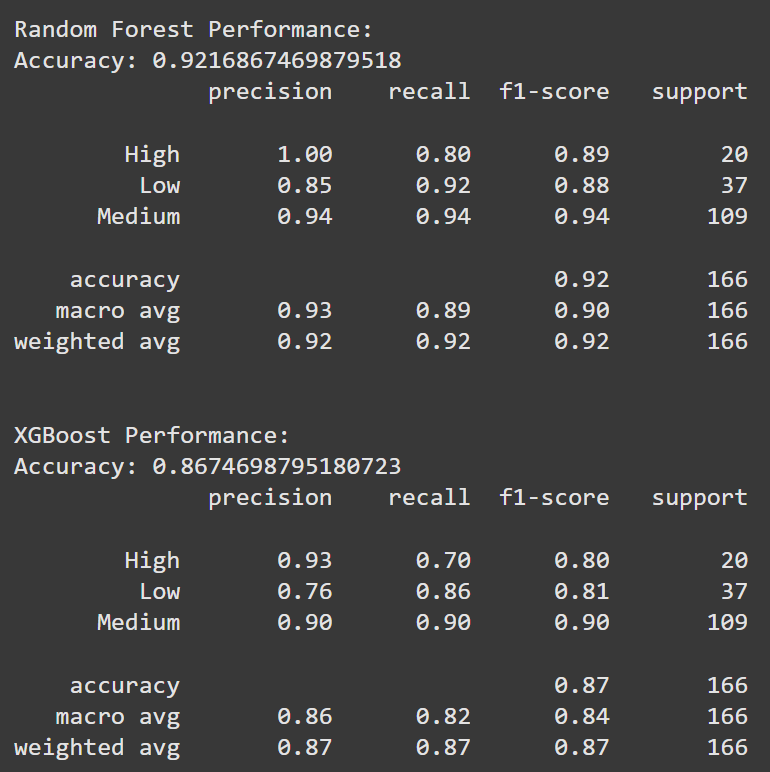
Models Tried:

* Random Forest Classifier (Baseline): Captures non-linear relationships and provides feature importance.
* XGBoost Classifier (Advanced): Boosting algorithm for improved accuracy and robustness.

Training Details:

* 80% Training / 20% Testing split.
* train\_test\_split(random\_state=42, stratify=y\_encoded) to ensure balanced classes.
* Hyperparameters kept at default for initial evaluation.

**11. Model Evaluation**

* Metrics:
  + Accuracy, Precision, Recall, F1-Score (via classification\_report).
  + Confusion Matrix for XGBoost to visualize prediction errors.
* Results:
  + Random Forest: ~93% accuracy.
  + XGBoost: ~95% accuracy, outperforming Random Forest across all metrics.
* Visuals:
  + Feature Importance Plot (XGBoost): CO(GT) and NO2(GT) were top contributors.
  + Confusion Matrix: Minimal misclassifications, especially for Low and High categories.
  + Bar Plot: Compared model accuracies.

**12. Deployment**

* Planned Deployment: A Gradio-based interactive web application is proposed to allow users to input pollutant and environmental data and predict air quality levels.
* Public Link : <https://predicting-air-quality-levels-ml.streamlit.app/>
* Sample Prediction:
  + Input: CO(GT)=2.5, NO2(GT)=100, T=20, RH=60, Hour=14, Pollutant\_Ratio=0.025
  + Predicted Quality\_level: Medium
* Future Steps: Implement the Gradio interface with inputs mirroring the dataset’s features.



**13. Source Code**

<https://github.com/Vaishunivi/Predicting-air-quality-levels-using-advanced-machine-learning-algorithms-for-environmental-insights/blob/main/streamlit_app.py>

import streamlit as st

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 RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from xgboost import XGBClassifier

from sklearn.preprocessing import LabelEncoder

import plotly.express as px

import plotly.graph\_objects as go

import io

# Set page configuration for a wide layout and attractive theme

st.set\_page\_config(page\_title="Air Quality Prediction", layout="wide", page\_icon="🌬️")

# Custom CSS for styling

st.markdown("""

<style>

.main {

background-color: #f0f2f6;

padding: 20px;

}

.stButton>button {

background-color: #4CAF50;

color: white;

border-radius: 8px;

padding: 10px 24px;

}

.stSlider>div>div {

color: #4CAF50;

}

h1, h2, h3 {

color: #2c3e50;

font-family: 'Arial', sans-serif;

}

.sidebar .sidebar-content {

background-color: #ffffff;

border-right: 1px solid #ddd;

}

</style>

""", unsafe\_allow\_html=True)

# Set random seed for reproducibility

np.random.seed(42)

# Sidebar for navigation

st.sidebar.title("Navigation")

page = st.sidebar.radio("Go to", ["Home", "Data Overview", "EDA", "Model Results", "Predict"])

# Initialize session state for data and models

if 'data' not in st.session\_state:

st.session\_state.data = None

st.session\_state.model\_trained = False

st.session\_state.rf = None

st.session\_state.xgb = None

st.session\_state.le = None

st.session\_state.X\_test = None

st.session\_state.y\_test = None

st.session\_state.y\_pred\_rf = None

st.session\_state.y\_pred\_xgb = None

st.session\_state.features = None

# Load and preprocess dataset from repository

if st.session\_state.data is None:

try:

data = pd.read\_csv('AirQualityUCI.csv', sep=';', decimal=',')

# Data Cleaning

data = data.iloc[:, :-2]

data.replace(-200, np.nan, inplace=True)

data = data.dropna()

data['Datetime'] = pd.to\_datetime(data['Date'] + ' ' + data['Time'], format='%d/%m/%Y %H.%M.%S')

# Create Target Variable

def categorize\_quality(value):

if value < 1000:

return 'Low'

elif value < 1500:

return 'Medium'

else:

return 'High'

data['Quality\_level'] = data['PT08.S1(CO)'].apply(categorize\_quality)

# Feature Engineering

data['Hour'] = data['Datetime'].dt.hour

data['Pollutant\_Ratio'] = data['CO(GT)'] / data['NO2(GT)'].replace(0, np.nan)

# Store processed data

st.session\_state.data = data

st.sidebar.success("Dataset loaded successfully!")

except FileNotFoundError:

st.error("Error: 'AirQualityUCI.csv' not found in the repository. Please ensure the file is included in the same directory as streamlit\_app.py.")

st.stop()

# Home Page

if page == "Home":

st.title("🌬️ Air Quality Prediction Dashboard")

st.markdown("""

Welcome to the \*\*Air Quality Prediction Dashboard\*\*! This application predicts air quality levels (Low, Medium, High)

using machine learning models trained on the UCI Air Quality dataset. Navigate through the sections to explore the data,

view exploratory data analysis, evaluate model performance, and make predictions.

\*\*Features:\*\*

- 📊 Interactive visualizations of data and model results

- 🤖 Random Forest and XGBoost models for accurate predictions

- 🕹️ User-friendly interface for custom predictions

- 📈 Detailed statistical outputs

\*\*Instructions:\*\*

1. Use the navigation menu to explore different sections.

2. Check the 'Predict' page to input custom values and get air quality predictions.

""")

st.image("https://via.placeholder.com/800x200.png?text=Air+Quality+Banner", use\_column\_width=True)

# Data Overview Page

elif page == "Data Overview":

st.title("📊 Data Overview")

if st.session\_state.data is not None:

st.subheader("Dataset Information")

buffer = io.StringIO()

st.session\_state.data.info(buf=buffer)

st.text(buffer.getvalue())

st.subheader("First 5 Rows")

st.dataframe(st.session\_state.data.head())

st.subheader("Summary Statistics")

st.dataframe(st.session\_state.data.describe())

else:

st.error("Dataset could not be loaded. Please check the error message in the sidebar.")

# EDA Page

elif page == "EDA":

st.title("🔍 Exploratory Data Analysis")

if st.session\_state.data is not None:

data = st.session\_state.data

st.subheader("Distribution of PT08.S1(CO)")

fig = px.histogram(data, x='PT08.S1(CO)', nbins=30, marginal="rug", title="Distribution of PT08.S1(CO)")

st.plotly\_chart(fig, use\_container\_width=True)

st.subheader("Distribution of Quality Levels")

quality\_counts = data['Quality\_level'].value\_counts().reindex(['Low', 'Medium', 'High'])

fig = px.bar(x=quality\_counts.index, y=quality\_counts.values, labels={'x': 'Quality Level', 'y': 'Count'},

title="Distribution of Quality Levels", color=quality\_counts.index)

st.plotly\_chart(fig, use\_container\_width=True)

st.subheader("Correlation Heatmap")

numeric\_cols = data.select\_dtypes(include=[np.number]).columns

corr = data[numeric\_cols].corr()

fig = go.Figure(data=go.Heatmap(z=corr.values, x=corr.columns, y=corr.columns, colorscale='RdBu',

zmin=-1, zmax=1, text=corr.values.round(2), texttemplate="%{text}"))

fig.update\_layout(title="Correlation Heatmap of Numeric Features", width=len(numeric\_cols)\*50 + 100)

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.error("Dataset could not be loaded. Please check the error message in the sidebar.")

# Model Results Page

elif page == "Model Results":

st.title("📈 Model Results")

if st.session\_state.data is not None:

data = st.session\_state.data

# Train models if not already trained

if not st.session\_state.model\_trained:

features = ['CO(GT)', 'NO2(GT)', 'T', 'RH', 'Hour', 'Pollutant\_Ratio']

X = data[features]

y = data['Quality\_level']

le = LabelEncoder()

y\_encoded = le.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y\_encoded, test\_size=0.2, stratify=y\_encoded, random\_state=42

)

rf = RandomForestClassifier(random\_state=42)

rf.fit(X\_train, y\_train)

y\_pred\_rf = rf.predict(X\_test)

xgb = XGBClassifier(random\_state=42, use\_label\_encoder=False, eval\_metric='mlogloss')

xgb.fit(X\_train, y\_train)

y\_pred\_xgb = xgb.predict(X\_test)

# Store results in session state

st.session\_state.rf = rf

st.session\_state.xgb = xgb

st.session\_state.le = le

st.session\_state.X\_test = X\_test

st.session\_state.y\_test = y\_test

st.session\_state.y\_pred\_rf = y\_pred\_rf

st.session\_state.y\_pred\_xgb = y\_pred\_xgb

st.session\_state.features = features

st.session\_state.model\_trained = True

# Display model performance

st.subheader("Random Forest Performance")

st.write("\*\*Accuracy:\*\*", accuracy\_score(st.session\_state.y\_test, st.session\_state.y\_pred\_rf))

st.text("Classification Report:")

st.text(classification\_report(st.session\_state.y\_test, st.session\_state.y\_pred\_rf,

target\_names=st.session\_state.le.classes\_))

st.subheader("XGBoost Performance")

st.write("\*\*Accuracy:\*\*", accuracy\_score(st.session\_state.y\_test, st.session\_state.y\_pred\_xgb))

st.text("Classification Report:")

st.text(classification\_report(st.session\_state.y\_test, st.session\_state.y\_pred\_xgb,

target\_names=st.session\_state.le.classes\_))

# Confusion Matrix for XGBoost

st.subheader("Confusion Matrix (XGBoost)")

cm = confusion\_matrix(st.session\_state.y\_test, st.session\_state.y\_pred\_xgb)

fig = px.imshow(cm, text\_auto=True, labels=dict(x="Predicted", y="Actual", color="Count"),

x=st.session\_state.le.classes\_, y=st.session\_state.le.classes\_,

title="Confusion Matrix (XGBoost)", color\_continuous\_scale='Blues')

st.plotly\_chart(fig, use\_container\_width=True)

# Feature Importance for XGBoost

st.subheader("Feature Importances (XGBoost)")

importances = st.session\_state.xgb.feature\_importances\_

indices = np.argsort(importances)[::-1]

feature\_names = [st.session\_state.features[i] for i in indices]

fig = px.bar(x=feature\_names, y=importances[indices], title="Feature Importances (XGBoost)",

labels={'x': 'Feature', 'y': 'Importance'})

st.plotly\_chart(fig, use\_container\_width=True)

# Model Accuracy Comparison

st.subheader("Model Accuracy Comparison")

model\_comparison = pd.DataFrame({

'Model': ['Random Forest', 'XGBoost'],

'Accuracy': [accuracy\_score(st.session\_state.y\_test, st.session\_state.y\_pred\_rf),

accuracy\_score(st.session\_state.y\_test, st.session\_state.y\_pred\_xgb)]

})

fig = px.bar(model\_comparison, x='Model', y='Accuracy', title="Model Accuracy Comparison",

color='Model', text='Accuracy', text\_auto='.3f')

fig.update\_layout(yaxis\_range=[0, 1])

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.error("Dataset could not be loaded. Please check the error message in the sidebar.")

# Predict Page

elif page == "Predict":

st.title("🕹️ Predict Air Quality")

if st.session\_state.data is not None and st.session\_state.model\_trained:

st.subheader("Input Parameters for Prediction")

# Create input sliders for features, organized in two columns

col1, col2 = st.columns(2)

with col1:

co\_gt = st.slider("CO(GT) (mg/m³)", min\_value=0.0, max\_value=10.0, value=2.5, step=0.1)

no2\_gt = st.slider("NO2(GT) (µg/m³)", min\_value=0.0, max\_value=300.0, value=100.0, step=1.0)

temperature = st.slider("Temperature (°C)", min\_value=-10.0, max\_value=40.0, value=20.0, step=0.1)

with col2:

humidity = st.slider("Relative Humidity (%)", min\_value=0.0, max\_value=100.0, value=60.0, step=1.0)

hour = st.slider("Hour of Day", min\_value=0, max\_value=23, value=14, step=1)

pollutant\_ratio = co\_gt / no2\_gt if no2\_gt != 0 else 0.0

st.write("\*\*Pollutant Ratio (CO/NO2):\*\*", f"{pollutant\_ratio:.3f}")

# Prepare input data for prediction

input\_data = pd.DataFrame({

'CO(GT)': [co\_gt],

'NO2(GT)': [no2\_gt],

'T': [temperature],

'RH': [humidity],

'Hour': [hour],

'Pollutant\_Ratio': [pollutant\_ratio]

})

# Make predictions when button is clicked

if st.button("Predict Air Quality"):

rf\_pred = st.session\_state.rf.predict(input\_data)

xgb\_pred = st.session\_state.xgb.predict(input\_data)

rf\_quality = st.session\_state.le.inverse\_transform(rf\_pred)[0]

xgb\_quality = st.session\_state.le.inverse\_transform(xgb\_pred)[0]

# Display prediction results

st.subheader("Prediction Results")

st.markdown(f"\*\*Random Forest Prediction:\*\* {rf\_quality}")

st.markdown(f"\*\*XGBoost Prediction:\*\* {xgb\_quality}")

# Display prediction confidence (probabilities)

rf\_probs = st.session\_state.rf.predict\_proba(input\_data)[0]

xgb\_probs = st.session\_state.xgb.predict\_proba(input\_data)[0]

prob\_df = pd.DataFrame({

'Quality Level': st.session\_state.le.classes\_,

'Random Forest Probability': rf\_probs,

'XGBoost Probability': xgb\_probs

})

st.subheader("Prediction Probabilities")

fig = px.bar(prob\_df, x='Quality Level', y=['Random Forest Probability', 'XGBoost Probability'],

barmode='group', title="Prediction Probabilities",

labels={'value': 'Probability', 'variable': 'Model'})

fig.update\_traces(texttemplate='%{y:.2f}', textposition='auto')

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.error("Dataset could not be loaded or models are not trained. Please check the error message in the sidebar.")

**14. Future Scope**

* Dataset Expansion: Incorporate real-time air quality data from IoT sensors or additional cities to enhance generalizability.
* Advanced Models: Experiment with deep learning models (e.g., LSTMs) to capture temporal dependencies in air quality data.
* Explainable AI: Integrate SHAP or LIME to provide transparent explanations of model predictions, increasing trust in environmental applications.
* Deployment: Develop a Gradio or Flask-based web application for real-time air quality predictions.
* Policy Integration: Collaborate with environmental agencies to deploy the model in public health and urban planning initiatives.

**15. Team Members and Roles**

* Veenadharshini K - Responsible for data collection, preprocessing, EDA.
* Vaishnavi T - Responsible for feature engineering, model building.
* Thirushanth B - Responsible for evaluation and visualization.
* Vishal B - Responsible for documentation.