**Student Name:** DHIVYATHARSHINI D

**Register Number:** 712523243008

**Institution:** PPG Institute Of Technology **Department:** B.Tech AI&DS 2nd year **Date of Submission:** 15-05-2025

**Github Repository Link:** [*https://github.com/dharshiniDuraimanickam/NM\_DHIVYATHARSHINI\_DS*](https://github.com/dharshiniDuraimanickam/NM_DHIVYATHARSHINI_DS)

# 

# Problem Statement

Air pollution is a major health and environmental concern, especially in urban areas like Coimbatore. Rising levels of pollutants such as PM2.5, PM10, NO₂, SO₂, CO, and O₃ contribute to respiratory issues and environmental harm. Traditional monitoring methods lack predictive power, limiting timely action. Leveraging advanced machine learning models enables accurate forecasting of air quality, supporting proactive interventions and informed policy decisions to reduce pollution impacts.

# Abstract

This project showcases an interactive **Air Quality Prediction Dashboard** built with **Streamlit**, enabling users to upload, clean, and analyze air quality data. It features a **before-and-after view of data processing**, followed by options for **data exploration**, **regression** to predict CO levels, and **classification** to label air quality as Good, Moderate, or Bad. The dashboard includes visualizations such as **time-series plots**, **correlation heatmaps**, **feature importance**, and **ROC curves**, and evaluates model performance using metrics like **MSE**, **R²**, and **confusion matrices**. It serves as a user-friendly tool for environmental analysis and air quality forecasting.

# System Requirements

**Hardware:**

* RAM: Minimum 4 GB
* CPU: Intel i3/i5 or equivalent

**Software:**

* Python 3.8 or above
* Libraries: pandas, numpy, matplotlib, seaborn, scikit-learn, xgboost, shap, streamlit
* IDE: Jupyter Notebook / Google Colab / VS Code
* Deployment: Streamlit Cloud

# Objectives

* Predict whether a customer will churn based on behavioral and demographic attributes.
* Evaluate multiple machine learning models to find the best performer.
* Interpret model predictions using SHAP values.
* Deploy the entire solution as an interactive Streamlit app.
* Provide actionable insights for customer retention.

# Flowchart of Project Workflow

# ┌──────────────────────────────┐

# │ Load Dataset (CSV) │

# └────────────┬─────────────────┘

# ↓

# ┌──────────────────────────────┐

# │ Data Preprocessing │

# └────────────┬─────────────────┘

# ↓

# ┌──────────────────────────────┐

# │ Exploratory Data Analysis │

# └────────────┬─────────────────┘

# ↓

# ┌──────────────────────────────┐

# │ Feature Engineering │

# └────────────┬─────────────────┘

# ↓

# ┌──────────────────────────────┐

# │ Model Training │

# └────────────┬─────────────────┘

# ↓

# ┌──────────────────────────────┐

# │ Model Evaluation │

# └────────────┬─────────────────┘

# ↓

# ┌──────────────────────────────┐

# │ SHAP Explainability │

# └────────────┬─────────────────┘

# ↓

# ┌──────────────────────────────┐

# │ Streamlit Deployment │

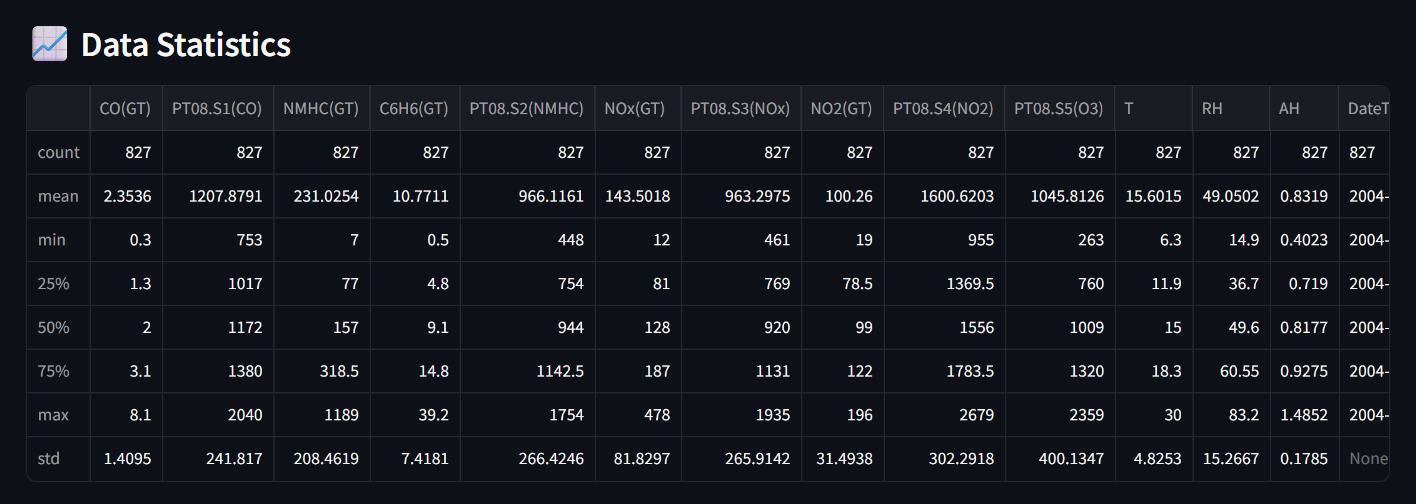
# └──────────────────────────────┘

# Dataset Description

 **Source:** Kaggle (uploaded as churn.csv)

 **Type:** Public dataset

 **Size:** 10,000 rows × 14 columns

**Df.Head() :**

# Data Preprocessing

** Dropped Columns**: RowNumber, CustomerId, Surname

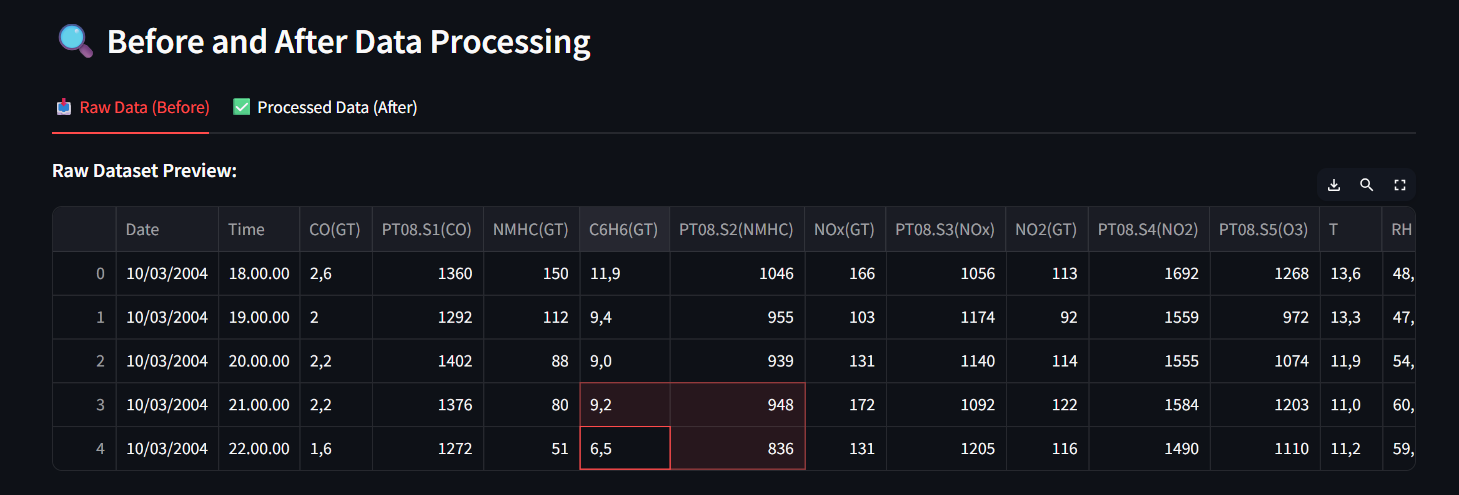
** Encoding:** Label encoding applied to Geography and Gender

** Scaling:** Applied StandardScaler to numerical features

** Missing Values & Duplicates**: None found

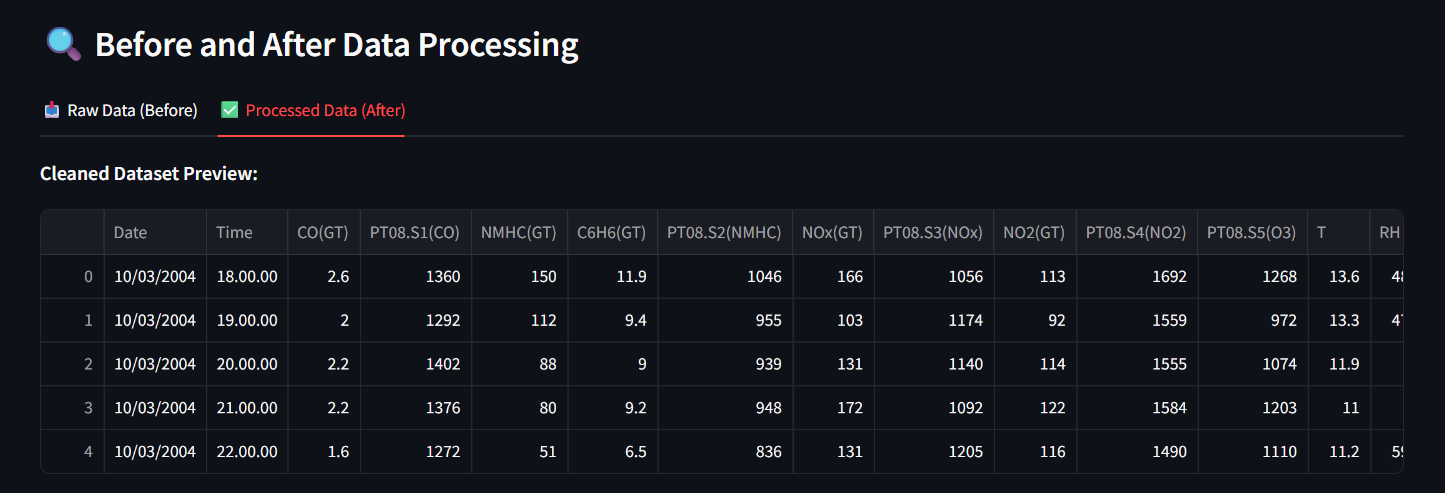
**Before Transformation:**

Raw categorical and unscaled numeric features



**After Transformation:**

All features encoded and standardized, ready for ML models



# Exploratory Data Analysis (EDA)

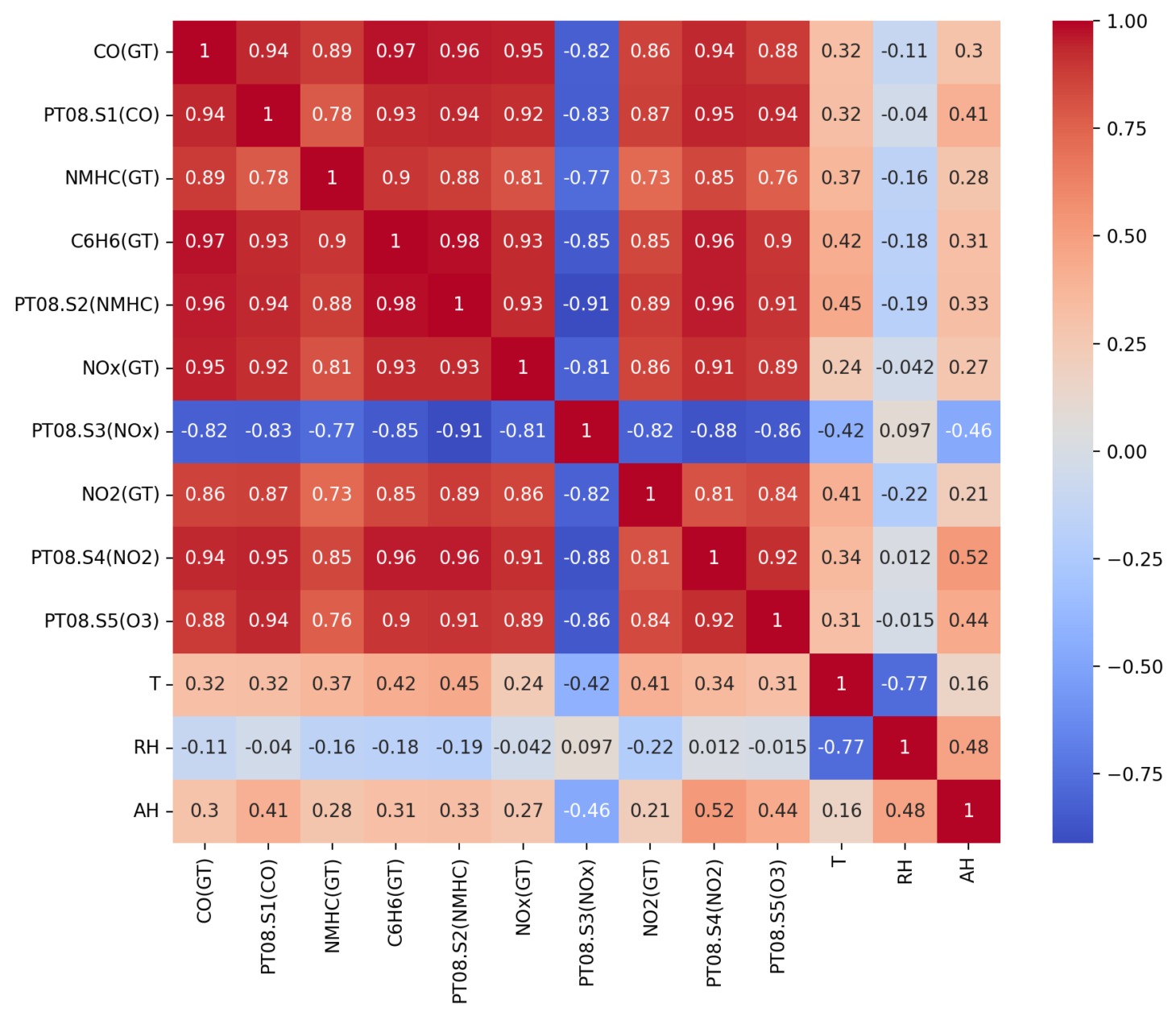
Visualizations Used:

* Correlation heatmap of numerical features
* Line plot of CO(GT) over time
* Histogram of air quality class distribution (in classification mode)

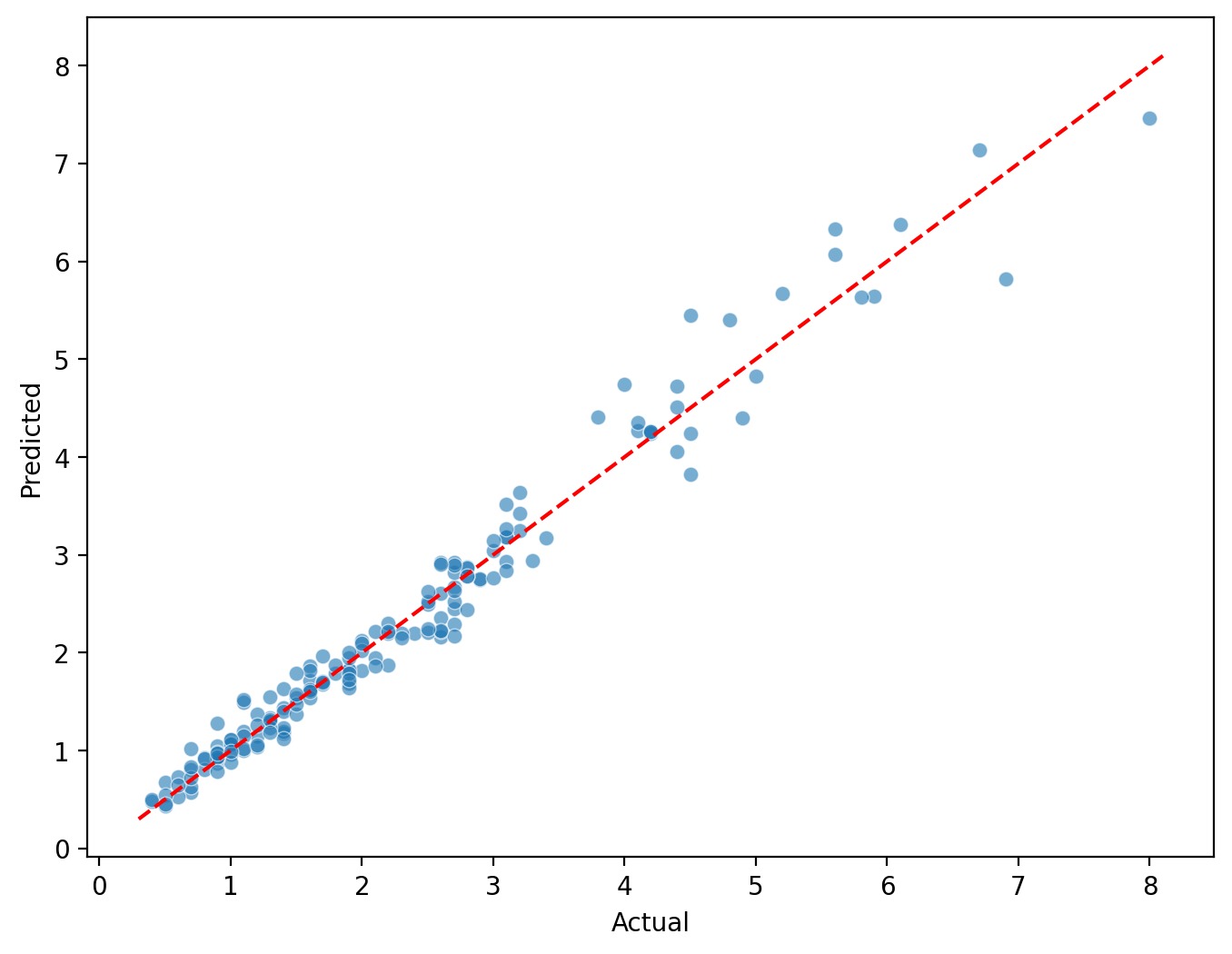
Key Insights:

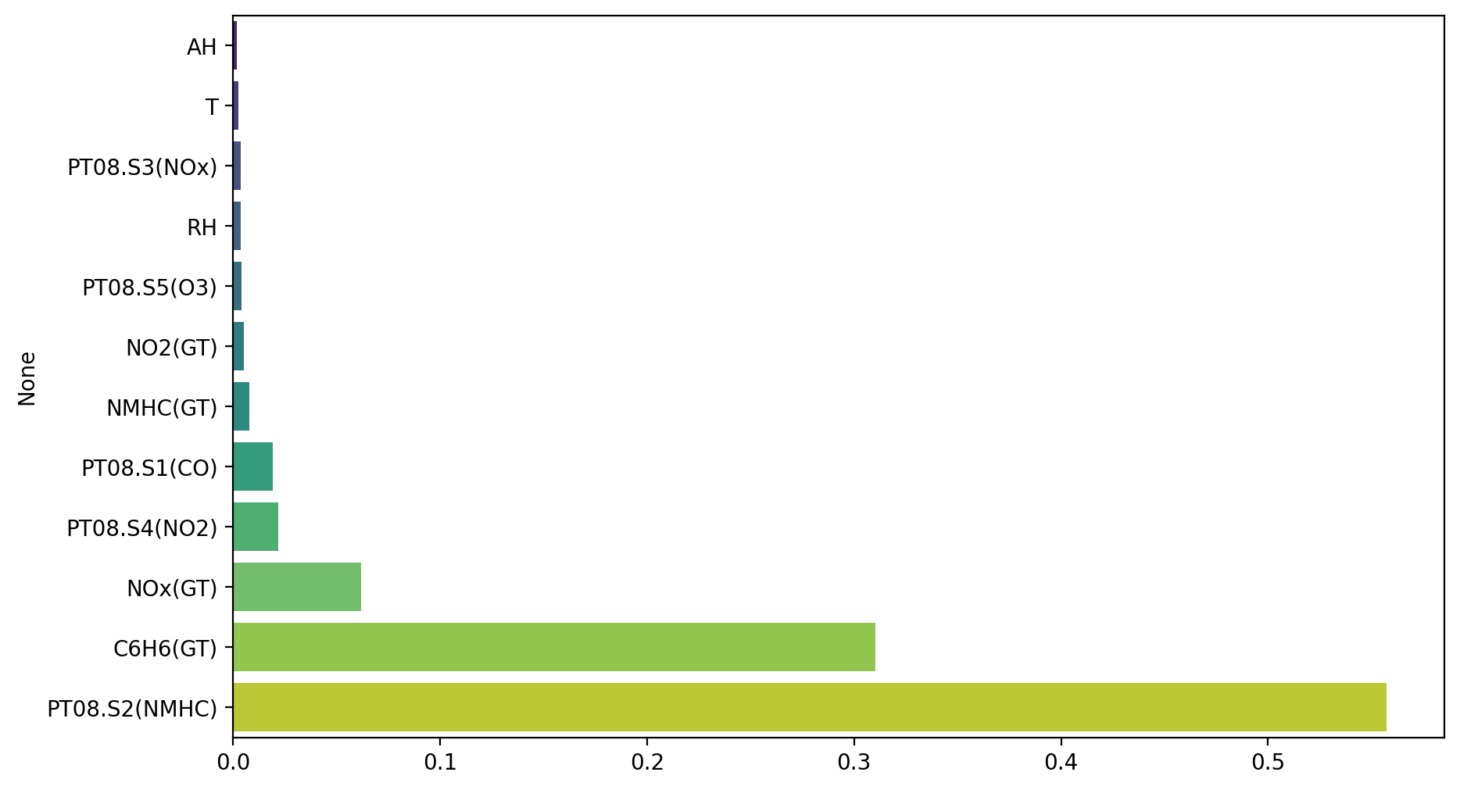
* CO(GT) is strongly correlated with C6H6(GT) and PT08.S1(CO), indicating sensor alignment
* Temperature is negatively correlated with relative humidity and positively with absolute humidity
* Most air quality readings are in the "Moderate" and "Good" categories; fewer instances are labeled "Bad"
* CO levels vary significantly over time, likely due to external factors such as traffic or weather

**Heatmap:**



**Scatter plot of actual vs predicted values:**





# Feature Engineering

* Cleaned missing and invalid values (-200, NaNs)
* Converted numeric strings (with commas) to float
* Combined Date and Time into DateTime
* Labeled air quality based on CO levels (Good, Moderate, Bad)

**Impact:**

* Enhanced data quality and model accuracy for both regression and classification, especially with Random Forest.

# Model Building

**Models Tried:**

* Random Forest Regressor
* Random Forest Classifier

**Training Details:**

* Train/test split = 80/20
* Minimal hyperparameter tuning via Streamlit sliders (n\_estimators, max\_depth)
* Used stratified split for classification to maintain class balance

# Model Evaluation

**Metrics:**

* Accuracy
* F1 Score
* Precision
* Recall
* ROC AUC

**Comparison Table:**

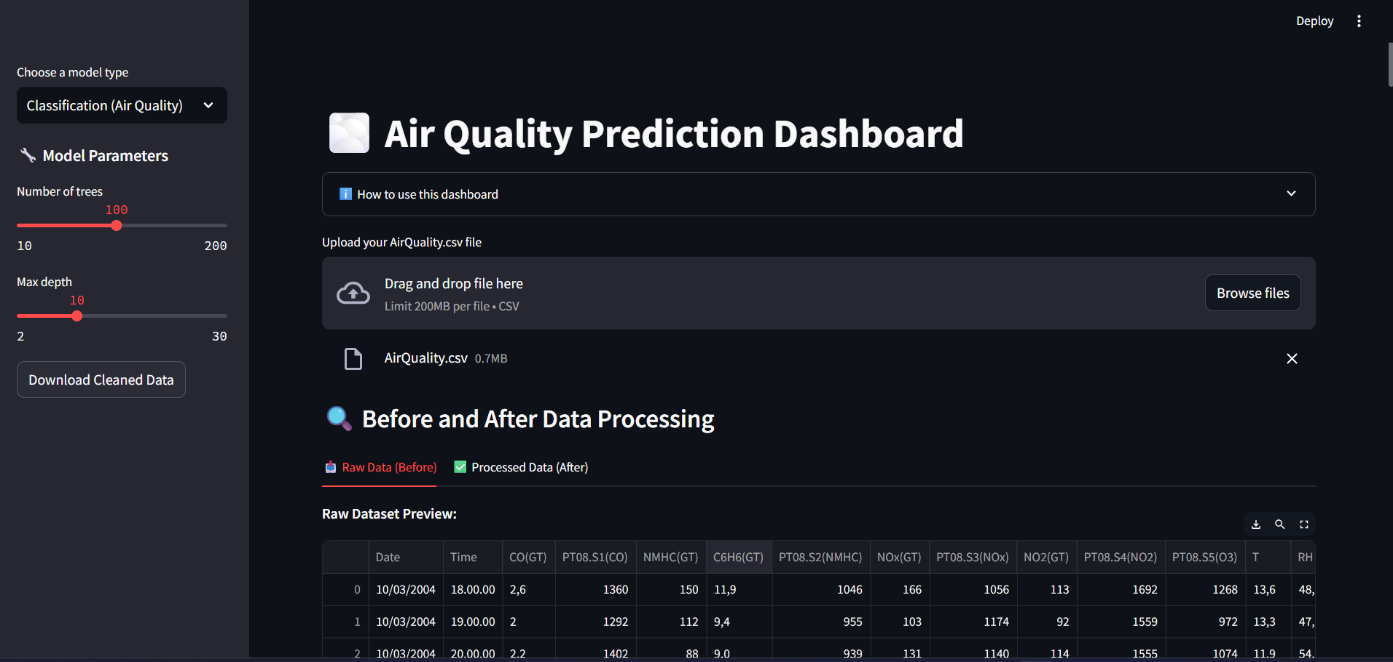


# Deployment

* **Platform:** Streamlit Cloud
* **Method:** Streamlit Python script
* **Link:** Air Quality Prediction Dashboard · Streamlit

** UI Features:**• Dataset Upload  
• Data Cleaning Overview (Before vs After)  
• Model Selection (Regression & Classification)   
• Visualization (Feature Importance, ROC Curve, Confusion Matrix)  
• Download Cleaned Data

**UI SCREENSHOT:**



**PREDICTION OUTPUT** :



# Source code

import streamlit as st

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.metrics import (

    mean\_squared\_error, r2\_score,

    classification\_report, confusion\_matrix,

    roc\_auc\_score, roc\_curve, auc

)

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import label\_binarize

# Page configuration

st.set\_page\_config(

    page\_title="🌫️ Air Quality Prediction Dashboard (Advanced)",

    page\_icon="🌍",

    layout="wide",

    initial\_sidebar\_state="expanded"

)

# Custom CSS

st.markdown("""

    <style>

    .main { background-color: #f8f9fa; }

    .stButton>button { background-color: #4CAF50; color: white; }

    .stSelectbox>div>div>select { background-color: #e9f7ef; }

    .css-1aumxhk { background-color: #e9f7ef; }

    </style>

    """, unsafe\_allow\_html=True)

@st.cache\_data(show\_spinner=True)

def load\_data(uploaded\_file):

    try:

        df\_raw = pd.read\_csv(uploaded\_file, sep=';', low\_memory=False)

        df = df\_raw.copy()

        df.drop(columns=[col for col in ["Unnamed: 15", "Unnamed: 16"] if col in df.columns], inplace=True)

        for col in df.columns:

            if col not in ["Date", "Time"]:

                df[col] = df[col].astype(str).str.replace(',', '.', regex=False)

                df[col] = pd.to\_numeric(df[col], errors='coerce')

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

        df.dropna(inplace=True)

        try:

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

        except ValueError:

            try:

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

            except ValueError:

                st.warning("Automatic datetime parsing used")

                df['DateTime'] = pd.to\_datetime(df['Date'] + ' ' + df['Time'], errors='coerce')

                df.dropna(subset=['DateTime'], inplace=True)

        return df\_raw, df

    except Exception as e:

        st.error(f"Error loading data: {str(e)}")

        return None, None

def model\_hyperparameters():

    st.sidebar.subheader("🔧 Model Parameters")

    n\_estimators = st.sidebar.slider("Number of trees", 10, 200, 100, 10)

    max\_depth = st.sidebar.slider("Max depth", 2, 30, 10, 1)

    return n\_estimators, max\_depth

def get\_feature\_target(df, target\_col):

    X = df.drop(columns=["Date", "Time", "DateTime", target\_col], errors='ignore')

    y = df[target\_col]

    return X, y

def show\_data\_summary(df):

    st.subheader("📊 Data Overview")

    col1, col2 = st.columns(2)

    with col1:

        st.write("\*\*Dataset Shape:\*\*", df.shape)

        st.write("\*\*Columns:\*\*", list(df.columns))

    with col2:

        st.write("\*\*Missing Values:\*\*")

        st.write(df.isnull().sum())

    st.subheader("📈 Data Statistics")

    st.write(df.describe())

    st.subheader("⏳ CO(GT) Over Time")

    fig, ax = plt.subplots(figsize=(12, 4))

    df.set\_index('DateTime')['CO(GT)'].plot(ax=ax)

    plt.title("CO Levels Over Time")

    st.pyplot(fig)

    st.subheader("🔥 Correlation Heatmap")

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

    fig, ax = plt.subplots(figsize=(10, 8))

    sns.heatmap(df[numeric\_cols].corr(), annot=True, cmap='coolwarm', ax=ax)

    st.pyplot(fig)

def regression\_model(df):

    st.subheader("🔢 Regression Model: Predict CO(GT)")

    n\_estimators, max\_depth = model\_hyperparameters()

    X, y = get\_feature\_target(df, "CO(GT)")

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    model = RandomForestRegressor(n\_estimators=n\_estimators, max\_depth=max\_depth, random\_state=42)

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    mse = mean\_squared\_error(y\_test, y\_pred)

    r2 = r2\_score(y\_test, y\_pred)

    col1, col2 = st.columns(2)

    with col1:

        st.metric("Mean Squared Error", f"{mse:.4f}")

    with col2:

        st.metric("R² Score", f"{r2:.4f}")

    st.subheader("📊 Feature Importance")

    importance = pd.Series(model.feature\_importances\_, index=X.columns).sort\_values()

    fig, ax = plt.subplots(figsize=(10, 6))

    sns.barplot(x=importance.values, y=importance.index, palette="viridis", ax=ax)

    st.pyplot(fig)

    st.subheader("🔄 Actual vs Predicted Values")

    fig, ax = plt.subplots(figsize=(8, 6))

    sns.scatterplot(x=y\_test, y=y\_pred, alpha=0.6, ax=ax)

    plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')

    plt.xlabel("Actual")

    plt.ylabel("Predicted")

    st.pyplot(fig)

def plot\_roc\_auc(y\_test, y\_proba, class\_labels):

    y\_bin = label\_binarize(y\_test, classes=class\_labels)

    fpr, tpr, roc\_auc = {}, {}, {}

    for i in range(len(class\_labels)):

        fpr[i], tpr[i], \_ = roc\_curve(y\_bin[:, i], y\_proba[:, i])

        roc\_auc[i] = auc(fpr[i], tpr[i])

    fig, ax = plt.subplots(figsize=(8, 6))

    for i, label in enumerate(class\_labels):

        ax.plot(fpr[i], tpr[i], label=f'{label} (AUC = {roc\_auc[i]:.2f})')

    ax.plot([0, 1], [0, 1], 'k--')

    ax.set\_xlabel('False Positive Rate')

    ax.set\_ylabel('True Positive Rate')

    ax.set\_title('ROC Curves')

    ax.legend()

    st.pyplot(fig)

def classification\_model(df):

    st.subheader("🏷️ Classification Model: Air Quality Labels")

    n\_estimators, max\_depth = model\_hyperparameters()

    def label\_air\_quality(co):

        if co <= 2:

            return 'Good'

        elif co <= 5:

            return 'Moderate'

        else:

            return 'Bad'

    df['AirQuality'] = df['CO(GT)'].apply(label\_air\_quality)

    st.write("\*\*Class Distribution:\*\*")

    dist = df['AirQuality'].value\_counts(normalize=True)

    st.write(dist)

    fig, ax = plt.subplots(figsize=(6, 4))

    dist.plot(kind='bar', color=['green', 'orange', 'red'], ax=ax)

    plt.title("Air Quality Class Distribution")

    st.pyplot(fig)

    X, y = get\_feature\_target(df, "AirQuality")

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42)

    model = RandomForestClassifier(n\_estimators=n\_estimators, max\_depth=max\_depth, random\_state=42, class\_weight='balanced')

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    y\_proba = model.predict\_proba(X\_test)

    st.subheader("📝 Classification Report")

    st.dataframe(pd.DataFrame(classification\_report(y\_test, y\_pred, output\_dict=True)).transpose())

    st.subheader("🔄 Confusion Matrix")

    cm = confusion\_matrix(y\_test, y\_pred, labels=['Good', 'Moderate', 'Bad'])

    fig, ax = plt.subplots(figsize=(8, 6))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Good', 'Moderate', 'Bad'], yticklabels=['Good', 'Moderate', 'Bad'], ax=ax)

    ax.set\_xlabel("Predicted")

    ax.set\_ylabel("Actual")

    st.pyplot(fig)

    st.subheader("📈 ROC Curves and AUC")

    plot\_roc\_auc(y\_test, y\_proba, ['Good', 'Moderate', 'Bad'])

def main():

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

    with st.expander("ℹ️ How to use this dashboard"):

        st.write("""

        1. Upload your AirQuality.csv file (semicolon-separated)

        2. View raw vs processed data

        3. Select the model type

        4. Adjust model parameters

        5. View the results and download cleaned data

        """)

        sample = """Date;Time;CO(GT);PT08.S1(CO);NMHC(GT);C6H6(GT);PT08.S2(NMHC);NOx(GT);PT08.S3(NOx);NO2(GT);PT08.S4(NO2);PT08.S5(O3);T;RH;AH\n01/01/2004;12.00.00;2,6;1360;150;11,9;1046;166;1056;113;1692;1268;13,6;48,9;0,7578"""

        st.download\_button("Download sample data format", data=sample.encode('utf-8'), file\_name="AirQuality\_sample.csv", mime="text/csv")

    uploaded\_file = st.file\_uploader("Upload your AirQuality.csv file", type=["csv"])

    if uploaded\_file:

        df\_raw, df = load\_data(uploaded\_file)

        if df is not None:

            st.subheader("🔍 Before and After Data Processing")

            tab1, tab2 = st.tabs(["📥 Raw Data (Before)", "✅ Processed Data (After)"])

            with tab1:

                st.write("\*\*Raw Dataset Preview:\*\*")

                st.dataframe(df\_raw.head())

                st.write("Shape:", df\_raw.shape)

                st.write("Missing Values:")

                st.write(df\_raw.isnull().sum())

            with tab2:

                st.write("\*\*Cleaned Dataset Preview:\*\*")

                st.dataframe(df.head())

                st.write("Shape:", df.shape)

                st.write("Missing Values:")

                st.write(df.isnull().sum())

            mode = st.sidebar.selectbox("Choose a model type", ["Data Exploration", "Regression (Predict CO)", "Classification (Air Quality)"])

            if mode == "Data Exploration":

                show\_data\_summary(df)

            elif mode == "Regression (Predict CO)":

                regression\_model(df)

            elif mode == "Classification (Air Quality)":

                classification\_model(df)

            csv = df.to\_csv(index=False, sep=';').encode('utf-8')

            st.sidebar.download\_button("Download Cleaned Data", data=csv, file\_name='cleaned\_air\_quality.csv', mime='text/csv')

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# Future scope

 Integrate live air quality data via APIs (e.g., OpenAQ, WAQI) for real-time prediction

 Automate hyperparameter tuning using GridSearchCV or Optuna for improved performance

 Add user feedback functionality to refine model predictions over time

 Implement model drift detection and schedule periodic retraining for sustained accuracy.

# 15. Team Members and Roles

|  |  |
| --- | --- |
| Name | Contributions |
| [Banumathi.V] | Team Leader; coordinated the project workflow, managed deadlines, performed data cleaning and preprocessing, and led exploratory data analysis (EDA). |
| [ Sakina. E] | Conducted in-depth exploratory data analysis, created visualizations to identify trends and patterns, and helped interpret insights to guide modeling. |
| [Divya Dharshini.D] | Responsible for feature engineering, including feature creation, transformation, and selection to enhance model accuracy and efficiency. |
| [Yogadharshini.R] | Carried out model evaluation by training multiple algorithms, comparing their performance, tuning hyperparameters, and selecting the best model. |
| [Kamalakaviya. J ] | Led deployment efforts by developing and deploying the web application using Streamlit, ensuring a smooth user interface and real-time predictions. |