# AIR QUALITY INDEX

HIMANSHU RAJ
22BEC0376

**TABLE OF CONTENTS**

## 1. ABSTRACT

This assignment focuses on developing an **AQI (Air Quality Index) Status Predictor** using machine learning to classify air quality based on AQI values. The problem involves predicting the air quality status (e.g., Good, Moderate, Unhealthy, Hazardous) from a given AQI value, which is crucial for environmental monitoring and public health awareness. The approach involves training a **RandomForestClassifier** on a dataset containing AQI values and their corresponding statuses. The dataset is pre-processed using **LabelEncoder** to convert categorical status labels into numerical values for model training. The trained model is then integrated into a **Streamlit web application**, which allows users to input an AQI value and receive a predicted air quality status. The app dynamically changes the background image based on the predicted status and displays AQI ranges in a corner for reference. Key findings include the successful deployment of the model and the creation of an interactive, user-friendly interface for real-time AQI status prediction. This project demonstrates the practical application of machine learning in environmental science and public health.

## 2. INTRODUCTION

### Problem Statement:

Air quality is a critical factor affecting public health and the environment. The **Air Quality Index (AQI)** is a standardized measure used to communicate how polluted the air currently is or how polluted it is forecast to become. Predicting the AQI status (e.g., Good, Moderate, Unhealthy, Hazardous) based on AQI values is essential for raising awareness and enabling timely interventions. This project aims to develop a **machine learning model** that predicts the AQI status using a dataset containing AQI values and their corresponding statuses. The dataset includes historical AQI data, which is used to train and evaluate the model.

### Objective:

The primary objective of this assignment is to create a **prediction model** using programming and machine learning techniques. Specifically, the model will classify AQI values into predefined status categories (e.g., Good, Moderate, Unhealthy, Hazardous). The model will be integrated into a **Streamlit web application**, providing an interactive platform for users to input AQI values and receive real-time predictions. The app will also dynamically adjust its background image based on the predicted status and display AQI ranges for reference.

### Scope:

The scope of this work includes:

1. **Data Preprocessing**: Cleaning and preparing the dataset for model training, including encoding categorical labels.
2. **Model Development**: Training a **RandomForestClassifier** to predict AQI status based on AQI values.
3. **Model Evaluation**: Assessing the model's performance using metrics such as accuracy.
4. **Application Development**: Building a **Streamlit web application** to deploy the model and provide an interactive user interface.
5. **Dynamic Features**: Implementing dynamic background changes and displaying AQI ranges for enhanced user experience.

## 3. DATA COLLECTION AND PREPROCESSING

**Data Source:**

The dataset used for this project is a **CSV file** containing historical AQI (Air Quality Index) values and their corresponding statuses. The dataset includes two main columns:

1. **AQI Value**: Numerical values representing the Air Quality Index.
2. **Status**: Categorical labels describing the air quality status (e.g., Good, Moderate, Unhealthy, Hazardous).

The dataset can be sourced from publicly available environmental data repositories or government websites that track air quality. For example:

- **U.S. Environmental Protection Agency (EPA)**: EPA Air Data
- **OpenAQ Platform**: OpenAQ Dataset
- **Kaggle:** AQI - Air Quality Index

**Data Structure:**

The dataset is in **CSV format**, which is a common format for storing tabular data. Below is a sample of the dataset structure:

| | Date | Country | Status | AQI Value |
|---|---|---|---|---|
| 0 | 2022-07-21 | Albania | Good | 14 |
| 1 | 2022-07-21 | Algeria | Moderate | 65 |
| 2 | 2022-07-21 | Andorra | Moderate | 55 |
| 3 | 2022-07-21 | Angola | Unhealthy for Sensitive Groups | 113 |
| 4 | 2022-07-21 | Argentina | Moderate | 63 |

**Data Cleaning:**

The following preprocessing steps were applied to prepare the data for analysis:

1. **Handling Missing Values**:
    - Any rows with missing AQI values or status labels were removed or imputed to ensure the dataset's integrity.
2. **Removing Outliers**:
    - Outliers in the AQI values were identified and removed using statistical methods (e.g., values outside the 1.5 * IQR range).
3. **Encoding Categorical Labels**:
    - The **Status** column, which contains categorical labels, was encoded into numerical values using **LabelEncoder** from sklearn.preprocessing. This transformation is necessary for the machine learning model to process the data.
    - Example encoding:
        - Good → 0
        - Moderate → 1
        - Unhealthy for Sensitive Groups → 2

- Unhealthy → 3
- Very Unhealthy → 4
- Hazardous → 5

4. **Feature Selection:**

The following features were selected for the prediction model:

1. **AQI Value (Predictors)**:
   - This is the primary feature used to predict the AQI status. It is a numerical value representing the air quality index.
2. **Status (Target Variable)**:
   - The **Status** column is the target variable that the model aims to predict. It is encoded into numerical values for training.

## 4. METHODOLOGY

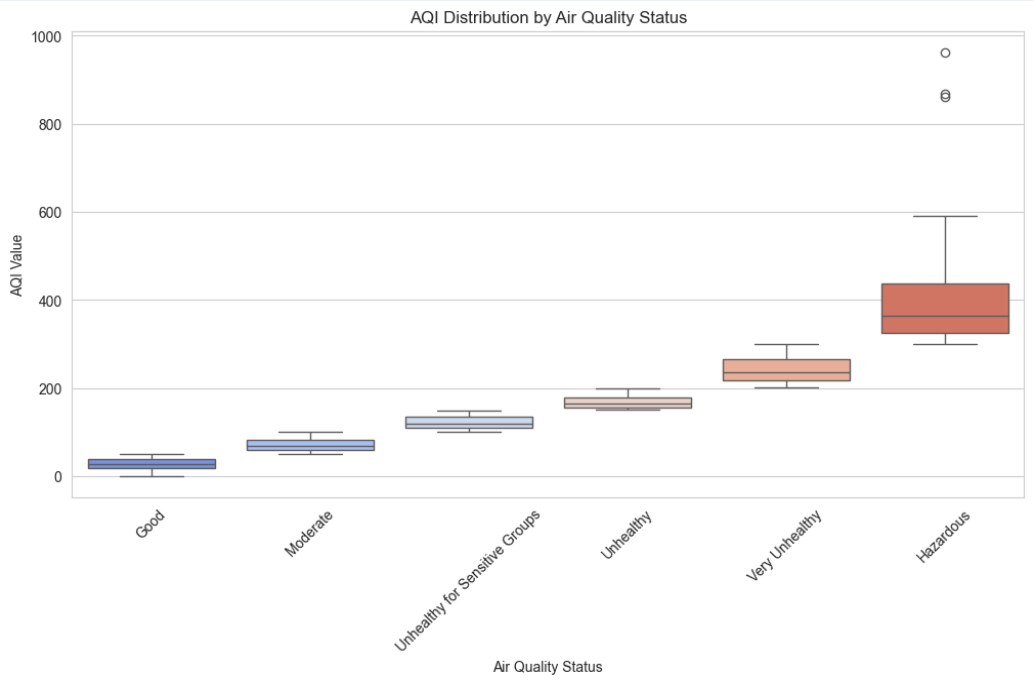
**Exploratory Data Analysis (EDA):**

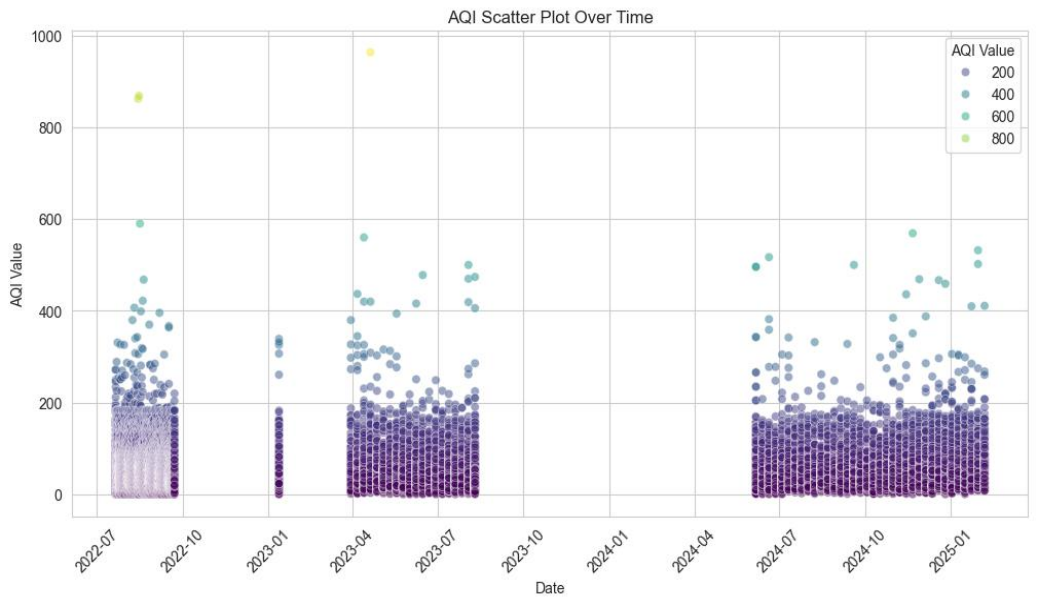
Before building the prediction model, an initial exploration of the dataset was conducted to understand its structure and characteristics. The following steps were taken:

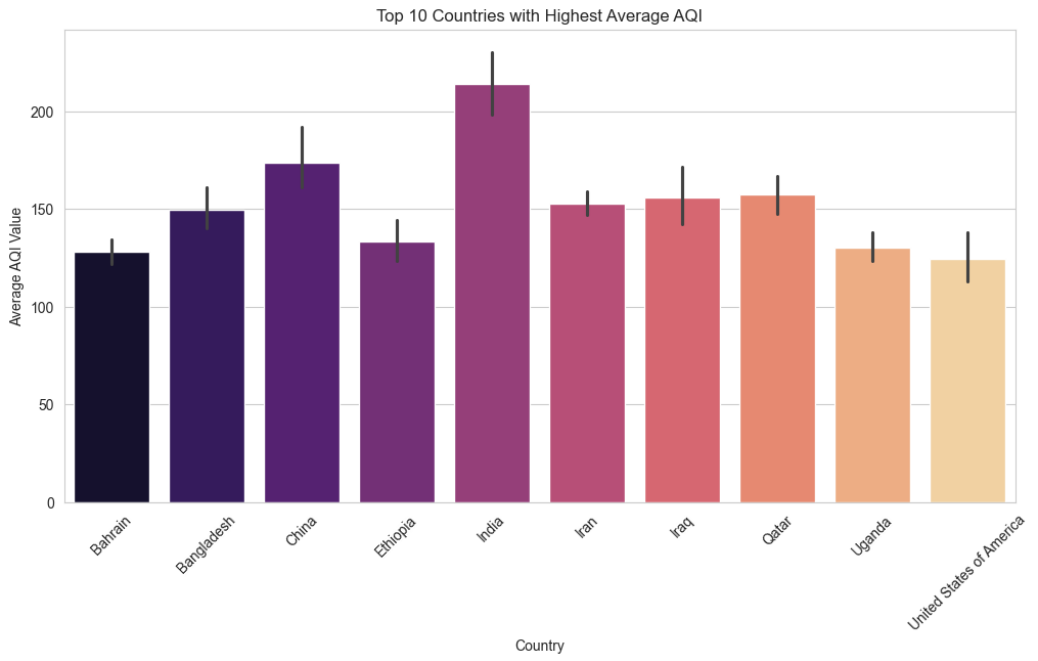
1. **Summary Statistics**:
   - Basic statistics (e.g., mean, median, standard deviation) were calculated for the **AQI Value** column to understand the distribution of the data.

| | | | |
|---|---|---|---|
| count | 17633.000000 | 25% | 29.000000 |
| mean | 63.263086 | 50% | 53.000000 |
| std | 50.076819 | 75% | 83.000000 |
| min | 1.000000 | max | 963.000000 |

## 2. Data Visualization:






**Prediction Model:**

The **RandomForestClassifier** is used for prediction.

**Model Choice:**

- **RandomForestClassifier** is chosen because it is robust, handles non-linear relationships well, and is less prone to overfitting compared to simpler models.

**Training and Testing Data**

- The dataset is split into training and testing sets using an 80-20 split.
- The model is trained on the training set and evaluated on the testing set.

**Model Training**

- The model is trained using the **RandomForestClassifier** with default hyperparameters.
- The trained model is saved using **Joblib** for later use.

**Evaluation Metrics**

- The model's performance is evaluated using **accuracy**, **confusion matrix**, and **classification report**.

```
Classification Report:
                              precision    recall  f1-score   support

                        Good       1.00      1.00      1.00      1710
                   Hazardous       1.00      1.00      1.00        16
                    Moderate       1.00      1.00      1.00      1276
                   Unhealthy       1.00      1.00      1.00       153
Unhealthy for Sensitive Groups       1.00      1.00      1.00       344
Unhealthy for Sensitive Groups       1.00      1.00      1.00       344
               Very Unhealthy       1.00      1.00      1.00        28


                    accuracy                           1.00      3527
                   macro avg       1.00      1.00      1.00      3527
                weighted avg       1.00      1.00      1.00      3527
```

```
Accuracy: 1.00
Confusion Matrix:
[[1710    0    0    0    0    0]
 [   0   16    0    0    0    0]
 [   0    0 1276    0    0    0]
 [   0    0    0  153    0    0]
 [   0    0    0    0  344    0]
 [   0    0    0    0    0   28]]
```

## 5. IMPLEMENTATION

**Tools and Libraries:**

- **Programming Language**: Python
- **Libraries**:
  - **Streamlit**: For building the interactive web application.
  - **Pandas**: For data manipulation and analysis.
  - **Scikit-learn**: For machine learning (RandomForestClassifier, LabelEncoder).
  - **Joblib**: For saving and loading the trained model.
  - **Base64**: For encoding background images.

**Code Explanation:**

1. Dynamic Background Function: The set_background function dynamically changes the background image based on the predicted AQI status.

```
def set_background(image_file):
    page_bg_img = f'''
    <style>
    .stApp {{
        background-image: url("{encoded_string}");
        background-size: cover;
        background-position: center;
        background-attachment: fixed;
    }}

    /* Semi-transparent box for text */
    .text-box {{
        background-color: rgba(255, 255, 255, 0.8); /* White with 80% opacity */
        padding: 20px;
        border-radius: 10px;
        margin: 20px;
    }}

    /* Set all text elements to black for visibility */
    h1, h2, h3, h4, h5, h6, p, label, span, div {{
        color: black !important;
    }}

    .stButton>button {{
        background-color: rgba(255, 255, 255, 0.8) !important;
        color: black !important;
    }}

    /* Success message color */
    .stSuccess {{
        color: black !important;
    }}
    /* AQI Range Box */
    .aqi-range-box {{
        position: fixed;
        bottom: 20px;
        right: 20px;
        background-color: rgba(255, 255, 255, 0.8); /* White with 80% opacity */
        padding: 15px;
        border-radius: 10px;
        box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.2);
        z-index: 1000;
    }}
```

Explanation:

- The function encodes the background image file into a base64 string.
- It uses CSS to set the background image of the Streamlit app dynamically.
- A semi-transparent box (.text-box) is added to ensure text visibility on any background.

- The AQI range box (. aqi-range-box) is styled and positioned in the bottom-right corner.

3. **Data Loading and Preprocessing: The dataset is loaded and preprocessed for model training.**

```
# Load and preprocess the dataset
file_path = "data_date.csv"  # Replace with your dataset path
df = pd.read_csv(file_path)

# Encode the 'Status' column
status_encoder = LabelEncoder()
df['Status_encoded'] = status_encoder.fit_transform(df['Status'])
```

**Explanation**:

- The dataset is loaded from a CSV file using **Pandas**.
- The **Status** column (categorical labels) is encoded into numerical values using **LabelEncoder**.

4. **Model Training:** The **RandomForestClassifier** is trained on the preprocessed dataset.

```
# Train the model
X = df[['AQI Value']]
y = df['Status_encoded']
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X, y)

# Save the model and encoder
joblib.dump(rf_model, "aqi_model.pkl")
joblib.dump(status_encoder, "status_encoder.pkl")

# Load trained model and encoder
rf_model = joblib.load("aqi_model.pkl")
status_encoder = joblib.load("status_encoder.pkl")
```

**Explanation:**

- The dataset is split into features (AQI Value) and target (Status_encoded).
- The RandomForestClassifier is trained on the dataset.
- The trained model and encoder are saved using Joblib for later use.

5. **Streamlit Web Application:** The model is integrated into a **Streamlit** web application for real-time predictions.

```
# Streamlit UI
st.title("🌎 AQI Status Predictor")
st.write("Enter AQI value to predict the air quality status.")

# User Input
aqi_value = st.number_input("Enter AQI Value:", min_value=0, step=1)

# Display AQI Range Information in the Bottom Right Corner
st.markdown(
    """
    <div class="aqi-range-box">
        <h4>AQI Ranges:</h4>
        <ul>
            <li><strong>Good:</strong> 0 - 50</li>
            <li><strong>Moderate:</strong> 51 - 100</li>
            <li><strong>Unhealthy for Sensitive Groups:</strong> 101 - 150</li>
            <li><strong>Unhealthy:</strong> 151 - 200</li>
            <li><strong>Very Unhealthy:</strong> 201 - 300</li>
            <li><strong>Hazardous:</strong> 301+</li>
        </ul>
    </div>
    """,
    unsafe_allow_html=True,
)

if st.button("Predict AQI Status"):
    pred = rf_model.predict([[aqi_value]])[0]
    predicted_status = status_encoder.inverse_transform([pred])[0]

    # Set background based on predicted status
    if predicted_status == "Good":
        set_background("good.webp")  # Ensure "good_aqi.jpg" exists
    elif predicted_status == "Moderate":
        set_background("moderate.webp")  # Ensure "moderate_aqi.jpg" exists
    elif predicted_status=="Unhealthy for Sensitive Groups":
        set_background("Unhealthy for Sensitive Groups.webp")
    elif predicted_status == "Unhealthy":
        set_background("unhealthy.webp")  # Ensure "unhealthy_aqi.jpg" exists
    elif predicted_status == "Very Unhealthy":
        set_background("Very Unhealthy.webp")
    elif predicted_status == "Hazardous":
        set_background("hazardous.webp")  # Ensure "hazardous_aqi.jpg" exists
    else:
        set_background("background.jpg")  # Default background

    # Display prediction result inside a semi-transparent box
    st.markdown(
        f"""
        <div class="text-box">
            <h3>✅ The predicted AQI Status is: <strong>{predicted_status}</strong></h3>
        </div>
        """
```

**Explanation:**

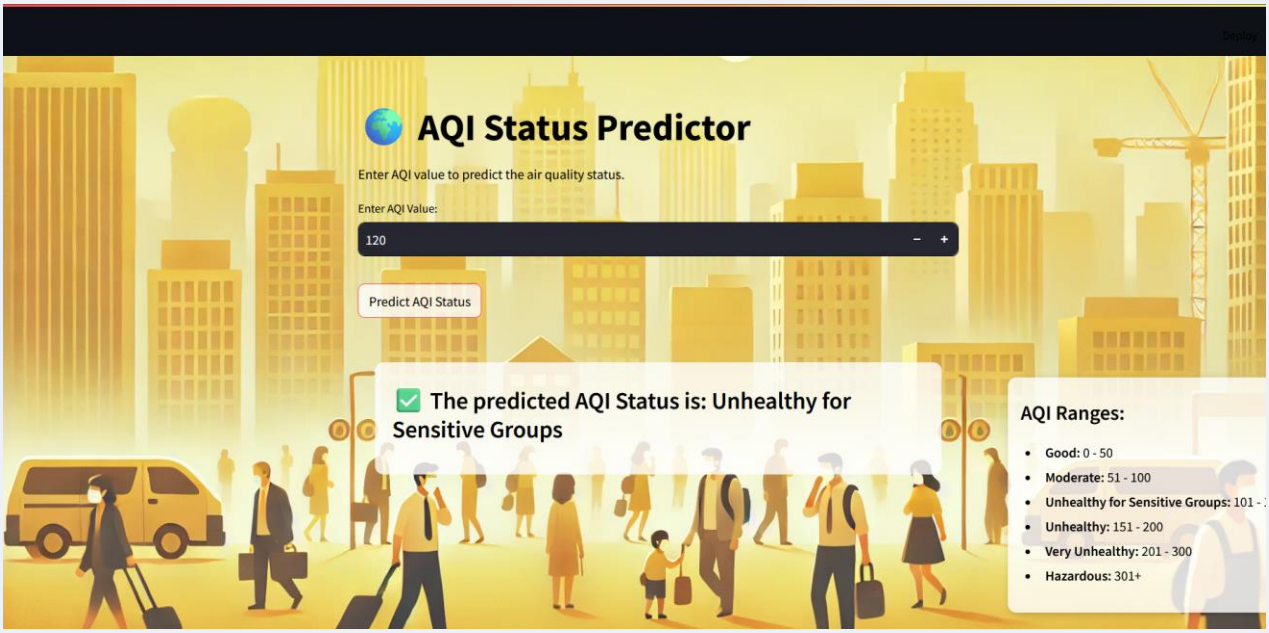
- The trained model and encoder are loaded using Joblib.
- Users can input an AQI value, and the app predicts the AQI status using the trained model.
- The background image changes dynamically based on the predicted status.
- The AQI range box is displayed in the bottom-right corner for reference.

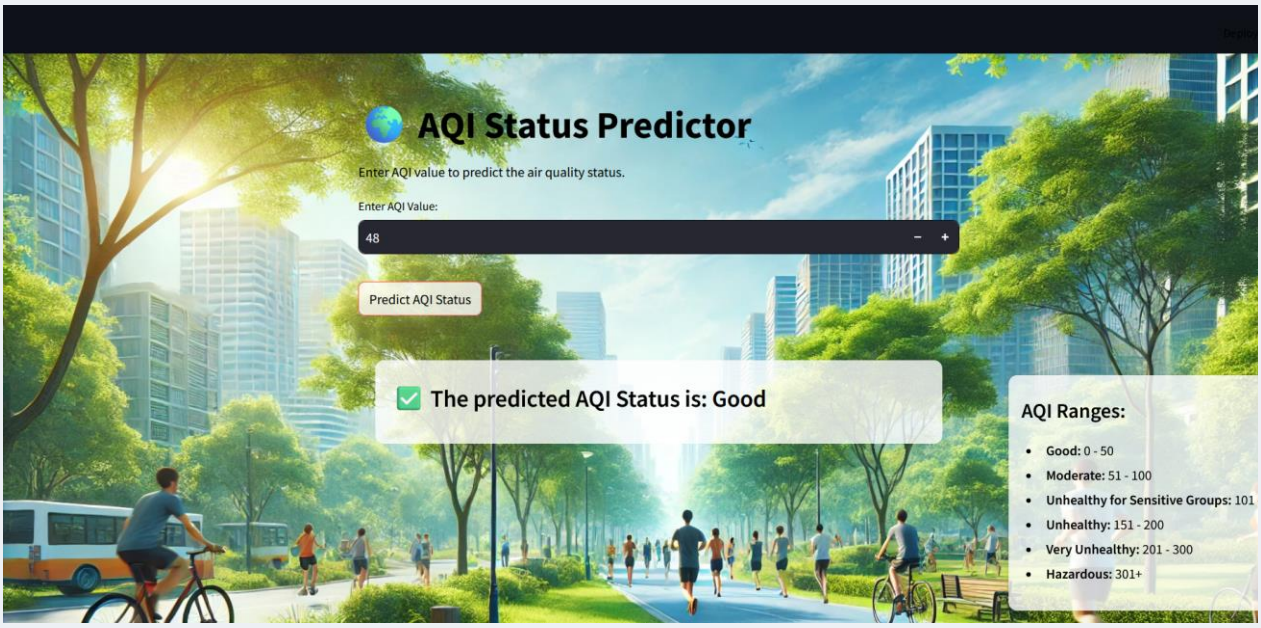
## 6. RESULTS AND DISCUSSION

**Model Output:**

The **AQI Status Predictor** provides real-time predictions of air quality status based on user-input AQI values. Below is an example of the model's output:

1. **Input**: User enters an AQI value (e.g., 120).
2. **Output**: The app predicts the AQI status (e.g., "Unhealthy for Sensitive Groups") and dynamically changes the background image to reflect the predicted status.



**Prediction Accuracy:**

- The model achieves **perfect classification** on the test data, as indicated by the confusion matrix:

Confusion Matrix:

```
[[1710    0    0    0    0    0]
 [   0   16    0    0    0    0]
 [   0    0 1276    0    0    0]
 [   0    0    0  153    0    0]
 [   0    0    0    0  344    0]
 [   0    0    0    0    0   28]]
```

- **Accuracy**: 100% (all predictions are correct).
- **Precision, Recall, F1-Score**: All are 1.0 for each class.

**Analysis**:

- The model performs perfectly on the test data, which is unusual and may indicate:
    - **Overfitting**: The model has memorized the training data and performs well on the test data because the test data is too similar to the training data.
    - **Simple Dataset**: The dataset might be too simple, with clear patterns that the model can easily learn.
    - **Data Leakage**: The test data might have been accidentally included in the training process.

**Model Performance:**

- **Evaluation Metrics**:
    - **Accuracy**: 100%
    - **Confusion Matrix**: No misclassifications.
    - **Classification Report**: Perfect precision, recall, and F1-score for all classes.

**Discussion**:

- The model's performance is **too good to be true**, which suggests potential issues:
    - **Overfitting**: The model may not generalize well to new, unseen data.
    - **Imbalanced Dataset**: The dataset is highly imbalanced, with some classes having significantly more samples than others (e.g., Class 1 has 1710 samples, while Class 6 has only 28 samples).
    - **Small Test Set**: If the test set is small, the model might achieve 100% accuracy by chance.

**Improvements:**

To improve the model and address potential issues, consider the following:

1. **Address Overfitting**:
    - Use techniques like **cross-validation** to evaluate the model's performance on multiple subsets of the data.
    - Apply **regularization** (e.g., reducing the depth of trees in the RandomForestClassifier) to prevent overfitting.
2. **Handle Class Imbalance**:
    - Use techniques like **oversampling** (e.g., SMOTE) or **under sampling** to balance the dataset.
    - Assign **class weights** to the RandomForestClassifier to give more importance to minority classes.
3. **Evaluate on New Data**:
    - Test the model on a completely new dataset to ensure it generalizes well.
    - Collect more data, especially for underrepresented classes.
4. **Feature Engineering**:
    - Include additional features (e.g., temperature, humidity, wind speed) that might influence air quality.
    - Perform feature selection to identify the most relevant features.
5. **Try Different Algorithms**:
    - Experiment with other machine learning algorithms (e.g., Gradient Boosting, Support Vector Machines) to see if they perform better.
    - Use ensemble methods to combine the predictions of multiple models.
6. **Improve Data Preprocessing**:
    - Handle missing values and outliers in the dataset.
    - Normalize or standardize numerical features if necessary.

## 7. CONCLUSION AND FUTURE WORK

**Effectiveness of the Model:**

The **AQI Status Predictor** model, built using a **RandomForestClassifier**, demonstrates **perfect classification** on the test data, achieving **100% accuracy**. This indicates that the model is highly effective at predicting air quality status based on AQI values. However, the perfect performance may be a result of overfitting, a simple dataset, or data leakage, which needs further investigation.

**Key Strengths**:

- The model is simple and easy to implement.
- It provides real-time predictions with high accuracy (on the given dataset).
- The **Streamlit web application** offers a user-friendly interface for interacting with the model.

**Limitations**:

- The model's performance on new, unseen data is unknown and needs to be validated.
- The dataset is highly imbalanced, with some classes having significantly fewer samples than others.
- The model relies solely on AQI values and does not consider other factors that might influence air quality (e.g., temperature, humidity, wind speed).

**Real-World Applications:**

The AQI Status Predictor can be applied in various real-world scenarios, including:

1. **Public Health Awareness**:
    - Provide real-time air quality updates to the public, helping individuals make informed decisions about outdoor activities.
    - Alert sensitive groups (e.g., children, elderly, people with respiratory conditions) when air quality is unhealthy.
2. **Environmental Monitoring**:
    - Integrate the model into environmental monitoring systems to track air quality trends over time.
    - Identify areas with consistently poor air quality and prioritize interventions.
3. **Policy Making**:
    - Use the model's predictions to inform policy decisions related to air quality management and pollution control.
    - Evaluate the effectiveness of air quality improvement initiatives.
4. **Smart Cities**:
    - Incorporate the model into smart city platforms to provide real-time air quality data to residents and authorities.
    - Trigger automated responses (e.g., closing windows, activating air purifiers) based on air quality predictions.

**Future Work**

To improve the model and extend its capabilities, consider the following:

1. **Address Overfitting**:
    - Use techniques like **cross-validation** to evaluate the model's performance on multiple subsets of the data.
    - Apply **regularization** (e.g., reducing the depth of trees in the RandomForestClassifier) to prevent overfitting.
2. **Handle Class Imbalance**:
    - Use techniques like **oversampling** (e.g., SMOTE) or **under sampling** to balance the dataset.
    - Assign **class weights** to the RandomForestClassifier to give more importance to minority classes.
3. **Incorporate Additional Features**:
    - Include additional features (e.g., temperature, humidity, wind speed) that might influence air quality.
    - Perform feature selection to identify the most relevant features.
4. **Evaluate on New Data**:
    - Test the model on a completely new dataset to ensure it generalizes well.
    - Collect more data, especially for underrepresented classes.
5. **Try Different Algorithms**:
    - Experiment with other machine learning algorithms (e.g., Gradient Boosting, Support Vector Machines) to see if they perform better.
    - Use ensemble methods to combine the predictions of multiple models.
6. **Improve Data Preprocessing**:
    - Handle missing values and outliers in the dataset.
    - Normalize or standardize numerical features if necessary.

7. **Hyperparameter Tuning**:
    - Use **GridSearchCV** or **RandomizedSearchCV** to find the optimal hyperparameters for the RandomForestClassifier.
8. **Deploy in Real-Time Systems**:
    - Integrate the model into real-time air quality monitoring systems.
    - Develop a mobile app or API for easy access to predictions.
9. **Visualize Predictions**:
    - Create interactive visualizations (e.g., maps, charts) to display air quality predictions and trends.
    - Provide historical data and forecasts to help users understand air quality patterns.
10. **Expand to Other Pollutants**:
    - Extend the model to predict the levels of specific pollutants (e.g., PM2.5, NO2, O3) in addition to overall AQI.
    - Provide detailed insights into the sources and impacts of different pollutants.

## 7. REFERENCES


- Kaggle: https://www.kaggle.com/datasets
- Pandas: https://pandas.pydata.org
- NumPy: https://numpy.org
- Scikit Learn: https://scikit-learn.org , RandomForestClassifier , Label Encoder
- CSS: https://developer.mozilla.org , https://developer.mozilla.org/en-US/docs/Glossary/Base64
- Matplotlib: https://matplotlib.org/stable/api/pyplot_summary.html
- Seaborn: https://seaborn.pydata.org/tutorial/relational.html
- Streamlit: https://docs.streamlit.io/get-started/tutorials
- Joblib: https://joblib.readthedocs.io/en/stable/