AIR QUALITY MANAGEMENT

Developing an air quality monitoring system using Python typically involves collecting data from air quality sensors and then processing, analyzing, and visualizing that data. Here's a simplified step-by-step guide on how to get started:

1. **Select Air Quality Sensors:**
   Choose appropriate sensors to measure various air quality parameters like particulate matter (PM2.5, PM10), carbon dioxide (CO2), carbon monoxide (CO), ozone (O3), nitrogen dioxide (NO2), etc. Popular sensors include the SDS011 for PM, MH-Z19 for CO2, and others.

2. **Data Acquisition:**
   Interface with the sensors and collect data. You might use libraries like `pyserial` or specialized libraries for sensor modules. Ensure you have the necessary hardware to connect and power the sensors.

3. **Data Processing:**
   Process the raw data to obtain meaningful values. This may involve data cleaning and calibration to ensure accuracy. For example, convert voltage readings into air quality measurements using sensor-specific calibration curves.

4. **Data Storage:**
   Store the processed data in a database (e.g., SQLite, MySQL, or cloud-based solutions like AWS DynamoDB). This makes it easier to manage and retrieve historical data.

5. **Data Analysis:**
   Perform data analysis to identify trends, anomalies, and patterns in air quality. Python libraries like Pandas and NumPy are useful for this task.

6. **Visualization:**
   Create visualizations of air quality data using libraries like Matplotlib, Seaborn, or Plotly. You can generate plots, charts, and graphs to make the data more understandable.

7. **Alerts and Notifications:**
   Implement a notification system to alert users when air quality falls below a certain threshold. You can use tools like Python's `smtplib` for email alerts or integrate with messaging platforms.

8. **Web Interface (Optional):**
   Create a web-based dashboard or user interface using web frameworks like Flask or Django. This allows users to access air quality data remotely.

9. **Real-time Monitoring (Optional):**
   If you want to provide real-time updates, use a service like MQTT to stream data from sensors to your monitoring system.

10. **Deployment:**
    Deploy your application on a server or a Raspberry Pi for continuous monitoring. Make sure it's accessible and secure.

11. **Maintenance:**
    Regularly calibrate and maintain the sensors to ensure accurate measurements. Update and maintain your Python code as needed.

Remember that the specific sensors you choose will impact the code and libraries you use, so you should refer to the sensor's datasheets and the documentation for any additional guidance. Additionally, consider environmental factors such as sensor placement and weatherproofing if needed.

Example program

```
# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.model_selection import train_test_split

# Load the air quality dataset

df = pd.read_csv('air_quality_data.csv')

# Explore the data

print(df.head())

# Split the data into training and testing sets

X = df[['Temperature', 'Humidity', 'Wind Speed']]
```

```
y = df['Air Quality']

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

# Train a linear regression model

model = LinearRegression()

model.fit(X_train, y_train)

# Evaluate the model on the test set

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error: ", mse)

print("Root Mean Squared Error: ", rmse)

print("R-squared: ", r2)

# Visualize the predicted vs actual values

plt.scatter(y_test, y_pred)

plt.xlabel('Actual Air Quality')

plt.ylabel('Predicted Air Quality')

plt.title('Air Quality Prediction')

plt.show()
```