Project Title: AQM - IoT

Phase 2: Innovation

Consider incorporating predictive modeling to forecast air quality trends based on historical data.

1. Project Objectives:

Clearly outline the goals of your air quality monitoring project, including the specific pollutants you want to measure (e.g., PM2.5, PM10, CO2, NO2) and the areas or locations you intend to monitor.

In this project we are going to make an **IoT Based Air Quality Monitoring System** in which we will **monitor the Air Quality over a webserver using internet** and will trigger a alarm when the air quality goes down beyond a certain level, means when there are sufficient amount of harmful gasses are present in the air like CO2, smoke, alcohol, benzene and NH3. It will show the air quality in PPM on the LCD and as well as on the webpage so that we can monitor it very easily.

MQ135 sensor used as the air quality sensor which is the best choice for monitoring Air Quality as it can detect most harmful gasses and can measure their amount accurately. In this IOT project, you can monitor the pollution level from anywhere using your computer or mobile. We can install this system anywhere and can also trigger some device when pollution goes beyond some level, like we can switch on the Exhaust fan or can send alert SMS/mail to the user.

2. Select Air Quality Sensors:

Choose appropriate air quality sensors based on your project objectives. Some commonly used sensors include:

- **Particulate Matter (PM) Sensors:** for PM2.5 and PM10 measurements.

- **Gas Sensors:** For measuring gasses like carbon dioxide (CO2), carbon monoxide (CO), nitrogen dioxide (NO2), sulfur dioxide (SO2), and ozone (O3).

- **Weather Sensors:** for measuring temperature, humidity, and other weather-related variables.

Ensure the selected sensors are accurate, reliable, and capable of providing real-time data.

3. Hardware Selection:

Choose IoT hardware platforms to connect and collect data from the sensors.
Common options include:

- **Raspberry Pi:** A popular choice for its versatility and community support.

- **Arduino:** Suitable for smaller-scale projects.

- **Dedicated IoT modules:** Specialized hardware designed for IoT applications.

Consider factors like power efficiency, connectivity options (Wi-Fi, LoRa, cellular, etc.), and compatibility with the chosen sensors.

4.Data Collection:

Gather historical air quality data from reliable sources, which may include government agencies, environmental monitoring stations, or research institutions. Ensure the data is comprehensive and covers a significant time period.

5.Data Preprocessing:

Clean the data by removing missing values, outliers, and inconsistencies.

Convert data into a suitable format for analysis, such as time series data with timestamps.

6.Feature Selection and Engineering:

Identify relevant features, such as pollutant concentrations, meteorological data, and geographical information.
Create new features if needed, like seasonality indicators, holidays, or special events.

7.Data Splitting:

Divide the dataset into training, validation, and testing sets to assess the model's performance.

8.Selecting a Predictive Model:

Choose a suitable predictive model for air quality forecasting. Common choices include time series models (e.g., ARIMA, SARIMA), machine learning models (e.g Random Forest, Gradient Boosting, LSTM, or CNN for deep learning), and hybrid models that combine various techniques.

9.Model Training:

Train the selected model using the training dataset. Fine-tune hyperparameters and assess model performance using the validation dataset. Techniques like cross-validation can be employed to prevent overfitting.

10.Model Evaluation:

Evaluate the model using appropriate metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) to assess its accuracy and generalization capabilities.

11.Model Interpretation:

Understand the model's feature importance to gain insights into what influences air quality.

12.Model Deployment:

Once satisfied with the model's performance, deploy it to make real-time or future air quality predictions. This can be in the form of a web application, API, or regular reports.

13.Continuous Monitoring and Retraining:

Air quality can be affected by various factors, so it's essential to monitor the model's performance over time. Periodically retrain the model to adapt to changing conditions, incorporate new data, and improve accuracy.

14.Communicate Results:

Share the forecasted air quality information with the relevant stakeholders, such as government agencies, the public, and environmental organizations. This can be done through websites, mobile apps, or public announcements.

15.Policy and Decision Support:

Use the forecasted air quality data to inform policy decisions and alert the public about potential air quality issues. For example, recommend actions like limiting outdoor activities during poor air quality days.

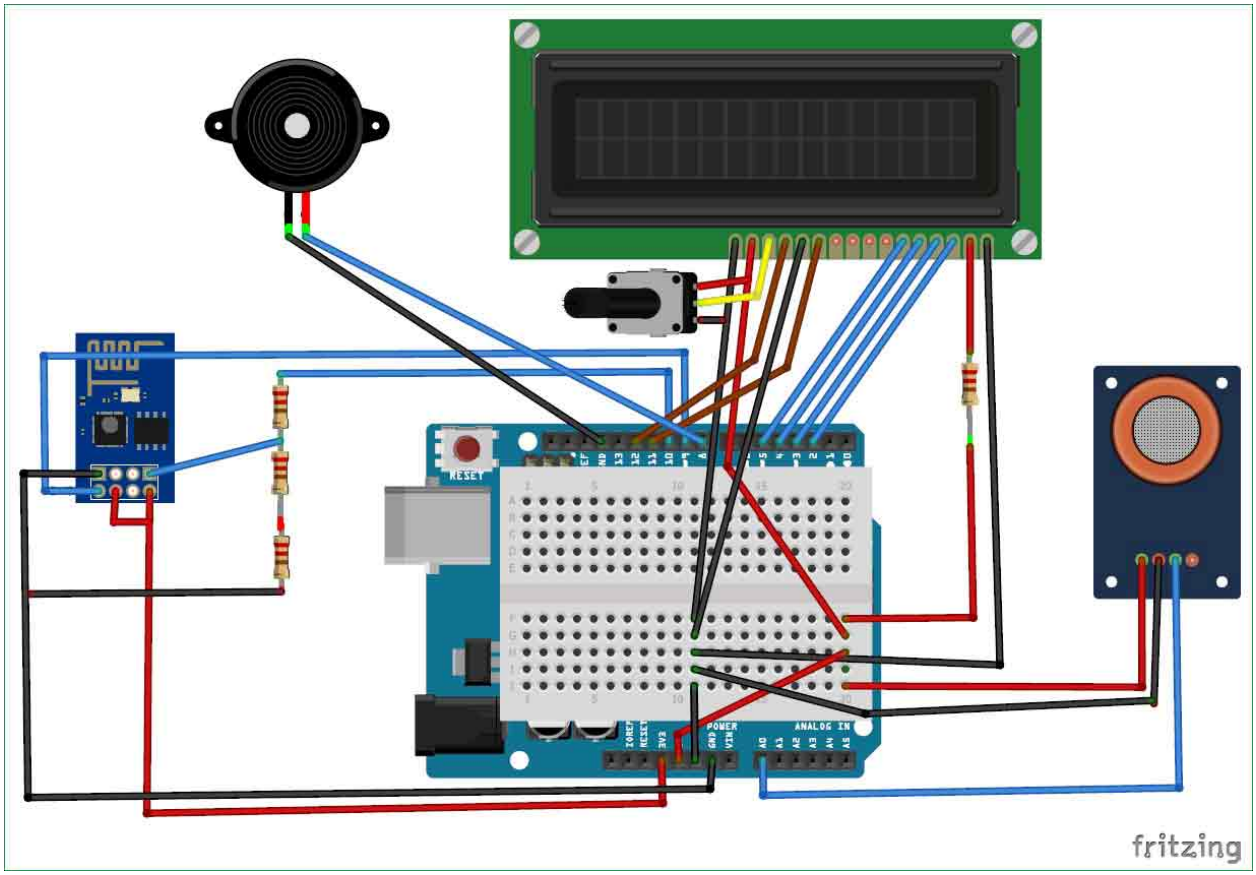
16.Feedback Loop:

Establish a feedback mechanism to collect data on the effectiveness of policies and measures taken to improve air quality. Use this feedback to improve the forecasting model further.

17.Required Components:

- MQ135 Gas sensor
- Arduino Uno
- Wi-Fi module ESP8266
- 16X2 LCD
- Breadboard
- 10K potentiometer
- 1K ohm resistors
- 220 ohm resistor
- Buzzer

Circuit Diagram:



- Connect pin 1 (VEE) to the ground.
- Connect pin 2 (VDD or VCC) to the 5V.
- Connect pin 3 (V0) to the middle pin of the 10K potentiometer and connect the other two ends of the potentiometer to the VCC and the GND. The potentiometer is used to control the screen contrast of the LCD. Potentiometer of values other than 10K will work too.
- Connect pin 4 (RS) to the pin 12 of the Arduino.
- Connect pin 5 (Read/Write) to the ground of Arduino. This pin is not often used so we will connect it to the ground.
- Connect pin 6 (E) to pin 11 of the Arduino. The RS and E pin are the control pins which are used to send data and characters.

The following four pins are data pins which are used to communicate with the Arduino.

- Connect pin 11 (D4) to pin 5 of Arduino.

- Connect pin 12 (D5) to pin 4 of Arduino.
- Connect pin 13 (D6) to pin 3 of Arduino.
- Connect pin 14 (D7) to pin 2 of Arduino.
- Connect pin 15 to the VCC through the 220 ohm resistor. The resistor will be used to set the backlight brightness. Larger values will make the backlight much darker.
- Connect pin 16 to the Ground.

Code Implementation:

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
# Simulated historical air quality data
historical_data = pd.read_csv('historical_air_quality_data.csv')
```

```
# Simulated real-time IoT data
# In a real application, you would receive this data from IoT devices.
real_time_data = pd.read_csv('real_time_iot_data.csv')
```

```
# Data Preprocessing
# You would typically need to preprocess and clean the data in a real scenario.
# In this simplified example, we'll assume the data is already clean.

# Feature Engineering
# Create features based on historical data

# Train-Test Split
X = historical_data[['Feature1', 'Feature2', ...]]  # Select relevant features
y = historical_data['AQI']  # Target variable (Air Quality Index)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Model Training
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
# Model Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

```
# Real-time Forecasting
# Assuming you have real-time IoT data, preprocess and predict the air quality.

# In a real application, you would continuously receive data from IoT devices.
# For simplicity, we'll use the simulated real-time data here.
real_time_predictions = model.predict(real_time_data)
```

```
# Visualize Historical and Real-time Data
plt.figure(figsize=(10, 5))
plt.plot(historical_data['Date'], historical_data['AQI'], label='Historical Data')
plt.plot(real_time_data['Date'], real_time_predictions, label='Real-time Forecast', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Air Quality Index')
plt.legend()
plt.show()
```

Conclusion:

Remember that air quality forecasting is a complex task, and the choice of model and data sources may vary based on your specific location and goals. Regular updates and improvements to your predictive model are essential to ensure its accuracy and relevance.