Project Report

'Project Vaayu'

IoT Based Air Pollution Monitoring System

B.Tech.

Department of Electronics and Communication Engineering

National Institute of Technology

Tiruchirappalli – 620 015

Abstract:

The level of pollution is increasing rapidly due to factors like industrialization, urbanization, increase in population and vehicle use which can affect human health. Quality of the air is the most important factor that directly causes diseases and decreases the quality of life. The development of air quality monitoring system will capture the amount of major pollutants and corresponding sources at appropriate level over a web server using the Internet. It will alert the stockholders when the air quality goes down beyond a certain level, i.e., when there is sufficient amount of harmful gases present in the air like CO_2, smoke, methane, NH_3 and NOx. It will show the air quality in PPM on a webpage as well as a mobile app so that air pollution can be monitored very easily. Future concentrations will also be predicted using machine learning approaches. The fundamental aspects of the proposed project are to reduce the cost of the infrastructure and enhance the data collection and dissemination to all stockholders.

Introduction:

Air pollution is the biggest problem of every nation, whether it is developed or developing. Health problems are growing at faster rate especially in urban areas of developing countries where industrialization and growing number of vehicles leads to release of a lot of gaseous pollutants. Air Pollution is the presence of substances in the atmosphere that are harmful to the health of humans and other animals. There are different types of air pollutants such as gases (Ozone, Nitrogen Dioxide, Carbon Monoxide, etc.) and particulates (PM10 and PM2.5). Harmful effects of pollution include mild allergic reactions such as irritation of the throat, eyes and nose as well as some serious problems like bronchitis, heart diseases, pneumonia, lung and aggravated asthma.

According to a study published in the scientific paper titled 'Health and economic impact of air pollution in the States of India: The Global Burden of Disease Study 2019', 1.7 million deaths were attributable to air pollution in India in 2019, which was 18% of the total deaths in the country, while the economic loss due to the lost output from premature deaths and morbidity from air pollution was 1.4% of the GDP in India during this time, which is equivalent to ₹260,000 crore. Furthermore, emerging researches, including a study from Harvard T.H. Chan School of Public Health, finds that breathing more polluted air over many years may itself worsen the effects of COVID-19.

Currently, the monitoring of air quality levels is achieved by placing sensors in various locations and the sensors alert if the pollution levels exceed the threshold level. However, there hardly exists a mobile solution wherein an individual can contribute to the pollution level checking from the convenience of their homes. Moreover, solutions for forecasting gas and particulate matter levels are location specific and require expensive training data (Meteorological data, wind speeds, etc.) which is unavailable in the reach of the common people.

Therefore, we aim to collect data about the concentration of NO2, CO_2, CO, as well as temperature and humidity via a compact and affordable solution, and host its visualization on a website as well as on a mobile app. We calculate the Air Quality Index (AQI) and also predict their future concentrations using machine learning approaches. The future concentration levels that are being predicted is aimed to provide maximum accuracy based on the regions in which the system is placed. The proposed device can be placed anywhere in the user's home and the user can obtain real time data of the pollution levels in their immediate locality. The accuracy of the prediction model can be improved based on the data sensed from the user's surroundings.

Components, Sensors, Boards:

- To Measure Concentration of Carbon Monoxide in ppm – MQ7 Sensor
- To Measure Concentration of Methane, Butane and LPG in ppm – MQ 2 sensor
- To measure General Air Quality in ppm (CO_2, NO_2, NOx) – MQ135 Sensor
- Node MCU
- Arudino UNO

Software Requirements:

- Arduino IDE 1.6.13 – To code the Arduino UNO and the Node MCU
- HTML, CSS, Javascript – For designing the website
- Google Collaboratory – For model training and Prediction
- Google Firebase – For hosting the website on the cloud
- Thingspeak IoT Cloud – For reading and visualizing sensor values
- Blynk App – Mobile App for reading sensor value

Bill of Materials:

Items	Quantity	Cost
MQ135, MQ7 and MQ2 Gas Sensors	1 each	₹ 525
Arduino UNO	1	₹ 390
Node MCU	1	₹ 290
DHT11 Temperature & Humidity Sensor	1	₹ 200
Breadboard	1	₹ 135
Jumper Wire (Male to Female)	12	₹ 20
Connecting Wires	6	₹ 10
10 kΩ Resistor	2	₹ 1
Grand Total		₹ 1571

Specification of Sensors:

MQ135 Sensor:

- Operating Voltage is +5V
- Detects/Measures NH3, NOx, alcohol, Benzene, smoke, CO2, etc.
- Analog output voltage: 0V to 5V
- Digital output voltage: 0V or 5V (TTL Logic)
- Preheat time over 24 hours
- Can be used as a Digital or analog sensor
- The Sensitivity of Digital pin can be varied using the potentiometer

MQ7 Sensor:

- Operating Voltage is +5V
- Detects/Measures Carbon Monoxide (CO)
- Analog output voltage: 0V to 5V
- Digital output voltage: 0V or 5V (TTL Logic)
- Preheat time no less than 48 hours
- Can be used as a Digital or analog sensor
- The Sensitivity of Digital pin can be varied using the potentiometer

MQ2 Sensor:

- Operating Voltage is +5V
- Detects/Measures LPG, Alcohol, Propane, Hydrogen, and even methane
- Analog output voltage: 0V to 5V
- Digital output voltage: 0V or 5V (TTL Logic)
- Preheat time over 48 hours
- Can be used as a Digital or analog sensor
- The Sensitivity of Digital pin can be varied using the potentiometer

DHT11 Sensor:

- Operating Voltage is +3.5V to +5V
- Operating Current: 0.3 mA (measuring) 60 μA (standby)
- Temperature Range: 0˚C to 50˚C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1˚C and ±1%

Specification of Boards:

Node MCU:

- Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
- Operating Voltage: 3.3 V
- Input Voltage: 7-12 V
- Digital I/O Pins (DIO): 16
- Analog Input Pins (ADC): 1
- UART: 1
- SPI: 1
- I2C: 1
- Flash Memory: 4 MB
- SRAM: 64 KB
- Clock Speed: 80 MHz

USB-TTL based on CP2102 is included onboard, Enabling Plug and Play.

Arduino UNO:

- Microcontroller: Microchip ATmega328P
- Operating Voltage: 5 V
- Input Voltage: 7-20 V
- Digital I/O Pins (DIO): 14 (of which 6 can provide PWM output)
- Analog Input Pins: 6
- UART: 1
- SPI: 1
- I2C: 1
- Flash Memory: 32 KB of which 0.5 KB used by bootloader
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz

Specification of Software Used:

Google Collaboratory:

CPU-only VMs:

- CPU Model Name: Intel(R) Xeon (R)
- CPU Frequency: 2.30 GHz
- No. of CPU cores: 2
- CPU Family: Haswell
- Available RAM: 12 GB (upgradable to 26.75 GB)
- Disk Space: 25 GB

GPU VMs:

- GPU: Nvidia K80 / T4
- GPU Memory: 12 GB / 16 GB
- GPU Memory Clock: 0.82 GHz / 1.59 GHz

- Performance: 4.1 TFLOPS / 8.1 TFLOPS
- Support Mixed Precision: No / Yes
- No. of CPU Cores: 2
- Available RAM: 12 GB (Upgradable to 26.75 GB)
- Disk Space: 358 GB

Maximum execution time is 12 hours and Maximum idle time is 90 minutes.

Google Firebase:

The free tier is enough for our use cases.

- Stored data: 1 GiB
- Document Reads: 50,000 per day
- Document Writes: 20,000 per day
- Document deletes: 20,000 per day
- Network egress: 10 GiB per month

Blynk Mobile App:

The free tier is enough for our use cases.

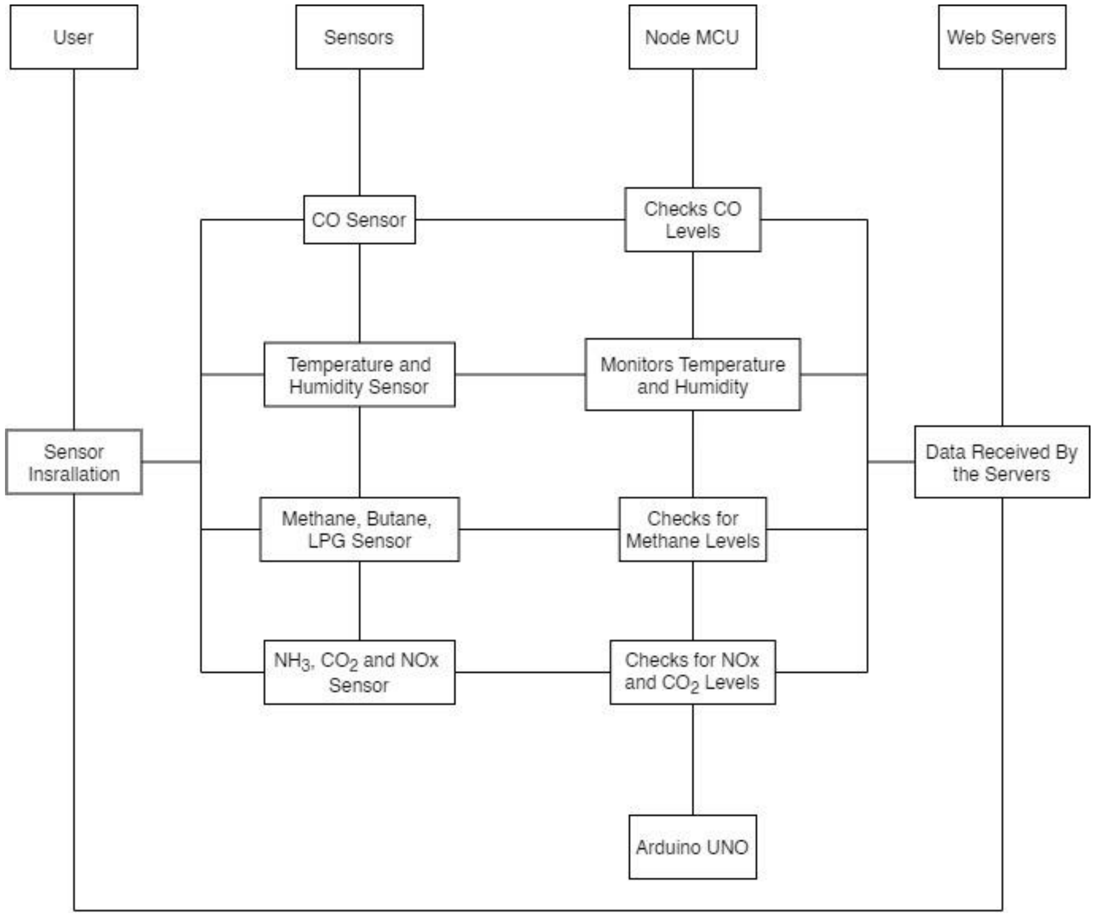
- Limited to 1 device
- Accessible to a maximum of 5 users
- Access to basic Widgets
- Historical Data Storage of up to 1 week
- Access to Blynk Cloud

Thingspeak IoT Cloud:

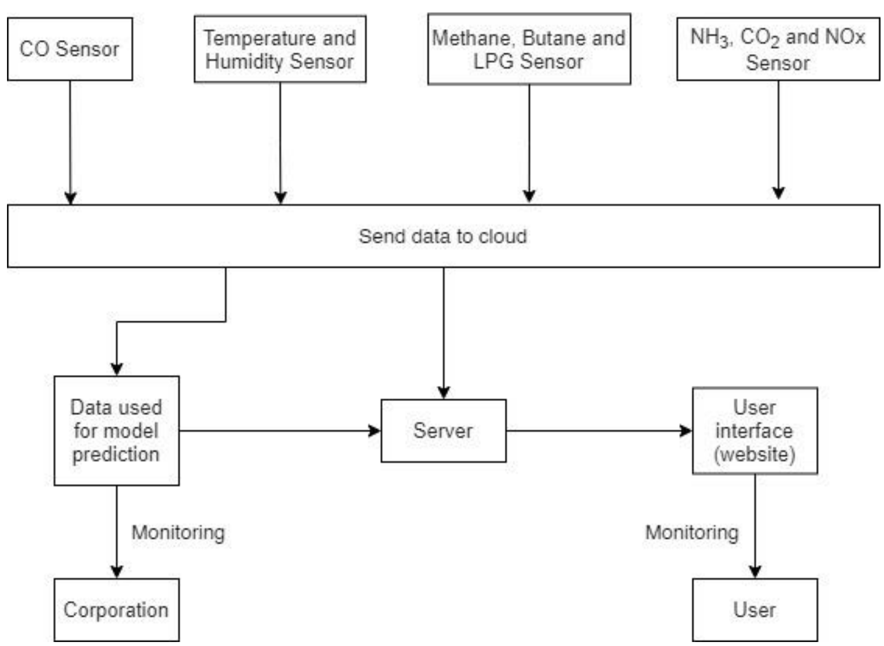
The student license provides most of the functionalities required.

- Limited to 8 channels per account (8 sensor readings can be visualized)
- MATLAB Tools available for performing analytics
- Requires a minimum of 10-15 seconds to receive sensor value
- Real Time visualization of Sensor Data

Basic System Architecture:



Block Schematic:



Principle of Working:

Perception Layer:

Here, the MQ135, MQ7, and MQ2 sensors are used to obtain raw values of the concentration of NH3, CO2, CO, NOx and Methane levels in the atmosphere. The Temperature and humidity sensor is also used to obtain the temperature and humidity in air that will aid us in checking the general air quality. All this information is sent to the processing layer

Processing Layer:

In this layer, we utilize the Arduino UNO. The data collected from the perception layer is received by the Arduino UNO and processed. There are two main processes that are involved here:

1) **Data Computation:** Here, the raw data obtained from the Gas sensors is used to obtain the proper ppm value of the gas concentrations in the atmosphere. This is done by referring to the datasheet of the gas sensors and performing appropriate calibration. The DHT Library can be used to obtain data from the Temperature and Humidity sensor.
2) **Data Serialization:** Data serialization is the process of converting data objects present in complex data structures into a byte stream for storage, transfer and distribution purposes. To achieve this, serialized data will be relayed to the Node MCU from the Arduino end by using ArduinoJSON. JSON (JavaScript Object Notation) is a popular, lightweight format for storing and transporting data, and the ArduinoJSON supports serialization and deserialization. The serialized data from one end will be deserialized in the other end of the node.

Note: As the Node MCU contains only one Analog Port, only one Gas sensor can be connected to the NodeMCU. Hence, the Arduino UNO is used to obtain reading from other gas sensors and Data Serialization is used to send the values obtained to the Node MCU.

Transport Layer:

Transport Layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers. The transport protocol that’s being utilized is UART, which stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data.

Application Layer

Here, we define how applications interface with the lower layer protocols to send data over the network. The application data, in files, is encoded by the application layer protocol. It is encapsulated in the transport layer protocol which provides transaction-oriented communication over the network. Application layer protocol enables process-to-process connection using ports. Here we are using the HTTP Protocol (Hypertext Transfer Protocol).

Implementation:

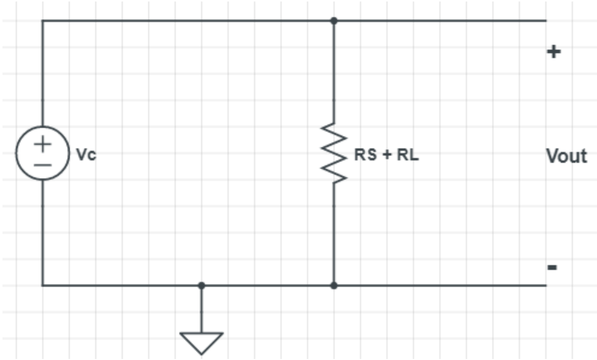
1. For recording the air quality level, we have used:
   - **Gas sensors:** To determine CO2, NOx, NH3, CO, Methane and Smoke level, we have used gas sensors like the MQ135, MQ7, and the MQ2.
   - **Temperature and Humidity Sensor:** Temperature and Humidity also plays a crucial role in determining the overall air quality, so we use the DHT11 sensor for the same.
2. The data we obtain from the sensors is sent at regular intervals to the Thingspeak IoT Cloud as well as the Blynk IoT Cloud using the Node MCU.
3. The Thingspeak IoT Cloud reads sensor value and performs a graph visualization over time.
4. The website, which is hosted in the Google Firebase Cloud, communicates with the Thingspeak IoT cloud and obtains the real time visualization of the sensor data, which can now be viewed through the website.
5. At the same time, The Blynk App obtains sensor data from the Blynk IoT cloud and displays it real time using widgets.
6. The user can opt for being alerted by e-mail, if the concentration of any of the gases exceeds the threshold level.
7. The values that are being obtained from the sensors can be exported as a .csv file from the Thingspeak IoT Dashboard. This file is sent to our prediction model to check and improve upon the accuracy of the model.
8. The visualization of the predicted values vs. the original values is performed in Google Collaboratory.

Due to Geographical and Time Constraints, the model is initially trained using existing datasets of India’s Air Pollution from 2015 to 2020.The training of the model is done in Google Collaboratory.

Software Codes:

Sensor Calibration:

The most important step is to calibrate the sensor in fresh air and then derive an equation that converts the sensor output voltage value into our convenient units PPM (parts per million). The following diagram is the internal circuit diagram of the MQ135 sensor with the R_S and the R_L values.



From Ohm's Law, at constant temperature, we can derive I as follows:

$$I = \frac{V}{R}$$

From the above figure, we can then obtain the equivalent equation as:

$$I = \frac{V_c}{R_s + R_l}$$

We can now obtain the output voltage at the load resistor using the value obtained for I and the Ohm's Law at constant temperature $V = I \times R$

$$V_{R_l} = \left[\frac{V_c}{R_s + R_l}\right] \times R_l$$

$$V_{R_l} = \left[\frac{V_c \times R_l}{R_s + R_l}\right]$$

So now we solve for R_s:

$$V_{R_l} \times (R_s + R_l) = V_c \times R_l$$

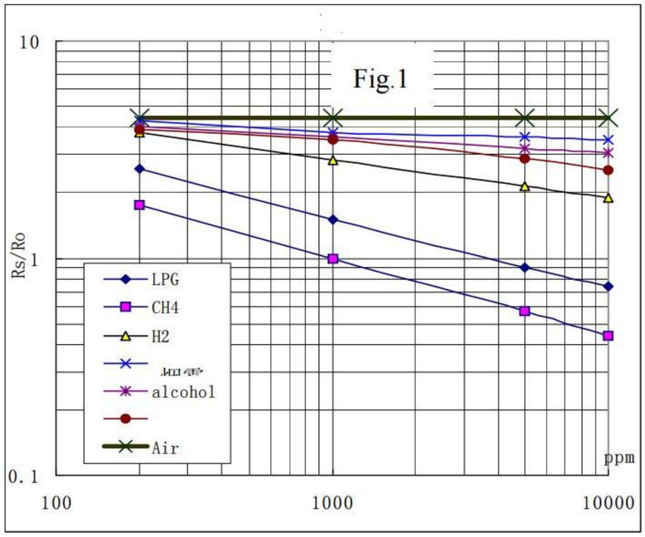
$$\left(V_{R_l} \times R_s\right) + \left(V_{R_l} \times R_l\right) = V_c \times R_l$$

$$V_{R_l} \times R_s = (V_c \times R_l) - \left(V_{R_l} \times R_l\right)$$

$$R_s = \frac{(V_c \times R_l) - (V_{R_l} \times R_l)}{V_{R_l}}$$

$$\boxed{R_s = \frac{(V_c \times R_l)}{V_{R_l}} - R_l}$$

The resultant equation helps us to find the internal sensor resistance for fresh air.



Now, from the graph shown above, we can see that the resistance ratio in fresh air is a constant:

$$\boxed{\frac{R_s}{R_0} = 3.7}$$

The value 3.7 which is mentioned in the above equation is depicted from the graph shown above which is obtained from the datasheet. To calculate R_0, we will need to find the value of the R_s in fresh air. This will be done by taking the analog average readings from the sensor and converting it to voltage. Then, we will use the R_s formula to find R_0. First of all, we will treat the lines as if they were linear. This way we can use one formula that linearly relates the ratio and the concentration. By doing so, we can find the concentration of a gas at any ratio value even outside of the graph's boundaries. The formula we will be using is the equation for a line, but for a log-log scale. The formula for a line is:

$$y = mx + b$$

For a log-log scale, the formula looks like this:

$$\log_{10} y = m \times \log_{10} x + b$$

Let's find the slope. To do so, we need to choose 2 points from the graph. In our case, we choose the points (200,2.6) and (10000,0.75) from the LPG (Liquified Petroleum gas) line from the above graph. The LPG line is a result of sensor under testing with various level of LPG as input. The formula to calculate slope m(here) is the following:

$$m = \frac{\log y - \log(y_0)}{\log x - \log(x_0)}$$

If we apply the logarithmic quotient rule, we get the following:

$$m = \frac{\log\left({}^{y}/_{y_0}\right)}{\log\left({}^{x}/_{x_0}\right)}$$

Now we substitute the values for x, x_0, y, and y_0:

$$m = \frac{\log\left({}^{0.75}/_{2.6}\right)}{\log\left({}^{10000}/_{200}\right)}$$

$$m = -0.318$$

Now that we have m, we can calculate the y intercept. To do so, we need to choose one point from the graph (once again from the LPG line). In our case, we chose (5000,0.9)

$$\log(y) = m \times \log(x) + b$$

$$b = \log(0.9) - (-0.318) \times \log(5000)$$

$$b = 1.13$$

Now that we have m and b, we can find the gas concentration for any ratio with the following formula:

$$\log(x) = \frac{\log(y) - b}{m}$$

However, in order to get the real value of the gas concentration according to the log-log plot we need to find the inverse log of x:

$$\boxed{x = 10^{\frac{\log(y) - b}{m}}}$$

Using the equations thus obtained, we will be able to convert the sensor output values into PPM (Parts per Million). The following procedure is repeated for the MQ-2 and well as the MQ-7 sensor for calibration.

Arduino Code:

Code to obtain R_0 value (MQ-135 Sensor):

```
void setup() {
  Serial.begin(9600); //Baud rate
  pinMode(A0,INPUT);
}

void loop() {
  float sensor_volt; //Define variable for sensor voltage
  float RS_air; //Define variable for sensor resistance
  float R0; //Define variable for R0
  float sensorValue=0.0; //Define variable for analog readings
  Serial.print("Sensor Reading = ");
  Serial.println(analogRead(A0));

  for(int x = 0 ; x < 500 ; x++) //Start for loop
  {
    sensorValue = sensorValue + analogRead(A0); //Add analog values of sensor
500 times
  }
  sensorValue = sensorValue/500.0; //Take average of readings
  Serial.print("Average = ");
  Serial.println(sensorValue);
  sensor_volt = sensorValue*(5.0/1023.0); //Convert average to voltage
  RS_air = ((5.0*10.0)/sensor_volt)-10.0; //Calculate RS in fresh air
  R0 = RS_air/3.7; //Calculate R0

  Serial.print("R0 = "); //Display "R0"
  Serial.println(R0); //Display value of R0
  delay(1000); //Wait 1 second
}
```

Code to obtain PPM value (MQ-135 Sensor):

```
int gas_sensor = A0; //Sensor pin
float m = -0.3376; //Slope
float b = 0.7165; //Y-Intercept
float R0 = 3.12; //Sensor Resistance in fresh air from previous code

void setup() {
  Serial.begin(9600); //Baud rate

  pinMode(gas_sensor, INPUT); //Set gas sensor as input
}

void loop() {
```

```
  float sensor_volt; //Define variable for sensor voltage
  float RS_gas; //Define variable for sensor resistance
  float ratio; //Define variable for ratio
  int sensorValue = analogRead(gas_sensor); //Read analog values of sensor
  Serial.print("SENSOR RAW VALUE = ");
  Serial.println(sensorValue);
  sensor_volt = sensorValue*(5.0/1023.0); //Convert analog values to voltage
  Serial.print("Sensor value in volts = ");
  Serial.println(sensor_volt);
  RS_gas = ((5.0*10.0)/sensor_volt)-10.0; //Get value of RS in a gas
  Serial.print("Rs value = ");
  Serial.println(RS_gas);
  ratio = RS_gas/R0;  // Get ratio RS_gas/RS_air

  Serial.print("Ratio = ");
  Serial.println(ratio);
  float ppm_log = (log10(ratio)-b)/m; //Get ppm value in linear scale
according to the the ratio value
  float ppm = pow(10, ppm_log); //Convert ppm value to log scale
  Serial.print("Our desired PPM = ");
  Serial.println(ppm);
  double percentage = ppm/10000; //Convert to percentage
  Serial.print("Value in Percentage = "); //Load screen buffer with
percentage value
  Serial.println(percentage);
  delay(1000);
}
```

Similar procedure is followed for the MQ-7 as well as the MQ-2 Sensors. The differences will occur only in the R_s/R_0 values, the obtained R_0 value and the linear curve derived (i.e. the m and b value)

For the MQ-7 sensor:

```
R0 = RS_air/27; //Calculate R0
```

```
float m = -0.6527; //Slope
float b = 1.30; //Y-Intercept
float R0 = 21.91; //Sensor Resistance in fresh air from previous code
```

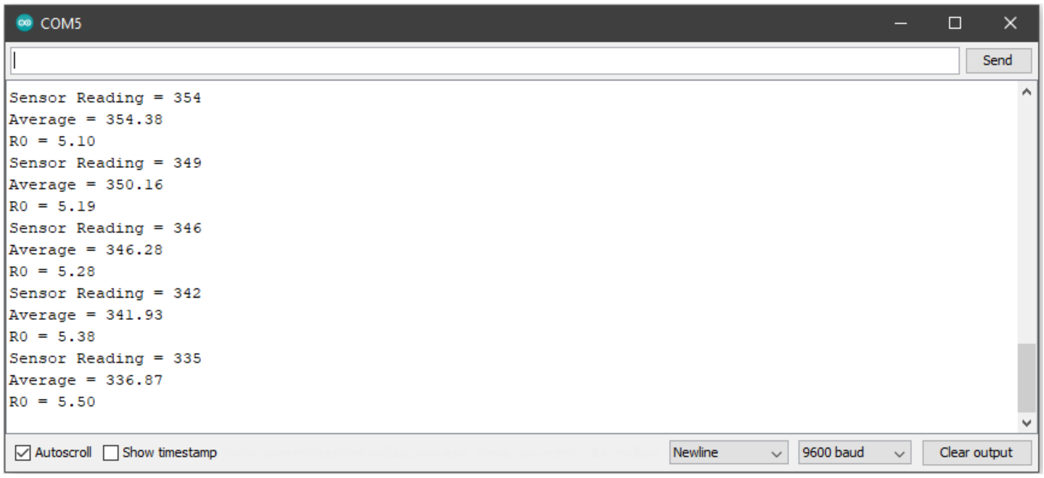
For the MQ-2 sensor:

```
R0 = RS_air/9.8; //Calculate R0
```

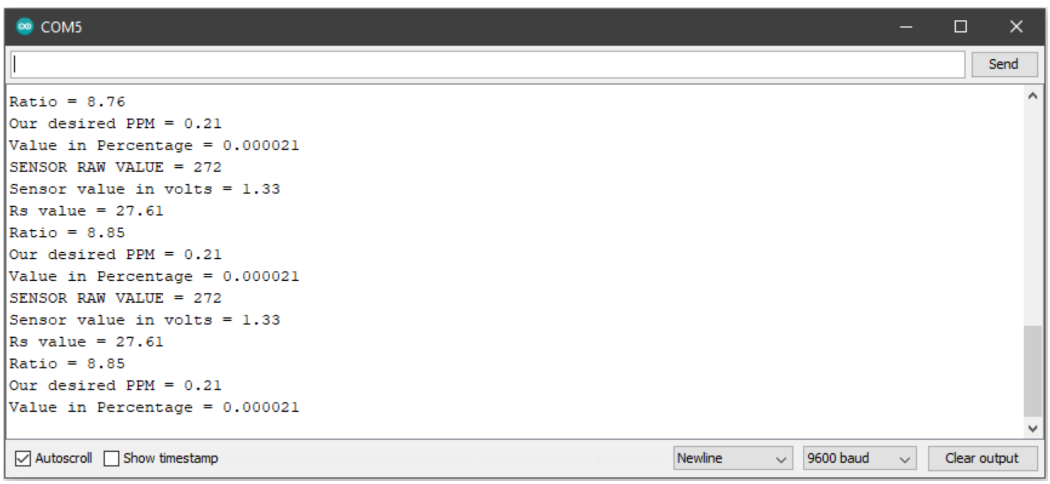
```
float m = -0.3720; //Slope
float b = 1.3492; //Y-Intercept
float R0 = 1.45; //Sensor Resistance in fresh air from previous code
```

The first code is run for 15-20 minutes to obtain an accurate value of R_0. Then, the second code is run for 10-15 minutes for obtaining an accurate value of the PPM for each sensor.

Serial Monitor Output (MQ-135 Sensor):



Obtaining R_0 value



Calculating PPM Value

Values will vary for the other gas sensors, but the serial monitor output listing is the same. Serial Communication is carried out with the PC at 9600 Baud Rate.

Arduino UNO Code:

The Node MCU Board only supports one analog pin. Hence, only one gas sensor can be connected to it and measured. However, for our purposes we require 3 Gas Sensor values in Real Time. Hence, an Arduino UNO Board is used to provide additional analog pins for the other 2 gas sensors. Once the values are obtained, it is sent as a JSON file to the Node MCU where the JSON file is parsed and the values obtained.

The mode of communication used between the Arduino UNO and the Node MCU is UART, and the Baud Rate is set to 57600. The ArduinoJson Library is used for this purpose. The Baud rate for communication between the Arduino UNO to the PC Serial Monitor is set at 9600.

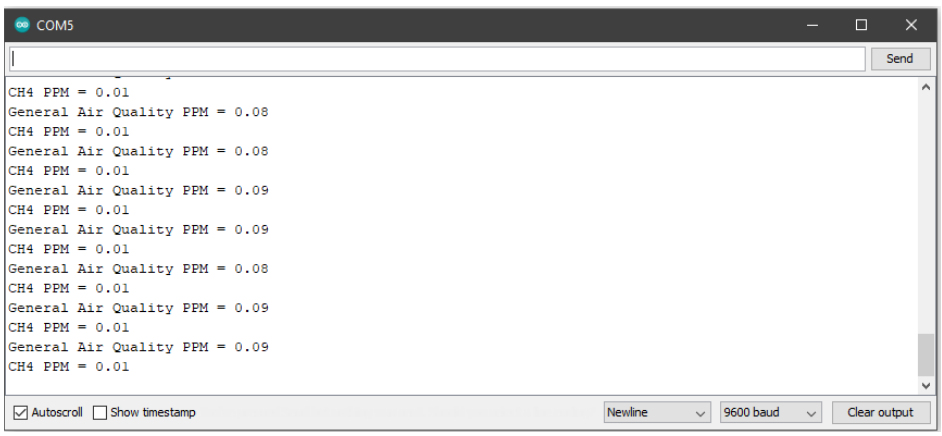
Code to compute MQ-135 and MQ-2 sensor values and send them to the Node MCU:

```
#include <SoftwareSerial.h>
#include <ArduinoJson.h>
SoftwareSerial s(5,6); // setting 5-Rx pin and 6-Tx pin

int gas_sensor = A0; //Sensor pin
float m = -0.3376; //Slope
float b = 0.7165; //Y-Intercept
float R0 = 2.82; //Sensor Resistance in fresh air from previous code

int CH4_sensor = A1; //Sensor pin
float m1 = -0.3720; //Slope
float b1 = 1.3492; //Y-Intercept
float R01 = 1.45; //Sensor Resistance

void setup() {


  Serial.begin(9600);      // PC to Arduino Serial Monitor
  pinMode(gas_sensor, INPUT);
  pinMode(CH4_sensor,INPUT);
  s.begin(57600); // Arduino UNO to Node MCU serial communication
 }

 StaticJsonBuffer<500> jsonBuffer;
 JsonObject& root = jsonBuffer.createObject();

void loop() {
  float sensor_volt; //Define variable for sensor voltage
  float RS_gas; //Define variable for sensor resistance
  float ratio; //Define variable for ratio
  float sensorValue = analogRead(gas_sensor); //Read analog values of sensor
  sensor_volt = sensorValue*(5.0/1023.0); //Convert analog values to voltage
    RS_gas = ((5.0*10.0)/sensor_volt)-10.0; //Get value of RS in a gas
  ratio = RS_gas/R0;  // Get ratio RS_gas/RS_air
  double ppm_log = (log10(ratio)-b)/m; //Get ppm value in linear scale
according to the the ratio value
  double ppm = pow(10, ppm_log); //Convert ppm value to log scale
  Serial.print("General Air Quality PPM = ");
  Serial.println(ppm);


  float sensor_volt1; //Define variable for sensor voltage
  float RS_gas1; //Define variable for sensor resistance
  float ratio1; //Define variable for ratio
  float sensorValue1 = analogRead(CH4_sensor); //Read analog values of sensor
  sensor_volt1 = sensorValue1*(5.0/1023.0); //Convert analog values to
voltage
  RS_gas1 = ((5.0*10.0)/sensor_volt1)-10.0; //Get value of RS in a gas
```

```
  ratio1 = RS_gas1/R01;  // Get ratio RS_gas/RS_air
  double ppm_log1 = (log10(ratio1)-b1)/m1; //Get ppm value in linear scale
according to the the ratio value
  double ppm1 = pow(10, ppm_log1); //Convert ppm value to log scale
  Serial.print("CH4 PPM = ");
  Serial.println(ppm1);

  root["ppm1"] = ppm;
  root["ppm2"] = ppm1;

  if(s.available()>0)
  {
    root.printTo(s);
  }

  delay(4000);                                // wait for 4 sec
 }
```

Pin 5 is used as the transmission Pin and Pin 6 is used as the Receiver Pin.

Serial Monitor Output:



Node MCU Code:

Node MCU has to read and compute the sensor value from the MQ-7 sensor as well as the DHT11 Temperature and Humidity Sensor. Hence, an analog pin and a digital pin is used for the same. At the same time, it needs to receive and parse the JSON document to obtain the values from the MQ-135 and the MQ-2 sensor.

After this, it communicates with the Thingspeak and Blynk IoT Cloud at a Baud Rate of 115200 and sends values every 10 seconds to the Thingspeak Cloud and every 1 second to the Blynk Cloud. Authentication keys are required to connect with these IoT clouds.

In order to receive data from the Arduino UNO, Pin D6 is set as receive pin and Pin D5 is set as transmission pin. The Baud Rate is set to 57600. The Baud rate for communication between the Node MCU to the PC Serial Monitor is set at 115200.

Code for sending Sensor Values to Thingspeak IoT Cloud:

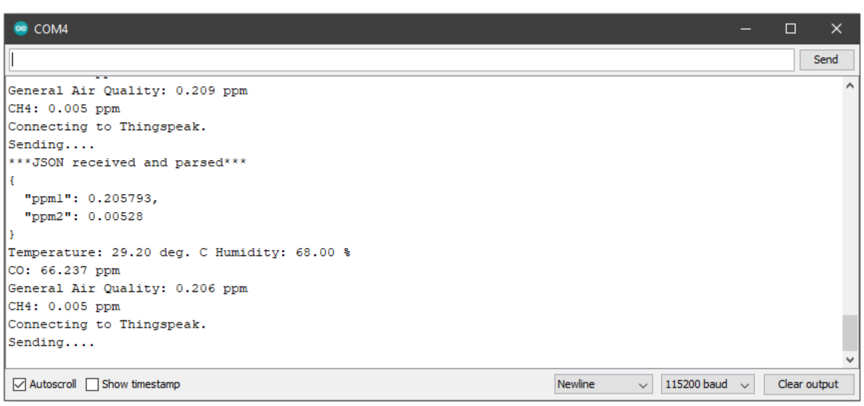
```
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>
#include <ArduinoJson.h>

SoftwareSerial s(D6,D5); // setting D6-Rx ping and D5-Tx pin

#define DHTPIN 2          //DHT11 is connected to GPIO Pin 2
int sensorValue;          //the AOUT pin of the CO sensor goes into analog
pin A0 of the arduino
float m = -0.6527; //Slope
float b = 1.30; //Y-Intercept
float R0 = 21.91; //Sensor Resistance in fresh air from previous code

String apiKey = "VK8KPOZH48MIM81E";     //  Enter your Write API key from
ThingSpeak
const char* ssid =  "";     // Enter your WiFi Network's SSID
const char* pass =  ""; // Enter your WiFi Network's Password
const char* server = "api.thingspeak.com";

float humi;
float temp;

DHT dht(DHTPIN, DHT11);
WiFiClient client;

void setup()
{
       Serial.begin(115200);
       s.begin(57600);
       delay(10);
       dht.begin();

       pinMode(16,INPUT);
       Serial.println("Connecting to ");
       Serial.println(ssid);


       WiFi.begin(ssid, pass);

      while (WiFi.status() != WL_CONNECTED)
     {
            delay(100);
            Serial.print("*");
     }
      Serial.println("");
      Serial.println("***WiFi connected***");
      Serial.println(" ");
```

```
}

void loop()
{
      s.write("s");
      StaticJsonBuffer<1000> jsonBuffer;
      JsonObject& root = jsonBuffer.parseObject(s);

      if (root == JsonObject::invalid())
      {
        return;
      }

      Serial.println("***JSON received and parsed***");
      root.prettyPrintTo(Serial);
      float ppm1 = root["ppm1"];
      float ppm2 = root["ppm2"];
      Serial.println(" ");

      float sensor_volt; //Define variable for sensor voltage
      float RS_gas; //Define variable for sensor resistance
      float ratio; //Define variable for ratio
      int sensorValue = analogRead(0); //Read analog values of sensor

      humi = dht.readHumidity();
      temp = dht.readTemperature();

      sensor_volt = sensorValue*(5.0/1023.0); //Convert analog values to
voltage
      RS_gas = ((5.0*10.0)/sensor_volt)-10.0; //Get value of RS in a gas
      ratio = RS_gas/R0;  // Get ratio RS_gas/RS_air

      double ppm_log = (log10(ratio)-b)/m; //Get ppm value in linear scale
according to the the ratio value
      double ppm = pow(10, ppm_log); //Convert ppm value to log scale
      delay(2000);

      if (client.connect(server,80))   //   "184.106.153.149" or
api.thingspeak.com
      {
       String sendData =
apiKey+"&field1="+String(temp)+"&field2="+String(humi)+"&field3="+String(ppm)
+"&field4="+String(ppm1)+"&field5="+String(ppm2)+"\r\n\r\n";

       //Serial.println(sendData);

       client.print("POST /update HTTP/1.1\n");
       client.print("Host: api.thingspeak.com\n");
       client.print("Connection: close\n");
       client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
       client.print("Content-Type: application/x-www-form-urlencoded\n");
       client.print("Content-Length: ");
       client.print(sendData.length());
       client.print("\n\n");
       client.print(sendData);
```

```
        Serial.print("Temperature: ");
        Serial.print(temp);
        Serial.print(" deg. C Humidity: ");
        Serial.print(humi);
        Serial.println(" %");
        Serial.print("CO: ");
        Serial.print(ppm, 3);  // prints the value read
        Serial.println(" ppm");
        Serial.print("General Air Quality: ");
        Serial.print(ppm1, 3);  // prints the value read
        Serial.println(" ppm");
        Serial.print("CH4: ");
        Serial.print(ppm2, 3);  // prints the value read
        Serial.println(" ppm");
        Serial.println("Connecting to Thingspeak.");
        }

       client.stop();

       Serial.println("Sending....");

 delay(10000);
}
```

The server that the Node MCU connects to is 184.106.153.149 or api.thingspeak.com from port 80.

Serial Monitor Output:



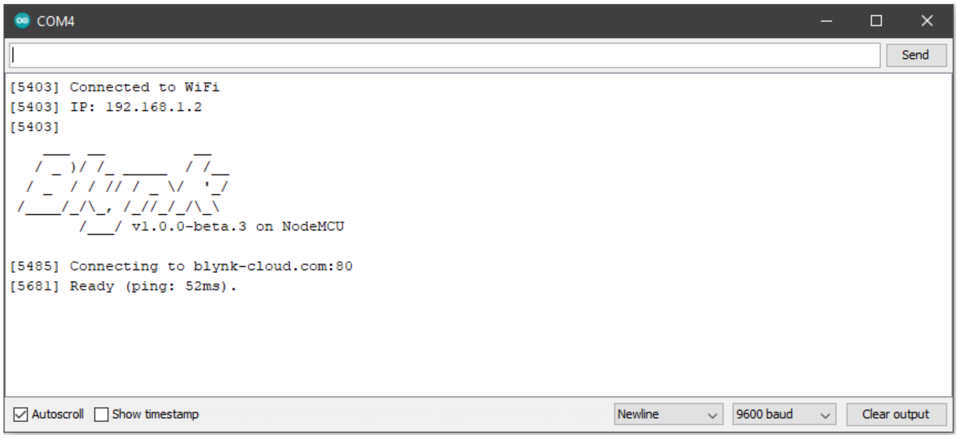
Code for sending Sensor Values to Blynk IoT Cloud:

In this case, the server that the Node MCU connects to is blynk-cloud.com or 192.168.1.100 from port 8442. The Baud Rate for communication between the Node MCU and the Blynk Cloud is 9600. The Baud rate for communication between the Node MCU to the PC Serial Monitor is set at 9600.

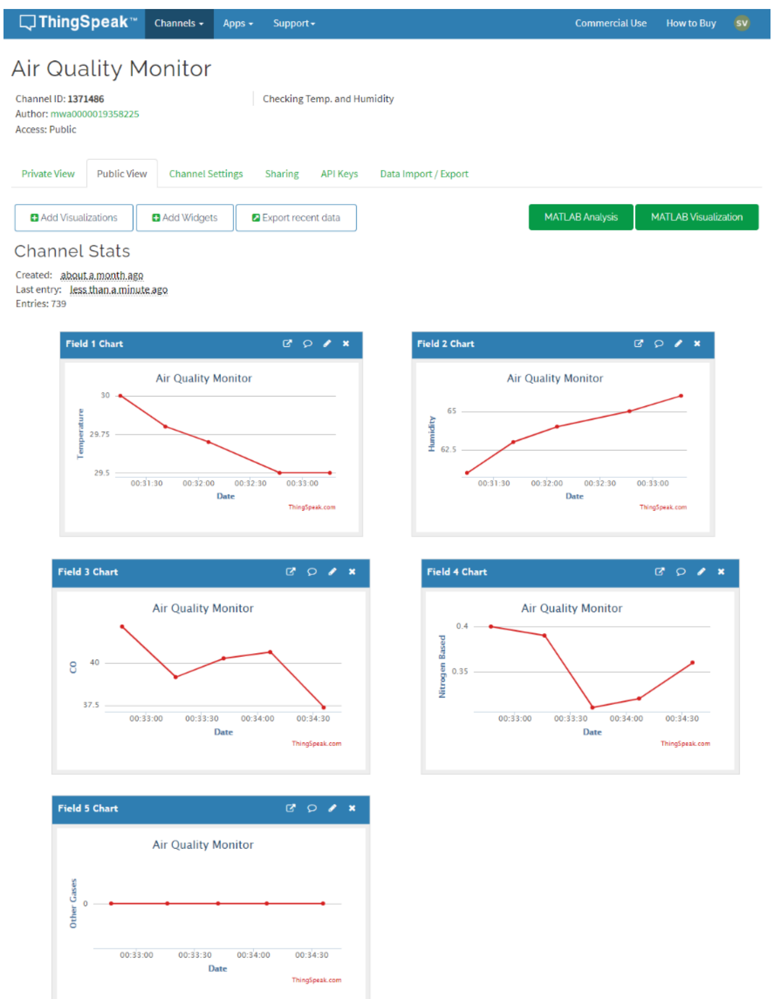
```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <DHT.h>
#include <SoftwareSerial.h>
#include <ArduinoJson.h>

SoftwareSerial s(D6,D5); // setting D6-Rx ping and D5-Tx pin

// We should get Auth Token from the Blynk App.
char auth[] = "g3lkjc1kZ4bW4jtN-jRR-4xSYGggwHN8";
int sensorValue;          //the AOUT pin of the CO sensor goes into analog
pin A0 of the arduino
float m = -0.6527; //Slope
float b = 1.30; //Y-Intercept
float R0 = 21.91; //Sensor Resistance in fresh air from previous code

// WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "";
char pass[] = "";

#define DHTPIN 2          // D3
#define DHTTYPE DHT11     // DHT 11

DHT dht(DHTPIN, DHTTYPE);
BlynkTimer timer;

// This function sends Arduino's up time every second to Virtual Pin.
// In the app, Widget's reading frequency should be set to PUSH. This means
// that you define how often to send data to Blynk App.
void sendSensor()
{
  s.write("s");
  StaticJsonBuffer<1000> jsonBuffer;
  JsonObject& root = jsonBuffer.parseObject(s);

  if (root == JsonObject::invalid())
  {
    return;
  }

  Serial.println("***JSON received and parsed***");
  root.prettyPrintTo(Serial);
  float ppm1 = root["ppm1"];
  float ppm2 = root["ppm2"];
```

```
  Serial.println(" ");

  float sensor_volt; //Define variable for sensor voltage
  float RS_gas; //Define variable for sensor resistance
  float ratio; //Define variable for ratio
  int sensorValue = analogRead(0); //Read analog values of sensor

  float h = dht.readHumidity();
  float t = dht.readTemperature(); // or dht.readTemperature(true) for
Fahrenheit

  sensor_volt = sensorValue*(5.0/1023.0); //Convert analog values to voltage
  RS_gas = ((5.0*10.0)/sensor_volt)-10.0; //Get value of RS in a gas
  ratio = RS_gas/R0;  // Get ratio RS_gas/RS_air

  double ppm_log = (log10(ratio)-b)/m; //Get ppm value in linear scale
according to the the ratio value
  double ppm = pow(10, ppm_log); //Convert ppm value to log scale

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // We can send any value at any time.
  // We cannot send more that 10 values per second.
  Blynk.virtualWrite(V5, t);
  Blynk.virtualWrite(V6, h);
  Blynk.virtualWrite(V7, ppm);
  Blynk.virtualWrite(V8, ppm1);
  Blynk.virtualWrite(V9, ppm2);
}

void setup()
{
  // Debug console
  Serial.begin(9600);
  s.begin(57600);

  Blynk.begin(auth, ssid, pass);
  // You can also specify server:
  //Blynk.begin(auth, ssid, pass, "blynk-cloud.com", 8442);
  //Blynk.begin(auth, ssid, pass, IPAddress(192,168,1,100), 8442);

  pinMode(16,INPUT);
  delay(10);
  dht.begin();

  // Setup a function to be called every second
  timer.setInterval(1000L, sendSensor);
}

void loop()
{
  Blynk.run();
  timer.run();
}
```

Serial Monitor Output:



Thingspeak IoT Dashboard and the Blynk App:



The Thingspeak IoT Dashboard can be viewed on both desktop as well as a smartphone.

The Designed Machine Learning Model:

Using the raw concentrations of SO_2, NO_2, SPM (Suspended Particulate Matter) and RSPM (Respirable Suspended Particulate Matter) we calculate their respective Pollutant Indices.

```
def calculate_si(so2):
    si=0
    if (so2<=40):
     si= so2*(50/40)
    if (so2>40 and so2<=80):
     si= 50+(so2-40)*(50/40)
    if (so2>80 and so2<=380):
     si= 100+(so2-80)*(100/300)
    if (so2>380 and so2<=800):
     si= 200+(so2-380)*(100/800)
    if (so2>800 and so2<=1600):
     si= 300+(so2-800)*(100/800)
    if (so2>1600):
     si= 400+(so2-1600)*(100/800)
    return si
data['si']=data['so2'].apply(calculate_si)
df= data[['so2','si']]
```

```
def calculate_ni(no2):
    ni=0
    if(no2<=40):
     ni= no2*50/40
    elif(no2>40 and no2<=80):
     ni= 50+(no2-14)*(50/40)
    elif(no2>80 and no2<=180):
     ni= 100+(no2-80)*(100/100)
    elif(no2>180 and no2<=280):
     ni= 200+(no2-180)*(100/100)
    elif(no2>280 and no2<=400):
     ni= 300+(no2-280)*(100/120)
    else:
     ni= 400+(no2-400)*(100/120)
    return ni
data['ni']=data['no2'].apply(calculate_ni)
df= data[['no2','ni']]
```

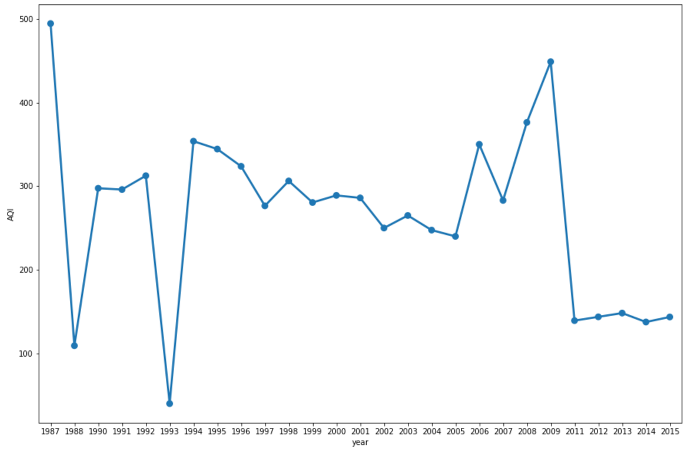
```
def calculate_spi(spm):
    spi=0
    if(spm<=50):
     spi=spm
    if(spm<50 and spm<=100):
     spi=spm
    elif(spm>100 and spm<=250):
     spi= 100+(spm-100)*(100/150)
    elif(spm>250 and spm<=350):
     spi=200+(spm-250)
    elif(spm>350 and spm<=450):
     spi=300+(spm-350)*(100/80)
    else:
     spi=400+(spm-430)*(100/80)
    return spi
data['spi']=data['spm'].apply(calculate_spi)
df= data[['spm','spi']]
```

```
def calculate_(rspm):
    rpi=0
    if(rpi<=30):
     rpi=rpi*50/30
    elif(rpi>30 and rpi<=60):
     rpi=50+(rpi-30)*50/30
    elif(rpi>60 and rpi<=90):
     rpi=100+(rpi-60)*100/30
    elif(rpi>90 and rpi<=120):
     rpi=200+(rpi-90)*100/30
    elif(rpi>120 and rpi<=250):
     rpi=300+(rpi-120)*(100/130)
    else:
     rpi=400+(rpi-250)*(100/130)
    return rpi
data['rpi']=data['rspm'].apply(calculate_si)
df= data[['rspm','rpi']]
```

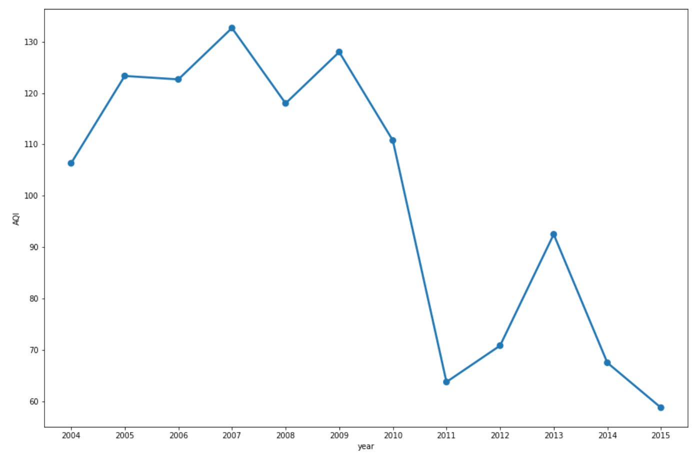
Using the above calculated Pollution Indices, we calculate the Air Quality Index (AQI) which is calculated as per the Government of India standards.

```
def calculate_aqi(si,ni,spi,rpi):
    aqi=0
    if(si>ni and si>spi and si>rpi):
     aqi=si
    if(spi>si and spi>ni and spi>rpi):
     aqi=spi
    if(ni>si and ni>spi and ni>rpi):
     aqi=ni
    if(rpi>si and rpi>ni and rpi>spi):
     aqi=rpi
    return aqi
data['AQI']=data.apply(lambda x:calculate_aqi(x['si'],x['ni'],x['spi'],x['rpi']),axis=1)
df= data[['sampling_date','state','si','ni','rpi','spi','AQI']]
df.head()
```

The AQI vs. Year plot is as follows:



For Delhi

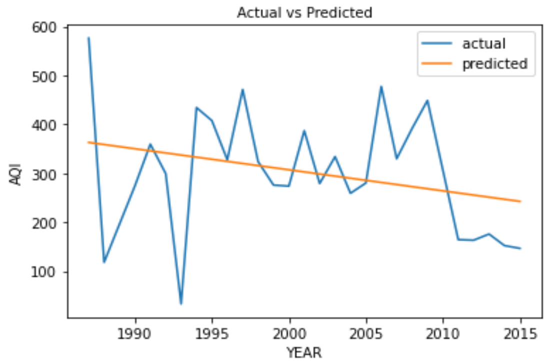


For Chennai

Applying Gradient Descent,

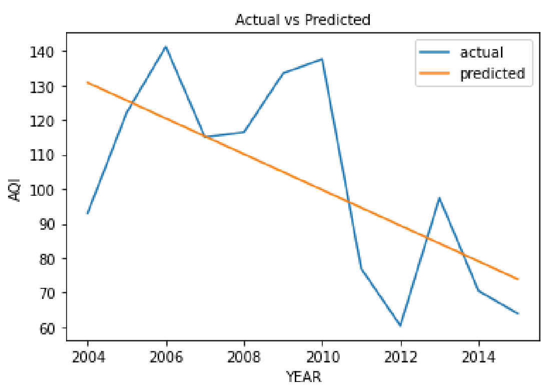
```
alpha = 0.01 #Step size
iterations = 3000 #No. of iterations
m = y.size #No. of data points
np.random.seed(4) #Setting the seed
theta = np.random.rand(2) #Picking random values to start with

def gradient_descent(x, y, theta, iterations, alpha):
    past_costs = []
    past_thetas = [theta]
    for i in range(iterations):
        prediction = np.dot(x, theta)
        error = prediction - y
        cost = 1/(2*m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)

    return past_thetas, past_costs
past_thetas, past_costs = gradient_descent(x, y, theta, iterations, alpha)
theta = past_thetas[-1]

#Printing the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))
```

For Delhi, we obtain the Actual vs. Predicted AQI as follows:



	year	AQI	Actual	Predicted
28	2015	101.258882	101.258882	197.637906
27	2014	102.785280	102.785280	197.818770
26	2013	106.729246	106.729246	197.999634
25	2012	99.696254	99.696254	198.180498
24	2011	117.824783	117.824783	198.361362
23	2010	188.283360	188.283360	198.542226
22	2009	221.368166	221.368166	198.723089
21	2008	214.378174	214.378174	198.903953
20	2007	211.160807	211.160807	199.084817
19	2006	219.623267	219.623267	199.265681
18	2005	207.546049	207.546049	199.446545
17	2004	164.661496	164.661496	199.627409
16	2003	163.875510	163.875510	199.808272
15	2002	149.706871	149.706871	199.989136
14	2001	205.138247	205.138247	200.170000
13	2000	195.772377	195.772377	200.350864
12	1999	225.439218	225.439218	200.531728
11	1998	234.740657	234.740657	200.712591
10	1997	220.903571	220.903571	200.893455
9	1996	216.850189	216.850189	201.074319
8	1995	227.102233	227.102233	201.255183
7	1994	205.636343	205.636343	201.436047
6	1993	65.754613	65.754613	201.616911
5	1992	177.485106	177.485106	201.797774
4	1991	238.060052	238.060052	201.978638
3	1990	239.071032	239.071032	202.159502
2	1989	236.513310	236.513310	202.340366
1	1988	211.076502	211.076502	202.521230
0	1987	242.438652	242.438652	202.702094

For Chennai, we obtain the Actual vs. Predicted AQI as follows:



	year	AQI	Actual	Predicted
11	2015	63.959596	63.959596	73.895559
10	2014	70.470657	70.470657	79.070912
9	2013	97.416667	97.416667	84.246265
8	2012	60.412286	60.412286	89.421618
7	2011	76.930233	76.930233	94.596971
6	2010	137.656463	137.656463	99.772324
5	2009	133.511905	133.511905	104.947676
4	2008	116.470238	116.470238	110.123029
3	2007	115.049242	115.049242	115.298382
2	2006	141.247549	141.247549	120.473735
1	2005	122.186508	122.186508	125.649088
0	2004	93.039907	93.039907	130.824441

The Website Designed:

The website that is designed periodically receives the values obtained from the Thingspeak IoT Cloud and visualizes the values in real-time. The website also contains information regarding the general information pertaining to air pollution, and also describes the various air pollutants so that the end users can gain insight from it.

The website is hosted in the Google Firebase Cloud and is accessible round the world. It is designed using HTML and CSS.

HTML Source Code of the Index Page:

HTML is used to design the basic structure of the webpage.

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<meta charset="utf-8">
<title>IoT PROJECT</title>
<link rel="stylesheet" type = "text/css" href="style1.css">
<script src="https://cdn.jsdelivr.net/npm/chart.js@3.2.0/dist/chart.min.js"> </script>
</head>
<body>
<h1>AIR POLLUTION MONITORING SYSTEM</h1>
<div class="topnav">
<a class="active" href="index.html">Home</a>
<a class href="Air_pollution1.html">Temperature</a>
<a class href="Air_pollution2.html">Humidity</a>
<a class href="Air_pollution3.html">CO</a>
<a class href="Air_pollution4.html">Nitrogen Based</a>
<a class href="Air_pollution5.html">Other Gases</a>
</div>
   <h2>AIR POLLUTTION</h2>
   <div class="home1">
     <h3>Introduction</h3>
     <p>Air pollution is a mix of hazardous substances from both human-made and natural sources. Air pollution is a familiar environmental
        health hazard. We know what we're looking at when brown haze settles over a city, exhaust billows across a busy highway, or a plume rises from a smokestack.
        Some air pollution is not seen, but its pungent smell alerts you.</p>
     <p>From smog hanging over cities to smoke inside the home, air pollution poses a major threat to health and climate.
        The combined effects of ambient (outdoor) and household air pollution cause about seven million premature deaths every year, largely as a result of increased
        mortality from stroke, heart disease, chronic obstructive pulmonary disease, lung cancer and acute respiratory infections.</p>
     <button class="b1" type="button" onclick="document.location='https://climatekids.nasa.gov/air-pollution/'">Causes of Air pollution</button>
   </div>
   <div class="home2">
     <h3>Health and Environmental Effects</h3>
     <p>Even healthy people can experience health impacts from polluted air including respiratory irritation or breathing difficulties during exercise or
        outdoor activities. Your actual risk of adverse effects depends on your current health status, the pollutant type and concentration, and the length of your
        exposure to the polluted air. It can also cause other health problems including:</p>
     <ul>
       <li>Aggravated respiratory disease such as emphysema, bronchitis and asthma</li>
       <li>Lung damage, even after symptoms such as coughing or a sore throat disappear</li>
       <li>Wheezing, chest pain, dry throat, headache or nausea</li>
       <li>Reduced resistance to infections</li>
       <li>Increased fatigue and weakened athletic performance</li>
     </ul>
     <button class="b1" type="button" onclick="document.location='https://www.tecamgroup.com/effects-air-pollution-environment/'">Effects of Air pollution</button>
   </div>
</body>
</html>
```

HTML Source Code of the Temperature Measurement Page:

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<meta charset="utf-8">
<title>IoT PROJECT</title>
<link rel="stylesheet" type = "text/css" href="style1.css">
<script src="https://cdn.jsdelivr.net/npm/chart.js@3.2.0/dist/chart.min.js"> </script>
</head>
<body>
<h1>AIR POLLUTION MONITORING SYSTEM</h1>
<div class="topnav">
<a class="active" href="index.html">Home</a>
<a class href="Air_pollution1.html">Temperature</a>
<a class href="Air_pollution2.html">Humidity</a>
<a class href="Air_pollution3.html">CO</a>
<a class href="Air_pollution4.html">Nitrogen Based</a>
<a class href="Air_pollution5.html">Other Gases</a>
</div>
<div class="home1">
  <h3>Temperature Measurement</h3>
  <iframe width="450" height="260" style="border: 2px solid #000000;"
   src="https://thingspeak.com/channels/1371486/charts/1?bgcolor=%23ffffff&color=%23d62020&dynamic=true&results=10&type=line"></iframe>
</div>
<div class="home2">
  <h3>Temperature</h3>
  <p>Temperature is a physical quantity that expresses hot and cold. It is the manifestation of thermal energy, present in all matter, which is the source
     of the occurrence of heat, a flow of energy, when a body is in contact with another that is colder or hotter. </p>
  <p>In India, temperatures typically range from -2 °C to 40 °C, but can reach 47 °C in summer and -4 °C in winter.</p>
  <button class="b1" type="button" onclick="document.location='https://www.mapsofindia.com/maps/india/annualtemperature.htm'">Temperature in India</button>
</div>

</body>
</html>
```

The Highlighted Code is the graph visualization that is obtained from the Thingspeak IoT Cloud.

CSS Source Code:

CSS is used for styling of the webpages like the background images, color, etc. which improves the readability and the user experience of the website.

```css
body{
  background-image: url("https://wallpapercave.com/wp/wp6100414.jpg");
  background-size: cover;
  background-repeat: no-repeat;
}
h1{
  padding-top: 10px;
  padding-bottom: 10px;
  text-align: center;
}
h2{
  padding-top: 10px;
  padding-bottom: 10px;
  text-align: center;
}
h3{
  padding-top: 10px;
  padding-bottom: 10px;
  text-align: center;
  margin: 30px;
}
p{
  padding: 10px;
}
body {
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
}
```

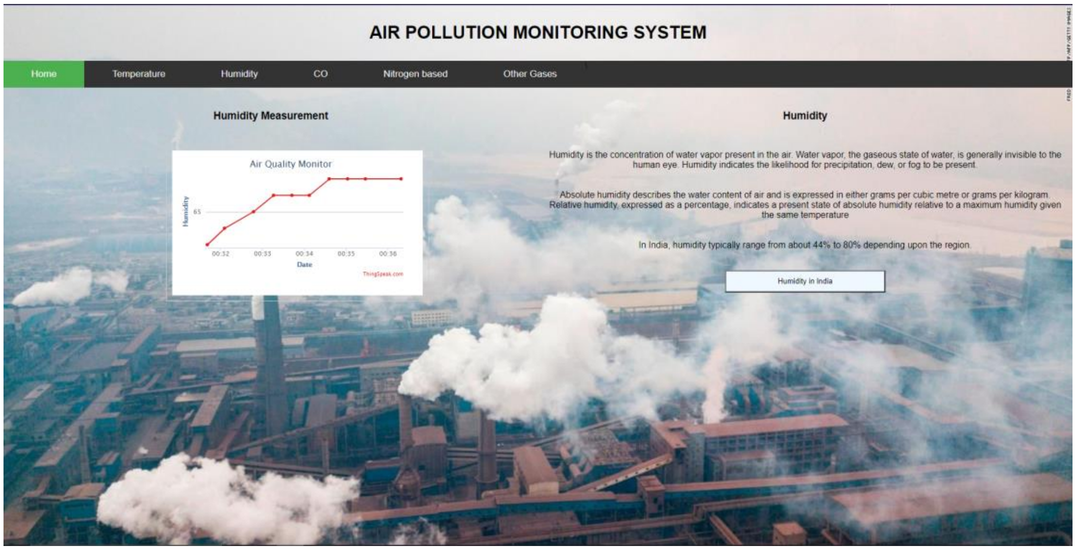
```css
.home1{
  width: 50%;
  float: left;
}
.home2{
  width: 50%;
  float: right;
}
.topnav {
  overflow: hidden;
  background-color: #333;
}

.topnav a {
  float: left;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 50px;
  text-decoration: none;
  font-size: 17px;
}
```

```css
.topnav a:hover {
  background-color: #ddd;
  color: black;
}

.topnav a.active {
  background-color: #4CAF50;
  color: white;
}
.topnav a.login {
  float: right;
  background-color: #4CAF50;
  color: white;
}
```

```css
.b1{
  background-color: rgb(240,248,255);
  color: black;
  text-align: center;
  alignment: centre;
  width: 80%;
  margin: 10px;
  height: 45px;
  border: 1px, solid;
}
iframe{
  padding: 10px;
  margin-left: 10%;
  align-items: center;
}
img{
  height: 400px;
  width: 400px;
  margin-left: 10%;
}
```

Website Visualization (2 out of 6 webpages have been shown here):

Index Page



Humidity Measurement Page

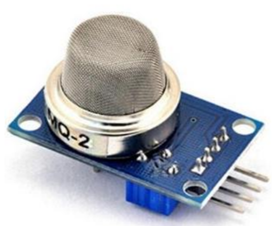
Detailed pictures of the Hardware:

Arduino UNO

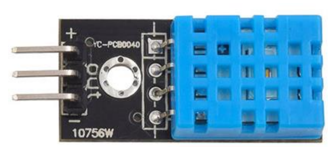
Node MCU

MQ7 – CO Sensor

MQ135 –Air quality Sensor



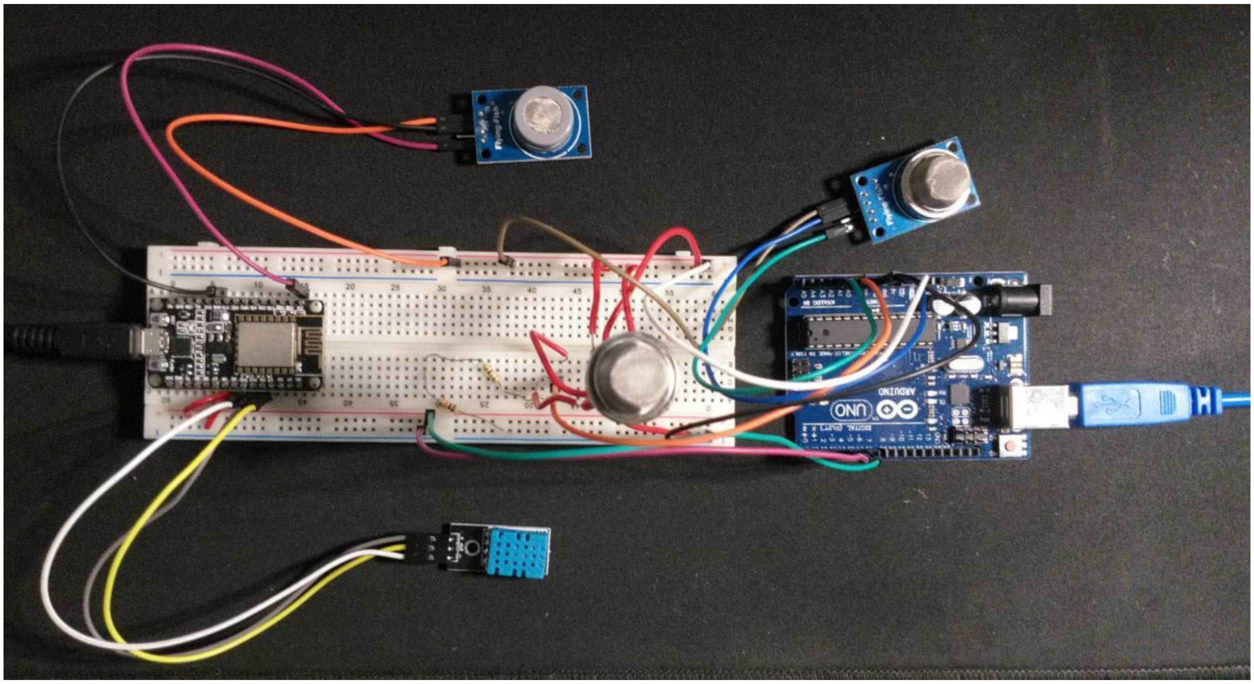
MQ2 – CH4 Sensor



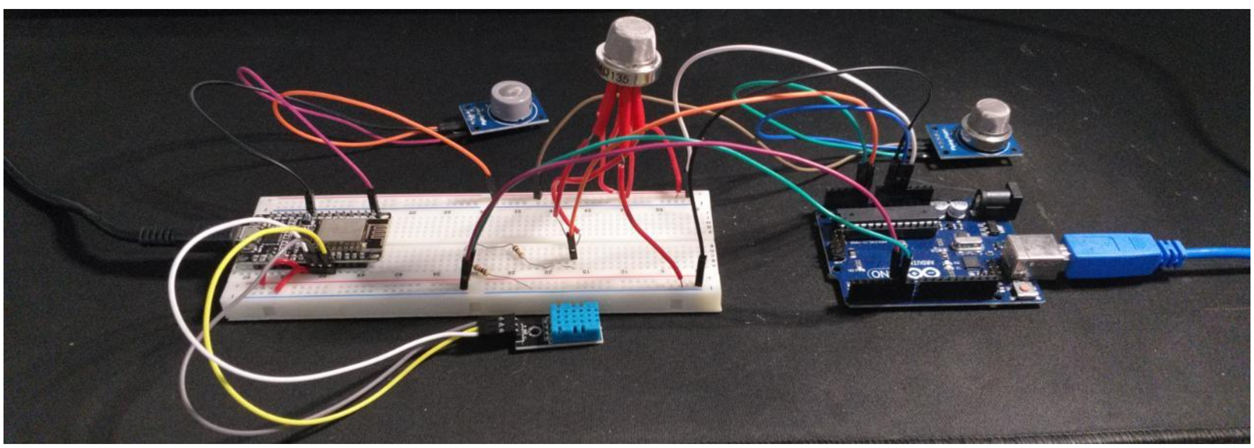
DHT11 – Temperature and Humidity Sensor

Detailed pictures of the Software:

Hardware Circuit Developed:



Top View



Side View

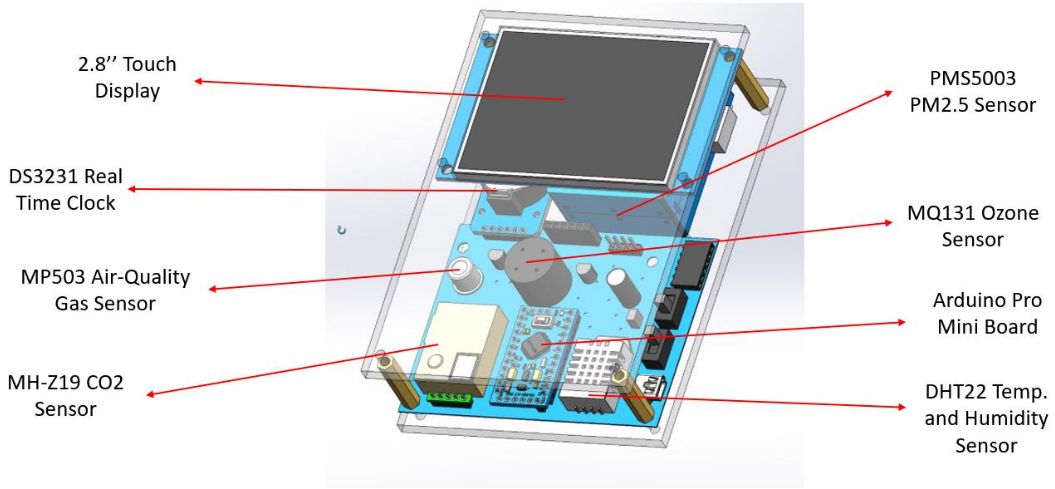
Real Time Applications:

- The readings collected by the sensors will be displayed on an interactive website.
- This will help the common people monitor the levels of air pollution in their neighborhood.
- The Trained Model can be used to gain insights on location specific future air quality levels.

- The government can also use the data to implement dynamic pollution control measures. (Eg: Like Delhi's odd-even rule but over a localized area and on particular days rather than over weeks)

Future Scope:

- The sensors can be miniaturized and integrated into smartphones.
- As a result, millions of datapoints can be generated for thousands of locations around the world.
- This can help to create better predictive models to estimate future patterns of air pollution across the world.
- With this information, governments can take more concrete steps to reduce air pollution like constructing Carbon Capture Systems (CCS) and rezoning industries in highly polluted regions.



The following CAD Design Model shows the envisioned miniaturized solution of the Air pollution monitoring system which is compact as well as portable.

Contribution of each student:

a) V. Shri Sarvesh (ECE – 108118109)

- Was involved in doing the hardware testing, calibration and interfacing of the various sensors used for data collection. Also worked on sending sensor data from the Arduino UNO to the Node MCU, and from the Node MCU to the IoT Cloud.

b) Balaji K S (ECE – 108118021)

- Was involved in designing the website using HTML and CSS, and hosting the website in the cloud. Also worked on the communication between the website and the IoT cloud, to enable the dynamic update of the values on the web interface.

c) Adithya Sineesh (ECE – 108118005)

- Was involved in Training and Development of the Machine Learning model used to visualize the air quality level over the years for various cities, and predict the future air quality level using the values obtained from the sensor data from the IoT Cloud.

References:

https://datasheetspdf.com/pdf/605076/Hanwei/MQ-135/1
https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf
https://www.pololu.com/file/0J309/MQ2.pdf
https://www.electronicoscaldas.com/datasheet/DHT11_Aosong.pdf
https://arduinojson.org/v6/doc/
https://www.mathworks.com/help/thingspeak/
https://how2electronics.com/iot-air-pollution-monitoring-esp8266/
http://help.blynk.cc/en/articles/512056-how-to-display-any-sensor-data-in-blynk-app
https://firebase.google.com/docs/build
https://www.researchgate.net/publication/337780520_IOT_based_Air_Quality_Monitoring_System_Using_MQ135_and_MQ7_with_Machine_Learning_Analysis
https://www.hindawi.com/journals/js/2020/8749764/
https://www.ijitee.org/wp-content/uploads/papers/v8i9S2/I11160789S219.pdf

****