

**REPORT FOR INDUSTRIAL TRAINING  
(EC781)  
ON  
IOT AND ARTIFICIAL INTELLIGENCE &  
ITS APPLICATION**



**Submitted By:**

**Name: Pralay Roy Chowdhury**

**University Roll Number: 13000320095**

**Year: 4<sup>th</sup>**

**Semester: 7<sup>th</sup>**

**Section: B**

**Department: Electronics and Communication  
Engineering**

**College: Techno Main Salt Lake, Kolkata  
(Affiliated to Maulana Abul Kalam Azad University of Technology)**

# CONTENTS

**Pg.No.**

## **Section 1: Artificial Intelligence and Machine Learning**

1. Project Objective.....	3
2. Component Used.....	4
3. Implementation.....	5
4. Output.....	6
5. Discussion.....	9

## **Section 2: Internet of Things**

1. Module 1: Introduction to Arduino Uno.....	10
2. Module 2: Bus Communication.....	12
3. Module 3: Node MCU.....	15
4. Module 4: Node MCU-based IoT.....	17
5. Module 5: Introduction to Raspberry Pi.....	22
6. Module 6: Raspberry Pi-based IoT.....	23

# Project Objective

Air pollution is a growing concern worldwide, and it has serious implications on human health, the environment, and the economy. In this project, we explore the prediction of the Air Quality Index (AQI) using the Random Forest algorithm. AQI is a measure of air pollution that is used to communicate the health risks associated with breathing polluted air. We use historical data collected from various air quality monitoring stations in a city and apply the Random Forest algorithm to predict AQI. This study aims to predict the AQI using machine learning algorithms.

AQI is a numerical value ranging from 0 to 500, and it is calculated based on the levels of major air pollutants such as particulate matter (PM), ozone (O<sub>3</sub>), nitrogen dioxide (NO<sub>2</sub>), and sulphur dioxide (SO<sub>2</sub>).

A Random Forest Regressor is a machine-learning algorithm that belongs to the ensemble learning category. It is an extension of the Random Forest algorithm, which is primarily used for classification tasks. In the case of a Random Forest Regressor, the algorithm is designed for regression problems, where the goal is to predict a continuous output variable rather than a categorical one.

- Air quality forecasting uses machine learning to predict the air quality index for a given region.
- To achieve better performance than the standard regression models.
- Our goal is for the model to accurately predict the Air Quality Index for India as a whole.
- By forecasting the Air Quality Index, we can track the main pollutants causing pollution and the locations across India that are severely affected by pollutants.

# Component Used

- Jupyter Notebook
- Python Libraries: Pandas, NumPy, Matplotlib.pyplot, Seaborn
- Data Set

## **Jupyter Notebook**

Jupyter Notebook is an open-source web application that allows to create and sharing of documents that contain live code, equations, visualizations, and explanatory text. It is a popular tool used for interactive data analysis, scientific computing, and machine learning.

## **NumPy**

NumPy (Numerical Python) is a powerful open-source library for the Python programming language, designed for efficient numerical computations and handling of large, multi-dimensional arrays and matrices. It provides a collection of high-level mathematical functions and operations that operate on these arrays, making it a fundamental building block for many scientific computing and data analysis tasks.

## **Pandas**

Pandas is a popular open-source data manipulation and analysis library for the Python programming language. It provides data structures and functions to efficiently handle and analyze structured data, such as numerical tables and time series data.

## **Matplotlib.pyplot**

In Python, matplotlib.pyplot, often abbreviated as plt, is a module within the matplotlib library that provides a collection of functions to create various types of plots and visualizations. Matplotlib is a widely used 2D plotting library for data visualization in Python.

## **Seaborn**

Seaborn is a statistical data visualization library in Python that is built on top of Matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn is particularly useful for creating complex visualizations with concise syntax and default themes that are aesthetically pleasing.

## **Data Set**

In our dataset for the prediction of Air Quality Index of a region we have used :

i. City, ii. Date, iii. PM2.5, iv. PM10, v. NO, vi. NO2, vii. NOx, viii. NH3, ix. CO, x. SO2, xi. O3, xii. Benzene, xiii. Toluene, xiv. Xylene, xv. AQI, xvi. AQI Bucket

# Implementation

## **1.Data sources**

Collection of data on various air quality parameters such as particulate matter (PM10, PM2.5), sulfur dioxide (SO<sub>2</sub>), nitrogen dioxide (NO<sub>2</sub>), ozone (O<sub>3</sub>), carbon monoxide (CO), etc. for a given location at different times. Import the file on Jupyter notebook.

## **2.Preprocessing of data**

Cleaning the data and removing any missing or inconsistent values. There are various techniques used in data preprocessing i.e. data cleaning, data reduction etc. The overall goal of data preprocessing is to ensure that the data is ready for analysis or machine learning and will produce accurate and meaningful results.

## **3.Parameter Selection and Calculation**

Selecting the relevant parameters from the dataset that can impact air quality. This can be done using statistical techniques or domain knowledge. Take the parameters and calculate the subindexes. Then calculate the AQI and specify the range for AQI bucket.

## **4.Train-Test Split**

Train-test split is a technique used in machine learning to evaluate the performance of a model on unseen data. The process involves splitting a dataset into two parts: a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate the model's performance. The goal is to train a model that can generalize well to new, unseen data.

## **5.Hyperparameter Tuning:**

Hyperparameter tuning is an essential step in optimizing the performance of a Random Forest model for air quality index prediction.

- 1) Split data into training and validation sets.
- 2) Define hyperparameters to tune.
- 3) Choose a performance metric to optimize.
- 4) Use grid search to try out all possible combinations of hyperparameters.
- 5) Perform k-fold cross-validation on each combination of hyperparameters.
- 6) Select the hyperparameters that give the best performance on the validation set.
- 7) Test the model with the selected hyperparameters on a new test dataset to evaluate its performance in real-world scenarios.

## **6.Model Evaluation**

Random Forest is a popular ML algorithm used for air quality index prediction. It is important to evaluate its performance using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

# OUTPUT

```
In [20]: plt.figure(figsize=(12,10))
sns.heatmap(df.corr(),cmap='coolwarm',annot=True);
```

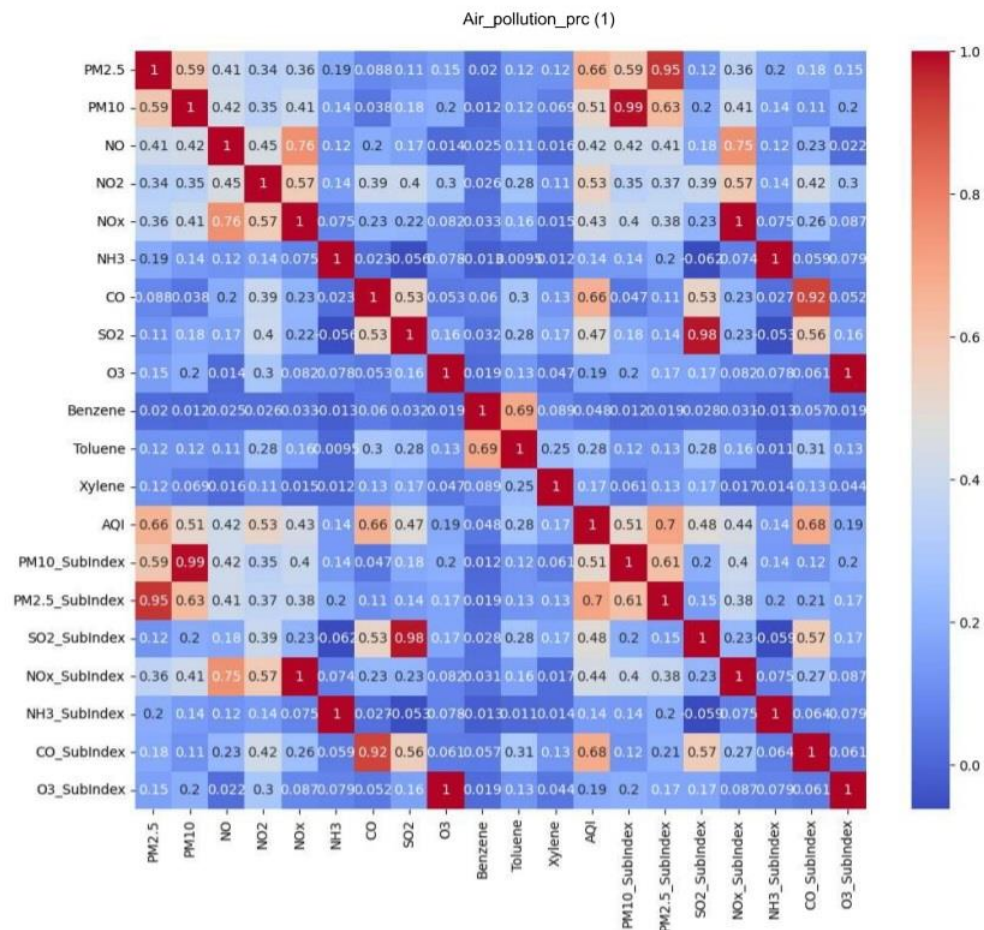
C:\Users\abhir\AppData\Local\Temp\ipykernel\_29140\2154488151.py:2: FutureWarning: The default value of numeric\_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric\_only to silence this warning.

```
sns.heatmap(df.corr(),cmap='coolwarm',annot=True);
```

localhost:8888/nbconvert/html/Air\_pollution\_prc (1).ipynb?download=false

7/13

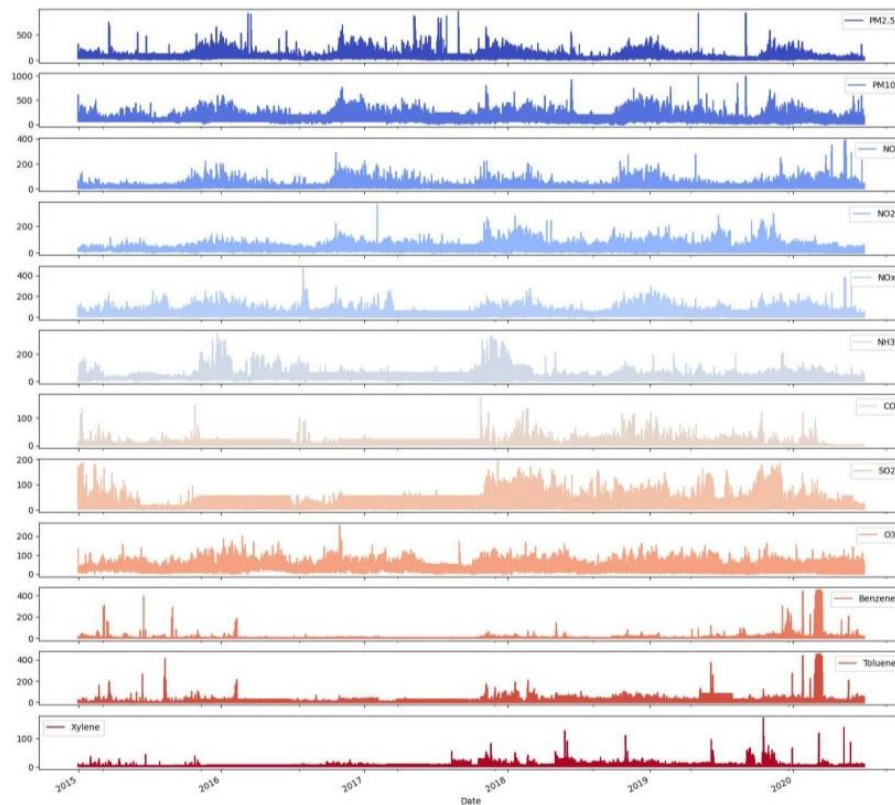
11/28/23, 1:43 PM



```
In [21]: pollutants = ['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3', 'Benzene',
df_city_day = df_city_day[pollutants]

print('Distribution of different pollutants in last 5 years')
df_city_day.plot(kind='line',figsize=(18,18),cmap='coolwarm',subplots=True,fontsize=16)
```

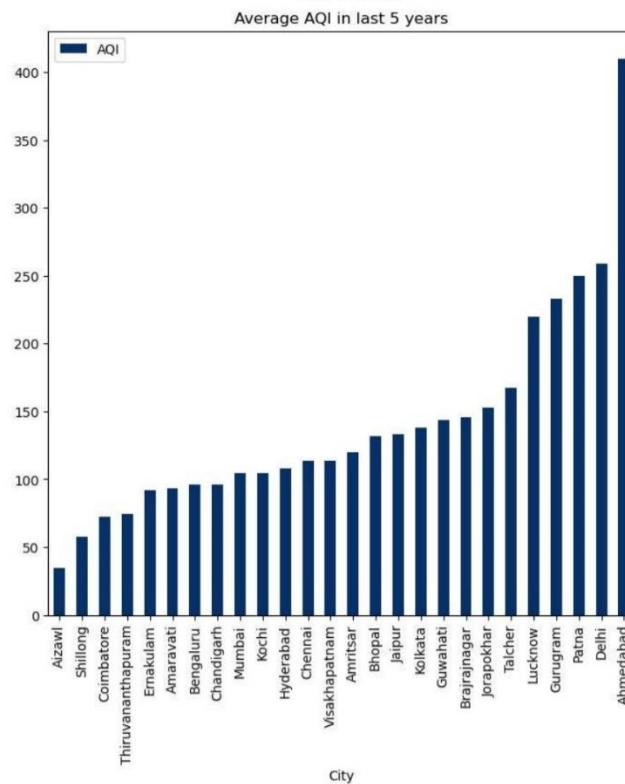
Distribution of different pollutants in last 5 years



```
In [22]: df[['City', 'AQI']].groupby('City').mean().sort_values('AQI').plot(kind='bar', cmap='Blu
plt.title('Average AQI in last 5 years');
```

11/28/23, 1:43 PM

Air\_pollution\_prc (1)



```
In [23]: final_df= df[['AQI', 'AQI_Bucket']].copy()
final_df
```

```
Out[23]:
```

	AQI	AQI_Bucket
Date		
2015-01-01	149.0	Moderate
2015-01-02	123.0	Moderate
2015-01-03	300.0	Poor
2015-01-04	123.0	Moderate
2015-01-05	329.0	Very Poor
...	...	...
2020-06-27	41.0	Good
2020-06-28	70.0	Satisfactory
2020-06-29	68.0	Satisfactory
2020-06-30	54.0	Satisfactory
2020-07-01	50.0	Good

29531 rows x 2 columns

```
In [24]: final_df['AQI_Bucket'].unique()
Out[24]: array(['Moderate', 'Poor', 'Very Poor', 'Severe', 'Satisfactory', 'Good'],
      dtype=object)
In [25]: #final_df = pd.get_dummies(final_df)
      final_df['AQI_Bucket'] = final_df['AQI_Bucket'].map({'Good':0, 'Satisfactory':1, 'Moderate':2, 'Poor':3, 'Very Poor':4, 'Severe':5})
      final_df.head()
```

```
Out[25]:
```

	AQI	AQI_Bucket
Date		
2015-01-01	149.0	2
2015-01-02	123.0	2
2015-01-03	300.0	3
2015-01-04	123.0	2
2015-01-05	329.0	4

```
In [29]: from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
      print(accuracy_score(y_test, y_pred))
      print(classification_report(y_test, y_pred))
      print(confusion_matrix(y_test, y_pred))
```

```
1.00
              precision    recall  f1-score   support

    0         1.00      1.00      1.00         388
    1         1.00      1.00      1.00        2397
    2         1.00      1.00      1.00        2608
    3         1.00      1.00      1.00         805
    4         1.00      1.00      1.00         861
    5         1.00      1.00      1.00         324

 accuracy
macro avg         1.00      1.00      1.00         7383
weighted avg         1.00      1.00      1.00         7383

[[ 388   0   0   0   0   0]
 [   0 2397   0   0   0   0]
 [   0   0 2608   0   0   0]
 [   0   0   0 805   0   0]
 [   0   0   0   0 861   0]
 [   0   0   0   0   0 324]]
```

```
In [30]: from sklearn.ensemble import RandomForestRegressor
```

```
In [31]: X = final_df.iloc[:,[0]].values
      y = final_df.iloc[:,[1]].values
```

```
In [32]: regressor = RandomForestRegressor(n_estimators = 100, random_state=0)
      regressor.fit(X,y)
```



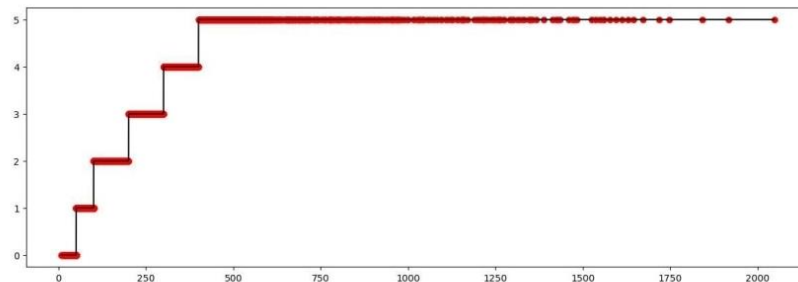
```
C:\Users\abhir\AppData\Local\Temp\ipykernel_29140\3226747133.py:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
regressor.fit(X,y)
```

```
Out[32]: RandomForestRegressor
RandomForestRegressor(random_state=0)
```

```
In [33]: Y_pred = regressor.predict(X)
```

```
In [34]: X_grid = np.arange(min(X),max(X),0.01)
X_grid=X_grid.reshape(len(X_grid),1)
```

```
In [35]: plt.figure(figsize=(15,5))
plt.scatter(X,y,color="red")
plt.plot(X_grid,regressor.predict(X_grid),color="black")
plt.show()
```



```
In [36]: from sklearn.metrics import mean_absolute_error as mae, mean_squared_error as mse, r2_score
```

```
In [37]: print(f"MAE: {mae(y_test, y_pred)}")
MAE: -0.0
```

```
In [38]: print(f"MSE: {mse(y_test, y_pred)}")
MSE: -0.0
```

```
In [39]: print(f"R squared: {r2_score(y_test, y_pred)}")
R squared: -1.0
```

## Discussion

In conclusion, random forest is a powerful machine-learning algorithm that can be used for air quality index prediction. It is a popular method for its ability to handle complex, high-dimensional datasets and to identify important features for prediction. By using random forest to analyse various air quality parameters, such as temperature, humidity, and particulate matter concentrations, it is possible to accurately predict the air quality index at a given location and time. However, it is important to note that prediction accuracy can be affected by the quality and quantity of data used to train the model, as well as other external factors such as weather conditions and human activity.

# MODULE: 1

## Introduction of Arduino Uno

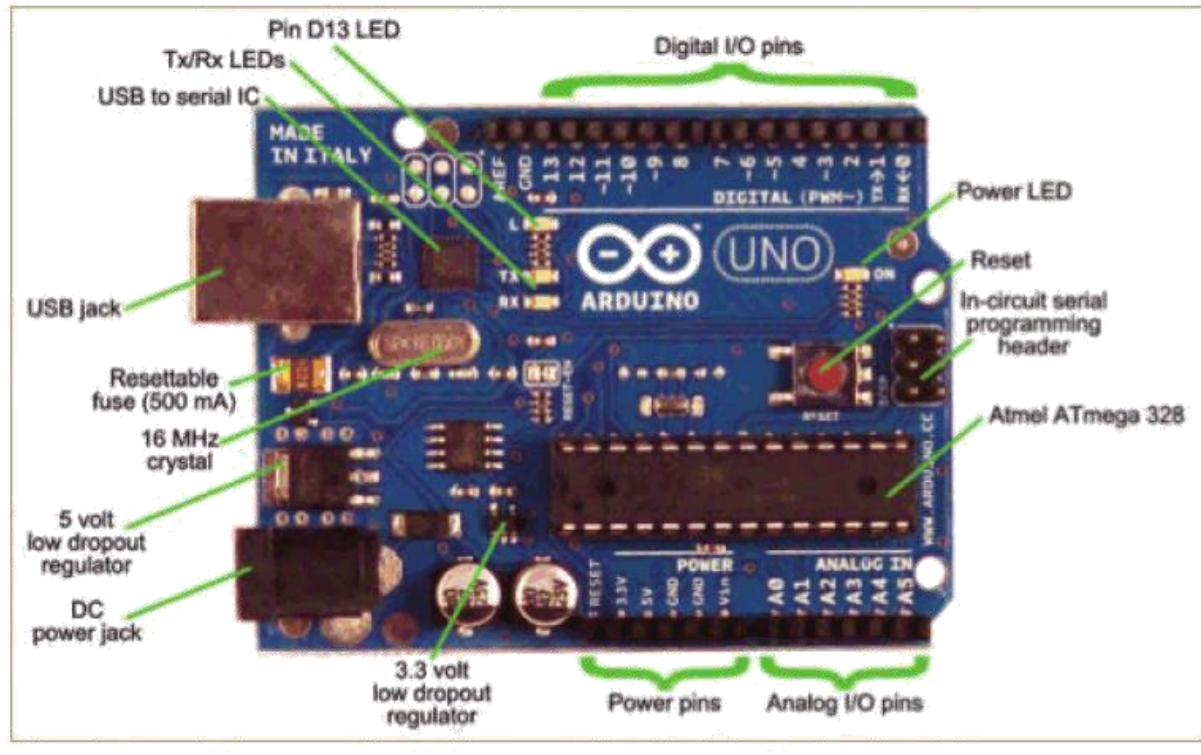


Fig No.1

Basically, the processor of the Arduino board uses the Harvard architecture where the program code and program data have separate memory.

The Atmega328 microcontroller has 32kb of flash memory, 2kb of SRAM 1kb of EPROM and operates with a 16MHz clock speed.

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

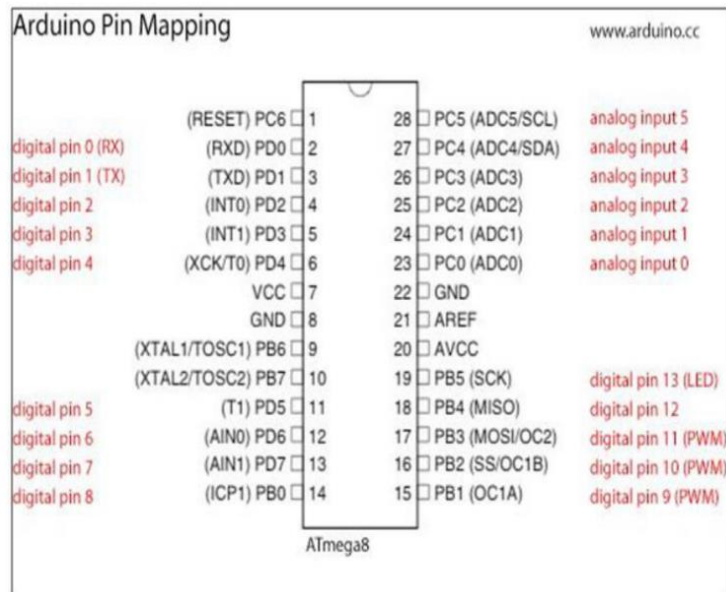
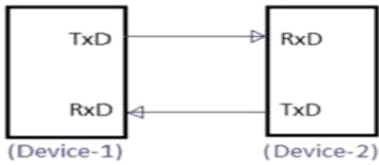
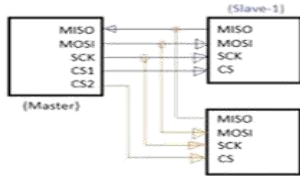
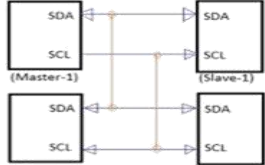


Fig No.2

- ✓ 8BIT AVR RISC MICROCONTROLLER
- ✓ Microcontroller ATmega 328
- ✓ Operating voltage: 5 V
- ✓ Input voltage recommended : 7-12 V
- ✓ Digital I/O Pins: 14 ( of which 6 PWM output)
- ✓ Analog input Pins:6
- ✓ DC current per I/O Pin: 40 mA
- ✓ DC current for 3.3 V Pin: 50 mA
- ✓ Flash memory : 32 kB (Atmega 328) of which 0.5 kB used by bootloader
- ✓ SRAM: 2 kB
- ✓ EEPROM: 1 kB
- ✓ Clock speed : 16 MHz
- ✓ 6 CHANNEL ,10 BIT ADC

# MODULE: 2

## Bus Communication

UART	SPI	I2C
Universal Asynchronous Receiver/Transmitter	Serial Peripheral Interface	Inter-Integrated Circuit
 <p>UART Interface Diagram</p>	 <p>SPI Interface Diagram</p>	 <p>I2C Interface Diagram</p>
<p>TxD: Transmit Data RxD: Receive Data</p>	<p>SCLK: Serial Clock MOSI: Master Output, Slave Input MISO: Master Input, Slave Output SS: Slave Select</p>	<p>SDA: Serial Data SCL: Serial Clock</p>
<p>As this is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.</p>	<p>Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps</p>	<p>I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also supports 10 Kbps and 1 Mbps.</p>
<p>Lower about 50 feet</p>	<p>highest</p>	<p>Higher</p>
<p>Asynchronous</p>	<p>Synchronous</p>	<p>Synchronous</p>

## Control an LED via Serial Communication:

### Code for Master Arduino:

```
#include <SoftwareSerial.h>
SoftwareSerial softSerial(10, 11);
void setup()
{
    softSerial.begin(9600);
    Serial.begin(9600);
}
void loop()
{
    softSerial.print(1);
    Serial.print(1);
    delay(5000);
    softSerial.print(0);
    Serial.print(0);
    delay(5000);
}
```

### Code for Slave Arduino:

```
#include <SoftwareSerial.h>
SoftwareSerial softSerial(10, 11);
char number = ' ';
int LED = 2;
void setup()
{
    softSerial.begin(9600);
    Serial.begin(9600);
}
void loop()
{
    if (softSerial.available())
    {
        char number = softSerial.read();
        if (number == '0')
        {
            digitalWrite(LED, LOW);
        }
        if (number == '1')
        {
            digitalWrite(LED, HIGH);
        }
        Serial.println(number);
    }
}
```

---

## Two-way communication between two Arduino using I2C:

### Sketch for Master:

```
#include <Wire.h>
void setup()
{
    Serial.begin(9600); /* begin serial comm. */
    Wire.begin(); /* join i2c bus as master */
    Serial.println("I am I2C Master");
}
void loop()
{
    Wire.beginTransmission(8); /* begin with device address 8 */
    Wire.write("Hello Slave"); /* sends hello string */
    Wire.endTransmission(); /* stop transmitting */
    Wire.requestFrom(8, 9); /* request & read data of size 9 from slave */
    while(Wire.available())
    {
        char c = Wire.read(); /* read data received from slave */
        Serial.print(c);
    }
    Serial.println();
    delay(1000);
}
```

### Sketch for Slave:

```
#include <Wire.h>
void setup()
{
    Wire.begin(8); /* join i2c bus with address 8 */
}
void loop() {
    delay(100);
}
//function that executes whenever data is received from master
void receiveEvent(int howMany)
{
    while (0 < Wire.available())
    {
        char c = Wire.read(); /* receive byte as a character */
        Serial.print(c); /*print the character*/
    }
    Serial.println(); /*to newline*/
}
//function that executes whenever data is requested from master
void requestEvent()
{
    Wire.write("Hi Master"); /*send string on request*/
}
```

## MODULE:3

### Node MCU



**Fig No. 3**

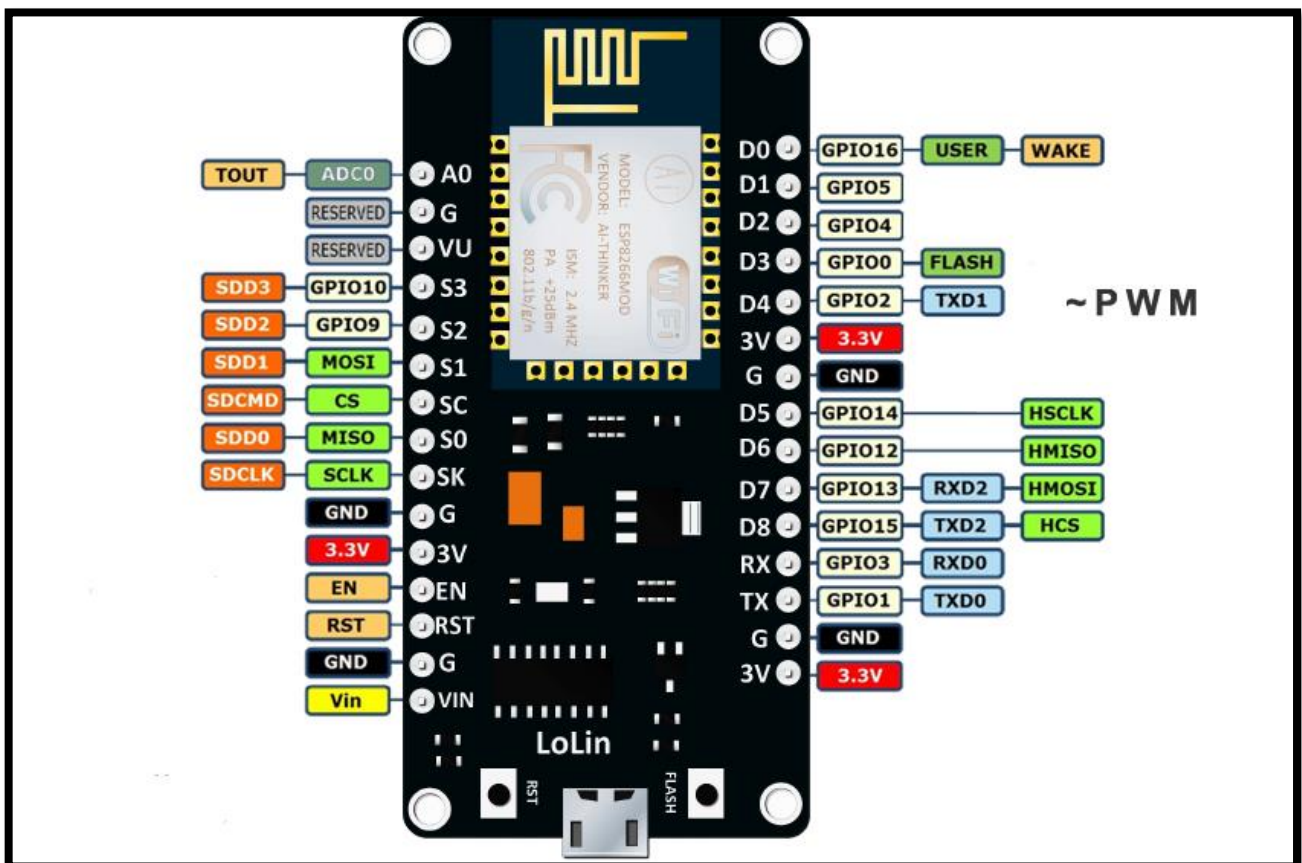
- The development board equips the ESP-12E module containing ESP8266 chip having Tensilica Xtensa® 32-bit LX106 RISC microprocessor which operates at 80 to 160 MHz adjustable clock frequency and supports RTOS.
- There's also 128 KB RAM and 4MB of Flash memory (for program and data storage) just enough to cope with the large strings that make up web pages, JSON/XML data, and everything we throw at IoT devices nowadays.
- The ESP8266 Integrates 802.11b/g/n HT40 Wi-Fi transceiver, so it can not only connect to a Wi-Fi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it.
- Power to the ESP8266 NodeMCU is supplied via the on- board Micro USB connector.
- You can also regulated 5V voltage source, the VIN pin can be used to directly supply current the ESP8266 and its peripherals.
- As the operating voltage range of ESP8266 is 3V to 3.6V, the board comes with a LDO(Low Dropout) voltage regulator to keep the voltage steady at 3.3V.



- It can reliably supply up to 600mA, which should be more than enough when ESP8266 pulls as much as 80mA during RF transmissions.

The ESP8266 NodeMCU has total 17 **GPIO** pins :

- **ADC channel** – A 10-bit ADC channel.
- **UART interface** – UART interface is used to load code serially.
- **PWM outputs** – PWM pins for dimming LEDs or controlling motors.
- **SPI, I2C & I2S interface** – SPI and I2C interface to hook up all sorts of sensors and peripherals.
- **I2S interface** – I2S interface if you want to add sound to your project.



**Fig No.4**



## MODULE: 4

### Node MCU Based IoT

4.1 IoT based Temperature and Humidity and Light Intensity Monitoring and control devices using Thing Speak cloud server.

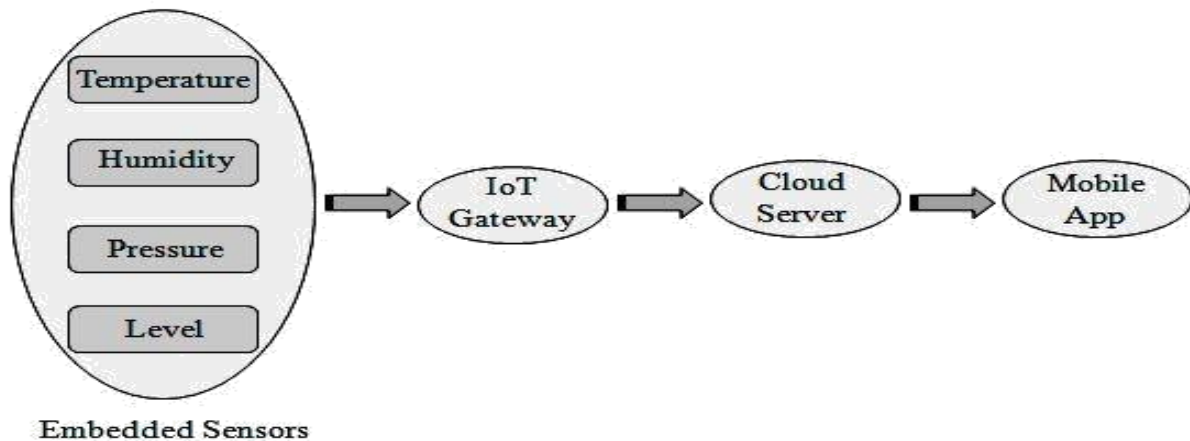


Fig No.5

```
#include <ESP8266WiFi.h>;
#include <WiFiClient.h>;
#include <ThingSpeak.h>;
#include <DHT.h>;
WiFiClient client;
const char* ssid = "Redmi"; //Your Network SSID
const char* password = "12345678"; //Your Network Password
unsigned long myChannelNumber = 2045473;
const char * myWriteAPIKey = "CDVUNQL9CCOTSUGB"; const char * myReadAPIKey =
"C86E152W4AMV3TGP";
#define DHTPIN D2 //pin where the dht11 is connected DHT
dht(DHTPIN, DHT11);
int temp;
int hum;
int light;
int led1 = D6;
int led2 = D7;
int led3 = D8;
int ldr = A0;
int val =0;
void setup()
{
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    dht.begin();
    ThingSpeak.begin(client);
    pinMode(led1,OUTPUT);
    pinMode(led2,OUTPUT);
```

```

        pinMode(led3,OUTPUT);
    }
    void loop()
    {
        float h = dht.readHumidity();
        float t = dht.readTemperature();
        Serial.println(h);
        Serial.println(t);
        val = analogRead(ldr);
        Serial.println(val);
        ThingSpeak.setField(1, h);
        ThingSpeak.setField(2, t);
        ThingSpeak.setField(3, val);
        ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
        hum = ThingSpeak.readIntField(myChannelNumber, 1, myReadAPIKey);
        temp = ThingSpeak.readIntField(myChannelNumber, 2, myReadAPIKey);
        light = ThingSpeak.readIntField(myChannelNumber, 3, myReadAPIKey);
        Serial.println(hum);
        if(hum<100)
        {
            digitalWrite(led1,HIGH);
        }
        else
        {
            digitalWrite(led1,LOW);
        }
        Serial.println(temp);
        if(temp<40)
        {
            digitalWrite(led2,HIGH);
        }
        else
        {
            digitalWrite(led2,LOW);
        }
        Serial.println(light);
        if(light<200)
        {
            digitalWrite(led3,HIGH);
        }
        else
        {
            digitalWrite(led3,LOW);
        }
        delay(15000);
    }

```

---

## 4.2 IoT based Light Intensity Monitoring and control devices using UBIDOTS cloud server (MQTT Protocol Based).

```
#include "UbidotsESPMQTT.h"
#include <ESP8266WiFi.h>
int ldrPin = A0;
/*****
Define Constants
*****/
#define TOKEN "BBFF-x5e4HGE4oiVIl6M34KiuJqD6qU3Cx" // Your Ubidots TOKEN
#define WIFINAME "Redmi" //Your SSID #define WIFIPASS "12345678" // Your Wifi Pass
Ubidots client(TOKEN);
int LED = D5;
char* variable1 = "led";
char* variable2 = "ldr";
/*****
Auxiliar Functions
*****/
void callback(char* topic, byte* payload, unsigned int length)
{
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");
    for (int i=0;i<length;i++)
    {
        Serial.print((char)payload[i]);
    }
    if ((char)payload[0]=='1')
    {
        digitalWrite(D5, HIGH);
    }
    else
    {
        digitalWrite(D5, LOW);
    }
    Serial.println();
}

void setup()
{
    put your setup code here, to run once: Serial.begin(115200);
    client.setDebug(true); // Pass a true or false bool value to activate debug messages
    client.wifiConnection(WIFINAME, WIFIPASS);
    client.begin(callback);
    pinMode(D5, OUTPUT);
    client.ubidotsSubscribe("MQTT",variable1);
}

void loop()
{
    if(!client.connected())
    {
        client.reconnect();
    }
}
```

```

    int ldrValue = analogRead(ldrPin);
    Serial.println(ldrValue);
    client.add(variable2, ldrValue);
    client.ubidotsPublish("MQTT");
    client.loop();
    delay(6000);
}

```

---

#### 4.3 IoT based Temperature and Humidity Monitoring and control devices using FireBase cloud server.

```

#include "FirebaseESP8266.h" // Install Firebase ESP8266 library #include <ESP8266WiFi.h>
#include <DHT.h> // Install DHT11 Library and Adafruit Unified Sensor Library
#define FIREBASE_HOST "https://subhajit-7af42-default-rtdb.asia-southeast1.firebaseio.com/" //Without http:// or https:// schemes
#define FIREBASE_AUTH "AIzaSyA6gMCNCc82nnRy5mfD51nr9ViPYc6LVWs"
#define WIFI_SSID "Redmi"
#define WIFI_PASSWORD "12345678"
#define DHTPIN D4 // Connect Data pin of DHT to D2
int led = D5; // Connect LED to D5
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
//Define FirebaseESP8266 data object
FirebaseData firebaseData;
FirebaseData ledData;
//FirebaseJson json;
void setup()
{
    Serial.begin(9600);
    dht.begin();
    pinMode(led,OUTPUT);
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
    Firebase.reconnectWiFi(true);
}
void sensorUpdate()
{
    float h = dht.readHumidity();
    float t = dht.readTemperature();
    if (isnan(h) || isnan(t) )
    {
        Serial.println(F("Failed to read from DHT sensor!"));
        return;
    }
    Serial.println(h);
    Serial.println(t);
    if (Firebase.pushFloat(firebaseData, "/FirebaseIOT/temperature", t))
    {
        Serial.println("PASSED");
    }
}

```

```

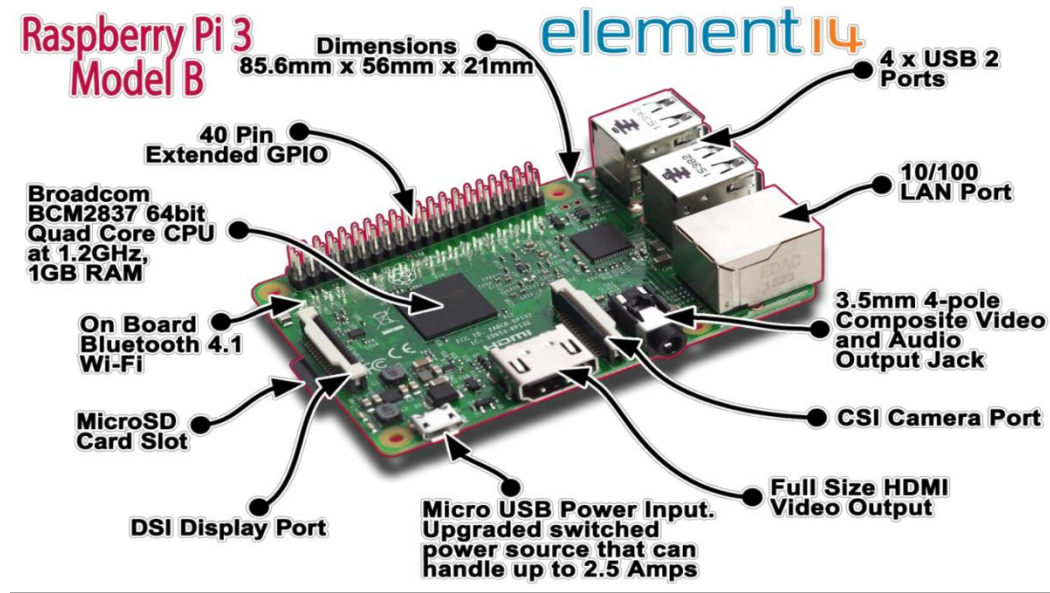
else
{
    Serial.println("FAILED");
}
if (Firebase.pushFloat(firebaseData, "/FirebaseIOT/humidity", h))
{
    Serial.println("PASSED");
}
else
{
    Serial.println("FAILED");
}
}
void loop()
{
    sensorUpdate();
    if (Firebase.getString(ledData, "/FirebaseIOT/led"))
    {
        Serial.println(ledData.stringData());
        if (ledData.stringData() == "1")
        {
            digitalWrite(led, HIGH);
        }
        else if (ledData.stringData() == "0")
        {
            digitalWrite(led, LOW);
        }
    }
    delay(100);
}

```

---

# MODULE: 5

## Introduction to Raspberry Pi



**Fig No. 6**

The Raspberry Pi is a small computer (which includes the processor, graphics card, and memory in a single package.), about the size of a credit card which was developed in UK by the Raspberry Pi Foundation. Name of Raspberry Pi is combination of Raspberry and Pi, where raspberry is fruit name and pi stand for Python.

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I <sup>2</sup> C)		DC Power 5v	04
05	GPIO03 (SCL1 , I <sup>2</sup> C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I <sup>2</sup> C ID EEPROM)		(I <sup>2</sup> C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

**Fig No. 7**

## **Control Led using RPi**

```
#import time
#import RPi.GPIO as GPIO
from RPi import GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(21,GPIO.OUT)
try:
    while True:
        GPIO.output(21,GPIO.HIGH)
        time.sleep(1)
        print("On")
        GPIO.output(21,GPIO.LOW)
        time.sleep(1)
        print("Off")
except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
```

---

# MODULE: 6

## Raspberry Pi Based IoT

### **6.1 IoT based Temperature and Humidity Monitoring using Thing Speak cloud server.**

```
import sys
import urllib.request
from time import sleep
import Adafruit_DHT as dht
# Enter Your API key here
myAPI = 'T0RGBR7N1ARD9XWB'
# URL where we will send the data, Don't change it
baseURL = 'https://api.thingspeak.com/update?api_key=%s' % myAPI
def DHT11_data():
    # Reading from DHT11 and storing the temperature and humidity
    humi, temp = dht.read_retry(dht.DHT11, 21)
return humi, temp
while True:
    try:
        humi, temp = DHT11_data()
        # If Reading is valid
        if isinstance(humi, float) and isinstance(temp, float):
            Formatting to two decimal places
            humi = '%.2f' % humi
            temp = '%.2f' % temp
            print(humi)
            print(temp)
            Sending the data to thingspeak
            conn = urllib.request.urlopen(baseURL + '&field1=%s&field2=%s' % (temp, humi))
            conn.read()
            Closing the connection conn.close()
        else:
            print ('Error')
            DHT11 requires 2 seconds to give a reading, so make sure to add delay of above 2 seconds.
            sleep(20)
    except:
        break
```



## 6.2 Human Motion Detection and post to UBIDOTS cloud server.

```
from gpiozero import MotionSensor
import json,requests
import time
PIR = MotionSensor(21)
url = "http://things.ubidots.com"
TOKEN = "BBFF-x5e4HGE4oiVil6M34KiuJqD6qU3Cx" # Put your TOKEN here
DEVICE_LABEL = "pir"
url = "{} /api/v1.6/devices/{}".format(url, DEVICE_LABEL)
headers = {"X-Auth-Token": TOKEN, "Content-Type": "application/json"}
#VARIABLE_LABEL_1 = "dust-sensor" # Put your first variable label here
#VARIABLE_LABEL_2 = "alcohol" # Put your second variable label here
VARIABLE_LABEL_3 = "PIR"
def value (PIR):
    payload = {VARIABLE_LABEL_3 : PIR}
    status = 400
    attempts = 0
    while status >= 400 and attempts <= 5:
        req = requests.post(url=url, headers=headers, json=payload)
        status = req.status_code
        attempts += 1
        time.sleep(1)
    if status >= 400:
        print("[ERROR] Could not send data after 5 attempts, please check \ your token
        credentials and internet connection")
        return
    while True:
        PIR.wait_for_motion()
        print("You moved")
        value(1)
        PIR.wait_for_no_motion()
        print("Don't Moved")
        value(0)
```

### 6.3 Data gets from UBIDOTS cloud server. (IoT based Automation)

```
import time
import requests
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(21,GPIO.OUT)
DEVICE ="LED"
VARIABLE1 ="BLINK"
TOKEN = "BBFF-x5e4HGEd4oiVIl6M34KiuJqD6qU3Cx"
try:
    while True:
        light1=requests.get("http://things.ubidots.com/api/v1.6/devices/"+DEVICE+"/"+
            VARIABLE1+"/lv?token="+TOKEN)
        print(light1.content) # Returns the content of the response, in bytes
        print(light1.json()) # Returns a JSON object of the result
        data = light1.json()
        print(type(data))
    if data==1.0:
        #GPIO.output(21,GPIO.HIGH)
        #GPIO.output(26,GPIO.LOW)
        print("On")
        #time.sleep(3)
    if data==0.0:
        #GPIO.output(21,GPIO.LOW)
        #GPIO.output(26,GPIO.HIGH)
        print("Off")
        #time.sleep(3)
    except KeyboardInterrupt:
        pass
finally:
    GPIO.cleanup()
    light1 = "
```