

## **Weather Prediction using ESP8266 and DHT11**

Weather prediction refers to the use of current and near real-time data to predict weather conditions. It involves the continuous analysis of data collected from various weather stations and sensors, such as satellites, radars, and ground-based weather stations, to generate weather prediction. Real-time weather prediction models use sophisticated ML algorithms that take into account a wide range of factors, such as temperature, humidity, air pressure, wind direction, and cloud cover, to generate weather predictions for any particular instant of time .

Real-time weather prediction is essential for various industries and applications, such as aviation, agriculture, energy, and transportation. For example, airlines rely on real-time weather prediction to make decisions regarding flight routes and schedules, while farmers use real-time weather prediction to determine the best time to plant crops and harvest them. In addition, energy companies use real-time weather prediction to optimize the production and distribution of energy, while transportation companies use it to plan and manage their operations based on weather conditions. Overall, real-time weather prediction plays a critical role in enabling individuals and organizations to make informed decisions that can impact their safety, productivity, and profitability.

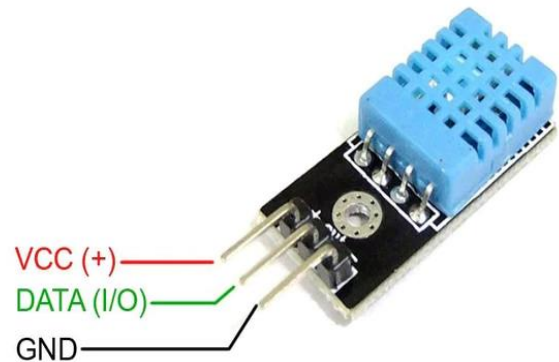
### **DHT11 :-**

The DHT11 sensor is a low-cost temperature and humidity sensor that is widely used in various applications due to its simplicity and affordability. It consists of a thermistor and a capacitive humidity sensor, which are integrated on a single chip along with an analog-to-digital converter and a digital signal processing unit.

The DHT11 sensor communicates with the microcontroller using a single-wire digital interface, making it easy to integrate with various microcontrollers such as Arduino, Raspberry Pi, and others. The sensor sends a 40-bit digital signal to the microcontroller, which includes the temperature and humidity data in binary format. The communication protocol is relatively simple, with the microcontroller sending a start signal to the sensor, and the sensor responding with the data signal.

### Features:

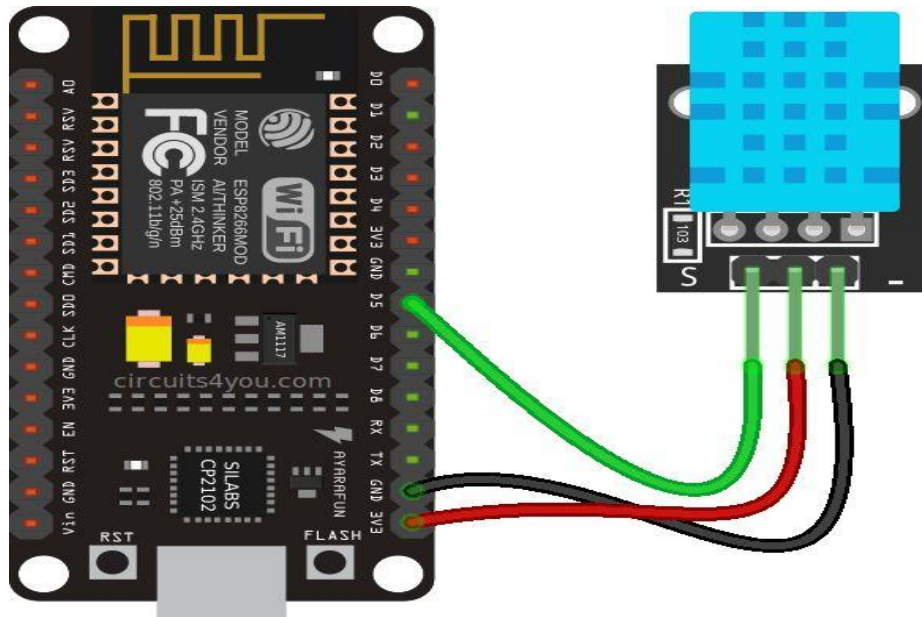
1. Low-cost temperature and humidity sensor
2. Measures temperature between 0°C and 50°C with an accuracy of  $\pm 2^\circ\text{C}$
3. Measures relative humidity between 20% and 90% with an accuracy of  $\pm 5\%$
4. Single-wire digital interface for easy integration with microcontrollers
5. Low power consumption
6. Compact size



### Pin details:

1. VCC: Power supply pin (3-5V DC)
2. Data: Digital data output pin
3. GND: Ground pin

### CIRCUIT DIAGRAM



Here, we have interfaced the **DHT11** sensor **Data pin** with the **Digital Pin D5** of **ESP8266**. **VCC** and **GND** pins of **DHT11** are connected to the **3.3 Volt** and **GND** pins of **ESP8266** respectively.

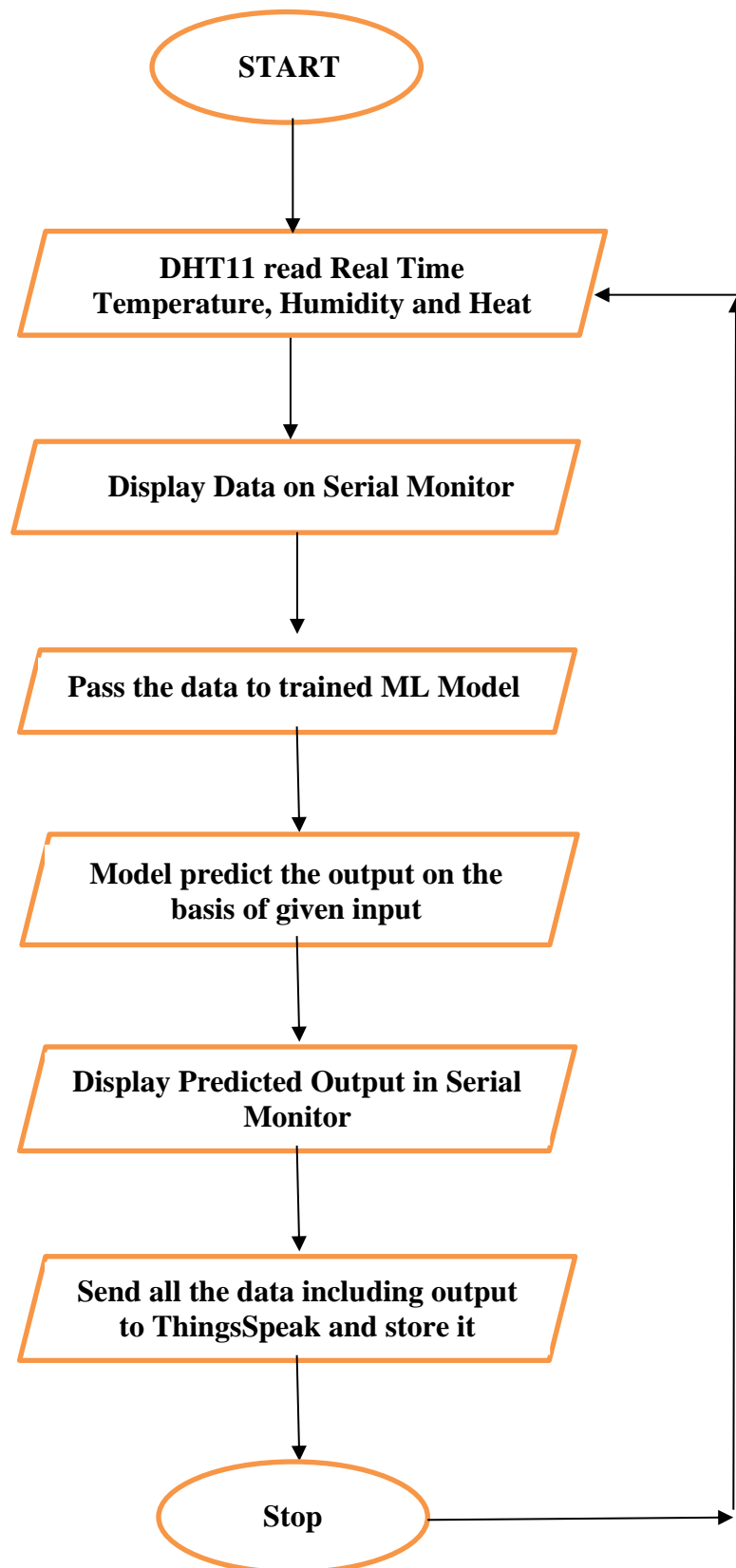
## **CONNECTION TABLE**

<b>ESP8266</b>	<b>DHT11 Sensor</b>
VV, Vin ( +5V )	( V ) VCC ( Positive + )
G, GND ( Ground )	( G ) GND ( Ground – )
D5 Pin	( S ) OUT Pin

## **STEPS FOR EXECUTION OF WEATHER PREDICTION**

1. Make a Decision Tree ML model by training it with past Weather dataset containing temperature, humidity, heat index and description of weather as columns.
2. After that, use a library **micromlgen** to make header file named “**DecisionTree.h**” for the trained model and add it to Arduino IDE libraries.
3. Write code for the DHT11 and ESP8266 interfacing in Arduino IDE and include the header file in code for calling the predict function from the ML model.
4. Add the code for ThingsSpeak in Arduino IDE, so that the real time data is send to the server for analyzing the change in weather with respect to change in temperature, humidity and Heat Index time to time.
5. Now, start and code the ESP8266 by plugging it with the source point and it takes real-time value of temperature, humidity and heat index and pass it to the predict function of Decision Tree Model for predicting the current weather.

## FLOW CHART



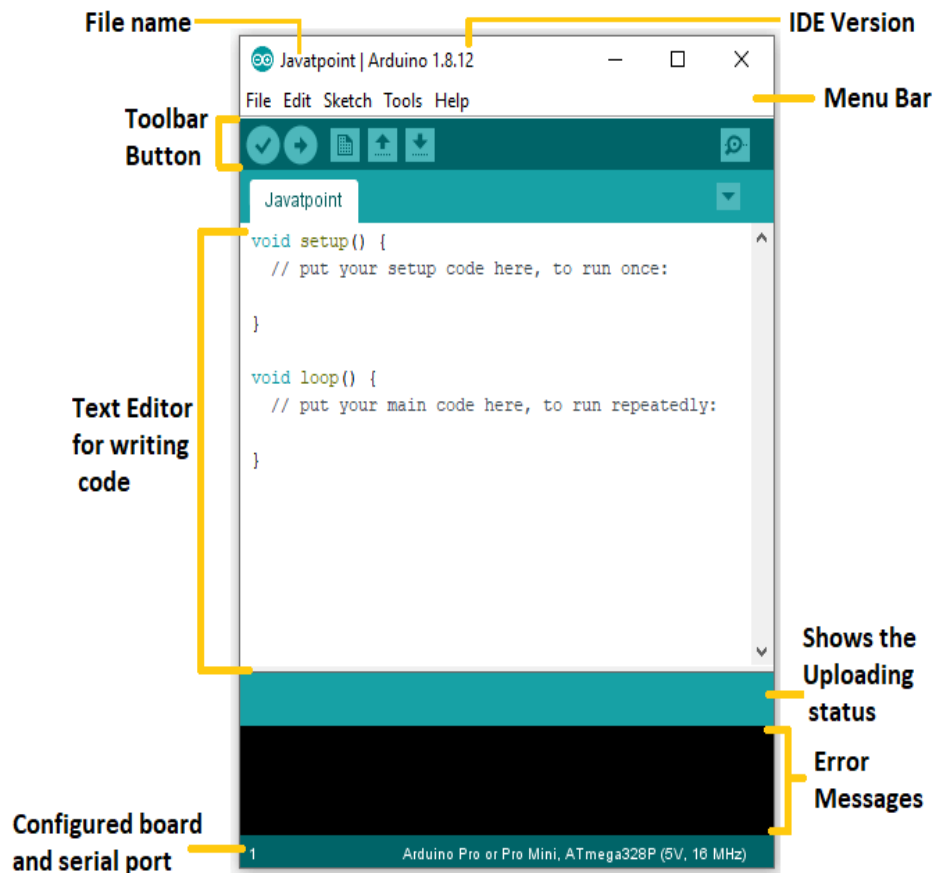
# SOFTWARE

## ARDUINO IDE

The Arduino IDE is an open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as **Windows, Mac OS X, and Linux**. It supports the programming languages C and C++. Here, IDE stands for **Integrated Development Environment**.

The program or code written in the Arduino IDE is often called as sketching. We need to connect the Genuino and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ino.'

The Arduino IDE will appear as:



## **JUPYTER NOTEBOOK**

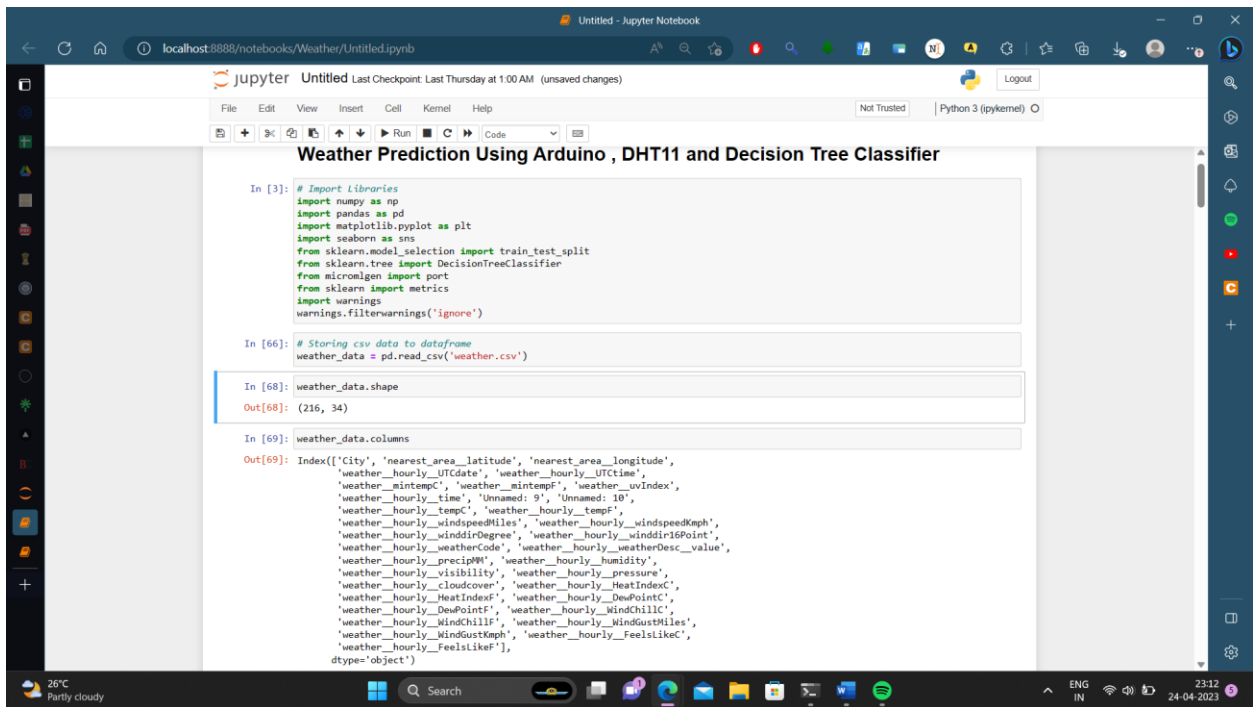
Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating notebook documents. A Jupyter Notebook document is a browser-based REPL containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media. Underneath the interface, a notebook is a JSON document, following a versioned schema, usually ending with the “.ipynb” extension.



Jupyter notebooks are built upon a number of popular open-source libraries. Jupyter Notebook can connect to many *kernels* to allow programming in different languages. A Jupyter kernel is a program responsible for handling various types of requests (code execution, code completions, inspection), and providing a reply. Kernels talk to the other components of Jupyter using ZeroMQ, and thus can be on the same or remote machines. Unlike many other Notebook-like interfaces, in Jupyter, kernels are not aware that they are attached to a specific document, and can be connected to many clients at once. Usually kernels allow execution of only a single language, but there are a couple of exceptions. By default Jupyter Notebook ships with the IPython kernel. As of the 2.3 release (October 2014), there are 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia and Haskell. Jupyter notebook is also used in exploratory data analysis, training of machine learning model and deep learning model.

# RESULTS

## 1. Code for Training ML Model :-



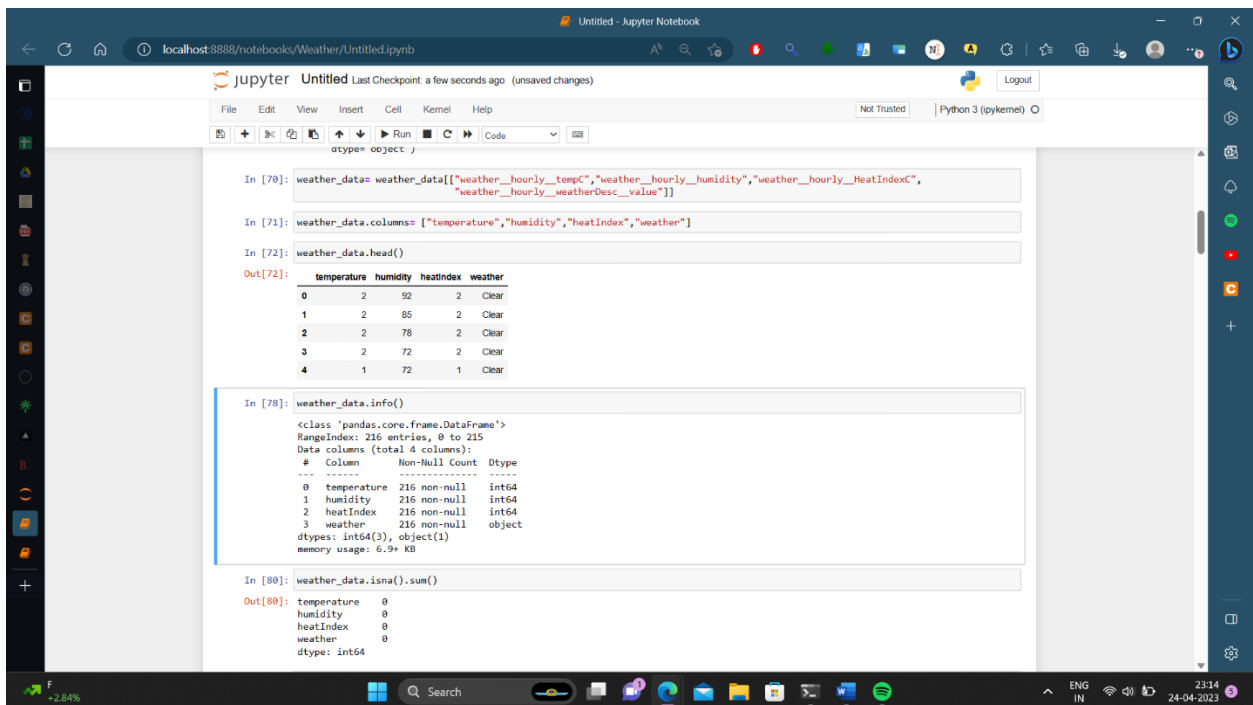
```

In [3]: # Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from microalgen import port
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')

In [66]: # Storing csv data to dataframe
weather_data = pd.read_csv('weather.csv')

In [68]: weather_data.shape
Out[68]: (216, 34)

In [69]: weather_data.columns
Out[69]: Index(['City', 'nearest_area_latitude', 'nearest_area_longitude',
'weather_hourly_UTdate', 'weather_hourly_UTctime',
'weather_mintemp', 'weather_mintemp', 'weather_uvindex',
'weather_hourly_time', 'Unnamed: 9', 'Unnamed: 10',
'weather_hourly_tempC', 'weather_hourly_temp',
'weather_hourly_windspeedMiles', 'weather_hourly_windspeedKmph',
'weather_hourly_winddirDegree', 'weather_hourly_winddir16Point',
'weather_hourly_weatherCode', 'weather_hourly_weatherDesc_value',
'weather_hourly_precipMM', 'weather_hourly_humidity',
'weather_hourly_visibility', 'weather_hourly_pressure',
'weather_hourly_cloudcover', 'weather_hourly_HeatIndexC',
'weather_hourly_HeatIndex', 'weather_hourly_DewPointC',
'weather_hourly_DewPoint', 'weather_hourly_WindChillC',
'weather_hourly_WindChillF', 'weather_hourly_WindGustMiles',
'weather_hourly_WindGustKmph', 'weather_hourly_FeelslikeC',
'weather_hourly_FeelslikeF'],
dtype='object')
```



```

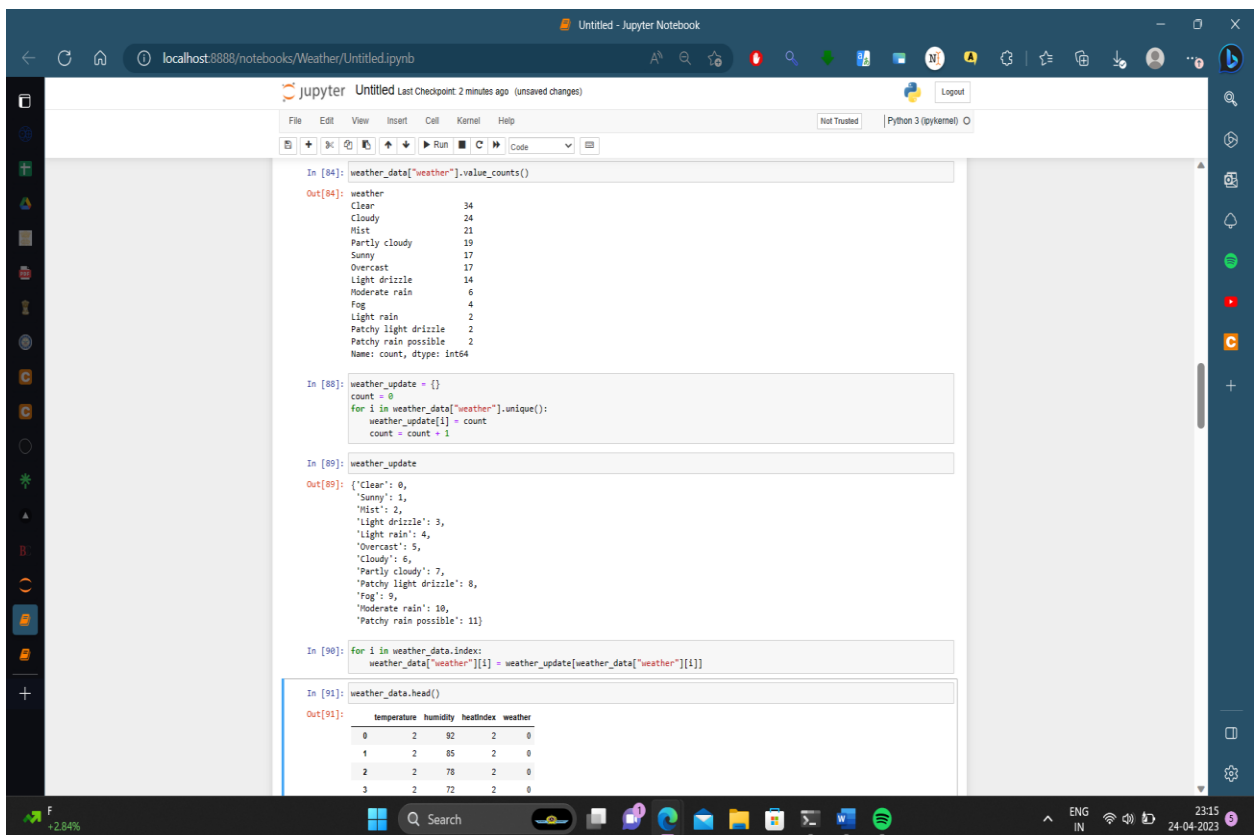
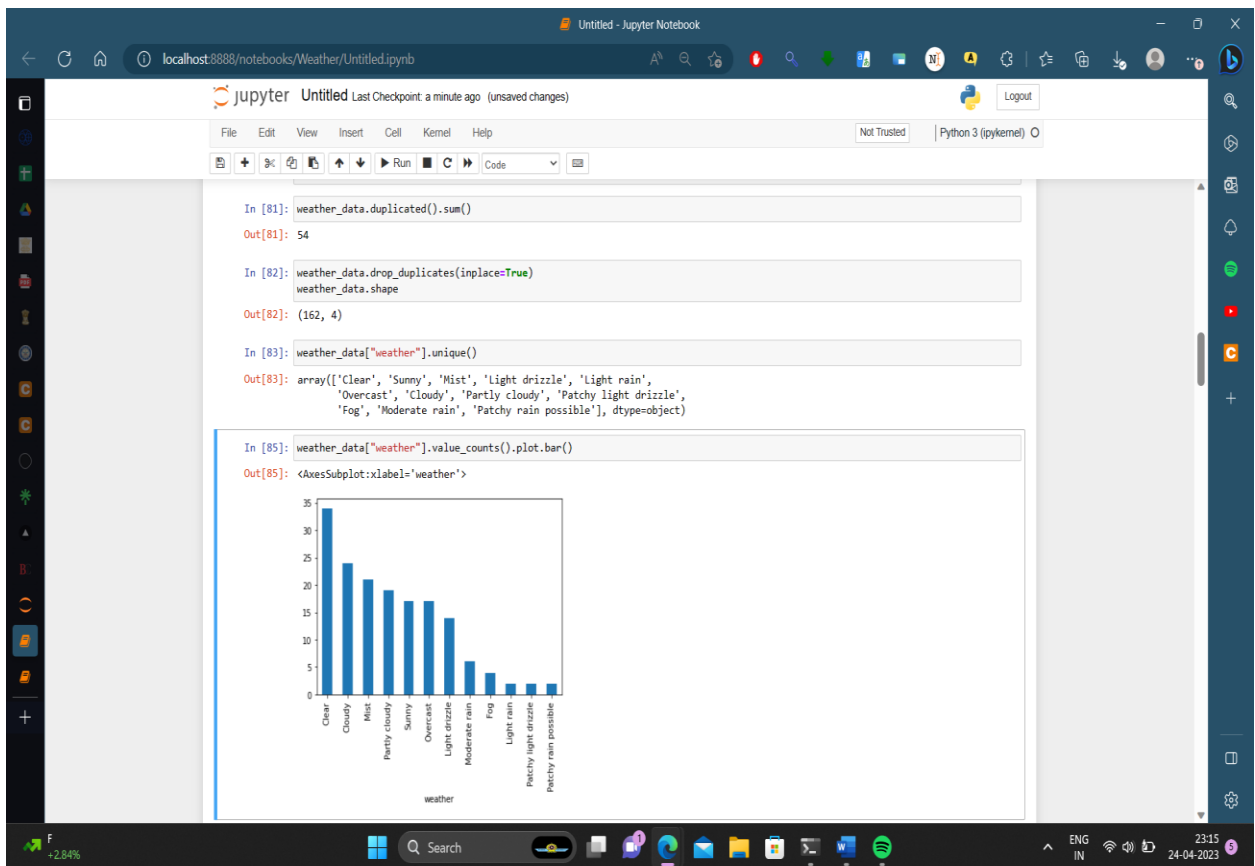
In [70]: weather_data= weather_data[['weather_hourly_tempC','weather_hourly_humidity','weather_hourly_HeatIndexC',
'weather_hourly_weatherDesc_value']]

In [71]: weather_data.columns= ['temperature','humidity','heatIndex','weather']

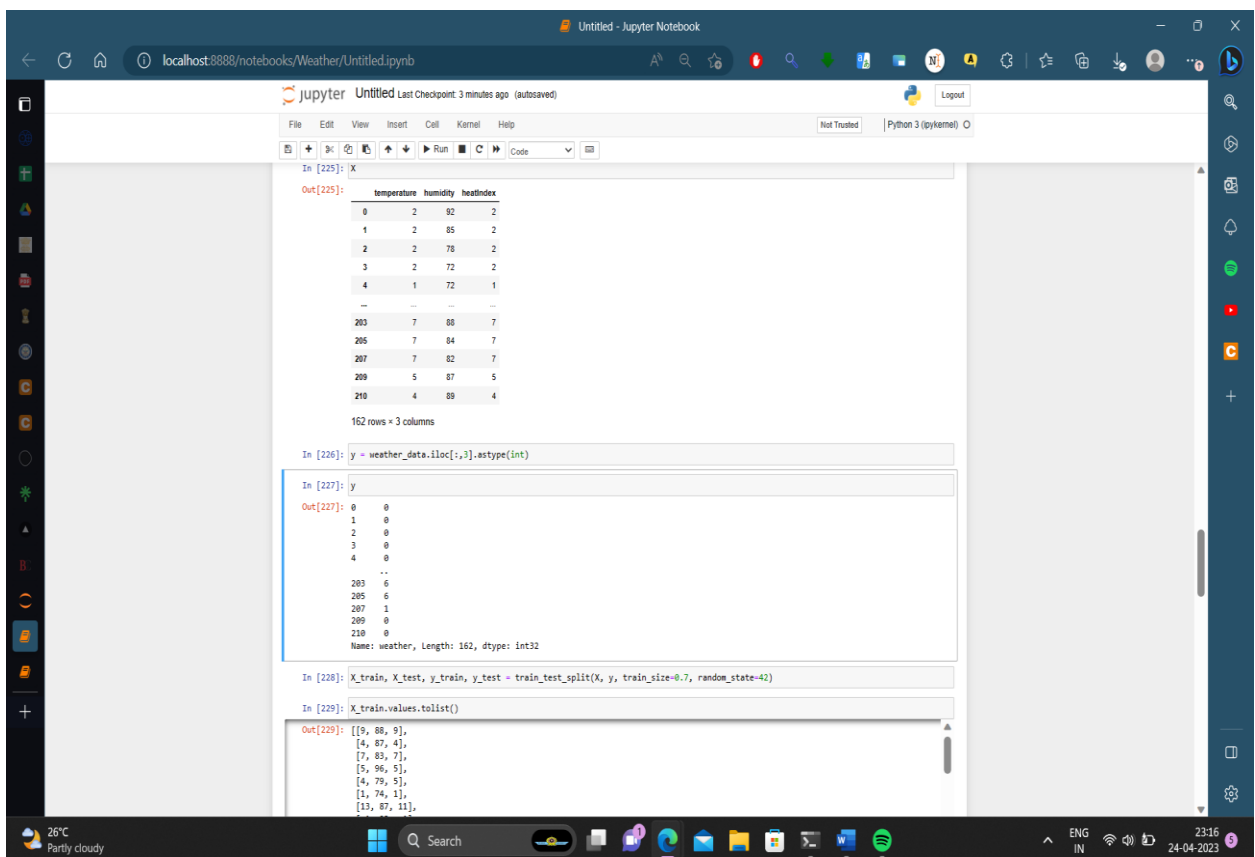
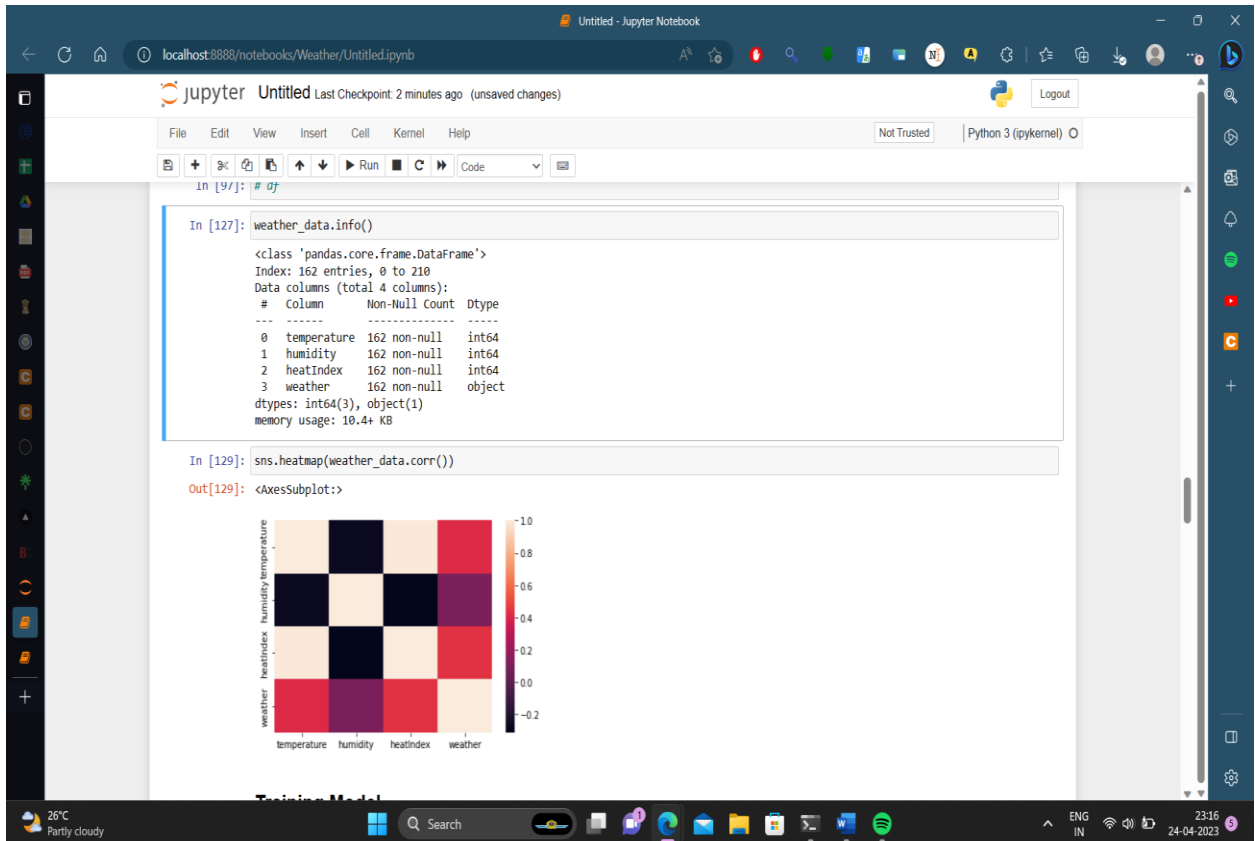
In [72]: weather_data.head()
Out[72]:
   temperature  humidity  heatIndex  weather
0             2         92           2    Clear
1             2         86           2    Clear
2             2         78           2    Clear
3             2         72           2    Clear
4             1         72           1    Clear

In [78]: weather_data.info()
Out[78]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 216 entries, 0 to 215
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ---
 0  temperature  216 non-null    int64
 1  humidity     216 non-null    int64
 2  heatIndex    216 non-null    int64
 3  weather      216 non-null    object
dtypes: int64(3), object(1)
memory usage: 6.9+ KB

In [80]: weather_data.isna().sum()
Out[80]:
temperature    0
humidity        0
heatIndex       0
weather         0
dtype: int64
```









The screenshot shows a Jupyter Notebook window titled 'Untitled - Jupyter Notebook' with the URL 'localhost:8888/notebooks/Weather/Untitled.ipynb'. At the top, a table displays performance metrics:

accuracy	0.32	0.33	0.31	49
macro avg	0.32	0.33	0.31	49
weighted avg	0.36	0.37	0.35	49

The main code cell contains the following C++ code:

```
In [249]: print(port(weather_classifier))

#pragma once
#include <csdang>
namespace Eloquent {
    namespace ML {
        namespace Port {
            class DecisionTree {
            public:
                /**
                 * Predict class for features vector
                 */
                int predict(float *x) {
                    if (x[2] <= 3.5) {
                        if (x[1] <= 93.5) {
                            if (x[0] <= 2.5) {
                                if (x[2] <= 1.5) {
                                    if (x[0] <= 0.5) {
                                        return 0;
                                    }
                                }
                            }
                        }
                    }
                }
            };
        };
    };
};
```

The output cell shows the result of the prediction:

```
In [250]: weather_classifier.predict([[39,43,98]])
Out[250]: array([6])

In [ ]:
```

## 2. Code for header file “DecisionTree.h” :-

```
#pragma once
namespace Eloquent {
    namespace ML {
        namespace Port {
            class DecisionTree {
            public:
                /**
                 * Predict class for features vector
                 */
                int predict(float *x) {
                    if (x[2] <= 3.5) {
                        if (x[1] <= 93.5) {
                            if (x[0] <= 2.5) {
                                if (x[2] <= 1.5) {
                                    if (x[0] <= 0.5) {
```

```

        return 0;
    }
    else {
        if (x[1] <= 91.0) {
            return 0;
        }
        else {
            return 1;
        }
    }
}
else {
    if (x[0] <= 1.5) {
        return 7;
    }
    else {
        if (x[1] <= 92.5) {
            return 0;
        }
        else {
            return 0;
        }
    }
}
else {
    if (x[1] <= 84.0) {
        return 0;
    }
    else {
        if (x[1] <= 87.5) {

```

```

        return 1;
    }
    else {
        if (x[1] <= 90.5) {
            if (x[1] <= 88.5) {
                return 0;
            }
            else {
                if (x[1] <= 89.5) {
                    return 0;
                }
                else {
                    return 0;
                }
            }
        }
        else {
            return 0;
        }
    }
}
else {
    if (x[2] <= 0.5) {
        if (x[1] <= 96.5) {
            return 0;
        }
        else {
            return 7;
        }
    }
}

```

```

    }
else {
    if (x[2] <= 2.5) {
        if (x[1] <= 94.5) {
            if (x[2] <= 1.5) {
                return 2;
            }
            else {
                return 0;
            }
        }
        else {
            if (x[1] <= 96.5) {
                return 2;
            }
            else {
                return 2;
            }
        }
    }
else {
    if (x[1] <= 95.5) {
        if (x[1] <= 94.5) {
            return 6;
        }
        else {
            return 5;
        }
    }
    else {
        return 2;
    }
}

```

```

        }
    }
}
}
}
else {
    if (x[1] <= 94.5) {
        if (x[0] <= 5.5) {
            if (x[1] <= 86.5) {
                if (x[2] <= 4.5) {
                    if (x[1] <= 85.5) {
                        if (x[1] <= 84.5) {
                            return 1;
                        }
                    }
                    else {
                        return 11;
                    }
                }
            }
            else {
                return 1;
            }
        }
    }
    else {
        if (x[0] <= 4.5) {
            return 3;
        }
        else {
            if (x[1] <= 84.0) {
                return 1;
            }
            else {

```

```

        if (x[1] <= 85.5) {
            return 3;
        }
        else {
            return 3;
        }
    }
}
}
else {
    if (x[1] <= 90.0) {
        if (x[2] <= 4.5) {
            return 0;
        }

        else {
            if (x[0] <= 4.5) {
                return 3;
            }
            else {
                return 0;
            }
        }
    }
    else {
        if (x[0] <= 4.5) {
            if (x[1] <= 91.5) {
                return 6;
            }
            else {

```



```

        return 3;
    }
}
else {
    return 6;
}
}
}
else {
    if (x[1] <= 86.5) {
        if (x[2] <= 7.5) {
            if (x[1] <= 85.0) {
                return 1;
            }
            else {
                return 6;
            }
        }
        else {
            if (x[0] <= 10.5) {
                if (x[0] <= 9.0) {
                    return 6;
                }
                else {
                    if (x[1] <= 81.0) {
                        return 6;
                    }
                    else {
                        return 7;
                    }
                }
            }
        }
    }
}

```

```

    }
}
else {
    if (x[1] <= 80.5) {
        return 6;
    }
    else {
        if (x[0] <= 11.5) {
            return 5;
        }
        else {
            if (x[0] <= 12.5) {
                if (x[1] <= 82.5) {
                    return 6;
                }
                else {
                    return 5;
                }
            }
            else {
                return 5;
            }
        }
    }
}
}
}
else {
    if (x[2] <= 9.5) {
        if (x[2] <= 7.5) {
            if (x[1] <= 92.5) {

```

```
    if (x[1] <= 90.5) {
        if (x[0] <= 6.5) {
            if (x[1] <= 89.5) {
                return 10;
            }
            else {
                return 2;
            }
        }
        else {
            if (x[1] <= 88.5) {
                return 6;
            }
            else {
                return 5;
            }
        }
    }
    else {
        if (x[2] <= 6.5) {
            return 5;
        }
        else {
            return 10;
        }
    }
}
else {
    return 2;
}
}
```

```
else {  
    if (x[1] <= 88.5) {  
        return 3;  
    }  
    else {  
        if (x[1] <= 89.5) {  
            return 6;  
        }  
  
        else {  
            return 7;  
        }  
    }  
}  
}  
else {  
    if (x[1] <= 88.0) {  
        if (x[2] <= 10.5) {  
            return 3;  
        }  
  
        else {  
            return 5;  
        }  
    }  
  
    else {  
        if (x[1] <= 89.5) {  
            return 4;  
        }  
    }  
}
```

```
        else {  
            return 3;  
        }  
    }  
}  
}  
}
```

```
else {  
    if (x[1] <= 96.5) {  
        if (x[2] <= 4.5) {  
            return 2;  
        }  
        else {  
            if (x[2] <= 7.5) {  
                if (x[1] <= 95.5) {  
                    return 7;  
                }  
                else {  
                    if (x[2] <= 6.0) {  
                        return 2;  
                    }  
                    else {  
                        return 2;  
                    }  
                }  
            }  
        }  
        else {  
            return 2;  
        }  
    }  
}
```

```
        }
    }
    else {
        if (x[1] <= 97.5) {
            if (x[2] <= 5.5) {
                return 7;
            }
            else {
                return 9;
            }
        }
        else {
            return 9;
        }
    }
}
}
}
protected:
};
}
}
```

### 3. Code for Arduino IDE :-

```
PBL3.1 | Arduino 1.8.20 Hourly Build 2022/04/25 09:33
File Edit Sketch Tools Help

PBL3.1 |
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ThingSpeak.h>
#include "DecisionTree.h"
Eloquent ML::Port::DecisionTree weatherClassifier;

#include "DHT.h"
float t;
float h;
float hif;
int prediction;

const char* ssid = "Sharat";//type your ssid
const char* password = "akhanda918";//type your password

//WiFiServer server(80);//Service Port
WiFiClient client;
unsigned long myChannelNumber = 2107048;
const char* myWriteAPIKey = "DRAFT9P2CE8MKVVB";

#define DHTPIN D1 // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHT11 test!");
  delay(10);
  WiFi.begin(ssid, password);
  ThingSpeak.begin(client);

  dht.begin();
}

88 NodeMCU 1.0 (ESP-12E Module), 80 MHz Flash, Disabled (new aborts on oom), Disabled, All SSL ciphers (most compatible), 32K cache + 32K IRAM (balanced), Use pgm_read macros for IRAM/PROGMEM, 4MB (FS 2MB OTA ~1019KB), 2 v2 Lower Memory, Disabled, None, Only Sketch, 115200 on COM7
26°C Partly cloudy Search 23:37 24-04-2023
```

```
PBL3.1 | Arduino 1.8.20 Hourly Build 2022/04/25 09:33
File Edit Sketch Tools Help

PBL3.1 |
void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  float h = dht.readHumidity();
  // Read temperature as Celsius
  float t = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Celsius (the default)
  float hif = dht.computeHeatIndex(t, h);

  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" Temperature: ");
  Serial.print(t);
  Serial.print(" Heat index: ");
  Serial.print(hif);
  Serial.println("C");

  float input[3] = {t, h, hif};
  int prediction = weatherClassifier.predict(input);

  Serial.print("Prediction: ");
  if (prediction == 0) {
    Serial.println("Clear");
  }
  else if (prediction == 1) {
    Serial.println("Sunny");
  }
}

69 NodeMCU 1.0 (ESP-12E Module), 80 MHz Flash, Disabled (new aborts on oom), Disabled, All SSL ciphers (most compatible), 32K cache + 32K IRAM (balanced), Use pgm_read macros for IRAM/PROGMEM, 4MB (FS 2MB OTA ~1019KB), 2 v2 Lower Memory, Disabled, None, Only Sketch, 115200 on COM7
26°C Partly cloudy Search 23:38 24-04-2023
```

```
PBL3.1 §
Serial.println("Start");
}
else if(prediction == 3){
  Serial.println("Light drizzle");
}
else if(prediction == 4){
  Serial.println("Light rain");
}
else if(prediction == 5){
  Serial.println("Overcast");
}
else if(prediction == 6){
  Serial.println("Cloudy");
}
else if(prediction == 7){
  Serial.println("Partly cloudy");
}
else if(prediction == 8){
  Serial.println("Patchy light drizzle");
}
else if(prediction == 9){
  Serial.println("Fog");
}
else if(prediction == 10){
  Serial.println("Moderate rain");
}
else if(prediction == 11){
  Serial.println("Patchy rain possible");
}
ThingSpeak.writeField(myChannelNumber, 1, t, myWriteAPIKey);
ThingSpeak.writeField(myChannelNumber, 2, h, myWriteAPIKey);
ThingSpeak.writeField(myChannelNumber, 3, hif, myWriteAPIKey);
ThingSpeak.writeField(myChannelNumber, 4, prediction, myWriteAPIKey);

delay(1000);
}
```

## Code :-

```
#include <ESP8266WiFi.h>
```

```
#include <WiFiClient.h>
```

```
#include <ThingSpeak.h>
```

```
#include "DecisionTree.h"
```

```
Eloquent::ML::Port::DecisionTree weatherClassifier;
```

```
#include "DHT.h"
```

```
float t;
```

```
float h;
```

```
float hif;
```

```
int prediction;
```

```
const char* ssid = "Bharat";//type your ssid
```

```
const char* password = "akhandta891@";//type your password
```



```

//WiFiServer server(80); //Service Port
WiFiClient client;
unsigned long myChannelNumber = 2107048;
const char* myWriteAPIKey = "DRAFT9P2CEBMXVRB";

#define DHTPIN D1    // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 test!"));
  delay(10);
  WiFi.begin(ssid, password);
  ThingSpeak.begin(client);

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  float h = dht.readHumidity();
  // Read temperature as Celsius
  float t = dht.readTemperature();
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println(F("Failed to read from DHT sensor!"));
  }
}

```

```

    return;
}

// Compute heat index in Celsius (the default)
float hif = dht.computeHeatIndex(t,h);

Serial.print(F("Humidity: "));
Serial.print(h);
Serial.print(F("%  Temperature: "));
Serial.print(t);
Serial.print(F("°C  Heat index: "));
Serial.print(hif);
Serial.println(F("°C"));

float input[3] = {t,h,hif};
int prediction = weatherClassifier.predict(input);

Serial.print("Prediction: ");
if (prediction == 0){
    Serial.println("Clear");
}
else if(prediction == 1){
    Serial.println("Sunny");
}
else if(prediction == 2){
    Serial.println("Mist");
}
else if(prediction == 3){
    Serial.println("Light drizzle");
}
else if(prediction == 4){

```

```

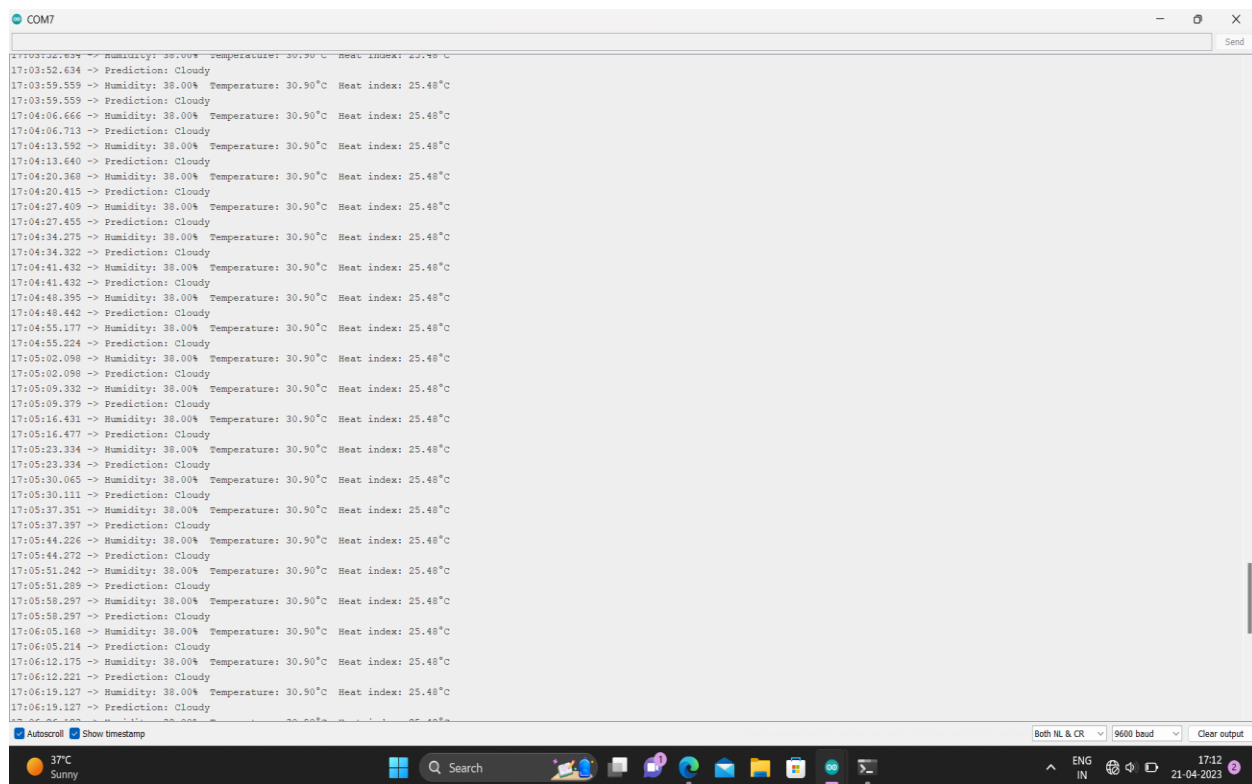
    Serial.println("Light rain");
}
else if(prediction == 5){
    Serial.println("Overcast");
}
else if(prediction == 6){
    Serial.println("Cloudy");
}
else if(prediction == 7){
    Serial.println("Partly cloudy");
}
else if(prediction == 8){
    Serial.println("Patchy light drizzle");
}
else if(prediction == 9){
    Serial.println("Fog");
}
else if(prediction == 10){
    Serial.println("Moderate rain");
}
else if(prediction == 11){
    Serial.println("Patchy rain possible");
}

ThingSpeak.writeField(myChannelNumber, 1, t, myWriteAPIKey);
ThingSpeak.writeField(myChannelNumber, 2, h, myWriteAPIKey);
ThingSpeak.writeField(myChannelNumber, 3, hif, myWriteAPIKey);
ThingSpeak.writeField(myChannelNumber, 4, prediction, myWriteAPIKey);

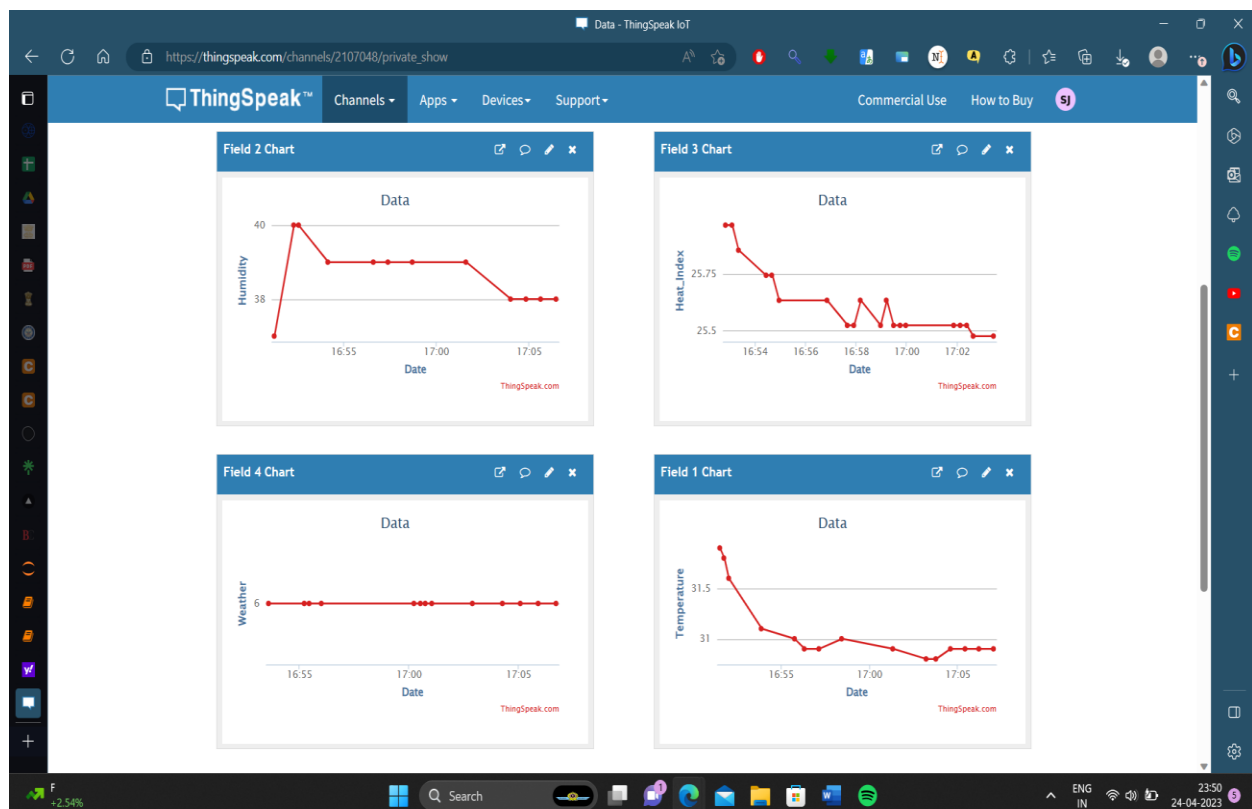
delay(1000);
}

```

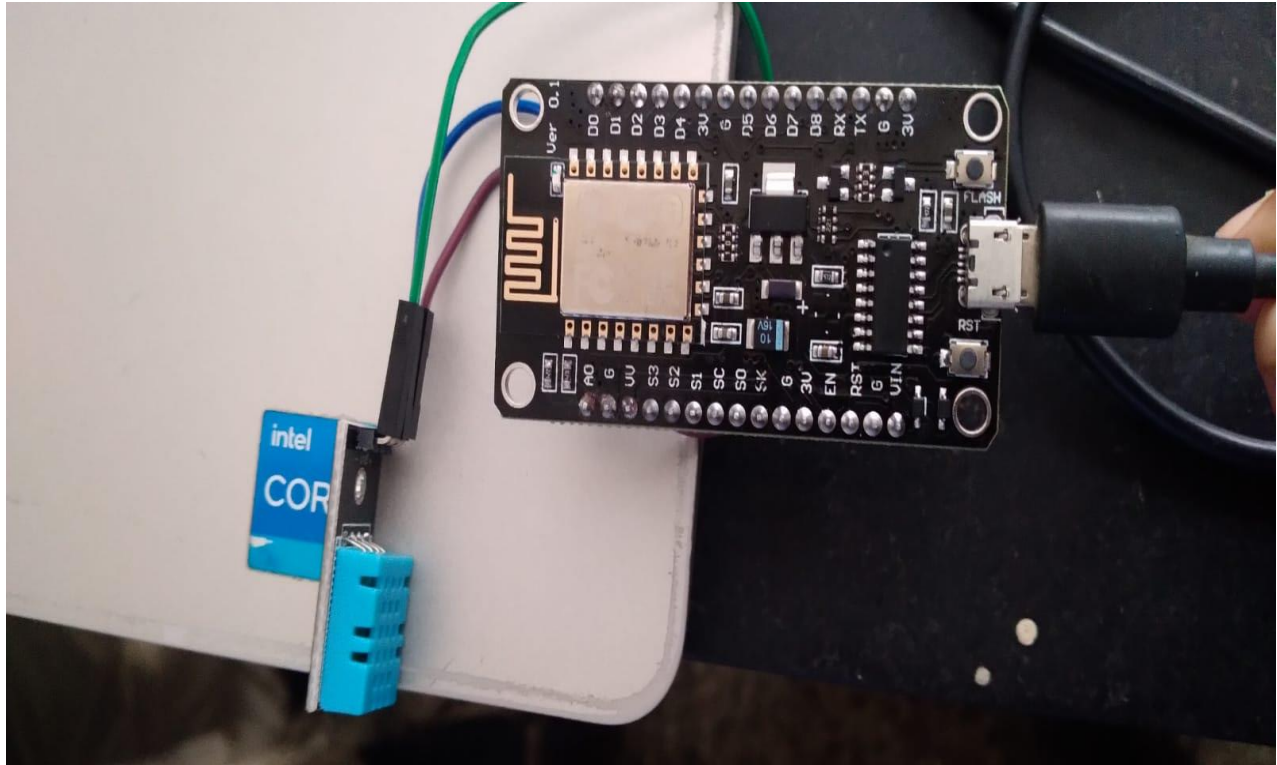
## Output of Serial Monitor :-



## 4. Things Speak Output :-



## 5. Circuit of Communication between DHT11 and ESP8266 :-



### Project Outcome

The project “Weather Prediction using ESP8266 and DHT11” has helped us to implement the practical application of IOT(Internet of Things). We learnt about model training on Jupyter Notebook and how to use it with Arduino IDE. The data from the DHT11 Sensor is sent to Arduino IDE and it predicts the current weather for real-time temperature and humidity. We also learned about sending live data to ThingSpeak.

The following course outcome is mapped for the project “Weather Prediction using ESP8266 and DHT11”: -

CO6: Design an IOT application with ML and Arduino /Raspberry Pi