IOT BASED WEATHER FORECASTING PROJECT:

1. Project Overview:

This project involves building a weather monitoring system using an ESP32 and DHT22 sensor to measure temperature and humidity, then uploading the data to ThingSpeak for visualization. Later, the data is downloaded as a CSV file and used in a Machine Learning model to predict environmental conditions like temperature trends or humidity forecasting.

2. Components Required:

| Component | Quantity |
| --- | --- |
| ESP32 Board | 1 |
| DHT22 Temperature & Humidity Sensor | 1 |
| Jumper Wires | as required |
| Breadboard | 1 |
| Wi-Fi Connection | 1 |

3. Working Principle:

1. The DHT22 sensor measures the temperature and humidity.
2. Data is read using the ESP32 microcontroller.
3. The data is uploaded to ThingSpeak every 20 seconds.
4. Later, data is downloaded from ThingSpeak in CSV format.
5. The CSV is used to train and test a Machine Learning model to predict future environmental conditions.

4. Libraries Used:

#include <WiFi.h> // Handles WiFi connectivity

#include "DHT.h" // Used to interface with the DHT22 sensor

#include "ThingSpeak.h" // Sends data to ThingSpeak

5. Pin Configuration:

| ESP32 Pin | Connected To |
| --- | --- |
| GPIO 21 | DHT22 Data Pin |

6. WiFi and ThingSpeak Setup:

* Connect ESP32 to Wi-Fi using:

const char\* ssid = "Wokwi-GUEST";

const char\* password = "";

* Set up ThingSpeak:
  + Channel ID: 2995393;( Example:)
  + Write API Key: "PLU2BLKFQ7AXOPG"(example:)

7. Program Flow:

1. Initialize serial and WiFi
2. Begin DHT22 sensor
3. Loop every 20 seconds:
   * Read temperature and humidity
   * Display on Serial Monitor
   * Upload to ThingSpeak to display in graphs (field 1 = temperature, field 2 = humidity)

8. ThingSpeak Setup:

1. Go to <https://thingspeak.com>
2. Create an account and channel
3. Add two fields:
   * Field 1: Temperature
   * Field 2: Humidity
4. Copy:
   * Channel ID
   * Write API Key
5. Paste into code

9. Downloading Data as CSV from ThingSpeak:

1. Go to your ThingSpeak Channel
2. Click on Export (top-right)
3. Choose CSV format
4. Download the file (.csv) for data analysis

Code to upload the data to thingspeak:

#include <WiFi.h>

#include "DHT.h"

#include "ThingSpeak.h"

#define DHTPIN 21

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

const char\* ssid = "Wokwi-GUEST";

const char\* password = "";

WiFiClient client;

unsigned long myChannelNumber = 2995393;

const char\* myWriteAPIKey = "PLU2BL3KFQ7AXOPG";

void setup() {

**Serial**.begin(115200);

  dht.begin();

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(1000);

  }

  ThingSpeak.begin(client);

}

void loop() {

  float humidity = dht.readHumidity();

  float temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature)) {

    delay(10000);

    return;

  }

**Serial**.print("Temperature: ");

**Serial**.print(temperature);

**Serial**.print("°C  | Humidity: ");

**Serial**.print(humidity);

**Serial**.println("%");

  ThingSpeak.setField(1, temperature);

  ThingSpeak.setField(2, humidity);

  int statusCode = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

  if (statusCode == 200) {

**Serial**.println("Channel update successful.");

  } else {

**Serial**.print("Problem updating channel. HTTP error code ");

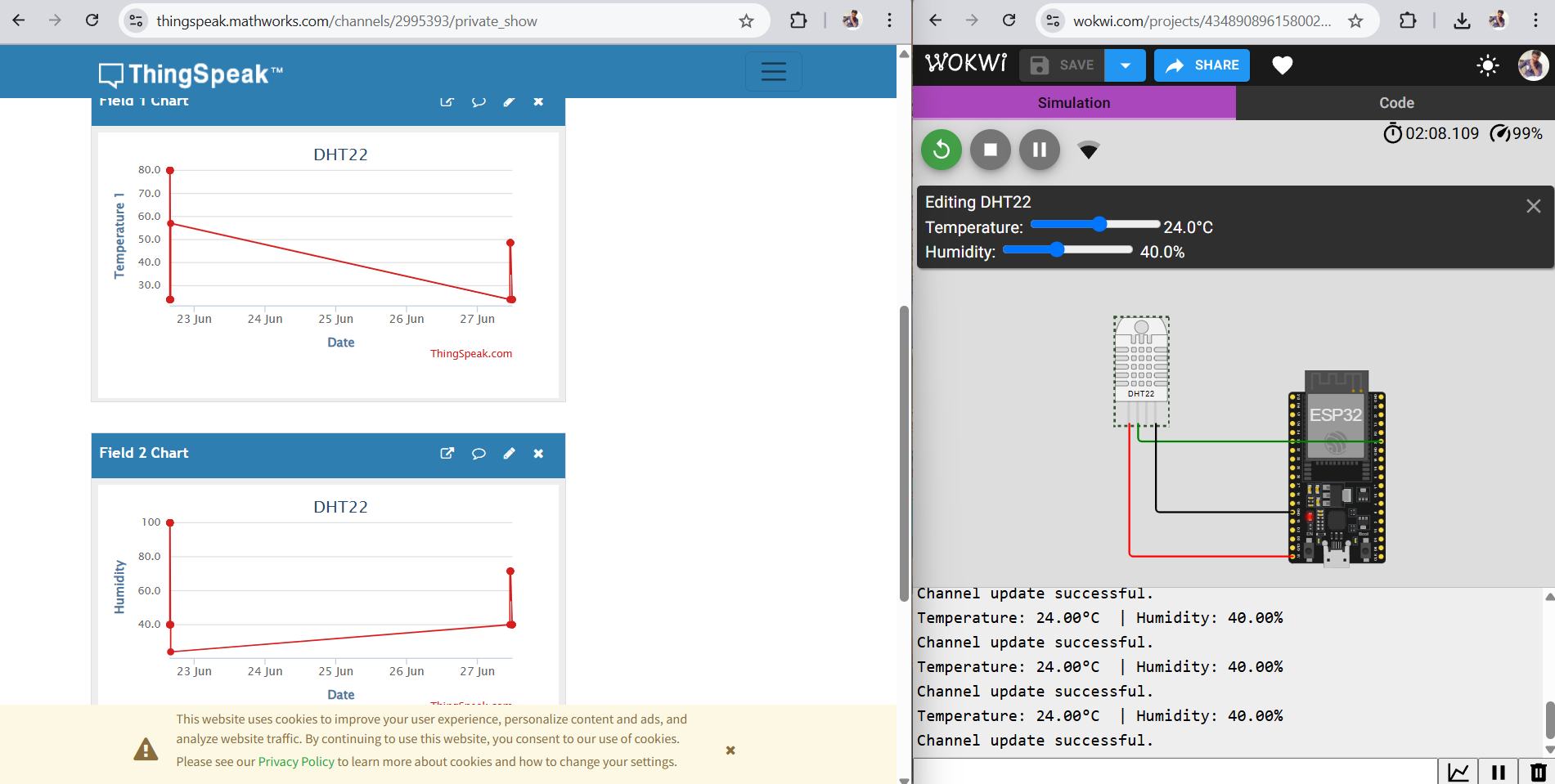
**Serial**.println(statusCode);

  }

  delay(20000);

}

Output:



Machine learning part to predict :

import tkinter as tk

from tkinter import messagebox

import pandas as pd

import numpy as np

import requests

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from datetime import datetime, timedelta

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

# === ThingSpeak Settings ===

CHANNEL\_ID = '2995393'

READ\_API\_KEY = 'CEAXKGRIZT8NVBA5'

TEMP\_FIELD = 1

HUMIDITY\_FIELD = 2

# === Fetch Data ===

def fetch\_data(field\_num):

    url = f'https://api.thingspeak.com/channels/{CHANNEL\_ID}/fields/{field\_num}.json?api\_key={READ\_API\_KEY}&results=800'

    response = requests.get(url)

    if response.status\_code != 200:

        return None

    data = response.json().get('feeds', [])

    if not data:

        return None

    df = pd.DataFrame(data)

    df['created\_at'] = pd.to\_datetime(df['created\_at'])

    df['value'] = pd.to\_numeric(df[f'field{field\_num}'], errors='coerce')

    df = df.dropna().sort\_values('created\_at')

    df['timestamp'] = (df['created\_at'] - df['created\_at'].min()).dt.total\_seconds() / 3600

    return df[['timestamp', 'value', 'created\_at']]

# === Train Model ===

def train\_model(df):

    X = df[['timestamp']]

    y = df['value']

    model = LinearRegression()

    model.fit(X, y)

    return model

# === Predict Future ===

def predict\_future(model, last\_timestamp, hours, reference\_time):

    future\_timestamps = np.arange(last\_timestamp + 1, last\_timestamp + hours + 1).reshape(-1, 1)

    future\_df = pd.DataFrame(future\_timestamps, columns=['timestamp'])

    predictions = model.predict(future\_df)

    future\_times = [reference\_time + timedelta(hours=int(h - last\_timestamp)) for h in future\_timestamps.flatten()]

    return future\_times, predictions

# === Rain Prediction Rule ===

def should\_rain(predicted\_temp):

    return ["Yes" if t < 28 else "no rain chances" for t in predicted\_temp]

# === Plot Graphs ===

def plot\_prediction(hours):

    temp\_df = fetch\_data(TEMP\_FIELD)

    hum\_df = fetch\_data(HUMIDITY\_FIELD)

    if temp\_df is None or hum\_df is None or temp\_df.empty or hum\_df.empty:

        return

    temp\_model = train\_model(temp\_df)

    hum\_model = train\_model(hum\_df)

    temp\_times, temp\_preds = predict\_future(temp\_model, temp\_df['timestamp'].iloc[-1], hours, temp\_df['created\_at'].iloc[-1])

    hum\_times, hum\_preds = predict\_future(hum\_model, hum\_df['timestamp'].iloc[-1], hours, hum\_df['created\_at'].iloc[-1])

    rain\_status = should\_rain(temp\_preds)

    fig, axs = plt.subplots(1, 2, figsize=(12, 4), dpi=100)

    month\_year = temp\_times[0].strftime("%B %Y")

    fig.suptitle(f"Weather Forecast - {month\_year}", fontsize=14, fontweight='bold')

    axs[0].plot(temp\_df['created\_at'].dt.strftime('%d'), temp\_df['value'], label="Actual Temp", color='blue')

    axs[0].plot([d.strftime('%d') for d in temp\_times], temp\_preds, label="Predicted Temp", color='red', linestyle='--')

    axs[0].set\_title("Temperature")

    axs[0].set\_xlabel("Date")

    axs[0].set\_ylabel("°C")

    axs[0].legend()

    axs[0].grid(True)

    axs[1].plot(hum\_df['created\_at'].dt.strftime('%d'), hum\_df['value'], label="Actual Humidity", color='green')

    axs[1].plot([d.strftime('%d') for d in hum\_times], hum\_preds, label="Predicted Humidity", color='orange', linestyle='--')

    axs[1].set\_title("Humidity")

    axs[1].set\_xlabel("Date")

    axs[1].set\_ylabel("%")

    axs[1].legend()

    axs[1].grid(True)

    for widget in plot\_frame.winfo\_children():

        widget.destroy()

    canvas = FigureCanvasTkAgg(fig, master=plot\_frame)

    canvas.draw()

    canvas.get\_tk\_widget().pack()

    summary = "🌧️ Rain Prediction (Next {} Hours):\n".format(hours)

    for i in range(len(temp\_times)):

        summary += "{} - {}\n".format(temp\_times[i].strftime("%d %b"), rain\_status[i])

    result\_label.config(text=summary)

# === GUI Setup ===

root = tk.Tk()

root.title("Weather Forecast Dashboard")

root.geometry("1100x700")

root.configure(bg="#eef6fb")

tk.Label(root, text="🌦️ Weather Forecast Dashboard", font=("Helvetica", 20, "bold"), bg="#eef6fb", fg="#333").pack(pady=20)

btn\_frame = tk.Frame(root, bg="#eef6fb")

btn\_frame.pack(pady=10)

tk.Button(btn\_frame, text="Predict Next Day", width=20, command=lambda: plot\_prediction(24), bg="skyblue", font=("Arial", 12)).grid(row=0, column=0, padx=10)

tk.Button(btn\_frame, text="Predict Next Week", width=20, command=lambda: plot\_prediction(24 \* 7), bg="lightgreen", font=("Arial", 12)).grid(row=0, column=1, padx=10)

tk.Button(btn\_frame, text="Predict Next Month", width=20, command=lambda: plot\_prediction(24 \* 30), bg="orange", font=("Arial", 12)).grid(row=0, column=2, padx=10)

plot\_frame = tk.Frame(root, bg="#ffffff", bd=2, relief="groove")

plot\_frame.pack(fill="both", expand=True, padx=20, pady=20)

result\_label = tk.Label(root, text="", bg="#eef6fb", font=("Arial", 12), justify="left")

result\_label.pack(pady=10)

root.mainloop()

output:

