

National Textile University, Faisalabad



Department of Computer Science

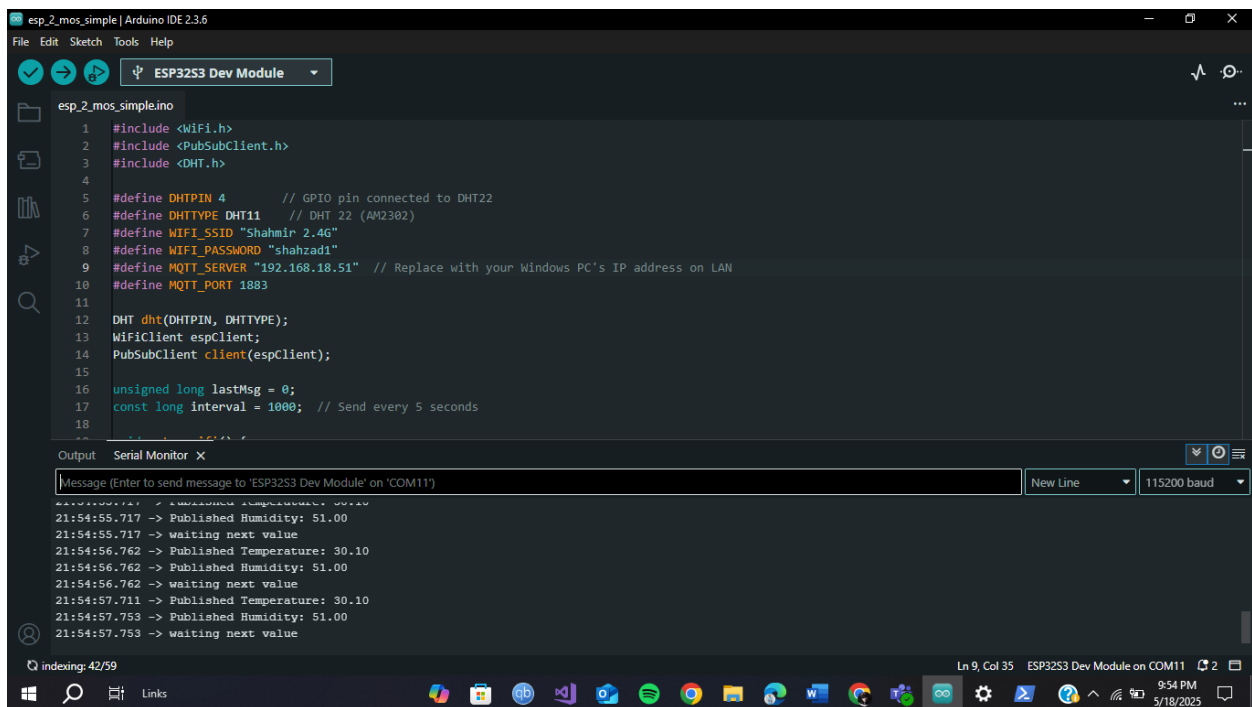
Name:	Muhammad Abdullah
Class:	BSAI – 6 ^h
Registration No:	22-NTU-CS-1358
Lab Report:	Lab 13 home task
Course Code:	AIE-3079
Course Name:	IoT Fundamentals
Submitted To:	Mr. Nasir Mehmood
Submission Date:	18-05-2025

IOT Lab13

Home tasks

Task1: Run the Arduino-based code to publish DHT sensor data to the Mosquitto MQTT broker.

Output:



The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar shows icons for running, uploading, and other functions. The main editor displays the code for `esp_2_mos_simple.ino`. The code includes headers for `WiFi.h`, `PubSubClient.h`, and `DHT.h`. It defines pins and constants for a DHT22 sensor, a WiFi network named "Shahmir 2.4G", and an MQTT broker at IP 192.168.18.51 on port 1883. The code initializes a DHT sensor, a WiFi client, and an MQTT client. It then enters a loop that publishes temperature and humidity data to the MQTT broker every 5 seconds.

```
1 #include <WiFi.h>
2 #include <PubSubClient.h>
3 #include <DHT.h>
4
5 #define DHTPIN 4 // GPIO pin connected to DHT22
6 #define DHTTYPE DHT11 // DHT 22 (AM2302)
7 #define WIFI_SSID "Shahmir 2.4G"
8 #define WIFI_PASSWORD "shahzad1"
9 #define MQTT_SERVER "192.168.18.51" // Replace with your Windows PC's IP address on LAN
10 #define MQTT_PORT 1883
11
12 DHT dht(DHTPIN, DHTTYPE);
13 WiFiClient espClient;
14 PubSubClient client(espClient);
15
16 unsigned long lastMsg = 0;
17 const long interval = 1000; // Send every 5 seconds
18
19 void setup() {
20   Serial.begin(115200);
21   while (!Serial) continue;
22   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
23   while (WiFi.status() != WL_CONNECTED) {
24     delay(5000);
25     Serial.print("Connecting to WiFi network: ");
26   }
27   Serial.println("Connected to WiFi network");
28   client.setServer(MQTT_SERVER, MQTT_PORT);
29   client.connect("ESP32");
30   Serial.println("Connected to MQTT broker");
31 }
32
33 void loop() {
34   if (!client.connected()) {
35     Serial.println("Reconnecting to MQTT broker");
36     client.setServer(MQTT_SERVER, MQTT_PORT);
37     client.connect("ESP32");
38     Serial.println("Connected to MQTT broker");
39   }
40   if (millis() - lastMsg > interval) {
41     float h = dht.readHumidity();
42     float t = dht.readTemperature();
43     if (isnan(h) || isnan(t)) {
44       Serial.println("Failed to read from DHT sensor!");
45       return;
46     }
47     String msg = "Temperature: " + String(t, 1) + "C, Humidity: " + String(h, 1) + "%\n";
48     client.publish("esp32_data", msg);
49     lastMsg = millis();
50     Serial.println(msg);
51   }
52   delay(1000);
53 }
```

The serial monitor shows the output of the code. It displays the time taken to connect to the WiFi network and the MQTT broker. The output shows that the ESP32 is successfully connected to the WiFi network and the MQTT broker. The serial monitor also shows the temperature and humidity data being published to the MQTT broker.

```
21:54:55.717 -> Published Humidity: 51.00
21:54:55.717 -> waiting next value
21:54:56.762 -> Published Temperature: 30.10
21:54:56.762 -> Published Humidity: 51.00
21:54:56.762 -> waiting next value
21:54:57.711 -> Published Temperature: 30.10
21:54:57.753 -> Published Humidity: 51.00
21:54:57.753 -> waiting next value
```

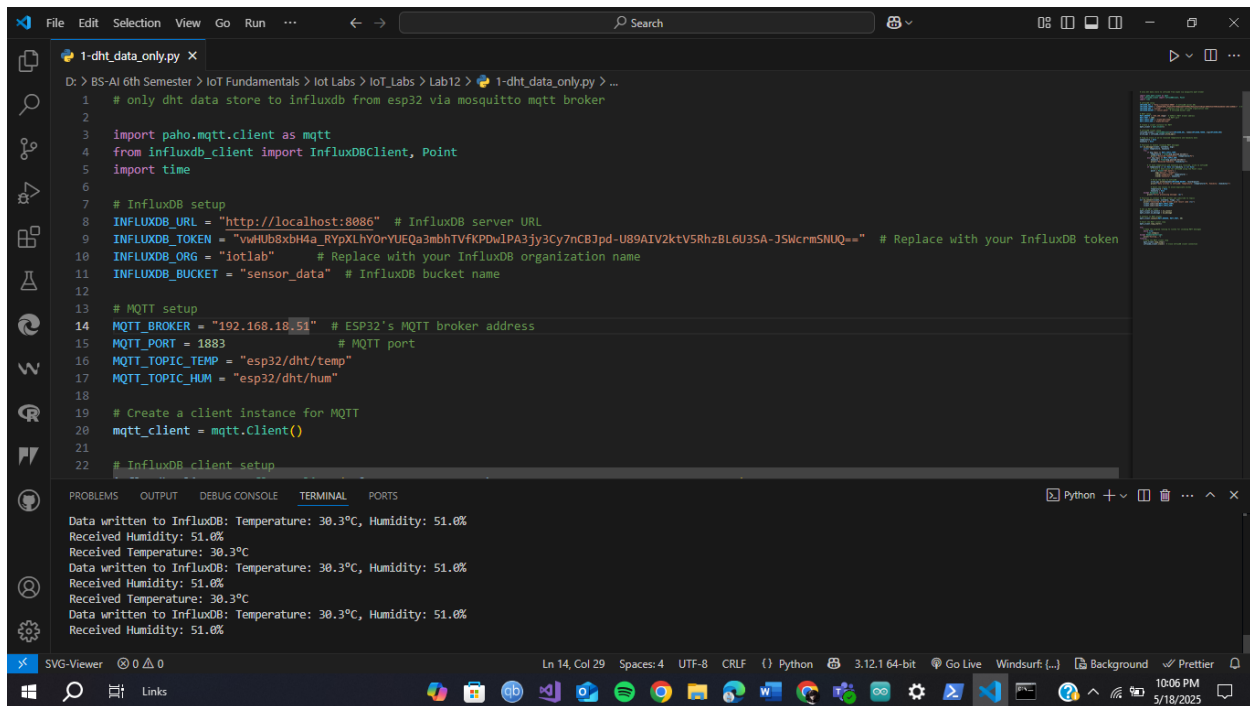
Observations:

The ESP32 was successfully connected to the WiFi network and communicated with the Mosquitto MQTT broker. Real-time temperature and

humidity data from the DHT11 sensor were accurately published to designated MQTT topics. This confirms the reliable integration of IoT sensor data with MQTT-based communication.

Task2: Execute the Python script 1-dht_data_only.py to store MQTT data in InfluxDB.

Output:

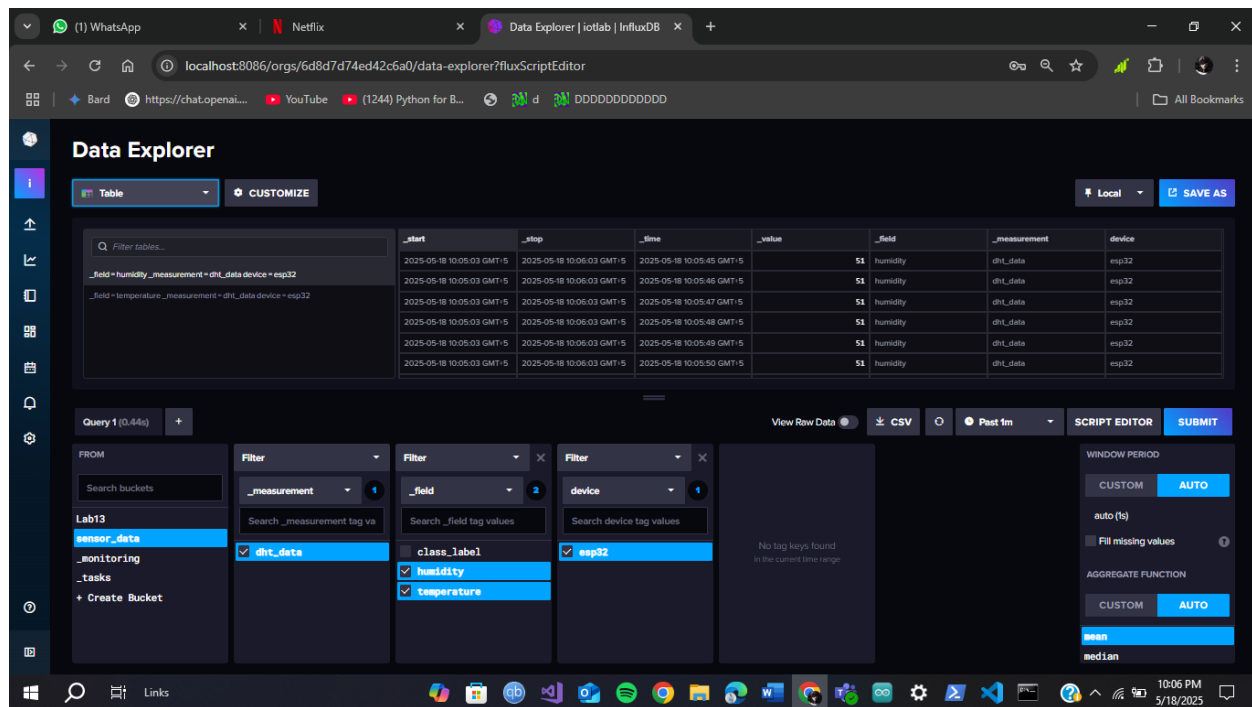


The screenshot displays a Visual Studio Code (VS Code) editor window. The main editor pane shows a Python script named `1-dht_data_only.py`. The script is configured to connect to an InfluxDB database and an MQTT broker. The terminal pane at the bottom shows the output of the script, indicating successful data transmission.

```
1 # only dht data store to influxdb from esp32 via mosquitto mqtt broker
2
3 import paho.mqtt.client as mqtt
4 from influxdb_client import InfluxDBClient, Point
5 import time
6
7 # InfluxDB setup
8 INFLUXDB_URL = "http://localhost:8086" # InfluxDB server URL
9 INFLUXDB_TOKEN = "vWmHUB8xbH4a_RYpXLhYOrYUEQa3mbhTVFKPDw1PA3jy3Cy7nCB3pd-U89AIV2ktV5RhzBL6U3SA-J5WcrnSNUQ==" # Replace with your InfluxDB token
10 INFLUXDB_ORG = "iotlab" # Replace with your InfluxDB organization name
11 INFLUXDB_BUCKET = "sensor_data" # InfluxDB bucket name
12
13 # MQTT setup
14 MQTT_BROKER = "192.168.10.51" # ESP32's MQTT broker address
15 MQTT_PORT = 1883 # MQTT port
16 MQTT_TOPIC_TEMP = "esp32/dht/temp"
17 MQTT_TOPIC_HUM = "esp32/dht/hum"
18
19 # Create a client instance for MQTT
20 mqtt_client = mqtt.Client()
21
22 # InfluxDB client setup
```

Terminal Output:

```
Data written to InfluxDB: Temperature: 30.3°C, Humidity: 51.0%
Received Humidity: 51.0%
Received Temperature: 30.3°C
Data written to InfluxDB: Temperature: 30.3°C, Humidity: 51.0%
Received Humidity: 51.0%
Received Temperature: 30.3°C
Data written to InfluxDB: Temperature: 30.3°C, Humidity: 51.0%
Received Humidity: 51.0%
```



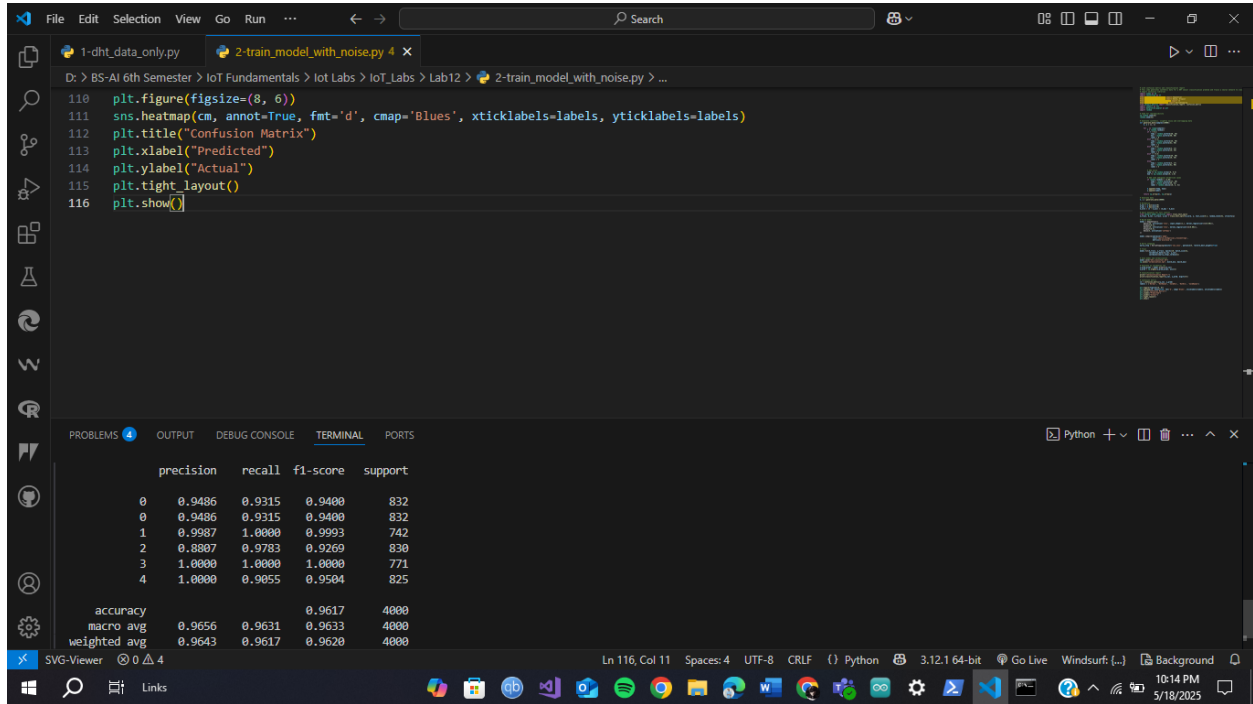
Observations:

After providing the correct credentials for Influxdb i.e Influxdb token, influxdb organization, influxdb bucket and IPv4 address for MQTT Broker, the python script runs successfully and show real time temperature humidity data at the terminal and on for displaying it on influxdb interface, run “influxd” command on cmd and provide the username and password which will show the real time data.

Task3: Run 2-train_model_with_noise.py and record the confusion matrix and classification report.

Output:

Confusion Matrix:



The screenshot shows a VS Code editor with two open files: `1-dht_data_only.py` and `2-train_model_with_noise.py 4`. The active file, `2-train_model_with_noise.py`, contains the following Python code:

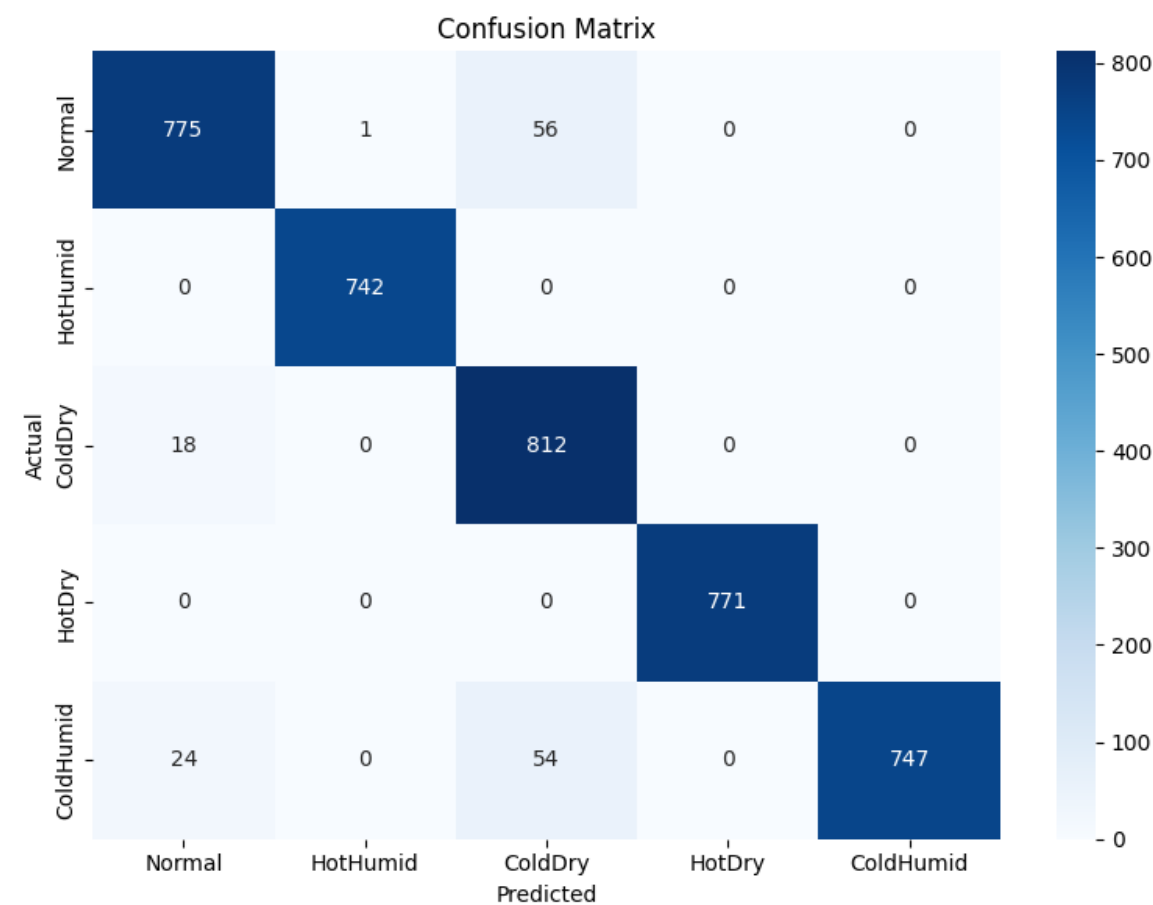
```
110 plt.figure(figsize=(8, 6))
111 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
112 plt.title("Confusion Matrix")
113 plt.xlabel("Predicted")
114 plt.ylabel("Actual")
115 plt.tight_layout()
116 plt.show()
```

The terminal window at the bottom displays the output of the script, which is a confusion matrix table:

	precision	recall	f1-score	support
0	0.9486	0.9315	0.9400	832
0	0.9486	0.9315	0.9400	832
1	0.9987	1.0000	0.9993	742
2	0.8807	0.9783	0.9269	830
3	1.0000	1.0000	1.0000	771
4	1.0000	0.9055	0.9504	825
accuracy			0.9617	4000
macro avg	0.9656	0.9631	0.9633	4000
weighted avg	0.9643	0.9617	0.9620	4000

The bottom status bar of VS Code shows the cursor is at line 116, column 11. The system tray at the bottom right indicates the time is 10:14 PM on 5/18/2025.

Classification Report:

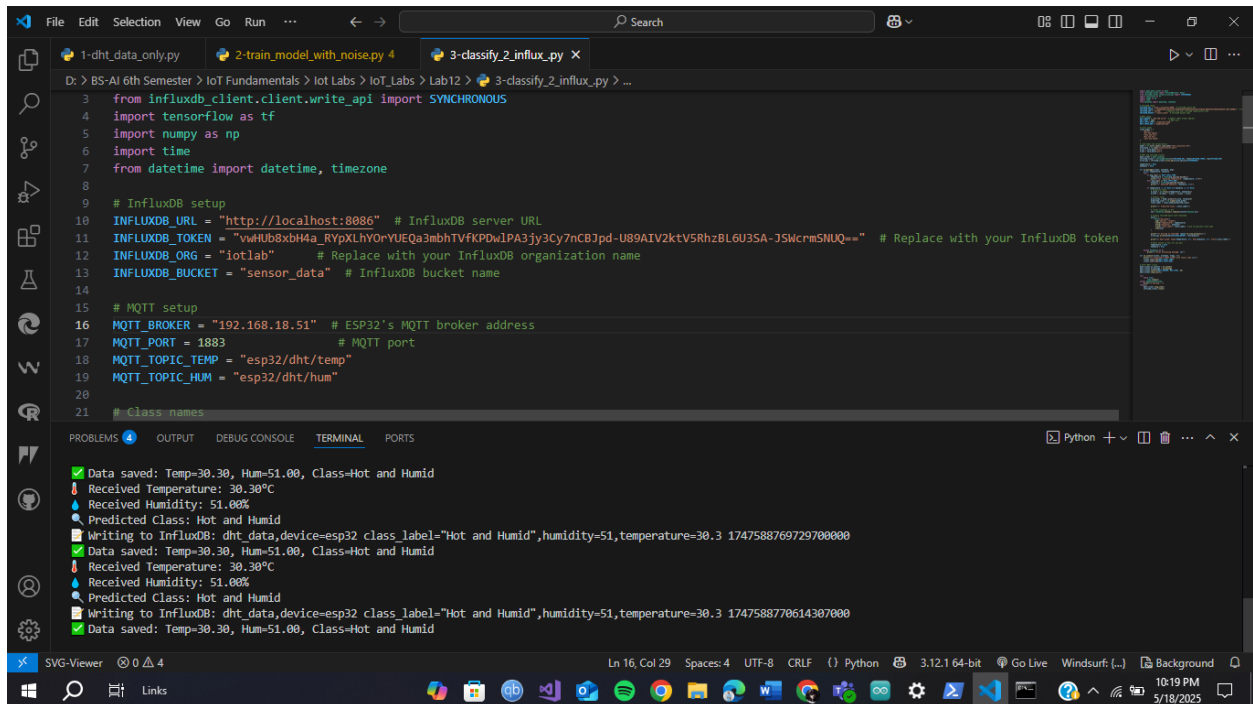


Observations:

The neural network accurately classified DHT sensor readings into five environmental conditions using synthetic, noisy data. The model showed balanced performance, with a clear confusion matrix and strong classification report indicating reliable generalization.

Task4: Execute 3-classify_2_influx.py and verify InfluxDB data for temperature, humidity, and classification results.

Output:



The screenshot displays a Visual Studio Code (VS Code) environment with a Python script named `3-classify_2_influx.py` open in the editor. The script is configured to connect to an InfluxDB instance at `localhost:8086` using a specific token and bucket named `sensor_data`. It also sets up an MQTT broker at `192.168.18.51` on port `1883`. The script's purpose is to receive temperature and humidity data via MQTT, predict a class (Hot or Humid), and store the results in InfluxDB.

The output window at the bottom of the VS Code interface shows the following log messages:

```
✓ Data saved: Temp=30.30, Hum=51.00, Class=Hot and Humid
⚠ Received Temperature: 30.30°C
⚠ Received Humidity: 51.00%
⚠ Predicted Class: Hot and Humid
📁 Writing to InfluxDB: dht_data,device=esp32 class_label="Hot and Humid",humidity=51,temperature=30.3 1747588769729700000
✓ Data saved: Temp=30.30, Hum=51.00, Class=Hot and Humid
⚠ Received Temperature: 30.30°C
⚠ Received Humidity: 51.00%
⚠ Predicted Class: Hot and Humid
📁 Writing to InfluxDB: dht_data,device=esp32 class_label="Hot and Humid",humidity=51,temperature=30.3 1747588770614307000
✓ Data saved: Temp=30.30, Hum=51.00, Class=Hot and Humid
```

The status bar at the bottom indicates the current file is `3-classify_2_influx.py`, located at `Ln 16, Col 29`, using `UTF-8` encoding and `CRLF` line endings. The Python interpreter is set to `3.12.1 64-bit`.

The screenshot displays the Data Explorer web interface in a browser. The top navigation bar includes tabs for WhatsApp, Netflix, and Data Explorer | InfluxDB. The address bar shows the URL: localhost:8086/orgs/6d8d7d74ed42c6a0/data-explorer?fluxScriptEditor. The interface features a sidebar with navigation icons and a main content area. The main area is divided into two sections: a table view and a query editor.

The table view shows a table with the following columns: `_time`, `class_label`, `humidity`, and `temperature`. The data rows are as follows:

<code>_time</code>	<code>class_label</code>	<code>humidity</code>	<code>temperature</code>
2025-05-18 10:19:27 GMT-5	Hot and Humid	S1	30.30
2025-05-18 10:19:27 GMT-5	Hot and Humid	S1	30.30
2025-05-18 10:19:28 GMT-5	Hot and Humid	S1	30.30
2025-05-18 10:19:29 GMT-5	Hot and Humid	S1	30.30
2025-05-18 10:19:30 GMT-5	Hot and Humid	S1	30.30
2025-05-18 10:19:31 GMT-5	Hot and Humid	S1	30.30

The query editor shows a Flux query:

```
1 from(bucket: "sensor_data")
2   |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
3   |> filter(fn: (r) => r["_measurement"] == "dht_data")
4   |> filter(fn: (r) => r["_field"] == "class_label" or r["_field"] == "humidity" or r["_field"] == "temperature")
5   |> filter(fn: (r) => r["device"] == "esp32")
6   |> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn: "_value")
7   |> keep(columns: ["_time", "temperature", "humidity", "class_label"])
```

The right sidebar contains a 'Filter Functions...' search bar and a list of transformations: `aggregate.rate`, `changeMomentumOscillator`, `columns`, `cov`, `covariance`, and `cumulativeSum`.

Observations:

This script integrates MQTT, a trained DHT classifier model, and InfluxDB to predict environmental conditions in real-time based on temperature and humidity. Predicted class labels and sensor data are successfully stored in InfluxDB for monitoring and analysis.
