FACULTY OF ELECTRICAL AND ELECTRONICS

ENGINEERING TECHNOLOGY

BVI 3114

TECHNOLOGY SYSTEM OPTIMIZATION II

MINI PROJECT

AI INTEGRATION SENSOR FORECASTING AND SUMMARIZATION

NO	STUDENT ID	STUDENT NAME
1	VC22015	ISYRAF HILMI BIN IMRAN
2	VC22030	RAZMIN BIN ABDUL RAHIM

DATE OF REPORT SUBMISSION	17.7.2025

TABLE OF CONTENT

1. INTRODUCTION

In today's smart lab environments, even minor environmental changes-like a spike in CO_2 or a shift in lighting-can compromise experiments, waste energy, or create safety risks. Traditionally, monitoring these issues has been reactive: something goes wrong, and we try to fix it. But what if we could predict and prevent those problems before they happen?

That's exactly what our PBL project aimed to achieve.

Our goal was to create a fully functional IoT-powered system that doesn't just monitor environmental data-but also predicts it and responds in real time. Using simple, affordable components and cloud services, we built a smart data forecasting and automation system that brings labs one step closer to full autonomy.

2. PROJECT OVERVIEW

Here's how we turned that vision into a working system:

- Hardware Setup: ESP32 + Sensors
- Sensor: Ultrasonic Distance Sensor
- Controller: ESP32 Microcontroller for real-time data collection
- Cloud Integration:
    - Sensor data is sent every 10 seconds to Google Sheets via Wi-Fi and a Google Apps Script Webhook.
    - This removes the need for local servers and ensures data is always accessible.
- Data Visualization: Google Looker Studio (formerly Data Studio) turns our raw sensor readings into real-time dashboards accessible from any browser or mobile device.
- Forecasting Intelligence:
    - We implemented short-term forecasting using:
        - EMA (Exponential Moving Average)
        - SMA (Simple Moving Average)
    - Forecasts predict the next 24 hours for temperature, light, CO_2 levels, and distance.
- Automation and Smart Responses
    - Based on predictions, the system can:
        - Trigger ventilation early if CO_2 is expected to rise.
        - Warn or reroute moving robots if obstacles are forecasted.
        - This helps shift lab operations from reactive to proactive.

3. CORE CODE AND IMPLEMENTATION

- Arduino (ESP32) Code Highlight
  - Reads ultrasonic sensor every 10 seconds.
  - Sends data to Google Sheets as JSON.
  - Uses HTTPClient and ArduinoJson libraries.
  - Connects via Wi-Fi with hardcoded credentials.
- GitHub Repo
  - https://github.com/RAZSTAR01/MiniProjectTSO2
- Google Apps Script Backend
  - Parses incoming JSON data.
  - Logs values into Google Sheets.
  - Forecast script (Simple Moving Average and Exponential Moving Average) predicts 24-hour trends.
  - Menu items added for manual controls and clearing old data

4. AI INTEGRATION

A. Real-Time Data Summary

- This section provides an interpretation of live sensor data collected over a short period. Its function is to:
  - ❖ Identify trends or anomalies in real-time sensor readings.
  - ❖ Highlight key patterns, such as:
    - Two distinct distance ranges.
    - Occasional "close proximity" events (sudden drops in distance).
    - Specific time ranges where notable changes occurred.
  - ❖ Offer a detailed breakdown of time-based sensor behavior (e.g., between 21:52 and 22:06).
  - ❖ Summarize behavior and suggest possible reasons (e.g., sensor type, environment, thresholds).
  - ❖ Help users understand unusual behavior in real-time conditions.

B. <u>Forecast Data Summary</u>

- This section evaluates predicted sensor data**,** looking at what is expected to happen in the future. Its function is to:
  - ❖ Analyze and explain the forecasted sensor trends over time.
  - ❖ Highlight phases like:
    - Initial decrease in readings.
    - Gradual increase.
    - Stabilization period.
  - ❖ Point out upper and lower bounds to understand the uncertainty in forecasts.
  - ❖ Summarize overall expected behaviour.

Overall, Purpose of Summarize Data

To make complex sensor data easier to understand, whether real-time or forecasted. It transforms raw data into meaningful insights so users can:

- Quickly detect patterns.
- Spot unusual changes.
- Anticipate future behaviour.
- Make informed decisions or investigations based on the sensor's performance.

C. Gemini AI

This system demonstrates how real-time sensor data can be logged and forecasted using Google Apps Script. The data is automatically recorded into Google Sheets, where forecasting is done using Exponential and Simple Moving Averages (EMA & SMA). To make the data easier to understand, Gemini AI is used to generate smart summaries that highlight key trends and insights.

- Process
  - User Start Summary

```
// Get summaries
const realtimeData = realtimeSheet.getDataRange().getValues();
const forecastData = forecastSheet.getDataRange().getValues();

const realtimePrompt = "This is real-time sensor data:\n" + convertToCSV(realtimeData) + "\nSummarize the trends.";
const forecastPrompt = "This is forecast sensor data:\n" + convertToCSV(forecastData) + "\nSummarize expected trends.";

const realtimeSummary = getGeminiSummary(realtimePrompt);
const forecastSummary = getGeminiSummary(forecastPrompt);

// Output to Summary sheet
summarySheet.clear();
summarySheet.getRange("A1").setValue("Real-Time Data Summary:");
summarySheet.getRange("A2").setValue(realtimeSummary);
summarySheet.getRange("A4").setValue("Forecast Data Summary:");
summarySheet.getRange("A5").setValue(forecastSummary);
summarySheet.getRange("A7").setValue("Last Updated: " + new Date().toLocaleString());

Logger.log("Real-Time Summary:\n" + realtimeSummary);
Logger.log("Forecast Summary:\n" + forecastSummary);
}
```

  - Data Prepared and Collected (Google Sheets)

	Timestamp	Distance
2	15/05/2025 21:52:02	235.263
3	15/05/2025 21:52:12	5.899
4	15/05/2025 21:52:21	235.229
5	15/05/2025 21:52:31	235.297
6	15/05/2025 21:52:41	235.705
7	15/05/2025 21:52:51	235.348
8	15/05/2025 21:53:01	235.314
9	15/05/2025 21:53:11	235.263
10	15/05/2025 21:53:21	5.508
11	15/05/2025 21:53:31	516.732
12	16/05/2025 21:53:41	516.477
13	16/05/2025 21:53:52	31.45
14	16/05/2025 21:54:01	516.511
15	16/05/2025 21:54:12	516.545
16	16/05/2025 21:54:22	516.562
17	16/05/2025 21:54:32	516.443
18	16/05/2025 21:54:42	35.071
19	16/05/2025 21:54:59	0.612
20	16/05/2025 22:01:09	233.988
21	16/05/2025 22:01:18	234.362
22	17/05/2025 22:01:28	234.277
23	17/05/2025 22:01:40	3.621
24	17/05/2025 22:01:49	9.35
25	17/05/2025 22:01:58	255.527

	Timestamp	Forecast	Upper	Lower
2	18/05/2025 23:05:18	203.84	224.23	183.46
3	19/05/2025 00:05:18	198.84	218.73	178.96
4	19/05/2025 01:05:18	223.04	245.35	200.74
5	19/05/2025 02:05:18	221.25	243.37	199.12
6	19/05/2025 03:05:18	219.15	241.06	197.23
7	19/05/2025 04:05:18	216.69	238.36	195.02
8	19/05/2025 05:05:18	213.8	235.18	192.42
9	19/05/2025 06:05:18	215.46	237.01	193.92
10	19/05/2025 07:05:18	218.23	240.06	196.41
11	19/05/2025 08:05:18	217.43	239.17	195.69
12	19/05/2025 09:05:18	216.79	238.47	195.11
13	19/05/2025 10:05:18	216.4	238.04	194.76
14	19/05/2025 11:05:18	216.35	237.99	194.72
15	19/05/2025 12:05:18	216.78	238.46	195.1
16	19/05/2025 13:05:18	217	238.7	195.3
17	19/05/2025 14:05:18	216.79	238.47	195.11
18	19/05/2025 15:05:18	216.69	238.36	195.02
19	19/05/2025 16:05:18	216.67	238.34	195
20	19/05/2025 17:05:18	216.71	238.38	195.04
21	19/05/2025 18:05:18	216.77	238.45	195.1
22	19/05/2025 19:05:18	216.77	238.45	195.1
23	19/05/2025 20:05:18	216.73	238.41	195.06
24	19/05/2025 21:05:18	216.72	238.4	195.05
25	19/05/2025 22:05:18	216.73	238.4	195.06

- Send to Gemini AI

```
function getGeminiSummary(prompt) {
  const payload = {
    contents: [{ parts: [{ text: prompt }] }]
  };
  const options = {
    method: 'POST',
    contentType: 'application/json',
    payload: JSON.stringify(payload),
    muteHttpExceptions: true
  };
  const url = 'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=' + apiKey;
  const response = UrlFetchApp.fetch(url, options);
  const result = JSON.parse(response.getContentText());
  return result.candidates?.[0]?.content?.parts?.[0]?.text || "No summary returned.";
}

// Get summaries
const realtimeData = realtimeSheet.getDataRange().getValues();
const forecastData = forecastSheet.getDataRange().getValues();

const realtimePrompt = "This is real-time sensor data:\n" + convertToCSV(realtimeData) + "\nSummarize the trends.";
const forecastPrompt = "This is forecast sensor data:\n" + convertToCSV(forecastData) + "\nSummarize expected trends.";

const realtimeSummary = getGeminiSummary(realtimePrompt);
const forecastSummary = getGeminiSummary(forecastPrompt);
```

- Summary is Received and Saved to Apps Script

4	Forecast Data Summary:
	Okay, let's analyze the forecast data and summarize the expected trends.
	Overall Trend:
	The forecast shows a relatively stable trend throughout Thursday, May 15, 2025. The "Forecast" values hover around 216-217 for most of the day.
	Detailed Observations:
	* **Initial Decrease:** The forecast starts at 203.84 on Wed May 14 2025 23:05:17 GMT+0700 (Indochina Time) and decreases to 198.84 at Thu May 15 2025 00:05:17 GMT+0700 (Indochina Time).
5	* **Morning Increase:** From midnight to 1 AM on Thursday, there's a noticeable jump in the forecast, rising to 223.04.
	* **Gradual Decline:** After the initial rise, the "Forecast" value tends to decrease from 223.04 at 1:05 AM to around 213.8 at 5:05 AM.
	* **Stabilization:** From 6:05 AM onward, the forecast stabilizes, fluctuating only slightly around the 216-217 range. The "Forecast" values remain relatively constant for the rest of the day.
	* **Upper and Lower Bounds:** The "Upper" and "Lower" values also remain relatively stable, indicating a consistent level of uncertainty associated with the forecast.
	Summary:
	The sensor data forecasts a value decreasing in the first few hours of May 15, 2025, starting from 223.04 and stabilizing around 216-217 after 6:05 AM (Indochina Time) and remains constant for the rest of the day.

5. VISUALIZATION & APP DEVELOPMENT

- Google Sheet (Live Data)
    - Our Google Sheet acts as the live database and is constantly updated with new readings.
- Looker Studio Dashboard
    - A clean, mobile-friendly dashboard shows all environmental metrics in real-time.
- Live Dashboard
- Android App (Made with Android Studio)
    - Uses a WebView to embed the dashboard.
    - Includes a Refresh button.
    - Features zoom and scroll support.
    - Designed with clean UI for usability.

6. SYSTEM HIGHLIGHTS & MODIFICATION

- We tweaked the ESP32 to also include a ultrasonic distance sensor.
- Data interval was changed from 10 minutes to 10 seconds for real-time behaviour.
- Mobile app UI was enhanced for a better user experience.
- Dashboard layout was optimized for small screen displays.

7. CHALLENGES WE FACED

CHALLENGE	SOLUTION
Data delay from ESP32	Reduced interval to 10 seconds
Unstable Wi-Fi	Added retry logic and feedback for reconnection
Limited forecasting tools	Used forecasting method before upgrading
Google Scripts limits	Optimized locking and payload handling

8. CONCLUSION

This PBL journey helped us explore the complete lifecycle of an IoT data system from sensing to cloud logging, forecasting, automation, and app integration. The most exciting takeaway? We built a system that doesn't just react it thinks ahead.

The skills gained include:

- IoT hardware integration.
- Cloud APIs (Google Apps Script)
- App development with Android Studio.
- Dashboard design with Looker Studio.

With just an ESP32 and a sensor, we made a lab smarter. And if we can do it here imagine the potential for smart classrooms, greenhouses, or even urban environments.

9. REFERENCES

- Arduino Code

```
#include <WiFi.h>
#include
<HTTPClient.h>
#include
<ArduinoJson.h>
#include <Ultrasonic.h>
// WiFi credentials
const char* ssid = "vivo V23 5G";
const char* password =
"123456789";
// Google Script ID - deploy as web app and get the
URL const char* scriptURL =
"https://script.google.com/macros/s/AKfycbz8tL6KiAsTixmGAz9g8FlHeyxLuQrTqSXmqcvMtZ
V
-P2PpRFp7_V4ZFJyI7zOALtkV/exec";
// Ultrasonic sensor configuration
const int ultrasonicPin = 13; // Digital pin connected to ultrasonic sensor
Ultrasonic ultrasonic(ultrasonicPin);
// Data sending interval (in milliseconds)
const unsigned long sendInterval = 10000; // 10
seconds unsigned long previousMillis = 0;
void setup() {
 // Initialize serial
 communication
 Serial.begin(115200);
 delay(1000);
 Serial.println("ESP32 Ultrasonic Ranger Data Logger");

 // Connect to WiFi
 WiFi.begin(ssid, password);
 Serial.print("Connecting to
 WiFi");
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println();
 Serial.print("Connected to WiFi with IP: ");
 Serial.println(WiFi.localIP());
}
```

```
void loop() {
 unsigned long currentMillis = millis();

 // Check if it's time to send data
 if (currentMillis - previousMillis >= sendInterval) {
 previousMillis = currentMillis;

 // Read distance from ultrasonic
 sensor (in cm) long distance =
 ultrasonic.read();

 Serial.print("Distan
 ce: ");
 Serial.print(distanc
 e); Serial.println("
 cm");

 // Send data to Google Sheets
 sendDataToGoogleSheets(distance);
 }
}
void sendDataToGoogleSheets(long distance) {
 // Check WiFi connection
 if (WiFi.status() !=
 WL_CONNECTED) {
 Serial.println("WiFi not
 connected"); return;
 }

 HTTPClient http;
 http.begin(scriptUR
 L);
 http.addHeader("Content-Type",  "application/json");

 // Create JSON
 data
 StaticJsonDocumen
 t<200> doc;
 doc["distance"] =
 distance;

String
 jsonString;
 serializeJson(do
 c, jsonString);
```

```
// Send HTTP POST request
int httpResponseCode = http.POST(jsonString);

if
(httpResponseCode
> 0) { String
response =
http.getString();
Serial.println("HTTP Response code: " +
String(httpResponseCode)); Serial.println("Response: "
+ response);
} else {
Serial.print("Error on sending POST: ");
Serial.println(httpResponseCode);
}

http.end();
```

- Google Apps Script Code

```
// Script to receive sensor data from ESP32 and log it to
Google Sheets function doGet(e) {
 return handleResponse(e);
}
function
 doPost(e) {
 return
 handleRespo
 nse(e);
}
function handleResponse(e) {
 // Process the incoming request
 var lock = LockService.getScriptLock();
 lock.tryLock(5000); // Wait 10 seconds for other processes to complete

 try {
 // Get the active sheet
 var spreadsheet =
SpreadsheetApp.openByUrl("https://docs.google.com/spreadsheets/d/1m3oH-
Ay2DI6iSvqG7pEf70r8MebjJTJhXH4hbwpEDcE/edit");
 var sheet = spreadsheet.getSheetByName("Sheet1");


 // Parse the
 incoming data
 var payload;
 if (e.postData &&
 e.postData.contents) { payload
 =
 JSON.parse(e.postData.content
 s);
```

```
        } else if
        (e.parameter)
        { payload =
        e.parameter;
        } else {
        return
        ContentService.createTextOutput(JSON.stringi
        fy({ 'status': 'error',
        'message': 'No data received'
        })).setMimeType(ContentService.MimeType.JSON);
        }

        // Prepare data array
        for the sheet var
        timestamp = new
        Date();
        var data = [timestamp];

        // Get sensor data based on what's available in the payload
        // Add appropriate sensor values to
       the data array if (payload.temperature
       !== undefined)
       data.push(parseFloat(payload.tempera
       ture));
        if (payload.humidity !==
       undefined)
       data.push(parseFloat(payload
       .humidity)); if
       (payload.moisture !==
       undefined)
       data.push(parseFloat(payload
       .moisture));
        if (payload.light !== undefined)
        data.push(parseFloat(payload.light)); if (payload.motion
        !== undefined) data.push(payload.motion);
        if (payload.distance !== undefined)
       data.push(parseFloat(payload.distance));

        // Insert data into the next row
        sheet.appendRow(data);

// Return success response
          return ContentService.createTextOutput(JSON.stringify({
        'status': 'success',
        'timestamp': timestamp.toString()
        })).setMimeType(ContentService.MimeType.JSON);

        } catch (error) {
        // Return error response
        return
        ContentService.createTextOutput(JSON.stringi
        fy({ 'status': 'error',
        'message': error.toString()
        })).setMimeType(ContentService.MimeType.JSON);
```

```
 } finally {
 lock.releaseLo
 ck();
 }
}
// Add menu to
sheet function
onOpen() {
 var ui =
 SpreadsheetApp.getUi();
 ui.createMenu('Sensor Data')
 .addItem('Clear All Data', 'clearData')
 .addToUi();
}
// Function to clear all data
except headers function
clearData() {
 var sheet =
 SpreadsheetApp.getActiveSpreadsheet().getActiveSheet(
 ); var lastRow = sheet.getLastRow();

 if (lastRow > 1) {
 sheet.deleteRows(2,
 lastRow - 1);
 }

 SpreadsheetApp.getUi().alert('All sensor data has been cleared!');
}

// Add this function to your existing Google Apps Script

// Function to generate forecasts (runs on time trigger or manual
execution) function generateForecasts() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Sheet1");
  var forecastSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Forecasts");

  // If forecast sheet doesn't
  exist, create it if
  (!forecastSheet) {
    forecastSheet = SpreadsheetApp.getActiveSpreadsheet().insertSheet("Forecasts");
    // Add headers based on your sensor type
    forecastSheet.appendRow(["Timestamp", "Forecasted Value", "Upper Bound",
"Lower Bound"]);
  }

  // Get historical data (last 24 hours or maximum available)
  var dataRange = sheet.getRange(2, 1, sheet.getLastRow()-1,
  sheet.getLastColumn()); var values = dataRange.getValues();

// Extract timestamps and sensor values


  var
```

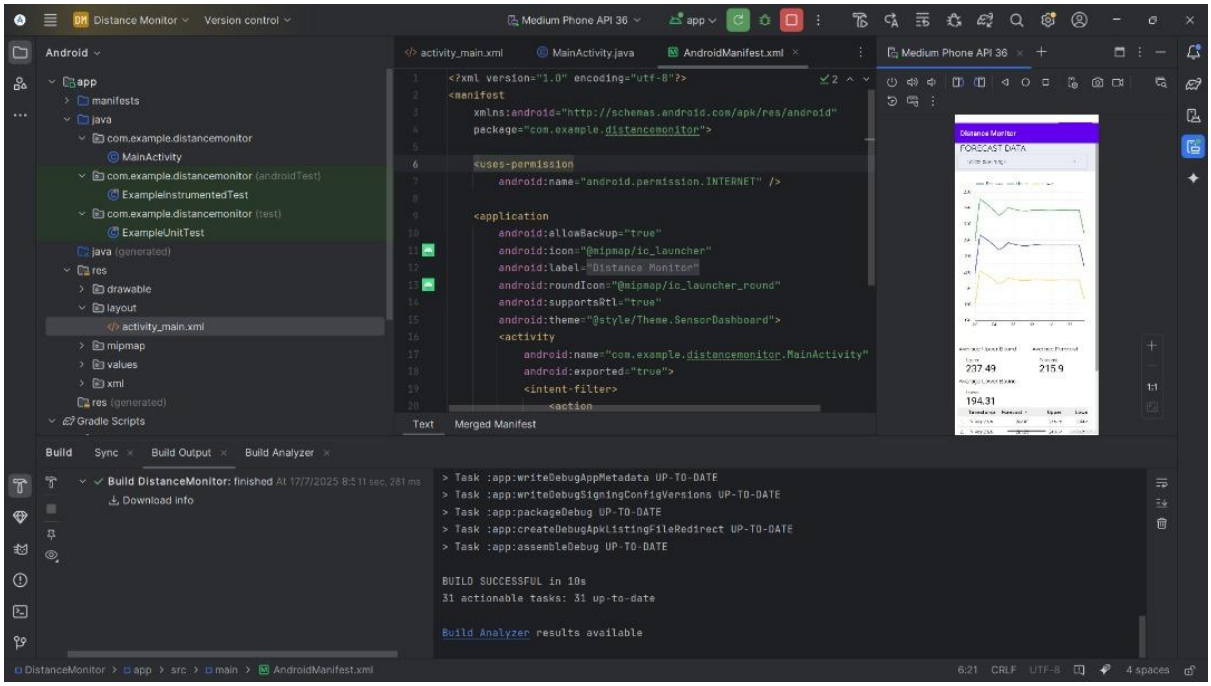
```
  timestam
  ps = [];
  var
  sensorVa
  lues = [];

  for (var i = 0; i < values.length; i++) {
    timestamps.push(values[i][0]); // Assuming timestamp is in column
    A sensorValues.push(values[i][1]); // Assuming sensor value is in
    column B
  }

  // Calculate forecasts using your chosen algorithm
  var forecasts = calculateForecasts(timestamps, sensorValues);

  // Clear previous forecasts
  if (forecastSheet.getLastRow() > 1) {
    forecastSheet.getRange(2, 1, forecastSheet.getLastRow()-1, 4).clear();
  }

  // Add new forecasts
  for (var i = 0; i <
    forecasts.length; i++) {
    forecastSheet.appendRow([
    forecasts[i].timestamp,
    forecasts[i].forecastValue,
    forecasts[i].upperBound,
    forecasts[i].lowerBound
    ]);
  }
}

// Implement your chosen forecasting
algorithm function
calculateForecasts(timestamps,
values) { var forecasts = [];

  // EXAMPLE: Simple Moving Average implementation
  // Replace with your chosen algorithm
  var windowSize = 6; // For 6-hour moving average

  // Generate forecasts for next 24 hours (at 1-hour intervals)
  var lastTimestamp = new Date(timestamps[timestamps.length - 1]);

for (var i = 1; i <= 24; i++) {
  var nextTimestamp = new Date(lastTimestamp.getTime() + (i * 60 * 60 * 1000));

  // Calculate forecast using rolling window
  var forecastValue = calculateSMA(values, windowSize);

  // Round to 2 decimal places
  forecastValue = Math.round(forecastValue * 100) / 100;

  // Add forecasted value to the values array for rolling updates
  values.push(forecastValue);
```

```
      // Add forecast with
      bounds forecasts.push({
        timestamp:
        nextTimestamp,
        forecastValue:
        forecastValue,
        upperBound: Math.round(forecastValue * 1.1 * 100) / 100,
      lowerBound: Math.round(forecastValue * 0.9 * 100) / 100
    });
  }


   return forecasts;
  }
  // Example: Simple Moving Average implementation
  function calculateSMA(values, windowSize) {
    if (values.length < windowSize) {
      windowSize = values.length; // Use all available data if not enough
    }

    var sum = 0;
    for (var i = values.length - windowSize; i < values.length; i++) {
      sum += values[i];
    }
    return sum / windowSize;
  }

  // Add button to sheet menu to generate forecasts manually
  function onOpen() {
    var ui = SpreadsheetApp.getUi();
    ui.createMenu('Sensor Data')
      .addItem('Generate Forecasts', 'generateForecasts')
      .addItem('Clear All Data', 'clearData')
      .addToUi();
  }
```

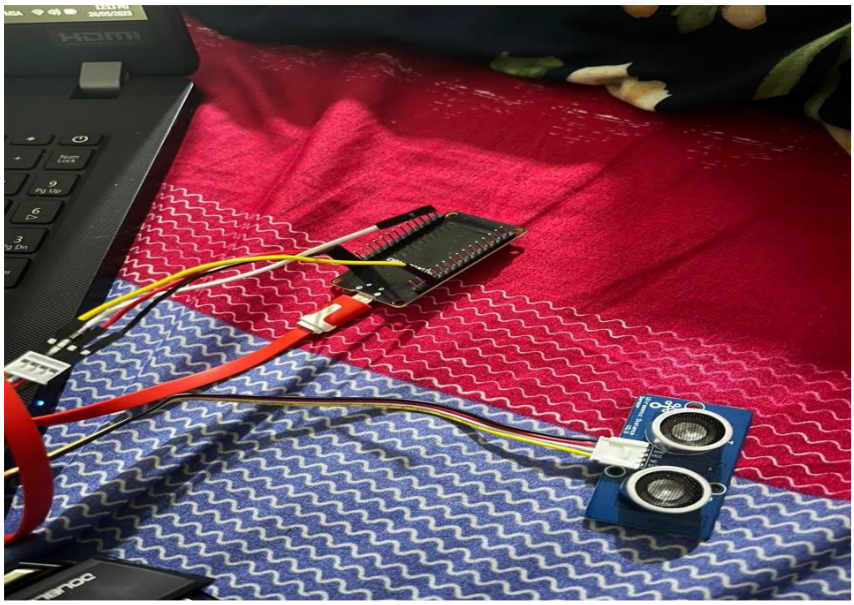
- Google Sheets Link
  - https://docs.google.com/spreadsheets/d/1i_UG_e6P4jl7CRTIaKHLuHTGTJW9Z1IRs1d-MNkviP4/edit?gid=0#gid=0

- Looker Studio Dashboard Link
  - https://lookerstudio.google.com/embed/reporting/f33214b6-2a00-4b58-99f8-512dea7a9f58/page/2EZRF

- Android Studio
    - As part of the project, we developed a mobile application using Android Studio. The app integrates real-time data by embedding an iframe from Looker Studio, displaying live readings from the distance sensors in a clean, user-friendly interface.



- Sensor Setup



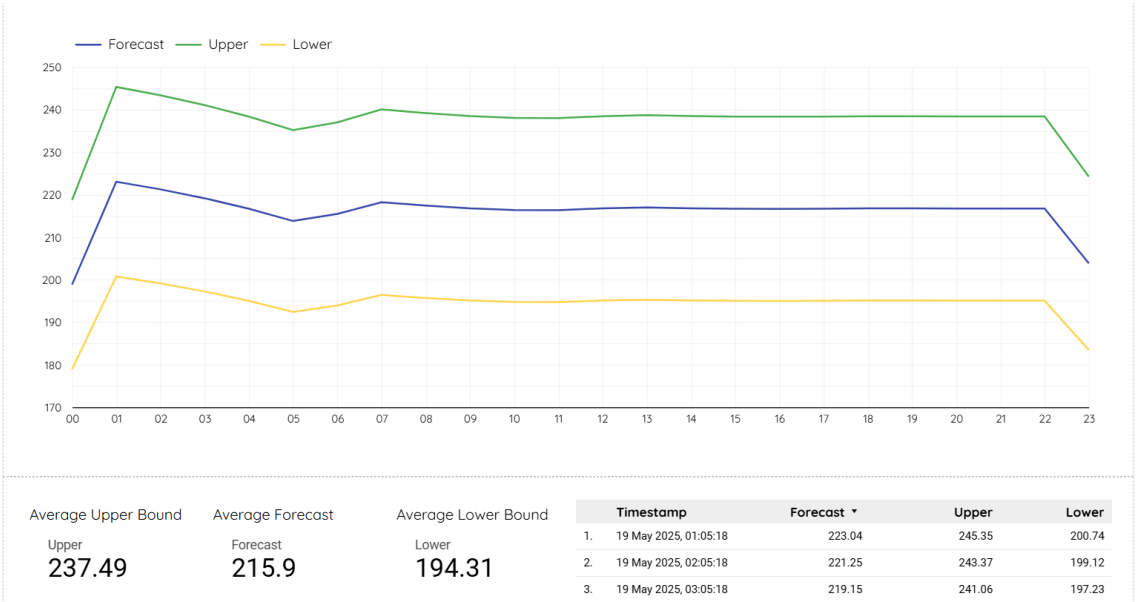
Ultrasonic Distance Sensor:

- Connect VCC to 5V on ESP32
- Connect GND to GND on ESP32
- Connect SIG to a digital GPIO

- Google Sheets

1	Timestamp	Distance
2	15/05/2025 21:52:02	235.263
3	15/05/2025 21:52:12	5.899
4	15/05/2025 21:52:21	235.229
5	15/05/2025 21:52:31	235.297
6	15/05/2025 21:52:41	235.705
7	15/05/2025 21:52:51	235.348
8	15/05/2025 21:53:01	235.314
9	15/05/2025 21:53:11	235.263
10	15/05/2025 21:53:21	5.508
11	15/05/2025 21:53:31	516.732
12	16/05/2025 21:53:41	516.477
13	16/05/2025 21:53:52	31.45
14	16/05/2025 21:54:01	516.511
15	16/05/2025 21:54:12	516.545
16	16/05/2025 21:54:22	516.562
17	16/05/2025 21:54:32	516.443
18	16/05/2025 21:54:42	35.071
19	16/05/2025 21:54:59	0.612
20	16/05/2025 22:01:09	233.988
21	16/05/2025 22:01:18	234.362
22	17/05/2025 22:01:28	234.277
23	17/05/2025 22:01:40	3.621
24	17/05/2025 22:01:49	9.35

- Forecast Chart & Visualization

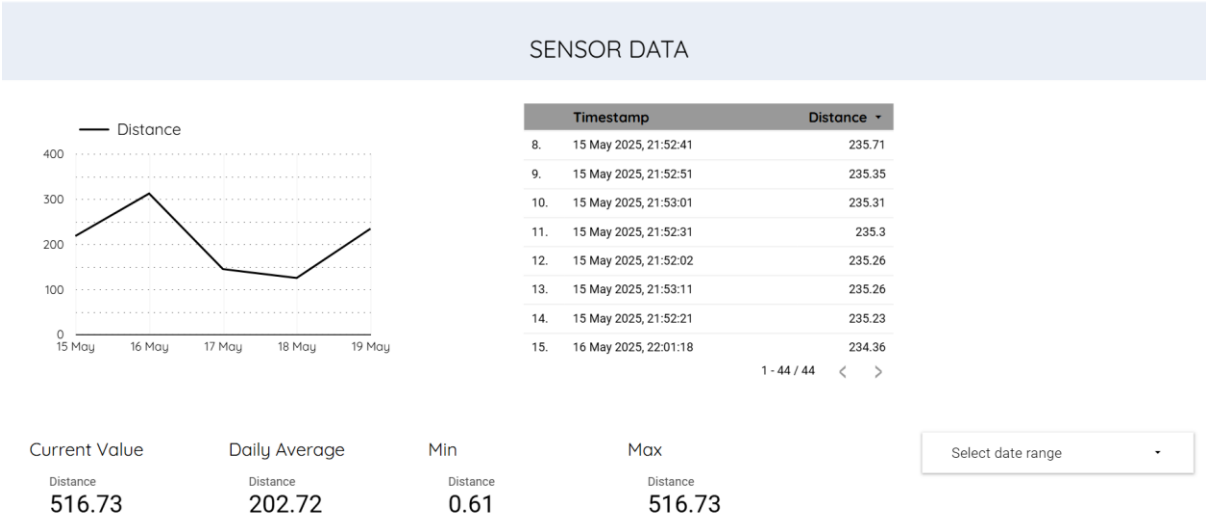


The chart is generated dynamically within the Forecasts sheet.

Line charts show:

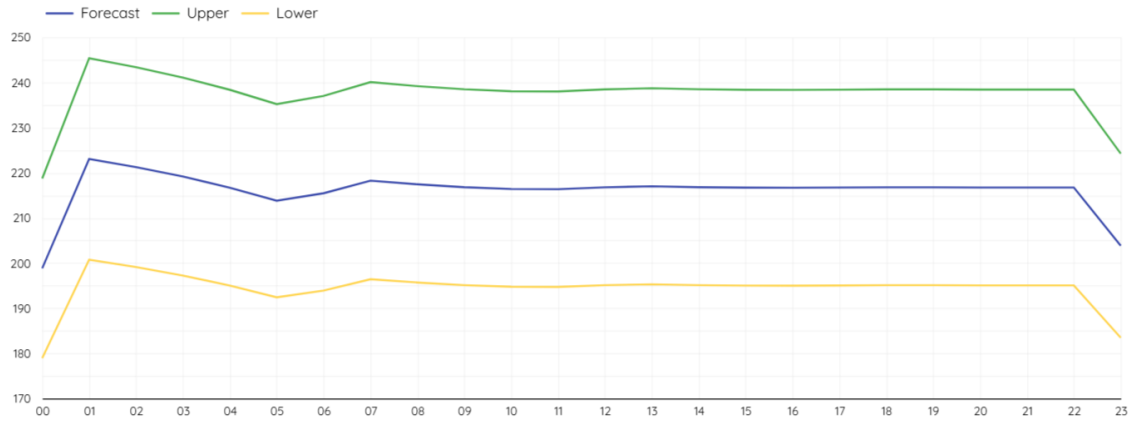
- Forecast and distance
- Upper/lower bounds for future estimates

- Complete Looker Studio Dashboard



FORECAST DATA

Select date range



Average Upper Bound	Average Forecast	Average Lower Bound	Timestamp	Forecast	Upper	Lower

SUMMARY FOR REAL-TIME

Select date range

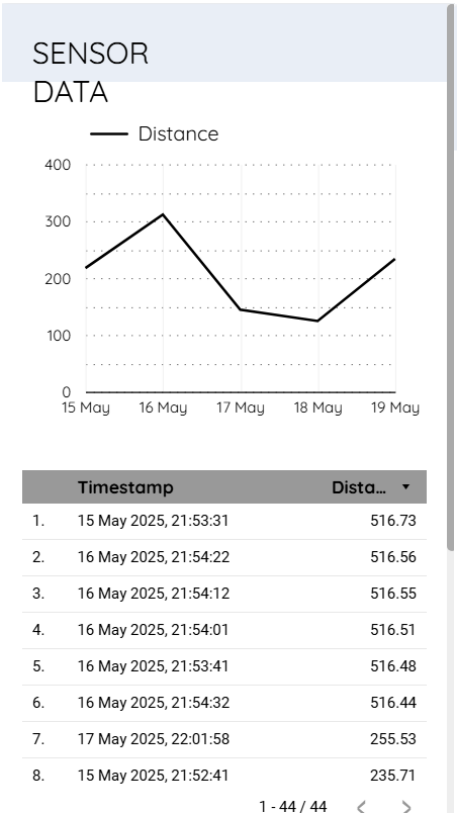
1	Real-Time Data	Okay, let's break down the trends in this sensor data. Overall Observations: Two Distinct Distance Ranges: The data largely clusters around two distance ranges: One around ~234 and another between 5-35. I noticed a third distinct range between 516-517 in the 21:53-21:54 timeframe. Intermittent Close Proximity: The sensor occasionally registers very close distances (around 0-12), which suggests a possible event or change in the environment. Detailed Trend Analysis: - 21:52 - 21:53: Starts with distances around 235, then sharply drops to ~5.899, indicating something moved close to the sensor. It returns to the 235 range before another drop to ~5.508. Finally, we see some distances within a range of 516.477-516.732. - 21:53 - 21:54: A shift in range between 31.45-516.562 before moving back to ~35.071 and a sharp drop to 0.612 at 21:54:59. - 22:01 - 22:05: There's a period where the distance is roughly in the ~234 range, with intermittent spikes/drops to lower values. This includes drops to values near 3.621, 9.35 and 3.706. Afterwards there are a few instances of drops in range between 1.768-12.274. This ends with a drop to ~53.635 at 22:04:40. Summary: - The sensor data shows that the measured distance is generally far. However, there are intermittent periods where an object seems to move very close to the sensor, as evidenced by the drops in distance. There are a few periods of extended distance between 516-517 as well. The frequency of these close proximity events varies. There is also a long gap between 21:54:59 and 22:01:08. To get a better understanding of what's happening, consider these factors: - Sensor Type: What kind of sensor is being used (e.g., ultrasonic, infrared)? This can help understand its limitations and potential sources of error.

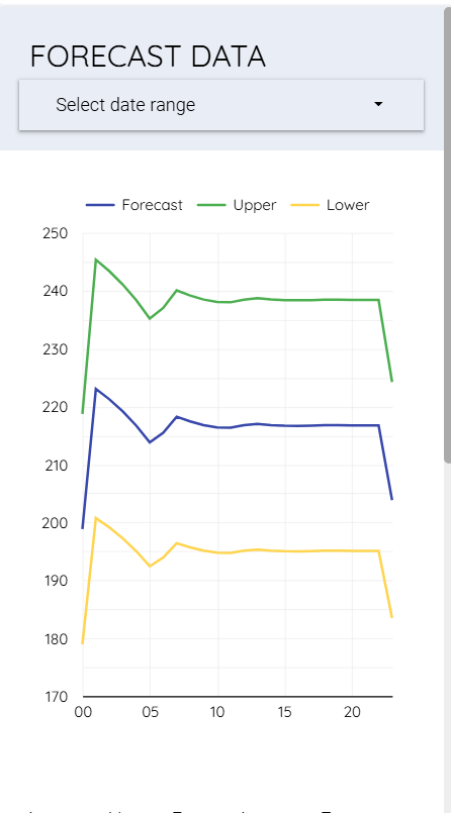
SUMMARY FOR FORECAST

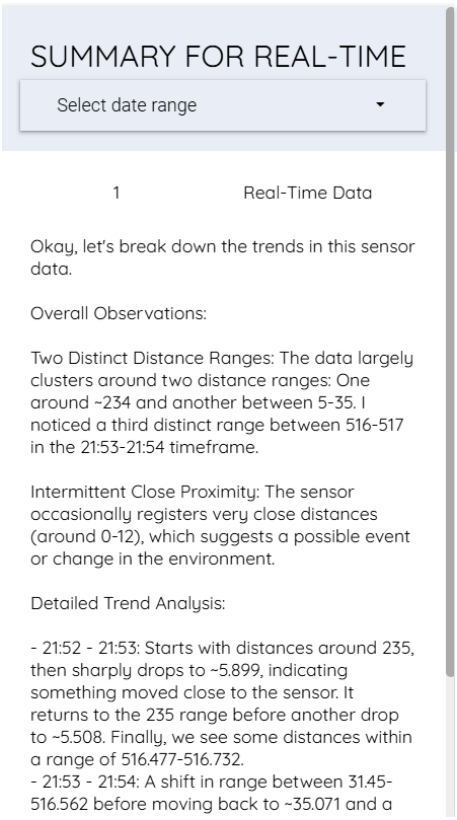
Select date range

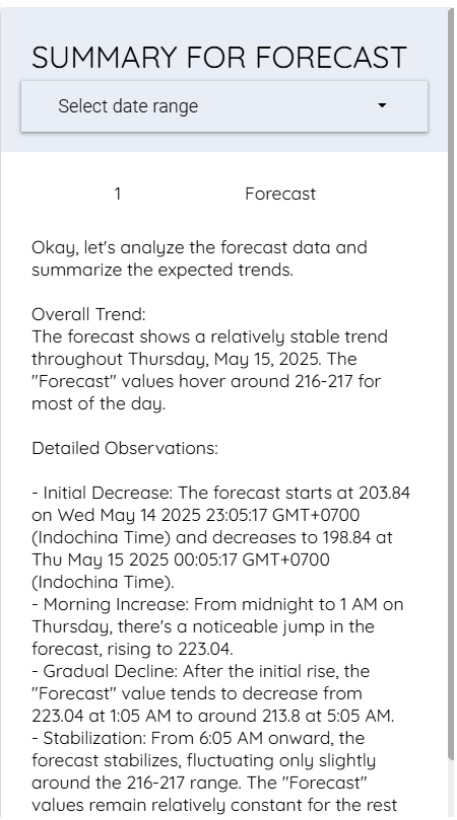
1	Forecast	Okay, let's analyze the forecast data and summarize the expected trends. Overall Trend: The forecast shows a relatively stable trend throughout Thursday, May 15, 2025. The "Forecast" values hover around 216-217 for most of the day. Detailed Observations: - Initial Decrease: The forecast starts at 203.84 on Wed May 14 2025 23:05:17 GMT+0700 (Indochina Time) and decreases to 198.84 at Thu May 15 2025 00:05:17 GMT+0700 (Indochina Time). - Morning Increase: From midnight to 1 AM on Thursday, there's a noticeable jump in the forecast, rising to 223.04. - Gradual Decline: After the initial rise, the "Forecast" value tends to decrease from 223.04 at 1:05 AM to around 213.8 at 5:05 AM. - Stabilization: From 6:05 AM onward, the forecast stabilizes, fluctuating only slightly around the 216-217 range. The "Forecast" values remain relatively constant for the rest of the day. - Upper and Lower Bounds: The "Upper" and "Lower" values also remain relatively stable, indicating a consistent level of uncertainty associated with the forecast. Summary: The sensor data forecasts a value decreasing in the first few hours of May 15, 2025, starting from 223.04 and stabilizing around 216-217 after 6:05 AM (Indochina Time) and remains constant for the rest of the day. The uncertainty, as represented by the "Upper" and "Lower" bounds, is relatively consistent throughout the day.

- Images of Android App









- Google App Script (Full Gemini AI Code)

```
function onOpen() {
const ui = SpreadsheetApp.getUi();
ui.createMenu("🔍 Gemini Summary")
.addItem("Summarize All Sensor Data",
"summarizeAllSensorDataWithGemini")
.addToUi();
}

function summarizeAllSensorDataWithGemini() {
const apiKey = 'AIzaSyDpN0BWNVj0HSM3K2maYdFel98_MtJcxRA';
// 🔐 Replace this with your Gemini API key
const ss = SpreadsheetApp.getActiveSpreadsheet();

const realtimeSheet = ss.getSheetByName('Sheet1');
const forecastSheet = ss.getSheetByName('Forecasts');
const summarySheet = ss.getSheetByName('Summary');

function convertToCSV(data) {
return data.map(row => row.join(',')).join('\n');
}

function getGeminiSummary(prompt) {
const payload = {
contents: [{ parts: [{ text: prompt }] }]
};
const options = {
method: 'POST',
contentType: 'application/json',
payload: JSON.stringify(payload),
muteHttpExceptions: true
};
const url =
'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=' + apiKey;
const response = UrlFetchApp.fetch(url, options);
const result = JSON.parse(response.getContentText());
return result.candidates?.[0]?.content?.parts?.[0]?.text
|| "No summary returned.";
}

// Get summaries
const realtimeData =
realtimeSheet.getDataRange().getValues();
const forecastData =
forecastSheet.getDataRange().getValues();

const realtimePrompt = "This is real-time sensor data:\n"
+ convertToCSV(realtimeData) + "\nSummarize the trends.";
```

```
-   const forecastPrompt = "This is forecast sensor data:\n" +
    convertToCSV(forecastData) + "\nSummarize expected
    trends.";
-
-   const realtimeSummary = getGeminiSummary(realtimePrompt);
-   const forecastSummary = getGeminiSummary(forecastPrompt);
-
-   // Output to Summary sheet
-   summarySheet.clear();
-   summarySheet.getRange("A1").setValue("Real-Time Data
    Summary:");
-   summarySheet.getRange("A2").setValue(realtimeSummary);
-   summarySheet.getRange("A4").setValue("Forecast Data
    Summary:");
-   summarySheet.getRange("A5").setValue(forecastSummary);
-   summarySheet.getRange("A7").setValue("Last Updated: " +
    new Date().toLocaleString());
-
-   Logger.log("Real-Time Summary:\n" + realtimeSummary);
-   Logger.log("Forecast Summary:\n" + forecastSummary);
-   }
-
```