FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING TECHNOLOGY

BVI 3114 TECHNOLOGY SYSTEM OPTIMIZATION II

SEMESTER 5 2025/2026

Lecturer:
Dr. Mohd Zamri Bin Ibrahim

Report:
IoT Real-time Sensor Data Collection with ESP32 and Google Sheets Integration Forecasting System

Prepared By	MUHAMMAD ARIF BIN AMRAN (VC23004)
	MAS IZZALIANA BINTI MOHAMAD MOKHTAR (VC23035)

Table of Content

Executive Summary

This project focuses on the development of an IoT based real time sensor data collection and forecasting system using an ESP32 microcontroller integrated with Google Sheets and Looker Studio. The system was designed to collect temperature, humidity, pressure and distance data from a BME280 temperature, humidity, and pressure sensor and an HC-SR04 ultrasonic distance sensor. The collected data is sent to Google Sheets to be stored, processed and prepared for forecasting and visualization.

Forecasting techniques were implemented using Google Apps Script. Holt-Winters forecasting method was used for temperature, humidity, pressure and distance data to capture trends and seasonal patterns. The system generates a 24 hours forecast along with upper and lower confidence bounds to anticipate future sensor behavior rather than relying solely on historical readings.

It also includes the creation of an interactive data visualization dashboard using Looker Studio. The dashboard presents real time sensor data and forecast results across multiple pages for clear monitoring of each sensor parameter. The dashboard was embedded into a mobile application developed using Android Studio to view sensor data and forecasts conveniently on a mobile device.

Throughout the development process, several technical challenges were encountered including dashboard rendering issues and data access errors. These challenges were successfully resolved through dashboard reconstruction and access permission reconfiguration. The system demonstrates reliable data collection, effective forecasting, and user friendly visualization to meet the project objectives.

Overall, this project showcases a complete IoT data flow from sensor data acquisition to Google Sheet to forecasting, visualization and mobile deployment. The system will also add future enhancements which are AI powered data summaries and improved forecasting techniques for scalable and intelligent solution to real time monitoring applications.

Timeline and Milestones

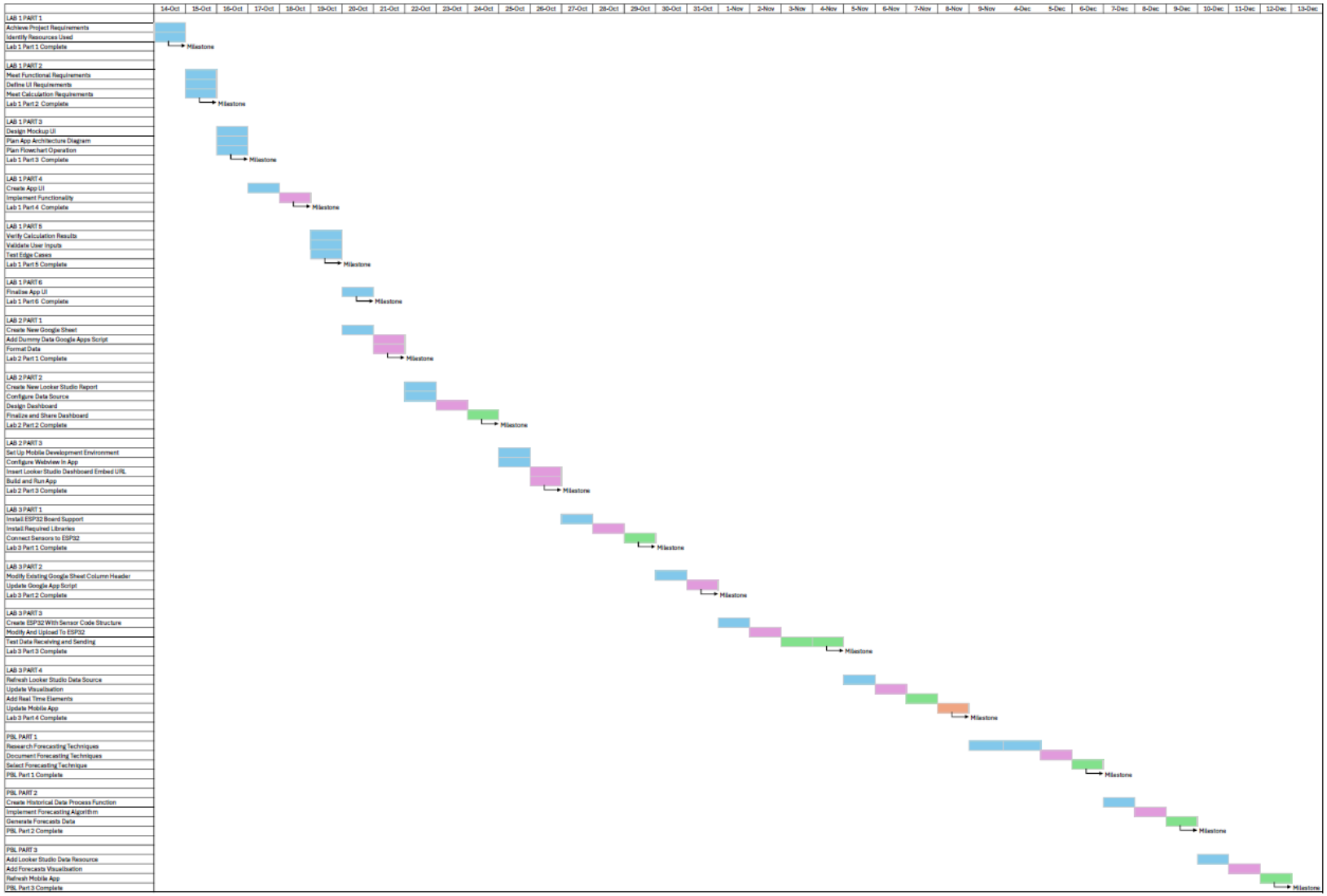


Figure 1: Timeline and Milestones.

System Architecture Diagram

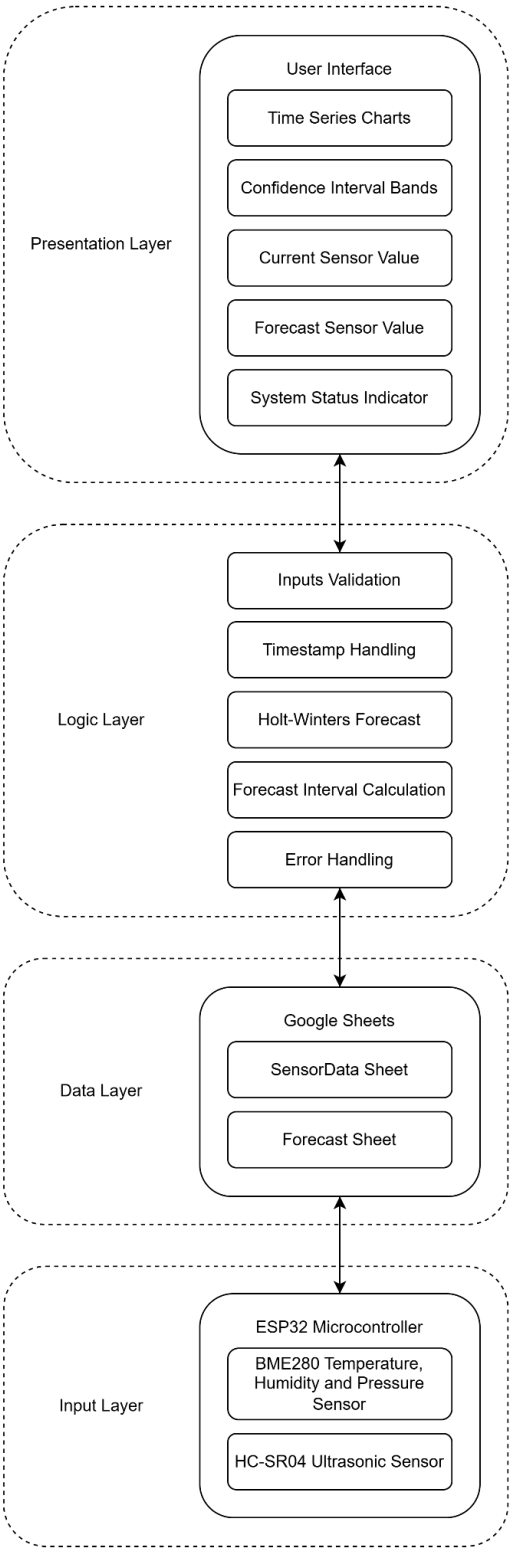


Figure 2: System Architecture Diagram.

Description of Hardware Setup

For this next lab exercise, the previous lab will be extended to replacing the dummy data with real sensor readings from the assigned ESP32 microcontroller, HC-SR04 Ultrasonic sensor and BME280 Temperature and Humidity Sensor.

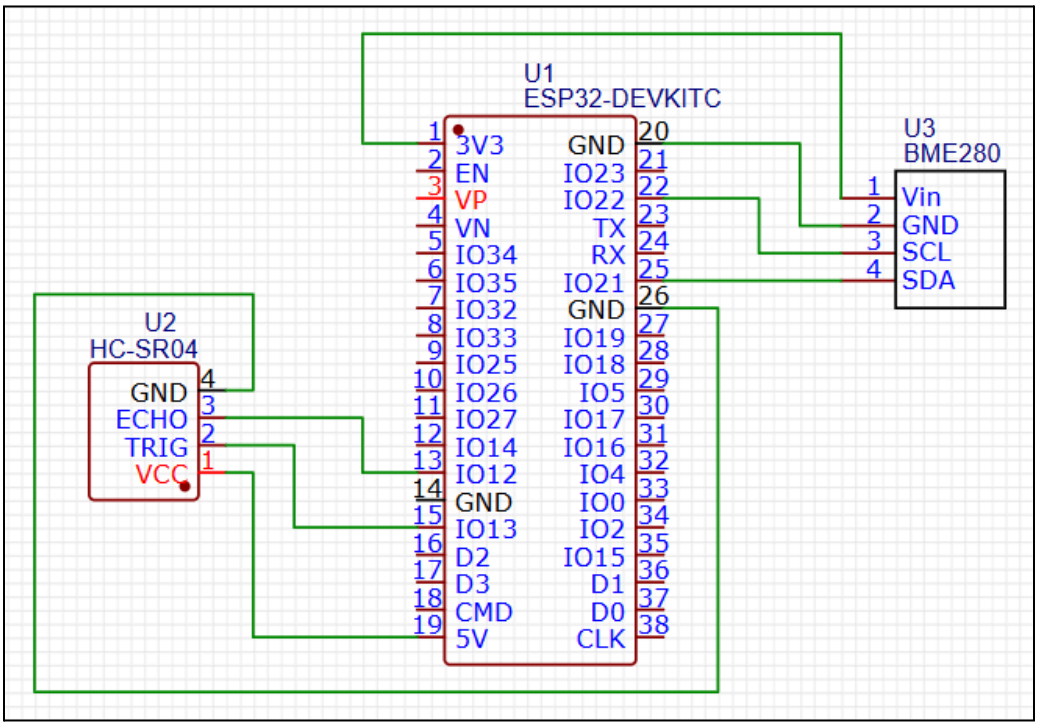


Figure 3: ESP32 Connection With HC-SR04 Ultrasonic sensor and BME280 Temperature and Humidity Sensor Schematic Diagram.

As shown in Figure 3, the wiring for ESP32 with BME280 Temperature and Humidity sensor and HC-SR04 Ultrasonic sensor are as follow;

a. BME280 Temperature and Humidity Sensor Wiring:
   - Vin Pin will connect to 3.3V ESP32.
   - GND Pin will connect to GND ESP32.
   - SCL Pin will connect to I2C Pin GPIO22 ESP32.
   - SDA Pin will connect to I2C Pin GPIO 21 ESP32.

b. HC-SR04 Ultrasonic Sensor Wiring:
   - VCC Pin will connect to 5V ESP32.
   - GND Pin will connect to GND ESP32.
   - ECHO Pin will connect to GPIO12 ESP32.
   - TRIG Pin will connect to GPIO13 ESP32.

Forecasting Algorithm Explanation

1. Exponential Moving Average (EMA)
   a. Basic Principles

      The Exponential Moving Average (EMA) is a time series smoothing technique that gives more weight to recent data while still considering older observations. The EMA reacts faster to sudden changes because the weighting decreases exponentially over time unlike the Simple Moving Average (SMA) which treats all data points equally.

      The EMA is calculated using the formula:

$$EMAt = \alpha \cdot xt + (1 - \alpha) \cdot EMAt - 1$$

      - xt is current measurement.
      - EMAt is new smoothed value
      - A is smoothing factor (0-1)

   b. Strengths
      - Fast reaction to new change because recent data is weighted more heavily.
      - Low computational cost and ideal for microcontrollers.
      - Good noise filtering, reducing random spikes in sensor readings.
      - Works well with streaming data.

   c. Limitations
      - Cannot detect trends or seasonality.
      - Heavily dependent on the alpha value.
      - Not suitable for long term forecasting.

   d. Specific Sensors Suitability

      Highly suitable for HC-SR04 Ultrasonic sensor because the distance readings often contain noise and sudden spikes so EMA will smooths distance measurements while react quick to small fluctuations.

2. Holt-Winters Method

a. Basic Principles

The Holt-Winters method or also known as Triple Exponential Smoothing is forecasting technique for seasonal time series. The extend of this method is by adding seasonal component which use 3 smoothing equations for Level (L), Trend (T) and Seasonality (S) that controlled by alpha, beta and gamma parameters to weight recent and past data with 2 variations to capture constant or proportional seasonal patterns.

Level (L) Smoothing Equation:

$$Lt=\alpha(xt-St-m)+(1-\alpha)(Lt-1+Tt-1)$$

- Take the current observation xt.
- Remove its seasonal effect St−m.
- Combine with previous level and trend.
- Balance both using smoothing factor α (0−1).

Trend (T) Smoothing Equation:

$$Tt=\beta(Lt-Lt-1)+(1-\beta)Tt-1$$

- Compute how the level changed from last step: Lt−Lt−1
- Smooth with previous trend using β(0−1).

Seasonality (S) Smoothing Equation:

$$St=\gamma(xt-Lt)+(1-\gamma)St-m$$

- Compare current value with new xt xt−Lt.
- Update the seasonal pattern using smoothing factor γ.

Holt-Winters Forecast Formula:

$$Ft+h=Lt+hTt+St-m+h$$

To forecast h steps ahead:

- Start with the current level.
- Add trend multiplied by forecast distance h.
- Add the seasonal value corresponding to the future period.

b. Strengths
    - Captures data trend and seasonality.
    - More accurate for a long term predictions.
    - Smooths the data while also forecasting future values.
    - Effective for environmental sensor data with daily cycles.

c. Limitations
    - More computationally heavy.
    - Not always suitable for low end microcontrollers.
    - Requires storing multiple past values.

d. Specific Sensors Suitability
    Highly suitable for BME280 Temperature, Humidity and Pressure sensor because environmental data often shows daily temperature cycles, pressure trends and humidity fluctuations with patterns. This method can model these patterns and forecast environment conditions.

Implementation

Part 1: Creating a Data Source with Google Sheets

Step 1.1: Create a New Google Sheet

Create a new spreadsheet in Google Sheets and rename it to “ESP32 Sensor Data Logger”. Then create 5 column headers in row 1 which is:

- A1: Timestamp.
- B1: Temperature (°C).
- C1: Humidity (%).
- D1: Pressure (atm).
- E1: Distance (cm).

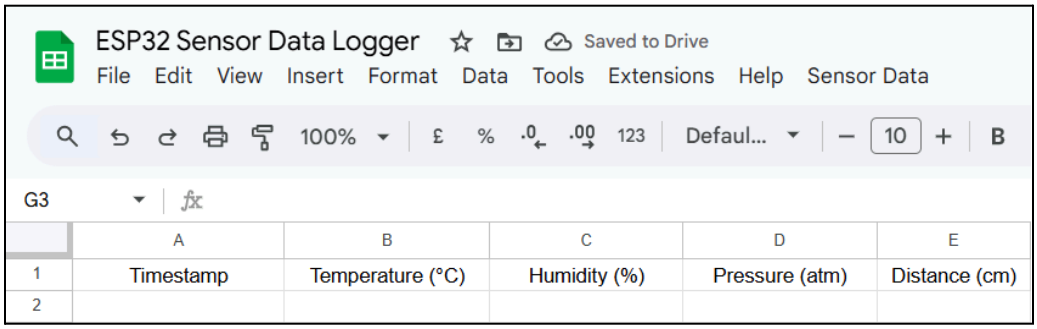


Figure 4: Created a Sheet with Timestamp, Temperature (°C), Humidity (%), Pressure (atm) and Distance (cm) columns.

Step 1.2: Format Data

Select all 5 columns filled with data and click Number on the Format menu. Then choose the appropriate formats for each columns:

- Timestamp column: Date time format.
- Temperature column: Number with 2 decimal places.
- Humidity column: Number with 2 decimal places.
- Pressure column: Number with 2 decimal places.
- Distance column: Number with no decimal places.

	A	B	C	D	E
1	Timestamp	Temperature (°C)	Humidity (%)	Pressure (atm)	Distance (cm)
2	03/12/2025 00:00:47	30.31	70.60	1.00	14
3	03/12/2025 01:00:47	30.08	71.55	1.00	57
4	03/12/2025 02:00:47	30.1	71.45	1.00	11
5	03/12/2025 03:00:47	30.84	68.86	1.00	13
6	03/12/2025 04:00:47	30.98	68.35	1.00	15
7	03/12/2025 05:00:47	31.07	67.93	1.00	17
8	03/12/2025 06:00:47	31.09	67.86	1.00	12
9	03/12/2025 07:00:47	31.06	67.93	1.00	18
10	03/12/2025 08:00:47	31.06	67.79	1.00	17
11	03/12/2025 09:00:47	31	67.85	1.00	9
12	03/12/2025 10:00:47	31.09	67.62	1.00	23
13	03/12/2025 11:00:47	30.66	68.60	1.00	21
14	03/12/2025 12:00:47	30.16	70.19	1.00	24
15	03/12/2025 13:00:47	30.19	70.38	1.00	23
16	03/12/2025 14:00:47	30.46	69.52	1.00	21
17	03/12/2025 15:00:47	30.22	70.35	1.00	17
18	03/12/2025 16:00:47	30.18	70.55	1.00	20
19	03/12/2025 17:00:47	30.12	70.88	1.00	8
20	03/12/2025 18:00:47	30.15	70.74	1.00	9
21	03/12/2025 19:00:47	30.23	70.60	1.00	10
22	03/12/2025 20:00:47	30.04	71.20	1.00	18
23	03/12/2025 21:00:47	30.07	71.22	1.00	78

Figure 5: Timestamp Column Data Formatted to Date Time, Temperature, Humidity and Pressure Column Data Formatted to 2 Decimal Place Number and Distance Column Data Formatted to No Decimal Place Number.

Part 2: Implement Forecasting in Google App Script

Step 2.1: Extend Existing Google Apps Script with Forecasting Functionality

Create new function to process historical data with the Holt-Winters Forecasting algorithm implemented to generate forecast for next 24 hours.

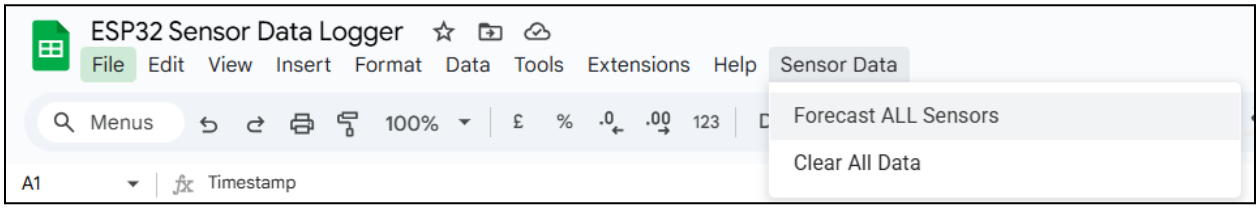
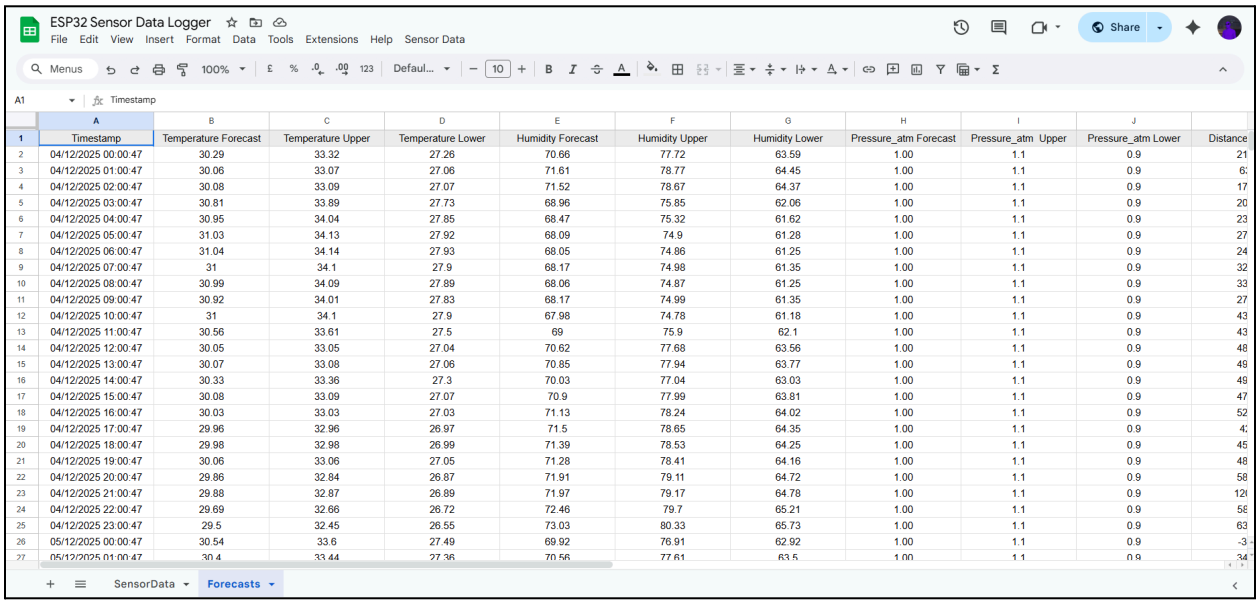


Figure 6: Forecast ALL Sensors Function Button In Google Sheet Menu to Generate 24 Hours Forecast Data.

The generated 24 hours forecast will be stored in a separate sheet named Forecasts within the same Google Sheet.



	A	B	C	D	E	F	G	H	I	J	
1	Timestamp	Temperature Forecast	Temperature Upper	Temperature Lower	Humidity Forecast	Humidity Upper	Humidity Lower	Pressure_atm Forecast	Pressure_atm Upper	Pressure_atm Lower	Distance
2	04/12/2025 00:00:47	30.29	33.32	27.26	70.66	77.72	63.59	1.00	1.1	0.9	21
3	04/12/2025 01:00:47	30.06	33.07	27.06	71.61	78.77	64.45	1.00	1.1	0.9	6
4	04/12/2025 02:00:47	30.08	33.09	27.07	71.52	78.67	64.37	1.00	1.1	0.9	17
5	04/12/2025 03:00:47	30.81	33.89	27.73	68.96	75.85	62.06	1.00	1.1	0.9	20
6	04/12/2025 04:00:47	30.95	34.04	27.85	68.47	75.32	61.62	1.00	1.1	0.9	23
7	04/12/2025 05:00:47	31.03	34.13	27.92	68.09	74.9	61.28	1.00	1.1	0.9	27
8	04/12/2025 06:00:47	31.04	34.14	27.93	68.05	74.86	61.25	1.00	1.1	0.9	24
9	04/12/2025 07:00:47	31	34.1	27.9	68.17	74.98	61.35	1.00	1.1	0.9	32
10	04/12/2025 08:00:47	30.99	34.09	27.89	68.06	74.87	61.25	1.00	1.1	0.9	33
11	04/12/2025 09:00:47	30.92	34.01	27.83	68.17	74.99	61.35	1.00	1.1	0.9	27
12	04/12/2025 10:00:47	31	34.1	27.9	67.98	74.78	61.18	1.00	1.1	0.9	43
13	04/12/2025 11:00:47	30.56	33.61	27.5	69	75.9	62.1	1.00	1.1	0.9	43
14	04/12/2025 12:00:47	30.05	33.05	27.04	70.62	77.68	63.56	1.00	1.1	0.9	48
15	04/12/2025 13:00:47	30.07	33.08	27.06	70.85	77.94	63.77	1.00	1.1	0.9	49
16	04/12/2025 14:00:47	30.33	33.36	27.3	70.03	77.04	63.03	1.00	1.1	0.9	49
17	04/12/2025 15:00:47	30.08	33.09	27.07	70.9	77.99	63.81	1.00	1.1	0.9	47
18	04/12/2025 16:00:47	30.03	33.03	27.03	71.13	78.24	64.02	1.00	1.1	0.9	52
19	04/12/2025 17:00:47	29.96	32.96	26.97	71.5	78.65	64.35	1.00	1.1	0.9	4
20	04/12/2025 18:00:47	29.98	32.98	26.99	71.39	78.53	64.25	1.00	1.1	0.9	45
21	04/12/2025 19:00:47	30.06	33.06	27.05	71.28	78.41	64.16	1.00	1.1	0.9	48
22	04/12/2025 20:00:47	29.86	32.84	26.87	71.91	79.11	64.72	1.00	1.1	0.9	58
23	04/12/2025 21:00:47	29.88	32.87	26.89	71.97	79.17	64.78	1.00	1.1	0.9	12
24	04/12/2025 22:00:47	29.69	32.66	26.72	72.46	79.7	65.21	1.00	1.1	0.9	58
25	04/12/2025 23:00:47	29.5	32.45	26.55	73.03	80.33	65.73	1.00	1.1	0.9	63
26	05/12/2025 00:00:47	30.54	33.6	27.49	69.92	76.91	62.92	1.00	1.1	0.9	-3
27	05/12/2025 01:00:47	30.4	33.44	27.36	70.56	77.61	63.5	1.00	1.1	0.9	34

Figure 7: Column of Data After Generate Forecast All Sensors Consisting Timestamp, Temperature Forecast, Temperature Upper, Temperature Lower, Humidity Forecast, Humidity Upper, Humidity Lower, Pressure_atm Forecast, Pressure_atm Upper, Pressure_atm Lower, Distance Forecast, Distance Upper, Distance Lower.

Step 2.2: Google Apps Script Structure With Holt-Winters Forecast Algorithm

The forecast generate function are added to existing Google Apps Script by manual using button added to the Google Sheet menu named Sensor Data. A forecast sheet will be created if not existed yet. The function will get all the historical sensor data available in SensorData sheet, calculate the forecast with Holt-Winters method algorithm and add the timestamp, each sensor 24 hours data forecast, each sensor upper bound and lower bound to every designated column made by the function.

```
function doGet(e) { return handleResponse(e); }
function doPost(e) { return handleResponse(e); }

function handleResponse(e) {
  var lock = LockService.getScriptLock();
  lock.tryLock(10000);

  try {
    var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("SensorData");

    var payload;
    if (e.postData && e.postData.contents) {
      payload = JSON.parse(e.postData.contents);
    } else if (e.parameter) {
      payload = e.parameter;
    } else {
      return ContentService
        .createTextOutput(JSON.stringify({ status: 'error',
message: 'No data received' }))
        .setMimeType(ContentService.MimeType.JSON);
    }

    var timestamp = new Date();
    var data = [
      timestamp,
      parseFloat(payload.temperature) || "",
      parseFloat(payload.humidity) || "",
      parseFloat(payload.pressure_atm) || "",
      parseFloat(payload.distance) || ""
    ];

    sheet.appendRow(data);

    return ContentService
      .createTextOutput(JSON.stringify({ status: "success",
```

```
timestamp: timestamp.toString() }))
      .setMimeType(ContentService.MimeType.JSON);

  } catch (error) {
    return ContentService
      .createTextOutput(JSON.stringify({ status: "error", message:
error.toString() }))
      .setMimeType(ContentService.MimeType.JSON);
  } finally {
    lock.releaseLock();
  }
}

function generateAllForecasts() {
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getSheetByName("SensorData");
  var lastRow = sheet.getLastRow();
  if (lastRow < 2) return;

  var sensors = {
    "temperature": 2,
    "humidity": 3,
    "pressure_atm": 4,
    "distance": 5
  };

  var sensorValues = {};
  Object.keys(sensors).forEach(sensor => {
    sensorValues[sensor] = sheet.getRange(2, sensors[sensor],
lastRow - 1, 1)
                                .getValues()
                                .flat()
                                .filter(x => x !== "" && !isNaN(x));
  });

  var forecastSheet = ss.getSheetByName("Forecasts");
  if (!forecastSheet) {
    forecastSheet = ss.insertSheet("Forecasts");

    var header = ["Timestamp"];
    Object.keys(sensors).forEach(sensor => {
      header.push(sensor + " Forecast", sensor + " Upper", sensor +
" Lower");
    });
```

```
    forecastSheet.appendRow(header);
  }
  var forecastLastRow = forecastSheet.getLastRow();
  var startTimestamp;
  if (forecastLastRow > 1) {

    startTimestamp = new
Date(forecastSheet.getRange(forecastLastRow, 1).getValue());
  } else {

    var timestamps = sheet.getRange(2, 1, lastRow - 1,
1).getValues();
    startTimestamp = new Date(timestamps[timestamps.length - 1]);
  }

  var alpha = 0.3, beta = 0.1, gamma = 0.1, seasonLength = 24;
  var forecastLength = 24;

  var forecasts = {};
  Object.keys(sensorValues).forEach(sensor => {
    forecasts[sensor] =
calculateHoltWintersForecast(sensorValues[sensor], startTimestamp,
alpha, beta, gamma, seasonLength, forecastLength);
  });

  for (var h = 0; h < forecastLength; h++) {
    var row = [];
    row.push(forecasts["temperature"][h].timestamp);
    Object.keys(sensors).forEach(sensor => {
      row.push(
        forecasts[sensor][h].forecastValue.toFixed(2),
        forecasts[sensor][h].upperBound.toFixed(2),
        forecasts[sensor][h].lowerBound.toFixed(2)
      );
    });
    forecastSheet.appendRow(row);
  }

  SpreadsheetApp.getUi().alert("All forecasts generated in single
sheet!");
}

function calculateHoltWintersForecast(values, lastTimestamp, alpha,
beta, gamma, m, forecastLength) {
```

```
  var seasonals = [];
  var initialSeasonAverage = average(values.slice(0, m));

  for (var i = 0; i < m; i++) {
    seasonals.push(values[i] - initialSeasonAverage);
  }

  var level = values[0];
  var trend = values[1] - values[0];

  for (var t = 0; t < values.length; t++) {
    var val = values[t];
    var seasonalIndex = t % m;
    var lastLevel = level;

    level = alpha * (val - seasonals[seasonalIndex]) + (1 - alpha)
* (level + trend);
    trend = beta * (level - lastLevel) + (1 - beta) * trend;
    seasonals[seasonalIndex] = gamma * (val - level) + (1 - gamma)
* seasonals[seasonalIndex];
  }

  var output = [];
  for (var h = 1; h <= forecastLength; h++) {
    var nextTime = new Date(lastTimestamp.getTime() + h * 60 * 60 *
1000);
    var forecastValue = level + h * trend +
seasonals[(values.length + h - 1) % m];

    output.push({
      timestamp: nextTime,
      forecastValue: forecastValue,
      upperBound: forecastValue * 1.10,
      lowerBound: forecastValue * 0.90
    });
  }

  return output;
}

function average(arr) {
  var sum = arr.reduce((a,b) => a+b, 0);
  return sum / arr.length;
}
```

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Sensor Data')
    .addItem('Forecast ALL Sensors', 'generateAllForecasts')
    .addItem('Clear All Data', 'clearData')
    .addToUi();
}

  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("SensorData");
  var lastRow = sheet.getLastRow();
  if (lastRow > 1) sheet.deleteRows(2, lastRow - 1);
  SpreadsheetApp.getUi().alert("SensorData cleared successfully!");
}
```

Part 3: Creating a Looker Studio Dashboard

Step 3.1: Create New Looker Studio Report

Go to Looker Studio, click the Create button and click Report. Choose Google Sheets at the data source selection and select where Temperature Humidity Sensor Data sheet are located. Then click ADD TO REPORT.

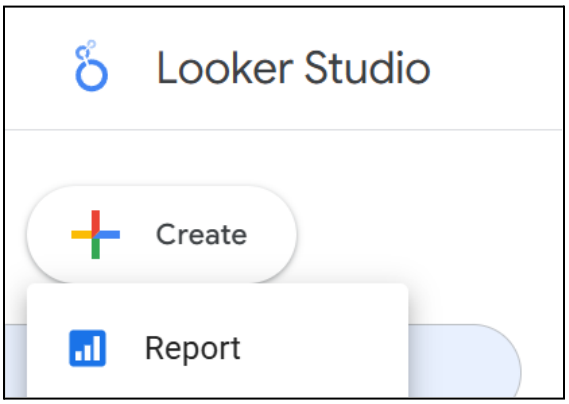


Figure 8: Create a New Report by Clicking the Create Button.

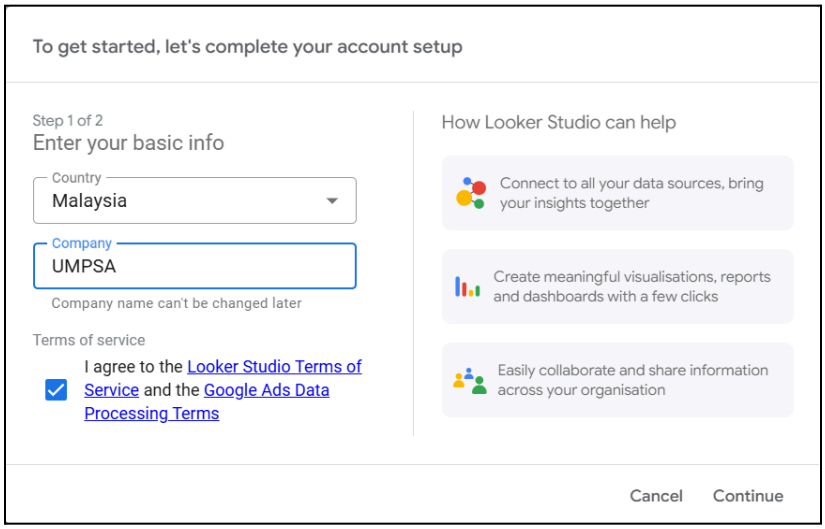


Figure 9: Enter the Basic Info Before Continuing.

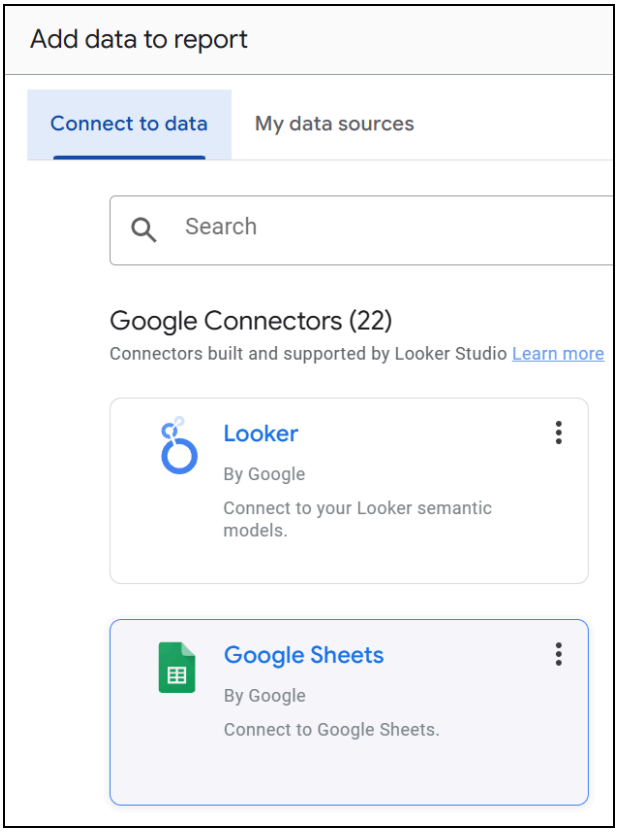


Figure 10: Select Google Sheets to Connect and Add Data to Report.

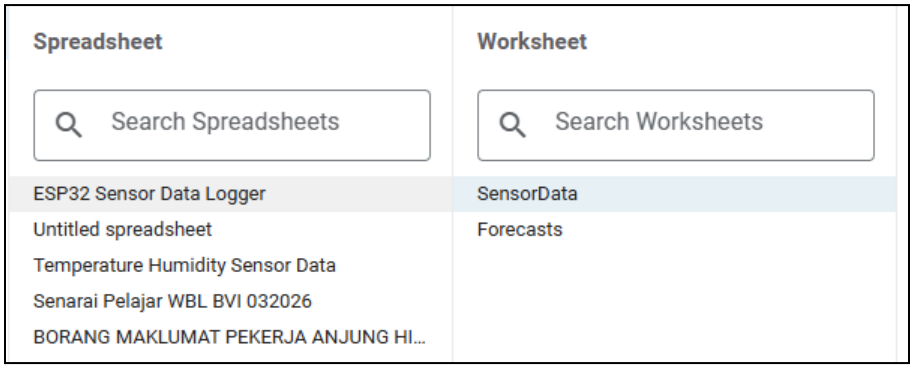


Figure 11: Select the ESP32 Sensor Data Logger Spreadsheet.

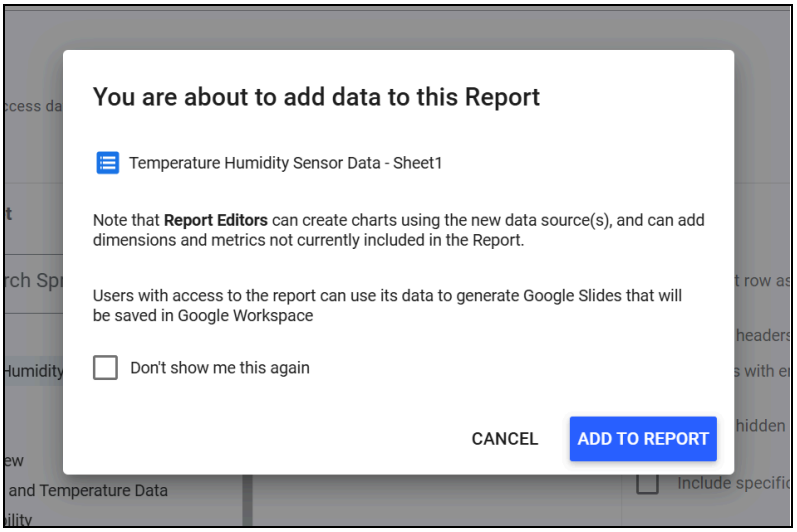


Figure 12: Confirm the Selected Data to Be Added to Report.

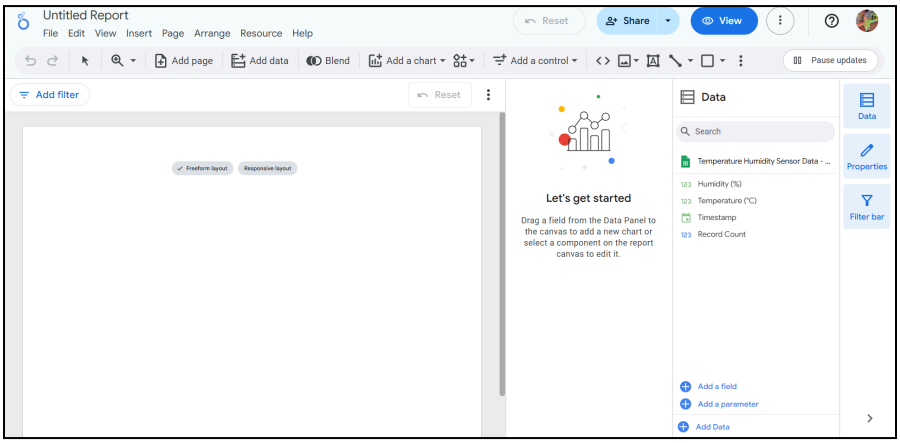


Figure 13: Blank Dashboard Report With ESP32 Sensor Data Logger Spreadsheet Resources.

Step 3.2: Configure Data Source

The data types in Looker Studio are correctly identified where in the SensorData sheet the Timestamp should be a Date & Time, Temperature, Humidity, Pressure and Distance should be a numeric metric.

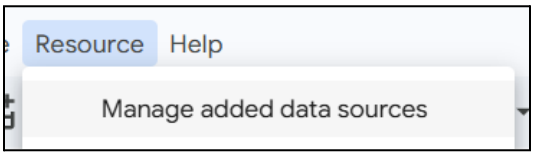


Figure 14: Select Manage added data sources to View Data Types.

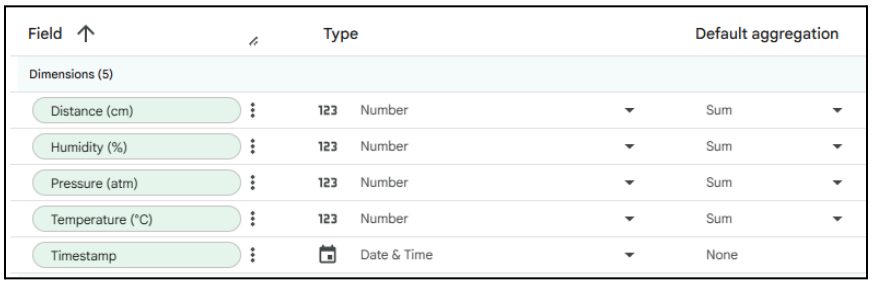


Figure 15: Temperature, Humidity, Pressure and Distance are Number Data Type and Timestamp are Date & Time Data Type.

And for Forecast sheet data types in Looker Studio is correctly identified too where the Timestamp should be a Date & Time too, Temperature Forecast, Temperature Upper, Temperature Lower, Humidity Forecast, Humidity Upper, Humidity Lower, Pressure_atm Forecast, Pressure_atm Upper, Pressure_atm Lower, Distance Forecast, Distance Upper and Distance Lower should be a numeric metric.

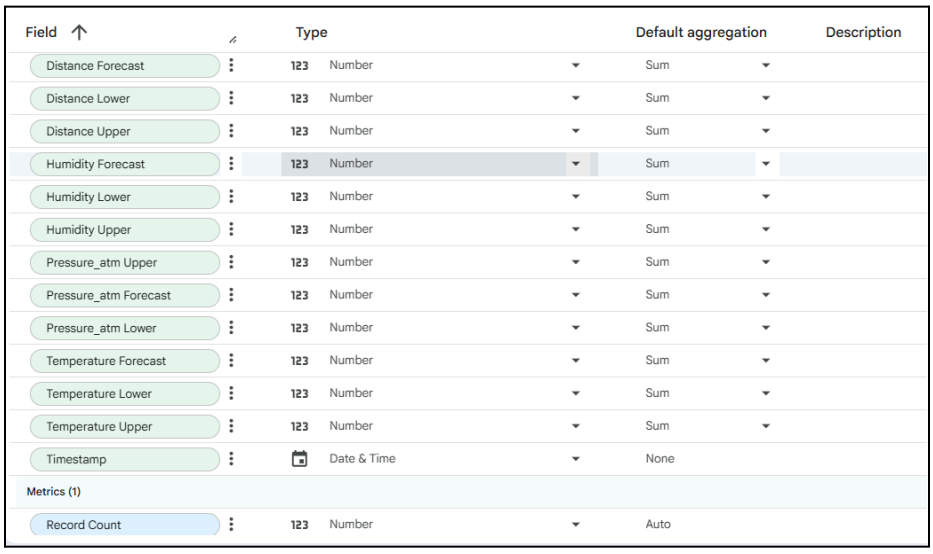


Figure 16: All the Forecast Data are Number Data Type and Timestamp are Date & Time Data Type.

Step 3.3: Design Dashboard

In the new report, the dashboard is in responsive layout and has multiple page. In the first page are added title, data range control, dual axis chart, separate metrics scorecard and 4 button to navigate to different pages in dashboard.

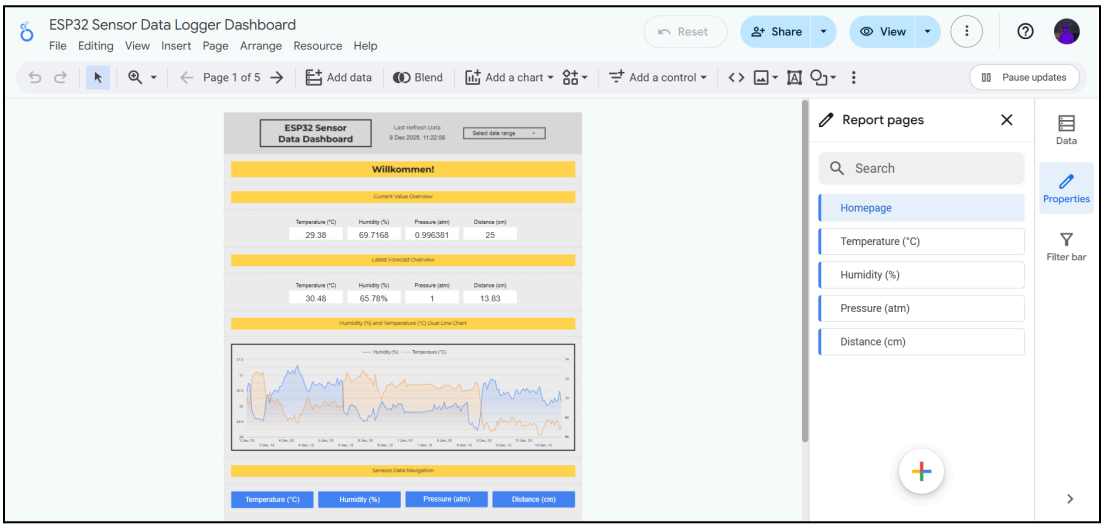


Figure 17: Dashboard First Page Design Included With Title, Date Range Control, Dual Axis Chart, Separate Metrics Scorecard and Buttons.

Every pages display each sensor data and with the forecast data individually but all the pages use same data display types. Every data pages use title, date range control, combo chart, separate metrics scorecard and 4 buttons.

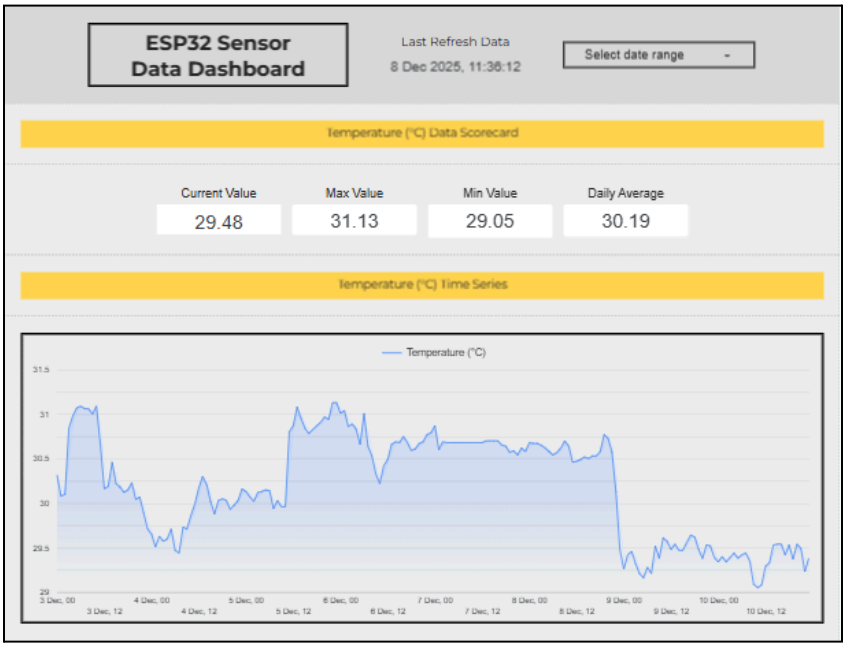


Figure 18: Upper View Temperature Dashboard Page Design Included With Title, Date Range Control, Time Series Chart and 4 Separate Metrics Scorecard.

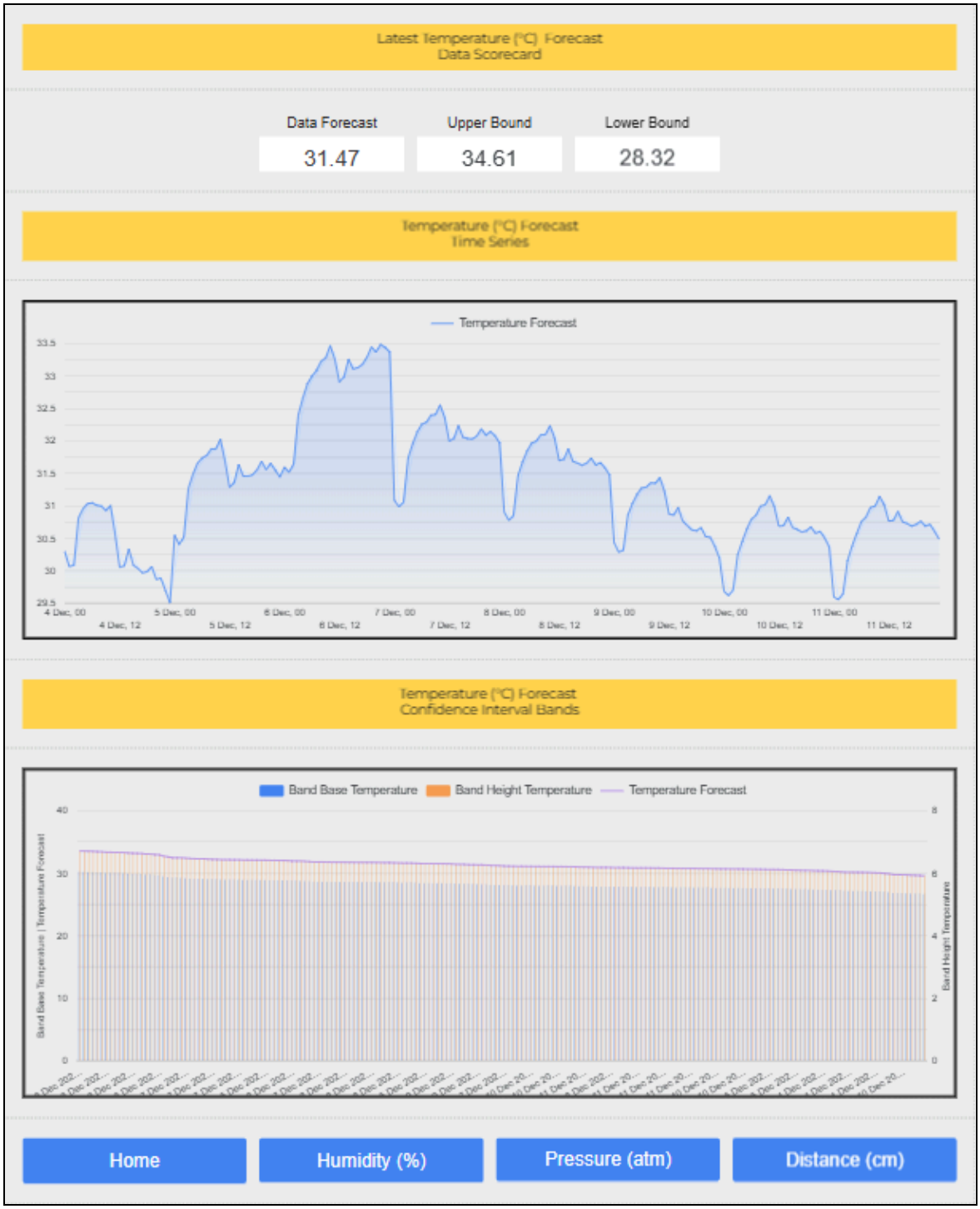


Figure 19: Lower View Temperature Dashboard Page Design Included Time Series Chart, Combo Chart and 3 Separate Metrics Scorecard.

Step 3.4: Finalize and Share Dashboard

Click View mode to test dashboard functionality and Edit mode for any adjustments needed.

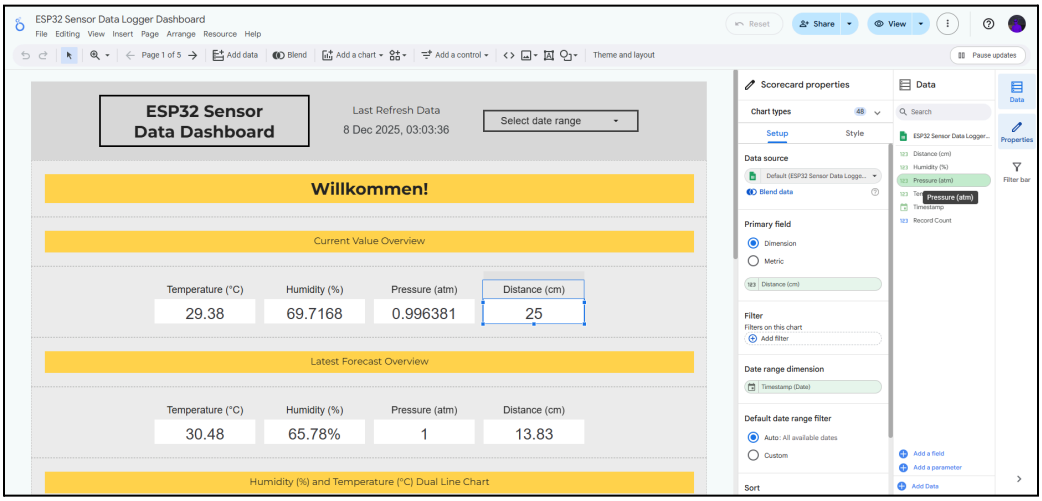


Figure 20: Looker Studio Edit Mode.

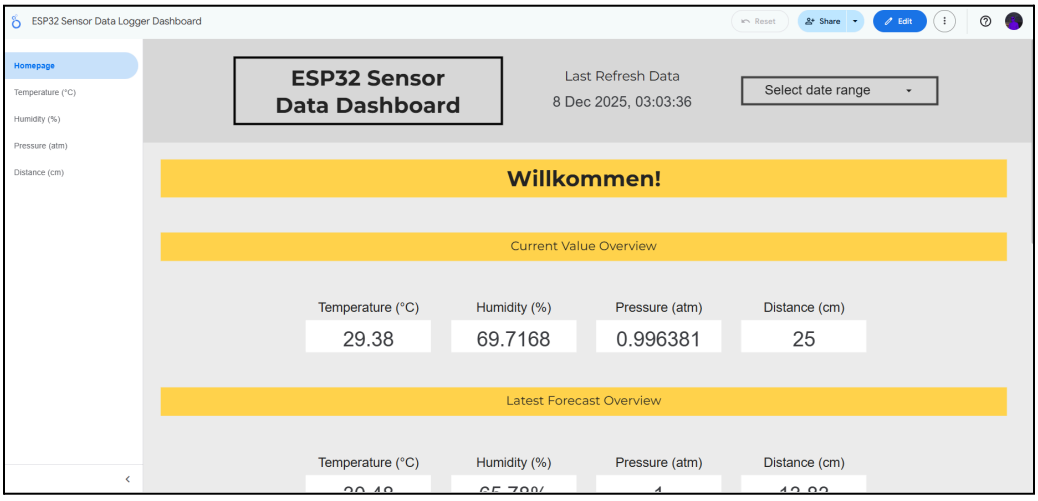


Figure 21: Looker Studio View Mode.

Then click Share in top right corner and set the access of the dashboard from Restricted to Anyone on the Internet with the link can find and view.

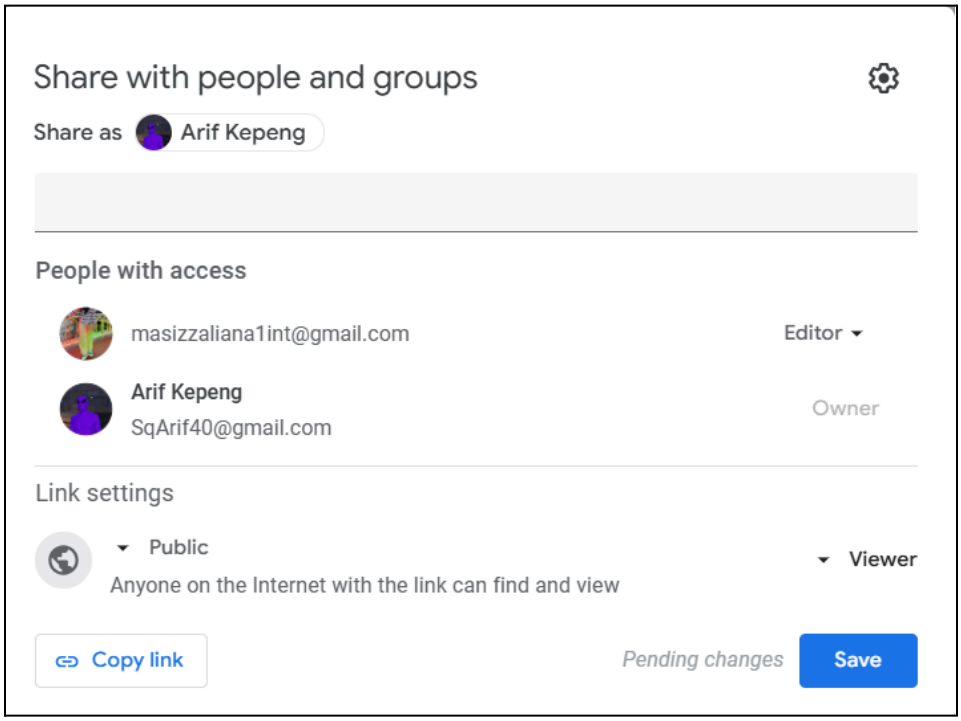


Figure 22: Changed the Access of Dashboard to Public but Viewer Only.

Get the dashboard embed URL by opening Embed Report from the File menu.

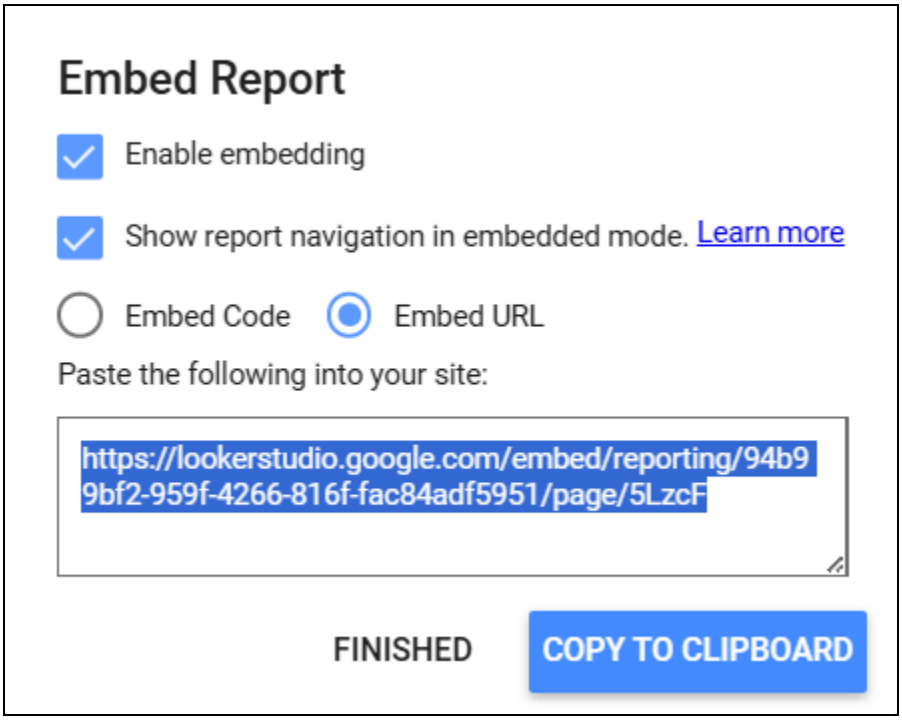


Figure 23: Click Both Check Boxes, Select Embed URL and COPY TO CLIPBOARD.

Part 4: Creating Mobile App With Embedded Dashboard

Step 4.1: Set Up Mobile Development Environment

Open Android Studio application and create a new project using Empty Views Activity template.

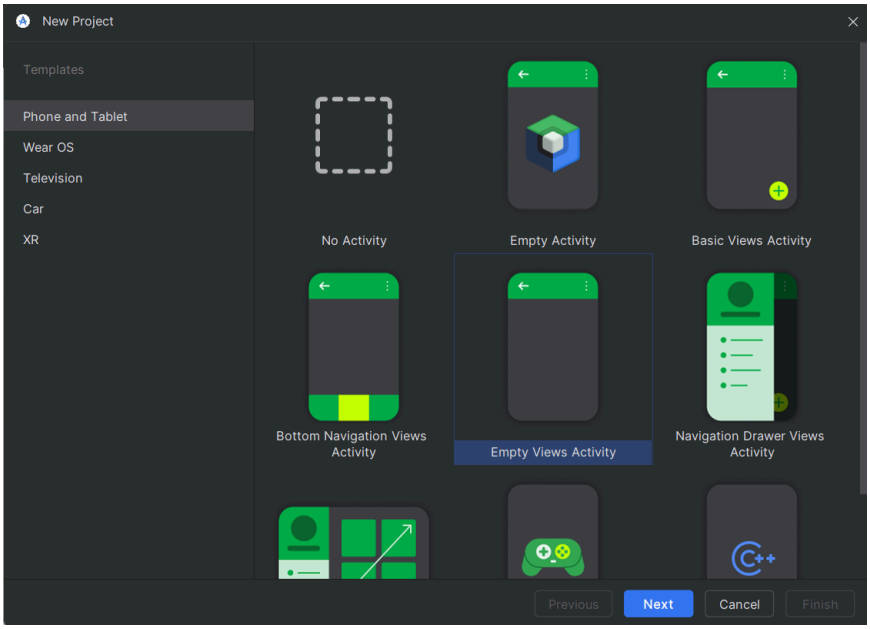


Figure 24: Select Empty Views Activity and Click Next.

Change the project name to SensorDashboard and use Android 5.0 Lollipop.

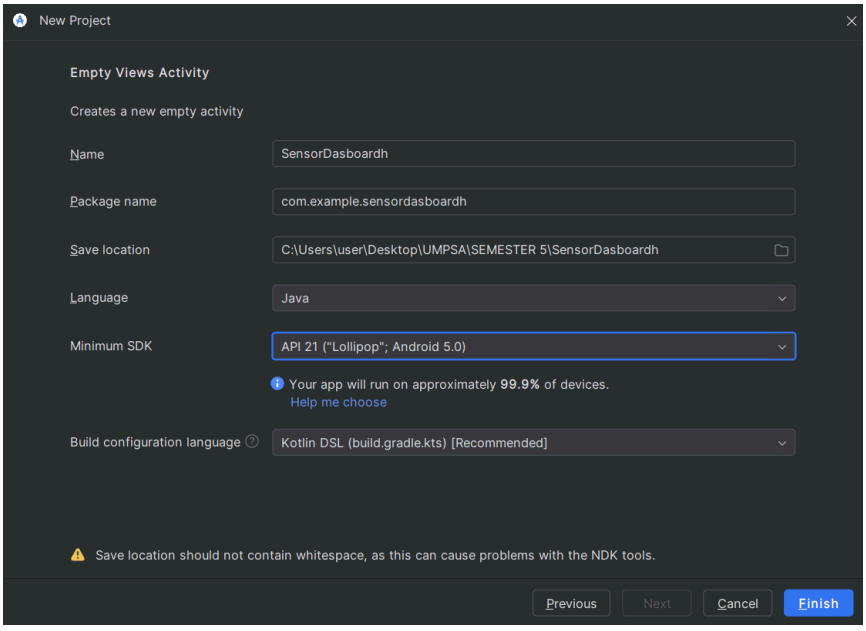


Figure 25: New Empty Views Activity Project Named SensorDashboard Using Java Language With Android 5.0 at Minimum SDK.

Step 4.2: Configure the Webview in App

Replace the existing activity_main.xml file code with this code:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <WebView
        android:id="@+id/dashboardWebView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Update the MainActivity.java file code with this code:

```
package com.example.sensordashboard;

import androidx.activity.OnBackPressedCallback;
import androidx.appcompat.app.AppCompatActivity;

import android.annotation.SuppressLint;
import android.os.Bundle;
import android.view.View;
import android.webkit.WebSettings;
```

```
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.ProgressBar;

public class MainActivity extends AppCompatActivity {
    private WebView dashboardWebView;
    private ProgressBar progressBar;

    @SuppressLint("SetJavaScriptEnabled")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        dashboardWebView = findViewById(R.id.dashboardWebView);
        progressBar = findViewById(R.id.progressBar);

        WebSettings webSettings = dashboardWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        webSettings.setDomStorageEnabled(true);
        webSettings.setLoadWithOverviewMode(true);
        webSettings.setUseWideViewPort(true);
        webSettings.setSupportZoom(true);
        webSettings.setBuiltInZoomControls(true);
        webSettings.setDisplayZoomControls(false);

        dashboardWebView.setWebViewClient(new WebViewClient() {
            @Override
            public void onPageFinished(WebView view, String url) {
                progressBar.setVisibility(View.GONE);
                super.onPageFinished(view, url);
            }
        });

        String dashboardUrl =

"https://lookerstudio.google.com/embed/reporting/94b99bf2-959f-4266
-816f-fac84adf5951/page/5LzcF";
        progressBar.setVisibility(View.VISIBLE);
        dashboardWebView.loadUrl(dashboardUrl);

        getOnBackPressedDispatcher().addCallback(this, new
OnBackPressedCallback(true) {
            @Override
```

```
            public void handleOnBackPressed() {
                if (dashboardWebView.canGoBack()) {
                    dashboardWebView.goBack();
                } else {
                    finish();
                }
            }
        });
    }
}
```

Update the AndroidManifest.xml file code to add internet permission with this code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sensordashboard">
    <uses-permission
        android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SensorDashboard">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Step 4.3: Insert Looker Studio Dashboard Embed URL

Open the MainActivity.java code and replace the existing URL with the created Looker Studio dashboard at the line containing `dashboardURL`.

```
String dashboardUrl =
        "https://lookerstudio.google.com/embed/reporting/743ce5e7-4615-4ce8-8b1f-c3613cfb5249/page/HdRcF";
```

Figure 26: Changed Existing URL With Created Dashboard Copied Embed URL.

Step 4.4: Build and Run App

Run the built app by clicking the Run button in Android Studio and wait for the app to install and launch.

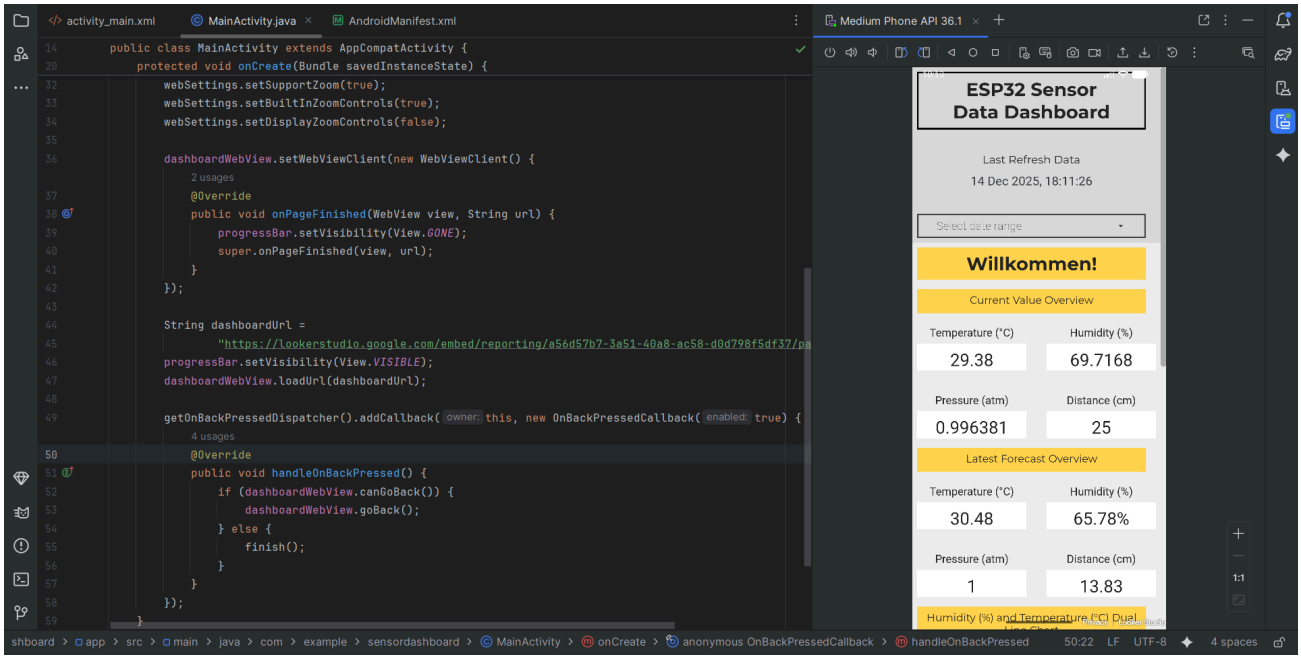


Figure 27: Created Looker Studio Dashboard Successfully Run in App Using Android Studio.

Base Code Modification

1. ESP32 Arduino IDE Code.
The original Ultrasonic code was modified by adding the BME280 sensor, adjusting the interval and updating the data sensor validation.

1. Added BME280 Sensor Support.
   - Libraries for BME280 Sensor.

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
```

   - BM280 Sensor Global Object.

```
Adafruit_BME280 bme;
```

   - BME280 Sensor Initialization Setup.

```
if (!bme.begin(0x76)) {
  Serial.println("Could not find BME280 sensor!");
  while (1);
}
```

   - BME280 Sensor Readings in Loop.

```
float temperature = bme.readTemperature();
float humidity = bme.readHumidity();
```

2. Fixed Ultrasonic Pin Usage.
   - In the original code, the pin for ultrasonic was unclear and the Ultrasonic library required 2 pins which are ECHO and TRIG. Then it was replaced with this code:

```
#define TRIG_PIN 13
#define ECHO_PIN 12
Ultrasonic ultrasonic(TRIG_PIN, ECHO_PIN);
```

3. Added Temperature and Humidity with Existing Distance to the JSON.
   - In the original code, only distance data sent to Google Sheets and with BME280 sensor being added, the temperature and humidity data are added to the existing code:

```
doc["temperature"] = temperature;
doc["humidity"] = humidity;
doc["distance"] = distance;
```

4. Added Sensor Data Validation.
   - Improved the original data validation code to prevent bad or incomplete readings from being sent to Google Sheets:

```
long distance = ultrasonic.read();
if (isnan(temperature) || isnan(humidity) || distance <= 0) {
  Serial.println("Failed to read from sensors!");
  return;
}
```

5. Updated Serial Output for Debugging.
   - Added a print statement to see all 3 sensor readings in the Serial Monitor.

```
Serial.print("Temperature: ");
Serial.print(temperature);
Serial.print("°C, Humidity: ");
Serial.print(humidity);
Serial.print("%, Distance: ");
Serial.print(distance);
Serial.println(" cm");
```

Challenges

1. The screen appeared completely blank white even after refreshing while running the app in Android Studio

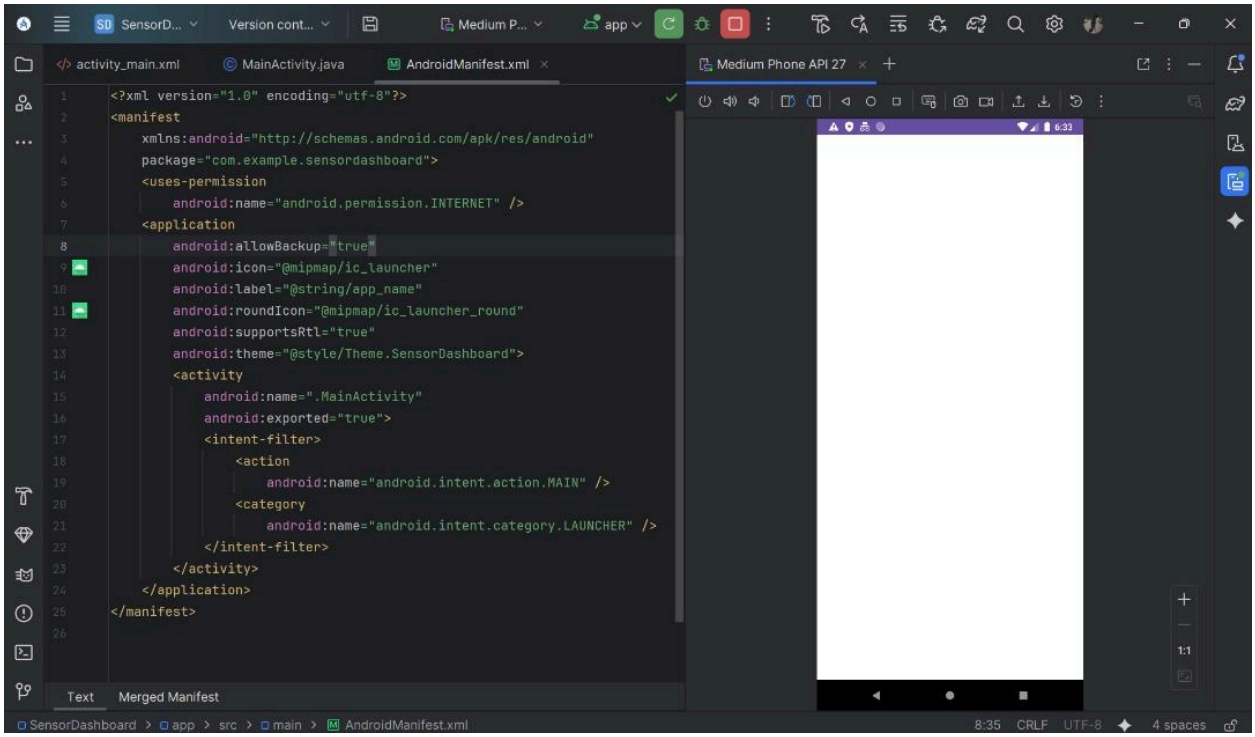


Figure 28: App Appear Blank White.

2. An error message pop-up saying “Couldn’t save the file. Sorry! We can’t save this report right now.” after the app was running.

Figure 29: Error Message Pop-up in App.

Solutions

1. The solution to first challenge was to recreate back the Looker Studio Dashboard. Then reuse the same activity_main.xml, MainActivity.java and AndroidManifest.xml codes in the same project file.

2. The solution to the second challenge was refreshing the Looker Studio page and changing manage access from public to restricted and rechange it back to public access that allow people with link to edit and view.

Results

1. Hardware Setup.

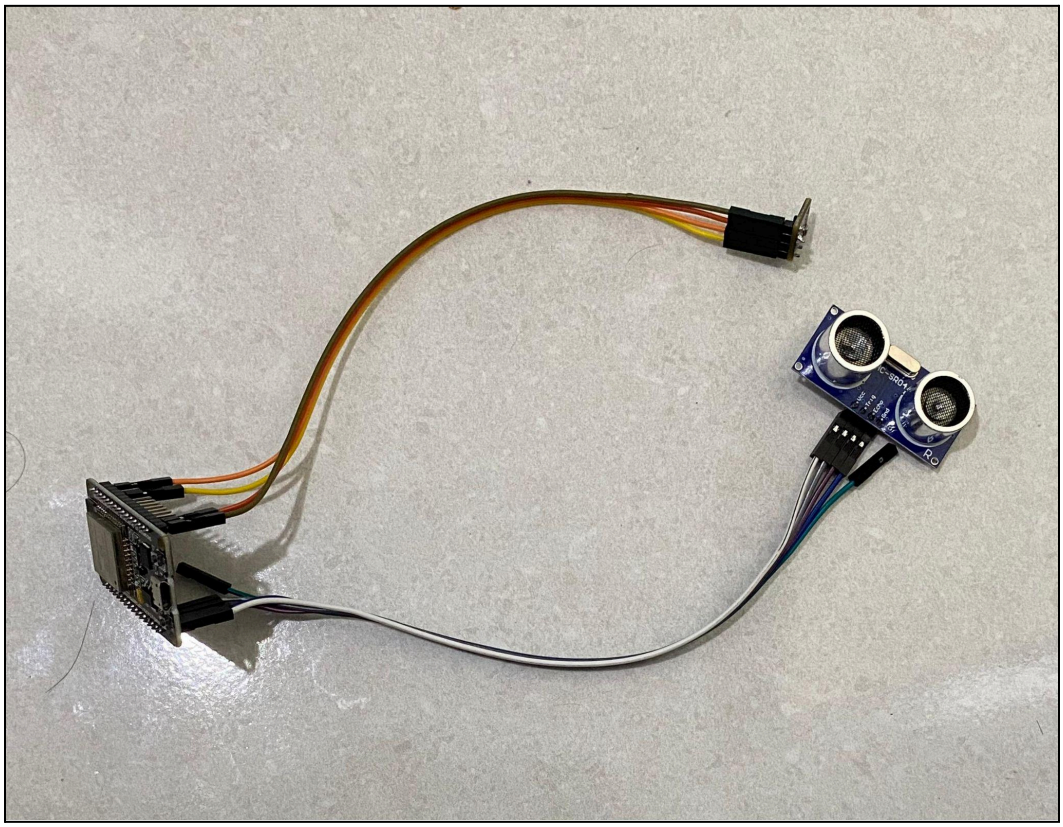


Figure 30: BME280 Temperature and Humidity sensor and HC-SR04 Ultrasonic sensor Hardware Connection to ESP32.

2. Looker Studio Dashboard.
   The dashboard is separated into 5 pages. First page is named Homepage that displays the overview of every sensors data.

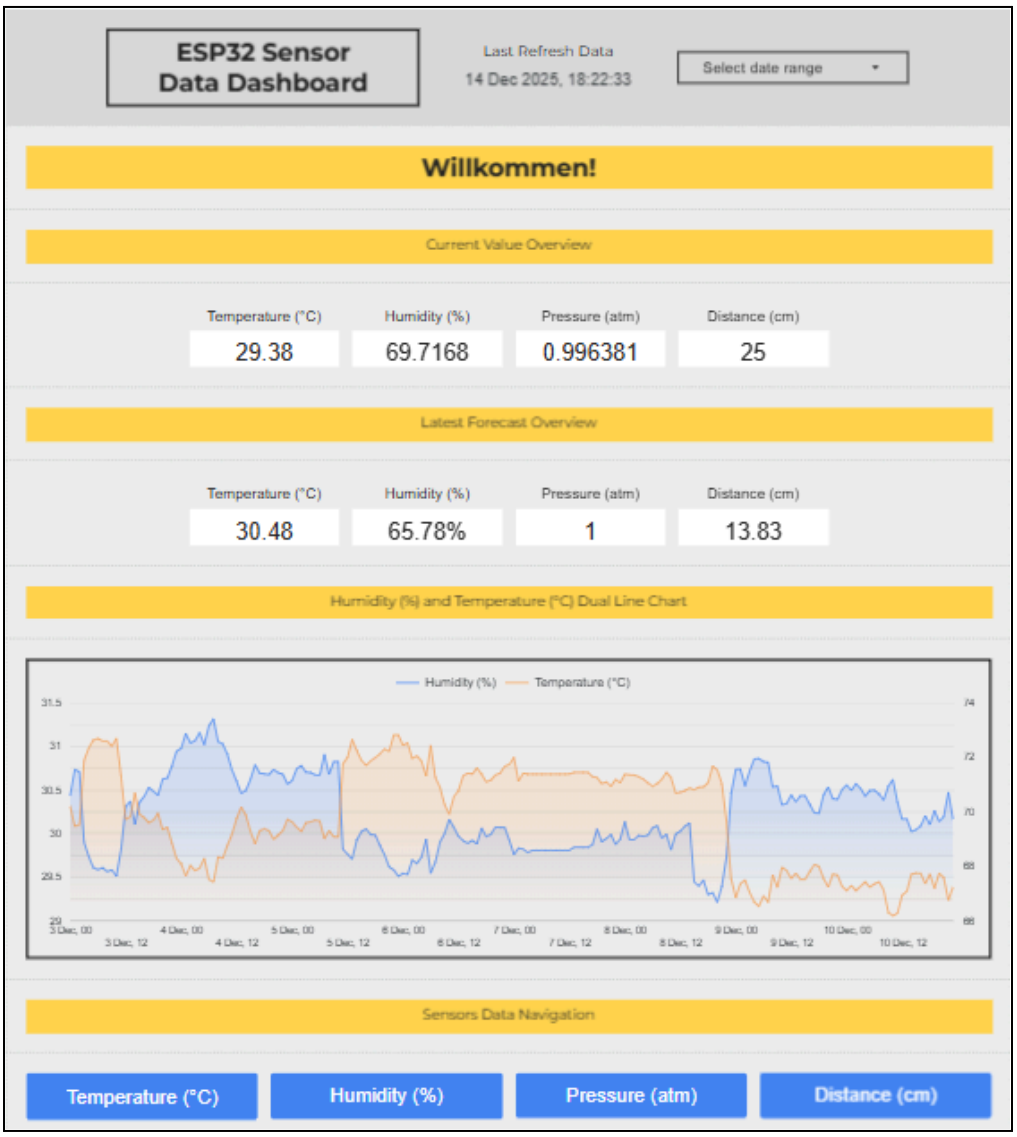


Figure 31: Homepage Dashboard.

In first page only displays title named “ESP32 Sensor Data Dashboard”, a date range control, 8 separate metrics scorecard which displays temperature current value and latest forecast, humidity current value and latest forecast, pressure current value and latest forecast, distance current value and latest forecast, a dual axis chart of temperature and humidity data and 4 buttons which navigates to Temperature (°C), Humidity (%), Pressure (atm) and Distance (cm) pages.

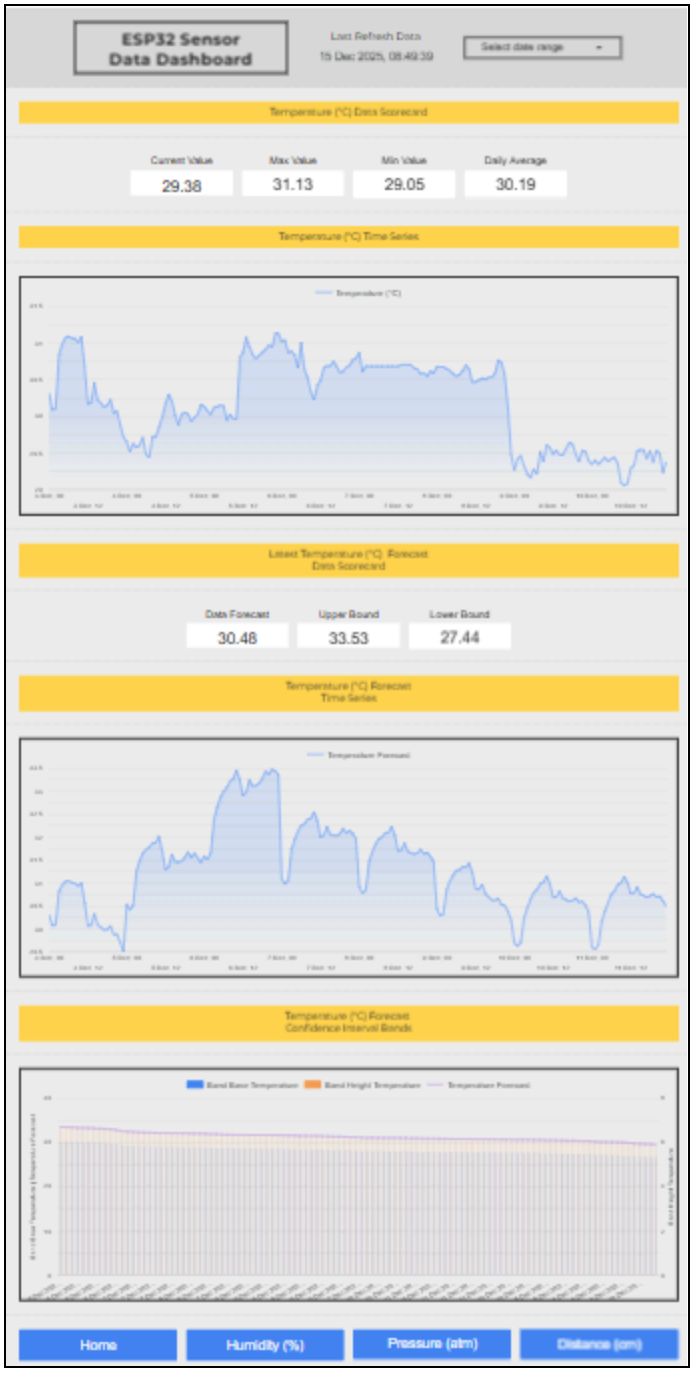


Figure 32: Temperature (°C) Dashboard.

For second page also displays title named “ESP32 Sensor Data Dashboard”, a date range control, 7 separate metrics scorecard which displays temperature current value, maximum value, minimum value and daily average and temperature forecast data, upper bound and lower bound, 2 time series chart of historical temperature data and temperature forecast data, a combo chart of temperature confidence interval bands and 4 buttons which navigates to Home, Humidity (%), Pressure (atm) and Distance (cm) pages.

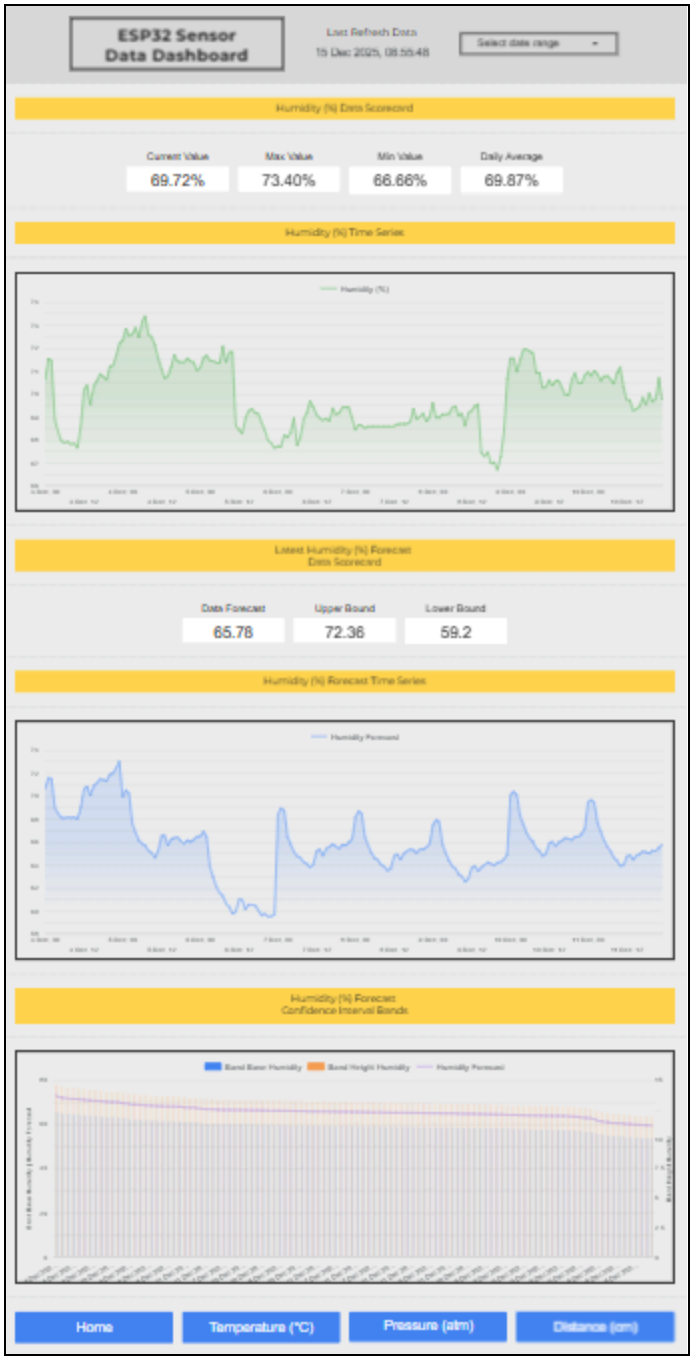


Figure 33: Humidity (%) Dashboard.

For third page also displays the same title, a date range control, 7 separate metrics scorecard which displays humidity current value, maximum value, minimum value and daily average and humidity forecast data, upper bound and lower bound, 2 time series chart of historical humidity data and humidity forecast data, a combo chart of humidity confidence interval bands and 4 buttons which navigates to Home, Temperature (°C), Pressure (atm) and Distance (cm) pages.

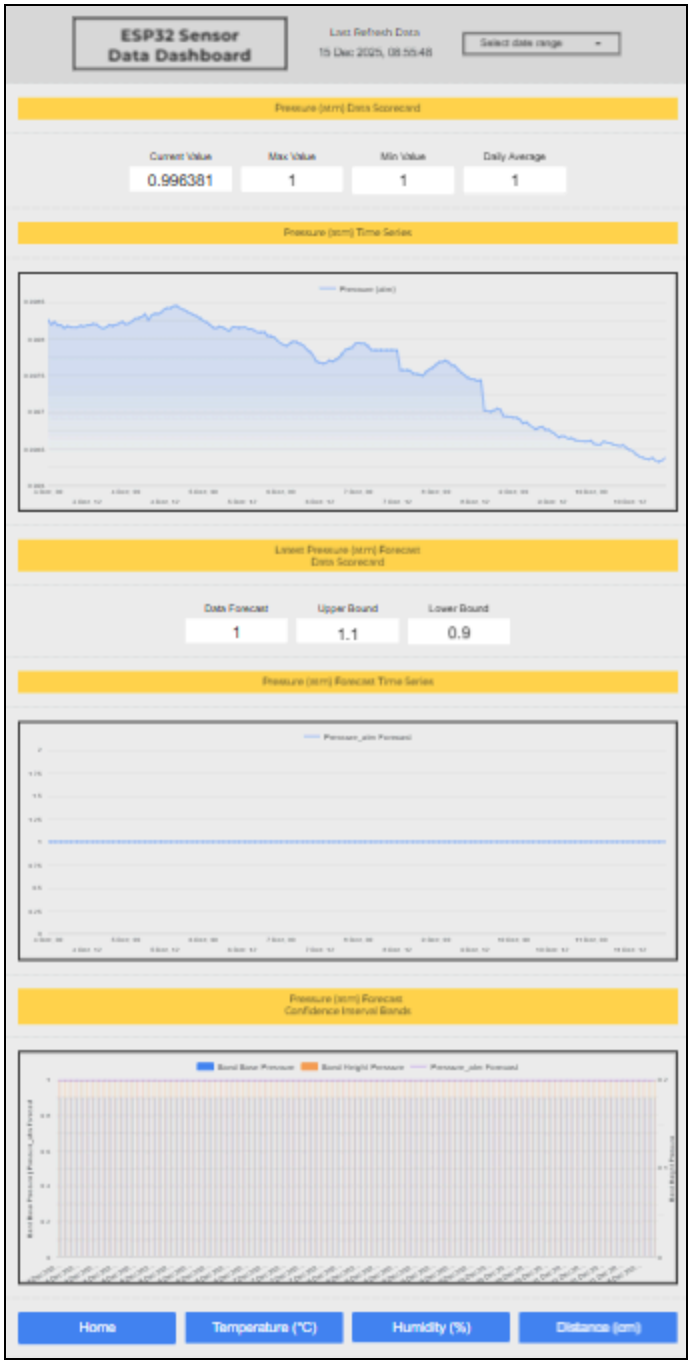


Figure 34: Pressure (atm) Dashboard.

At fourth page also displays title named “ESP32 Sensor Data Dashboard”, a date range control, 7 separate metrics scorecard which displays pressure current value, maximum value, minimum value and daily average and pressure forecast data, upper bound and lower bound, 2 time series chart of historical pressure data and pressure forecast data, a combo chart of pressure confidence interval bands and 4 buttons which navigates to Home, Temperature (°C), Humidity (%) and Distance (cm) pages.

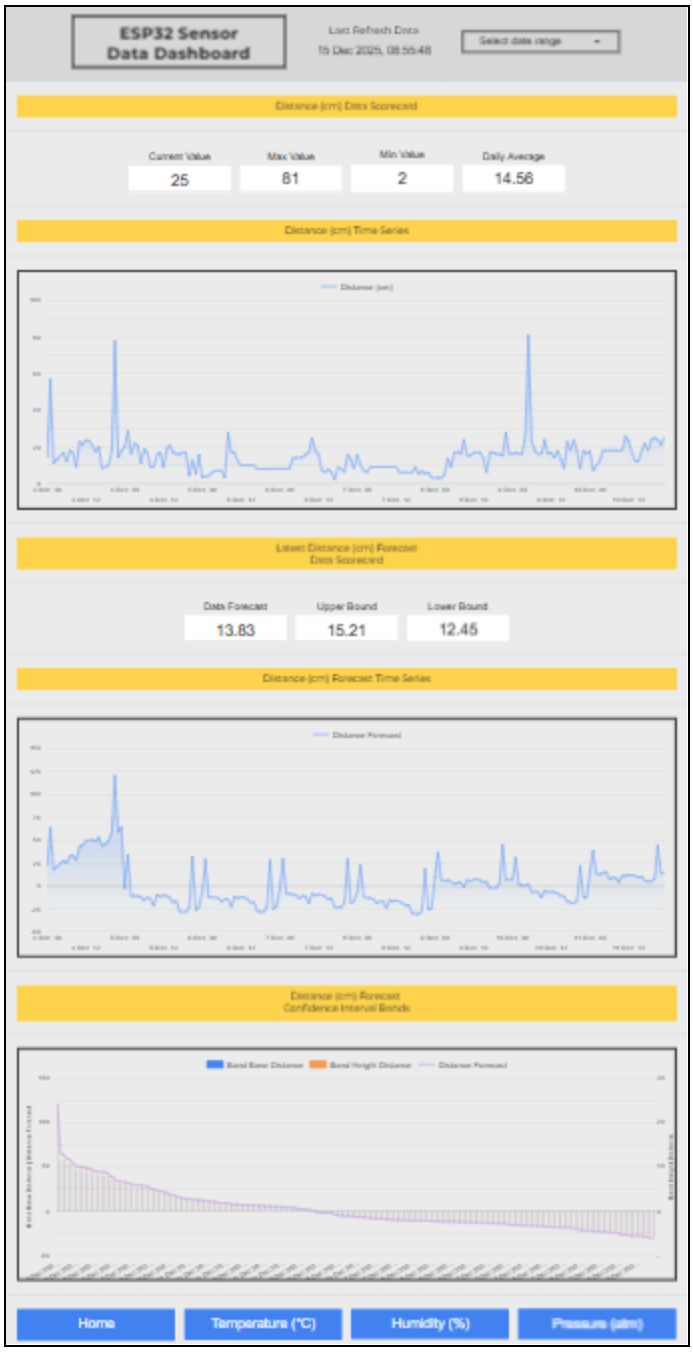


Figure 35: Distance (cm) Dashboard.

In fifth page also displays title named “ESP32 Sensor Data Dashboard”, a date range control, 7 separate metrics scorecard which displays distance current value, maximum value, minimum value and daily average and distance forecast data, upper bound and lower bound, 2 time series chart of historical distance data and distance forecast data, a combo chart of distance confidence interval bands and 4 buttons which navigates to Home, Temperature (°C), Humidity (%) and Pressure (atm) pages.

3. Android Studio Dashboard App.

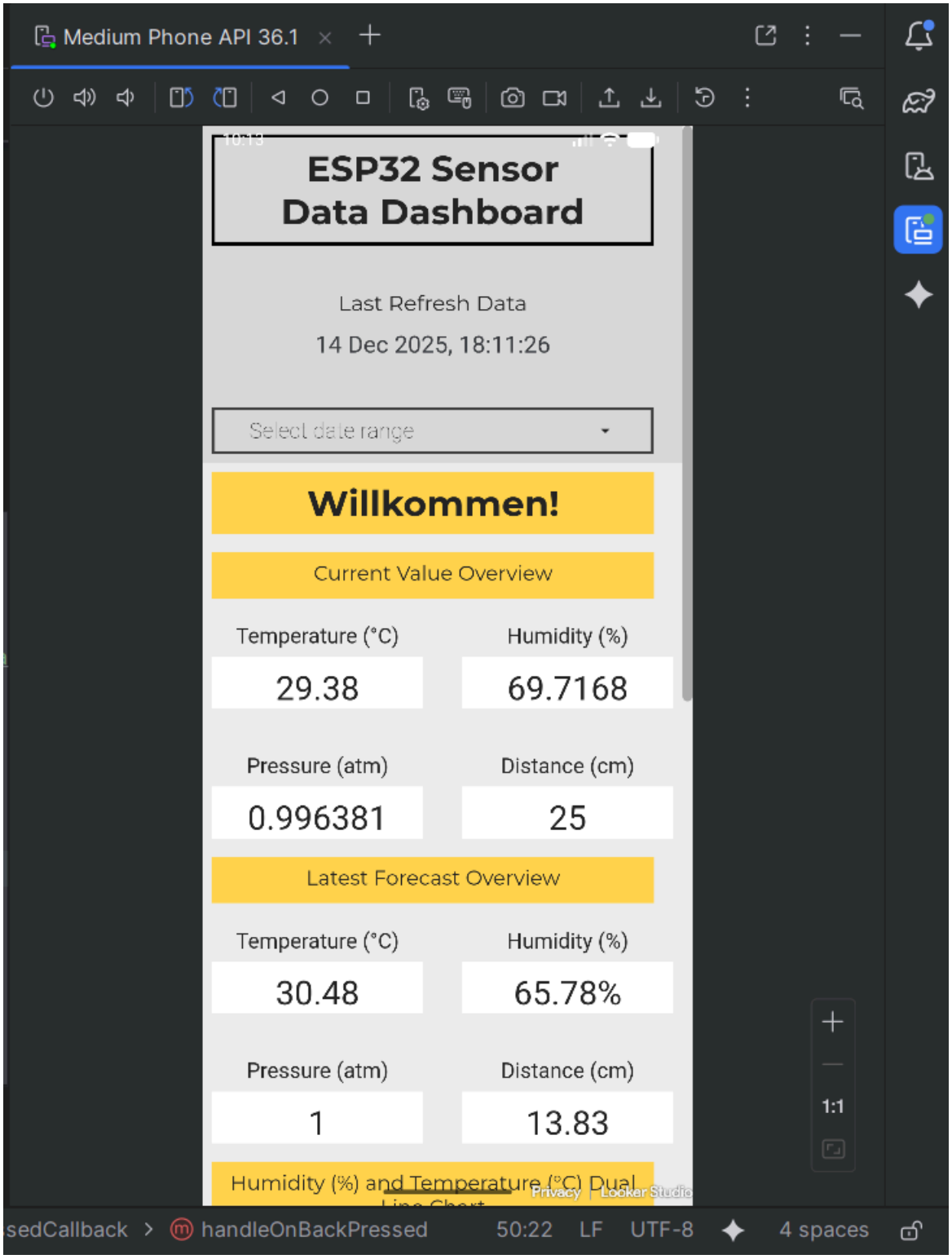


Figure 36: ESP32 Sensor Data Logger Dashboard App Running in Android Studio.

Collected Data Analysis

The sensors data reading collected from the ESP32 includes temperature (°C), humidity (%), pressure (atm) and distance (cm) that were then logged into Google Sheets and visualized through Looker Studio dashboards. The recording period from 3 December 2025 to 10 December 2025 shows normal sensor behavior with small variations across all 3 parameters.

1. Temperature Data Analysis and Insight:

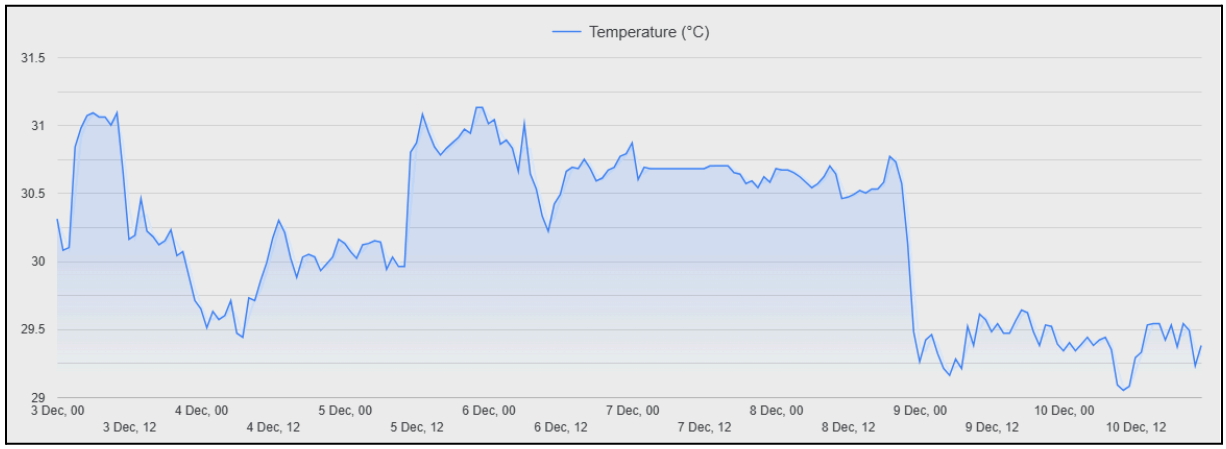


Figure 37: Temperature Data Line Chart.

In Figure 24, the temperature began ranged 31°C before decreasing steadily to around 29.5°C. A noticeable rise occurred during 4 December 2025 and return back to 31°C to 31.3°C the next day due to a momentary change in surrounding airflow or heat from nearby components. The sudden drop occurred on 9 December 2025 suggests a significant environmental change, such as cooling or ventilation activation. The data shown is stable in overall which suggests that the sensor is performing reliably.

2. Humidity Data Analysis and Insight:

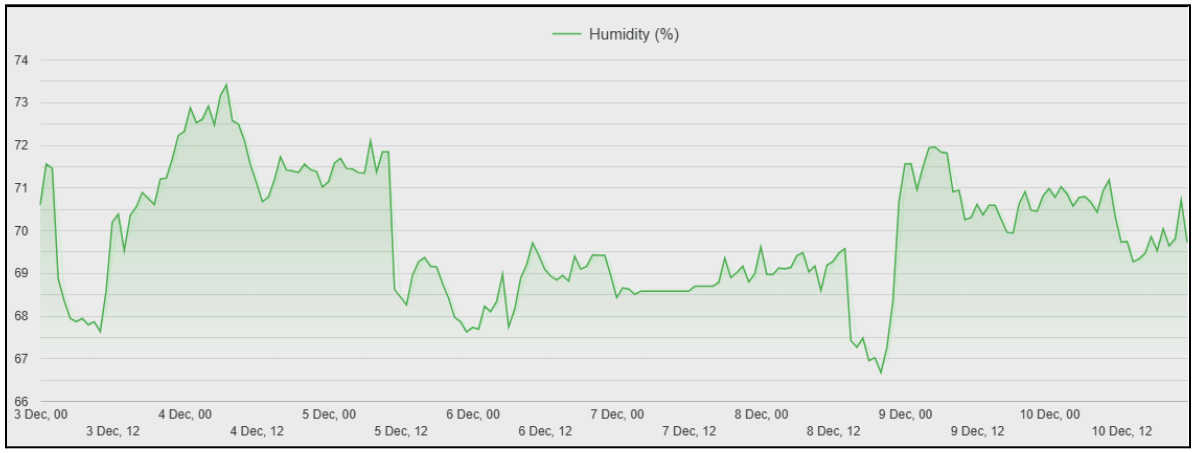


Figure 38: Humidity Data Line Chart.

In Figure 25, humidity readings were ranged between 68% to 74%. Humidity peak gradually when temperature reached its lowest. This chart shows the expected inverse relationship between temperature and humidity which was when temperature decreased, humidity will increased. The data remained smooth and stable thus demonstrating reliable sensor behaviour with minimal external disturbances.

3. Pressure Data Analysis and Insight.

Figure 39: Pressure Data Line Chart

In Figure ?, atmospheric pressure ranged between 0.9965 atm and 0.9985 atm which shows a slow and gradual decline. Small variations like rise and drop were observed during the first few days, but the overall downward trend suggests natural environmental or weather changes, such as approaching low-pressure conditions. The readings appeared clean and consistent, with no unusual spikes, confirming accurate pressure measurement.

4. Distance Data Analysis and Insight:

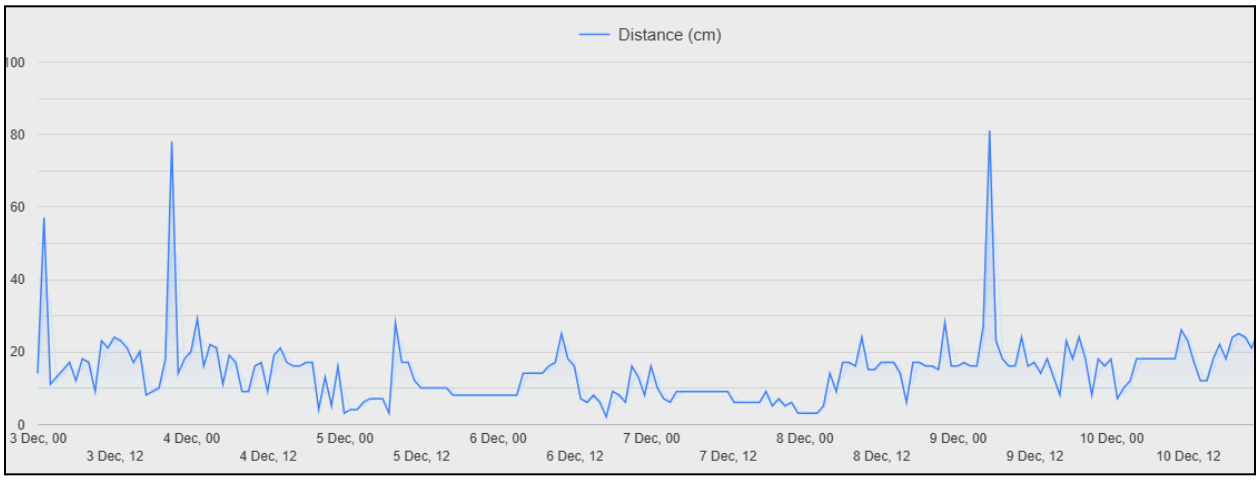


Figure 40: Distance Data Line Chart.

In Figure 26, the distance ranged between 2 cm to 29 cm and multiple spikes reached 57 cm and above. The reading that dropped, stabilized and spiked indicates the measured object was moved very close or far to the sensor. Despite the noise, the core measurements suggested that the object being monitored generally remained at a fixed and stable distance from the sensor.

Future Improvements

Future improvement for this system is the integration of Gemini AI to generate automated summaries and insights based on the collected sensor data. By leveraging Gemini advanced language and data interpretation capabilities, the system could automatically analyze daily or hourly sensor trends, detect notable patterns and provide concise, human readable reports. This AI powered summary feature would allow users to quickly understand key observations such as unusual temperature fluctuations, consistent humidity trends or forecast deviations without manually reviewing raw data or charts. Incorporating Gemini AI will enhance decision making and system monitoring, elevate the platform into a more intelligent, user friendly and proactive IoT solution.

Another potential future improvement is implement Exponential Moving Average (EMA) forecasting for the distance sensor data only because distance measurements are often more sensitive to short term fluctuations and noise caused by object movement or environmental interference. EMA forecasting would allow the system to smooth out sudden spikes while still remaining responsive to recent changes. This approach can improve the reliability of distance trend predictions, support early detection of abnormal variations and enhance the accuracy of proximity based monitoring or alert mechanisms.

GitHub Repositories Link

https://github.com/Kepengan/TSO2_PBL