FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING TECHNOLOGY

REPORT

BVI 3114

TECHNOLOGY SYSTEM OPTIMIZATION II

Mini Project: IoT Sensor Data Forecasting System With AI

No.	Student ID	Student Name	Section/Grp
1	VC22013	MUHAMMAD AKMAL HAIKAL BIN AZMI	01P/Group 3
2	VC22017	FARIS AZRI BIN FAISAL RIDZA	
3	VC22029	MUHAMMAD FAUZAN SYAFIQ BIN ROSLAN	

DATE OF REPORT SUBMISSION	19.0.2025

Contents

1.0 INTRODUCTION 3
Core Components 3
2.0 SYSTEM DESIGN AND IMPLEMENTATION 4
2.1 Hardware and Data Flow 4
2.2 ESP32 Firmware 4
2.3 Google Apps Script Backend 4
2.4 Forecasting Logic 4
3.0 MOBILE APPLICATION 5
3.1 Overview 5
3.2 Features 5
4.0 SYSTEM LINKS 5
5.0 SYSTEM SETUP & CHALLENGES 6
5.1 Sensor Configuration 7
5.2 Enhancements & Modifications 8
5.3 Common Issues & Fixes 8
6.0 VISUAL PROOF OF SYSTEM 9
7.0 CONCLUSION 10
8.0 APPENDICES – FULL CODE LISTINGS 11
APPENDIX A – APPS SCRIPT CODE 11
APPENDIX B – ARDUINO CODE 15
APPENDIX C – ANDROID MainActivity.java 17
APPENDIX D – activity_main.xml 19
APPENDIX E – Android Manifest 20

1.0 INTRODUCTION

Modern laboratories are highly sensitive environments where subtle changes in temperature, humidity, or unexpected obstacles can disrupt operations, compromise safety, and waste resources. Traditional monitoring methods are reactive and slow to address issues.

This project aimed to develop a predictive, automated monitoring solution using Internet of Things (IoT) technologies. By integrating real-time sensors with cloud analytics, mobile visualization and AI summarization, the system offers an intelligent, proactive approach to lab monitoring.

Core Components:

- **Sensing**: ESP32-based data collection from ultrasonic, temperature-humidity, light, and air quality sensors.
- **Cloud Integration**: Real-time data is transmitted to Google Sheets via HTTP POST using Google Apps Script.
- **Forecasting**: Short-term data prediction using Simple Moving Average (SMA) algorithms for environmental variables.
- **Dashboard Visualization**: Live metrics are displayed through a Google Looker Studio dashboard.
- **Gemini AI:** Help summarize the data and forecast that has been collected with the question that has been given
- **Mobile App**: A native Android application was built using Android Studio to provide portable real-time visualization.

2.0 SYSTEM DESIGN AND IMPLEMENTATION

2.1 Hardware and Data Flow

- **Microcontroller**: ESP32 Wi-Fi-enabled board
- **Sensors**: Grove ultrasonic and light sensors
- **Connectivity**: Wi-Fi + Google Apps Script Web App endpoint
- **Output**: Real-time logs in Google Sheets, analytics in Looker Studio, Gemini AI and app-based visualization

2.2 ESP32 Firmware

The ESP32 was programmed to:

- Connect to Wi-Fi
- Measure distance (and optionally light)
- Format and send data as JSON
- POST to a Google Apps Script endpoint every 10 seconds

✅ *Improved sampling interval from 10 minutes to 10 seconds for real-time tracking.*

2.3 Google Apps Script Backend

- Logs JSON payloads into a dedicated Google Sheet
- Automatically handles timestamps and data parsing
- Features a script-triggered forecast generator
- Supports menu-driven actions: Clear Data, Generate Forecasts

2.4 Forecasting Logic

Implemented in Apps Script using SMA:

- Generates 24-hour rolling forecasts
- Outputs forecast values with upper/lower bounds
- Automatically updates the “Forecasts” sheet

2.5 Gemini AI

- Implemented within Apps Acript
- Summarize the data within Sheet1(Real-time data) and Forecasts Data.

The ESP32 is programmed using Arduino IDE to send ultrasonic sensor data every 10 seconds via HTTP POST to a Google Apps Script endpoint. Full code provided in Appendix A.

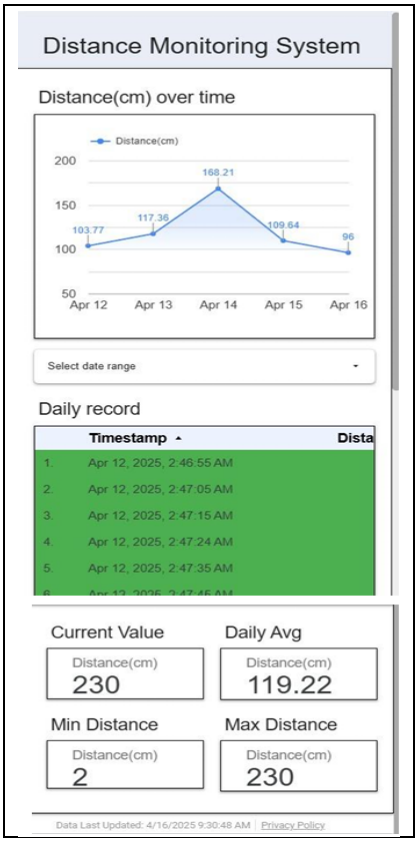
3.0 MOBILE APPLICATION

3.1 Overview

A lightweight Android app was developed using WebView to embed the real-time dashboard into a mobile-friendly layout.

3.2 Features

- Embedded Looker Studio dashboard
- Real-time readings
- Progress bar for UX
- Zoom and scroll enabled for responsive viewing



4.0 SYSTEM LINKS

📊 Google Sheets (Live Data): Google Sheets

📈 Looker Studio Dashboard: Looker Studio

5.0 GEMINI AI

Demonstrates a real-time sensor data logging and forecasting system integrated with AI-based summarization using Google Apps Script. Data is collected and logged into Google Sheets. Forecasting is performed using Exponential and Simple Moving Averages (EMA & SMA), while Gemini AI generates automatic summaries to provide meaningful insights.

5.1 Sequence Of Operation

5.1.1 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);
}
```

5.1.2 Data Prepared And Collected

	A	B
1	Timestamp	Distance(cm
2	4/12/2025 2:46:55	127.00
3	4/12/2025 2:47:05	100.00
4	4/12/2025 2:47:15	122.00
5	4/12/2025 2:47:24	128.00
6	4/12/2025 2:47:35	130.00
7	4/12/2025 2:47:45	125.00
8	4/12/2025 2:47:58	90.00
9	4/12/2025 2:48:08	17.00
10	4/12/2025 2:48:15	130.00
11	4/12/2025 2:48:24	122.00
12	4/12/2025 2:48:32	135.00
13	4/12/2025 2:48:45	112.00
14	4/12/2025 2:48:55	11.00
15	4/13/2025 2:49:04	122.00
16	4/13/2025 2:49:16	123.00
17	4/13/2025 2:49:25	102.00
18	4/13/2025 2:49:35	108.00
19	4/13/2025 2:49:44	109.00
20	4/13/2025 2:49:55	112.00
21	4/13/2025 2:50:05	59.00
22	4/13/2025 2:50:15	12.00
23	4/13/2025 2:50:25	123.00
24	4/13/2025 2:50:34	124.00
25	4/13/2025 2:50:45	180.00
26	4/13/2025 2:50:55	120.00
27	4/13/2025 2:51:05	227.00

	A	B	C	D
1	timestampe	forecast	upper	lower
2	4/16/2025 3:58:14	120	132	108
3	4/16/2025 4:58:14	128.33	141.16	115.5
4	4/16/2025 5:58:14	140.56	154.62	126.5
5	4/16/2025 6:58:14	143.98	158.38	129.58
6	4/16/2025 7:58:14	129.65	142.62	116.69
7	4/16/2025 8:58:14	113.42	124.76	102.08
8	4/16/2025 9:58:14	129.32	142.25	116.39
9	4/16/2025 10:58:14	130.88	143.97	117.79
10	4/16/2025 11:58:14	131.3	144.43	118.17
11	4/16/2025 12:58:14	129.76	142.74	116.78
12	4/16/2025 13:58:14	127.39	140.13	114.65
13	4/16/2025 14:58:14	127.01	139.71	114.31
14	4/16/2025 15:58:14	129.28	142.21	116.35
15	4/16/2025 16:58:14	129.27	142.2	116.34
16	4/16/2025 17:58:14	129	141.9	116.1
17	4/16/2025 18:58:14	128.62	141.48	115.76
18	4/16/2025 19:58:14	128.43	141.27	115.59
19	4/16/2025 20:58:14	128.6	141.46	115.74
20	4/16/2025 21:58:14	128.87	141.76	115.98
21	4/16/2025 22:58:14	128.8	141.68	115.92
22	4/16/2025 23:58:14	128.72	141.59	115.85
23	4/17/2025 0:58:14	128.67	141.54	115.8
24	4/17/2025 1:58:14	128.68	141.55	115.81
25	4/17/2025 2:58:14	128.72	141.59	115.85

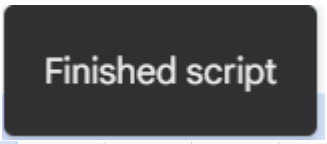
5.1.3 Send to Gemini AI

```
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);
```

5.1.4 Send to Gemini AI

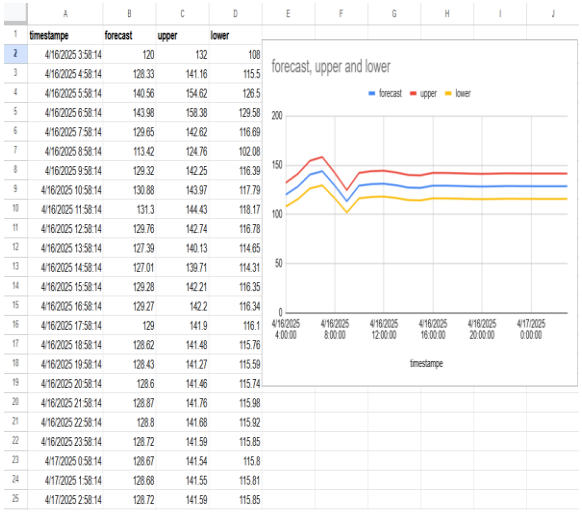
```
function getGeminiSummary(prompt) {
  const payload = {
    contents: [{ parts: [{ text: prompt }] }]
  };
  const options = {
    method: 'POST',
    contentType: 'application/json',
    payload: JSON.stringify(payload),
    muteHttpExceptions: true
  };
  const url = 'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?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);
```

5.1.5 Summary is received and saved to Apps Script

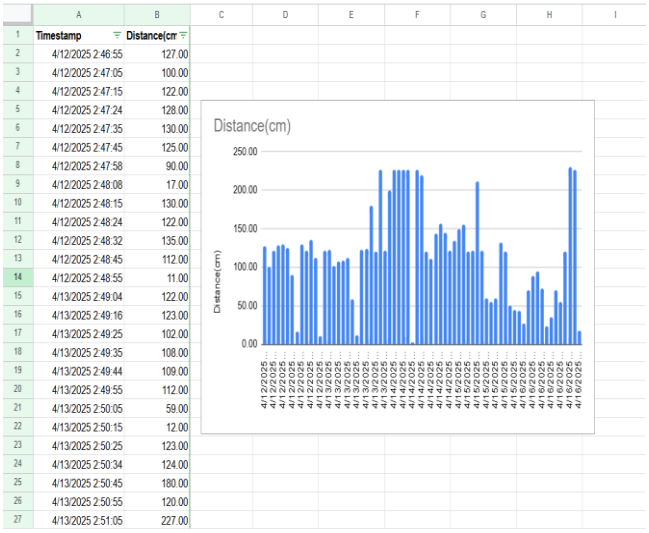


1	Real-Time Data Summary:
2	Okay, let's break down the trends in the distance sensor data. **Overall Observations:** * **Large Fluctuations:** The distance readings are quite variable, suggesting movement or changes in the environment being measured. * **Short-Term Variability:** There are significant changes in distance within short timeframes (seconds to minutes). This is a key characteristic. **Trends by Day:** * **Sat Apr 12 2025:** Generally, the distances are fluctuating between 90-135cm, with a sudden drop to 17cm before increasing back up. * **Sun Apr 13 2025:** Initially, the distances stay around the 100-125cm range. Later, there are more significant drops to shorter distances (12cm, 59cm). The data ends with distances fluctuating from 120 to 227 cm. * **Mon Apr 14 2025:** The distances start high, frequently at 227cm. Then drops sharply to a very low value of 2 cm. The distances then settle into a range of 111cm to 156cm before a reading of 122. * **Tue Apr 15 2025:** The distances are inconsistent, varying from 50cm to 211cm * **Wed Apr 16 2025:** The distances are primarily short (23-95cm), except for the large spike at the end. **Summary:** The distance readings show considerable variability. The sensor data suggests that the object or environment being measured is dynamic and subject to frequent changes in distance. In the beginning, the distance fluctuates around 120cm, before deci

6.0 CHARTING AND VISUALIZATION REAL-TIME DATA AND FORECASTS



	timestampe	forecast	upper	lower
2	4/16/2025 3:58:14	120	132	108
3	4/16/2025 4:58:14	128.33	141.16	115.5
4	4/16/2025 5:58:14	140.56	154.62	126.5
5	4/16/2025 6:58:14	143.98	158.38	129.58
6	4/16/2025 7:58:14	129.65	142.62	116.69
7	4/16/2025 8:58:14	113.42	124.76	102.08
8	4/16/2025 9:58:14	129.32	142.25	116.39
9	4/16/2025 10:58:14	130.88	143.97	117.79
10	4/16/2025 11:58:14	131.3	144.43	118.17
11	4/16/2025 12:58:14	129.76	142.74	116.78
12	4/16/2025 13:58:14	127.39	140.13	114.65
13	4/16/2025 14:58:14	127.01	139.71	114.31
14	4/16/2025 15:58:14	129.28	142.21	116.35
15	4/16/2025 16:58:14	129.27	142.2	116.34
16	4/16/2025 17:58:14	129	141.9	116.1
17	4/16/2025 18:58:14	128.62	141.48	115.76
18	4/16/2025 19:58:14	128.43	141.27	115.59
19	4/16/2025 20:58:14	128.6	141.46	115.74
20	4/16/2025 21:58:14	128.87	141.76	115.98
21	4/16/2025 22:58:14	128.8	141.68	115.92
22	4/16/2025 23:58:14	128.72	141.59	115.85
23	4/17/2025 0:58:14	128.67	141.54	115.8
24	4/17/2025 1:58:14	128.68	141.55	115.81
25	4/17/2025 2:58:14	128.72	141.59	115.85



	Timestamp	Distance(cm
2	4/12/2025 2:46:55	127.00
3	4/12/2025 2:47:05	100.00
4	4/12/2025 2:47:15	122.00
5	4/12/2025 2:47:24	128.00
6	4/12/2025 2:47:35	130.00
7	4/12/2025 2:47:45	125.00
8	4/12/2025 2:47:58	90.00
9	4/12/2025 2:48:08	17.00
10	4/12/2025 2:48:15	130.00
11	4/12/2025 2:48:24	122.00
12	4/12/2025 2:48:32	135.00
13	4/12/2025 2:48:45	112.00
14	4/12/2025 2:48:55	11.00
15	4/13/2025 2:49:04	122.00
16	4/13/2025 2:49:16	123.00
17	4/13/2025 2:49:25	102.00
18	4/13/2025 2:49:35	108.00
19	4/13/2025 2:49:44	109.00
20	4/13/2025 2:49:55	112.00
21	4/13/2025 2:50:05	59.00
22	4/13/2025 2:50:15	12.00
23	4/13/2025 2:50:25	123.00
24	4/13/2025 2:50:34	124.00
25	4/13/2025 2:50:45	180.00
26	4/13/2025 2:50:55	120.00
27	4/13/2025 2:51:05	227.00

7.0 SYSTEM SETUP & CHALLENGES

7.1 Sensor Configuration

- **Ultrasonic Ranger**:
  - VCC → 5V
  - GND → GND
  - SIG → GPIO 13 on ESP32

7.2 Enhancements & Modifications

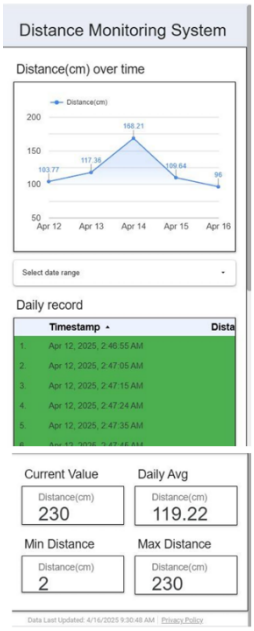
- Added light sensor integration with ultrasonic readings
- Developed web endpoint for Google Sheets using Apps Script

7.3 Common Issues & Fixes

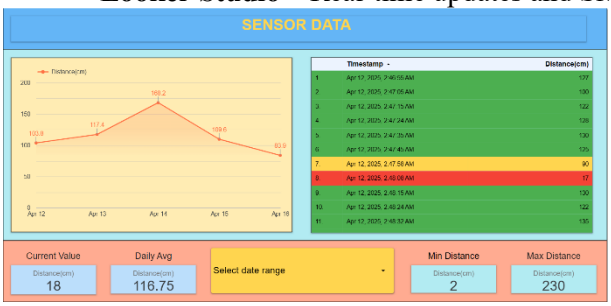
Issue	Resolution
Initial slow data updates	Reduced interval from 10 minutes → 10s
Data overflow in sheets	Added "Clear Data" menu in Apps Script
Mobile dashboard overflow issues	Used horizontal scroll and scaling

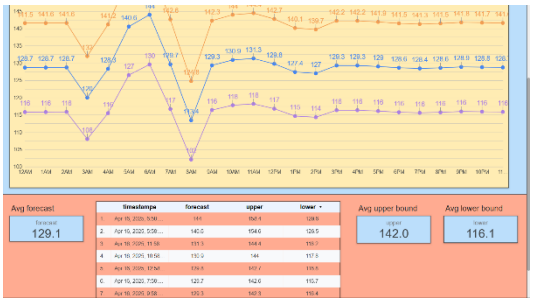
8.0 VISUAL PROOF OF SYSTEM

- **App Screenshots** – Showing embedded dashboard and refresh feature



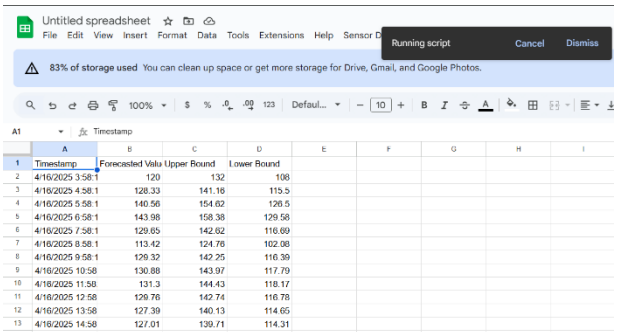
- **Looker Studio** – Real-time updates and SMA forecast



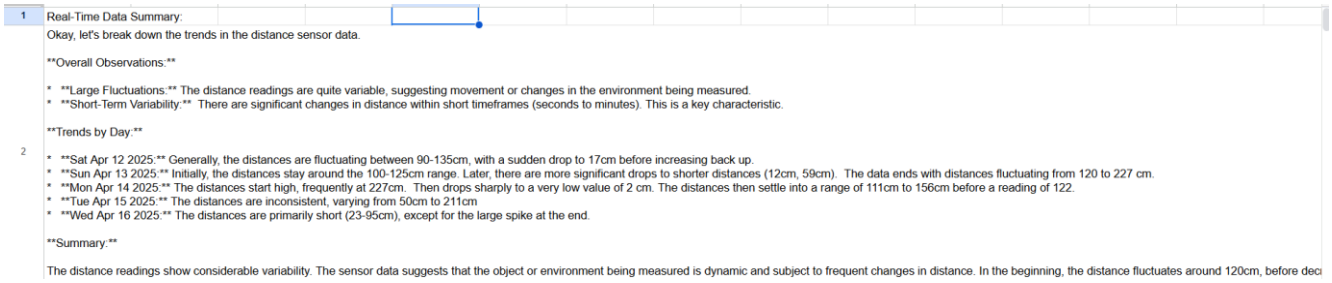


- **Google Sheets** – Live logging of sensor values

	A	B
1	Timestamp	Distance(cm
2	4/12/2025 2:46:55	127.00
3	4/12/2025 2:47:05	100.00
4	4/12/2025 2:47:15	122.00
5	4/12/2025 2:47:24	128.00
6	4/12/2025 2:47:35	130.00
7	4/12/2025 2:47:45	125.00
8	4/12/2025 2:47:58	90.00
9	4/12/2025 2:48:08	17.00
10	4/12/2025 2:48:15	130.00
11	4/12/2025 2:48:24	122.00
12	4/12/2025 2:48:32	135.00
13	4/12/2025 2:48:45	112.00
14	4/12/2025 2:48:55	11.00
15	4/13/2025 2:49:04	122.00
16	4/13/2025 2:49:16	123.00
17	4/13/2025 2:49:25	102.00
18	4/13/2025 2:49:35	108.00
19	4/13/2025 2:49:44	109.00
20	4/13/2025 2:49:55	112.00
21	4/13/2025 2:50:05	59.00
22	4/13/2025 2:50:15	12.00
23	4/13/2025 2:50:25	123.00
24	4/13/2025 2:50:34	124.00



- **Gemini AI –** Results of summarization of Real-Time data and Forecasts data



Real-Time Data Summary:

Okay, let's break down the trends in the distance sensor data.

Overall Observations:

* **Large Fluctuations:** The distance readings are quite variable, suggesting movement or changes in the environment being measured.
* **Short-Term Variability:** There are significant changes in distance within short timeframes (seconds to minutes). This is a key characteristic.

Trends by Day:

* **Sat Apr 12 2025:** Generally, the distances are fluctuating between 90-135cm, with a sudden drop to 17cm before increasing back up.
* **Sun Apr 13 2025:** Initially, the distances stay around the 100-125cm range. Later, there are more significant drops to shorter distances (12cm, 59cm). The data ends with distances fluctuating from 120 to 227 cm.
* **Mon Apr 14 2025:** The distances start high, frequently at 227cm. Then drops sharply to a very low value of 2 cm. The distances then settle into a range of 111cm to 156cm before a reading of 122.
* **Tue Apr 15 2025:** The distances are inconsistent, varying from 50cm to 211cm
* **Wed Apr 16 2025:** The distances are primarily short (23-95cm), except for the large spike at the end.

Summary:

The distance readings show considerable variability. The sensor data suggests that the object or environment being measured is dynamic and subject to frequent changes in distance. In the beginning, the distance fluctuates around 120cm, before deci

9.0 CONCLUSION

This PBL project showcased how low-cost IoT solutions can significantly improve lab safety and monitoring through real-time sensing, cloud data management, and mobile visualization. Forecasting environmental trends allowed us to act proactively rather than reactively, setting the stage for scalable smart lab environments.

10.0 APPENDICES – FULL CODE LISTINGS

APPENDIX A – APPS SCRIPT CODE

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

  ui.createMenu('Sensor Data')
    .addItem('Generate Forecasts', 'generateForecasts')
    .addItem('Clear All Data', 'clearData')
    .addToUi();
}

function summarizeAllSensorDataWithGemini() {
  const apiKey = 'AIzaSyDkOMALdQGQv6SHaastcv7FmIEjrSXusp4'; // 🔐 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);
}
```

```
// Script to receive sensor data from ESP32 and log it to Google Sheets
function doGet(e) {
  return handleResponse(e);
}

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

function handleResponse(e) {
  var lock = LockService.getScriptLock();
  lock.tryLock(5000); // Wait 5 seconds for other processes to complete
  try {
    // Get the active sheet
    var spreadsheet =
SpreadsheetApp.openByUrl("https://docs.google.com/spreadsheets/d/1jlimos4C4gYYEcMx9zxddugdrxA-
mfcRvSGrOPzSXEE/edit?gid=1020590958#gid=1020590958");
    var sheet = spreadsheet.getSheetByName("Sheet1");

    // Parse the incoming data
    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);
    }

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

    // Get sensor data based on what's available in the payload
    if (payload.temperature !== undefined) data.push(parseFloat(payload.temperature));
    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.stringify({
      'status': 'error',
      'message': error.toString()
    })).setMimeType(ContentService.MimeType.JSON);
  } finally {
    lock.releaseLock();
  }
}

// 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!');
}

// 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 timestamps = [];
  var sensorValues = [];

  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
  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;
}
```

APPENDIX B – 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/AKfycbz8tL6KiAsTixmGAz9g8FlHeyxLuQrTqSXmqcvMtZV-
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());
}

 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();
2.0 PROCCES
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("Distance: ");
 Serial.print(distance);
```

```
 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(scriptURL);
 http.addHeader("Content-Type", "application/json");

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

String jsonString;
 serializeJson(doc, 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();
```

APPENDIX C – ANDROID MainActivity.java

```
package com.example.distancemonitor;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.ProgressBar;
import android.widget.Button;


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

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

      dashboardWebView =
            findViewById(R.id.dashboardWebView);
      progressBar = findViewById(R.id.progressBar);
      Button refreshButton = findViewById(R.id.refreshButton);
      refreshButton.setOnClickListener(v -> dashboardWebView.reload());


      // Configure WebView settings
      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);

      // Hide progress bar when page loads
      dashboardWebView.setWebViewClient(new
                                    WebViewClient() {
                                       @Override
                                       public void onPageFinished(WebView view, String
                                             url) {
                                          progressBar.setVisibility(View.GONE);
                                          super.onPageFinished(view, url);
                                       }
                                    });
      // Replace with your actual Looker Studio dashboard embed URL
      String dashboardUrl = "https://lookerstudio.google.com/embed/reporting/33854f6a-849e-4a8f-85da
868509b91880/page/cr8GF";
      dashboardWebView.loadUrl(dashboardUrl);

   }
```

```
    @Override
    public void onBackPressed() {
        if (dashboardWebView.canGoBack()) {
            dashboardWebView.goBack();
        } else {
            super.onBackPressed();
        }
    }
}
```

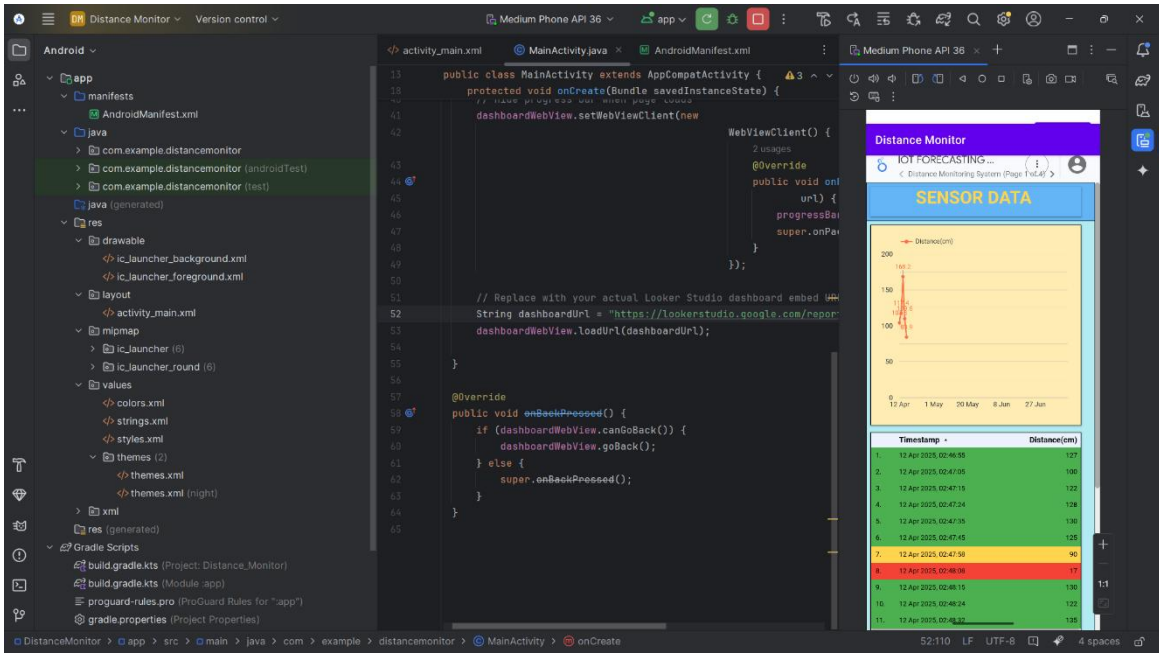
APPENDIX D – activity_main.xml

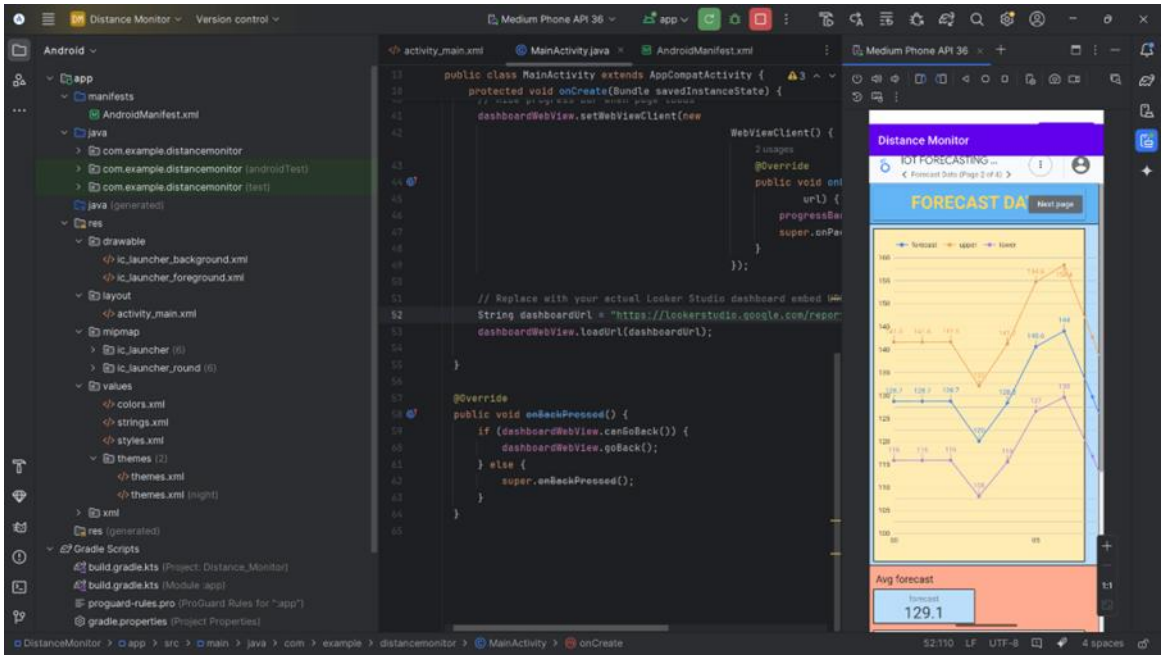
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"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

    <Button
        android:id="@+id/refreshButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Refresh"
        android:layout_margin="16dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <!-- HorizontalScrollView wrapping WebView -->
    <HorizontalScrollView
        android:id="@+id/horizontalScrollView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:fillViewport="true"
        android:scrollbars="horizontal"
        app:layout_constraintTop_toBottomOf="@id/refreshButton"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent">

        <WebView
            android:id="@+id/dashboardWebView"
            android:layout_width="wrap_content"
            android:layout_height="match_parent" />
    </HorizontalScrollView>

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

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

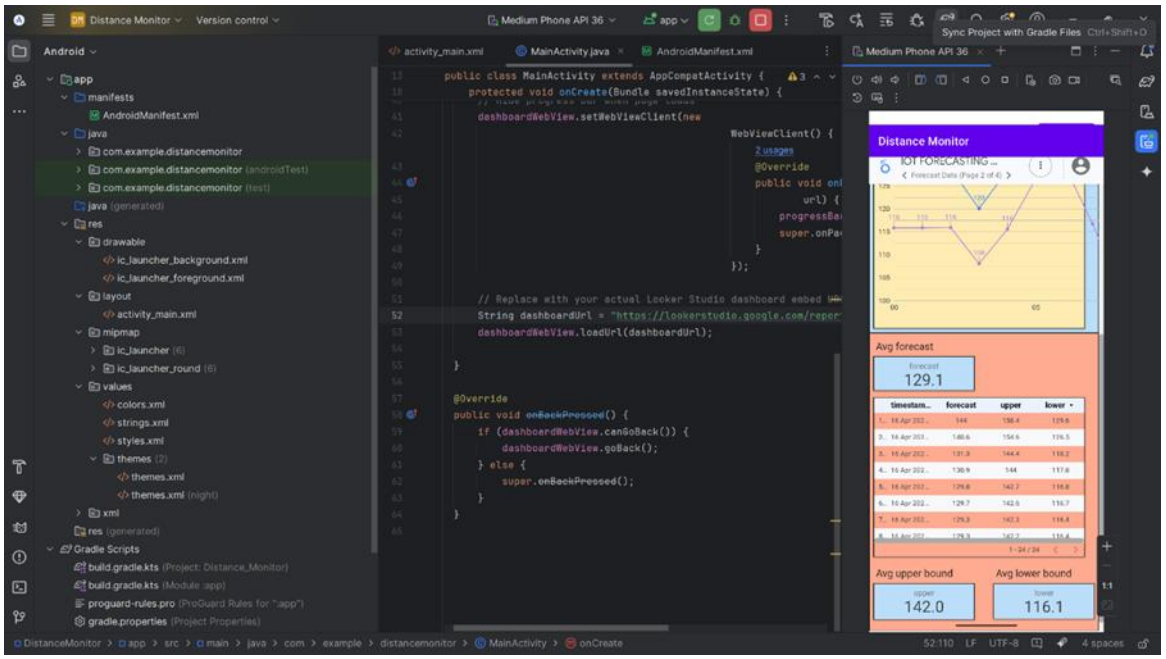
APPENDIX E – Android Manifest

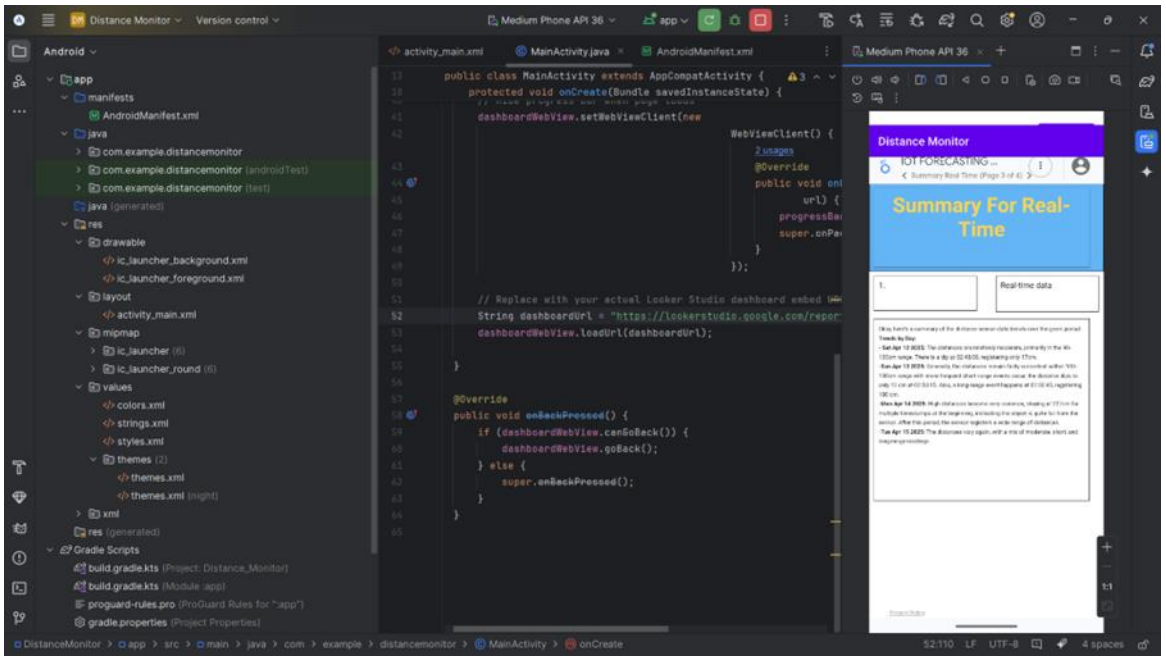
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
   xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.distancemonitor">

   <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="com.example.distancemonitor.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
```

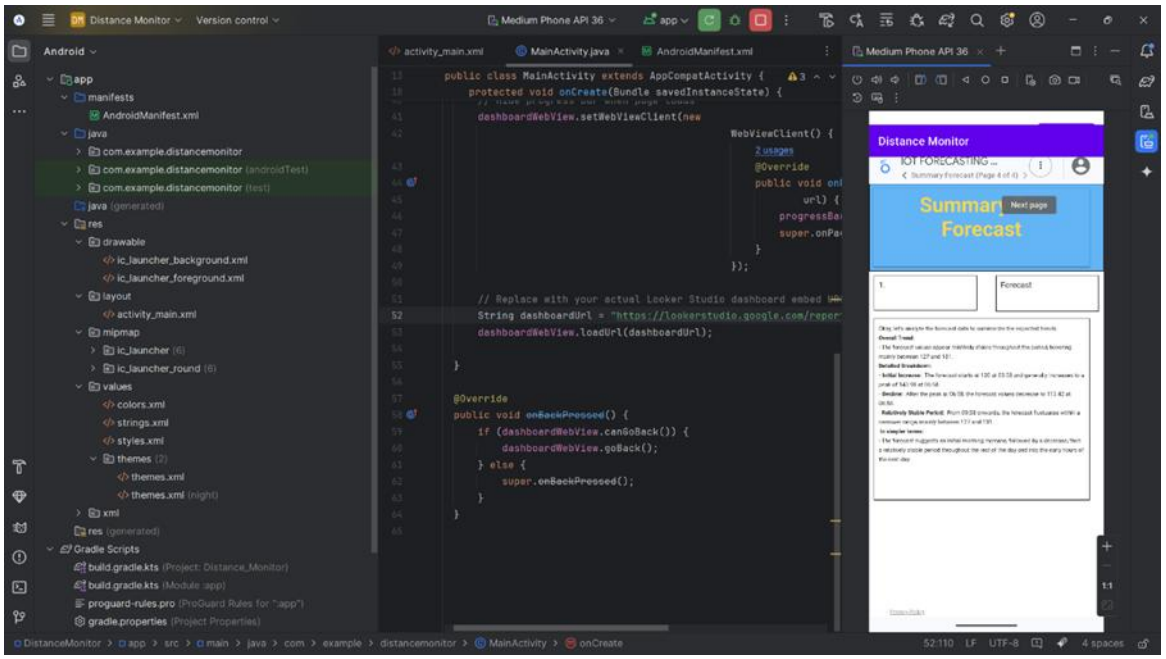
APPENDIX F – ANDROID STUDIO APP RESULTS






Distance Monitor
Version control
Medium Phone API 36
app
Sync Project with Gradle Files
Android
activity_main.xml
MainActivity.java
AndroidManifest.xml
Medium Phone API 36
Distance Monitor
IOT FORECASTING ...
Avg forecast
129.1
Avg upper bound
142.0
Avg lower bound
116.1
DistanceMonitor › app › src › main › java › com › example › distancemonitor › MainActivity › onCreate
52:110 LF UTF-8 4 spaces


Distance Monitor
Version control
Medium Phone API 36
app
Android
activity_main.xml
MainActivity.java
AndroidManifest.xml
Medium Phone API 36
Distance Monitor
IOT FORECASTING ...
Summary For Real-Time
Real-time data
DistanceMonitor › app › src › main › java › com › example › distancemonitor › MainActivity › onCreate
52:110 LF UTF-8 4 spaces


Distance Monitor
Version control
Medium Phone API 36
app
Android
activity_main.xml
MainActivity.java
AndroidManifest.xml
app
manifests
AndroidManifest.xml
java
com.example.distancemonitor
com.example.distancemonitor (androidTest)
com.example.distancemonitor (test)
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
ic_launcher (6)
ic_launcher_round (6)
values
colors.xml
strings.xml
styles.xml
themes (2)
themes.xml
themes.xml (night)
xml
res (generated)
Gradle Scripts
build.gradle.kts (Project: Distance_Monitor)
build.gradle.kts (Module :app)
proguard-rules.pro (ProGuard Rules for ":app")
gradle.properties (Project Properties)
public class MainActivity extends AppCompatActivity {
protected void onCreate(Bundle savedInstanceState) {
dashboardWebView.setWebViewClient(new
WebViewClient() {
2 usages
@Override
public void onP
url) {
progressBa
super.onPa
}
});
// Replace with your actual Looker Studio dashboard embed UR
String dashboardUrl = "https://lookerstudio.google.com/reporti
dashboardWebView.loadUrl(dashboardUrl);
}
@Override
public void onBackPressed() {
if (dashboardWebView.canGoBack()) {
dashboardWebView.goBack();
} else {
super.onBackPressed();
}
}
Medium Phone API 36
Distance Monitor
IOT FORECASTING ...
Summary Forecast (Page 4 of 4)
Summary Forecast
Next page
1.
Forecast
DistanceMonitor
app
src
main
java
com
example
distancemonitor
MainActivity
onCreate
52:110
LF
UTF-8
4 spaces

ANDROID APP RESULT