

Forecast

```
// 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) {
  // 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/1m3oHAY2DI6iSvqG
      7pEf7Or8MebjJTJhXH4hbwpEDcE/edit");
    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
    // Add appropriate sensor values to the data array
    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();
}
}
// 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 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
  // 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();
}

```

Gemini AI

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

function summarizeAllSensorDataWithGemini() {
  const apiKey = 'AlzaSyDkOMALdQGQv6SHaastcv7FmIEjrSXusp4'; // 🔒 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);
}
```