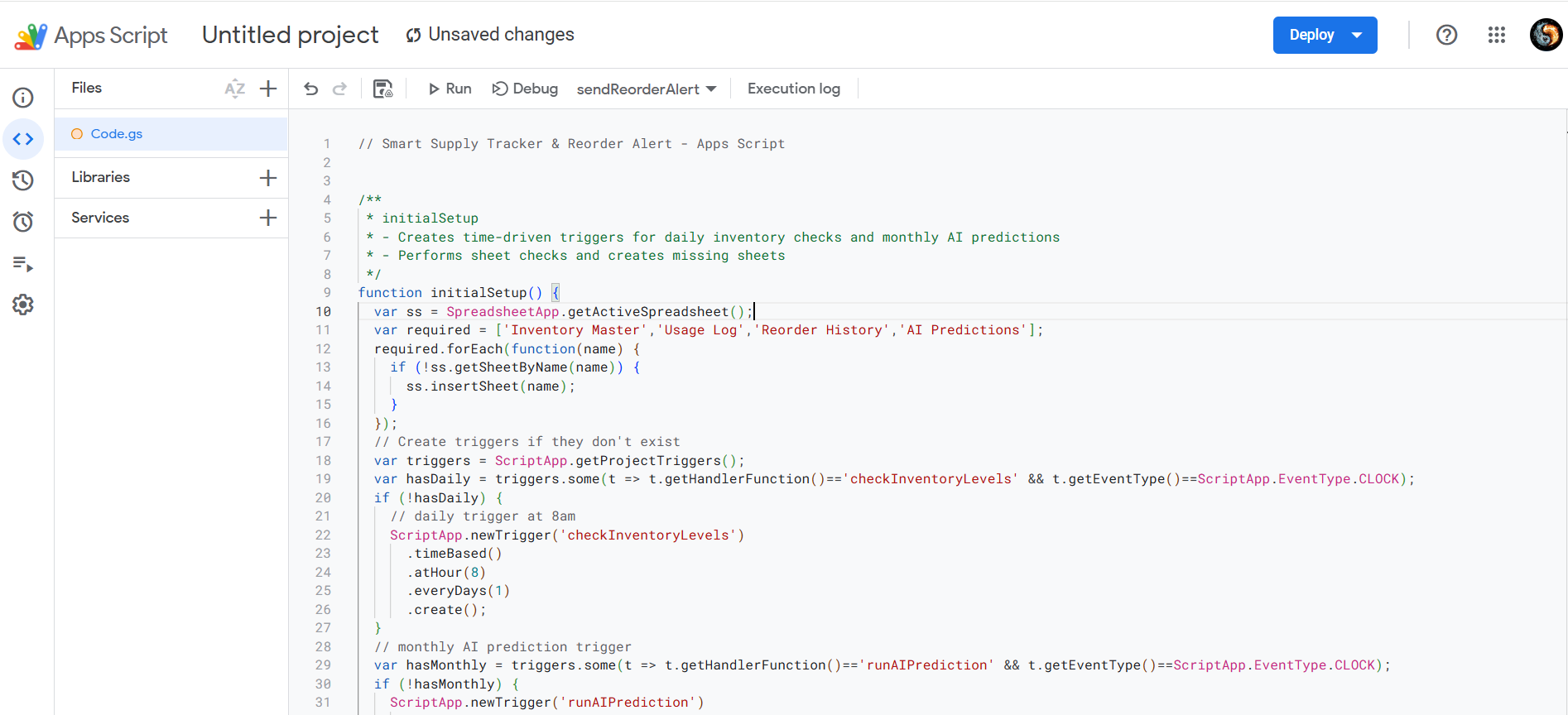
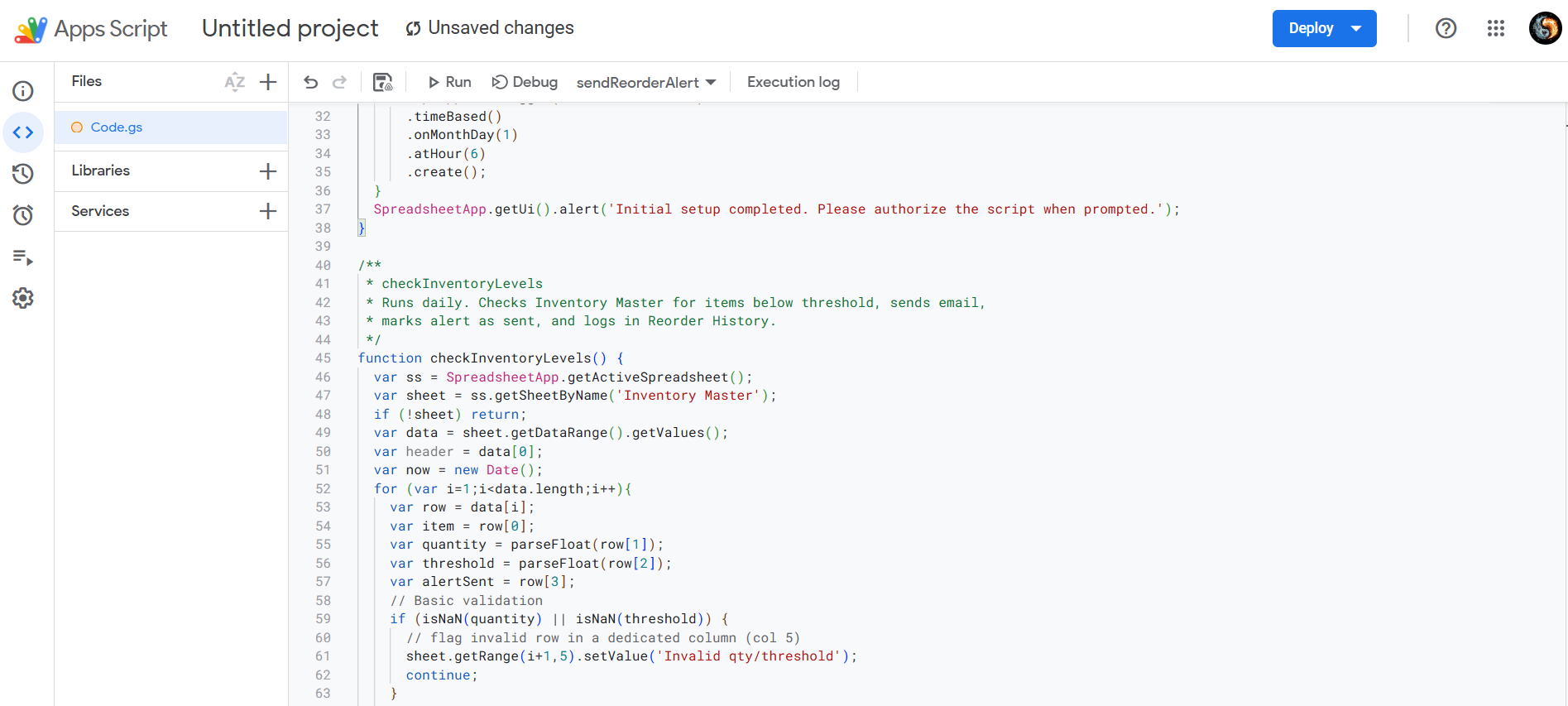
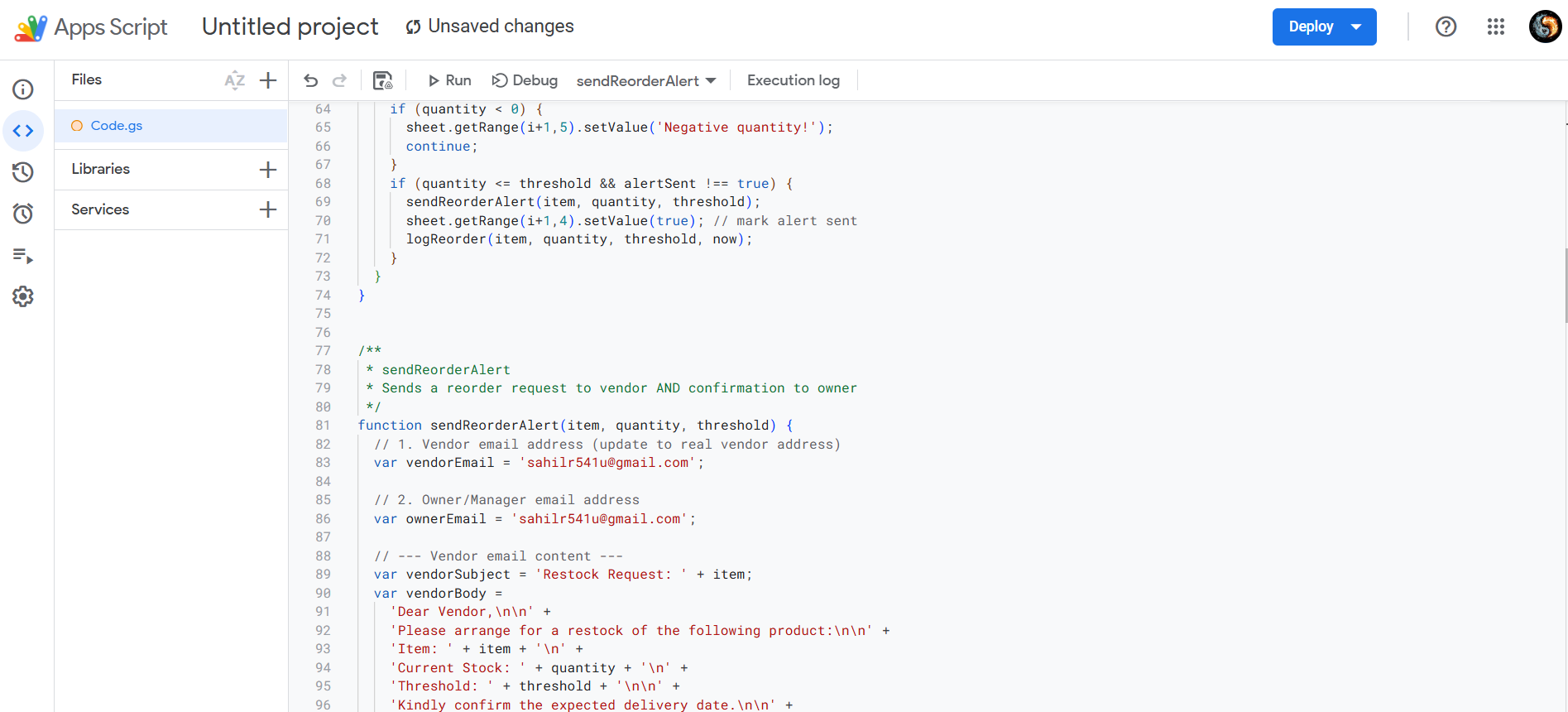
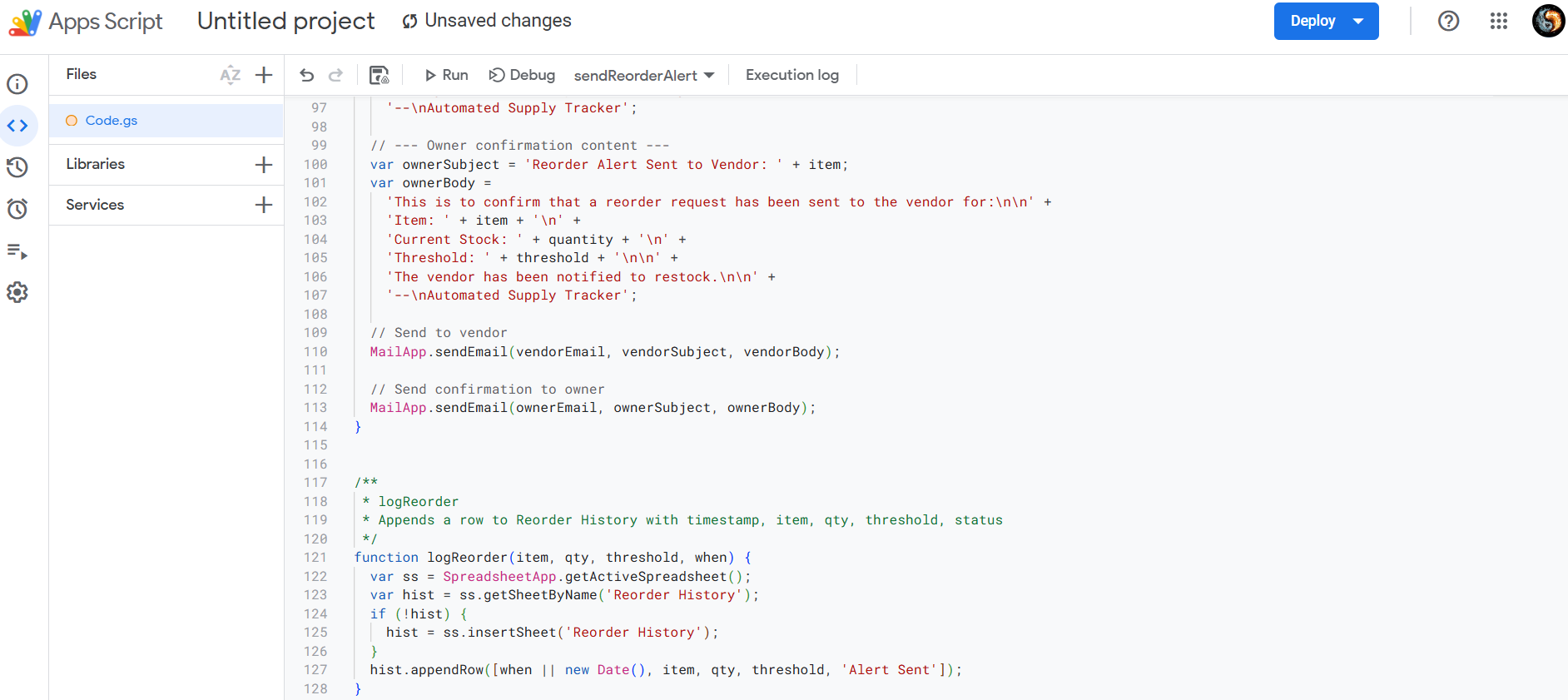
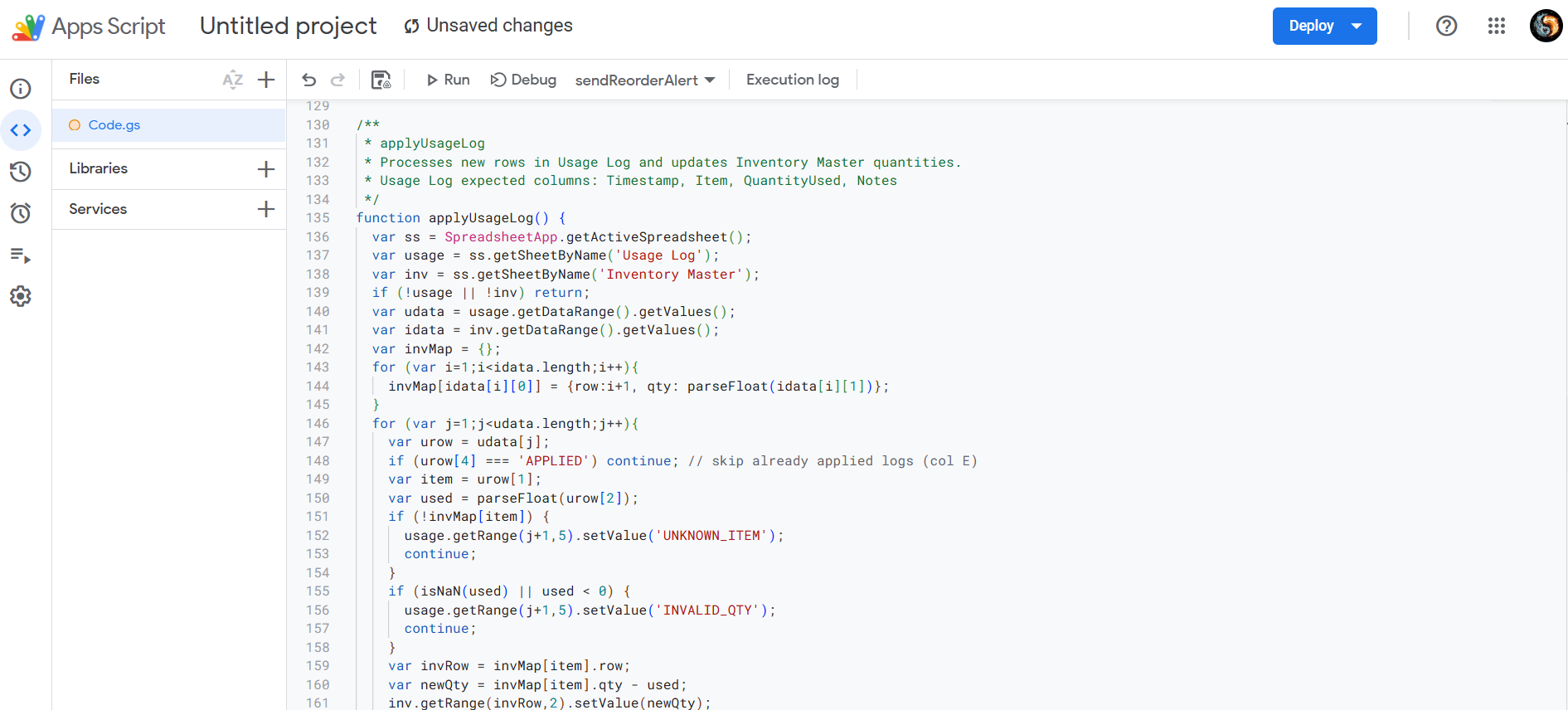
**Script.js Snapshot**

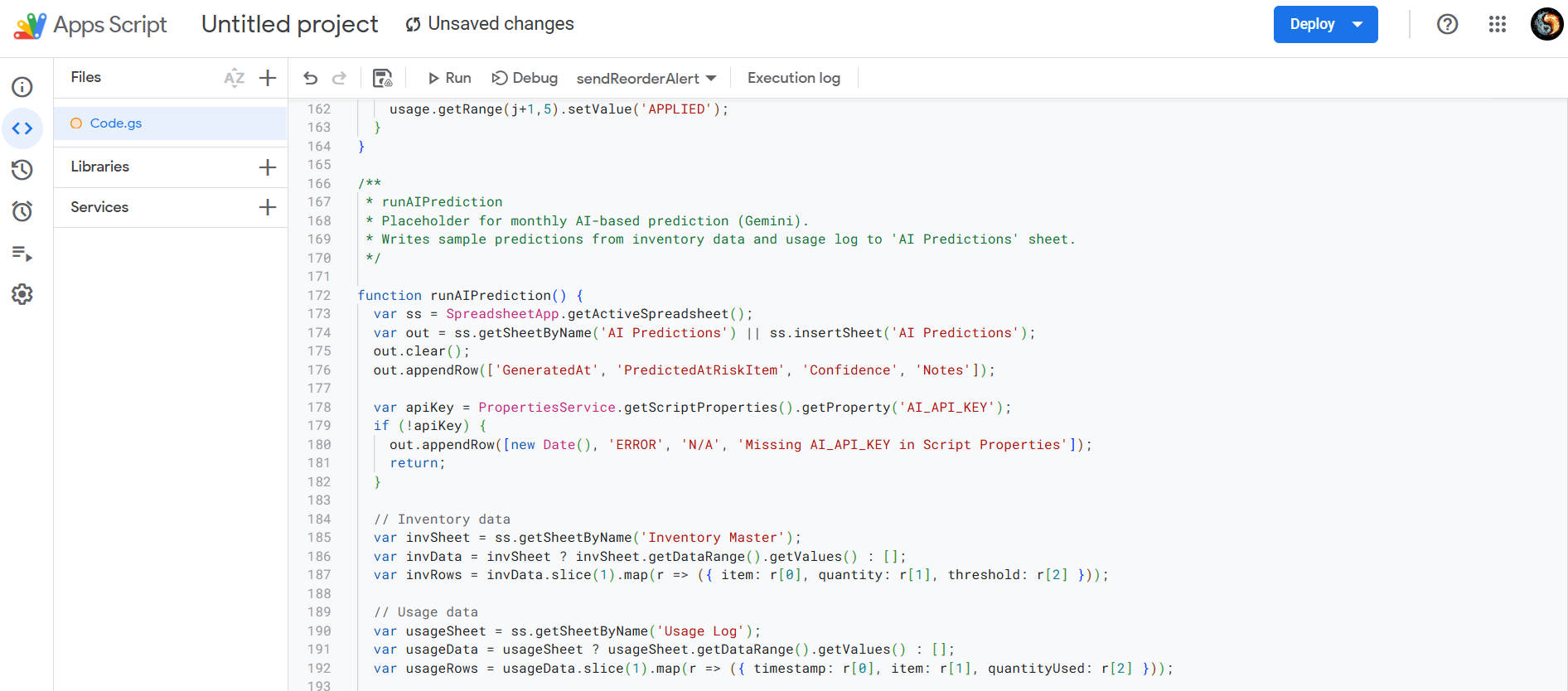


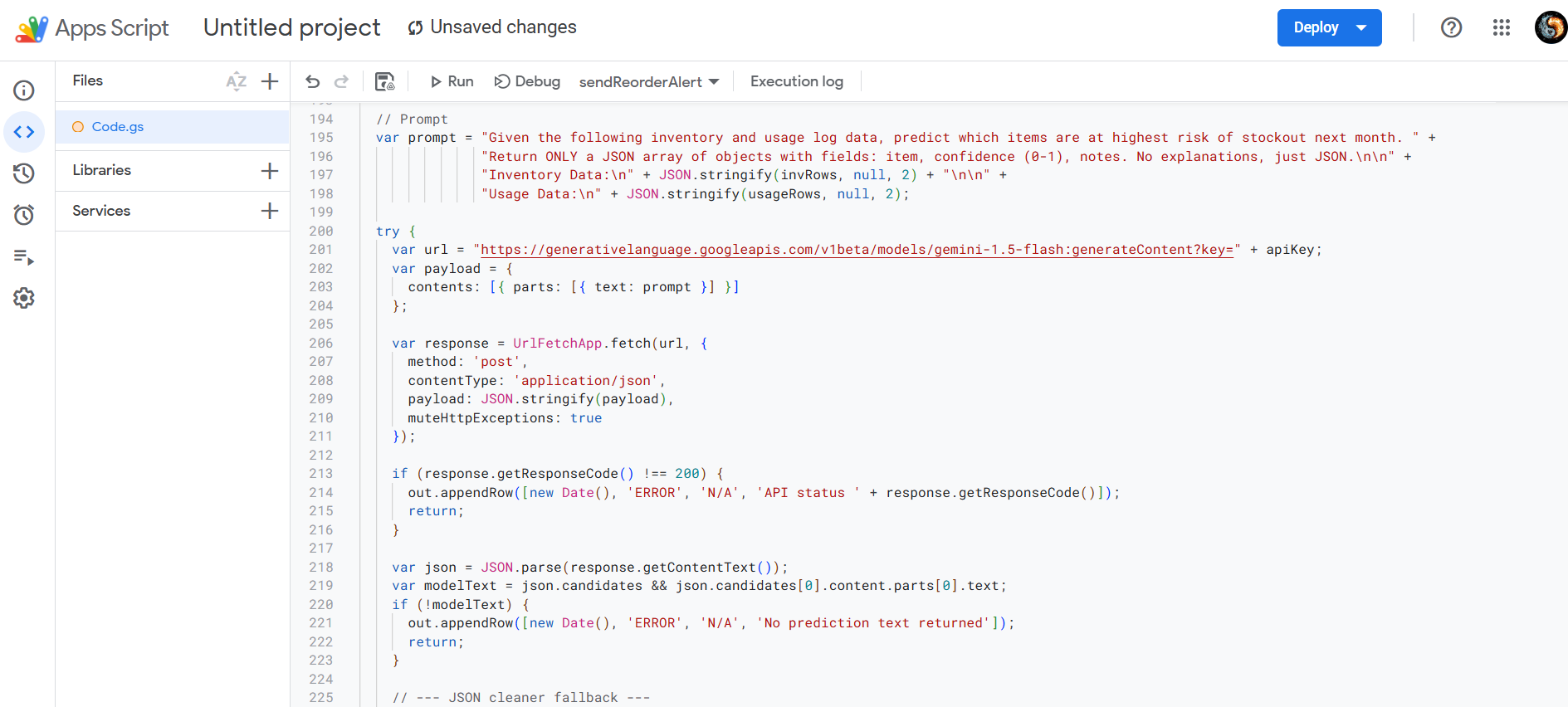




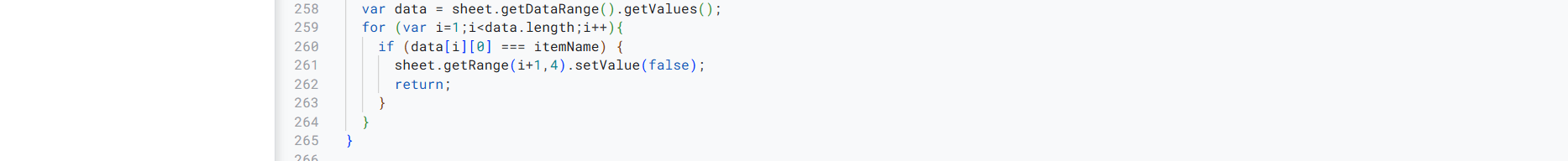












**Script.js Code**

// Smart Supply Tracker & Reorder Alert - Apps Script

/\*\*

 \* initialSetup

 \* - Creates time-driven triggers for daily inventory checks and monthly AI predictions

 \* - Performs sheet checks and creates missing sheets

 \*/

function initialSetup() {

  var ss = SpreadsheetApp.getActiveSpreadsheet();

  var required = ['Inventory Master','Usage Log','Reorder History','AI Predictions'];

  required.forEach(function(name) {

    if (!ss.getSheetByName(name)) {

      ss.insertSheet(name);

    }

  });

  // Create triggers if they don't exist

  var triggers = ScriptApp.getProjectTriggers();

  var hasDaily = triggers.some(t => t.getHandlerFunction()=='checkInventoryLevels' && t.getEventType()==ScriptApp.EventType.CLOCK);

  if (!hasDaily) {

    // daily trigger at 8am

    ScriptApp.newTrigger('checkInventoryLevels')

      .timeBased()

      .atHour(8)

      .everyDays(1)

      .create();

  }

  // monthly AI prediction trigger

  var hasMonthly = triggers.some(t => t.getHandlerFunction()=='runAIPrediction' && t.getEventType()==ScriptApp.EventType.CLOCK);

  if (!hasMonthly) {

    ScriptApp.newTrigger('runAIPrediction')

      .timeBased()

      .onMonthDay(1)

      .atHour(6)

      .create();

  }

  SpreadsheetApp.getUi().alert('Initial setup completed. Please authorize the script when prompted.');

}

/\*\*

 \* checkInventoryLevels

 \* Runs daily. Checks Inventory Master for items below threshold, sends email,

 \* marks alert as sent, and logs in Reorder History.

 \*/

function checkInventoryLevels() {

  var ss = SpreadsheetApp.getActiveSpreadsheet();

  var sheet = ss.getSheetByName('Inventory Master');

  if (!sheet) return;

  var data = sheet.getDataRange().getValues();

  var header = data[0];

  var now = new Date();

  for (var i=1;i<data.length;i++){

    var row = data[i];

    var item = row[0];

    var quantity = parseFloat(row[1]);

    var threshold = parseFloat(row[2]);

    var alertSent = row[3];

    // Basic validation

    if (isNaN(quantity) || isNaN(threshold)) {

      // flag invalid row in a dedicated column (col 5)

      sheet.getRange(i+1,5).setValue('Invalid qty/threshold');

      continue;

    }

    if (quantity < 0) {

      sheet.getRange(i+1,5).setValue('Negative quantity!');

      continue;

    }

    if (quantity <= threshold && alertSent !== true) {

      sendReorderAlert(item, quantity, threshold);

      sheet.getRange(i+1,4).setValue(true); // mark alert sent

      logReorder(item, quantity, threshold, now);

    }

  }

}

/\*\*

 \* sendReorderAlert

 \* Sends a reorder request to vendor AND confirmation to owner

 \*/

function sendReorderAlert(item, quantity, threshold) {

  // 1. Vendor email address (update to real vendor address)

  var vendorEmail = 'sahilr541u@gmail.com';

  // 2. Owner/Manager email address

  var ownerEmail = 'sahilr541u@gmail.com';

  // --- Vendor email content ---

  var vendorSubject = 'Restock Request: ' + item;

  var vendorBody =

    'Dear Vendor,\n\n' +

    'Please arrange for a restock of the following product:\n\n' +

    'Item: ' + item + '\n' +

    'Current Stock: ' + quantity + '\n' +

    'Threshold: ' + threshold + '\n\n' +

    'Kindly confirm the expected delivery date.\n\n' +

    '--\nAutomated Supply Tracker';

  // --- Owner confirmation content ---

  var ownerSubject = 'Reorder Alert Sent to Vendor: ' + item;

  var ownerBody =

    'This is to confirm that a reorder request has been sent to the vendor for:\n\n' +

    'Item: ' + item + '\n' +

    'Current Stock: ' + quantity + '\n' +

    'Threshold: ' + threshold + '\n\n' +

    'The vendor has been notified to restock.\n\n' +

    '--\nAutomated Supply Tracker';

  // Send to vendor

  MailApp.sendEmail(vendorEmail, vendorSubject, vendorBody);

  // Send confirmation to owner

  MailApp.sendEmail(ownerEmail, ownerSubject, ownerBody);

}

/\*\*

 \* logReorder

 \* Appends a row to Reorder History with timestamp, item, qty, threshold, status

 \*/

function logReorder(item, qty, threshold, when) {

  var ss = SpreadsheetApp.getActiveSpreadsheet();

  var hist = ss.getSheetByName('Reorder History');

  if (!hist) {

    hist = ss.insertSheet('Reorder History');

  }

  hist.appendRow([when || new Date(), item, qty, threshold, 'Alert Sent']);

}

/\*\*

 \* applyUsageLog

 \* Processes new rows in Usage Log and updates Inventory Master quantities.

 \* Usage Log expected columns: Timestamp, Item, QuantityUsed, Notes

 \*/

function applyUsageLog() {

  var ss = SpreadsheetApp.getActiveSpreadsheet();

  var usage = ss.getSheetByName('Usage Log');

  var inv = ss.getSheetByName('Inventory Master');

  if (!usage || !inv) return;

  var udata = usage.getDataRange().getValues();

  var idata = inv.getDataRange().getValues();

  var invMap = {};

  for (var i=1;i<idata.length;i++){

    invMap[idata[i][0]] = {row:i+1, qty: parseFloat(idata[i][1])};

  }

  for (var j=1;j<udata.length;j++){

    var urow = udata[j];

    if (urow[4] === 'APPLIED') continue; // skip already applied logs (col E)

    var item = urow[1];

    var used = parseFloat(urow[2]);

    if (!invMap[item]) {

      usage.getRange(j+1,5).setValue('UNKNOWN\_ITEM');

      continue;

    }

    if (isNaN(used) || used < 0) {

      usage.getRange(j+1,5).setValue('INVALID\_QTY');

      continue;

    }

    var invRow = invMap[item].row;

    var newQty = invMap[item].qty - used;

    inv.getRange(invRow,2).setValue(newQty);

    usage.getRange(j+1,5).setValue('APPLIED');

  }

}

/\*\*

 \* runAIPrediction

 \* Placeholder for monthly AI-based prediction (Gemini).

 \* Writes sample predictions from inventory data and usage log to 'AI Predictions' sheet.

 \*/

function runAIPrediction() {

  var ss = SpreadsheetApp.getActiveSpreadsheet();

  var out = ss.getSheetByName('AI Predictions') || ss.insertSheet('AI Predictions');

  out.clear();

  out.appendRow(['GeneratedAt', 'PredictedAtRiskItem', 'Confidence', 'Notes']);

  var apiKey = PropertiesService.getScriptProperties().getProperty('AI\_API\_KEY');

  if (!apiKey) {

    out.appendRow([new Date(), 'ERROR', 'N/A', 'Missing AI\_API\_KEY in Script Properties']);

    return;

  }

  // Inventory data

  var invSheet = ss.getSheetByName('Inventory Master');

  var invData = invSheet ? invSheet.getDataRange().getValues() : [];

  var invRows = invData.slice(1).map(r => ({ item: r[0], quantity: r[1], threshold: r[2] }));

  // Usage data

  var usageSheet = ss.getSheetByName('Usage Log');

  var usageData = usageSheet ? usageSheet.getDataRange().getValues() : [];

  var usageRows = usageData.slice(1).map(r => ({ timestamp: r[0], item: r[1], quantityUsed: r[2] }));

  // Prompt

  var prompt = "Given the following inventory and usage log data, predict which items are at highest risk of stockout next month. " +

               "Return ONLY a JSON array of objects with fields: item, confidence (0-1), notes. No explanations, just JSON.\n\n" +

               "Inventory Data:\n" + JSON.stringify(invRows, null, 2) + "\n\n" +

               "Usage Data:\n" + JSON.stringify(usageRows, null, 2);

  try {

    var url = "https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-flash:generateContent?key=" + apiKey;

    var payload = {

      contents: [{ parts: [{ text: prompt }] }]

    };

    var response = UrlFetchApp.fetch(url, {

      method: 'post',

      contentType: 'application/json',

      payload: JSON.stringify(payload),

      muteHttpExceptions: true

    });

    if (response.getResponseCode() !== 200) {

      out.appendRow([new Date(), 'ERROR', 'N/A', 'API status ' + response.getResponseCode()]);

      return;

    }

    var json = JSON.parse(response.getContentText());

    var modelText = json.candidates && json.candidates[0].content.parts[0].text;

    if (!modelText) {

      out.appendRow([new Date(), 'ERROR', 'N/A', 'No prediction text returned']);

      return;

    }

    // --- JSON cleaner fallback ---

    var cleanedText = modelText.match(/\[([\s\S]\*)\]/);

    if (cleanedText) {

      modelText = "[" + cleanedText[1].trim() + "]";

    }

    var predictions;

    try {

      predictions = JSON.parse(modelText);

    } catch (e) {

      out.appendRow([new Date(), 'ERROR', 'N/A', 'Could not parse cleaned JSON']);

      return;

    }

    predictions.forEach(function(p) {

      out.appendRow([new Date(), p.item || '', p.confidence || '', p.notes || '']);

    });

  } catch (e) {

    out.appendRow([new Date(), 'ERROR', 'N/A', e.toString()]);

  }

}

/\*\*

 \* manualResetAlert

 \* Utility: manually reset the Alert Sent flag for an item (for testing)

 \*/

function manualResetAlert(itemName) {

  var ss = SpreadsheetApp.getActiveSpreadsheet();

  var sheet = ss.getSheetByName('Inventory Master');

  var data = sheet.getDataRange().getValues();

  for (var i=1;i<data.length;i++){

    if (data[i][0] === itemName) {

      sheet.getRange(i+1,4).setValue(false);

      return;

    }

  }

}