LAB MANUAL — SUPPLEMENT

Lab Test 12
Attendance Alert System

Workflow, agent, n8n and Apps Script — the same job four ways

---

Faculty Development Workshop

Symbiosis Institute of Technology, Pune

Facilitator: Mr. Dinesh Wadhwani

Day 3: 11 September 2026, afternoon

Duration: 90 minutes

Sample data: Attendance_Sample_Data.xlsx

Replaces: Lab Tests 9A and 9B

Lab Test 9 — The Attendance Alert System

Time: 90 minutes **Objective:** Build a system that reads attendance from a Google Sheet on a schedule, decides which parents to contact and what to say, and prepares the messages for your review. **Prerequisites:** Lab Tests 0 to 8A. Runs before the capstone.

This lab exists because a faculty member asked for it. It is the most directly useful thing in the workshop, and it is also where the distinction between a **workflow** and an **agent** stops being theoretical.

9.0 Read this before you build anything

The question this lab settles

You will build the same system twice.

Part A is a workflow. It reads the sheet, buckets students by attendance percentage, drafts a message for each one below the threshold, and writes it out for review. Every run does the same five things in the same order.

Part B is an agent. It gets tools and decides for itself which students need what — checking history where history matters, escalating instead of emailing where that is the right call.

Then you compare the two on the same cohort.

> **A SCHEDULE DOES NOT MAKE SOMETHING AN AGENT**
>
> A cron job that runs a backup at 2 a.m. is not an agent. Windows Update is not an agent.
>
> **When** something starts has nothing to do with **who chooses the steps**.
>
> Similarly, branching on a threshold — "below 75% use the concerned tone, below 60% use the urgent tone" — is an if-statement. You decided that mapping in advance, at design time. The model is filling in a template with better prose. That is genuinely useful, and it is not autonomy.
>
> Apply the rule from Chapter 1: *do the steps change depending on what I find?* In Part A, no. Read, compute, bucket, draft, write — the same five steps, every time.

Why Part B is a real agent, and why it is better

Part A treats every student at 74% identically. A real teacher does not.

- A student at 74% who was at **90% last month** is falling, and needs a note now

- A student at 74% who was at **58% last month** is recovering, and a warning email would be actively counterproductive
- A student flagged for the **third consecutive week** does not need a fourth parent email — they need the class mentor
- A student with a **recorded medical absence** should not be flagged at all

Handling that requires deciding **what to look up**. Do I need this student's history? Have we contacted this parent recently? Should this go to the parent at all?

That is a genuine runtime decision, and it is the difference between a threshold alarm and something a teacher would actually trust.

9.1 The rules for this lab

DUMMY DATA ONLY. NO EXCEPTIONS.

Nobody sends anything to a real parent from a lab machine today.

Your facilitator supplies `Attendance_Sample_Data.xlsx` in the shared workshop folder. Use it as it is.

- Every student and parent name in it is **invented**
- Every email address uses the `.invalid` domain, which by internet standard can never resolve to a real mailbox. Mail sent there cannot reach anyone.
- Do **not** paste your real class list, and do **not** replace those addresses with real ones

Attendance data about real, named students is personal data about minors. It does not go into a public sheet, a workshop exercise, or a free-tier AI service.

HOW YOU WILL TEST SENDING, SAFELY

You still get to send a real email and read it in a real inbox — that matters, and section 12.3 covers it.

Every version has a `testRecipientOverride` input. Set it to your own address and **every** message is redirected there, whatever the sheet says. The intended recipient is shown in the subject line so you can check the routing was right.

This is the standard pattern in production mail systems, usually called a catch-all or staging redirect. It exists so that a wrong row, a bad merge or a stale address cannot reach anyone. You are learning it here because you should use it every time you touch a mail system, not only today.

DRAFTS BY DEFAULT. SENDING IS A SWITCH, AND IT SHIPS OFF.

The failure mode here is not a red error node.

It is one hundred and eighty parents being told their child missed fourteen classes when they missed four. Or a row misalignment sending Parent A the record of Parent B's child.

You cannot unsend that, and it lands on families.

Every version in this lab has a `sendMode` input with three values:

Value	What happens
`review_sheet`	Writes messages to a dataset or sheet for you to read. **Default.**
`draft`	Creates Gmail drafts you open and send by hand
`send`	Sends. Only after you have read a full run in review mode.

The approval gate is not a nicety. It is the architecture.

9.2 Setup — 20 minutes

Step 1 — Upload the sample sheet

Take `Attendance_Sample_Data.xlsx` from the shared workshop folder.

1. Go to **https://drive.google.com** and upload it
2. Right-click the file → **Open with → Google Sheets**
3. It will save as a Google Sheet. Rename it `Attendance Demo - <your initials>`

Two tabs: **Sheet1** holds the data, **READ ME** explains what each row is testing. Read the second one.

The eight columns:

Column	Meaning
`roll_no`	Student roll number
`student_name`	Invented name
`parent_name`	Invented name
`parent_email`	Fake `.invalid` address. **Leave it alone.**
`classes_held`	Total classes conducted
`classes_attended`	Classes attended. **Blank for R015, on purpose.**
`previous_percentage`	Attendance a month ago — this is what makes Part B possible
`last_alert_date`	When a parent was last contacted, or blank

FOUR ROWS ARE SHADED, AND THEY ARE THE POINT OF THE LAB

R004 is falling fast. **R007** is recovering. **R011** was contacted five days ago. All three sit at a similar percentage today, so a threshold rule treats them identically.

R015 has a blank `classes_attended`. Your code must skip it and log why — code that reads blank as zero will tell a parent their child attended no classes at all.

Step 2 — Publish it as CSV

1. **File → Share → Publish to web**
2. Under the first dropdown, select the sheet tab
3. Under the second dropdown, select **Comma-separated values (.csv)**
4. Click **Publish**, confirm
5. Copy the URL. It looks like:

```
https://docs.google.com/spreadsheets/d/e/2PACX-1vT.../pub?gid=0&single=true&output=csv
```

1. Open that URL in a new tab. You should see raw CSV text.

PUBLISHING TO WEB MAKES THE SHEET PUBLICLY READABLE

Anyone with that URL can read it. That is why this lab uses invented data.

For real deployment you use a **Google service account** with the sheet shared privately to it, or the Apps Script version in 9.5 which reads the sheet natively with no publishing at all. Section 9.6 covers this.

Never publish a sheet containing real student data.

Step 3 — Create a Gmail app password

Google finished shutting off password-only SMTP access on 1 May 2025. Every SMTP connection to Gmail now requires either OAuth or an app password.

1. Go to **https://myaccount.google.com/security**
2. Confirm **2-Step Verification** is **ON**. App passwords do not exist without it.
3. Go to **https://myaccount.google.com/apppasswords**
4. Enter a name: `Apify Attendance Actor`
5. Click **Create**
6. Copy the 16-character code. Google will not show it again. You may remove the spaces.

THREE THINGS ABOUT APP PASSWORDS

Personal Gmail only. Institutional Workspace accounts often have app passwords disabled by the administrator. If `myaccount.google.com/apppasswords` shows nothing, that is why — use a personal Gmail.

Google calls them "not recommended." Anyone holding the 16-character code has full SMTP access to your account and it cannot be scoped. Treat it like your password, and **revoke it at the end of Day 3.**

Sending limits. Personal Gmail caps at roughly 500 recipients per day. Fine for one class. Not enough for a department.

9.3 PART A — The workflow version (Apify)

Time: 30 minutes

Step 4 — New Actor

Develop new → Empty Node.js project. Name it `<initials>-lab9-attendance`.

Step 5 — Add three secret environment variables

Settings → Environment variables. All three ticked as **Secret**:

Name	Value
`GEMINI_API_KEY`	Your Gemini key
`GMAIL_USER`	Your personal Gmail address
`GMAIL_APP_PASSWORD`	The 16-character app password

ADD ALL THREE, THEN BUILD

Environment variables are applied at build time. Add them first.

Step 6 — Add the email dependency

Open `package.json` and add `nodemailer` to the dependencies block:

```
{
  "name": "attendance-alert",
  "version": "0.0.1",
  "type": "module",
  "dependencies": {
    "apify": "^3.2.0",
    "nodemailer": "^6.9.14"
  },
  "scripts": {
    "start": "node src/main.js"
  }
}
```

Keep whatever version numbers the template gave you for `apify`. Only add the `nodemailer` line.

Step 7 — Replace `.actor/input_schema.json`

```
{
  "title": "Attendance Alert System",
  "type": "object",
  "schemaVersion": 1,
  "properties": {
    "sheetCsvUrl": {
      "title": "Published Google Sheet CSV URL",
      "type": "string",
      "editor": "textfield",
      "description": "File > Share > Publish to web > CSV. Dummy data only."
    },
    "thresholdPercent": {
      "title": "Attendance threshold (%)",
      "type": "integer",
      "editor": "number",
      "description": "Students below this are flagged.",
      "default": 75
    },
    "testRecipientOverride": {
      "title": "Redirect ALL email to this address (safety)",
      "type": "string",
      "editor": "textfield",
      "description": "Strongly recommended. Every message goes here instead of the real
recipient. Leave set during the workshop.",
      "prefill": "your.address@gmail.com"
    },
    "allowLiveSend": {
      "title": "I understand this will email the real addresses in the sheet",
      "type": "boolean",
      "description": "Only tick this once the override is cleared AND you have read a full
review run.",
      "default": false
    },
    "sendMode": {
      "title": "What to do with the messages",
      "type": "string",
      "editor": "select",
      "enum": ["review_sheet", "send"],
      "enumTitles": ["Write to dataset for review (safe)", "Actually send the emails"],
      "description": "Leave on review until you have read a full run.",
      "default": "review_sheet"
    },
    "courseName": {
      "title": "Course name",
      "type": "string",
      "editor": "textfield",
      "default": "Database Management Systems (CS204)"
    },
    "teacherName": {
      "title": "Your name, for the signature",
      "type": "string",
      "editor": "textfield",
      "default": "Prof. A. Sharma"
    },
    "language": {
      "title": "Message language",
      "type": "string",
      "editor": "select",
```

```
      "enum": ["English", "Marathi", "Hindi", "English and Marathi"],
      "default": "English"
    },
    "promptTemplate": {
      "title": "Message prompt",
      "type": "string",
      "editor": "textarea",
      "prefill": "You are a college professor writing to the parent of a student about
attendance.\n\nSTUDENT: {{student_name}} (Roll {{roll_no}})\nPARENT: {{parent_name}}
\nCOURSE: {{course}}\nATTENDANCE: {{attended}} of {{held}} classes = {{percentage}}
percent\nTHRESHOLD: {{threshold}} percent\nSEVERITY: {{severity}}\n\nWrite a short email
body in {{language}}.\n\nRules:\n- Address the parent respectfully by name\n- State the
attendance figures plainly and accurately. Do NOT round, exaggerate, or add figures that
were not given to you.\n- Explain briefly why attendance matters for this course\n- Invite
them to reply or meet, do not demand\n- Do NOT threaten, and do NOT mention marks, grades or
examination eligibility - you have not been given that information\n- 5 to 7 sentences\n-
Sign as {{teacher}}\n\nWrite only the email body. No subject line. No preamble."
    },
    "model": {
      "title": "Gemini model",
      "type": "string",
      "editor": "textfield",
      "default": "gemini-2.5-flash"
    },
    "delayMs": {
      "title": "Delay between AI calls (ms)",
      "type": "integer",
      "editor": "number",
      "default": 4500
    }
  },
  "required": ["sheetCsvUrl"]
}
```

LOOK AT WHAT IS IN THE SCHEMA

Threshold, language, prompt, send mode, course name, your name — everything a teacher would want to change is an input.

Deploy this once and a colleague can use it for their own course by filling in a form. They never open the code.

Step 8 — Replace `src/main.js`

Complete file:

```
import { Actor } from 'apify';
import nodemailer from 'nodemailer';

await Actor.init();

const input = await Actor.getInput() ?? {};
const sheetCsvUrl      = input.sheetCsvUrl ?? '';
const thresholdPercent = input.thresholdPercent ?? 75;
const sendMode         = input.sendMode ?? 'review_sheet';
const overrideTo       = (input.testRecipientOverride ?? '').trim();
const allowLiveSend    = input.allowLiveSend === true;
const courseName       = input.courseName ?? 'this course';
const teacherName      = input.teacherName ?? 'Your instructor';
const language         = input.language ?? 'English';
const promptTemplate   = input.promptTemplate ?? '';
const model            = input.model ?? 'gemini-2.5-flash';
const delayMs          = input.delayMs ?? 4500;

const API_KEY  = process.env.GEMINI_API_KEY;
const GM_USER  = process.env.GMAIL_USER;
const GM_PASS  = process.env.GMAIL_APP_PASSWORD;

if (!API_KEY) throw new Error('GEMINI_API_KEY not set. Add it in Settings, then REBUILD.');
if (sendMode === 'send' && (!GM_USER || !GM_PASS)) {
    throw new Error('sendMode is "send" but GMAIL_USER / GMAIL_APP_PASSWORD are not set.');
}

// ---------------------------------------------------------------
//  THE SAFETY GATE
//  Live sending needs BOTH the override cleared AND an explicit
//  acknowledgement. One of them alone is not enough.
// ---------------------------------------------------------------
if (sendMode === 'send' && !overrideTo && !allowLiveSend) {
    throw new Error(
        'Refusing to send. testRecipientOverride is empty and allowLiveSend is false.\n' +
        'Either set testRecipientOverride to your own address (recommended), or tick ' +
        'allowLiveSend to confirm you intend to email the real addresses in the sheet.'
    );
}
if (overrideTo) {
    console.log(`OVERRIDE ACTIVE - every message will go to ${overrideTo}, not the sheet
addresses.`);
}

function sleep(ms) { return new Promise((r) => setTimeout(r, ms)); }

// ---------------------------------------------------------------
//  CSV PARSING
//  Handles quoted fields containing commas.
// ---------------------------------------------------------------
function parseCsv(text) {
    const rows = [];
    let row = [], field = '', inQuotes = false;

    for (let i = 0; i < text.length; i++) {
        const c = text[i];
        if (inQuotes) {
            if (c === '"' && text[i + 1] === '"') { field += '"'; i++; }
```

```
            else if (c === '"') { inQuotes = false; }
            else { field += c; }
        } else if (c === '"') {
            inQuotes = true;
        } else if (c === ',') {
            row.push(field); field = '';
        } else if (c === '\n') {
            row.push(field); rows.push(row); row = []; field = '';
        } else if (c !== '\r') {
            field += c;
        }
    }
    if (field.length > 0 || row.length > 0) { row.push(field); rows.push(row); }

    const header = rows.shift().map((h) => h.trim());
    return rows
        .filter((r) => r.some((cell) => cell.trim() !== ''))
        .map((r) => {
            const obj = {};
            header.forEach((h, i) => { obj[h] = (r[i] ?? '').trim(); });
            return obj;
        });
}

// ---------------------------------------------------------------
//  1. READ THE SHEET
// ---------------------------------------------------------------
console.log('Fetching attendance sheet...');
const csvResponse = await fetch(sheetCsvUrl);
if (!csvResponse.ok) {
    throw new Error(
        `Could not fetch the sheet: HTTP ${csvResponse.status}. ` +
        'Check that the sheet is published to web as CSV and the URL is correct.'
    );
}
const students = parseCsv(await csvResponse.text());
console.log(`Read ${students.length} student records.`);

if (students.length === 0) {
    throw new Error('The sheet returned no rows. Check the published URL opens as CSV in a
browser.');
}

// ---------------------------------------------------------------
//  2. COMPUTE AND BUCKET
//  Deterministic. No model involved. This must be exact.
// ---------------------------------------------------------------
const flagged = [];
const skipped = [];

for (const s of students) {
    const held     = Number(s.classes_held);
    const attended = Number(s.classes_attended);

    if (!held || Number.isNaN(attended)) {
        skipped.push({ ...s, reason: 'classes_held or classes_attended missing or not a
number' });
        continue;
    }
```

```
    if (!s.parent_email || !s.parent_email.includes('@')) {
        skipped.push({ ...s, reason: 'parent_email missing or malformed' });
        continue;
    }

    const percentage = Number(((attended / held) * 100).toFixed(2));

    if (percentage >= thresholdPercent) continue;

    let severity;
    if      (percentage < 50) severity = 'critical';
    else if (percentage < 65) severity = 'serious';
    else                      severity = 'early warning';

    flagged.push({ ...s, held, attended, percentage, severity });
}

console.log(`${flagged.length} student(s) below ${thresholdPercent}%. ${skipped.length}
skipped.`);
for (const s of skipped) console.log(`  SKIPPED ${s.roll_no}: ${s.reason}`);

// ------------------------------------------------------------
//  3. DRAFT THE MESSAGES
// ------------------------------------------------------------
function fillTemplate(template, values) {
    return template.replace(/\{\{\s*(\w+)\s*\}\}/g, (m, k) =>
        values[k] !== undefined ? String(values[k]) : m);
}

async function callGemini(prompt) {
    const url = `https://generativelanguage.googleapis.com/v1beta/models/$
{model}:generateContent?key=${API_KEY}`;
    const response = await fetch(url, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            contents: [{ parts: [{ text: prompt }] }],
            generationConfig: { temperature: 0.5, maxOutputTokens: 1024 }
        })
    });
    if (!response.ok) {
        const body = await response.text();
        throw new Error(`Gemini API ${response.status}: ${body.slice(0, 300)}`);
    }
    const data = await response.json();
    const text = data?.candidates?.[0]?.content?.parts?.[0]?.text;
    if (!text) throw new Error('No text in Gemini response.');
    return text.trim();
}

const messages = [];

for (let i = 0; i < flagged.length; i++) {
    const s = flagged[i];
    console.log(`[${i + 1}/${flagged.length}] Drafting for ${s.student_name} ($
{s.percentage}%, ${s.severity})`);

    const prompt = fillTemplate(promptTemplate, {
        student_name: s.student_name,
```

```
        roll_no:      s.roll_no,
        parent_name:  s.parent_name,
        course:       courseName,
        attended:     s.attended,
        held:         s.held,
        percentage:   s.percentage,
        threshold:    thresholdPercent,
        severity:     s.severity,
        teacher:      teacherName,
        language:     language
    });

    try {
        const body = await callGemini(prompt);

        // A cheap but important check: does the drafted message contain
        // the real attendance figures? If not, the model has invented
        // something and this draft must not go out unreviewed.
        const mentionsFigures =
            body.includes(String(s.attended)) || body.includes(String(s.percentage));

        messages.push({
            roll_no:       s.roll_no,
            student_name:  s.student_name,
            parent_name:   s.parent_name,
            parent_email:  s.parent_email,
            percentage:    s.percentage,
            attended:      s.attended,
            held:          s.held,
            severity:      s.severity,
            subject:       `Attendance update - ${s.student_name} - ${courseName}`,
            body,
            figures_check: mentionsFigures ? 'ok' : 'REVIEW - figures may be missing or
altered',
            status:        'drafted',
            error:         null
        });
    } catch (err) {
        console.log(`  FAILED: ${err.message}`);
        messages.push({
            roll_no: s.roll_no, student_name: s.student_name,
            status: 'failed', error: err.message
        });
    }

    if (i < flagged.length - 1) await sleep(delayMs);
}

// ------------------------------------------------------------
//  4. REVIEW OR SEND
// ------------------------------------------------------------
if (sendMode === 'send') {
    console.log('\n*** SEND MODE - emails will actually go out ***');

    const transporter = nodemailer.createTransport({
        host: 'smtp.gmail.com',
        port: 465,
        secure: true,
        auth: { user: GM_USER, pass: GM_PASS }
```

```
    });

    for (const m of messages) {
        if (m.status !== 'drafted') continue;

        // The override wins over whatever is in the sheet.
        const actualTo = overrideTo || m.parent_email;
        const subject  = overrideTo
            ? `[TEST - intended for ${m.parent_email}] ${m.subject}`
            : m.subject;

        try {
            await transporter.sendMail({
                from: `"${teacherName}" <${GM_USER}>`,
                to: actualTo,
                subject,
                text: m.body
            });
            m.status       = 'sent';
            m.delivered_to = actualTo;
            m.redirected   = Boolean(overrideTo);
            console.log(`  sent to ${actualTo} re ${m.student_name}`
                        + (overrideTo ? ' [REDIRECTED]' : ''));
        } catch (err) {
            m.status = 'send_failed';
            m.error = err.message;
            console.log(`  SEND FAILED for ${m.student_name}: ${err.message}`);
        }
        await sleep(1500);
    }
} else {
    console.log('\nReview mode. Nothing was sent. Read the dataset, then set sendMode to
"send".');
}

await Actor.pushData(messages);

console.log('\n=== SUMMARY ===');
console.log(`Students read      : ${students.length}`);
console.log(`Skipped (bad data) : ${skipped.length}`);
console.log(`Flagged below ${thresholdPercent}% : ${flagged.length}`);
console.log(`Messages drafted   : ${messages.filter((m) => m.status !== 'failed').length}`);
console.log(`Mode               : ${sendMode}`);

await Actor.exit();
```

Step 9 — Build and run in review mode

Build. Set **Memory 512 MB**. On the Input tab paste your published CSV URL. Leave `sendMode` on **review_sheet**. Start.

Step 10 — Read every draft

Open the **Dataset** tab and read all of them properly. Check specifically:

- Does each message state the **correct** attendance figures?

- Any message mentioning marks, grades or exam eligibility? The prompt forbids it — did it obey?
- Any message that threatens, rather than invites?
- Is `figures_check` reading `ok` for every row?

> **THIS STEP IS THE ENTIRE POINT OF REVIEW MODE**
>
> You are looking for a message that reads perfectly and states a number that is not in your data.
>
> That is the failure this whole design exists to catch. It will not look like an error. It will look like a good email.

Step 11 — Deliberately break it

On the Input tab, edit the prompt and remove this line:

```
- State the attendance figures plainly and accurately. Do NOT round, exaggerate, or add figures that were not given to you.
```

Re-run. **No rebuild** — it is an input.

Now read the drafts again. You will typically see rounded figures, invented context ("has missed several important sessions"), or claims about exam eligibility that you never supplied.

Put the line back.

Step 12 — One real send, redirected to yourself

The sheet's addresses all end in `.invalid` and can never deliver. To see a real email you use the override.

1. On the Input tab, set **testRecipientOverride** to your own Gmail address
2. Set **sendMode** to `send`
3. Start

Every message lands in **your** inbox, with a subject reading:

```
[TEST - intended for parent.r004@example.invalid] Attendance update - Deepa Joshi - ...
```

Read one as a parent would, and check the bracketed address — that is how you verify the routing was correct without anything leaving your own mailbox.

DO THIS ONCE, DELIBERATELY

Reading your own output in an actual inbox, in the format a parent will see it, changes how you judge the tone. It reads differently there than in a dataset table.

Step 12b — Try to send without the safety net

Clear **testRecipientOverride**, leave **allowLiveSend** unticked, keep `sendMode` on `send`, and Start.

The Actor refuses and tells you why.

WHY THE GATE NEEDS TWO CONDITIONS, NOT ONE

A single checkbox gets ticked absent-mindedly. A single override gets cleared by someone copying a config.

Requiring the override to be empty **and** a separate acknowledgement means live sending is never one careless click away. This is worth carrying into anything you build that can reach people.

Step 13 — Schedule it

On the Actor page, go to the **Schedules** tab → **Create schedule** → set a cron expression, for example every Friday at 5 p.m.:

```
0 17 * * 5
```

THIS IS WHY THE ACTOR VERSION MATTERS

Apify runs your Actor on its own servers. The schedule fires whether or not your laptop is on.

Compare with n8n in Part C: n8n's Schedule Trigger only fires **while n8n is running**, which on a lab machine means only while that Command Prompt window is open. Friday 5 p.m. arrives, the PC is off, nothing happens.

That is the single most important practical difference between the two platforms for this use case.

PART A CHECKPOINT

- [ ] Sheet published as CSV, URL opens as raw CSV in a browser
- [ ] Three environment variables added as secrets, Actor rebuilt after
- [ ] Ran in review mode and read every draft
- [ ] Verified figures against the sheet by hand for at least two students
- [ ] Removed the accuracy line, re-ran, and saw the drafts degrade
- [ ] Sent one real email to myself and read it as a parent
- [ ] Schedule created
- [ ] Dataset exported

9.4 PART B — Now make it an actual agent

Time: 30 minutes

Step 14 — What Part A cannot do

Look at your sheet. Three students are at similar percentages but their situations are not similar:

Roll	Now	A month ago	Last alert	What a teacher would do
R004	72%	91%	never	**Falling fast.** Contact now.
R007	71%	55%	3 weeks ago	**Recovering.** A warning would be counterproductive.
R011	68%	69%	5 days ago	**Third alert.** Stop emailing. Escalate to the mentor.

Part A sent all three the same kind of message. It could not do otherwise, because you never told it to look at history — and if you had, you would have had to decide in advance exactly when history matters.

An agent decides that at runtime.

Step 15 — Duplicate your Lab 6A agent

Name it `<initials>-lab9-attendance-agent`. Add `GEMINI_API_KEY` as a secret.

THE LOOP DOES NOT CHANGE. AGAIN.

This is the third agent you have built on the same sixty lines.

Only the tools and the instructions change.

Step 16 — Replace the tools block

```
// ========================================================================
//  THE TOOLS
// ========================================================================

const SHEET_CSV_URL = process.env.SHEET_CSV_URL ?? '';

// Same parser as Part A
function parseCsv(text) {
    const rows = [];
    let row = [], field = '', inQuotes = false;
    for (let i = 0; i < text.length; i++) {
        const c = text[i];
        if (inQuotes) {
            if (c === '"' && text[i + 1] === '"') { field += '"'; i++; }
            else if (c === '"') { inQuotes = false; }
            else { field += c; }
        } else if (c === '"') { inQuotes = true; }
        else if (c === ',') { row.push(field); field = ''; }
        else if (c === '\n') { row.push(field); rows.push(row); row = []; field = ''; }
        else if (c !== '\r') { field += c; }
    }
    if (field.length > 0 || row.length > 0) { row.push(field); rows.push(row); }
    const header = rows.shift().map((h) => h.trim());
    return rows.filter((r) => r.some((c) => c.trim() !== '')).map((r) => {
        const o = {}; header.forEach((h, i) => { o[h] = (r[i] ?? '').trim(); }); return o;
    });
}

let CACHE = null;
async function loadSheet() {
    if (CACHE) return CACHE;
    const res = await fetch(SHEET_CSV_URL);
    if (!res.ok) throw new Error(`Sheet fetch failed: HTTP ${res.status}`);
    CACHE = parseCsv(await res.text());
    return CACHE;
}

function pct(row) {
    const held = Number(row.classes_held);
    const att  = Number(row.classes_attended);
    if (!held || Number.isNaN(att)) return null;
    return Number(((att / held) * 100).toFixed(2));
}

const tools = {
    get_attendance_summary: {
        description:
            'Returns every student with their current attendance percentage, ' +
            'calculated exactly. Takes no arguments. ' +
            'Call this FIRST. Do not calculate percentages yourself.',
        run: async () => {
            const rows = await loadSheet();
            return rows.map((r) => ({
                roll_no: r.roll_no,
                student_name: r.student_name,
                classes_held: Number(r.classes_held),
                classes_attended: Number(r.classes_attended),
                percentage: pct(r)
```

```
            }));
        }
    },

    get_student_history: {
        description:
            'Returns the previous attendance percentage and the date a parent was last ' +
            'contacted, for specific students. ' +
            'args: { "roll_numbers": ["R004","R007"] }. ' +
            'Use this to tell a falling student from a recovering one, and to avoid ' +
            'contacting a parent who was contacted recently.',
        run: async (args) => {
            const wanted = args.roll_numbers;
            if (!Array.isArray(wanted)) {
                return { error: 'roll_numbers must be an array, e.g. ["R004"]' };
            }
            const rows = await loadSheet();
            return rows
                .filter((r) => wanted.includes(r.roll_no))
                .map((r) => ({
                    roll_no: r.roll_no,
                    student_name: r.student_name,
                    current_percentage: pct(r),
                    previous_percentage: r.previous_percentage
                        ? Number(r.previous_percentage) : null,
                    change: r.previous_percentage
                        ? Number((pct(r) - Number(r.previous_percentage)).toFixed(2)) :
null,
                    last_alert_date: r.last_alert_date || 'never'
                }));
        }
    },

    days_since: {
        description:
            'Returns the number of days between a date and today. ' +
            'args: { "date": "2026-08-18" }. ' +
            'Use this for anything involving how long ago something happened. ' +
            'NEVER work out date differences yourself.',
        run: async (args) => {
            const d = new Date(args.date);
            if (Number.isNaN(d.getTime())) {
                return { error: `Could not read "${args.date}" as a date. Use YYYY-MM-
DD.` };
            }
            const days = Math.floor((Date.now() - d.getTime()) / 86400000);
            return { date: args.date, days_ago: days };
        }
    },

    record_decision: {
        description:
            'Records what you have decided for one student. Call once per student ' +
            'you have made a decision about. ' +
            'args: { "roll_no": "R004", "action": "email_parent" | "escalate_to_mentor" |
"no_contact", ' +
            '"severity": "critical" | "serious" | "early warning" | "none", ' +
            '"reason": "one sentence explaining why, referencing the data you saw" }',
        run: async (args) => {
```

```
            if (!args.roll_no || !args.action || !args.reason) {
                return { error: 'roll_no, action and reason are all required.' };
            }
            const valid = ['email_parent', 'escalate_to_mentor', 'no_contact'];
            if (!valid.includes(args.action)) {
                return { error: `action must be one of: ${valid.join(', ')}` };
            }
            DECISIONS.push({
                roll_no: args.roll_no,
                action: args.action,
                severity: args.severity ?? 'none',
                reason: args.reason
            });
            return { recorded: true, total_recorded: DECISIONS.length };
        }
    }
};

const DECISIONS = [];
```

Step 17 — Add the decisions to the output

At the bottom of the file, change the `Actor.pushData` call to:

```
await Actor.pushData([{
    question,
    answer: finalAnswer,
    iterations_used: trace.length,
    decisions: DECISIONS,
    trace
}]);
```

Step 18 — Add the sheet URL as an environment variable

Settings → Environment variables → `SHEET_CSV_URL` → your published CSV URL. Not secret — it is not a credential, though it is a public link.

Rebuild.

Step 19 — Set the agent instructions (Input tab, no rebuild)

```
You are assisting a college professor with attendance follow-up
for Database Management Systems (CS204).

YOUR JOB:
Decide, for each student below the attendance threshold, whether to
contact the parent, escalate to the class mentor, or do nothing.
Record every decision with record_decision.

RULES:
1. Never calculate percentages or date differences yourself. Use
   get_attendance_summary and days_since.
2. A student below threshold is not automatically a parent email.
   Look at direction of travel and recent contact history before
   deciding.
3. If a student is IMPROVING compared to their previous percentage,
   a warning email is counterproductive. Prefer no_contact and say
   so in your reason.
4. If a parent was contacted within the last 14 days, do NOT email
   again. Escalate to the mentor instead.
5. If a student has been below threshold and contacted twice or more
   already, escalate to the mentor rather than sending a third email.
6. Every decision must cite the actual figures you saw. Do not
   record a decision you cannot justify from tool output.
7. Do not invent students, figures or dates. If the data is missing
   or malformed, record no_contact and say the data was unusable.

EFFICIENCY:
Call get_student_history ONCE with all the roll numbers you need,
not once per student.

WHEN TO STOP:
Once you have recorded a decision for every student below threshold,
give a short summary and stop. Do not re-verify decisions already
recorded.
```

Set **question** to:

```
Review this week's attendance for CS204. The threshold is 75 percent.
Today's date is 11 September 2026. Decide what to do about each
student below threshold and record your decisions.
```

Set **maxIterations** to **8** for this one — there are more tools and more students.

Step 20 — Run it and read the trace

Watch the log. A good run looks roughly like:

```
--- Iteration 1 ---
  thought: I need the current figures first.
  calling: get_attendance_summary({})
--- Iteration 2 ---
  thought: Five students are below 75%. I need their history to see
           who is falling and who is recovering.
  calling: get_student_history({"roll_numbers":["R004","R007","R011",...]})
--- Iteration 3 ---
  thought: R011 was contacted 5 days ago. I need to confirm how recent.
  calling: days_since({"date":"2026-09-06"})
--- Iteration 4 ---
  thought: R004 fell from 91 to 72. That is a sharp drop, no prior
           contact. Email the parent.
  calling: record_decision({"roll_no":"R004","action":"email_parent",...})
```

Step 21 — Check the three cases

Look at the recorded decisions for R004, R007 and R011.

Roll	Expected decision	Because
R004	`email_parent`	Fell 91 → 72, never contacted
R007	`no_contact`	Rose 55 → 71, improving
R011	`escalate_to_mentor`	Contacted 5 days ago, repeat case

THIS IS THE MOMENT THE LAB EXISTS FOR

Part A treated all three identically because it could not do otherwise.

Part B looked at each one, decided what it needed to know, went and got it, and reached three different conclusions — with a reason for each that cites the actual numbers.

That is the difference between a schedule with an if-statement and an agent.

IF IT GOT THEM WRONG

Read the trace and find where. Usually one of three things:

- It never called `get_student_history` → strengthen Rule 2
- It called history one student at a time and ran out of iterations → the efficiency instruction is being ignored, restate it more firmly
- It emailed R007 anyway → Rule 3 needs to be more explicit that improving means no contact

Fix the instructions on the **Input tab** and re-run. No rebuild.

Step 22 — Try to make it invent something

Change **question** to:

```
What is the attendance for roll number R999?
```

It must say there is no such student. **It must not produce a figure.**

PART B CHECKPOINT

- [ ] Agent called get_attendance_summary before anything else
- [ ] Agent fetched history for multiple students in ONE call
- [ ] R004 → email_parent, R007 → no_contact, R011 → escalate_to_mentor
- [ ] Every decision cites real figures in its reason
- [ ] Agent refuses to invent data for R999
- [ ] I read the full trace and can explain each decision
- [ ] Dataset exported

9.5 PART C — The n8n version

Time: 20 minutes **Track B only — if the machines are ready.**

Step 23 — Build the workflow

`<initials> - Lab 9C - Attendance`

Node	Configuration
Schedule Trigger	Interval: Weeks. Day: Friday. Hour: 17.
HTTP Request	GET, URL = your published CSV URL, Response Format = **Text**
Code (JavaScript)	Parse the CSV and filter — code below
Google Gemini	Prompt with `{{ $json.student_name }}` etc.
Send Email	SMTP credential using your app password

Step 24 — The Code node

Mode: **Run Once for All Items**.

```
const csv = $input.first().json.data;
const lines = csv.split('\n').filter(l => l.trim() !== '');
const header = lines.shift().split(',').map(h => h.trim());

const threshold = 75;
const results = [];

for (const line of lines) {
  const cells = line.split(',').map(c => c.trim());
  const row = {};
  header.forEach((h, i) => { row[h] = cells[i] ?? ''; });

  const held = Number(row.classes_held);
  const att  = Number(row.classes_attended);
  if (!held || Number.isNaN(att)) continue;
  if (!row.parent_email || !row.parent_email.includes('@')) continue;

  const percentage = Number(((att / held) * 100).toFixed(2));
  if (percentage >= threshold) continue;

  results.push({
    json: {
      roll_no: row.roll_no,
      student_name: row.student_name,
      parent_name: row.parent_name,
      parent_email: row.parent_email,
      held, attended: att, percentage,
      severity: percentage < 50 ? 'critical'
              : percentage < 65 ? 'serious'
              : 'early warning'
    }
  });
}

return results;
```

THIS SIMPLE PARSER BREAKS ON COMMAS INSIDE QUOTES

Fine for the demo data. Not fine for a real sheet where a name might contain a comma.

The Actor version in Part A has a proper parser. Use that one for anything real.

Step 25 — The SMTP credential

Credentials → Add credential → SMTP

Field	Value
User	your personal Gmail address
Password	your 16-character app password
Host	`smtp.gmail.com`
Port	`465`
SSL/TLS	on

Step 25b — Apply the same override in n8n

n8n has no built-in catch-all, so add one in the Code node. Immediately before `results.push`, insert:

```
// SAFETY: redirect everything to one address during testing.
// Set to '' only when you intend to mail the real recipients.
const OVERRIDE_TO = 'your.address@gmail.com';
```

and change the pushed object's recipient to:

```
    parent_email: OVERRIDE_TO || row.parent_email,
    intended_for: row.parent_email,
```

Then put `{{ $json.intended_for }}` into the Send Email node's subject line so you can verify routing.

DO THIS BEFORE YOU RUN IT EVEN ONCE

The `.invalid` addresses in the sample sheet cannot deliver, so a run without the override simply bounces. That is safe today.

The habit matters for the day you swap in a real sheet, when a run without an override is not safe at all.

Step 26 — The critical limitation

THE N8N SCHEDULE ONLY FIRES WHILE N8N IS RUNNING

n8n runs inside that Command Prompt window. Close it, or turn the machine off, and the Schedule Trigger never fires.

Friday 5 p.m. arrives, the lab PC is off, no emails go out, and nobody knows until a parent asks why they were not told.

For anything that must run on a schedule, n8n needs an always-on host — a college server, or n8n Cloud. On a desktop it is a manual-run tool.

This is why Part A used Apify's own scheduling, and why Part D exists.

Step 27 — Compare

PART C CHECKPOINT

- [ ] n8n workflow produces the same flagged list as the Actor
- [ ] I understand why its schedule will not fire on a lab machine
- [ ] I can state which of the three versions I would actually deploy

---

9.6 PART D — Google Apps Script, the version you will actually deploy

Time: 10 minutes

Honest advice: for this specific task, **Apps Script is the best tool**, and you should know that even though it is neither of the platforms we have been teaching.

	Apify Actor	n8n	Apps Script
Runs when your PC is off	Yes	**No**	Yes
Needs the sheet published publicly	Yes (or a service account)	Yes	**No**
Needs an app password	Yes	Yes	**No**
Data leaves Google	Yes	Yes	**No**
Setup time	30 min	20 min	**5 min**
Can use an AI agent loop	**Yes**	Limited	Possible, awkward

Use Apify when the deciding is genuinely hard — Part B's agent. Use Apps Script when the job is read a sheet and send mail, which is most weeks.

Step 28 — Open the script editor

In your attendance sheet: **Extensions → Apps Script**. Delete the placeholder code.

Step 29 — Paste this complete script

```
// ====================================================================
//  ATTENDANCE ALERT - Google Apps Script
//  Runs on Google's servers. Reads this sheet directly.
//  Sends from your own Gmail. No app password, no publishing.
// ====================================================================

const CONFIG = {
  SHEET_NAME:   'Sheet1',
  THRESHOLD:    75,
  COURSE:       'Database Management Systems (CS204)',
  TEACHER:      'Prof. A. Sharma',
  LANGUAGE:     'English',
  DRAFT_ONLY:   true,        // <<<< leave true until you have read a full run
  RECIPIENT_OVERRIDE: '',    // <<<< set to YOUR address: every mail is redirected here
  ALLOW_LIVE_SEND:    false, // <<<< must be true to mail the real addresses
  GEMINI_KEY:   '',          // paste your key, or leave blank for template mode
  GEMINI_MODEL: 'gemini-2.5-flash'
};

function runAttendanceAlerts() {
  // ---- SAFETY GATE ----
  // Live sending needs the override cleared AND an explicit acknowledgement.
  if (!CONFIG.DRAFT_ONLY && !CONFIG.RECIPIENT_OVERRIDE && !CONFIG.ALLOW_LIVE_SEND) {
    throw new Error(
      'Refusing to send. RECIPIENT_OVERRIDE is empty and ALLOW_LIVE_SEND is false. ' +
      'Set RECIPIENT_OVERRIDE to your own address, or set ALLOW_LIVE_SEND to true ' +
      'to confirm you intend to email the real addresses in the sheet.');
  }
  if (CONFIG.RECIPIENT_OVERRIDE) {
    Logger.log('OVERRIDE ACTIVE - all mail goes to ' + CONFIG.RECIPIENT_OVERRIDE);
  }

  const sheet = SpreadsheetApp.getActive().getSheetByName(CONFIG.SHEET_NAME);
  const data  = sheet.getDataRange().getValues();
  const head  = data.shift().map(String);

  const col = {};
  head.forEach((h, i) => { col[h.trim()] = i; });

  const required = ['roll_no','student_name','parent_name','parent_email',
                    'classes_held','classes_attended'];
  for (const r of required) {
    if (col[r] === undefined) throw new Error('Missing column: ' + r);
  }

  let drafted = 0, skipped = 0;

  data.forEach(function (row) {
    const held = Number(row[col.classes_held]);
    const att  = Number(row[col.classes_attended]);
    const email = String(row[col.parent_email] || '');

    if (!held || isNaN(att) || email.indexOf('@') === -1) {
      skipped++;
      Logger.log('Skipped ' + row[col.roll_no] + ' - bad data');
      return;
    }
```

```
    const percentage = Math.round((att / held) * 10000) / 100;
    if (percentage >= CONFIG.THRESHOLD) return;

    const severity = percentage < 50 ? 'critical'
                   : percentage < 65 ? 'serious'
                   : 'early warning';

    const details = {
      student:    String(row[col.student_name]),
      roll:       String(row[col.roll_no]),
      parent:     String(row[col.parent_name]),
      attended:   att,
      held:       held,
      percentage: percentage,
      severity:   severity
    };

    const body = CONFIG.GEMINI_KEY
      ? draftWithGemini(details)
      : draftFromTemplate(details);

    // The override wins over whatever is in the sheet.
    const actualTo = CONFIG.RECIPIENT_OVERRIDE || email;
    const subject  = (CONFIG.RECIPIENT_OVERRIDE
                      ? '[TEST - intended for ' + email + '] ' : '')
                   + 'Attendance update - ' + details.student + ' - ' + CONFIG.COURSE;

    if (CONFIG.DRAFT_ONLY) {
      GmailApp.createDraft(actualTo, subject, body);
      Logger.log('DRAFT created for ' + details.student + ' (' + percentage + '%)');
    } else {
      GmailApp.sendEmail(actualTo, subject, body);
      Logger.log('SENT to ' + actualTo + ' re ' + details.student
                 + (CONFIG.RECIPIENT_OVERRIDE ? ' [REDIRECTED]' : ''));
    }
    drafted++;
  });

  Logger.log('Done. ' + drafted + ' message(s), ' + skipped + ' skipped. '
             + (CONFIG.DRAFT_ONLY ? 'DRAFT MODE - check Gmail Drafts.' : 'SENT.'));
}

// ---------------------------------------------------------------
//  Template version - no AI, completely predictable
// ---------------------------------------------------------------
function draftFromTemplate(d) {
  return 'Dear ' + d.parent + ',\n\n'
    + 'I am writing regarding ' + d.student + ' (Roll ' + d.roll + ') and their '
    + 'attendance in ' + CONFIG.COURSE + '.\n\n'
    + 'To date they have attended ' + d.attended + ' of ' + d.held + ' classes, '
    + 'which is ' + d.percentage + ' percent. Our expectation for this course is '
    + CONFIG.THRESHOLD + ' percent.\n\n'
    + 'Regular attendance matters a great deal in this subject because each topic '
    + 'builds directly on the previous one, and missed sessions are difficult to '
    + 'recover from independently.\n\n'
    + 'If there is something affecting their attendance that I should know about, '
    + 'please do reply to this email or come and meet me. I would rather understand '
    + 'the situation than simply record the absence.\n\n'
    + 'Warm regards,\n' + CONFIG.TEACHER;
```

```
}

// ------------------------------------------------------------
//  AI version - warmer, and can write in another language
// ------------------------------------------------------------
function draftWithGemini(d) {
  const prompt =
    'You are a college professor writing to the parent of a student about attendance.\n\n'
    + 'STUDENT: ' + d.student + ' (Roll ' + d.roll + ')\n'
    + 'PARENT: ' + d.parent + '\n'
    + 'COURSE: ' + CONFIG.COURSE + '\n'
    + 'ATTENDANCE: ' + d.attended + ' of ' + d.held + ' classes = ' + d.percentage + '
percent\n'
    + 'THRESHOLD: ' + CONFIG.THRESHOLD + ' percent\n'
    + 'SEVERITY: ' + d.severity + '\n\n'
    + 'Write a short email body in ' + CONFIG.LANGUAGE + '.\n\n'
    + 'Rules:\n'
    + '- Address the parent respectfully by name\n'
    + '- State the attendance figures plainly and accurately. Do NOT round, '
    + 'exaggerate, or add figures that were not given to you.\n'
    + '- Explain briefly why attendance matters for this course\n'
    + '- Invite them to reply or meet, do not demand\n'
    + '- Do NOT threaten, and do NOT mention marks, grades or examination '
    + 'eligibility - you have not been given that information\n'
    + '- 5 to 7 sentences\n'
    + '- Sign as ' + CONFIG.TEACHER + '\n\n'
    + 'Write only the email body. No subject line. No preamble.';

  const url = 'https://generativelanguage.googleapis.com/v1beta/models/'
            + CONFIG.GEMINI_MODEL + ':generateContent?key=' + CONFIG.GEMINI_KEY;

  try {
    const response = UrlFetchApp.fetch(url, {
      method: 'post',
      contentType: 'application/json',
      muteHttpExceptions: true,
      payload: JSON.stringify({
        contents: [{ parts: [{ text: prompt }] }],
        generationConfig: { temperature: 0.5, maxOutputTokens: 1024 }
      })
    });

    if (response.getResponseCode() !== 200) {
      Logger.log('Gemini error ' + response.getResponseCode()
                 + ' - falling back to template for ' + d.student);
      return draftFromTemplate(d);
    }

    const json = JSON.parse(response.getContentText());
    const text = json.candidates
              && json.candidates[0].content.parts[0].text;

    if (!text) return draftFromTemplate(d);

    // Safety check: the drafted message must contain the real figures
    if (text.indexOf(String(d.attended)) === -1
        && text.indexOf(String(d.percentage)) === -1) {
      Logger.log('WARNING: draft for ' + d.student
                 + ' does not contain the attendance figures. Using template.');
```

```
      return draftFromTemplate(d);
    }

    return text.trim();

  } catch (err) {
    Logger.log('Gemini call failed for ' + d.student + ': ' + err
               + ' - using template.');
    return draftFromTemplate(d);
  }
}

// ---------------------------------------------------------------
//  Run this ONCE to schedule the weekly job
// ---------------------------------------------------------------
function createWeeklyTrigger() {
  ScriptApp.getProjectTriggers().forEach(function (t) {
    if (t.getHandlerFunction() === 'runAttendanceAlerts') {
      ScriptApp.deleteTrigger(t);
    }
  });

  ScriptApp.newTrigger('runAttendanceAlerts')
    .timeBased()
    .onWeekDay(ScriptApp.WeekDay.FRIDAY)
    .atHour(17)
    .create();

  Logger.log('Weekly trigger created: Fridays around 17:00.');
}
```

Step 30 — Run it

1. Select `runAttendanceAlerts` in the function dropdown
2. Click **Run**
3. Google asks for authorisation the first time — review the permissions and allow
4. Open **Gmail → Drafts**. Your messages are waiting.

> **NOTE THE FALLBACK DESIGN**
>
> Every AI path in this script falls back to the template on failure — a bad response, a rate limit, a missing figure.
>
> A parent communication system that silently stops working on a Friday is worse than one that sends a plainer email. **Design the fallback before you design the AI.**

Step 31 — Schedule it

Select `createWeeklyTrigger` in the dropdown and click **Run** once.

Check **Triggers** (the clock icon in the left sidebar) to confirm.

NO API KEY AT ALL IS A LEGITIMATE CHOICE

Leave `GEMINI_KEY` blank and the script uses `draftFromTemplate`. No AI, no key, no rate limits, no possibility of an invented figure, and it never breaks.

For a weekly attendance note, that is arguably the right answer. Use the AI version when you want warmth, personalisation, or Marathi — not by default.

Knowing when *not* to use the AI is part of the skill.

PART D CHECKPOINT

- [ ] Script runs and creates Gmail drafts
- [ ] I read a draft and checked the figures against the sheet
- [ ] Weekly trigger created and visible in the Triggers panel
- [ ] I tried it once with `GEMINI_KEY` blank and compared the template output

---

9.7 Before you use this on a real class

SEVEN THINGS TO SETTLE FIRST

1. Institutional approval. Contacting parents is a formal communication with weight. Your HoD signs off on the wording, the threshold and the frequency before the first send.

2. Never publish a real sheet to the web. Use the Apps Script version, which reads the sheet natively, or a Google service account with the sheet shared privately. Attendance data about named minors on a public URL is a serious data protection problem.

3. Keep the override on for the first real run. Point it at your own address, run against the real class sheet, and read what *would* have gone out. Only then clear it.

4. Draft mode for at least the first three weeks after that. Read every message. When you have gone three weeks without finding an error, consider sending directly — and even then, keep a log.

5. Verify the source data first. The most likely wrong email is not caused by the AI. It is caused by a marking error in the sheet, or a row that shifted. **Garbage in, confidently worded garbage out.**

6. Check the parent email column every semester. An out-of-date address means either nothing arrives, or a stranger receives information about somebody's child.

7. Tell students the system exists. A student who knows a parent will be contacted at 75% has a chance to talk to you first. That is the entire point — the alert is meant to start a conversation, not to be a surprise.

THE HONEST SUMMARY

The technology here is easy. Reading a sheet and sending an email is a first-year exercise.

Everything that makes this hard is judgement: who to contact, when it helps and when it makes things worse, what tone, and what to do when the data is wrong.

Part A automates the easy bit. Part B automates a little of the judgement, and shows its reasoning so you can check it. Neither replaces the decision to pick up the phone about a student you are worried about.

Appendix E — The sample data

The file `Attendance_Sample_Data.xlsx` is in the shared workshop folder. A `.csv` twin is there too if you prefer to import that.

Upload it to Google Drive, open with Google Sheets, and publish Sheet1 as CSV. Do not retype it.

```
roll_no,student_name,parent_name,parent_email,classes_held,classes_attended,previous_percent
age,last_alert_date
R001,Aarti Kulkarni,Sunita Kulkarni,parent.r001@example.invalid,40,36,88,
R002,Bhavesh Patil,Ramesh Patil,parent.r002@example.invalid,40,38,93,
R003,Chetan Deshmukh,Vaishali Deshmukh,parent.r003@example.invalid,40,19,52,2026-08-12
R004,Deepa Joshi,Anil Joshi,parent.r004@example.invalid,40,29,91,
R005,Eshan Rane,Meera Rane,parent.r005@example.invalid,40,34,84,
R006,Farhan Shaikh,Nasreen Shaikh,parent.r006@example.invalid,40,15,41,2026-08-05
R007,Gauri Naik,Prakash Naik,parent.r007@example.invalid,40,28,55,2026-08-21
R008,Harsh Mehta,Kiran Mehta,parent.r008@example.invalid,40,37,90,
R009,Isha Bhosale,Sanjay Bhosale,parent.r009@example.invalid,40,39,95,
R010,Jatin Kale,Suresh Kale,parent.r010@example.invalid,40,33,80,
R011,Kavya Pawar,Nitin Pawar,parent.r011@example.invalid,40,27,69,2026-09-06
R012,Lalit Gaikwad,Asha Gaikwad,parent.r012@example.invalid,40,35,86,
R013,Manasi Thorat,Dilip Thorat,parent.r013@example.invalid,40,22,60,2026-08-28
R014,Nikhil Sawant,Rekha Sawant,parent.r014@example.invalid,40,38,92,
R015,Omkar Chavan,Vijay Chavan,parent.r015@example.invalid,40,,87,
```

WHY `.INVALID` AND NOT `EXAMPLE.COM`

`.invalid` is a reserved top-level domain that, by internet standard, can never be registered and never resolves. Mail addressed there cannot be delivered anywhere, by anyone, ever.

`example.com` is a real registered domain. It is reserved for documentation and does not accept mail, but it exists, and "does not currently accept mail" is a weaker guarantee than "cannot exist."

When you invent test addresses, use `.invalid`. It costs nothing and removes a category of accident.

WHAT THE DATA IS DESIGNED TO TEST

Roll	Now	Before	Last alert	Tests
R003	47.5%	52	4 weeks ago	Critical, still falling
R004	72.5%	91	never	**Sharp fall.** Contact now.
R006	37.5%	41	5 weeks ago	Most critical case
R007	70.0%	55	3 weeks ago	**Recovering.** Should NOT be warned.
R011	67.5%	69	**5 days ago**	**Repeat case.** Escalate, do not email.
R013	55.0%	60	2 weeks ago	Serious, recently contacted
R015	—	87	never	**Missing data.** Must be skipped, not guessed.

R015 exists to check that your code skips bad rows rather than treating a blank as zero and telling a parent their child attended nothing.

R004, R007 and R011 are the three that separate Part A from Part B.