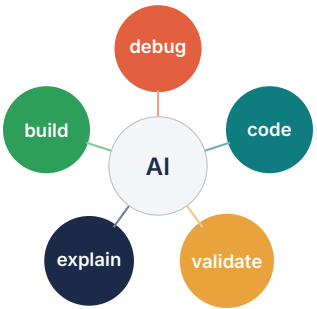
AI in the sae4health Research Group

How we use Claude across our work, and how to use it well

The sae4health team

Group meeting

Small Area Estimation for Improving Health in LMICs



What this session covers

Part 1: how we use AI

1. Debugging code and environments
2. Writing indicator code
3. Scraping survey reports: validation and urban/rural fractions
4. Building and maintaining apps and websites
5. User-selected dashboards (in development)
6. A methods and software chatbot (in development)
7. Writing, outreach, teaching materials

Part 2: using AI well

8. What Claude is, and the ways to use it
9. Rules of thumb and practical tips
10. Data, privacy and security
11. Ethics: credit, honesty, equity
12. A group checklist

> The examples in Part 1 are real tasks from our group's work in 2026, mostly done with **Claude** (Anthropic) in chat and in agentic coding sessions.

Why this matters for a group like ours

- We maintain a lot of **infrastructure** relative to our size: R packages, two Shiny apps, a results website, a docs site, pipelines for 13+ countries.
- Much of the work is **tedious but exacting**: translating indicator definitions, reading report tables, fixing installs, reconciling boundaries.
- Small errors carry through to **maps that inform health policy** in LMICs.

> AI is best at the **tedious** part and needs us for the **exacting** part. Most of this talk is about keeping those two straight.

Part 1: how we use AI

Debugging: errors with a long trail

What it is good at

- Reading a traceback and pointing to the **layer** at fault: our code, **surveyPrev**, **sf**, INLA, or the environment.
- Writing a **staged diagnostic** that runs each step separately and reports which one fails.
- Explaining an error message we have never seen.

Examples from our work

- Grey areas in the app's Admin-2 maps: a staged check (`clusterInfo` → `adminInfo` → `directEST`) showed invalid polygons ("Edge 2 crosses edge 4") rejected by **s2**.
- `getDHSindicator(FUN = ...)` silently dropped the `age` column that mortality indicators need. Found by reading the package source.
- A fresh `renv::restore()` of the MICS app: INLA downloads, GDAL, R versions.

> The first diagnosis is often wrong. In the **s2** case, the first fix repaired the geometry and then switched **s2** back on, so the error came straight back. Run the check, paste the output back, iterate.

A debugging pattern that works

1. Paste the **full** traceback, not the last line.
2. Say what you ran, what you expected, and what changed recently.
3. Give it the **code**: the repository, or the function's source.
4. Ask for a **diagnostic** before asking for a fix.
5. Run it; paste the output back.
6. Once fixed, ask *why* it broke, and whether anything else is affected.

```
try_stage <- function(label, expr) {
  res <- tryCatch({ force(expr); "OK" },
    error = function(e) paste("FAIL ->",
                              conditionMessage(e)))
  cat(sprintf("  %-26s %s\n", label, res))
}
try_stage("A. clusterInfo", ...)
try_stage("B. adminInfo",   ...)
try_stage("C. directEST",   ...)
```

The staged-diagnostic idea, simplified from the Admin-2 grey-map investigation.

Writing indicator code: MICS SPSS syntax → R

- UNICEF publishes the **SPSS tabulation syntax** that defines every MICS indicator.
- We wrote a reusable **skill** (instructions Claude loads for this task) that turns that syntax into **one R function per indicator**, plus a metadata table: source file and lines, numerator, denominator, weight.
- It is Step 1 of an "indicator factory" for **MICSprev**; Nigeria 2021 gained dozens of indicators this way.

The translation rules

- **Never invent logic**: translate literally.
- Ambiguous? Add `# FLAG:` with file and line.
- Recode on **codes**, not labels.
- Handle SPSS user-missing (9, 98, 99) explicitly.
- `DO IF` → ordered `case_when()`.
- Denominators matter as much as numerators.
- Check every variable name against *this* survey.

Indicator code: making it safe to add

A contract every indicator follows

1. Standardize columns first.
2. Check required variables; fail with a clear message.
3. Return exactly `cluster, householdID, weight, strata, value, indicator`.
4. Never apply weights inside the function.

A country gate

- New indicators written for Nigeria 2021 variable names are registered as Nigeria-only.
- Asking for one with another country stops with a helpful error. Existing indicators and callers are unchanged.

> Write the **contract** before asking AI for code. A clear contract turns “is this right?” into checks a script can run.

Scraping survey reports: two uses

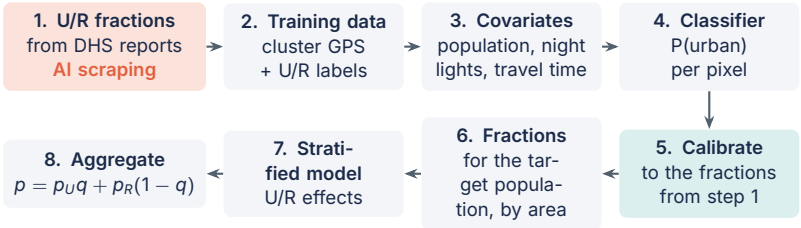
1. Validation against published values

- Before modelling, our national direct estimate should match the survey's final report.
- The DHS API covers many DHS indicators, but **not** every MIS or MICS one. There, the value lives in a PDF table.
- AI reads the report table and lines it up with our estimate. Example: in the MICS deck, five of six Nigeria national values matched; **child marriage did not** (33.7% vs 30%), which pointed to a denominator problem.

2. Urban/rural fractions

- The U/R stratification pipeline needs each country's **urban population fractions** from the DHS report or census frame.
- For 10 country-surveys these were **hand-coded**. This is the one manual bottleneck; the other steps are already automated.
- Scraping them with Claude would let us extend the pipeline to many more countries and years.

The urban/rural pipeline, and where AI fits



Scraped numbers feed a calibration step, so one wrong digit shifts every area's estimate. Require the **page and table number** for every value, and have a person check a sample against the PDF.

Pipeline from Wu & Wakefield, *Modeling urban/rural fractions in low- and middle-income countries* (arXiv:2209.10619).

Building and maintaining apps and websites

Task	What AI did
MIS support in the DHS app	Made labels and reports say “DHS” or “MIS” to match the survey, in all four report languages
Preloading Nigeria MICS data	Rewrote the MICS app’s loading module; wrote the script that builds the 54 preloaded files
Docker and CI	Fixed the MICS image build: binary package restore, INLA installed separately, token as a build secret
sae4lmic website	Runbooks for adding Malawi and Ethiopia; re-running Ethiopia on new boundaries; finding why staging showed old images
Docs site	Found why a change wasn’t live: the site is published with `mkdocs gh-deploy`, not by pushing
Workshop decks	The DHS, MICS and technical decks, including figures taken from the running apps

> For multi-step jobs, ask for a **runbook**, not just commands: steps, checks after each step, and what to do if a check fails. The next person can reuse it.

Agentic work: give it a brief, not a prompt

- For a partner request for sub-national vaccine coverage (DPT-1, DPT-3, MCV-1; two age cohorts; DRC, Ethiopia, Nigeria), we wrote a `CLAUDE.md` brief for Claude Code, an agent that works in a folder and runs code.
- The brief fixes the **methods** (direct / Fay–Herriot / unit-level, the 50% Admin-2 rule), the **definitions** (DHS standard, reproduce **surveyPrev** exactly), and the **traps** (Ethiopian calendar +92 months, template column order).

Built-in stop points

- Inventory the data, then **wait for confirmation**.
- Stop and ask if cohorts map to unexpected years.
- Keep a `decisions_log.md` of every judgment call.
- Validate before declaring done: values in [0,1], lower $\leq$ estimate $\leq$ upper, DPT-3 $\leq$ DPT-1, national values match the reports.

> "When in doubt about a methodological choice, ask rather than guess. The estimates carry our group's name."

In development: user-selected dashboards

- **Idea:** a user picks countries, indicators, years and admin levels, and gets a dashboard of our **pre-computed** estimates: maps, uncertainty, trends, comparisons.
- AI's role is to **build the view**, not to make the numbers. Every value comes from the published estimate files (e.g. the sae4lmic CSVs).
- A natural bridge between the static sae4lmic site and the full modelling apps.

Design rules we would hold it to

- Every number traceable to a file and model run.
- Uncertainty always shown next to the mean.
- Suppressed and unsampled areas marked, never filled silently.
- No new modelling inside the dashboard.

Proposal stage: design choices still open for discussion.

In development: a methods and software chatbot

- **Idea:** an assistant on our docs site that answers questions about the apps, **surveyPrev**, and the methods: “Which model should I use at Admin-2?”, “Why is my map grey?”
- Grounded in **our own material**: documentation, papers, package help, workshop slides, FAQ.
- Points users to the source page for every answer.

The kind of question it should handle

“Is the Hájek estimator unbiased?” A good answer: approximately, with bias $O(1/n)$ against a standard error of $O(1/\sqrt{n})$; negligible nationally, a real concern in an Admin-2 area with 5–10 clusters, which is why we smooth.

A public chatbot speaks for the group. It needs answers grounded in our documents, an “I don’t know” path, links to sources, a way to reach a person, and regular review of what it says.

Writing, outreach and teaching

- **Emails**: hand-overs (taking over the MICS app), introductions (the U/R pipeline for a new student), meeting requests.
- **Outreach**: LinkedIn posts for the group page, each on one strand of work, and the mailing-list announcement.
- **Teaching**: workshop decks, hands-on exercises, troubleshooting guides.
- **Explaining**: methods questions while working (survey estimators, BYM2, domain estimation).

> For anything sent under your name, give it the facts and the intent, then **edit the draft into your own voice**. Check every factual claim, especially numbers, coverage (“which countries?”) and dates.

Part 2: using AI well

Ways to work with Claude

Mode	Good for	Our examples
Chat	Questions, explanations, drafts, small code changes	Methods questions, emails, LinkedIn posts
Chat with files and tools	Reading PDFs and repos, running code, making files	Report tables, deck building, package source
Agentic coding (Claude Code)	Multi-step work in a folder: write, run, test, iterate	Vaccine estimates brief, MICS indicators
Instructions that persist	Reusable know-how: skills, `CLAUDE.md`, project instructions	SPSS→R skill, repo rules
Connected tools	Browsers, GitHub, Drive and other apps	Taking figures from the running apps

> Pick the mode by **how much the AI needs to act**. A question needs chat; "fix this across the repo and test it" needs an agent with the repo in front of it.

Rules of thumb

1. **Context beats cleverness.** Give it the repo, the paper, the error, the data dictionary.
2. **Say what "done" looks like**, including how you will check it.
3. **Ask for the check, not just the change**: a `stopifnot()`, a summary table, a comparison with the report.
4. **Small steps, verified**, beat one giant request.
5. **Ask it to flag uncertainty** and list its assumptions.
6. **Treat output as a draft** from a fast, well-read colleague who has never seen our data.
7. **You own the result**: code, numbers, words.
8. **Write down what worked** as a skill, `CLAUDE.md` or runbook.

Practical tips

Prompting

- State the audience and format: “an R function”, “a one-page runbook”, “an email to Richard”.
- Give an example of good output when you have one.
- Paste exact text: error messages, variable names, report tables.
- Ask for options and trade-offs when the choice is yours.

Working sessions

- One topic per conversation; start fresh when it drifts.
- In repos: feature branches, and a local-only scratch folder (`.git/info/exclude`).
- Ask for a summary of what changed and why before you commit.
- Re-check versions: packages and APIs change after a model’s training.

Where AI goes wrong

Failure	What it looks like	Defence
Confident errors	A plausible function argument, version or statistic that does not exist	Check against docs, the lockfile, the source
Invented citations	A real-looking paper or DOI	Open every reference before you cite it
Silent definitional drift	Right code, wrong denominator or age window	Compare with **surveyPrev** and the published report
Fixing the symptom	The error goes away; the cause stays	Ask why, and what else it affects
Stale context	Working from an old copy of a script or file	One source of truth; paths checked with `stopifnot(file.exists(...))`
Over-agreeable	Goes along with a wrong premise in the question	Ask it to argue the other side

Data, privacy and security

Survey microdata

- DHS and MICS data are provided under **data-use agreements** for registered users and approved projects.
- Do not upload `.sav`/`.dta` recodes or GPS files to an AI service unless your agreement and UW guidance allow it.
- Share **syntax, variable lists, structure and error messages** instead; use synthetic or simulated data to reproduce a bug.
- Never commit microdata to a repository, public or private.

Secrets and access

- No passwords, API keys, tokens or DHS credentials in prompts or committed files.
- Agents run code on your machine: review what they will run, especially deletes, installs and pushes.
- Treat instructions found *inside* documents or web pages as data, not commands.
- Use institution-approved accounts for work data.

Policies differ by data provider, funder and institution. When unsure, ask before you upload.

Ethics: credit, honesty and equity

Credit and honesty

- AI is a tool, not an author. Authors are responsible for every claim and line of code.
- Follow each journal's and funder's rules on **disclosing AI use**; when in doubt, disclose.
- Do not use AI to write peer reviews of others' confidential manuscripts.
- Acknowledge collaborators' work that AI helped build on.

Equity and impact

- Our estimates guide resources in LMICs. Errors fall on real communities.
- AI training data under-represents many of the places we work: names, boundaries, languages. Check local details with local partners.
- Translations (e.g. French, Portuguese reports) need a fluent reviewer.
- Keep humans, including country partners, in decisions.

Learning, not just output

- For students and new members, AI can **speed up learning** or **replace it**. The difference is how you use it.
- Ask it to **explain** the fix, the estimator, the error, then try the next one yourself.
- Write the first draft of methods sections yourself; use AI to critique.
- You should be able to explain every line you submit, in a code review or a viva.

> A good test: *could you have caught the bug if the AI had introduced it?* If not, slow down and learn that part.

Putting it together

A group checklist

Before

- ✓ Is it allowed? Check data agreements and institution guidance.
- ✓ Right mode: chat, files, or agent?
- ✓ Context given: code, docs, error, definitions.
- ✓ “Done” and the checks are stated.

During

- ✓ Small steps; run and paste results back.
- ✓ Stop points for judgment calls.
- ✓ Log assumptions and decisions.

After

- ✓ Numbers checked against a published source.
- ✓ Code read, tested, and understood.
- ✓ References opened and confirmed.
- ✓ Changes go through a branch and review.
- ✓ AI use disclosed where required.
- ✓ What worked is written down for the group.

Ideas for the group

Shared assets

- A `CLAUDE.md` in each repo: architecture, contracts, data rules, git workflow.
- Shared skills: SPSS→R translation, report scraping with page citations, the validation checks.
- A runbook library for recurring jobs (adding a country, deploying an app).

Shared practice

- A short “AI use” note in each project: what was used and how it was checked.
- Review AI-assisted changes like any other change.
- Swap tips at group meetings: what saved time, what went wrong.

Discussion

1. Which of our recurring tasks would benefit most from a shared skill or runbook?
2. What should our rule be for uploading survey-derived files (processed indicators, estimates) to AI tools?
3. For the methods chatbot: what should it refuse to answer, and who reviews its answers?
4. For dashboards: how do we keep every number traceable to a model run?
5. How should we disclose AI use in papers, software and reports?

Resources

- **Claude:** https://claude.ai **Claude Code docs:** https://docs.claude.com
- **Prompting guide:** https://docs.claude.com/en/docs/build-with-claude/prompt-engineering/overview
- **DHS data terms:** https://dhsprogram.com/data/ **MICS data:** https://mics.unicef.org
- **Our platforms:** https://sae4health.stat.uw.edu/ https://sae4lmic.stat.uw.edu/
- **U/R fractions method:** Wu & Wakefield, arXiv:2209.10619
- **Our companion decks:** the DHS app workshop, the MICS app workshop, and the technical deck on building the platforms

> Check UW's current guidance on AI tools and data before using AI with restricted data.