ECON 594: Applied Economics

How to Use AI

Sam Norris
University of British Columbia

Where we are

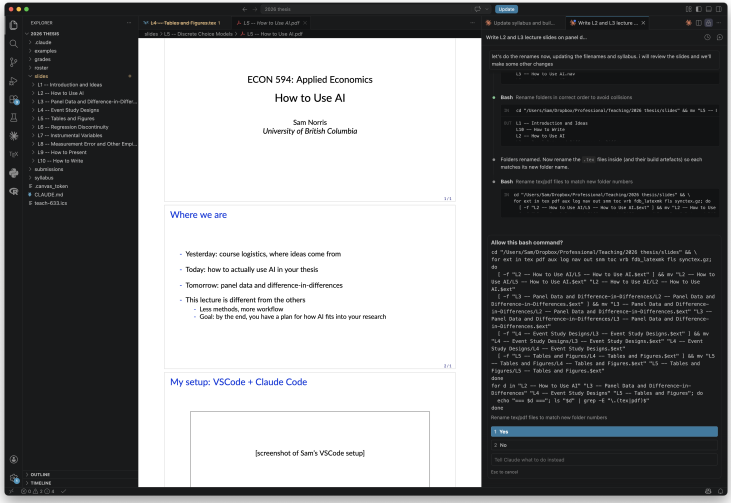
The arc of the week

- Yesterday: course logistics, where ideas come from
- Today: how to actually use AI in your thesis
- Tomorrow: panel data and difference-in-differences

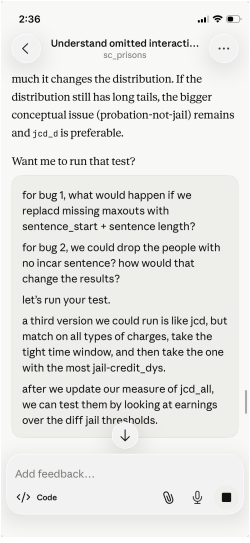
This lecture is different from the others

- Less methods, more workflow
- I want you to have a plan for how AI fits into your work

My setup: VSCode + Claude Code



Go for a walk! You can remote control Claude from your phone



The tool landscape

Claude Code is what I'll demo today

- Terminal/IDE integration, can edit files, run code
- claude.ai: web interface, good for writing/exploration

Other options: ChatGPT, Cursor, Copilot, Gemini

- All roughly $20/month for the cheap tier
- Principles very similar

Pick one, learn it, then branch out

The big idea: context is everything

AI is good when it has context, bad when it doesn't

- Without context, you get generic slop

Most of your job is feeding it the right context

- What is the project about?
- What does your data look like?
- What conventions do you follow?
- What papers are you building on?

Where AI helps

Writing code

- Quick descriptive figures (binscatter, histogram, time series)
- Stata code generation (regression specifications, data cleaning)
- Draft regression tables, LaTeX formatting

Digesting things

- Summarizing papers (research design, results, mechanisms)
- Explaining things you don't understand (a method, a paper, a result)

Reviewing your work

- Code review ("what's wrong with this regression?")
- Proofreading and clarity edits on your own writing

Where AI fails

Taste

- Knowing what's important: research-design judgment
- Generating novel research ideas
- Sticking to your conventions without being told

Self-awareness

- Knowing when it's wrong (it almost never says "I don't know")
- Telling you to stop or rethink
  - Claude is always "yes, and", never "no, but"

Citations: often wrong, sometimes invented

Don’t outsource your understanding

AI is (incredible) auto-correct

- It does not quite understand

The oral exam tests you, not your AI

- You should be able to explain why your code does X

Use AI to help you learn, not to skip learning

- “Walk me through what this code is doing”
- “Why is this research design appropriate?”
- “What would change if I controlled for X?”

Use CLAUDE.md to consistently manage context

Every new chat starts cold

- Without context, AI defaults to generic slop
- You will get tired of explaining the same things over and over

Solution: a project file the AI auto-loads

- In Claude Code, this is `CLAUDE.md`
- Lives in the project directory
- Loaded into every conversation in that directory

Real example: this course's CLAUDE.md

```
## Notation conventions

### Regression equations: coefficient before regressor
When writing regression equations, the coefficient comes BEFORE
the regressor when it's a scalar: write $\beta X_{st}$, not
$X_{st}\beta$.

### Event-study time indexing: omit period 0
Period 0 is the last period before treatment (the omitted
reference). Period 1 is the first period of treatment.

## Slide style
- Frame titles in sentence case
- No \textbf or \emph in slide content
- No "Plan for today" auto-section roadmaps

## Stata conventions for event studies
- Manual relative-time dummies (gen pre3 = (ktime == -3))
- Cluster at the unit level
```

What to put in your CLAUDE.md

CLAUDE.md should include info that will be relevant across tasks

- Project overview: one paragraph on what you're doing and why
- Data dictionary: variable names, units, sample restrictions
- Conventions: file structure, naming, regression notation
- Curated paper summaries: what each one does, why it matters, what design
- Open questions and known pitfalls
- Things you've already ruled out (so AI doesn't propose them again)

Treat it as a living document

- Update as the project evolves.

Context management

Long sessions get less reliable

- Models forget early instructions
- They hallucinate more
- They drift away from your conventions

Solution: start a fresh session after each discrete task

- “Fix this bug” is a discrete task
- “Help me with my thesis” is not
- Goal: don’t get to more than 40% of context used

Working with code

AI will be confidently wrong

- Especially when it doesn't have the right context or can't check facts
- More mistakes explaining the world / fewer mistakes with code

Check its work

- Use Github Desktop to check its changes before accepting
- Check that the regression spec is what you asked for
  - Look at the code
  - Get fresh agents to write descriptions of design from the code only

"Does this match what I asked for?" is a question only you can answer

Use a second agent to audit

One agent does the work, a second one reviews

- In Claude Code: spawn a subagent for code review
- Especially valuable before committing or sharing results
- You can use a different model as the reviewer to fill in blindspots

The reviewer will catch things the writer missed

- Wrong cluster level
- Sample restrictions that drop the wrong observations
- Unused variables, dead code

A second agent is no substitute for your brain

AI gives you confident-sounding stories

- Sometimes narratively correct, quantitatively wrong

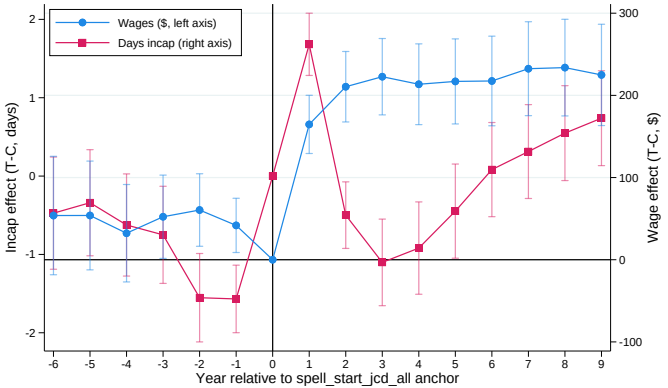
Make it work through the logic

- “What would have to be true for that to be the explanation?”
- “Multiply through and tell me if the magnitudes work”
- “What is the implied elasticity?”
- “Is this consistent with what we know from prior literature?”

Example: a pre-trend with a confident story

Event study of earnings on prison security level, pre-period = before prison

- A pre-trend in earnings, and a similarly-shaped pre-trend in time in jail
- Claude: “This nails it: incapacitation is causing the earnings pre-trend”



... but the magnitudes don't work

Average wages: $\approx \$700$ per quarter

- Pre-trend incapacitation effect: ≈ 1.5 days incarcerated per quarter
- Implied earnings effect: $\$700 \times 1.5/90 \approx \$12$ per quarter
- Observed pre-trend in earnings: \$50-70
- Incapacitation can't plausibly explain it!

Lesson: Claude's story sounded plausible until common sense killed it

Using git as a safety net

Git is a tool to track changes in code

- Github Desktop is a gentle, non-command line version

Every AI change should be a git change

- You can see exactly what changed (`git diff`)
- You can roll back when AI breaks something (`git reset`)
- You build an audit trail of what AI did and when

Commit early, commit often

- Claude will write you commit messages
- These help with getting the context right if you have to revisit decisions

Putting it together

The task → commit → clear → next-task loop

1. Define one discrete task in a sentence or two
2. Let AI do the task
3. Review the diff, run the code, and read the output
4. Commit when it works
5. Clear the context (start a fresh session)
6. Pick the next task

Writing with AI

You need to know what you want to say first

- The hard work is the planning

Outline first: paper $\rightarrow$ section $\rightarrow$ paragraph

- Within each section, sketch what each paragraph will say
- AI can help with this, but it needs guidance

AI is best at discrete, high-context tasks

- Drafting a paragraph is easy when you can point it in the right direction

Where AI shines for writing

Reviewing what you wrote

- “Is this paragraph saying what I think it’s saying?”
- Clarity edits: cut wordiness, fix awkward phrasing
- Smoother transitions between paragraphs
- Proofreading typos and grammar
- Reformatting for journal style (e.g., AER:Insights)

Not for: generating ideas, inventing arguments, or citing papers

Live demo: “Make me a binscatter”

Goal: a binscatter of Y on X, conditional on controls

- I’ll ask Claude and we’ll watch what it produces
- Then we’ll ask whether it’s actually right.
- Watch for: confident answer, plausible code, hidden problem.

What Claude probably gave you

```
ssc install binscatter
binscatter wages security_level, ///
    controls(age prior_offenses) n(20) linetype(lfit)
```

Under the hood: residualize Y on Z, residualize X on Z, then bin and plot

- AI's framing: "equivalent to a partial regression plot, with binning for visualization"

But is it actually right?

The problem with double-residualization

You bin the residualized X, not X

- Within each bin, the residualized X varies (because Z varies)
- You're conflating within-bin variation in X with the projected-out variation
- The bins are no longer bins of X

Cattaneo, Crippa, Farrell, Feng, Jansson (2024, "On Binscatter"):

- Use `binsreg`: regress Y on bin dummies of X plus controls Z linearly
- The bin coefficients are the conditional means
- Same regression produces both the bins and the partialling out

The right way: `binsreg`

```
ssc install binsreg
binsreg wages security_level age prior_offenses, ///
    nbins(20) polyreg(1)
```

What this fixes

- Bins of X are bins of X, not of residualized X
- Controls enter the regression linearly, in the same step
- Confidence bands are honest (uniform inference, not pointwise)

The lesson of the demo

- AI gave you confident, fluent, plausible code
- It was wrong
- You catch it by knowing what the answer should look like

One last thing: subagents, skills, slash commands

One golden rule of coding: don't duplicate code

- Same principle applies with AI

Workflow tricks that let you set it and forget it

- Skills: reusable instruction packs ("make a regression table," knows your formatting)
- Slash commands: shortcuts you create (e.g., `/binsreg` runs your binscatter workflow)

Another useful trick

- Subagents: spin up a second AI to audit work, do parallel research, or give a second opinion
  - "Have another agent review this regression for bugs"
- Helps with context management

Next up

Tomorrow: panel data and difference-in-differences

- This week: review of the empirical methods you'll use in your thesis
- Office hours and TA office hours start this week (see syllabus)