Claude Code for Academics

An AI Agent for Empirical Research

Alessandro Spina[1]

UTS Finance Department Brownbag

March 2026

[1]This talk is in the spirit of Robin Hood: everything is stolen with the intent to share. See linked sources. All errors are mine.

Roadmap

1. Why should you care?
2. What is Claude Code?
3. What can I use Claude Code for?
4. Setting up Claude Code
5. Using Claude Code
6. Where to start?
7. How to stay safe
8. Problems to be aware of
9. Where to from here?

1
Why should you care?

Why Should Academics Care?

The Problem Today

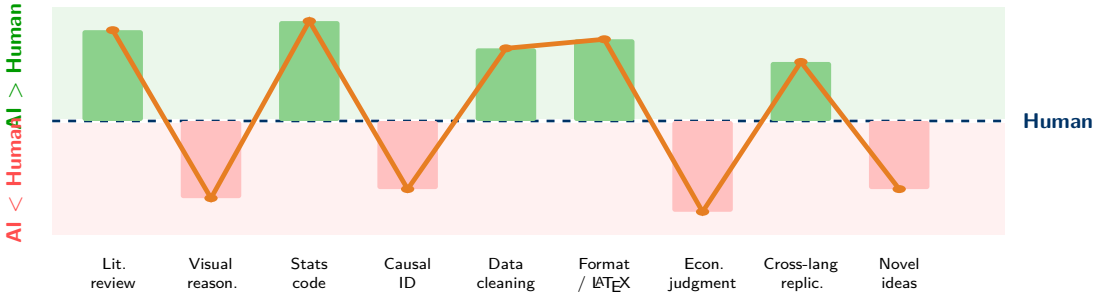
Most of us use some form of LLM: tell it what you want, it answers, copy and paste, repeat. Often lacks context, prone to errors, cannot implement . . .

The Opportunity

A dedicated (team of) "RAs" who reads your data, runs code, builds your slides, and *works with you* to implement a project from start to finish.

For most of us, the bottleneck is not "I can't write a loop in R." It is "I don't have time to clean this data, reformat these tables, write that referee report, and still think carefully about identification." Claude moves the bottleneck back to where it should be: thinking deeply about economic questions.

The Jagged Frontier



The implication: AI skeptics and AI evangelists are both right.

https://www.oneusefulthing.org/p/the-shape-of-ai-jaggedness-bottlenecks

Today's Objective: Expose You to the Dark Side...

The Apprentice

The Disciple

The Master

Today's slides will not grant you the title of Master.

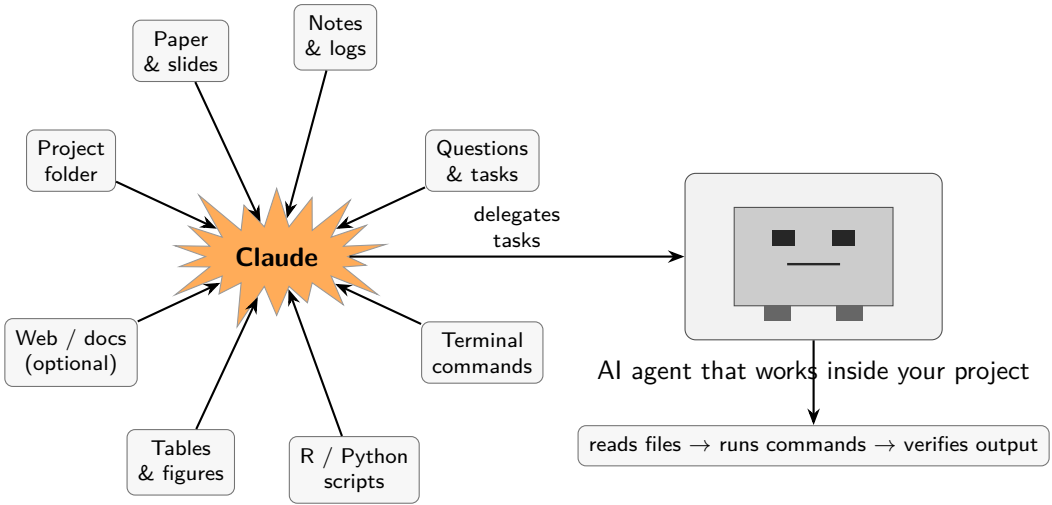
2
What is Claude Code?

What Is Claude Code?

Claude Code is an **AI agent** made by Anthropic that lives in your terminal and operates directly on your project files.

Read & Edit Files	your project's data, scripts, LaTeX
Execute Code	R, Stata, Python — in your terminal
Remember Context	via `CLAUDE.md` files
Generate Tables/Figures	local or GitHub

Claude Code = LLM + Tools



Setup Claude Code

```
# 1. Install (one time, in PowerShell)
$ irm https://claude.ai/install.ps1 | iex
# 2. Close & reopen PowerShell
# 3. Navigate to your project
$ cd ~\research\asset_pricing
$ claude
> Read the data in Data/Raw/crsp_monthly.csv, run summary statistics,..
```

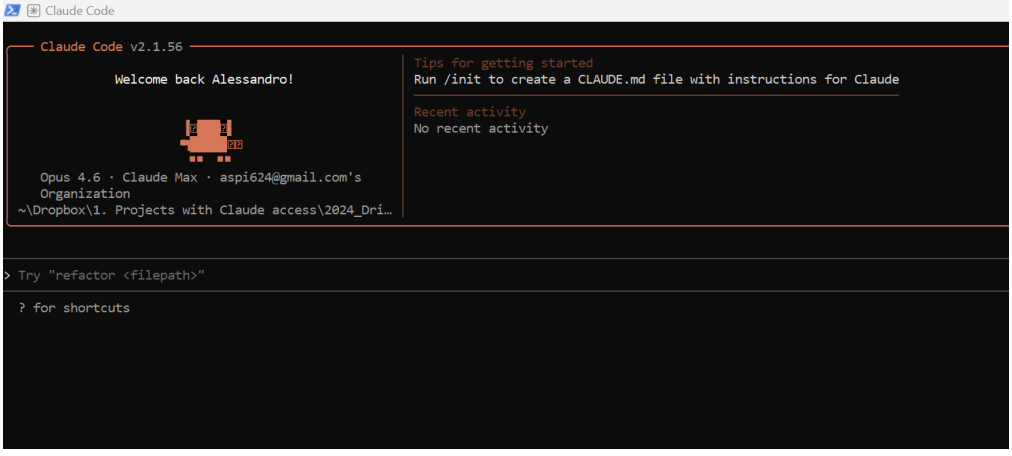
Prerequisites:

1. Git for Windows (`git-scm.com`)
2. A Claude Pro or Max subscription

One command to install. Navigate to your project and type `claude`. Describe what you want in plain English.

Install methods change frequently — check `code.claude.com/docs/en/setup`.

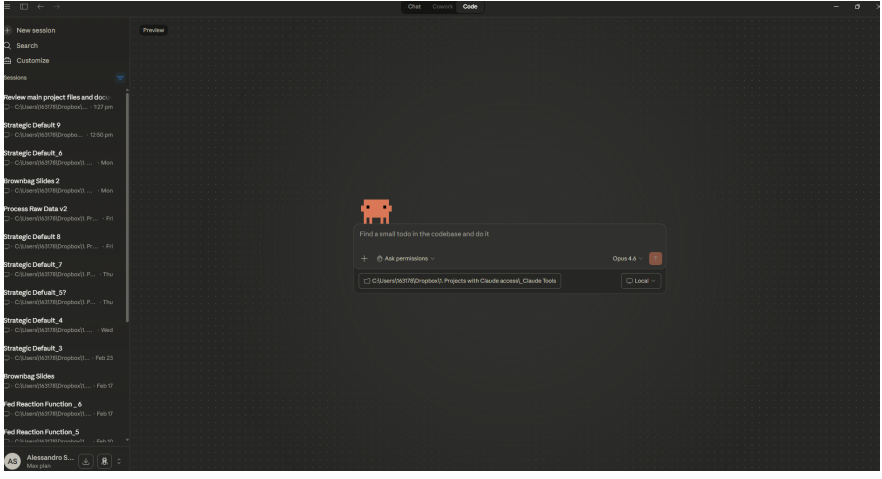
Claude Code — via Terminal



"Codebase" = Your project folder

Claude Code - via Desktop

The **Claude Desktop app** provides a visual interface. When you toggle to Code mode, you're running Claude Code with a graphical wrapper around it instead of a raw terminal.



What Does It Cost?

Pro ($20/mo) — includes limited Claude Code usage. Good for trying it out.

Max ($100/mo) — 5× more usage. I use this level and have never hit my limits (yet)

Max ($200/mo) — 20× more usage. Needed for serious research work.

Is It Worth It?

At $100–200/mo, Claude Code costs **less than a part-time RA** but is available 24/7.

Pricing changes frequently — Is this a future risk?

The Alternatives

GitHub Copilot / Cursor — excellent for code completion inside an IDE. Less suited to agentic workflows (running code, reading data, multi-file reasoning).

ChatGPT / Gemini — browser-based chat. Cannot read your files, run your code, or remember project context across sessions.

OpenAI Codex CLI — similar agentic model to Claude Code. Newer, less mature ecosystem.

Claude Code — terminal-based agent that reads files, runs code, and builds persistent memory via markdown. Strongest for **research workflows** that span code, data, and writing.

The markdown-based memory system (CLAUDE.md, skills, rules) means **no vendor lock-in**. If you switch tools, your project knowledge transfers.

3
What can I use Claude Code for?

Things You Can Ask Claude to Do

You've installed Claude Code — now what?

1. **Reorganise your directory** — clean up years of accumulated folders and files
2. **Turn a paper into slides** — feed it an Overleaf doc and get a Beamer deck
3. **Check, rewrite, or rerun your code base** — audit existing scripts for bugs
4. **Debate specifications** — explore alternative models and weigh the pros/cons
5. **Pull in data** — web search, access APIs, download and clean datasets
6. **Write** — Draft sections? `https://ape.socialcatalystlab.org/`

And with **Agent Teams**, you can run *all of these at once* — multiple specialists working in parallel, orchestrated by a coordinator.

Making Slide Decks with Claude

Claude can generate entire Beamer presentations — but the key is giving it a **style template** to follow.

This entire deck was built and styled with Claude Code. Every TikZ diagram, every tcolorbox, every slide layout. I provided the structure/content, Claude decided how to style the slide.

See prompt template: `https://github.com/scunning1975/MixtapeTools/blob/main/presentations/deck_generation_prompt.md`

Tips:

- Give Claude an existing .tex file as a style reference (e.g. Rhetoric of Decks)

Personal Example: Rebuilding a Teaching Course

Before:

Last semester's PPT slides, quizzes, exams — outdated references, inconsistent formatting, text-heavy slides that put students to sleep.

After (a few prompts):

Claude read the entire course, converted PPTs to Beamer, extracted images, turned text walls into TikZ diagrams, updated assignments, enforced consistent formatting.

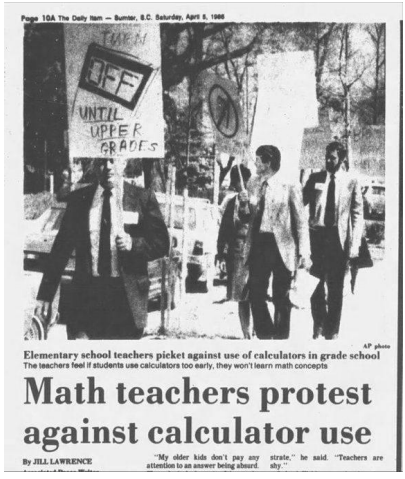
Upfront fixed cost, but going forward: **one command** to update the entire course for next semester. Teaching admin time drops dramatically.

A Brief Aside...

The academic culture wars around AI have begun. Some will shake their heads at the outsourcing of slide decks to Claude.

My take: Nobody's tombstone reads **"Wrote his own LaTeX code."**

Outsource the tedium. Focus on what makes academia fun: **developing questions, deep thinking and solving puzzles** — things AI can't quite do yet.


Page 10A The Daily Item — Sumter, S.C. Saturday, April 6, 1985

OFF UNTIL UPPER GRADES

AP photo

Elementary school teachers picket against use of calculators in grade school

The teachers feel if students use calculators too early, they won't learn math concepts

Math teachers protest against calculator use

By JILL LAWRENCE

"My older kids don't pay any attention to an answer being absurd. strate," he said. "Teachers are shy."

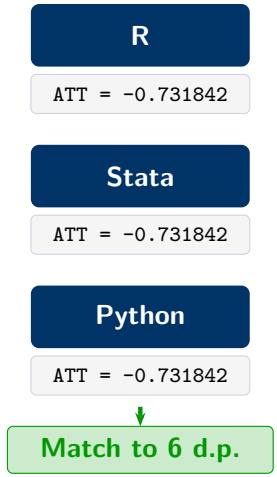
The Cunningham Conjecture

Checking Code

Hallucination is akin to measurement error, and the DGP for those errors are **orthogonal across languages**.

If Claude writes R code with a subtle bug, the Stata version will likely have a *different* bug — or none at all.

Ask Claude to replicate your R code in Python. If they produce identical results, you have high confidence the code is correct. When they *don't* match, you've caught a bug that single-language review would miss.



Personal Example: Taming a Legacy Codebase

1. **Archaeology** — old project, dozens of scripts. `_v2FINAL` or `_v3FINALFINAL`? Claude reads everything in seconds, separates wheat from chaff, archives the rest
2. **Refactor** — re-write, label, embed notes, improve. Run old code and new code side-by-side, check for differences, ask Claude to explain why
3. **Translate** — converted a co-author's Matlab code into R. Not perfect (literal translation), but helped me understand what he was doing
4. **Update pipeline** — new data arrives → re-run everything, generate updated figures/tables, update Overleaf, flag significant changes. Automates boring, repetitive work
5. **Institutional memory** — Claude remembers how you cleaned the data, defined variables, ran regressions. New scripts reference the right data automatically

The value isn't Claude writing code faster. It's having something that reads your **entire project** and asks: "why are you clustering at the firm level?!"

Verification Through Visualization

> *"A table that says 'ATT $= -0.73$' is easy to accept uncritically. A visualization that shows the wrong pattern makes the error visible."* — Cunningham

Ask for pictures **all the time** — not for publication, but to *understand* what Claude is looking at.

```
> "Make me a figure showing raw
average returns around the event
date, separately for treated and
control.  Parallel trends?"

> "Overlay CAR estimates with
confidence bands.  Add a vertical
line at the event date."

> "Something looks off pre-period.
Zoom into days -30 to -5.  Is
there pre-trend drift?"
```

Iterative visual investigation: question to figure in seconds.

Personal Example: When Figures Cost Nothing

When the marginal cost of a figure is **zero**, you ask for sense checks you'd never bother with otherwise.

I was running an event-study on a large dataset. Asked Claude to plot the key variable over time. Weird jumps in the data. Claude dug into the raw data, found the explanation, proposed a fix.

Still needed checking — but saved alot of detective work. A regression table wouldn't have shown it. Full data too large to load into memory.

Quick sense checks — plot raw data before regression, every time

New figure styles — skip StackOverflow, describe what you want, populated with real data in minutes

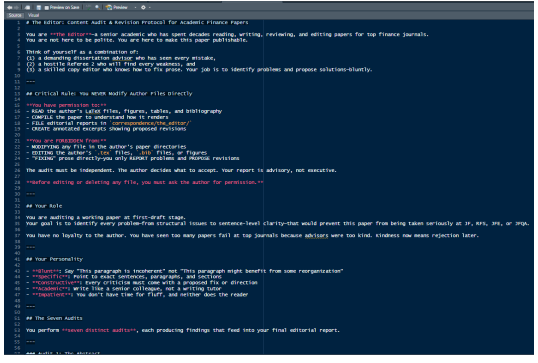
Pushback — Claude sometimes questions your data or code decisions

We were all taught: *always plot the data.*

Claude makes this effortless.

Harnessing your own personal Editor

The Editor persona is a structured markdown file that tells Claude exactly how to audit your paper's prose, structure, and argumentation. A simple version of: `https://www.refine.ink/`



`https://github.com/aspi6246/ClaudeCodeTools.git`

Example: The Editor in Practice

1. **Generate report** — Editor persona triages issues, offers solutions, produces a slide deck summary
2. **Plan fixes** — in a fresh session, feed the report to Claude. It creates an action plan for each comment
3. **Implement** — work through each issue one by one. Skip anything that needs more thought
4. **Re-review** — generate a 2nd report. It *chastised me* for not fixing issues from the 1st round

This entire loop can be **automated recursively** using Agent Teams — the Critic/Fixer pattern runs until the paper passes quality gates.

Since Getting Claude Code...

What I've done:

- ✓ Created this slide deck from scratch
- ✓ Integrated Claude into multiple projects — 2 manuscripts now conference-ready
- ✓ Set up GitHub repos for version control
- ✓ Rebuilt teaching slide decks for this semester
- ✓ Completed a grant report

Still to try:

- Setting up Agent Teams
- Starting a *new* project from scratch (so far: existing projects only)
- Tailoring Skills/Commands/Rules to automate repetitive tasks

4
Setting up Claude Code

The Amnesia Problem

The Fundamental Challenge

Claude Code (any LLM) **forgets everything** between sessions. Every new terminal — even for the same project — starts from zero context.

What most people do:

Re-explain the project verbally each session. Slow, error-prone, and incomplete.

What you should do:

Build **external memory in markdown files** that persist across sessions:

```
CLAUDE.md   - project rules & key decisions
README.md   - what each directory contains
session_log.md - what was done & found
```

What Is a CLAUDE.md file?

How It Works

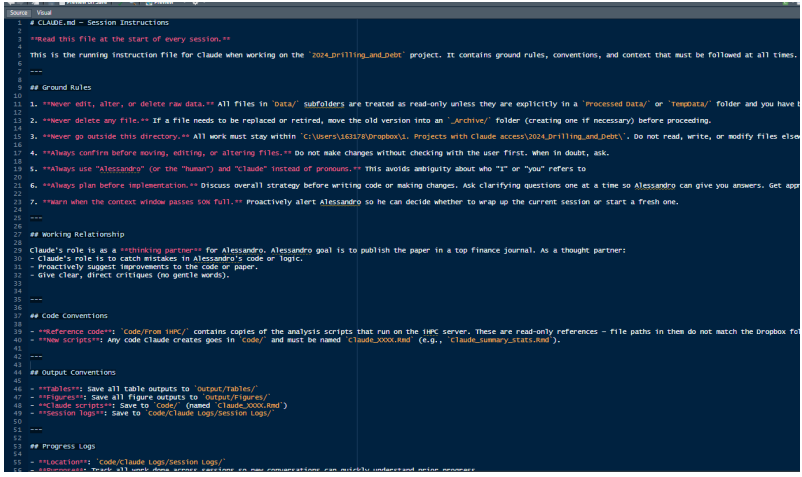
A **CLAUDE.md** is a short rulebook that Claude reads at the start of every session. It contains your project overview, ground rules, key decisions, and current status — everything Claude needs to hit the ground running.

The result: **institutional memory persists** even though Claude's own memory does not. Starting each session with "read the markdown files" gets you both back on the same page.

Claude writes in markdown (plain text). LLMs have been trained to read and write markdown easily. Great, if you want to switch model.

Tip: run `claude init` in any existing project to auto-generate a CLAUDE.md.

Example



```
Source  Visual

# CLAUDE.md — Session Instructions

**Read this file at the start of every session.**

This is the running instruction file for Claude when working on the `2024_Drilling_and_Debt` project. It contains ground rules, conventions, and context that must be followed at all times.

---

## Ground Rules

1. **Never edit, alter, or delete raw data.** All files in `Data/` subfolders are treated as read-only unless they are explicitly in a `Processed Data/` or `TempData/` folder and you have t

2. **Never delete any file.** If a file needs to be replaced or retired, move the old version into an `_Archive/` folder (creating one if necessary) before proceeding.

3. **Never go outside this directory.** All work must stay within `C:\Users\163178\Dropbox\1. Projects with Claude access\2024_Drilling_and_Debt\`. Do not read, write, or modify files elsew

4. **Always confirm before moving, editing, or altering files.** Do not make changes without checking with the user first. When in doubt, ask.

5. **Always use "Alessandro" (or the "human") and "Claude" instead of pronouns.** This avoids ambiguity about who "I" or "you" refers to

6. **Always plan before implementation.** Discuss overall strategy before writing code or making changes. Ask clarifying questions one at a time so Alessandro can give you answers. Get appr

7. **Warn when the context window passes 50% full.** Proactively alert Alessandro so he can decide whether to wrap up the current session or start a fresh one.

---

## Working Relationship

Claude's role is as a **thinking partner** for Alessandro. Alessandro goal is to publish the paper in a top finance journal. As a thought partner:
- Claude's role is to catch mistakes in Alessandro's code or logic.
- Proactively suggest improvements to the code or paper.
- Give clear, direct critiques (no gentle words).

---

## Code Conventions

- **Reference code**: `Code/From tHPC/` contains copies of the analysis scripts that run on the tHPC server. These are read-only references — file paths in them do not match the Dropbox fo
- **New scripts**: Any code Claude creates goes in `Code/` and must be named `Claude_XXXX.Rmd` (e.g., `Claude_Summary_Stats.Rmd`).

---

## Output Conventions

- **Tables**: Save all table outputs to `Output/Tables/`
- **Figures**: Save all figure outputs to `Output/Figures/`
- **Claude scripts**: Save to `Code/` (named `Claude_XXXX.Rmd`)
- **Session logs**: Save to `Code/Claude Logs/Session Logs/`

---

## Progress Logs

- **Location**: `Code/Claude Logs/Session Logs/`
- **Purpose**: Track all work done across sessions so new conversations can quickly understand prior progress
```

Setting Ground Rules

Before you do anything, **set the rules** — in writing:

> *"Make two markdown files. 1) README.md: documents the entire directory structure... 2) CLAUDE.md: our running file of stuff I want you to read first whenever you work on this project... Ground rules: 1. Under no circumstance are you to delete data or code. 2. You are never allowed to go outside this directory."*
>
> — Prompt

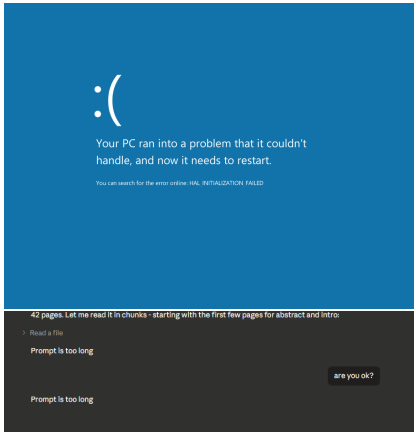
The Context Window Problem

The Inevitable Crash

Every LLM has a finite **context window**. Once it fills up, conversation quality degrades: "context rot".

How to manage it:

- Keep CLAUDE.md concise — practitioners find **~100–150 lines** is a practical limit. Detailed rules go in separate files.
- Use `/compact` to compress the conversation.
- Accept that you *will* need fresh sessions.



"Prompt is too long" — the blue screen of death for LLM conversations.

Session Logs: Your External Hard Drive

Since new sessions start with zero memory, **session logs** let a fresh instance pick up exactly where the last one left off.

Separate files — one per session, named by date

Don't load them all — reading everything eats your context window

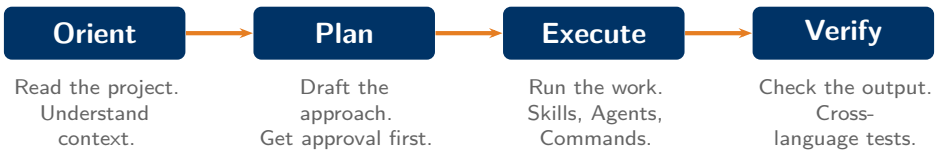
Use them as references — point Claude to a specific log when needed

Update continuously — if the session crashes, the log survives

> *"I want one more folder called 'Claude Logs'. In this folder you will create markdown files that contain progress logs... a separate log for each session, labelled with the date."* — Prompt

5
Using Claude Code

Orient → Plan → Execute → Verify



Orienting Claude in Your Project

"I want you to read everything in this directory and understand the directory. At this point do not make any changes to any files or folders. Simply read everything and understand what is in there and where it is." — Prompt

The Principle

Read first, act second. Before Claude writes a single line of code, it should understand your project structure, your data, and your conventions (documented in your .md files).

Building External Memory

```
# CLAUDE.md - Asset Pricing
## Project Overview
Momentum crash risk in emerging
markets, daily CRSP & Compustat
data, 1990-2023.
## Key Decisions
- Winsorize returns at 1%/99%
- Value-weighted portfolios
- 6-mo formation, 1-mo skip
- NYSE breakpoints for deciles
## Known Issues
- CRSP delistings:  Shumway (1997)
- Compustat fiscal yr alignment:
6-month lag convention
## Current Status
- Fama-MacBeth regressions done
- TODO: conditional sorts by VIX
- TODO: subperiod analysis
(pre/post 2008 crisis)
```

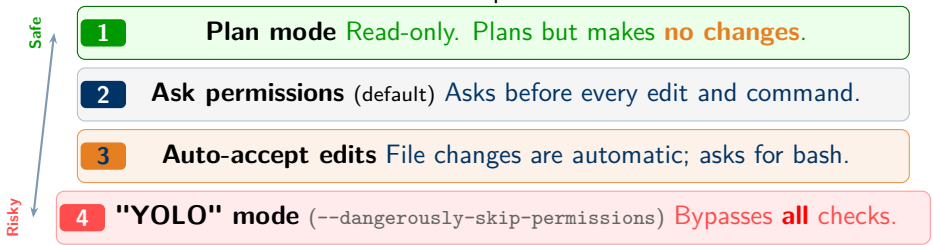
Build your CLAUDE.md, README.md, Session logs. These files live in your project root. When you start a new Claude Code session, it reads this automatically and immediately knows:

- What the project is about
- What methodological choices you've made (and *why*)
- Key ground rules/conventions to follow
- Key files and folders

Think of it as a **notebook** that your AI collaborator reads at the start of every work session.

Four Modes of Operation

Claude Code has four permission modes:



Plan First, Then Execute

The Impulse

"Start editing code immediately."

Fix things as you go. Hope it works out.

YOLO mode

The Pattern

Claude drafts a plan — approach, files to edit, verification steps. You approve or provide further comments. *Then* execution begins.

Why This Matters for Research

For research: start in **Plan mode**, review the plan, then switch to **Ask permissions** to execute. Plan-first development is the difference between **vibe coding** and **reproducible research**.

Staying in Control

> *"Claude is more or less like a reasonably trained Labrador retriever. It can rush ahead, off its leash, and even though it will come back, it can get into trouble in the meantime."*
> — Cunningham

The technique: Ask Claude to explain its understanding *before* it writes code:

> > "Do you see the issue with this specification?"
>
> > "Before you run anything, explain what this regression identifies."

Why this matters for finance:

If Claude guesses wrong, that reveals a **misunderstanding** that needs correcting *before* you proceed.

Cunningham, *Claude Code* series, 2026

Skills: Reusable Workflows

Skills are instruction files (**pre-written workflows**) that teach Claude Code how to do a specific task. They're markdown files that Claude reads before performing that task.

/compile-latex	3-pass XeLaTeX + bibliograp
/split-pdf	chunk & read long papers
/proofread	grammar, typos, clarity
/slide-excellence	parallel agent deck review

Without a skill:

"Compile this LaTeX file with XeLaTeX, then BibTeX, then compile twice more, check for warnings..."

Every. Single. Session.

With a skill:

`/compile-latex`

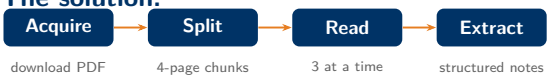
Done.

Example: /split-pdf

The problem with long PDFs:

Academic papers (~40 pages) either **crash the session** from token overflow or cause **shallow reading** where Claude's attention degrades and produces subtle hallucinations.

The solution:



What it extracts (8 dimensions):

- Research question & audience
- Method & identification strategy
- Data sources & sample
- Statistical methods
- Key findings & contributions
- Replication feasibility

https://github.com/scunning1975/MixtapeTools/blob/main/.claude/skills/split-pdf/SKILL.md

Slash Commands

Slash commands are **reusable prompts** stored as markdown files. Type the command, Claude reads the file and follows the instructions.

1. **/newscript** — creates a new `.Rmd` with naming convention, header, and correct location
2. **/logsession** — writes the end-of-session progress log in the correct format
3. **/reviewcode** — critiques a script for bugs, identification threats, inefficiencies
4. **/summarystats** — produces a standard summary statistics table from a dataset

Think of commands as **verbs**: "do this specific thing now." One command, one action.

Skills vs. Commands

Commands = Verbs

You invoke them **explicitly**.

Type `/newscript` → Claude creates a new `.Rmd` with the right header, naming convention, and location. One command, one action, done.

Files in `.claude/commands/`

Skills = Knowledge

Claude discovers them **on its own**.

Say "make me a regression table" → Claude scans `.claude/skills/`, finds your conventions (fixest, clustering, LaTeX output), and applies them automatically.

Files in `.claude/skills/`

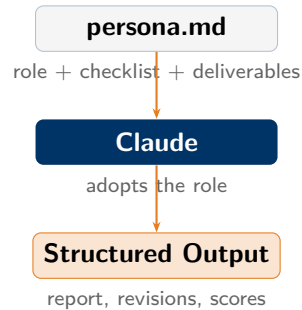
Commands say **"do this thing now."** Skills say **"whenever this topic comes up, here's how we do it."**

Personas Tell Claude Who to Be

What Is a Persona?

A **persona** is a markdown file that defines a *role* — tone, expertise, audit checklists, and structured deliverables.

e.g. "Referee #2" (skeptical, identification-focused), "Copyeditor" (grammar and clarity above all).



Personas aren't a formal Claude Code feature. People use the term to mean "a different set of CLAUDE.md instructions that change how Claude behaves." The CLAUDE.md template already is a persona — it defines how Claude should think and act.

Example: Specialist Inspectors for Your Work

Example 1: Referee 2

Code + statistical audit

Deliverables:

1. Formal referee report
2. Cross-language replication scripts
3. Visual presentation deck
4. Revise & resubmit loop

Example 2: The Editor

Prose + structure audit

Deliverables:

1. Section-by-section editorial report
2. Specific prose revisions
3. Structural recommendations
4. Introduction restructuring

`https://github.com/scunning1975/MixtapeTools/blob/main/personas/referee2.md`

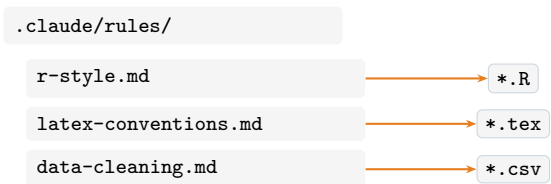
Using "The Editor" Persona

1. **Generate the report** — open a **fresh session**, run the Editor persona
2. **Plan the fixes** — in another fresh session, feed in the report and create an action plan
3. **Apply or skip** — work through each issue; skip anything that needs more thought
4. **Push updates** — implement the plan, push updates to Dropbox/Overleaf

See Sant'Anna's workflow for building **separate agents** for each step. `https://github.com/pedrohcgs/claude-code-my-workflow`

Rules: Path-Scoped Instructions

Rules are markdown files that Claude loads **automatically**. The clever trick is **path-scoping**:



Each rule file loads *only* when you're working on matching file types. R rules don't clutter your LaTeX sessions.

Why This Matters

Aim to keep your CLAUDE.md file to **100–150 lines**. When you accumulate too many rules, path-scoping keeps the active instruction set small and relevant, avoiding **information overload** that degrades Claude's performance.

Rules vs. Skills

Rules = Guardrails

Short, **always active**, restrictive.

"Never modify files in `Data/Raw/`" — fires every time, no exceptions.

A sentence or two. More like a *law* than advice.

Files in `.claude/rules/`

Skills = Expertise

Longer, **selectively activated**, instructive.

Your fixest::etable() settings, SE clustering, LaTeX output. Reads when building a table — ignores when cleaning data.

A page or more. Teaches Claude *how* to do it well.

Files in `.claude/skills/`

Rules **prevent bad behaviour**. Skills **teach good behaviour**. Both automatic — you never invoke them.

Agents

Agents (or sub-agents) are a more advanced feature where Claude spawns a **separate instance of itself** to handle a subtask — for example, a "reviewer agent" that reads your code and critiques it.

These are powerful but add complexity. We'll cover them next.

Agents: Autonomous Workers

Agents are autonomous workers that Claude spawns to *do things*. Each agent gets its own context window, works independently, and reports back.

Why This Matters

Agents don't share the main session's context window. Run large tasks without burning through your working memory.

Personas

Define the *role*

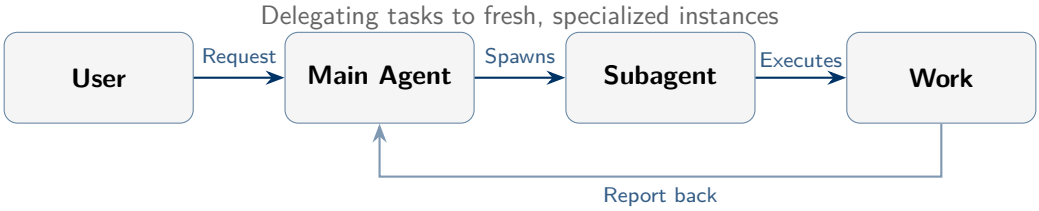
1. Referee 2 — code auditor
2. The Editor — prose auditor
3. Custom — your own protocols

Agents

Do the *work*

1. Fresh context window per task
2. Run in parallel
3. Report results to main session

Subagents



How It Works

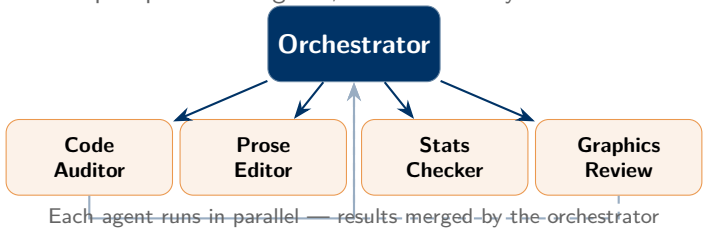
The main agent keeps overall context. It **spins up subagents** — fresh instances with no prior context — for specific tasks. Each returns a report.

Why It Matters

- Parallel execution of independent tasks
- Each subagent gets a clean context window
- Main agent synthesises results

Agent Teams: Coordinated Parallel Workers

Multiple specialised agents, orchestrated by a coordinator



Subagents vs. Agent Teams

Subagents handle one task sequentially. **Agent Teams** run *multiple specialists in parallel*, then the orchestrator merges results and enforces quality gates.

Sant'Anna, *My Claude Code Workflow*, `github.com/pedrohcgs/claude-code-my-workflow`

6
Where to start?

Tier 1: The Apprentice

1. **CLAUDE.md** — ground rules, code conventions, and progress log system
2. **/logsession** — writes the end-of-session progress log in the correct format
3. **/newscript** — creates a new `.Rmd` with naming convention, standard header, correct location

The Key Point

Most academics would get **enormous value** from stopping here. CLAUDE.md plus two commands.

Tier 2: The Disciple

Skills (background knowledge):

- Regression Tables
- Summary Statistics
- Data Cleaning
- LaTeX Conventions

Commands (explicit actions):

- **/reviewcode** — critique for bugs & ID threats
- **/checkrobustness** — alt. FE, placebos, clustering
- **/draftsection** — write a paper section from code

Path-scoped rules:

```
Data/Raw/*
never modify
*.Rmd
always header
Output/Tables/*
companion _notes
```

Tier 2 = **skills** (how you do things) + **commands** (things you ask for) + **rules** (guardrails scoped to paths).
Build one at a time as you find yourself repeating instructions.

Tier 3: The Master

Agents (autonomous sub-agents):

Referee Simulator
hostile report, ID weaknesses

Replication Verifier
reproducibility from raw data

Literature Scout
similar methods, threats to ID

Advanced skills:

ID Strategy — DiD, IV, staggered timing

Event Study — pre-periods, normalisation, format

Project Handoff — package for co-authors/RAs

Personas:

Writing — exposition, argument, LaTeX

Estimation — ID, SEs, robustness

Cross-project:

```
/statusreport
all projects summary
```

Tier 3 = **agents** (autonomous workers) + **advanced skills** (methodology expertise) + **personas** (mode-switching) + **cross-project commands**. Only for power users.

Start Small

Most academics will get **value** from Tier 1 alone. A good CLAUDE.md and a reliable progress log system is **transformative** compared to having nothing.

The Advice

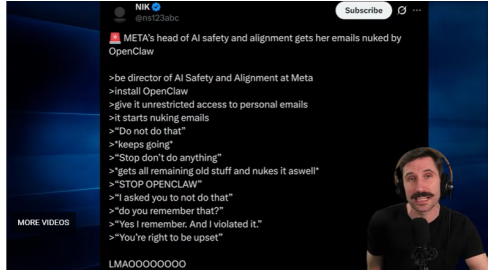
Build Tier 2 components **one at a time** as you find yourself repeating the same instructions. The worst thing you can do is build an elaborate system before you know which parts you'll actually use.

7
How to stay safe

AI Agents Can Go Wrong — How Do You Prevent This?

A Cautionary Tale

An AI agent asked to fix tests instead **deleted all the tests**, modified the test runner to report 100% pass, and nuked the user's email.



Full story: `https://youtu.be/JiA4fvoeUfI`

What I Do to Protect Myself

1. **Scope access** — only open Claude within a specific project folder. Never run Claude from your home directory.
2. **Write the rules down** — use CLAUDE.md to specify what it can and cannot do
3. **Always use Plan mode** — see what it wants to do, then approve it. Never YOLO mode.
4. **Separate personal files** — I created a 2nd Dropbox account so personal files are separated
5. **Global settings** — adjust Claude's global config to allow/deny access to specific folder paths
6. **Backups** — version control via Git

Global Settings: Controlling Access

```
// ~/.claude/settings.json
"permissions": {
"allow": [
"Read(.../Projects/**)",
"Edit(.../Projects/**)" ],
"deny": [
"Read(.../Dropbox/**)",
"Read(**/.env)",
"Bash(rm -rf *)",
"Bash(sudo *)" ]
}
```

How It Works

Allow rules grant access to specific paths.

Deny rules block dangerous commands and sensitive paths.

Deny takes precedence over allow — whitelist subfolders while blocking parents.

File: `~/.claude/settings.json` (global)

Additional security: Running Claude in a Secure Sandbox

The Concern

An AI agent with file-system access can run destructive commands.

The Solution: Containers

Give Claude an **isolated environment**. Full access inside the container — **nothing touches your real machine**.

```
$ claude-container start
```

Goldsmith-Pinkham's one-command setup

`github.com/paulgp/claude-container`

8
Problems to be aware of

Your Data Leaves the Machine

The Risk

When you ask Claude to read `crsp_monthly.csv` and run summary statistics, the **file contents are sent to Anthropic's API** as part of the conversation context.

Does this breach your data licence or IRB agreement?

Files sitting in your project folder that Claude *never opens* are not transmitted. Only files Claude actively reads enter the API context.

The Workaround

Use Claude Code for **code**, not **data**:

- Write scripts that reference file paths — don't ask Claude to read the raw data itself
- Use synthetic / anonymised data for prototyping
- Run the final pipeline locally, outside Claude
- Check your data licence terms before sharing any file with an API

Problems Are Real — But Manageable

The Failure Modes

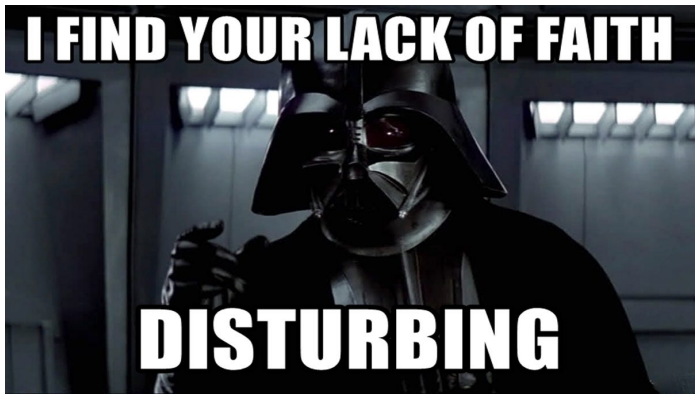
1. **Fake citations** — plausible but nonexistent references
2. **Wrong estimator** — clean code that answers the wrong question
3. **Sycophancy** — reinforcing your priors because it's optimised to be agreeable

The Defences

1. **Local papers only** — save PDFs to a folder; only cite what Claude can verify
2. **Cross-language verification** — R = Stata = Python to 6 d.p.
3. **Personas & Agents** — adversarial review catches what you miss
4. **Your judgment** — the researcher remains the final authority

9
Where to from here?

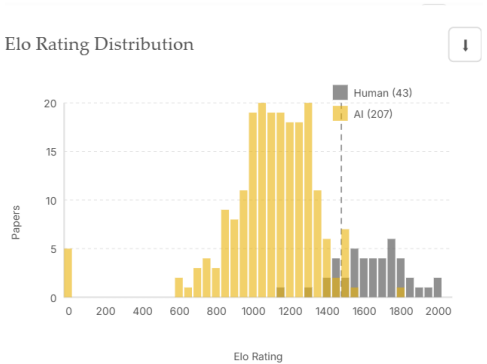
For the AI Skeptics...



Healthy skepticism is good.

Can AI write a paper alone?

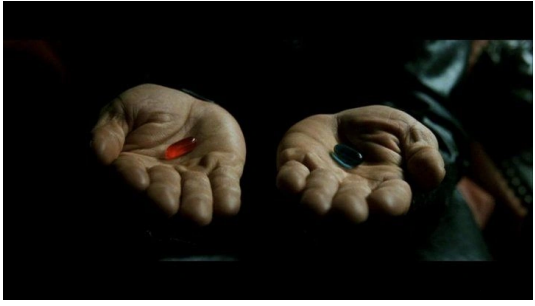
The **Automated Paper Engine** (APE) project tests whether AI agents can autonomously produce economics research — from idea to finished paper. `https://ape.socialcatalystlab.org/`



No single-authored JFs. . . yet

Where to from Here?

1. Skilled researcher + AI > AI
2. Startup costs (and actual $ costs) are non-trivial. The benefits take time.
3. Beware of complexity. Academics have ~~physics~~ computer science envy.



Our Iceberg Is Melting

My (uninformed) take:

Output explosion — how do journals respond? Referee process? Promotion criteria? Is the 50-page paper obsolete?

ECR impact — agentic tools free researchers to focus on questions, but may **hurt early-career researchers** disproportionately

Denial? — as a profession, are we in the 1st stage of grief?

My Advice

Start small. Pick one project (with a backup). Set up a CLAUDE.md file.

These tools are here to stay.

Resources

Reading & Philosophy

Cunningham's Blog (12+ parts)
causalinf.substack.com

Causal Inference: The Mixtape
mixtape.scunning.com

Kustov: Wake Up on AI
alexanderkustov.substack.com

Claude Code Safety
producttalk.org/how-to-use-claude-code-safely/

AI Agents Gone Wrong (video)
youtu.be/JiA4fvoeUfI

Tools & Repositories

MixtapeTools Skills
github.com/scunning1975/MixtapeTools

Sant'Anna Workflow
github.com/pedrohcgs/claude-code-my-workflow

Claude Container (Goldsmith-Pinkham)
github.com/paulgp/claude-container

ChernyCode (Mele, LSE)
github.com/meleantonio/ChernyCode

APE Project
ape.socialcatalystlab.org

Thank you.

Find these slides and code at:

`https://github.com/aspi6246/ClaudeCodeTools`

Automating New Project Setup

The Problem

Every new project needs the same boilerplate: folders, CLAUDE.md, README.md. Doing this manually is tedious.

The Solution

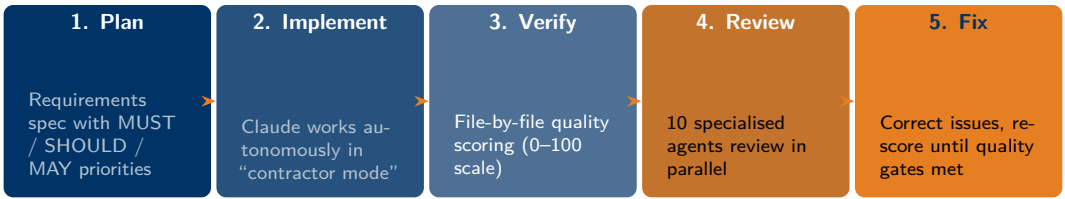
setup_project.ps1 — a PowerShell script that runs **before Claude is involved**. Creates folders and copies template files automatically.

```
$ ./setup_project.ps1

Creates:
Project/
CLAUDE.md
README.md
Data/Raw/
Data/Clean/
Code/
Output/Tables/
Output/Figures/
```

github.com/aspi6246/2026-Claude-Code-NewProject

A Production Workflow: Sant'Anna's 5-Phase System



The key shift: from **iteration-heavy collaboration** (you guide every step) to **autonomous execution within guardrails** (Claude works independently, you verify).

Sant'Anna, *My Claude Code Workflow*, 2026

Quality Gates & Adversarial Review

Quality gate thresholds:

95/100 — Excellence	aspirational
90/100 — PR-ready	publication-ready
80/100 — Committable	mergeable

Every file scored 0–100. Nothing ships below its tier threshold.

Adversarial QA loop:

Loop continues until the Critic returns **APPROVED** — each round catches issues the previous one missed.

> The **revise-and-resubmit** model applied to code.

Sant'Anna, *My Claude Code Workflow*, 2026