

SESSION I

AI Agents for Empirical Research

Using Claude Code in finance, accounting, and tax workflows

Core message

It will change how we do research. Use it so you do not fall behind.

Anecdote 1: one day, five paper projects

Personal motivation

I built a fairly elaborate Claude Code research-workflow setup: project memory, reusable prompts, Skills, audit routines, and a structured way to move from idea to implementation.

With that setup, I created five paper projects in one day.

Almost indistinguishable paper drafts with interesting questions. Clearly not publication-ready — but shocking as a result of a few hours.

Key idea

The surprising part was not that Claude wrote text. The surprising part was how much of the research-production workflow became compressible.

Anecdote 2: a master's thesis in an afternoon

The benchmark

Last semester, I supervised a very strong master's student. He invested a lot of time, wrote the best thesis I had supervised so far, and received a 1.0.

A few weeks ago, I replicated the entire thesis workflow in one afternoon.

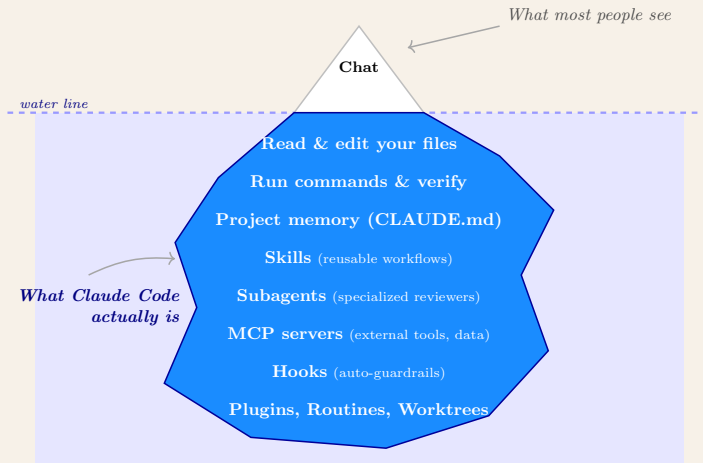
The quality was probably similar, perhaps slightly above, at least for the implementation and documentation layer.

Who has used AI?

Quick show of hands.

- ▶ Who has **used AI**?
- ▶ Who has **used Claude Code**?

The chatbot is only the tip



Most people stop at chat. The capability that matters lives below the surface.

What is Claude Code?

Plain definition

Claude Code is an agentic coding environment. It works *inside a project folder*, reads and edits files, runs commands, uses tools, remembers project instructions, and keeps session history.

Not just	But actually
chat about code	interaction with the project folder
copy-paste snippets	file edits, diffs, tests, and logs
one prompt	a session that can explore, plan, implement, and verify
generic advice	project-specific behavior through CLAUDE.md, Skills, settings

Follow-up: [Docs: How Claude Code works](#)

Today you learn

Agenda

A step-by-step tour through the building blocks of Claude Code — and together we build a small research project that uses all of them.

Foundations

- ▶ How to set up (VS Code)
- ▶ Sessions, models, modes
- ▶ `CLAUDE.md`: global + local
- ▶ Memory

Building blocks

- ▶ Agents and subagent teams
- ▶ Skills (reusable workflows)
- ▶ Plugins (shareable bundles)
- ▶ MCP servers (external tools, GitHub)
- ▶ Hooks, worktrees, routines

Key idea

Goal: by the end you have a mental model and a concrete starter setup for empirical research.

Section 2

How to start?

How to set up

My recommendation

Use **VS Code** with the Claude Code extension — file tree, editor, terminal, and Claude panel side by side. This is where I work day-to-day and what I recommend for this workshop.

- ▶ Install [VS Code](#) and enable the Claude Code extension.
- ▶ Open a fresh project folder and launch Claude Code from the panel.
- ▶ Make sure your subscription/account has Claude Code access (see next slide).

Alternative: the pure **terminal CLI** also works well — more direct, less visual. Useful once you are comfortable.

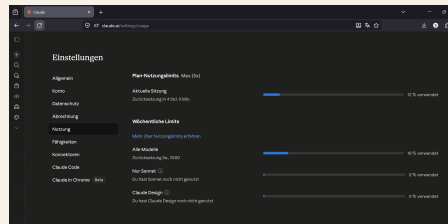
Follow-up: [Docs: Use Claude Code in VS Code](#)

Subscription plans

Plan	Price	Best for
Pro	\$20 / mo	individual researchers, occasional use
Max 5×	\$100 / mo	active research workflows
Max 20×	\$200 / mo	heavy daily use, autonomous pipelines
API	pay-as-you-go	teams, programmatic, custom integrations

- ▶ Start with **Pro**; upgrade to Max if you hit caps.
- ▶ Track your usage at claude.ai/settings/usage.

Follow-up: Pricing: anthropic.com/pricing



Live usage tracking on claude.ai

How sessions work

Concept	Practical meaning
Session	saved conversation tied to a project directory
Context	what Claude currently sees: prompt, files, logs, previous messages
Resume	continue a previous thread when the task is still the same
Branch	try a different path without mixing it into the old session
Clear/restart	useful when the task changes or context gets stale

Key idea

Good habit: one session, one task. Explore, implement, summarize, then restart or branch.

Follow-up: [Docs: Manage sessions](#)

Models, modes, and settings

Control	What it does
/model	choose or inspect the model for the session
Thinking	extended deliberation before answering — better on hard problems, slower and more tokens
Effort	how thorough Claude is per task (low / medium / high) — trades cost for quality
Permission mode	decide how often Claude asks before edits/commands
settings.json	persistent user/project/local configuration

- ▶ Default/Sonnet-style models are usually enough for most coding and research workflow tasks.
- ▶ Use stronger models or higher effort for architecture, debugging, and complex audits.

Further reading: Docs: <https://code.claude.com/docs/en/settings>,
<https://code.claude.com/docs/en/permission-modes>

Meta-skill: ask Claude how to use Claude

Default move: if setup, installation, package errors, or next steps are unclear, paste the situation into Claude Code and ask for the next concrete step.

```
I am using Claude Code for an empirical accounting project.  
Help me set up the environment step by step.  
Ask before running commands.
```

```
This command failed:  
<paste error>  
  
Explain the likely cause.  
Suggest the smallest safe next diagnostic.
```

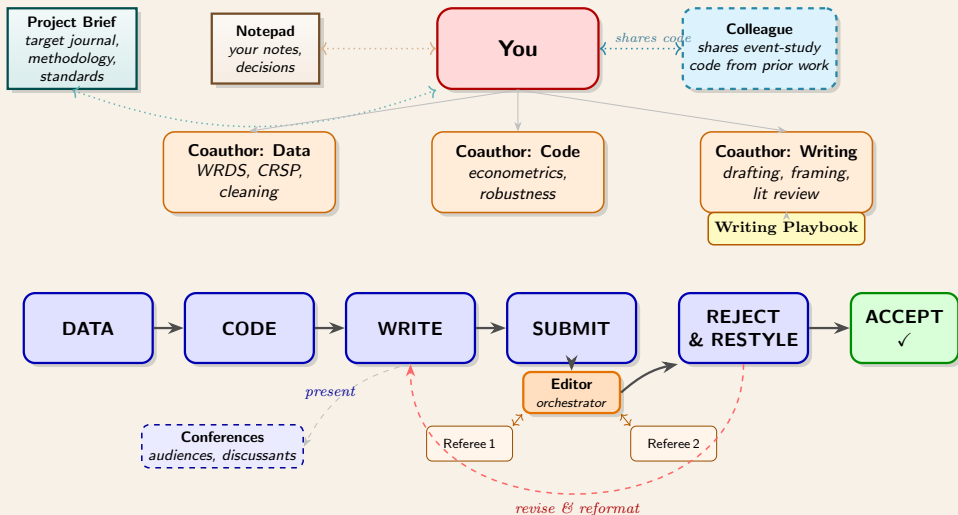
```
Inspect this project read-only.  
Tell me the next smallest useful step.  
Do not edit files yet.
```

Point: you do not need to know the whole workflow first; use Claude to navigate it.

Section 3

Claude Code features — with real world examples

Recap: how empirical research works today



Feature 1 — Global CLAUDE.md

What it replaces

YOU bring your own standards into every project: language, methodological preferences, tone. In the human team, this is invisible — it just *is* you. In Claude Code, it lives in one file that Claude reads at the start of *every* session.

Lives at	~/.claude/CLAUDE.md (Windows: C:\Users\<you>\.claude\CLAUDE.md)
Loaded when	every Claude Code session, regardless of project
Good content	“answer in English, be terse, prefer DiD over IV, double-check numerical claims, I am a Mannheim PhD in finance”
Bad content	project-specific data conventions (those go to the <i>local</i> CLAUDE.md)

Key idea

Global CLAUDE.md is who you are as a researcher — across every project, every session.

Follow-up: [Docs: Memory & CLAUDE.md](#)

Example in Claude Code: Global CLAUDE.md

Open the file: C:\Users\<you>\.claude\CLAUDE.md (create if it does not exist).

```
# Who I am
- PhD candidate, accounting & finance, University of Mannheim
- Working with Stata + R + Python, mostly R

# How to talk to me
- Answer in English unless I ask in German
- Be terse, no filler, no "I'd be happy to help"
- Be critical: challenge my assumptions before validating them
- Double-check numerical claims by re-reading the source file

# Methodological defaults
- Prefer DiD or event-study designs over IV when both fit
- Cluster standard errors by firm; mention if I should also cluster by year
- Top-5 citations preferred (JAR, TAR, JAE, RAST, CAR) when available
```

Key idea

This file is short, opinionated, and reused across every project you ever open.

Feature 2 — Local CLAUDE.md

What it replaces

The **Project Brief** sheet pinned to your desk: target journal, sample period, methodology, citation style. In Claude Code, it sits at the root of the project folder.

Lives at	<project>/CLAUDE.md (committed to Git, shared with coauthors)
Loaded when	only when Claude opens this project
Good content	target journal, sample period, exclusions, SE clustering rule, paper voice, file naming conventions
Bad content	personal preferences (those belong in the <i>global</i> CLAUDE.md)

Key idea

Local CLAUDE.md is the contract: what this specific project values and how it is run.

Example in Claude Code: Local CLAUDE.md

Create: earnings-call-event-study/CLAUDE.md

```
# Project: Earnings-Call Event Study
Target journal: TAR (fallback: JAR, RAST). Deadline: 2026-09-30.

## Sample
- CRSP + Compustat, 1995-2024
- Exclude SIC 6000-6999 (financials), 4900-4999 (utilities)
- Event window: [-1, +1] trading days; estimation [-250, -11]

## Methodology
- Standard errors clustered by firm AND year (Petersen 2009)
- Industry FE: Fama-French 48
- Robustness: 3-day window, alternative benchmarks

## Conventions
- Cite in Chicago Author-Date; Top-5 first when available
- Hypotheses numbered H1, H2a, H2b
- No raw-data edits; outputs go to output/tables/
```

Key idea

Local CLAUDE.md is committed to the repo and shared with coauthors.

Feature 3 — Memory

What it replaces

The **Notepad** you keep next to you with quick observations: “Anna prefers footnotes short”, “WRDS connection drops after 90 min”. In Claude Code, Claude maintains this file itself *automatically* as it works with you.

Lives at	~/.claude/projects/<project-hash>/memory/MEMORY.md
Loaded when	the project is opened
Written by	Claude itself, when it learns something surprising about you or the project
Good content	“user dictates audio — single words may be misrecognized”, “run Rscript build.R not R CMD BATCH”

Key idea

Memory is what Claude noticed; CLAUDE.md is what you told it.

Example in Claude Code: Memory

You usually do not write to Memory directly. You either let Claude learn passively, or you say something like:

```
Please remember for future sessions: when I dictate, single words  
may be misrecognized. Use context to infer the likely meaning  
instead of asking me to repeat.
```

Claude then writes a memory file (one per fact), and adds a one-line pointer to `MEMORY.md`. You can browse them with `/memory` or open the folder directly.

Key idea

Memory grows organically. Review and prune occasionally so old facts do not mislead.

Feature 4 — Agents

What it replaces

The **3 Coauthors** (data, code, writing). Each coauthor becomes a specialized *agent*: a separate Claude instance with its own prompt, its own tools, and its own narrow responsibility.

Lives at	.claude/agents/<name>.md
Each agent has	a name, a description, an allowed-tool list, and an instruction prompt
Empirical examples	data-agent (WRDS, cleaning), code-agent (econometrics, robustness), writer-agent (drafting, framing)
You call them	“use the data-agent to pull and clean the sample for me”

Key idea

An agent is a coauthor with a narrow job description and a small toolbox — no scope creep.

Follow-up: [Docs: Subagents \(general agent definitions\)](#)

Example in Claude Code: an agent definition

Create: `.claude/agents/data-agent.md`

```
---  
name: data-agent  
description: Pulls and cleans firm-level panel data for the event study.  
tools: Read, Write, Bash, Grep, Glob  
---
```

You are the data-coauthor for an earnings-call event study.

Responsibilities:

- pull CRSP daily returns and Compustat fundamentals
- restrict to SIC outside 6000-6999 and 4900-4999
- merge with CCM links; verify link validity on each event date
- write the cleaned panel to `data/intermediate/panel.parquet`
- report row counts, date coverage, and any dropped observations

Constraints:

- never modify `data/raw/`
- never expose credentials
- cite file paths and counts when reporting

Key idea

Once written, you invoke this agent by name — it is now part of your project.

Feature 5 — Skills

What it replaces

The **Writing Playbook** the writer keeps reaching for. A Skill is a reusable instruction package for a task you expect to repeat — a written checklist the agent can execute.

Lives at	<code>.claude/skills/<name>/SKILL.md</code>
You invoke it	with <code>/<skill-name></code> or Claude picks it automatically based on the description
Empirical examples	<code>event-study-audit</code> , <code>paper-code-consistency</code> , <code>restyle-paper</code>
Good Skill	repeatable, has a clear input/output, has a checklist or recipe
Bad Skill	one-off idea, vague judgment call, “find significant results”

Key idea

A Skill is a lab standard: the checklist you wish every RA would remember, written once.

Follow-up: [Docs: Skills](#)

Example in Claude Code: event-study-audit Skill

Create: `.claude/skills/event-study-audit/SKILL.md`

```
---  
name: event-study-audit  
description: Audit an empirical event-study project for common errors.  
---
```

Audit checklist:

1. Identify event source and event date.
2. Check timing: no look-ahead bias.
3. Check identifier links valid on event date.
4. Check estimation window vs event window.
5. Check duplicates, missing returns, delisting returns.
6. Check whether raw data are modified.
7. Check whether tables reproduce from scripts.

Default: read-only. Do not edit files unless explicitly asked.
Return: Issue | Evidence | File path | Severity | Suggested fix

Invoke: "Run the event-study-audit Skill on this project."

Feature 6 — Subagents (orchestrator pattern)

What it replaces

The **Editor + 2 Subagents**. The editor (orchestrator) dispatches the paper to two specialized subagents in parallel, each with a narrow lens. Subagents report *only* back to the editor, which then synthesizes and forwards the decision. Same pattern works with 2–4 subagents.

What changes	a Subagent is an agent specifically meant to be called <i>by</i> another agent — with its own isolated context window
Why it matters	long codebase scans, parallel reviews, independence between reviewer and implementer
Empirical examples	empirical-auditor (timing, identification), paper-code-consistency (tables vs. scripts), data-safety-reviewer (secrets, raw data)
Pattern	one main agent + 2–4 specialized subagents in parallel

Key idea

Subagents do not add autonomy — they add narrower responsibility and independent judgment.

Example in Claude Code: orchestrator + 3 referees

Prompt to the main agent:

Act as editor. Dispatch the current draft to three referees in parallel:

1. empirical-auditor -> check timing, identification, leakage
2. paper-code-consistency -> check whether paper tables match scripts
3. data-safety-reviewer -> check secrets, raw-data edits, Git hygiene

Each must return: Issue | Evidence | File:line | Severity | Suggested fix.

After all three return, synthesize the findings into a single prioritized report. Do not edit files.

Key idea

This is the orchestrator pattern: one editor agent + parallel specialized referees + a synthesis step.

Follow-up: [Sant'Anna writes about exactly this pattern](#)

Feature 7 — MCP servers

What it replaces

The **Colleague** who shares his event-study code with you. He gives Claude bounded access to an external system — another repo, a database, a calendar, an email inbox — through a clean, permissioned interface.

What it is	Model Context Protocol — a standard for letting Claude use external tools without pasting credentials into chat
Examples	GitHub MCP (other repos), WRDS MCP (research data), Google Drive, Gmail, Slack, Calendar
Install	<code>claude mcp add <name> "<command>"</code>

Key idea

MCP is how Claude reaches the outside world through approved doors — not through pasted secrets.

Follow-up: [Docs: MCP](#)

Example in Claude Code: MCP (GitHub)

Add the GitHub MCP server:

```
claude mcp add github "npx -y @modelcontextprotocol/server-github"
```

Then ask Claude:

```
Look at my GitHub repo "paulseidel/event-study-template".  
Find the latest commit that touches the abnormal-return function.  
Summarize what it changed and why.  
Do not copy anything into this project yet.
```

Key idea

The colleague who once emailed you a zip file is now an MCP connection that Claude can query whenever you ask.

Bonus 1 — Plugins

What they are

A Plugin is a *bundle* of Skills + Agents + Hooks + MCP configuration, packaged so a research group can install it with one command. The Sant'Anna workflow is essentially one big plugin.

- ▶ Lives at `.claude/plugins/<name>/` with a `plugin.json` manifest
- ▶ Namespaced: `/<plugin>:<skill>` — no collisions
- ▶ Distribution: forkable Git repos, eventual marketplaces
- ▶ Example bundle: “Mannheim empirical research” = `event-study-audit` + `replication-package-map` + `empirical-auditor` + `data-safety-reviewer` + WRDS MCP

Key idea

Plugins are how good research habits become *shareable infrastructure* instead of folklore.

Follow-up: [Docs: Plugins](#)

Bonus 2 — Hooks

What they are

Hooks are *shell commands that fire automatically* at defined moments: before a tool call, after a file edit, on session start. The agent does not have to remember — the system enforces.

- ▶ Configured in `settings.json` under `hooks`
- ▶ Common: log every Bash command, auto-format after every edit, block `.env` reads
- ▶ Research-specific: refuse writes to `data/raw/`, refuse to commit secrets
- ▶ Deterministic: prompts are advisory, hooks are enforced

Key idea

When you cannot trust the agent to remember a rule, encode the rule as a Hook.

Follow-up: [Docs: Hooks](#)

Bonus 3 — Worktrees and Routines

Worktrees

- ▶ Parallel Git checkouts in isolated directories
- ▶ Try *two different paper framings* simultaneously without conflict
- ▶ Each session has its own branch + folder
- ▶ Best for: exploration, robustness across specifications

Follow-up: [Docs: Worktrees](#)

Routines

- ▶ Cloud-side scheduled tasks (Claude.ai/code/routines)
- ▶ Run *without* your machine being on
- ▶ Example: nightly SSRN scan for new papers on your topic
- ▶ Best for: passive background work

Follow-up: [Article: Claude Code Routines](#)

Key idea

Worktrees parallelize *you*. Routines parallelize *time*.

What we have done

#	Human role in the team	Claude Code feature
1	YOU's standing preferences	Global CLAUDE.md
2	Project Brief	Local CLAUDE.md
3	Notepad	Memory
4	3 Coauthors	Agents
5	Writing Playbook	Skills
6	Editor + 2 Subagents	Subagents (orchestrator pattern)
7	Colleague (shares code)	MCP servers
<i>bonus</i> — (no human analog)		Plugins, Hooks, Worktrees, Routines

Key idea

Every human role in the research team now maps to a concrete Claude Code feature.

Section 4

Full workflow — Sant'Anna's setup

A working reference: Sant’Anna’s workflow

Why this one?

Pedro H. C. Sant’Anna (Emory; the DiD-methodologist behind Callaway–Sant’Anna 2021) has published an end-to-end Claude Code workflow for academic work. It is the most complete public template currently available.

Component	What it does
14 specialized agents	proofreading, visual audit, pedagogy, code review, fact-checking
Orchestrator pattern	one skill runs verify → review → fix → score internally
Quality gates 80/90/95	advisory scoring for commits, PRs, and excellence
15+ workflow patterns	plan-first, contractor mode, replication-first, devil’s advocate
Memory + plans + logs	cross-session persistence and compression-resistant task memory

Follow-up: [Workflow guide \(Sant’Anna, 2026\)](#)

How to implement it

1. **Fork the repository** from Sant'Anna's setup and clone it locally.
2. **Start Claude Code** in the cloned folder; paste the bootstrap prompt provided in the guide.
3. **Inspect the included files:** skills, agents, rules, hooks, CLAUDE.md.
4. **Adapt to your domain:** replace lecture-specific reviewers with empirical-research reviewers (event-study audit, replication map, paper-code consistency).
5. **Edit one skill at a time** and verify by running it on a toy project.
6. **Iterate over sessions** — the workflow was built over 6+ rounds of refinement.

Key idea

The point is not to copy his choices but to inherit a well-thought structure and then make it yours.

Section 5

Other sources

Where to keep learning

Official documentation

- ▶ Claude Code overview: <https://code.claude.com/docs/>
- ▶ Skills: <https://code.claude.com/docs/en/skills>
- ▶ Subagents: <https://code.claude.com/docs/en/sub-agents>
- ▶ Hooks: <https://code.claude.com/docs/en/hooks>
- ▶ MCP: <https://code.claude.com/docs/en/mcp>
- ▶ Plugins: <https://code.claude.com/docs/en/plugins>

Workflows from researchers

- ▶ Sant'Anna (Emory): psantanna.com/claude-code-my-workflow
- ▶ clo-author, claudeblattman, autoresearch, MixtapeTools
- ▶ AEA Data Editor Template (reproducibility)

Video introductions

- ▶ Claude Code Live (Anthropic webinars) — demos and best practices
- ▶ Tutorial DE: <https://www.youtube.com/watch?v=Lu95f1ZBIos>
- ▶ Tutorial EN/35 min: https://www.youtube.com/watch?v=vLwfguLz_qQ
- ▶ MCP focused: <https://www.youtube.com/watch?v=4fFPGvZ3gEQ>

Section 6

Review of our project

TODO: Section 6 placeholder

Live review of the autonomous research project.

Planned content:

- ▶ Show what the parallel-running project produced
- ▶ Walk through the generated paper draft / table / audit report
- ▶ Discuss what worked, what failed, what required human judgment
- ▶ Caveats: identification, interpretation, ethics, peer review

Section 7

Q&A

Thank you

Questions?

Paul Seidel
PhD Candidate in Finance
University of Mannheim

Slides + materials will be made available after the session.