# AI Coding Agents for Economists

From Messy Code to AER-Ready Replication Package

Alex Rieber (@AlexRieber) · Ulm University

2026-02-24

## Table of contents

**Who is this for?** Graduate students and post-docs who want to learn how to use AI coding agents (Claude Code, Gemini CLI, Codex CLI) from the terminal to organize, document, and package research code to professional standards.
**Prerequisites:** Docker container built and running from the **Docker Setup How-To**. A Claude Max subscription, Gemini API key, or OpenAI API key.
**Workshop materials:** The messy project folder, this handout, and all supporting files are at github.com/AlexRieber/Workshops.

# 1 Part 1: What You Will Build Today

## 1.1 The Task

You have inherited a **messy project folder** — the kind every researcher has at some point. It contains:

- R scripts with inconsistent names (`analysis_v2_FINAL.R`, `make figs.R`)
- No README, no master script, no documentation
- Data files mixed with code
- Hardcoded file paths and commented-out code
- No dependency management
- Exploratory scripts that should not be in a final package

Your job: **use an AI coding agent to transform this mess into a publication-ready replication package** that meets the standards of the American Economic Association (AEA).

## 1.2 The Data

The project uses the actual replication data from **Kessler & Roth (2025)**, "Increasing Organ Donor Registration as a Means to Increase Transplantation" (AEJ: Economic Policy, openICPSR 195641). The experiment tested whether changing how organ donor registration questions are framed (opt-in vs. active choice) affects real registration decisions.

The messy folder contains three Stata datasets (`.dta`) and two raw CSV files, mixed together with no organization:

| File | Description | Observations |
|---|---|---|
| `1_experiment_clean.dta` | Lab experiment waves 1–3 (real registrations) | 1,043 |
| `2_nextofkin_clean.dta` | Next-of-kin experiment (MTurk) | 803 |
| `3_dmv_quarterly_clean.dta` | State-level DMV registration panel (51 states) | 1,416 |
| `dmv_quarterly_raw.csv` | Raw quarterly DMV data | — |
| `nextofkin_raw.csv` | Raw next-of-kin Qualtrics data | — |

And seven R scripts of varying quality, plus a `notes.txt` with scattered reminders.

## 1.3 What a Finished Package Looks Like

By the end, you will have:

```
replication_package/
  README.md                  # Structured per AEA template
  LICENSE.txt                # MIT for code, CC-BY for data
  Makefile                  # One command reproduces everything
  master.R                   # Runs all scripts in correct order
  code/
      00_setup.R             # Install/load packages
      01_descriptive.R       # Table 1: summary statistics
      02_main_analysis.R     # Tables 2-3: treatment effects
      03_nok_analysis.R      # Table 4: next-of-kin results
      04_dmv_analysis.R      # Table 5: state-level DiD
      05_robustness.R        # Appendix robustness checks
      06_figures.R          # All figures
  data/
      raw/                   # Original CSV files
      README_data.md         # Variable descriptions
  output/
```

```
    tables/                 # LaTeX and CSV tables
    figures/               # PDF and PNG figures
  CLAUDE.md                 # (bonus) Agent instructions
```

---

# 2 Part 2: Meet the Agents

## 2.1 Claude Code

Claude Code is Anthropic's terminal-native coding agent. It reads files, writes code, runs commands, sees errors, and iterates — all autonomously.

**Key features for researchers:**

| Feature | What It Does |
|---|---|
| **CLAUDE.md** | A Markdown file the agent reads on startup. Your "briefing document." |
| **`/plan`** | Enters planning mode — the agent drafts a step-by-step approach for your approval before executing. |
| **`/commands`** | Lists available slash commands. |
| **`/skills`** | Auto-discovered project-specific skills (from `.claude/commands/`). |
| **`/insights`** | Shows things the agent has learned about your project during this session. |
| **`/init`** | Auto-generates a CLAUDE.md by scanning your project. Great for getting started. |
| **Permission modes** | Control what the agent is allowed to do without asking. |

**Starting Claude Code:**

```
# Inside your Docker container:
claude                          # interactive mode
claude --model opus --verbose   # use Opus model, show thinking
claude --dangerously-skip-permissions  # full autonomy (use with care!)
```

**Useful keyboard shortcuts (inside Claude Code):**

| Shortcut | Action |
|---|---|
| `Ctrl+O` | Toggle verbose output (shows thinking) |
| `Esc` | Cancel current generation |
| `Ctrl+C` | Cancel / exit |

## 2.2 Gemini CLI

Gemini CLI is Google's terminal coding agent. It follows a similar pattern: reads instructions, executes commands, iterates on errors.

**Key features:**

| Feature | What It Does |
|---|---|
| **GEMINI.md** | Equivalent of CLAUDE.md — instructions read on startup. |
| **`/plan`** | Enters planning mode (added in v0.29). |
| **`/memory`** | Manage persistent project memories. |
| **Sandbox mode** | `gemini -s` runs in a sandboxed environment. |
| **1M token context** | Gemini's huge context window (1 million tokens) handles very large codebases. |
| **Free tier** | 1,000 requests/day with Gemini API key (no subscription needed). |

**Starting Gemini CLI:**

```
# Inside your Docker container:
gemini                    # interactive mode
gemini -s                 # sandboxed mode (safer)
```

### 2.3 Codex CLI

Codex CLI is OpenAI's open-source terminal coding agent. Written in Rust, it features OS-level sandboxing and a hierarchical instruction file system.

**Key features:**

| Feature | What It Does |
|---|---|
| **AGENTS.md** | Equivalent of CLAUDE.md — hierarchical discovery from root to project directory. |
| **Sandbox** | OS-level sandboxing via Seatbelt (macOS), Landlock + seccomp (Linux). Network disabled by default. |
| **`--full-auto`** | Full autonomy mode (applies all changes without asking). |
| **Profiles** | Named configurations in `~/.codex/config.toml` for different workflows. |
| **Open source** | Apache 2.0 license, community-extensible. |

**Starting Codex CLI:**

```
# Inside your Docker container:
codex                          # interactive mode (suggest approval policy)
codex --full-auto              # full autonomy (applies all changes)
codex --full-auto --model o4-mini   # specific model + full autonomy
```

### 2.4 Feature Comparison

| Capability | Claude Code | Gemini CLI | Codex CLI |
|---|---|---|---|
| Instructions file | `CLAUDE.md` | `GEMINI.md` | `AGENTS.md` |
| Planning mode | `/plan` | `/plan` (v0.29+) | Prompt-based |
| Permission control | 3 modes + settings.json | `-s` sandbox + `--approval-mode` | 3 policies + OS sandbox |
| Slash commands | `/plan`, `/skills`, `/compact`, `/cost` | `/plan`, `/memory`, `/compress` | Not available |

| Capability | Claude Code | Gemini CLI | Codex CLI |
|---|---|---|---|
| Custom commands | `.claude/commands/` folder | `.gemini/commands/` (TOML) | Not available |
| Model selection | `--model opus/sonnet/haiku` | Uses configured Gemini model | `--model o4-mini/o3` |
| Context management | `/compact` (summarize & continue) | `/compress` + 1M token window | Automatic checkpointing |
| Session memory | Auto within session | `/memory` (persistent) | Auto within session |

Which agent should I use?

All three agents can complete today's exercise. **Choose the one you have access to** (Claude Max subscription, Gemini API key, or OpenAI API key). If you have multiple, try one for the first half and switch to compare the experience. Gemini CLI has a generous free tier (1,000 requests/day).

---

# 3 Part 3: Instruction Files — Your Agent's Brain

## 3.1 What Are Instruction Files?

Each agent reads a Markdown file from your project root on startup:

| Agent | File | Discovery |
|---|---|---|
| Claude Code | `CLAUDE.md` | Project root, then parent directories up to ~ |
| Gemini CLI | `GEMINI.md` | Project root |
| Codex CLI | `AGENTS.md` | Hierarchical: root `~/.codex/AGENTS.md` → parent dirs → project dir (all merged) |

Think of these as a **detailed briefing** you hand to a new research assistant on their first day. The agent reads them automatically — you never need to paste them into the conversation.

## 3.2 Why Instruction Files Matter

Without an instruction file, the agent will:

- Not know the project's goals or context
- Make arbitrary decisions about file organization
- Use whatever coding style it prefers
- Not know your field's conventions (AER formatting, clustering choices, etc.)

With a good instruction file, the agent will:

- Follow your exact specifications
- Use the right packages and methods
- Produce output in the format you need
- Handle edge cases the way you want

## 3.3 Anatomy of a Good Instruction File

All three agents follow the same general template:

```
# Project Name

## Mission
One paragraph: what is this project and what should the agent achieve?

## Principles
Numbered list of non-negotiable rules.

## Project Structure
Show the expected directory layout.

## Tasks / Phases
What needs to be done, in what order.

## Technical Specifications
Packages to use, output formats, coding style.

## Known Issues / Pitfalls
Domain-specific gotchas the agent should watch for.

## Communication Protocol
What to do when stuck, where to log progress.
```

### 3.4 Best Practices for Instruction Files

These tips apply to CLAUDE.md, GEMINI.md, and AGENTS.md alike:

1. **Be specific, not vague.** "Use `modelsummary()` with `output = 'latex'`" beats "make nice tables."
2. **Show the expected directory layout.** A tree diagram removes ambiguity about where files go.
3. **State what NOT to do.** "Never delete files in `data/raw/`" prevents costly mistakes.
4. **Use emphasis for critical rules. Bold** or ALLCAPS for non-negotiable constraints — agents pay more attention.
5. **Keep it concise.** Long instruction files dilute important rules. Aim for 50–200 lines.
6. **Update as the project evolves.** Your instruction file is a living document — revise it as you learn what works.
7. **Use `/init` (Claude Code) to bootstrap.** Running `claude /init` auto-generates a CLAUDE.md by scanning your project. Edit the result to add your domain-specific rules.

> Agent-specific tips
>
> **Claude Code:** You can have multiple CLAUDE.md files at different levels. A `~/.claude/CLAUDE.md` sets global preferences (e.g., "always use tidyverse"), while a project-level CLAUDE.md adds project-specific rules. Both are loaded.
> **Gemini CLI:** Use `/memory add "always use fixest for regressions"` to persist preferences across sessions. Gemini also reads `GEMINI.md` from the working directory on startup.
> **Codex CLI:** AGENTS.md files are discovered hierarchically: `~/.codex/AGENTS.md` → parent directories → project root. Rules cascade from general to specific. Configure default approval mode in `~/.codex/config.toml`.

### 3.5 Writing Your CLAUDE.md for Today's Exercise

Here is a CLAUDE.md you can use for the replication package task. You should **read it carefully, understand each section, then customize it** to your preferences.

> Use an LLM to write your instruction file!
>
> The meta-move: **use the chatbot version (claude.ai, gemini.google.com, chatgpt.com) to draft the instructions for the terminal agent.** Tell it about your project, your field's conventions, and your preferences. Iterate: "Add a section about X" / "Make the output format more specific."

### 3.6 The GEMINI.md — Gemini CLI Pendant

Gemini CLI reads a `GEMINI.md` file on startup. The content is **identical in structure** to a CLAUDE.md. Here is the same example adapted for Gemini CLI:

> What changes between CLAUDE.md and GEMINI.md?
>
> **Nothing about the content.** Only the filename. The rules, structure, and specifications are exactly the same. You can literally copy one to the other. The same holds for Codex CLI's `AGENTS.md`.

```
# Write once, use everywhere:
cp CLAUDE.md GEMINI.md
cp CLAUDE.md AGENTS.md
```

### 3.7 Gemini CLI Extras: `/memory`

One unique feature of Gemini CLI is persistent memory across sessions. Use it to save preferences:

```
> /memory add "Always use fixest for regressions, not lm()"
> /memory add "Standard errors: heteroskedastic-robust unless otherwise specified"
> /memory add "Output format: tables as .tex + .csv, figures as .pdf + .png at 300 DPI"
```

These memories persist even after you close and reopen Gemini CLI. They complement the GEMINI.md by storing preferences you discover during a session.

### 3.8 Codex CLI Extras: `config.toml`

Codex CLI uses `~/.codex/config.toml` for persistent configuration beyond AGENTS.md:

Start Codex with a specific profile: `codex --profile econ-replication`

---

## 4 Part 4: Permission Modes — When to Trust the Agent

### 4.1 Why Permissions Matter

By default, Claude Code asks for your approval before running **every** shell command. This is safe but slow. Understanding permission modes lets you find the right balance between safety and speed.

### 4.2 The Three Permission Modes

| Mode | How to Activate | What Happens |
|---|---|---|
| **Default** | `claude` | Agent asks before each bash command. You approve/deny individually. |
| **Allow-list** | `.claude/settings.json` | You pre-approve specific command patterns. Agent asks for everything else. |
| **Dangerously skip** | `claude --dangerously-skip-permissions` | Agent runs **everything** without asking. Full autonomy. |

### 4.3 Setting Up an Allow-list

Create `.claude/settings.json` in your project root:

This lets the agent run R scripts, create directories, copy files, and read/write files — without asking. It will still ask before doing anything not on the list (e.g., `rm`, `pip install`, `git push`).

### 4.4 When to Use Each Mode

| Situation | Recommended Mode |
|---|---|
| First time using an agent | **Default** — watch and learn what it does |
| Reorganizing code (today's exercise) | **Allow-list** — let it work, block destructive commands |
| You understand the task and trust the CLAUDE.md | **Dangerously skip** — maximum speed |
| Agent is accessing the internet or external APIs | **Default** — review each action |
| Working inside Docker (sandbox) | **Dangerously skip** is safer here — the container isolates you |

> **!** "Dangerously skip" + Docker = reasonable risk
>
> Inside a Docker container, the agent **cannot escape the sandbox**. Your host files are safe. This makes `--dangerously-skip-permissions` much less dangerous than running it on your bare laptop. For today's exercise inside Docker, this is a reasonable choice.

### 4.5 Gemini CLI Permissions

Gemini CLI offers several approval modes:

```
gemini                                  # default: asks before commands
gemini -s                              # sandbox mode: restricted environment
gemini --approval-mode=auto_edit        # auto-approve file edits, ask for
  ↪  shell commands
gemini --approval-mode=yolo             # full autonomy (like
  ↪  --dangerously-skip-permissions)
```

### 4.6 Codex CLI Permissions

Codex CLI uses OS-level sandboxing (Seatbelt on macOS, Landlock + seccomp on Linux) combined with three approval policies:

| Policy | Flag | Behavior |
|---|---|---|
| **Suggest** | (default) | Agent suggests changes, you apply them manually |
| **Auto-edit** | `--approval-mode auto-edit` | File edits applied automatically; shell commands need approval |
| **Full-auto** | `--full-auto` | Everything runs without asking. Network disabled by sandbox. |

```
codex                          # suggest mode (safest)
codex --full-auto              # full autonomy + OS sandbox
```

> **i** Codex CLI's unique sandbox
>
> Unlike Claude Code and Gemini CLI, Codex CLI uses **OS-level sandboxing** that restricts network access and filesystem writes even in full-auto mode. The agent can only write to the current project directory. This makes `--full-auto` safer than the equivalent modes in other agents.

---

# 5 Part 5: Planning Mode — Think Before You Act

## 5.1 Why Planning Matters

Complex tasks benefit from planning before execution. Without planning, the agent might:

- Start reorganizing files before understanding the full scope
- Miss dependencies between scripts
- Create a structure that needs to be redone later

## 5.2 Using `/plan` in Claude Code

Type `/plan` in the Claude Code prompt to enter planning mode:

```
you> /plan

Please analyze the messy_project folder and create a plan to
reorganize it into an AER-compliant replication package.
```

In planning mode, the agent will:

1. **Read all files** in the project
2. **Analyze dependencies** between scripts
3. **Draft a step-by-step plan** with specific actions
4. **Present the plan** for your review
5. **Wait for your approval** before executing anything

You can then:

- **Approve**: "Looks good, proceed."
- **Modify**: "Skip the robustness checks. Rename Table 5 to Table 4."
- **Reject**: "Start over with a different structure."

> When to use `/plan`
>
> Use `/plan` when:
> - The task has many steps (like today's exercise)
> - You want to review the approach before execution
> - The agent might make structural decisions you want to influence
> - You are not sure what the agent will do
>
> You do NOT need `/plan` for:
> - Simple, single-file edits ("fix the axis label in Figure 3")
> - Running existing scripts ("run master.R")
> - Questions about the project ("what packages does this use?")

## 5.3 Planning with Gemini CLI

Gemini CLI now supports `/plan` (added in v0.29). Use it the same way as in Claude Code:

```
> /plan
> Analyze all files in messy_project/ and create a detailed plan
  for reorganizing into an AER-compliant replication package.
```

If your Gemini CLI version does not have `/plan`, explicitly ask for a plan in your prompt:

```
> Before doing anything, analyze all files in messy_project/ and
  create a detailed plan for reorganizing into an AER-compliant
  replication package. Present the plan and wait for my approval
  before executing.
```

### 5.4 Planning with Codex CLI

Codex CLI does not have a `/plan` command. Include the planning request directly in your prompt:

```
codex "Analyze all files in messy_project/. Create a detailed plan
to reorganize into an AER-compliant replication package per AGENTS.md.
Present the plan and wait for my approval before executing."
```

---

## 6 Part 6: Slash Commands and Skills

### 6.1 Built-in Commands in Claude Code

Type `/commands` to see all available commands:

| Command | What It Does |
|---|---|
| `/plan` | Enter planning mode |
| `/commands` | List all slash commands |
| `/skills` | Show project-specific skills |
| `/insights` | Show what the agent learned about your project |
| `/status` | Show current session status |
| `/help` | General help |
| `/clear` | Clear conversation history |
| `/compact` | Compress conversation to save context |
| `/cost` | Show API usage for this session |
| `/model` | Switch model mid-session |
| `/config` | View/change configuration |

### 6.2 The `/insights` Command

As the agent works, it builds up knowledge about your project. Type `/insights` to see what it has learned:

```
you> /insights
```

Example output:

```
Project Insights:
- This project uses R with fixest for regressions
- Data is in CSV format (3 files, ~2000 total observations)
- The analysis covers a 2x2 experimental design (frame x info)
- Tables should use modelsummary with heteroskedastic-robust SEs
- The DMV analysis uses a two-way fixed effects DiD specification
```

These insights are built automatically. They help the agent maintain context, especially in long sessions where earlier conversation might be compressed.

### 6.3 Custom Skills with `.claude/commands/`

You can create **project-specific slash commands** by placing Markdown files in `.claude/commands/`:

```
mkdir -p .claude/commands
```

Example: Create a `/check-package` command:

Now you can type `/check-package` and the agent will execute this checklist.

Another example — a `/compare-output` command:

> **Skills are project-specific**
>
> Custom commands in `.claude/commands/` are scoped to your project. They travel with your repository — anyone who clones it gets the same slash commands. This is powerful for team workflows.

### 6.4 Gemini CLI Commands and Skills

Gemini CLI has its own set of commands:

| Command | What It Does |
|---|---|
| `/plan` | Enter planning mode (v0.29+) |
| `/memory add "..."` | Add a persistent project memory |
| `/memory show` | Show all saved memories |
| `/memory clear` | Clear all memories |
| `/compress` | Compress conversation to save context |

**Custom commands in Gemini CLI** use `.gemini/commands/` with TOML files:

### 6.5 Codex CLI Equivalent

Codex CLI does not have slash commands. Pass instructions directly:

```
# Run a one-off command
codex "Verify that the replication package is AER-compliant.
Check README, all scripts, and run master.R."

# Or pipe from a file
codex < prompts/check_package.txt
```

Codex CLI uses profiles in `~/.codex/config.toml` for different workflows:

Activate a profile: `codex --profile econ`

---

## 7 Part 7: The AER Replication Package Standard

### 7.1 Why Standards Matter

The American Economic Association requires all published papers to include a replication package deposited on openICPSR. The AEA Data Editor reviews every package before publication. Understanding these standards is essential for any economist submitting to AEA journals.

### 7.2 The AEA README Template

The AEA uses the Social Science Data Editors' README template. Your README must include these sections:

#### 7.2.1 Required Sections

1. **Overview**: Paper title, authors, one-paragraph summary of what the package contains
2. **Data Availability and Provenance Statements**: Where each dataset came from, how to access it
3. **Statement about Rights**: Certify you have permission to use and redistribute the data
4. **Summary of Availability**: All public / some restricted / none available

5. **Dataset List**: Table mapping data files to their sources
6. **Computational Requirements**: Software (with versions), packages (with versions), hardware, runtime estimate
7. **Description of Programs**: What each script does
8. **Instructions to Replicators**: Numbered steps to reproduce everything
9. **List of Tables and Programs**: Map every table/figure to the script that produces it
10. **References**: Full citations for all data sources

#### 7.2.2 Computational Requirements Checklist

Your README must specify:

☐ Software and version (e.g., R 4.4.2)
☐ All packages with versions (e.g., fixest 0.12.1)
☐ Operating system (or "platform-independent")
☐ Expected runtime (e.g., "< 10 minutes on a standard laptop")
☐ Storage requirements (e.g., "< 25 MB")
☐ Whether a random seed is set (and where)

### 7.3 Other Required Files

| File | Purpose |
|---|---|
| `README.md` | Documentation (see above) |
| `LICENSE.txt` | Terms of use (MIT for code, CC-BY 4.0 for data) |
| `master.R` or `Makefile` | Single entry point that runs everything |
| `00_setup.R` | Installs/loads all package dependencies |

> ! The cardinal rule
>
> "The replication package should reproduce the tables and figures, as well as any in-text numbers, by running code without manual intervention, starting from the raw data." — AEA Data Editor

### 7.4 Common Compliance Failures

The AEA Data Editor rejects packages for these reasons:

| Problem | How to Avoid |
|---|---|
| Missing or incomplete README | Use the template, fill every section |
| Code does not run | Test from a clean environment |
| Missing data citations | Cite every dataset in the manuscript's references |
| No master script | Create `master.R` that sources all scripts in order |
| Hardcoded paths | Use relative paths from the project root |
| Missing dependencies | List every package in `00_setup.R` with version |
| Exploratory files included | Remove scratch scripts, keep only final code |
| Manual steps required | Automate everything; the only manual step should be setting the root path |

# 8 Part 8: Hands-On Exercise

## 8.1 Step 1: Set Up Your Workspace

Open your terminal and enter the Docker container:

```
docker exec -it econ-replication-agent bash
```

Navigate to the messy project:

```
cd /workspace/messy_project
ls -la
```

You should see:

```
1_experiment_clean.dta
2_nextofkin_clean.dta
3_dmv_quarterly_clean.dta
analysis_v2_FINAL.R
dmv_analysis_v3.R
dmv_figures.R
dmv_quarterly_raw.csv
figure8_nok.R
make figs.R
nextofkin_raw.csv
nok_analysis.R
notes.txt
old_robustness_checks.R
quick_look.R
Table1_descriptive.R
```

> **Take a moment**
>
> Before starting the agent, **look at the files yourself**. Open a few scripts. Notice the problems: inconsistent naming, no structure, hardcoded paths, commented-out code, exploratory scripts mixed with analysis code. This is what the agent will fix.

## 8.2 Step 2: Create Your Instruction File

Copy the CLAUDE.md from Part 3 into the project folder:

```
nano CLAUDE.md
# (paste the content from Part 3, save with Ctrl+O, Enter, Ctrl+X)
```

If using Gemini or Codex, copy it under the right filename:

```
cp CLAUDE.md GEMINI.md     # for Gemini CLI
cp CLAUDE.md AGENTS.md     # for Codex CLI
```

Or use Claude Code's auto-generate feature to bootstrap one:

```
claude /init
```

## 8.3 Step 3: Start the Agent with Planning

### 8.3.1 Option A: Claude Code

```
claude --dangerously-skip-permissions
```

Then type:

```
/plan

Analyze all files in this project folder. Create a detailed plan to
reorganize everything into an AER-compliant replication package.
The package should follow the structure in CLAUDE.md. Show me the
plan before you start.
```

Review the plan. If it looks good:

```
Approved. Please execute the plan.
```

#### 8.3.2 Option B: Gemini CLI

```
gemini
```

Then type:

```
/plan

Read GEMINI.md and analyze all files in this directory. Create a
detailed plan to reorganize everything into an AER-compliant
replication package.
```

#### 8.3.3 Option C: Codex CLI

```
codex --full-auto
```

Then type:

```
Read AGENTS.md and analyze all files in this directory. Create a
detailed plan to reorganize everything into an AER-compliant
replication package. Present the plan and wait for my approval
before executing.
```

### 8.4 Step 4: Watch the Agent Work

The agent will typically:

1. Read all existing scripts to understand what they do
2. Create the directory structure (`code/`, `data/raw/`, `output/tables/`, `output/figures/`)
3. Copy data files to `data/raw/`
4. Refactor each script: clean code, fix paths, add proper output saving
5. Create `00_setup.R` with all package dependencies
6. Create `master.R` that sources all scripts in order
7. Write `README.md` following the AEA template
8. Create `LICENSE.txt` and `Makefile`
9. Test by running `master.R`
10. Fix any errors that come up

> **This takes 5–15 minutes**
>
> Depending on the model and the agent's approach, this may take a few minutes. Use `--verbose` (Claude Code) to watch the agent's thinking process in real time.

### 8.5 Step 5: Review the Results

After the agent finishes, check the output:

```
# Check directory structure
tree replication_package/
```

```
# Check that master.R runs cleanly
cd replication_package
Rscript master.R

# Check that tables were produced
ls output/tables/

# Check that figures were produced
ls output/figures/

# Read the README
cat README.md
```

### 8.6 Step 6: Iterate and Improve

The first pass will not be perfect. Use the agent to fix issues:

```
The README is missing the computational requirements section.
Please add it with the correct R version and package list.
```

```
Table 2 should use clustered standard errors at the session level,
not heteroskedasticity-robust. Please fix this.
```

```
The Makefile target "clean" should also remove the output/tables/
directory. Please update it.
```

This iterative refinement is where agents excel. Each fix takes seconds, not minutes.

---

## 9 Part 9: Advanced Techniques

### 9.1 Prompt Engineering for Agents

Good prompts for coding agents follow a **three-part structure**: context, objective, constraints.

| Weak Prompt | Strong Prompt |
|---|---|
| "Clean up this code" | "Refactor `analysis_v2_FINAL.R` into `code/02_main_analysis.R` with relative paths, modelsummary output to `output/tables/`, and heteroskedastic-robust SEs" |
| "Make a README" | "Write `README.md` following the AEA Social Science Data Editors template. Include all 10 required sections. List every R package with its version number." |
| "Fix the figures" | "In `code/06_figures.R`, change Figure 1 to use `theme_minimal()`, save as both PDF and PNG at 300 DPI in `output/figures/`" |

**Best practices for prompting:**

- **Be specific and verifiable.** The agent should know exactly what "done" looks like.
- **Reference files explicitly.** Use `@filename` (Claude Code) or full paths to point the agent at the right context.
- **Give context before objectives.** "This project replicates Kessler & Roth (2025). The data is in `data/raw/`. Refactor the analysis scripts."
- **State constraints up front.** "Use only relative paths. Do not install new packages. Output to `output/tables/`."

- **One task per prompt for complex work.** Don't ask the agent to reorganize AND extend AND debug in one message.

## 9.2 Context Management

Long sessions will eventually exceed the agent's context window. Each agent handles this differently:

### 9.2.1 Claude Code: `/compact`

```
/compact
```

The agent summarizes the conversation and continues with a compressed context. You can add instructions to the compaction: `/compact Focus on the regression results and any unresolved errors.`

**Best practice:** Use `/compact` proactively when you notice the conversation is long, rather than waiting for the agent to lose context. Between major subtasks, consider using `/clear` to start fresh (the agent will re-read CLAUDE.md).

### 9.2.2 Gemini CLI: `/compress` and the 1M token window

Gemini has a 1 million token context window, so context exhaustion is rarer. When it happens:

```
/compress
```

**Best practice:** Gemini's huge context means you can paste entire papers or large datasets directly into the conversation. Take advantage of this.

### 9.2.3 Codex CLI: Automatic Checkpointing

Codex CLI automatically checkpoints its state. If it hits a context limit, it resumes from the last checkpoint. No manual intervention needed.

## 9.3 The "Referee 2" Protocol

A powerful workflow for verifying replication packages, inspired by Scott Cunningham:

1. **Agent 1** builds the replication package (the author)
2. **Agent 2** reviews the package in a fresh session (the referee)

```
# Terminal 1: Build the package
claude --dangerously-skip-permissions
> (build the replication package)

# Terminal 2: Independent audit
claude --dangerously-skip-permissions
> You are a referee for the AEA Data Editor. Review the replication
  package in /workspace/replication_package/. Check:
  1. Does master.R run without errors from a clean R session?
  2. Does the README follow the AEA template?
  3. Are all tables and figures produced?
  4. Are all packages listed with versions?
  Report a detailed referee report with pass/fail for each criterion.
```

This catches issues that the building agent might overlook. The fresh context means the reviewing agent has no bias from the construction process.

> Cross-agent Referee 2
>
> For maximum robustness, use a **different agent** for the review. Build with Claude Code, review with Gemini CLI (or vice versa). Different models catch different issues.

### 9.4 Hooks (Claude Code)

Hooks are shell commands that execute automatically in response to agent events. They let you enforce project rules without relying on the agent to remember them.

Configure hooks in `.claude/settings.json`:

**Use cases for researchers:**

- Lint R code after every file write
- Prevent writes to `data/raw/` (enforce data protection)
- Log all shell commands for reproducibility auditing
- Auto-format code after edits

### 9.5 Multi-Agent Workflow

For ambitious projects, you can run multiple agents:

1. **Agent 1** (Claude Code): Reorganize code and create the replication package
2. **Agent 2** (Gemini CLI): Review the package for AER compliance and suggest fixes
3. **Agent 3** (Codex CLI): Run the package in its sandbox to verify it works in isolation
4. **You**: Final review and submission

This is especially powerful when comparing how different agents approach the same task.

### 9.6 Creating a Makefile

A good Makefile is the hallmark of a reproducible package:

Ask the agent to create this, or let it generate one automatically as part of the reorganization.

### 9.7 Using `.claudeignore` / `.geminiignore`

Like `.gitignore`, these files tell the agent to skip certain files and directories. Useful for large projects:

This speeds up the agent (fewer files to scan) and prevents it from accidentally modifying data files.

---

## 10 Part 10: Agent Commands Reference

### 10.1 Claude Code Quick Reference

| Action | Command |
|---|---|
| Start interactively | `claude` |
| Use Opus model | `claude --model opus` |
| Show verbose output | `claude --verbose` or `Ctrl+O` mid-session |
| Skip all permission prompts | `claude --dangerously-skip-permissions` |
| Auto-generate CLAUDE.md | `claude /init` |
| Enter planning mode | `/plan` |
| List commands | `/commands` |
| Show project-specific skills | `/skills` |
| Show project insights | `/insights` |
| Compress conversation | `/compact` |
| Show cost | `/cost` |

| Action | Command |
|---|---|
| Switch model | `/model` |
| Clear history | `/clear` |
| Get help | `/help` |

### 10.2 Gemini CLI Quick Reference

| Action | Command |
|---|---|
| Start interactively | `gemini` |
| Sandboxed mode | `gemini -s` |
| Auto-edit mode | `gemini --approval-mode=auto_edit` |
| Full autonomy | `gemini --approval-mode=yolo` |
| Enter planning mode | `/plan` |
| Add a memory | `/memory add "always use fixest"` |
| Show memories | `/memory show` |
| Compress conversation | `/compress` |
| Non-interactive (pipe) | `echo "prompt" \| gemini` |

### 10.3 Codex CLI Quick Reference

| Action | Command |
|---|---|
| Start interactively (suggest mode) | `codex` |
| Auto-edit mode | `codex --approval-mode auto-edit` |
| Full autonomy | `codex --full-auto` |
| Choose model | `codex --model o4-mini` |
| Use a profile | `codex --profile econ` |
| Non-interactive | `codex "your prompt here"` |

**Model context**

**Claude Code** supports models: `opus` (most capable), `sonnet` (balanced), `haiku` (fastest). For today's reorganization task, `sonnet` is a good default. Use `opus` if the agent struggles with complex decisions.
**Gemini CLI** uses the model configured via your API key. The free tier uses Gemini 2.5 Flash (1,000 requests/day). Gemini 2.5 Pro has a lower free-tier limit (100 requests/day).
**Codex CLI** defaults to `codex-mini-latest` (fast, optimized for CLI). Also supports `o4-mini` and `o3`. Configure in `~/.codex/config.toml` or pass `--model`.

---

# 11 Part 11: Troubleshooting

### 11.1 Common Issues

| Problem | Solution |
|---|---|
| Agent asks for permission on every command | Set up `.claude/settings.json` (Part 4) or use `--dangerously-skip-permissions` |
| Agent takes a wrong approach | Type "stop" or Ctrl+C, then redirect: "Instead of X, please do Y" |
| Agent runs out of context | Use `/compact` to compress the conversation |
| R package not installed | Ask the agent: "Install the fixest package" |
| Agent creates files in wrong location | Be specific in your prompt: "Create `code/01_descriptive.R` in the replication_package directory" |
| Agent modifies original data files | Add to CLAUDE.md/GEMINI.md/AGENTS.md: "Never modify files in data/raw/" |
| Gemini CLI does not read GEMINI.md | Ensure the file is in the current working directory when you start `gemini` |
| Codex CLI cannot access the network | This is by design (sandbox). Install packages beforehand or use `suggest` mode. |
| Agent generates code that does not run | Let it see the error — it will debug and fix autonomously |

### 11.2 When to Intervene

Let the agent work autonomously **unless**:

- It is about to delete original data files
- It is taking a fundamentally wrong approach (wrong statistical method)
- It has been stuck in a loop for more than 3 attempts
- It is making network requests you did not expect

### 11.3 Getting Help

- Claude Code documentation: docs.anthropic.com/en/docs/agents-and-tools/claude-code
- Gemini CLI documentation: github.com/google-gemini/gemini-cli
- Codex CLI documentation: github.com/openai/codex
- AEA Data Editor guidance: aeadataeditor.github.io/aea-de-guidance
- AEA README template: social-science-data-editors.github.io/template_README

---

## 12 Part 12: Checklist — Before You Submit

Use this checklist to verify your replication package:

### 12.1 Package Structure

☐ `README.md` exists in the root directory
☐ `LICENSE.txt` exists
☐ `master.R` or `Makefile` exists
☐ `code/` directory contains all analysis scripts
☐ `data/raw/` contains all data files
☐ `output/tables/` and `output/figures/` contain results
☐ No exploratory or scratch files remain

### 12.2 README Completeness

☐ Overview section with paper title and summary

☐ Data Availability Statement
☐ Dataset List table
☐ Computational Requirements (software, packages with versions)
☐ Description of Programs (every script listed)
☐ Instructions to Replicators (numbered steps)
☐ Table/Figure to Program mapping
☐ References for all data sources

### 12.3 Code Quality

☐ All paths are relative (no `C:/Users/...` or `/home/user/...`)
☐ `00_setup.R` installs/loads all required packages
☐ `master.R` runs all scripts in the correct order
☐ Code runs without errors from a clean R session
☐ No `install.packages()` calls in analysis scripts (only in `00_setup.R`)
☐ Commented-out code removed
☐ Random seeds set where applicable (`set.seed()`)

### 12.4 Reproducibility

☐ `Rscript master.R` completes without errors
☐ All tables in `output/tables/` match expected results
☐ All figures in `output/figures/` are produced
☐ `make clean && make all` reproduces everything (if Makefile exists)

Use the agent to verify!

Ask the agent to run through this checklist:

```
Please verify that the replication package meets all AER requirements.
Go through each item in the checklist and report pass/fail.
```

---

## 13 Glossary

| Term | Definition |
|---|---|
| **AEA** | American Economic Association — publisher of the AER, AEJ journals, and others |
| **AER** | American Economic Review — the flagship economics journal |
| **Agent** | An AI system that can plan, execute code, and iterate autonomously |
| **AGENTS.md** | Markdown instructions file read by Codex CLI, discovered hierarchically |
| **CLAUDE.md** | Markdown instructions file read automatically by Claude Code on startup |
| **Claude Code** | Anthropic's terminal-native AI coding agent |
| **Codex CLI** | OpenAI's open-source terminal AI coding agent with OS-level sandboxing |
| **Container** | A running instance of a Docker image — an isolated environment |
| **`/compact`** | Claude Code command to compress conversation and free context space |
| **Dangerously skip permissions** | Claude Code mode that runs all commands without asking for approval |
| **DiD** | Difference-in-Differences — a causal inference method comparing treatment/control groups before/after |

| Term | Definition |
|---|---|
| **Docker** | Platform for running isolated containers |
| **`--full-auto`** | Codex CLI flag for full autonomy (all changes applied without asking) |
| **Gemini CLI** | Google's terminal AI coding agent |
| **GEMINI.md** | Markdown instructions file read by Gemini CLI on startup |
| **Hooks** | Shell commands in Claude Code that execute automatically on agent events |
| **LPM** | Linear Probability Model — OLS with a binary dependent variable |
| **Makefile** | Build automation file — `make all` runs the full pipeline |
| **Master script** | A single script (`master.R`) that runs all other scripts in order |
| **openICPSR** | Repository where AEA replication packages are deposited |
| **`/plan`** | Command to enter planning mode (Claude Code and Gemini CLI) |
| **Referee 2 protocol** | Workflow where a second agent independently audits a replication package |
| **Replication package** | The complete set of data, code, and documentation needed to reproduce a paper's results |
| **Sandbox** | Restricted execution environment (Docker, or OS-level in Codex CLI) |
| **`settings.json`** | Claude Code configuration file for permission allow-lists and hooks |
| **`/skills`** | Claude Code command to show project-specific custom commands |
| **Slash command** | A `/command` in Claude Code or Gemini CLI (e.g., `/plan`, `/compact`, `/memory`) |

**Listing 1** `CLAUDE.md`

```
# Replication Package Agent

## Mission
Transform the messy project folder into an AER-compliant replication
package. Reorganize code, write documentation, create a master script,
and produce all tables and figures in a clean, reproducible pipeline.

## Principles
1. Never delete the original data files. Copy them to data/raw/.
2. All code must run from the project root using relative paths.
3. Use R with tidyverse conventions. Load packages with library().
4. Output tables as both .csv and .tex (using modelsummary).
5. Output figures as both .pdf and .png (300 DPI).
6. Follow the AEA README template structure exactly.
7. Document every decision in the README.

## Project Structure
replication_package/
  README.md
  LICENSE.txt
  Makefile
  master.R
  code/
      00_setup.R
      01_descriptive.R
      02_main_analysis.R
      03_nok_analysis.R
      04_dmv_analysis.R
      05_robustness.R
      06_figures.R
  data/
      raw/
      README_data.md
  output/
       tables/
       figures/

## Technical Specifications
- R packages: tidyverse, fixest, modelsummary, ggplot2, marginaleffects
- Tables: modelsummary() with output formats "latex" and "csv"
- Figures: ggsave() with width=8, height=5, dpi=300
- Standard errors: heteroskedasticity-robust (vcov = "hetero")
- Regression tables: report coefficients with SEs in parentheses

## AER README Requirements
The README.md must follow the Social Science Data Editors template:
1. Overview (title, authors, data/software summary)
2. Data Availability Statement
3. Dataset List (file, source, provided yes/no)
4. Computational Requirements (software, packages, runtime)
5. Description of Programs (what each script does)
6. Instructions to Replicators (step-by-step)
7. Table of Tables/Figures → Programs mapping

## Communication Protocol
- If a script fails, log the error and continue with other scripts.
- After completing all tasks, summarize what was done and any issues.
```

**Listing 2** `GEMINI.md`

```
# Replication Package Agent

## Mission
Transform the messy project folder into an AER-compliant replication
package. Reorganize code, write documentation, create a master script,
and produce all tables and figures in a clean, reproducible pipeline.

## Principles
1. Never delete the original data files. Copy them to data/raw/.
2. All code must run from the project root using relative paths.
3. Use R with tidyverse conventions. Load packages with library().
4. Output tables as both .csv and .tex (using modelsummary).
5. Output figures as both .pdf and .png (300 DPI).
6. Follow the AEA README template structure exactly.
7. Document every decision in the README.

## Project Structure
replication_package/
  README.md
  LICENSE.txt
  Makefile
  master.R
  code/
      00_setup.R
      01_descriptive.R
      02_main_analysis.R
      03_nok_analysis.R
      04_dmv_analysis.R
      05_robustness.R
      06_figures.R
  data/
      raw/
      README_data.md
  output/
       tables/
       figures/

## Technical Specifications
- R packages: tidyverse, haven, fixest, modelsummary, ggplot2,
  ↪  marginaleffects
- Tables: modelsummary() with output formats "latex" and "csv"
- Figures: ggsave() with width=8, height=5, dpi=300
- Standard errors: heteroskedasticity-robust (vcov = "hetero")
- Data files are Stata .dta format - read with haven::read_dta()

## AER README Requirements
The README.md must follow the Social Science Data Editors template:
1. Overview
2. Data Availability Statement
3. Dataset List
4. Computational Requirements
5. Description of Programs
6. Instructions to Replicators
7. Table/Figure → Program mapping
```

**Listing 3** `~/.codex/config.toml`

```
model = "o4-mini"
approval_policy = "auto-edit"

[profiles.econ-replication]
model = "o3"
approval_policy = "full-auto"
```

**Listing 4** `.claude/settings.json`

```
{
  "permissions": {
    "allow": [
      "Bash(Rscript:*)",
      "Bash(R:*)",
      "Bash(mkdir:*)",
      "Bash(cp:*)",
      "Bash(mv:*)",
      "Bash(cat:*)",
      "Bash(ls:*)",
      "Bash(make:*)",
      "Write(*)",
      "Read(*)"
    ]
  }
}
```

**Listing 5** `.claude/commands/check-package.md`

```
Verify that the replication package is AER-compliant:

1. Check that README.md exists and follows the template
2. Check that all scripts referenced in README exist
3. Run master.R and verify it completes without errors
4. Check that all output files (tables/, figures/) are produced
5. Verify the Makefile works: make clean && make all
6. Report any issues found
```

**Listing 6** `.claude/commands/compare-output.md`

```
Compare the output of the replication package with expected results:

1. List all files in output/tables/ and output/figures/
2. For each table, show the key coefficients and standard errors
3. Flag any tables that are missing or empty
4. Summarize: how many tables/figures were produced vs. expected
```

**Listing 7** `.gemini/commands/check_package.toml`

```
description = "Verify AER compliance of the replication package"
prompt = """
Verify that the replication package is AER-compliant:
1. Check README.md exists and follows the template
2. Check all scripts referenced in README exist
3. Run master.R and verify it completes without errors
4. Check that all output files are produced
5. Report pass/fail for each item
"""
```

**Listing 8** `~/.codex/config.toml`

```
[profiles.econ]
model = "o4-mini"
approval_policy = "auto-edit"
```

**Listing 9** `.claude/settings.json`

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "command": "echo 'Running shell command...'"
      }
    ],
    "PostToolUse": [
      {
        "matcher": "Write",
        "command": "Rscript -e 'lintr::lint(\"$CLAUDE_FILE_PATH\")'"
      }
    ]
  }
}
```

**Listing 10** `Makefile`

```
.PHONY: all tables figures clean

all: tables figures

tables: code/00_setup.R code/01_descriptive.R code/02_main_analysis.R \
        code/03_nok_analysis.R code/04_dmv_analysis.R
  ↪  code/05_robustness.R
__Rscript code/00_setup.R
    Rscript code/01_descriptive.R
    Rscript code/02_main_analysis.R
    Rscript code/03_nok_analysis.R
    Rscript code/04_dmv_analysis.R
    Rscript code/05_robustness.R

figures: code/00_setup.R code/06_figures.R
__Rscript code/00_setup.R
    Rscript code/06_figures.R

clean:
__rm -rf output/tables/* output/figures/*
```

**Listing 11** `.claudeignore`

```
# Skip large data files the agent doesn't need to read
data/raw/*.csv
data/raw/*.dta
# Skip compiled output
output/
# Skip renv library
renv/library/
```