Lab 2 — From Spec to Code

AI-Assisted Programming for PhD Researchers

In Lab 1 you got the inherited script running and catalogued its bugs, but the loading and cleaning logic is still a tangle buried inside `legacy_analysis.py`. In this lab you replace it with a proper, testable module, driven by a **specification you write first**. The agent does the typing; the spec is your contract, and you verify the result against it rather than trusting the agent's report.

Budget about **45 minutes**. Continue in the same repository, on the state you reached at the end of Lab 1.

Task 1 — Write the Spec

Goal: write a short specification for the cleaning module before any code exists, so both you and the agent know exactly what "done" means.

Create the file `docs/spec-cleaning.md` and paste this skeleton into it:

```
# Spec: data loading and cleaning
Goal: load data/raw/measurements.csv into a clean table ready for analysis.
Module: src/pipeline/io.py — functions load_raw(path), clean(df).
Acceptance criteria:
- [ ] all three timestamp formats parse into one datetime column
- [ ] temperatures unified to °C (temp_unit column consulted, then dropped)
- [ ] humidity: "", "n/a", -99 → missing (NaN); decimal commas parsed
- [ ] wind: "calm" → 0.0; column numeric
- [ ] station names stripped of whitespace
- [ ] exact duplicate rows dropped; report how many
Out of scope: statistics, plotting, the CLI.
```

Every criterion is a checkbox because every criterion is a pass/fail test: no debate later about whether the code "handles missing values."

Now make it yours. Look back at what **you** found while reading the data in Lab 1 and add **one or two** more acceptance criteria of your own. For example, a guarantee on the resulting column types (`timestamp` is `datetime64`, `temperature` / `humidity_pct` / `wind_ms` are `float`), or that the row order is reset after duplicates are dropped. Write them as checkboxes in the same style.

> Checkpoint
>
> `docs/spec-cleaning.md` exists, contains the skeleton above verbatim, and has one or two extra acceptance criteria in your own words. No code has been written yet.

⚠ Explain it

1. The spec turns "–99 → NaN" instead of "delete rows where humidity is –99." Why is marking those readings as missing better than dropping the whole row? Think about the other columns in that same row: temperature, wind, timestamp.
2. Which one or two criteria did you add, and what did you observe in Lab 1 that made you add them? Point to the evidence, not a guess.

Task 2 — Interview, Then Plan

Goal: let the agent turn your spec into a step-by-step plan, but only after it has interrogated the spec for gaps.

Start the agent in the repository root and switch it to **plan mode** (press `Tab` to toggle). Plan mode lets it read and reason without editing anything.

```
opencode
```

Paste this prompt:

ℹ Prompt

Read `docs/spec-cleaning.md` and `AGENTS.md`. Before proposing a plan, interview me: ask clarifying questions until the spec is unambiguous. Then propose a step-by-step plan where every step ends in something I can verify.

The agent will ask questions before it plans. Answer them from what you know about the project. Expect questions like these, with sensible answers in the right column:

The agent asks…	Your answer
Which library for the data work: pandas, polars, plain `csv`?	pandas
How should the package be laid out?	a `uv` project with a `src/` layout; package name `pipeline`
Should `load_raw` do any parsing, or just read the file?	read only; all cleaning happens in `clean`
Keep `legacy_analysis.py`, or remove it?	delete it once the new module reproduces its loading behavior
Where do the summary statistics and plot go?	out of scope for now, cleaning only

When the plan comes back, do not approve it on sight. Review it against the three questions from the lecture:

- Does it cover **every** acceptance criterion in your spec?
- Does it invent scope you never asked for (statistics, a CLI, extra files)?
- Does **each step end in a check** you could actually run?

If any answer is "no," tell the agent what to change and have it revise the plan.

> Checkpoint
>
> The agent asked clarifying questions, you answered them, and you now have a plan whose steps each end in a verifiable result and whose scope is cleaning only: no statistics, no plotting, no CLI.

> Explain it
>
> 1. Name **one** question the agent asked that your spec should have answered on its own. Now go add that answer to `docs/spec-cleaning.md`: the spec, not just the chat, should carry it.
> 2. Of the three review questions above, which one caught a real problem in the agent's first plan (or would have, if the plan had drifted)? What did it catch?

Task 3 — Implement and Verify

Goal: let the agent build the module from the approved plan, then confirm the result yourself against the spec.

Switch the agent to **build mode** (`Tab`) and let it execute the plan. It will initialize the `uv` project, write a `pyproject.toml` that declares the `pipeline` package and its dependencies (pandas), and create `src/pipeline/io.py` with `load_raw` and `clean`.

When it reports it is done, verify. Do not take its word for it. Run:

```
uv run python -c "from pipeline.io import load_raw, clean; df = clean(load_raw('data/raw/measurements.csv')); print(df.dtypes); print(len(df))"
```

Check the output against your spec:

- `timestamp` is `datetime64`, and `temperature`, `humidity_pct`, `wind_ms` are `float64` (the type guarantees from your acceptance criteria).
- The temperatures are all in a plausible °C range (single digits to low tens for this station data), not a mix with the 40s–50s that raw °F readings show.
- The printed row count is **smaller** than the number of rows in the raw file: the exact duplicates are gone.

If anything is off, hand the mismatch back to the agent and point at the specific criterion it violated.

Checkpoint

`src/pipeline/io.py` exists, `uv run python -c ...` prints the expected dtypes and a row count below the raw count, and every acceptance criterion in your spec is satisfied.

Explain it

1. Open `src/pipeline/io.py` and find the line that converts Fahrenheit to Celsius. Take a raw row whose `temp_unit` is `F` (say a reading of `47.5`) and **compute the Celsius value yourself** with the formula on that line. Does your result land in the same range as the native °C readings? Show your arithmetic.
2. Find the line that drops duplicate rows. If two different sensors genuinely recorded the **exact same values** at the same timestamp, `drop_duplicates()` would remove one of them as a "duplicate." Explain in your own words why that happens, and argue whether it is acceptable for this dataset, given what a row represents. What would you change if it were not?

Task 4 — Retire the Legacy

Goal: remove the old script now that the module has taken over its loading job, and record the new commands for the next agent session.

Delete the legacy script:

```
git rm legacy_analysis.py
```

Then update `AGENTS.md` so its **Commands** section points at the new package: how to run code and add dependencies with `uv`, and the fact that loading and cleaning now live in `src/pipeline/io.py`. Keep it short, and keep the read-only `data/raw/` and explain-before-commit rules you wrote in Lab 1.

Commit the whole change together:

```
git add -A
git commit -m "Replace legacy loading with a spec-driven cleaning module"
```

Checkpoint

`legacy_analysis.py` is gone, `AGENTS.md` describes the `uv` package and the new module, and the work is committed. Your repository now matches the reference state for this lab.

If you fell behind, catch up with:

```
git fetch --tags && git checkout tags/lab-02-done -b catchup-02
```

Explain it

1. You deleted `legacy_analysis.py` and added the new module in the **same** commit. Why is that reasonable here, when in Lab 1 you were careful to keep unrelated changes in separate commits?
2. The next agent session will read the updated `AGENTS.md` before it reads any code. Which single line you just wrote there would save it the most wasted effort, and why?

If You Are Ahead

Your last acceptance criterion says "report how many" duplicates were dropped, but right now `clean()` drops them silently and reports nothing at all. Make that count a real, usable output.

1. **Spec first.** Add two acceptance criteria to `docs/spec-cleaning.md`: that `clean()` can optionally return a small report of how many fixes of each kind it applied (duplicates dropped, humidity values marked missing, `calm` winds set to zero, Fahrenheit rows converted), and that the default behavior (a plain cleaned frame) is unchanged so existing callers keep working.
2. **Then implement.** Have the agent add an optional `report` flag (or a second return value) that hands back those counts, driven by your two new criteria.
3. Verify by running `clean()` both ways and confirming the counts match what you can see in the raw data, and that the plain call still returns exactly the frame it did before.

If the counts do not match your own tally, the gap is in the code, not your arithmetic: find out which fix is miscounted.

Note

That's it for this lab. If anything is unclear, ask now — the next session builds on this state. If you fell behind, use the checkpoint tag from the last checkpoint box to catch up.

Bibliography