LLM Coding Workflow Workshop

2-Hour Hands-On Walkthrough

Razvan Marinescu — UC Santa Cruz

February 10, 2026

Goal

By the end of this session, each participant should be able to:

- Use LLMs in **Cursor** to write new code and modify existing code.
- Use an **LLM CLI agent** (Codex or Claude CLI) for file/system operations.
- Build a small local data workflow: load CSV data, compute summaries, and generate plots.
- Produce a short document artifact (Markdown and optional LaTeX table) generated from local data.

Materials in this folder

- `data/students_scores.csv`, `data/model_runs.csv`
- `src/starter_analysis.py`
- `src/text_utils.py`
- `src/generate_report.py`
- `src/plot_metrics.py`
- `tests/test_text_utils.py`
- `requirements.txt`, `Makefile`

Session timeline (120 min)

Time	Activity
0–10 min	Environment checks + quick LLM sanity prompts in Cursor and CLI
10–35 min	Exercise A: write new code with LLM assistance (analysis function)
35–60 min	Exercise B: modify existing code + test-driven bug fixing
60–85 min	Exercise C: use CLI agent for file tasks and report generation
85–105 min	Exercise D: generate plots/charts from local CSV data

105–120 min	Exercise E: produce document artifacts (Markdown + optional LaTeX table), wrap-up

Exercise 0 — Setup check (10 min)

Run in terminal:

```
cd llm-coding
command -v uv >/dev/null || curl -LsSf https://astral.sh/uv/install.sh |
    sh
uv venv .venv
source .venv/bin/activate
uv pip install -r requirements.txt
uv run python src/starter_analysis.py
uv run python src/generate_report.py
uv run python src/plot_metrics.py
```

Quick prompts (Cursor and CLI):

- "Read this folder and explain what each file is for."
- "Suggest 3 improvements to make this mini-project clearer for a beginner."

Exercise A — Write new code in Cursor (25 min)

Open `src/starter_analysis.py`. There are TODOs in:

- `compute_group_summary(...)`
- `find_top_improvers(...)`

Task:

1. Ask Cursor to implement both TODOs.
2. Ask Cursor to add docstrings and type hints if missing.
3. Run:

   ```
   uv run python src/starter_analysis.py
   ```

Prompt examples:

- "Implement TODO functions in this file only. Keep output deterministic and concise."
- "Before editing, explain your plan in 3 bullets; after editing, show what changed."

Exercise B — Modify code + tests (25 min)

Open `src/text_utils.py`. It contains intentionally weak behavior.

Task:

1. Run tests: `uv run pytest -q` (or `make test`)

2. Use Cursor to fix failing tests in `tests/test_text_utils.py`
3. Ensure all tests pass.

Prompt examples:

- "Read failing tests and patch only src/text_utils.py. Do not change tests unless absolutely necessary."
- "Keep functions simple and robust for edge cases (extra spaces, mixed case, punctuation)."

Exercise C — CLI agent workflow (25 min)

Use Codex/Claude CLI to do a small multi-file task:

1. Ask the CLI to inspect project structure.
2. Ask it to run `uv run python src/generate_report.py`.
3. Ask it to improve `reports/workshop_report.md` by adding one extra section: "Potential failure modes in this dataset and mitigation ideas."

Prompt template:

```
You are helping with a local Python project.
1) inspect files
2) run report generation
3) update the report with a short "failure modes" section
4) do not add new dependencies
5) show me exactly what you changed
```

Exercise D — Plots/charts from local data (20 min)

Run:

```
uv run python src/plot_metrics.py
```

This should generate:

- `reports/score_by_group.png`
- `reports/accuracy_vs_latency.png`

Stretch task: Ask an LLM to add one more plot (example: token usage distribution by model).

Exercise E — Document workflow (15 min)

Option 1 (fast): update `reports/workshop_report.md`.
Option 2 (LaTeX): ask the LLM to generate a table from CSV summaries and paste into a `tabular` block.

Example prompt:

```
From data/students_scores.csv, compute mean assignment/midterm/final by
   group,
and produce a LaTeX tabular snippet with booktabs formatting.
```

Best practices to emphasize

- Ask for a plan first, then ask for edits.
- Scope the edit: file(s), constraints, and expected output.
- Require verification steps (tests, script runs, lint checks).
- Prefer iterative prompts over one giant prompt.
- Keep humans in the loop for correctness, reproducibility, and security.

Common failure modes to discuss

- Silent hallucinations (invented functions, files, APIs).
- Over-broad code edits when prompt scope is vague.
- Incomplete environment assumptions (wrong Python/package versions).
- Data leakage/privacy risks when sharing sensitive content.
- Overconfidence in generated analysis without validation.

Wrap-up checklist

[ ] I can use Cursor to create and edit multi-file code changes.

[ ] I can use an LLM CLI agent to run commands and modify files safely.

[ ] I can generate and validate local data plots and summaries.

[ ] I understand at least 3 failure modes and mitigation strategies.

Optional extension for next session: multi-agent orchestration where one agent writes code, one runs tests, and one performs review.