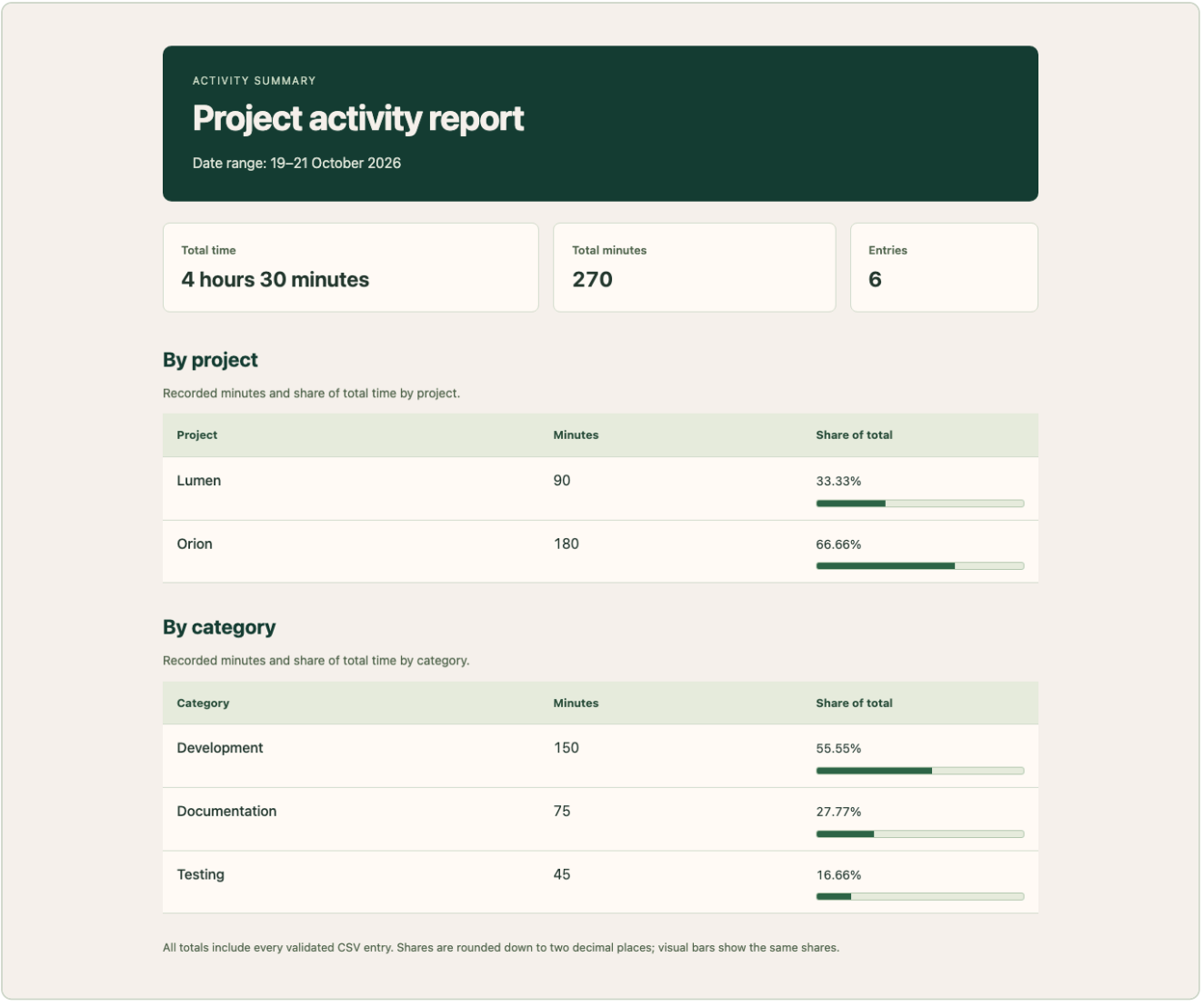
Build your first report

A useful result. Clear boundaries. Evidence you can inspect.

PLAN → BUILD

See what you can build

Turn six activity entries into a formatted report you can open, explain, and print. Let your AI assistant handle the implementation while you direct the work.



ACTIVITY SUMMARY

Project activity report

Date range: 19–21 October 2026

Total time	Total minutes	Entries
4 hours 30 minutes	270	6

By project

Recorded minutes and share of total time by project.

Project	Minutes	Share of total
Lumen	90	33.33%
Orion	180	66.66%

By category

Recorded minutes and share of total time by category.

Category	Minutes	Share of total
Development	150	55.55%
Documentation	75	27.77%
Testing	45	16.66%

All totals include every validated CSV entry. Shares are rounded down to two decimal places; visual bars show the same shares.

The formatted report generated from the six-row synthetic CSV.

Click **Open formatted report** to explore the actual HTML file.

Harness gives your assistant a defined goal, file boundaries, protected checks, and a separate reviewer. You decide what may change and assess the result.

This lesson follows the earlier text-report build and its checked repair. The current example adds the formatted HTML report shown here. You can inspect both without starting AI work.

DEFINE → PLAN

Start with data you can check

The projects are fictional. The question is practical: where did the work go?

Six synthetic activity entries

Project	Category	Minutes
Orion	Development	90
Orion	Testing	30
Orion	Documentation	45
Lumen	Development	60
Lumen	Documentation	30
Orion	Testing	15

Add the minutes: **90 + 30 + 45 + 60 + 30 + 15 = 270.** Orion totals 180 minutes; Lumen totals 90.

Your project brief needs an input example, a target result, and a clear rule for bad input. Here, invalid rows must produce a helpful error and no partial report.

sample data

```
date,project,category,minutes
2026-10-19,Orion,Development,90
2026-10-19,Orion,Testing,30
2026-10-20,Orion,Documentation,45
2026-10-20,Lumen,Development,60
2026-10-21,Lumen,Documentation,30
2026-10-21,Orion,Testing,15
```

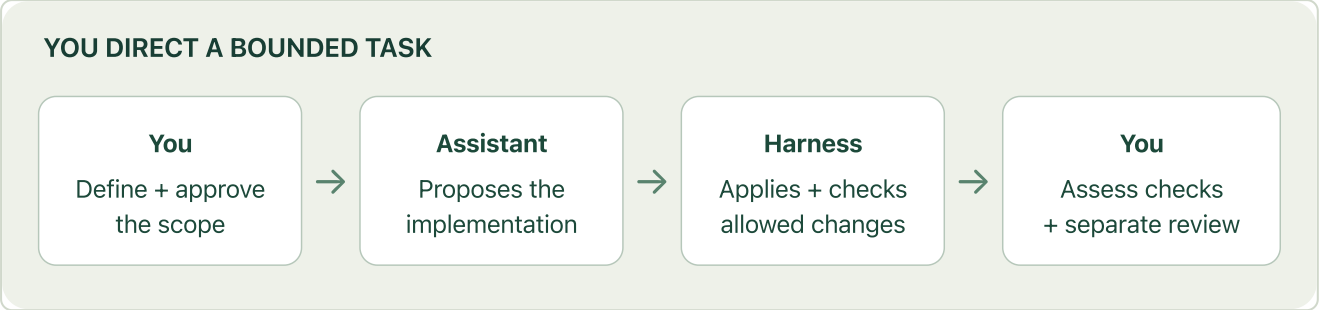
PLAN

Ask for an approach you understand

You do not need to design the parser yourself. Ask the assistant to explain the work and its checks before it starts.

1. Validate the CSV and required fields.
2. Count every valid row and add whole-number minutes.
3. Group totals by project and category.
4. Format the report and run the protected checks.
5. Get separate review and retain the outcome.

A short plan answers *what will change, what stays protected, and how we will check it.*



The person directs the work; the assistant and Harness carry out the bounded task.

planning prompt

```
Prepare a short plan for the activity-reporting task. Show the intended report, the file that may change, protected inputs, and the checks. Keep this preparation read-only. Do not start AI work or publish anything.
```

PLAN · SCOPE

Agree to the files and checks

In the historical build, only report.py could change. The checks and their expected answers stayed protected.

Allowed in the recorded build

`report.py` — the reporting implementation.

▼ **The eight protected files**

- `tests/test_report.py`
- `data/activity.csv`
- `expected/activity-report.txt`
- `README.md`
- `AGENTS.md`
- `participants.json`
- `pytest.ini`
- `.gitignore`

Project filtering and new dependencies were outside that request. Keeping tests and fixtures protected helps you see whether the implementation meets the agreed result.

▼ **About these controls**

File boundaries, frozen inputs, explicit approvals, and bounded attempts are workflow controls. They do not establish a general security sandbox. Use the supported profile and trusted example data.

PLAN · APPROVAL

Approve the scope before execution

Approval records agreement with the goal, allowed files, and checks. Execution has its own decision.

In an eligible Harness 1.5.1 draft form, **Approve this work request** records scope approval. **Keep as draft** leaves the request unapproved. Neither starts the work.

This page is a training lesson. It does not approve a Harness task, and the completed historical task is not an active draft approval form.

Ask your assistant to show the current request and explain the effect of approval in plain language.

inspection prompt

```
Help me inspect this reporting task. Show me the intended result, what may change, what stays protected, and the checks and review that support the result. Explain what approving the request permits and what separately authorizes execution. Keep this inspection read-only.
```

Check your understanding

After approving the work request, has AI execution started?

- No. Execution needs separate authorization.
- Yes. Approval automatically starts the worker.

Choose an answer to see feedback.

BUILD · EXECUTION

Give the assistant a bounded go

When the request is ready, explicitly authorize execution for that accepted task and its reviewed limits.

The historical run used one configured worker and a separate reviewer, one attempt each, with 180 seconds per role. Harness applied the worker's allowed-file proposal and ran the checks before review.

Harness 1.5.1 browser forms can handle intent. This historical run accepted scope through `plan --accept` and executed through separately authorized CLI `build` commands. There is no browser *Start build* control in this lesson.

Copying a prompt only copies text. Read it before sending it to your assistant.

▼ **What the assistant checks first**

The actual task identity, current checkpoint, accepted scope, configured participants, supported effects, protected checks, budgets, and the authorized subscription route.

execution readiness prompt

```
Before starting this accepted reporting task, show me its current checkpoint, allowed files,
protected checks, configured worker and reviewer, and attempt and time limits. Confirm the agreed
subscription route. Wait for my explicit execution go. Do not publish.
```

BUILD → VERIFY

Check the report and the evidence

The historical build produced the expected sample report and passed both protected probes.

49 passing cases in the task probe.

49 passing cases in the final probe.

Only report.py changed. All eight protected files stayed unchanged.

Each probe ran the same 49 cases. The host also ran the actual sample CLI and matched its output to the target byte for byte.

Check the total and headings yourself. Then ask the assistant for the saved checks and file comparison that support its answer.

sample report

```
Project activity report
Entries: 6
Total minutes: 270

By project
- Lumen: 90
- Orion: 180

By category
- Development: 150
- Documentation: 75
- Testing: 45
```

VERIFY → REVIEW

Look at what the reviewer found

Passing the chosen cases is useful evidence. Separate review can still uncover an assumption the tests missed.

> **R1 · Medium severity**
>
> A valid minutes value longer than 4,300 digits caused a traceback. The original checks had not covered that boundary.

The reviewer identified the concern through static review. The host then reproduced it in a separate check.

Actual read-only Harness snapshot of the earlier run, before the later host repair. It is historical evidence.

▼ **Preview Tutorial 4: what completed meant**

The configured workflow stopped for high-severity findings. This medium finding was retained while the run was marked completed. Human assessment still needed to address it.

Keep the successful checks and the concern together. Preserve the record, authorize a bounded correction, and add a check that exposes the failure.

Check your understanding

Both probes passed, but the reviewer found R1. What should happen next?

- Preserve the record, inspect R1, and choose a bounded correction with suitable checks.
- Ignore R1 because the tests passed.

Choose an answer to see feedback.

REPAIR → VERIFY

Make the correction traceable

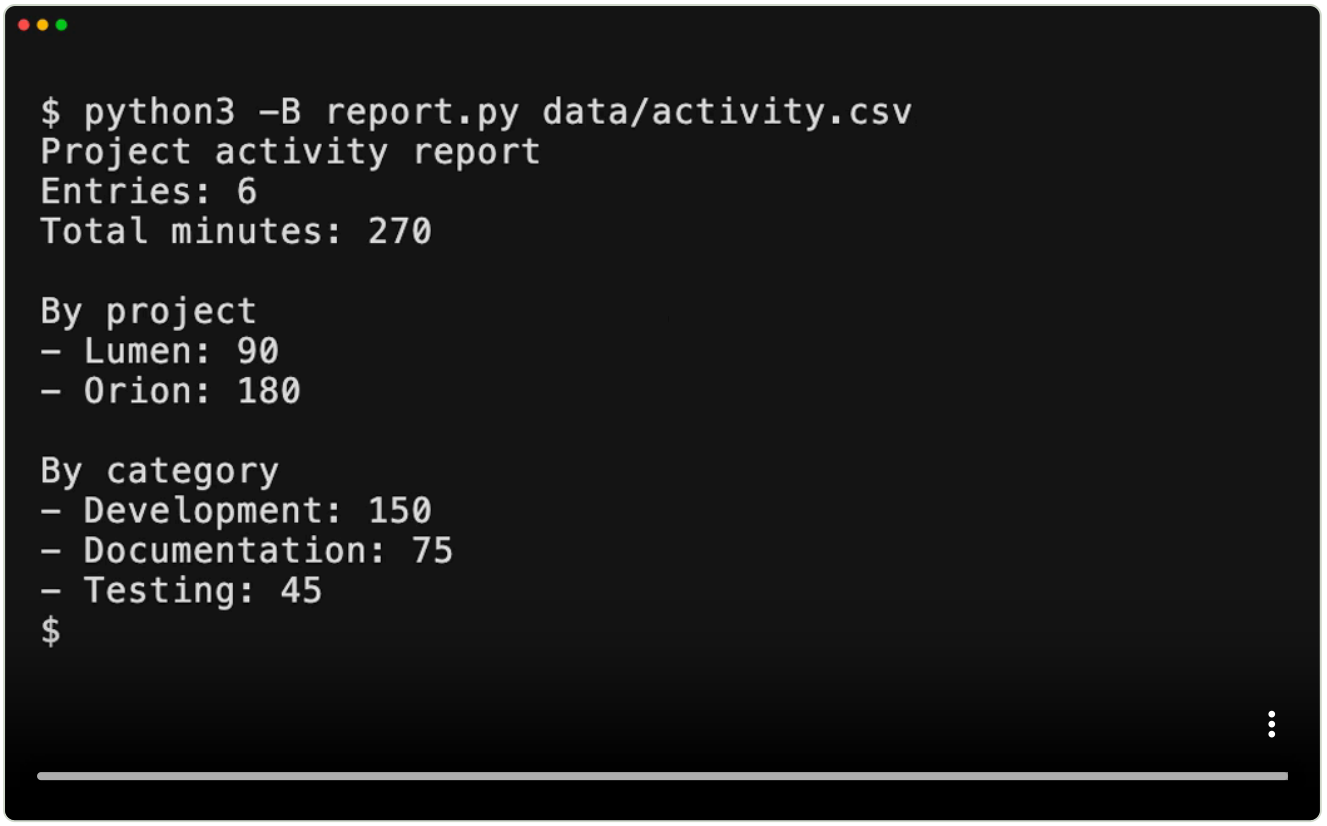
A later host repair used a separate project copy. The original AI rehearsal and its finding stayed intact.

Separate host repair evidence

Check	Observed result
New cases against original code	4 failed, 4 passed
Same new cases after repair	8 passed
Full repaired suite	57 passed

The repair changed `report.py` and appended regression coverage. All original 49 cases remained intact. Python's conversion safeguard stayed unchanged.

This was a host repair, with no native AI rerun. The repaired sample still produces the same useful report.



VHS capture: the current repaired CLI reads the synthetic sample and prints the report. The AI build is documented by its saved record.

▼ **Read the terminal transcript**

Current repaired reporting example. The command prints six entries, 270 minutes, Lumen 90, Orion 180; Development 150, Documentation 75, Testing 45. This local command uses no model call.

Read the runnable reference instructions or inspect the sanitized evidence summary.

sample command

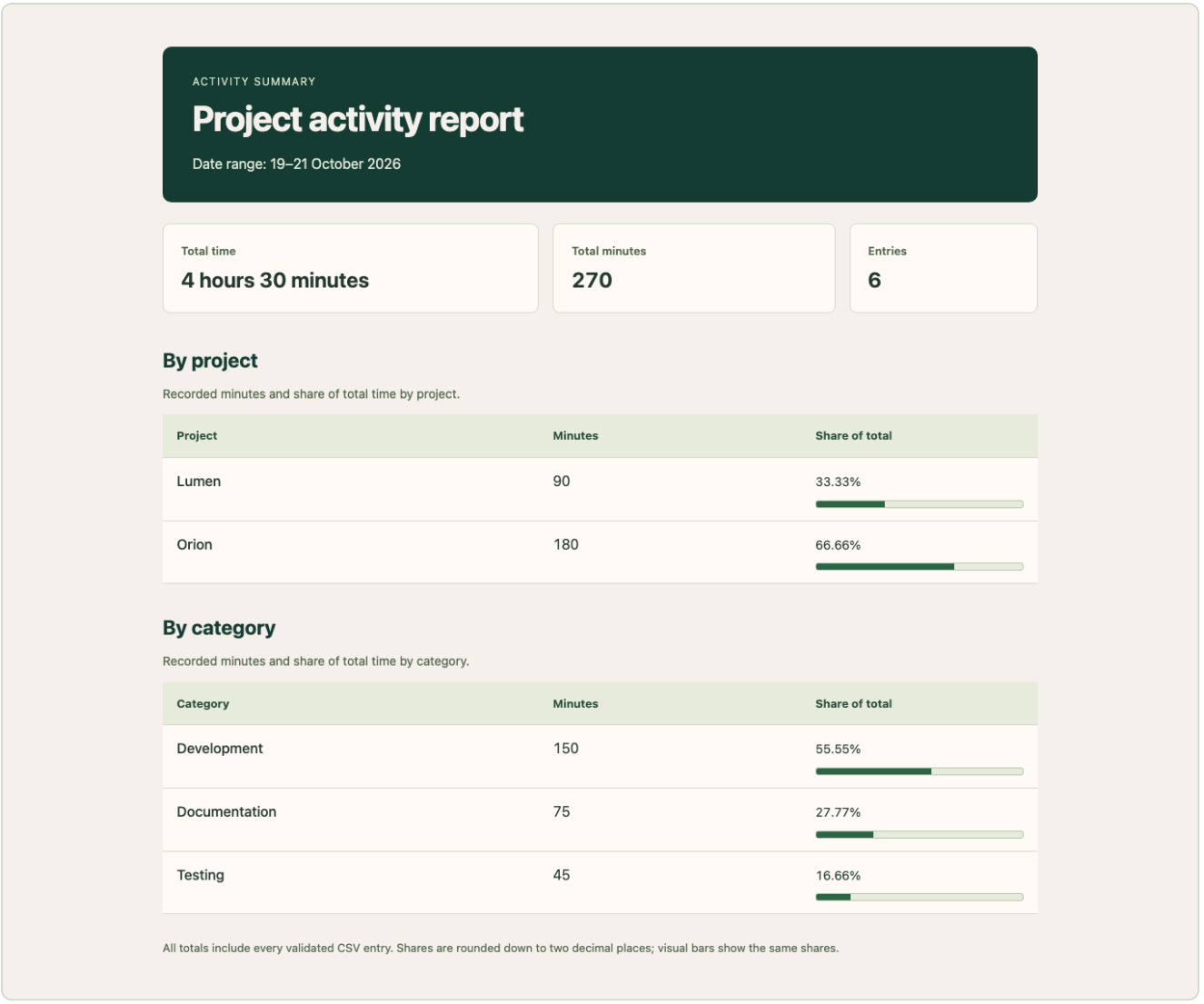
```
python report.py data/activity.csv
```

PLAN → BUILD · RESULT

Your report, ready to use

The goal is a readable report from the activity CSV. The current tool writes a formatted HTML file you can open in your browser and print.

The sample covers **19–21 October 2026**: **6 entries and 270 minutes**, or **4 hours 30 minutes**. Its project tables show Lumen at 90 minutes and Orion at 180. The category totals are Development 150, Documentation 75, and Testing 45.



ACTIVITY SUMMARY

Project activity report

Date range: 19–21 October 2026

Total time	Total minutes	Entries
4 hours 30 minutes	270	6

By project

Recorded minutes and share of total time by project.

Project	Minutes	Share of total
Lumen	90	33.33%
Orion	180	66.66%

By category

Recorded minutes and share of total time by category.

Category	Minutes	Share of total
Development	150	55.55%
Documentation	75	27.77%
Testing	45	16.66%

All totals include every validated CSV entry. Shares are rounded down to two decimal places; visual bars show the same shares.

This is the formatted report generated from the supplied synthetic data.

Click **Open formatted report** to inspect the HTML file. Its tables and bars show where the time went, and its print layout is included in the same file.

Generate the report on your computer

1. Click **Download checked example ZIP** and extract the files.
2. Find the extracted folder that contains `report.py` and `data/activity.csv`.

3. Ask your assistant for the exact `cd` command to enter that folder, then run the Python 3.12 command below. On Mac, you can also type `cd` followed by a space in Terminal, drag the extracted folder from Finder into the window, and press **Return**.
4. When the command finishes, ask your assistant to open `activity-report.html` in your browser, or open the file yourself.
5. Check the dates and totals against the CSV. Use your browser's print command if you want a paper or PDF copy.

Run this command from the extracted example folder. That folder must contain `report.py` and `data/activity.csv`.

run command

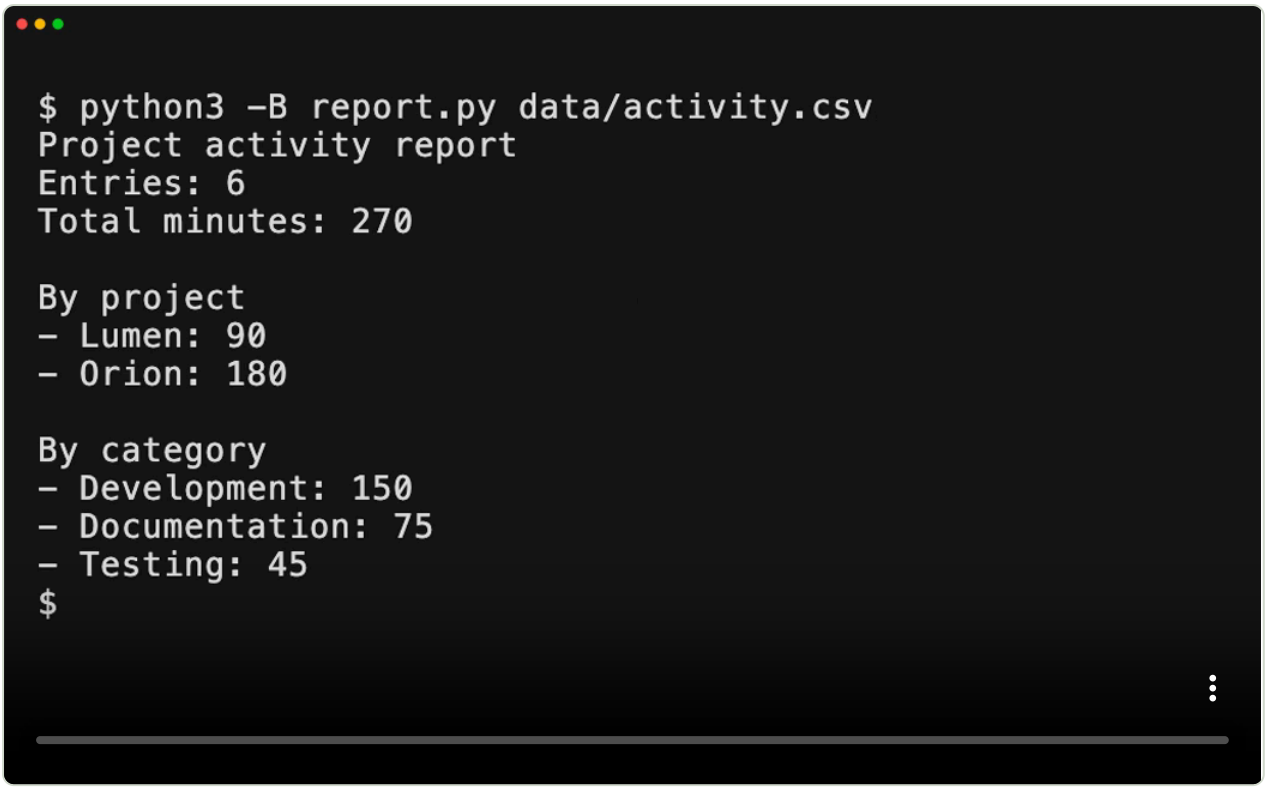
```
python3.12 report.py data/activity.csv --format html --output activity-report.html
```

The command creates `activity-report.html` in the current folder. It refuses to replace an existing file; choose a new output filename if you run it again, then open that same file.

For the reference files and optional checks, click **Runnable reference instructions**.

▼ **Optional: use the text report**

Text output remains available with `python3.12 report.py data/activity.csv`.



Earlier VHS recording of the repaired text-report example. It does not demonstrate the later HTML format or a new AI run.

Transcript: The earlier command prints six entries and 270 minutes: Lumen 90, Orion 180; Development 150, Documentation 75, Testing 45.

▼ **Optional: make your next request**

Write a read-only request that names the goal, file boundary, evidence, and next decision. Use your own words.

Your next request

What should the assistant show you before taking the next action?

Ask the assistant to show the current saved task, explain what changed and what remains protected, summarize checks and review, and stop before execution.

saved-progress prompt

```
Show me the current saved reporting task and where it stopped. Explain the goal, accepted
scope, recorded execution, checks, review findings, and any later repair. Keep the inspection
read-only and ask before starting new work.
```

When you are ready to return to saved work, continue with **Tutorial 3: Come back tomorrow**. **Tutorial 4** goes deeper into evidence and review.

▼ **Version, evidence, and limits**

The historical native text-report build passed the same 49 cases in each of two probes and retained the medium R1 finding. A later host repair preserved those 49 cases, added eight regression cases, and passed 57. Both stages came before this HTML addition. Their original records remain intact; no native AI rerun followed the host repair.

The current example retains the existing text-report checks. The current suite passed 77 checks: the unchanged 57 text cases and 20 new HTML cases. Click **Current verification results** for the source-bound summary. Running the reporting program on the supplied CSV uses no model call.

The recorded Harness workflow used version 1.5.1 on macOS. Native planning and building remain experimental. Windows users can install Harness and use documented workflows within supported profiles; the versioned capability guide explains remaining storage, recovery, and qualification limits.

Read the 1.5.1 capability and reliability guide.

Local software checks support this example. A newcomer pilot has not run, so learning speed and effectiveness remain to be observed.

Local unpublished draft. Lesson progress stays in this browser; it does not approve or run Harness work.