

INSTRUCTOR GUIDE AND NOTES

Python and AI

Wrangle a dataset, then talk to a local model with Python scripts. A 50-minute lesson on data cleaning, the agent loop, and keeping an agent safe.

Series	Hack Your Future, PrivacySafe Foundation's series for building what's next
Audience	Learners who can type commands in a terminal. Some programming helps, but the scripts are short and readable.
Length	50 minutes, with a 30-minute cut
Format	4 slides, live lab demo, 4 slides, hands-on script run, discussion
You will need	Python 3.9 or newer, Ollama with a model that supports tools, the scripts folder, slides (light for print, dark for projecting), flash cards
Facts checked	October 27, 2026. Libraries and models change. Re-check before you teach.

PrivacySafe Foundation, Inc. | Hack Your Future | These materials are dedicated to the public domain under CC0 1.0 and shared for the public good. The PrivacySafe and PrivacySafe Foundation names and logos are trademarks and are not included in that dedication. Python is a trademark of the Python Software Foundation. Ollama, Olmo and other product names belong to their owners.

The lesson in five sentences

1. Python reads close to plain English and steers fast compiled libraries, which is why its ecosystem covers data, machine learning and models^{[1][6][7][8][9][10]}.
2. Data work is a pipeline: load, clean, transform, summarize, ask. Never change the raw file. Write a clean copy.
3. Cleaning is a list of small named steps. Skipping one often gives wrong numbers silently instead of crashing, which the lab shows.
4. An agent is a model, some tools and a loop. When the model "calls a tool" it only asks. Your script runs the tool and sends the result back^[11].
5. Keep agents safe with narrow read-only tools, checked arguments, a step limit, a log, and a human approval step for any write. Treat text in your data as untrusted^{[15][16]}.

LEARNING OUTCOMES

By the end, students can:

1. Explain why Python is widely used for data and AI work.
2. Read a short script that loads a CSV, cleans it and summarizes it^{[2][17]}.
3. Describe the cleaning steps and what goes wrong when one is skipped.
4. Explain the agent loop and say who runs the code.
5. Name at least four safety measures for a tool-using agent^[15].

Session plan

SEGMENT	MINUTES	NOTES
Before the demo, slides 1 to 4	12	Includes the prediction: who runs the code when an agent calls a tool.
Lab demo	12	Sections 1 to 3. Section 4 lists the commands for the hands-on run.
After the demo, slides 1 to 4	10	Wrangling, talking to a local model, building an agent, checklist.
Hands-on script run	10	Learners run the scripts. Pre-install to save time.
Discussion	6	Two questions.

30-MINUTE CUT

Show the scripts on the instructor machine instead of the hands-on run. In the lab, do sections 1 and 3 only. Cover after-demo slides 1 and 3. Use one discussion question.

Before class

- Optional: the lesson page has a walkthrough video of under four minutes, in a light version for the page and a dark version for projecting, each with a subtitles file. Play it as a recap, or share it with learners who missed the session. Check that it plays on the room computer first.
- **Check Python.** Run `python3 --version` (or `python --version` on Windows). The scripts need 3.9 or newer.
- **Install everything beforehand.** Install Ollama, run `ollama pull olmo-3:7b-instruct`, create a virtual environment and run `pip install ollama` ^{[3][4][12]}. A 4.5 GB download and a pip install are slow on shared Wi-Fi.
- **Run every script once** on the instructor machine: `python wrangle.py`, `python hello_local.py`, `python local_agent.py "Which branch had the most loans?"`. Compare `wrangle.py` with `expected_output.txt`.
- **Check tool support.** The Olmo 3 library page lists tools among its capabilities ^[14]. If your model does not call tools reliably, pass another with `--model`. Smaller models make more mistakes with tools.
- Open the lab page and try each section. Set browser zoom to about 125% for the back row.
- Print the flash cards, double-sided, flipped on the long edge.
- Remind learners to use only the made-up dataset. Do not run the agent on real personal data in class.

WINDOWS AND TERMINAL NOTES

- On Windows, the command is often `python` or `py`, not `python3`.
- Activate a virtual environment with `.venv\Scripts\activate`. PowerShell may block the activation script until you allow it. See the `venv` documentation ^[3].
- If learners skip the virtual environment, `pip install` may install into the wrong Python. Teach the habit early ^[4].

WHAT THE LAB IS, AND IS NOT

The lab runs in the browser and does not run Python. It mirrors the cleaning and summary logic of `wrangle.py` in JavaScript, and its numbers match the script. The agent stepper computes tool results from the cleaned data, but the model's replies are scripted for teaching. Real models vary in how well they choose tools. Run the real scripts to see that.

Slide notes: before the demo

Slide 1: Title

about 2 min

SAY

Why is Python the default language for data and AI? Our promise: short, readable scripts that run on your own computer. Our job: clean a messy dataset, then let a local agent answer questions about it.

DO

Point at the five-segment bar: load, clean, transform, summarize, ask. It is the shape of the whole session.

CHECK FOR UNDERSTANDING

Ask: "Who has written code before? In what language?" Use answers to pace the session.

Slide 2: Why Python for AI

about 4 min

SAY

- Readable: code is close to plain English^[1].
- A huge ecosystem: pandas for tables^[6], NumPy for arrays^[7], scikit-learn for classical machine learning^[8], PyTorch for neural networks^[9], Transformers for pretrained models^[10].
- The glue: your Python script steers fast compiled libraries underneath.
- Local-model tools ship Python libraries, including Ollama^[12].

DO

Draw the three layers: your script, the libraries, compiled code.

CHECK FOR UNDERSTANDING

Ask: "If the heavy math is in compiled code, what is Python for?" Answer: describing the steps clearly, and connecting pieces.

WATCH FOR

Do not claim Python is the fastest or the only choice. It is popular for its readability and libraries.

Slide 3: From raw data to answers

about 4 min

SAY

Load, clean, transform, summarize, ask. Walk the four example fixes: `" Harbor "` becomes `"Harbor"`, `"FICTION"` becomes `"Fiction"`, `"1,204"` becomes `1204`, and a blank becomes `None`. Show `.strip().lower()`, `.replace(",","")` and `is None` ^[1].

DO

Stress the rule: never change the raw file. Write a clean copy so you can always start again.

CHECK FOR UNDERSTANDING

Ask: "What if two rows are exact duplicates and both are real?" Good discussion: cleaning rules encode assumptions. Write them down.

WATCH FOR

Learners may think cleaning is trivial. The lab shows how skipped steps silently change totals.

Slide 4: What to watch for, and the prediction

about 2 min

SAY

Python ideas: functions and docstrings, lists and dictionaries, reading a CSV with the csv module^[2], loops, sorting and grouping, and checking for missing values. Agent ideas: an agent is a model, tools and a loop; tools are plain Python functions; your script runs the tools; check arguments; limit steps, use read-only tools and keep logs.

DO

Read the prediction aloud: "When an agent calls a tool, who actually runs the code, the model or your script?" Give 60 seconds. Take a show of hands.

WATCH FOR

Many learners say the model runs it. The model only produces text asking for a tool.

Lab demo script (about 12 minutes)

Section 1: Clean a messy dataset

4 min

SAY

The raw file has 125 rows. Switch to the raw view: dashed cells are stray spaces, odd capitals or commas. Red cells are blanks. With every step on, the counts match the script: **125 rows read, 28 names fixed, 1 number with a comma, 5 duplicates removed, 4 rows dropped, 116 rows clean.**

DO

Switch to the cleaned view. Then switch cleaning steps off one at a time and watch section 2.

CHECK FOR UNDERSTANDING

Ask: "Which step would you skip if you were in a hurry, and what would happen?"

WATCH FOR

With names not cleaned, one branch is counted as several groups. With numbers not parsed, summing raises a `TypeError`. The first fails silently, which is more dangerous.

Section 2: Summarize it

3 min

SAY

Group by branch and measure total loans: Central 15,749, Riverside 13,239, Harbor 11,950, Hillside 9,322. Switch to late-return rate: Hillside 6.3%, Harbor 5.8%, Riverside 5.7%, Central 5.7%. Show the Python panels: the standard library code and the pandas code give the same result^{[6][17]}.

DO

Change the grouping and the branch filter and watch the code update.

CHECK FOR UNDERSTANDING

Ask: "Why does the late rate use only rows where late_returns is known?" Missing values would otherwise be counted as zero.

Section 3: Step through a local agent

4 min

SAY

Pick "Which branch had the most loans?" and click Next. User asks, model asks for `sum_by`, your script validates and runs it, the result goes back, the model answers. **The model never runs code.** Then try "Please delete all 2025 rows": the model asks for a tool that does not exist, and the script refuses.

DO

Then run the notes question. A note contains a hidden instruction. With no save tool, the request fails. Turn on the save tool and turn off human approval: a write happens. Turn approval on: the script asks first^{[15][16]}.

CHECK FOR UNDERSTANDING

Ask: "What stopped the damage in each case?" Missing tool, then human approval.

WATCH FOR

The model replies in the stepper are scripted for teaching. Say that real models vary: some resist injected text, some do not. Design as if it can happen.

Section 4: Run the scripts yourself

1 min

SAY

The page lists the commands: create a virtual environment, install the library, pull a model, run the scripts. Each has a copy button^{[3][12]}.

DO

Hand over to the hands-on run.

Close the prediction

1 min

SAY

Your script runs the code. The model only produces a tool name and arguments^[11].

DO

Return to slide 4. Ask who guessed correctly.

Slide notes: after the demo

Slide 1: Wrangling a dataset with Python

about 3 min

SAY

Walk the code: open the CSV, read rows as dictionaries with `csv.DictReader` ^[2], clean them, total loans by branch with a `defaultdict` ^[17], sort and print. The cleaning report shows 125 rows read, 5 duplicates removed, 4 dropped, 116 clean. `wrangle_pandas.py` gives the same numbers in fewer lines ^[6].

DO

Show `expected_output.txt` so learners can check their own run.

CHECK FOR UNDERSTANDING

Ask: "When would you choose pandas over the csv module?" Bigger tables and heavier grouping, at the cost of an install.

Slide 2: Talking to a local model

about 3 min

SAY

Four lines: import `chat` from `ollama`, call it with a model name and a list of messages, print the reply. The model runs on your machine through the Ollama server, so nothing leaves your computer ^{[13][12]}. Do not paste a whole table into the prompt. It may not fit in the context window and models miscount. Let Python do the arithmetic.

CHECK FOR UNDERSTANDING

Ask: "What are the roles in a message list?" System, user, assistant, tool ^[11].

Slide 3: Building a local agent

about 4 min

SAY

The loop: you ask, the model replies, your script runs the tool, the result goes back, the model answers. In code: call `chat` with `tools`, check `message.tool_calls`, run each tool, append a message with role `tool`, and repeat ^[11]. Plain Python functions can be passed as tools. The library builds the description from the type hints and docstring ^{[11][12]}.

DO

Read the "Safe by design" card: narrow read-only tools, allowlisted arguments, a step limit and a log, human approval for writes, data text treated as untrusted ^[15].

CHECK FOR UNDERSTANDING

Ask: "Why is a tool that runs any shell command a bad idea?" It gives the model, and anyone who can influence it, open-ended power ^[15].

Slide 4: Scripts and agents checklist

about 4 min

SAY

1. Is the raw data left untouched?
2. Do I check types, blanks and duplicates?
3. Can I run it twice and get the same result?
4. Did I check a few rows by hand?
5. Are the tools narrow and read-only?
6. Do I check arguments before running anything?
7. Is there a step limit and a log of every call?
8. Did I verify answers and keep secrets out?

DO

Optional activity: give pairs a messy list and ask them to write down the cleaning rules before coding.

Hands-on script run (about 10 minutes)

1. Open a terminal in the `scripts` folder. Create and activate a virtual environment^{[3][4]}.
2. `python wrangle.py`. Compare the output with `expected_output.txt`.
3. Edit `wrangle.py`: add a table of total loans by category using `sum_by(rows, "category", "loans")`. Run it again.
4. `python hello_local.py`, then change the question.
5. `python local_agent.py "Which branch had the most loans?"`. Read the printed tool call and compare the answer with `wrangle.py`.
6. Ask the agent "Summarize the notes for the Harbor branch." Read the log in `agent_log.jsonl`. Did the model follow the hidden note? Either way, no tool can act on it.
7. Stretch: change the cleaning so a missing `late_returns` counts as 0. Re-run and compare the late-return rates. Discuss how the assumption changes the answer.

Expected output of wrangle.py

```
CLEANING REPORT
rows read          125
names fixed        28
thousands separators 1
duplicates removed 5
dropped missing loans 4
unknown names      0
rows clean         116
```

```
TOTAL LOANS BY BRANCH
Central    15,749
Riverside  13,239
Harbor     11,950
Hillside   9,322
```

```
LATE-RETURN RATE BY BRANCH
Hillside    6.3%
Harbor      5.8%
Riverside   5.7%
Central     5.7%
```

```
AVERAGE WAIT (DAYS) BY CATEGORY
Reference    3.21
Non-fiction  3.36
Children     3.45
Audio        3.58
Fiction      3.91
```

```
Saved 116 clean rows to loans_clean.csv
```

Troubleshooting

SYMPTOM	LIKELY CAUSE AND FIX
<code>python3: command not found</code> or <code>python</code> not found	Use the other name (<code>python</code> , <code>py</code>). Install Python from python.org if missing.
<code>ModuleNotFoundError: No module named 'ollama'</code>	The package is not installed in this Python. Activate the virtual environment, then run <code>pip install ollama</code> ^[3] ^[4] .
"Could not reach Ollama"	Ollama is not running. Start the app or run <code>ollama serve</code> ^[13] .
"Model is not downloaded"	Run <code>ollama pull olmo-3:7b-instruct</code> , or pass a model you have with <code>--model</code> .
"Does not list tool support"	Pick a model that supports tools. The script checks and tells you ^[14] ^[11] .
The agent answers without calling a tool, or picks the wrong one	Smaller models are less reliable with tools. Read the log, try a larger or different model, or ask the question more directly.

The first answer takes a long time

The model is loading into memory. Later calls are faster. See the Local AI lesson.

Numbers differ from
expected_output.txt

The data file changed. Re-create it with `python make_data.py`.

Discussion (about 6 minutes)

QUESTION 1: AUTOMATE AND VERIFY

"Which step of your own data work would you automate first? What could go wrong if the script silently skipped bad rows?"

- Good candidates: repeated cleaning, weekly summaries, merging exports.
- Silent skipping is the danger. Report counts of what was dropped, as `wrangle.py` does.
- Write down cleaning rules as assumptions that someone else could review.

QUESTION 2: HARM WITH READ-ONLY TOOLS

"An agent can only call read-only tools. What harm is still possible, and how would you notice?"

- Wrong answers presented confidently: verify against the data.
- Data disclosure: a read-only tool can still reveal sensitive rows to the wrong person^[15].
- Misleading output shaped by injected text in the data^[16].
- Noticing: logs, spot checks, and comparing with a script you trust.

Answer key and assessment

QUESTION	ANSWER
When an agent calls a tool, who runs the code?	Your script. The model only requests a tool and arguments ^[11] .
What does <code>csv.DictReader</code> give you?	Each row as a dictionary keyed by column name ^[2] .
Why keep the raw file untouched?	So you can start again and audit what cleaning changed.
Why check arguments before running a tool?	Models can send wrong or unexpected values. Allowlists limit what can happen ^[15] .
Name the three root causes of excessive agency.	Excessive functionality, excessive permissions, excessive autonomy ^[15] .
Why is the notes column risky?	It is untrusted free text and can carry hidden instructions ^[16] .

EXIT TICKET (3 MINUTES)

1. When an agent calls a tool, who runs the code?
2. Name two cleaning steps and one thing that goes wrong if you skip it.
3. Name two safety measures for a tool-using agent.

HOMEWORK

- Add a new summary to `wrangle.py` (for example, average late returns per month) and print it.
- Add one new read-only tool to `local_agent.py` with a docstring and an allowlist for its arguments, and ask the agent a question that needs it.
- Stretch: write three tests for `clean_rows` using small hand-made rows.

Common questions and misconceptions

QUESTION OR BELIEF	RESPONSE
"The model runs my function."	No. The model returns a tool name and arguments as data. Your script runs the function ^[11] .
"If the model says it ran code, it did."	Only your script's output is real. Check the printed tool calls and the log.
"An agent is autonomous."	Only as far as you let it be. The loop, the tools and the limits are your code.
"pandas is always better."	It is shorter for tables. The csv module needs no install and is clearer for learning ^{[2][6]} .
"Cleaning is just tidying."	Cleaning decisions change the numbers. Document them.
"A local model makes the agent safe."	Local keeps data on your machine. Tool permissions and injected text still matter ^{[15][16]} .

References and further reading

Citations in this guide use bracketed numbers. Links were checked on October 27, 2026. Library APIs change, so confirm the tool-calling details against the current Ollama documentation before you teach.

#	SOURCE	WHAT IT SUPPORTS
[1]	Python documentation, The Python Tutorial https://docs.python.org/3/tutorial/	Functions, docstrings, lists, dictionaries, None, modules and imports.
[2]	Python documentation, csv module https://docs.python.org/3/library/csv.html	Reading and writing CSV files, including csv.DictReader.
[3]	Python documentation, venv: creation of virtual environments https://docs.python.org/3/library/venv.html	Creating and activating a virtual environment.
[4]	Python Packaging User Guide, Installing packages using pip and virtual environments https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/	Installing packages with pip inside a virtual environment.
[5]	PEP 8, Style Guide for Python Code https://peps.python.org/pep-0008/	Readable code style.
[6]	pandas documentation https://pandas.pydata.org/docs/	DataFrames, grouping, cleaning and summarizing tables.
[7]	NumPy documentation https://numpy.org/doc/stable/	Fast array computing that many data and AI libraries build on.
[8]	scikit-learn documentation https://scikit-learn.org/stable/	Classical machine learning in Python.
[9]	PyTorch documentation https://pytorch.org/docs/stable/index.html	Deep learning framework used to build and run neural networks.
[10]	Hugging Face, Transformers documentation https://huggingface.co/docs/transformers	Loading and running pretrained language models in Python.
[11]	Ollama documentation, Tool calling https://docs.ollama.com/capabilities/tool-calling	Passing Python functions as tools, reading tool_calls, sending tool results with the tool role, and the multi-turn agent loop.
[12]	Ollama Python library (GitHub) https://github.com/ollama/ollama-python	The ollama package, installed with pip.
[13]	Ollama documentation, FAQ https://docs.ollama.com/faq	Local server on 127.0.0.1, default context, and turning off cloud features.
[14]	Ollama library, olmo-3 https://ollama.com/library/olmo-3	Olmo 3 tags and sizes. The page lists tools among its capabilities.

[15]	OWASP GenAI Security Project, LLM06:2025 Excessive Agency https://genai.owasp.org/llmrisk/llm062025-excessive-agency/	Excessive functionality, permissions and autonomy; minimizing tools, least privilege, human approval, logging and rate limiting.
[16]	OWASP GenAI Security Project, LLM01:2025 Prompt Injection https://genai.owasp.org/llmrisk/llm01-prompt-injection/	Direct and indirect prompt injection, including instructions hidden in content a model reads.
[17]	Python documentation, collections module https://docs.python.org/3/library/collections.html	defaultdict, used for grouping.
[18]	Creative Commons, CC0 1.0 Universal https://creativecommons.org/publicdomain/zero/1.0/	The public-domain dedication that covers these lesson materials.

ABOUT THESE MATERIALS

The slides, lab, scripts, flash cards and this guide were written for the PrivacySafe Foundation Hack Your Future series. They are dedicated to the public domain under CC0 1.0^[18]. The Foundation names and logos are trademarks and are not included in that dedication. The dataset is made up. This lesson is independent of the projects named in the references.