NOTEBOOKS

CODE NOTEBOOKS

In order to make analysis **practically reproducible**, one should strive to make analysis

1. easy to interact with
2. easy to understand

code notebooks help achieve these goals via a **literate programming** format that interweaves

1. text
2. code
3. output

all together.

POPULAR NOTEBOOKS AND SOFTWARE

Often, code notebooks and their editing software are discussed as a single object. However one may separate

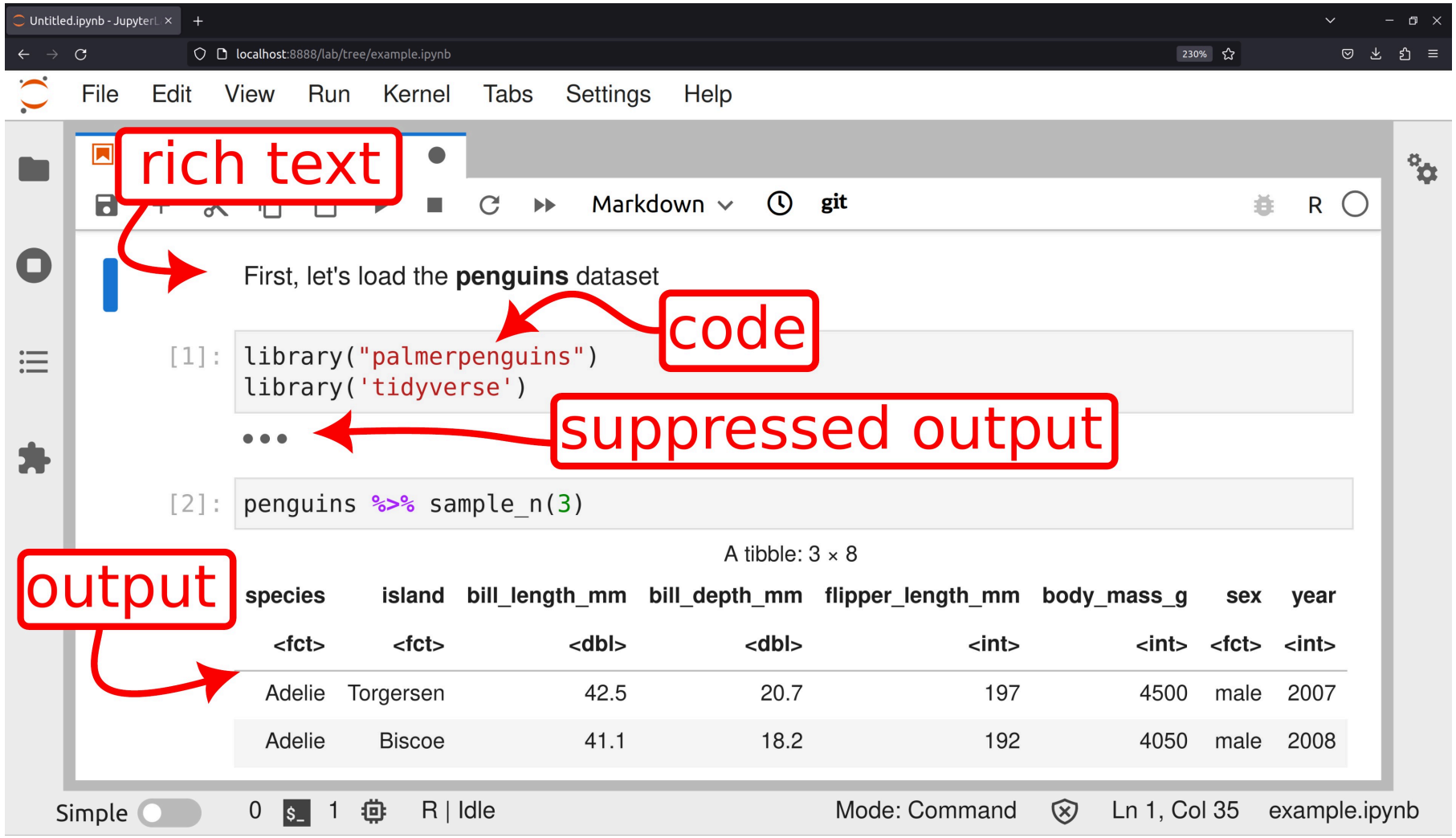
1. the notebook file format, from
2. the software used to interact with that format

Two most popular notebook formats/software:

1. “jupyter”: **jupyter lab** software and `.ipynb` format
2. “quarto”: **Rstudio** software and `.qmd` format

JUPYTER LAB EXAMPLE

An example of **jupyter lab**:



TEXT IN `markdown`

`jupyter` allows text to be written in `markdown` which is a light-weight markup language.

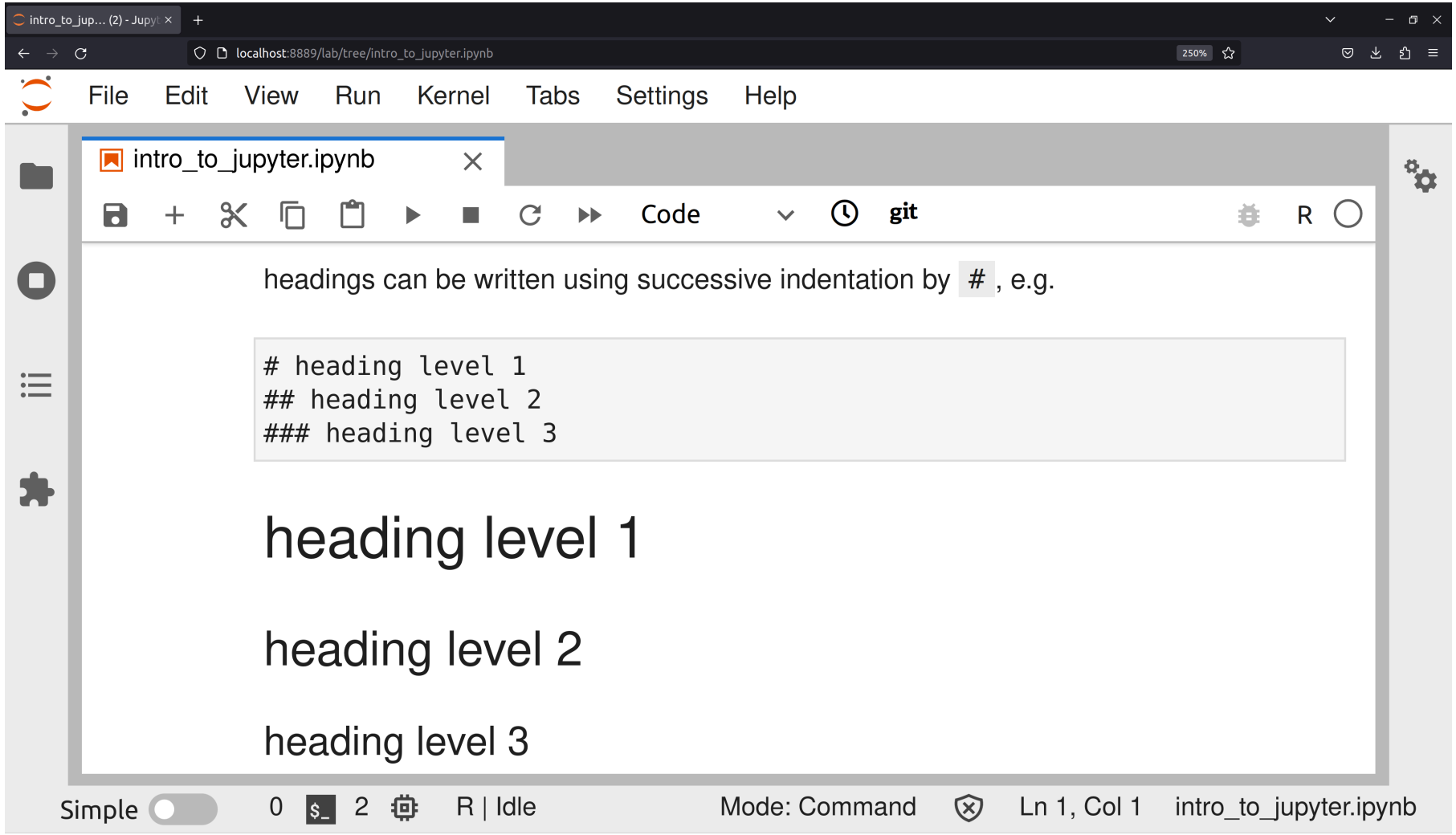
More-or-less: if you can display it on a webpage, you can write it in `markdown`. (Additionally, one can directly embed `html`)

One can also use extended markdown languages like `myst` which enables features like references, figures, bibliographies, …

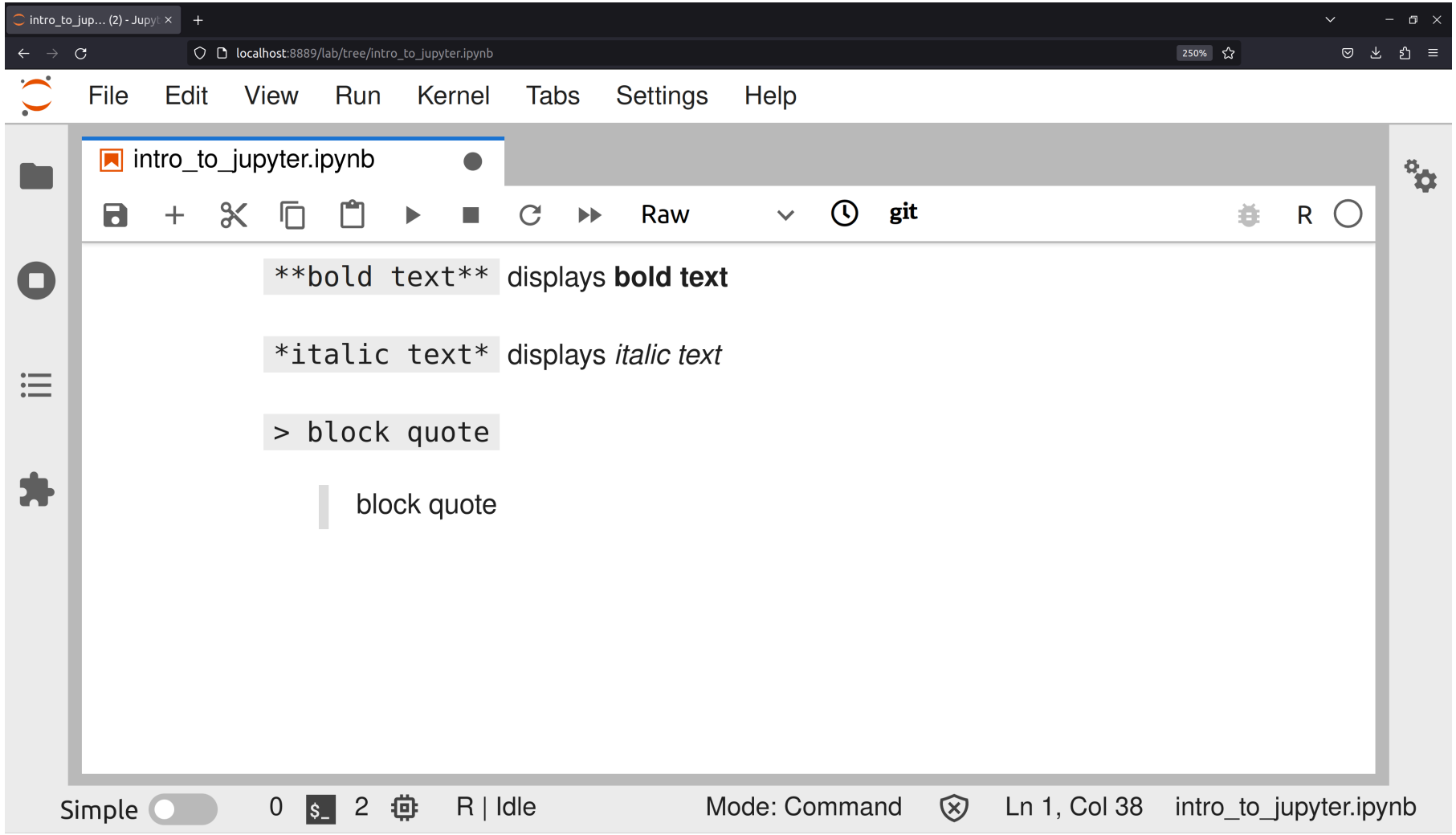
In any case, `jupyter`'s markdown enables rich-text commentary on **both the code and the output**

(In fact, this presentation is written in `markdown`)

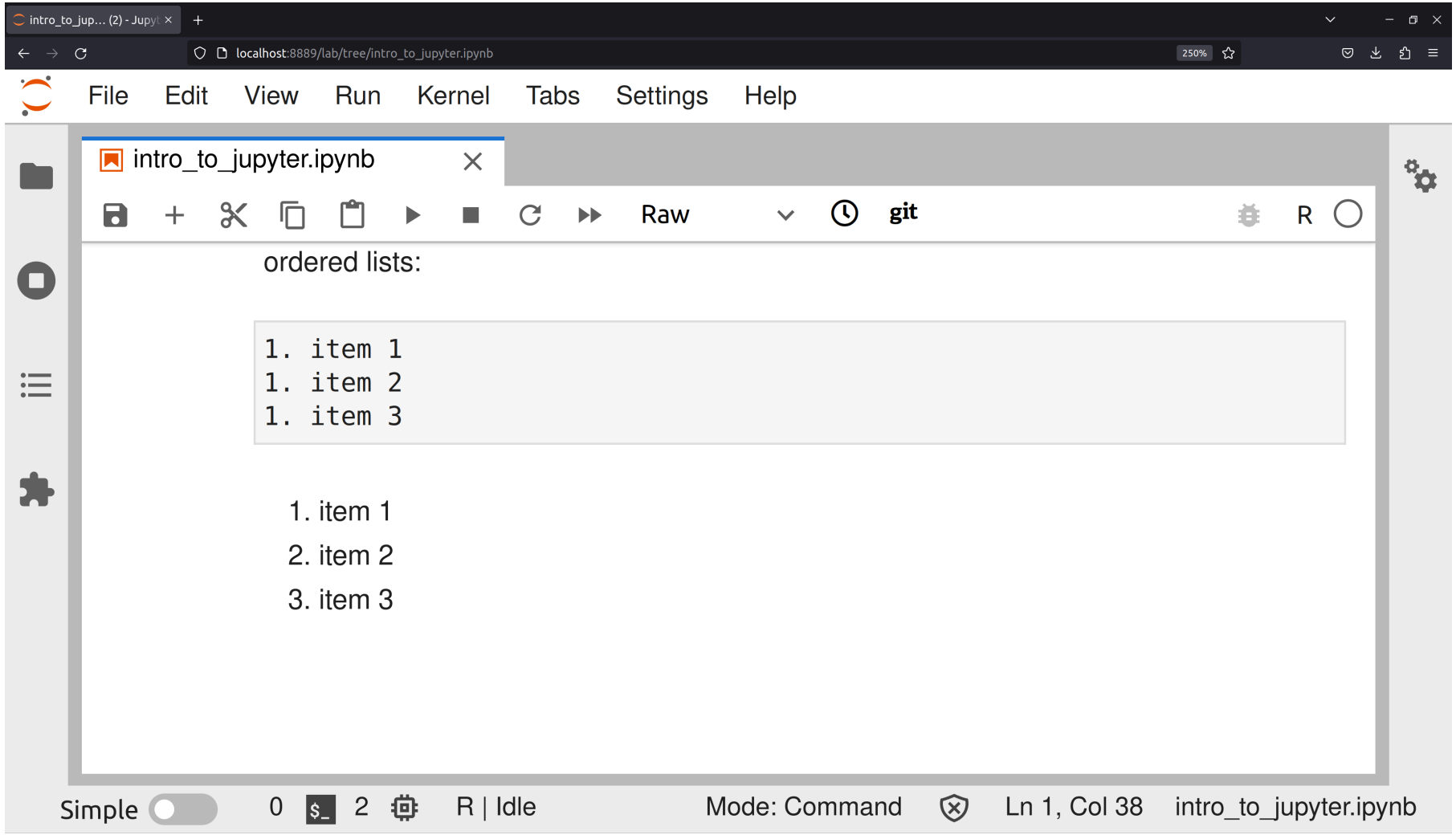
BASIC MARKDOWN



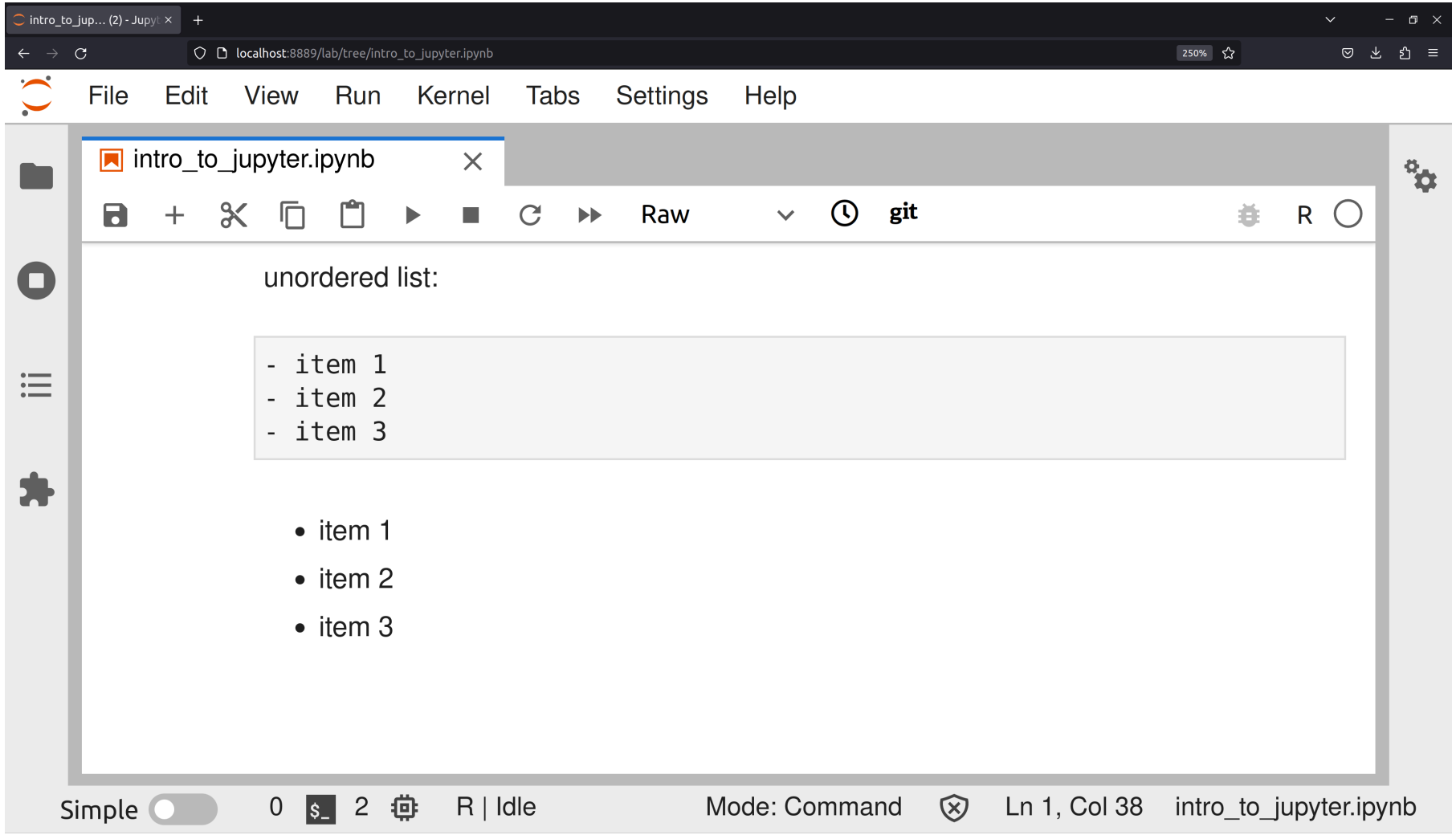
BASIC MARKDOWN



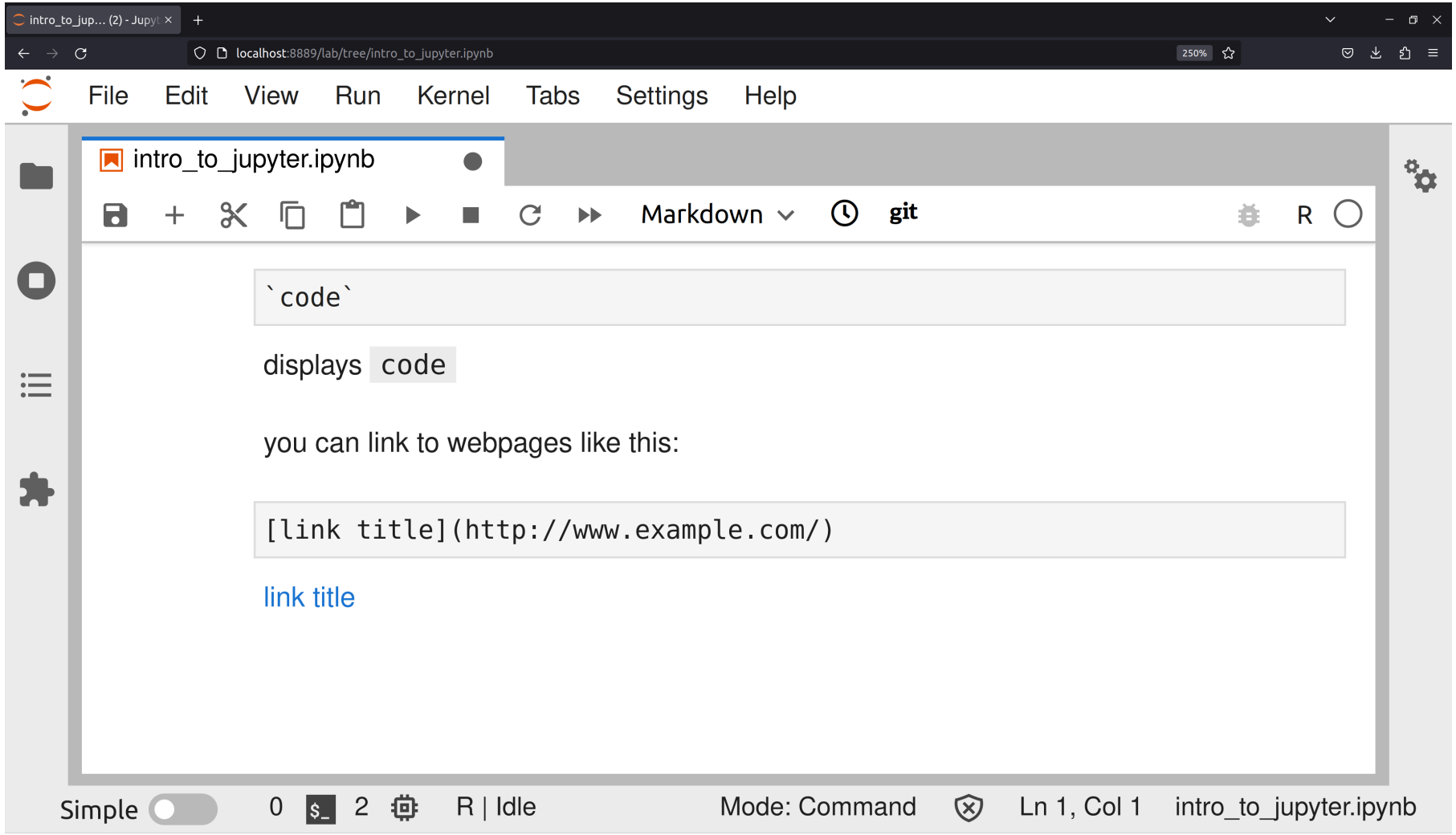
BASIC MARKDOWN



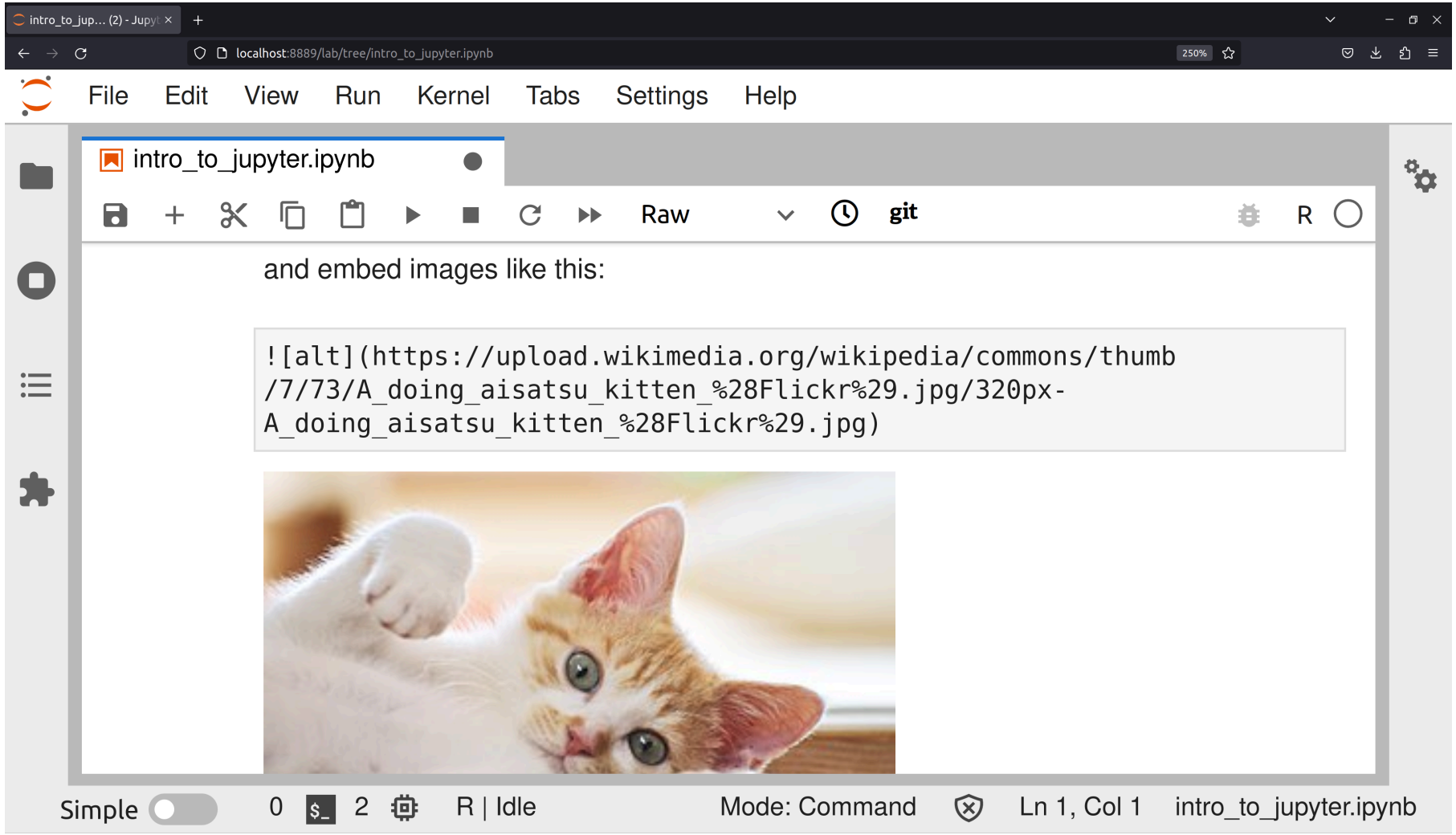
BASIC MARKDOWN



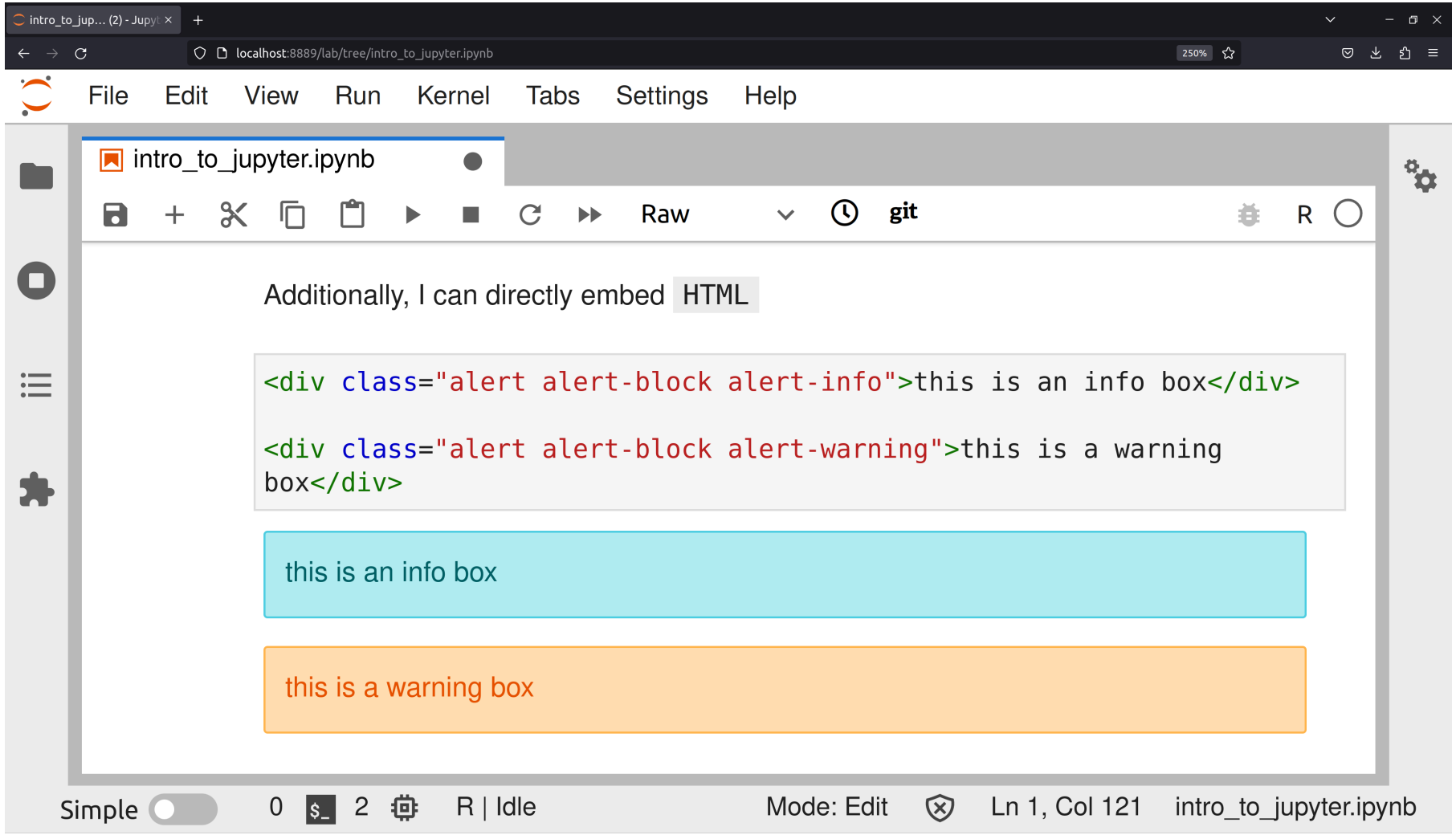
BASIC MARKDOWN



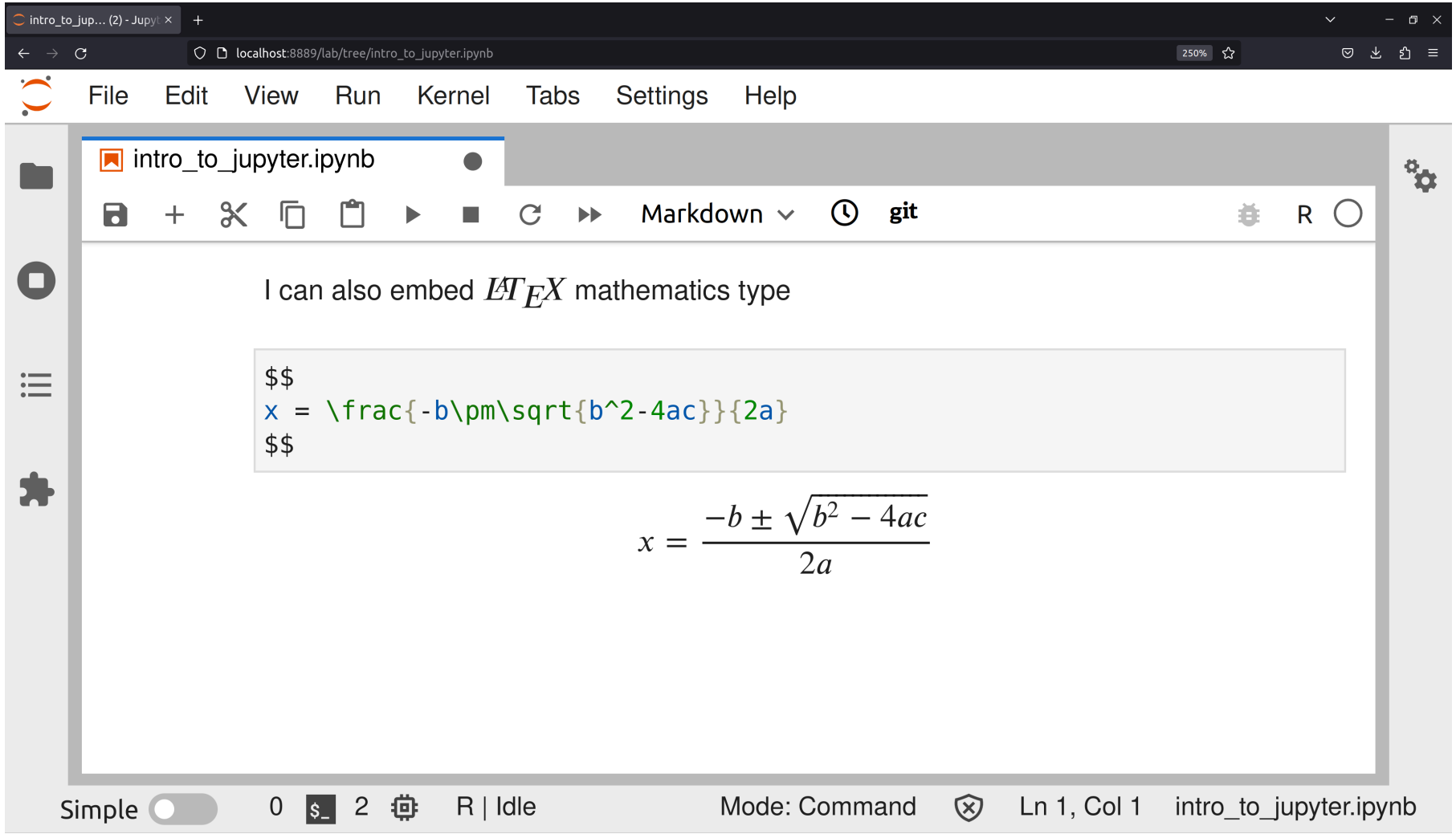
BASIC MARKDOWN



BASIC MARKDOWN

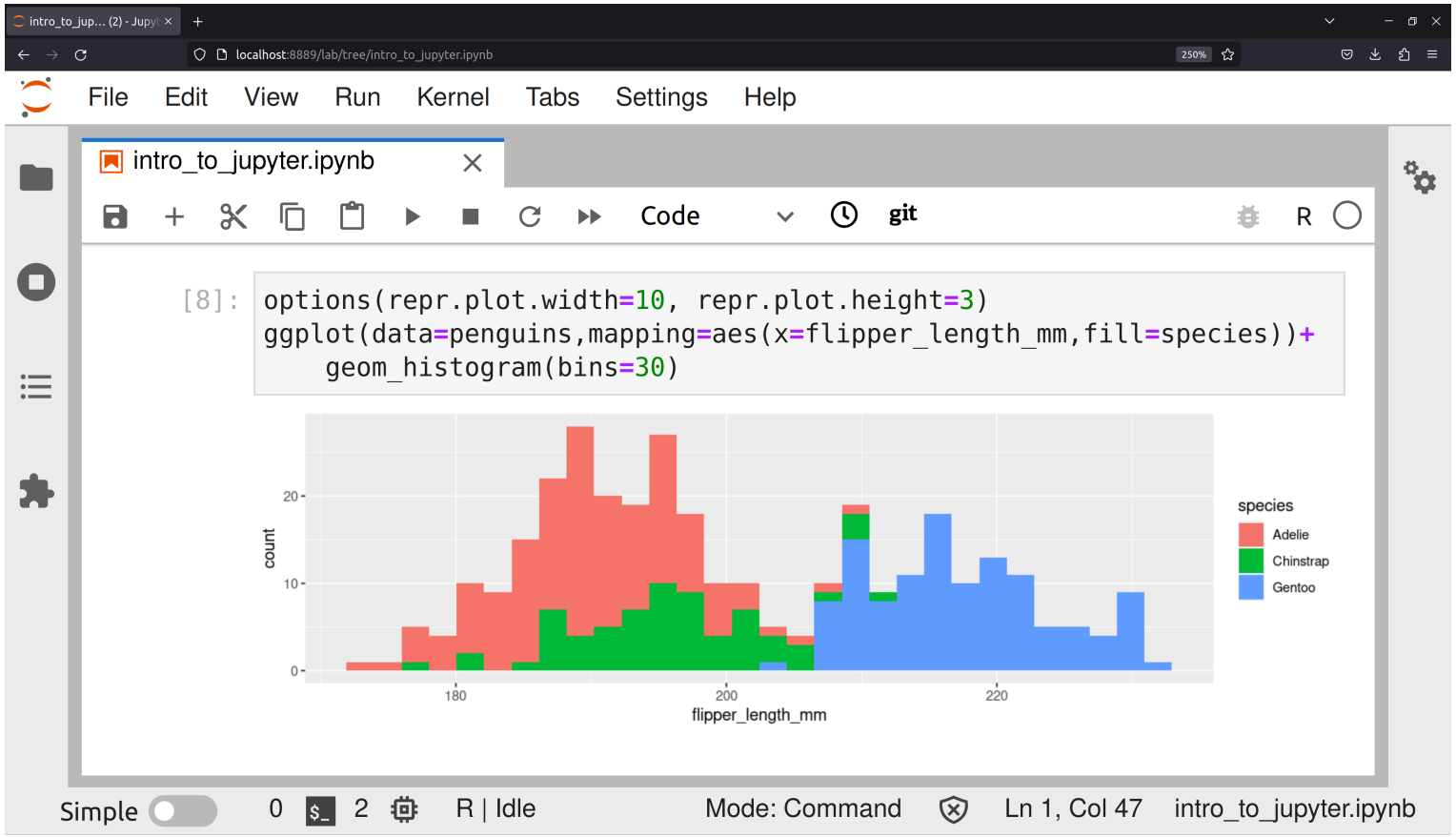


BASIC MARKDOWN



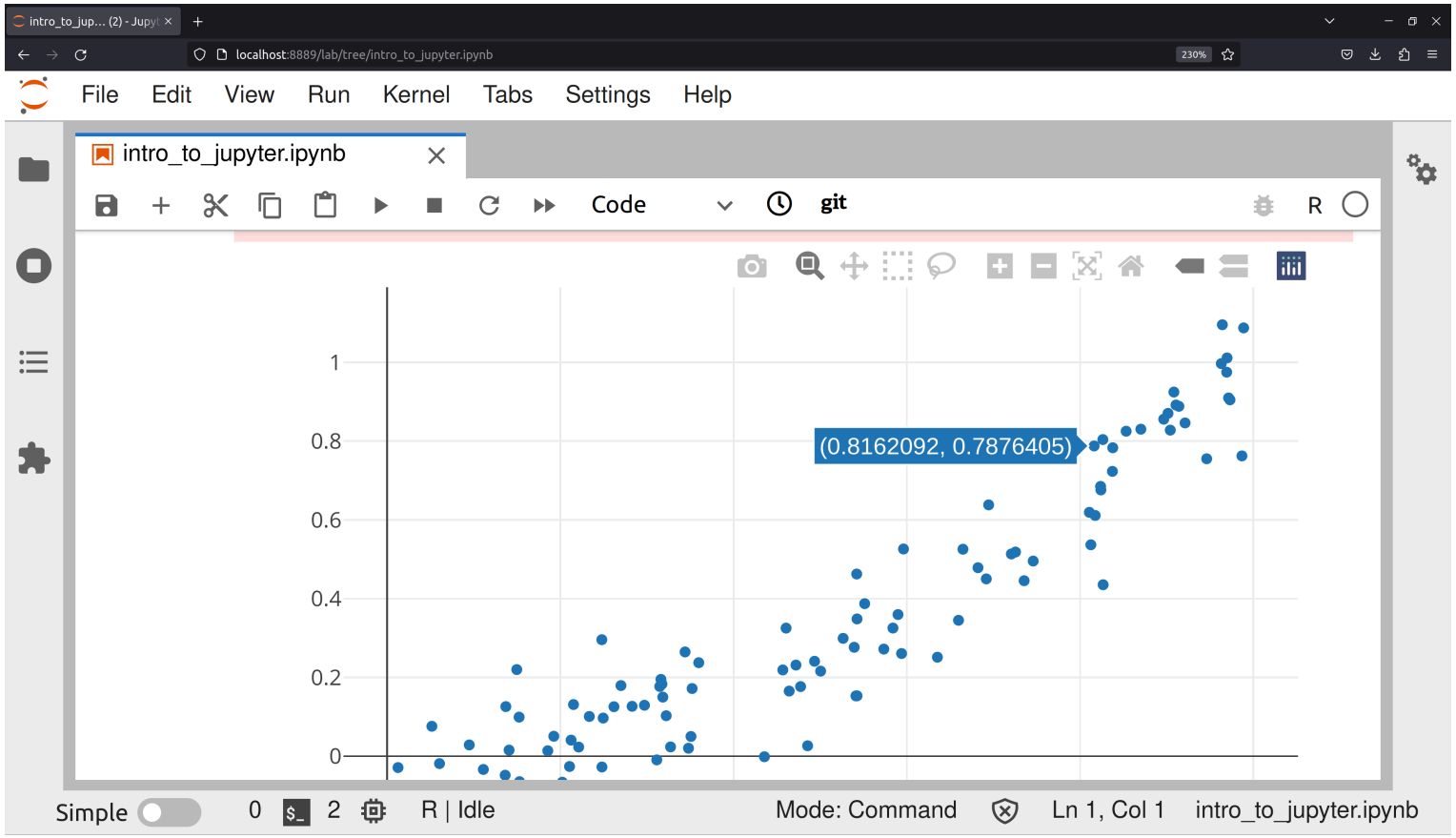
CODE: BASIC OUTPUT

Interweaved with the rich-text markdown commentary, one intersperses **code and in-line output**, e.g.,



CODE: INTERACTIVE WIDGETS

One can also embed **interactive widgets**, though this is somewhat notebook/language-specific. For example, in `R` one can use `htmlwidgets` or `plotly`, e.g.,



CODE: LANGUAGES

There are **many** language backends that jupyter can use. (These are called **kernels** in jupyter-speak). Jupyter lists well over 100 available kernels here including kernels for:

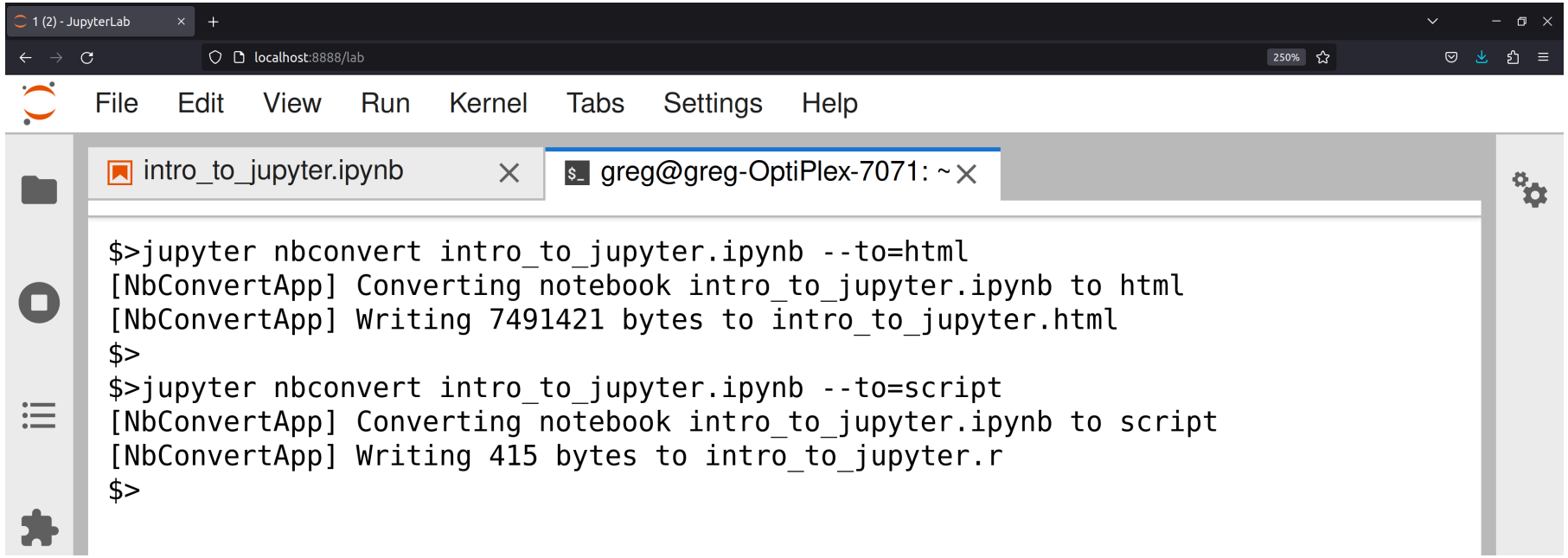
- `python`,
- `r`,
- `julia`,
- `stata`,
- `sql`,
- `octave`,
- `matlab`,
- `java`,
- `go`,
- `C`,
- ...

EXPORTING

One can **export** a **.ipynb** notebook using **jupyter** to many differnt formats like: `html`, `markdown` `pdf`, `reveal.js` html slides, ..., and even an executable script.

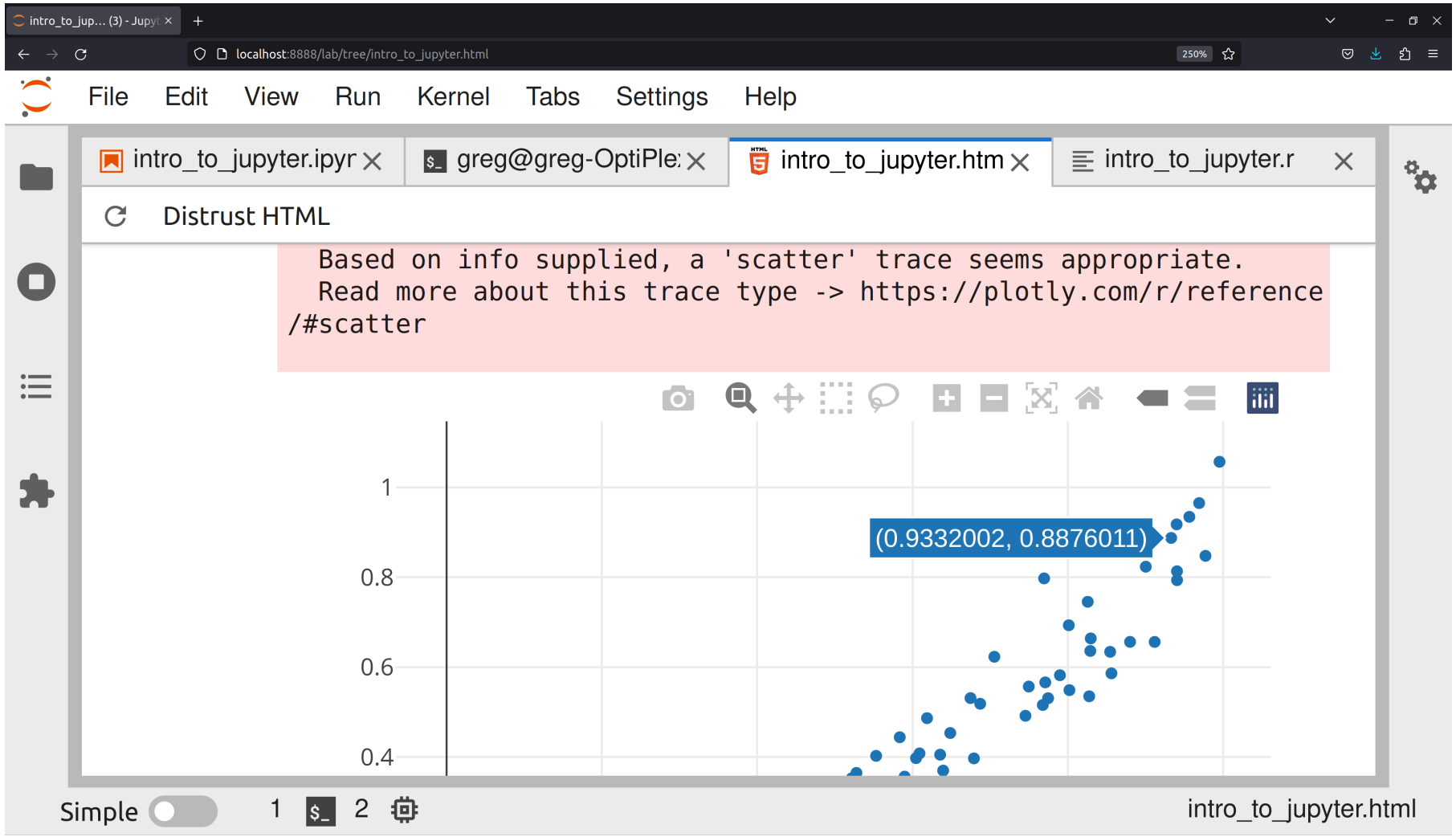
This can all be done using the command

```
jupyter nbconvert notebook.ipynb --to=format
```



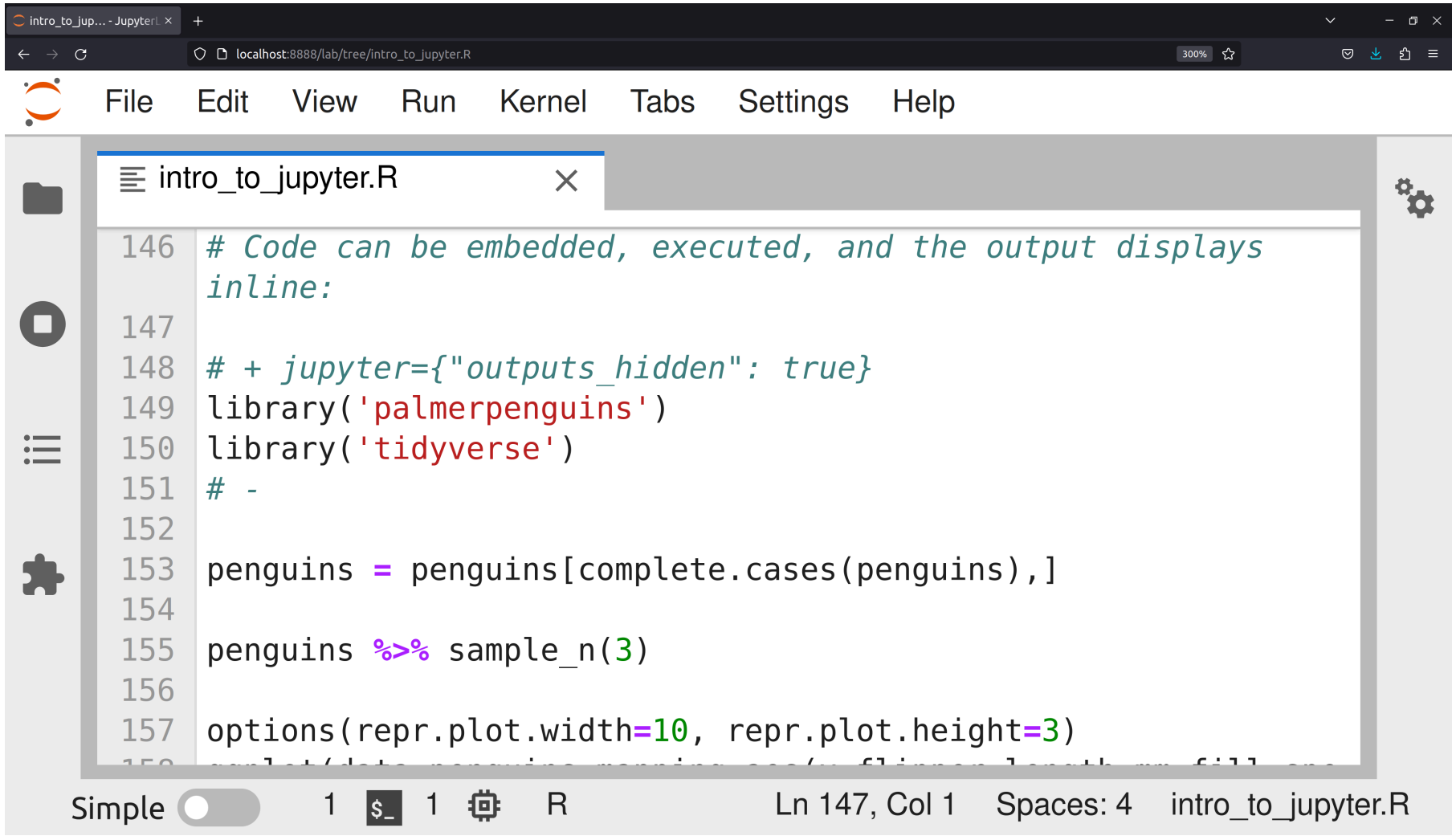
EXPORTING: HTML

Nicely, for a HTML export many of the interactive widgets still work!



EXPORTING: SCRIPT

Exporting as a script gives us basic executable script



NOTEBOOK INTEROPERABILITY

One can also use third-party tools to convert/maintain versions in **other notebook formats**.

For this, we find the tool `jupytext` to be invaluable.

We can install via

```
pip install jupytext
```

intro_to_jupyter.ipynb | greg@greg-OptiPlex-7071: ~

```
$>pip install jupytext
Collecting jupytext
  Downloading jupytext-1.14.7-py3-none-any.whl (299 kB)
     |▏                                | 10 kB 37.8 MB/s eta 0:00:
     |▎                                | 20 kB 12.7 MB/s eta 0:00:
     |▌                                | 30 kB 18.1 MB/s eta 0:00:
```

NOTEBOOK INTEROPERABILITY

`jupytext` **synchronously** propagates changes in the `jupyter` notebook (when the `.ipynb` is saved) to other formats like `markdown`, annotated scripts, `rmarkdown`, `quarto`, ...

via the command

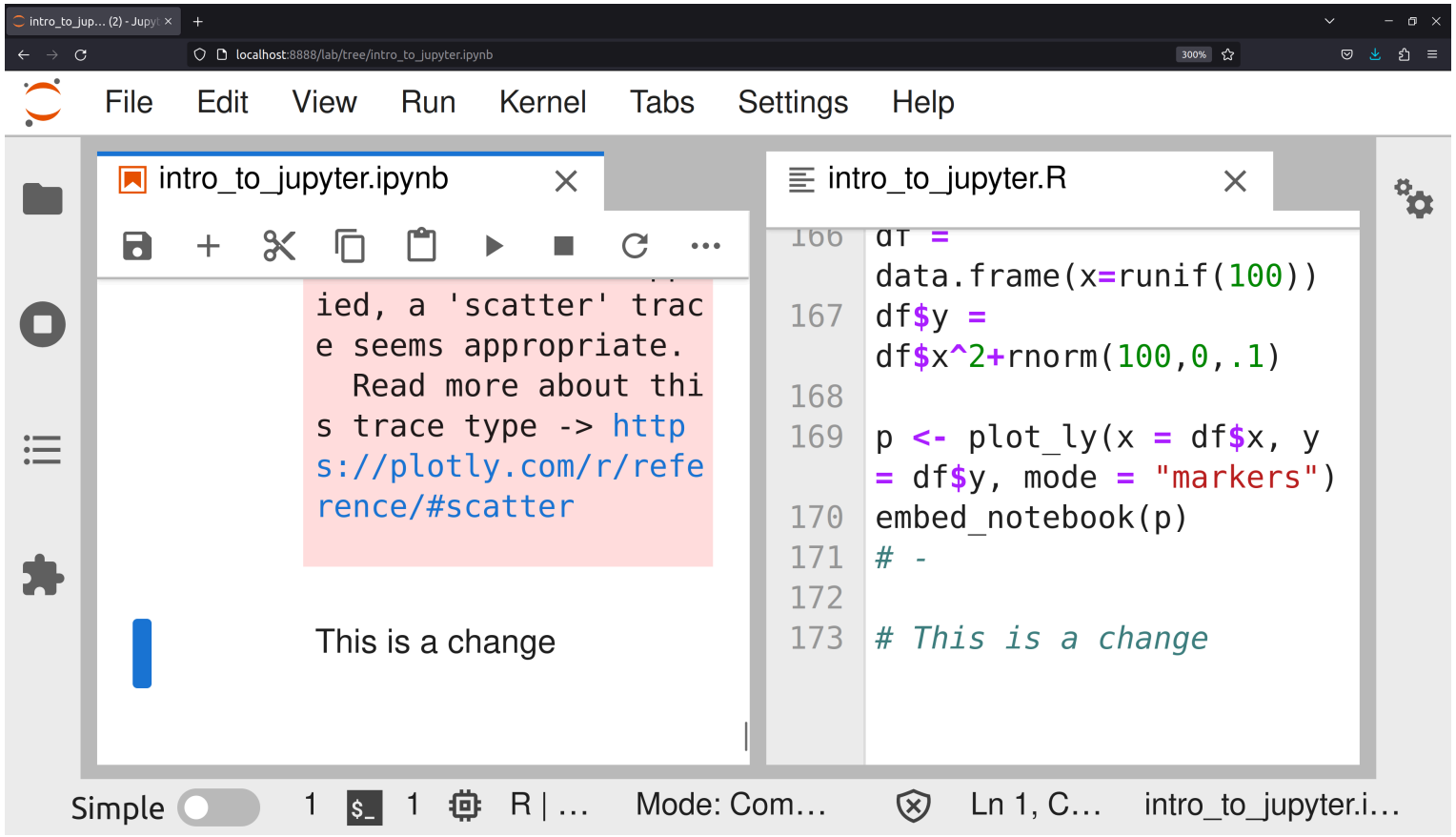
```
jupytext notebook.ipynb --set-formats=format1,format2,...
```

intro_to_jupyter.ipynb | greg@greg-OptiPlex-7071: ~

```
$>jupytext intro_to_jupyter.ipynb --set-formats=ipynb,R,qmd,md
[jupytext] Reading intro_to_jupyter.ipynb in format ipynb
[jupytext] Updating notebook metadata with '{"jupytext": {"format
s": "ipynb,R,qmd,md"}}'
[jupytext] Updating intro_to_jupyter.ipynb
[jupytext] Updating intro_to_jupyter.md
1.3.353
0.2.134
[jupytext] Updating intro_to_jupyter.qmd
[jupytext] Updating intro_to_jupyter.R
```

JUPYTEXT CHANGE

Changes are **synchronously** propagated from `.ipynb` to the other formats and vice-versa. (Just be wary of editing two files at once!)



EXPORTING AND INTEROPERABILITY

- **Exporting** via `nbconvert` mostly immortalizes analysis for display e.g. as `HTML` or a `PDF`
- We can also export executable scripts and (via tools like `jupytext`) other notebook formats for
  - deployment in **production**,
  - deployment in a **non-interactive cluster**,
  - or making analysis more amenable for **version control**
- Exporting to display formats and other notebook formats also makes them more **sharable** and thus analysis more easily **reproducible**
  - we provide same analysis in many notebook formats so users have choice
  - this is probably a good final step before sharing analysis

QUARTO: LATEST R STUDIO NOTEBOOK FORMAT

- `quarto` is the latest notebook format from Posit (RStudio).
- Largely, it supplants R notebooks and R markdown formats (and is back-compatable).
- Technically, `quarto` is a command line publishing tool. This means:
  - can be run via the command line
  - don't need `R` or `Rstudio` **at all** to run `quarto`
  - can be used to weave documents in `python` or `julia` (can even use a `jupyter` back-end)
- Nonetheless, still has great integration in `Rstudio` with a nice WYSIWYG editor.
- Rstudio/`quarto` are a similar, nice alternative to jupyter lab/jupyter notebooks (in my opinion)
- can edit `quarto` documents in `jupyter`

QUARTO AND JUPYTER

`quarto`/Rstudio and `jupyter`/jupyter lab are both great notebook formats/tools. One major difference:

- `quarto` uses markdown-like `.qmd` while `jupyter` uses JSON `.ipynb`
  - `.qmd` are probably easier for version control (e.g. `git`) and other editors (e.g. VSCode)
- `quarto` is more geared towards publishing polished formats (e.g. figures, tables, references, …)
  - `myst` extension to markdown can enable this for `jupyter`
- `.qmd` doesn't store output, `.ipynb` encodes/stores output
- **A useful idiom**: do your analysis in `jupyter` and mirror into several formats e.g. ipynb, md, R, qmd,…

NOTEBOOKS AND REPRODUCIBILITY

Why do notebooks help **reproducibility**:

1. literate programming: interweaving code/commentary/output
    - allows rich commentary on code, output
    - develops a narrative that is easy to read
    - document diagnostic/exploratory/micro-decision analysis
2. keeps commentary/output close to code
    - good tool for playing with code, immediately observing output
3. good for showcasing results and interoperability
    - can be converted to many sharable formats (html, pdf, …)
    - can convert **among** the notebook formats
4. creates a reproducible record
    - code automatically generates results from data
    - this forces documentation on how the results were produced
5. promotes good code organization via chunking
6. software/formats not proprietary, easy to distribute

SOME POTENTIAL DOWNSIDES

While code notebooks can be great, there are some **potential issues**, including:

1. chunks can be run in non-sequential order, making them not reproducible (soln: re-run all analysis at the end sequentially)
2. saved format of notebook may make version control difficult (soln: jupytext)
3. not great for non-interactive environments (soln: jupytext)
4. conversion among various formats isn’t 100% fool-proof

DISCUSSION

- Do you use notebooks regularly? Might you, now?
- Where do you find notebooks to be helpful?
- Where do you find notebooks **not** to be helpful?