Code Notebooks

Exercise 0: Install Jupyter

Install jupyter. First, you need an installation of python/pip:

- https://www.python.org/downloads/

Then, install jupyter following https://jupyter.org/install:

```
pip install jupyterlab
jupyter lab
```

To use R, run these commands in an R session

```
install.packages("devtools")
devtools::install_github("IRkernel/IRkernel")
IRkernel::installspec()
```

Alternatively, you can use the interactive web-apps

- https://jupyter.org/try
- https://colab.research.google.com/

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

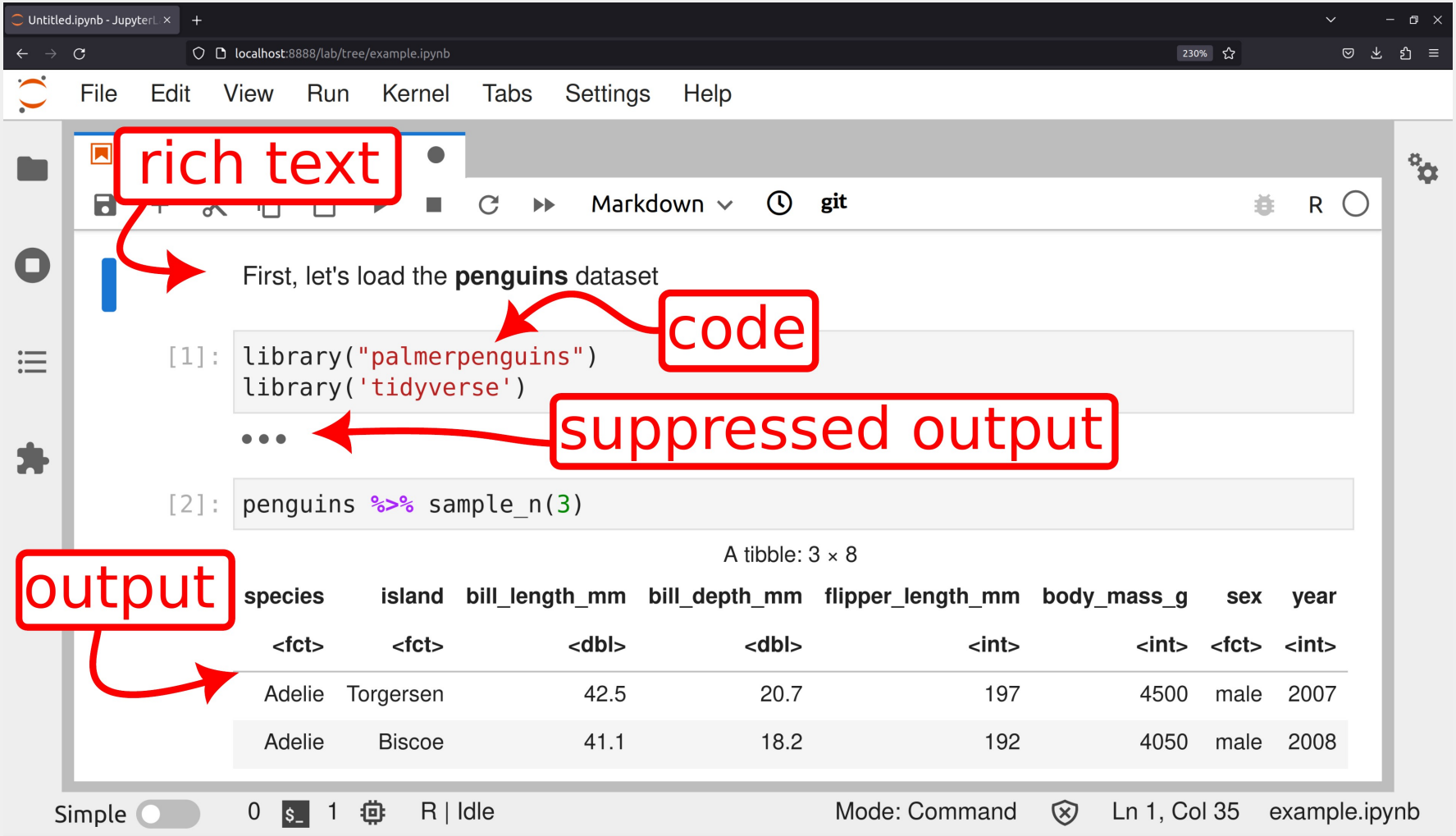
Two most popular notebook formats/software:

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

Primarily, we will give a brief overview of **jupyter** and discuss its advantages for reproducibility.

Jupyter Lab Example

An example of **jupyter lab**:



Text in markdown

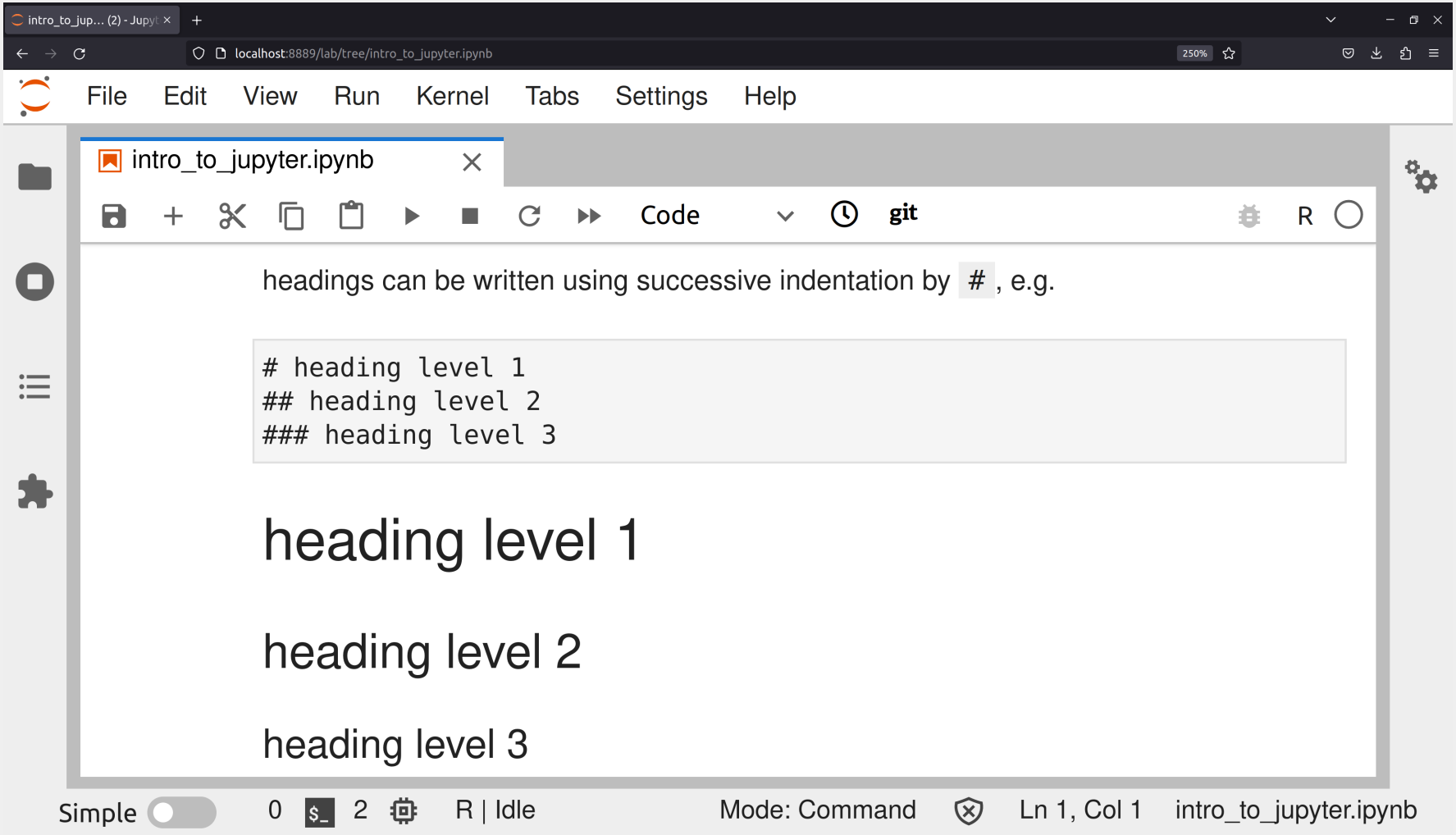
`jupyter` allows text to be written in `markdown` which is a light-weight markup language.

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

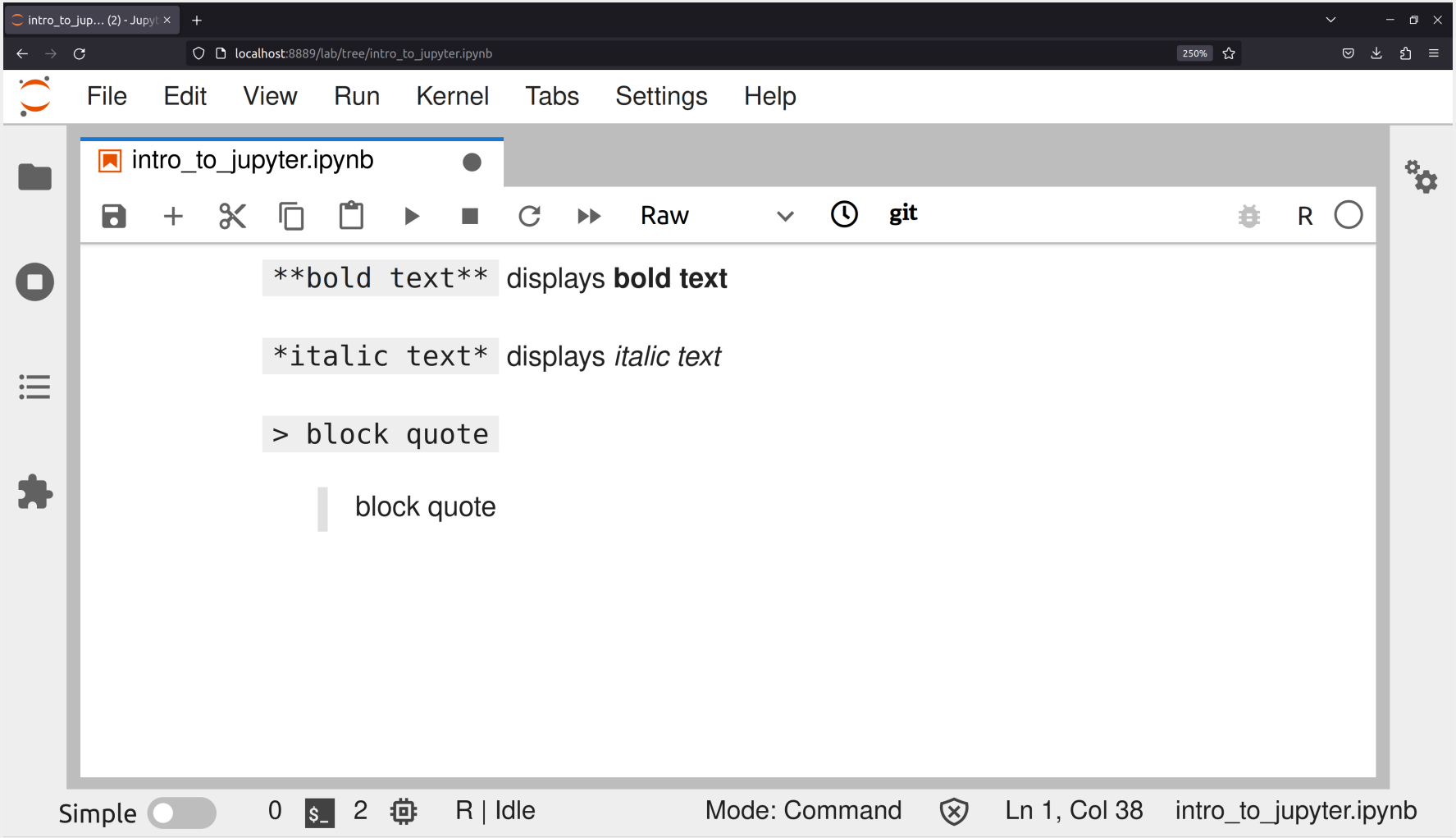
One can also use extended markdown languages like `myst` which enables features like references, figures, bibliographies, ...

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

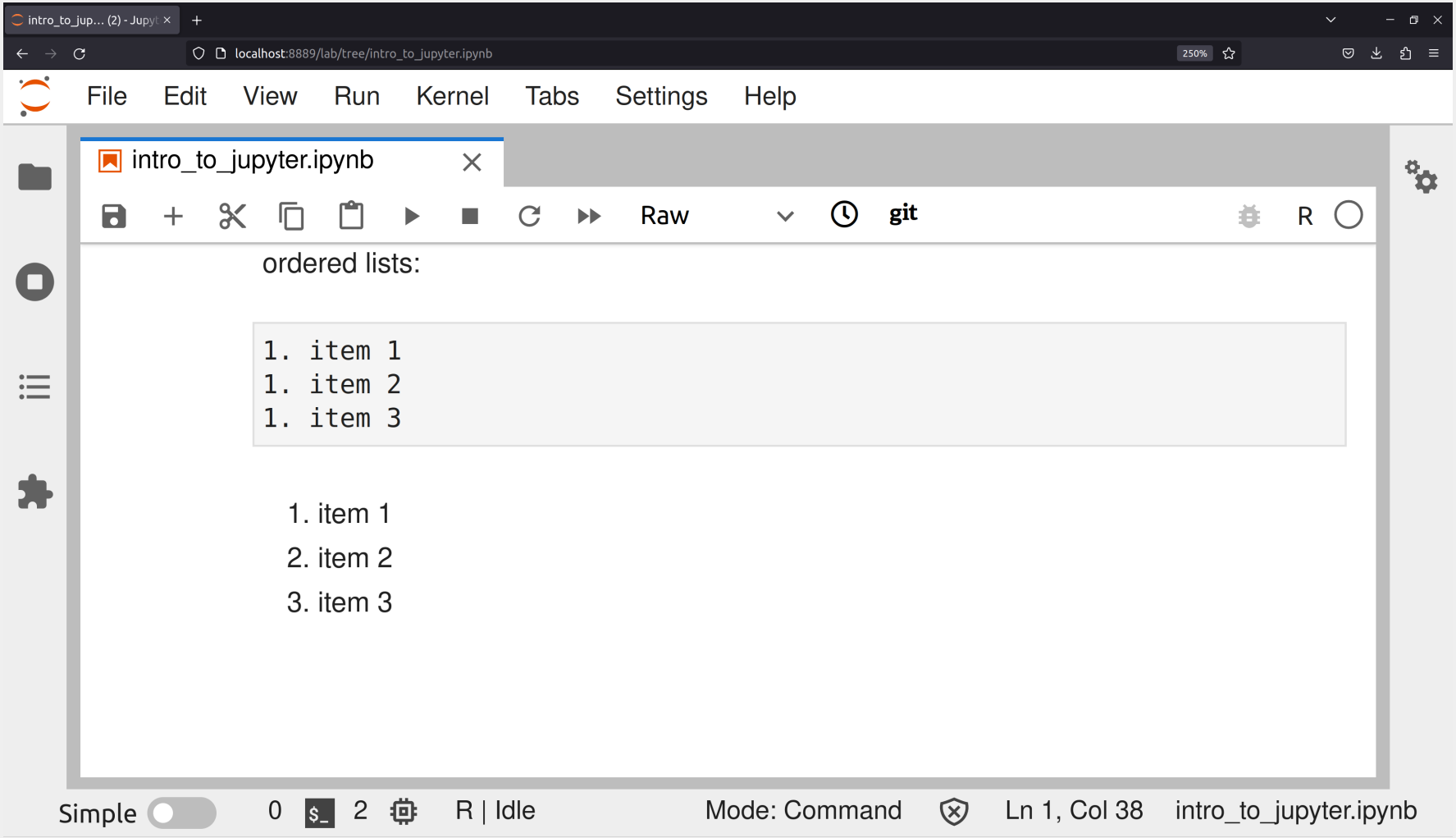
Basic Markdown



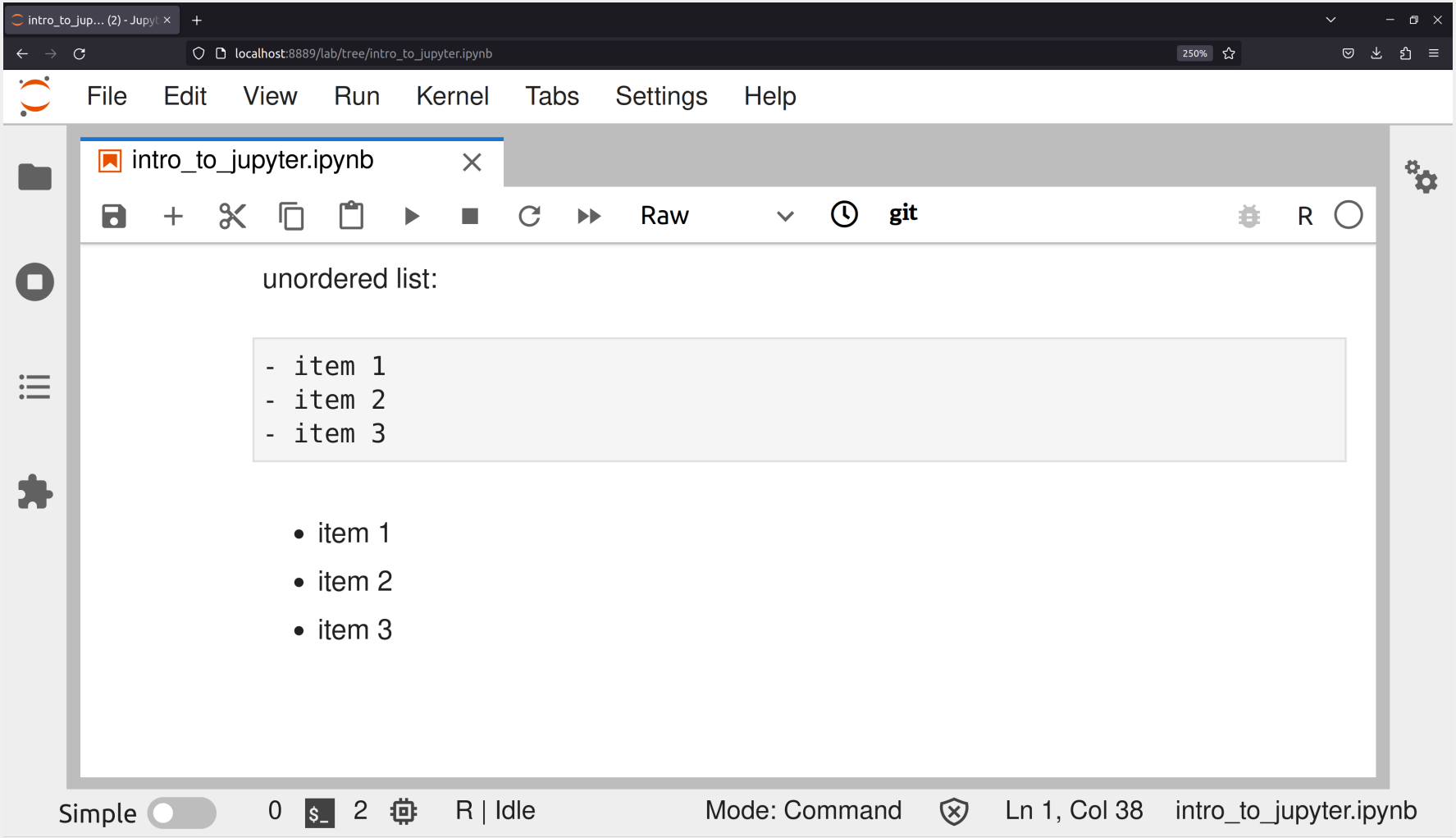
Basic Markdown



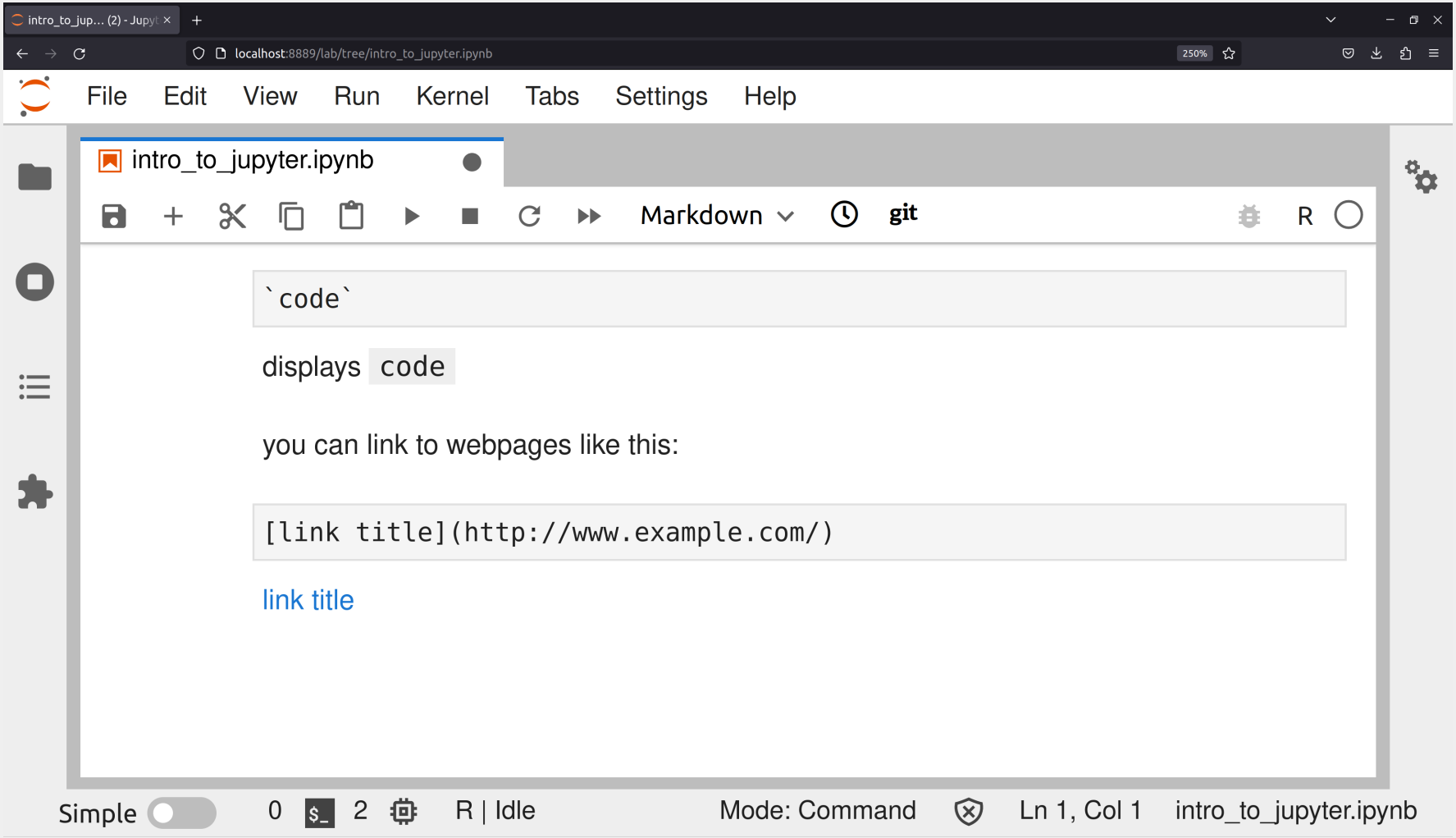
Basic Markdown



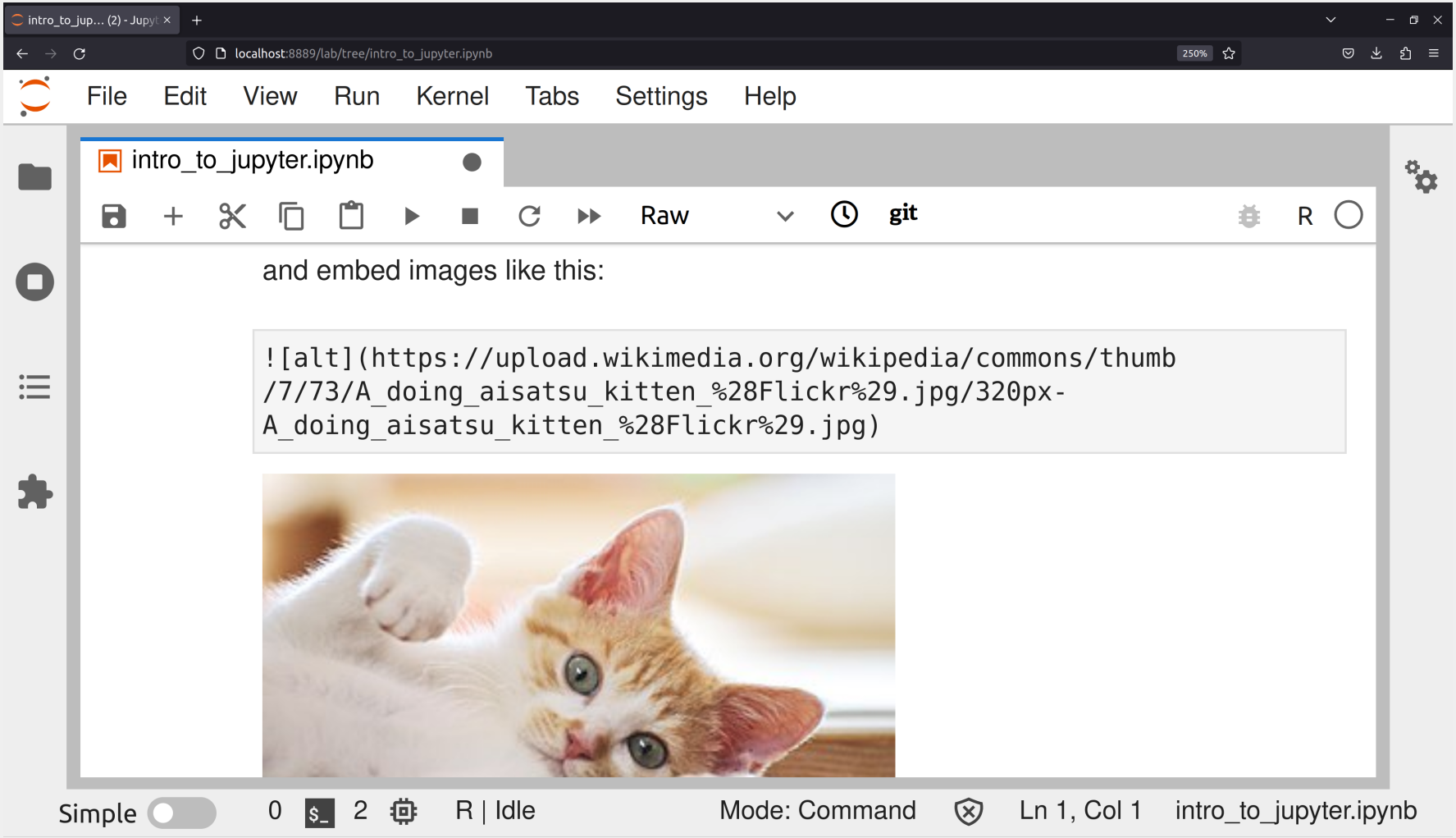
Basic Markdown



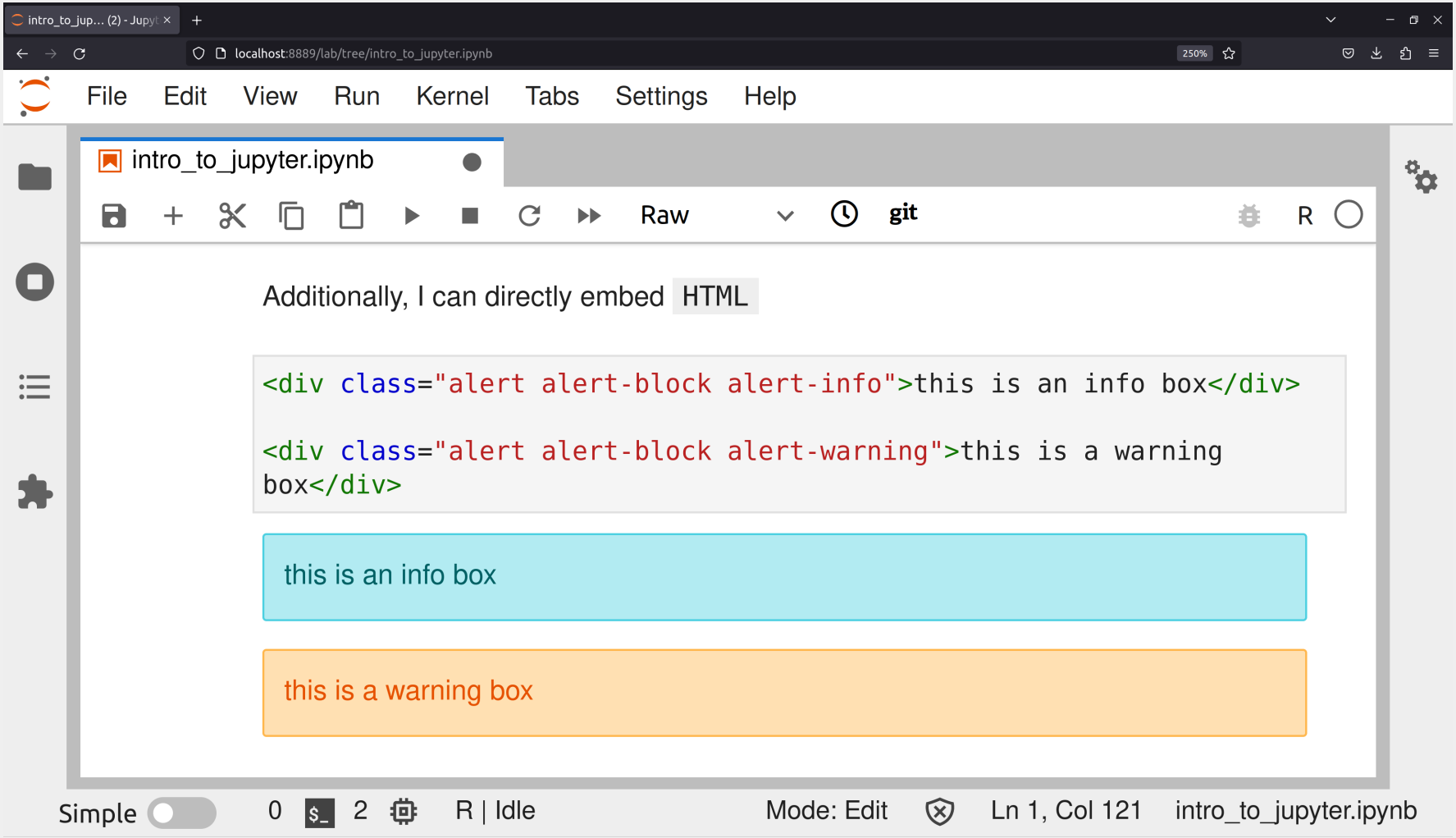
Basic Markdown



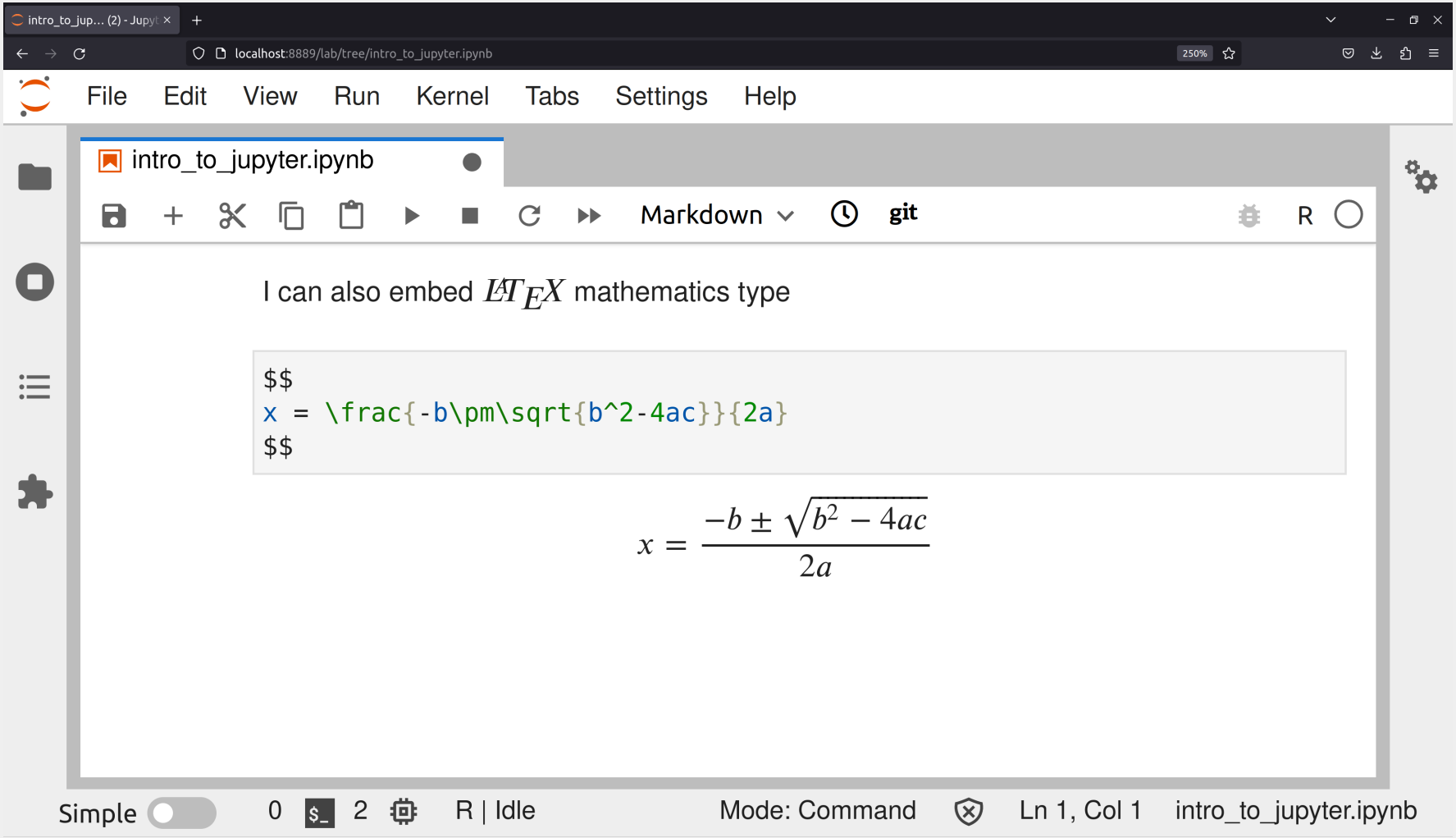
Basic Markdown



Basic Markdown



Basic Markdown



Exercise 1: Add Markdown to a Jupyter notebook

Load up jupyter and start a new notebook `.ipynb` file.

In several different cells add markdown and render it. Your markdown should include:

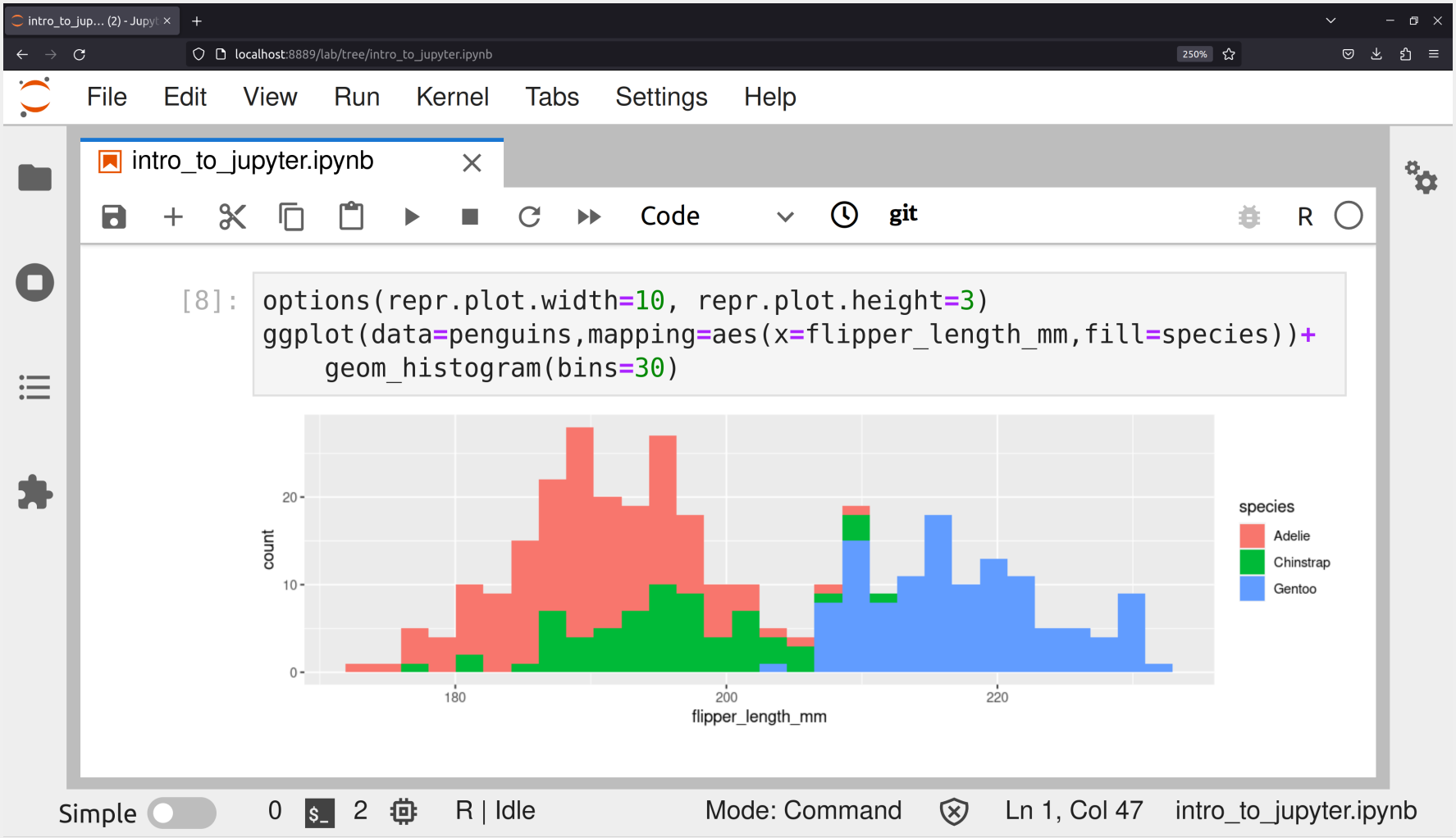
- a heading
- bold and italic text
- a numbered and un-numbered list of items
- displayed pseudocode
- a link to an external webpage
- an image
- embedded math in LaTeX

After writing these blocks of markdown, re-arrange them

- copy and past one
- delete one
- move one

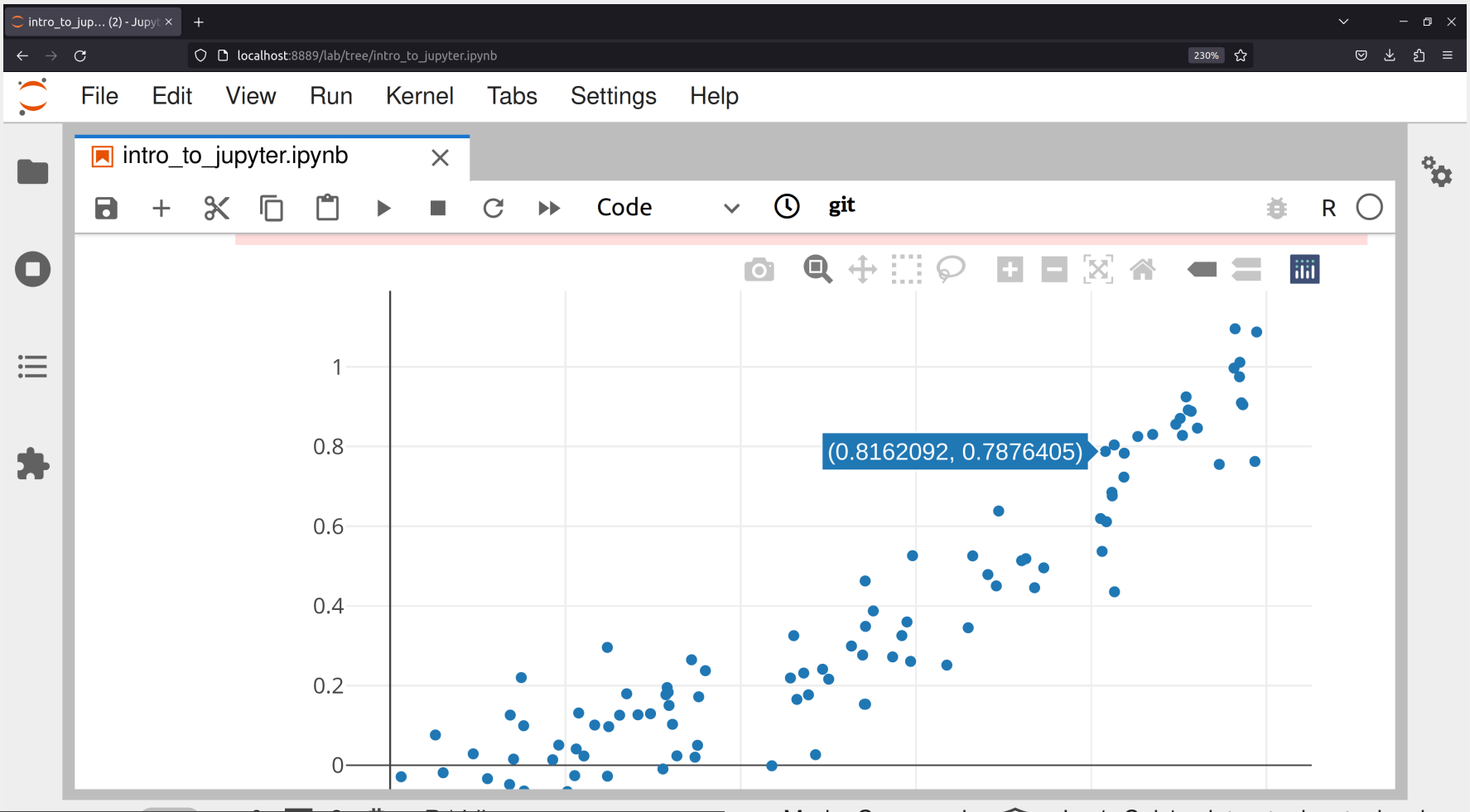
Code: Basic Output

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`,

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`,

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`,

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`,

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`,

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`,

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`,

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`,

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`,

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`,

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`,
- ...

Exercise 2: Add Some Code

Add some code to your notebook file. Here, we'll use the palmer penguins data set. You can download it and load it up with

```
install.packages("palmerpenguins")
library('palmerpenguins')
head('palmerpenguins')
```

Explore this data:

- make a histogram of flipper length for each species
- add some markdown commentary to your plots
- install `plotly` via `install.packages('plotly')` and then add an interactive scatter plot:

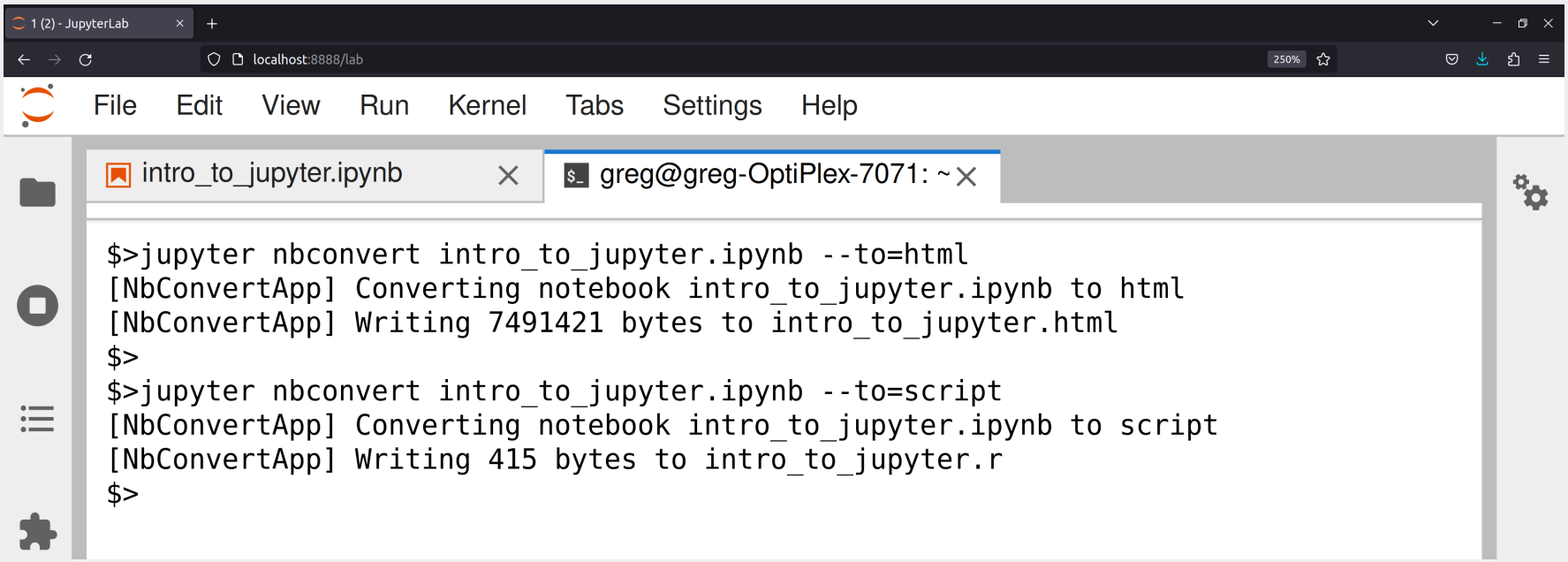
```
p <- plot_ly(x = ____, y = ____,
  mode = "markers", type = "scatter")
embed_notebook(p)
```

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.

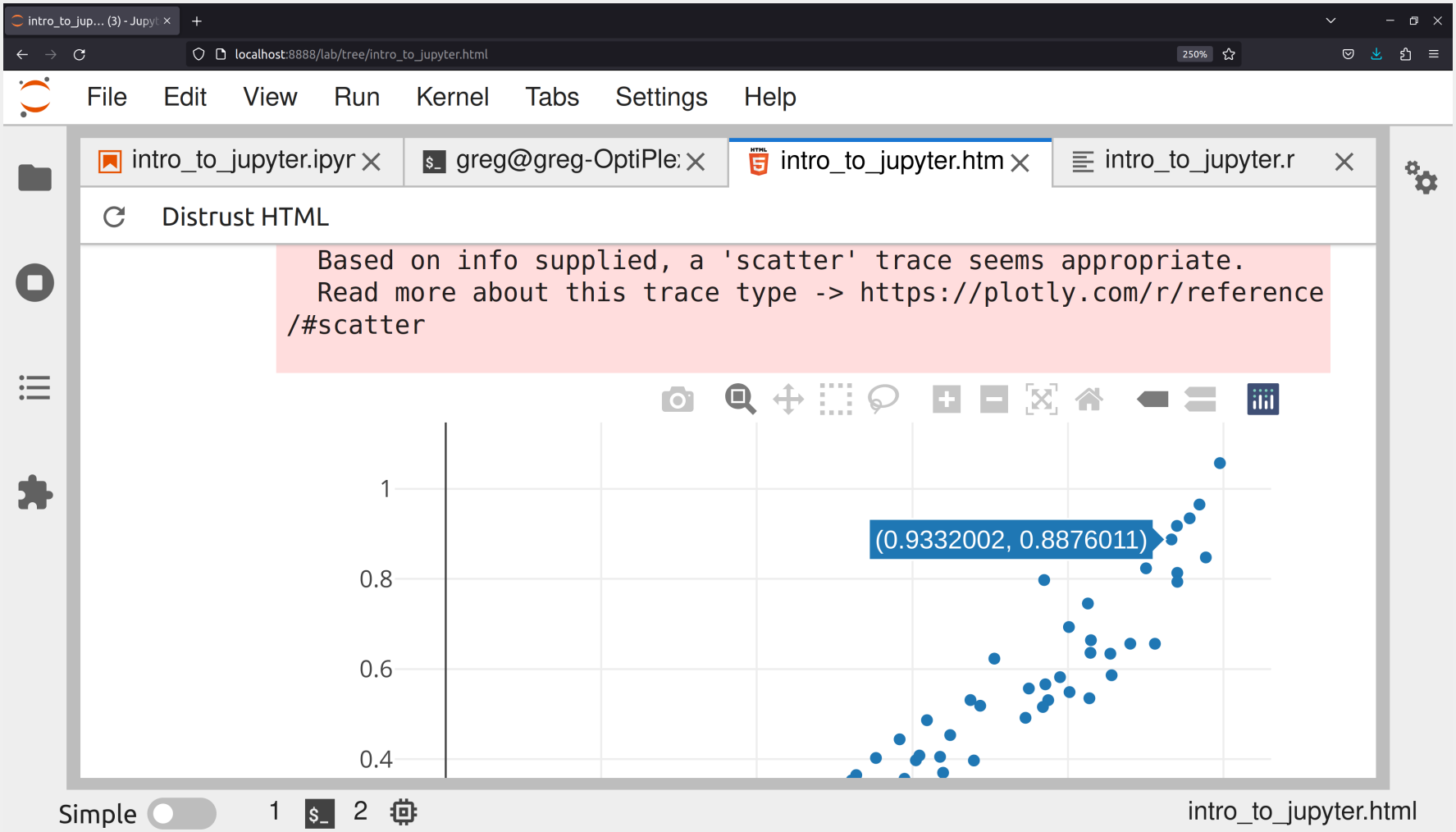
This can all be done using the command

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



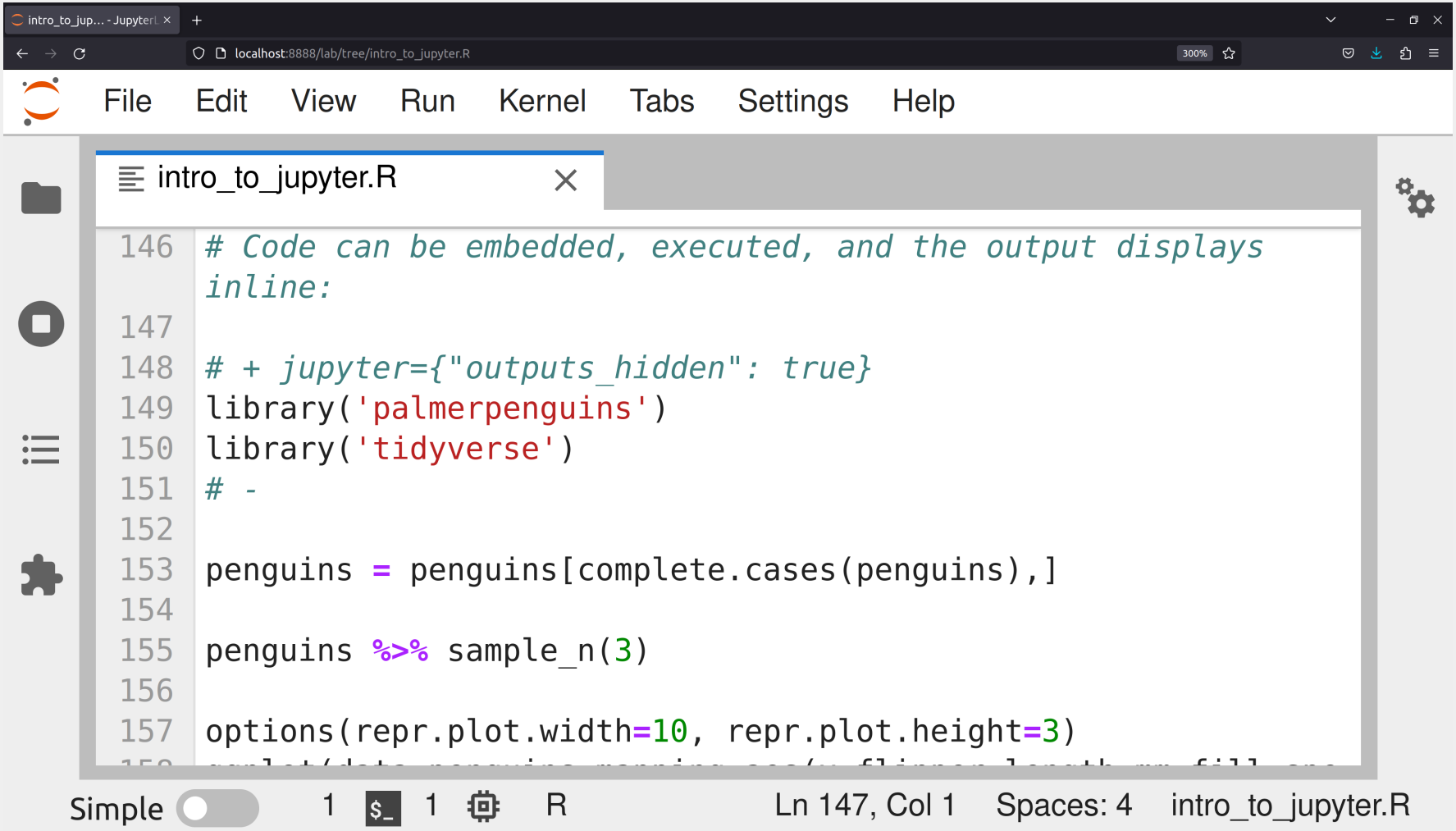
Exporting: HTML

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



Exporting: Script

Exporting as a script gives us basic executable script



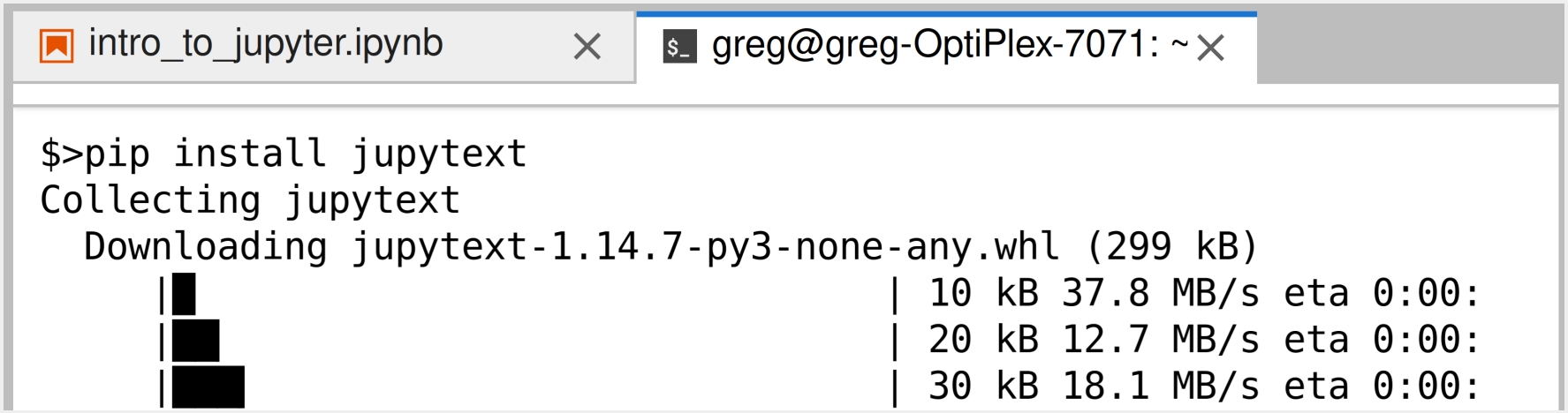
Notebook Interoperability

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
```



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

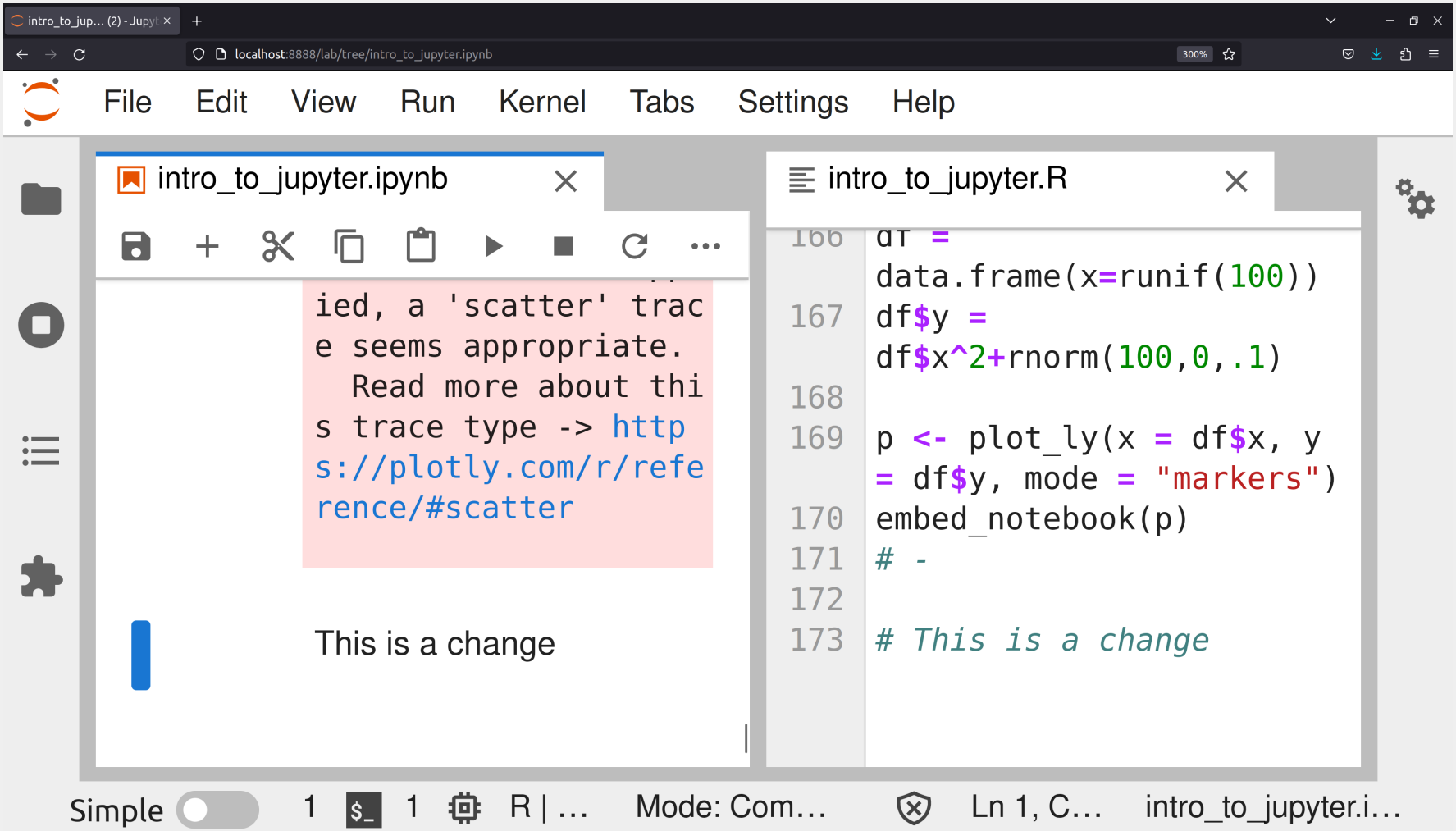
```
jupytext notebook.ipynb --set-formats=format1,format
```

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

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

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**,

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**,

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 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**

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

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

Exercise 3: Exporting

Export your notebook to:

- `html`, then inspect the interactive html
- a `.R` script, then try running the script separately

Install jupytext via

```
pip install jupytext
```

Mirror your notebook into

- an `.R` script,
- a `.md` markdown file

Open up the .R script and edit it. Go back and re-load the `.ipynb` jupyter notebook and observe the changes.

quarto: latest R studio notebook format

- `quarto` is the latest notebook format from Posit (RStudio).

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).

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:

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

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

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)

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.

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)

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

Notebooks and Reproducibility

Why do notebooks help **reproducibility**:

1. literate programming: interweaving code/commentary/output
   - allows rich commentary on code, output

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

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

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

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

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

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, ...)

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

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

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

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

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

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)

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)

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)

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

Exercise 4: Putting It All Together

1. Load up the `plates.csv` data from Bray et al.
2. Using `jupyter`, conduct some exploratory analysis.

- A good example of an exploratory analysis is to conduct PCA on the data and visualize the first several principal components, coloring the data using the metadata. Some data-cleaning might be in order.
- Make sure to document your code and use the markdown text to write comments on the analysis.

1. Export your analysis as a HTML document using jupyter.
2. Using `jupytext`, mirror the analysis to a `.R` script.

- Run the script independently after exporting to it.

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?