Code Notebooks

Module 2: 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

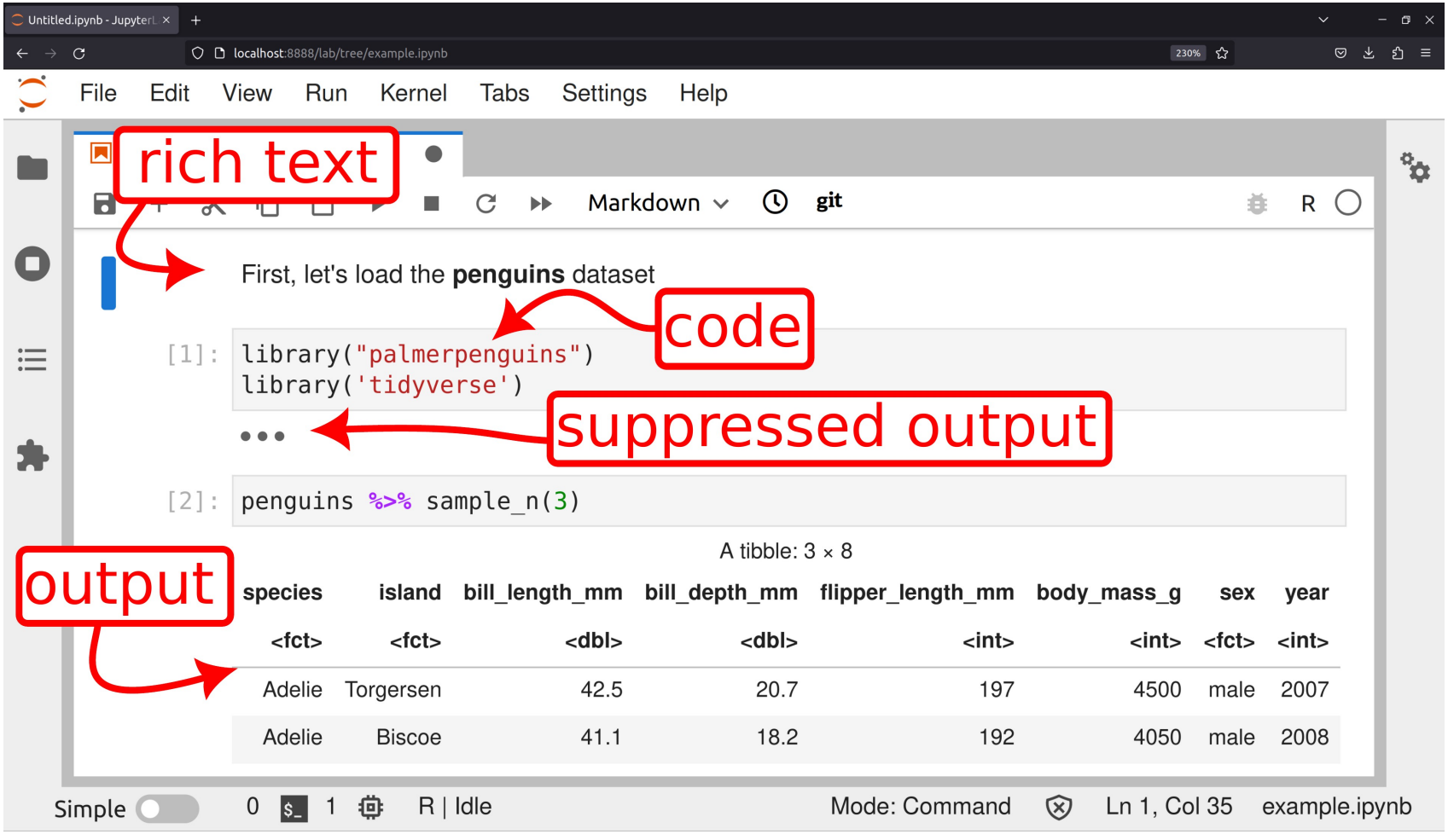
Two most popular notebook formats/software:

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

We will give a brief overview of these, and then turn to comparing them and their advantages for reproducibility.

Jupyter Lab Example

An example of **jupyter lab**:



Text in `markdown`

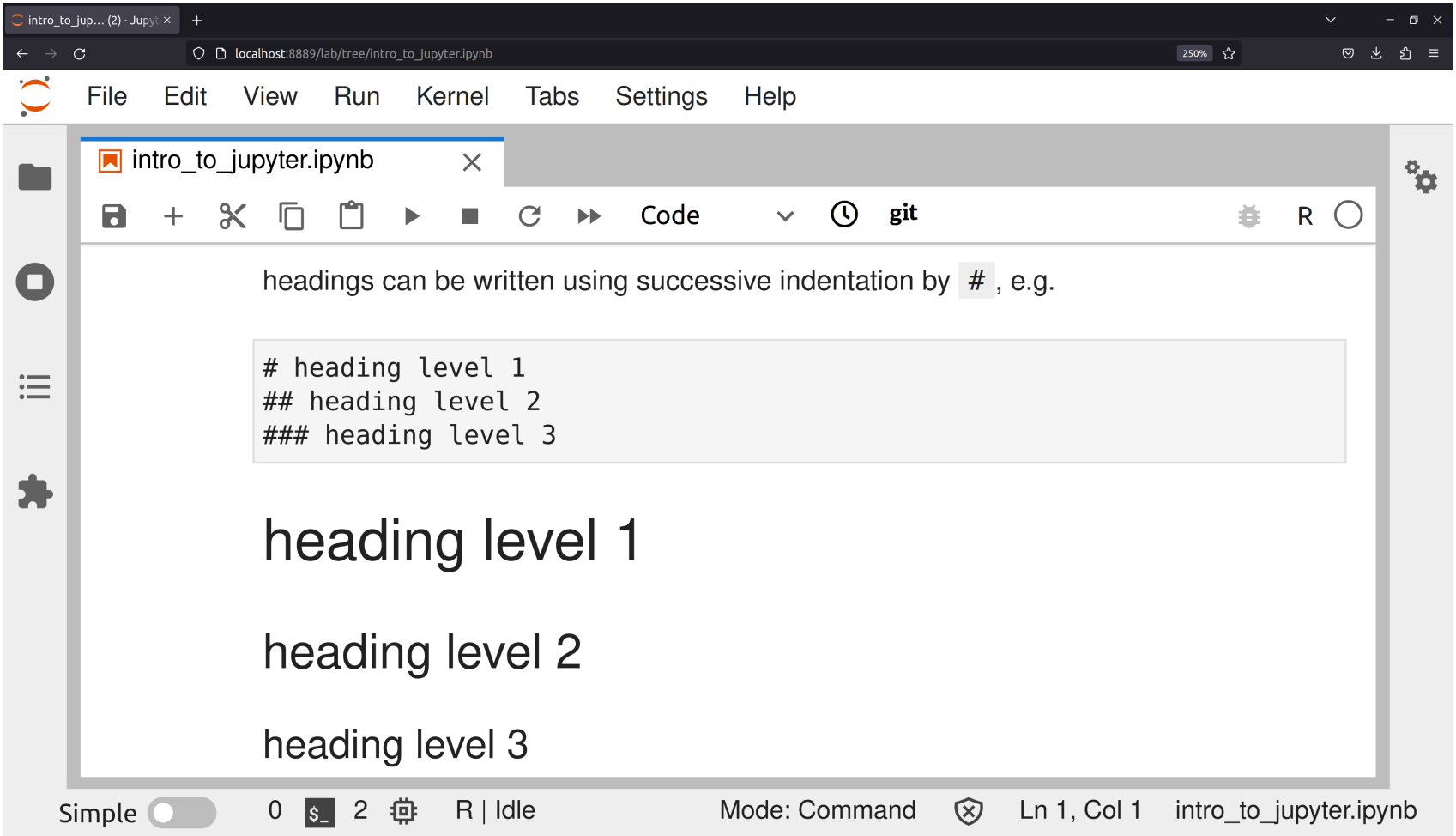
`jupyter` allows text to be written in `markdown` which is a light-weight markup language.

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

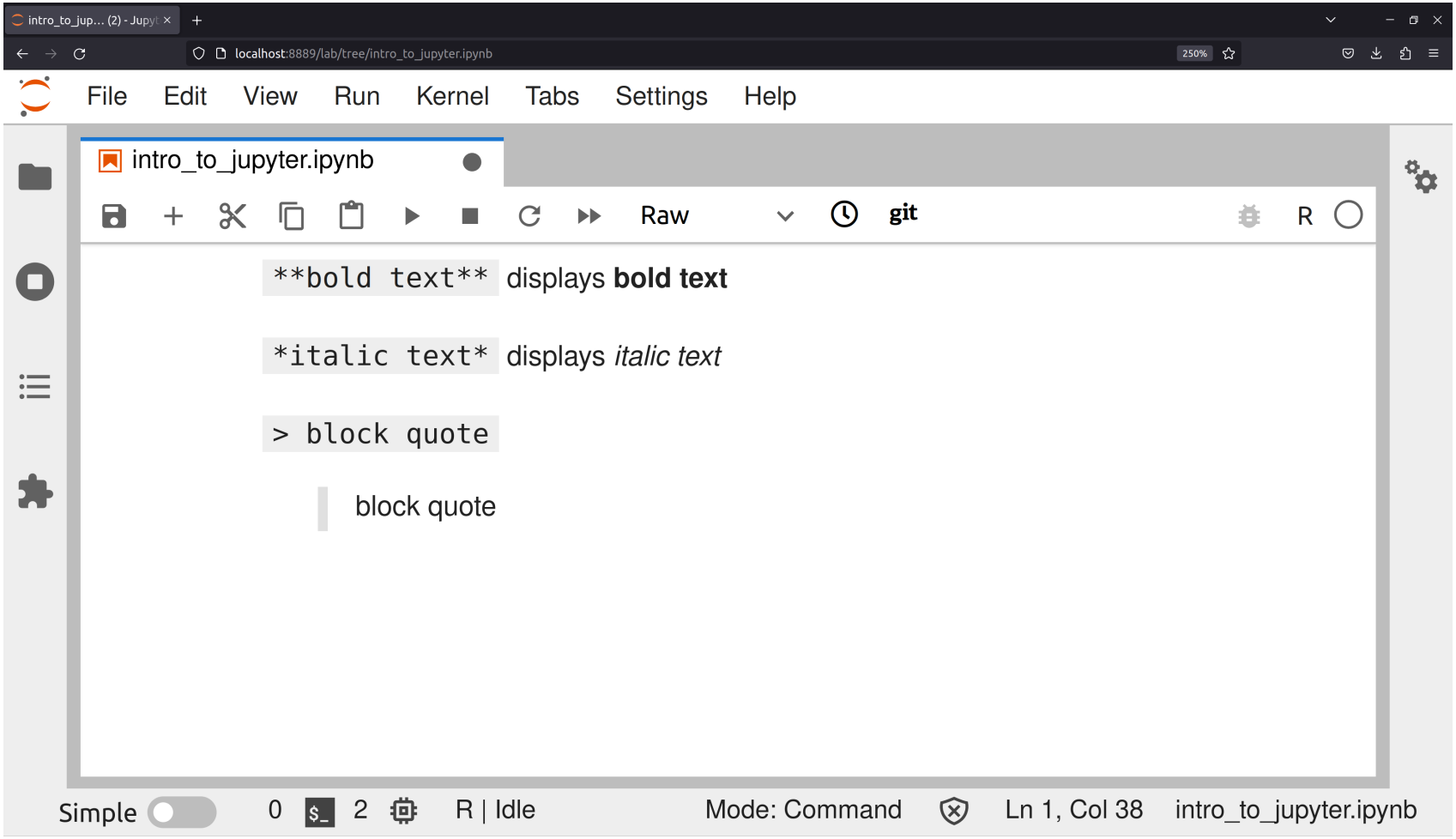
One can also use extended markdown languages like `myst` which enables features like references, figures, bibliographies, ...

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

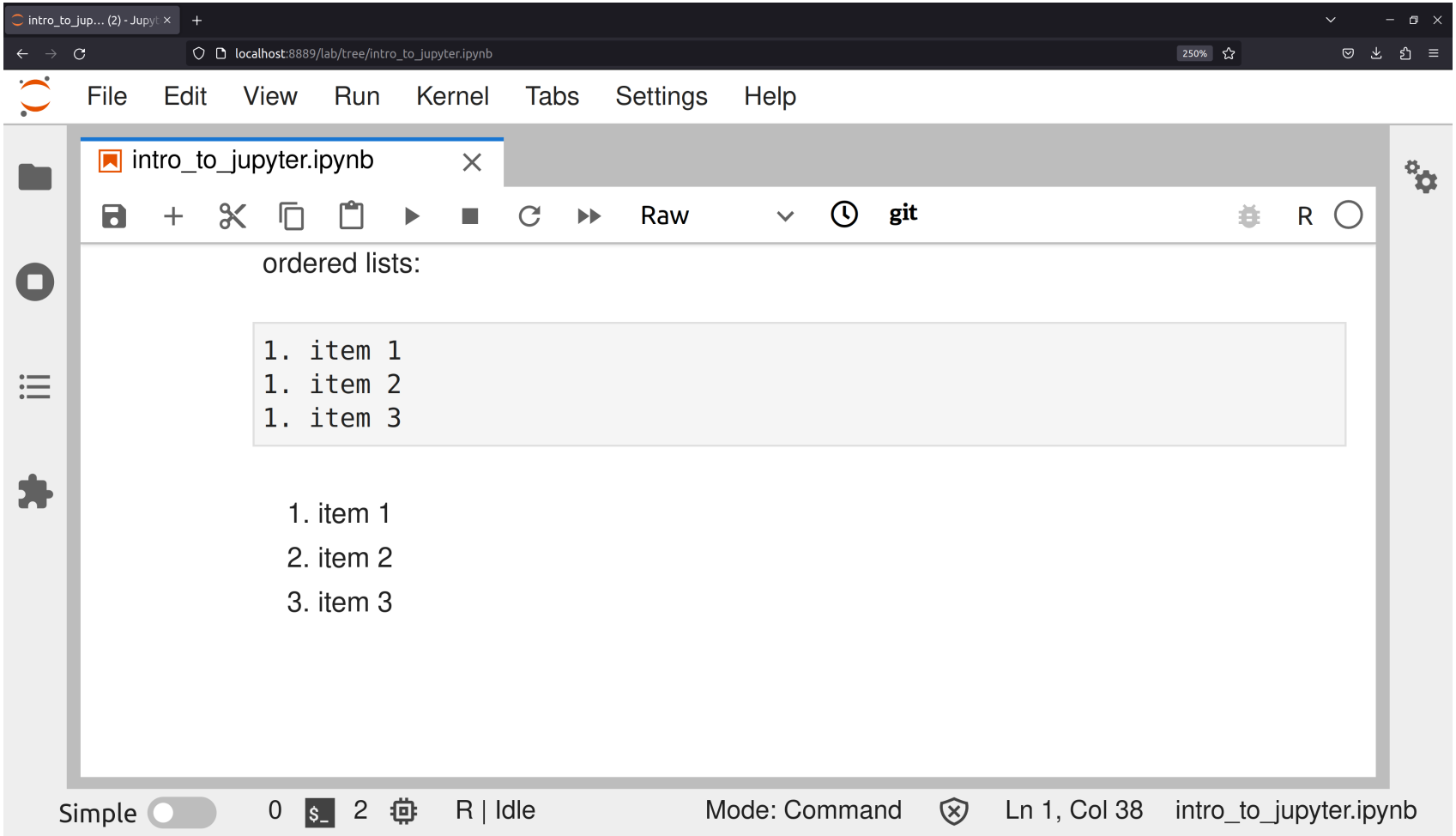
Basic Markdown



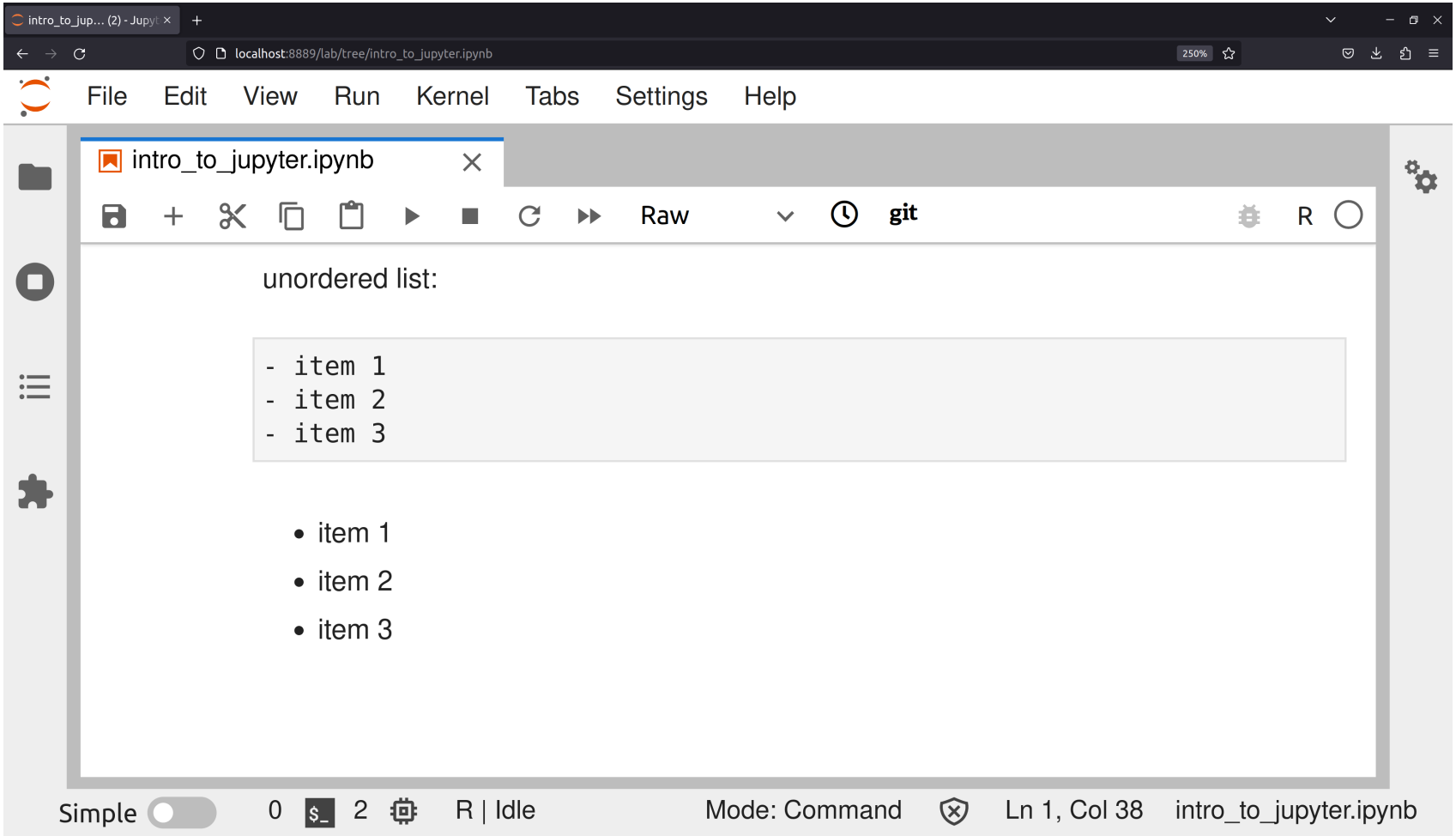
Basic Markdown



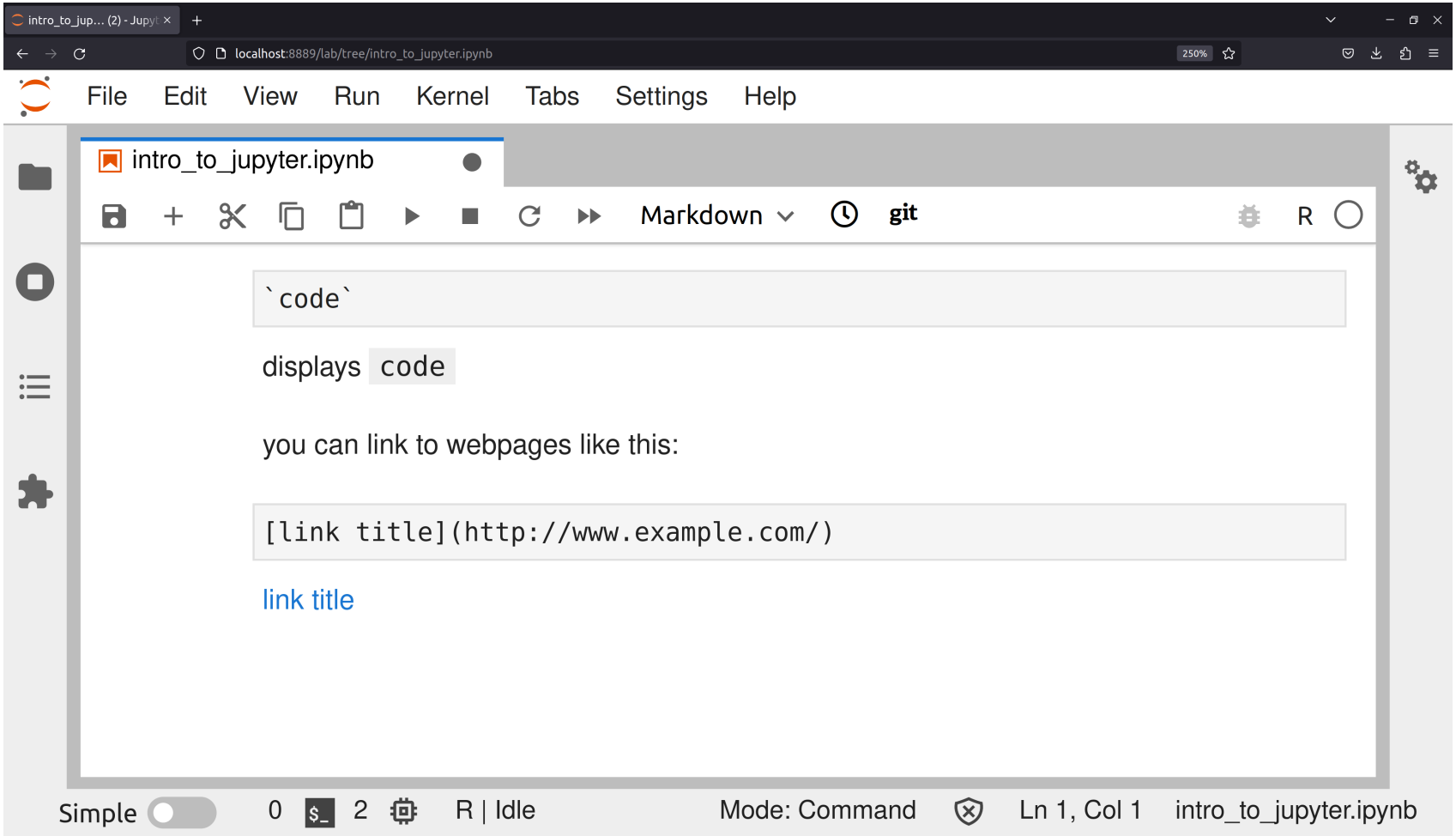
Basic Markdown



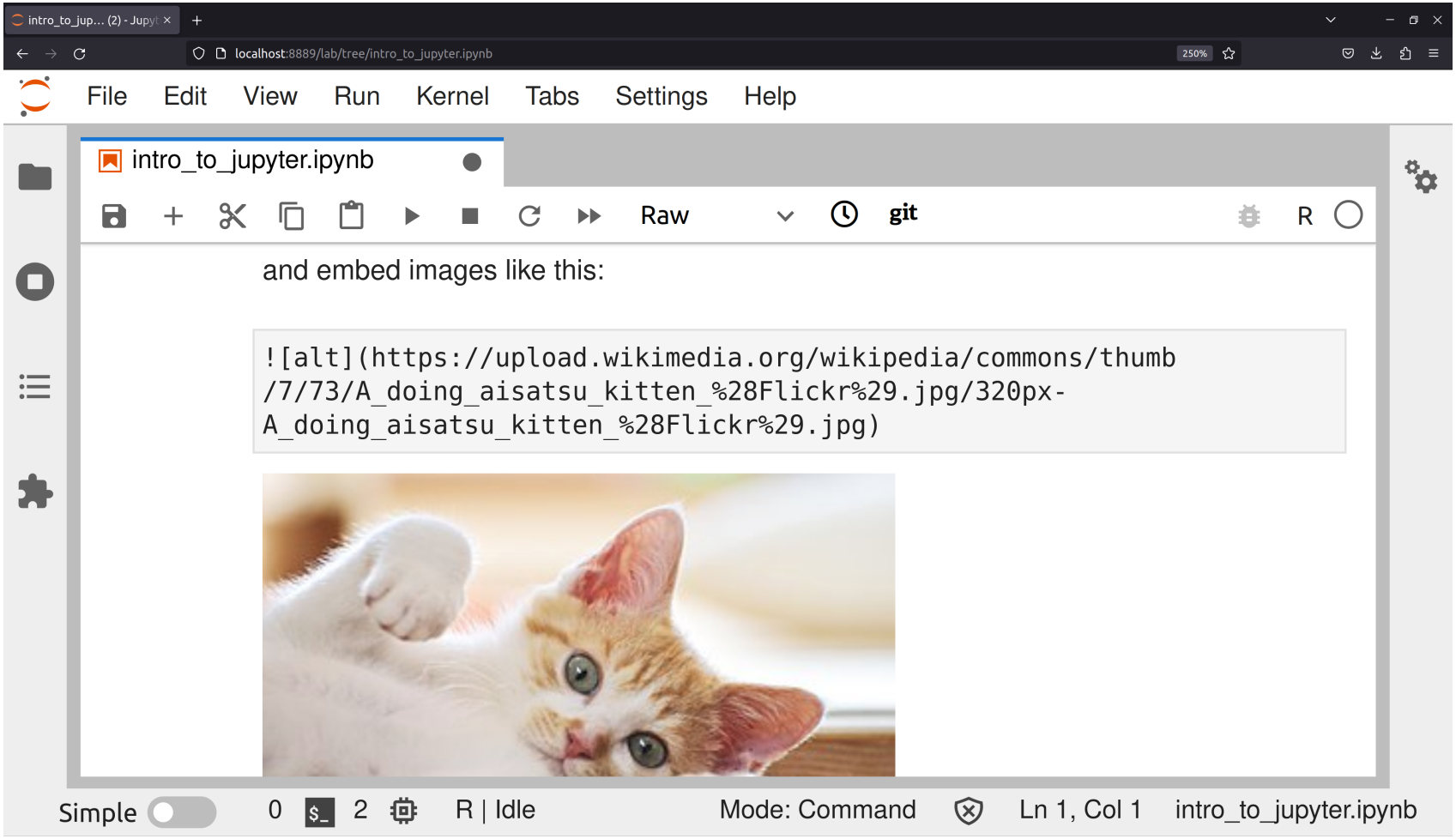
Basic Markdown



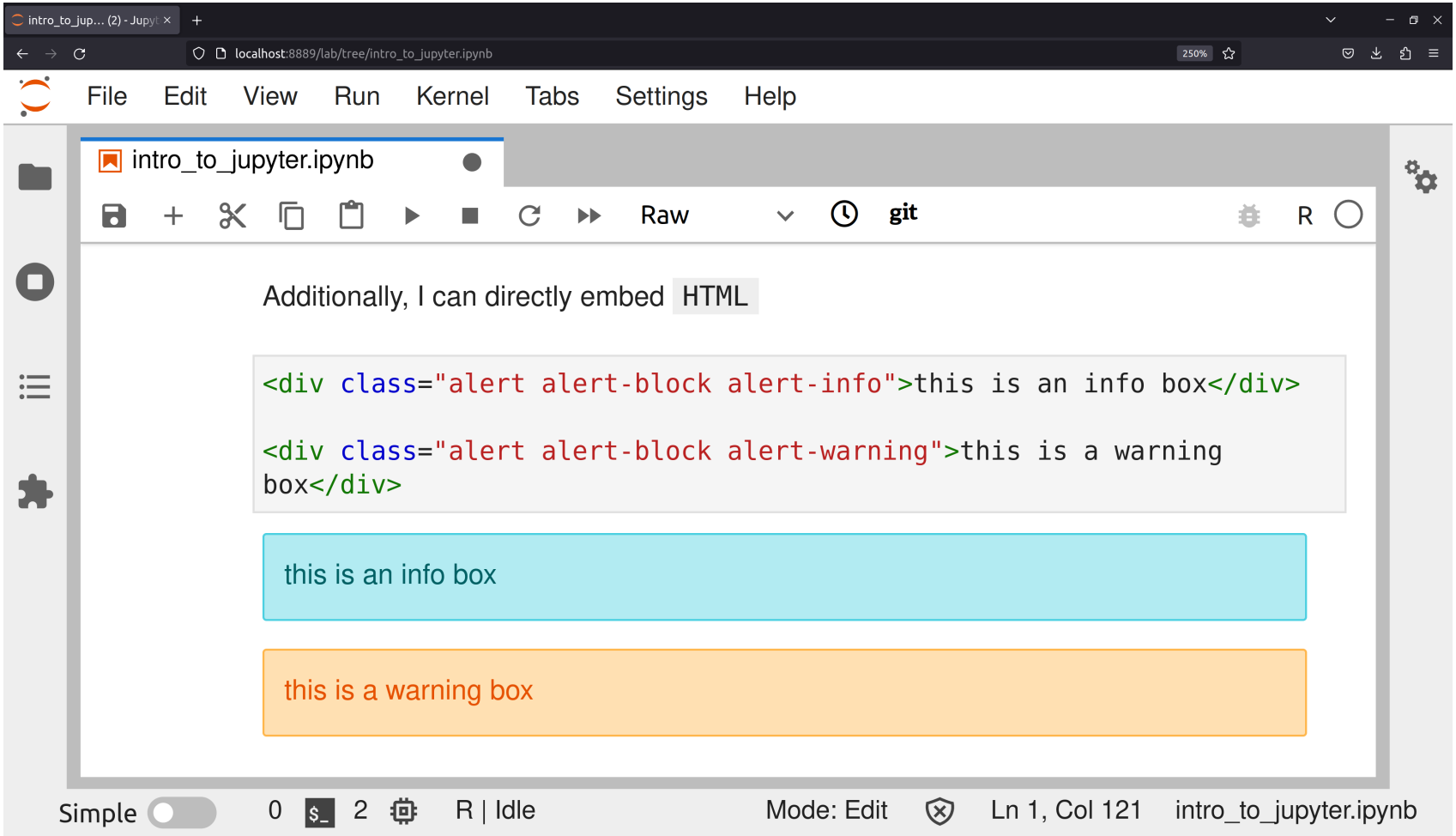
Basic Markdown



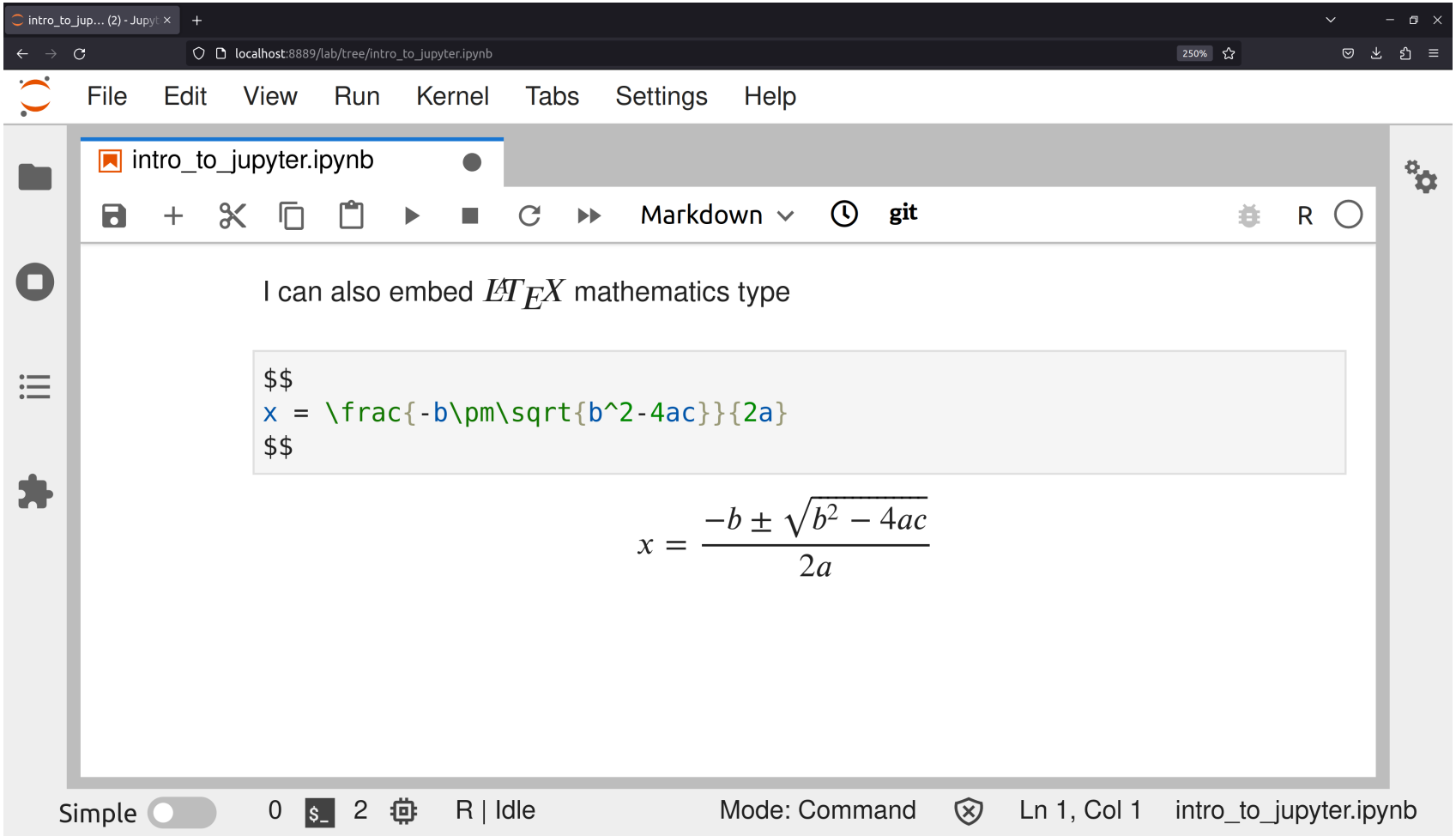
Basic Markdown



Basic Markdown

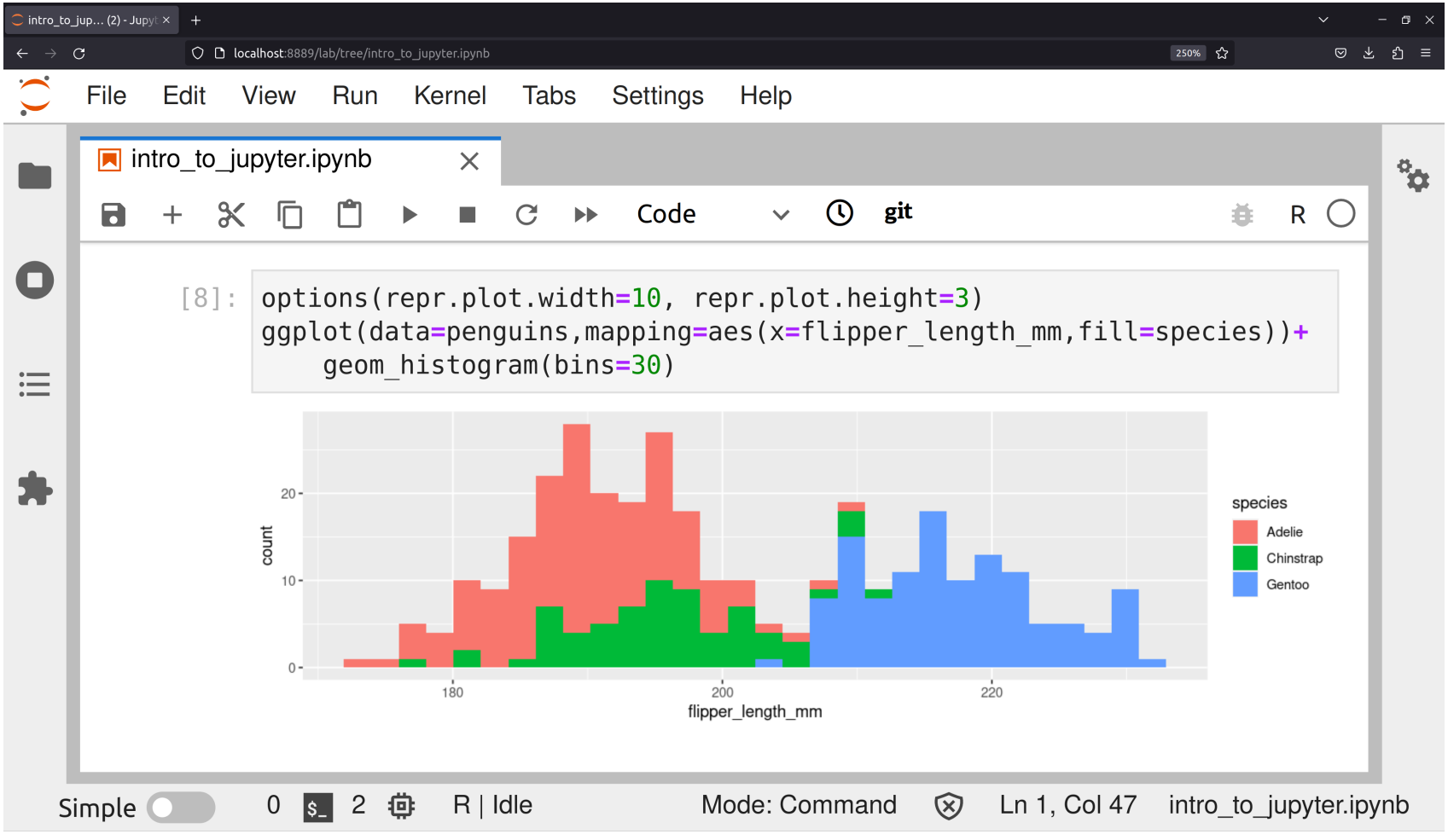


Basic Markdown



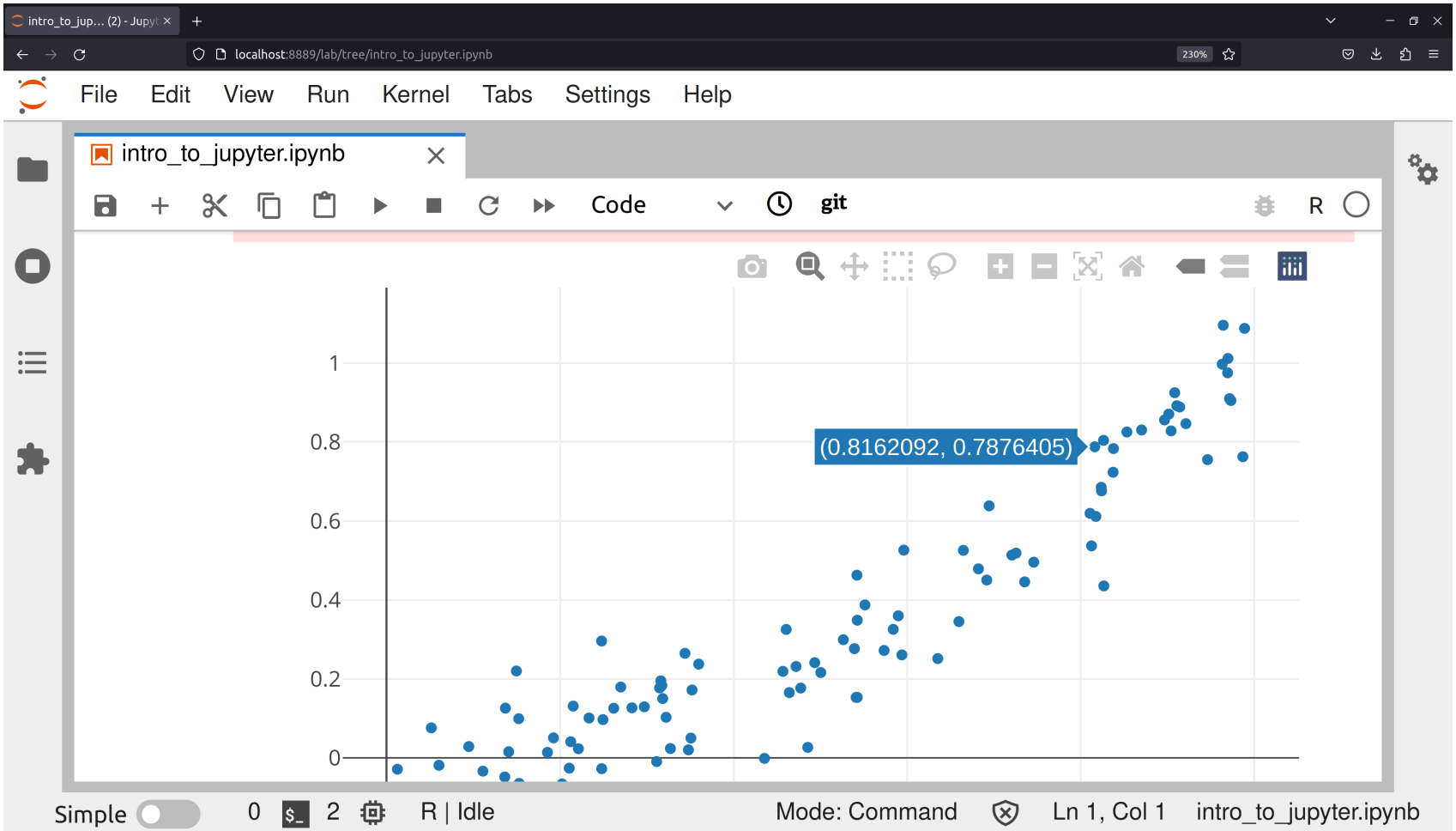
Code: Basic Output

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:

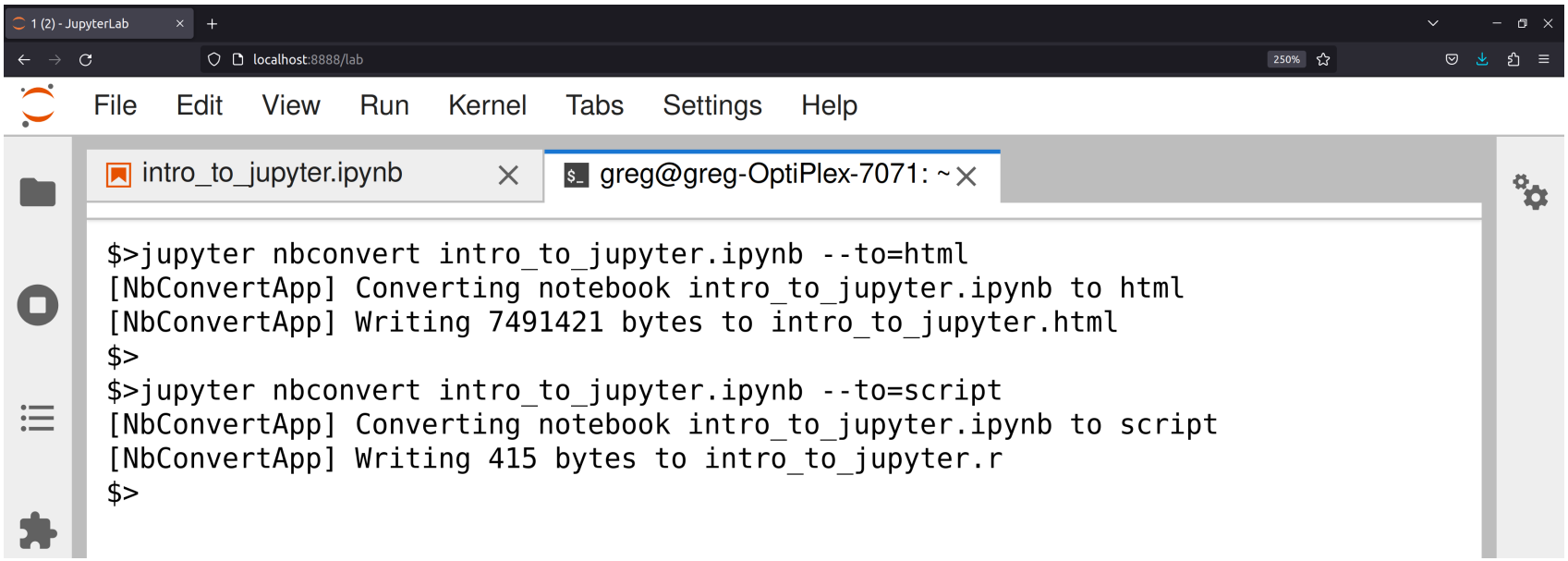
- `python`,
- `r`,
- `julia`,
- `stata`,
- `sql`,
- `octave`,
- `matlab`,
- `java`,
- `go`,
- `C`,
- `...`

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.

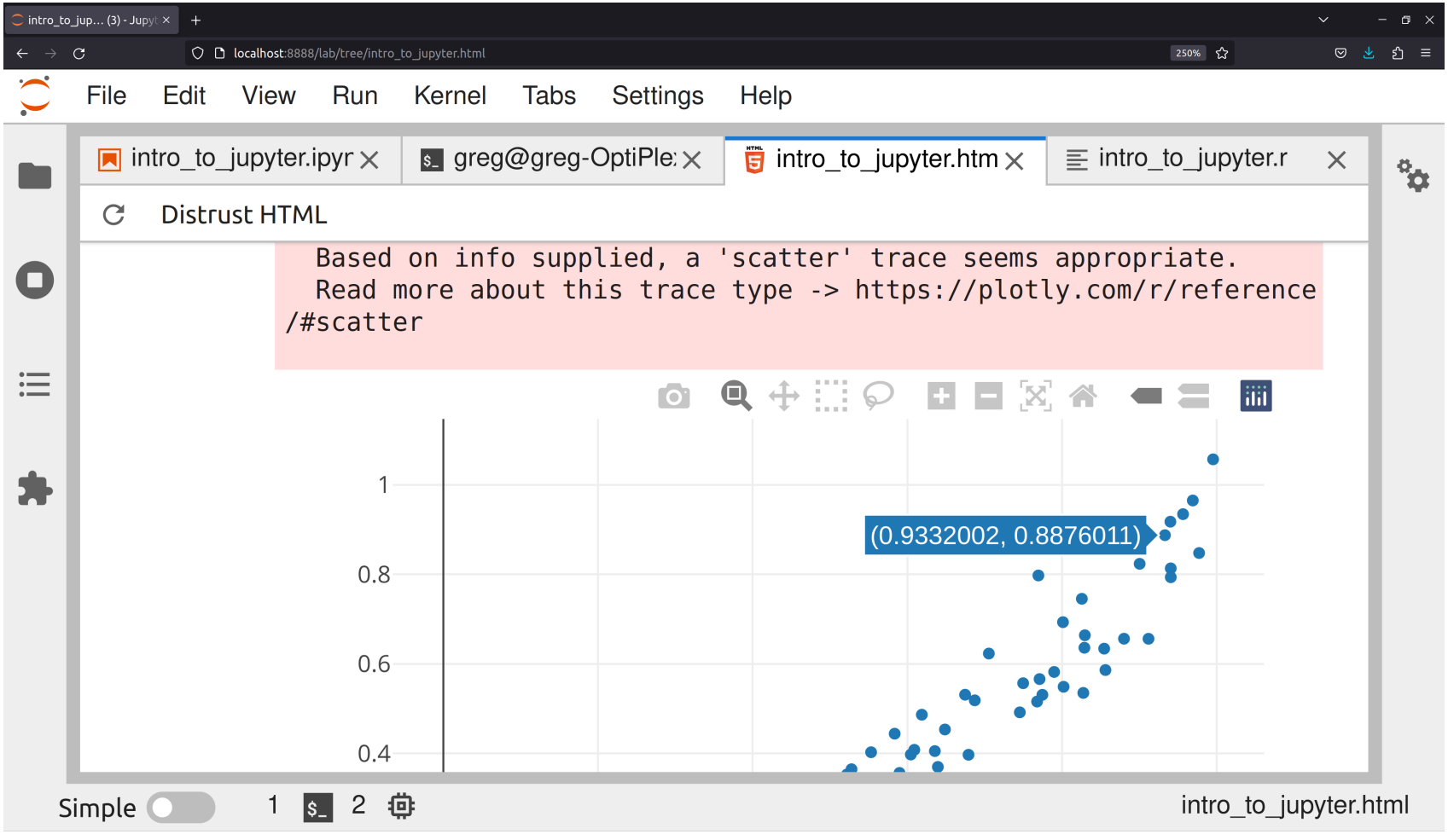
This can all be done using the command

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



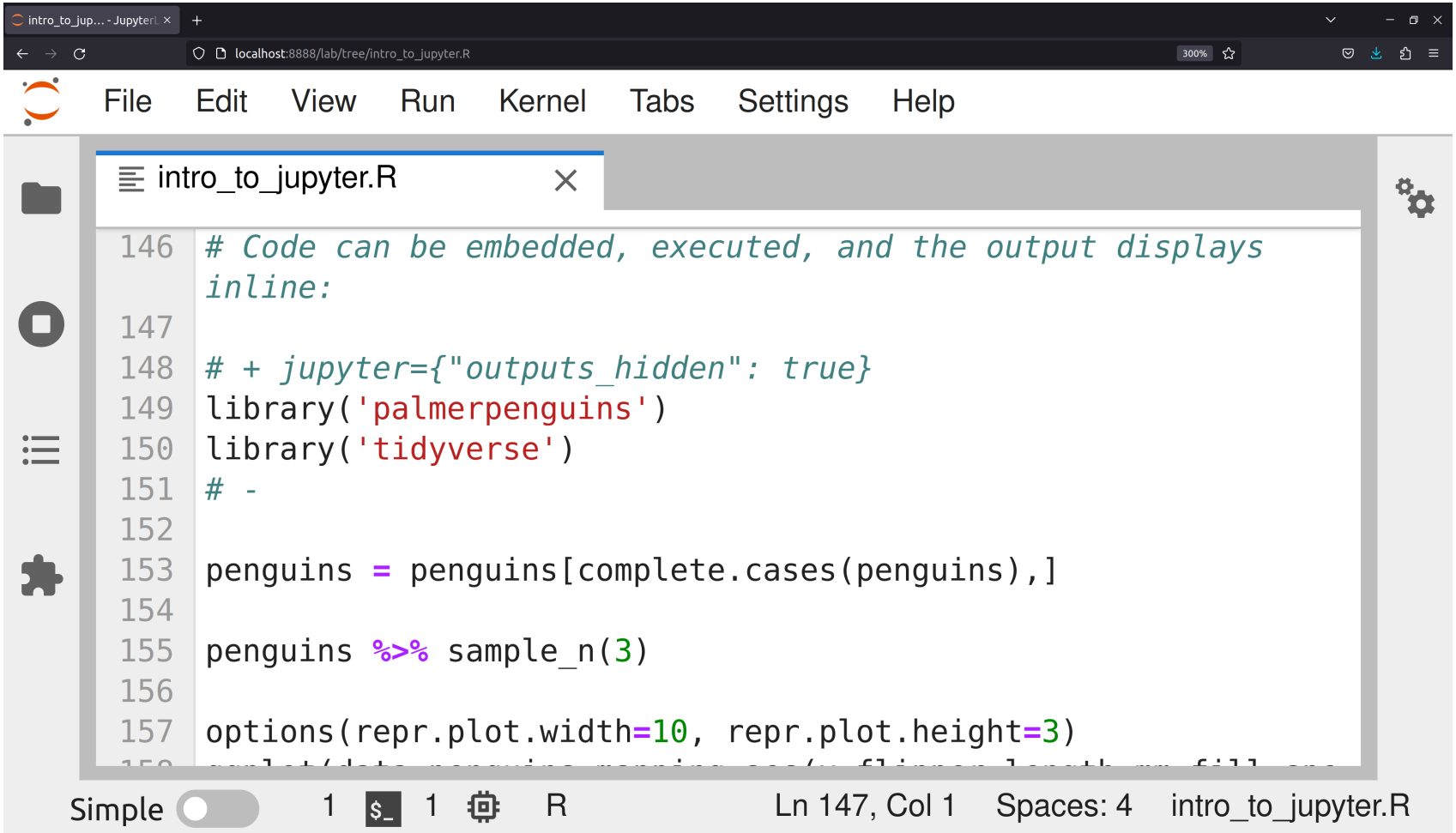
Exporting: HTML

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



Exporting: Script

Exporting as a script gives us basic executable script



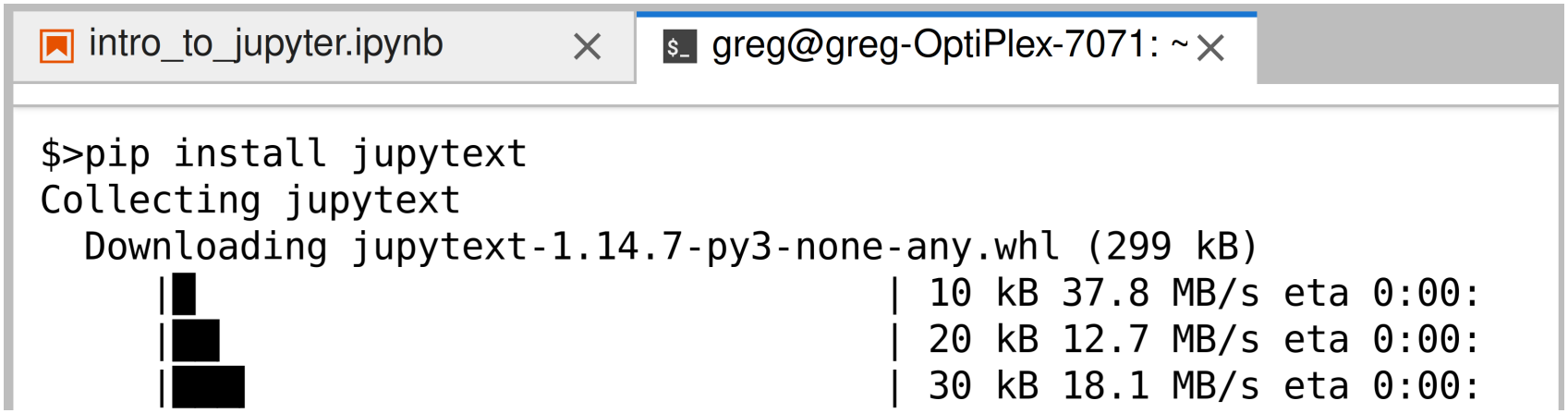
Notebook Interoperability

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
```

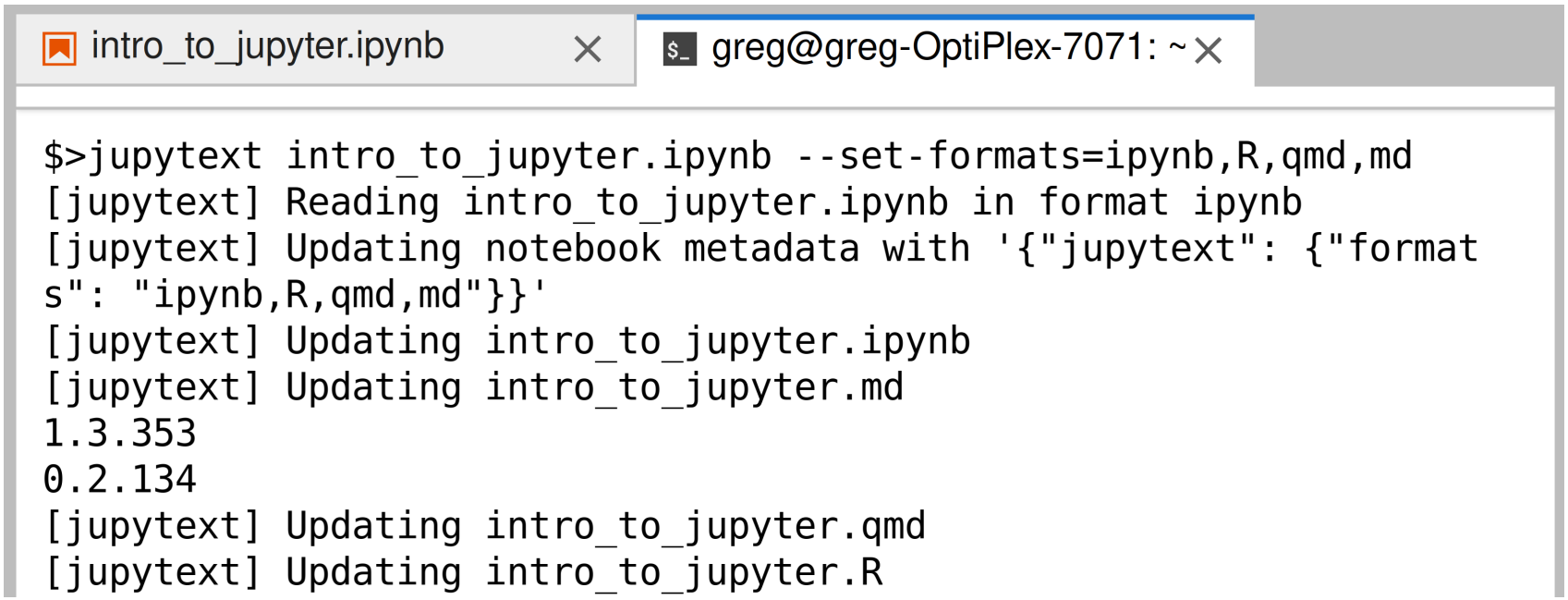


Notebook Interoperability

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

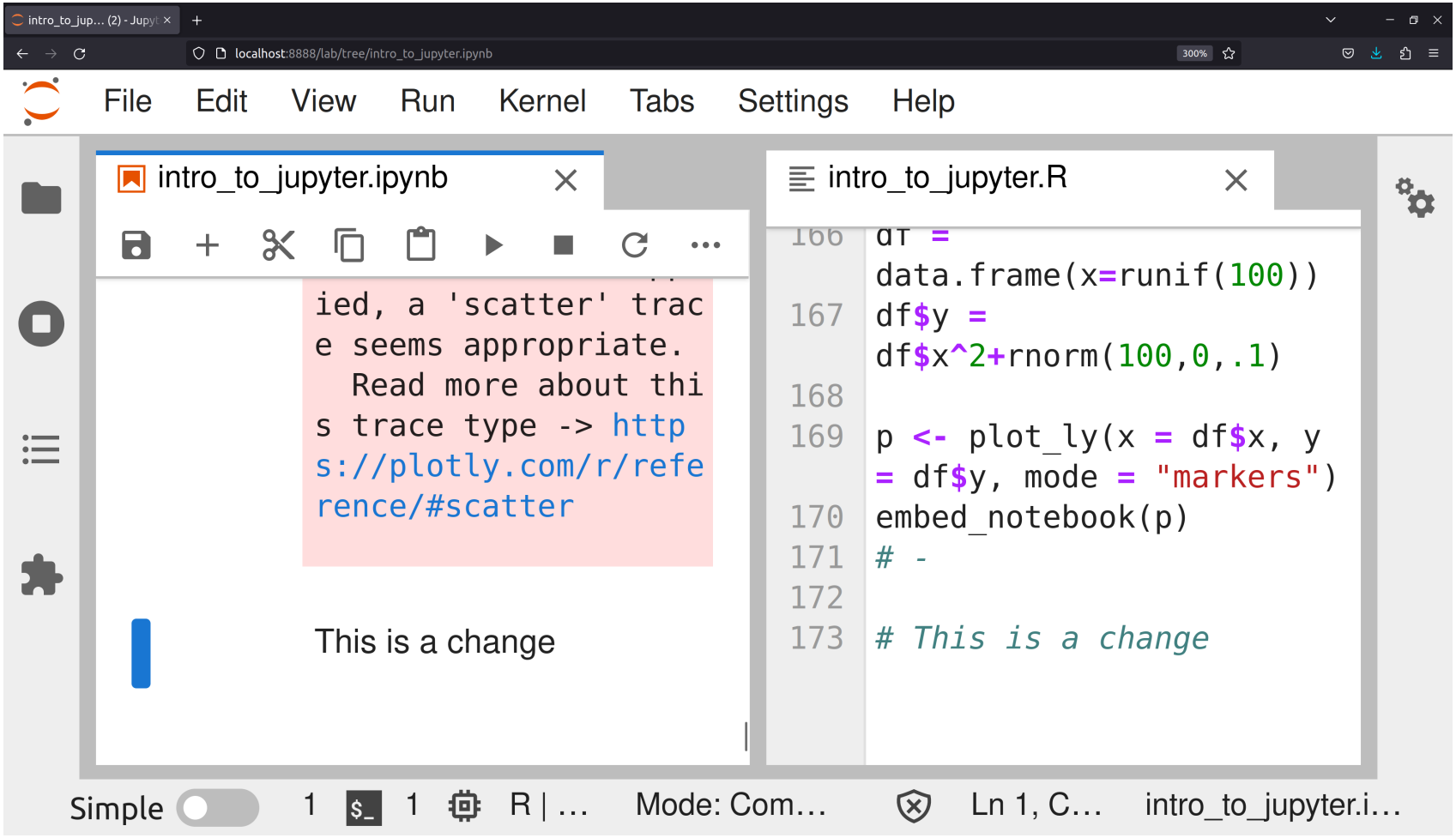
via the command

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



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

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.

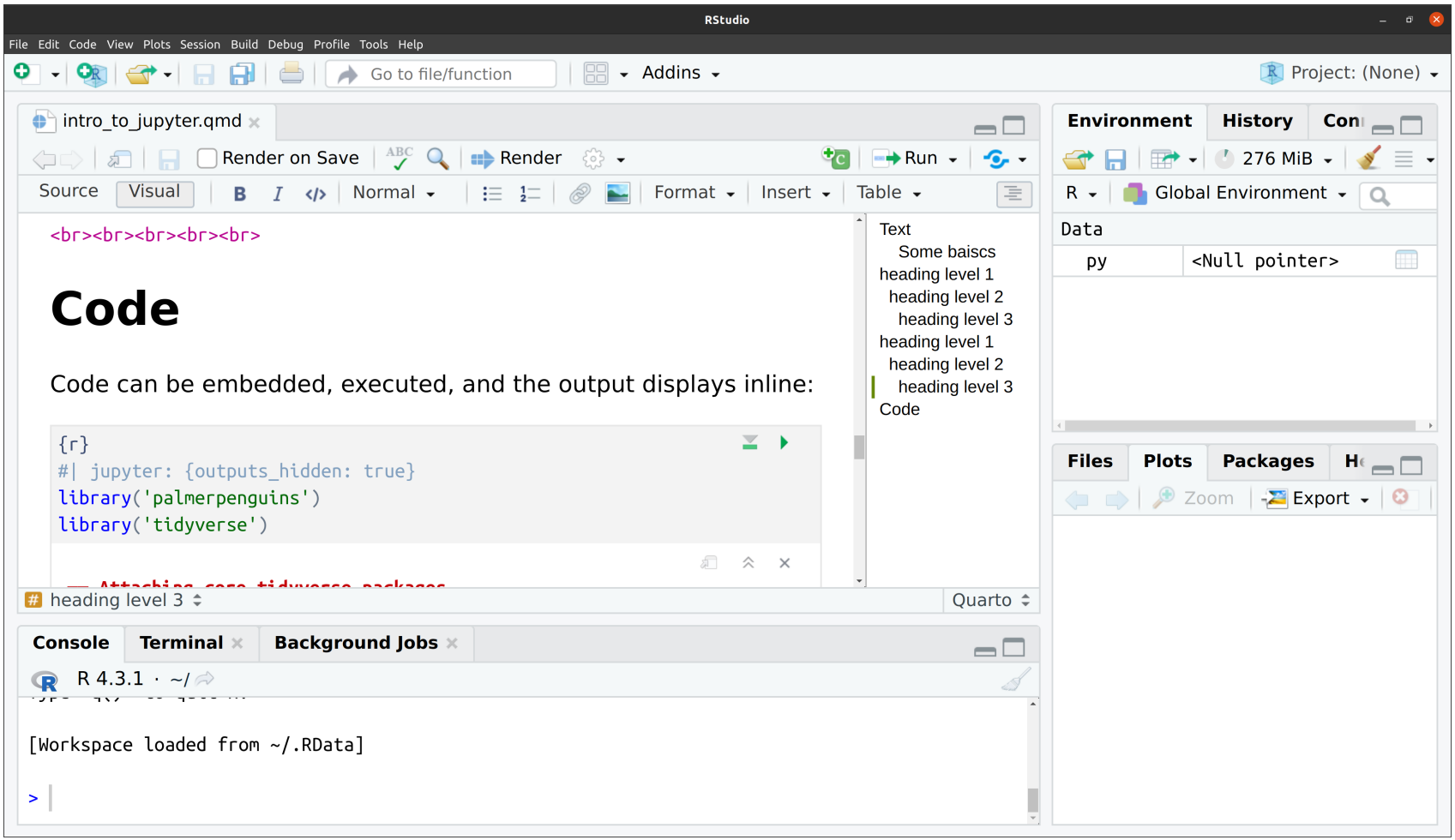
quarto: latest R studio notebook format

- `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 examples

`quarto` uses the markdown-like `.qmd` format (an evolution of `.rmd`, `.md`).

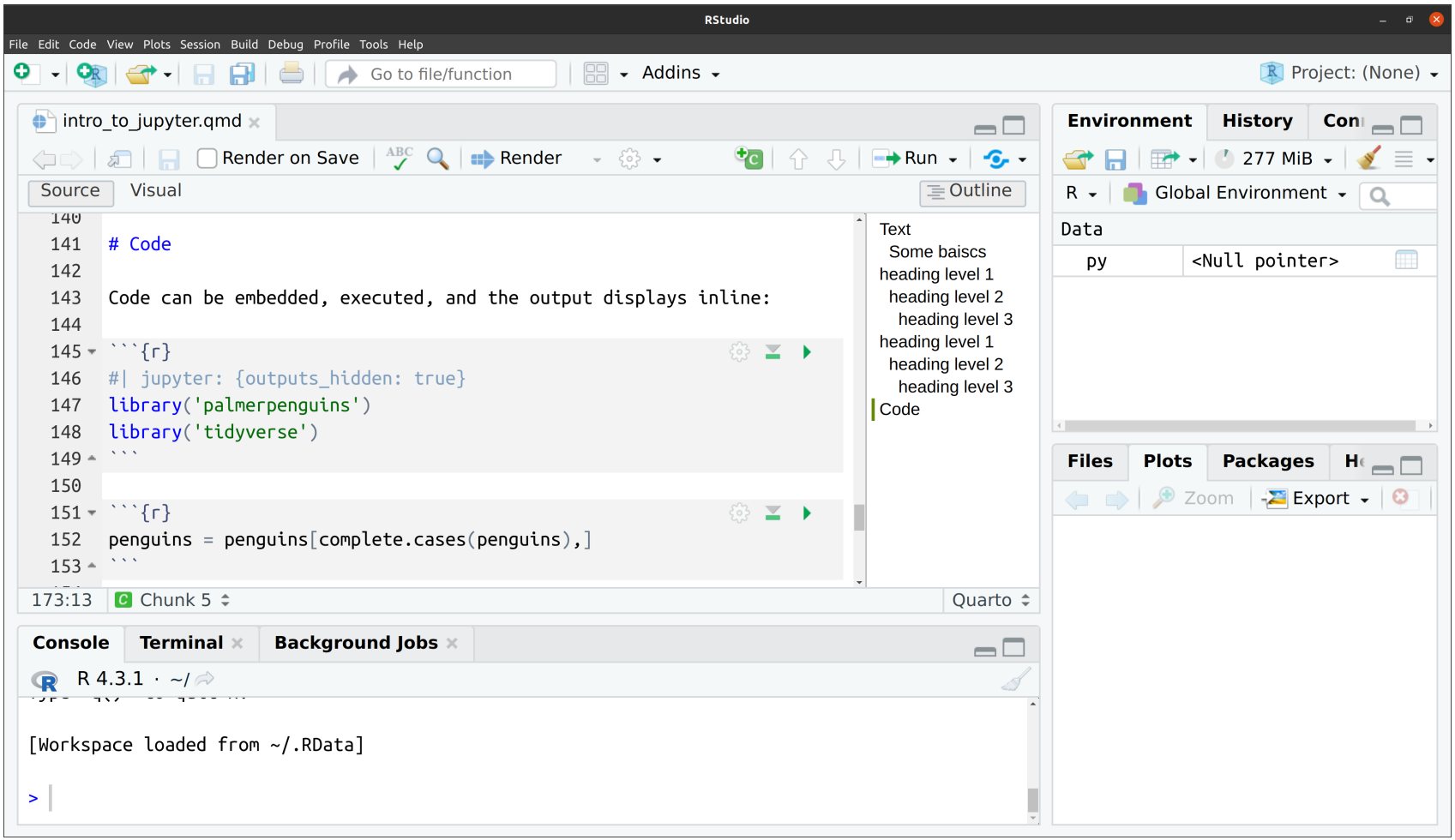
`Rstudio` visual editor:



quarto examples

`quarto` uses the markdown-like `.qmd` format (an evolution of `.rmd`, `.md`).

`Rstudio` visual markdown-like source:



quarto rendering

- `quarto` can `render` documents to a static display format (e.g. HTML, PDF, ...) either via command line or GUI (e.g. Rstudio calls `quarto` CLI)

```
#> quarto render intro_to_jupyter.qmd
Starting ir kernel...Done
Executing 'intro_to_jupyter.ipynb'
  Cell 1/5...Done
...
Output created: intro to jupyter.html
```

- `quarto` can also `render` jupyter `.ipynb` notebooks

```
#> quarto render intro_to_jupyter.ipynb --execute
...
Output created: intro_to_jupyter.html
```

- default: render does not **re-run** code when rendering `.ipynb`, so you'll need to `--execute` flag if you want to re-run code before rendering.

quarto preview and convert

- `quarto` can also `preview` documents
- this maintains a running background server and displays a real-time updated preview of the document in the web browser

```
#> quarto preview intro_to_jupyter.qmd
Executing 'intro_to_jupyter.ipynb'
  Cell 1/5...Done
...
Watching files for changes
Browse at http://localhost:6971/
```

`quarto` has a command to `convert` from `.qmd` to `.ipynb`

```
#>quarto convert intro_to_jupyter.ipynb --output quarto_nb.qmd
Converted to quarto_nb.qmd
```

and from `.ipynb` to `.qmd`

```
#>quarto convert quarto_nb.qmd --output jupyter_nb.ipynb
Converted to jupyter_nb.ipynb
```

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

Discussion

- Do you use notebooks regularly?
- Where do you find notebooks to be helpful?
- Where do you find notebook **not** to be helpful?

Example: Writing and converting notebooks.

1. Download and read in the `titanic` dataset:

```
url = 'https://tinyurl.com/35vnmbzm'
titanic = read.csv(url)
```

1. Explore the data, and make a plot or two.
2.
   - If using `jupyter`, use `jupytext` to mirror the analysis to a `.qmd` and `html`
   - If using `Rstudio`, use `quarto` to mirror the analysis to a `.ipynb` and `html`

We'll reconviene and discuss thoughts.

A notebook example:

We are going to generate some simulated data with the following model: First, we sample $N = 200$ samples

$$X_i \overset{iid}{\sim} U(-4\pi, 4\pi)$$

and then we calculate

$$Y_i = \sin(X_i) + \varepsilon_i$$

where $\varepsilon_i \overset{iid}{\sim} N(0, \sigma^2)$. Here $\sigma = .25$.

[1]:
```
N = 200
sig = .25
```

[2]:
```
x = sort(runif(N,-4*pi,4*pi))
eps = rnorm(N,0,sig)
y = sin(x)+eps
```

A notebook example:

Now we're going to fit using a kernel regression from the `locpol` package

[3]:
```
library('locpol')
```

a particularly important tuning parameter is the kernel bandwidth `bandw`, in our analysis we set it to 0.5

[4]:
```
bandw = 0.5
```

[5]:
```
smoothed = locPolSmootherC(x=x,y=y,xeval=x,
                                    deg=0,kernel=gaussK,
                                    bw=bandw)
preds = smoothed$beta0
```

Notebooks and Reproducibility

A notebook example:

[6]:
```
plot(x,y)
lines(x,preds,col='red',lwd=2)
```