

Getting Started with Quarto

Aidan Marnane

2023-04-28

Table of contents

0.1	What is Quarto?	2
0.1.1	Best features	2
0.2	Why use quarto?	2
1	writing Code	2
1.1	NumPy	2
1.2	Matplotlib	3
1.3	Plotly	5
1.3.1	Default plotting without options	5
1.3.2	Customisation With Options	6
1.4	From Notebooks	6
1.5	rendering code	8
2	Formatting	8
2.1	callout blocks	8
2.2	Cross referencing	9
2.3	Citations	10
2.4	Equations	11

In this document we will introduce a number of the key features Quarto markdown files support. We will be following the [basic tutorial](#) from Quarto as well as summarising some of the more advanced features introduced in their comprehensive guide. ¹

¹ Blog photo by [Setyaki Irham](#) from [Unsplash](#)

0.1 What is Quarto?

Quarto allows you to easily share and publish your code/analysis/research through any of `markdown/jupyter/knitr`. It is an extension of `pandoc` and offers support for `python/R/Julia`.

0.1.1 Best features

- render `jupyter` notebooks
- render markdown with code
- advanced visual customisation - figures, layout, citations
- create simple and easily customisable websites
- Great integration with [VSCode](#) and [Github Pages](#).

0.2 Why use quarto?

Sharing jupyter notebooks is tedious. Either you share a link to a prerendered notebook on github or you awkwardly convert to html/pdf with a variety of tools.

The problem? The conversion of a notebook is awkward. Often you have to choose between removing all or including all the code. And while the markdown support within `jupyter` is good again the customisation is limited.

Quarto solves this. It is a rendering tool that gives you a number of options with customise how code, figures and text are arranged when converting to `html` and `pdf`.

Even better it provides a new markdown format `.qmd` that allows you to write `python` code within a markdown document (similar to `R markdown`) or link to figures within a precomputed `jupyter` notebook.

1 writing Code

1.1 NumPy

Create code blocks in markdown using ````{python}`

```

```{python}
import numpy as np
a = np.arange(15).reshape(3, 5)
a
```

import numpy as np
a = np.arange(15).reshape(3, 5)
a

```

```

array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])

```

1.2 Matplotlib

Control how figures appear with comments `#| keyword: value.`²

```

```{python}
#| label: fig-limits-eg
#| fig-cap: "Errorbar limit selector"

import matplotlib.pyplot as plt

fig = plt.figure()
x = np.arange(10)
y = 2.5 * np.sin(x / 20 * np.pi)
yerr = np.linspace(0.05, 0.2, 10)

plt.errorbar(x, y + 3, yerr=yerr, label='both limits (default)')
plt.errorbar(x, y + 2, yerr=yerr, uplims=True, label='uplims=True')
plt.errorbar(x, y + 1, yerr=yerr, uplims=True, lolims=True,
 label='uplims=True, lolims=True')

upperlimits = [True, False] * 5
lowerlimits = [False, True] * 5
plt.errorbar(x, y, yerr=yerr, uplims=upperlimits, lolims=lowerlimits,

```

<sup>2</sup> The [getting started tutorial](#) has a nice description of the main options for figures and code output

```

 label='subsets of uplims and lolims')

plt.legend(loc='lower right')
plt.show(fig)
```

import matplotlib.pyplot as plt

fig = plt.figure()
x = np.arange(10)
y = 2.5 * np.sin(x / 20 * np.pi)
yerr = np.linspace(0.05, 0.2, 10)

plt.errorbar(x, y + 3, yerr=yerr, label='both limits (default)')
plt.errorbar(x, y + 2, yerr=yerr, uplims=True, label='uplims=True')
plt.errorbar(x, y + 1, yerr=yerr, uplims=True, lolims=True,
             label='uplims=True, lolims=True')

upperlimits = [True, False] * 5
lowerlimits = [False, True] * 5
plt.errorbar(x, y, yerr=yerr, uplims=upperlimits, lolims=lowerlimits,
             label='subsets of uplims and lolims')

plt.legend(loc='lower right')
plt.show(fig)

```

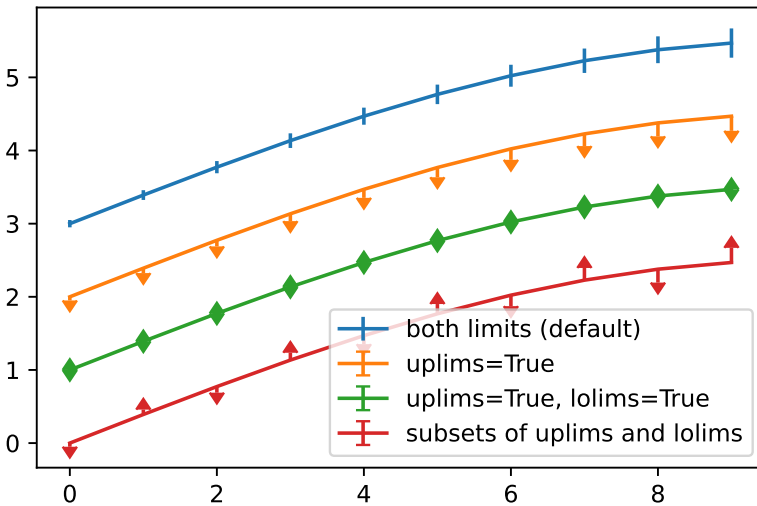


Figure 1: Errorbar limit selector

1.3 Plotly

1.3.1 Default plotting without options

```
import plotly.express as px
import plotly.io as pio
gapminder = px.data.gapminder()
gapminder2007 = gapminder.query("year == 2007")
fig = px.scatter(gapminder2007,
                 x="gdpPercap", y="lifeExp", color="continent",
                 size="pop", size_max=60,
                 hover_name="country")
fig.show()
```

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

1.3.2 Customisation With Options

```
```{python}
#| label: fig-gapminder
#| fig-cap: "Life Expectancy and GDP"
#| fig-subcap:
#| - "Gapminder: 1957"
#| - "Gapminder: 2007"
#| layout-ncol: 2
#| column: page

...

import plotly.express as px
import plotly.io as pio
gapminder = px.data.gapminder()
gapminder2007 = gapminder.query("year == 2007")
fig = px.scatter(gapminder2007,
 x="gdpPercap", y="lifeExp", color="continent",
 size="pop", size_max=60,
 hover_name="country")

fig.show()

gapminder1957 = gapminder.query("year == 1957")
fig = px.scatter(gapminder1957,
 x="gdpPercap", y="lifeExp", color="continent",
 size="pop", size_max=60,
 hover_name="country")

fig.show()
```

Unable to display output for mime type(s) text/html

(a) Gapminder: 1957

(b) Gapminder: 2007

Figure 2: Life Expectancy and GDP

## 1.4 From Notebooks

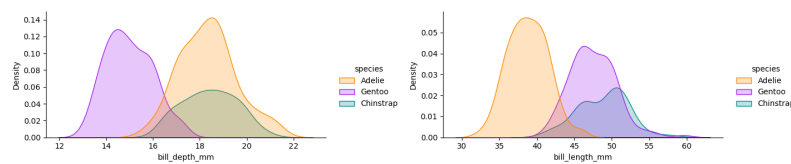
It is simple to embed plots from precomputed notebooks and include a link.

```
{{< embed penguins.ipynb#fig-bill-scatter >}}
```

```
alt.Chart(...)
```

Figure 3: A scatterplot of bill dimensions for penguins, made with Altair.

```
{{< embed penguins.ipynb#fig-bill-marginal >}}
```



- (a) Gentoo penguins tend to have thinner bills, (b) and Adelie penguins tend to have shorter bills.

Figure 4: Marginal distributions of bill dimensions

You can also include the code by specifying `echo=true` in the call.

```
{{< embed penguins.ipynb#species-counts echo=true >}}
```

```
penguins.groupby("species").size().reset_index(name = "count")
```

	species	count
0	Adelie	152
1	Chinstrap	68
2	Gentoo	124

You can control how the link to the notebook appears in the title metadata <sup>3</sup>

<sup>3</sup> See the [notebook embedding tutorial](#) for more info

notebook-view:

```
- notebook: penguins.ipynb
```

```
title: "Plots and Computations"
```

## 1.5 rendering code

There are a number of ways to control the rendering of code. For example, if you have a notebook with some code that takes a long time to run you won't want to recompute it everytime you change formatting. There are many options to control this behaviour.<sup>4</sup>

<sup>4</sup> see the [python tutorial](#) for a more in depth explanation

```
execute:
 freeze: true
```

## 2 Formatting

### 2.1 callout blocks

Quarto has nice control for adding note blocks. There are five formats<sup>5</sup> - note - warning - important - tip - caution

<sup>5</sup> See [their tutorial](#) for a more in depth explanation To add a callout note use `{.callout-note}` or `{.callout-important}` etc

```
::: {.callout-note}
Note that there are five types of callouts, including:
'note', 'warning', 'important', 'tip', and 'caution'.
:::
```

```
::: {.callout-tip}
Tip with Title
```

This is an example of a callout with a title. See [their tutorial](https://quarto.org/docs/autobibliography.html)  
:::

```
::: {.callout-caution collapse="true"}
Expand To Learn About Collapse
```

This is an example of a 'folded' caution callout that can be expanded by the user. You can use  
:::



### Note

Note that there are five types of callouts, including: **note**, **warning**, **important**, **tip**, and **caution**.

### Tip with Title

This is an example of a callout with a title.

### Expand To Learn About Collapse

This is an example of a ‘folded’ caution callout that can be expanded by the user. You can use `collapse="true"` to collapse it by default or `collapse="false"` to make a collapsible callout that is expanded by default.

## 2.2 Cross referencing

```
import matplotlib.pyplot as plt
plt.plot([1,23,2,4])
plt.show()
```

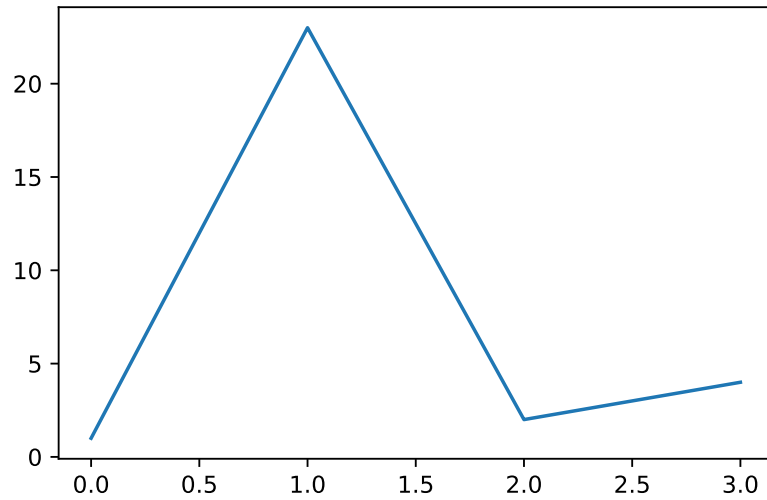


Figure 5: A line plot

This is a cross reference to our figure Figure 5 with `@fig-line-plot`. We can also reference the matplotlib figure above Figure 1 `@fig-limits`.

#### **i** Note

You must start your label with `fig-` e.g. `fig-line-plot` for the cross reference to work.

## 2.3 Citations

You can include citations by including a bibliography.

For example, Antoine et al produced some sick work (Lain et al. 2022).

Add a path to your bibliography in the title metadata

```

title: "My Document"
bibliography: references.bib

```

You can customise citation style and more.<sup>6</sup>

<sup>6</sup> Footnotes are also possible. For more guidance see the [quarto tutorial](#)

```

title: "My Document"
bibliography: references.bib
csl: nature.csl

```

#### Note

You can download csl files from the [CSL repo](#). nature.csl was taken from [here](#) (there are so many I had to search to get it.)

Citation can also be placed in the margin by adding

```
citation-location: margin
```

## 2.4 Equations

We can also write equations.

They can be placed inline  $\frac{d}{dx} \left( \int_a^x f(u) du \right) = f(x)$ .  
`\frac{d}{dx}\left( \int_{a}^{x} f(u)\,du\right)=f(x).`

They can be placed centrally.

`$$\frac{d}{dx}\left( \int_{a}^{x} f(u)\,du\right)=f(x).$$`

$$\frac{d}{dx} \left( \int_a^x f(u) du \right) = f(x).$$

But they can also be placed in the margin

::: {.column-margin}

We know from *the first fundamental theorem of calculus* that for  $x$  in  $[a, b]$ :

$$\frac{d}{dx} \left( \int_a^x f(u) du \right) = f(x).$$

:::

Lain, Antoine, Wonjin Yoon, Hyunjae Kim, Jaewoo Kang, and Ian Simpson. 2022. “KU\_ED at SocialDisNER: Extracting Disease Mentions in Tweets Written in Spanish.” In *Proceedings of the Seventh Workshop on Social Media Mining for Health Applications, Workshop & Shared Task*, 78–80. Gyeongju, Republic of Korea: Association for Computational Linguistics. <https://aclanthology.org/2022.smm4h-1.23>.

We know from *the first fundamental theorem of calculus* that for  $x$  in  $[a, b]$ :

$$\frac{d}{dx} \left( \int_a^x f(u) du \right) = f(x).$$