

Quarto が完成すると Jupyter でも R Markdown のように簡単に スライドや文書を作れるかもしれない

ill-identified (片桐 智志)

2021/10/04

目次

概要	2
はじめに	2
Quarto の特徴	3
従来通り Markdown + チャンクで書く	3
すでに RStudio に対応している	3
YAML メタデータの構文が使いやすくなった	5
(u)pLaTeX の使用もしやすくなりそう	6
新しい記法のチャンクオプション	7
相互参照を使いやすくした上で最初からサポート	7
Jupyter や VS Code への対応	8
多様な Jupyter カーネルへの対応	9
現時点での制約	10
Quarto の導入方法	11
RStudio 上での使用法	11
RStudio 上で knitr/jupyter を使用する場合	12

Jupyter 上で Python3 カーネルを使用する場合	13
VS Code 上で Jupyter を操作する場合	13
Jupyter 上で Julia カーネルを使用する場合	14
まとめ	14
付録: 各種設定での出力例	15
補足: R Markdown と Quarto の LaTeX 設定の比較	16

概要

- [Quarto](#) という R Markdown のような動的ドキュメント生成プログラムが開発中
- まだ開発版だが野心的な機能をめざしているらしい
- 単なる R Markdown の再設計ではなく, **Jupyter Notebook の変換にも対応**している
- R や Python だけでなく Jupyter カーネルで使える任意の言語でも R Markdown のように扱えるかもしれない
- 使い方の参考になるようにこの投稿も **Quarto** を利用して書いている
- 付録としてプレゼンテーション資料への変換, Jupyter Notebook の変換例も用意した
<https://github.com/Gedevan-Aleksizde/quarto-demo-ja>

注意

Quarto は最近公開されたばかりで開発中なのでこの記述もすぐ時代遅れになる可能性があることに注意してほしい.

(このように注記のためのブロックも追加の事前準備なしに簡単に書ける^{a)})

^a <https://quarto.org/docs/authoring/callouts.html>

はじめに

最近公開された [Quarto](#) というソフト, 最初はこちらでわかりにくかった R Markdown の YAML メタデータのフォーマットをすっきりさせただけと思っていたが, それどころではないということがわかってきたので紹介してみる. ただし, 現時点ではまだ開発が始まったばかりであり, 頻繁に更新され, 細かなバグも多くやや不安定である.

そんな不安定な状況でなんで公開したか。自分もこれに数ヶ月前に関心を持ち、興味深いとは思いつつも、もう少し安定してきてから紹介しようかと思っていた。しかし昨日リリースされた、**knitr** v1.35 の [NEWS](#) を見ると、あきらかに Quarto を意識したように見える機能追加が行われていた。RStudio 的には結構本気で^{*1}取り組んでいるプロジェクトなのかもしれない。公式ドキュメントも久々に見返すと以前よりさらに充実していたので、そろそろ紹介してみるタイミングかと思い公開してみた。

Quarto の動作確認は以下のような環境で行った。^{*2} 開発版ではあるが、ドキュメントは既にそれなりに充実しており、R Markdown 使用者なら想像で補って書けるところも多い。

- Ubuntu 20.04^{*3}
- R 4.1.1
- RStudio 2021.09.0 Build 351
- Pandoc: 上記最新版 RStudio に同梱されているもの (v2.14.0.3) を使用
- Quarto [v0.2 Build 178](#)
- **knitr** 1.36^{*4}
- Jupyter 関連:
 - Python 3.9.6 (on Pyenv)
 - ipykernel: 6.4.1
 - jupyter-lab: 3.1.14
 - Julia: 1.6.3

Quarto の特徴

従来通り Markdown + チャンクで書く

YAML メタデータの項目名やチャンクの設定方法といった一部に変更があるが、Markdown + チャンクによるプログラムの埋め込み、そして YAML メタデータによる文書のメタデータや出力媒体の設定という基本は変わらない。よって R Markdown ユーザにとっては移行が容易だろう。

すでに RStudio に対応している

Quarto がインストールされていなければ表示されないのが気づいてない人のほうが多いだろうが、実はすでに RStudio のエディタも Quarto に対応しており、R Markdown のようにファイルやプロジェクトを新規作成し

^{*1} [github](#) リポジトリを見ると一番コミットしてるのはなぜか RStudio CEO の J.J. Allaire.

^{*2} Quarto について実際に調べたのは数ヶ月前で、この記事公開に合わせて再度簡単に確認した程度

^{*3} Windows や Mac ではまったく動作確認していないが、WSL とか AWS とか RStudio Workbench とか使えば Ubuntu 環境を得るのは簡単なので省略

^{*4} 1.36 は 1.35 の翌日にリリースされた…

たりできる (図 1).

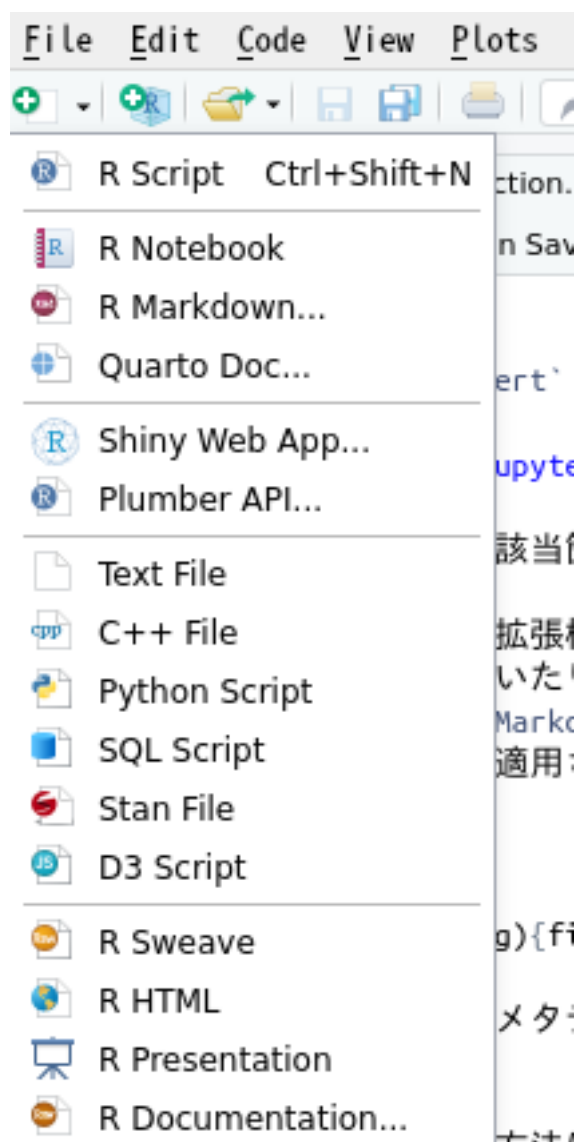


図 1: RStudio の新規作成メニュー

また, RStudio v1.4 から R Markdown のビジュアルエディタ機能が追加されたことは既に知っている人も多いかもしれないが, このビジュアルエディタは **Quarto** にも対応している. (ただし私は Visual Editor をあまりつかっていないので既知の不具合にも詳しくない)

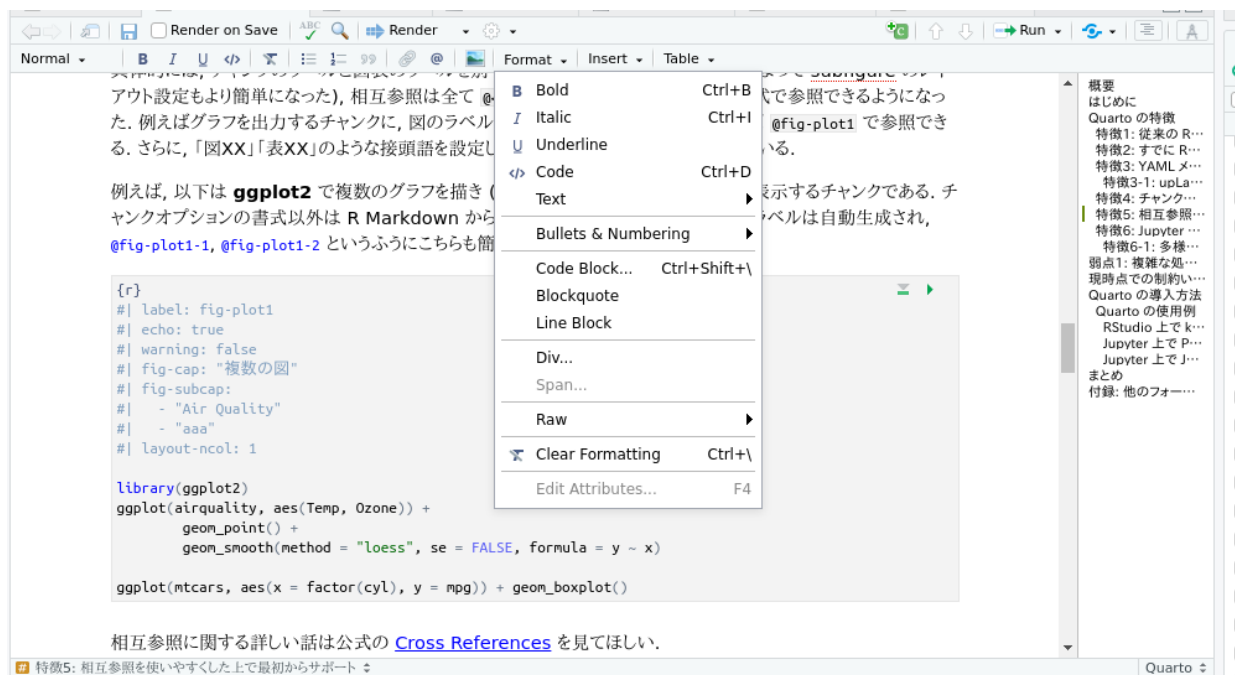


図 2: ビジュアルエディタ上での Quarto

YAML メタデータの構文が使いやすくなった

これは R Markdown 利用者でないと良さが分かりづらいかもしれない。それ以外のユーザにとっては、R Markdown 開発開始当時にはなかった、最近の Pandoc で追加された機能 (v2.14 あたりに準拠?) をベースに、YAML メタデータで文書全般の設定を指定できる、という程度の認識で良いと思う。

以下、R Markdown ユーザを想定とした解説。

以前『[R Markdown の YAML ヘッダでハマったおまえのための記事](#)』と題して書いたように、R Markdown の YAML メタデータは Pandoc が使用する項目と、**knitr** や **rmarkdown** パッケージが参照する項目が複雑に入り混じっていて、初学者を混乱させる可能性があった。Quarto ではこのようなメタデータの構造が一新され、かなり洗練された。例えば `header-includes` のように、従来は HTML や PDF の出力ごとに個別設定できない項目というのがいくつもあった。^{*5} 複数媒体でレイアウトを揃えようとするこの制約が邪魔になることが結構あったのだが、Quarto では `format` フィールド以下にネストすることで、出力フォーマットごとに設定できたりする。knitr のデフォルトのチャンクオプション (Quarto では Execution options と呼ばれる) もほぼ全て YAML メタデータの `execute` フィールドに書くことができるようだ (以前はこれもまた `knitr::opts_chunk$set()` と YAML メタデータ内の項目とで重複があったり複雑だった)。

^{*5} 指摘があったので補足: 工夫すれば `header-includes` でも出力ごとに競合しづらい書き方もできるが、ソースファイル 1 つで自己完結したい、編集しやすくないという観点ではあまりおすすめできない

R Markdown ユーザなら以下の書き方がこれまで不可能であったこと、そして可能だったらどれだけ便利かわかるかもしれない。

```
format:
  pdf:
    header-includes:
      \usepackage[hiragino-pro]{zxjafont}
    execute:
      echo: true
  html:
    execute:
      echo: false
execute:
  eval: false
```

💡 Tip

ただし、Quarto の実行オプションと knitr のチャンクオプションは似ているが完全互換ではないようだ。(例: `fig.cap` と `fig-cap`) しかし knitr エンジンでは実行オプションと knitr のチャンクオプション両方が使えるようで、少し混乱するかもしれない。

さらに、`editor_options` というフィールドで、編集時の挙動や、Pandoc 変換時の文書全般の設定をわかりやすく制御できるようになった。^{*6} その中には参考文献の表示位置 (セクション末尾ごとか、巻末にまとめてか) とか自動折返しの文字数などが含まれている。

(u)pLaTeX の使用もしやすくなりそう

R Markdown では TinyTeX の latexmk エミュレーション機能 (`tinytex::latexmk` 関数) を利用して LaTeX のコンパイルを行っていた。しかし R Markdown 側が (u)pLaTeX の使用を想定していなかったので、昔からのコード資産 (例えば科研費 LaTeX とか?) に頼っていて XeLaTeX や LuaLaTeX に移行できない人にとってはやりづらかった。YAML メタデータを一新したことで、メタデータ上でこの辺の制御も設定できるようになった。^{*7}

^{*6} <https://quarto.org/docs/visual-editor/markdown.html#writer-options>

^{*7} 動作保証はできないが、<https://zrbabbler.hatenablog.com/entry/2019/05/06/235854> なんかを参考に対応する項目を YAML メタデータに設定したり `.latexmkrc` を用意したりすればできるかもしれない。

新しい記法のチャンクオプション

これも従来の R Markdown ユーザ向けの解説. R Markdown にあまり詳しくなくて, Jupyter の利用環境が快適になることを期待するしているなら, セクション まで飛ばしても良い.

Quarto でも R Markdown (**knitr**) のコードチャンクが使用できる. それとは別に, **実行オプション** (Execution Options) と呼ばれる機能が用意されている. これは先日リリースされた knitr v1.35 での記法とよく似ている (なお, デフォルトでは実行オプションは表示されないが, `echo: fenced` というオプションを使うことでコードチャンクの囲み部分も含めて表示することができる.).

```
```{r}
#| label: code1
#| eval: true
#| output: asis

1 + 1
```
```

[1] 2

knitr でこの記法もサポートするようになった理由ははっきりしないが, 将来的に R Markdown の仕様を Quarto に寄せるか, あるいは knitr の機能を Quarto に使えるようにする意図があるのかもしれない. ただし現時点では, それぞれで使えるオプションはかなり似ているものの, 互換性があるわけではなさそうだ.

ではなぜ Quarto で新しい記法を採用するようになったか, それはおそらく, R Markdown 以外, つまり Jupyter でも使えるようにするためだろう.

相互参照を使いやすくした上で最初からサポート

Jupyter の話の前に, Quarto の実行オプションによるもっと直接のメリットも紹介する. R Markdown は単体では相互参照をサポートしていなかった. Pandoc の文献引用機能, LaTeX のコマンドの直接入力などによって部分的にはできるが, 統一感がなく便利とはいえない. 図表, ヘッダ, 引用文献の相互参照には **bookdown** パッケージが必要で, その上技術的な理由で参照の構文が統一されていなかった. 例えば図は `\@ (fig:image)` と書けるが, 章や節を参照するには `\@ (section1)` のように書いていた. セクションヘッダと図と表とでキャプションラベルの付け方が全部異なるのもややこしい. その点 Quarto の実行オプションでは, 一貫性のある構文で相互参照をサポートしている. もちろん **bookdown** で使用できた, 数学書にあるような「定理 XX」「命題 XX」といった `amsthm.sty` 準拠の相互参照も使用できる.

具体的には, チャンクのラベルと図表のラベルを別々に定義できるように変更し (これによって subfigure のレイアウト設定もより簡単になった), 相互参照は全て `@<媒体の種類>-<ラベル名>` という形式で参照できるように

なった。例えばグラフを出力するチャンクに、図のラベルとして `fig-plot1` などと定義すれば `@fig-plot1` で参照できる。さらに、「図 XX」「表 XX」のような接頭語を設定して自動で付加できるようにもなっている。

以下の例は **ggplot2** で複数のグラフを描き (図 3), キャプションをつけて表示するもの。オプションの名称は R Markdown からあまり変わっていない。サブフィギュアのラベルは自動生成され, 図 3a, 図 3b というふうにこちらも簡単に参照できる。

```
library(ggplot2)
ggplot(airquality, aes(Temp, Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", se = F, formula = y ~ x)

ggplot(mtcars, aes(x = factor(cyl), y = mpg)) + geom_boxplot()
```

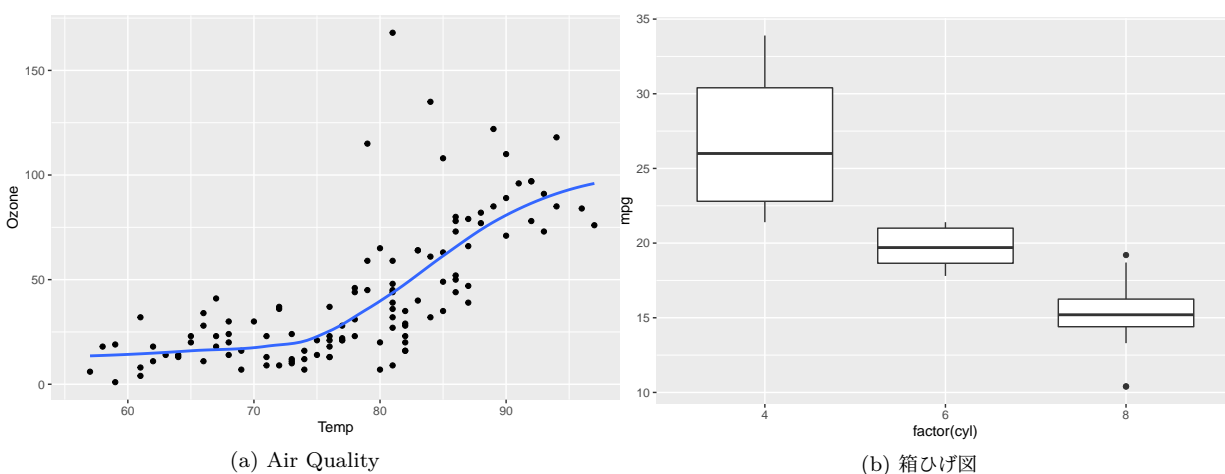


図 3: 複数の図

セクション見出しへの相互参照も, 見出し側に `{#sec-<ラベル名>}` のように指定すれば, セクション というふうに参照できる。

相互参照に関する詳しい話は公式の [Cross References](#) のセクションを見てほしい。

Jupyter や VS Code への対応

私自身は Jupyter をあまり好まないが, これが利用者層を増やす可能性が一番ありそうな, **大きな特徴**であると思う。Jupyter(lab) は IDE としてもドキュメントエディタとしても機能がどっちつかずなので (この

辺の不満を書いていたらとても長くなったので脚注にした^{*8}), Jupyter にも knitr のチャンクオプションのような機能が用意され、簡単にソースコードの表示を切り替えたりできないか、あるいは公式に互換性をサポートしてくれたら良いなどと思いながら実現されないで私は Jupyter を使わない作業スタイルを追求し続けていたが、先に Quarto がこれを実現することになりそうだ。

Quarto は Jupyter でも Rmarkdown のチャンクオプションと似た機能を提供するため、**実行オプション** (Execution Options) という機能を用意している。これによって、

- 長大なコードを折りたたんで出力する (クリックで展開するアレ)
- 読み込みログや大して意味のない大量の警告メッセージ (しかし開発者にとっては一応確認する必要があるもので消せない) を文書生成時に非表示にする
- 逆にコードの解説のため、コードのみの出力する
- **図表のキャプション表示と相互参照**する

といったことが全てできるようになり、整った文書を簡単に出力できるようになる。もちろん YAML メタデータも与えられるから、HTML や PDF 生成するときの全体的なスタイル変更も、従来の Jupyter 側のエクスポート機能を使うよりかなり簡単だ。

具体的にはこのような使い方になる。セル内の冒頭に `#|` の後に続けて書く記法で、HTML や PDF に変換したときの表示方法 (notebook 上での表示ではない) を制御することができる。例えば以下の `echo: false` はコードを表示しない、`warning: false` は警告を表示しない、という意味である。グラフのみ掲載して共有したい場合などに便利だろう。

同様に、VS Code の Jupyter Notebook インターフェイス上でも Quarto を使用できるようだ。Google Colaboratory も ipynb 形式でファイルをダウンロードすればたぶん使えると思う (普段使わないので詳細は不明)。

(実は先日公開した [Pysocviz](#) も、notebook で用例を作成した上で Quarto で PDF などの静的なドキュメントに変換しようと考えていたが、開発版で不安定なのでメンテが大変になるかもと思い直してやめたという経緯がある。)

多様な Jupyter カーネルへの対応

Jupyter への対応についてももう 1 つ強調すべきは、Quarto 文書ファイルの新規作成時の設定項目に Jupyter カーネルを指定する `jupyter` という項目があること。公式ドキュメントでは Python の例しかなかったが、こ

^{*8} Jupyter Notebook (公式は今も Jupyter の開発を積極的には行わないので JupyterLab を使えと言っているが、便利なプラグインのほとんどが使えなくなってしまったので私は JupyterLab を使っていない) は IDE としては機能が少なすぎるし、「インタラクティブな開発環境」と自称するわりには Python はじめインタラクティブに扱える言語の特性を活かした機能が少ない。かと言ってドキュメントやレポートを作成するツールとして見ると、長大なソースコードや警告メッセージがそのまま表示され、スライドとしての体をなさない、PDF エクスポート時のオプションがなさすぎるなど、機能が貧弱である。Jupyter notebook をプレゼンテーションに変換する RISE という拡張機能があるが、R Markdown に比べると機能が少なすぎて使いづらい (というか相変わらずコードや警告を非表示にできない/難しい) ので諦めてしまった。

```
[1]: #| echo: false
      #| warning: false

      1 + 1
```

図 4: Jupyter Notebook のセルに実行オプションを記述する例

のような項目があるということは、公式ドキュメントには明記していないものの **Quarto は Python 以外の Jupyter カーネルも対応している**、あるいは将来そうする予定なのではないか、ということで **Julia** カーネルを指定してみたところ、HTML 版のみオプションも含め動作した。R Markdown では **JuliaCall** というパッケージを使い Julia コードのチャンクを書くことができるが、現在はグラフの表示がうまくいかなかったりと制約が多い。しかし Quarto ではそのような問題は見られなかった。^{*9}

Jupyter カーネルはかなりいろいろな言語に対応している^{*10}が、実際にこのレポート作成などと相性がよさそうなのは R, Julia, Matlab, Octave あたりぐらいだと思うが後二者は持ってない & ほとんど使ってないので R と Julia の例のみ付録した。^{*11}

なお、Julia 側のインターフェースも作ってるようだ (おそらく作りかけなので未検証): <https://github.com/quarto-dev/quarto-julia>

現時点での制約

現時点で気になった点・注意点を挙げていく。まだ開発版なのでおそらく今後どんどん不具合が修正され機能も追加されていくと思うが。

- グラフィックデバイスの扱いが不明瞭。日本語ユーザにとって現状これが一番ネックな気がする。最近書いたように Windows や Mac でグラフに書いた日本語を文字化けさせないためには **ragg** パッケージのデバイス関数や `cairo_pdf` が使えるといいのだが、指定方法がはっきり書かれていない。現状は `.qmd` な

^{*9} 実際には、現時点では Gadfly.jl のグラフがどうしても SVG で保存されてしまうため、PDF 形式で保存するのが少し面倒である。PDF のみの処理を場合分けして作れば動作するだろうが、面倒だったので作らなかった。

^{*10} <https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>

^{*11} なお、カーネルの名称は `jupyter kernelspec list` で一覧を表示できる。

らば `#| dev: cairo_pdf` のように書けば適用されるようだが、一方で YAML メタデータの `execute:` に書いても認識されないようだ (Quarto ではなく **knitr** の機能が適用されている?)。

- YAML メタデータのブール値で `yes/no` が使えない。 `true/false` のみ
- R Markdown の出力フォーマット関数に対応するものがない
 - 私が作った **rmdja** は出力フォーマット内で日本語文書用の独自テンプレートを呼び出したり、設定を自動調整したりしていたが、それができなくなった。 `knitr` のフックやフィルタで代用できそうなどところもあるが、複雑な処理を内部化し、ユーザがほとんど何も設定しなくてもいいようなテンプレートを用意するのは難しい
 - Markdown や HTML に対応しているように見えて実は完全に互換性があるわけではなかったり独自仕様だったりするサービスは結構あるのでこれは少し困る
- Jupyter の変換も便利だがまだ足りない機能も多い
 - たとえば Python で描いたグラフに対して相互参照やキャプションをつけることができるが、 `kable()` のような表は作れない (Pandas の表示に対応していない)
 - Julia は HTML ならおおむね問題なく動くが、PDF 出力時の画像表示がやや使いづらい
 - * `qmd` を使用した場合は従来どおり **JuliaCall** が使われてしまう

Quarto の導入方法

Quarto の導入に特に難しいことはないが、上記のように開発版であるため動作が不安定である可能性が高いことには注意してほしい。 Quarto はすでに公式ドキュメントがある程度充実しており、[インストール方法に関するページ](#)も作られているので、ここでは多くを書かない。 GitHub から Quarto CLI の最新のリリースをダウンロードする。 `.deb`, `.pkg`, `.msi` の 3 通りのバイナリ/インストーラが用意されているので、主要な OS は簡単にインストールできる。 Quarto CLI 単体だとターミナルでの呼び出しでしか使えないため、「Pandoc の機能拡張版」というような認識で良いだろう。 インストールできたら、RStudio を再起動すると新規作成ファイルの一覧に “Quarto Doc” が追加されているはずであるので選択する。 R 側で操作するための **quarto** パッケージもインストールが必要である。^{*12}

その他 TeX や Pandoc の設定が必要な場合もあるが、R Markdown を既に使っているならその設定を引き継げることが多いので説明は省略する。 Jupyter 上で使いたい場合はもちろん Python や Jupyter が使えるようにする必要がある。 これらの環境設定の多くはターミナルの `quarto` コマンドで確認できる。

RStudio 上での使用法

今後変更される可能性が高いので、あまり細かいことは書かない。

^{*12} 必要だとは書いているが、現時点ではなくても動作するのであまり気にする必要はないかもしれない。 中身を少し見た限りではドキュメントのエクスポートなどに関する関数ばかりで使わない人のほうが多そうである。 RStudio ではなくコンソールのみで操作したい人向け？

RStudio 上で knitr/jupyter を使用する場合

RStudio で Quarto Doc を新規作成すると、設定ウィンドウが開かれる。この設定は .Rmd の YAML メタデータに対応する部分で、後から書き換えることもできる。とりあえず エンジンが knitr になっていることだけ確認してほしい (図 5)。

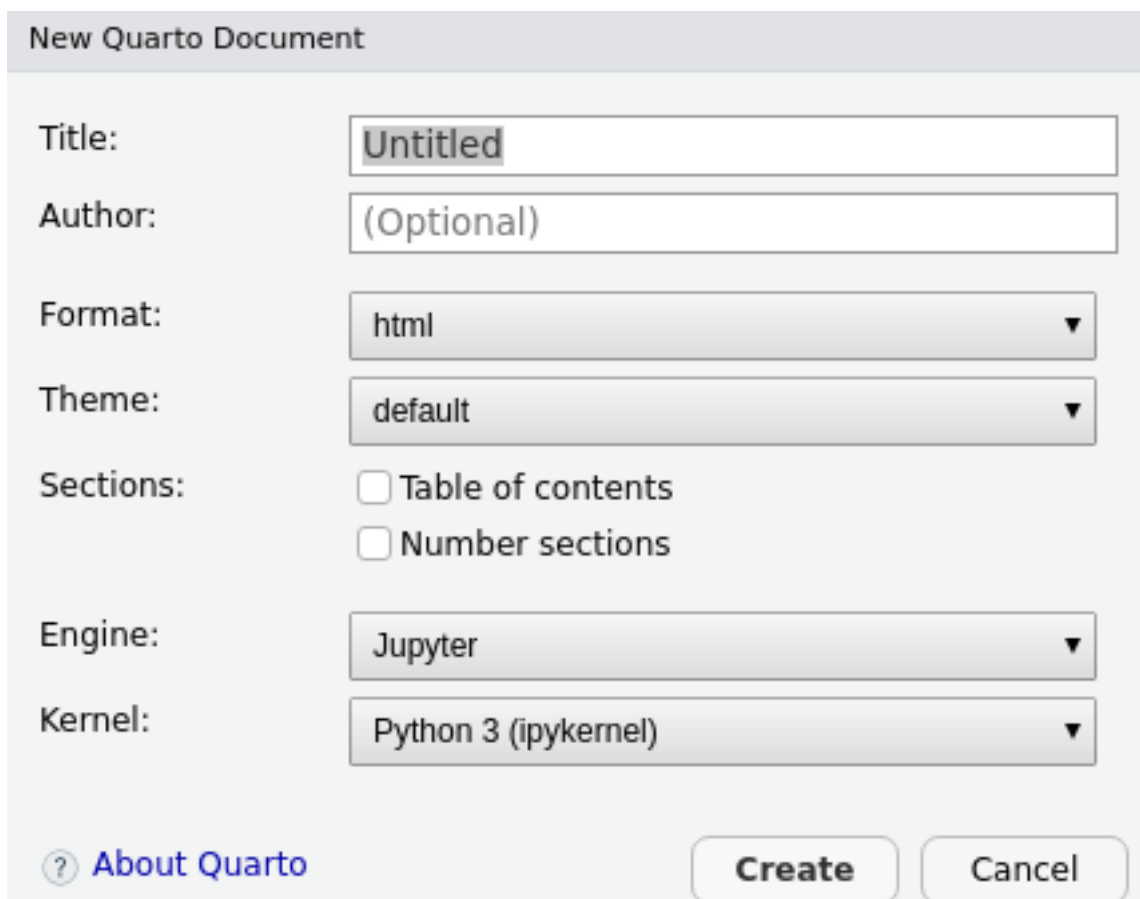


図 5: qmd ファイル新規作成時の画面

“Create” を押すと .qmd ファイルが作られる。エディタ上での見た目は .Rmd とそっくりそのままであり、YAML メタデータ + Markdown + knitr チャンクで構成される。細かい構文の違いは既にかいたとおり。これも既にかいたがビジュアルエディタモードも使える。

R Markdown で “knit” ボタンだったものが “Render” に変わっており、これで qmd ファイルをコンパイルできる。

エンジンに Jupyter を選択した場合も、テンプレートのチャンクエンジンが Python になるだけでほぼ同じである。この場合は **reticulate** のインストールと設定が必要なことに注意。

(なお、現時点では“Preview in Window”を選択してもなぜか Viewer ペーンに表示されてしまう。PDF の場合はこの手の組み込みビューアの例にもれず、文字表示がちょくちょくおかしくなるので注意。PDF が正しく出力されないと慌てる前に Okular とか Sumatra とかで見たほうがいい。)

Jupyter 上で Python3 カーネルを使用する場合

公式ドキュメントの対応する箇所はこちら: <https://quarto.org/docs/tools/jupyter-lab.html>

Jupyter(lab) を起動した状態で、ターミナルで以下を実行する (JupyterHub を使ってリモートサーバ上で動いている場合はもちろんホスト側のターミナルで)

```
quarto preview notebook.ipynb
```

これはリアルタイムプレビューで、notebook 側の変更がすぐに反映される。つまり R Notebook のような機能。R Markdown の Knit ボタンのように文書を生成するには `quarto render` コマンドを使う。

```
quarto render notebook.ipynb
```

`format:` の形式が複数書かれている場合は 1 番上のものが適用されるので、明示的に指定したい場合は、`--to pdf` のようなオプションを追加する。

Jupyter は元から Markdown セルを使用できるため、Markdown セル内の記述が `qmd/Rmd` の地の文に対応する。出力オプションも R の場合と同様に notebook のセル内に実行オプションを書くことで適用される。

! 重要

R Markdown はデフォルトの設定では Knit 時に全てのチャンクを実行するが、`.ipynb` ファイルに対して `quarto render` を使っただけでは**セルが再実行されず**、`.ipynb` 内部に保存されたものだけを反映する。生成時に改めて実行するには `--execute` オプションを追加する。逆に言えば、既存の `.ipynb` ファイルを `quarto render` で変換するだけなら Jupyter 本体は不要である。

また、`quarto convert` コマンドで `.ipynb` と `.qmd` を相互に変換できるらしい (今回は試していないので詳しい挙動はわからない。)

VS Code 上で Jupyter を操作する場合

公式ドキュメントの該当箇所: <https://quarto.org/docs/tools/vscode.html>

VS Code で Python 拡張機能を入れていれば Jupyter Notebook が扱える。ここでもセルに出力オプションを書いたり YAML フロントマターを書いたりできる。VS Code のエディタではセルの使用言語を Python と Markdown 以外にする方法が分かりづらいが、セル右下の [Python] とか [Markdown] とか表示されている箇所

をクリックすればドロップダウンリストが出てくるので、“raw” を選べば YAML メタデータとして認識される (図 6).

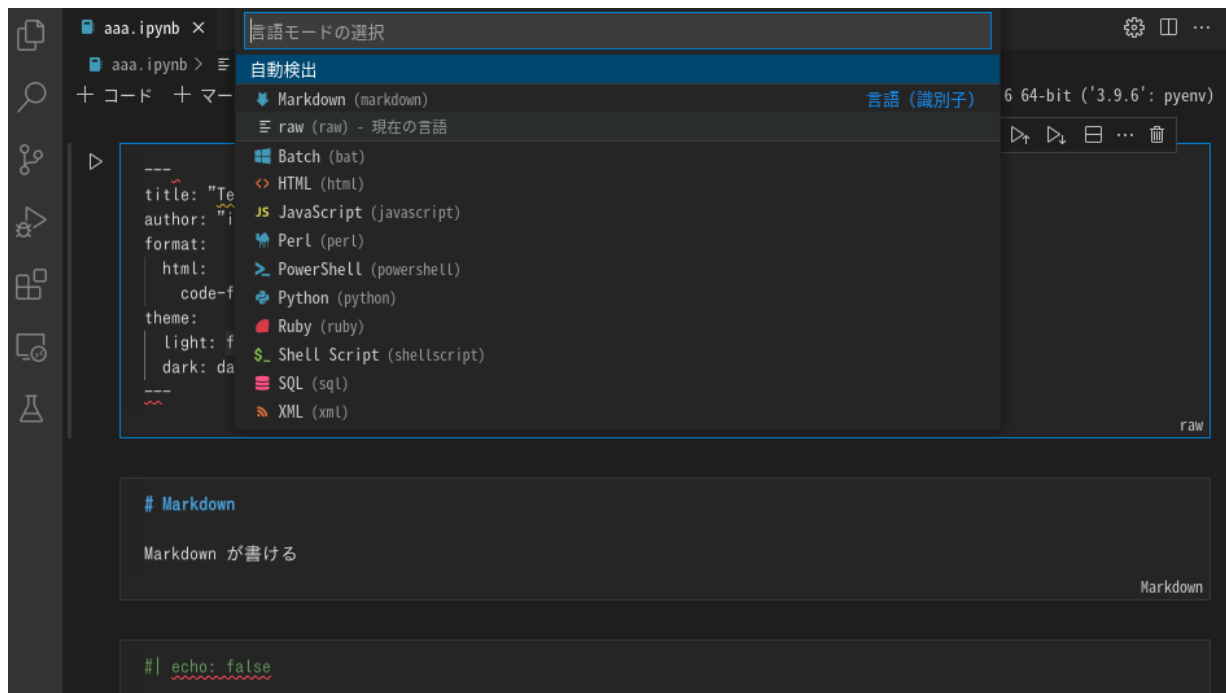


図 6: VS Code 上で YAML メタデータを書くための設定

ドキュメントの生成方法は通常の Jupyter と同じ.

💡 Tip

インタラクティブモードではセルの種類を指定できないので使えない?

Jupyter 上で Julia カーネルを使用する場合

公式ドキュメントには書かれていないが, Jupyter に Julia カーネルを登録すれば, 同じ要領で Julia のコードを埋め込んだ文書を作ることでもできた. ただし, Julia の Plots や Gadfly はデフォルトでは SVG 形式で画像を出力するため, PDF での出力がうまくいかない. 今回はとりあえず PNG に変更して対処したが, Gadfly のみうまく動作していないようだ.

まとめ

Jupyter (Jupyterlab) は結構使われているが, 個人的には IDE としてもドキュメント作成ツールとしてもどっちつかずな感があった. それは既に述べたように, IDE としての機能が少なく, 対話的処理であると便利な機能

もほとんどない (Jupyter にはプラグインで一応あるが, lab にはまだ対応していない). 逆にドキュメント作成ツールとして見ると, 長大なコードを隠すことができず, 技術文書や論文等で必要なフォーマットに変換するのも大変である. 去年登場した Jupyter Book もオンラインドキュメントの作成には便利そうだが, R Markdown のようにいろいろな媒体への出力を想定していないようだった.

しかし, Quarto の変換機能を使えばドキュメント作成ツールとしての使い勝手はかなり改善されると思う. Jupyter はもともと Markdown に対応しており, また実行オプションや YAML メタデータの記述は既存の ipynb ファイルの設定と競合することは少ないと思われるため, 気軽に試すことができる. もちろん, 既にかいたように細かい不具合や不満はあるが, それでも清書してない .ipynb を共有するよりは見た目が良くなるのではないだろうか.

先日更新された **knitr** の新機能は, Quarto の新機能を狙ったものにみえる. RStudio 社としては, 今後積極的に Quarto をアップデートしていくのではないかと勝手に期待している.

つまり, おまへは万物を Quarto で書ける日が近いことが証明されたかたちだ. ~~そして **rmrj**~~ も来年くらいには不要になってそう

付録: 各種設定での出力例

- 冒頭に書いたようにこの投稿は Quarto で生成したものだが, R Markdown および Quarto の強みは同一のソースを少し設定を変えるだけで Markdown や HTML や PDF の文書で生成できる点にある. よって, この同一のファイルで作成した例を [GitHub](#) に投稿しておいた. ただし, DOCX (Word) や PPTX (パワーポイント) は確認がめんどくさいので Libre Office でざっと閲覧したのみで細かく確認してない. 興味のある人だけ各々確認してほしい.

まだ開発途上なので, 思ったよりうまく動作しない箇所も多い. ちょっと遠回りな方法で解決できそうなケースもあったが, 将来のアップデートで使いやすくなることを期待して放置した.

- Quarto-introduction.qmd はこの投稿を生成した原稿である. 一応 PDF 出力もできるように設定してある.
- Knitr-Ja/ は最低限の日本語組版ができそうな PDF の設定例である. 一応 HTML 出力ともある程度互換性があるように設定している.
 - 「最低限」なので文献リストは BibLaTeX を使用している. 日本語ユーザの多くは (u)pBibTeX に頼っているが, これは設定や使い方少し複雑なので, 別のファイル Quarto-Ja-upbibtex.* を用意している.
 - (追記) upLaTeX でも動作させる例, Knitr-ja-up-bx.* を追加した. しかし現状は BXjscls の文書クラスにしか対応していない.
- Jupyter-Py/ は Python カーネルを使用した Jupyter Notebook および, それを Quarto で HTML と PDF に変換したものである. Quarto の機能を見るためにいろいろ変わった設定を試している.

- Jupyter-R/* は R カーネルを使用した Jupyter Notebook および、それを Quarto で HTML と PDF に変換したものである。(これが必要な状況があるのかはわからない)
- Jupyter-Julia/ は Julia カーネルを使用した Jupyter Notebook および、それを Quarto で変換したものである。PDF は作っていないが、Gadfly.jl の保存形式の問題を回避できれば生成できると思われる。
- Jupyter-slide/ は脚注で Jupyter や RISE の機能を批判しまくったので、責任を取るべく Quarto ではスライド資料をより簡単に作れると実証したものである。カーネルは Python を使っている。PDF (beamer), PPTX, HTML (revealjs) の設定をしてあるが、実際やってみたら HTML の出力はあまりよろしくなかった。pandas の出力はうまく表示できないが、Markdown の表はたぶん表示できると思う。
- Knitr-slide/* は .qmd でスライドを作成する場合の例である。こちらは R を使っているので、従来の R Markdown と比較してみてほしい。

公式ドキュメントでもいくつかの使用例が紹介されている。

補足: R Markdown と Quarto の LaTeX 設定の比較

`rmdja` では柔軟な設定ができるよう、pandoc テンプレートに大幅に手を加えたりフォーマット関数による動的な調整を行ったりしていたが、そういうのなしでエンドユーザ向けの設定だけでなるべく簡単に済ませようとした場合の設定を比較すると、両者の設定はフィールド名以外同じになる。

R Markdown では YAML メタデータの記述はこうなる。

```
output:
  pdf_document:
    latex_engine: lualatex
documentclass: bxjsarticle
classoption:
- pandoc
- ja=standard
- jafont=haranoaji
```

Quarto で同等のことをやろうとすると、こうなる。

```
format:
  pdf:
    documentclass: bxjsarticle
    classoption:
      - pandoc
      - ja=standard
```



```
- jafont=haranoaji
pdf-engine: lualatex
```

どちらもほぼ Pandoc の引数に対応させているだけなので、結果として同じになる。また、文書クラスに `bxjsarticle` を使用^{*13}することで、日本語フォントの設定も簡単にしている。^{*14} ^{*15} 上記では LuaLaTeX を使用しているが、XeLaTeX でも動作するはずである。これは `bxjsarticle` が自動的にそれぞれのエンジンでの組版に必要なパッケージを読み込んでくれるためである。プリセットが気に入らず、メインフォント、サンセリフフォントなどを個別に設定したい、ウエイトを調整したいという場合は LaTeX コマンドで追加設定する必要がある。この設定は XeLaTeX か LuaLaTeX かで変わってくる。前者は `zxjatype` を、後者は `luatex-japan` のマニュアルを参考にしてほしい。

しかし、現在は `xelatex` よりも `lualatex` の使用を推奨する。理由は以下の 2 つ。

1. `zxjatype` 作者が述懐するように、XeLaTeX を選択した場合に使われる `zxjatype` の組版はあまり厳格ではない
2. LuaLaTeX は以前から (up-TeX や XeLaTeX と比べ) 処理が遅いと言われているが、TinyTeX を使用する環境では LuaLaTeX の遅さ以外にボトルネックがあるのであまり気にならない

ただし、beamer に関しては逆に禁則処理が厳格でなくても気になることがあまりなく、かつ人気のある metropolis テーマは XeLaTeX での動作を想定しているので XeLaTeX を使用したほうが良いだろう。

なお、ネット上では別のパターンの設定も存在するので、蛇足気味だがそのあたりの違いについても書いておく。

例えば、『[TinyTeX のインストール& Rmarkdown で PDF on Windows10](#)』というページ。この設定例であればタイトルの Windows 10 に限らず比較的他の OS でも動作しやすい。ただし誤りでないがいくつか注意が必要である。

- TinyTeX を R パッケージを使わずにインストールしてしまっている。R も使うのなら `tinytex` パッケージ経由でインストールしたほうが管理が簡単である (ブログ筆者は Atom で編集する想定のためこうした?)。詳細は[私の翻訳した公式ドキュメント](#)参照。
- Noto フォントで揃える場合は `header-includes:` 以下は不要で、上記のように、`classoption` 内で `jafont=noto` のようにプリセットを設定できる。
 - 私の例では Noto ではなく TinyTeX からインストールできる原ノ味フォント (`haranoaji`) を使用している。私が `haranoaji` を指定しているのは、LuaLaTeX で Noto フォントを使う場合は `noto-otc` とか `noto-otf` とか名前が変わってややこしいため。

^{*13} `bxjscls` というまともりで、`bxjreport` や `bxjsbook` も存在するが、以降は便宜上 `bxjsarticle` と表記する。

^{*14} ここでは、最近の TeX Live の標準日本語フォントである `haranoaji` (原ノ味) を指定している。OS ごとに標準日本語フォントが異なるため、OS 依存をなくすためにこれを選択した。TeX Live 2020 以降ならそのまま使用でき、`tinytex` を使っているなら `tinytex::tlmgr_install("haranoaji")` でインストールできる。

^{*15} この他にも `bxjsarticle` には、1 行あたりの文字数、1 ページあたりの行数を指定するオプションがあるという利点がある。

- あと IPA フォントをインストールするという指示があるが結局使っていない。

さらに、例えば以下のようなページではまた別の設定例が紹介されている。

- 『[R Markdown + XeLaTeX で日本語含め好きなフォントを使って PDF を出力する](#)』
- 『[メモ: Pandoc+LaTeX で気軽に日本語 PDF を出力する - Qiita](#)』

CJKmainfont などを使う場合も確かに日本語フォントを指定し表示させられるが、現在ではややバッドノウハウと思える。CJKfont は中国語の組版を想定したもの (xeCJK) であり、日本語環境を想定した設定ではないため、微妙に不自然になることがある (zxjatype も組版設定が甘いのでそこばかりしつこく突くのもフェアではないが)。現状一番簡単な設定の書き方として、私が最初に提示した、そしてこれらの記事の下の方でも少し紹介されている bxjscls シリーズの文書クラスを使うことをお勧めする。現在は特にプリアンプルを書き換えたりしなくともエラーが発生しないようになっている (はず)。このあたりの仕様が変わったのは比較的最近なので、2-3 年前の記事だとあまり反映されていないのだろう。

ただし、もしこのプリセットが納得行かず、3 種類のフォントでそれぞれ違う系統のものを使いたい場合はこの限りではない (しかしその場合も bxjscls でプリセットを指定した上で上書きしたほうが使いやすいはずである)。

その他、多くのブログ等で

```
header-includes:  
- \usepackage{zxjatype}  
- \usepackage[HOGEHOG]{zxjafont}
```

のような zxjatype や zxjafont を手動で読み込む LaTeX コマンドを書く例を紹介している人がいるが、これらは XeLaTeX でのみ使用可能であり、また使用できるフォントは OS によって変わってくるため書き換えが面倒であり私はおすすめしない。これもやはり、bxjsarticle であれば LuaLaTeX を使うか XeLaTeX を使うかによって必要なパッケージを自動で読み込んでくれる。ただし、beamer の場合は日本語フォントを設定するオプションがないので自分で書く必要がある (詳しくは beamer のサンプルを参考に)。

さらに日付の古いページだと、

```
output:  
  pdf_document:  
    ...  
documentclass: bxjsarticle  
mainfont: ...  
sansfont: ...
```

といった記述例を紹介するものが見られる。あるいは documentclass: の設定が抜けているものも見られる。こ

れらも一応動作するが、欧文用の設定に日本語フォントを放り込んだだけなので、フォントスタイルの適用がおかしくなったり、見出しの一部が英語になったりしてしまう。よって現在は積極的に使う必要はない。