New Ways of Publishing: A Roadmap to Authoring Online Training Material

Contents

- Charting the landscape
- From static and dynamic to interactive content
- Some itineraries through the landscape
- Concluding remarks
- Appendix A: Open Courses
- Appendix B: Development environment
- References

JB jupyter book

This website provides a set of guidelines to publish online training material using state-of-the-art web authoring tools, and also serves as an example authored using the `Jupyter Book` framework for producing documents from computational content. It has been prepared as a deliverable of the TAILOR Network of Trustworthy AI through Integrating Learning, Optimisation and Reasoning as part of Work Package 9: Network Collaboration, and is made freely available to the academic community.

These new authoring and formatting tools give rise to **new ways of working and publishing**. For example, `Jupyter Notebooks`[1] can be used in teaching, for self-study, as lab notebooks, for research collaborations, and in a host of other ways. But there are many other recent developments that open further avenues for authoring and publishing dynamic and interactive training material. Knowing about these developments and opportunities helps academic writers to publish their training material in the best possible forms.

This **roadmap** therefore has twin objectives:

- to chart the ever-growing landscape of publishing workflows, formatting tools and authoring tools;
- to provide some itineraries through this landscape, such as converting an existing LaTeX Beamer presentation to `Quarto`.

The companion website https://tailor-uob.github.io/mooc_trustworthy_ai/ shows the rendered results. The content can be viewed in a variety of ways, following the Single-In-Multiple-Out paradigm.

Who should read this

We prepared this material for an audience with experience in authoring AI-related training material using well-established tools such as LaTeX, Overleaf, Google Docs etc. You will learn about the latest tools and frameworks such as `Jupyter Book` and `Quarto`. These tools make it easier to deliver content in a variety of ways, and also offer the opportunity to add interactive elements. We give examples of possible workflows to get you started.

How to read this

The structure of the document is fairly self-explanatory.

- If your main experience is with tools such as LaTex and Google Docs, you probably want to read the whole document.
- If you already have experience with `Jupyter Book` and the markdown format, you may want to skip to Formatting tools which describes the most recent developments, and follow on from there with the From static and dynamic to interactive content section which gives many examples of static, dynamic and interactive content.
- If you are particularly interested in how to produce `Quarto` content, head to Some itineraries through the landscape which provides concrete use cases of authoring training material using `Quarto`, the rendered output of which can be viewed on the companion website.

The full table of contents is given in the panel on the left, each element of which is a separate webpage. You can step through these pages in order by following the `Next` link at the bottom of each page. If the page has an internal heading structure this is indicated in a panel on the top-right. The `Jupyter Book` button at the top downloads the entire website as a single PDF file.

Note

This is a near-final draft currently under review.

[1] `Jupyter Book` and `Jupyter Notebook` are related but different, see Dynamic content.

Charting the landscape

The landscape of tools for authoring and publishing training material is vast, and creating new or upgrading legacy material may appear daunting in the face of all available options. This roadmap surveys the landscape and provides actionable guidelines for publishing modern online training materials with state-of-the-art tools. We focus on Publishing workflows that use standard authoring artefacts (such as plain text, tables and figures) that can generate multiple publishing outputs (like slides, websites, pdfs, or documents).

The proliferation and adoption of the World Wide Web has allowed the modernisation of teaching material that is cross-platform and dynamic. Examples of modern teaching material include online videos (e.g. Khan Academy), online exams (e.g. Massive Open Online Courses or MOOCs), personal blogs with commenting facilities, and animated or interactive illustrations. However, large parts of the currently available teaching material still focus on printable formats, with the possible addition of hyperlinks for online delivery.

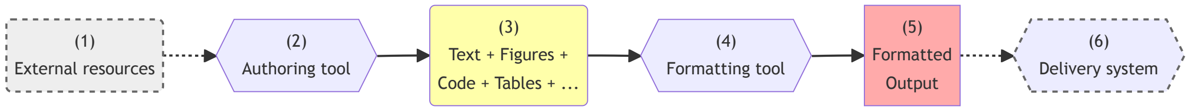
The lack of standards for publishing hinders the adoption of authoring formats and increases the efforts to improve individual publishing workflows. Furthermore, the time spent getting familiar with one technology may not be productive in future with the rapidly changing publishing environment. Additionally, multiple publishing outputs require different input formats which forces the duplication of efforts to produce very similar material (e.g. printed pdf, website, blog post, slides, posters).

This roadmap provides an introduction to modern Authoring tools and Formatting tools. We explore the most broadly adopted authoring and publishing platforms, as well as online collaboration tools to facilitate the creation of shared online material. We pay special attention to the possible standardisation of a workflow to minimise the effort of authoring artefacts while maximizing the range of output modalities. Two such publishing systems are `Jupyter Book` and `Quarto` for which multiple examples are provided.

We also provide some use-case examples of a full workflow to generate online training material (and this website is in fact one of them). All material is kept in the `GitHub` version control environment to maximise transparency and reproducibility while allowing the automatic generation of the output artefacts with `GitHub Actions`. The input artefact is constructed in such a way that multiple types of output formats can be generated automatically from them (e.g. formats like an HTML website, `reveal.js` slides, online or print PDFs, and e-books).

Publishing workflows

The **publishing workflow** is the process in which external resources are prepared and collated together with the help of an **authoring tool** and subsequently rendered with a **formatting tool** to generate a publishable output that can be delivered in various forms. This is visualised by the following figure.[1]



An example of a publishing workflow is the following. A research group may carry out an investigation which ends up with **(1)** data, figures, analyses, and results, those resources are put together with an **(2)** authoring tool like Overleaf which produces an **(3)** organised hierarchy of files and text that includes formatting information, then using a **(4)** formatting tool like pdfLaTeX all the artefacts are combined into a **(5)** pdf document that can be **(6a)** printed as a stand-alone document, or **(6b)** included into the proceedings of a conference.

Note

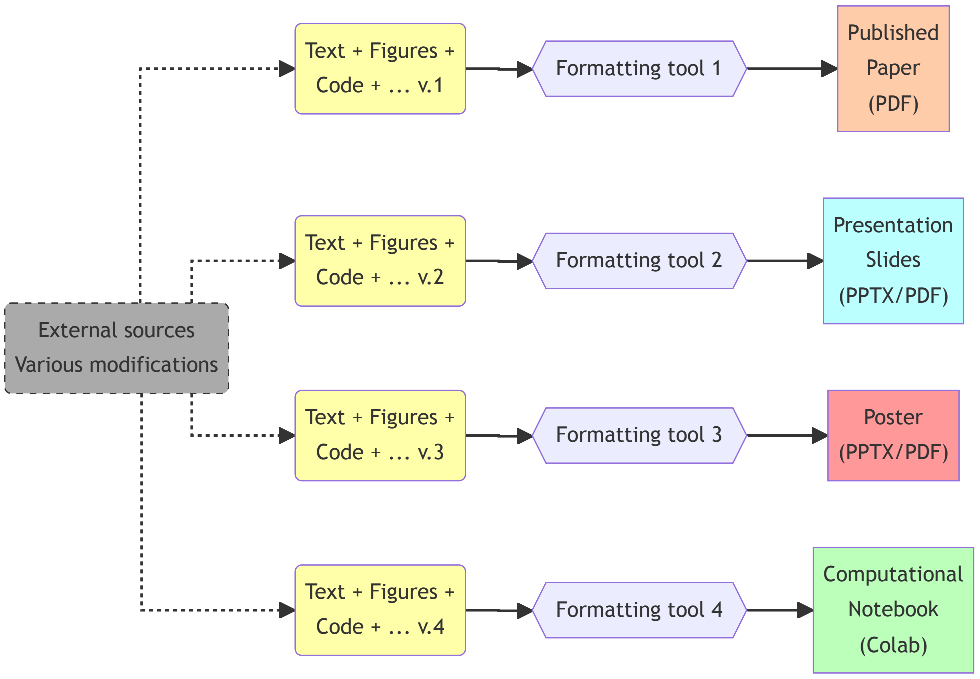
For the purposes of this roadmap we emphasise the difference between the *authoring* and the *formatting* steps of the full publishing workflow. The authoring step consists in creating a structured composition of input artefacts that are necessary to generate the intended publication through formatting.

The creation of research or academic material usually involves the production of slides, documents, posters and other output formats from the same material. In this roadmap, we are interested in workflows to generate this multitude of outputs minimising the *authoring* effort. Here, we describe two types of paradigms which were discussed in the previous TAILOR deliverable D9.3 [SF21]: Single-In-Single-Out (SISO) and Single-In-Multiple-Out (SIMO).

Single-In-Single-Out

Formatting tools commonly specialise in generating only one type of output from one or more input files. This is one of the most common approaches when producing an article for publication in a journal or conference proceedings. The text, figures, tables and other artefacts are assembled with some authoring tool and compiled with a specialised formatting tool to produce a printable pdf.

Once the paper has been accepted by a conference, a presentation will need to be prepared, often using different *formatting tools*. It is then common to have duplicates of the same content with mild modifications in a separate environment. The same process is often repeated if a poster, computational notebooks, or a website need to be created. We refer to this paradigm of publishing system as Single-In-Single-Out as each system works independently, while the authors need to keep up to date copies of similar content in each of them.



External sources
Various modifications
Text + Figures + Code + … v.1
Text + Figures + Code + … v.2
Text + Figures + Code + … v.3
Text + Figures + Code + … v.4
Formatting tool 1
Formatting tool 2
Formatting tool 3
Formatting tool 4
Published Paper (PDF)
Presentation Slides (PPTX/PDF)
Poster (PPTX/PDF)
Computational Notebook (Colab)

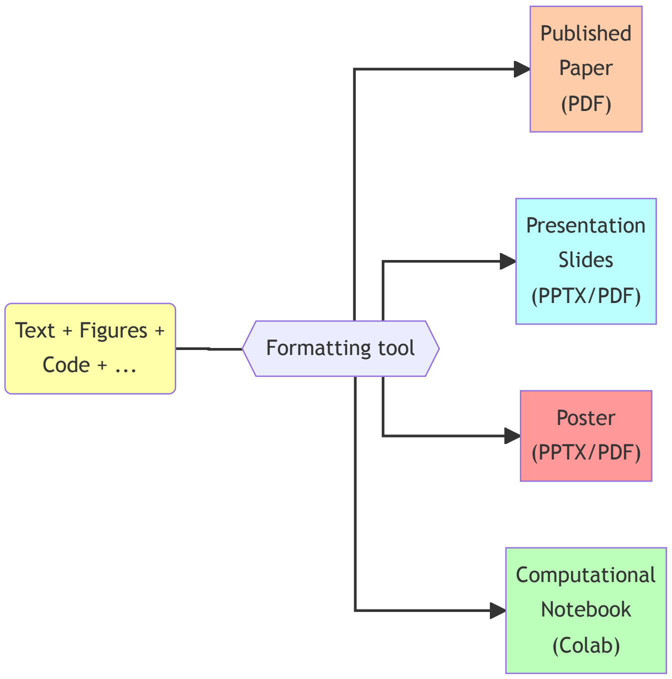
Common examples of this type of paradigm are the authoring and publishing workflows for creating

- text documents with Microsoft Word, Libre Office Writer, LaTeX;
- slide decks with Microsoft PowerPoint, LibreOffice Impress, LaTeX Beamer, Reveal.js, or Google Slides; and
- posters with Microsoft Publisher, Google Slides, LaTeX, etc.

Single-In-Multiple-Out

More recent publishing systems allow the generation of multiple types of publication formats from a joint set of input artefacts. We refer to this paradigm as Single-In-Multiple-Out (SIMO). This paradigm offers various benefits, among them:

- keeping the artefacts in a single, well-defined location which facilitates consistency, management and findability;
- keeping a unique source of history changes and versions which is useful for auditing and transparency; and
- not duplicating artefacts that are not changed between different publishing systems.



Text + Figures + Code + …
Formatting tool
Published Paper (PDF)
Presentation Slides (PPTX/PDF)
Poster (PPTX/PDF)
Computational Notebook (Colab)

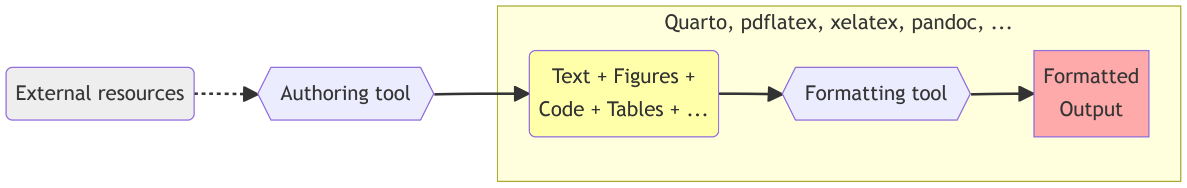
Nowadays, the SIMO type of formatting tool is becoming more common, and we focus on the state-of-the-art formatting tools that fall into this category in this roadmap. The roadmap document itself is an example of one particular tool (`Jupyter Book`), while the rest of the document and use cases also include examples built with `Quarto`. A description of these and other publishing tools is provided with more detail in the next section Formatting tools.

---

[1] In this roadmap we do not cover **(1)** the process of generating the external resources and **(6)** the delivery system. However, note that the delivery system may have some impact on the formatted artefact, even within the same format. For example, hyperlinks should ideally be handled differently for printed and online PDFs.

Formatting tools

Formatting tools take input artefacts produced by authoring tools and produce publishable formatted output. While many of the authoring tools discussed in the previous section are well-known, there have been many recent developments in formatting tools that deserve to be better known in the academic community as they take opportunities for publishing training material to the next level. In the previous section we described the two paradigms Single-In-Single-Out (SISO) and Single-In-Multiple-Out (SIMO). In this section we describe some *formatting tools* that can be used for the SIMO paradigm.


Quarto, pdflatex, xelatex, pandoc, ...
External resources
Authoring tool
Text + Figures + Code + Tables + ...
Formatting tool
Formatted Output

Pandoc

Pandoc is a tool to convert files between multiple markup formats. Pandoc is used within more generic tools such as Quarto. It is customizable thanks to a Haskell library and a template system to feet your needs. It is able to generate bibliographies, footnotes, LaTeX math, tables, definitions, and most common publication assets. Some supported formats are lightweight markup (Markdown, reStructuredText, AsciiDoc, Textile, Emacs Org-Mode, …), HTML, Ebooks, TeX, word processing (docx, rtf, odt), wiki markup, slide show (LaTeX Beamer, reveal.js, Microsoft PowerPoint, …), and even PDF (via pdflatex, lualatex, xelatex, latexmk and others).

Sphinx

Sphinx is an open-source documentation engine that is based on Docutils; extending it to multi-page documentation. Docutils is an open-source text processing system that uses the plain text easy-to-read reStructuredText to create documentation in multiple formats, such as HTML, LaTeX, Linux man pages, OpenDocument, or XML. Sphinx supports reStructuredText and MyST markdown as input files and can generate multiple output formats including HTML, LaTeX, PDF, ePub, and Texinfo (the official documentation format of the GNU project).

Jupyter Book

Jupyter Book is one of the main projects of the Executable Books Project, together with the other project MyST Markdown. The Executable Books Project is an international collaboration to build open-source tools for publishing computational documents based on the Jupyter ecosystem. `Jupyter Book` can read markdown, MyST Markdown, `Jupyter notebooks` and reStructuredText. It is based on the Sphinx documentation engine being able to produce html websites, pdf, and computational narratives. It supports multiple programming languages in the Jupyter notebooks provided that a Jupyter kernel exist (e.g. Python, Julia, Rubi, Haskell, and many other languages).

Websites that include computational narratives can also benefit from live environments thanks to the integration of Binder, Thebe and Google Colab. It supports multiple types of narrative content like highlighted notes, code cells, quotations, epigraphs, glossaries, index of terms, footnotes, references, grids, cards, dropdown menus, tab content, maths, equations, proofs, theorems, algorithms, and more. The `Jupyter Book` system has been used in multiple ocasions to publish online material, an extensive gallery can be found at https://executablebooks.org/en/latest/gallery/. This roadmap has been created with `Jupyter Book` and serves as an example of some of its functionalities. Another great example of a collaboratively authored `Jupyter Book` is the TAILOR Handbook of Trustworthy AI.

R markdown

R markdown is a flavoured markdown type with special focus on the R programming language and a publishing system. The publishing system uses R markdown files (with extension `.rmd`) or standard markdown and can produce various output formats including HTML, PDFs, Microsoft Word documents, Beamer presentations, HTML5 slides, scientific articles and books (with the the help of the bookdown R package. It also support other programming languages including Python, SQL and others with a language engine. R markdown is also integrated in Rstudio.

Bookdown

Bookdown is an open-source R package that facilitates the creation of books from R Markdown documents. It is an extension for R Markdown to work with long documents. The rest of the functionalities are shared with R Markdown. A list of books written with Bookdown can be found at https://bookdown.org/home/archive/.

Quarto

Quarto is another open-source publishing system with the objective of facilitating the collaboration to create scientific content. Quarto is sponsored by Posit, and follows the development of the R Markdown publishing system extending the focus from the programming language R to Python, R, Julia and Observable. It supports Jupyter notebooks, markdown and their own extension `Quarto` markdown. The conversion to different output formats is done with pandoc, which is able to produce presentations (Reveal.js), dashboards, websites, blogs, books, PDFs, Microsoft Word, ePub and more. Quarto is integrated into multiple authoring environments like Microsoft Visual Studio, Jupyter Lab, Rstudio, and Atlassian Confluence among others.

Reveal.js

Reveal.js is an open-source framework for in-browser presentations. It supports features like animations, export to PDF, an intuitive navigation of the slides, speaker notes, Markdown support, LaTeX support, laser-like pointer, and drawing tools. `reveal.js` presentations can be authored in markdown, HTML or a mix using any text editor and served locally using the Jekyll static website generator. The following HTML code is a fully working `reveal.js` presentation

```html
<html>
  <head>
    <link rel="stylesheet" href="dist/reveal.css" />
    <link rel="stylesheet" href="dist/theme/white.css" />
  </head>
  <body>
    <div class="reveal">
      <div class="slides">
        <section>Slide 1</section>
        <section>Slide 2</section>
      </div>
    </div>
    <script src="dist/reveal.js"></script>
    <script>
      Reveal.initialize();
    </script>
  </body>
</html>
```

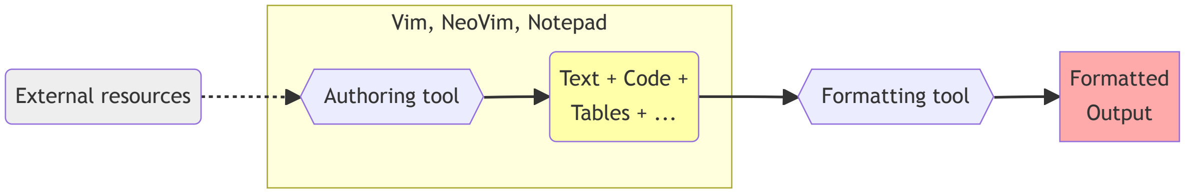
`reveal.js` is directly supported by formatting tools inclusing `Quarto`, `Jupyter Notebook`, and `pandoc`.

D2L-Book: A Toolkit for Hands-on Books

Building on several of these formatting tools, d2l-book is a Python package and a toolkit to build online and printed books, in support of the Dive into Deep Learning book [ZLLS23]. The authoring formats are mainly MyST markdown and Jupyter Notebooks. Formatting is done with pandoc and Sphinx.

Authoring tools

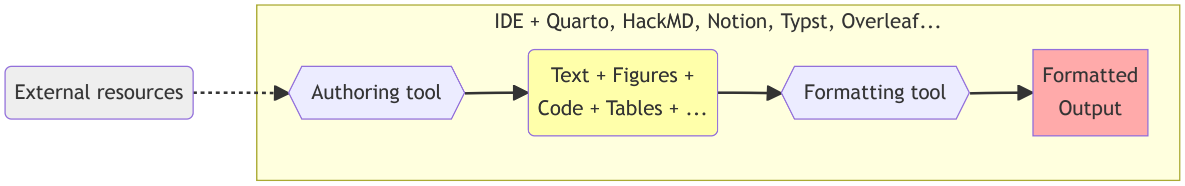
Authoring tools facilitate the creation of input artefacts (e.g. plain text, markup language, tables, figures, code, or equations) which will be compiled and rendered by a *formatting tool* to generate publishable outputs. For example, a markup language editor to create LaTeX is an authoring tool, while `pdflatex`, `XeLaTeX`, or `LuaTeX` are formatting tools that compile input artefacts to pdf. These tools need to integrate metadata about the format in which the artefacts need to be formatted in different output types. For example, the font of the text and its position, the position of figures, tables and other elements. In this roadmap we focus on authoring tools that can be used for the purpose of multi-output publishing systems. We provide some guidelines on the type of files that need to be considered during the authoring process, which tools can directly help on the generation of those artefacts. We consider the collaboration of teams in the authoring process as a desirable feature.



Authoring tools that do not incorporate part of the formatting are rare, as it is common to provide at least a preview of the formatted output. Examples of authoring that can be separated from the formatting step are simple text editors like vim, NeoVim, Notepad, and general-purpose Integrated Development Environments (IDEs).

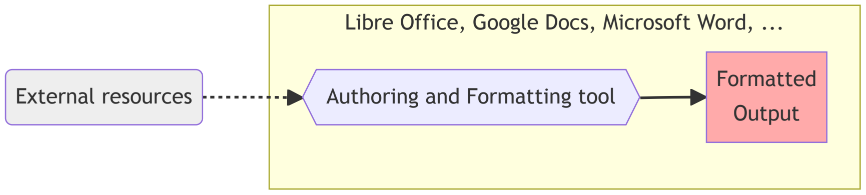
Authoring with formatting

There are **authoring tools** that incorporate a **formatting tool**. While in some situations it is a good practice to separate the two aspects, some computer programs integrate them in one tool that may (or may not) provide access to the intermediate artefacts. For example, Overleaf provides an online collaborative authoring platform that integrates with *pdfLatex* in the back-end.


IDE + Quarto, HackMD, Notion, Typst, Overleaf...
External resources
Authoring tool
Text + Figures + Code + Tables + ...
Formatting tool
Formatted Output

WYSIWYG (What You See Is What You Get)

WYSIWYG editors provide an interface to write rich text documents like Microsoft Word, Libre office, or Google Docs. Such files are difficult to convert automatically to other formats as they do not separate the source text from the presentation layer. Benefits include the possibility of collaborative editing, which comes at the expense of reduced control over formatting.


Libre Office, Google Docs, Microsoft Word, ...
External resources
Authoring and Formatting tool
Formatted Output

Markup language editors

These editors separate the source text from the publishable output. Unlike WYSIWYG editors they typically have a dual-pane user interface with one pane displaying the editable source and the other pane providing a preview. Overleaf is an online collaborative authoring tool that allows the edition of LaTeX files, managing the errors and generating PDF files with a LaTeX compiler. However the free tier has some collaboration limitations. HackMD, Notion and Typst are other online collaborative text editing tools that can be used for authoring publishable material and work with markdown files.

Integrated Development Environments

The separation of the source code and the publishable outputs is something that all Integrated Development Environments (IDEs) provide. These are tools for writing computer programs that commonly require a compilation phase which is usually integrated in the same tool. The idea of authoring tools that can create generic input artefacts that are later combined by a formatting tool is very similar to the common process followed in programming compiled programming languages. This has facilitate the adoption of IDEs as authoring tools. Microsoft Visual Studio and Posit Workbench (formerly RStudio) have tools to work with the `Quarto` environment. Both of them provide options for collaborative and contemporaneous editing.

Computational notebook editors

`Jupyter Notebooks` are documents that offer *computational narratives* by interlacing formatted markdown text and executable code cells, providing the output as additional content. The idea has been adopted by other platforms that have created similar notebook environments. Most of these platforms can convert the resulting computational narratives into various formats like html, pdf, or text documents. Jupyter Notebook and JupyterLab offer the official environment for the original `Jupyter Notebooks`. All of the following projects offer cloud-based platforms to edit `Jupyter Notebooks` or similar. Google Colab is hosted by Google and free for low-intensive programs, Kaggle Notebooks is hosted by Kaggle which focuses on Python for Data Science, and Machine Learning, offering a free solution to participate into Kaggle competitions. Microsoft Azure Notebooks is hosted in Azure and focused on Data Science and Machine Learning with CPUs and GPUs available. It supports various programming languages Python, R, F# and Julia, among others. Other platforms that are similar to support functionalities similar to `Jupyter Notebook` are CoCalc, JetBrains Datalore, Deepnote, Hex, Pluto.jl, Nextjournal, and Starboard. In particular JetBrains Datalore integrates with the JetBrains ecosystem tools such as the IDEs PyCharm and IntelliJ.

From static and dynamic to interactive content

In this section we provide a range of examples of the kinds of content that can be authored with publishing systems such as `Jupyter Book` and `Quarto`. We consider three different types of content:

- Static content is rendered during the creation of the website/book and is kept static;
- Dynamic content includes computational code cells that are rendered during the creation of the website/book but can be modified and re-run on demand; and
- Interactive content provides interactive elements such as buttons, sliders, interactive figures and more. Many more examples can be found in the documentation of Quarto and Jupyter Book.

Computational infrastructure

An important consideration when designing online teaching material is the location where the material will be hosted and served. The host capabilities may affect the type of content that can be included. For example, the content of this roadmap is being hosted as a static webpage thanks to the free service offered by GitHub pages. However, most free hosting services do not provide computational capabilities for dynamic or interactive functionalities. For instance, the Dynamic content material is visualised as a static web page by Github pages but the same content can be run by a third party server for its dynamic functionalities. Services such as *MyBinder* and *Google Colab*, offer free tier versions but they may be slow or fail to run for various reasons (e.g. connection errors or high server workload). Parts of the Interactive content of the roadmap may also require a third party service. Additional information about these requirements is specified when showcasing each example.

Static content

Prior to the World Wide Web, **static content** was the only type of content that could be published. It is primarily content that can be printed without loss of information, the most common forms of which are text, tables and figures. More broadly, static content is authored once after which it is formatted and rendered the same every time. We therefore also include basic video and audio in this category. In this section we provide several examples on how to produce static content in `Jupyter Book` and `Quarto`.

Both formatting systems are capable of reading `Markdown` and `Jupyter Notebooks` and convert them to a variety of output formats. They support particular versions of `Markdown` as there is no Markdown standard yet. `Quarto` has its own `Quarto markdown` with extension `.qmd`, while `Jupyter Book` uses `MyST Markdown`.

Markdown

`Markdown` is a markup language to create formatted text from plain text that can be easily read by humans. Markdown was initiated in 2004 by John Gruber and Aaron Swartz to convert plain text to html [KW17]. However, there is still no consensus on concrete specification because of unsolved issues[1], which has created diverging versions of markdown. In 2014 an unambiguous specification was released by Markdown contributors under the name of CommonMark.

Roles and Directives

MyST Markdown provides roles and directives in order to extend the basic functionalities of Markdown. By defining a set of terms and how to interpret the code on it via extensions, which may be already integrated in Jupyter Book.

Roles are used in-line and have the form ``{rol-name}`role content` `` for example the role ``{math}`E=mc^2` `` is rendered as $E = mc^2$.

Similarly, directives are multi-line versions of the form

~~~
```{directive-name} arguments
:key1: val1
:key2: val2

Content of
the directive
```
~~~

Examples of directives can be found in Callout Blocks.

Callout Blocks

Callout Blocks are special boxes with a colored title and a main textual content. These can be created with different directives with `{note}` being a common example.

~~~
```{note}

Here is a note.
```
~~~

Note

Here is a note.

The content of a callout block can be hidden by adding the optional tag `:class: dropdown`

~~~
```{note}
:class: dropdown

Hidden text.
```
~~~

Note

Hidden text.

Other types of notes can be created with the directives `attention`, `caution`, `danger`, `error`, `important`, `warning`, `tip`, `seealso`, and more that can be found at https://mystmd.org/guide/directives. The following are three examples.

Tip

Tip note.

Attention

Attention note.

Error

Error note.

It is possible to personalise your own notes with `{admonition}`

This is a warning block

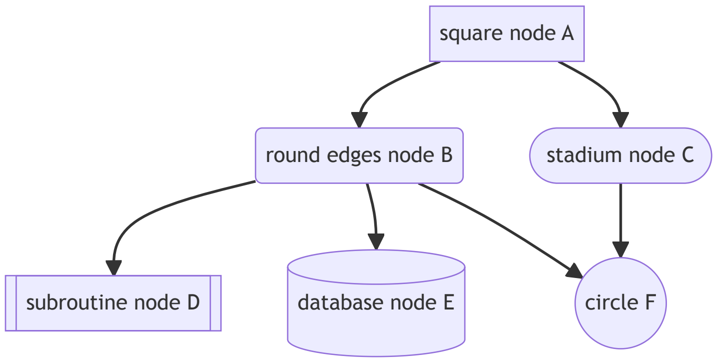
with personalised title and body text.

Diagrams

With additional directives it is possible to create diagrams from plain text. There are multiple sphinx plugins that can be installed in `Jupyter Book` and some that are already integrated in `Quarto`. In this section we show some examples. The Mermaid diagramming and charting tool allows the creation of a multitude of diagrams including flowcharts, sequences, mindmaps and more. There is an online live editor that allows the exploration of various examples, modification and creation of new diagrams in the following link https://mermaid.live/edit. WaveDrom is another rendering engine to draw timing diagrams. Some examples can be found below.

Mermaid flowchart

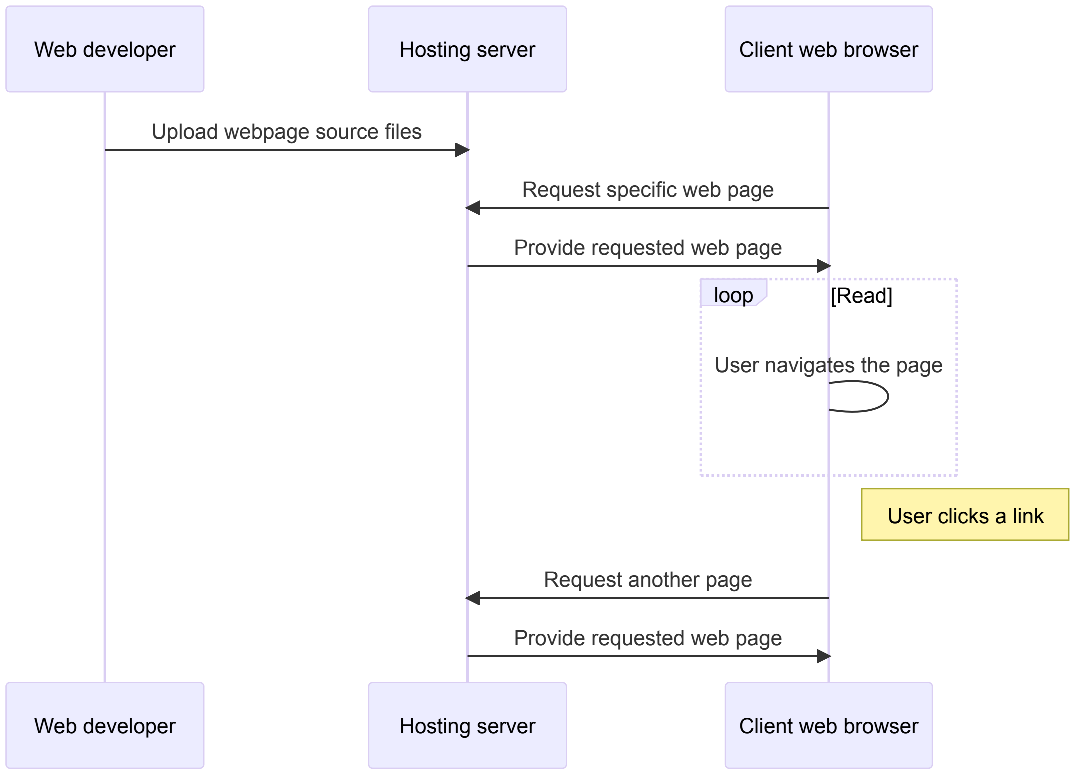
~~~
```{mermaid}
flowchart TD
  A[square node A] --> B(round edges node B)
  A --> C([stadium node C])
  B --> D[[subroutine node D]]
  B --> E[(database node E)]
  B --> F((circle F))
  C --> F
```
~~~



Mermaid sequence diagram

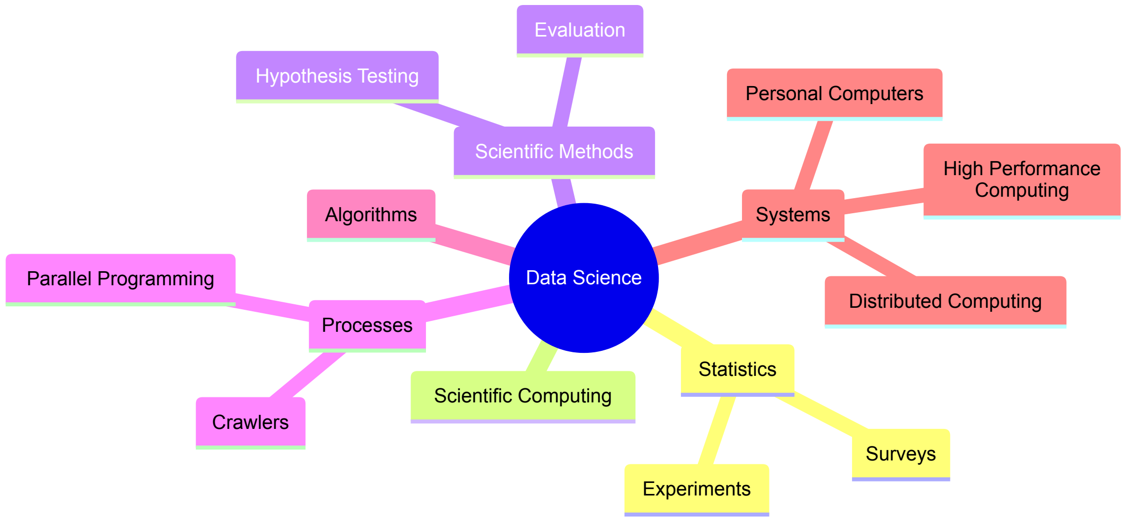
Example of a sequence diagram with Mermaid showing the interactions between a web developer, the hosting and a client web browser.

~~~
```{mermaid}
sequenceDiagram
        participant Web developer
        participant Hosting server
  participant Client web browser
        Web developer->>Hosting server: Upload webpage source files
  Client web browser->>Hosting server: Request specific web page
  Hosting server->>Client web browser: Provide requested web page
        loop Read
                        Client web browser->Client web browser: User navigates the page
        end
  Note right of Client web browser: User clicks a link
  Client web browser->>Hosting server: Request another page
  Hosting server->>Client web browser: Provide requested web page
```
~~~



Mermaid mindmap

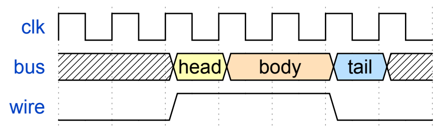
~~~
```{mermaid}
mindmap
  root((Data Science))
    Statistics
      Surveys
      Experiments
    Scientific Computing
    Scientific Methods
      Hypothesis Testing
      Evaluation
    Processes
      Parallel Programming
      Crawlers
    Algorithms
    Systems
      High Performance Computing
      Personal Computers
      Distributed Computing
```
~~~



WaveDrom timing diagrams

[WaveDrom](#) can draw timing diagrams.

~~~
```{wavedrom}
{ signal : [
  { name: "clk",  wave: "p......" },
  { name: "bus",  wave: "x.34.5x",   data: "head body tail" },
  { name: "wire", wave: "0.1..0." },
]}
```
~~~



Cross-references

Published documents often have internal references to other content in the document. By default, all the titles have their own anchor points (move the mouse cursor on top of a title to see a `#` symbol indicating a clickable anchor point). It is possible to reference sections within the same page by creating a link writing the full title in lower case and dashes instead of spaces like `[](cross-references)` which creates a link like **`cross-references`**. With this method it is not possible to reference other pages; e.g. the link **`conclusions`** should not work but [Static content](#) does work. A more flexible method is to manually indicate anchor points in titles, figures, tables and other content. By adding a label before a title as follows

```
(my-label)=
# Section title
```

the section can be references with `[](my-label)` or the more flexible role `{ref}`my-label``. For example the code `[](sec:conclusion)` and `{ref}`sec:conclusion`` will generate a link to the Concluding remarks.

Code	Result
`[](sec:conclusion)`	Concluding remarks
`[Textual description](sec:conclusion)`	Textual description
`{ref}`sec:conclusion``	Concluding remarks

Labels can also be added to figures, tables and equations within their own directive tags. For example, in the `{figure}` directives the tag `name` specifies the label.

~~~
```{figure} images/example.svg
:name: fig:ex:1

Caption of the example figure ex1
```
~~~

The previous code generates the next figure that can be referenced in multiple ways.

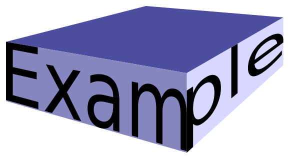


Fig. 1 Caption of the example figure number 1

Code	Result
`[](fig:ex:1)`	Caption of the example figure number 1
`[Textual description](fig:ex:1)`	Textual description
`{ref}`fig:ex:1``	Caption of the example figure number 1
`{numref}`fig:ex:1``	Fig. 1
`{numref}`Figure %s and more text <fig:ex:1>``	Figure 1 and more text

Tables use the tag `name`

~~~
```{table} Caption of the table ex1
:name: tab:ex:1

| header 1 | header 2 |
| -------- | -------- |
|    a     |    b     |
```
~~~

Table 1 Caption of the table ex1

header 1	header 2
a	b

Code	Result
`[](tab:ex:1)`	Caption of the table ex1
`[Textual description](tab:ex:1)`	Textual description
``{ref}`tab:ex:1` ``	Caption of the table ex1
``{numref}`tab:ex:1` ``	Table 1
``{numref}`Figure %s and more text <tab:ex:1>` ``	Figure 1 and more text

Finally, equations use the tag `label`

~~~
```{math}
:label: eq:ex:1

E = mc^2
```
~~~

$$E = mc^2 \tag{1}$$

And can only be referenced by their number

Code	Result
`[](eq:ex:1)`	(1)
``{eq}`eq:ex:1` ``	eq:ex:1

`Quarto` has its own way of making cross references which can be consulted in their documentation (Quarto cross references).

Code segments

Another common aspect of documents in STEM (Science, technology, engineering, and mathematics) is the publication of short extracts of pseudocode or source code. `Markdown` languages usually support the highlight of code based on the different programming language specifications.

For example the following code is for Python

```
print("Hello world!")
```

while the following code is written in `ANSI C`

```
#include <stdio.h>

int main() {
  printf("Hello world!");
  return 0;
}
```

Inline-tabs

Some interactive books that contain code may be aimed at diverse audiences with different background programming knowledge. For those cases, `MyST Markdown` can create tabs to select which content to display. The following is an example of `Python` and `C++` code:

Python C++

```
def main():
    return
```

Tab choices persist across code segments:

Python C++

```
print("Hello World!")
```

Citations and bibliography

It is possible to add citations and a bibliography using a `bibtex` file. For example, the references in this webpage are all stored in the `../references.bib` file. The bibliography can be inserted in any particular section with the directive `{bibliography}`. The current References section contains the following markdown content:

~~~
(sec:references)=
# References

```{bibliography}
```
~~~

Once a bibliography has been added in the document, it is possible to cite any of the references with the `{cite}` role. The following role ``{cite}`sokol21` `` will generate the citation [SF21], which provides information about the reference when hovering the mouse over it. The list of references in the bibliography will only contain those reference that are cited at least once in the whole website.

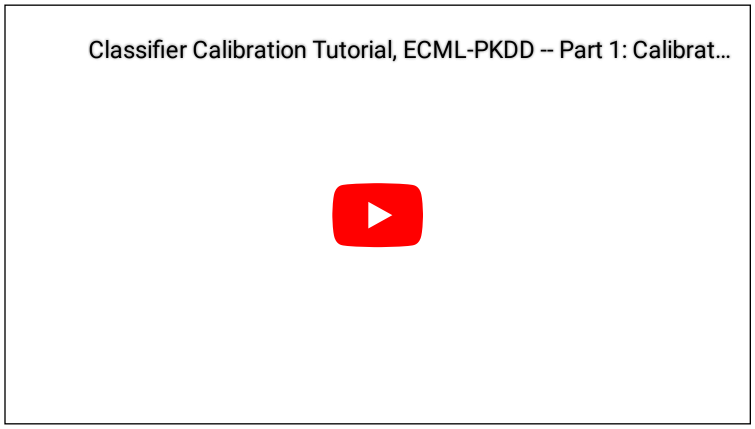
Videos

The wide spread of personal computers, smart phones and tables has accelerated the content of audiovisuals in the internet and streaming services. Furthermore, It is common that in academic and teaching contexts the same lessons are repeated multiple times to different audiences. This makes recorded audiovisual content well suited to reduce the limited time commitment of the teachers, while maximising the reach of the content. There are multiple online services to host video material with YouTube, Vimeo, and SlidesLive being the most well known.

Most video hosting services offer ways to embed their videos into any `html` website by using the `iframe` tag. The following is an example of a video hosted in `YouTube`

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/4kwEMHZJx5A" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write;
encrypted-media; gyroscope; picture-in-picture; web-share"
allowfullscreen></iframe>
```

which is rendered like this


Classifier Calibration Tutorial, ECML-PKDD -- Part 1: Calibrat...

[1] https://talk.commonmark.org/t/issues-we-must-resolve-before-1-0-release-6-remaining/1287

Dynamic content

Dynamic content is content that includes computational code that is executed when the website/book is built. With extra configurations, the content can be re-run in real time locally or with a third party service like MyBiner, Google Colab, or others. This type of content is very useful for educational material in science, technology, engineering and mathematics. Jupyter Notebooks and other programming environments that integrate a textual narrative, computational cells and their results are really common. Publishing systems like `Jupyter Book` and `Quarto` support multiple ways to write this type of material by using directives (See Roles and Directives) for code in markdown files, or directly using Jupyter Notebooks. This section serves as an example of the functionalities that can be integrated in a Markdown file in a Jupyter Book. We also indicate what are the differences in the `Quarto` publishing system when necessary.

Configuration

Adding dynamic content that changes during the execution requires the specification of a **kernel** which is able to read the code indicated in the apropiate directives and produce an ouptut result. This configuration differs slighly depending on the platform and the type of file (e.g. markdown and Jupyter Notebook).

Jupyter Notebook files

`Jupyter Notebooks` are web-based documents that combine a textual narrative (written in a markup language) with computational elements (supporting a multitude of programming languages). The code of the notebooks can be executed with the help of a kernel; a *programming language specific* process that can interpret the code, run it and provide the results to the authoring application. The default kernel is the ipykernel built on top of IPython. Common functionalities added via the code cells are the generation of figures, tables, plots, and interactive elements and the analysis of data. Jupyter Notebooks have a file extension `.ipynb` and can be edited with authoring tools like `Jupyter Notebook`, `Jupyter Lab`, *Google Colab*, and most *integrated development environments* (IDEs) like *PyCharm*, *Microsoft Visual Studio*, *Posit*, and *RStudio* (See also Computational notebook editors).

In `Jupyter Book` and `Quarto` projects the configuration of `Jupyter Notebooks` is general across the project. However, `Quarto` requires a *Raw* cell at the beginning of the notebook with the `title`, `author`, and any additional options that you want to include in order to be rendered. The following is a configuration example for `Quarto`:

```
---
title: Title of the page
authors: "Miquel Perello Nieto"
date: "July 11th, 2024"
format:
  html: default
  pdf: default
  refealjs: default
---
```

Markdown files

In `Jupyter Book`, **Markdown** files with dynamic content require a YAML configuration in the header indicating some parameters about the type of file and the kernel to use. There are kernels for different programming languages like R, Python, Julia, and more. For example, the markdown file for this page has been configured to run Python3 with the following `yaml` configuration

```
---
jupytext:
  cell_metadata_filter: -all
  formats: md:myst
  text_representation:
    extension: .md
    format_name: myst
    format_version: 0.13
    jupytext_version: 1.11.5
kernelspec:
  display_name: Python 3
  language: python
  name: python3
---
```

The header can be generated automatically with the help of the `jupyter-book` tool by running the following command in a terminal at the root of the project

```
jupyter-book myst init markdownfile.md --kernel kernelname
```

Additional documentation can be found at https://jupyterbook.org/en/stable/file-types/myst-notebooks.html.

Running Python code in the `Quarto` publishing system requires the `jupyter` Python package and the specification of the `python` command to use in the `YAML` header or in the `_quarto.yml` configuration file.

```
jupyter: python3
```

Live code

In `Jupyter Book`, Thebe offers a solution to launch the kernel in the current page (without the need to jump to a third party website). The web page you are currently reading has been configured to run Thebe with the third party service *MyBinder*. This was achieved by adding the following line of code before the first title of the markdown document:

```
(launch:thebe)=
```

The top of the page now displays the *spaceship* icon. You can launch *Thebe* by clicking on it and select `Live code` in the drop down menu:

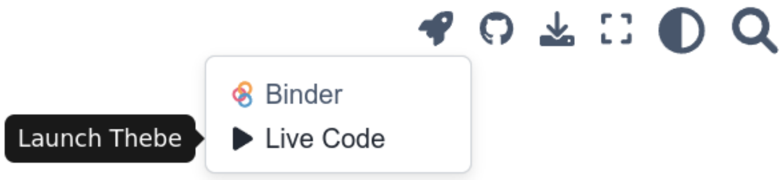


Fig. 2 Menu to launch Live Code with Thebe

This will launch the code in your configured server (in this case *MyBinder*) and a new loading text will be shown at the top of the current page with several steps, building the code, publishing and launching. Depending on the configured server this process may take some time to complete, and occasionally could fail to launch.

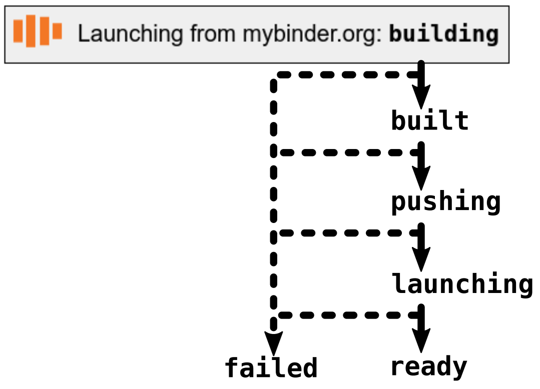


Fig. 3 Steps that Live Code will show as it prepares the running environment.

Once the kernel is ready, any Python code that is written in a `{code-cell}` with `ipython3` can be modified and re-run in real time. The current page is an example in which you can modify all the cells.

`Quarto` does not currently support *Thebe*, but it supports launching the code in a third party environment like *MyBinder* by adding the following `yaml` configuration in the header of the Markdown file

```yaml
code-links: binder
```

`Quarto` has good integration with Shiny which has additional functionalities that can bee seen in Section Interactive content.

Simple examples

In Jupyter Books by using the `code-cell` directive it is possible to execute code and print out its response.

~~~
```{code-cell}
print("Hello world!")
```
~~~

When your book is being built, the contents of any `{code-cell}` block will be executed with the default Jupyter kernel. The outputs will be displayed in-line with the rest of your content.

```python
print("2 + 2 = ", 2 + 2)
```

```
2 + 2 =  4
```

It is possible to modify the behaviour of `code-cells` by adding `tags`. For example the tag `hide-input` will make the code hidden until it is clicked with the mouse.

~~~
```{code-cell} ipython3
:tags: [hide-input, thebe-init]

# Python code
```
~~~

The following code is an example:

▶ Show code cell source

The previous example generates a random number between 1 and 6 and stores the result in a variable. The content of the variable can be printed in a separate cell

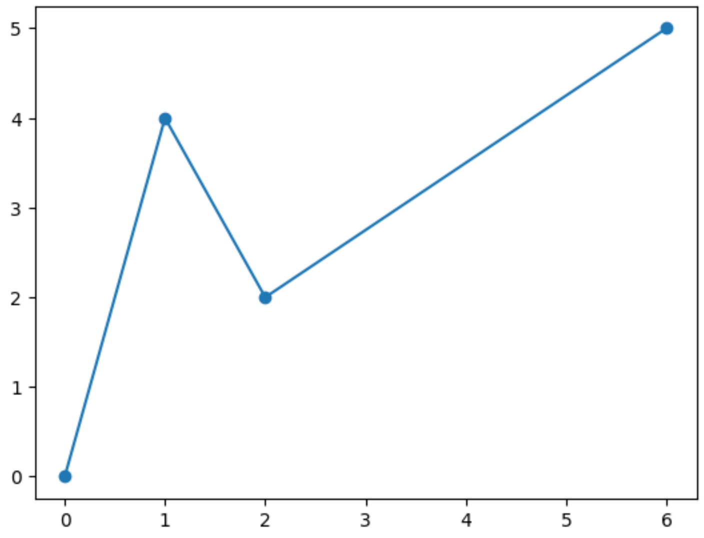
```
print(hidden_text)
```

```
The result of this draw of a six sided dice is 4
```

The following code illustrates how to create a lineplot which can be easily modified in a Live environment.

```
import matplotlib.pyplot as plt

plt.plot([0, 1, 2, 6], [0, 4, 2, 5], 'o-')
```

```
[<matplotlib.lines.Line2D at 0x7f8bc329b610>]
```



Exploration of tables

In data analysis it is common to inspect large amounts of data which in certain cases is stored in a tabular form. `Pandas` is a Python library that is able to produce `Data Frames` (similar to the `Data Frames` from the `R` programming language), and can produce summaries and visualisations in various output formats. The following example shows the recorded temperatures (Celsius) in Bristol as shown in its Wikipedia article[1].

```
import pandas

features = ['Record low', 'Record high']
date = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
        'Nov', 'Dec']
data = [[-14.4, 14.2], [-9.7, 18.3], [-8.3, 21.7], [-4.7, 25.7], [-2.0, 27.4],
        [0.6, 32.5], [4.7, 34.5], [3.9, 33.3], [0.6, 28.3], [-3.2, 26.8],
        [-6.5, 17.5], [-11.9, 15.8]]
df = pandas.DataFrame(data=data, index=date, columns=features)

df.style.background_gradient(cmap='coolwarm', vmin=df.min().min(), vmax=df.max().max())
```

	Record low	Record high
Jan	-14.400000	14.200000
Feb	-9.700000	18.300000
Mar	-8.300000	21.700000
Apr	-4.700000	25.700000
May	-2.000000	27.400000
Jun	0.600000	32.500000
Jul	4.700000	34.500000
Aug	3.900000	33.300000
Sep	0.600000	28.300000
Oct	-3.200000	26.800000
Nov	-6.500000	17.500000
Dec	-11.900000	15.800000

The cells are not independent, and following computations can be performed making reference to variables instantiated in previous cells. The following example shows some statistics of the previous table.

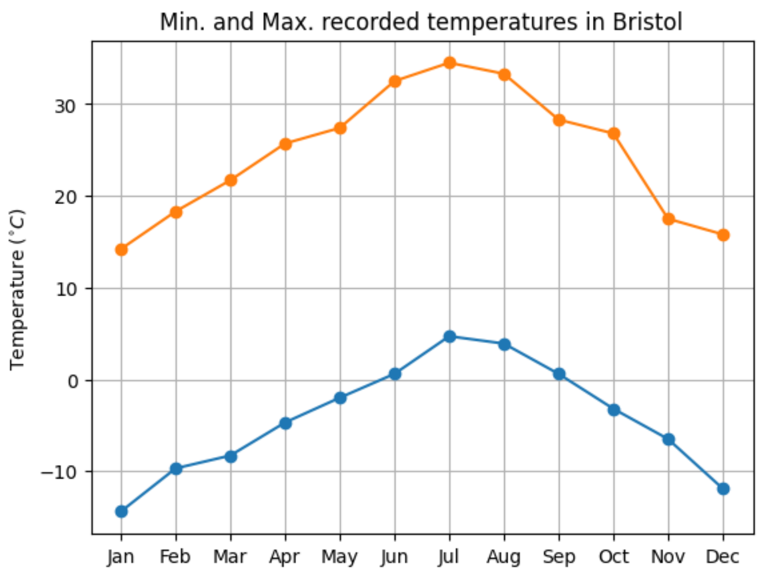
```
df.describe()
```

	Record low	Record high
count	12.000000	12.000000
mean	-4.241667	24.666667
std	6.124831	7.056568
min	-14.400000	14.200000
25%	-8.650000	18.100000
50%	-3.950000	26.250000
75%	0.600000	29.350000
max	4.700000	34.500000

Static plots

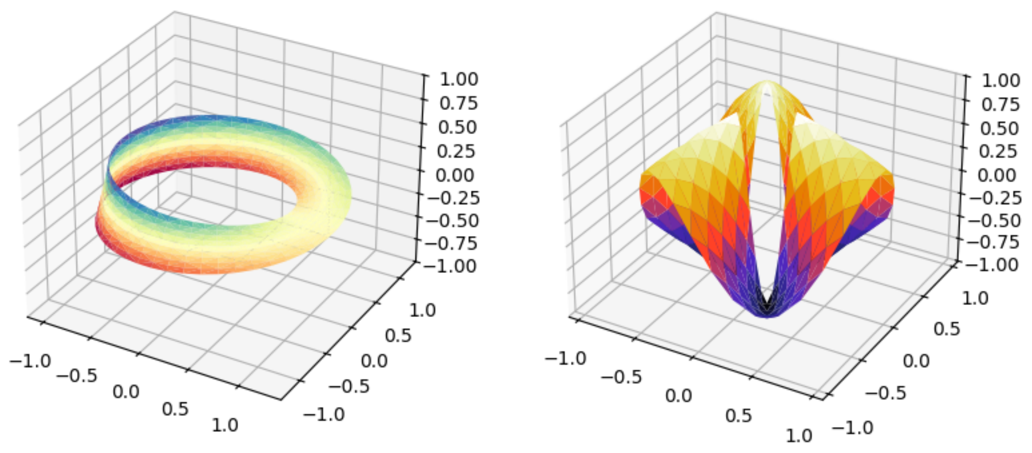
Figures generated with plotting libraries can also be rendered and shown as static images (e.g. matplotlib, pyplot, bokeh). The following is an example with the previous temperatures rendered with `Matplotlib``.

```
import matplotlib.pyplot as plt
plt.plot(df, '-o')
plt.ylabel('Temperature ($^{\circ}C$)')
plt.title('Min. and Max. recorded temperatures in Bristol')
plt.grid()
```



Or the following examples of 3D surfaces from the Matplotlib documentation.

```
from bpython.examples import matplotlib_trisurf3d_2

matplotlib_trisurf3d_2()
```



Code listing

It is possible to print the source code of any Python function with the package `inspect`. The following is an example for a function in the `bpython` package.

```
def function_example(a: float, b: float):
    """Sums two numbers

    Parameters
    ----------
    a : float
      First number to sum
    b : float
      Second number to sum
    """
    return a + b
```

Then it is possible to get the function documentation

```
import inspect

documentation = inspect.getdoc(function_example)
print(documentation)
```

```
Sums two numbers

Parameters
----------
a : float
  First number to sum
b : float
  Second number to sum
```

or its source code

```
source = inspect.getsource(function_example)
print(source)
```

```
def function_example(a: float, b: float):
    """Sums two numbers

    Parameters
    ----------
    a : float
      First number to sum
    b : float
      Second number to sum
    """
    return a + b
```

Reuse of complex code

In order to reuse complex code across the website, it is recommended to write a package with the functions. In the case of the current roadmap, we have created a package `bpython` in the folder /lib/book-python/. It can be installed after the requirements with

```
pip install -e /lib/book-python/
```

The `-e` option keeps the library in its current folder, allowing the modification of the library during the development of the website. In this way, every time that the virtual environment is loaded the library is loaded anew. The following code should print the current version of this auxiliary library.

```
import bpython

print(f"The current BPython version is {bpython.__version__}")
```

```
The current BPython version is 0.0.1.dev1
```

Another example with a complex 3D surface visualisation from the Matplotlib documentation was already shown in Section Static plots.

[1] https://en.wikipedia.org/wiki/Bristol

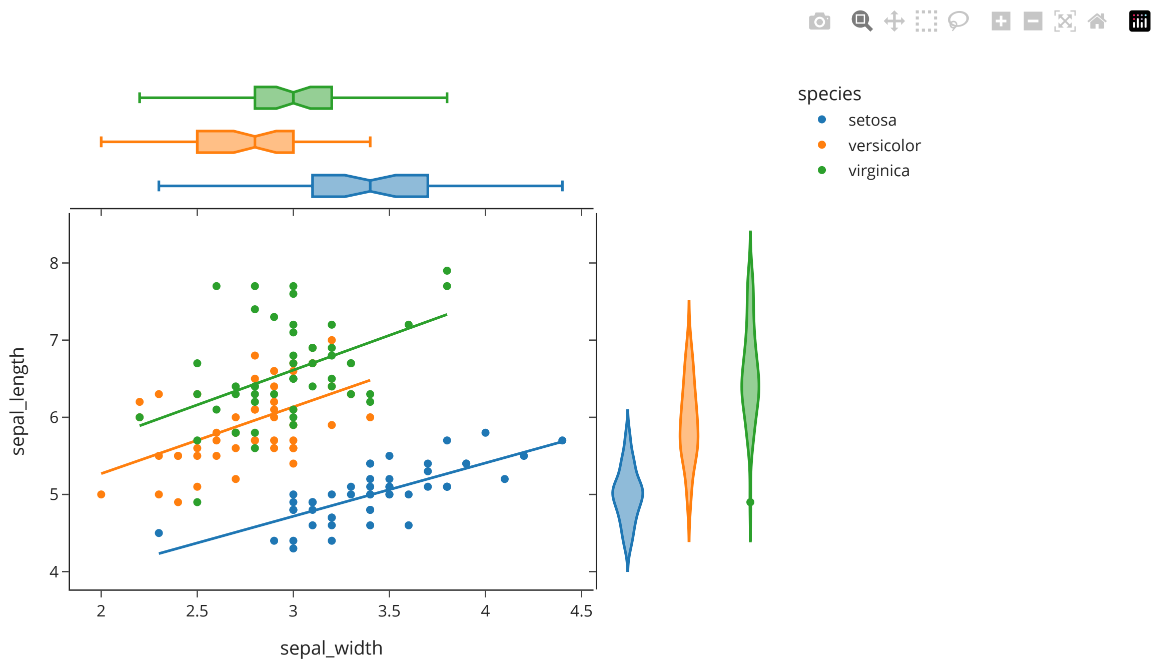
Interactive content

Programming examples shown in the previous sections are executed during the compilation of the publication and their result is embedded into the resulting publication file. We have also seen that with *Thebe* it is possible to have live code that can be modified and re-run to generate different results. However, it requires a third party server (e.g. Google Colab, MyBinder, SageMaker Studio Lab, or others). On the other hand, there are methods to run code in the same web browser used for viewing the formatted content, without the need of an external computation resource. In this section we demonstrate how this can be achieved with Shinylive and other libraries.

Plots with Plotly

Some libraries can produce figures that can be interacted with. For example Plotly provides tools like zooming, selection, hover information, filtering, and more. The following example shows the sepal length and width of flowers from the Iris dataset. Notice that it is possible to remove some classes from the visualisation by clicking on the species in the legend, it is possible to get additional information on the summary statistics or the individual samples by hovering with the mouse, it is possible to select a rectangular subregion to zoom in, download the plot as a `png`, and more.

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species",
                 marginal_y="violin", marginal_x="box", trendline="ols",
                 template="simple_white")
fig.show()
```



Maps with IpyLeaflet

The Python library ipyleaflet can display maps using the JavaScript library Leaflet which are interactive and mobile-friendly.

```
from ipyleaflet import Map, Marker, basemaps, basemap_to_tiles
m = Map(
  basemap=basemap_to_tiles(
    basemaps.NASAGIBS.ModisTerraTrueColorCR, "2017-04-08"
  ),
  center=(51.4545, -2.5879),
  zoom=4
)
m.add_layer(Marker(location=(51.4545, -2.5879), draggable=False))
display(m)
```

Jupyter Widgets

Jupyter Widgets provide a set of interfaces to interact with Python code. Some of the interfaces are buttons, sliders, In a *Live environment* this interactions result in the code being executed.

```python
import ipywidgets as widgets

a = widgets.FloatText()
b = widgets.FloatSlider()
display(a,b)

mylink = widgets.jslink((a, 'value'), (b, 'value'))
```

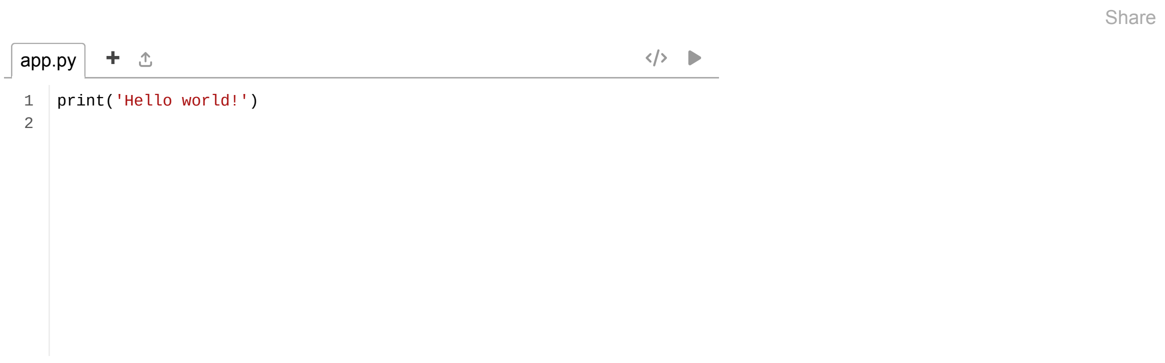
The results of the interactions with the widgets can be used in the following code cells.

```python
display(a)
```

Shinylive: Shiny + WebAssembly

Shinylive is a technology that unifies Shiny and WebAssembly. WebAssebmly is a binary format for compiled programs that can run in the web browser at near-native speeds. Pyodide is a port of Python and various packages compiled in WebAssembly.

It is possible to edit Shiny applications in the online editor https://shinylive.io/py/editor/. Once the application has been finished it is possible to copy a link to the end result. By default, Shiny provides an interface with a code editor on the top left, an output console at the bottom left and a user interface on the right to output the application result. The following example uses the output console and not the user interface.



```python
print('Hello world!')
```

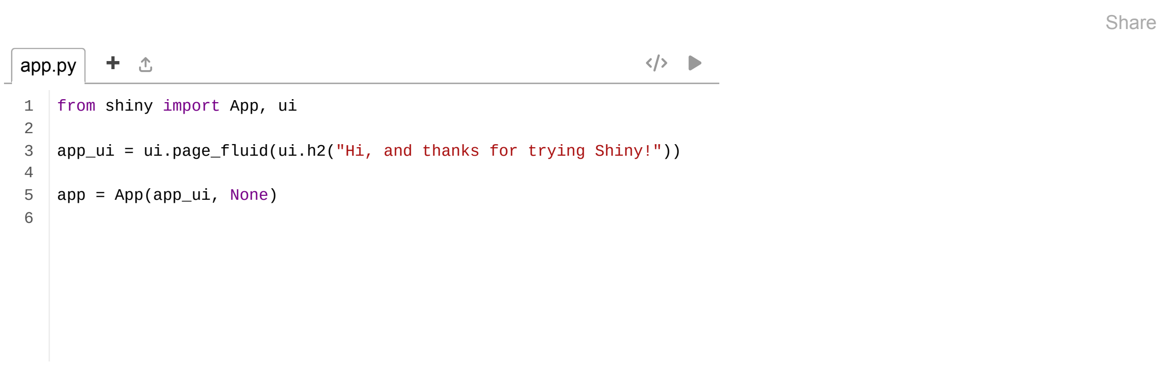
It is possible to modify the code and obtain the result on the output console. For example, you may want to try and change `print('Hello world!')` by `print(2+2)` and click `Re-run app` (or press `(Ctrl)-Shift-Enter`).

Shinylive in Jupyter Book

Jupyter Books (like this roadmap) need to edit the application in the online editor and embed the final application using the resulting url and an `<iframe>`. For example the following iframe

```
<iframe src="https://shinylive.io/py/editor/#code=NobwRAdghgtgpmAXGKAHVA6VBPMAaMAYwHsIAXOcpMAMwCdiYACAZwA
data-external="1" width="100%" height="400px">
</iframe>
```

which is rendered as follows



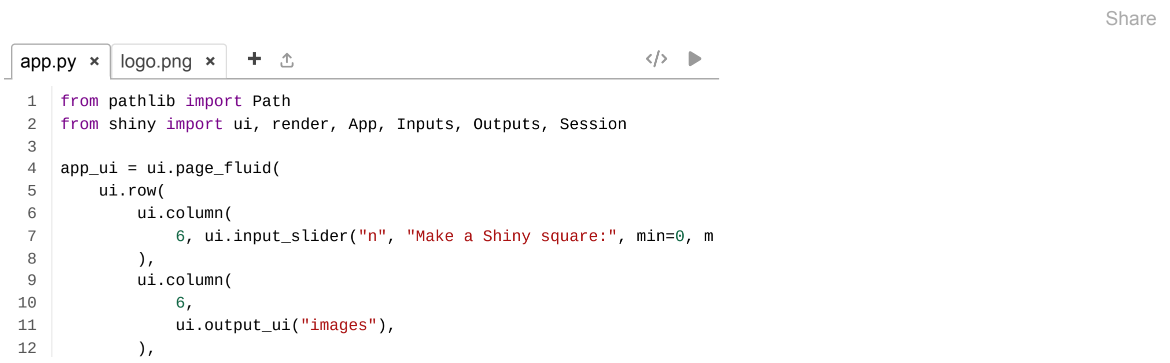
```python
from shiny import App, ui

app_ui = ui.page_fluid(ui.h2("Hi, and thanks for trying Shiny!"))

app = App(app_ui, None)

```

All the code for the previous example is encoded in the URL as a GET method. It is possible to modify the code in the editor, and generate the new URL by clicking the Share button on the top right corner. This provides a link to the editor (including editor, console and user interface (ui)) or only the application (ui).

Furthermore, the original source code could be store in a Github Gist and provide the gist id in the url. For example, the following GitHub Gist https://gist.github.com/wch/e62218aa28bf26e785fc6cb99efe8efe with `id=e62218aa28bf26e785fc6cb99efe8efe` can be deployed with

```
<iframe src="https://shinylive.io/py/editor/#gist=e62218aa28bf26e785fc6cb99efe8efe"
data-external="1" width="100%" height="400px">
</iframe>
```

which results in the following application



```python
from pathlib import Path
from shiny import ui, render, App, Inputs, Outputs, Session

app_ui = ui.page_fluid(
    ui.row(
        ui.column(
            6, ui.input_slider("n", "Make a Shiny square:", min=0, m
        ),
        ui.column(
            6,
            ui.output_ui("images"),
        ),
```

There are multiple prebuild common Python packages like Matplotlib, Numpy, Seaborn, Scipy, and scikit-learn (See list of packages for Pyodide 0.25.1 at https://shiny.posit.co/py/docs/shinylive.html#installed-packages). This makes it flexible to create plots with interactive parts like the following histogram of random points.

Share

app.py

```python
import matplotlib.pyplot as plt
import numpy as np
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.input_slider("n", "Number of bins", 0, 100, 20),
    ui.output_plot("plot"),
)


def server(input, output, session):
    @output
```

Or a simple line plot which can be easily modified.

Share

app.py

```python
import matplotlib.pyplot as plt
from shiny import App, render, ui

app_ui = ui.page_fluid(
    ui.output_plot("plot"),
)

def server(input, output, session):
    @output
    @render.plot(alt="A line plot")
    def plot():
        return plt.plot([0, 1, 2, 6], [0, 4, 2, 7], 'o-')
```

Shinylive in Quarto

Shiny and `Quarto` are both developed by Posit (formerly RStudio). Quarto integrates very well with Shinylive being able to embed any Shinylive application in a Markdown file by writing the source code directly in a directive of the type `{shinylive-python}`. There are some options that can be adjusted in the header, and the code goes directly below. The following example in `Quarto` would render as the first example of the previous subsection Shinylive in Jupyter Book.

~~~
```{shinylive-python}
#| standalone: true
#| components: [editor, viewer]
#| viewerHeight: 480

from shiny import App, ui

app_ui = ui.page_fluid(ui.h2("Hi, and thanks for trying Shiny!"))

app = App(app_ui, None)
```
~~~

More complex examples are demonstrated in the following sections.

Some itineraries through the landscape

Now that we have charted the landscape of new ways of authoring and publishing online material, and given a number of examples of “places to visit”, we are ready to describe some “itineraries” through the landscape:
practical use cases that people who want to create or update training material might encounter.

On the one hand, somebody may want to use `Quarto` to develop new training material, which is covered in Create a Quarto course from scratch. But perhaps a more common use case is that somebody already has material in a legacy format which they want to transform into a more interactive and multi-purpose form. We give a concrete example of how to do this in Convert a LaTeX Beamer presentation to Quarto.

Note

There are of many other possible use cases! Please get in touch if you have ideas to add other use cases to this roadmap.

Create a Quarto course from scratch

In this section we describe how we created an online course entirely with `Quarto`. The main idea was to create a website with all the necessary content to give a brief introduction to a topic, with a set of slides created with the same content, a printable version as a pdf, video recordings for each section, all with Python interactive examples.

Overall structure of the course

The idea is to create a self-contained course with multiple publishing options that adapt to the device capabilities. The most comprehensive format to access the course is in its website format which includes textual narrative, tables, figures, equations, interactive code and video recordings. The other format that includes the full narrative, but without interactive code or video, is the printable PDF. Finally, the `reveal.js` slides focus on the key points but include the tables, figures, equations and the interactive code.

In order to generate multiple types of outputs with `Quarto` it is necessary to configure the different formats in the _quarto.yml file (See lines 52-84) and in the quarto markdown file of the specific section (See lines 20-38 of odm.qmd).

The following is a small extract of the configuration in YAML format.

```yaml
format:
  html:
    theme:
      light: [yeti, mooctai.scss]
      dark: [superhero, mooctai_dark.scss]
    code-link: true
    css: style.css
    toc: true
    number-chapters: true
  reveal.js:
    logo: "../images/logos/tailor_and_uob.svg"
    output-file: slides-intro-to-opt-dec-mak.html
    slide-number: true
    incremental: true
    smaller: false
    auto-stretch: false
    chalkboard: true
  pdf:
    include-in-header:
      - file: packages.tex
      - operators.tex
      - colors.tex
      - definitions.tex
```

Course narrative and key points

The course follows a textual narrative explaining the details, but the slides need to concentrate on the main points, leaving out some details. This can be achieved as follows:

```
::: {.content-hidden when-format="reveal.js"}

Text shown in all formats excepts reveal.js slides.

:::
```

In order to keep some content in the slides we decided to create lists with the key points, which can be shown in all formats. We have added a `Key Points` title as a subsection in the other formats (an example can be found at https://tailor-uob.github.io/mooc_trustworthy_ai/cha_odm/odm.html#key-points).

Figures, tables and equations

Figures, tables and equations can be visualised in all formats and are automatically adjusted to fit the available space of the output format.

Figures can be easily added in plain markdown

```
![Rounded rectangle](./images/example.png)
```

which would be rendered as follows:

However, `Quarto` markdown also allows changes to the style of the image, for example the alignment and size

```
![Rounded rectangle](./images/example.png){fig-align="center" width="100px"}
```

or with HTML code as follows.

```
<img src="images/example.png" alt="Rectangle with the text: example of an
image." style="width=100px">
```

Warning

Using `html` syntax to render images in `Jupyter Book` is not recomended, and requires the activation of the `html_image` extension (which is not active in this roadmap).

Tables can be written in markdown

```
|  | A  | B  |
|--|----|----|
|1 | A1 | B1 |
|2 | A2 | B2 |
```

which is rendered as

	A	B
1	A1	B1
2	A2	B2

Finally, equations can be written in `LaTeX` and are interpreted by `MathJax`. Equations can be written inline with `$E=mc^2$` shown as $E = mc^2$, or in display mode:

```
$$
\begin{equation}
  \mathbb{E}_{j \sim P(\cdot|\vec{x})} (c_{i|j}) = \sum_{j = 1}^K P(C_j|\vec{x}) c_{i|j}.
\end{equation}
$$
```

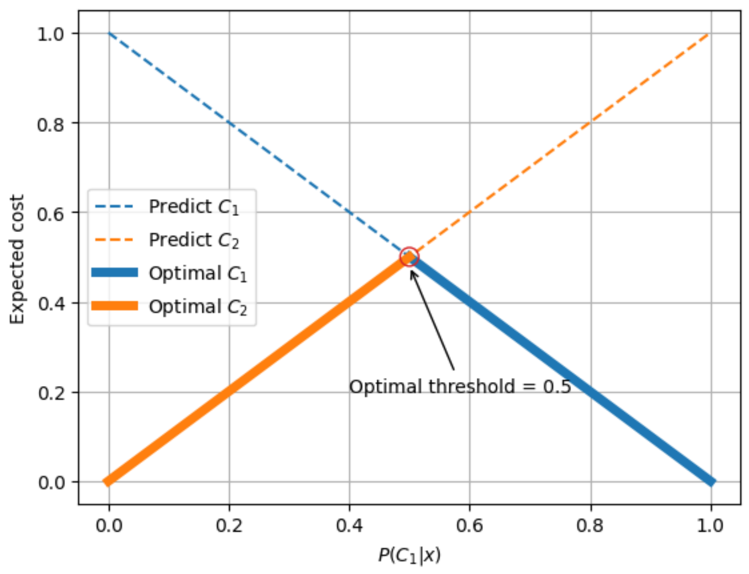
which is rendered as

$$\mathbb{E}_{j \sim P(\cdot|\vec{x})} (c_{i|j}) = \sum_{j = 1}^K P(C_j|\vec{x}) c_{i|j}.$$

Programming examples

This course is designed for a technical audience that may benefit from programming examples that serve both to teach a concept and learn how to code the example. In this use-case we generate figures based on the explained mathematical concepts. Accessing the source code can provide further intuition to better understand the resulting figures. The next example has been extracted from the use-case which shows the source code and the generated figure below.

```python
import matplotlib.pyplot as plt

C = [[0, 1], [1, 0]]
threshold = (C[0][1] - C[1][1])/(C[0][1] - C[1][1] + C[1][0] - C[0][0])
cost_t = threshold*C[0][0] + (1-threshold)*C[0][1]
plt.grid(True)
plt.plot([0, 1], [C[0][1], C[0][0]], '--', label="Predict $C_1$")
plt.plot([0, 1], [C[1][1], C[1][0]], '--', label="Predict $C_2$")
plt.plot([threshold, 1], [cost_t, C[0][0]], lw=5, color='tab:blue', label="Optimal $C_1$")
plt.plot([0, threshold], [C[1][1], cost_t], lw=5, color='tab:orange', label="Optimal $C_2$")
plt.xlabel('$P(C_1|x)$')
plt.ylabel('Expected cost')
plt.legend()
plt.annotate("Optimal threshold = 0.5", (0.5, 0.48), xytext=(0.4, 0.2),
             arrowprops=dict(arrowstyle='->', facecolor='black'))
plt.scatter(0.5, 0.5, s=100, facecolors='none', edgecolors='tab:red', zorder=10)
plt.show()
```



Interactive examples

An important part of the attraction of novel publishing tools is the possibility of creating interactive and dynamic applications online. `Shinylive` is a method that combines `Shiny` and `WebAssembly` to run `Python` code in your own client web browser. `Quarto` has a great integration with this technology, allowing to include code directly in the markdown that is executed in real time when the page is loaded. The following code is an example extracted from the use case. (In this instance we chose not to show the code in the course to focus the learner on the interactive example.)

~~~
```{shinylive-python}
#| standalone: true
#| components: viewer
#| viewerHeight: 480

import matplotlib.pyplot as plt
from shiny import App, render, ui
import pandas as pd

app_ui = ui.page_fluid(
    ui.layout_sidebar(
        ui.panel_sidebar(
    ui.input_slider("TP", "Cost True C1",  value=-5, min=-10, max=0),
    ui.input_slider("TN", "Cost True C2",  value=-1, min=-10, max=0),
    ui.input_slider("FN", "Cost False C2", value=10, min=1,   max=10),
    ui.input_slider("FP", "Cost False C1", value=1,  min=1,   max=10),
    ),
    ui.panel_main(
    ui.output_plot("plot")
    )
    ),
)

def server(input, output, session):
    @output
    @render.plot(alt="A histogram")
    def plot():
        TP = input.TP() # C_1|1
        FN = input.FN() # C_1|2
        FP = input.FP() # C_2|1
        TN = input.TN() # C_2|2
        fig = plt.figure()
        ax = fig.add_subplot()
        ax.grid(True)
        ax.plot([0, 1], [FP, TP], '--', label="Predict $C_1$")
        ax.plot([0, 1], [TN, FN], '--', label="Predict $C_2$")

        threshold = (FP - TN)/(FP - TN + FN - TP)
        cost_t = threshold*TP + (1-threshold)*FP
        ax.plot([threshold, 1], [cost_t, TP], lw=5, color='tab:blue', label="Optimal $C_1$")
        ax.plot([0, threshold], [TN, cost_t], lw=5, color='tab:orange', label="Optimal $C_2$")

        C = [[TP, FP], [FN, TN]]
        bbox = dict(boxstyle="round", fc="white")
        ax.annotate(r'$C_{2|2}$', (0, C[1][1]), xytext=(2, -1),
                    textcoords='offset fontsize',
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)
        ax.annotate(r'$C_{1|1}$', (1, C[0][0]), xytext=(2, 0),
                    textcoords='offset fontsize',
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)
        ax.annotate(r'$C_{1|2}$', (0, C[0][1]), xytext=(0, 2),
                    textcoords='offset fontsize',
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)
        ax.annotate(r'$C_{2|1}$', (1, C[1][0]), xytext=(2, 0),
                    textcoords='offset fontsize',
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)

        ax.annotate(f'$t*={threshold:0.2}$', (threshold, cost_t),
                    xytext=(0, --3),
                    textcoords='offset fontsize',
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)

        ax.set_xlabel('$P(C_1|x)$')
        ax.set_ylabel('Expected cost')
        ax.legend()

        return fig

app = App(app_ui, server, debug=True)
~~~

This `roadmap` has been created with `Jupyter Book` which does not support `Shinylive`. However, it is possible to create the Shinylive example in the online editor at https://shinylive.io/py/editor/ and then insert an iframe with the result as follows

```
<iframe src="https://shinylive.io/py/app/#code=NobwRAdghgtgpmAXGKAHVA6VBPMAaMAYwHsIAXOcpMASxlWICCyACGKM1A
data-external="1" width="100%" height="400px">
</iframe>
```

which is rendered as follows

Edit

Video recordings

The video recordings required the accompanying set of slides generated from the same course. Given that the slides contain a reduced version of the website, it is possible to create the slides and the videos before the narrative is finalised, which has been the approach taken for this course. The video were recorded in a home environment with non-professional devices such as a common laptop and its internal microphone. In order to record both the slides and the speaker we used OBS (Open Bradcaster Software) Studio which is a free and open-source software for video recording (and live streaming) available for Windows, Mac and Linux.

The background of the speaker was removed using an OBS Studio plugin that does not require a green screen . However, the parameters need to be adjusted which can affect the computation cost and the precision of the background removal. For online courses it is recommended to create short videos of a very specific topic (less than 10 minutes) to facilitate the time management of the students, and to potentially reuse some video recordings in similar courses. A total of four videos were recorded with an average length of approximately 9 minutes. The videos are currently hosted on YouTube and can be used without the other content.

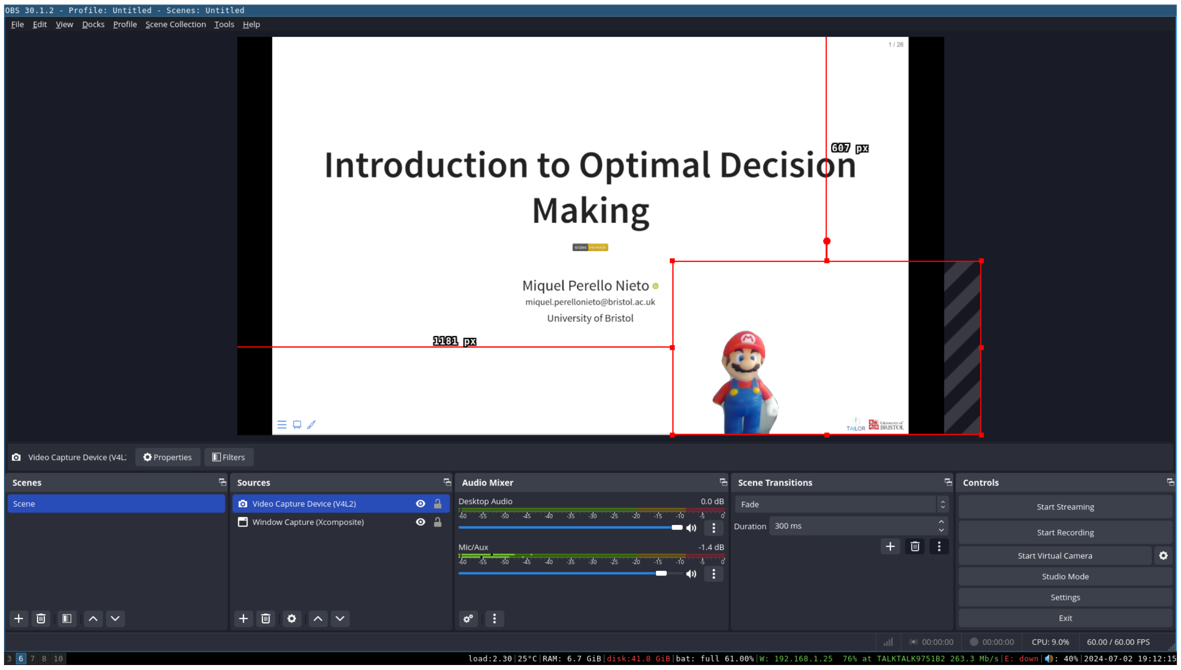
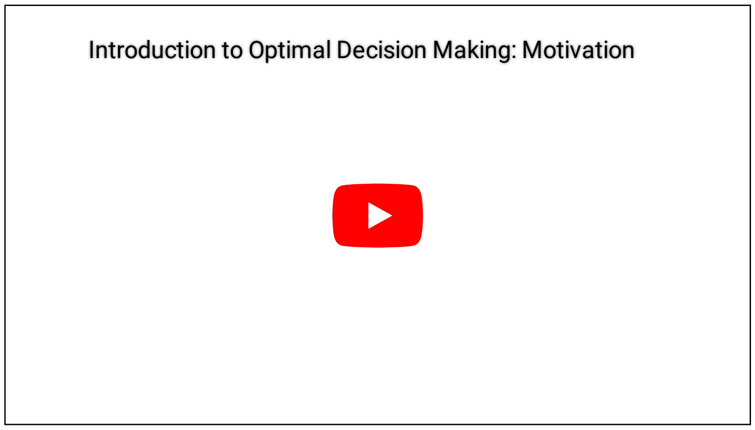


Fig. 4 OBS Studio configured to record the set of slides and video capture with a plugin that automatically removes the background.

In order to install the background removal plugin in Ubuntu 20.04 it is necessary to install OBS and the plugin from `flatpak` with the following commands:

```
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
flatpak install flathub com.obsproject.Studio
flatpak install flathub com.obsproject.Studio.Plugin.BackgroundRemoval
```

The following video is an example of the results obtained using the `reveal.js` slides generated with `Quarto` and the video recorded with `OBS` and the automatic removal of the background.



Convert a LaTeX Beamer presentation to Quarto

In this use case we start from an already existing set of slides made with LaTeX package Beamer. The course generated in this use case is available at https://tailor-uob.github.io/mooc_trustworthy_ai/cha_wahcc/wahcc.html.

This use case explains the process followed to generate the `Quarto` course material. In order to replicate the steps involved you can `download the presentation source files from this link`.

To generate the slides first unzip the downloaded file and run `pdflatex` and `biber` as follows

```
unzip -X cla_cal_slides.zip
pdflatex main.tex
biber main
pdflatex main.tex
```

The rest of this section uses the content of the zip file to extend the `Quarto` project started in the use case Create a Quarto course from scratch.

Pandoc: from LaTeX to markdown

Depending on the size of the LaTeX project it may be possible to manually copy the main content and edit it in such a form that is markdown compliant. This may be a good solution as at the end there is no perfect automation to convert LaTeX to markdown. However, there are a few alternatives that will do part of the job automatically. In particular, `pandoc` is a tool to convert text documents into a multitude of other formats (see Pandoc).
The command to convert a LaTeX file to `markdown` is the following.

```
pandoc main.tex -s -o main.md
```

However, this will result in a markdown with only the titles of the sections and empty content. This is because the LaTeX source code of this example contains several definitions and new commands that can not be converted by `pandoc` without manual modifications.

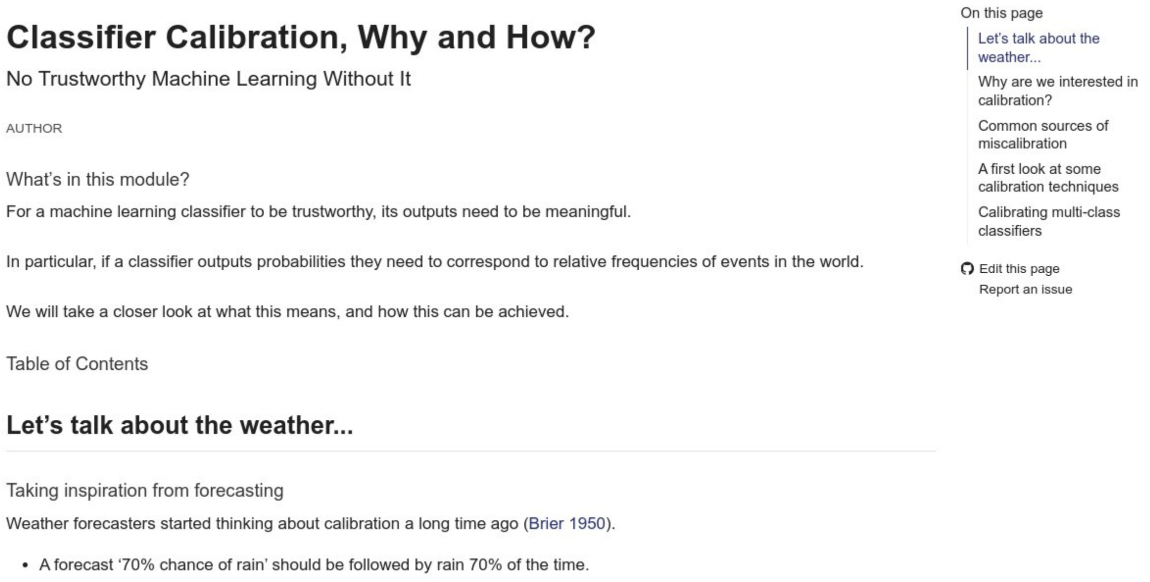
The first thing that needed to be changed was to define the command `brightFrame` which was previously defined in a separate file that contained lots of other definitions and commands. Remove the line 44 that loads the external file `frame-commands.tex`.

```
44: \input{latex/frame-commands}
```

And define the new command `brigthFrame` in the `main.tex` file

```
\newcommand{\brightFrame}[2]{
  \begin{frame}
    \frametitle{#1}
    #2
  \end{frame}
}
```

Only with this change the resulting `markdown` file will contain most of the original information including tables, some figures, equations, and references. The resulting markdown file needs to be renamed to include the `Quarto` extension `.qmd`, and it can be copied to a `Quarto` project together with the figures folder, bibliography and other documents. The next image shows the header of the resulting markdown content rendered into html.



Classifier Calibration, Why and How?

No Trustworthy Machine Learning Without It

AUTHOR

What's in this module?

For a machine learning classifier to be trustworthy, its outputs need to be meaningful.

In particular, if a classifier outputs probabilities they need to correspond to relative frequencies of events in the world.

We will take a closer look at what this means, and how this can be achieved.

Table of Contents

Let's talk about the weather...

Taking inspiration from forecasting

Weather forecasters started thinking about calibration a long time ago (Brier 1950).

- A forecast '70% chance of rain' should be followed by rain 70% of the time.

On this page

Let's talk about the weather...

Why are we interested in calibration?

Common sources of miscalibration

A first look at some calibration techniques

Calibrating multi-class classifiers

Edit this page

Report an issue

Fig. 5 Begining of the markdown document converted by Pandoc and rendered by Quarto.

Some of the figures that are originally in the pdf format are rendered inside of a pdf reader interface. This will need to be fixed manually by converting the pdfs into images.

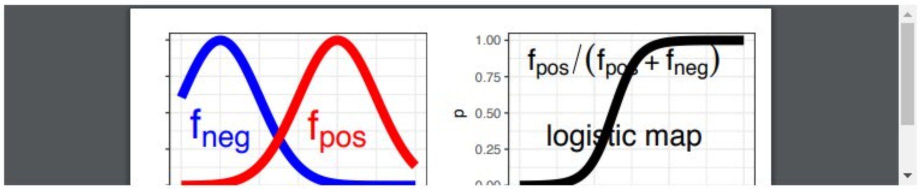
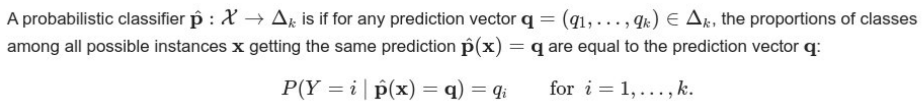


Fig. 6 PDF figures in the markdown file rendered by Quarto.

Equations in LaTeX are kept in their original form in the generated markdown files as pandoc will convert the equations to MathJax which is a JavaScript display engine for mathematics.



A probabilistic classifier $\hat{\mathbf{p}}: \mathcal{X} \to \Delta_k$ is if for any prediction vector $\mathbf{q} = (q_1, \ldots, q_k) \in \Delta_k$, the proportions of classes among all possible instances $\mathbf{x}$ getting the same prediction $\hat{\mathbf{p}}(\mathbf{x}) = \mathbf{q}$ are equal to the prediction vector $\mathbf{q}$:

$$P(Y = i \mid \hat{\mathbf{p}}(\mathbf{x}) = \mathbf{q}) = q_i \qquad \text{for } i = 1, \ldots, k.$$

Fig. 7 Equations generated by `Quarto` from the markdown file.

Another problem is that the original LaTeX file did not indicate the extension of **some** figures that were originally PDFs. With the absence of the file extension `Quarto` assumes that the figures are `png` files and adds that extension which is not correct. An option would be to manually modify all the figures in the `Quarto` markdown file clearly specifying the extension. However, in this case it is better to convert all the `pdf` figures to `png`, as those will be rendered better in the website.

Before converting the `pdf` files to `png` it will be necessary to crop all the `pdf` files as some of them have invisible parts that will show incorrectly in the `png` version. Running the following shell script will crop all the pdf files in the current folder.

```
#!/bin/sh

# it depends on texlive-extra-utils
# Install:
# sudo apt-get install texlive-extra-utils

for f in ./*.pdf
do
    echo "Cropping file $f"
    pdfcrop "${f}" "${f}"
done
```

Then we can proceed to convert all the `pdf` files. Here is a shell script to convert a set of `pdf` files to `png`

```
#!/bin/sh

# depends on gs
# sudo apt-get install gs

for in_file in "$@"
do
    out_file="${in_file%.pdf}.png"
    echo "Converting file ${in_file} to ${out_file}"
    gs -sDEVICE=pngalpha -o ${out_file} -sDEVICE=pngalpha -r1200 ${in_file}
done
```

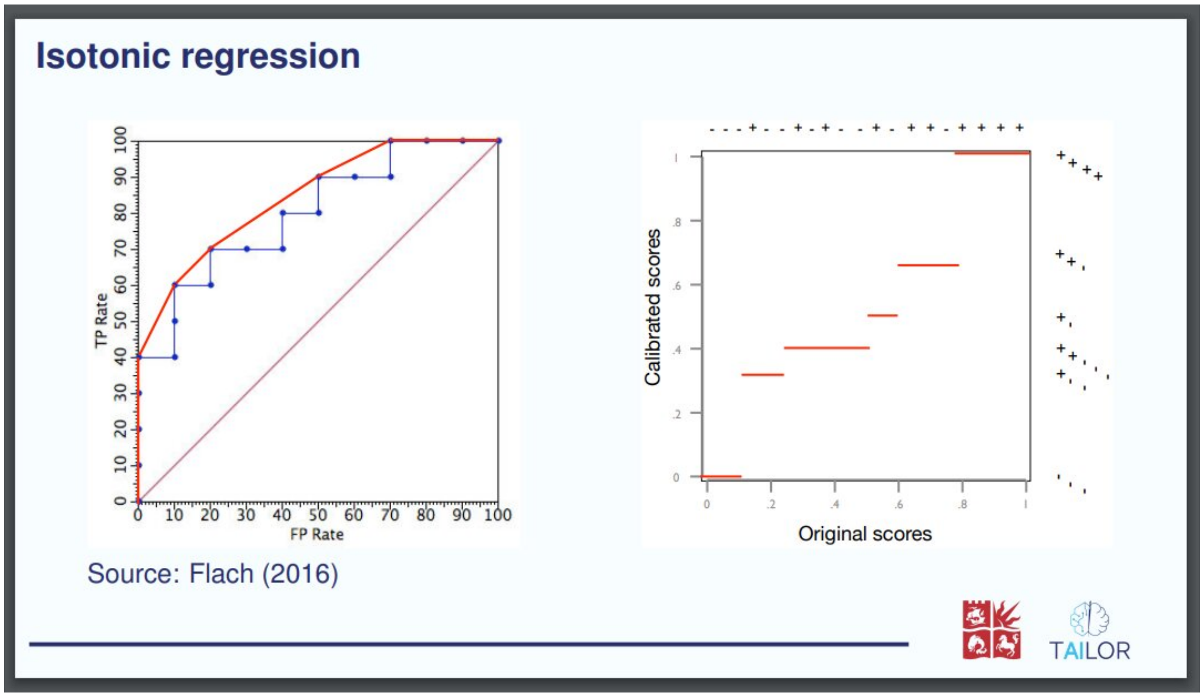
Only with this change all the figures that did not specify the extension will be loaded. It is also convenient to change the remaining extensions `.pdf` to `.png` now that we have the bitmap version of all figures.

Figure sizes and positions

Figures in the original set of slides were positioned and adjusted to fit the spacing provided by LaTeX Beamer. The change of spacing in the `Quarto` output makes it difficult to fit the figures in an appealing manner. For that reason, we had to manually adjust the automated generated markdown code. The original LaTeX code

```
\includegraphics[height=0.7\textheight]{figures/ROCCH.pdf}\hfill
\includegraphics[height=0.7\textheight]{figures/ROCcal2.pdf}\\
Source: \textcite{flach2016roc}
```

which considered the height of the slides to position them correctly as follows



Isotonic regression

Source: Flach (2016)

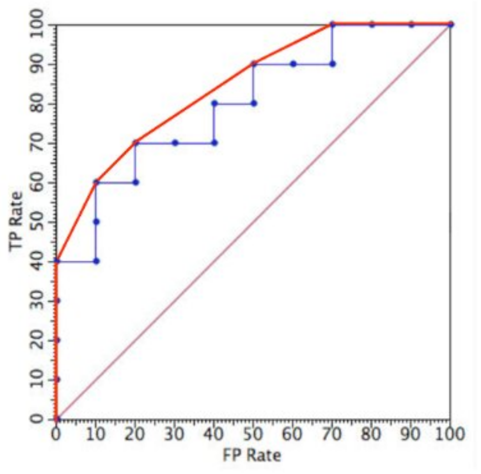
The code was automatically converted to markdown keeping the original height proportions

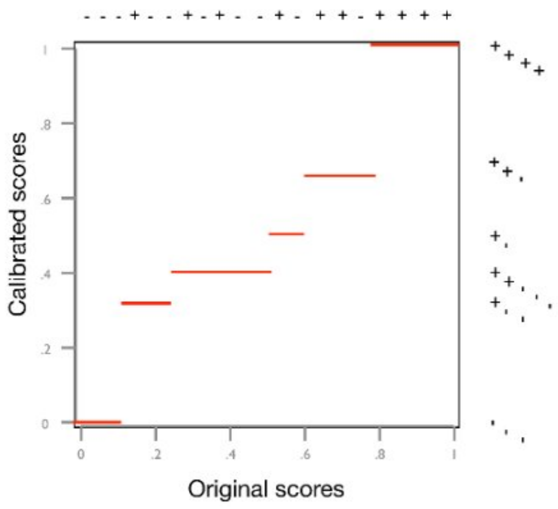
```
![image](figures/ROCCH.pdf){height="0.7\textheight"}
![image](figures/ROCcal2.pdf){height="0.7\textheight"}\
Source: @flach2016roc
```

However, the spacing in the `Quarto` webpage didn't have the same vertical dimensions, which made the figures extend to a large portion of the webpage. We manually changed the code by partitioning the body space into columns as follows

```
:::: {.columns .v-center-container}

::: {.column width="40%"}

![image](figures/ROCCH.png)

:::
::: {.column width="10%"}
:::
::: {.column width="47%"}
![image](figures/ROCcal2.png)

Source: @flach2016roc

:::
::::
```

which resulted in the following output





Source: Flach (2016)

Comments and line breaks

It turned out that some of the comments in LaTeX (lines that start with the % symbol) were not ignored during the conversion to markdown. The fourth line of the following example should be removed from the generated markdown file.

```
confidence $c$, is equal to $c$:
\begin{align*}
P(Y=i \: | \: \ph_i(\vx)=c)=c\qquad\text{where }\ i=\argmax_j \ph_j(\vx).
%P\Big(Y=\argmax\big(\vph(X)\big) \: \Big| \: \max\big(\vph(X)\big)=c\Big)=c.
\end{align*}
}
```

However it resulted in the following text, which kept the % symbol and joined the `\end{align*}` into the previous line.

```
confidence $c$, is equal to $c$: $$\begin{aligned}
P(Y=i \: | \: \hat{p}_i(\mathbf{x})=c)=c\qquad\text{where }\ i=\argmax_j \hat{p}_j(\mathbf{x}).
%P\Big(Y=\argmax\big(\vph(X)\big) \: \Big| \: \max\big(\vph(X)\big)=c\Big)=c.\end{aligned}$$
```

Errors like this need to be manually edited which depending on the number of occurrences can be time-consuming.

```
confidence $c$, is equal to $c$: $$\begin{aligned}
P(Y=i \: | \: \hat{p}_i(\mathbf{x})=c)=c\qquad\text{where }\ i=\argmax_j \hat{p}_j(\mathbf{x}).
\end{aligned}$$
```

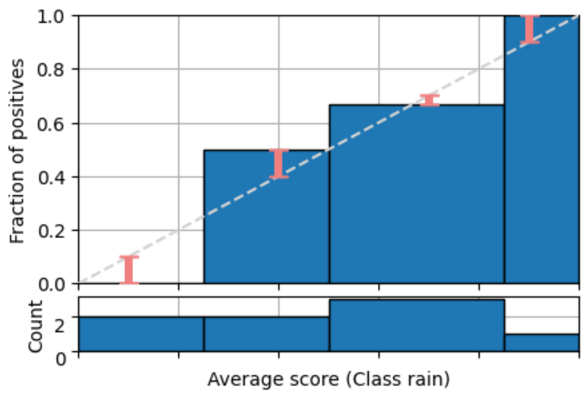
Generating figures from source code

The original slides required the figures to be generated in advance and imported from LaTeX. However, given that we had the `Python` code to generate the figures it is better to embed the code, which can be modified in the `markdown` file if we want to change the example. For example, the following markdown code loaded a figure

```
![image](figures/Forecaster1-fixed-gaps-v2){width="90%"}
```

which was originally generated with the following Python code. By adding the code to the `markdown` it is possible to generate the same figure during compilation both in the `Quarto` example, and in this same Jupyter Book roadmap.

```python
#| code-fold: true
#| code-summary: "Show the code"

import numpy as np
import matplotlib.pyplot as plt

from pycalib.visualisations import plot_reliability_diagram

labels = np.array([0, 0, 0, 1, 0, 1, 1, 1])
scores = np.array([0.1, 0.1 ,0.4, 0.4,0.7, 0.7, 0.7, 0.9])
bins = [0, 0.25, 0.5, 0.85, 1.0]
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram(labels, np.vstack([1 - scores, scores]).T,
                               class_names=['not 1', 'rain'], bins=bins,
                               fig=fig, show_gaps=True,
                               show_bars=True)
```

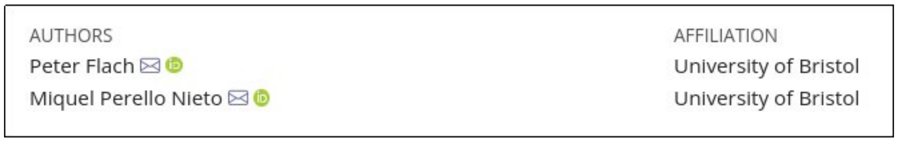


Authors in the header

`Quarto` allows the inclusion of metadata in the YAML header section of each markdown file, which can be used to display authors at the top of each page. This makes it very easy to collaborate in one `Quarto` publication where multiple authors worked in different sections, and clearly indicate their contributions. The following metadata

```yaml
title: Classifier Calibration
author:
  - name: Peter Flach
    orcid: 0000-0001-6857-5810
    email: peter.flach@bristol.ac.uk
    affiliations:
      - name: University of Bristol
        city: Bristol
        country: United Kingdom
        postal-code: BS8 1QU
  - name:
      given: Miquel
      family: Perello Nieto
    orcid: 0000-0001-8925-424X
    email: miquel.perellonieto@bristol.ac.uk
    affiliations:
      - name: University of Bristol
        city: Bristol
        country: United Kingdom
        postal-code: BS8 1QU
    attributes:
        equal-contributor: False

# Other configuration ommited
# ...
```

results in a list of authors with their affiliations, links to their e-mail addresses and ORCID profiles.

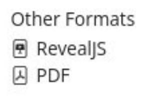


AUTHORS	AFFILIATION
Peter Flach ✉ iD	University of Bristol
Miquel Perello Nieto ✉ iD	University of Bristol

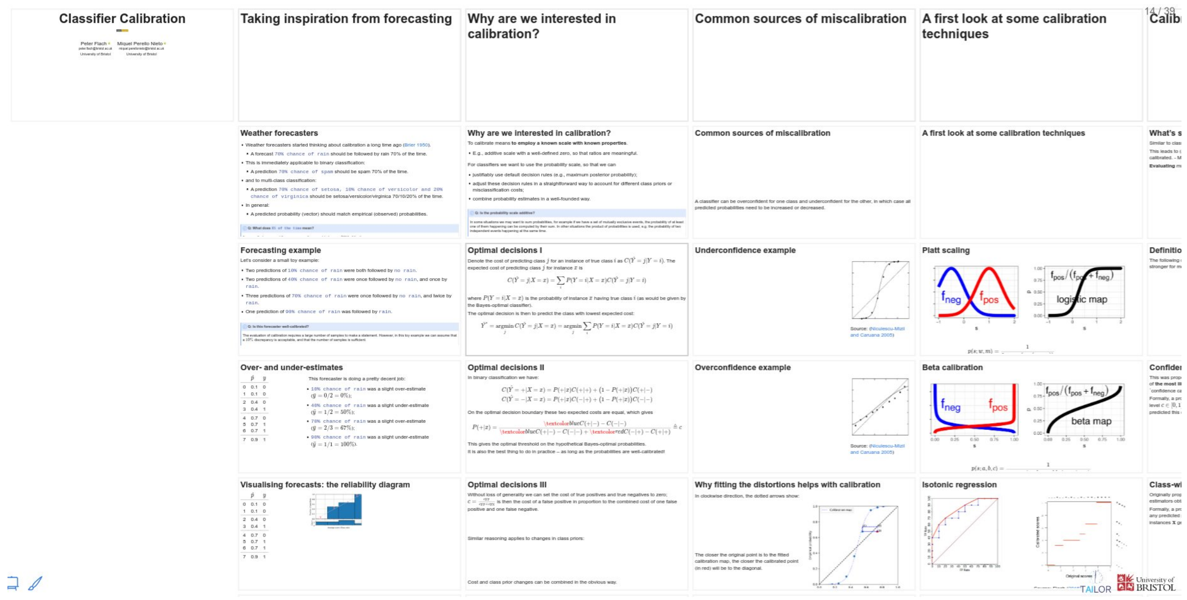
Generating Reveal.js slides

`Quarto` allows the generation of multiple types of outupt formats from the same `markdown` file. By adding the following configuration to the header, `Quarto` will generate `reveal.js` slides and provide a link to view the content in this format in the right-hand navigation bar.

```yaml
format:
  html:
    css: wahcc_style.css
  revealjs:
    logo: "../images/logos/tailor_and_uob.svg"
    output-file: slides-cla-cal.html
    slide-number: true
    width: 100%
    height: 100%
    incremental: true
    smaller: false
    auto-stretch: false
    chalkboard: true
bibliography: references.bib
```



Other Formats
RevealJS
PDF


Classifier Calibration

Taking inspiration from forecasting

Why are we interested in calibration?

Common sources of miscalibration

A first look at some calibration techniques

Weather forecasters

Why are we interested in calibration?

Common sources of miscalibration

A first look at some calibration techniques

Forecasting example

Optimal decisions I

Underconfidence example

Platt scaling

Over- and under-estimates

Optimal decisions II

Overconfidence example

Beta calibration

Visualising forecasts: the reliability diagram

Optimal decisions III

Why fitting the distortions helps with calibration

Isotonic regression

Fig. 8 Some of the slides created by Quarto in the Reveal.js engine.

Questions and answers

There are various ways in which questions and answers can be incorporated in the resulting website. In this particular case we opted to use notes with hidden text that can be inspected when clicking.

Q: What is going to happen when you click this question?

It is possible to work on multiple answers by stacking blocks of text together.

Question 1

Is only one of the following answers correct?

Answer: Yes

Answer: No

Video recordings

Peter recorded a video using the facilities of TAILOR partner Universitat Politècnica de València. The recording session took place in front of a green screen, with accompanying slides being placed in the background after recording. Some of the functionalities provided are the automatic detection of slide changes that presents the different slides in the video timeline for easy navigation, and the automatic generation of captions in various languages. For this use case Peter recorded a short video for each subsection of the presentation.

Videos can be embedded into websites with the `<iframe>` tag, which allows to embed one web page into another. The following HTML code is an example.

```html
<iframe allowfullscreen
src="https://media.upv.es/player/embed.html?id=003597b0-bf29-11ed-83a4-bf04f88f22c5"
style="border:0px #FFFFFF none;" name="Paella Player" scrolling="no"
frameborder="0" marginheight="0px" marginwidth="0px" width="640" height="360">
</iframe>
```

This shows the first video of the currently explained use-case.

Classifier Calibration, Why and How?

No Trustworthy Machine Learning Without It

Peter Flach
Peter.Flach@bristol.ac.uk
classifier-calibration.github.io/

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
University of BRISTOL
TAILOR
Foundation of Trustworthy AI:
Integrating Learning, Optimisation and Reasoning
poliMedia

It is a good idea to make short videos (between 5 and 10 minutes) covering a single topic or idea. This provide flexibility to the students to watch the videos during multiple sessions, and allows the reuse of content among different courses if the concepts are concise and general.

Breaking into slides and sections

The original set of slides had a specific structure that may not be good for a website. Furthermore, the `reveal.js` slides generated from the resulting markdown will not preserve the exact same divisions, which need to be manually specified in the markdown file. In this use-case some of the slide titles have been converted into sections or subsections manually. A new slide is automatically generated for each header of level 1 (`#`), including only the title, and headers of level 2 (`##`) create a new slide with that title and the following content. If a slide has too much information this can be split into other slides with an horizontal rule.

```
---
```

Conditional content

In certain situations we may want to show different content for different output formats. For example, in the current use case the videos are only shown in the HTML website, but not in other static documents or the revealjs website. Using the `.content-visible` and `.content-hidden` directives it is possible to specify parts of the markdown that are rendered only in the specified formats. The following are two self-explanatory examples.

```
::: {.content-visible when-format="html" unless-format="revealjs"}
This content will be visible in html websites, but not in html revealjs.
:::
```

```
::: {.content-hidden when-format="doc"}
This content will not be shown in word documents.
:::
```

Other changes

The markdown file required additional modifications to make the website and the `reveal.js` slides look good. Among the those changes are:

- Remove empty lines between each element of a list
- Manually added `highlights` with `text to highlight`

- Fix multiple issues with LaTeX and added definitions and packages in the header
- Adjusted the size of tables, figures and code to better fit the webpage and the slides

MyST client: LaTeX to HTML

We end this use-case with a short discussion of a way to convert LaTeX Beamer content directly to HTML using the MyST client Python package. However, we didn't find a method to obtain the intermediate MyST markdown files that are automatically generated, which may have made the process described above with pandoc much easier. A guide on how to do this conversion can be found at https://mystmd.org/guide/writing-in-latex. This section also provides a quick summary of the guide that results in a good HTML version.

This method requires the installation of the MyST client, which also requires an updated version of `node`. The installation of the required version of `node` can be done by creating a virtual environment for the specific version.

```
pip install nodeenv
```

You can list the available `node` version for virtual environments with

```
nodeenv --list
```

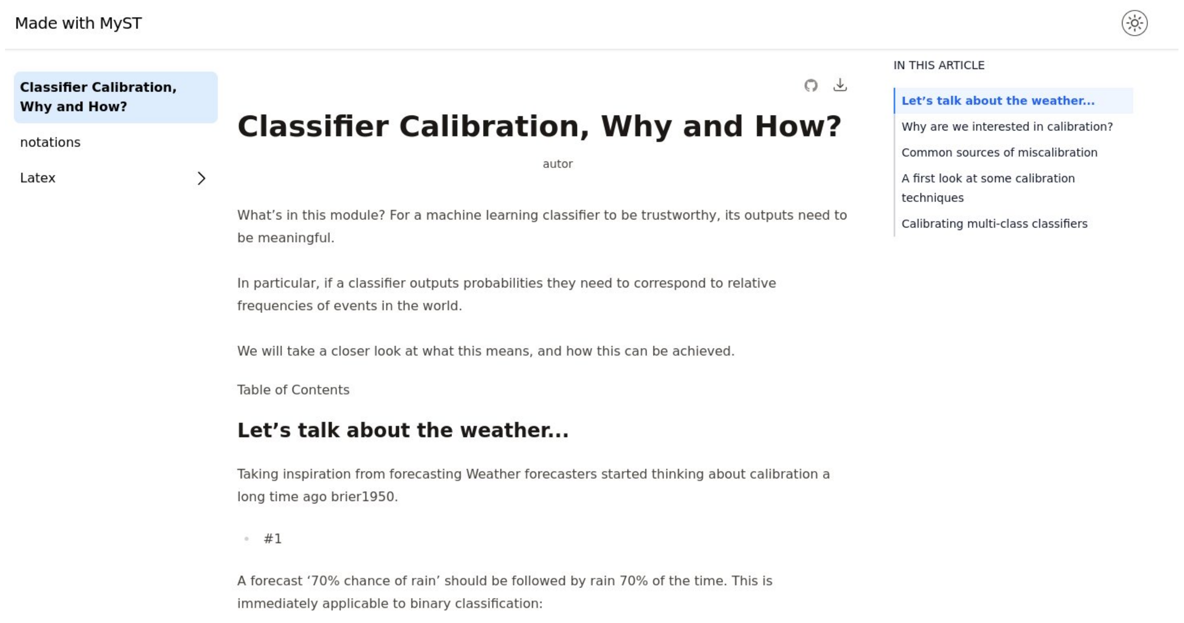
At the time of writing the required version was `20.11.1` which can be loaded by creating the virtual environment with the specific version and loading it.

```
nodeenv -n 20.11.1 node_env
source node_env/bin/activate
node -v
# v20.11.1
npm -v
# 10.2.4
```

One consideration from this example is that the LaTeX environment `refsection` is not currently supported, and all the environments need to be removed. This can be done by removing the beginning and end of each environment.

```
\begin{refsection}
...
\end{refsection}
```

The resulting website is in general very well formatted with some minor problems. The following is a capture of the resulting front page with a navigation bar on the left for the different documents and on the right side for the sections of the current document.


Made with MyST

Classifier Calibration, Why and How?

notations

Latex

IN THIS ARTICLE

Let’s talk about the weather...

Why are we interested in calibration?

Common sources of miscalibration

A first look at some calibration techniques

Calibrating multi-class classifiers

Classifier Calibration, Why and How?

autor

What’s in this module? For a machine learning classifier to be trustworthy, its outputs need to be meaningful.

In particular, if a classifier outputs probabilities they need to correspond to relative frequencies of events in the world.

We will take a closer look at what this means, and how this can be achieved.

Table of Contents

Let’s talk about the weather...

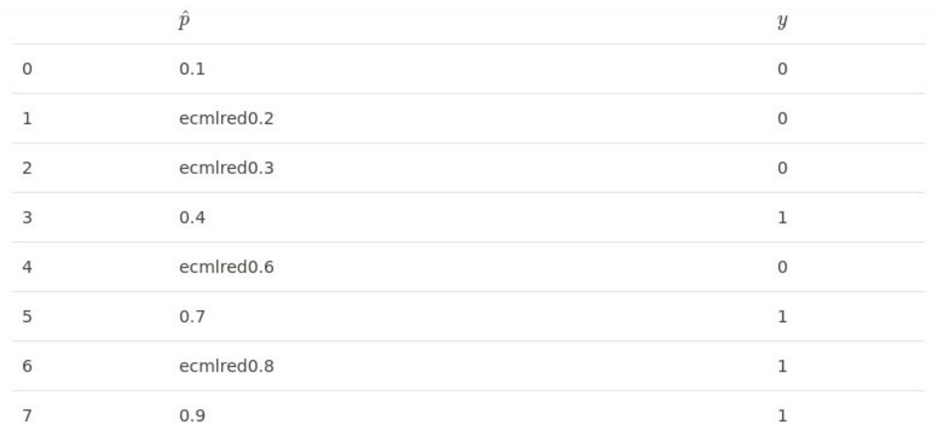
Taking inspiration from forecasting Weather forecasters started thinking about calibration a long time ago brier1950.

- #1

A forecast ‘70% chance of rain’ should be followed by rain 70% of the time. This is immediately applicable to binary classification:

Fig. 9 Automatic conversion of LaTeX Beamer file into a website with MyST client.

The previous figure shows one of the issues with the lists, which in this particular case were not converted to markdown correctly. However, tables, images and equations were correctly rendered in most cases. The table shown in Fig. 10 looks correct, but is missing the definition of colors specified in a separate LaTeX file.



	$\hat{p}$	y
0	0.1	0
1	ecmlred0.2	0
2	ecmlred0.3	0
3	0.4	1
4	ecmlred0.6	0
5	0.7	1
6	ecmlred0.8	1
7	0.9	1

Fig. 10 Simple LaTeX table converted with MyST client to markdown. The defined colors in LaTeX are not rendered correctly.

Images in different formats (even without clearly indicated extension) are correctly converted into bitmaps and placed in the right clocation as shown in Fig. 11.

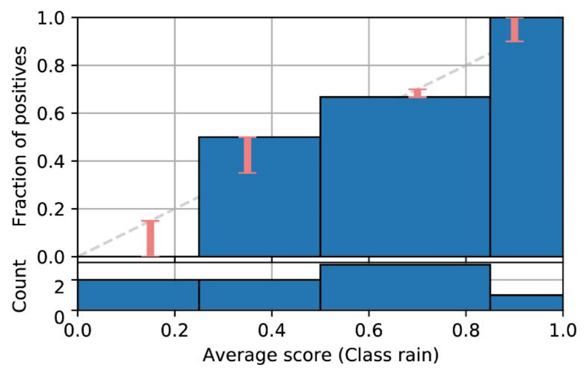
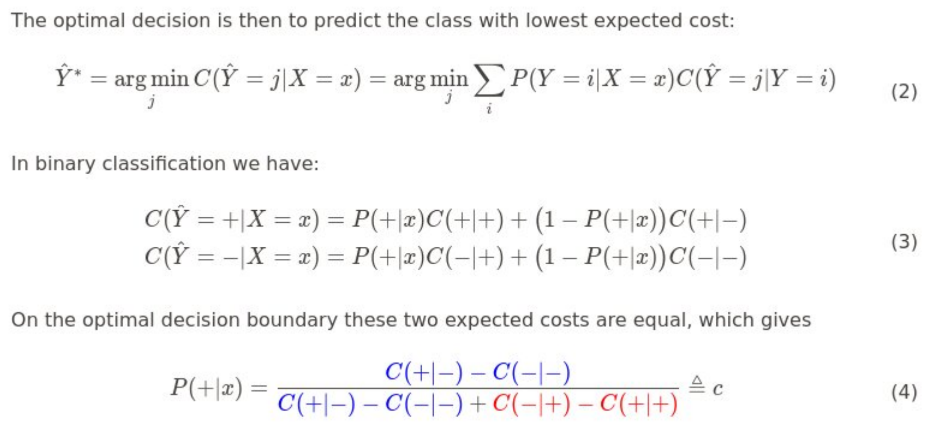


Fig. 11 Bitmap and pdf images are correctly converted into bitmaps and placed in the right location.

While equations are correctly formated even with special definitions of new commands and colors as shown in Fig. 12.



The optimal decision is then to predict the class with lowest expected cost:

$$\hat{Y}^* = \arg\min_j C(\hat{Y} = j | X = x) = \arg\min_j \sum_i P(Y = i | X = x) C(\hat{Y} = j | Y = i) \qquad (2)$$

In binary classification we have:

$$\begin{aligned} C(\hat{Y} = + | X = x) &= P(+|x)C(+|+) + \big(1 - P(+|x)\big)C(+|-) \\ C(\hat{Y} = - | X = x) &= P(+|x)C(-|+) + \big(1 - P(+|x)\big)C(-|-) \end{aligned} \qquad (3)$$

On the optimal decision boundary these two expected costs are equal, which gives

$$P(+|x) = \frac{C(+|-) - C(-|-)}{C(+|-) - C(-|-) + C(-|+) - C(+|+)} \triangleq c \qquad (4)$$

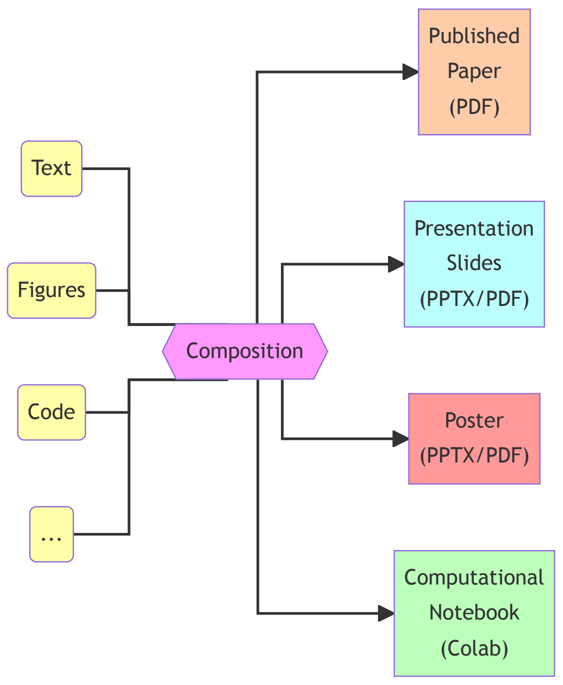
Fig. 12 Most of the equations are correctly displayed, even with colors.

Concluding remarks

Technology has moved far beyond the printing press, yet academic publishing is to a large extent still informed by the legacy format of printable documents. The growth of the World Wide Web is enabling the modernisation of teaching material which can be accessed from a large variety of devices. It also provides a platform for dynamic and interactive material that is automatically adjusted to the publishing medium. This roadmap has been written to help AI researchers embrace these new ways of publishing and the new modes of learning that they facilitate.

It should also be said that the rapid growth of web technologies creates a transitional environment for software and publishing tools that are still being developed and do not address all the requirements of such systems. Publishing systems like `Jupyter Book` and `Quarto` are among the most curated ones but are under heavy development. In a year from now they will have many new features that will push the boundaries even further. The challenge is for all of us to keep abreast of these developments and to make sure that we use the best tools for our purposes, while keeping in mind that this should not just be driven by the technological push but strike a balance with well-understood educational needs.

One area where current technology does not quite meet users' needs is in collaborative authoring. Markdown-based formatting tools such as `Quarto` offer maximum freedom how the authoring artefacts are produced, which some will see as an advantage but also means that a collaborative environment for authoring markdown files (e.g., HackMD) is separate from `Quarto` formatting. This situation is similar to LaTeX authoring prior to 2011 when Overleaf and ShareLaTeX were developed. Interestingly, the collaborative features in Overleaf such as commenting and tracked changes followed the model set by WYSIWYG editors such as Microsoft Word and Google Docs. A good collaborative environment combined with the flexibility of the Single-In-Multiple-Out paradigm would pave the way to the **Multiple-In-Multiple-Out (MIMO)** publishing paradigm that was already envisaged in the TAILOR deliverable D9.3 [SF21]. Such a system would pull together different authoring artefacts, possibly from different locations, and composes them in all the required output formats.



Appendix A: Open Courses

The idea of Open Courses is to take some of the principles of Open Research and adapt for the creation of online training material. Open Research is a set of principles to promote sharing research outcomes in a way that makes research reproducible and more transparent. There are at least four aspects that are considered:

- **Open Code** (Open source licenses)
- **FAIR Data** (Findable Accessible Interoperable Reusable)
- **Research Profile** (e.g. ORCID)
- **Open Access** of publications (Green, Gold, Hybrid, Diamond)

New ways of publishing potentially needs to consider all of the aforementioned points as the objective is to publish training material online with maximum reach. For transparency and accessibility the source code of the course should be open and any data used in the course should adhere to best practices. The authors of the course may also benefit from an open research profile which would give credibility to their content, transparency and increase the visibility to other work created by the same authors.

Open Code

As indicated for the datasets, publishing the code of the training material online does not entitle the way in which the course material can be used. When the course is created it is automatically protected by copyright. The course source code material without a license can not be used. It is important then to understand what are the licenses available and to add them to the course material properly. The information provided in this roadmap is a personal overview and should not be taken as legal advise, additional information can be found at https://opensource.guide/legal/. Furthermore, an interactive guide on how to choose a license is available at https://choosealicense.com/.

One of the most common licenses for open-source is `Creative Commons`. It has lots of variations depending on the flexibility that you want to provide to the users. The Creative Common licenses are short named with acronyms, and understanding them makes much easier their interpretation.

- **0 (No rights Reserved, Public Domain)**: There are no restrictions on how the code is used nor shared.
- **BY (By Attribution)**: Any use of the code needs to attribute the original work and author/s.
- **SA (Share-Alike)**: Any use of the code even if modified needs to keep the same license as the original work.

- **ND (No Derivatives)**: The work is shared as a whole and can not be modified.
- **NC (Non Commercial)**: The work can not be used for commercial uses.

The following are some common Creative Commons licenses:

- **CC-0**: No Rights Reserved, allows the distribution without accreditation, it is commonly used to share tabular data or other databases from which knowledge could be derived.
- **CC-BY**: Attribution, allows the use of the work even for commercial purposes but requires the attribution of the original form (e.g. with a citation). It is recommended for the widest dissemination of work.
- **CC-BY-SA**: Attribution-ShareAlike, allows the use of the work even for commercial purposes but requires the attribution and the same type of license to any derivatives.
- **CC-BY-ND**: Attribution-NoDerivatives, allows the use of the code as it is even for commercial uses, but does not allow the modification of the code. It also requires the accreditation of the original author.
- **CC-BY-NC**: Attribution-NonCommercial), allows the use and modification of the code for non-commercial use, subject to accreditation of the author and does not require the same license on its derivatives.
- **CC-BY-NC-SA**: Attribution-NonComercial-ShareAlike, allows the use and modification of the work for non commercial applications, requires accreditation of the original work and authors and the derived code needs to use the same license as the original.
- **CC-BY-NC-ND**: Attribution-NonComercial-NoDerivatives, allows the use of the code without modifications for non commercial uses and requires accreditation of the original work and authors.
- **The Restrictive License Template**: is a license developed by the Australian Government Open Access Licensing framework for material that contains personal or other confidential information. It can include multiple restrictions on its use like time limits, permissions or ethics required, or contractual arrangements). See more at https://library.unimelb.edu.au/Digital-Scholarship/restrictive-licence-template

Other common licenses for software are

- **MIT** (license) allows the use of the Software free of charge, with no restrictions but under the condition that there is an accreditation of the original software and authors and that the permission notice is included in all the copies or substantial portions of the Software.
- **Apache** (Apache License, Version 2.0) (license)
- **GPL** (General Public License) guarantees the end users the four freedoms to run, study, share and modify the Software.

And the BSD license which includes several versions

- **BSD 0-clause** (aka BSD Zero Clause License) allows the use, copy, modification and/or distribution of the Software for any purposes with or without fees.
- **BSD 2-clause license** (aka “Simplified BSD License” or “FreeBSD License”) same as 0-clause but requires to retain the copyright notice, the list of conditions and a disclaimer in the source code and in the documentation or other materials provided if used in its binary form.
- **BSD 3-clause** (aka “BSD License 2.0”, “Revised BSD License”, “New BSD License”, or “Modified BSD License”) same as 2-clause license but does not allow the endorsement or promotion of products in the name of the original copyright holders and contributors without specific prior written permission.
- **BSD 4-clause** (aka original “BSD License”) same as 3-clause license but all advertising material that mentiones the use of the original sofware must display the following acknowledgement: This product includes software developed by the <copyright holder>.

FAIR Data

The FAIR (Findable Accessible Interoperable Reusable) principles of sharing data were defined by a consortium of scientists and organisations, and published in the journal Scientific Data [WDA+16]. In STEM fields it is very common to share datasets with the students in order to better understand the training material. It is important to ensure that any data shared online in this manner is correctly licensed. The four principles that shape the acronym are:

- **Findable**: Additional metadata is added in order to easily identify and find the data with a search engine. The metadata needs to contain clear information and the data requires a unique identifier.
- **Accessible**: There is a specific protocol that can be followed to retrieve the metadata of the data of interest (even if the data is not available). The metadata is also understandable by humans and can be processed by machines. The data is stored in a trusted repository that ensures its accessibility for a particular period of time.
- **Interoperable**: The metadata follows a formal structure that is commonly accepted by multiple parties. Ideally the metadata has a vocabulary that follows the FAIR principles and can form a knowledge representation.
- **Reusable**: It is clearly specified how the data can be reused by others, including a license and any usage limitations.

Data format and storage

Data should be stored in such a way that is easy to understand by humans, machine readable and accessible; including metadata human and computer readible. The data should be stored in open file formats facilitating the accessibility, and not requiring propertary applications to be opened. Should provide good documentation (e.g. a README plain text file), and be stored in a trusted dat repository for long term storage.

Some examples of data repositories are:

- figshare allows to upload any file format and assigns a DOI identifier for citations.
- Mendeley Data allows the storage of public or private data, keeps versioning and ensures long-term storage by Data Archiving & Networked Services[2].
- Zenodo is a general-purpose open-access repository that facilitates making the repository private and can automatically make it open once an associated paper is published. No restrictions on the file type and datasets up to 50 GB.
- The Open Science Framework[3] is an open-source research management and collaboration tool to facilitate the documentation of a full project lifecycle.

Data Licenses

Publishing a dataset online does not entitle the ways in which the data can be used. When the data is generated it is automatically protected by copyright and without a license, users do not have permission to access, use and share under copyright and database laws. For example, in 2015, ~54% of the open data in open data catalogues across the European Union was not truly open as it was not licensed[1].

There are multiple licenses to choose from, and as legal documents they may differ in some small details. In order to facilitate the understanding of the licenses and maximise the use of your material it is advisable to use well known licenses.

Some of the most common licenses are the Creative Commons. Similar principles shown previously in the open-source licenses section apply here. For example the **Open Data Commons by Attribution License (ODC-BY v4.0**) allows to copy, distribute and use a database, produce new works derived from it, modify, transform and build upon it subject to attribution to the original work. A similar license that forces any derived work to be shared under the same license is the **Creative Commons by Attribution share-alike (CC-BY-SA v4.0)**. On the other hand, a less restrictive license for public domain is the **CC0** which does not require attribution.

The licenses can also be created to fit particular purposes, or to facilitate its adoption in certain organisations. This is the case for some governments may create their own open data licenses to facilitate its adoption among governmental organisations. Some examples of governmental licenses are the UK Open Government License[4], the French Government open License[5] and the Singapore Open Data Licence[6].

The European Union offers an online data licensing assistant[7]

The following video provides some quick guidelines on Data Sharing by the UK Reproducibility Network. The animation is shared under a CC BY license. Other UKRN primers are available at https://www,ukrn.org/primers.

Research Profile

The authoring of any material should clearly and uniquely identify the identity of the contributors. In printing material it was common to use the name and surnames of the authors as an identification. However, in multiple occasions this is not enough for disambiguation. With more recent online tools it is possible to disambiguate authors by providing additional information and creating unique identifiers for each of them. Webpages like ORCID[8] (Open Researcher and Contributor ID) provide unique identifiers for researchers and connect them with their contributions. Other identifiers are the Scopus Author Profile[9] by Elsevier, ResearcherID[10] from Clarivate, the Digital Author Identifier (DAI)[11] by the Dutch research system, among others.

Open Access

We provide a short introduction to Open Access for completeness of the Open Research principles, which is mainly directed for research publications. However, we believe this principle may be considered if the training material is reused for a research publication. Open Access is a broad international movement that specifies a set of principles to make research free of access and online, defined as “digital, online, free of charge, and free of most copyright and licensing restrictions.”. One of the main objectives is to publish research work under the CC-BY license. If the material for a course is later reused to publish a book or some other type of publication it may be necessary to know what are the possible licensing restrictions.

OPEN ACCESS

Fig. 13 Open Access logo

There are multiple types of Open Access models that are used by journals. However, the most common ones are gold, green and hybrid OA journals, while some journals are hybrid. In all the cases the article is free to read but they have the following differences.

Gold OA: The article is free to read, but the authors need to pay the publisher, requiring an external funding. The ‘Article Processing Charge’ costs an average of £2,000 and can reach £10,000.

Green OA: The author accepted manuscript can be hosted in some repository (e.g. Pure is a repository of scholarly works for the Universtiy of Bristol). The publisher retains the final version in their website. The article has a CC-BY license and there is no embargo on its use. Additional considerations may be required by the authors, for example the inclusion of a statement in the publishers submission, like in the case of the University of Bristol recomendation:

```
"For the purpose of open access, the author(s) has applied a Creative Commons
Attribution (CC BY) licence to any Author Accepted Manuscript version arising
from this submission."
```

Diamond OA: These are crowd-funded by libraries and scholarly organisations that pay for the processing charges. Then in the same manner as the **Green OA**, the accepted manuscript can be hosted in a repository, while the final copy is available in the publisher's website.

More about Open Research

Additional sources of information about open research can be found in the following links:

- Open Research at Bristol (UOB)
- Open Access for journal articles (UOB)
- Research Data Evaluation Guide (UOB)
- Managing research data (UOB)
- Dealing with sensitive data (UOB)

---

[2] https://www.nwo.nl/en/data-archiving-and-networked-services-dans

[3] https://osf.io/dashboard

[1] https://data.europa.eu/elearning/en/module4/#/id/co-01

[4] https://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/

[5] https://etalab.gouv.fr/licence-ouverte-open-licence

[6] https://beta.data.gov.sg/open-data-license

[7] https://data.europa.eu/en/training/licensing-assistant

[8] https://info.orcid.org/

[9] https://www.elsevier.com/en-gb/products/scopus/author-profiles

[10] https://clarivate.com/products/scientific-and-academic-research/research-discovery-and-workflow-solutions/researcher-profiles/

[11] https://en.wikipedia.org/wiki/Digital_Author_Identifier

Appendix B: Development environment

In order to build this website first you need to `clone` or `fork` the source repository. You can find a link to the repository by clicking on the GitHub logo at the top of this page and `Repository` in the pop-up menu (or click on the following link TAILOR-UoB/new_ways_of_publishing).

You can then `fork` or `clone` the respository and follow the instructions provided by GitHub.

Once you have a local version of the repository you need to create a virtual environment with all the dependencies. The following instructions have been designed for an **Ubuntu 20.04** machine, but will probably work in most operating systems.

Create a virtual environmnet with `Python3.9` at the root of the repository by opening a terminal at the root of the repository and running the following command

~~~
python3.9 -m venv venv
```

then activate the virtual environment

```shell
source /venv/bin/activate
```

We have created a `Makefile` that can help in all the following steps, here we
provide details about the process without using the `Makefile` and indicate
wahat

upgrade pip and install all the required dependencies

````{tab} shell
```shell
pip install --upgrade pip
pip install -r requirements.txt
```
~~~

Makefile

```
make install
```

```
pip install --upgrade pip
pip install -r requirements.txt
```

You will also need to install an additional python library that serves as an example to include your own Python code in the book

```
pip install -e lib/book-python/
```

The book can then be build by running the following command

```
jupyter-book build booksource
```

Once the virtual environment has been generated and loaded (`source /venv/bin/activate`), the book can be built with the shell command

```
make build
```

The previous build will compile only the source files that have changed. Some of the table of contents on other pages may still be out of date. To fully build the book from anew run the shell command

```
make cleanbuild
```

In case of wanting to run Jupyter Notebooks, you will need to create a kernel of the current virtual environment and load it from the Notebook.

```
pip install ipykernel
python -m ipykernel install --user --name data-science
```

Then, you can open the notebooks with the following command

```
jupyter notebook
```

References

[KW17] Albert Krewinkel and Robert Winkler. Formatting open science: agilely creating multiple document formats for academic manuscripts with pandoc scholar. *PeerJ Computer Science*, May 2017. URL: https://doi.org/10.7717/peerj-cs.112, doi:10.7717/peerj-cs.112.

[SF21] Kacper Sokol and Peter Flach. Training platform – beta report. Technical Report, Foundations of Trustworthy AI – Integrating Reasoning, Learning and Optimization (TAILOR), 2021. URL: https://tailor-network.eu/wp-content/uploads/2021/11/Extended-Deliverable-9.3-Report-v1.1.pdf.

[WDA+16] Mark D Wilkinson, Michel Dumontier, I Jsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, Jildau Bouwman, Anthony J Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J G Gray, Paul Groth, Carole Goble, Jeffrey S Grethe, Jaap Heringa, Peter A C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J Lusher, Maryann E Martone, Albert Mons, Abel L Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR guiding principles for scientific data management and stewardship. *Sci Data*, 3:160018, March 2016.

[ZLLS23] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. https://D2L.ai.