Florida AFS 2023 Open Science Workshop

2023-05-09

Table of contents

Course synopsis

Welcome to the 2023 Florida AFS open science workshop! Open science (OS) has been advocated as an effective approach to create reproducible, transparent, and actionable research products. However, widespread adoption among the research and management community has not occurred despite its perceived benefits. In the face of major environmental challenges, the collaborative framework provided by OS is needed now more than ever. This workshop will cover material introducing participants to core concepts of OS. The target audience includes anyone interested in applying OS in their own workflows as part of a larger research and resource management team.

By the end of this workshop, you should have a good understanding of fundamental concepts in open science and how they can be applied to help bridge the research-management divide.

You will also have the skills to understand how collaborative open science tools can be used to increase efficiency and transparency, understand fundamental best practices for working with data to facilitate openness, and create reproducible Quarto documents.

Much of the content on this web page was adopted from the TBEP Data Management SOP.

Prepare

Please attend the workshop with a personal laptop and power supply. Make sure your laptop can access publicly available WiFi. **You need to install software prior to the workshop, visit the setup page for full instructions**. We will have limited capacity to help with installation issues the day of the workshop, so please come prepared. The setup instructions will guide you through the following.

1. Install R: link
2. Install RStudio: link
3. Install Quarto: link
4. GitHub create account: link
5. Install Git (optional): link

We also assume some knowledge about R. Please visit this page for a crash course if you need to brush up on your R skills.

Agenda

1. The basics of open science: 12:30 - 1:00
2. Open science for collaboration: 1:00 - 2:00
3. Open science for impactful products: 2:15pm - 3:30pm

Each module uses a set of common icons to orient you to specific tasks or experiences during this workshop. These include the following:

Exercise and discussion

Watch and learn

Description of a collaborative tool

Pros of a collaborative tool or solution to an open science challenge

Cons of a collaborative tool

Challenge to overcome for open science

Instructor

Dr. Marcus Beck is the Program Scientist for the Tampa Bay Estuary Program and is developing data analysis and visualization methods for Bay health indicators. He received his PhD in Conservation Biology from the University of Minnesota in 2013. Marcus has experience researching environmental indicators and developing open science products to support environmental decision-making. Marcus is also an open source software and dashboard developer to facilitate science application. CV, Google Scholar, GitHub

---

This website is licensed under a Creative Commons Attribution 4.0 International License.

This version of the website was built automatically with GitHub Actions on 2023-05-09.

Part I

Modules

1 Open science basics

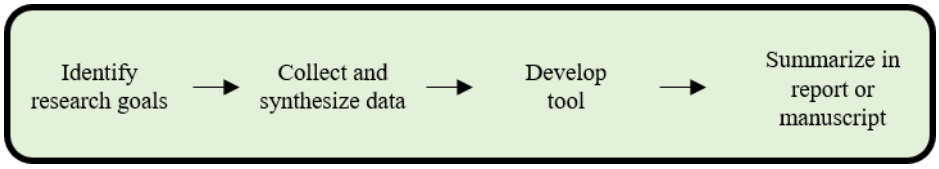
1.1 Goals and motivation

This is the first module in our workshop on open science. This module describes the need for open science, how it can improve research applications, and exposes you to common ideas and terminology that we'll be using throughout the day. Consider this your 30,000 foot view of open science. Our later modules will provide more detail on specific topics in open science that you can use for continued learning.

- **Goal**: get comfortable with key ideas and concepts for understanding open science
- **Motivation**: this is the first step in your open science journey!

1.2 Why open science?

Let's start with revisiting the scientific process. I'm sure this looks familiar to all of you. This is geared towards an applied research question.



Our basic scientific approach to discovery is motivated by a question or research goal, developing a hypothesis for the question, collecting data based on the hypothesis, developing a tool that can be used for decision-making, and summarizing the results in a conventional format.

Many scientists, especially early career researchers (my past self included), may assume that this is sufficient to affect change. We write the report, send it out into the world, and move on to the next project. This is a common mentality:

> "This 500-page report will answer all of their questions!"

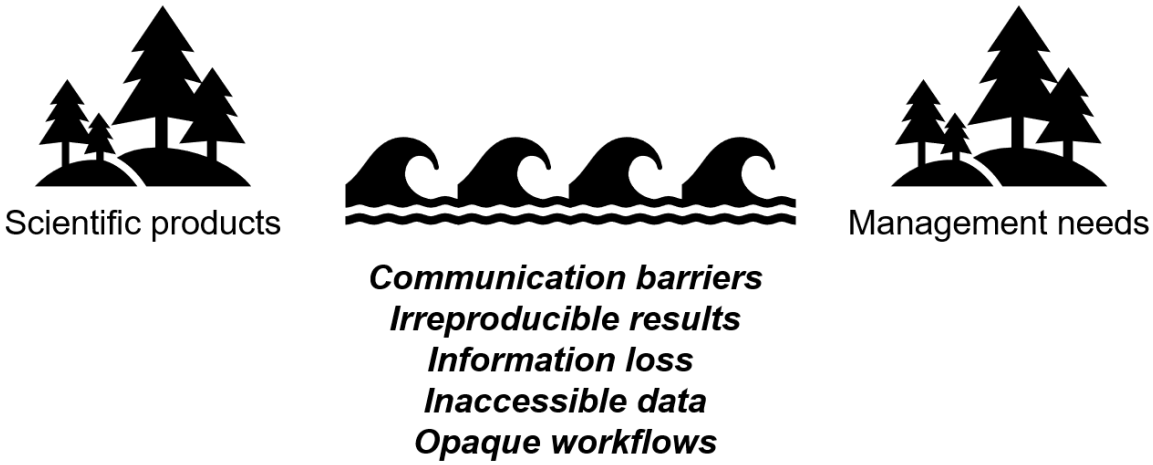
From the other side, such as the manager or policy-maker, the report may be received like this:

> "This 500-page report answers none of my questions!"

It's dense, inaccessible, and there are probably questions about the underlying data and methods used to achieve the results. More importantly, it doesn't present the information in an easily digestible format to quickly make the right decision. Sometimes, if you think you're doing applied science, it may just be *implied* science that falls short of application.

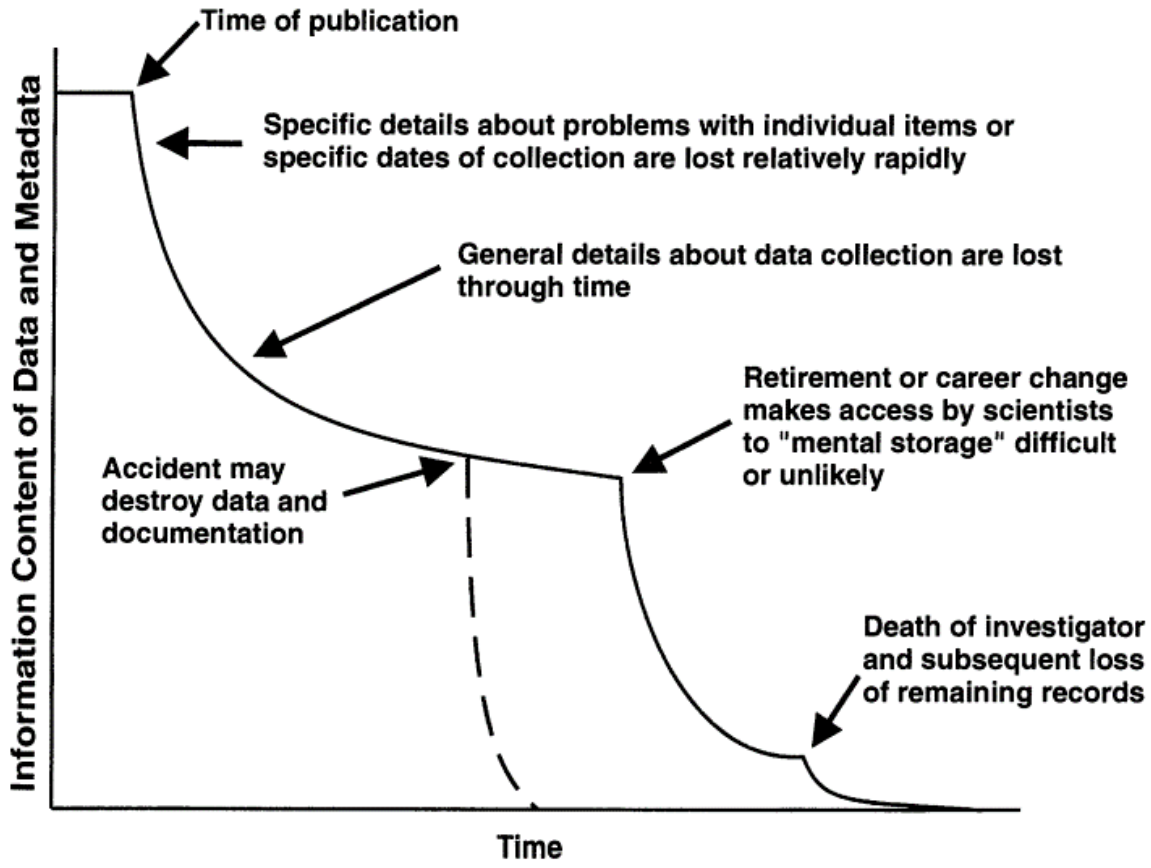
Why is this conventional approach to science ineffective at seeding change?

The environmental management community is often siloed with each branch doing their own thing and speaking their own language. Between the research (typically academic) and management community, we call this the research-management divide.

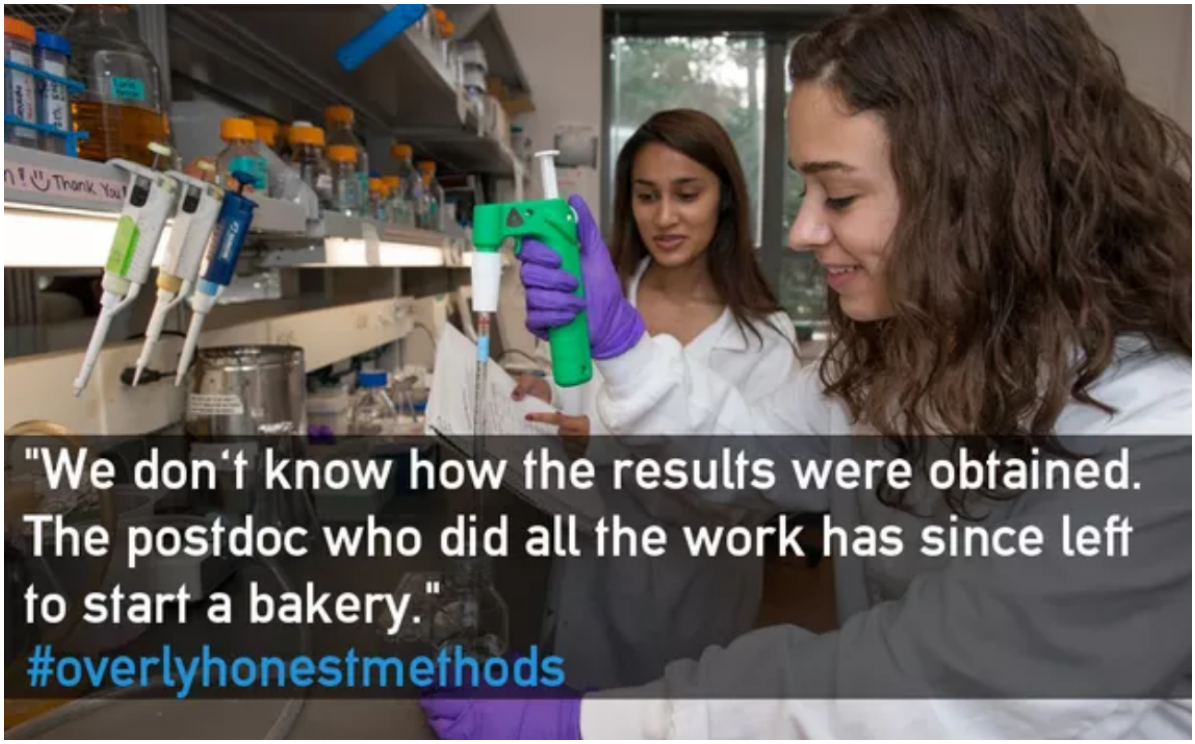


A distinct gap exists between how scientific products are developed and how they can be used to meet management needs. This is often the result of communication barriers, irreproducible results, information loss with poor documentation, inaccessible data, and opaque workflows known only to the analyst.

These barriers can occur at any stage of the research process. This compelling graphic from Michener et al. (1997) describes the atrophy of information in a closed approach to creating science.



The last part is especially morbid. Sometimes, this is called the bus factor. What would happen to your important work and life achievements if you were hit by a bus? Would others be able to pick it up? Research products with a high bus factor are at risk of being lost if critical team members are no longer available. This is a very real problem for continuity of science.



So how do we make changes to our workflows to ensure we can achieve truly applied science using open tools and philosophies?

1.3 Learning and speaking the language of open science

The tools and broader philosophy behind open science can help us bridge the research-management divide. It involves a fundamental shift in how we approach the scientific process, both for your own internal workflows and how you can engage others in the process. By others, we mean not just researchers, but specifically those that need the information to make informed decisions. This also includes your *future self*.

Now, let's settle on a definition for open science (from Open Knowledge International, http://opendefinition.org/, https://creativecommons.org/):

> "The practice of science in such a way that others can *collaborate* and *contribute*, where research data, lab notes and other research processes are *freely available*, under terms that enable *reuse*, *redistribution* and *reproduction* of the research and its underlying data and methods."

Key words from this definition are italicized. There are very specific tools in the open science toolbox that enable each of these key words. We'll cover some of these later.

Similarly, the current administration has declared 2023 the Year of Open Science. Their definition is:

> "The principle and practice of making research products and processes available to all, while respecting diverse cultures, maintaining security and privacy, and fostering collaborations, reproducibility, and equity."

We can can breakdown these definitions into key principles.

1. **Open data**
    - Public availability of data
    - Reusability and transparent workflows
    - Data provenance and metadata
2. **Open process**
    - Iterative methods using reproducible workflows
    - Collaboration with colleagues using web-based tools
    - Leveraging external, open-source applications
3. **Open products**
    - Interactive web products for communication
    - Dynamic documents with source code
    - Integration with external networks for discoverability

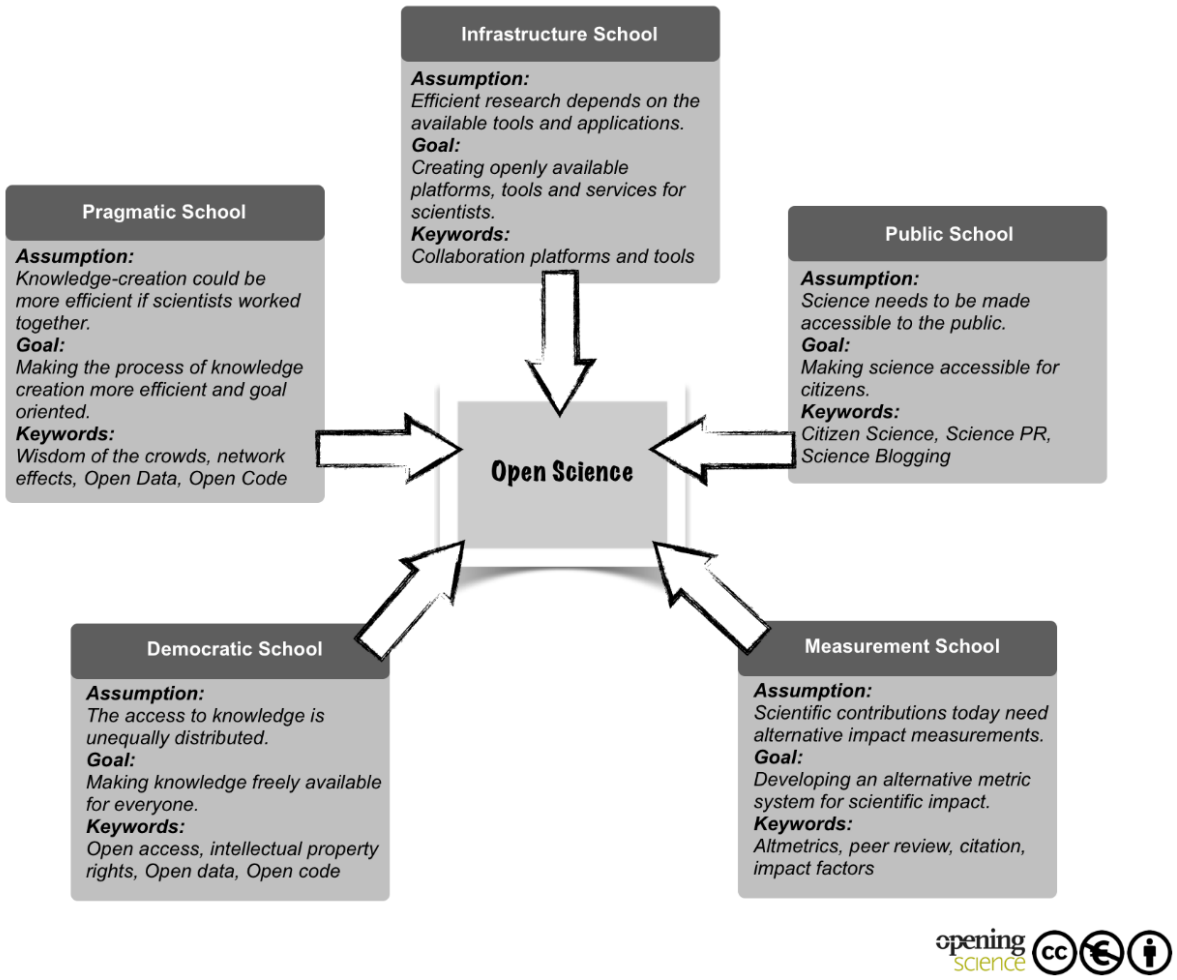
You'll notice that web-based tools and open science are often discussed at the same time. Science existed before the internet. Open science often focuses on how the two can leverage and support one another despite the latter being a relatively new addition to society.

Advocates of open science also use the FAIR principles (Wilkinson et al. 2016) as guidelines. The FAIR acronym stands for Findable, Accessible, Interoperable, and Reusable. Anybody should be able to find your science, access it once it's found, use it in different environments, and reproduce it for additional analysis.

1.4 Schools of thought

Finally, it's useful to make a distinction of how different people may talk about open science. This can help you better navigate conversations to become an advocate for open science.

A useful paradigm is provided by Fecher and Friesike (2014) that describes open science as five distinct schools of thought:



These are of course only conceptual boxes and there's considerable overlap when open science is used in practice. For our purposes, we'll mostly be talking about ideas and tools from the pragmatic, infrastructure, and democratic schools of thought. The end goal is to provide you with the means to create more efficient and impactful science that can more readily be used by others in a collaborative setting.

2 Open science for collaboration

2.1 Goals and motivation

This is the second module in our workshop on open science. This module will explore some open science tools to help you and your team become better collaborators and to better engage your science with external partners. We'll introduce some essential elements of collaboration and discuss some readily available tools for doing so.

- **Goal**: understand methods of collaboration and the pros/cons of various tools
- **Motivation**: start building the tools for your open science toolbox

2.2 Essential elements of collaboration

We start our deep dive into open science by focusing on collaboration as a fundamental activity that can be enhanced through transparent, efficient, and reproducible tools. Having effective tools to work together is a critical theme of many open science practices.

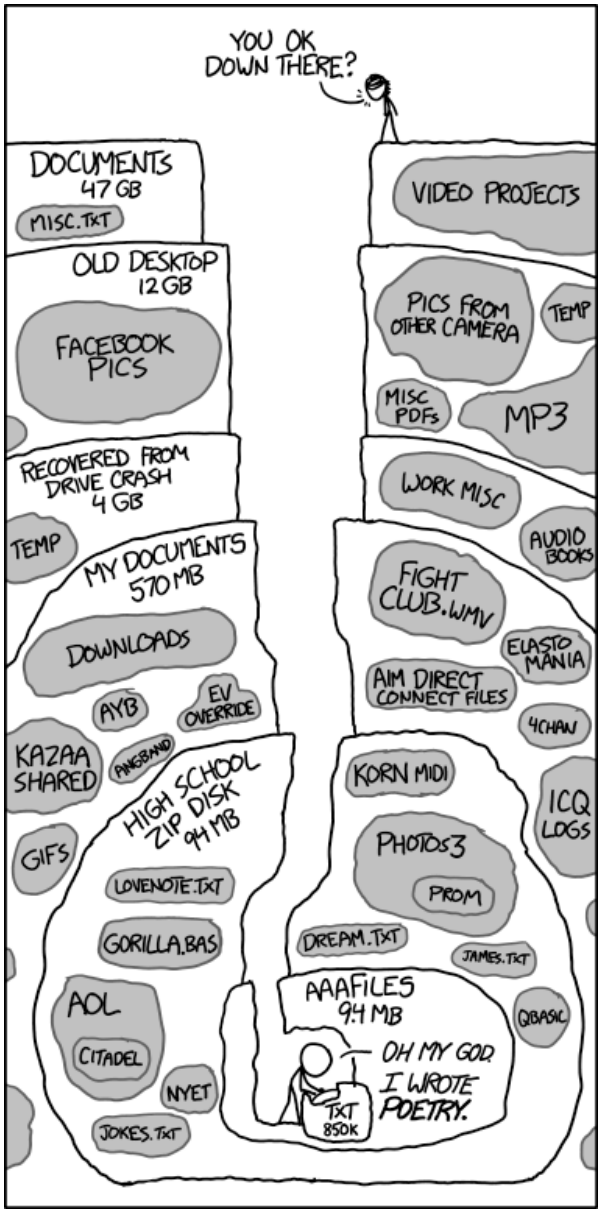
2.2.1 Workflow management

How do you organize your work each day? How do you make sure projects are on schedule and pressing deadlines are met? How do you plan for short-term and long-term goals? Do you have a five-year, ten-year, or longer career plan?

Work to achieve goals cannot be accomplished without a systematic approach to organizing tasks. Chances are, we each have our own system that works for us that was probably developed through trial and error. Although everyone has familiar workflows, they are often idiosyncratic and deeply entrenched by habit. These comfortable workflows can be in direct conflict with collaboration when we try to mesh them with the habits of others.

Does this look familiar?

Although the above comic from xkcd speaks directly to file management, it hints at a broader problem of personal information management that can seriously complicate working with others. I'm sure we've all struggled to find that one file for that one project from a vague recollection of seeing it a few months ago.

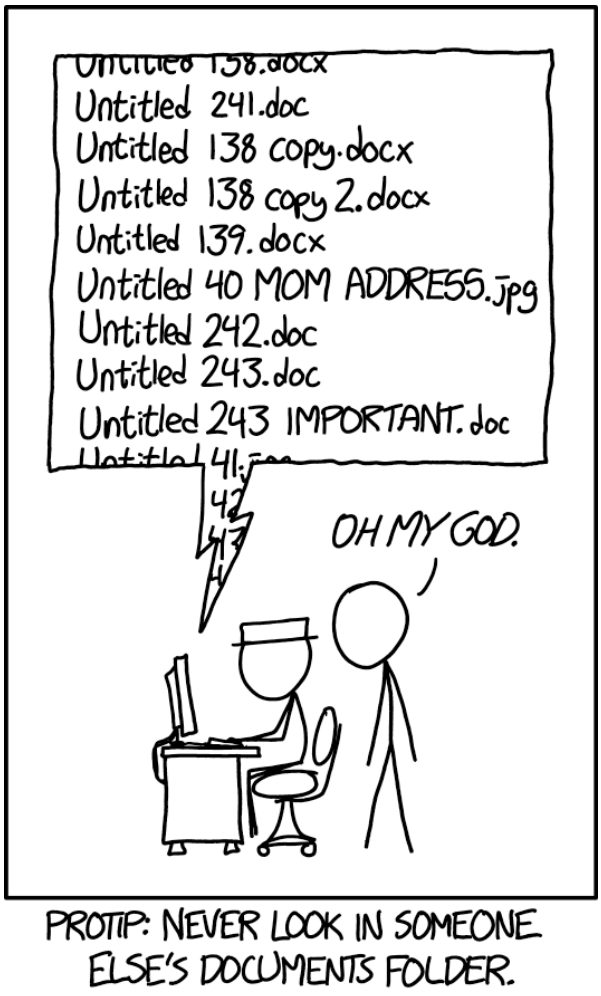

YOU OK DOWN THERE?
DOCUMENTS 47 GB
MISC.TXT
OLD DESKTOP 12 GB
FACEBOOK PICS
RECOVERED FROM DRIVE CRASH 4 GB
TEMP
MY DOCUMENTS 570 MB
DOWNLOADS
AYB
EV OVERRIDE
KAZAA SHARED
PINGBAND
HIGH SCHOOL ZIP DISK 94 MB
GIFS
LOVENOTE.TXT
GORILLA.BAS
AOL
CITADEL
NYET
JOKES.TXT
VIDEO PROJECTS
PICS FROM OTHER CAMERA
TEMP
MISC PDFS
MP3
WORK MISC
AUDIO BOOKS
FIGHT CLUB.WMV
ELASTO MANIA
AIM DIRECT CONNECT FILES
4CHAN
KORN MIDI
ICQ LOGS
PHOTOS3
PROM
DREAM.TXT
JAMES.TXT
AAAFILES 9.1 MB
QBASIC
OH MY GOD I WROTE POETRY.
TXT 850K

Collaborative work can be facilitated through workflow management that helps you break out of old habits. We'll introduce some specific internet-based tools below to facilitate workflows either for yourself or, better yet, working with others. These can help propel you towards open science.

2.2.2 Version control

A specific problem for workflow management that can be solved by open science tools is file management. Workflows can be immensely enhanced by tools that use strict guidelines for tracking changes and allowing a complete view of the evolution of a project. This is where version control comes in.

I'm sure many of you have fallen into this trap:

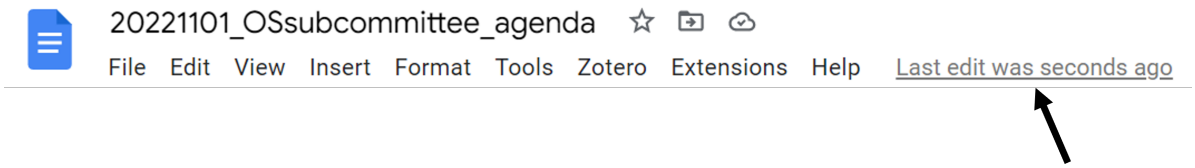


Version control is a way to track the development history of a project. It serves the joint purposes of:

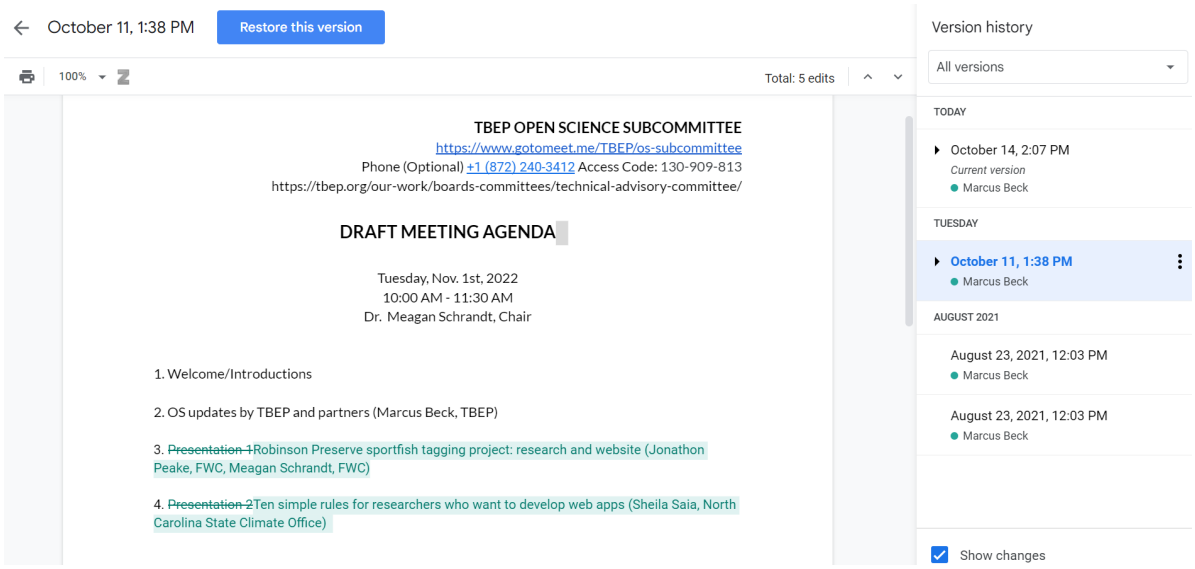
1. Formally documenting the changes that have been made to code, software, or text
2. Making sure that the development history is permanent
3. Providing a system for collaborating across platforms (with friends!)

It's more than saving files. Documenting changes with a set of commands that follow strict rules provides a transparent record for yourself and others, and establishing permanency ensures that any of the changes that are made can be vetted and accessed as needed. Think of it as an insurance plan for your project.

If you've ever used Google Docs, you might have noticed a feature that looks a lot like version control. The Google Drive platform is a great way to start working together and to familiarize yourself with the basics of version control.



For any Google Doc, clicking on the link shown by the arrow will open the Version history pane which shows all of the edits that were made to the document. You can view any of the edits, who made the edits, view the changes (before/after) in the document, or even restore the document to a previous version.



These are the building blocks of version control:

1. No iterative and ambiguous file naming
2. History of changes assigned to each editor
3. Ability to restore a previous version

Perhaps more importantly, these tools are in the cloud and openly accessible (unlike other cloud-based services). File links (via a URL) also do not change if a file is moved to a different location in the drive. Overall, the Google platform is an accessible means of improving collaboration (but not without cons).

2.2.3 Git and GitHub

Although Google products can get you a long way towards better collaboration, they do not use dedicated version control software. These tools become more important as your projects become more complex - those beyond simple documents or spreadsheets.

The most widely used software for version control is Git. Although we do not cover the specifics of this software, it's useful to understand the purpose and what it can do in making your work more open and impactful. Git is integrated with many popular open source development platforms, such as RStudio.

Many people often confuse Git with GitHub. GitHub is an online platform for working collaboratively through Git AND it allows you to be open with your work. We'll provide some examples below of how this can be done. Importantly, you do not need to be an expert in Git to be able to use GitHub. This speaks volumes for how team efficiency can be improved with GitHub through better collaboration.

This recent blog provides a helpful introduction to Git/GitHub for the casual user.

A common workflow for using Git and GitHub is shown below. One developer creates the core content on their own computer and uses GitHub to host the repository online. From there, collaborators can contribute to the repository through their web browser. A web page hosted on GitHub can create a public-friendly format for others to view important content.

Watch and learn

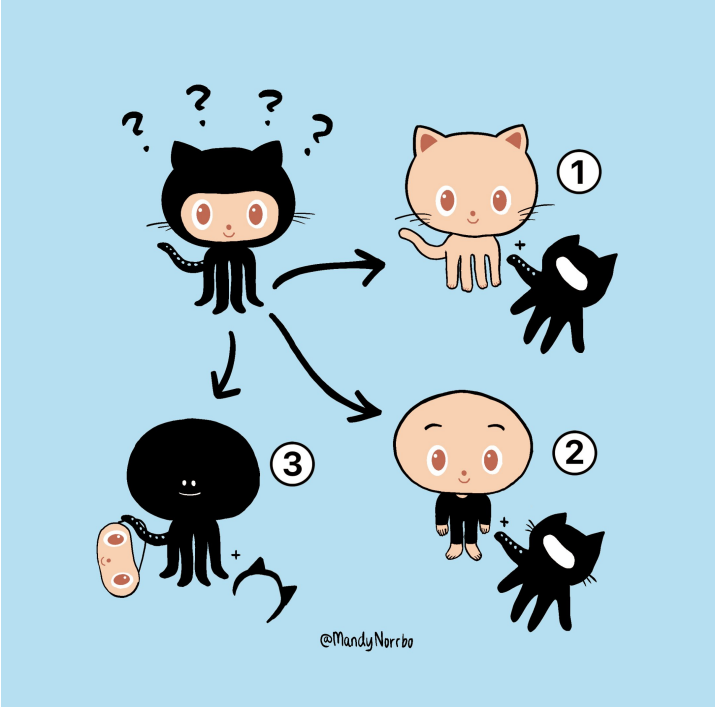


Figure 2.1: Octocat, the strange and loveable mascot of GitHub.

Workflow management in the real world - using GitHub to collaborate. Here we present some examples from the Tampa Bay Estuary Program State of the Bay report and water quality report card.

Watch and learn

Now we'll demonstrate how to setup a version control project with RStudio, Git, and GitHub. This example will cover:

1. Creating the project in GitHub
2. Creating a file, adding content, and committing it to the project
3. Setting up issues in GitHub
4. Adding members to the project
5. Creating a Kanban project board to assign tasks

Exercise and discussion

In small groups, setup a shared workspace using GitHub and create a project management board. Some real world examples of why you might do this were presented in the earlier watch and learn.

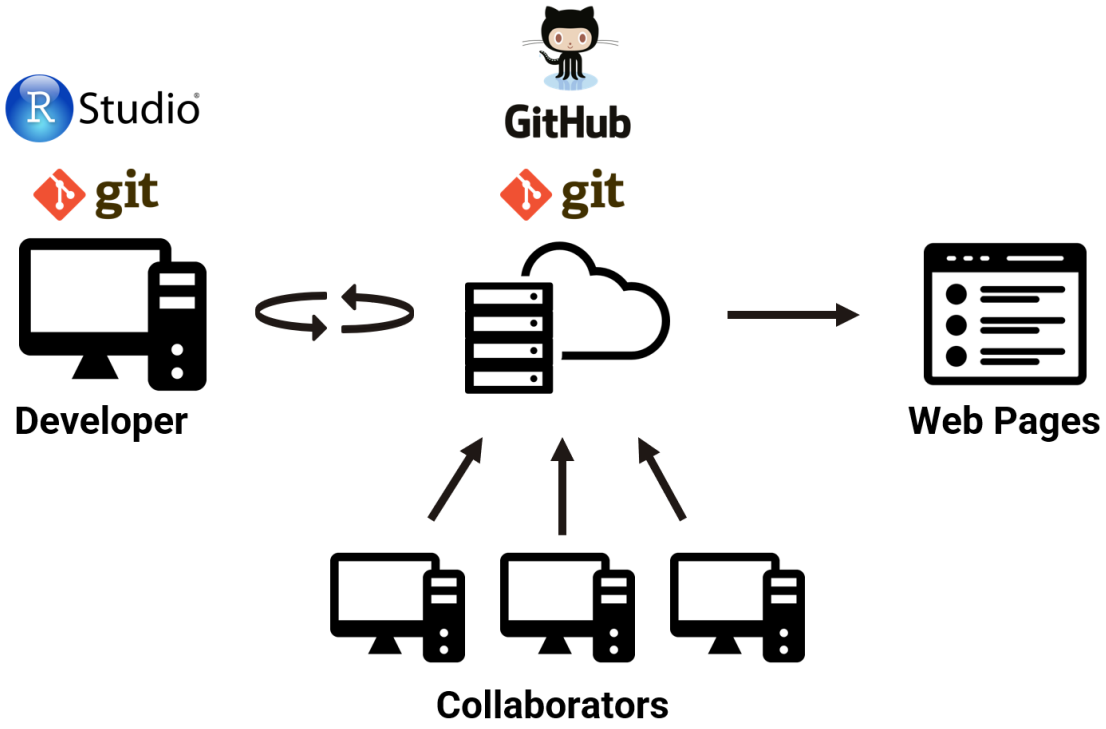


Figure 2.2: A GitHub workflow engaging collaborators.

1. Open GitHub in a web browser and have one person create a new repository (the big, green "New" button in Repositories). Add each member to the repository after it's created (hint: Settings -> Collaborators)
2. Have that same person create a project board for the repository (Hint: Projects -> New project -> board format)
3. After each person accepts the invitation to the repository (check your email!), each new member create a new file in the repository (Hint: Click "Add file" near the top). Name it something unique, save and commit the changes
4. Assign issues to different members of the repository to do something to the new files (Hint: on the right menu, select "Assignees"). Add the issue to the project board (Hint: on the right menu, select "Projects" and click the new project).
5. Work on the issues until the time is up. Close each issue as they're completed.

3 Open science for impactful products

3.1 Goals and motivation

This is the third module in our workshop on open science. Now we focus on how Quarto can be used as a document preparation system to generate easily shared web content.

- **Goal**: understand best practices for reproducible documents using Quarto
- **Motivation**: cultivate your analyses as living, shared resources

3.2 Quarto

Quarto is a relatively new document preparation system that lets you create reproducible and dynamic content that is easily shared with others. Quarto is integrated with RStudio and allows you to combine plain text language with analysis code in the same document.

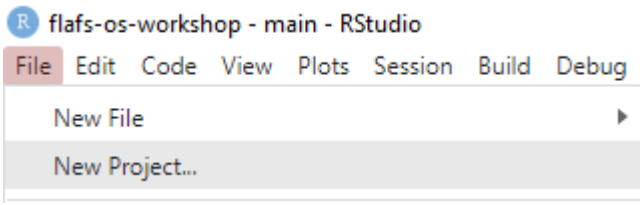
Quarto belongs to a class of reporting tools called dynamic documents or literate programming (Knuth 1984). It is not the first of its kind, but it builds substantially on its predecessors by bridging multiple programming langues.

Advantages of creating analyses using Quarto include:

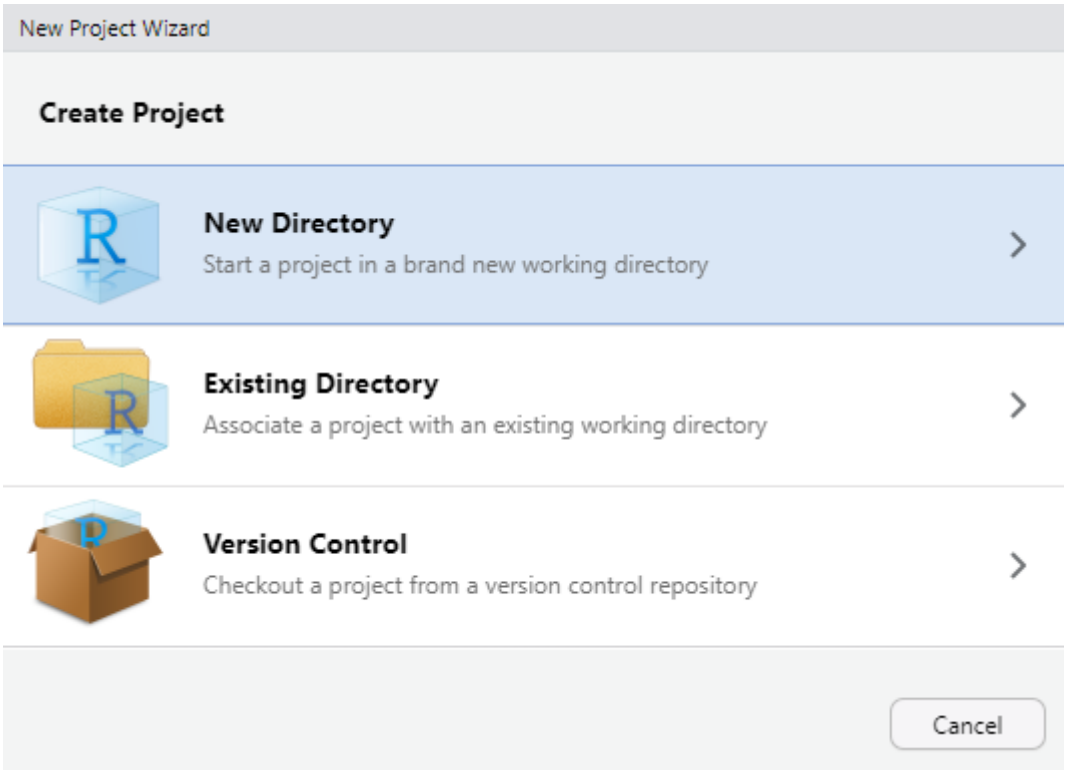
1. Clear demonstration of a workflow using plain text and code
2. Reproducible materials allow others to use your work
3. Easily shared content (e.g., on GitHub)
4. Keeping the data, analysis, and writing all in the same place

This next section will run through the very basics of creating a Quarto document, some of the options for formatting, and how to generate shared content. You'll follow along in this module.

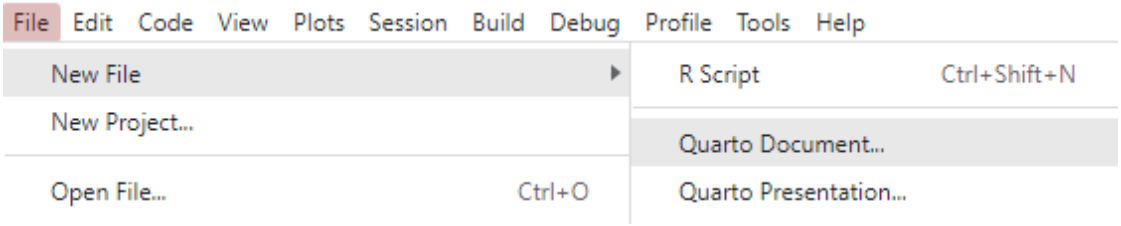
(1) Create a new project in RStudio, first open RStudio and select "New project" from the File menu at the top.



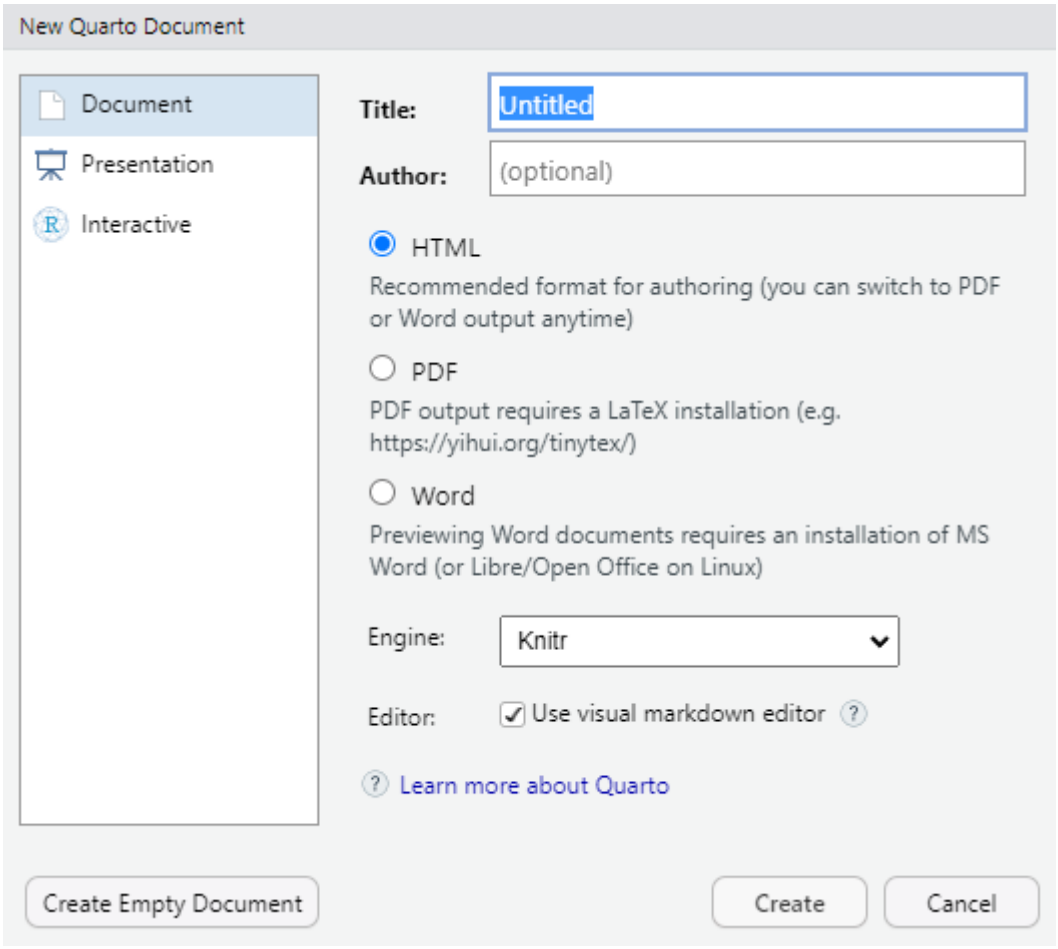
Then select "New Directory". Create a directory in a location that's easy to find.



(2) Open a new Quarto file from the File menu under New file > Quarto Document.



Enter a title for the document (e.g., "Quarto practice") and your name as the author. Use the defaults for the other options and hit "Create".



Save the file in the project root directory (give it any name you want).

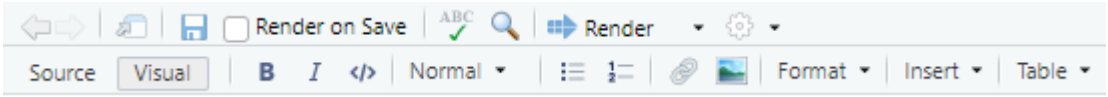
(3) Let's get familiar with the components of a Quarto document.

> **Tip**
>
> The three main components of a Quarto document are:
>
> - YAML
> - Code chunks
> - Plain or Markdown text

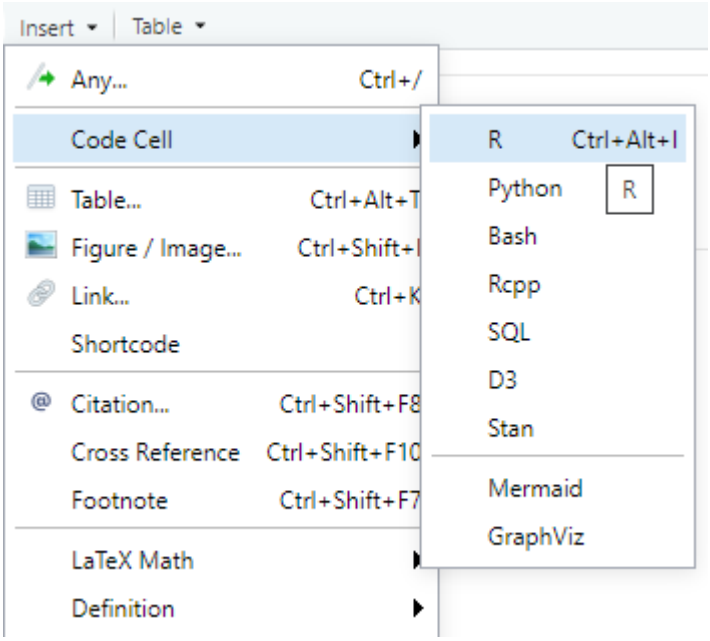
The new file is completely empty except for the title, name, and editor type at the top. The content at the top is called YAML, which defines global options for the document.

```
---
title: "Quarto practice"
author: "Marcus Beck"
editor: visual
---
```

You'll also notice that there's a button on the top-left that lets you toggle between "source" or "visual" editor mode. The source editor simply lets you add text to the document, whereas the visual editor lets you add content that is partially rendered. First time Quarto users may prefer the visual editor.



Using the visual editor, we can insert a code chunk (or code cell). This can be done by selecting the appropriate option from the Insert menu. Note the variety of programming langues that can be used with the code chunk.



We can enter any code we want in the code chunks, including options for how the code chunk is evaluated. Options are specified using the hashpipe notation, `#|`.

```
```{r}
#| echo: true
print('Hello Quarto!')
```
```

```
```

When the file is rendered, the code is run and results displayed in the output. There are many options to change how code chunks are executed, which we'll discuss below.

```
print('Hello Quarto!')
```

```
[1] "Hello Quarto!"
```

We can also run the code chunks separately without rendering the file using the arrow buttons on the top right in the source document. This can be useful for quickly evaluating your code as you include it in the file.

> **Tip**
>
> Code chunks are executed in the order they appear in the document when a .qmd file is rendered.

Descriptive text can be entered anywhere else in the file. This is where we can describe in plain language what our analysis does or any other relevant information. Text can be entered as-is or using simple markdown text that can format the appearance of the output. If you're using the visual editor, you can use some of the items in the file menu to modify the text appearance. In the source editor, you can manually enter markdown text:

When the file is rendered, the markdown text will be formatted. The text will already be formatted if you're using the visual editor:

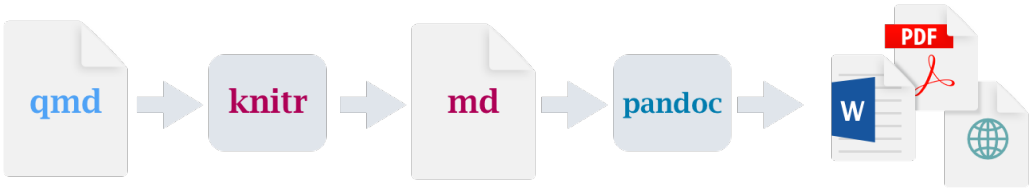
I can write anything I want right here. Here's some **bold text**.

I can also make lists

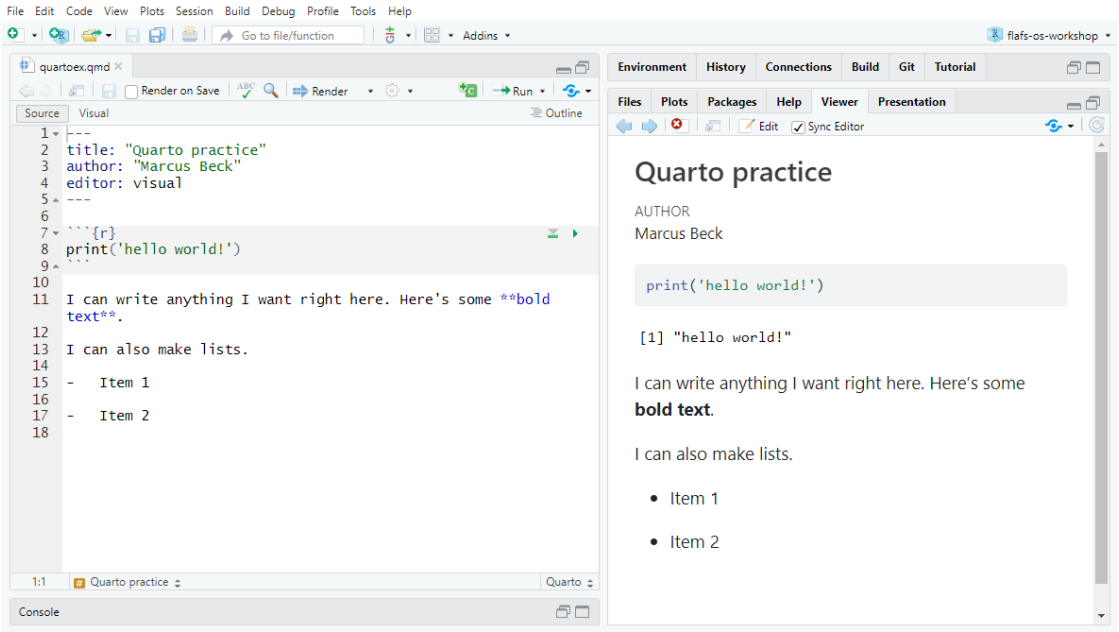
1. Item 1
2. Item 2

(4) Render the .qmd file to the output format.

The source file is a .qmd document. We need to render the document to create the output format - HTML (default), PDF, or Word. The following happens when you hit the render button at the top.



Here's what your RStudio session should look like (note the three parts of the source .qmd document - YAML, code chunk, and Markdown text). The rendered HTML file will appear in the Viewer pane on the right.



> **Tip**
>
> A rendered Quarto document as an HTML, PDF, Word, or other file format is stand-alone and can be shared with anybody!

3.2.1 Code chunk options

The behavior of the code chunks when the file is rendered can be changed using the many options available in Quarto. This can be useful for a few reasons.

1. Only displaying the output of a code chunk
2. Only displaying the code and not running the chunk
3. Running the code without displaying output for use in other parts of the document

4. Suppressing warnings and messages
5. Defining table or figure options (e.g., height, width, captions, etc.)

Code chunk options can be applied **globally** to all chunks in the document or **separately** for each chunk.

To apply them globally, they'll look something like this in the YAML, where options are added after `execute:`

```yaml
---
title: "My Document"

execute:
  echo: false
  warning: false
---
```

> **Tip**
>
> Be careful with indentation in the YAML, the document won't render if the indentation is incorrect.

To apply to individual code chunks, use the `#|` (hashpipe) notation at the top of the code chunk. This will override any global options if you've included them in the top YAML. Below, `echo: true` indicates that the code will be displayed in the rendered output.

````
```{r}
#| echo: true
plot(1:10)
```
````

Here's a short list of other useful execution options:

Option	Description
`eval`	Evaluate the code chunk (if `false`, just echos the code into the output).
`echo`	Include the source code in output
`output`	Include the results of executing the code in the output (`true`, `false`, or `asis` to indicate that the output is raw markdown and should not have any of Quarto's standard enclosing markdown).
`warning`	Include warnings in the output.
`error`	Include errors in the output (note that this implies that errors executing code will not halt processing of the document).
`include`	Catch all for preventing any output (code or results) from being included (e.g. `include: false` suppresses all output from the code block).

Option	Description
`message`	Include messages in rendered output

R code can also be executed "inline" outside of code chunks. This can be useful if you want to include statements that reference particular values or information that is linked directly to data. Inline R code is entered using the `r` syntax.

Text with inline R code will look like this when the document is rendered.

I can enter inline text like 2.

3.2.2 Figures and tables

Figures and tables are easily added in Quarto, using either R code or importing from an external source.

Any figures created in code chunks will be included in the rendered output. Relevant code chunk options for figures include `fig-height`, `fig-width`, `fig-cap`, `label` (for cross-referencing) and `fig-align`.

````
```{r}
#| label: fig-myhist
#| fig-height: 4
#| fig-width: 6
#| fig-cap: "Here's my awesome histogram."
#| fig-align: "center"
vals <- rnorm(100)
hist(vals)
```
````

Figures can be cross-referenced in the text using the `@` notation with the figure label.

```
Here's a cross-reference to @fig-myhist.
```

When the file is rendered, the appropriate figure number will be displayed with a link to the figure:

Here's a cross-reference to Figure 3.1.

Similarly, tabular output can be created inside code chunks.

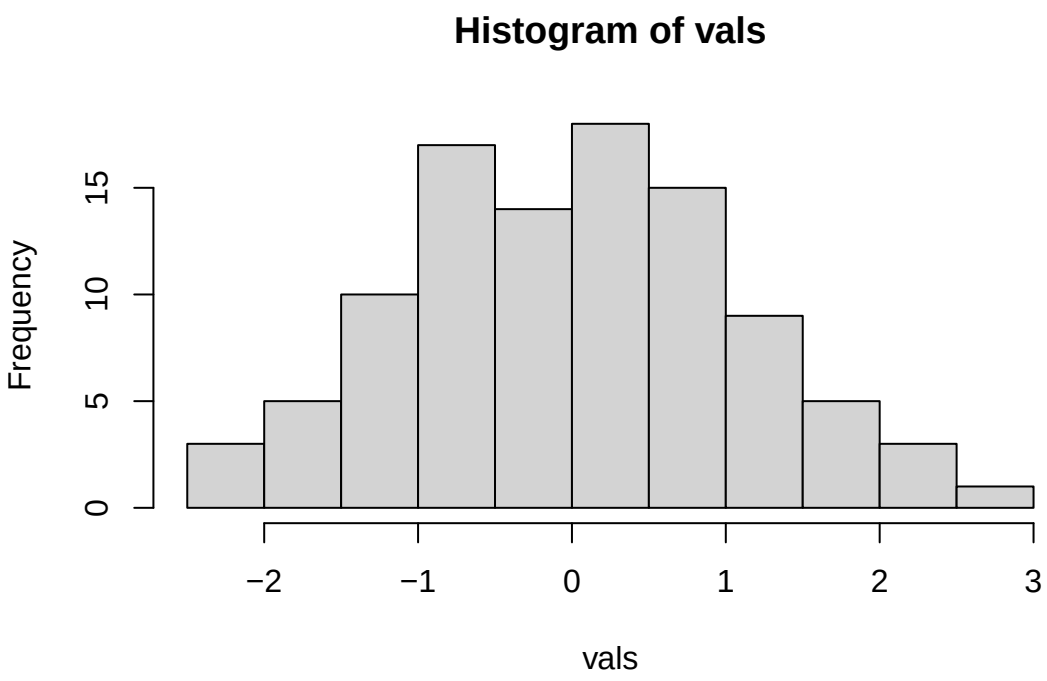


Figure 3.1: Here's my awesome histogram.

```{r}
#| label: tbl-mytable
#| tbl-cap: "Here's my awesome table."
totab <- data.frame(
  Species = c('Bluegill', 'Largemouth bass', 'Crappie'),
  Count = c(12, 5, 4)
)
knitr::kable(totab)
```

Table 3.2: Here's my awesome table.

Species	Count
Bluegill	12
Largemouth bass	5
Crappie	4

And a cross-reference:

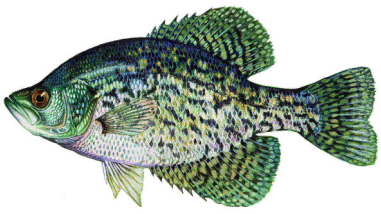
```
Here's a cross-reference to @tbl-mytable.
```

Here's a cross-reference to Table 3.2.

> **Tip**
>
> Label tags for tables and figures should include the `tbl-` or `fig-` prefix for proper cross-referencing.

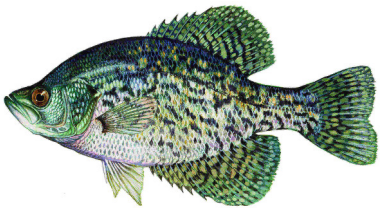
Figures can also be imported from an external source (e.g., from your computer or the web) using the `![]()` notation, where the image is in the `img` folder in my working directory. You can also simply add a figure from the file menu using the Visual editor.

```
![](img/blackcrappie.jpg)
```



You can also add a figure from a URL using the same notation.

```
![](https://www.bigcatchflorida.com/media/1174/blackcrappie.jpg)
```



Adding captions and labels to external figures looks something like this:

```
![Here's a beautiful crappie](img/blackcrappie.jpg){#fig-crappie}
```

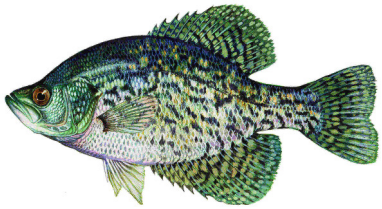


Figure 3.2: Here's a beautiful crappie

The cross-reference is done the same.

```
Here's a cross-reference to @fig-crappie
```

Here's a cross-reference to Figure 3.2.

Likewise, tables can be imported from an external source (e.g., Excel). You'll want to do this in a code chunk and add the appropriate options (e.g., to cross-reference Table 3.3).

````
```{r}
#| label: tbl-habitats
#| tbl-cap: "The first six rows of our tidy data"
mytab <- readxl::read_excel('data/tidy.xlsx')[1:6, ]
knitr::kable(mytab)
```
````

Table 3.3: The first six rows of our tidy data

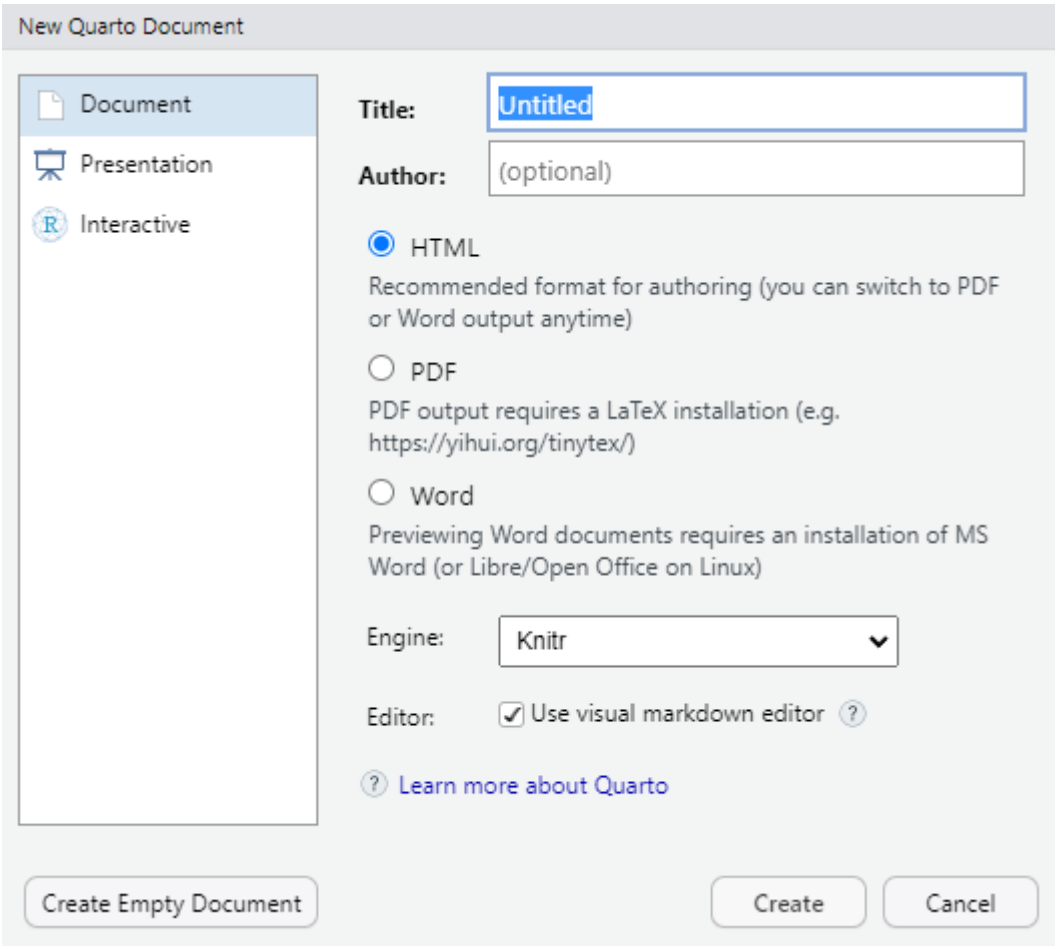
Location	Habitat	Year	Acres	Category
Clear Bay	Seagrass	2019	519	B

Location	Habitat	Year	Acres	Category
Clear Bay	Oysters	2019	390	B
Clear Bay	Sand	2019	742	C
Fish Bay	Seagrass	2019	930	B
Fish Bay	Oysters	2019	680	A
Fish Bay	Sand	2019	611	A

Visit these links for full details on figures and tables in Quarto. R also has a rich library of packages for producing tables, most of which play nice with Quarto.

3.2.3 Output options

Rendering a Quarto file to an HTML, PDF, or Word document is as simple as adding the appropriate option to the YAML. This is done by choosing the format when you create a new Quarto file:



The output format can also be added in the YAML of the document.

```
---
title: "Quarto practice"
author: "Marcus Beck"
editor: visual
format: html
---
```

> **Tip**
>
> The default output format is HTML and it does not need to be added explicitly to the YAML.

Alternative formats are specified the same way (i.e., Word and PDF).

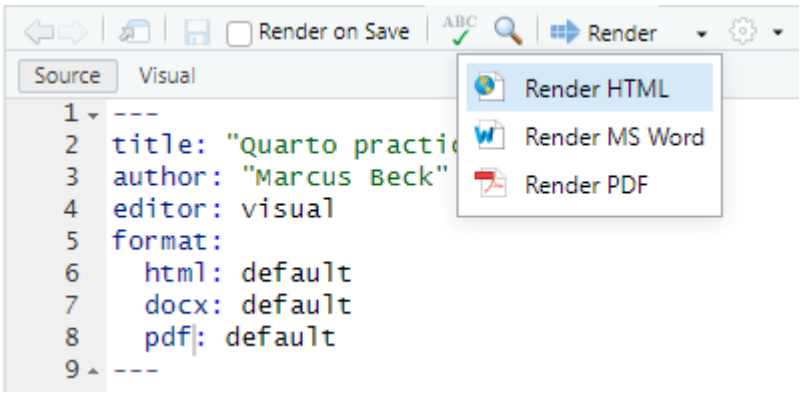
```yaml
---
title: "Quarto practice"
author: "Marcus Beck"
editor: visual
format: docx
---
```

```yaml
---
title: "Quarto practice"
author: "Marcus Beck"
editor: visual
format: pdf
---
```

You can also specify multiple formats (note the indentation). The `default` setting just indicates that we want to use all default options for each format and this option must be included.

```yaml
---
title: "Quarto practice"
author: "Marcus Beck"
editor: visual
format:
  html: default
  docx: default
  pdf: default
---
```

You can use the dropdown menu to render each file format one at a time. The dropdown menu will show the options that are included in the YAML.



You can also render all formats at once using the `quarto` package in the console. The path to

your file will differ depending on where it is in your working directory.

```
quarto::quarto_render('data/quartoex.qmd')
```

Or you can render all formats at once using the `render` command in the terminal (terminal tab, bottom left pane of RStudio). This requires the separate installation of Quarto described in the setup. This is the Quarto Command Line Interface (CLI).

Listing 3.1 `Terminal`

```
quarto render data/quartoex.qmd
```

Rendering a file to PDF uses LaTeX and you'll need to install tinytex before you can use this option. This can also be done in the terminal.

Listing 3.2 `Terminal`

```
quarto install tool tinytex
```

There are several options you can include in the YAML to control the formatting of the output. Some of the options apply to all format types, whereas others are specific to a type. Here's an example building out these options.

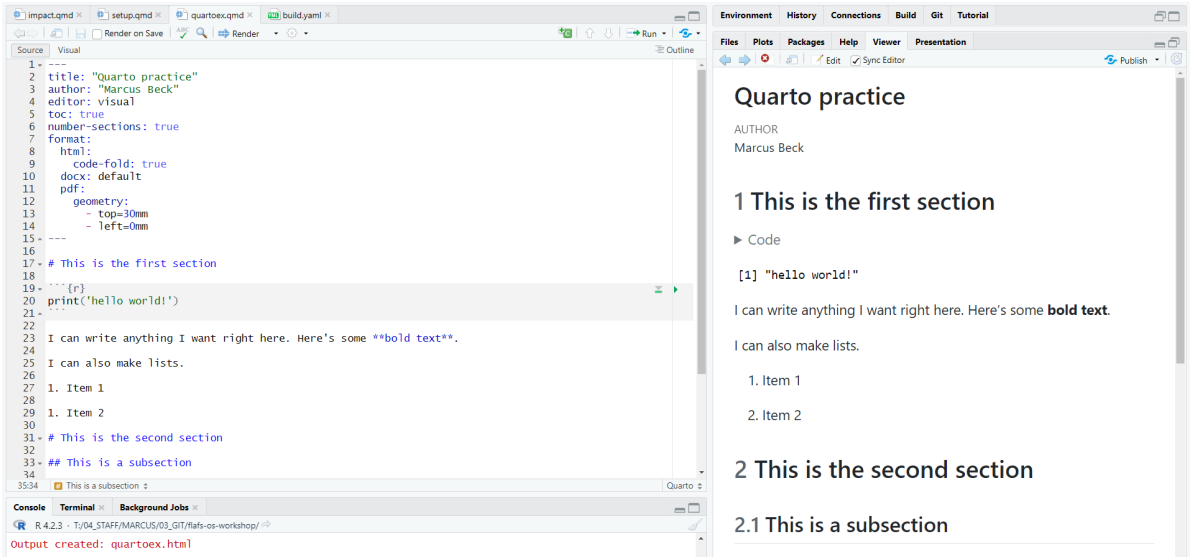
```
---
title: "Quarto practice"
author: "Marcus Beck"
editor: visual
toc: true
number-sections: true
format:
  html:
    code-fold: true
  docx: default
  pdf:
    geometry:
      - top=30mm
      - left=0mm
---
```

In the above example, there are two new options that apply globally to the HTML, Word, and PDF outputs. Specifically, we've indicated that we'd like a table of contents (`toc: true`) and

that the sections should be numbered (`number-sections: true`) in the rendered documents.

We've also added some specific options to the HTML and PDF output. For the HTML output, we've indicated that we want the code chunks to be folded (i.e., toggle between seen and not seen, `code-fold: true`). For the PDF output, we've changed the geometry of the margins using the `geometry` options.

Here's what the HTML output would look like:



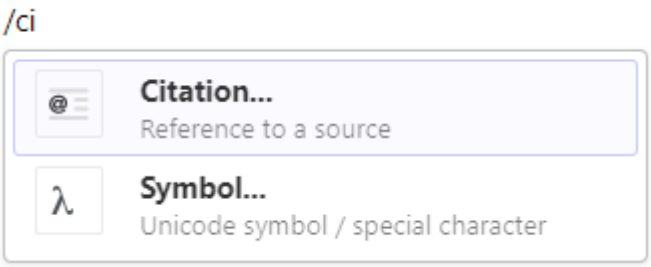
There are many other options available for each output format, as well as other format types. View the full list here.

3.2.4 Citations and References

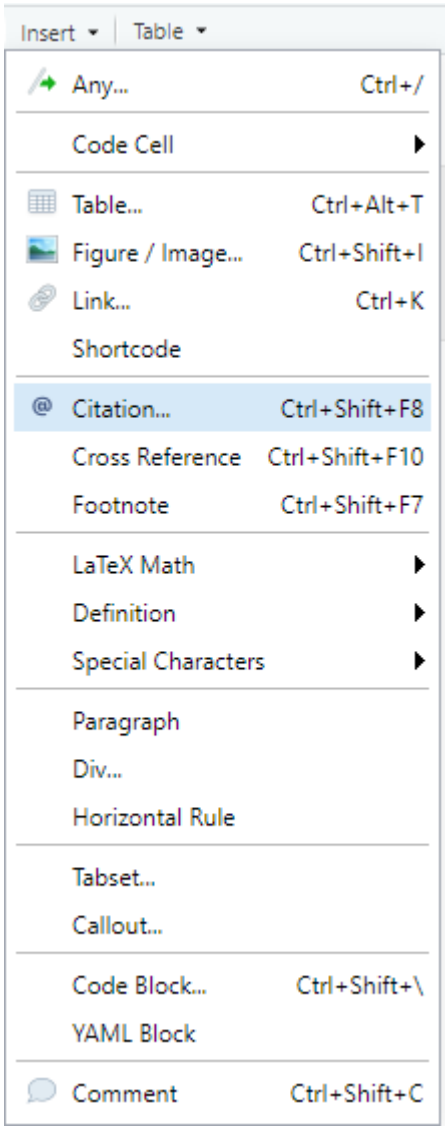
One of the more valuable aspects of Quarto is the ability to easily add and reference other works in your document. This includes finding papers and reports, citing them in your document, and formatting references - all with relative ease in Quarto.

You can of course do this using the source editor, but it's slightly easier using the visual editor. If we switch to visual mode (top-left button of the .qmd file), you can type a forward-slash to view a menu of items to insert in the document. Just start typing text to search items to insert.

> **Tip**
>
> The Insert Anything tool in the visual editor is useful to... insert anything! Just execute `/` at the beginning of a line or `Ctrl/Cmd + /` after some text.

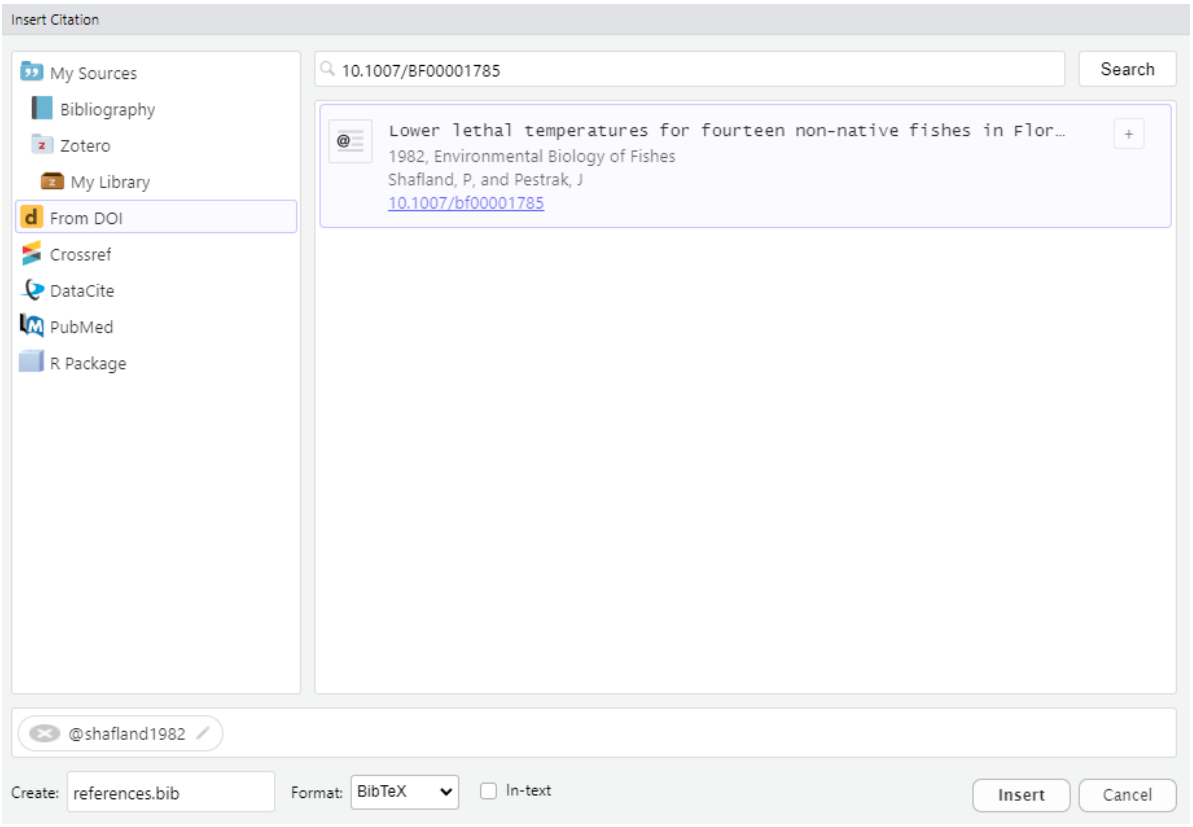


You can also insert a citation from the menu at the top of the .qmd file.

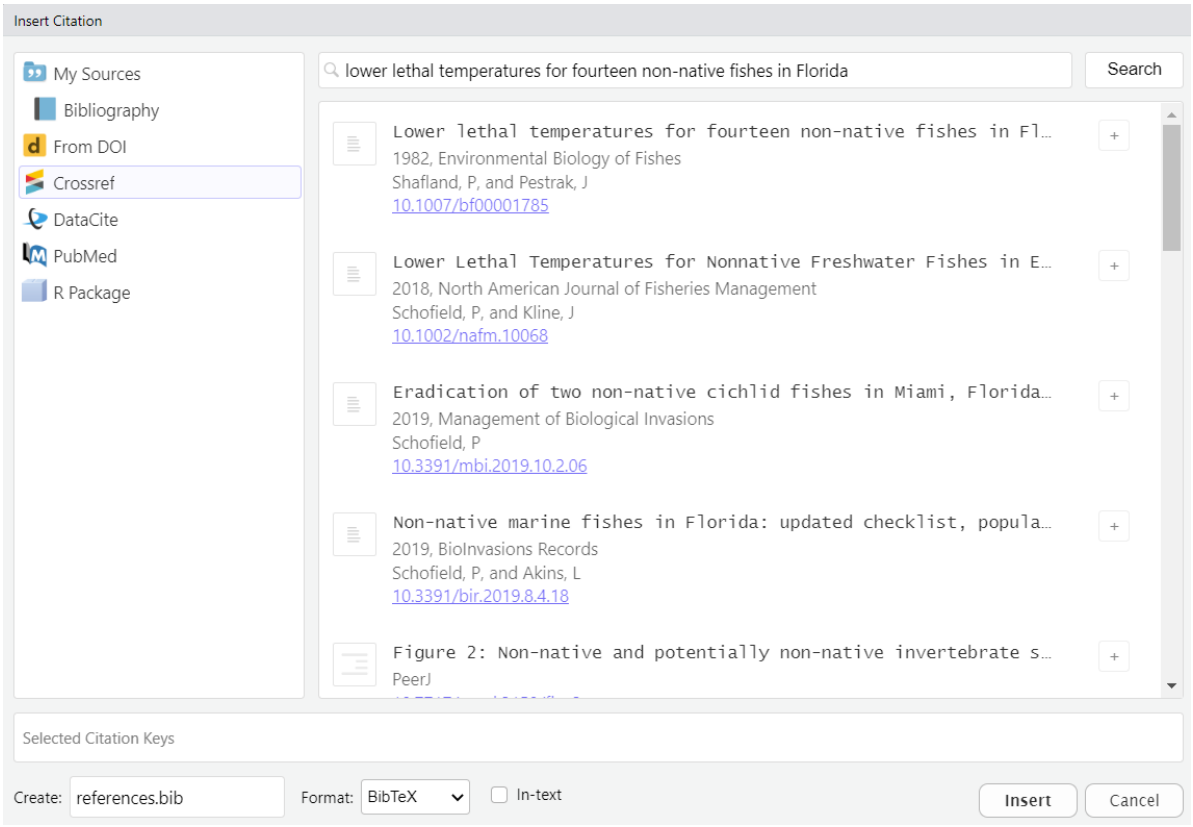


Either option will open the citation menu where you can add citations from a variety of sources (ie., Zotero, DOI, CrossRef, PubMed, or DataCite).

For example, we can copy/paste a DOI to find a reference of interest.



Or we can search by title.



Once the paper is found, you can click the Insert button to add it to your document. This adds a reference file, information in the YAML, and the in-text citation. The reference file will be called `references.bib` by default and includes a BibTeX formatted reference that looks like this:

```
@article{shafland1982,
    title = {Lower lethal temperatures for fourteen non-native fishes in Florida},
    author = {Shafland, Paul L. and Pestrak, James M.},
    year = {1982},
    month = {03},
    date = {1982-03},
    journal = {Environmental Biology of Fishes},
    pages = {149--156},
    volume = {7},
    number = {2},
    doi = {10.1007/bf00001785},
    url = {http://dx.doi.org/10.1007/BF00001785},
    langid = {en}
```

```
}
```

The YAML file will now indicate the reference file to use that includes the references (`bibliography: references.bib`).

```
---
title: "Quarto practice"
author: "Marcus Beck"
editor: visual
bibliography: references.bib
toc: true
number-sections: true
format:
  html:
    code-fold: true
  docx: default
  pdf:
    geometry:
      - top=30mm
      - left=0mm
---
```

The text citation will look like this, where `@` is the tag used to reference the citation using the identifier from the references file.

```
Many non-native species in Florida have lower lethal temperatures [@shafland1982].
```

When the Quarto file is rendered, the citation will be formatted and you'll see it added to the references section at the end of the document.

Many non-native species in Florida have lower lethal temperatures (Shafland and Pestrak 1982).

References

Shafland, Paul L., and James M. Pestrak. 1982. "Lower Lethal Temperatures for Fourteen Non-Native Fishes in Florida." *Environmental Biology of Fishes* 7 (2): 149–56. https://doi.org/10.1007/bf00001785.

The `@` citation syntax also has different options for displaying the citation in the text (full explanation here). For example, omitting the brackets does the following:

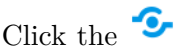
```
@shafland1982 state that many non-native species in Florida have lower lethal temperatures
```

Shafland and Pestrak (1982) state that many non-native species in Florida have lower lethal temperatures.

Additional information about citations in Quarto can be found here.

3.2.5 Publishing

A rendered Quarto file can be shared with anyone as a standalone document. The file can also be hosted online and shared by URL. This latter approach is useful to make the document available to anyone with the web address.

The easiest way to do this is to publish your document to RPubs, a free service from Posit for sharing web documents. Click the  publish button on the top-right of the editor toolbar. You will be prompted to create an account if you don't have one already.

This can also be done using the `quarto` R package in the console.

```
quarto::quarto_publish_doc(
  "data/quartoex.qmd",
  server = "rpubs.com"
  )
```

You can also use the Quarto CLI in the terminal. Here we are publishing the document to Quarto Pub.

Listing 3.3 `Terminal`

```
quarto publish quarto-pub data/quartoex.qmd
```

If your Quarto document is in an RStudio project on GitHub, you can also publish to GitHub Pages.

Listing 3.4 `Terminal`

```
quarto publish gh-pages data/quartoex.qmd
```

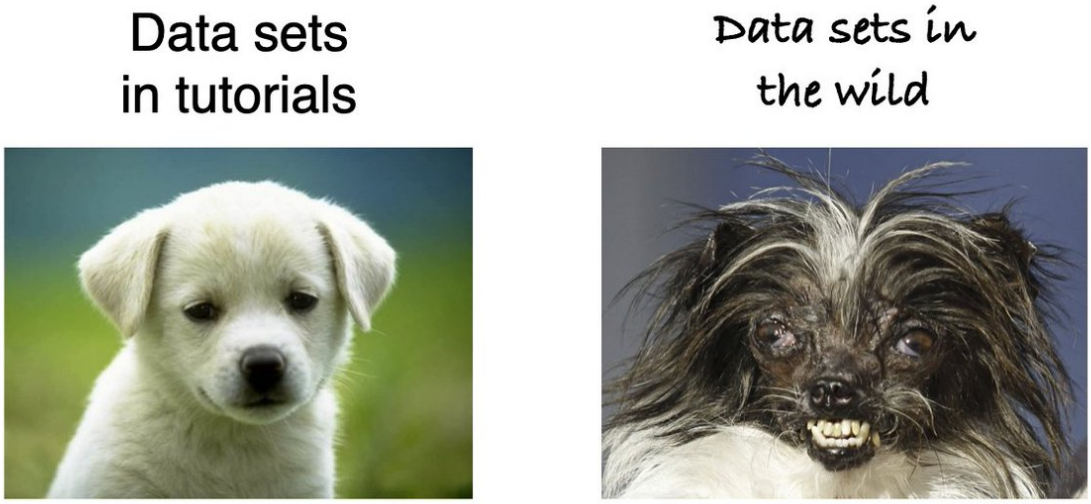
3.3 Summary

In this module we learned the basics of creating dynamic documents with Quarto that combine markdown text with R code. There's much, much more Quarto can do for you. Please visit https://quarto.org/ for more information on how you can use these documents to fully leverage their potential for open science.

Part II

Extra modules

4 Principles of tidy data



Tabular data allow you to store information, where observations are in rows and variables are in columns. It's very common to try to make tabular data more than it should be. Unless you spend a lot of time working with data, it can be difficult to recognize common mistakes that lead to *table abuse*.

Before we get into tidy data, we need to discuss some of the downfalls of Excel as a data management system. There are many examples that demonstrate how Excel has contributed to costly mistakes through table abuse or outright negligence, often to the detriment of science (Ziemann, Eren, and El-Osta 2016).

Excel allows you to abuse your data in many ways, such as adding color to cells, embedding formulas, and automatically formatting cell types. This creates problems when the organization is ambiguous and only has meaning inside the head of the person who created the spreadsheet. Embedding formulas that reference specific locations in or across spreadsheets is also a nightmare scenario for reproducibility.

slate
@PleaseBeGneiss

excel: is that a date?

me: 57.39 is very much not a date

excel: strong date vibes to me

me: h-how

excel: fixed it

me: 57/39/2020?

excel: you're welcome

	A	B	C	D	E	F
1	OTBT gmtpmt& mnpmt					
2		OTBT Rzpsmr	OTBT N/B	Tsct OTBT NB		
3		11434	64037	200000		
4	Osrkzt vglmOzs mnpmt					
5		Lsst 12 Ognths sslzs snd rzpsmr zxpgsmrz pzrmgd cslcmlstmgn				
6		Tgtsl bssz	svzrsgz rmntmOz	BzTgrz 12 O	R/Onth -1	R/Onth -2
7	yslzs bmmlt:	1412	11.97	1400	0	0
8	Tgtsl rzpsmr bmmlt:	306	4.36	0	2	23
9	ATT bmmlt mnpmt	272	4.01	0	1	15
10	Bzndgr 2 bmmlt mnp	0	#DIV/0!	0	0	0
11	Bzndgr 3 bmmlt mnp	34	7.18	0	1	8
12	RzAsmR mNAUT					
13			Rzpsmr mssgz		Rzpsmr pgpmlstmgn	
14		Tgtsl Rzpsmr	Rzpsmrzd DOs	Rzpsmrzd Wsrr	NgrOsl Rzpsmr	Lgcsl Rzpsmr
15		306	2	40	264	0
16	Argdmct mnTgrOstmgn				% cmrrznt mnstsllzd bssz sTtzr Tmtmrz ssl	
17		Argdmct NsOz	ytylz NsOz	Rzvmzw Dstz	Ognths sctmvz	svzr Tmtmrz rmntmOz
18		DzhmOmdmī	sB6065903CT	25/06/1998	76.8	27
19	Asrt NmObzr mnTgrOstmgn			Ognth/dsy/yzsr		
20		sDy Asrt NmObz	yzt-mp Dstz	ytsndsrd cgst	Ognthly mssgz	12 Ognth's mssgz
21		43588136-00.	1/02/1992	675.92	9.7	116.4
22			Ognth/dsy/yzsr			
23						

If you absolutely must use Excel to store data, the only acceptable format is a rectangular, flat file. This is typically saved as a *.csv* file. What do we mean by this?

A **rectangular** file:

> Store data only in rows and columns in matrix format (e.g., 10 rows x 5 columns), with no "dangling" cells that have values outside of the grid or more than one table in a spreadsheet.

A **flat** file:

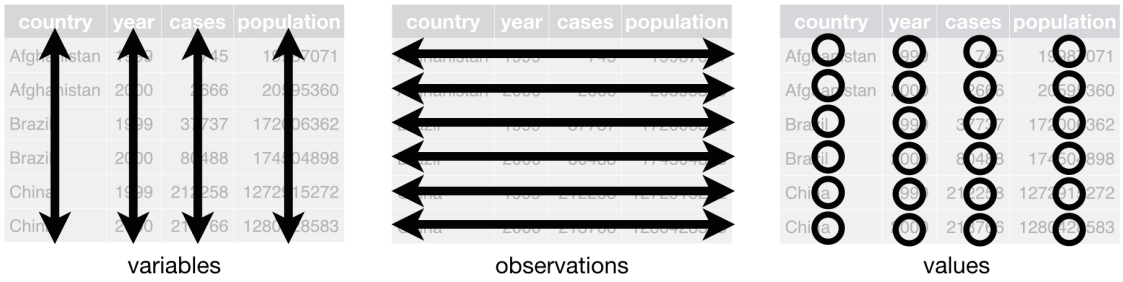
> No cell formatting, no embedded formulas, no multiple spreadsheets in the same file, and data entered only as alphanumeric characters.

Broman and Woo (2018) provide an excellent guide that expands on these ideas. Essentially, these best practices force you to isolate the analysis from the data - many people use Excel to mix the two, leading to problems.

Now we can talk about tidy data. The tidy data principles developed by Hadley Wickham (Wickham 2014) are a set of simple rules for storing tabular data that have motivated the development of the wildly popular tidyverse suite of R packages (Wickham et al. 2019). The rules are simple:

1. Each variable must have its own column;
2. Each observation must have its own row; and,
3. Each value must have its own cell.

Graphically, these rules are shown below (from Wickham and Grolemund 2017):



The following examples show five tables represented in different arrangements. Only one of the tables is tidy - which one?

Only the first table is tidy - each variable has its own column, each observation has its own row, and each value has its own cell. Table 2 violates the first rule, Table 3 violates the third rule, and tables 4a and 4b violate the first and second rules.

Habitat	Year	Acres	Category
Seagrass	2019	519	B
Oysters	2019	390	B
Sand	2019	742	C
Seagrass	2020	438	C
Oysters	2020	875	B
Sand	2020	702	A
Seagrass	2021	375	A
Oysters	2021	724	A
Sand	2021	505	C

Figure 4.1: Table 1

Habitat	Year	Attribute	Value
Seagrass	2019	Acres	519
Seagrass	2019	Category	B
Oysters	2019	Acres	390
Oysters	2019	Category	B
Sand	2019	Acres	742
Sand	2019	Category	C
Seagrass	2020	Acres	438
Seagrass	2020	Category	C
Oysters	2020	Acres	875
Oysters	2020	Category	B
Sand	2020	Acres	702
Sand	2020	Category	A
Seagrass	2021	Acres	375
Seagrass	2021	Category	A
Oysters	2021	Acres	724
Oysters	2021	Category	A
Sand	2021	Acres	505
Sand	2021	Category	C

Figure 4.2: Table 2

Habitat	Year	Acres and Category
Seagrass	2019	519/B
Oysters	2019	390/B
Sand	2019	742/C
Seagrass	2020	438/C
Oysters	2020	875/B
Sand	2020	702/A
Seagrass	2021	375/A
Oysters	2021	724/A
Sand	2021	505/C

Figure 4.3: Table 3

Habitat	2019	2020	2021
Oysters	B	B	A
Sand	C	A	C
Seagrass	B	C	A

Figure 4.4: Table 4a

Habitat	2019	2020	2021
Oysters	390	875	724
Sand	742	702	505
Seagrass	519	438	375

Figure 4.5: Table 4b

◢ Exercise and discussion

Download this untidy dataset and make it tidy using your preferred software.

5 Addressing implementation barriers

5.1 Goals and motivation

What does it mean to use open science in the real world? It's great to talk about the value of open science and the tools you can use, but it's a completely different ball game when it comes to putting these ideas into practice. Our goal is that you leave this workshop an advocate and early adopter for the ideas we discussed today - spread these ideas to your peers and colleagues! To realistically achieve this goal, we will talk about some of the challenges you will face so you can develop a realistic expectation of what's to come.

- **Goal**: Understand common hurdles in adopting open science and how to overcome them
- **Motivation**: Become the "open science" expert at your institution!

5.2 Learning curves

Challenge

It's hard to learn new tools!

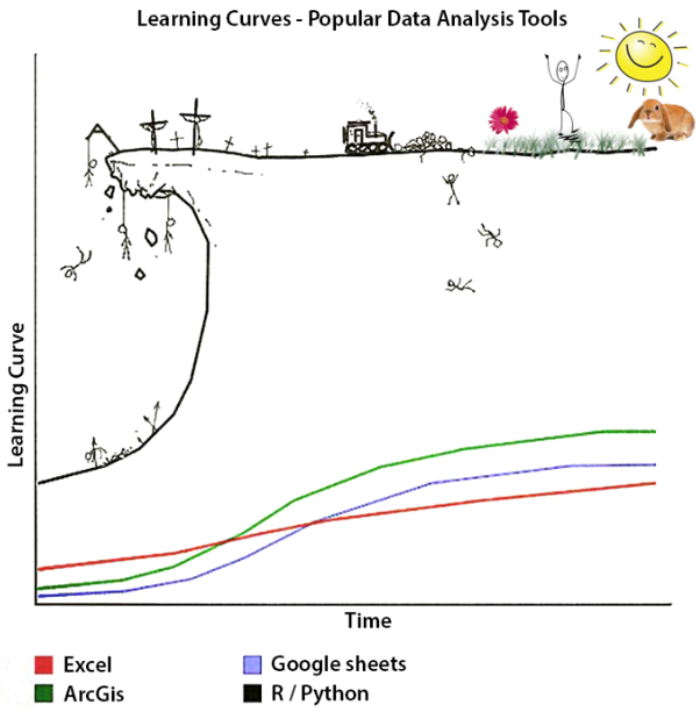
Solution

It's an investment, look to the community!

You've probably seen a graphic like this if you've ever taken a course in R or Python. The hope is that you're able to quickly reach the land of sunshine and bunnies, but the path is treacherous and even insurmountable for some.

A huge obstacle in using open science is that the toolsets can have steep learning curves. More popular platforms, such as Excel, are used by many because they're simple and intuitive. However, as noted earlier, FAIR workflows and tools are sacrificed for ease of use.

Although it's true that adopting new tools will slow forward progress, this is only temporary. Consider your path towards learning new platforms an investment in your future. The immediate benefit may not be apparent, but you'll soon wonder how you ever got by before.



It's also helpful to think about the broader community that can support you along this journey. Learning alone can be discouraging and we strongly recommend that you tap into the diverse community of educators, mentors, bloggers, and friends that can help. Even you can create a community of practice!

Exercise and discussion

How can you engage your peers to develop a shared workspace to learn new tools? What tools will you learn?

5.3 Fear of exposure

Challenge

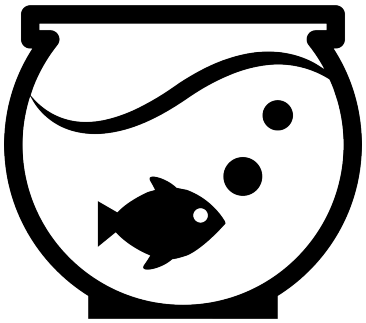
Being open makes me nervous!

Solution

Being open helps you collaborate, increases competitiveness, and creates a better scientific product!

Practicing open science can feel like science in a fish bowl. Although this is kind of the goal, many view this transparency as a liability. Many fear having their ideas "scooped" or


maintain
mentors!
friends
developers!
teachers!
bloggers!
contrib
i'm doing a thing all on my own!
code hero
support
community
@allison_horst



losing credibility because of greater exposure of mistakes. These are real concerns that require consideration when working towards more open workflows.

In conventional academic settings, competition for resources (e.g., via grant funding) is a real issue and being open can be seen as a risk to the competitive edge. We cannot dismiss this fact, but rather we can think about a lack of openness as a hindrance to forward progress and stifled creativity.

Think about being open as a means to finding your next collaborator. Creating FAIR data opens the door for others to engage with your science. In fact, being open can increase the competitiveness of research proposals by building a stronger team that collaborates and shares data through better workflows.

First time practitioners of open science also worry about the risk of "airing their dirty laundry". By exposing the process and potential mistakes, many worry that their integrity as scientists may be questioned.

These fears are unfounded as the scientific process by definition is iterative. Hypotheses are supported or refuted through trial and error - if you're getting your answer after one pass, you're probably not doing it right. Making the process more transparent can help build trust as your collaborators can better appreciate how decisions and conclusions were made.

Mistakes in research are also very common, much more so than many people realize. By being open, it is true that mistakes are more visible, but this also provides a mechanism for fixing. Being open can lead to a better product by simply having more eyes on the process. It also helps normalize mistakes as part of the process - perfection is an unrealistic expectation.

Exercise and discussion

What are your personal concerns about adopting open science?

5.4 What does it mean to be open?

Challenge

People and institutions define open differently!

Solution

Understand the context and demonstrate the value!

Also realize that open science can mean different things to different people. By extension, this also applies to institutions. We presented the five schools of open science to help conceptualize ideas and tools when we discuss what it means to different groups.

Think about your employer and what they might care about if you advocate for adoption of open science. Do you need to convince them that there is value in being open? What is their value proposition? What are the hurdles to achieving openness at your institution?

For many institutions, being open may come with IT hurdles as you push for alternative software platforms. Working with IT staff to develop trust and comfort for new software may be your burden, but as always, it's an investment in the future.

Maybe there are legal contexts to being open. For example, Florida has the "Sunshine" law that makes all government communications public record. What does this mean for using new workflows in open science? Is this is an improvement or a liability (see previous section)?

If you're an administrator or manager, maybe you're the one that makes the call about being open. It's important for you to create a culture that promotes and supports open science. Allow space and time for your staff to learn new skills. Realize that investing time in open science is an investment in the future.

Exercise and discussion

What does being open mean to you? What do you think being open means to your employer?

5.5 Something is better than nothing

Challenge

Doing all the things is impossible!

Solution

Start small, incremental progress is the name of the game!

First time open science enthusiasts can be overwhelmed by the apparent need to check all the boxes on the open science list. There's often a prevailing sentiment that you're not doing open science unless you do all the things. This is simply not true. Just remember that doing something is a huge improvement over doing nothing.

Openness in science exists on a spectrum. Your goal should be incremental movement away from the completely closed end of the spectrum. Perhaps you set a goal of only accomplishing one open science task for a particular project. Maybe you start by developing a simple metadata text file or developing a data dictionary. Or maybe you make a commitment to try a new communication platform for collaborative engagement.

Channeling this concept, Wilson et al. (2017) discuss "good enough practices" in scientific computing, acknowledging that very few of us are professionally trained in these disciplines and sometimes "good enough" is all we can ask for. Lowenberg et al. (2021) also advocate for simple adoption, rather than perfection, when it comes to data citation practices.

Also, be mindful of complacency (and apathy, at its very worst). Just because you think you've mastered a task doesn't mean you can't continue to learn. Always strive to improve yourself and the tools you use to be open. The fact that the toolbox is constantly evolving makes this a necessity.

So, be kind to yourself when learning new skills and realize that the first step will likely be frustration, but through frustration comes experience. The more comfortable you become with a task, the more likely you'll attempt new tasks in the future. I promise you will see a return on your investment.

◢ Exercise and discussion

What are some simple things you can do to begin adopting open science?

6 Additional tools for collaboration

Below we introduce some web-based tools that you can use to improve collaboration and openness. We present them as a suite of options to consider based on the pros and cons associated with each tool. This is by no means a comprehensive list, but it should get you started towards better collaboration in an open environment.

6.1 Slack

https://slack.com/

What

An online messaging platform for internal communication. Conversations can be organized by topic (via channels) or you can send direct messages to one or more team members. You can have multiple workspaces for different groups.

Pros

Alleviate email overload through quick, informal messaging. Offers a fresh approach to online communication.

Cons

Yet another thing to monitor. Free subscription limits archive of messages. Communication is limited to those in the same workspace.

6.2 Trello

https://trello.com/

What

A Kanban style workflow organization platform. Can be used for personal organization or in teams. Card management allows you to assign due dates, add attachments, make checklists, assign tasks to yourself or team members, and label by themes.

Pros

Easy to use and can upgrade with "power-ups" for integration with other services (e.g., Google). Use across locations (e.g., from home or in the office) is easy because it's based in a web browser.

Cons

Not entirely open because it's only visible to yourself or those you explicitly invite. Free version is limited to only a handful of "power-ups".

6.3 Google Drive

https://google.com/drive

What

Cloud-based platform for sharing documents, worksheets, slides, etc. Follows a familiar file-based structure that is common to most operating systems.

Pros

Easy to use and can be a very open space for collaboration. Fairly interoperable with different file formats. Some functionality with version control (i.e., ability to "revert" to previous versions and to view changes).

Cons

Requires a Google account and access can be tricky depending on institution. Even though some versioning is provided, the format can encourage poor file management. Who knows what Google is doing with your data.

6.4 Office 365

https://www.microsoft.com/en-us/microsoft-365

What

Cloud-based platform for secure sharing of Microsoft documents, worksheets, slides, etc.

Pros

Easy to use and fully supports Microsoft products. Low barrier of inclusion to others that are already using Microsoft products.

Cons

Requires a Microsoft account and access can be tricky depending on institution. Maintains dependency on expensive Microsoft products that aren't reproducible or interoperable. Very often used in closed workflows.

6.5 GitHub

https://github.com

What

Cloud-based platform for sharing code with Git version control. Supports sharing of most file types, although code and text-based files are the primary use.

Pros

Collaborative and fully transparent work environment for files under version control. Supports workflow management through issue tracking and Kanban style project boards. Links to third-party platforms for archiving and DOI generation (e.g., Zenodo). Octocat mascot is super cute.

Cons

Learning curve is steep if you want to fully leverage version control. Not a formal data archival service by itself and file sizes are limited.

A Setup for the workshop

Thanks for your interest in the open science workshop. You will need to do the following, outlined below, before the workshop. The last item is optional, but strongly encouraged.

1. Install R: link
2. Install RStudio: link
3. Install Quarto: link
4. GitHub create account: link
5. Install Git (optional): link

Most of these steps will require administrative privileges on a computer. Work with your IT staff to complete the setup if you do not have these privileges. Please reach out if you have any issues with installation: mbeck@tbep.org

A.1 Install R and RStudio

R and **RStudio** are separate downloads and installations. R is the underlying statistical computing software. RStudio is a graphical integrated development environment (IDE) that makes using R much easier and more interactive. *You need to install R before you install RStudio.*

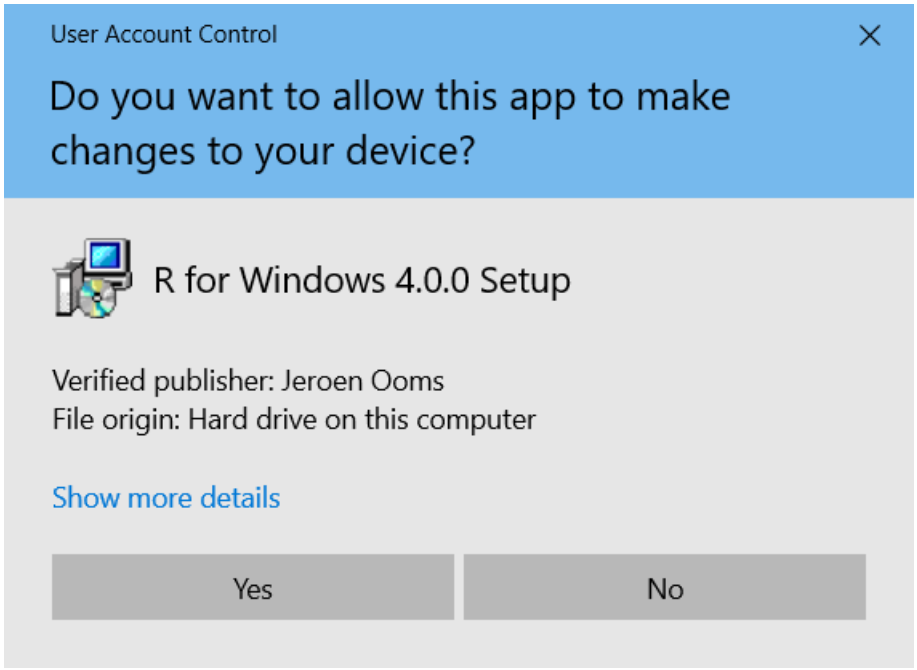
Thanks to the USGS-R Training group and Data Carpentry for making their installation materials available. The following instructions come directly from their materials, with a few minor edits to help you get set up.

A.1.1 Windows: Download and install R

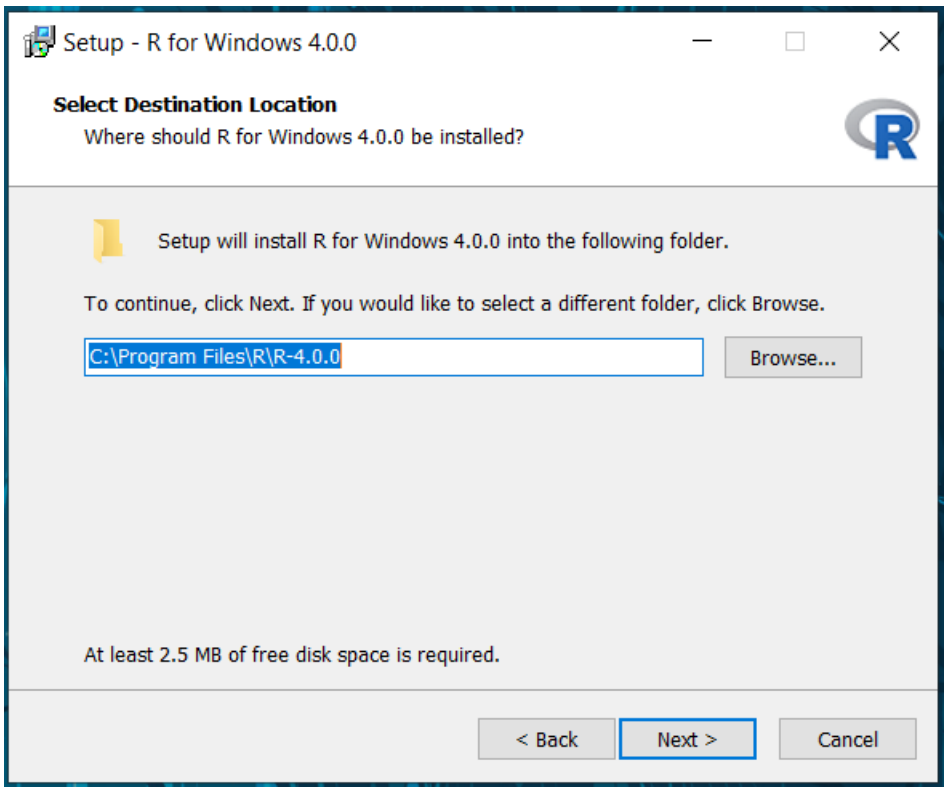
Go to CRAN and download the R installer for Windows. Make sure to choose the latest stable version (v4.2.3 as of April 2023).

Once the installer downloads, Right-click on it and select "Run as administrator".

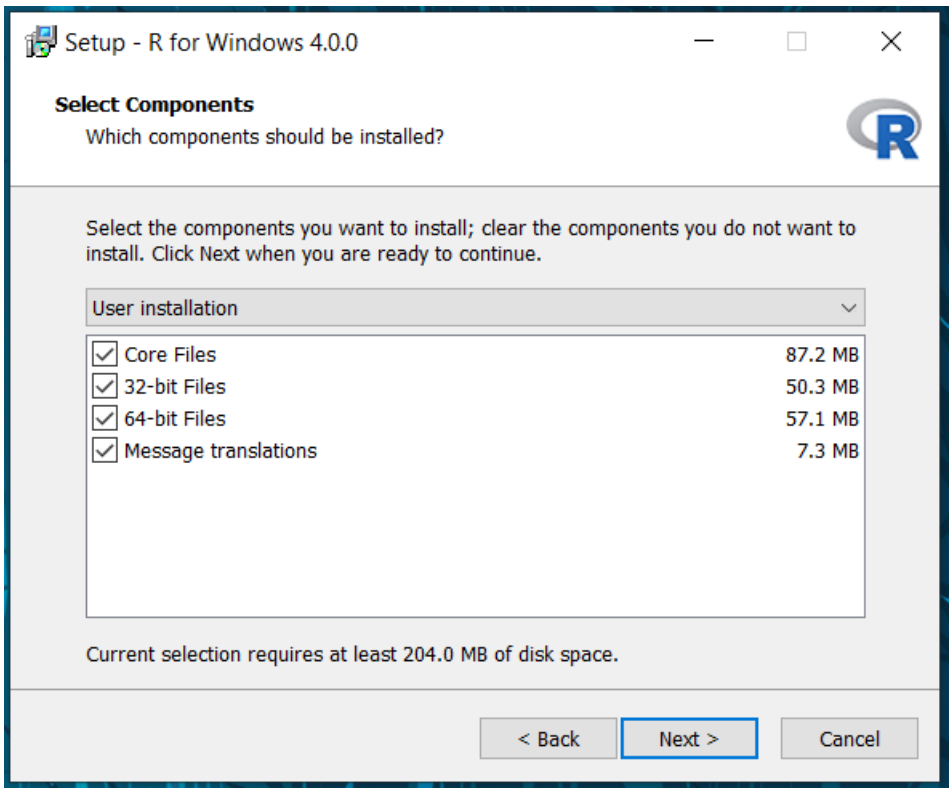
Type in your credentials and click yes (or if you don't have administrator access have your IT rep install with Admin privileges).



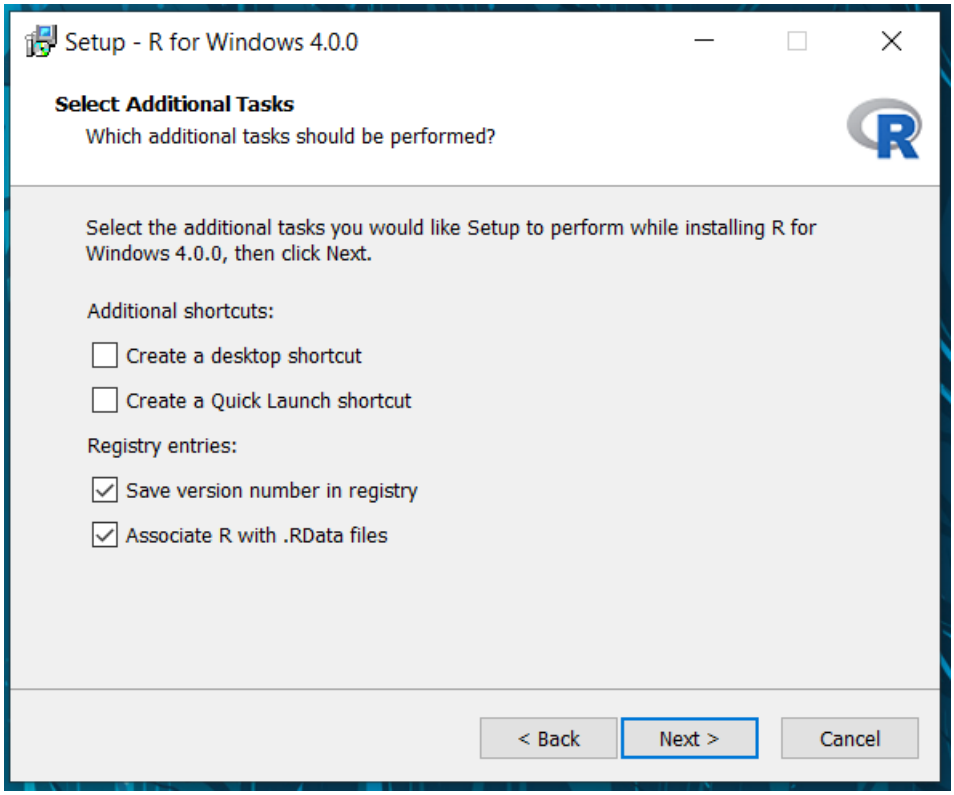
You can click next through the standard dialogs and accept most defaults. But at the destination screen, please verify that it is installing it to C:\Program Files\R



At the "Select Components" screen, you can accept the default and install both 32-bit and 64-bit versions.



At this screen, uncheck 'Create a desktop icon' because non-admin users in Windows will be unable to delete it.

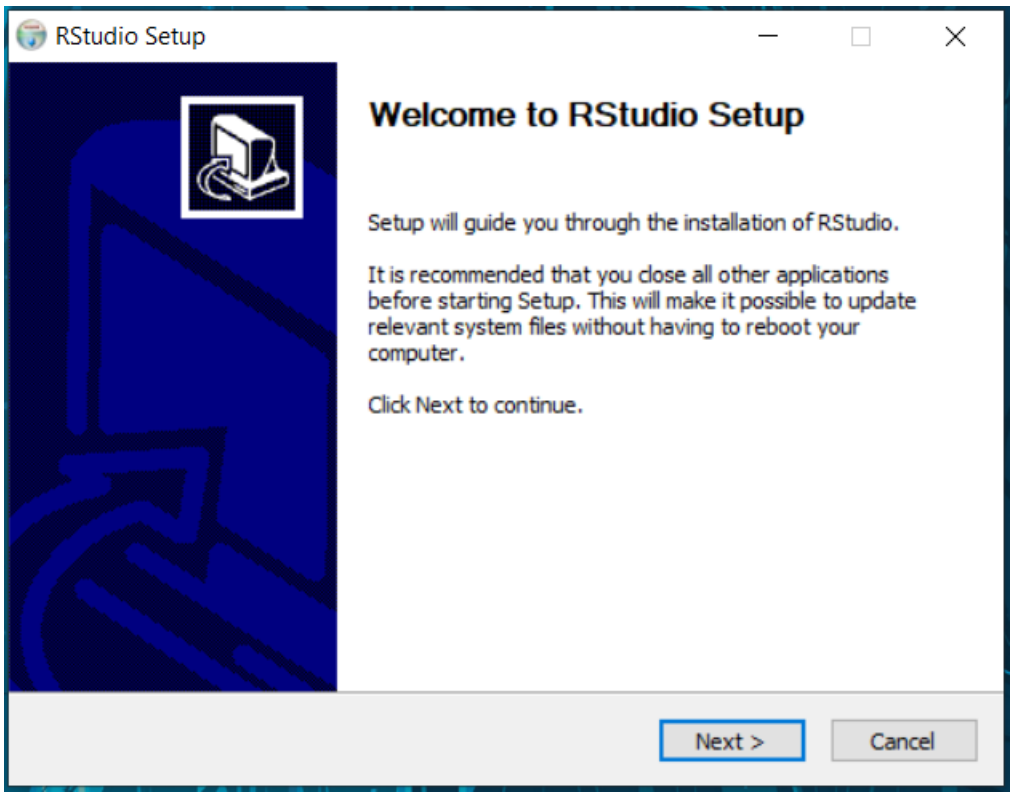


A.1.2 Windows: Download and install RStudio

Download RStudio from here.

After download, double-click the installer. It will ask for your administrator credentials to install (you might need to have your IT rep install again).

Accept all the default options for the RStudio install.



A.1.3 macOS: Download and install R

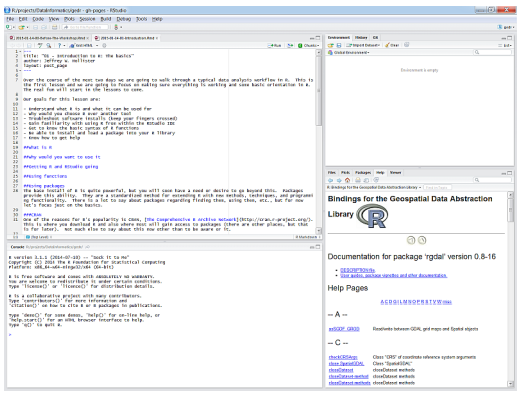
- Download and install R from the CRAN website for Mac here.
- Select the `.pkg` file for the latest R version
- Double click on the downloaded file to install R
- It is also a good idea to install XQuartz (needed by some packages)

A.1.4 macOS: Download and install RStudio

- Go to the RStudio download page
- Under Installers select the appropriate RStudio download file for macOS
- Double click the file to install RStudio

A.1.5 Check Install

Once installed, RStudio should be accessible from the start menu. Start up RStudio. Once running it should look something like this:



A.2 Install Quarto

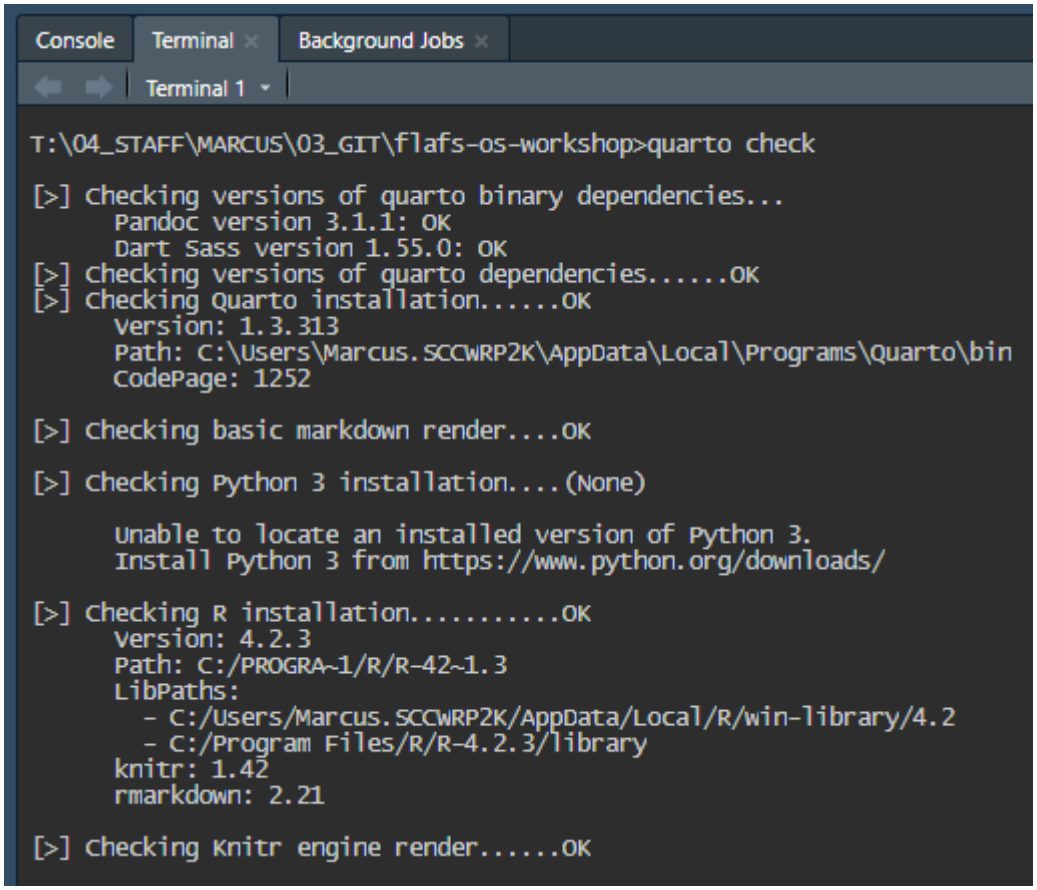
A visual editor for Quarto is installed with RStudio. However, you'll need to install Quarto CLI to make full use of its features.

Navigate to https://quarto.org/docs/get-started/. You'll see a screen that looks like this:

Platform	Download	Size	SHA-256
Ubuntu 18+/Debian 10+	quarto-1.3.318-linux-amd64.deb	84.57 MB	8d48b9c
Linux Arm64	quarto-1.3.318-linux-arm64.deb	84.4 MB	d9ab433
Mac OS	quarto-1.3.318-macos.pkg	105.26 MB	2a2e815
Windows	quarto-1.3.318-win.msi	77.82 MB	374e0bb
	Release notes and more downloads…		

Select the download appropriate for your operating system (Windows is the big blue button). After the file is downloaded, navigate to the folder containing the file, double-click to install, and accept the default settings at the prompts.

After installation is done, open RStudio (or close and open again) and select the Terminal tab. This tab is located on the bottom-left pane, next to the Console tab. Type `quarto check` at the prompt and press enter. You should see something like this if installation was successful.



A.3 Create GitHub account

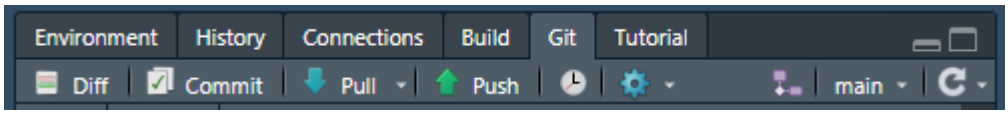
Open a web browser and enter the url https://github.com. On the top-right, you should see a button to sign up. Click the button and register an account by choosing an email, username, and password.

A.4 Install Git (optional)

After you've registered a new GitHub account, you can install the Git software on your computer. Git is version control software used by RStudio that allows you to access GitHub.

Open the url https://git-scm.com/book/en/v2/Getting-Started-Installing-Git and follow the instructions for your operating system.

After Git is installed, open RStudio (or close and open again) to verify the installation. You should see a new "Git" tab located in the top-right pane of RStudio.



A.4.1 Make sure RStudio can talk to GitHub via Git

The next step can be a bit tricky, but is essential if you want to access your GitHub using RStudio and Git. First, install the `usethis` R package in RStudio.

```
install.packages("usethis")
```

You must let Git know who you are and that you have permission to write to a GitHub repository. First, let Git know who you are, where you enter your user name and email associated with the account from the previous step.

```
usethis::use_git_config(user.name="Jane Doe", user.email="jane@example.org")
```

Next, you need to setup a personal access token (PAT) that defines the permissions to write to a repository. This can be done as follows:

```
usethis::create_github_token()
```

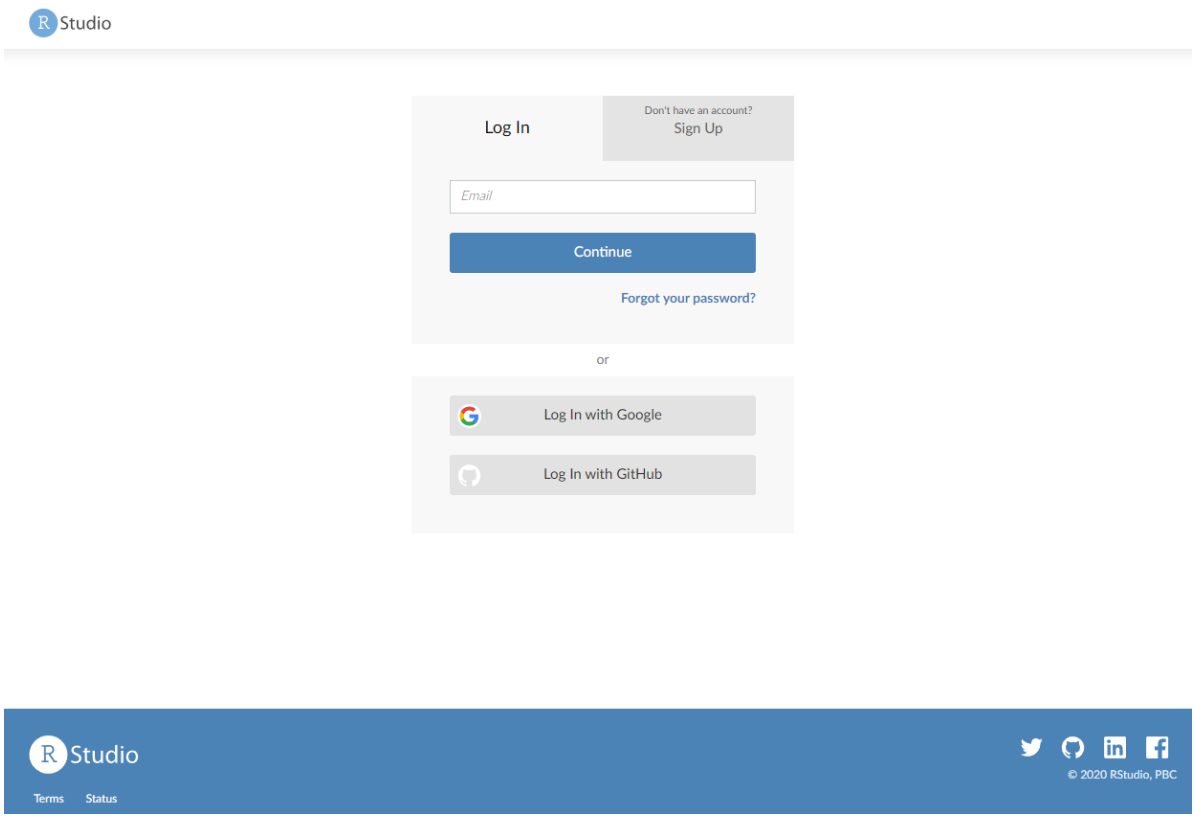
Then follow the remaining prompts to complete the PAT creation. A more thorough explanation can be found here.

A.5 This is hard!

If you have trouble installing any of the software prior to the workshop, you can use use RStudio in the cloud on the Posit website. This is only a backup option and we strongly encourage you to troubleshoot the installation when able.

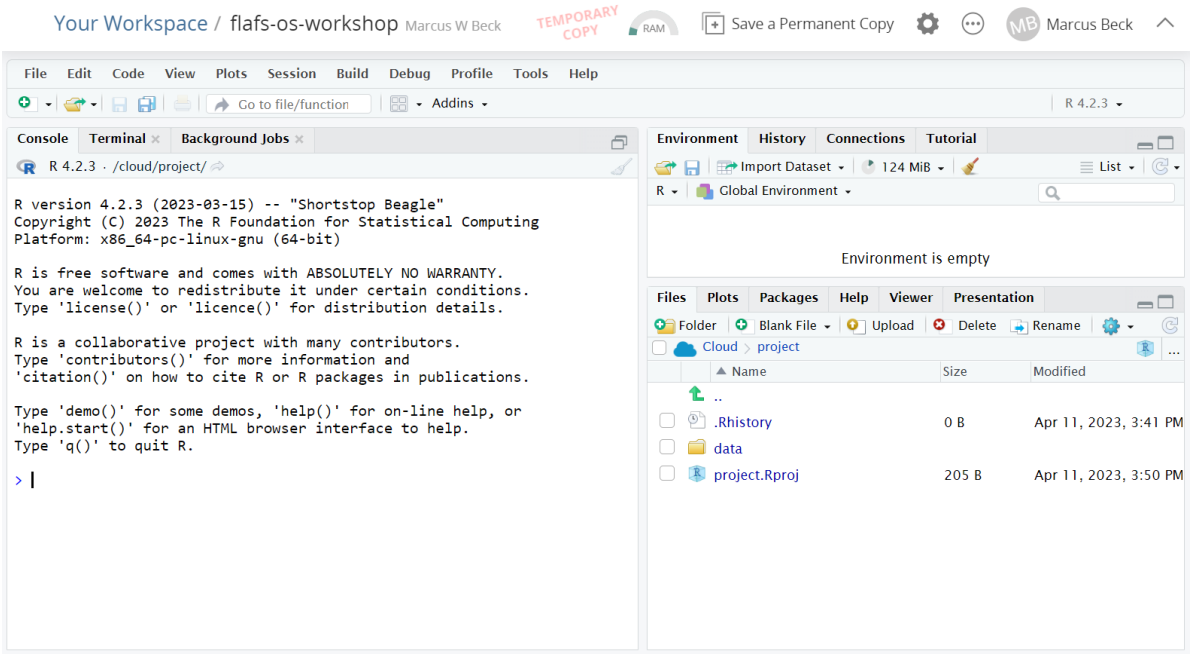
To use RStudio in the cloud, copy this link and paste it in a web browser: https://posit.cloud/content/5775087

If you do not have a Posit Cloud account, you will see this screen when you first visit the URL:



You can setup an account for free using a login you create or through a third-party (Google or GitHub).

After your account is setup, you should a screen that looks like this:



You'll see that this is a TEMPORARY COPY under your account. Make it permanent by clicking the button on top. This will save any changes you make to this project under your account.

B Introduction to R

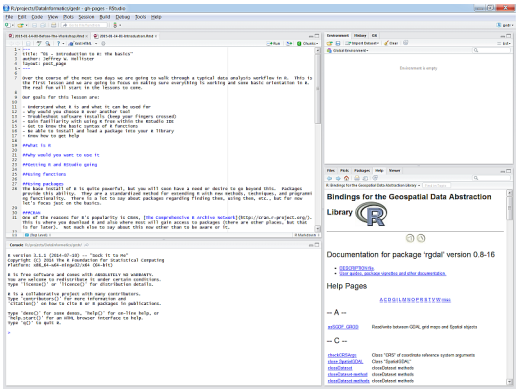
B.1 RStudio

RStudio is the go-to Interactive Development Environment (IDE) for R. Rstudio includes many features to improve the user's experience.

Let's get familiar with RStudio.

B.1.1 Open R and RStudio

Find the RStudio shortcut on your computer and fire it up. You should see something like this:



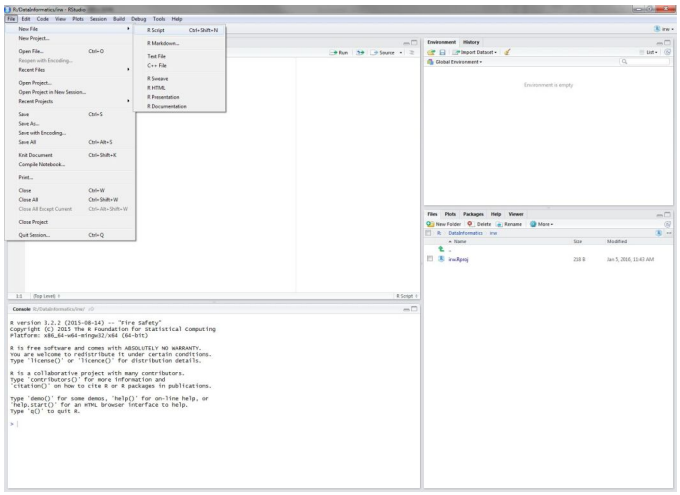
There are four panes in RStudio:

- **Source**: Your primary window for writing code to send to the console, this is where you write and save R "scripts"
- **Console**: This is where code is executed in R
- **Environment, History, etc.**: A tabbed window showing your working environment, code execution history, and other useful things
- **Files, plots, etc.**: A tabbed window showing a file explorer, a plot window, list of installed packages, help files, and viewer

B.1.2 Scripting

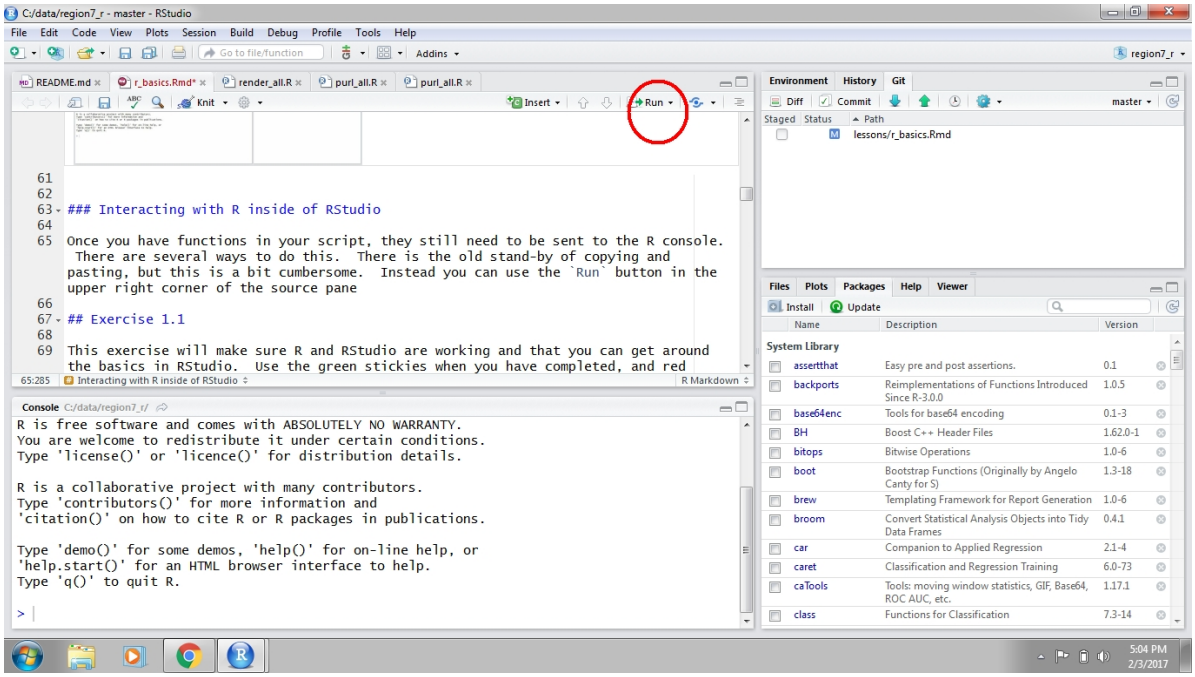
In most cases, you will not enter and execute code directly in the console. Code can be written in a script and then sent directly to the console.

Open a new script from the File menu…



B.1.3 Executing code in RStudio

After you write code in an R script, it can be sent to the Console to run the code. There are two ways to do this. First, you can hit the `Run` button at the top right of the scripting window. Second, you can use `ctrl+enter` (`cmd+enter` on a Mac). Either option will run the line(s) of script that are selected.



B.2 R language fundamentals

R is built around functions. The basic syntax of a function follows the form: `function_name(arg1, arg2, ...)`.

With the base install, you will gain access to many functions (2344, to be exact). Some examples:

```
# print
print("hello world!")
```

```
[1] "hello world!"
```

```
# sequence
seq(1, 10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```r
# random numbers
rnorm(100, mean = 10, sd = 2)
```

```
  [1]  9.510315  9.472886 13.371396  3.928072 11.590777  6.229769 12.926082
  [8]  7.133993 11.682513 10.769137 11.726609  9.741981  9.482421  7.757421
 [15] 12.736586 10.720136 10.041760 10.075359 10.303576  9.251305  5.161833
 [22]  7.366161 13.142874  8.401254 12.213756  7.975022  9.981634 12.020838
 [29] 12.892318  9.999828 10.034307  8.684290  9.393115 13.710205  9.792573
 [36]  7.663600 11.369314  8.025594 10.376753 11.231402 13.409885 10.120308
 [43] 10.753252 12.583008  9.095326 13.571648  9.144894  7.227999 14.364129
 [50]  5.731186  6.915156  8.874169  8.564031 12.377787 12.313535 12.093675
 [57] 15.146901  8.118614 10.221204 13.897908 10.480256  8.527210  7.280279
 [64]  9.314042  9.178460 14.269812 10.426083  9.633042 12.429730 10.297976
 [71]  7.832244  8.463819  7.409067  9.964568 12.913677  8.252506  9.563548
 [78] 10.564277  8.179686 10.916762  9.549043  8.757387  9.296954 11.852775
 [85]  8.134484 11.963646 10.603542 13.787894 12.653587  7.513778  9.163553
 [92]  8.584831 11.193082 10.901496  6.834456 11.380845 10.738945 10.746967
 [99]  8.382742 10.185587
```

```r
# average
mean(rnorm(100))
```

```
[1] 0.1390361
```

```r
# sum
sum(rnorm(100))
```

```
[1] -1.650461
```

Very often you will see functions used like this:

```r
my_random_sum <- sum(rnorm(100))
```

The first part of the line is the name of an object that you make up. The second bit, `<-`, is the assignment operator. This tells R to take the result of `sum(rnorm(100))` and store it in an object named, `my_random_sum`. It is stored in the environment and can be used by just executing it's name in the console.

```r
my_random_sum
```

```
[1] -17.28476
```

B.2.1 What is the environment?

There are two outcomes when you run code. First, the code will simply print output directly in the console. Second, there is no output because you have stored it as a variable using `<-`. Output that is stored is saved in the `environment`. The environment is the collection of named objects that are stored in memory for your current R session.

B.3 Packages

The base installation of R is quite powerful. Packages allow you to include new methods for use in R.

B.3.1 CRAN

Many packages are available on CRAN, The Comprehensive R Archive Network. This is where you download R and also where most will gain access to packages. As of 2023-05-09, there are 19474 packages on CRAN!

B.3.2 Installing packages

When a package gets installed, that means the source code is downloaded and put into your library. A default library location is set for you.

We use the `install.packages()` function to download and install a package. Here, we install the readxl package, used below, which is used to upload data from and Excel file.

```
install.packages("readxl")
```

You should see some text in the R console showing progress of the installation and a prompt after installation is done.

After installation, you can load a package using the `library()` function. This makes all functions in a package available for you to use.

```
library(readxl)
```

An important aspect of packages is that you only need to download them once, but every time you start RStudio you need to load them with the `library()` function.

B.4 Data structures in R

Now we can talk about R data structures. Simply put, a data structure is a way for programming languages to handle information storage.

B.4.1 Vectors (one-dimensional data)

The basic data format in R is a vector - a one-dimensional grouping of elements that have the same type. These are all vectors and they are created with the `c` (concatenate) function:

```
dbl_var <- c(1, 2.5, 4.5)
int_var <- c(1L, 6L, 10L)
log_var <- c(TRUE, FALSE, T, F)
chr_var <- c("a", "b", "c")
```

The four types of vectors are `double` (or numeric), `integer`, `logical`, and `character`. The following functions can return useful information about the vectors:

```
class(dbl_var)
```

```
[1] "numeric"
```

```
length(log_var)
```

```
[1] 4
```

B.4.2 Data frames (two-dimensional data)

A collection of vectors represented as one data object are often described as two-dimensional data, like a spreadsheet, or in R speak, a data frame. Here's a simple example:

```
ltrs <- c("a", "b", "c")
nums <- c(1, 2, 3)
logs <- c(T, F, T)
mydf <- data.frame(ltrs, nums, logs)
mydf
```

```
  ltrs nums  logs
1    a    1  TRUE
2    b    2 FALSE
3    c    3  TRUE
```

The only constraints required to make a data frame are:

1. Each column (vector) contains the same type of data
2. The number of observations in each column is equal.

B.5 Getting your data into R

It is the rare case when you manually enter your data in R. Most data analysis workflows typically begin with importing a dataset from an external source. We'll be using `read_excel()` function from the readxl package.

We can import the `ExampleSites.xlsx` dataset as follows. Note the use of a *relative* file path. You can see what R is using as your "working directory" using the `getwd()` function.

```
sitdat <- read_excel("data/ExampleSites.xlsx")
```

Let's explore the dataset a bit.

```
# get the dimensions
dim(sitdat)
```

```
[1] 11  5
```

```
# get the column names
names(sitdat)
```

```
[1] "Monitoring Location ID"        "Monitoring Location Name"
[3] "Monitoring Location Latitude"  "Monitoring Location Longitude"
[5] "Location Group"
```

```
# see the first six rows
head(sitdat)
```

```
# A tibble: 6 x 5
  `Monitoring Location ID` `Monitoring Location Name` Monitoring Location Lati~1
  <chr>                    <chr>                                          <dbl>
1 ABT-026                  Rte 2, Concord                                  42.5
2 ABT-062                  Rte 62, Acton                                   42.4
3 ABT-077                  Rte 27/USGS, Maynard                            42.4
4 ABT-144                  Rte 62, Stow                                    42.4
5 ABT-237                  Robin Hill Rd, Marlboro                         42.3
6 ABT-301                  Rte 9, Westboro                                 42.3
# i abbreviated name: 1: `Monitoring Location Latitude`
# i 2 more variables: `Monitoring Location Longitude` <dbl>,
#   `Location Group` <chr>
```

```
# get the overall structure
str(sitdat)
```

```
tibble [11 x 5] (S3: tbl_df/tbl/data.frame)
 $ Monitoring Location ID       : chr [1:11] "ABT-026" "ABT-062" "ABT-077" "ABT-144" ...
 $ Monitoring Location Name     : chr [1:11] "Rte 2, Concord" "Rte 62, Acton" "Rte 27/USGS, M
 $ Monitoring Location Latitude : num [1:11] 42.5 42.4 42.4 42.4 42.3 ...
 $ Monitoring Location Longitude: num [1:11] -71.4 -71.4 -71.4 -71.5 -71.6 ...
 $ Location Group               : chr [1:11] "Assabet" "Assabet" "Assabet" "Assabet" ...
```

You can also view a dataset in a spreadsheet style using the `View()` function:

```
View(sitdat)
```

B.6 Summary

In this intro we learned about R and Rstudio, some of the basic syntax and data structures in R, and how to import files. You'll be able to follow the rest of the workshop with this knowledge. View the Resources page for additional training materials.

C Resources

The following is a non-exhaustive list of additional resources you can use for continued learning on your open science journey.

C.1 Open Science Websites

- NCEAS Open Science for Synthesis workshop
- NCEAS Reproducible Research Techniques
- Open Science Foundation open science workshop
- Openscapes
- Openscapes Champions Lesson Series
- Supercharge your research: A 10 week plan for open data science
- ROpenSci guidance on creating a Code of Conduct
- NOAA Reproducible Reporting with R
- PeerJ collection on practical data science

C.2 Data Management Tools

- Environmental Data Initiative Data Management Resources
- University of California DMPTool
- US Geological Survey resources for Metadata Creation
- ELIXIR and others Data Stewardship Wizard
- TBEP Data Management SOP

C.3 TBEP R Trainings

- Peconic Estuary Program R training, recording
- TBEP June 2020 R training, recordings
- Writing functions in R
- R package development workflow
- A soft introduction to Shiny

C.4 R Lessons & Tutorials

- Software Carpentry: R for Reproducible Scientific Analysis
- Data Carpentry: Geospatial Workshop
- Data Carpentry: R for Data Analysis and Visualization of Ecological Data
- Data Carpentry: Data Organization in Spreadsheets
- R for Water Resources Data Science
- RStudio Webinars, many topics
- R For Cats: Basic introduction site, with cats!
- Topical cheatsheets from RStudio, also viewed from the help menu
- Cheatsheet from CRAN of base R functions
- Totally awesome R-related artwork by Allison Horst
- Color reference PDF with text names, Color cheatsheet PDF from NCEAS

C.5 R eBooks/Courses

- Jenny Bryan's Stat545.com
- Garrett Grolemund and Hadley Wickham's R For Data Science
- Chester Ismay and Albert Y. Kim's Modern DiveR
- Julia Silge and David Robinson Text Mining with R
- Hadley Wickham's Advanced R
- Hadley Wickham's R for Data Science
- Yihui Xie R Markdown: The Definitive Guide
- Winston Chang R Graphics Cookbook
- Wegman et al. Remote Sensing and GIS for Ecologists: Using Open Source Software
- Lovelace et al. Geocomputation with R
- Edszer Pebesma and Roger Bivand Spatial Data Science

C.6 Git/Github

- Jenny Bryan's Happy Git and Github for the useR
- Git and GitHub for the Casual User
- Coding Club Intro to Github

D Contributor Covenant Code of Conduct

D.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

D.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

D.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

D.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

D.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at mbeck@tbep.org. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

D.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

D.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

D.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

D.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

D.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

D.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

References

Broman, K. W., and K. H. Woo. 2018. "Data Organization in Spreadsheets." *The American Statistician* 72 (1): 2–10. https://doi.org/10.1080/00031305.2017.1375989.

Fecher, B., and S. Friesike. 2014. "Open Science: One Term, Five Schools of Thought." In *Opening Science*, 17–47. Springer, Cham.

Knuth, Donald Ervin. 1984. "Literate Programming." *The Computer Journal* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Lowenberg, Daniella, Rachael Lammey, Matthew B Jones, John Chodacki, and Martin Fenner. 2021. "Data Citation: Let's Choose Adoption over Perfection." Zenodo. https://doi.org/10.5281/zenodo.4701079.

Michener, W. K., J. W. Brunt, J. J. Helly, T. B. Kirchner, and S. G. Stafford. 1997. "Non-geospatial Metadata for the Ecological Sciences." *Ecological Applications* 7 (1): 330–42. https://doi.org/10.1890/1051-0761(1997)007%5B0330:NMFTES%5D2.0.CO;2.

Shafland, Paul L., and James M. Pestrak. 1982. "Lower Lethal Temperatures for Fourteen Non-Native Fishes in Florida." *Environmental Biology of Fishes* 7 (2): 149–56. https://doi.org/10.1007/bf00001785.

Wickham, H. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10): 1–23. https://doi.org/10.18637/jss.v059.i10.

Wickham, H., M. Averick, J. Bryan, W. Chang, L. D'Agostino McGowan, R. François, G. Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, H., and G. Grolemund. 2017. *R for Data Science*. Sebastopol, California: O'Reilly.

Wilkinson, M. D., M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, et al. 2016. "The FAIR Guiding Principles for Scientific Data Management and Stewardship." *Scientific Data* 3 (160018). https://doi.org/10.1038/sdata.2016.18.

Wilson, G., J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal. 2017. "Good Enough Practices in Scientific Computing." *PLoS Computational Biology* 13 (6): e1005510. https://doi.org/10.1371/journal.pcbi.1005510.

Ziemann, M., Y. Eren, and A. El-Osta. 2016. "Gene Name Errors Are Widespread in the Scientific Literature." *Genome Biology* 17 (1): 1–3. https://doi.org/10.1186/s13059-016-1044-7.