# TADA! Simple guidelines to improve analytical code sharing for transparency and reproducibility

*Edward R. Ivimey-Cook[1@], Antica Culina[2], Shreya Dimri[3], Matthew J. Grainger[4], Fonti Kar[5,6], Malgorzata Lagisz[6,7], Nicholas P. Moran[8], Shinichi Nakagawa[7], Dominique G. Roche[9], Sean Tattan[1], Alfredo Sánchez-Tójar[3,10,11], Saras M. Windecker[12], Joel L. Pick[13]*

*1 University of East Anglia, Norwich, UK; 2 Ruder Boskovic Institute, Croatia; 3 Department of Evolutionary Biology, Bielefeld University, Germany; 4 Norwegian Institute for Nature Research, Trondheim, Norway; 5 Research School of Finance, Actuarial Studies & Statistics, The Australian National University, Canberra, Australia. 6 School of Biological, Earth & Environmental Sciences, University of New South Wales, Sydney, Australia; 7 Department of Biological Sciences, University of Alberta, Edmonton, Canada; 8 Centre of Excellence for Biosecurity Risk Analysis, Biosciences, University of Melbourne, Parkville, Victoria, Australia; 9 Institut de Biologie, Université de Neuchâtel, NE, Switzerland; 10 CNC, Center for Neuroscience and Cell Biology, University of Coimbra, Portugal; 11 CIBB, Center for Innovative Biomedicine and Biotechnology, University of Coimbra, Portugal; 12 The Kids Research Institute Australia, Nedlands, WA, Australia; 13 Institute of Ecology and Evolution, University of Edinburgh, Edinburgh, UK;*

@corresponding author: e.ivimeycook@gmail.com; authors aside from the first and last are ordered alphabetically.

## *Acknowledgements*

We thank Sarah Wilson Kemsley for discussion of the TADA guidelines in other coding languages across disciplines.

## *Conflict of Interest*

EIC, JLP, SN, ML, DGR, NPM, SD, and AS-T are members of the Society for Open, Reliable, and Transparent Ecology and Evolutionary Biology (SORTEE). EIC is the Past-President. EIC and AS-T are past board members. ML is a current board member and president-elect.

## *Author contributions*

EIC and JLP conceptualised the idea. EIC wrote the first draft. EIC, ML, and SD made figures. All authors (EIC, AC, SD, MJG, FK, ML, NPM, SN, DGR, ST, AS-T, SMW, and JLP) contributed to reviewing and editing subsequent drafts.

## *AI declaration*

GPT-4.0 was used to generate the dog and wizard used in the figures.

## *Data availability*

No data were collected for this paper, however a Zenodo DOI has been generated to archive the TADA example, 10.5281/zenodo.20757675.

## *Funding*

AC was supported by the Croatian Science Foundation under the project number HRZZ-IP-2022-10-2872. AS-T was partially supported by Portuguese national funds via Fundação para a Ciência e a Tecnologia (FCT) under projects LA/P/0058/2020, UID/PRR/4539/2025 and

UID/04539/2025, and project EXCELScIOR, funded by the EU's Horizon Europe under Grant Agreement No. 101087416.

## *Abstract*

Code sharing is essential to ensure transparency and computational reproducibility of published research, which in turn increases trust in scientific results. However, despite the growing number of journals that mandate code sharing, the prevalence of open code remains low and substantially lags behind that of open data. Furthermore, even when it is openly shared, code is often non-functional, which hinders computational reproducibility. One reason for low levels of code sharing is uncertainty around how to properly archive functional analytical code associated with published research. Existing resources for best coding practices often do not sufficiently address how to archive analytical code, do not adhere to the established FAIR (*Findable, Accessible, Interoperable, and Reusable*) principles, do not align with journal requirements, or are complex and primarily developed for software development. To address this gap, we provide simple code sharing guidelines: TADA (*Transferable, Available, Documented and Annotated*). TADA details the minimum requirements necessary for a researcher to produce functional code for sharing that directly supports best practices and aligns with the FAIR principles and common journal requirements. TADA aims to streamline the process of archiving and sharing functional code for researchers across all levels of coding experience, to increase the transparency, reproducibility, and the reliability of research results. These guidelines were developed based on our experience in ecology and evolutionary biology but can be applicable to researchers in other disciplines.

## *Keywords*

Open science, Research integrity, Reliability, Replicability, Research methods, Methodological rigour

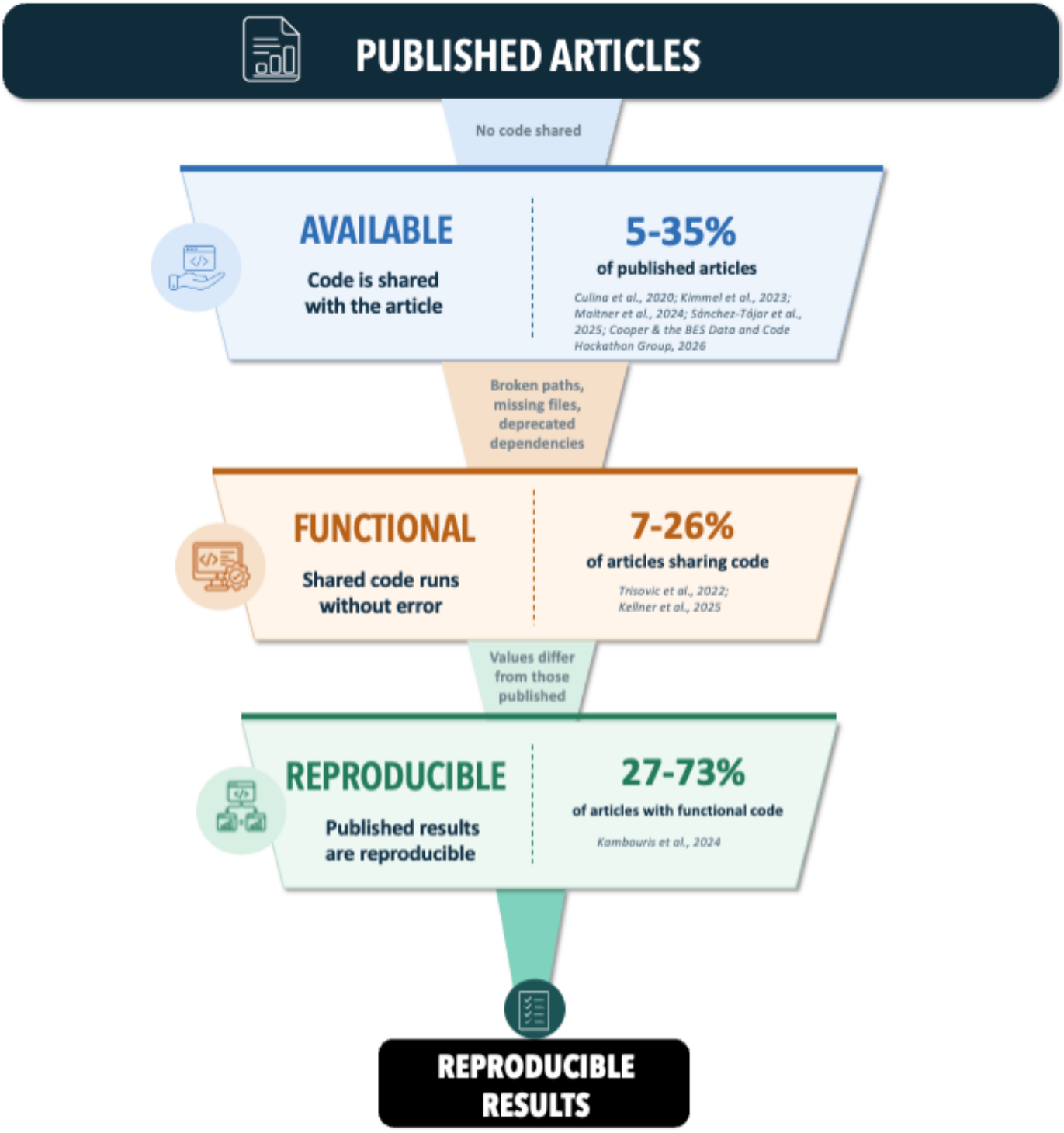
## *Introduction*

Publicly sharing code (i.e., open code) offers numerous benefits for researchers and the broader scientific community. For authors, open code may increase citation rates of associated articles (Vandewalle, 2012; Maitner *et al.*, 2024) and can provide future career advantages (McKiernan *et al.*, 2016; Allen & Mehler, 2019; König *et al.*, 2025). For the broader community, open code, alongside open data, is essential for ensuring computational reproducibility (the ability to reproduce analyses and results using the same data, code, and computational conditions; National Academies of Sciences *et al.*, 2019), reduces research waste (Purgar *et al.*, 2022), and is a key part of the scientific process that promotes reliability and builds trust in research (Fidler *et al.*, 2017; Powers & Hampton, 2019). The presence of publicly available analytical code associated with a research article also enhances the transparency of analytical methods and the overall research process (Goldacre *et al.*, 2019; Fernández-Juricic, 2021; Ivimey-Cook *et al.*, 2023) and enables other researchers to efficiently build upon published work (Barnes, 2010; Eglen *et al.*, 2017). Furthermore, as awareness of these benefits grows amongst researchers and the wider scientific community (Eynden *et al.*, 2016; Cadwallader & Hrynaszkiewicz, 2022; Ferguson *et al.*, 2023), an increasing number of journals in ecology and evolutionary biology are promoting open code by implementing code sharing policies (from 15% in 2015 to 88% in 2024; Mislan *et al.*, 2016; Culina *et al.*, 2020; Ivimey-Cook *et al.*, 2025). These policies encourage or require authors to share code, although mandatory code sharing is often not enforced and code is rarely reviewed (see Ivimey-Cook *et al.*, 2025). Ideally, when shared, open code should follow the

FAIR principles, initially published for data (Wilkinson *et al.*, 2016) and later adapted for Research Software (FAIR4RS; (Barker *et al.*, 2022; Chue Hong *et al.*, 2022). FAIR stands for *Findable*: the ability for both machines and humans to easily find digital assets (including metadata, data, and code); *Accessible*: digital assets are retrievable via their identifier, and can be accessed with or without the need for additional authorisation or authentication; *Interoperable*: digital assets must be able to interoperate with other digital assets and be readable using standard documented formats; and lastly, *Reusable*: digital assets must be described sufficiently to enable reuse and attribution, ideally via a licence (see Wilkinson *et al.*, 2016; Barker *et al.*, 2022; Chue Hong *et al.*, 2022).

Despite incremental progress towards more transparent and reproducible research in ecology and evolutionary biology (Cao *et al.*, 2023), evidence suggests there are significant barriers to code sharing. First, the proportion of articles with open code in ecology and evolutionary biology remains alarmingly low, with rates of code sharing ranging between 5% and 35% (Culina *et al.*, 2020; Kimmel *et al.*, 2023; Maitner *et al.*, 2024; Kellner *et al.*, 2025; Sánchez-Tójar *et al.*, 2025; Cooper & the BES Data and Code Hackathon Group, 2026; Figure 1). Second, even when code is provided, its functionality (i.e., the ability to run code without errors) is often low (Trisovic *et al.*, 2022; Kellner *et al.*, 2025; Figure 1). In a recent study examining R code from species distribution and abundance articles, the authors had to abandon the reproducibility aspect of their analysis due to the overwhelmingly high proportion of code that did not run (93% of coding scripts; Kellner *et al.*, 2025). Similarly, a recent review of over 9000 R files from the Harvard Dataverse repository found that 74% of code failed to complete without error, which only decreased to 56% after basic code cleaning was applied (e.g., removal of absolute file paths and

ensuring libraries and dependencies were properly installed and loaded; Trisovic *et al.*, 2022). Third, even if code is present and functional, computational reproducibility is not always achieved (Campbell *et al.*, 2023; Kambouris *et al.*, 2024; Kellner *et al.*, 2025; Figure 1). For instance, the ability to reproduce the results of meta-analyses in ecology and evolutionary biology has been shown to vary from 27% (all results within an article exactly matched) to 73% (50% of results within an article were within 10% of the original value) when data and code were shared and functional (Kambouris *et al.*, 2024). The low prevalence of code archiving, low functionality of archived code, and low computational reproducibility of results when functional code is archived paints a concerning picture for ecology and evolutionary biology and suggests that many of the benefits of code sharing are not being achieved.




126

127 **Figure 1.** The code sharing cascade. Reproducible results depend on code that is available (i.e.,

128 shared with the article) and functional (i.e., runs without error). The range of percentages given

129 for articles with published code and articles with functional code represents the lowest and

highest percentages in data aggregated from: Culina *et al.*, 2020; Trisovic *et al.*, 2022; Kimmel *et al.*, 2023; Kambouris *et al.*, 2024; Maitner *et al.*, 2024; Kellner *et al.*, 2025; Sánchez-Tójar *et al.*, 2025; Cooper & the BES Data and Code Hackathon Group, 2026.

A major reason for the limited availability and functionality of code and, therefore, the low rates of computational reproducibility, is likely due to a lack of knowledge of how to share code with transparency and reproducibility in mind (Gomes *et al.*, 2022). Whilst several interdisciplinary resources have been created to help authors prepare and share code (Sandve *et al.*, 2013; Cooper, 2017; Jiménez *et al.*, 2017; Barker *et al.*, 2022; Chue Hong *et al.*, 2022; Filazzola & Lortie, 2022; Ivimey-Cook *et al.*, 2023; Patel *et al.*, 2023; Abdill *et al.*, 2024; Rokem, 2024; Sharma *et al.*, 2024; Hillemann *et al.*, 2025), they are not focused on how to practically *archive* functional code used for analyses in research articles (i.e., analytical code). Instead, they mostly focus on how to write or review code rather than how to specifically prepare analytical code for archiving in a manner that is simple and accessible for all levels of coding experience. In addition, few explicitly link to the FAIR principles, and those that do, such as FAIR4RS (Barker *et al.*, 2022; Chue Hong *et al.*, 2022), are too broad in scope and focused on software development, potentially explaining why they have not been widely adopted beyond this specific community. Lastly, they are also not designed to help authors directly adhere to journal requirements for code sharing (e.g., Pick *et al.*, 2026), which have been specifically written to aid functionality and computational reproducibility.

The term ‘code reusability’ is often used in two different contexts. In the context of the FAIR principles, reusability involves sharing code in a way that clearly specifies what can be done with

it: for example, via a license and a documentation file. In a software development context, designing code for reuse is a far more complex process, as code needs to be written in a generalised and modular way, and tested, enabling it to function across different systems and with various compatible datasets as input (e.g., Hillemann *et al.*, 2025). Current guidelines focus on the latter context, and although they are extremely useful and important in ensuring best practices for open-source software, they likely set too high a bar for analytical code that does not need to meet the standards of reusable software to achieve its intended benefits. Analytical code is typically far more unique and tailored to a specific dataset than open-source software. The main goal of producing and sharing analytical code is typically not to create tools or for broad reuse but rather to produce a transparent and reproducible record of the analysis for a particular study. Therefore, establishing simple best practices for code to align with the FAIR principles and minimum standards for transparency and computational reproducibility is an important step towards increasing the rate and quality of analytical code sharing in ecology and evolutionary biology. Here, we provide simplified and easy-to-follow guidelines built with the FAIR principles in mind but tailored to analytical code for both empirical and theoretical research. We call these guidelines TADA (Transferable, Available, Documented, and Annotated) and believe they will help researchers at all coding levels prepare functional code that facilitates reproducible and transparent research, ultimately strengthening trust in published results. TADA aligns with standardised journal guidelines for data and code sharing (Pick *et al.*, 2026), specifically Stages 3 (and to some extent Stage 4-5), and provides a useful educational resource for journals and research institutions.

## TADA!

TADA consists of four easy-to-follow steps to help researchers share functional code. By following the TADA guidelines (Figures 3-6), researchers can produce analytical code that follows best practices, aligns with the FAIR principles (Wilkinson *et al.*, 2016; Barker *et al.*, 2022; Chue Hong *et al.*, 2022), increases transparency, adheres to journal requirements (Stage 3 of new standardised data editor guidelines by Pick *et al.*, 2026), and lastly, facilitates computational reproducibility. TADA is tailored mainly to R and Python, as these open-source coding languages are widely used in ecology and evolutionary biology (Lai *et al.*, 2019; Gao *et al.*, 2025). However, the basic principles of the guidelines can be widely applied to other languages, including workflow (e.g., Snakemake) and compiled languages (e.g., C++), and to disciplines beyond ecology and evolutionary biology. For a checklist of the TADA guidelines in a variety of languages see Figure S1.

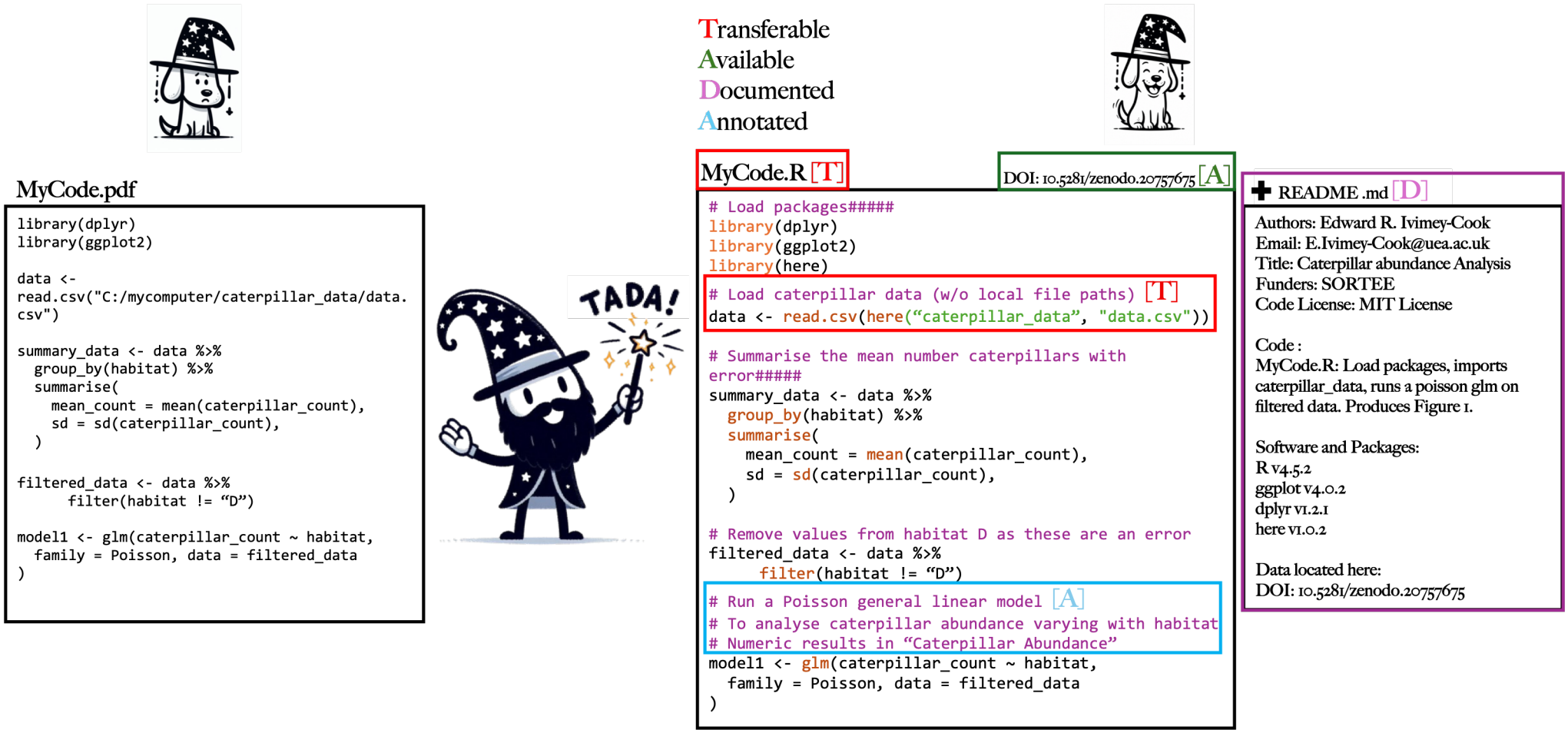



**Figure 2.** An example of the TADA guidelines (Transferable, Available, Documented, and Annotated) applied to analytical code written in R, showing a pre-TADA script (left) and a post-TADA script (right). Coloured letters correspond to Transferable (red), Available (green), Documented (purple), and Annotated (blue). The code shown is generic and designed to showcase the TADA guidelines.

## Transferable

Transferability refers to the ability for anyone to open a file containing code, view and run it without conversion or alteration (Figure 3). This corresponds to the FAIR principle of interoperability that is specific to code (in simple terms, implies that anyone should be able to open and use your code regardless of computer or operating system) and extends it to allow code to be *run* on different computers and operating systems. Ensuring code transferability greatly increases its reusability and computational reproducibility.

To be transferable, code must first be saved and encoded (i.e., *how* the data is saved) in a file type that can be opened by any text editor or integrated development environment (IDE; an application combining programming tools into a graphical interface to aid coding such as RStudio, VSCode, PyCharm, or Positron). In Figure 2, the non-transferable, pre-TADA code is in the form of a .pdf file. This file can be viewed but cannot be opened and edited within an IDE without using additional libraries or software, or without conversion. Importantly, copying and pasting code from certain file types (i.e., .pdf or .docx) can lead to changes in characters (e.g., apostrophes or the addition of line numbers and headers) or white spaces. These can easily result in code errors that are sometimes difficult to spot or time-consuming to fix. We suggest saving code in a transferable file extension with appropriate encoding, such as .R, .py, or .cpp, as these can be readily viewed, edited and saved using any text editor or IDE. Whilst a .txt file can be used to share code that allows for copying and pasting, it can lead to issues with code interpretation in many IDEs (e.g., without the. R file extension, IDEs may not recognise and allow for execution of the R coding language without saving the .txt file as a. R file).

Second, to ensure that your code runs on different computers and operating systems, file paths must be relative rather than absolute (also referred to as local or hard-coded file paths) and should be written in a way that is not specific to a user's local environment or specific directory structure. For example,

```
setwd("C:/Users/Ed/Desktop/MyProject/data/raw")
data <- read.csv("caterpillar_data.csv")
```

or

```
data <- read.csv("C:/Users/Ed/Desktop/MyProject/data/raw/caterpillar_data.csv")
```

will only work on Ed's computer, assuming that the file in question is in a folder called 'MyProject' and is itself on Ed's Desktop. Importantly, data, code, and all necessary materials should be organised in a single project directory with an appropriate substructure (e.g., data and code within distinct subfolders inside a project; for a nice example of appropriate file structure see Figure 3 in Brun *et al.*, 2025). For example,

```
MyProject/
      data/
            raw/
                  catepillar_data.csv
            processed/
      code/
      figures/
```

Everything in that main directory should remain in the same structure when archiving, to ensure that the specified file paths continue to work. In this example, if only the code folder was shared, another user would not know where the data was meant to be located relative to the code files. In Figure 2, the use of absolute user-specific file paths in the pre-TADA code will cause all other users to encounter errors when importing the required data file. In contrast, the post-TADA panel is agnostic of operating system and file paths, allowing prospective users to load the necessary data file provided the data is present in the archive.

Although beyond the scope of these guidelines, reproducible analyses can also be supported by dependency and workflow managers, and containerisation. Dependency managers such as {renv} in R (Ushey, 2023) or environment managers such as {conda} in Python (contributors, 2026) can be used as reliable methods of ensuring the same computational environment (using renv.lock and conda-lock). Workflow managers such as Snakemake (Köster & Rahmann, 2012) and Nextflow (Di Tommaso *et al.*, 2017) promote reproducibility by specifying the sequence of scripts or computational steps in a pipeline and their dependencies. This ensures each step in the pipeline executes in a defined and reproducible order (Di Tommaso et al., 2017). Lastly, containerisation platforms such as Docker (Merkel, 2014) and Apptainer (previously called Singularity) (Kurtzer *et al.*, 2017) use images that encapsulate a complete software environment, including all required programs and libraries. This helps ensure environment reproducibility between systems to avoid the common “works on my computer” problem (Mitra-Behura *et al.*, 2022).

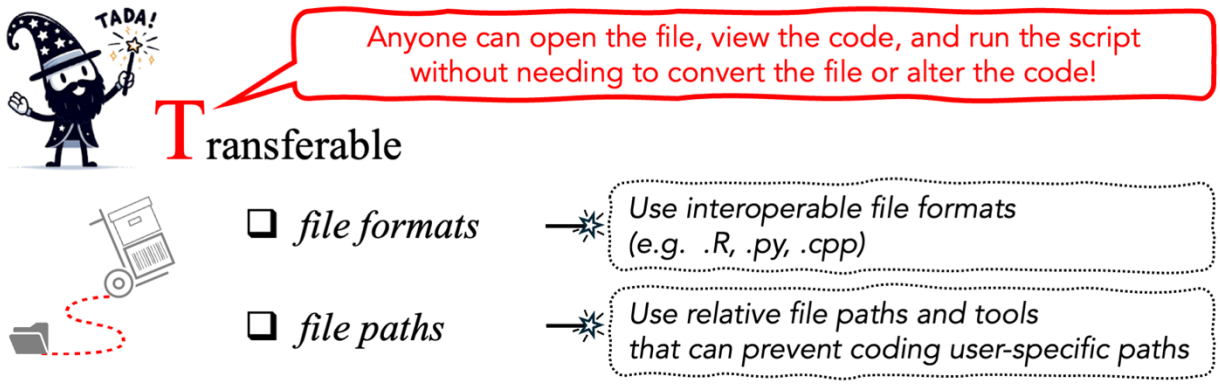



**Figure 3.** Summary of advice on making analytical code *Transferable*.

**Transferability How To (See also Figure 3):** **When sharing R or Python code, ensure that each code file is appropriately saved and encoded as either a .R, .py, or .cpp, file (or other standard format, as appropriate). Avoid sharing code within Word documents (.doc or .docx) or PDFs. If the coding language or IDE does not use or save code in a standard file type, check to see if the resulting file can be opened by a text editor (e.g., SPSS syntax .sps files can be readily viewed in a text editor).**

**There are several options to specify relative file paths and avoid absolute file paths and the use of setwd() or os.chdir() in your code. RStudio users can simply create a new RStudio project (File --> New Project; see https://docs.posit.co/ide/user/ide/get-started/), which eliminates the need for absolute file paths. RStudio projects can be used in combination or separately from packages such as {here} (Müller & Bryan, 2020), which create file paths relative to any project directory regardless of operating system (i.e., relative file paths). In Python, this is instead managed by the {pyprojroot} package (Chen, 2023). We recommend using both RStudio and {here} to maximise transferability across operating systems.**

**Additional methods include navigating to the project file and opening it within VSCode or Positron or running an instance of R or Python within the specific project folder. The latter will remove the need for absolute file paths that will almost certainly lead to errors when other users try to run the code on different systems. Whichever method is chosen should be in the code documentation (see below). One additional tip is to ask a colleague to run the code on their machine prior to archiving to ensure the code is *Transferable*.**

## **A**vailable

Availability refers to the act of publicly archiving the code in a way that provides long-term access to any external user via a unique and persistent identifier (Figure 4). In this context, Available corresponds to the FAIR principles of both Findable (provision of a unique identifier) and Accessible (code is retrievable via this identifier).

To store code in an open and easily available manner, code must have an associated globally unique persistent identifier or PID (e.g., a Digital Object Identifier, DOI), which must be cited in the corresponding manuscript. Whilst GitHub might be a commonly used platform for developing code and provides a transparent platform for version control during the development phase (Braga *et al.*, 2023; Kang *et al.*, 2023), it does not readily provide a PID and files can be changed (or even deleted) at any time, including after manuscript publication and after archiving (i.e., GitHub is not immutable). This limits reproducibility of published results if the exact code is no longer available or is edited. As such, GitHub and similar platforms (e.g., Codeberg, Bitbucket, GitLab) are not suitable for archiving analytical code used in an article. Repositories such as Zenodo and Figshare (which can both connect to a GitHub project) are immutable and

can provide both a base project-level DOI that never changes and version-specific DOIs, created whenever a new version of the code is released. Another useful resource is Software Heritage (https://archive.softwareheritage.org/), which can preserve GitHub projects for long-term storage and provides PIDs in the form of Software Hash Identifiers (SWHIDs). In Figure 2, the lack of archived code and associated DOI in the pre-TADA code limits code sharing and prevents permanent, immutable, and citable storage of the code.

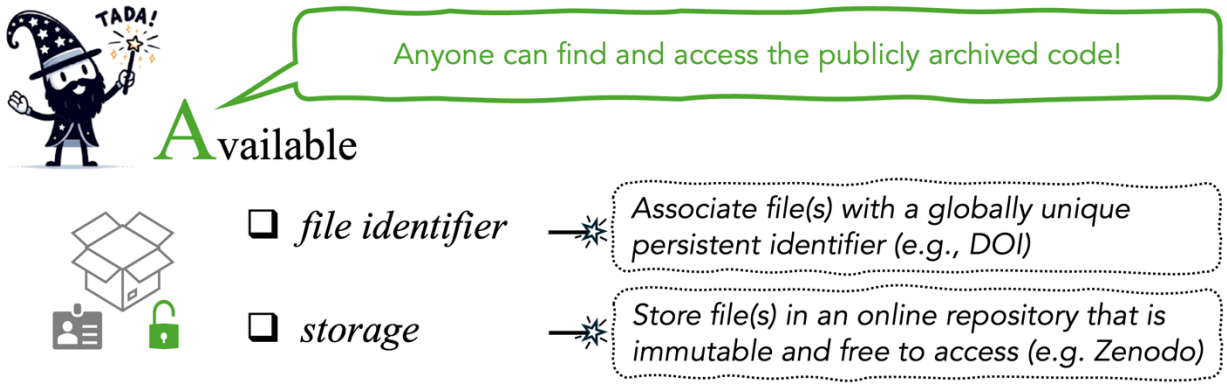



**Figure 4.** Summary of advice on making analytical code *Available*.

**Availability How To (See also Figure 4): Upload your code to Zenodo (https://zenodo.org/), Figshare (https://figshare.com/) or any other repository that assigns a DOI and guarantees immutability and preservation. A unique DOI will be created for the project, and a new one produced whenever it is subsequently updated (known as DOI versioning). Assigning a DOI facilitates citing the code and linking to it in the related manuscript; we recommend citing the DOI associated with the version of the code that produced the results in the manuscript. GitHub is not ideal for archiving and sharing analytical code associated with an article because it is not immutable and does not generate a DOI. Instead, users can link their GitHub repository to Zenodo or Figshare and create a release version on GitHub**

**which will automatically be captured on Zenodo or Figshare (see https://help.zenodo.org/docs/profile/linking-accounts/ or https://info.figshare.com/user-guide/how-to-connect-figshare-with-your-github-account/ for more information regarding linking projects with Zenodo and Figshare, respectively).**

## Documented

Documentation refers to providing accurate and detailed metadata files that describe the code files and their usage (Figure 5). This corresponds to to the FAIR principle of Reusable (anyone can understand how to reuse and attribute the code) and Findable (the provision of rich metadata).

Typically, documentation is often provided as an additional text file (.txt) or Markdown file (.md) (typically called a README). Documentation could be provided as a combined README containing metadata for both code and data, or as two separate READMEs, one for code and one for data. Metadata for data include a description of the data files, variables, and units of measurement; for more details on data-specific metadata, see Lortie *et al.*, (2022).
Figure 2 shows an example of essential information that should be contained within a documentation file such as a README. For instance, it should include information on the author(s) of the code with some form of contact information, the title and abstract of the corresponding manuscript, and any other information (e.g., any funding bodies). It should also specify the computational environment used, as different software or package versions can produce different results (for instance, the sample() function in base R changed after v3.6.0 and the distinct() function changed after v0.4.0, producing differing results; see also Abdill *et al.*,

2024). The documentation file should therefore include the software version (e.g., R v4.5.2) and list all packages with associated version numbers (e.g., {ggplot} v4.0.2). This information could also be provided as a text file that lists every loaded package and version number; retrieved by sessionInfo() in R or session_info in Python. The documentation should also include any PID or other information that enables locating the corresponding dataset, alongside any additional information needed to run the code (e.g., what each file contains, the order in which to run them, whether the code takes a long time to run, what it requires data-wise to run and what it produces).

The documentation must also specify an appropriate licence detailing how others can use, modify and share the code. Code-specific licences can take many forms, such as the Massachusetts Institute of Technology (MIT) or GNU General Public Licence (GPL) and can differ in their permission levels and conditions. For instance, the licence usually details if attribution is required (i.e., whether you are required to cite the creator of the code), and whether code can be modified and/or used for commercial purposes. Licences can range from completely open and permissive, such as MIT, which has little to no restrictions on use, to more restrictive, such as the GPL, which imposes several conditions that must be met. For instance, applying the same licence to any derivative works and listing any changes made from the source code (e.g., GPL v3.0). Data and code can be associated with their own separate and specific licences in the same repository; a researcher should carefully consider what form of code-specific licence is needed or whether the repository they choose to use has a default repository-wide licence that covers both data and code (e.g., Dryad only supports the CC0 licence, which is not best suited for code). Websites such as choosealicense.com provide detailed guidance on selecting and

adding an appropriate licence to a project (although, in essence, it can simply involve copying the respective license text and saving the file to the project; see also Pick *et al.*, 2026 for further details on both data and code licensing). Many factors will influence what licence to choose and how open you want your code to be, including who the audience is (i.e., is it intended for commercial applications?), whether you want to allow others to modify or extend your code, and how this aligns with journal, institutional and funder policies. For instance, some journals require the use of a specific licence upon archiving (e.g., a GPL license in the Journal of Statistical Software). Figure 2 illustrates the implications of licensing choices. The pre-TADA code lacks a licence, which legally restricts others from using, sharing, or modifying the archived code. In contrast, the post-TADA code has an MIT licence, explicitly granting users permission to copy, modify, merge, publish, and share the archived code with proper attribution.

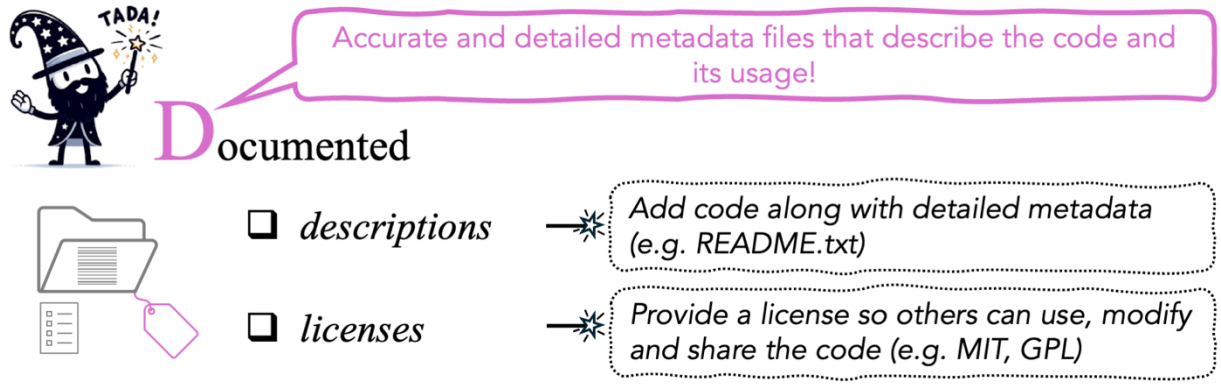



**Figure 5.** Summary of advice on making analytical code *Documented*.

**<u>Documented How To (See also Figure 5):</u> Code documentation can provide important information that code annotation lacks. A README.txt or .md file describing the code should contain additional information on the manuscript that the code is associated with**

**(including the title and abstract of the manuscript, any relevant funders, and a list of authors with emails for correspondence; if necessary, this can be anonymised during peer review to adhere to double-blind reviewing policies), software used (e.g., R or Python, including version number), any important libraries or packages used (with version numbers), information about where relevant data is located (if appropriate, with a PID), a mention of the code-specific licence, and any other important pieces of information, such as the order in which the code should be run, whether the code takes a long time to run (especially for computing intensive processes), what data the code requires to run, where the data is located, and what data the code produces. Packages such as READMEBuilder (https://github.com/EIvimeyCook/READMEBuilder) can help authors create and share detailed READMEs that facilitate "Documented".**

**For licences, there exists a multitude to choose from. We recommend consulting choosealicence.com and considering which license is most relevant to your project, copying the relevant licence text, and producing a licence.txt (often given as a LICENSE.md file) to add to your project alongside your code. In some repositories, such as Zenodo, you can specify the licence when you choose to archive your code, which will then be attached to the specific project without the need to create your own file.**

## Annotated

Annotation refers to adding comments within each code file (e.g., denoted with a "#" in R and Python) or embedding code within an R Markdown or Quarto document alongside descriptive text (Figure 6; see also https://eivimeycook.github.io/TADA/ for an example of a RMarkdown

document that combines code, code output, and descriptive text in one document). Annotation can dramatically improve the ability for someone else to understand (transparency) and run archived code (functionality and reproducibility). Annotation, similar to documentation, corresponds to the FAIR principle of Reusable (anyone can understand how to reuse and attribute the code) but rather than describing the code 'externally', provides clear commenting within each coding script.

In addition to script headers, which describe what the code is doing and what it will produce at the very top of the script, logical sections of code can be broken into 'chunks', which can be annotated to include informative details. Such as, what the chunk is doing (e.g., "# Run a Poisson generalised linear model…"), why it is needed (e.g., "…to analyse caterpillar abundance varying with habitat…"), and provide signposting for the locations of specific results in the manuscript body (when applicable; e.g., "Numeric results shown in Caterpillar Abundance section" or "Figure 5A"). Although annotation can be done line by line, simply denoting and describing relevant code chunks in sufficient detail is often more helpful for tracking what code does and what it produces (Note, "#####" or "#----" in RStudio or "#%%" in VSCode or Spyder (a Python IDE) creates collapsible sections in your code that increase readability and facilitate structuring). In Figure 2, the pre-TADA code has no internal annotation, and thus it remains unclear what is being run, why it is run, and what it produces (i.e., there is no signposting). Several useful resources provide additional information on producing clean, well annotated code (Filazzola & Lortie, 2022; Cooper & Hsing, 2025).

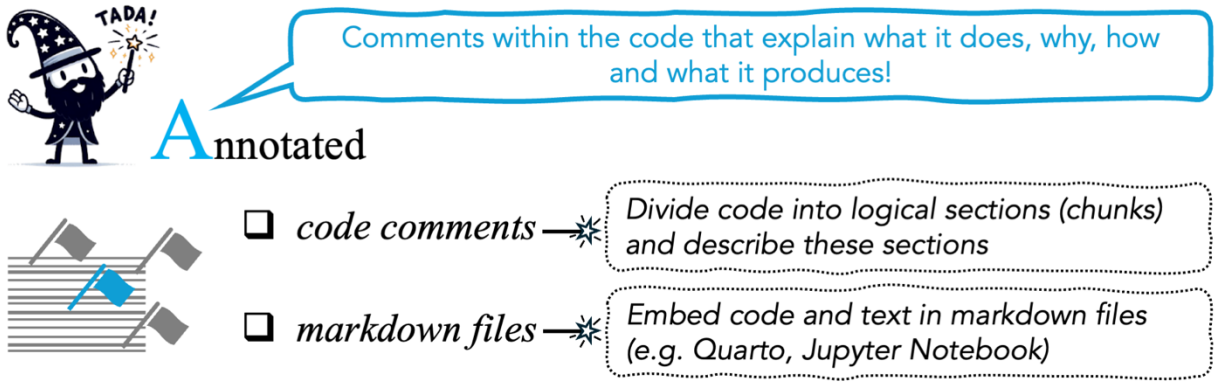



**Figure 6.** Summary of advice on making analytical code *Annotated*.

**<u>Annotated How to (See also Figure 6):</u> Annotation in both R and Python is done by simply providing a # (hashtag) before writing text. We also recommend producing a script header at the top of the code describing what it will do and what it will produce in addition to annotating code chunks (instead of every line of code). Each code chunk annotation should briefly include a description of what the code is doing, why, and if it produces any results in the manuscript. An example annotation is given in Figure 2. Alternatively, users could provide annotated code embedded within a RMarkdown or Quarto file, or using IDEs such as a Jupyter Notebook.**

## *Beyond the author: The role of journals, publishers, and institutions.*

The TADA guidelines provided here offer a simple author-centric guide to producing functional code that better enables reproducible research. However, there are multiple other stakeholders that need to play their part to ensure credible research across the broader research ecosystem. As noted by Ivimey-Cook *et al.*, (2025), journals and publishers need to have consistent requirements for code sharing that are both easy to locate and understand. Importantly, these

requirements should be implemented and monitored by journals via dedicated editorial staff such as data editors which can ensure a minimum quality of archived code; ideally following the new standardised data editor guidelines by Pick *et al.*, (2026). Authors, on the advice of journals and publishers, can then use TADA to ensure their code meets all the requirements of Stage 3 of these data editor guidelines prior to submission. For institutions, embedding open science education in under- and postgraduate courses is essential to raise awareness of reproducible research (Kohrs *et al.*, 2023). TADA has been designed to be memorable, easy-to-follow, and accessible to beginner coders. As such, these guidelines are ideally suited to be taught in courses and modules introducing students to the principles of open science and code sharing, for instance, data science. To aid this, we have provided the checklist in a number of different languages (Figure S1). In addition, students can be introduced to code review, the act of reviewing each other's code (Ivimey-Cook *et al.*, 2023), and check adherence to the data editor guidelines mentioned above (Pick *et al.*, 2026).

## *Conclusion*

By following the TADA guidelines, which are easy to understand, easy to remember, and which embody the FAIR principles, researchers at all coding levels will be better equipped to produce functional and transparent analytical code. TADA complements new guidelines for data editors across ecology and evolutionary biology (Pick *et al.*, 2026) to support the computational reproducibility of results. Importantly, not only will TADA help in the short-term, through improved code functionality, but it will also enable better long-term persistence when researchers return to their code, through improved annotation, documentation, and transparency. Through the use of TADA, combined with improved editorial practices at journals (e.g., the

presence of data editors at journals; Ivimey-Cook *et al.*, 2025; Pick *et al.*, 2026; and pre-submission code reviews; Ivimey-Cook *et al.*, 2023), we hope that the rate and quality of code sharing will continue to increase in ecology and evolutionary biology. Furthermore, while our advice for implementing TADA is tailored towards common practices in ecology and evolutionary biology, the core foundational goals of transferability, availability, documentation, and annotation are broadly applicable across research disciplines. We encourage researchers across disciplines to adapt and apply these core principles to support widespread adoption of open science practices.

## *References*


Abdill, R.J., Talarico, E. & Grieneisen, L. 2024. A how-to guide for code sharing in biology. *PLoS Biol* **22**: e3002815.

Allen, C. & Mehler, D.M.A. 2019. Open science challenges, benefits and tips in early career and beyond. *PLOS Biology* **17**: e3000246.

Barker, M., Chue Hong, N.P., Katz, D.S., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., *et al.* 2022. Introducing the FAIR Principles for research software. *Sci Data* **9**: 622.

Barnes, N. 2010. Publish your computer code: it is good enough. *Nature* **467**: 753–753.

Braga, P.H.P., Hébert, K., Hudgins, E.J., Scott, E.R., Edwards, B.P.M., Sánchez Reyes, L.L., *et al.* 2023. Not just for programmers: How GitHub can accelerate collaborative and reproducible research in ecology and evolution. *Methods Ecol Evol* **14**: 1364–1380.

Brun, J., Lyon, N.J., Chen, A., Slette, I., De La Rosa, G., Caselle, J.E., *et al.* 2025. Enabling data-driven collaborative and reproducible environmental synthesis science. *Methods in Ecology and Evolution* **16**: 1061–1074.

Cadwallader, L. & Hrynaszkiewicz, I. 2022. A survey of researchers' code sharing and code reuse practices, and assessment of interactive notebook prototypes. *PeerJ* **10**: e13933. PeerJ Inc.

Campbell, T., Dixon, K.W. & Handcock, R.N. 2023. Restoration and replication: a case study on the value of computational reproducibility assessment. *Restoration Ecology* **31**: e13968.

Cao, H., Dodge, J., Lo, K., McFarland, D.A. & Wang, L.L. 2023. The Rise of Open Science: Tracking the Evolution and Perceived Value of Data and Methods Link-Sharing Practices. arXiv.

Chen, D. 2023. pyprojroot.

Chue Hong, N.P., Katz, D.S., Barker, M., Lamprecht, A.-L., Martinez, C., Psomopoulos, F.E., *et al.* 2022. FAIR Principles for Research Software (FAIR4RS Principles). , doi: 10.15497/RDA00068. Zenodo.

contributors, conda. 2026. conda: A system-level, binary package and environment manager running on all major operating systems and platforms. , doi: 10.5281/zenodo.20723337. Zenodo.

Cooper, N. 2017. A Guide to Reproducible Code in Ecology and Evolution. British Ecological Society.

Cooper, N., & BES Data and Code Hackathon Group (2026). Data- and code-archiving in the British Ecological Society journals: Present status and recommendations for future improvements. Methods in Ecology and Evolution, 17, 1954–1966.

Cooper, N. & Hsing, P.-Y. 2025. *Guide to reproducible code*. British Ecological Society.
Culina, A., van den Berg, I., Evans, S. & Sánchez-Tójar, A. 2020. Low availability of code in ecology: A call for urgent action. *PLoS Biol* **18**: e3000763.
Di Tommaso, P., Chatzou, M., Floden, E.W., Barja, P.P., Palumbo, E. & Notredame, C. 2017. Nextflow enables reproducible computational workflows. *Nat Biotechnol* **35**: 316–319.
Eglen, S.J., Marwick, B., Halchenko, Y.O., Hanke, M., Sufi, S., Gleeson, P., *et al.* 2017. Toward standard practices for sharing computer code and programs in neuroscience. *Nat Neurosci* **20**: 770–773.
Eynden, V.V.D., Knight, G., Vlad, A., Radler, B., Tenopir, C., Leon, D., *et al.* 2016. Survey of Wellcome researchers and their attitudes to open research. *Wellcome Trust*, doi: 10.6084/m9.figshare.4055448.v1. Wellcome Trust.
Ferguson, J., Littman, R., Christensen, G., Paluck, E.L., Swanson, N., Wang, Z., *et al.* 2023. Survey of open science practices and attitudes in the social sciences. *Nat Commun* **14**: 5401.
Fernández-Juricic, E. 2021. Why sharing data and code during peer review can enhance behavioral ecology research. *Behav Ecol Sociobiol* **75**: 103.
Fidler, F., Chee, Y.E., Wintle, B.C., Burgman, M.A., McCarthy, M.A. & Gordon, A. 2017. Metaresearch for Evaluating Reproducibility in Ecology and Evolution. *BioScience* **67**: 282–289.
Filazzola, A. & Lortie, C. 2022. A call for clean code to effectively communicate science. *Methods Ecol Evol* **13**: 2119–2128.
Gao, M., Ye, Y., Zheng, Y. & Lai, J. 2025. A comprehensive analysis of R's application in ecological research from 2008 to 2023. *Journal of Plant Ecology* **18**: rtaf010.
Goldacre, B., Morton, C.E. & DeVito, N.J. 2019. Why researchers should share their analytic code. *BMJ* **367**: l6365.
Gomes, D.G.E., Pottier, P., Crystal-Ornelas, R., Hudgins, E.J., Foroughirad, V., Sánchez-Reyes, L.L., *et al.* 2022. Why don't we share data and code? Perceived barriers and benefits to public archiving practices. *Proc. R. Soc. B.* **289**: 20221113. Royal Society.
Hillemann, F. [freddy], Burant, J.B., Culina, A. & Vriend, S.J.G. 2025. Code review in practice: A checklist for computational reproducibility and collaborative research in ecology and evolution. EcoEvoRxiv.
Ivimey-Cook, E.R., Pick, J.L., Bairos-Novak, K.R., Culina, A., Gould, E., Grainger, M., *et al.* 2023. Implementing code review in the scientific workflow: Insights from ecology and evolutionary biology. *Journal of Evolutionary Biology* **36**: 1347–1356.
Ivimey-Cook, E.R., Sánchez-Tójar, A., Berberi, I., Culina, A., Roche, D.G., A. Almeida, R., *et al.* 2025. From policy to practice: progress towards data- and code-sharing in ecology and evolution. *Proc Biol Sci* **292**: 20251394.
Jiménez, R.C., Kuzak, M., Alhamdoosh, M., Barker, M., Batut, B., Borg, M., *et al.* 2017. Four simple recommendations to encourage best practices in research software. *F1000Res* **6**: ELIXIR-876.
Kambouris, S., Wilkinson, D.P., Smith, E.T. & Fidler, F. 2024. Computationally reproducing results from meta-analyses in ecology and evolutionary biology using shared code and data. *PLOS ONE* **19**: e0300333.
Kang, D., Kang, T. & Jang, J. 2023. Papers with code or without code? Impact of GitHub repository usability on the diffusion of machine learning research. *Information Processing & Management* **60**: 103477.
Kellner, K.F., Doser, J.W. & Belant, J.L. 2025. Functional R code is rare in species distribution and abundance papers. *Ecology* **106**: e4475.
Kimmel, K., Avolio, M.L. & Ferraro, P.J. 2023. Empirical evidence of widespread exaggeration bias and selective reporting in ecology. *Nat Ecol Evol* **7**: 1525–1536.
Kohrs, F.E., Auer, S., Bannach-Brown, A., Fiedler, S., Haven, T.L., Heise, V., *et al.* 2023. Eleven strategies for making reproducible research and open science training the norm at research institutions. *eLife* **12**: e89736. eLife Sciences Publications, Ltd.
König, L., Gärtner, A., Slack, H., Dhakal, S., Adetula, A., Dougherty, M., *et al.* 2025. How to bolster employability through open science. *OSF Preprints*, doi: 10.31219/osf.io/zgfkd_v1.
Köster, J. & Rahmann, S. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* **28**: 2520–2522.

Kurtzer, G.M., Sochat, V. & Bauer, M.W. 2017. Singularity: Scientific containers for mobility of compute. *PLOS ONE* **12**: e0177459..
Lai, J., Lortie, C.J., Muenchen, R.A., Yang, J. & Ma, K. 2019. Evaluating the popularity of R in ecology. *Ecosphere* **10**: e02567.
Lee, B.D. 2018. Ten simple rules for documenting scientific software. *PLOS Computational Biology* **14**: e1006561.
Maitner, B., Santos Andrade, P.E., Lei, L., Kass, J., Owens, H.L., Barbosa, G.C.G., *et al.* 2024. Code sharing in ecology and evolution increases citation rates but remains uncommon. *Ecology and Evolution* **14**: e70030.
McKiernan, E.C., Bourne, P.E., Brown, C.T., Buck, S., Kenall, A., Lin, J., *et al.* 2016. How open science helps researchers succeed. *eLife* **5**: e16800.
Merkel, D. 2014. Docker: lightweight Linux containers for consistent development and deployment. *Linux J.* **2014**: 2:2.
Mislan, K.A.S., Heer, J.M. & White, E.P. 2016. Elevating The Status of Code in Ecology. *Trends in Ecology & Evolution* **31**: 4–7.
Mitra-Behura, S., Fiolka, R.P. & Daetwyler, S. 2022. Singularity Containers Improve Reproducibility and Ease of Use in Computational Image Analysis Workflows. *Front. Bioinform.* **1**.
Müller, K. & Bryan, J. 2020. here: A Simpler Way to Find Your Files.
National Academies of Sciences, E., Affairs, P. and G., Committee on Science, E., Information, B. on R.D. and, Sciences, D. on E. and P., Statistics, C. on A. and T., *et al.* 2019. Reproducibility. In: *Reproducibility and Replicability in Science*. National Academies Press (US).
Patel, B., Soundarajan, S., Ménager, H. & Hu, Z. 2023. Making Biomedical Research Software FAIR: Actionable Step-by-step Guidelines with a User-support Tool. *Sci Data* **10**: 557.
Pick, J.L., Allen, B.J., Bachelot, B., Bairos-Novak, K.R., Brand, J.A., Class, B., *et al.* 2026. The SORTEE guidelines for data and code quality control in ecology and evolutionary biology. *Peer Community Journal* **6**.
Powers, S.M. & Hampton, S.E. 2019. Open science, reproducibility, and transparency in ecology. *Ecological Applications* **29**: e01822.
Purgar, M., Klanjscek, T. & Culina, A. 2022. Quantifying research waste in ecology. *Nat Ecol Evol* **6**: 1390–1397.
Rokem, A. 2024. Ten simple rules for scientific code review. *PLOS Computational Biology* **20**: e1012375.
Sánchez-Tójar, A., Bezine, A., Purgar, M. & Culina, A. 2025. Code-sharing policies are associated with increased reproducibility potential of ecological findings. *Peer Community Journal* **5**.
Sandve, G.K., Nekrutenko, A., Taylor, J. & Hovig, E. 2013. Ten Simple Rules for Reproducible Computational Research. *PLOS Computational Biology* **9**: e1003285.
Sharma, N.K., Ayyala, R., Deshpande, D., Patel, Y., Munteanu, V., Ciorba, D., *et al.* 2024. Analytical code sharing practices in biomedical research. *PeerJ Comput. Sci.* **10**: e2066. PeerJ Inc.
Trisovic, A., Lau, M.K., Pasquier, T. & Crosas, M. 2022. A large-scale study on research code quality and execution. *Sci Data* **9**: 60.
Ushey, K. 2023. renv: Project Environments.
Vandewalle, P. 2012. Code Sharing Is Associated with Research Impact in Image Processing. *Comput. Sci. Eng.* **14**: 42–47.
Wilkinson, M.D., Dumontier, M., Aalbersberg, Ij.J., Appleton, G., Axton, M., Baak, A., *et al.* 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3**: 160018.