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.

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, Reusable*) principles, or are complex and primarily developed for software. 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 complements the FAIR principles. TADA aims to streamline the process of archiving and sharing functional code for researchers across all levels of coding experience, with the goal of increasing transparency, reproducibility, and the reliability of research results. Although these guidelines were developed based on our experience in Ecology and Evolutionary Biology, we believe they will be useful to researchers in other disciplines.

Keywords

Research integrity, Reliability, Replicability, Reproducibility, 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 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 more efficiently build upon published work (Barnes, 2010; Eglen *et al.*, 2017). Furthermore, code, alongside 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, 2019) - a key part of the scientific process that promotes reliability and builds trust in research (Fidler *et al.*, 2017; Powers & Hampton, 2019). 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 before manuscript publication, or in some cases, upon first submission. Ideally, open code should follow the FAIR principles, which were initially published for data in 2016 (Wilkinson *et al.*, 2016) and later adapted for Research Software in 2022 (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 appear to be 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 from between 5 and 33% (Culina *et al.*, 2020; Kimmel *et al.*, 2023; Kambouris *et al.*, 2024; Maitner *et al.*, 2024; Kellner *et al.*, 2025; Sánchez-Tójar *et al.*, 2025). Second, even when code is provided, its functionality (i.e., the ability to run code without error) is often low (Trisovic *et al.*, 2022; Kellner *et al.*, 2025). In a recent study examining R code in research articles analysing species distribution and abundance, the authors had to abandon the reproducibility aspect of their analysis due to the overwhelmingly high proportion of code that did not run or ran with errors (93% of coding scripts; Kellner *et al.*, 2025). Similarly, a recent review of over 9000 unique R files shared in the Harvard Dataverse repository found that 74% of code failed to complete without error, which only decreased to 56% after code cleaning was applied (e.g., removal of local file paths and ensuring libraries and dependencies were properly installed and loaded; Trisovic *et al.*, 2022). Finally, 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). For instance, the ability to reproduce the results of meta-analyses in ecology and evolutionary biology has been shown to range 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 rates 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 likely not being achieved.

A major reason for the limited availability and functionality of code and, therefore, low rates of computational reproducibility, might be 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), these resources are not focused on how to practically archive functional code used for analyses in research articles (analytical code). Few refer to 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 towards software developers, potentially explaining why they have not been widely adopted.

The term ‘code reusability’ is often used in two different contexts. In the context of 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 README file. In a software development context, designing code for reuse is a far more complicated 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 in order 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 that enable code to align with FAIR principles and minimum standards for transparency and computational reproducibility. is an important first 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 FAIR4RS principles in mind but tailored to analytical code for research. We call these guidelines *TADA!* (Transferable, Available, Documented, Annotated) and believe they will help researchers at all coding levels prepare functional code that facilitates reproducible and transparent research which will help to build trust in published results.

Figure 1. An example of the TADA guidelines (Transferable, Available, Documented, 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 (dark green), Documented (purple), and Annotated (blue). The code shown is generic and designed to showcase the TADA guidelines. Figure by EIC.

TADA!

We outline below four easy-to-follow steps to help researchers share functional and transparent code. By following the TADA guidelines (Figures 2-5), a researcher can produce analytical code that follows best practices, aligns with the FAIR and FAIR4RS principles (Wilkinson *et al.*, 2016; Barker *et al.*, 2022; Chue Hong *et al.*, 2022), increases transparency, and facilitates computational reproducibility. TADA is tailored mainly to R and Python, as these open-source 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 coding languages including workflow (e.g., Snakemake) and compiled languages (e.g., C++). Furthermore, whilst we provide guidance in the context of research in ecology and evolutionary biology, TADA can be applied broadly across other disciplines. For a checklist of the TADA guidelines, see Figure S1.

Transferable

Transferability refers to the ability for anyone to open the file, view and run the code without conversion or alteration (Figure 2). This includes the FAIR principle of interoperability (a simple definition implies that anyone will be able to open and use your code) and extends it to allow code to be *run* on different computers and operating systems. Ensuring transferability greatly increases the computational reproducibility of results from analytical code. First, code must be saved and encoded in a file type that can be opened by any text editor or integrated development environment (IDE; e.g., RStudio, VSCode, PyCharm). In Figure 1, 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 to a different file type. Importantly, copying and pasting code from certain file types (i.e., .PDF or .docx) may lead to changes in characters (e.g., apostrophes) or white spaces, or the inclusion of additional, unwanted characters (e.g., line numbers, headers), which can easily result in code errors that are sometimes difficult to spot or time consuming to fix. We suggest saving code in an interoperable 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 text in a manner that readily allows for copying and pasting without the aforementioned issues, it can lead to issues with interpretation of the coding language 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 code runs on different computers and operating systems, file paths must be written in a way that is not specific to the user's local environment or directory structure (i.e., local or user-specific file paths as opposed to relative file paths). Importantly, data, code, and all necessary materials should be organised in a single project directory. To avoid local file paths, one can use an RStudio project, which automatically sets the working directory to the appropriate location (e.g., a project folder), alongside packages such as *here* (Müller & Bryan, 2020) or *pyprojroot* (Chen 2023), which create file paths relative to any project directory regardless of operating system (i.e., relative file paths). This will ultimately avoid the use of the setwd() function (in R), or the os.chdir() function (in Python), which set both operating system and user-specific file paths that can cause other users to encounter errors when running the code. For other software, simply opening the project folder (in VSCode) or launching R (when standalone without an IDE) within the project directory performs a similar action to using an RStudio project. In Figure 1, the use of local and 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 (assuming it exists). Although beyond the scope of this paper, reproducible analyses can also be supported by containerisation and workflow managers. Containerisation platforms such as Docker (Merkel, 2014) and 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., 2021). Workflow managers such as Snakemake (Koster & 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).

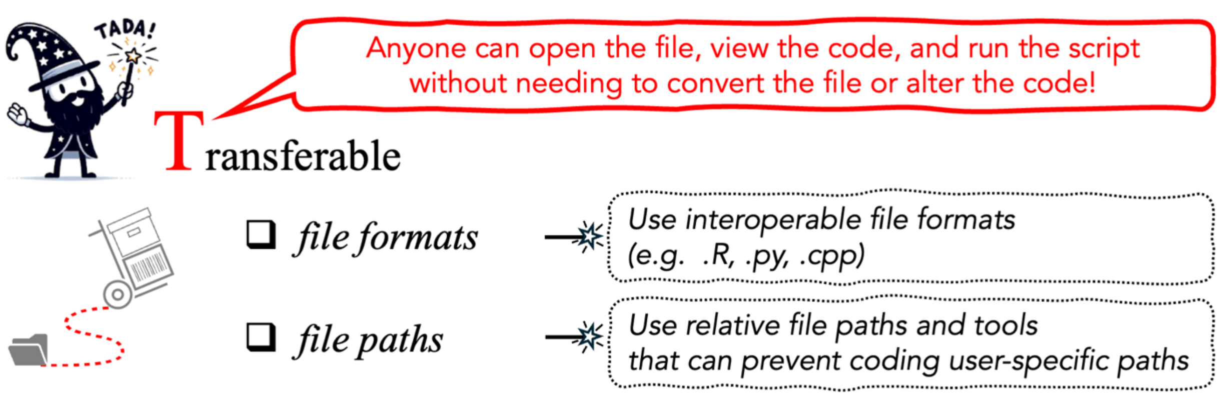

Figure 2. Summary of advice on making analytical code *Transferable*. Figure by ML.

<u>Transferability How To (See also Figure 2):</u> 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 local file paths 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 local file paths. RStudio projects can be used in combination or separately from using packages such as *here*. We recommend using both to maximise transferability across operating systems. Additional methods include navigating to the project file and opening it within VSCode or running an instance of R or Python within the specific project folder. The latter will remove the need for local file paths that may 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).

Available

Availability refers to the act of publicly archiving the code in a way that provides long-term to any external user (Figure 3). Available, in this context covers 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 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 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 a particular publication. Repositories such as Zenodo (which can connect to a GitHub project) and Figshare 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, which can preserve GitHub projects for long-term storage and provides PIDs in the form of Software Hash Identifiers (SWHIDs). In Figure 1, 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.

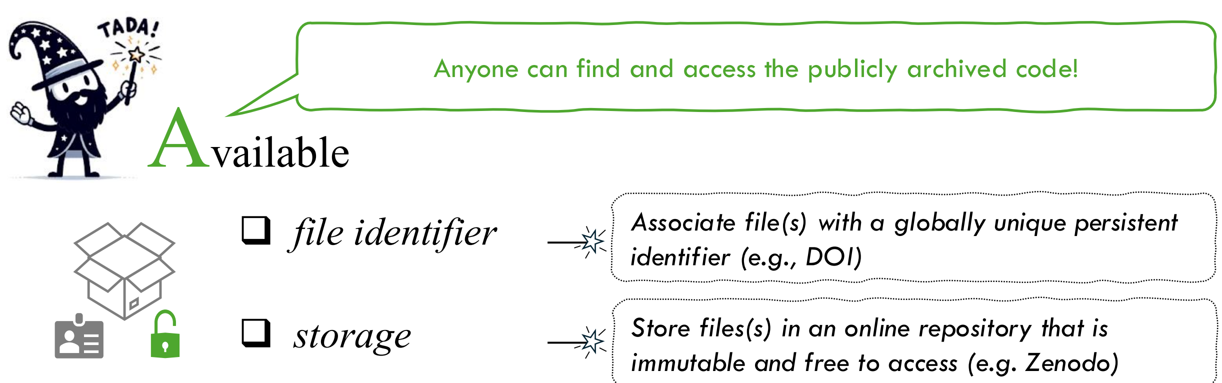

Figure 3. Summary of advice on making analytical code *Available*. Figure by ML.

<u>Availability How To (See also Figure 3):</u> Upload your code to Zenodo (https://zenodo.org/) or 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 code, and a new one whenever it is subsequently updated (known as DOI versioning). Assigning a DOI facilitates citing the code and linking to it in the related manuscript. GitHub is not ideal to archive and share analytical code associated with a paper because it is not immutable and does not generate a DOI. Instead, users can create a release version on GitHub and link to Zenodo (see https://help.zenodo.org/docs/profile/linking-accounts/ for more information regarding linking projects).

Documented

Documentation refers to providing accurate and detailed metadata files that describe the code files and their usage (Figure 4). This documentation is often provided as an additional .txt file or .md file (typically a README.txt or markdown file). Documentation could be provided as a combined README containing both code- and data-specific metadata, or as two separate READMEs, one for code and one for data, if relevant. Figure 1 shows an example of essential information that should be contained within a README file. This includes information related to the author of the code along with some form of contact information, as well as the title of the corresponding manuscript and any relevant funders. In addition, the computational environment used, such as software version (e.g., R v.4.3.3), packages with associated versions (e.g., *ggplot* v2.3.2; this could also be provided alongside a text file which lists every loaded package and version number; given by sessionInfo() in R or session-info in Python), licences (e.g., MIT licence), and the data-specific PID or other important information as to where the relevant data are located 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 specify an appropriate licence detailing how others can use, modify and share the code. Licences can take many forms, such as the Massachusetts Institute of Technology (MIT) or General Public Licence (GPL) and can differ in their permission levels and conditions. For instance, the licence 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 licences, which has 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). 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 (e.g., Dryad only supports the CC0 licence, which is not best suited for code). Websites such as choosealicence.com provide detailed guidance on selecting a licence (although, in essence, it can simply involve copying the respective license text and saving the file to the project). 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 in the Journal of Statistical Software). Figure 1 illustrates the implications of licencing 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.

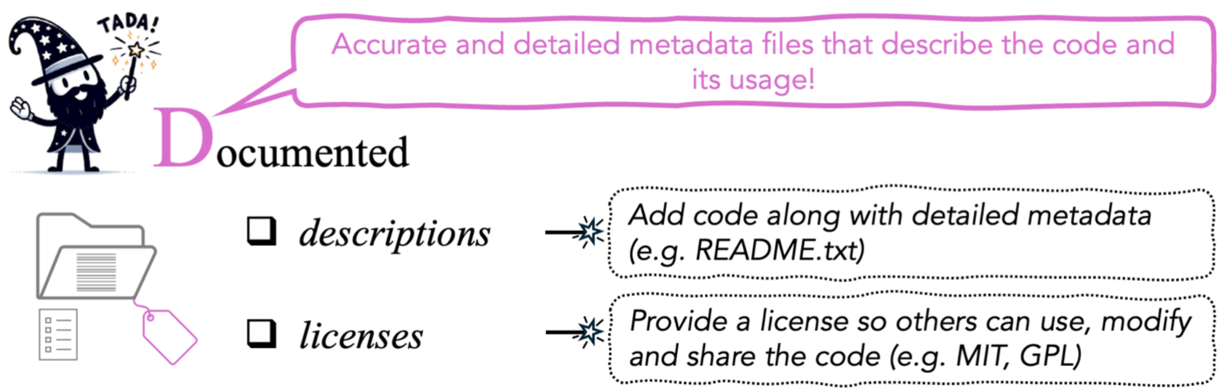

Figure 4. Summary of advice on making analytical code *Documented*. Figure by ML.

<u>Documented How To (See also Figure 4):</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 of the manuscript, any relevant funders, and 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 and what data it produces, if any.

For licences, as mentioned above, 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 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 RMarkdown or Quarto document alongside descriptive text (Figure 5; see also https://eivimeycook.github.io/TADA/) and can dramatically improve the ability for someone else to understand (transparency) and run archived code (functionality and reproducibility). 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, "#####" in RStudio or "#%%" in Python creates collapsible sections in your code that increase readability and facilitate structuring). In Figure 1, 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).

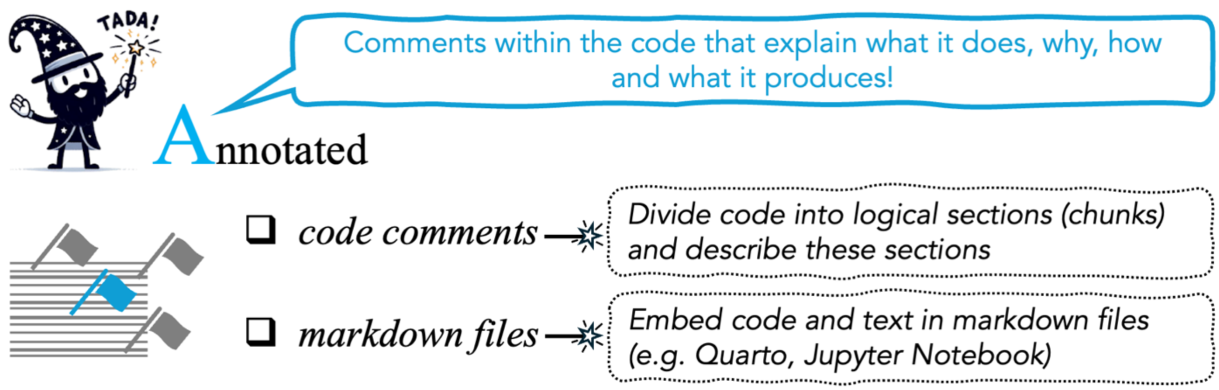

Figure 5. Summary of advice on making analytical code *Annotated*. Figure by ML.

<u>Annotated How to (See also Figure 5):</u> Annotation in both R and Python is done by simply providing a # (hashtag) before writing text. We recommend annotating code chunks instead of every line of code. Each 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 1. Alternatively, users could provide annotated code embedded within a RMarkdown or Quarto file, or using IDEs such as a Jupyter Notebook.

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 to support computational reproducibility. 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.*, 2025, 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 transparency, availability, documentation, and annotation are broadly applicable across research disciplines. We encourage researchers to adapt and apply these core principles beyond ecology and evolutionary biology, to support widespread adoption of open science practices.

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, AS-T are past board members. ML is a current board member.

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, AS-T, SMW, and JLP) contributed to reviewing and editing subsequent drafts.

AI declaration

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

Data availability

No data was used in this paper.

Funding

AC was supported by the Croatian Science Foundation under the project number HRZZ-IP-2022-10-2872. AST 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.

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. Public Library of Science.

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. Nature Publishing Group.

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 <scp>GitHub</scp> can accelerate collaborative and reproducible research in ecology and evolution. *Methods Ecol Evol* **14**: 1364–1380.

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.

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.

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

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. Public Library of Science.

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. Nature Publishing Group.

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. British Medical Journal Publishing Group.

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., Almeida, R.A., *et al.* 2025. From Policy to Practice: Progress towards Data- and Code-Sharing in Ecology and Evolution. EcoEvoRxiv.

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. F1000Research.

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. Public Library of Science.

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. Nature Publishing Group.

König, L., Gärtner, A., Slack, H., Dhakal, S., Adetula, A., Dougherty, M., *et al.* 2025. How to bolster employability through open science. OSF.

Lai, J., Lortie, C.J., Muenchen, R.A., Yang, J. & Ma, K. 2019. Evaluating the popularity of R in ecology. *Ecosphere* **10**: e02567.

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. eLife Sciences Publications, Ltd.

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.

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. Understanding Reproducibility and Replicability. 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. Nature Publishing Group.

Pick, J.L., Bairos-Novak, K.R., Bachelot, B., Brand, J.A., Class, B., Dallas, T., *et al.* 2025. The SORTEE Guidelines for Data and Code Quality Control in Ecology and Evolutionary Biology.

Powers, S.M. & Hampton, S.E. 2019. Open science, reproducibility, and transparency in ecology. *Ecological Applications* **29**: e01822.

Chen. D., *pyprojroot*: Project-oriented workflow in Python. 2023.

Rokem, A. 2024. Ten simple rules for scientific code review. *PLOS Computational Biology* **20**: e1012375. Public Library of Science.

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. Public Library of Science.

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. Nature Publishing Group.

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. Nature Publishing Group.