TADA! Simple guidelines to improve code sharing

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], Alfredo Sánchez-Tójar[3], Saras M. Windecker[10], Joel L. Pick[11]

1 School of Biodiversity, One Health, and Veterinary Medicine, University of Glasgow, 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 The Kids Research Institute Australia, Nedlands, WA, Australia; 11 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 important for transparency and facilitates computational reproducibility of published research. However, even as the number of journals that encourage or mandate code sharing continues to increase, the prevalence of open code remains low. Furthermore, even when shared, code is often non-functional, which hinders computational reproducibility. One reason for low levels of code sharing is uncertainty around how to prepare functional (i.e., the ability to run code without error) and reproducible (i.e., the ability to reproduce the analysis and results using the same data, code, and computational conditions) code as existing principles for best coding practices are both complex and primarily developed for software. To improve code sharing, there is an urgent need for clear and simple guidance on how to prepare functional and reproducible code for sharing. To address this, we provide simple code sharing guidelines: TADA (*Transferable, Accessible, Documented and Annotated*). TADA details the minimum requirements necessary for a researcher to produce functional and reproducible code for sharing that directly supports open science best practices and the FAIR (*Findable, Accessible, Interoperable, Reusable*) principles for code. TADA aims to streamline the process of depositing and sharing functional code for researchers with all levels of coding experience, with the ultimate goal of increasing the transparency, reproducibility, and reliability of research results across ecology and evolution, and more broadly.

Keywords

Research integrity, Reliability, Replicability, Reproducibility, Research methods, Methodological rigour

Introduction

Public sharing of code (i.e., open code) offers numerous benefits for researchers. It enhances the transparency of methods and the overall research process (Goldacre *et al.*, 2019; Fernández-Juricic, 2021; Ivimey-Cook *et al.*, 2023), increases the citation rates of associated articles (Vandewalle, 2012; Maitner *et al.*, 2024), enables other researchers to build upon published work (Barnes, 2010; Eglen *et al.*, 2017), and can provide future career advantages for early career researchers (McKiernan *et al.*, 2016; Allen & Mehler, 2019; König *et al.*, 2025). Furthermore, code, alongside data, is essential for ensuring computational reproducibility - the ability to reproduce the analysis and results using the same data, code, and computational conditions (National Academies of Sciences *et al.*, 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 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), where authors are encouraged or required to share code following manuscript acceptance, or in some cases, upon first submission.

To date, several recommendations exist for how to prepare and archive code to facilitate review and computational reproducibility (Sandve *et al.*, 2013; Cooper, 2017; Filazzola & Lortie, 2022; Ivimey-Cook *et al.*, 2023; Abdill *et al.*, 2024; Rokem, 2024; Sharma *et al.*, 2024; Hillemann *et al.*, 2025). These guidelines aim to follow the FAIR principles, which were published for data in 2016 (Wilkinson *et al.*, 2016) and later adapted for Research Software in 2022 (FAIR4RS;

Barker *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 every user must understand how they can be accessed which may or may not require 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 attributed alongside an appropriate licence (see Wilkinson *et al.*, 2016; Barker *et al.*, 2022).

Despite these guidelines and progress towards more transparent and reproducible research (Cao *et al.*, 2023), there are still clear limitations with code sharing. First, the proportion of articles with open code remains alarmingly low. For example, in ecology and evolution, rates range from between 5 - 32% (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 about 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 archived in the Harvard Dataverse found that 74% of code failed to complete without error, which 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 has been shown to range from 26.9% (all results within an article exactly matched) to 73.1% (50% of results within an article were within 10% of the original value) when data and code were shared and available (Kambouris *et al.*, 2024). Putting these three components together (the low rates of code archiving, low functionality of archived code, and low computational reproducibility of functional archived code) presents a dismal picture, and suggests that many of the benefits of code sharing for both authors and the scientific community more broadly are not being achieved.

It has been suggested that a major reason for the limited functionality of code and, therefore, low rates of computational reproducibility is a lack of knowledge on how to prepare code with transparency and reproducibility in mind (Gomes *et al.*, 2022). We suggest that complete guidelines, such as the FAIR4RS principles (Barker *et al.*, 2022), are too broad in scope and largely focused towards software developers, which may explain why they have not been widely adopted by the research community. Furthermore, analytical code is typically far more unique and tailored to a specific dataset as opposed to software code. The main goal of producing and sharing code in this case is not for *reuse* and provision of general analytical tools, but rather to produce a transparent and reproducible record of the analysis for a particular study. Therefore, we contend that there is a significant distinction between guidelines developed to ensure code *reuse* and those designed to ensure code *transparency* and *reproducibility.* Generating code for reuse is a far more complicated process than "simply" ensuring reproducibility, as code needs to be written in a generalised way to provide analytical tools that can run on any system with any appropriate data as input. Although extremely useful and important in ensuring best practices for data sharing and open-source software, current principles are thus likely setting too high a bar for

analytical research code that does not need to meet the level of reusable software. Therefore, an important first step to increase the rate and quality of code sharing is to lower this bar and to establish best practices for code to meet the minimum standards for transparency and computational reproducibility. This increase in transparency will help to build trust in published results, which should be the norm for all open analytical code. Here, we provide simplified and easy-to-follow guidelines built with the FAIR4RS principles in mind but tailored for sharing analytical code for research. We call these guidelines, *TADA* (Transferable, Accessible, Documented, Annotated) and believe it will help research coders of all levels to prepare functional, transparent, and reproducible code.

Figure 1. An example of the TADA guidelines (Transferable, Accessible, 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), Accessible (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 transparent and reproducible code. By following the TADA (Figs. 2-5) guidelines, a researcher can produce analytical code that follows open science best practices, aligns with the FAIR and FAIR4RS principles (Wilkinson *et al.*, 2016; Barker *et al.*, 2022), increases transparency, and facilitates computational reproducibility. We discuss each component in detail below. Whilst our advice is tailored mainly to R and Python, as these open-source languages are widely used, particularly in ecology and evolution (Mislan *et al.*, 2016; Lai *et al.*, 2019; Gao *et al.*, 2025), the basic principles of these guidelines can be widely applied to other coding languages. Furthermore, whilst we provide guidance from an ecology and evolution perspective, these guidelines 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 the code, and run the script without conversion or alteration (Fig. 2). Ensuring transferability greatly increases the computational reproducibility of research code. First, code must be saved in a file extension 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 or Word file. These files can be viewed but cannot be opened and edited within an IDE without using additional libraries (or software) or converting to a different file extension. Furthermore, copying and pasting code directly from these file extensions may result in changing characters (e.g., apostrophes) or white spaces, or the inclusion of other additional unwanted characters (e.g.,

line numbers, headers), which can easily result in code errors that are sometimes difficult to spot. We suggest saving any script in a transferable (often referred to as interoperable) file extension, such as .txt, .R or .py, as these can be readily viewed, edited and saved using any text editor or IDE.

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., absolute file paths). This is also inherently related to appropriate folder organisation, where data, code, and all necessary materials are organised in a single project directory. To avoid local (or absolute) 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 *pyrpojroot* (pyprojroot 2023), which creates file paths relative to any project directory regardless of operating system (i.e., relative file paths). By doing so, this will ultimately avoid the use of the setwd() function (in R) or the os.chdir() (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 operating system and user-agnostic and allows anyone to load the necessary data file.

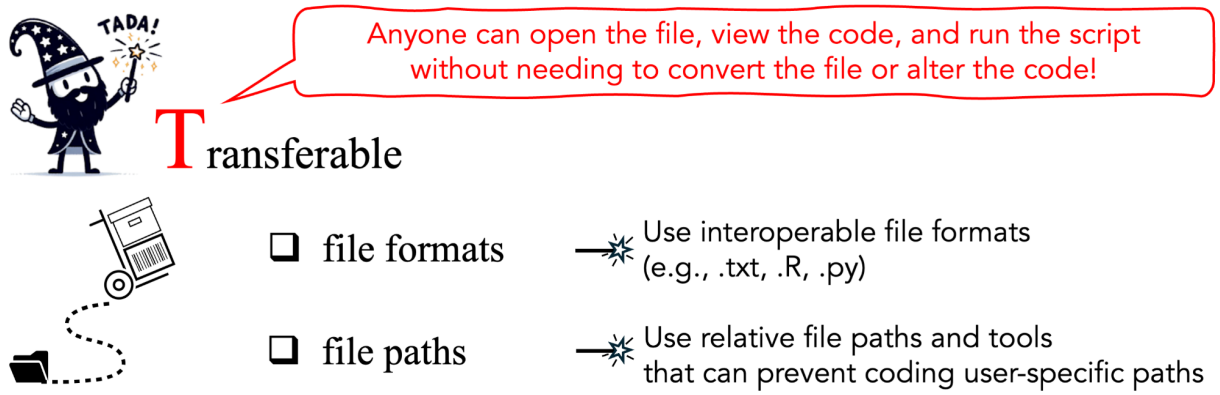

Figure 2. Summary of advice on making code sharing *Transferable*. Figure by ML.

<u>How to (Fig. 2):</u> When writing code in R or Python, be sure to save and share each script as a .txt, .R, or .py file extension. Avoid providing code within supplementary files or Word documents and PDFs. If the coding language or IDE does not use or save scripts or code syntax in the previously stated file extensions, 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 prevent the use of local file paths in your scripts. 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 user-specific file paths. This 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 repository's documentation (see below).

Accessible

Accessibility refers to the act of publicly archiving the code in a way that provides access to any external user (Fig. 3). Code must be stored in an open and easily accessible manner with an associated globally unique persistent identifier (e.g., a DOI), which must be cited in the corresponding manuscript to enable others to find and access the code. Whilst GitHub might be a commonly used repository 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 DOI and files can be changed (or even deleted) after archiving (i.e., GitHub is not immutable). As such, GitHub is not suitable for archiving analytical code used in a particular publication. Repositories such as Zenodo (which can connect to a GitHub repository), 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 by the owner. In Figure 1, the lack of archived code and associated DOI in the pre-TADA code limits code sharing and prevents permanent, uneditable, and citable storage of the code.

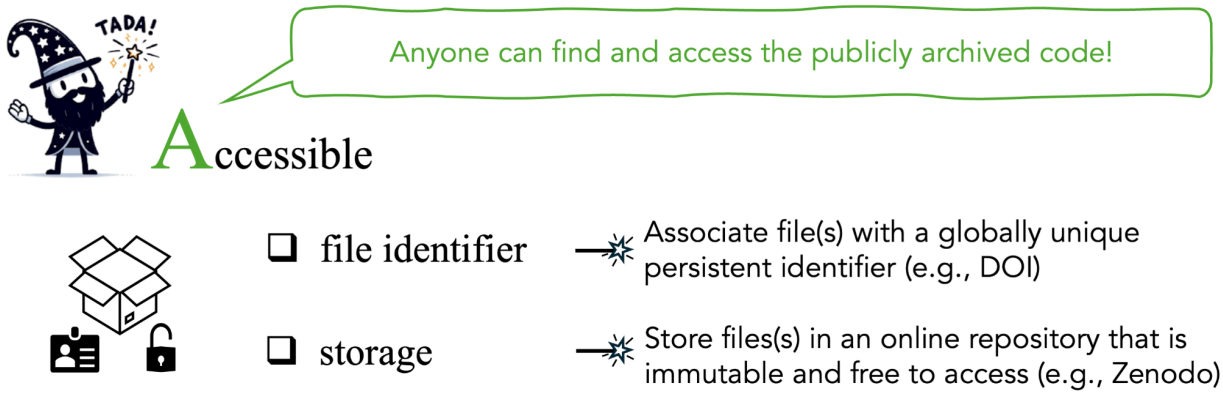

Figure 3. Summary of advice on making code sharing *Accessible*. Figure by ML.

How to (Fig. 3): 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 long-term persistence. A unique DOI will be created when the repository goes live, and a new one whenever it is subsequently updated (known as DOI versioning). Assigning a DOI facilitates citing and linking in the related manuscript. We do not recommend using GitHub as a standalone repository because it is not immutable and does not create 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 accounts).

Documented

Documentation refers to providing accurate and detailed metadata files that describe the code files and their usage (Fig. 4). This documentation is often provided as an additional .txt file (typically a README.txt). 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. Figure 1 provides additional essential information that should be contained within a README file. This includes information on the computational environment used, such as software version (e.g., R v4.3.3), packages with associated versions (e.g., *ggplot* v2.3.2), licences (e.g., MIT licence), and the data-specific DOI or other important information as to where the relevant data is located (e.g., doi.org/12345; see below), alongside any additional information needed to run the files (e.g., what each file contains, the order in which to run them and whether the code takes a long time to run).

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), whether code can be modified, and/or used for commercial purposes. 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 has a generic CC0 licence on all its repositories that is not suitable for code). Websites such as choosealicence.com provide detailed guidance on how to assign a licence to a repository (although, in essence, it can simply involve copying the respective license text and saving the file to the repository). Licences can range from completely open and permissive, such as MIT, which has little to no restrictions on use, to more restrictive, such as GPL, 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. 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 the option for collaboration (i.e., can others modify or extend your code), and how this aligns with journal, institutional and funding policies. For instance, some journals require the use of a specific licence upon archiving (e.g., GPL in the Journal of Statistical Software). In Figure 1, the pre-TADA code has no associated licence, which restricts its use as other users are not legally permitted to use, share, or modify the archived script. In contrast, the post-TADA code has an associated MIT licence, which tells users explicitly that they are free to copy, modify, merge, publish, and share the archived script.

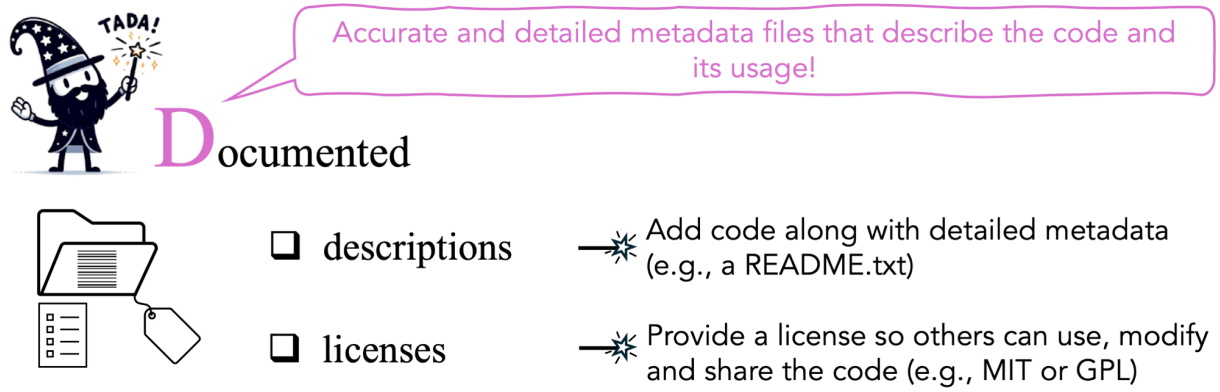

Figure 4. Summary of advice on making code sharing *Documented*. Figure by ML.

How to (Fig. 4): External documentation can provide important information that internal code annotation lacks. A README.txt file describing the code should contain additional information on the manuscript that the code is associated with (title, abstract, and authors with corresponding emails, including who wrote the code; if necessary this can be anonymised during review to adhere to double-blind reviewing policies), software used (e.g., R or Python including version number), any important libraries used (with version numbers; this should 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), information about where relevant data is located (if appropriate), a mention of the code-specific licence, and any other important pieces of information such as the order the scripts should be used and whether the code takes long time to run.

For licences, as mentioned above, there exists a multitude to choose from. Common licences that will suit code are MIT and GPL, but many more exist that differ in how permissive they are (see https://choosealicense.com/appendix/). We recommend consulting choosealicence.com, copying the relevant licence text, and producing a licence.txt file to add to your repository 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 repository without the need to create your own file.

Annotated

Annotation refers to the addition of comments within each script (e.g., denoted with a "#" in R and Python) or embedding code within an RMarkdown or Quarto document alongside descriptive text (Fig. 5; see also https://eivimeycook.github.io/TADA/). Annotation dramatically improves the ability for someone else to understand (transparency) and run the code (functionality and reproducibility). Annotation can include informative details such as what the section of code 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 in "Caterpillar Abundance" "). Although this could be line-by-line annotation, simply

denoting and describing relevant sections 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, which means that it is unclear what is being run, why it might be run, and ultimately what it produces (i.e., no signposting).

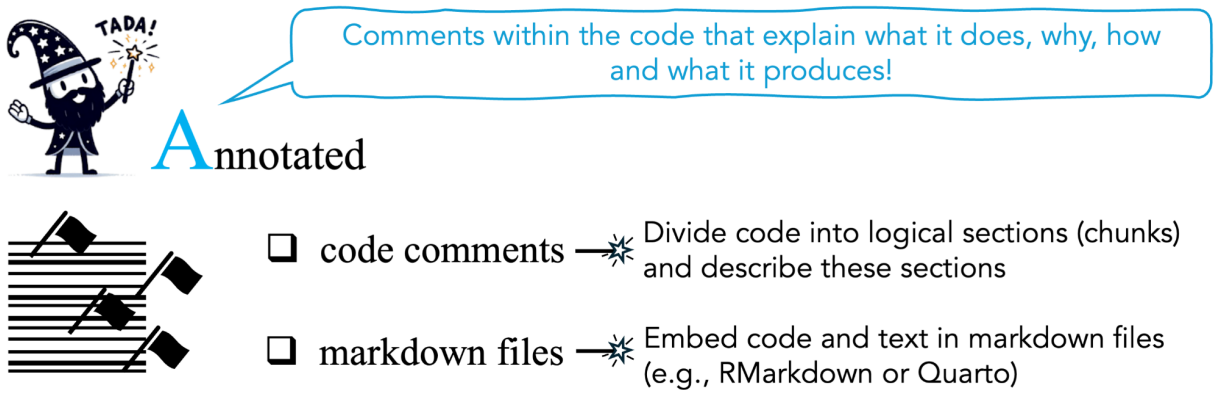

Figure 5. Summary of advice on making code sharing *Annotated*. Figure by ML.

<u>How to (Fig. 5):</u> Annotation in both R and Python is done by simply providing a # (hashtag) before writing text. We recommend that, rather than annotating every line of code, to annotate each code 'chunk', where multiple lines of code are described in sufficient detail. Each comment 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 Fig. 1. Alternatively, as mentioned above, a user could provide annotated code embedded within an RMarkdown or Quarto file, which could be shared.

Conclusion

By following these simple guidelines, which are both easy to understand, easy to remember, and which embody the FAIR principles, code creators of all experience levels will be better equipped to produce transparent and reproducible analytical code. 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 evolution. Furthermore, while our advice for implementing TADA is tailored towards common practices in ecology and evolution, the core foundational goals of transparency, accessibility, documentation, and annotation are broadly applicable across research disciplines. We encourage researchers to adapt and apply these core principles beyond ecology and evolution, 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 acting President. EIC, ML and AS-T are current board members.

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 of subsequent drafts.

AI declaration

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

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.

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

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).

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.

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.

Supplementary material

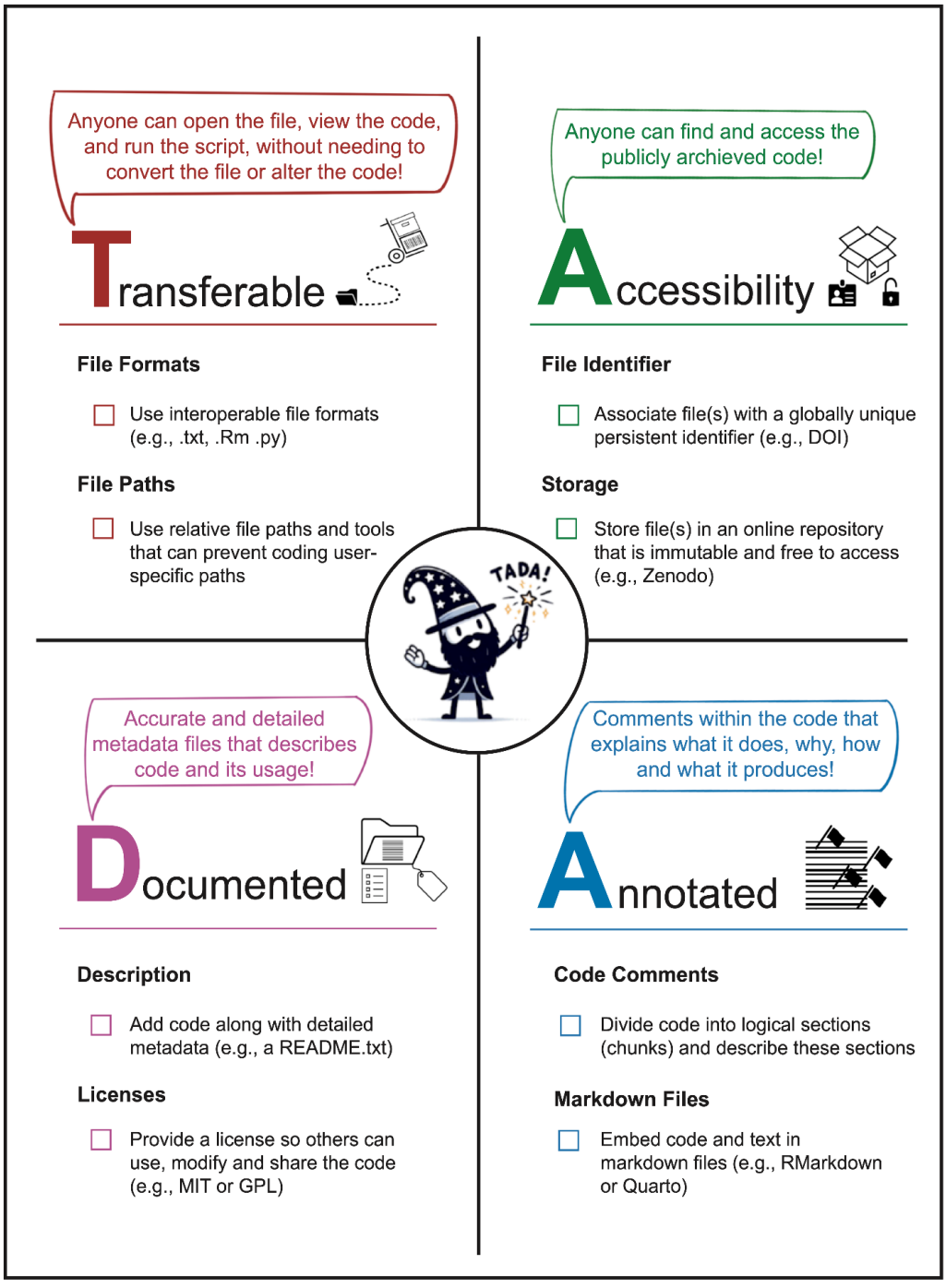

Figure S1. A checklist highlighting the key points of TADA: *Transferable, Accessible, Documented, and Annotated* code. Figure by SD.