Reproducing Spatial Data Science Publications

Edzer Pebesma[0000−0001−8049−7069]

Institute for Geoinformatics, Münster University, Germany,
edzer.pebesma@uni-muenster.de

Abstract. Reproducibility is one of the corner stones of science: when studies cannot be reproduced it is hard to convey that they contain new findings of general truth. We constrain ourselves here to computational aspects of spatial data science, and discuss the challenges posed by always evolving software, scientific software developer communities, upstream and downstream dependencies, the publishing industry, and report on experiences from developer communities, and look at convergence in the spatial data science software ecosystems.

Keywords: data science languages· R · Python · Julia

DOI: https://doi.org/10.25436/E23K5H

1 Introduction

In spatial data science, reproducible research [2][6] is increasingly seen as a property of good science, or a requirement for getting work published. We focus here on studies whose contribution is entirely computational. This means that the study itself did not involve carrying out new experiments or collecting new sample data – the repeatability of those aspects is usually referred to as *replication* – but started off from existing data and contributed by applying novel computational methods to analyse this data.

Publishing reproducible research seems as easy as it never was: share the paper, the data, the software in a way that allows readers to re-execute the computational steps in some form, done. This form can be Jupyter notebook, an R or Julia script, or an R-markdown file or its successor, a quarto file along with rendered pdf or html. Doing this in practical research, e.g. with the goal that 10 years after publication the work is still reproducible (a common requirement from public research funding bodies), is however harder. Roadblocks are found in many places, including (i) readers may not understand the reproducible materials because of the **language** used, (ii) reproducing requires a **run-time**, software that can execute the scripts even 10 years from now, (iii) the software stack used is often not confined to an interpreter ("R", "Python", "Julia") but depends on other packages in those languages and external, **upstream libraries** written in e.g. C++, (iv) computers, compilers, compiled software, interpreters, packages and upstream libraries constantly **evolve**, (v) the **publication industry** does not welcome reproducible publications. In this short contribution we will discuss these aspects in some detail, and sketch a future outlook.

2 SDS languages

The ultimate goal of science, beyond scientific discovery itself, is the *communication* of discoveries. Being able to recreate figures and tables in a paper without understanding what is going on is of little value. The value lies in being able to assess, and hopefully confirm, that what is going on in the paper was the correct procedure to create the results, or in case of doubts or clear errors or omissions to communicate these with the authors, or in case of correct procedures to reuse what was learned (as the basis) for further research for instance using other datasets or comparisons with other methods. The language used (typically: Python, R or Julia) then plays a role, but also the way this language was used and which available extension packages (see Section 4) were used. Using more modern software approaches (be it the language or the packages used) will typically allow authors to express computations cleaner and more compact, but may come at the cost of reaching a smaller readership that is sufficiently familiar with the approach to really grasp what is going on and verify correctness.

3 Run-time preservation

Publishing a study means fixing it: only very rarely do publications get an erratum. While insights continuously develop, scientists are not supposed to adjust publications to newer insights, once published. Software however also changes continuously, and reproduction scripts would typically need to be adjusted to work with updated software. The question is whether or when this should happen. To "fix" the problem of updated software one could preserve the *complete* software used at the time of publication along with the paper [5]. This would mean that e.g. a virtual machine or docker container image (along with the recipe how it was made) would be stored along with the publication, and that that machine can be run (years) later to reproduce the publication. This assumes that such VMs or container images can still be run at a later stage. The technology used for this is relatively new, and at this moment it is hard to say how easy this will be in 5, 10 or 20 years from now.

An alternative approach is to store the reproduction materials (in addition to a fixed version alongside the publication) at a location (e.g. GitHub) where the author(s) can continuously update it. This is in particular useful when it can be expected that readers will want to use the methods published with updated software. We have done this with [1] (both the 2008 and 2013 editions) by running nightly checks on a dedicated machine that would update R and R packages from CRAN before every run. For the successor book [8] we also do this but using continuous integration (GitHub Actions) so that the process that does nightly checks is even more isolated and can be understood and reused by others. Breaking builds of the books are used to communicate with authors of software in case of unintentional software changes, or are used to update online errata of the books in case of intentional changes.

4 Upstream libraries

Open source data science languages have the advantage that the interpreters and libraries/packages are available for anyone without restrictions, and that in case of unexpected results the source code can be scrutinised. Closed source environments such as Matlab or ArcGIS require licences to be available to readers, and licence restriction may hinder the creation and publicly sharing of virtual machines or docker container images with fixed software versions that can be run later.

In open source languages, reproduction scripts typically use packages that make the handling of spatial data easier, such as geopandas, shapely or rasterio in Python, or sf, terra or stars in R. These packages augment the interpreter with useful functions but typically also link to external system requirements, for instance the C++ OSGEO libraries GDAL, PROJ and GEOS. Figure 1 shows the dependency tree (upstream and downstream) for R package sf [7] as an example.

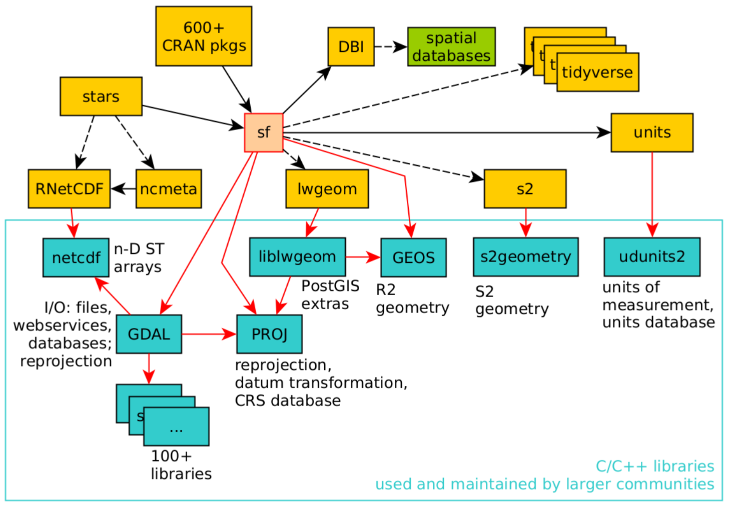


Fig. 1. Dependencies (arrow direction) and reverse dependencies for R package sf

5 Evolving software

Open source software used by spatial data scientists is often also developed by (spatial) data scientists. When software developers introduce breaking changes, they know this will have an effect on users, and reproducibility.

The R system, arguably being the oldest data science language and most closely associated with the statistics community, uses CRAN as its central distribution platform for packages. R packages contain self-tests that check whether the software correctly runs, and that typically involves running packages a package depends on. When a package author submits a new version to CRAN, CRAN runs all reverse dependency checks: for package sf, over 600 packages (figure 1) are checked against the submitted version, and any regression is reported as blocking release. Package authors need to confirm (and show) that dependent package that now break have been informed timely, to justify a changes that has adverse effects on packages depending on it. This mechanism only works within a package archive: it does not run all reproducible publications depending on a package. However, one could (also) publish a reproducible paper as a package on CRAN [3], and have it checked by CRAN when a dependency changes – putting the responsibility with the upstream developer. New submissions to CRAN are reviewed by humans, and checked whether they make any sense and whether authorship and software licenses are clear and complete.

Python does not have the luxury of a package distribution system that automatically checks reverse dependencies: authors need to set up their own testing. Package systems like PiPy (recently plagued by many submissions containing malicious code) only check packages using their own tests (if present), no reverse dependencies are checked. By pinning projects to particular versions of packages reproducibility is assured, but users are not helped by the question how to move on with newer package versions. This approach is considered "developer-friendly", as opposed to the CRAN strategy which is considered "user-friendly".

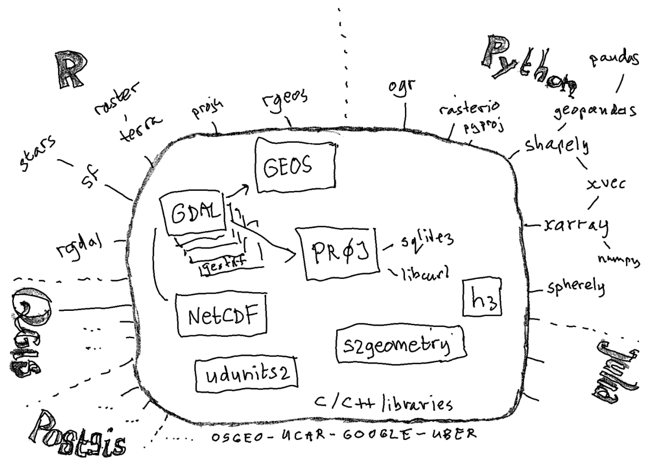


Fig. 2. Incomplete sketch of the open source spatial data science software ecosystem

A further issue is how packages deal with system requirements that are not part of the interpreter. All open source spatial data science languages depend on a set of common libraries (figure 2). When multiple versions of these libraries are present and confused at run time, it will typically lead to a crash of the interpreter. R binary packages for Windows and MacOSX *statically* link all system requirements, leading to huge packages and duplication, but avoiding a dynamic linking hell. Python packaging systems like Conda try to manage this hell with varying success, seemingly trying to do what also Docker images do. Several well-known educators in the spatial Python community develop courses that start with explaining participants how to set up docker and run a docker image. In R spatial courses, this is rarely seen. R spatial users using Linux now also have access to binary packages for Ubuntu, Debian and Fedora but only on systems carrying the upstream spatial libraries from the system; users installing upstream libraries e.g. from PPA's or from source need to install (compile) R packages from source.

6 The publication industry

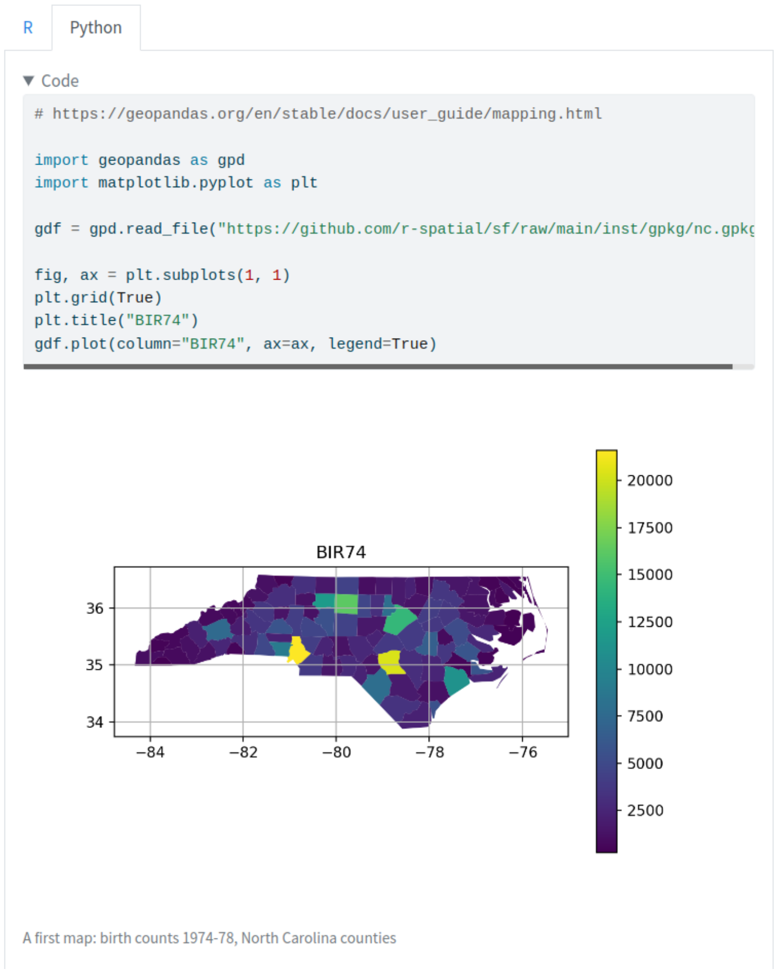
I am not aware of a single journal published by a commercial publisher that accepts a Jupyter notebook, R markdown or quarto document as submission format. Although the technology is solid, proven, and widely used, publishers want to publish pdf and html and minimise the trouble getting there. In two funded DFG projects ("Opening Reproducible Research", `https://o2r.info/`) we have experienced that publishers are happy to be involved in talks about how to push the boundaries of publishing towards publishing reproducibly, but refuse participating in any experiments beyond business-as-usual. The list of publishers we experienced this with includes big ones like Elsevier, but also smaller and more agile ones like Copernicus.

The current practice of journals is that authors are encouraged to publish reproducibly, but journals will not tell them how this should work. Also, papers are rarely rejected because data and/or scripts are not shared: this is typically up to the reviewer(s). Editors invite reviewers to review papers that claim to be reproducible, but typically do not check whether the reproducibility materials are available to the reviewer.

7 The future

The Council of the EU calls for transparent, equitable, and open access to scholarly publications – that is the about "A" (accessible) in FAIR. The "R" (reusable) of publications involves sharing data and software. If we, scientists, continue working with commercial publishers (submitting papers, reviewing papers, participating in editorial boards) I predict that in 10 years from now at least 95% of the published papers will be as non-reusable as they are now.

On the positive side I see an increase in communication between the different communities identifiable in figure 2. For spatial data science, we all reuse much of the same software stack. Communication takes place on social networks but also e.g. on OpenGeoHub summer schools where last year people from R-spatial, GeoPython and GeoJulia met, as well as this year on the GeoPython 2023 conference. Cross language topics included spherical geometries (how do we leave the flat, 2D Cartesian GIS world?), raster and vector data cubes, links to the climate, weather and Earth observation communities, teaching, and software packaging.



```
# https://geopandas.org/en/stable/docs/user_guide/mapping.html

import geopandas as gpd
import matplotlib.pyplot as plt

gdf = gpd.read_file("https://github.com/r-spatial/sf/raw/main/inst/gpkg/nc.gpkg

fig, ax = plt.subplots(1, 1)
plt.grid(True)
plt.title("BIR74")
gdf.plot(column="BIR74", ax=ax, legend=True)
```

A first map: birth counts 1974-78, North Carolina counties

Fig. 3. Screen shot of a bilingual (R+Python) Python tab, from [8]

We are in the process of trying to make [8] bilingual, with all code sections containing an R and Python tab (figure 3). The quarto publishing system has (currently) support for tabs using R, Python, Julia and Observable JS. Another initiative, ”geocompx”, develops the ”Geocomputation with Python” book alongside [4]; this is a less one-to-one translation, but covering the same general idea. Julia will catch up.

References

1. Bivand, R., Pebesma, E., Gómez-Rubio, V.: Applied spatial data analysis with R, Second edition. Springer, NY (2013), http://www.asdar-book.org/
2. Goodchild, M.F., Fotheringham, A.S., Kedron, P., Li, W.: Introduction: Forum on reproducibility and replicability in geography. Annals of the American Association of Geographers **111**(5), 1271–1274 (2021). https://doi.org/10.1080/24694452.2020.1806030, https://doi.org/10.1080/24694452.2020.1806030
3. Leisch, F., Eugster, M., Hothorn, T.: Executable papers for the R community: The R2 platform for reproducible research. Procedia Computer Science **4**, 618–626 (2011). https://doi.org/https://doi.org/10.1016/j.procs.2011.04.065, https://www.sciencedirect.com/science/article/pii/S1877050911001232, proceedings of the International Conference on Computational Science, ICCS 2011
4. Lovelace, R., Nowosad, J., Muenchow, J.: Geocomputation with R. Chapman & Hall/CRC (2019), https://r.geocompx.org/
5. Nüst, D., Pebesma, E.: Practical reproducibility in geography and geosciences. Annals of the American Association of Geographers **111**(5), 1300–1310 (2021). https://doi.org/10.1080/24694452.2020.1806028, https://doi.org/10.1080/24694452.2020.1806028
6. Pebesma, E., Nüst, D., Bivand, R.: The R software environment in reproducible geoscientific research. Eos, Transactions American Geophysical Union **16**, 163–164 (2012). https://doi.org/10.1029/2012EO160003, https://doi.org/10.1029/2012EO160003
7. Pebesma, E.: Simple Features for R: Standardized Support for Spatial Vector Data. The R Journal **10**(1), 439–446 (2018). https://doi.org/10.32614/RJ-2018-009, https://doi.org/10.32614/RJ-2018-009
8. Pebesma, E., Bivand, R.: Spatial Data Science: With Applications in R. Chapman and Hall/CRC (2023). https://doi.org/10.1201/9780429459016, https://doi.org/10.1201/9780429459016