

Computational reproducibility of Jupyter notebooks from biomedical publications

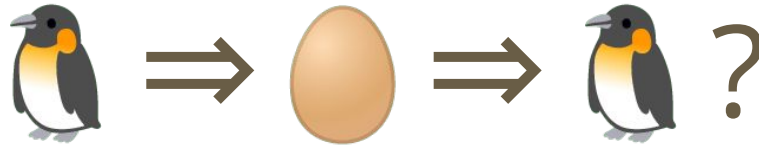


Table of contents

- Background
- Study design
- Implementation
- Results
- Self-replication
- Implications
- Jupyter and Wikimedia
- Environmental footprint
- Recommendations
- How you can get involved

Background

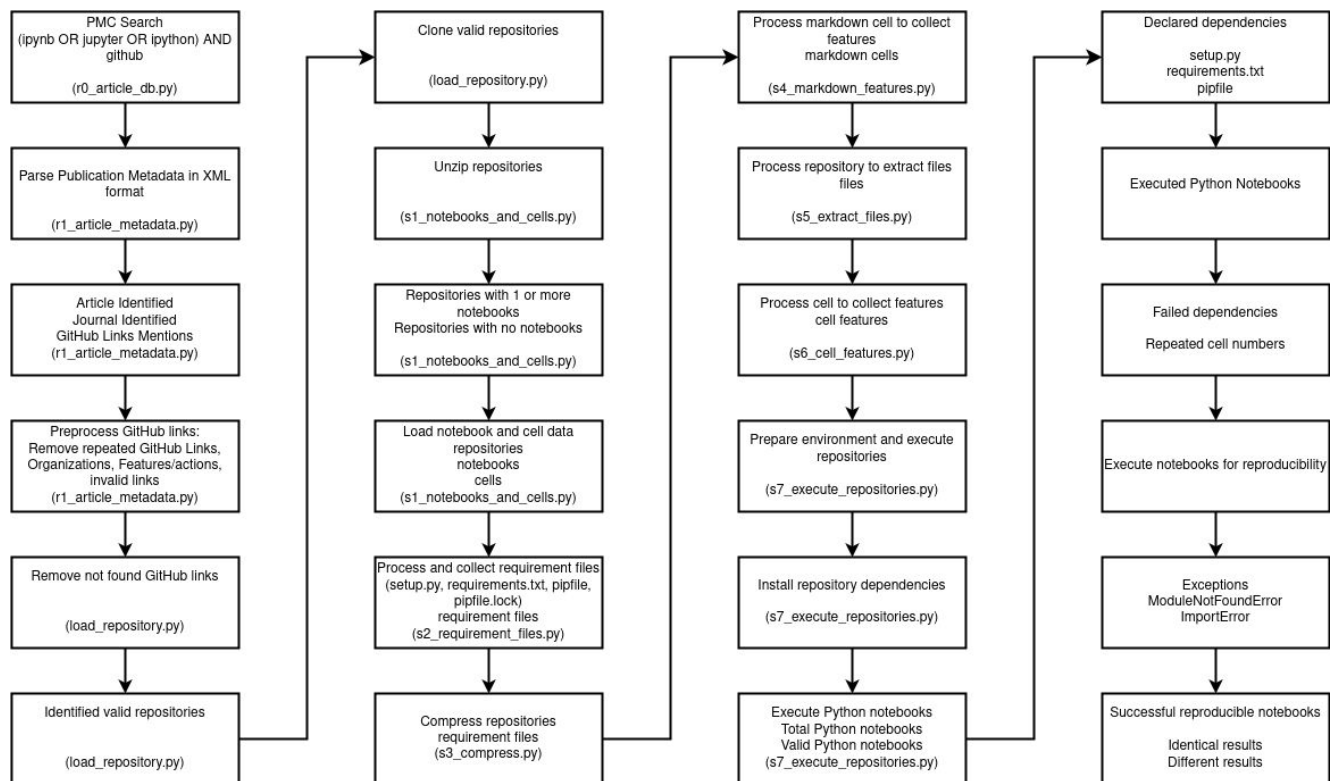
- General awareness of rep(lica | roduci)bility issues, scientific vs. computational
- [Open data doathon 2017 \(107 notebooks then, 5505 now\)](#)
- JupyterCon talks
 - 2017: [Postpublication Peer Review of Jupyter Notebooks Referenced in Articles on PubMed Central](#)
 - Manual efforts at computational reproducibility do not scale
 - 2020: [Jupyter in the Wikimedia ecosystem](#)
 - Lots of interactions, but potential for more
 - 2020: [Analyzing the use and reproducibility of Jupyter Notebooks using ReproduceMeGit](#)
 - Computational reproducibility can be assessed in an automated fashion
- [Dataset of A Large-scale Study about Quality and Reproducibility of Jupyter Notebooks](#)
 - focus on notebooks from GitHub, not from research publications
- [Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks](#)
 - useful, but no accompanying data

Study design

- Search PubMed Central to get full text XML
- Parse XML to get repo/ notebook URLs & metadata
- Normalize the URLs
- Crawl GitHub repos for notebooks and requirements
- Install declared requirements
- Run notebooks, log errors and outputs
- Compare outputs to original runs
- Analyze errors
- **Preprint:** <https://doi.org/10.48550/arXiv.2209.04308>
- **Data:** <https://doi.org/10.5281/zenodo.6802158>
- **Code:** <https://github.com/fusion-jena/computational-reproducibility-pmc>

DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)

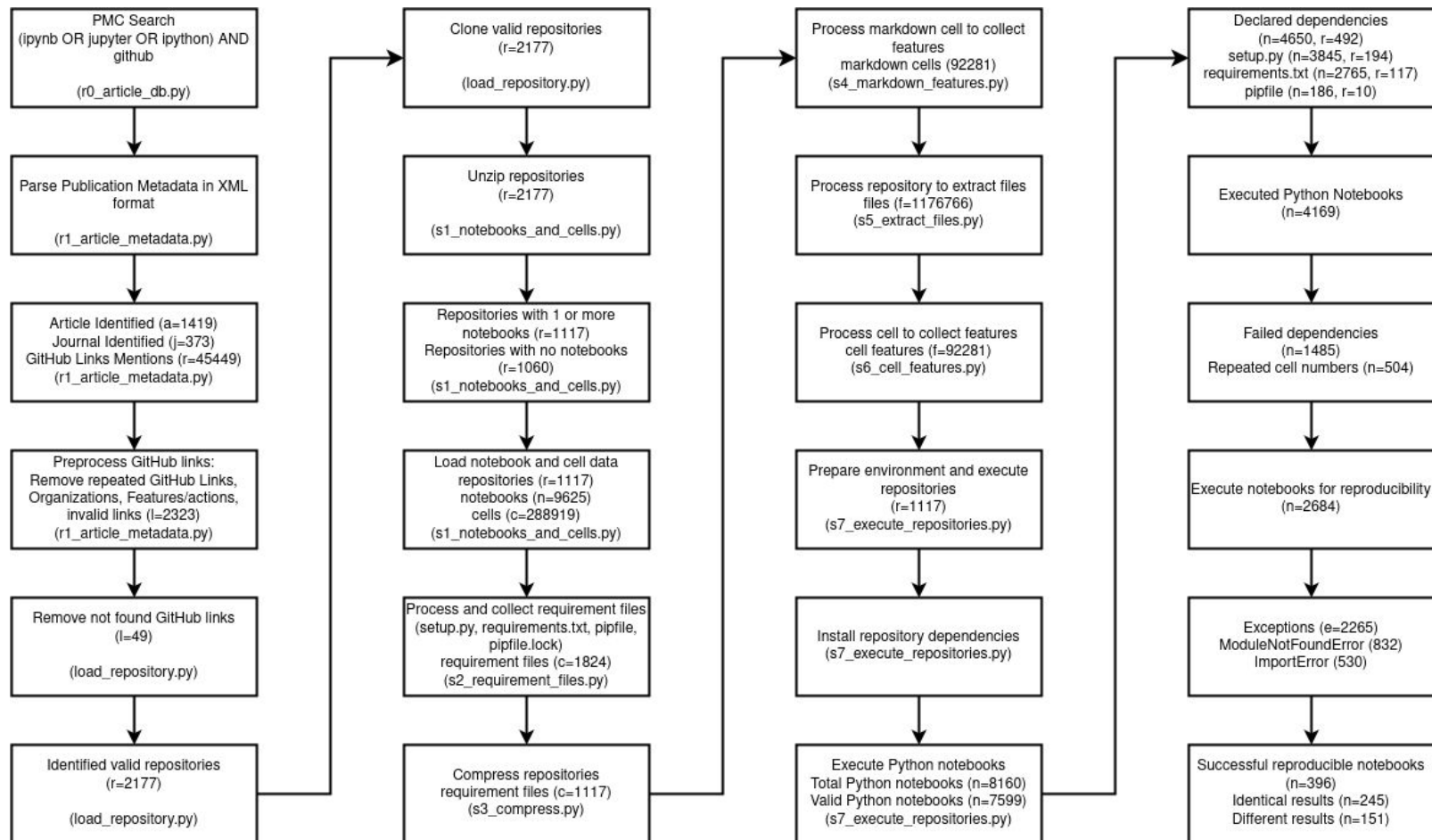
Study design



Implementation

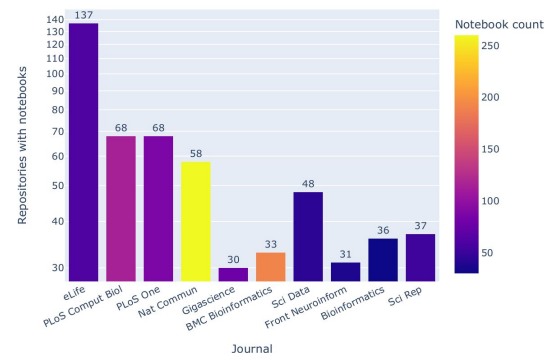
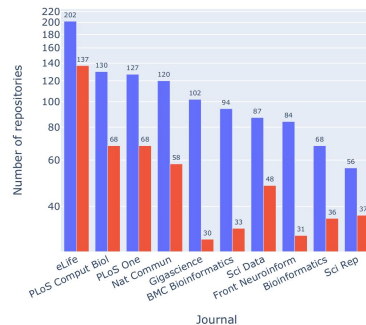
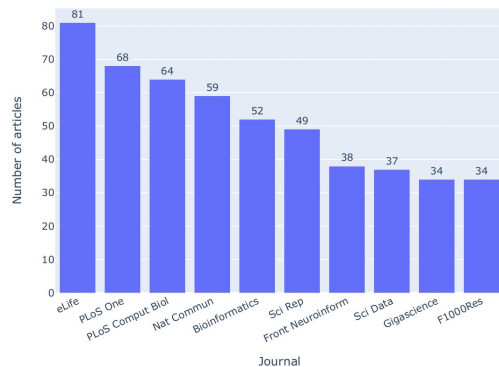
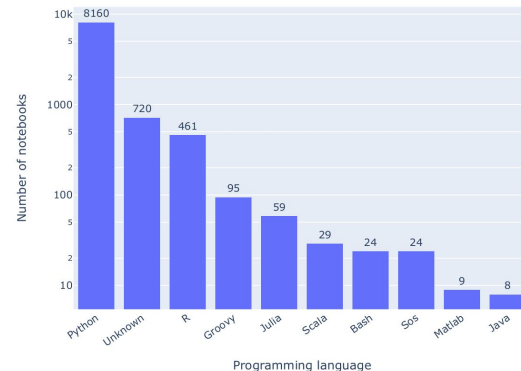
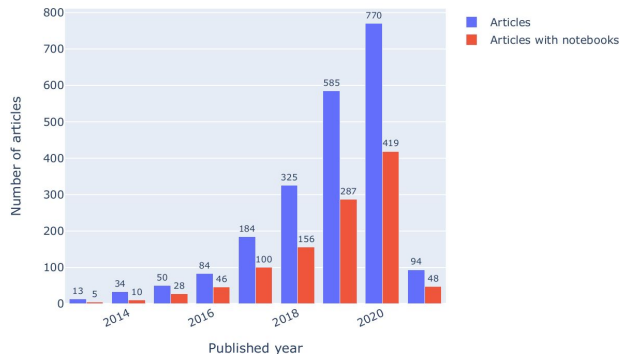
- Tech stack
 - Cloud infrastructure, i.e. a shared resource
 - Python scripts under conda environments
- Assumptions
 - Papers using Jupyter mention at least one of “Jupyter”/ “iPython” / “ipynb” as well as “GitHub”
 - this will miss non-GitHub repos, which are also on the rise
 - this might find repos by other authors
 - Notebooks are just mentioned, not cited
 - Mentions of GitHub repos can be normalized
 - Within a given GitHub repository, the default branch is the best one to mine
 - Python is representative for Jupyter notebooks in biomedical publications

Results



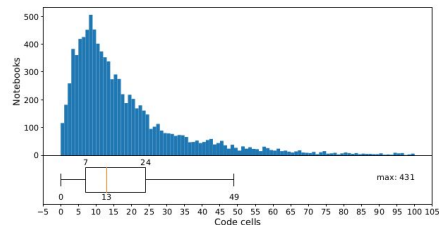
Results - general statistics about our corpus

- Articles
- Repositories
- Notebooks
- Languages
- Journals

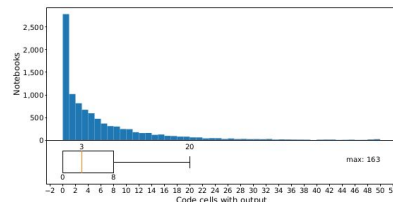


DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)

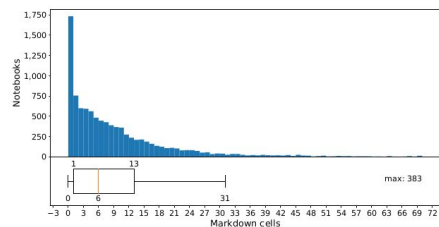
Results - general statistics about notebooks in our corpus



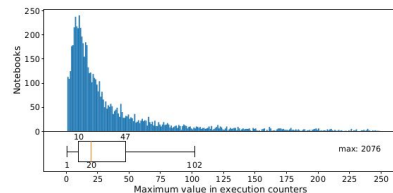
(a) Distribution of the number of code cells across notebooks in our corpus.



(b) Distribution of the number of code cells with outputs across notebooks in our corpus.



(c) Distribution of the number of Markdown cells across notebooks in our corpus.



(d) Distribution of the maximum execution count across notebooks in our corpus.

Figure 11. Analysis of the notebook structure

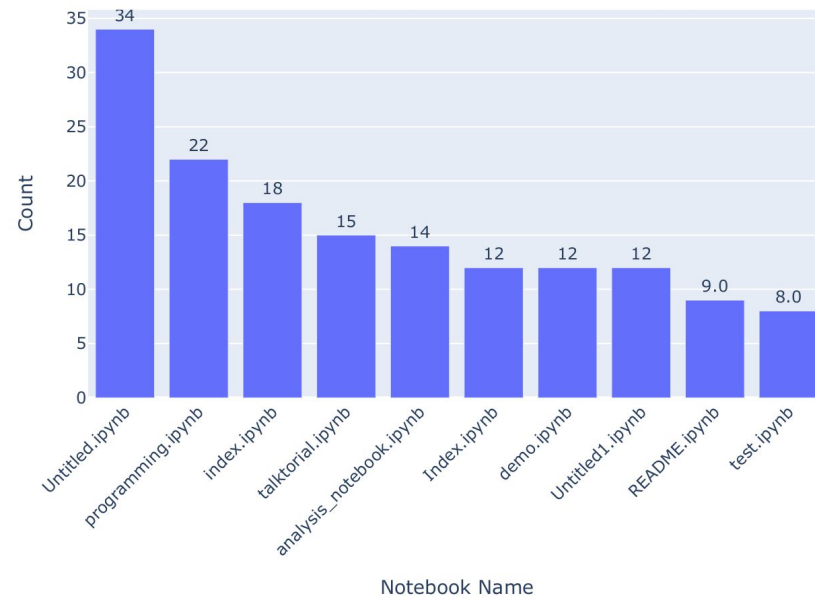


Figure 12. Most frequent notebook titles.

Results - Trends

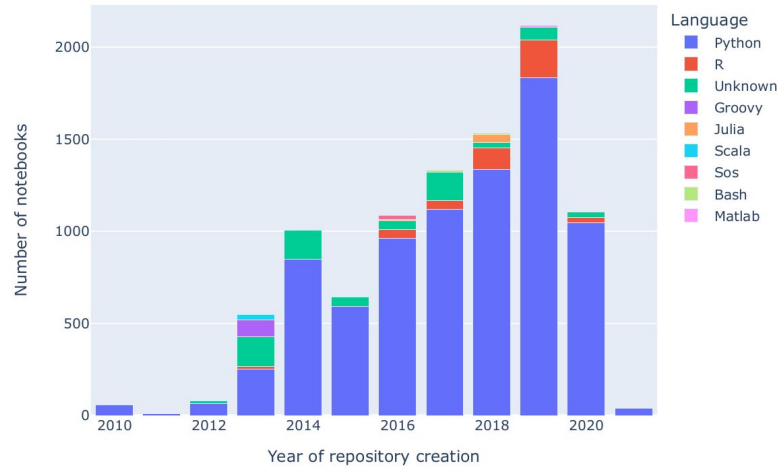


Figure 8. Relative proportion of the most frequent programming languages used in the notebooks per year.

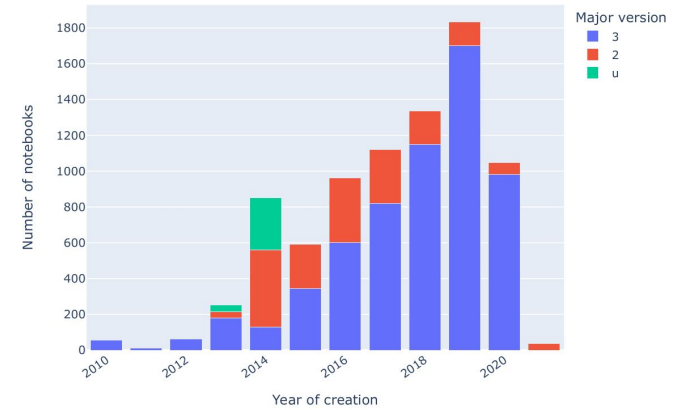


Figure 10. Python notebooks by major Python version by year of first commit.

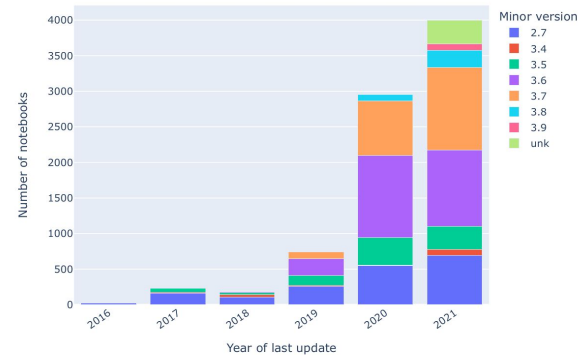
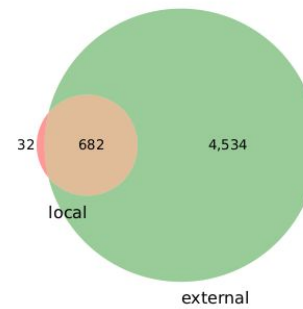
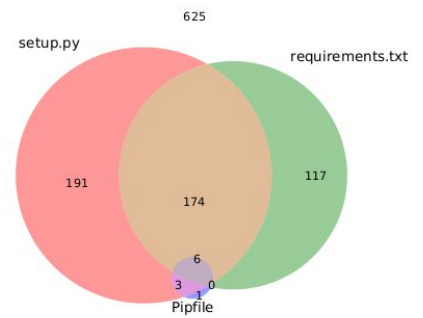


Figure 9. Python notebooks by minor Python version by year of last commit to the GitHub repository.

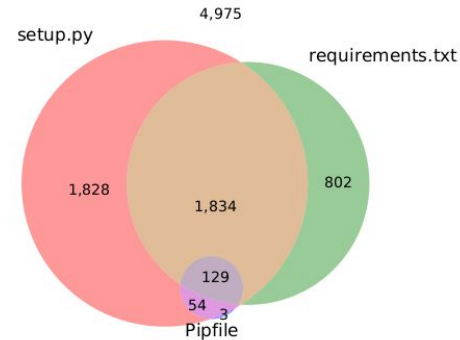
Dependencies of notebooks and repositories



(a) External versus local modules declared in Jupyter notebooks.



(b) Repositories with dependencies.



(c) Notebooks with dependencies.

Figure 17. Dependencies of Jupyter Notebooks and GitHub repositories. In (a), the notebooks depending on external modules (green) are plotted against notebooks depending on local modules (red) and notebooks that had both (brown). In (b) and (c), GitHub repositories and Jupyter notebooks are shown as to whether they declared their dependencies via any combination of `setup.py` (red), `requirements.txt` (green) or a `pipfile` (pink).

Notebook dependencies: Python modules

Anaconda defaults:

- scikit-learn
- numpy
- matplotlib
- pandas
- ...

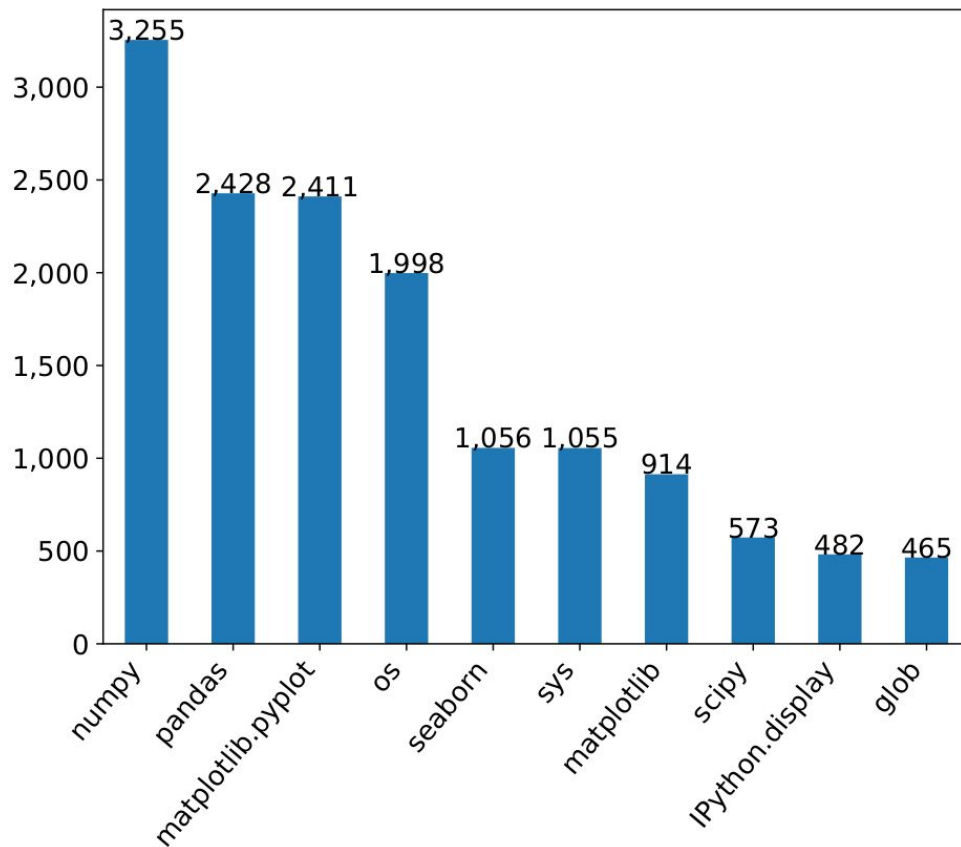


Figure 15. Top Python modules declared in Jupyter Notebooks.

DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)

Results - Common issues

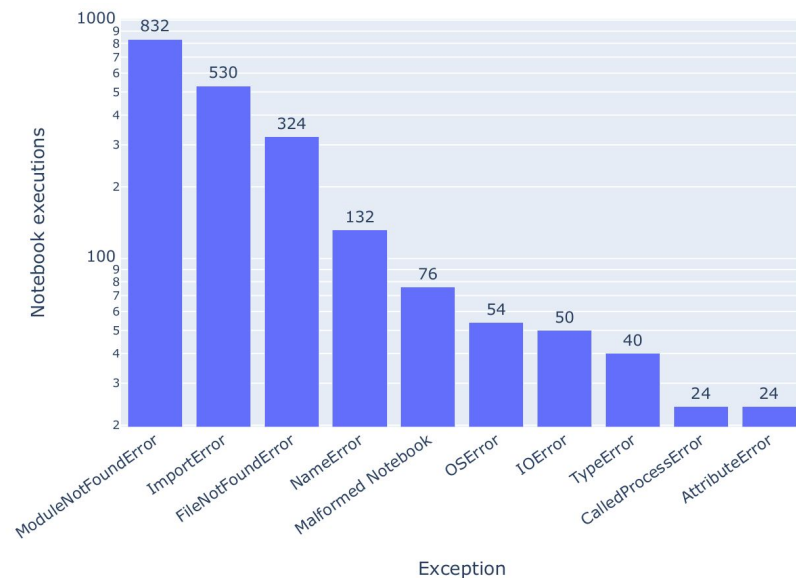


Figure 18. Exceptions occurring in Jupyter Notebooks.

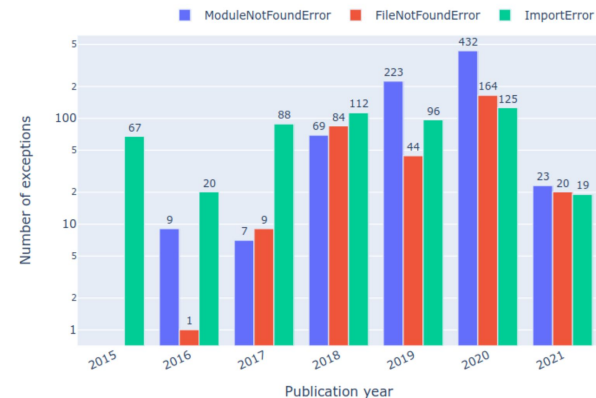


Figure 19. ModuleNotFoundError, ImportError and FileNotFound exceptions by year of publication.

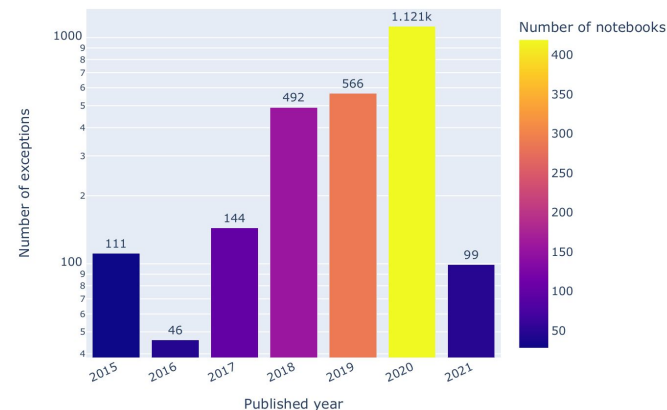


Figure 20. Exceptions by year of publication.

Completed executions by Python version

Table 2. Comparison of most frequent Python versions declared for notebooks that were successfully executed without errors, grouped by whether their results were different from or identical to the results documented for the original notebook. Versions listed in *italics* occur in both top-5 groups, versions listed in **bold** in only one. The absolute columns give total number of notebooks per version and group, while the relative columns normalize the absolute values as a percentage of the total number of notebooks per group, i.e. 151 for different and 245 for identical, as per Table 1. In both groups, the top-5 versions account for slightly over half of the notebooks.

	different			identical		
rank	version	absolute	relative	version	absolute	relative
1	3.6.9	28	18.5	3.7.3	46	18.8
2	2.7.6	17	11.3	3.6.9	30	12.2
3	2.7.10	13	8.6	3.7.1	26	10.6
4	3.6.5	12	7.9	3.7.4	22	9.0
5	2.7.9	11	7.3	3.6.5	15	6.1

Results - characteristics of reproducible notebooks

Table 1. Comparison of notebooks that were successfully executed without errors, grouped by whether their results were different from or identical to the results documented for the original notebook. For features listed in *italics*, the mean values per notebook are indicated, otherwise totals across all notebooks per group.

	Notebooks with different results	Notebooks with identical results
Number of notebooks	151	245
setup.py	0	98
requirement.txt	0	107
pipfile	0	0
<i>Total cells</i>	17.9	17.1
<i>Code cells</i>	12.3	9.8
<i>Markdown cells</i>	5.6	6.7
<i>Ratio of Markdown vs. code cells</i>	0.46	0.68
<i>Empty cells</i>	0.7	0.7
<i>Differences</i>	5.3	0
<i>Execution time (s)</i>	22.1	16.4
<i>Execution time per code cell (s)</i>	1.80	1.67

Common style issues

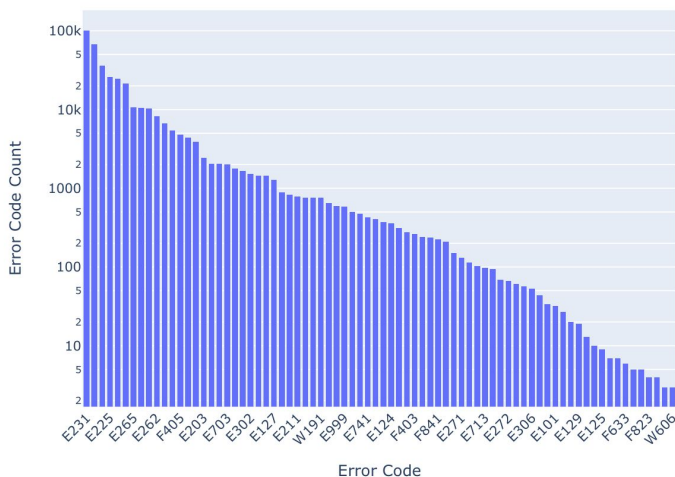


Figure 23. Frequent notebook code style errors as per the Python code style guide.

- E: styling
- F: definitions
- W: deprecations

Table 3. Common Python Notebook Code Warning/Style Error found in our Study

Error code	Description	Count
E231	missing whitespace after commas, semicolons or colons	102218
E225	missing whitespace around operator	25979
E265	block comment should start with '#'	10769
E402	module level import not at top of file	10478
E262	inline comment should start with '#'	8369
E703	statement ends with a semicolon	2023
E127	continuation line over-indented for visual indent	1290
E701	multiple statements on one line	500
E741	do not use variables named 'l', 'O', or 'I'	432
E401	multiple imports on one line	95
E101	indentation contains mixed spaces and tabs	32
F405	<i>name</i> may be undefined, or defined from star imports: <i>module</i>	4840
F401	<i>module</i> imported but unused	3938
F821	undefined name 'X'	2071
F403	'from module import *' used; unable to detect undefined names	263
F841	local variable 'X' is assigned to but never used	225

Self-replication

- Practical considerations
 - Dynamic data in an exponential growth phase
 - How much memory/ storage to allocate in advance?
 - used default allocation originally, now 128 GB RAM
 - How much time to allow for a re-run?
 - current one has been running for 1.5 months
 - Parallelization
 - none so far
 - also everything on CPU, no accommodation of GPUs/ TPUs
- Reviewer comment
 - code does not run

Self-replication

Error	Fix (Commit)
External Dependency/ Deprecation error / Compatibility Error/ Dependency Conflicts	removed the deprecation parameter 'convert_unicode' (C1) recent compatibility issue in setuptools (C9) Added additional flag to ignore installed versions (C12) Added additional flag for pip's dependency resolver (C13)
Empty/Collection/Topic repository	Added check for empty GitHub repository (C3) Added check for a collection/topic GitHub url (C4)
Typo/Missing Checks	Missing parenthesis (C7) Change to correct directory for installation (C13)
Additional support	Added support for new Python versions (C2 , C10 , C11) Added support for Bioyphthon library in requirements file (C8) Moved all the installation instructions from the README file to conda-setup.sh file (C15)

Implications

- Computational reproducibility is a key element of scientific reproducibility
- Automatically assessed computational reproducibility is low
- Opportunities for
 - Scientific reproducibility
 - Learning/ teaching through errors and fixing them
 - Standardization
 - List dependencies at the top
 - Linking standards to sample problems they solve
 - Tooling, e.g. [ReproduceMeGit](#)
- Pick some low-hanging fruits, e.g.
 - notebooks had no declared dependencies
 - 504 notebooks were sorted out for duplicate cell numbers ⇒ check
 - identify point in publication process where replication should occur, and who is to do it
 - support Binder, and help scale it (see also [poster by Fangohr et al.](#))

Jupyter and Wikimedia

- Wikipedia
 - Jupyter notebooks as *reliable* references?
- Wikidata
 - [Co-usage graph](#)
 - [Map of published Jupyter usage](#)
- JupyterHub instances
 - used to edit Wikimedia
 - opportunities for standardization

Co-usage

Resources used together.

Show 10 entries

Reload

Search:

Count	Coused	Zoom	Coused description	Example work
248	NumPy		numerical programming package for the Python programming language	The connectome viewer toolkit: an open source framework to manage, analyze, and visualize connectomes
111	scikit-learn		machine learning library for the Python programming language	Open Drug Discovery Toolkit (ODDT): a new open-source player in the drug discovery field
66	ImageJ		image processing software	COMPASS: Continuous Open Mouse Phenotyping of Activity and Sleep Status
58	scikit-image		open source image processing library for the Python programming language	The connectome viewer toolkit: an open source framework to manage, analyze, and visualize connectomes
50	Cytoscape		open source software platform for visualizing molecular interaction networks and biological pathways	Open source libraries and frameworks for biological data visualisation: a guide for developers
50	ggplot2		data visualization package for the statistical programming language R	phyloseq: an R package for reproducible interactive analysis and graphics of microbiome census data
42	Python		general-purpose programming language	Bioalerts: a python library for the derivation of structural alerts from bioactivity and toxicity data sets
30	DESeq2		R package	Epiviz: a view inside the design of an integrated visual analysis software for genomics
30	SQL		relational database language that allows to extract from data tables a series of records with selection, sorting and computation criteria, or to update, delete or add new records	Ten Simple Rules for Digital Data Storage
24	Neuron		simulation environment for modeling neurons	Nengo: a Python tool for building large-scale functional brain models

Wikidata Query Service

use: co-used.spargl

Showing 1 to 10 of 95 entries

DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)

Environmental footprint

- Simplified calculation: $\text{carbon footprint} = \text{energy needed} * \text{carbon intensity}$

(<http://calculator.green-algorithms.org/>)

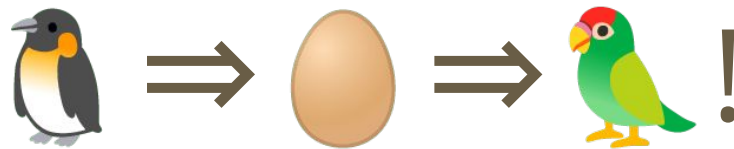
- At a runtime of 117 hours and 52 minutes and considering the local energy mix of the cluster, this gives a footprint of 16.05 kg CO₂e
 - 1/3 of a flight Paris-London
 - 90 km in a passenger car
- Caveats
 - We did not account for test runs, failed runs, our code development, hardware footprint, external code development, trips to JupyterCons.
 - The calculation is thus an underestimation/ lower boundary.

Recommendations

- Separate talk
- Existing recommendations are good but need to be
 - integrated into actual workflows
 - e.g. see [Jupyter](#)
 - adapted to the role(s) of different stakeholders
 - expanded to take into account different types of dependencies
- Make notebooks [citable](#)
 - run them before citing
- Routinely re-run notebooks
 - both your own and those of others
- Monitor the environmental footprint

How you can get involved

- How shall we share the dataset?
- How would you use it?
 - Which variables (article/ repo/ notebook etc.) to look at?
 - Research questions
 - Teaching/ learning opportunities
- What would be useful follow-ups?
- Anyone interested in helping turn this into a service? NumFOCUS?
- Who wants to do this for other languages?
- Would it be useful to publish reports about computational reproducibility?
- Would you like to publish such reports?



Merci! Danke! मन्नी! धन्यवाद!

- [2017 doathon participants](#), Jupyter community, JupyterCons
- [Pimentel et al., 2019](#)
- Carl Zeiss Foundation for the project “A Virtual Werkstatt for Digitization in the Sciences (K3)”
- Alfred P. Sloan Foundation under grant number G-2021-17106.
- Ara Cluster of Friedrich Schiller University Jena under DFG grants INST 275/334-1 FUGG and INST 275/363-1 FUGG.

contact: daniel.mietchen@ronininstitute.org & sheeba.samuel@uni-jena.de

DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)

Bonus slides

The following slides are here to facilitate anticipated discussion.

ORCID usage

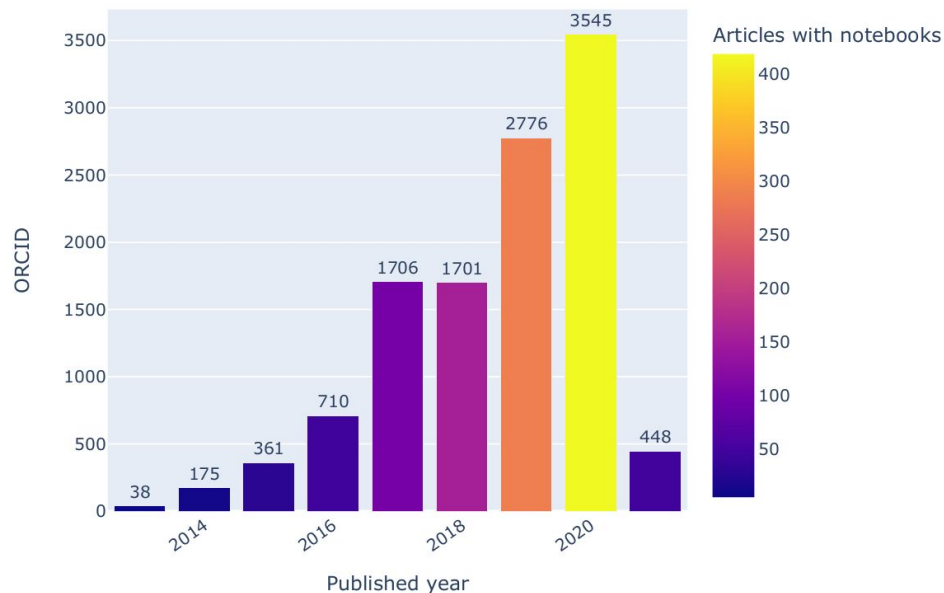


Figure 6. ORCID usage in our collection. Bars indicate the total number of ORCID IDs found each year for authors of articles in our collection. Colors indicate the number of articles that year with Jupyter notebooks. Note that data for 2021 is incomplete, as only articles published by mid-February have been included.

Notebook titles containing the string “test”

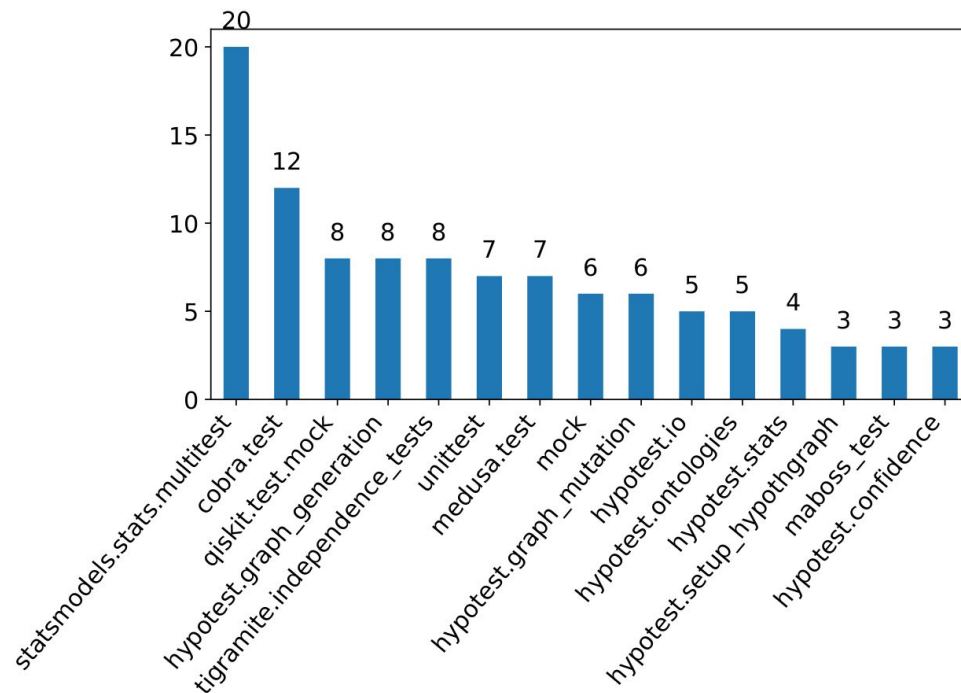


Figure 13. Notebooks with the string test

Notebook title length

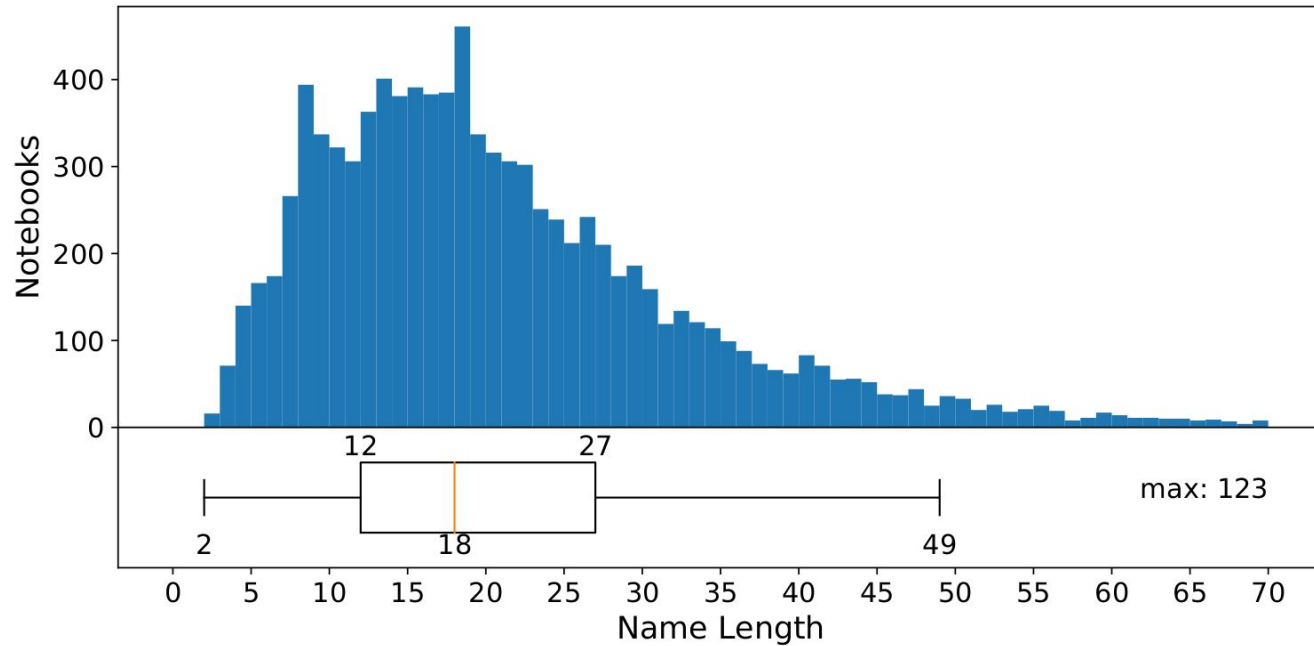


Figure 14. Notebook title length

Notebooks with load extension modules

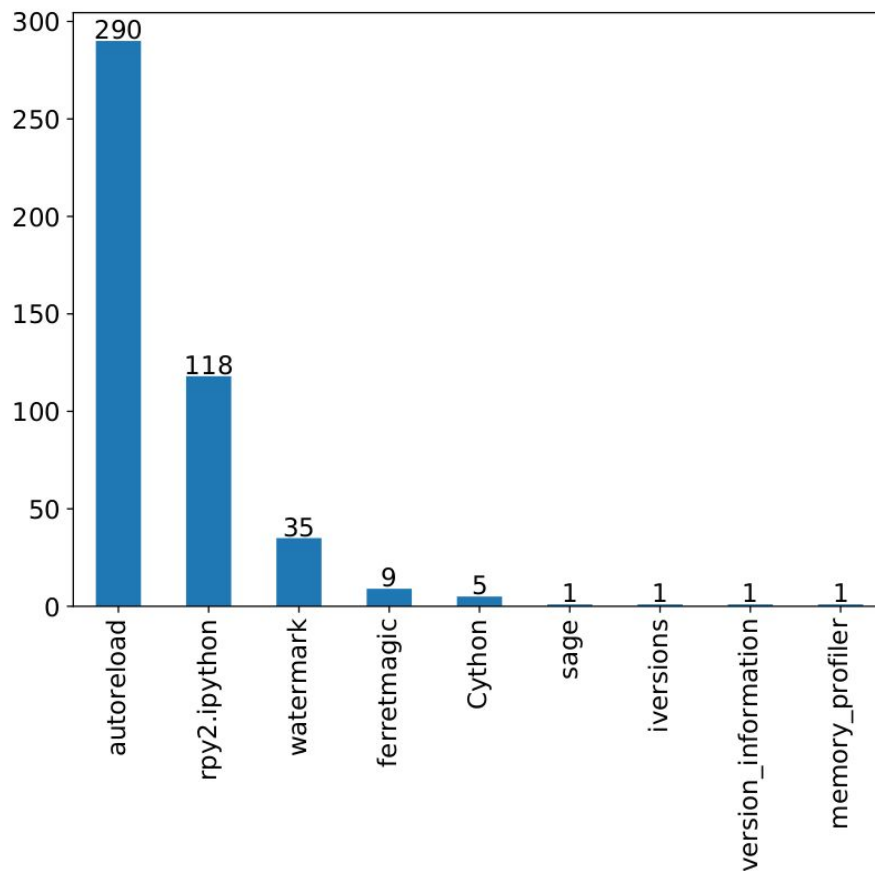


Figure 16. Load extension modules in Jupyter Notebooks

DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)

Runtime exceptions by article type

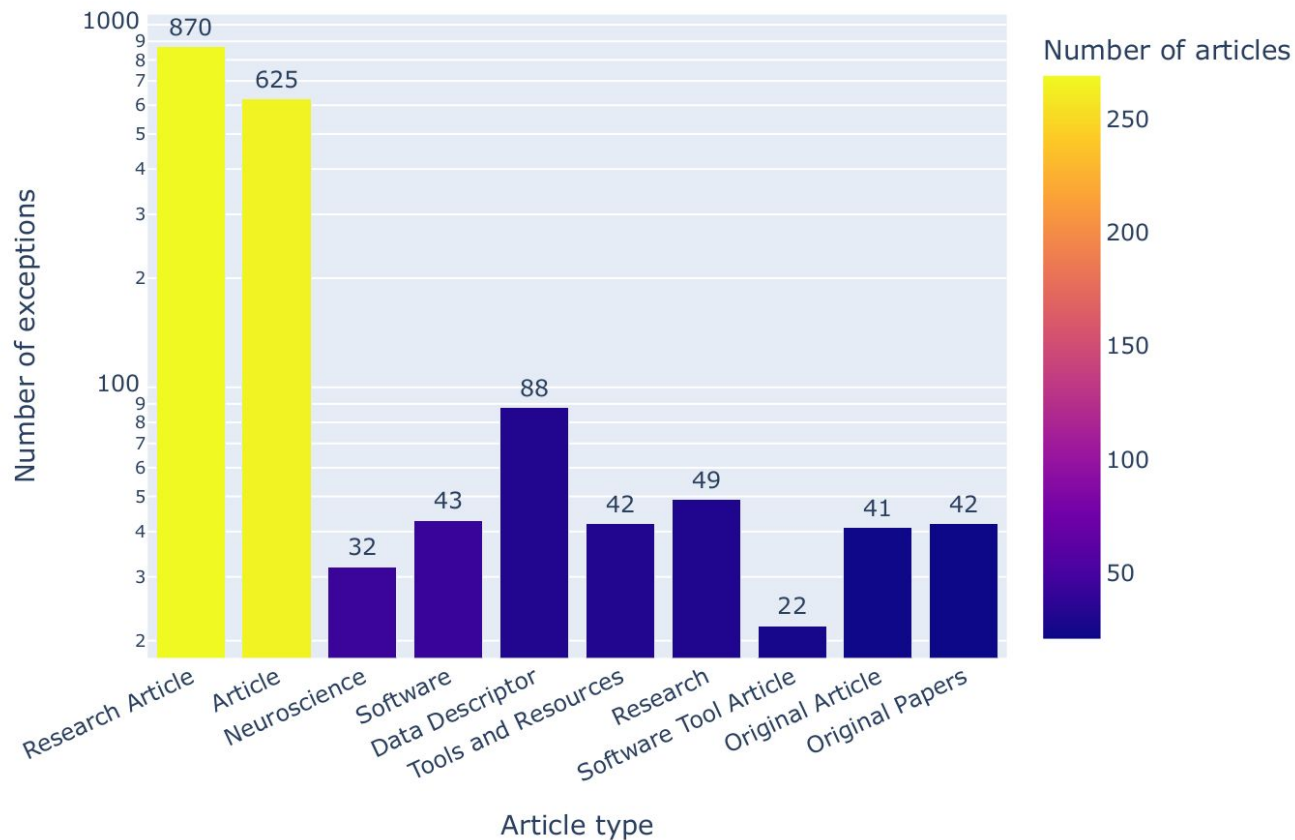


Figure 21. Exceptions by article type.

DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)

Runtime exceptions by journal

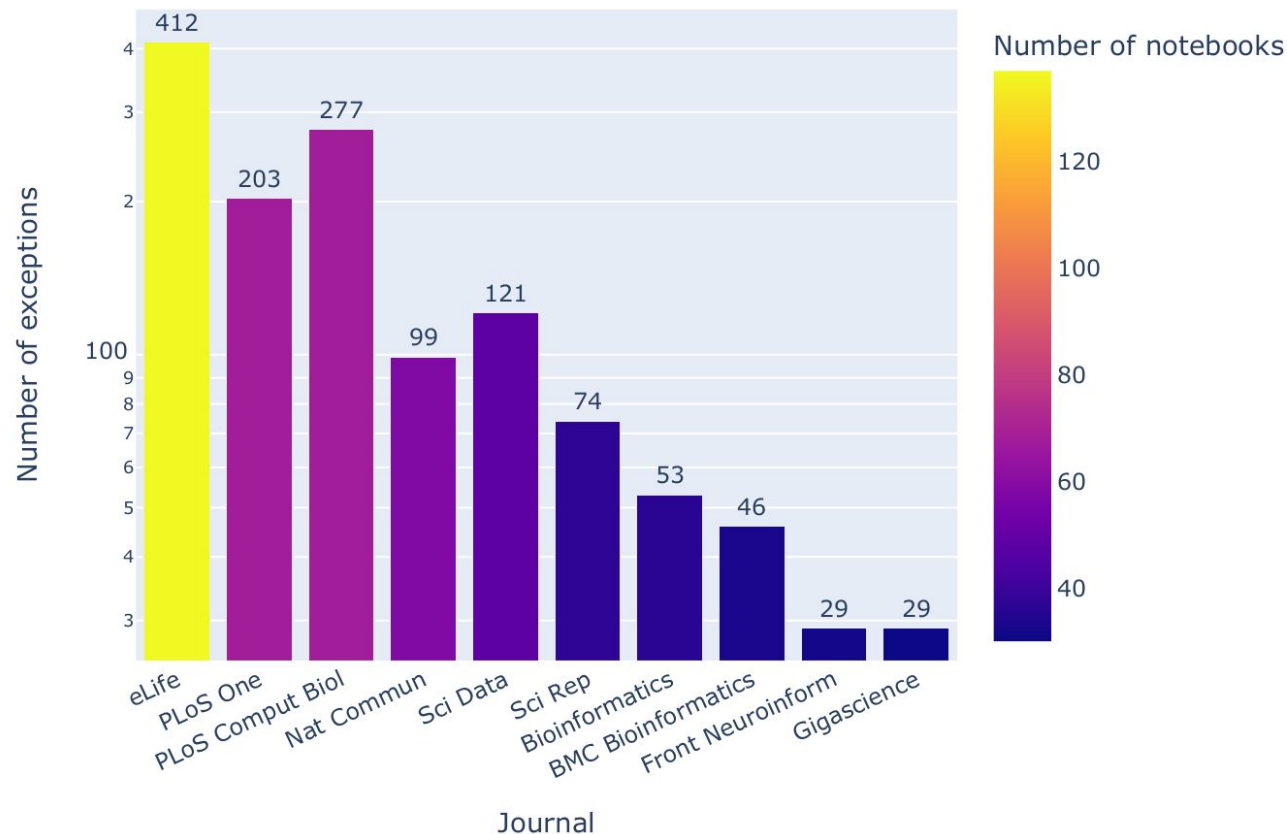


Figure 22. Exceptions by journal.

DOI: [10.5281/zenodo.7854503](https://doi.org/10.5281/zenodo.7854503)