Doing Data Science on the Shoulders of Giants: The Value of Open Source Software for the Data Science Community

Katie Malone¹ Rich Wolski²

¹Tempus Labs, Chicago, Illinois, United States of America,

²Department of Computer Science, College of Engineering, University of California Santa Barbara, Santa Barbara, California, United States of America

Published on: May 31, 2020

DOI: <https://doi.org/10.1162/99608f92.268cc8e4>

License: [Creative Commons Attribution 4.0 International License \(CC-BY 4.0\)](https://creativecommons.org/licenses/by/4.0/)

ABSTRACT

Open source software is ubiquitous throughout data science, and enables the work of nearly every data scientist in some way or another. Open source projects, however, are disproportionately maintained by a small number of individuals, some of whom are institutionally supported, but many of whom do this maintenance on a purely volunteer basis. The health of the data science ecosystem depends on the support of open source projects, on an individual and institutional level.

Keywords: open source software, data science community, software licenses, computing

1. Free Open-Source Software: The Foundational Layer of Data Science Computing

The world runs on open source software, from the Linux operating system to the Apache web server to Git version control. If you're reading this on an Android phone, your operating system is open source. The research and drafting of this column happened in a Firefox web browser, which is also open source. The data visualization was created by Matplotlib (open source visualization software) in a Jupyter notebook (also open source). By definition, the source code for these programs is freely available to use or modify (albeit sometimes with caveats—see Appendix 1), and the dynamics that arise from that simple fact have created an ecosystem surrounding software that is unlike any other type of marketplace (Eghbal, 2016). Open source software underpins many different types of computing, including the software stack that powers most modern data science (Piatetsky, 2019). And yet, it's easy to be a beneficiary of this ecosystem without realizing it's there. The more the users of this software (which is all of us) appreciate and contribute to the ecosystem though, even in small ways, the more we all benefit.

In the data science world, open source projects permeate all levels. R and Python, the two most popular general purpose data science programming languages, are themselves open source. More importantly, the rich ecosystem of libraries and packages built on top of these languages is heavily dominated by open source software. Open source packages do everything from implementing machine learning models to creating visualizations to storing and accessing raw data. For the vast majority of data scientists, a day's work in programming looks like loading up these libraries, and knitting together their features and functions into a workflow that solves a data science problem. It's simple but very powerful: for common data science needs, like cross-validation or missing value imputation, data scientists don't have to create their own implementation. They can build upon the implementations created by their colleagues, and as we'll see, become contributors to those implementations themselves, creating a virtuous cycle.

There are a few foundational tenets that define open source software. First, the underlying code of an open source project is publicly visible. Anyone is able to read, modify, and, at least in theory, contribute new

features to it (the open source community often calls this “free as in speech”—although as we detail in Appendix 1, the license terms of any particular project can have a large impact on how the software can actually be used). Second, open source software is monetarily free: you can go to a code repository like GitHub or CRAN right now and, in a few keystrokes, have your own copy of the software to use without paying anybody anything (in the open source community, this is often referred to as “free as in beer”). Anyone, whether they have copious financial resources or not, can download and use the software. (Later in this article we explore how companies can nonetheless create viable business models around free open source software). And the third tenet, which arises as a result of the first two, is that an ethos of volunteer stewardship and community pervades open source software. With no money changing hands when the software gets downloaded and licensed (free as in beer) and anyone who wants to contribute to the software having the opportunity to do so (free as in speech), the software democratizes who gets to write the source code of data science (Raymond, 1999).

The implications of this democratization are profound. The fact that open source software is free and freely available promotes individual exploration, particularly in a research context. There is no ‘paperwork’ needed to experiment with an open source package. One may need to overcome the lack of professional support tools, but open source software promotes scientific exploration by individuals and small teams where information and ideas can be readily exchanged and enhanced.

Second, while it’s easy just to focus on the software itself, the community surrounding the software is arguably equally impactful. There is a strong social networking component within open source communities: in addition to software contributions, open source community members participate in issue trackers, write blogs, and remediate problems online. This community participation (which is, itself, a contribution) both increases the awareness of the software’s capabilities while providing tangible software-engineering benefit. For example, StackExchange is a site where open source community members ask questions and receive help from other community members. When the community is large (i.e., the technology is popular), the time to problem remediation using a site like StackExchange is far shorter than with even the best, most expensive support contract for proprietary software. Put another way, a researcher that is having a ‘problem’ with an R package will almost assuredly find someone who has had a similar problem, and a plethora of contributed ‘solutions’ using these social networking venues almost instantly. The quality of the various solutions varies and there are some methods (like voting) that the venues employ to try and help identify the best solutions, but no professional support contract can provide the almost instantaneous remediation opportunity that the open source community does for popular technologies.

What does this social network aspect contribute to data science? First, it is fast, but more importantly, it is a form of peer review. Scientists are well trained in being able to understand the truth revelation mechanisms inherent in peer review. As a result, working with open source software feels natural and scientists quickly develop useful intuition about how to judge the value of different open source technologies.

Whether it's writing the software itself, contributing to the surrounding community, or both, as in any voluntary operation some individuals and groups do more than their fair share of contributions and maintenance work. In some cases, it's a labor of love and completely uncompensated volunteer work. In other cases, large institutions (companies, national labs, nonprofit organizations) invest in open source because of its positive externalities and out of self-interest. Before unpacking the motivations and incentives surrounding open source, a look at a few large Python open source libraries will show the contribution patterns.

2. The Few, Building for the Many

Many popular open source data science packages manage their source code on GitHub,¹ so we can use the features of GitHub to gather data about contribution patterns. We use a few of the most popular Python packages as the focus of our analysis: pandas, a data manipulation and analysis tool (McKinney, 2010); Scikit-Learn, which implements machine learning algorithms (Pedregosa et al., 2011); Matplotlib, for data visualization (Hunter, 2007); NumPy for numerical and scientific computing (Oliphant, 2006; van der Walt et al., 2011); StatsModels, for statistical modeling (Seabold & Perktold, 2010); and TensorFlow, foundational code for building and training neural networks (Abadi et al., 2015).² Table 1 shows a few standard metrics for these packages as of February 8, 2020.³

Table 1. GitHub statistics on stars, forks, and contributors for several popular open source Python data science libraries as of February 8, 2020.

	Stars	Forks	Contributors
pandas	23.5k	9.3k	1,787
Scikit-Learn	39.2k	19.2k	1,594
Matplotlib	10.8k	4.8k	881
NumPy	12.9k	4.2k	865
StatsModels	4.7k	1.8k	208
TensorFlow	141k	79.8k	2,384

Note. All packages in this sample show a large differential between the large number of people starring and forking a project versus the comparatively small number of people contributing to it. Stars denote people marking a package as one of their favorites, forks denote people making their own copy of the code that they can modify for themselves, and contributors are people who have written source code for the library.

These numbers paint a picture of many more users than contributors for the most popular open source software. The projects we examine have tens or hundreds of thousands of likes but hundreds or low thousands

of people contributing to the software.⁴ There is further stratification within the contributor class, as shown in Figure 1; all the packages follow a trend of a small number of extremely active contributors followed by a long tail (see Eghbal, 2019, for a deeper exploration of this pattern).

Figure 1. The number of lines of code added and deleted for the top 100 contributors to six different Python open source data science projects, as of February 8, 2020. Note the log scale on both the x- and y-axis.

Matt Rocklin (maintainer of Dask, a parallel computing platform for Python) explains the different types of roles in open source, reflected by the data (Rocklin, 2019):

1. **Developers** fix bugs and create features. They write code and docs and generally are agents of change in a software project. There are often many more developers than reviewers or maintainers.
2. **Reviewers** are known experts in a part of a project and are called on to review the work of developers, mostly to make sure that the developers don't break anything, but also to point them to related work, ensure common development practices, and pass on institutional knowledge. There are often more developers than reviewers, and more reviewers than maintainers.
3. **Maintainers** are loosely aware of the entire project. They track ongoing work and make sure that it gets reviewed and merged in a timely manner. They direct the orchestra of developers and reviewers, making sure that they connect to each other appropriately, often serving as dispatcher. Maintainers also have final

responsibility. If no reviewer can be found for an important contribution, they review. If no developer can be found to fix an important bug, they develop. If something goes wrong, it's eventually the maintainer's fault.

When we see outliers in the data around open source contributions, we are seeing the maintainers at work. They are tending to the core architecture of the codebase, writing the infrastructure code that isn't always externally visible but keeps the libraries internally coherent and functional. They make payments on technical debt. And, of course, in many cases, they're also writing the externally facing code like the API, the documentation, and the examples. If we were to argue that open source projects are owned by anyone, these are clearly the owners. And as Figure 1 makes clear, it's a lot of work, amounting to hundreds of thousands, sometimes millions, of lines of code. What motivates and incentivizes this work?

3. The Experience of Maintaining Open Source Software

From a labor economics perspective, it's all rather paradoxical: why do maintainers maintain (Lerner & Tirole, 2000)? Project maintenance is a huge amount of work, but nobody is paying to license the code. In most cases, no money is changing hands at all. And yet the maintainers show up, day after day, doing the work so the rest of us can continue doing data science.

A complex mix of incentives drives maintainers. There are indirect economic benefits: open source maintenance looks good on a resume and the network effects of open source collaboration means maintainers often have a leg up when looking for good jobs (Eghbal) because in open source software, contributions are not anonymous. In GitHub, for example, most contributions are via something called a "pull request," in which a developer requests that a reviewer (more properly called a 'curator') adopt a particular set of contributions. However, the developer (once a pull request is accepted) has an indelible record of having contributed that is publicly available. One of the authors of this piece (RW) was told by an executive at a high-profile Silicon Valley tech company that part of the reason that company makes its software available as open source software is because it provides talent-recruiting opportunities in a competitive job market. In an era where signing bonuses for top coding talent occasionally rival those of public sports figures, there is tremendous value in being able to demonstrate the proficiency necessary to make a contribution to a specific project.

But especially in the long tail of smaller open source projects, where there is no real money to be made, open source maintainers often speak about their contributions in less tangible terms. Maintainers often speak about the satisfaction of having an impact in the community. And they do have an outsized impact: because open source software is used so widely, maintaining a popular project is arguably one of the highest leverage activities that a programmer can undertake, and the altruistic appeal (and, in some cases, ego boost) of that leverage can be powerful. One maintainer put it this way:

You know, everyone kind of has I feel like emotional attachment to open source as a big component to everyone's relationships to open source projects, and at the time, for all the other turmoil in my life, I had a strong, positive emotional attachment to [my open source project] WP-CLI, so it was like, "Of course I

wanna maintain it. I don't care how much time it's gonna take. This is the most meaningful part of my career right now." [laughter] (Hiller et al., 2017).

However, open source maintenance can be burdensome too. Volunteer maintainers usually have 'day jobs,' which means that the open source work happens in their limited free time. Maintenance work is often not particularly fun: its tasks tend more toward project management and gap-filling (Rocklin, 2019), which leaves smaller, more visible, and more incrementally gratifying programming tasks to other contributors. Burnout is common (Arslan, 2018; Cannon, 2017; Hiller et al., 2017). Unfortunately the problem is often exacerbated by the small subset of users who complain, often loudly, about perceived shortcomings of the software. The impossibility of pleasing everybody all the time, in combination with the anonymity and frequent bad etiquette on the internet, can unfortunately become toxic for the maintainers of open source projects, who bear the brunt of user complaints.

4. The Gaps Between Open Source Software and the Needs or Aspirations of the Data Science Community

Open source software has much to recommend it, but a data scientist expecting open source software to hold the solution to all his or her problems will likely come away disappointed. The incentive system for contributing, and the legal implications of some open source licenses, are structural forces that enable the ecosystem as a whole, but at somewhat of a cost.

Open source projects attract contributors based on the incentive system of the ecosystem, which is to say that when someone is looking to contribute to a project, the choice of which project isn't necessarily made on the merits of the projects themselves. This has knock-on effects for which projects 'win' or 'lose' in terms of accumulating a critical mass of users. Google's TensorFlow, for example, is popular as a target for contributions because having made a contribution may provide future employment opportunities with Google. From Google's perspective, a contribution to TensorFlow unequivocally demonstrates the ability to work with an important Google technology. However, there may be other, better open source projects available that compete with TensorFlow that never garner the energy or the interest that TensorFlow does. These projects receive very few contributions and typically (even when their approach is better) fail. Usually, the project's progenitors or curators switch to the more popular (but inferior) project, thereby creating cross-pollination of the ideas—but the 'better' software often doesn't win in the open source community because of the self-interest in the incentive system, both commercial and among volunteers.

A second rude awakening for some users of open source comes when software bugs come into the picture. As mentioned, most data science today is done by practitioners who download and stitch together open source libraries. However, these libraries have not necessarily gone through a quality assurance process to determine that they are actually correct. This issue comes up frequently in the larger open source community, especially regarding security flaws in widespread open source projects (Vaughan-Nichols, 2014; Whitesource, 2020). In

the data science world, the methodologies we use for our everyday tasks are themselves full of subtlety, corner cases, assumptions, and other nuances that layer on additional complexity and opportunity for bugs to creep in.

One of the authors of this piece (RW) offers a real-world example. In the course of some routine research, he asked a graduate student to run a Kolmogorov-Smirnov test on some data (a fairly straightforward and common nonparametric statistical test). The graduate student downloaded some Kolmogorov-Smirnov code in Python and ran the test. It didn't seem right, though, and after about a week's analysis it turns out that the code was implementing the test incorrectly. It worked correctly most of the time, but the researchers were working with data that had properties the authors had not considered.

Imagine if the researchers had not found the error: the results would have been incorrect. There is absolutely no recourse. As a result, the data science community should, of course, feel empowered to use open source for data science to get a good guess about some problem, but if the answer really matters (say when we are working on medically related projects), a researcher should consider alternatives such as implementing the method from scratch or the use of a validated (and possibly for-fee) implementation for the purposes of quality assurance.

Thus, one question that the data science community must face is 'how can it be assured that the methodologies are correctly implemented?' In a closed source context, the same problem exists, but the incentives (legal liability, loss of revenue, etc.) are strong. In an open source context, the incentives are purely reputational and there is little in the way of a structured methodology for quality assurance. The assumption is that there are so many users of the code, someone will come across and correct any errors quickly. For immensely popular projects, that might be true, but not for all projects.

To be clear, the issues of code provenance and how to define or prove that software is 'correct' are issues that are open to healthy debate, and they do not pertain exclusively to open source software. Any data set or research paper can be flawed. Further, chasing down and criticizing a person who introduced a bug, or requiring large amounts of test code to accompany any new contribution, would disincentivize many contributors. The fact remains, though, that the data science community can't thrive in the long term if mistakes are frequently slipping into our analyses via buggy code. For the foreseeable future, it's let the buyer (or downloader, as the case may be) beware.

There are also examples (particularly in data science) where open source software simply does not meet the requirements of the discipline. MATLAB, for example, is a for-fee proprietary software system for numerical computation. It has several open source competitors (GNU Octave being the most notable (Eaton et al., 2014)) but a great deal of data science taking today is done with MATLAB. Indeed, the MATLAB software licenses can be one of the biggest licensing expenses for IT departments at top research universities.

Why has Octave not surpassed MATLAB even though it is free open source software? The first reason is described above: there is not the incentive to contribute to Octave because the reputation for having done so does not carry great professional value. The second reason is that numerical software is notoriously difficult to write, debug, and maintain. Contributing to a project like Octave requires more time and energy (and is more error prone) than contributing to a project such as Matplotlib. Finally, because Octave carries a viral GPL license (see Appendix 1 for a fuller discussion of open source licenses), there is some concern that using it in a nonprofit context (e.g., at a university) will not translate to a commercial context where Octave would be too great a business risk. This familiarity with the technology that is useful in industry may be important to the researchers who are doing the work. Thus, while open source software is certainly a strong catalyst for data science, it does not provide all of the software ecosystem that is required.

5. Institutional Participation in Open Source Projects

Not all open source support is purely voluntary: there can be money to be made in open source, albeit indirectly. Although the code itself is free, there are numerous examples of business models that can be established around open source projects, with the goal of economic viability. Perhaps the most prominent example is Red Hat, a company that built and maintained enterprise installations of the open source Linux operating system. In data science and engineering, prominent examples of companies with significant open source projects include the Databricks data science platform (built by core contributors to the Spark codebase, and making heavy use of that infrastructure), the TensorFlow neural net library (built and maintained by Google, with a look inside this process available in Warden, 2017), Kafka event streaming infrastructure (originally built at LinkedIn and currently maintained largely by the spinoff company Confluent), and the Prophet time-series modeling library (built and maintained by Facebook). In some of these cases, the open source project becomes the engine powering the company's product (as in the case of Databricks and RStudio, a company heavily invested in the R open source community). In other cases, companies provide training, enterprise installation, prioritized bug fixing for customers, and consulting around the open source project (examples include Confluent and Anaconda, which specializes in Python package management). And in still other cases, like Facebook's Prophet and the popular neural net libraries Torch (also built by Facebook) and TensorFlow, there is the appeal of thought leadership in high-profile categories and influence over how the data science community builds models.

Companies also end up supporting open source software when they adopt it and then end up maintaining it to support their own needs. Some companies adopt open source software (in part, because it is free) but then find that as the software ages, it requires resources to enhance and maintain. Once the business case has been established, they then devote engineering resources to the software as a way of assuring its future availability to their business. Hyperscale e-commerce and technical companies (e.g., Google, Amazon, Microsoft), where the software licensing costs would be prohibitive due to their scale, are famous examples of open source software adopters.

In this scenario, though, the positive externalities of this maintenance work are more limited. While many large companies pay employees to work on open source software, they are under no obligation to share the output of this labor with the wider community. Indeed, ‘viral’ open source licenses like GPL2 (the license used with the Linux kernel) state explicitly that a company that shares any software that ‘touches’ a GPL2-licensed software component must share *all* of the software (not just the open source part) with the community. For this reason, many companies fear that their internal proprietary technologies will be ‘tainted’ by viral open source licensing. Crucially, the obligation to share is triggered by the act of distributing (sharing) software that may depend, in some small way, on virally licensed open source software. Thus, many companies elect to forbid the distribution of the open source software enhancements they make because they cannot be completely certain that any such an enhancement does not also contain something common to a propriety software component and, as a result, require the release of the proprietary component.

There are other maintainers who are funded by companies but aren’t their direct employees. For example, Wes McKinney (author of pandas) and Hadley Wickham (author of dplyr, ggplot2, and many more) founded Ursa Labs in 2018 upon a business model of getting support grants from companies that benefit from open source projects. Numerous open source maintainers will teach training courses, write books, or accept consulting work centered on their projects. GitHub introduced a sponsorship feature in 2019 that allowed individuals to make financial donations toward projects and maintainers (Wiggers 2019).

And last but not least, a variety of nonprofit and not-for-profit foundations and government grants are available, although especially in the latter case there is not a strong track record of open source projects getting funding. As recounted in Nowogrodzki (2019), for example, within a 5-day period in 2019, Matplotlib was used to reconstruct the first image of a black hole and also denied grant funding for not having a big enough impact. So even when there’s money theoretically available for it, open source maintenance can be a risky career move and certainly does not guarantee ease or riches.

6. What the Data Science Community Can Do to Support Open Source Projects

As users of open source software, which we all are, there are steps we can take to keep this ecosystem healthy.

Individual actions can have significant impact. On an interpersonal level, we should remember that maintainers do a lot of valuable work: thank them when you get the chance, and refrain from unnecessary hostility. Buy their books, attend their seminars, and grant them bonus points when they apply to be hired at your company. For projects that accept monetary contributions from users, for example, via the GitHub Sponsors feature, consider making a donation. And last but not least, consider getting involved in open source as a contributor yourself.

Academic and research institution users also have a powerful set of tools for incentivizing open source projects, starting with the citation system. If a research lab uses an open source package in analyzing data, the resulting papers should cite the software the same as any other reference (readers wishing to go deeper into this topic may refer to (“Giving Software Its Due,” 2019; Jackson 2019; Katz & Chue Hong 2018; Katz et al, 2019; Smith et al., 2016). For the open source developers that are in academic institutions, maintaining a widely cited software package should convey academic bona fides on par with writing a widely cited paper.

Second, given the reach and importance of open source software, it’s deserving of far more consideration in the grant process. Research grants for open source development would incentivize academic participants to invest in building and sharing software analogously to how they build and share research. Critically, grant funding could focus on the projects that for-profit organizations might not prioritize, but which are especially important to the scientific community (think of your favorite particle physics simulator, or a gene transcription package). These grants could come from private philanthropies: the Alfred P. Sloan Foundation, Chan Zuckerberg Initiative, Helmsley Charitable Trust, and Gordon and Betty Moore Foundation are a few examples that have funded open source development in the past. Another option is for government grants, like those offered through the National Science Foundation (NSF), to favor research proposals that include creating or maintaining important open source initiatives. Spark is a success story here: it was originally funded through a large-scale NSF “Expedition” grant and then evolved to become self-sustaining (most notably through Databricks), but both the origin story and the transformation to self-sustainability are more the exception than the rule.

When it comes to thinking about open source development, maintenance is a concept worth further emphasis. As many of us know too well, starting new projects can generate more excitement than keeping legacy projects running smoothly. But maintenance requires real investment, it’s often not glamorous, and it’s critical for the overall health of the ecosystem: having software that ‘just works’ is the expectation of users. Accordingly, investment (of whatever kind) in open source should support ongoing maintenance of features, not just the initial build phase.

Organizations can also have an impact on the health of the open source ecosystem. Commercial organizations should consider sponsoring NumFOCUS, which supports a number of popular Python data science open source packages, or hosting meetups for contributors to work on open source projects together. Companies with the resources can also sponsor open source directly, for example, by employing maintainers and allowing some of their work hours to go toward open source work. Every organization can find open source projects that benefit them, and should feel an obligation to pay it back toward those projects or pay it forward to the new projects that are up and coming.

Data science is a popular field right now, with many people and organizations looking to find their role in it. Investing in the foundational people and projects, namely, in the data science open source software community, is a worthwhile place to focus. Without these people and projects, data science as we know it could not exist. It

follows that the contributions (code, money, psychological support and encouragement) we make today will keep our field healthy for years to come.

Disclosure Statement

Katie Malone's former employer (Civis Analytics) is a silver corporate sponsor of NumFOCUS.

Appendix

Guide to Common Open Source Licenses

The legal terms of an open source project are covered by its license, which spells out the terms and conditions of using, modifying, and distributing the software. The license can strongly shape the incentives and dynamics surrounding a project.

For example, 'viral' open source licenses like GPL2 (the license used with the Linux kernel) state explicitly that a company that shares any software that 'touches' a GPL2-licensed software component must share *all* of the software (not just the open source part) with the community. For this reason, many companies fear that their internal proprietary technologies will be 'tainted' by viral open source licensing. Crucially, the obligation to share is triggered by the act of distributing (sharing) software that may depend, in some small way, on virally licensed open source software. Therefore many companies that pay employees to work on open source software (but are under no obligation to share the output of this labor with the wider community) elect to forbid the distribution of the open source software enhancements they make, because they cannot be completely certain that any such an enhancement does not also contain something common to a propriety software component and, as a result, require the release of the proprietary component.

The dynamics of an open source project can be starkly different depending on how permissive or viral that project's license is (open source software licenses are categorized as being either 'viral' or 'permissive' based on whether they have the power to taint other software; viral licenses are often also referred to as 'copyleft'). Google and Netflix are famously companies that make large corpuses of their open source software and enhancements publicly available under a nonviral open source license. Amazon, Microsoft, and IBM make comparatively little of their open source software contributions available to the community.

This situation is presently in flux. For example, IBM recently acquired Red Hat (a company that makes many of its products available as open source software), signaling a greater commitment to open source. Moreover, viral licenses (once the dominant open source license style) are waning in popularity. Both of these trends bode well for open source software and data science.

A few of the most prevalent licenses are briefly described below. Developers and users of open source software are encouraged to consult the licenses themselves, or legal professionals, for further details.

Apache License

Under the terms of Apache-licensed code, anyone may copy or distribute the code or any derived/modified version of the code, and may charge a fee or do this free of charge. The software itself can be run for any purpose, including commercial purposes. The Apache license has explicit terms covering patenting rights to the software, in addition to copyright.

A distributor must include original copyright, patent, trademark, and attribution notices with the software. Modifications must be explicitly stated, by default are covered under Apache terms, but can be covered under other licenses if explicitly stated. Unmodified code remains covered by Apache terms. The Apache license is a permissive license.

BSD License

Under the terms of BSD-licensed code, anyone may copy or distribute the code or any derived/modified version of the code, and may charge a fee or do this free of charge. The software itself can be run for any purpose, including commercial purposes.

There are several versions of the BSD license, some of which require that the distributor must include the BSD terms and conditions with the software. The three-clause BSD license forbids using the names or affiliations of the contributors as endorsements or promotions of any products derived from the software. The four-clause BSD license requires attribution of the original authors.

If BSD-licensed code is modified, the original software remains BSD-licensed but the modifications themselves can be licensed under other terms. BSD-licensed software is distributed as is, without any guarantees or liability on the part of the software's authors, distributors, and so on. The BSD license is a permissive license.

MIT License

Under the terms of MIT-licensed code, anyone may copy or distribute the code or any derived/modified version of the code, and may charge a fee or do this free of charge. The software itself can be run for any purpose, including commercial purposes.

A distributor must include the MIT terms and conditions with the software.

If MIT-licensed code is modified, the original software remains MIT-licensed, but the modifications themselves can be licensed under other terms. MIT-licensed software is distributed as is, without any

guarantees or liability on the part of the software's authors, distributors, and so on. The MIT license is a permissive license.

GNU General Public License (GPL)

Under the terms of GPL-licensed code, anyone may copy or distribute the code or any derived/modified version of the code, and may charge a fee or do this free of charge. The software itself can be run for any purpose, including commercial purposes.

A distributor must include the GPL terms and conditions with the software, and may not impose further restrictions than those imposed by the GPL.

If anyone wants to sell or distribute GPL-licensed code, the entirety of the source code must be made available to users, including any modifications and additions, covered under the GPL terms. Therefore the GPL license places a copyleft/viral requirement on distributing modified versions of the source code. This requirement only applies to distribution, and does not mean the source code is subject to GPL terms if modifications are kept private.

References

- Abadie, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, M., Dandelion-Man, ... (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems* [Computer software]. TensorFlow. <https://www.tensorflow.org/>
- Arslan, F. (2018, October 10). *Taking an indefinite sabbatical from my projects*. [Blog post]. <https://arslan.io/2018/10/09/taking-an-indefinite-sabbatical-from-my-projects/>
- Cannon, B. (2017, August 29). *The give and take of open source* [Presentation]. JupyterCon 2017, New York, NY. <https://www.youtube.com/watch?v=y19s6vPpGXA>.
- Eaton, J. W., Bateman, D., Hauberg, S., & Wehbring, R. (2014). *GNU Octave (Version 3.8.1): A high-level interactive language for numerical computations* [Computer software]. CreateSpace Independent Publishing Platform. <http://www.gnu.org/software/octave/doc/interpreter/>
- Eghbal, N. (2016). *Roads and bridges: The unseen labor behind our digital infrastructure*. Ford Foundation. <https://www.fordfoundation.org/work/learning/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/>
- Eghbal, N. (2019, May). The rise of few-maintainer projects. *Increment*, 9. <https://increment.com/open-source/the-rise-of-few-maintainer-projects/>

Giving software its due. (2019). *Nature Methods*, 16(3), 207. <https://doi.org/10.1038/s41592-019-0350-x>

Hiller, C., Eghbal, N., & Rogers, M. (2017, November 1). *Request for commits #15: Maintaining a popular project and managing burnout* [Audio podcast]. Changelog. <https://changelog.com/rfc/15>

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>

Jackson, M. (2019). *How to cite and describe software*. Software Sustainability Institute. <https://www.software.ac.uk/how-cite-software>

Katz, D. S., Bouquin, D., Chue Hong, N. P., Hausman, J., Jones, C., Chivvis, D., Clark, T., Crosas, M., Druskat, S., Fenner, M., Gillespie, T., Gonzalez-Beltran, A., Gruenpeter, M., Habermann, T., Haines, R., Harrison, M., Henneken, E., Hwang, L., Jones, M., Kelly, A., ... Zhang, Q. (2019, May 21). Software citation implementation challenges. *arXiv*. <https://doi.org/10.48550/arXiv.1905.08674>

Katz, D. S., & Chue Hong, N. P. (2018). Software citation in theory and practice. In *Lecture Notes in Computer Science: Vol. 10931. Mathematical Software – ICMS 2018* (pp 289–296). https://doi.org/10.1007/978-3-319-96418-8_34

Lerner, J., & Tirole, J. (2000, March). *The simple economics of open source*. National Bureau of Economic Research, Working Paper No. 7600. <https://www.nber.org/papers/w7600.pdf>

McKinney, W. (2010). Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference* (pp. 56–61). <https://doi.org/10.25080/MAJORA-92BF1922-00A>

Nowogrodzki, A. (2019, July 1). How to support open-source software and stay sane. *Nature*, 571, 133–134. <https://doi.org/10.1038/d41586-019-02046-0>

Oliphant, T. E. (2006). *A guide to NumPy*. Trelgol Publishing.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011, November). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. <https://scikit-learn.org/stable/about.html>

Piatetsky, G. (2019, May). Python leads the 11 top data science, machine learning platforms: Trends and analysis. *KDnuggets*. <https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html>

Raymond, E. S. (1999, October). The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary. O'Reilly Media. <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>

Rocklin, M. (2019, May 18). *The role of a maintainer* [Blog post].

<https://matthewrocklin.com/blog/2019/05/18/maintainer>

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. In *Proceedings of the 9th Python in Science Conference*. <https://www.statsmodels.org/stable/index.html>

Smith, A. M., Katz, D. S., & Niemeyer K. E. (2016). FORCE11 Software Citation Working Group. Software citation principles. *PeerJ Computer Science*, 2, Article e86. <https://doi.org/10.7717/peerj-cs.86>

Thakker, D., Schireson, M., & Nguyen-Huu, D. (2017, April 17). *Tracking the explosive growth of open-source software*. TechCrunch. <https://techcrunch.com/2017/04/07/tracking-the-explosive-growth-of-open-source-software/>

van der Walt, S., Colbert, S. C., & Varouquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2), 22–30. <https://doi.org/10.1109/MCSE.2011.37>

Vaughan-Nichols, S. (2014, April 14). *Heartbleed: Open source's worst hour*. ZDNet. <https://www.zdnet.com/article/heartbleed-open-sources-worst-hour/>

Warden, P. (2017, May 4). *How the TensorFlow team handles open source support*. O'Reilly. <https://www.oreilly.com/ideas/how-the-tensorflow-team-handles-open-source-support>

Wiggers, K. (2019, May 23). *GitHub adds donation button, token scanning, and enterprise tools*. VentureBeat. <https://venturebeat.com/2019/05/23/github-adds-donation-button-token-scanning-and-enterprise-tools/>

WhiteSource. (2020). *Annual report: The state of open source security vulnerabilities*. <https://www.whitesourcesoftware.com/open-source-vulnerability-management-report/#>

©2020 Katie Malone and Rich Wolski. This article is licensed under a Creative Commons Attribution (CC BY 4.0) [International license](https://creativecommons.org/licenses/by/4.0/), except where otherwise indicated with respect to particular material included in the article.

Footnotes

1. GitHub is a website for code management built on top of the (open source) Git version control system. ↵
2. The R ecosystem has similar dynamics, but it can be harder to quantify because of the large number of packages managed outside of GitHub, which makes the data on contributions and downloads more opaque. ↵
3. Unfortunately, total downloads of a packages is not made available to the public on GitHub, and none of these metrics provide a perfect proxy for the overall popularity of a package. Visibility is increasing, though,

as open source software continues to occupy a central spot in the tech ecosystem (Thakker et al., 2017). [↵](#)

4. It's worth noting that the packages examined in this analysis are among the most popular in the data science ecosystem, and there is a long tail of projects with far smaller numbers of users and contributors. [↵](#)