`davos`: The Python package smuggler

Paxton C. Fitzpatrick, Jeremy R. Manning*

Department of Psychological and Brain Sciences
Dartmouth College, Hanover, NH 03755

Abstract

Keywords: Python, Jupyter Notebook, Google Colaboratory, Reproducibility, Package management, Pip install

---

*Corresponding author
Email address: `Jeremy.R.Manning@Dartmouth.edu` (Jeremy R. Manning)

Required Metadata

Current code version

Nr.	**Code metadata description**	**Please fill in this column**
C1	Current code version	v0.1.1
C2	Permanent link to code/repository used for this code version	https://github.com/ContextLab/davos/tree/v0.1.1
C3	Code Ocean compute capsule	
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, JavaScript, pip, IPython, Jupyter, Ipykernel, PyZMQ. Additional tools used for tests: pytest, Selenium, Requests, mypy, GitHub Actions
C7	Compilation requirements, operating environments & dependencies	Dependencies: Python>=3.6, packaging, setuptools. Supported OSes: MacOS, Linux, Unix-like. Supported IPython environments: Jupyter notebooks, JupyterLab, Google Colaboratory, Binder, IDE-based notebook editors.
C8	If available Link to developer documentation/manual	https://github.com/ContextLab/davos#readme
C9	Support email for questions	contextualdynamics@gmail.com

Table 1: Code metadata

1. Motivation and significance

Modern scientific research often involves writing code for a variety of purposes throughout the research process, from designing experiments, to analyzing and visualizing data, to presenting findings or techniques via tutorials or public demos. One requirement common to nearly all research-related code it that its behavior and/or outputs must remain consistent and predictable every time it is run, both over time and across users. For example, this stability can be critical to ensuring that data are collected under constant conditions (e.g., across recordings, trials, or subjects) and that statistical analyses yield accurate, reliable results. Additionally, modern open science practices encourage sharing code and data publicly to enable others to explore, reproduce, learn from, and build upon existing work. Scientists, researchers, and educators may seek to share research code with audiences that have a wide range of scientific and technical backgrounds, including collaborators, students, the broader scientific community, and the general public.

Python has become one of the most widely used and fastest-growing scientific programming languages by combining an accessible, high-level syntax with the availability of powerful third-party tools that facilitate rapid development and collaboration [1]. The Python ecosystem offers an extensive data science toolkit, with platforms for interactive programming (e.g., Project Jupyter [2], Google Colaboratory), community-maintained libraries for data manipulation (e.g., NumPy [3], SciPy [4], Pandas [5]) and visualization (e.g., Matplotlib [6], seaborn [7]), frameworks for training complex machine learning models (e.g., scikit-learn [8], TensorFlow [9], Hugging Face [10]), and myriad other resources. However, this heavy emphasis on third-party libraries also presents a challenge to writing and sharing stable, reproducible scientific Python code: different versions of the same library may behave differently, adopting different syntax, adding or dropping support for specific functions or other libraries, addressing (or introducing) bugs, and so on. While these issues are present to some extent in any language or ecosystem, they have a particular impact on the Python community due to its unusually rapid growth relative to other languages. Ensuring Python code behaves consistently over time and across users therefore typically requires making sure it is always run with the same specific set of versions for each package used.

One common approach to solving this problem is to create containerized or virtualized Python environments (e.g., using Docker, Singularity, or conda) tailored to individual applications. Researchers can then share these environments alongside their code as configuration files that explicitly list required package versions, enabling other users to build identical copies for themselves. While often effective, this approach comes with two notable drawbacks: First, it can add significantly to the technical knowledge, system resources, and initial setup required to share and run the actual code of interest. For example, sharing research code that relies on a particular Docker image to run properly not only necessitates distributing extra configuration files and setup instructions, but also requires that both the original author and anyone with whom the code may be shared install and navigate additional software that is likely more complex and resource-intensive than the Python code it is used to manage. These added prerequisites clash with the simplicity and accessibility that have contributed to Python's popularity, and can create significant barriers to both contributing to and taking advantage of open science resources.

Second, although many tools exist for installing Python packages in an isolated environment from a fixed set of required packages (e.g., in a `requirements.txt`, `pyproject.toml`, `environment.yml`, `Pipfile`, `Dockerfile RUN` instruction, etc.), few if any enforce that the specified requirements *remain* satisfied after this initial setup. The ability to modify an environment after its creation is useful in many cases (e.g., to install additional software as needed); however, it also makes it easy to inadvertently alter existing packages, potentially leading to subtle issues with code that relies on them. For instance, suppose a researcher has implemented a series of analyses using version 1.0 of "Package X," and decides to perform an additional analysis that requires installing "Package Y." If Package Y depends on version 0.9 of Package X, then Package X will be downgraded to accommodate this new requirement, potentially altering or breaking prior analyses for both the researcher and anyone with whom their code may be shared. Further, if certain analyses require Package Y while others rely on a feature of Package X not implemented until version 1.0, it's unclear which version the researcher should install in the environment.

2. Software description

2.1. Software Architecture

2.2. Software Functionalities

2.3. Sample code snippets analysis (optional)

3. Illustrative Examples

4. Impact

5. Conclusions

Funding

Our work was supported in part by NSF grant number 2145172 to J.R.M. The content is solely the responsibility of the authors and does not necessarily represent the official views of our supporting organizations.

Declaration of Competing Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

References

[1] E. Muller, J. A. Bednar, M. Diesmann, M.-O. Gewaltig, M. Hines, A. P. Davison, Python in neuroscience, Frontiers in Neuroinformatics 9 (2015) 11. doi:10.3389/fninf.2015.00011.

[2] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, Jupyter notebooks – a publishing format for reproducible computational workflows., in: F. Loizides, B. Scmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas, IOS Press, Netherlands, 2016, pp. 97–90. doi:10.3233/978-1-61499-649-1-87.

[3] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant, Array programming with NumPy, Nature 585 (7825) (2020) 357–362. doi:10.1038/s41586-020-2649-2.

[4] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (3) (2020) 261–272. doi:10.1038/s41592-019-0686-2.

[5] W. McKinney, Data structures for statistical computing in Python, in: Proceedings of the Python in Science Conference, 2010, pp. 51–56. doi:10.25080/Majora-92bf1922-00a.

[6] J. D. Hunter, Matplotlib: A 2D graphics environment, Computing in Science and Engineering 9 (3) (2007) 90–95. doi:10.1109/MCSE.2007.55.

[7] M. L. Waskom, seaborn: statistical data visualization, Journal of Open Source Software 6 (60) (2021) 3021. doi:10.21105/joss.03021.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[10] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush, Transformers: State-of-the-Art Natural Language Processing, Association for Computational Linguistics, 2020, pp. 38–45.
URL https://www.aclweb.org/anthology/2020.emnlp-demos.6