`davos`: The Python package smuggler

Paxton C. Fitzpatrick, Jeremy R. Manning*

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

Abstract

`davos` is a Python package for sharing

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 open science practices encourage sharing code and data to enable others to explore, reproduce, learn from, and build upon existing work. Scientists, researchers, and educators produce many different forms of research-related code (e.g., experiments, data analyses, tutorials, demos) that they may seek to share with a wide range of audiences, 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 offering both powerful research functionality and high-level accessibility [1]. In addition to the language's intuitive, readable grammar and large standard library, the Python ecosystem provides an extensive data science toolkit designed to facilitate rapid development and collaboration, including 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]), and myriad other tools.

However, this also presents a challenge to sharing reproducible scientific Python code: different versions of a given package or library can behave differently, use different syntax, add or drop support for specific functions or other libraries, address (or introduce) 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 stable and reproducible results over time and across users therefore typically requires ensuring that shared code is always run with a specific set of versions for each package used.

One common approach to solving this problem involves creating containerized or virtualized Python environments (e.g., using Docker, Singularity, or conda) tailored to individual applications. Authors may then share these environments alongside their code as configuration files from which users may build identical copies themselves. While effective, a significant drawback to this approach is that it introduces additional prerequisite knowledge

a significant drawback to this approach is that it adds to the requisite knowledge

in many cases, it introduces a level of complexity beyond . For example, distributing research code that relies on a particular Docker image to run properly not only necessitates extra configuration files and setup steps, but requires that both the author and end user install and navigate additional software that is often more complicated and resource-intensive than the actual code being shared. These added prerequisites clash with the simplicity and accessibility that have helped popularize Python among researchers, and can create barriers to both contributing to and taking advantage of open science. `davos` was developed with the goal of addressing

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.