

davos: The Python package smuggler

Paxton C. Fitzpatrick, Jeremy R. Manning*

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

Abstract

Reproducible code plays many important roles in modern scientific research: it enables scientists to collaborate on projects, replicate and extend prior work, and teach students new concepts and techniques via hands-on, interactive tutorials. However, existing tools that facilitate creating, sharing, and running reproducible code are often highly complex and resource-intensive, creating barriers to both contributing to and benefitting from open science resources. Here, we introduce **davos**, a Python package that allows users to create and share reproducible workflows

davos is also more effective at ____ than typical tools,

Keywords: Reproducibility, Open science, Python, Jupyter Notebook, Google Colaboratory, Package management

*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 \geq 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 entails writing software code for a wide variety of purposes throughout the scientific process. Researchers across disciplines may design and implement complex experiments; collect, store, and analyze large datasets; create visualizations for presentations and publications; and share their findings and techniques with peers, students, and the broader public through tutorials, demos, workshops, and classes. However, one requirement common to virtually all forms of research-related code is that its behavior and outputs remain consistent and predictable, no matter when, where, or by whom it is run. This stability can be crucial to ensuring, for example, that data are collected under the same conditions (e.g., across

12 recordings, trials, or subjects) over multiple months or years, and that they
13 can be accessed, processed, and analyzed consistently by a research team
14 that may be spread across multiple institutions or countries. Additionally,
15 modern open science practices encourage publicly sharing research code and
16 data so that others may explore, reproduce, learn from, and build upon ex-
17 isting work. Much of the benefit afforded by freely available research code
18 depends on users’ ability to execute it and achieve the same result as its
19 original author.

20 Python [1] has become one of the most widely used and fastest-growing
21 scientific programming languages, in part by combining an accessible, high-
22 level syntax with a rich ecosystem of powerful third-party tools that facilitate
23 rapid development and collaboration [2]. The Python ecosystem offers an ex-
24 tensive data science toolkit with platforms for interactive programming (e.g.,
25 Project Jupyter [3], Google Colaboratory), community-maintained libraries
26 for data manipulation (e.g., NumPy [4], SciPy [5], Pandas [6]) and visual-
27 ization (e.g., Matplotlib [7], seaborn [8]), frameworks for training complex
28 machine learning models (e.g., scikit-learn [9], TensorFlow [10], Hugging
29 Face [11]), and myriad other resources. However, this heavy emphasis on
30 third-party libraries also presents a challenge to writing and sharing stable,
31 reproducible scientific Python code: different versions of the same library
32 may behave differently, adopt changes in syntax, expose different functions
33 and interfaces, add or drop support for specific hardware or software, write
34 (or expect to read) files in different formats, fix (or introduce) bugs, and
35 so on. While these issues exist to some extent in any software language
36 or ecosystem, they have a particular impact on the Python community due
37 to its unusually rapid growth relative to other languages. Ensuring Python
38 code behaves consistently over time and across users therefore typically re-
39 quires ensuring it is always run with the same specific set of versions for each
40 third-party package used.

41 One common approach to solving this problem is to create containerized
42 or virtualized Python environments (e.g., using Docker [12], Singularity [13],
43 or conda [14]) tailored to individual applications. Such an environment may
44 be “entered” by a researcher in order to run a specific experiment or analy-
45 sis, and “exited” once they are finished. These environments can alongside
46 their code as configuration files that explicitly list required package versions,
47 enabling other users to build identical copies for themselves. While often
48 effective, this approach carries two notable drawbacks: First, it can add sub-
49 stantially to the technical knowledge, computing resources, and initial setup
50 required to share and run the actual code of interest. For example, sharing an
51 analysis or tutorial that relies on a particular Docker image to run properly
52 not only necessitates distributing extra configuration files and setup instruc-

tions, but also requires both the original author and anyone with whom the code is shared to first install and navigate additional software that is likely more complex and resource-intensive than the actual code it facilitates running. 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, while many existing tools allow users to populate a Python environment with a fixed set of packages and package versions (e.g., from a `requirements.txt`, `pyproject.toml`, `environment.yml`, `Pipfile`, `Dockerfile` `RUN` instruction, etc.), few, if any, enforce that these specified requirements *remain* satisfied after they are initially installed. 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 previous analyses for both the researcher and anyone with whom they may later share their code. Further, if some 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 their environment.

The `davos` package provides a novel, Python-native framework for creating, sharing, and running reproducible workflows that was designed to address each of these issues.

Simply importing `davos` in a Jupyter notebook

`davos` was designed specifically for use in Jupyter notebook (formerly called IPython notebook [15]) environments, and supports Jupyter notebooks [3], JupyterLab [16], Google Colaboratory, Binder [17], and IDE-based notebook editors.

In simple use cases, `davos` can be used in lieu of more complicated environment management software, entirely eliminating the need for additional configuration files, pre-execution setup, and `.`. For more complex

`davos` can be used on its own, in lieu of

2. Software description

2.1. Software architecture

The `davos` package is structured as two subpackages: a set of “core” modules that implement

92 2.2. Software functionalities

93 2.2.1. The *smuggle* statement

94 Importing `davos` enables an additional Python keyword: “`smuggle`”. The
95 `smuggle` statement can be used as a drop-in replacement for Python’s built-
96 in `import` statement to load libraries, modules, and other objects into the
97 current namespace. However, whereas `import` will fail if the requested pack-
98 age is not installed locally, `smuggle` statements can handle missing packages
99 on the fly. If a smuggled package does not exist in the local environment,
100 `davos` will install it, expose its contents to Python’s `import` machinery, and
101 load it into the namespace for immediate use.

102 2.2.2. The *onion* comment

103 For greater control over the behavior of `smuggle` statements, `davos` de-
104 fines an additional construct called the *onion comment*. An onion comment
105 is a special type of inline comment that may be placed on a line containing a
106 `smuggle` statement to customize how `davos` searches for the smuggled pack-
107 age locally and, if necessary, how it should be installed. Onion comments
108 follow a simple syntax based on the “type comment” syntax introduced in
109 PEP 484 [18], and are designed to make controlling installation via `davos`
110 intuitive and familiar. To construct an onion comment, simply provide the
111 name of the installer program (e.g., `pip`) and the same arguments one would
112 use to install the package as desired manually via the command line (see
113 Fig. 1).

```
import davos

# if numpy is not installed locally, pip-install it and display verbose output
smuggle numpy as np      # pip: numpy --verbose

# pip-install pandas without using or writing to the package cache
smuggle pandas as pd     # pip: pandas --no-cache-dir

# install scipy from a relative local path, in editable mode
from scipy.stats smuggle ttest_ind      # pip: -e ../../pkgs/scipy
```

Figure 1: **Figure 1**

114 2.2.3. *The **davos** config*
115 2.2.4. *Additional functionality*
116 2.3. *Sample code snippets analysis (optional)*
117 **3. Illustrative Examples**

118 **4. Impact**

119 Despite the short time since its conception and initial release, **davos** has
120 already found use in a variety of applications. In addition to managing data
121 analysis environments for multiple ongoing research studies, **davos** is being
122 used by both students and instructors in programming courses such as *Sto-*
123 *rytelling with Data* [19] (an open course on data science, visualization, and
124 communication) to simplify distributing lessons and submitting assignments,
125 as well as in online demos such as **abstract2paper** [20] (an example appli-
126 cation of **GPT-Neo**) to share ready-to-run code that installs dependencies
127 automatically.

128 **5. Conclusions**

129 **Author Contributions**

130 **Paxton C. Fitzpatrick:** Conceptualization, Methodology, Software,
131 Validation, Writing - Original Draft, Visualization. **Jeremy R. Man-**
132 **ning:** Conceptualization, Resources, Writing - Review & Editing, Super-
133 vision, Funding acquisition.

134 **Funding**

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

138 **Declaration of Competing Interest**

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

142 Acknowledgements

143 References

- 144 [1] G. van Rossum, Python reference manual, Department of Computer
145 Science [CS] (R 9525) (1995).
- 146 [2] E. Muller, J. A. Bednar, M. Diesmann, M.-O. Gewaltig, M. Hines, A. P.
147 Davison, Python in neuroscience, *Frontiers in Neuroinformatics* 9 (2015)
148 11. [doi:10.3389/fninf.2015.00011](https://doi.org/10.3389/fninf.2015.00011).
- 149 [3] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier,
150 J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov,
151 D. Avila, S. Abdalla, C. Willing, Jupyter notebooks – a publish-
152 ing format for reproducible computational workflows., in: F. Loizides,
153 B. Schmidt (Eds.), *Positioning and Power in Academic Publishing: Play-*
154 *ers, Agents and Agendas*, IOS Press, Netherlands, 2016, pp. 97–90.
155 [doi:10.3233/978-1-61499-649-1-87](https://doi.org/10.3233/978-1-61499-649-1-87).
- 156 [4] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Vir-
157 tanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith,
158 R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane,
159 J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Shep-
160 pard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant,
161 *Array programming with NumPy*, *Nature* 585 (7825) (2020) 357–362.
162 [doi:10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2).
- 163 [5] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy,
164 D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright,
165 S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. May-
166 orov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey,
167 Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perk-
168 told, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M.
169 Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0
170 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Com-*
171 *puting in Python*, *Nature Methods* 17 (3) (2020) 261–272. [doi:](https://doi.org/10.1038/s41592-019-0686-2)
172 [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
- 173 [6] W. McKinney, Data structures for statistical computing in Python, in:
174 *Proceedings of the Python in Science Conference*, 2010, pp. 51–56. [doi:](https://doi.org/10.25080/Majora-92bf1922-00a)
175 [10.25080/Majora-92bf1922-00a](https://doi.org/10.25080/Majora-92bf1922-00a).

- [7] J. D. Hunter, Matplotlib: A 2D graphics environment, *Computing in Science and Engineering* 9 (3) (2007) 90–95. doi:10.1109/MCSE.2007.55.
- [8] M. L. Waskom, seaborn: statistical data visualization, *Journal of Open Source Software* 6 (60) (2021) 3021. doi:10.21105/joss.03021.
- [9] 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.
- [10] 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. Wattemberg, 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/>
- [11] 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>
- [12] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, *Linux Journal* 239 (2) (2014) 2.
- [13] G. M. Kurtzer, V. Sochat, M. W. Bauer, Singularity: Scientific containers for mobility of compute, *PLoS One* 12 (5) (2017) e0177459.
- [14] Anaconda, Inc., conda, <https://docs.conda.io> (2012).
- [15] F. Pérez, B. E. Granger, Ipython: a system for interactive scientific computing, *Computing in science & engineering* 9 (3) (2007) 21–29. doi:10.1109/MCSE.2007.53.

- 210 [16] B. Granger, J. Grout, Jupyterlab: Building blocks for interactive com-
211 puting, Slides of presentation made at SciPy (2016).
- 212 [17] B. Ragan-Kelley, C. Willing, Binder 2.0-reproducible, interac-
213 tive, sharable environments for science at scale, in: F. Akici,
214 D. Lippa, D. Niederhut, , M. Pacer (Eds.), Proceedings of the 17th
215 Python in Science Conference, 2018, pp. 113–120. doi:10.25080/
216 MAJORA-4AF1F417-011.
- 217 [18] G. van Rossum, J. Lehtosalo, L. Langa, Type Hints, PEP 484 (Septem-
218 ber 2014).
219 URL <https://www.python.org/dev/peps/pep-0484>
- 220 [19] J. R. Manning, Storytelling with Data (June 2021). doi:10.5281/
221 zenodo.5182775.
222 URL <https://doi.org/10.5281/zenodo.5182775>
- 223 [20] J. R. Manning, abstract2paper (2021).
224 URL <https://github.com/ContextLab/abstract2paper>