

davos: a Python package “smuggler” for constructing lightweight reproducible notebooks

Paxton C. Fitzpatrick, Jeremy R. Manning*

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

Abstract

Reproducibility is a core requirement of modern scientific research. For computational research, reproducibility means that code should produce the same results, even when run on different systems. A standard approach to ensuring reproducibility entails packaging a project’s dependencies along with its primary code base. Existing solutions vary in how deeply these dependencies are specified, ranging from virtual environments, to containers, to virtual machines. Each of these existing solutions requires installing or setting up a system for running the desired code that must be packaged alongside the primary code base. Here we propose a lighter-weight solution than virtual environments: the **davos** library. When used in combination with a notebook-based Python project, the **davos** library provides a mechanism for specifying (and automatically installing) the correct package versions of the project’s dependencies. The **davos** library also ensures that those versions are in use any time the notebook’s code is executed. This enables researchers to share a complete reproducible copy of their code within a single Jupyter notebook file.

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	Metadata value
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, PyPI/pip, IPython, Jupyter, Ipykernel, PyZMQ. Additional tools used for tests: pytest, Selenium, Requests, mypy, GitHub Actions
C7	Compilation requirements, operating environments, and 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	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

The same computer code may not behave identically under different circumstances. For example, when code depends on external libraries, different versions of those libraries may function differently. Or when CPU or GPU instruction sets differ across machines, the same high-level code may be compiled into different machine instructions. Because executing identical code does not guarantee identical outcomes, code sharing in and of itself is often insufficient for enabling researchers to reproduce each others' work.

Within the Python [1] community, external packages that are published in the most popular repositories [2, 3] are associated with version numbers and tags that enable users to guarantee that they are installing exactly the same

code across different computing environments. Despite that it is *possible* to manually install the intended version numbers of every dependency of a Python script or package, manually tracking down those dependencies can impose a substantial burden on the user.

Researchers, programmers, and others have developed a broad set of approaches and tools to facilitate code sharing and reproducible outcomes (Fig. 1). At one extreme, simply publishing a set of Python scripts (.py files) may enable others to use or gain insights into the relevant work. Because Python is installed by default on most modern operating systems, for some projects this may be sufficient. Another popular approach entails creating JSON files, called Jupyter notebooks [4], that comprise a mix of text, executable code, and embedded media. Notebooks may call or import external scripts or libraries in order to provide a more compact and readable experience for users. Each of these systems (Python scripts and notebooks) provides a convenient means of sharing code, with the caveat that they do not specify the computing environment in which the code is executed. Therefore the functionality of code shared using these systems cannot be guaranteed across different computing environments.

At another extreme, virtual machines [5, 6, 7] provide a hardware-level simulation of the desired system. Virtual machines are typically isolated from the user’s system, such that installing or running software on a virtual machine does not impact the user’s primary operating system or computing environment. Containers [e.g., 8, 9] provide a similar “isolated” experience. Although containerized environments do not specify hardware-level operations, they are typically packaged with a complete operating system, in addition to a complete copy of Python and any relevant package dependencies. Virtual environments [e.g., 10] also provide a computing environment that is largely separated from the user’s main environment. They incorporate a copy of Python and the target software’s dependencies, but virtual environments do not specify or reproduce an operating system for the runtime environment. Each of these systems (virtual machines, containers, and virtual environments) guarantees (to differing degrees— at the hardware level, operating system level, and Python environment level, respectively) that the relevant code will run similarly for different users. However, each of these systems also relies on additional software that can be resource intensive or burdensome to install or configure.

We designed **davos** to occupy a “sweet spot” between these extremes. **davos** is a notebook-installable package that adds functionality to the default notebook experience. Like standard Jupyter notebooks, **davos**-enhanced notebooks allows researchers to include text, executable code, and media within a single file. No further setup or installation is required, beyond what

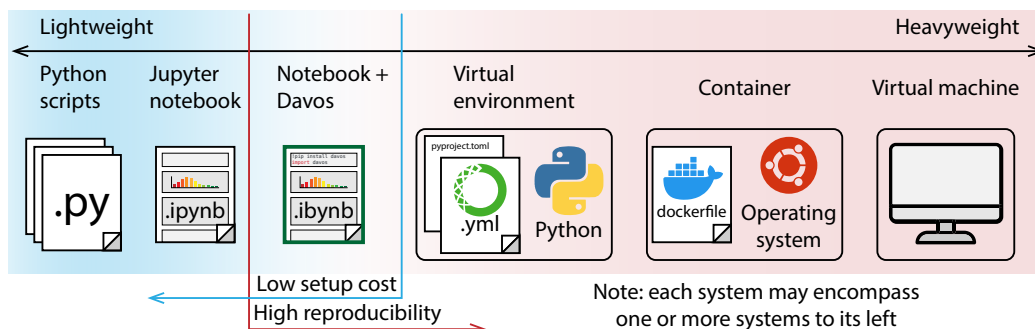


Figure 1: **Systems for sharing code within the Python ecosystem.** From left to right: plain-text **Python scripts** (`.py` files) provide the most basic “system” for sharing raw code. Scripts may reference external libraries, but those libraries must be manually installed on other users’ systems. Further, any checking needed to verify that the correct versions of those libraries were installed must also be performed manually. **Jupyter notebooks** (`.ipynb` files) comprise embedded text, executable code, and media (including rendered figures, code output, etc.). When the **davos** library is imported into a Jupyter notebook, the notebook’s functionality is extended to automatically install the required external libraries (at their correct versions, when specified). **Virtual environments** install an isolated copy of Python and all required dependencies. This typically requires defining a `requirements.txt` file or an environment (`.yml`) file that specifies all project dependencies (including version numbers of external libraries). **Containers** provide a means of defining an isolated environment that includes a complete operating system (independent of the user’s operating system), in addition to (optionally) specifying a virtual environment or other configurations needed to provide the necessary computing environment. Containers are typically defined using specification files (e.g., a plain-text **Dockerfile**) that instruct the virtualization engine regarding how to build the virtual environment. **Virtual machines** provide a complete hardware-level simulation of the computing environment. In addition to simulating specific hardware, virtual machines (typically specified using binary images files) must also define operating system-level properties of the computing environment. Systems to the left of the blue vertical line entail sharing individual files, with no additional installation or configuration needed to run the target code. Systems to the right of the red vertical line support precise control over dependencies and versioning. Notebooks enhanced using the **davos** library are easily shareable and require minimal setup costs, while also facilitating high reproducibility by enabling precise control over project dependencies.

53 is needed to run standard Jupyter notebooks. And like virtual environments
54 **davos** provides a convenient mechanism for fully specifying (and installing, as
55 needed) a complete set of Python dependencies, including package versions.

56 2. Software description

57 2.1. Software architecture

58 The **davos** package consists of two interdependent subpackages. The
59 first, **davos.core**, comprises a set of modules that implement the bulk of the
60 package’s core functionality, including pipelines for installing and validating
61 packages, custom parsers for the **smuggle** statement (see Section 2.2.1) and
62 onion comment (see Section 2.2.2), and a runtime interface for configuring
63 **davos**’s behavior (see Section 2.2.3). However, certain critical aspects of
64 this functionality require (often substantially) different implementation ap-
65 proaches depending on various properties of the notebook environment in
66 which **davos** is used (e.g., whether the frontend is provided by Jupyter or
67 Google Colaboratory, or which version of IPython [11] is used by the note-
68 book kernel). To deal with this, environment-dependent parts of core fea-
69 tures and behaviors are isolated and abstracted to “helper functions” in the
70 **davos.implementations** subpackage. This second subpackage defines multi-
71 ple, interchangeable versions of each helper function, organized into modules
72 by the conditions that trigger their use. At runtime, **davos** detects various
73 features in the notebook environment and selectively imports a single version
74 of each helper function into the top-level **davos.implementations** names-
75 pace, allowing **davos.core** modules to access the correct implementations
76 for the current notebook environment from a single, constant location. An
77 additional benefit of this design pattern is that it makes adding support for
78 new or updated notebook variants to **davos** in the future relatively easy.

79 2.2. Software functionalities

80 2.2.1. The **smuggle** statement

81 Importing **davos** in a Jupyter notebook enables an additional Python
82 keyword: “**smuggle**”. The **smuggle** statement can be used as a drop-in re-
83 placement for Python’s built-in **import** statement to load libraries, modules,
84 and other objects into the current namespace. However, whereas **import** will
85 fail if the requested package is not installed locally, **smuggle** statements can
86 handle missing packages on the fly. If a smuggled package does not exist in
87 the local environment, **davos** will install it, expose its contents to Python’s
88 **import** machinery, and load it into the namespace for immediate use.

89 2.2.2. The onion comment

90 For greater control over the behavior of `smuggle` statements, `davos` de-
91 fines an additional construct called the “onion comment”. An onion comment
92 is a special type of inline comment that may be placed on a line containing a
93 `smuggle` statement to customize how `davos` searches for the smuggled pack-
94 age locally and, if necessary, downloads and installs it. Onion comments
95 follow a simple syntax based on the “type comment” syntax introduced in
96 PEP 484 [10], and are designed to make managing packages with `davos` intu-
97 itive and familiar. To construct an onion comment, simply provide the name
98 of the installer program (e.g., `pip`) and the same arguments one would use
99 to manually install the package as desired via the command line (see Fig. 2,
100 lines 3–10 for examples).

101 Onion comments are useful when smuggling a package whose distribution
102 name (i.e., the name used when installing it) is different from its top-level
103 module name (i.e., the name used when importing it; e.g., Fig. 2, lines 12–13).
104 However, the most powerful use of the onion comment is making `smuggle`
105 statements *version-sensitive*. If an onion comment includes a version specifier
106 [12] (e.g., Fig. 2, lines 15–19), `davos` will ensure that the version of the
107 package loaded into the notebook matches the specific version requested, or
108 satisfies the given version constraints. If the smuggled package exists locally,
109 `davos` will extract its version info from its metadata and compare it to the
110 specifier provided. If the two are incompatible (or no local installation is
111 found), `davos` will install and load a suitable version of the package instead.
112 Onion comments can similarly be used to smuggle specific VCS references
113 (e.g., Git [13] branches, commits, tags, etc.; Fig. 2, lines 21–22).

114 `davos` processes onion comments internally before forwarding arguments
115 to the installer program. In addition to preventing onion comments from
116 being used as a vehicle for shell injection attacks, this allows `davos` take cer-
117 tain logical actions when particular arguments are passed (e.g., Fig. 2, lines
118 24–28). For example, the `--force-reinstall`, `-I/--ignore-installed`,
119 and `-U/--upgrade` flag will all cause `davos` to skip searching for a smug-
120 gled package locally before installing a new copy; `--no-input` will disable
121 `davos`’s input prompts in addition to the installer program’s; and installing
122 a package into `<dir>` with `--target <dir>` will cause `dir` to be prepended
123 to the module search path (`sys.path`), if necessary, so the package can be
124 imported.

```

1  import davos
2
3  # if numpy is not installed locally, pip-install it and display verbose output
4  smuggle numpy as np    # pip: numpy --verbose
5
6  # pip-install pandas without using or writing to the package cache
7  smuggle pandas as pd   # pip: pandas --no-cache-dir
8
9  # install scipy from a relative local path, in editable mode
10 from scipy.stats smuggle ttest_ind    # pip: -e ../../pkgs/scipy
11
12 smuggle dateutil        # pip: python-dateutil
13 from sklearn.decomposition smuggle PCA    # pip: scikit-learn
14
15 # specifically use matplotlib v3.4.2, pip-installing it if needed
16 smuggle matplotlib.pyplot as plt    # pip: matplotlib==3.4.2
17
18 # use a version of seaborn no older than v0.9.1, but before v0.11
19 smuggle seaborn as sns    # pip: seaborn>=0.9.1,<0.11
20
21 # use quail as the package existed on GitHub at commit 6c847a4
22 smuggle quail    # pip: git+https://github.com/ContextLab/quail.git@6c847a4
23
24 # install hypertools v0.7 without first checking for it locally
25 smuggle hypertools as hyp    # pip: hypertools==0.7 --ignore-installed
26
27 # always install the latest version of requests, including pre-releases
28 from requests smuggle Session    # pip: requests --upgrade --pre

```

Figure 2: Example `smuggle` statements and accompanying onion comments.

125 *2.2.3. The `davos` config*

126 *2.2.4. Additional functionality*

127 *2.3. Sample code snippets analysis (optional)*

128 **3. Illustrative Examples**

129 **4. Impact**

130 Like virtual environments, containers, and virtual machines, the `davos` li-
 131 brary (when used in conjunction with Jupyter notebooks) provides a lightweight
 132 mechanism for sharing code and ensuring reproducibility across users and
 133 computing environments (Fig. 1). Further, `davos` enables users to fully
 134 specify (and install, as needed) any project dependencies within the same
 135 notebook. This provides a system whereby executable code (along with text

136 and media) *and* code for setting up and configuring the project dependencies,
137 may be combined within a single notebook file.

138 We designed **davos** for use in research applications. For example, in many
139 settings **davos** may be used as a drop-in replacement for more-difficult-to-
140 set-up virtual environments, containers, and/or virtual machines. For re-
141 searchers, this lowers barriers to sharing code. By eliminating most of the
142 setup costs of reconstructing the original researchers’ computing environ-
143 ment, **davos** also lowers barriers to entry for members of the scientific com-
144 munity and the public who seek to *benefit* from shared code.

145 Beyond research applications, **davos** is also useful in pedagogical settings.
146 For example, in programming courses, instructors and students may import
147 the **davos** library into their notebooks to provide a simple means of ensur-
148 ing their code will run on others’ machines. When combined with online
149 notebook-based platforms like Google Colaboratory, **davos** provides a con-
150 venient way to manage dependencies within a notebook, without requiring
151 any software (beyond a web browser) to be installed on the students’ or in-
152 structors’ systems. For the same reasons, **davos** also provides an elegant
153 means of sharing ready-to-run notebook-based demonstrations that install
154 their dependencies automatically.

155 Since its initial release, **davos** has found use in a variety of applications. In
156 addition to managing computing environments for multiple ongoing research
157 studies, **davos** is being used by both students and instructors in programming
158 courses such as *Storytelling with Data* [14] (an open course on data science,
159 visualization, and communication) to simplify distributing lessons and sub-
160 mitting assignments, as well as in online demos such as **abstract2paper**
161 [15] (an example application of **GPT-Neo**) to share ready-to-run code that
162 installs dependencies automatically.

163 Our work also has several more subtle “advanced” use cases and poten-
164 tial impacts. Whereas Python’s built-in **import** statement is agnostic to
165 packages’ version numbers, **smuggle** statements (when combined with onion
166 comments) are version-sensitive. This enables multiple versions of a single li-
167 brary to be imported within the same notebook. This could be useful in cases
168 where specific features were added or removed from a package across differ-
169 ent versions, or in comparing the performance or functionality of particular
170 features across different versions of the same package.

171 A second advanced use case is in providing a proof-of-concept of how one
172 can add new “keyword-like” operators to the Python language by leverag-
173 ing notebooks’ error-handling mechanisms. This could lead to exciting new
174 tools that, like **davos**, extend the Python language in useful ways within
175 notebook-based environments. We note that our approach to adding the
176 **smuggle** keyword to Python when **davos** is imported into a notebook-based

environment also has the potential to be exploited for more nefarious purposes. For example, a malicious user could use a similar approach (e.g., in a different library) to substantially change a notebook’s functionality by adding new *unexpected* keyword-like objects (e.g., based around common typos). This could lead to difficult-to-predict changes in a notebook’s behavior once the malicious library was imported. This highlights an important reason why security-conscious users would be well-served to only make use of libraries from trusted sources, or whose code is publicly available for review.

5. Conclusions

The **davos** library supports reproducible research by providing a novel lightweight system for sharing notebook-based code. But perhaps the most exciting uses of the **davos** library are those that we have *not* yet considered or imagined. We hope that the Python community will find **davos** to provide a convenient means of managing project dependencies to facilitate code sharing. We also hope that some of the more advanced applications of our library might lead to new insights or discoveries.

Author Contributions

Paxton C. Fitzpatrick: Conceptualization, Methodology, Software, Validation, Writing - Original Draft, Visualization. **Jeremy R. Manning:** Conceptualization, Resources, Validation, Writing - Review & Editing, Supervision, Funding acquisition.

Funding

Our work was supported in part by NSF grant number 2145172 to JRM. 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

We acknowledge useful feedback and discussion from the students of JRM’s *Storytelling with Data* course (Winter, 2022 offering) who used preliminary versions of our library in several assignments.

210 References

- 211 [1] G. van Rossum, Python reference manual, Department of Computer
212 Science [CS] (R 9525) (1995).
- 213 [2] Python Software Foundation, The Python Package Index (PyPI),
214 <https://pypi.org> (2003).
- 215 [3] conda-forge community, The conda-forge Project: Community-based
216 Software Distribution Built on the conda Package Format and Ecosys-
217 tem, <https://doi.org/10.5281/zenodo.4774217> (July 2015). doi:
218 [10.5281/zenodo.4774217](https://doi.org/10.5281/zenodo.4774217).
- 219 [4] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier,
220 J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov,
221 D. Avila, S. Abdalla, C. Willing, Jupyter notebooks – a publish-
222 ing format for reproducible computational workflows., in: F. Loizides,
223 B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Play-
224 ers, Agents and Agendas, IOS Press, Netherlands, 2016, pp. 97–90.
225 doi:[10.3233/978-1-61499-649-1-87](https://doi.org/10.3233/978-1-61499-649-1-87).
- 226 [5] R. P. Goldberg, Survey of virtual machine research, Computer 7 (6)
227 (1974) 34–45.
- 228 [6] Y. Altintas, C. Brecher, M. Weck, S. Witt, Virtual ma-
229 chine tool, CIRP Annals 54 (2) (2005) 115–138. doi:[https://doi.org/10.1016/S0007-8506\(07\)60022-5](https://doi.org/10.1016/S0007-8506(07)60022-5).
230 URL [https://www.sciencedirect.com/science/article/pii/](https://www.sciencedirect.com/science/article/pii/S0007850607600225)
231 [S0007850607600225](https://www.sciencedirect.com/science/article/pii/S0007850607600225)
232
- 233 [7] M. Rosenblum, VMware’s Virtual Platform: A virtual machine monitor
234 for commodity PCs, in: IEEE Hot Chips Symposium, IEEE, 1999, pp.
235 185–196.
- 236 [8] D. Merkel, Docker: lightweight linux containers for consistent develop-
237 ment and deployment, Linux Journal 239 (2) (2014) 2.
- 238 [9] G. M. Kurtzer, V. Sochat, M. W. Bauer, Singularity: Scientific contain-
239 ers for mobility of compute, PLoS One 12 (5) (2017) e0177459.
- 240 [10] G. van Rossum, J. Lehtosalo, L. Langa, Type Hints, PEP 484, Python
241 Software Foundation (September 2014).
242 URL <https://www.python.org/dev/peps/pep-0484>

- 243 [11] F. Pérez, B. E. Granger, Ipython: a system for interactive scientific
244 computing, *Computing in science & engineering* 9 (3) (2007) 21–29.
245 [doi:10.1109/MCSE.2007.53](https://doi.org/10.1109/MCSE.2007.53).
- 246 [12] N. Coghlan, D. Stufft, [Version identification and dependency specifica-](https://peps.python.org/pep-0440)
247 [tion](https://peps.python.org/pep-0440), PEP 440, Python Software Foundation (March 2013).
248 URL <https://peps.python.org/pep-0440>
- 249 [13] L. Torvalds, J. Hamano, Git: Fast version control system, [https://](https://git.kernel.org/pub/scm/git/git.git)
250 git.kernel.org/pub/scm/git/git.git (April 2005).
- 251 [14] J. R. Manning, Data wrangler, Zenodo 10.5281/zenodo.5123310 (2021).
- 252 [15] J. R. Manning, Episodic memory: mental time travel or a quantum
253 “memory wave” function?, *Psychological Review* 128 (4) (2021) 711–
254 725.