

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

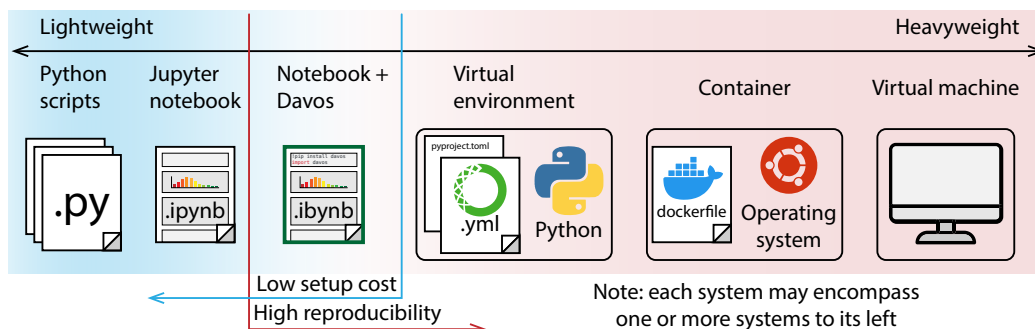


Figure 1: **Systems for sharing code within the Python ecosystem.** From left to right: plain-text **Python scripts** (`.py` files) provide the most basic “system” for sharing raw code. Scripts may reference external libraries, but those libraries must be manually installed on other users’ systems. Further, 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 is structured as two sub-packages: a set of “core”
59 modules that implement...

60 2.2. Software functionalities

61 2.2.1. The *smuggle* statement

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

70 2.2.2. The *onion* comment

71 For greater control over the behavior of **smuggle** statements, **davos** de-
72 fines an additional construct called the *onion comment*. An onion comment
73 is a special type of inline comment that may be placed on a line containing a
74 **smuggle** statement to customize how **davos** searches for the smuggled pack-
75 age locally and, if necessary, how it should be installed. Onion comments
76 follow a simple syntax based on the “type comment” syntax introduced in
77 PEP 484 [10] and are designed to make managing packages via **davos** intu-
78 itive and familiar. To construct an onion comment, simply provide the name
79 of the installer program (e.g., **pip**) and the same arguments one would use
80 to install the package as desired manually via the command line (see Fig. 2).

81 2.2.3. The *davos* config

82 2.2.4. Additional functionality

83 2.3. Sample code snippets analysis (optional)

84 3. Illustrative Examples

85 4. Impact

86 Like virtual environments, containers, and virtual machines, the **davos** li-
87 brary (when used in conjunction with Jupyter notebooks) provides a lightweight

```

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 2: **FILL THIS IN...**

88 mechanism for sharing code and ensuring reproducibility across users and
 89 computing environments (Fig. 1). Further, **davos** enables users to fully
 90 specify (and install, as needed) any project dependencies within the same
 91 notebook. This provides a system whereby executable code (along with text
 92 and media) *and* code for setting up and configuring the project dependencies,
 93 may be combined within a single notebook file.

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

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

111 Since its initial release, **davos** has found use in a variety of applications. In
 112 addition to managing computing environments for multiple ongoing research
 113 studies, **davos** is being used by both students and instructors in programming
 114 courses such as *Storytelling with Data* [11] (an open course on data science,
 115 visualization, and communication) to simplify distributing lessons and sub-
 116 mitting assignments, as well as in online demos such as **abstract2paper**
 117 [12] (an example application of **GPT-Neo**) to share ready-to-run code that

118 installs dependencies automatically.

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

127 A second advanced use case is in providing a proof-of-concept of how one
128 can add new “keyword-like” operators to the Python language by leverag-
129 ing notebooks’ error-handling mechanisms. This could lead to exciting new
130 tools that, like `davos`, extend the Python language in useful ways within
131 notebook-based environments. We note that our approach to adding the
132 `smuggle` keyword to Python when `davos` is imported into a notebook-based
133 environment also has the potential to be exploited for more nefarious pur-
134 poses. For example, a malicious user could use a similar approach (e.g.,
135 in a different library) to substantially change a notebook’s functionality by
136 adding new *unexpected* keyword-like objects (e.g., based around common ty-
137 pos). This could lead to difficult-to-predict changes in a notebook’s behavior
138 once the malicious library was imported. This highlights an important rea-
139 son why security-conscious users would be well-served to only make use of
140 libraries from trusted sources, or whose code is publicly available for review.

141 5. Conclusions

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

149 Author Contributions

150 **Paxton C. Fitzpatrick:** Conceptualization, Methodology, Software,
151 Validation, Writing - Original Draft, Visualization. **Jeremy R. Manning:**
152 Conceptualization, Resources, Validation, Writing - Review & Editing, Su-
153 pervision, Funding acquisition.

154 Funding

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

158 Declaration of Competing Interest

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

162 Acknowledgements

163 We acknowledge useful feedback and discussion from the students of
164 JRM's *Storytelling with Data* course (Winter, 2022 offering) who used pre-
165 liminary versions of our library in several assignments.

166 References

- 167 [1] G. van Rossum, Python reference manual, Department of Computer
168 Science [CS] (R 9525) (1995).
- 169 [2] [Python package index - pypi](https://pypi.org/).
170 URL <https://pypi.org/>
- 171 [3] Anaconda, Inc., conda, <https://docs.conda.io> (2012).
- 172 [4] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier,
173 J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov,
174 D. Avila, S. Abdalla, C. Willing, Jupyter notebooks – a publish-
175 ing format for reproducible computational workflows., in: F. Loizides,
176 B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Play-
177 ers, Agents and Agendas, IOS Press, Netherlands, 2016, pp. 97–90.
178 [doi:10.3233/978-1-61499-649-1-87](https://doi.org/10.3233/978-1-61499-649-1-87).
- 179 [5] R. P. Goldberg, Survey of virtual machine research, Computer 7 (6)
180 (1974) 34–45.
- 181 [6] Y. Altintas, C. Brecher, M. Weck, S. Witt, [Virtual ma-](https://doi.org/10.1016/S0007-8506(07)60022-5)
182 [chine tool](https://doi.org/10.1016/S0007-8506(07)60022-5), CIRP Annals 54 (2) (2005) 115–138. [doi:https:](https://doi.org/10.1016/S0007-8506(07)60022-5)
183 [//doi.org/10.1016/S0007-8506\(07\)60022-5](https://doi.org/10.1016/S0007-8506(07)60022-5).
184 URL [https://www.sciencedirect.com/science/article/pii/](https://www.sciencedirect.com/science/article/pii/S0007850607600225)
185 [S0007850607600225](https://www.sciencedirect.com/science/article/pii/S0007850607600225)

- 186 [7] I. VMware, R. Calculator, VMware (2018).
- 187 [8] D. Merkel, Docker: lightweight linux containers for consistent develop-
188 ment and deployment, Linux Journal 239 (2) (2014) 2.
- 189 [9] G. M. Kurtzer, V. Sochat, M. W. Bauer, Singularity: Scientific contain-
190 ers for mobility of compute, PLoS One 12 (5) (2017) e0177459.
- 191 [10] G. van Rossum, J. Lehtosalo, L. Langa, [Type Hints](#), PEP 484 (Septem-
192 ber 2014).
193 URL <https://www.python.org/dev/peps/pep-0484>
- 194 [11] J. R. Manning, [Storytelling with Data](#) (June 2021). [doi:10.5281/](https://doi.org/10.5281/zenodo.5182775)
195 [zenodo.5182775](https://doi.org/10.5281/zenodo.5182775).
196 URL <https://doi.org/10.5281/zenodo.5182775>
- 197 [12] J. R. Manning, [abstract2paper](#) (2021).
198 URL <https://github.com/ContextLab/abstract2paper>