

# 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	<a href="https://github.com/ContextLab/davos/tree/v0.1.1">https://github.com/ContextLab/davos/tree/v0.1.1</a>
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 $\geq 3.6$ , packaging, setuptools. Supported OSes: MacOS, Linux, Unix-like. Supported IPython environments: Jupyter notebooks, JupyterLab, Google Colaboratory, Binder, IDE-based notebook editors.
C8	Link to developer documentation/manual	<a href="https://github.com/ContextLab/davos#readme">https://github.com/ContextLab/davos#readme</a>
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

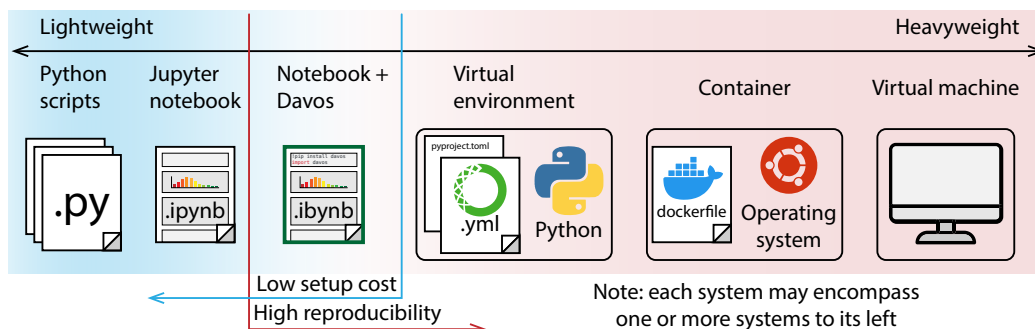


Figure 1: **Systems for sharing code within the Python ecosystem.** From left to right: plain-text **Python scripts** (`.py` files) provide the most basic “system” for sharing raw code. Scripts may reference external libraries, but those libraries must be manually installed on other users’ systems. Further, 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 (`.yaml`) 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)  
62 and 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  
64 of this functionality require (often substantially) different implementation  
65 approaches depending on various properties of the notebook environment  
66 in which **davos** is used (e.g., whether the frontend is provided by Jupyter  
67 or Google Colaboratory, or which version of IPython [11] is used by the  
68 notebook kernel). To deal with this, environment-dependent parts of core  
69 features and behaviors are isolated and abstracted to “helper functions” in  
70 the **davos.implementations** subpackage. This second subpackage defines  
71 multiple, interchangeable versions of each helper function, organized into  
72 modules by the conditions that trigger their use. At runtime, **davos** detects  
73 various features in the notebook environment and selectively imports a single  
74 version of each helper function into the top-level **davos.implementations**  
75 namespace, allowing **davos.core** modules to access the correct implementa-  
76 tions for the current notebook environment from a single, constant location.  
77 An additional benefit of this design pattern is that it allows maintainers or  
78 contributors to extend **davos** to support new, updated, or custom notebook  
79 variants simply by creating a new **davos.implementations** module with any  
80 necessary tweaks to existing helper functions.

### 81 2.2. Software functionalities

#### 82 2.2.1. The *smuggle* statement

83 Importing **davos** in a Jupyter notebook enables an additional Python  
84 keyword: “**smuggle**” (also see Section 2.3). The **smuggle** statement can be  
85 used as a drop-in replacement for Python’s built-in **import** statement to load  
86 libraries, modules, and other objects into the current namespace. However,  
87 whereas **import** will fail if the requested package is not installed locally,  
88 **smuggle** statements can handle missing packages on the fly. If a smuggled  
89 package does not exist in the local environment, **davos** will install it, expose

90 its contents to Python’s `import` machinery, and load it into the namespace  
91 for immediate use.

### 92 2.2.2. The onion comment

93 For greater control over the behavior of `smuggle` statements, `davos` de-  
94 fines an additional construct called the “onion comment”. An onion comment  
95 is a special type of inline comment that may be placed on a line containing a  
96 `smuggle` statement to customize how `davos` searches for the smuggled pack-  
97 age locally and, if necessary, downloads and installs it. Onion comments  
98 follow a simple syntax based on the “type comment” syntax introduced in  
99 PEP 484 [12], and are designed to make managing packages with `davos` intu-  
100 itive and familiar. To construct an onion comment, simply provide the name  
101 of the installer program (e.g., `pip`) and the same arguments one would use  
102 to manually install the package as desired via the command line (see Fig. 2,  
103 lines 3–10 for examples).

104 Onion comments are useful when smuggling a package whose distribution  
105 name (i.e., the name used when installing it) is different from its top-level  
106 module name (i.e., the name used when importing it; e.g., Fig. 2, lines 12–13).  
107 However, the most powerful use of the onion comment is making `smuggle`  
108 statements *version-sensitive*. If an onion comment includes a version specifier  
109 [13] (e.g., Fig. 2, lines 15–19), `davos` will ensure that the version of the  
110 package loaded into the notebook matches the specific version requested, or  
111 satisfies the given version constraints. If the smuggled package exists locally,  
112 `davos` will extract its version info from its metadata and compare it to the  
113 specifier provided. If the two are incompatible (or no local installation is  
114 found), `davos` will install and load a suitable version of the package instead.  
115 Onion comments can similarly be used to smuggle specific VCS references  
116 (e.g., Git [14] branches, commits, tags, etc.; Fig. 2, lines 21–22).

117 `davos` processes onion comments internally before forwarding arguments  
118 to the installer program. In addition to preventing onion comments from  
119 being used as a vehicle for shell injection attacks, this allows `davos` take cer-  
120 tain logical actions when particular arguments are passed (e.g., Fig. 2, lines  
121 24–28). For example, the `--force-reinstall`, `-I/--ignore-installed`,  
122 and `-U/--upgrade` flag will all cause `davos` to skip searching for a smug-  
123 gled package locally before installing a new copy; `--no-input` will disable  
124 `davos`’s input prompts in addition to the installer program’s; and installing  
125 a package into `<dir>` with `--target <dir>` will cause `dir` to be prepended  
126 to the module search path (`sys.path`), if necessary, so the package can be  
127 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.

### 128 2.2.3. The *davos* config

### 129 2.2.4. Additional functionality

### 130 2.3. Implementation details

131 Functionally, importing `davos` appears to define “`smuggle`” as a Python  
132 keyword, similar to “`import`”, “`def`”, or “`return`”. It also appears to cause  
133 comments to be parsed and potentially able to affect the code’s behavior,  
134 which they normally are not. However, `davos` doesn’t actually modify the  
135 rules of Python’s parser or lexical analyzer—in fact, modifying the Python  
136 grammar isn’t possible at runtime, as doing so would require rebuilding the  
137 interpreter. Instead, `davos` leverages the IPython notebook backend to im-  
138 plement the `smuggle` statement and onion comment via a combination of  
139 namespace injections and its own (far simpler) custom parser.

140 The `smuggle` keyword can be enabled and disabled at any time by “ac-  
141 tivating” and “deactivating” `davos` (see Section 2.2.3, above); when `davos`  
142 is first imported, it is activated automatically. Activating `davos` triggers  
143 two things: (1) the `smuggle()` function is injected into the IPython user  
144 namespace, and (2) the `davos` parser is registered as a custom IPython input  
145 transformer. IPython preprocesses all executed code as plain text before it is  
146 sent to the Python parser, in order to handle special constructs like `%magic`  
147 and `!shell` commands. `davos` hooks into this process to transform `smuggle`  
148 statements into syntactically valid Python code. The `davos` parser uses a  
149 complex regular expression [15] to match lines of code containing `smuggle`  
150 statements (and, optionally, onion comments), extract relevant information  
151 from their text, and replace them with equivalent calls to the `smuggle()`  
152 function. For example, if the code visible to the user reads

```
153 smuggle numpy as np      # pip: numpy>1.16,<=1.20 -vv,
```

154 the code that is actually executed by the Python interpreter would be

```
155 smuggle(name="numpy", as_="np", installer="pip", args_str="""numpy>1.16,<=1.20 -  
156 vv""", installer_kwargs={'editable': False, 'spec': 'numpy>1.16,<=1.20', 'verbos
```

### 157 3. Illustrative Examples

### 158 4. Impact

159 Like virtual environments, containers, and virtual machines, the `davos` li-  
160 brary (when used in conjunction with Jupyter notebooks) provides a lightweight  
161 mechanism for sharing code and ensuring reproducibility across users and  
162 computing environments (Fig. 1). Further, `davos` enables users to fully  
163 specify (and install, as needed) any project dependencies within the same  
164 notebook. This provides a system whereby executable code (along with text  
165 and media) *and* code for setting up and configuring the project dependencies,  
166 may be combined within a single notebook file.

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

174 Beyond research applications, `davos` is also useful in pedagogical settings.  
175 For example, in programming courses, instructors and students may import



176 the **davos** library into their notebooks to provide a simple means of ensur-  
177 ing their code will run on others’ machines. When combined with online  
178 notebook-based platforms like Google Colaboratory, **davos** provides a con-  
179 venient way to manage dependencies within a notebook, without requiring  
180 any software (beyond a web browser) to be installed on the students’ or in-  
181 structors’ systems. For the same reasons, **davos** also provides an elegant  
182 means of sharing ready-to-run notebook-based demonstrations that install  
183 their dependencies automatically.

184 Since its initial release, **davos** has found use in a variety of applications. In  
185 addition to managing computing environments for multiple ongoing research  
186 studies, **davos** is being used by both students and instructors in programming  
187 courses such as *Storytelling with Data* [16] (an open course on data science,  
188 visualization, and communication) to simplify distributing lessons and sub-  
189 mitting assignments, as well as in online demos such as **abstract2paper**  
190 [17] (an example application of **GPT-Neo**) to share ready-to-run code that  
191 installs dependencies automatically.

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

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

## 214 5. Conclusions

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

## 222 Author Contributions

223 **Paxton C. Fitzpatrick:** Conceptualization, Methodology, Software,  
224 Validation, Writing - Original Draft, Visualization. **Jeremy R. Manning:**  
225 Conceptualization, Resources, Validation, Writing - Review & Editing, Su-  
226 pervision, Funding acquisition.

## 227 Funding

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

## 231 Declaration of Competing Interest

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

## 235 Acknowledgements

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

## 239 References

- 240 [1] G. van Rossum, Python reference manual, Department of Computer  
241 Science [CS] (R 9525) (1995).
- 242 [2] Python Software Foundation, The Python Package Index (PyPI),  
243 <https://pypi.org> (2003).

- [3] conda-forge community, The conda-forge Project: Community-based Software Distribution Built on the conda Package Format and Ecosystem, <https://doi.org/10.5281/zenodo.4774217> (July 2015). doi: [10.5281/zenodo.4774217](https://doi.org/10.5281/zenodo.4774217).
- [4] 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. Schmidt (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](https://doi.org/10.3233/978-1-61499-649-1-87).
- [5] R. P. Goldberg, Survey of virtual machine research, Computer 7 (6) (1974) 34–45.
- [6] Y. Altintas, C. Brecher, M. Weck, S. Witt, Virtual machine 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).  
URL <https://www.sciencedirect.com/science/article/pii/S0007850607600225>
- [7] M. Rosenblum, VMware’s Virtual Platform: A virtual machine monitor for commodity PCs, in: IEEE Hot Chips Symposium, IEEE, 1999, pp. 185–196.
- [8] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux Journal 239 (2) (2014) 2.
- [9] G. M. Kurtzer, V. Sochat, M. W. Bauer, Singularity: Scientific containers for mobility of compute, PLoS One 12 (5) (2017) e0177459.
- [10] Anaconda, Inc., conda, <https://docs.conda.io> (2012).
- [11] 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](https://doi.org/10.1109/MCSE.2007.53).
- [12] G. van Rossum, J. Lehtosalo, L. Langa, Type Hints, PEP 484, Python Software Foundation (September 2014).  
URL <https://www.python.org/dev/peps/pep-0484>
- [13] N. Coghlan, D. Stufft, Version identification and dependency specification, PEP 440, Python Software Foundation (March 2013).  
URL <https://peps.python.org/pep-0440>

- 279 [14] L. Torvalds, J. Hamano, Git: Fast version control system, [https://](https://git.kernel.org/pub/scm/git/git.git)  
280 [git.kernel.org/pub/scm/git/git.git](https://git.kernel.org/pub/scm/git/git.git) (April 2005).
- 281 [15] K. Thompson, Programming Techniques: Regular expression search al-  
282 gorithm, Communications of the ACM 11 (6) (1968) 419–422. doi:  
283 [10.1145/363347.363387](https://doi.org/10.1145/363347.363387).
- 284 [16] J. R. Manning, Data wrangler, Zenodo 10.5281/zenodo.5123310 (2021).
- 285 [17] J. R. Manning, Episodic memory: mental time travel or a quantum  
286 “memory wave” function?, Psychological Review 128 (4) (2021) 711–  
287 725.