`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 (which specify all Python package versions), to containers (which also specify the operating system), to virtual machines (which also specify hardware layers of the system). 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. This enables researchers to share a complete reproducible environment using 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 $\geq$ 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	`https://github.com/ContextLab/davos#readme`
C9	Support email for questions	contextualdynamics@gmail.com

Table 1: Code metadata

1. Motivation and significance

Code sharing is a core component of the open science movement that has inspired the development of a full ecosystem of tools and packages. However, sharing code, in and of itself, does not guarantee that others will be able to *reproduce* the desired results. For example, research code often requires installing other software packages that extend the implementation language's basic functionality. Within the Python community [1], 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, doing so may cause conflicts within the user's computing environment that interfere with the functionality of *other* code.

To facilitate code sharing, the Python community has developed a broad set of approaches and tools (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 is needed to run standard Jupyter notebooks. And like virtual environments `davos` provides a convenient mechanism for fully specifying (and installing, as needed) a complete set of Python dependencies, including package versions.

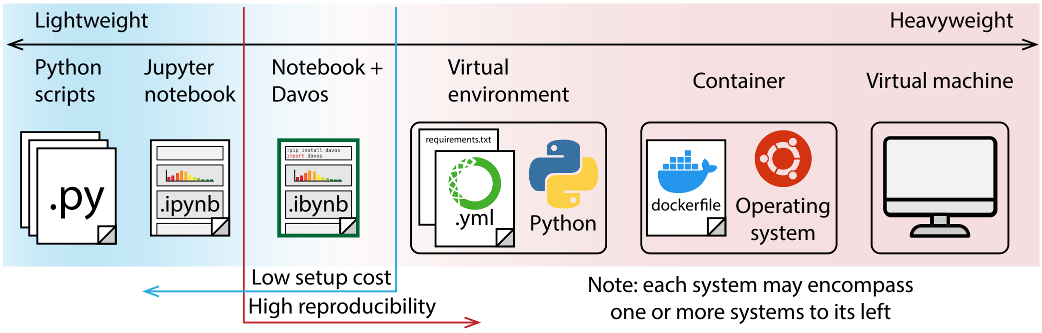

Figure 1: **Systems for sharing code within the Python ecosystem.** From left to right: plain-text **Python scripts** (.py files) provide the most basic "system" for sharing raw code. Scripts may reference external libraries, but those libraries are not automatically installed on other users' systems, nor is any version checking performed by default. **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 that lists all dependencies (including version numbers) along with an environment (.yml) file that specifies how the virtual environment should be configured. **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 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 software. Systems to the right of the red vertical line provide high reproducibility by supporting 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.

2. Software description

2.1. Software architecture

The `davos` package consists of two interdependent subpackages. The first, `davos.core`, comprises a set of modules that implement the bulk of the package's core functionality, including pipelines for installing and validating packages, custom parsers for the `smuggle` statement (see Section 2.2.1) and onion comment (see Section 2.2.2), and a runtime interface for configuring `davos`'s behavior (see Section 2.2.3). However, certain critical aspects of this functionality require (often substantially) different implementation approaches depending on various properties of the notebook environment in which `davos` is used (e.g., whether the frontend is provided by Jupyter or Google Colaboratory, or which version of IPython [11] is used by the notebook kernel). To deal with this, environment-dependent parts of core features and behaviors are isolated and abstracted to "helper functions" in the `davos.implementations` subpackage. This second subpackage defines multiple, interchangeable versions of each helper function, organized into modules by the conditions that trigger their use. At runtime, `davos` detects various features in the notebook environment and selectively imports a single version of each helper function into the top-level `davos.implementations` namespace, allowing `davos.core` modules to access the correct implementations for the current notebook environment from a single, constant location. An additional benefit of this design pattern is that it makes adding support for new or updated notebook variants to `davos` in the future relatively easy.

2.2. Software functionalities

2.2.1. The `smuggle` statement

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

2.2.2. The onion comment

For greater control over the behavior of `smuggle` statements, `davos` defines an additional construct called the "onion comment". An onion comment is a special type of inline comment that may be placed on a line containing a `smuggle` statement to customize how `davos` searches for the smuggled package locally and, if necessary, downloads and installs it. Onion comments follow a simple syntax based on the "type comment" syntax introduced in PEP 484 [10], and are designed to make managing packages with `davos` intuitive and familiar. To construct an onion comment, simply provide the name of the installer program (e.g., `pip`) and the same arguments one would use to manually install the package as desired via the command line (see Fig. 2, lines ___-___ for examples).

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

`davos` processes onion comments internally before forwarding arguments to the installer program. In addition to preventing onion comments from being used as a vehicle for shell injection attacks, this allows `davos` take certain logical actions when particular arguments are passed (e.g., Fig. 2, lines ___-___). For example, `--force-reinstall`, `-I`/`--ignore-installed`, and `-U`/`--upgrade` will all cause `davos` to skip searching for a smuggled package locally before installing a new copy; `--no-input` will disable `davos`'s input prompts in addition to `pip`'s; and installing a package into `<dir>` with `--target <dir>` will cause `dir` to be prepended to `sys.path`, if necessary, so the package can be imported.

2.2.3. The `davos` config

2.2.4. Additional functionality

2.3. Sample code snippets analysis (optional)

3. Illustrative Examples

4. Impact

Like virtual environments, containers, and virtual machines, the `davos` library (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

smuggle dateutil    # pip: python-dateutil
from sklearn.decomposition smuggle PCA    # pip: scikit-learn

# specifically use matplotlib v3.4.2, pip-installing it if needed
smuggle matplotlib.pyplot as plt    # pip: matplotlib==3.4.2

# use a version of seaborn no older than v0.9.1, but before v0.11
smuggle seaborn as sns    # pip: seaborn>=0.9.1,<0.11

# use quail as the package existed on GitHub at commit 6c847a4
smuggle quail    # pip: git+https://github.com/ContextLab/quail.git@6c847a4

# install hypertools v0.7 without first checking for it locally
smuggle hypertools as hyp    # pip: hypertools==0.7 --ignore-installed

# always install the latest version of requests, including pre-releases
from requests smuggle Session    # pip: requests --upgrade --pre
```

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

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

We designed `davos` for use in research applications. For example, in many settings `davos` may be used as a drop-in replacement for more-difficult-to-set-up virtual environments, containers, and/or virtual machines. For researchers, this lowers barriers to sharing code. By eliminating most of the setup costs of reconstructing the original researchers' computing environment, `davos` also lowers barriers to entry for members of the scientific community and the public who seek to *benefit* from shared code.

Beyond research applications, `davos` is also useful in pedagogical settings. For example, in programming courses, instructors and students may import the `davos` library into their notebooks to provide a simple means of ensuring their code will run on others' machines. When combined with online notebook-based platforms like Google Colaboratory, `davos` provides a convenient way to manage dependencies within a notebook, without requiring any software (beyond a web browser) to be installed on the students' or instructors' systems. For the same reasons, `davos` also provides an elegant means of sharing ready-to-run notebook-based demonstrations that install their dependencies automatically.

Our work also has several more subtle "advanced" use cases and potential impacts. Whereas Python's built-in `import` statement is agnostic to packages' version numbers, `smuggle` statements (when combined with onion comments) are version-sensitive. This enables multiple versions of a single library to be imported within the same notebook. This could be useful in cases where specific features were added or removed from a package across different versions, or in comparing the performance or functionality of particular features across different versions of the same package.

A second advanced use case is in providing a proof-of-concept of how one can add new "keywords" to the Python language by leveraging the error-handling mechanisms. This could lead to exciting new tools that, like `davos`, extend the Python language in useful ways. We note that our approach to adding the `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

Conceptualization: PCF and JRM. Methodology: PCF and JRM. Implementation: PCF. Validation: PCF. Testing: PCF and JRM. Writing: PCF and JRM.

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.

References

[1] G. van Rossum, Python reference manual, Department of Computer Science [CS] (R 9525) (1995).

[2] Python Software Foundation, The Python Package Index (PyPI), https://pypi.org (2003).

[3] Anaconda, Inc., conda, https://docs.conda.io (2012).

[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. 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.

[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.
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] G. van Rossum, J. Lehtosalo, Ł. Langa, Type Hints, PEP 484, Python Software Foundation (September 2014).
URL https://www.python.org/dev/peps/pep-0484

[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.

[12] N. Coghlan, D. Stufft, Version identification and dependency specification, PEP 440, Python Software Foundation (March 2013).
URL https://peps.python.org/pep-0440

[13] L. Torvalds, J. Hamano, Git: Fast version control system, https://git.kernel.org/pub/scm/git/git.git (April 2005).