`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

A core requirement of modern scientific research is replicability. For computational research, replicability means that code should produce the same results, even when run on different systems. The standard solution to improving replicability 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 an even lighter-weight solution than virtual environments: the `davos` library. When used in combination with a notebook-based Python project, `davos` library provides a mechanism for specifying (and automatically installing) the correct package versions of the project's. 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 & 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

Code sharing is a core component of the open science movement that has inspired 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 extends the 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 interferes 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 the users. Each of these systems (Python scripts and notebooks) provides a convenient means of sharing code, with the caveat that they do not specify a complete computing environment. 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 complete hardware-level simulation of the desired system. Virtual machines are typically fully 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 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 a complete copy of the original operating system. Each of these systems (virtual machines, containers, and virtual environments) guarantees (to differing degrees– at the hardware-level, operating system level, and Python environment, respectively) that the relevant code will run similarly for different users. However, each of these systems also requires installing independent tools 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 original. 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.

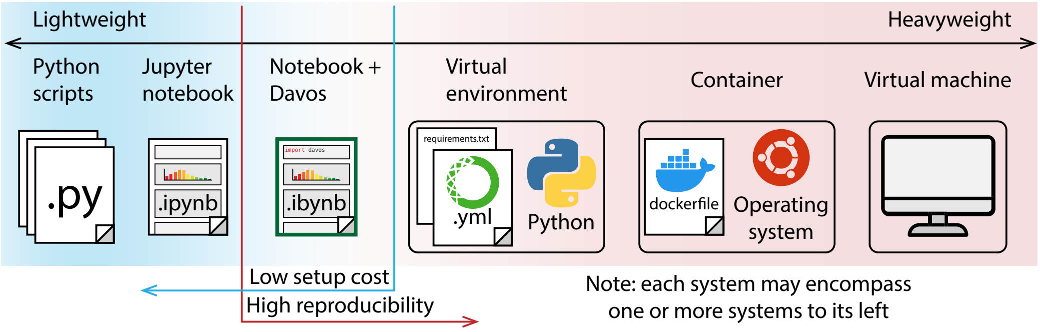

Figure 1: **Systems for sharing code within the Python ecosystem.** From left to right: plain-text Python scripts (.py) provide the most basic system for sharing raw code. Scripts may reference external libraries, but those libraries are not automatically installed on other users' systems, nor is any 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 using 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.

2. Software description

2.1. Software architecture

The `davos` package is structured as two sub-packages: a set of "core" modules that implement...

2.2. Software functionalities

2.2.1. The `smuggle` statement

Importing `davos` 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, how it should be installed. Onion comments follow a simple syntax based on the "type comment" syntax introduced in PEP 484 [10] and are designed to make managing packages via `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 install the package as desired manually via the command line (see Fig. 2).

```
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...**

*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 notebook) provides a lightweight mechanism for sharing code and ensuring reproducibility across users and computing environments. 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, all within a single notebook file.

We designed `davos` for use in research applications. For example, in many cases `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. And by eliminating most of the setup costs of reconstructing the researchers' computing environment, `davos` also serves to lower barriers to entry for members of the scientific community and of the public to *benefiting* from shared code.

Beyond research applications, `davos` is also useful in pedagogical settings. For example, in programming courses, instructors and students may choose to 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 means of managing 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.

In addition to the above common uses of `davos`, our work also provides several more subtle "advanced" use cases. Whereas Python's built-in `import` statement is agnostic to packages' version numbers, `smuggle` statements (when combined with onion comments) are version-sensitive. This setup 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 built into Jupyter notebooks. 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 carries some potential risk. For example, a malicious user could use a similar approach to substantially change a notebook’s functionality, in difficult-to-predict ways when a particular library was imported into the current environment. 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

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 package index - pypi.
URL https://pypi.org/

[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] I. VMware, R. Calculator, Vmware (2018).

[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 (September 2014).
URL https://www.python.org/dev/peps/pep-0484