

nbinteract: Generate Interactive Web Pages From Jupyter Notebooks

Samuel Lau

April 26, 2018

`nbinteract` provides a Python library and a command-line tool to convert Jupyter notebooks to standalone, interactive HTML web pages. These web pages may be viewed by any web browser running JavaScript, regardless of whether the viewer has Python or Jupyter installed locally. `nbinteract`'s built-in support for function-driven plotting makes interactive visualizations simpler to create by allowing authors to focus on declarative data changes instead of callbacks. `nbinteract` has use cases for data analysis, visualization, and especially education, where it is used for a prominent course textbook.

Introduction

Jupyter notebooks provide a popular document format for authoring, executing, and publishing code alongside analysis [13]. Although Jupyter notebooks were originally designed for use in scientific workflows for data preparation and analysis, they are becoming an increasingly common choice for university courses—a survey in 2016 reported that over one hundred courses across multiple countries use Jupyter in their course content [7].

An increasing number of universities now offer data science courses, many of which use Jupyter because of its broad adoption for data analysis workflows in both academia and industry. These courses often use Jupyter notebooks as the preferred medium for homeworks, labs, projects, and lectures. UC Berkeley's flagship data science courses serve thousands of students every year and use Jupyter for all of these course components.

As a web technology, Jupyter notebooks also provide a platform for interaction authoring. For example, the popular `ipywidgets` Python library allows users to create web-based user interfaces to interact with arbitrary Python functions. Users can create these interfaces using Python directly in the notebook environment instead of having to use HTML and JavaScript, significantly lowering the time usually needed to create these interfaces [9]. This ease-of-use encourages instructors and researchers to create interactive explanations of their work.

Unfortunately, it is difficult to share these interactive notebooks with the public. Sharing the notebook file itself retains full interactivity but requires viewers to have Jupyter, Python, and all other packages used in the notebook installed on their own machines. The

freely available Binder service circumvents this by hosting notebook servers that come pre-packaged with necessary software. However, both of these options still require viewers to have prior familiarity with the Jupyter environment, making them less suitable for use with non-technical viewers. Authors can convert a Jupyter notebook to a static HTML document and host the document as a publicly-accessible web page. However, this method does not preserve the interactive elements of the notebook; the resulting web page only contains text and images.

`nbinteract` is a Python package that allows authors to convert Jupyter notebooks into interactive, standalone HTML pages. The interactive elements can use arbitrary Python code to generate output, including Python libraries that use C extensions (e.g. `numpy` and `pandas`) and libraries that create images (e.g. `matplotlib`). The resulting web pages can be used by anyone with a modern web browser even if the viewer does not have Python or Jupyter installed on their computer. The `nbinteract` package also includes specialized methods for interactive plots designed for fast interaction prototyping in the notebook and smooth interaction on static HTML web pages. We discuss the design of the package, its features and limitations, and its implications for interaction authoring and sharing.

Related Work

Jupyter Technologies

The Jupyter notebook platform provides an environment to author code, images, and written explanations together in a single document composed of multiple cells. The platform is composed of two main components. It includes a frontend—a web-based authoring environment that users open in their web browsers. The frontend connects to a Jupyter kernel, a process on the users’ computers that runs code and returns the output to the frontend to display [13].

The `ipywidgets` library makes use of Jupyter’s web-based frontend to create interactive elements directly in the notebook. The library includes Python functions that produce HTML and JavaScript to display interactive widgets. When a user interacts with a widget—selecting an option from a dropdown menu, for example—the `ipywidgets` library executes user-defined Python functions on the Jupyter kernel and renders the result in the cell [9]. A number of other specialized libraries are built on top of `ipywidgets`, such as the interactive plotting library `bqplot` [4] and the molecular visualization library `nglview` [1].

Jupyter notebooks use the `nbconvert` tool to convert between note-

book formats. nbconvert also allows notebooks to be converted to static HTML pages [8]. However, these pages do not retain widget functionality because they do not have access to a Jupyter kernel by default.

The Binder project hosts ephemeral Jupyter notebook servers as a free service for the general public. It takes a repository of Jupyter notebooks, starts a Jupyter frontend and Jupyter kernel, and gives users the ability to run the notebook over the internet instead of having on their local machines [2].

Interaction Authoring in JavaScript

JavaScript is the most commonly used language to design interactions that run in a web browser. Because most modern web browsers run JavaScript natively, viewers do not have to install additional software in order to make use of these interactive elements, a key advantage of the language. A number of authors use JavaScript to create interactive articles [6, 10] and textbooks [12].

There are a number of JavaScript libraries that provide higher level abstractions for interaction creation such as D3 and Tangle [3, 5]. Fundamentally, most JavaScript libraries require fluency with aspects of web programming such as JavaScript syntax and the document-object model. This additional requirement makes JavaScript more difficult to use for many data scientists; most data science analysis uses Python and R rather than JavaScript.

The Vega project provides a promising alternative to directly using Javascript for interaction data visualizations. By defining a grammar of visualization and interaction, Vega and its ecosystem of projects allow users to declaratively generate plots that support filtering, panning and zooming. Since Vega prespecifies available interaction types, however, it does not allow arbitrary user-defined code to run in response to interaction [11].

Features

Installation

Installing nbinteract requires Python version 3.4 or higher. To install nbinteract, run the following command in the terminal:

```
pip install nbinteract
```

If the Jupyter notebook package version is lower than 5.3, run these two additional commands to enable nbinteract in the notebook environment:

```
jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

```
jupyter nbextension enable --py --sys-prefix bqplot
```

After installation, the nbinteract package is available to import in a Python program and a nbinteract command-line tool is available to use on the terminal.

Preparing Notebooks for nbinteract

The simplest method to prepare notebooks for conversion using nbinteract is to place the notebooks in a GitHub repository with a `requirements.txt` file in the root directory. The `requirements.txt` file should contain all packages required to run the notebooks. These steps will prepare the GitHub repository for use on the Binder service, a prerequisite for nbinteract. For additional configuration options, consult the Binder documentation¹.

¹ <http://bit.ly/binder-docs>

Command-line API

nbinteract provides a command line tool to convert Jupyter notebook files to HTML files. It requires that a GitHub repository with the notebooks is set up for use with the Binder service. To convert a notebook to HTML, run the following command in a terminal shell:

```
nbinteract {owner}/{repo}/{branch} {notebook_name}
```

Where `{owner}`, `{repo}`, `{branch}`, and `{notebook_name}` are replaced with the repository's owner, repository name, branch containing the files, and the name of the notebook to convert. For example, to convert a notebook called `hello.ipynb` residing on the default master branch of `https://github.com/SamLau95/nbinteract`, run:

```
nbinteract SamLau95/nbinteract/master hello.ipynb
```

This command creates a `hello.html` file using the original `hello.ipynb` notebook. The resulting HTML file may be uploaded to the World Wide Web using any hosting service, including the free GitHub Pages service².

² <https://pages.github.com/>

Python API for Notebook Conversion

As a convenience, nbinteract also provides a Python interface to convert notebooks to HTML files. To use Python to convert the `hello.ipynb` notebook mentioned above, run:

```
import nbinteract as nbi
nbi.publish('SamLau95/nbinteract-image/master', 'hello.ipynb')
```

This Python code performs the same conversion as the shell command above.

Python API for Interactive Plotting

The nbinteract Python package provides a set of plotting methods for generating visualizations controlled by interactive widgets. While most plotting methods in other visualization libraries (e.g. matplotlib) take data as input, the plotting methods in nbinteract take in functions that generate data as input. For example, the following code generates an interactive histogram where the user can change the mean and spread of a normal distribution:

```
import numpy as np
import nbinteract as nbi

def normal(mean, sd):
    '''Returns 1000 points drawn at random from N(mean, sd)'''
    return np.random.normal(mean, sd, 1000)

# Pass in the 'normal' function and let user change mean and sd.
# Whenever the user interacts with the sliders, the
# 'normal' function is called and the returned data are plotted.
nbi.hist(normal, mean=(0, 10), sd=(0, 2.0), options=options)
```

The plotting methods in nbinteract take in functions as input and use the function signature to generate widgets placed above the resulting visualization³.

³ The complete API is documented on nbinteract's website: <http://nbinteract.com/>

Implementation

Interactivity for Generated HTML Pages

Using the base nbconvert library to convert notebooks to HTML results in a static HTML page that includes code, text, and images. If the notebook uses ipywidgets library to generate widgets, the HTML page also renders static widgets. Although these widgets respond to user interaction, since the page does not have access to a Jupyter kernel the widgets will not generate new output⁴.

When a notebook is converted to HTML using nbinteract, the library replaces all static widgets with “Run Widget” buttons and embeds an additional JavaScript library in the page. When a “Run Widget” is pressed, the JavaScript library starts a Jupyter kernel using the publicly available Binder service. Once a kernel is available, the JavaScript library runs the code on the page and renders live widgets for each cell that originally generated widgets. The library also handles future communication between the widgets on HTML page and the kernel so that interacting with the widgets also updates the output in the HTML. Connecting to a live Jupyter kernel using

⁴ For example, [the ipywidgets documentation](#) has widgets embedded in the page that are detached from their original Python output.

the Binder service allows the static HTML page generated from a notebook to contain dynamic elements.

The JavaScript library for nbinteract called `nbinteract-core` is publicly available on the JavaScript package registry⁵ but is neither designed for general use nor documented on the nbinteract website. `nbinteract-core` combines three existing APIs in order to enable widget rendering for static HTML pages: the Binder Web API, the JupyterLab Services JavaScript API, and the `ipywidgets` JavaScript API.

⁵ <https://www.npmjs.com/>

The Binder Web API allows other programs to start Jupyter servers on the Binder service. Although it is undocumented, the code repository contains an example of API usage⁶ which we use to implement `nbinteract-core`.

⁶ <http://bit.ly/binder-api>

The JupyterLab Services API contains methods to start and manage Jupyter kernels on a Jupyter server⁷. After starting a Jupyter server using the Binder API, `nbinteract-core` uses the JupyterLab Services API to start a Jupyter kernel in order to run Python code on the HTML page.

⁷ <https://www.npmjs.com/package/@jupyterlab/services>

When running code that generates widgets, the `nbinteract-core` library uses the `ipywidgets` JavaScript API for rendering the widgets onto the page⁸ and sets up the necessary connection between widgets and kernel so that future interactions with the widgets will generate new output.

⁸ <https://github.com/jupyter-widgets/ipywidgets>

Interactive Plotting Implementation

`nbinteract` combines the `ipywidgets` widget library and the `bqplot`⁹ plotting library to implement function-driven interfaces to interactive plotting. The `nbinteract` plotting methods use `ipywidgets` to generate and display widgets, inferring the widget type as needed. When a user interacts with a widget, a Python callback updates the visualization without a complete rerender. This noticeably lowers visualization update time compared to using `ipywidgets` alone to render static images.

⁹ <https://github.com/bloomberg/bqplot>

Discussion

Use Cases

Because `nbinteract` is designed for use with the Python programming language in Jupyter notebooks, it provides the most utility for users with prior familiarity with Python and Jupyter. These users include course staff for computer science or data science courses, students in these courses, and online blog authors that use Jupyter notebooks for written content. `nbinteract` can also be used to cre-

ate dashboards from Jupyter notebooks by hiding the code used to generate widgets. Since Jupyter notebooks converted to HTML are an increasingly popular format for online content¹⁰, we've designed nbinteract to easily fit into existing notebook publishing workflows.

UC Berkeley's upper division data science course Data 100 uses nbinteract to build its online textbook¹¹. The textbook is written using Jupyter notebooks and receives a few hundred views per day. nbinteract allows the authors to include interactive widgets anywhere in the textbook. For example, the book uses widgets to display complete views of large data tables that would normally require truncation by allowing viewers to scrub through the rows and columns of the data table. The book also allows viewers to interactively change parameters and data of statistical models and displays updated visualizations in response.

Comparison with JavaScript

Compared to JavaScript, nbinteract gives authors lower flexibility and fidelity in exchange for easier interaction creation.

Since web browsers run JavaScript natively, JavaScript users have almost complete control over every element of the interaction, including the visual appearance of interactive elements and animations. nbinteract, however, limits interactions to the widgets supported by the ipywidgets and bqplot libraries.

Compared to nbinteract, JavaScript-based interactions will typically have a lower startup time and a lower latency between user interaction and visual change. In order to display an interactive element for the first time, nbinteract requests a Jupyter server from the Binder service. This initial request adds 5-10 seconds to the initial startup time. On user interaction, nbinteract runs Python code on a remote Jupyter kernel, incurring overhead from network latency in addition to the time it takes the kernel to run the Python code. Typically, interactions can be structured in a way to minimize code execution time when widgets are manipulated, making network latency the most significant overhead for user interaction. On fast connections, this latency is typically around 100 milliseconds. JavaScript-based interactions, however, typically do not make requests to a remote server and thus do not incur network latency overhead.

Interactions that both nbinteract and JavaScript support are typically easier to create in nbinteract than in JavaScript. Authors fluent in Python can often write interactions with an order-of-magnitude fewer lines of code in nbinteract than authors fluent in JavaScript. The simplicity of nbinteract makes it attractive for authors already knowledgeable in Python but not JavaScript and HTML.

¹⁰ The first detection of gravitational waves and Peter Norvig's pytudes are recent high-profile examples of notebooks used as online content formats.

¹¹ <https://www.textbook.ds100.org/>

Future Work

Although `nbinteract` is not designed to fully replicate the flexibility and fidelity of JavaScript, we hope to make several improvements to the library in order to increase utility for authors that use Jupyter notebooks.

We plan to release a more flexible API for creating interactive visualizations using `nbinteract`. Currently, visualizations are restricted to single plots that contain one type of mark (e.g. line, point, or bar). Our future API will provide a declarative syntax for connecting multiple widgets, functions, and plot elements to create more complex interactive visualizations.

Currently, `nbinteract` only permits widgets from the `ipywidgets` and `bqplot` libraries. Since there are now many libraries built on top of the `ipywidgets` library¹², we plan to allow authors to specify additional widget libraries when converting notebooks. This improvement will allow `nbinteract` to support the entire ecosystem of interaction frameworks that use `ipywidgets`.

¹² Any of the widget libraries listed at <http://jupyter.org/widgets>, for example.

We also plan to work with the Binder team to decrease the startup time of Jupyter servers on Binder. This improvement would benefit both `nbinteract` users and any user of the Binder service itself. We are also investigating ways to reduce latency after widget initialization, including caching previous interaction outputs on the client and batching visualization updates together.

Conclusion

`nbinteract` combines recently developed projects in the Jupyter ecosystem to allow authors to create interactive explanations and visualizations directly in the Jupyter notebook environment. The library aims to use the broad adoption of Python and Jupyter to allow many more individuals to create and share interactive content online.

References

- [1] Arose/Nglview: IPython Widget to Interactively View Molecular Structures and Trajectories. <https://github.com/arose/nglview>.
- [2] Binder (Beta). <https://mybinder.org/>.
- [3] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. “Data-Driven Documents”. In: *IEEE transactions on visualization and computer graphics* 17.12 (2011), pp. 2301–2309.
- [4] Bqplot: Plotting Library for IPython/Jupyter Notebooks. Apr. 2018.
- [5] bret. Tangle: A JavaScript Library for Reactive Documents. Apr. 2018.
- [6] Explorable Explanations. <http://explorables/>.
- [7] Jessica B. Hamrick and Jupyter Development Team. 2016 Jupyter Education Survey. May 2016. DOI: [10.5281/zenodo.51701](https://doi.org/10.5281/zenodo.51701).
- [8] Jupyter/Nbconvert: Jupyter Notebook Conversion. <https://github.com/jupyter/nbconvert>.
- [9] Jupyter-Widgets/Ipywidgets: Interactive Widgets for the Jupyter Notebook. <https://github.com/jupyter-widgets/ipywidgets>.
- [10] Daniel Kunin. *Seeing Theory*. en. <http://seeingtheory.io>.
- [11] Arvind Satyanarayan et al. “Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization”. In: *IEEE transactions on visualization and computer graphics* 22.1 (2016), pp. 659–668.
- [12] Philip B. Stark. “SticiGui: Statistics Tools for Internet and Classroom Instruction with a Graphic User Interface”. In: *Available at Web site: http://stat-www.berkeley.edu/users/stark/SticiGui/index.htm (8 August 2004)* (2004).
- [13] Kluyver Thomas et al. “Jupyter Notebooks—a Publishing Format for Reproducible Computational Workflows”. In: *Stand Alone* (2016), pp. 87–90. ISSN: 0000-0000. DOI: [10.3233/978-1-61499-649-1-87](https://doi.org/10.3233/978-1-61499-649-1-87).