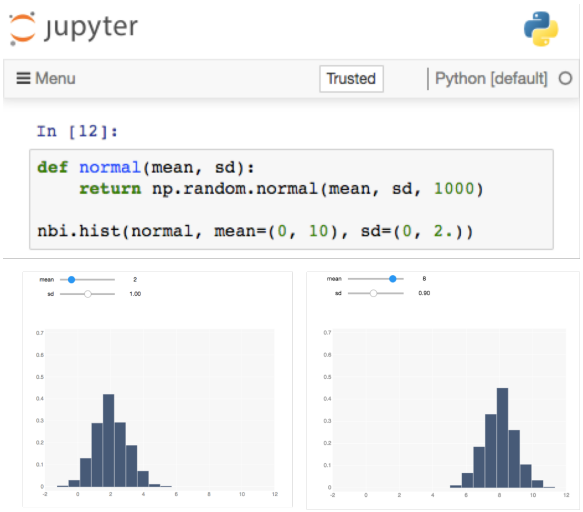
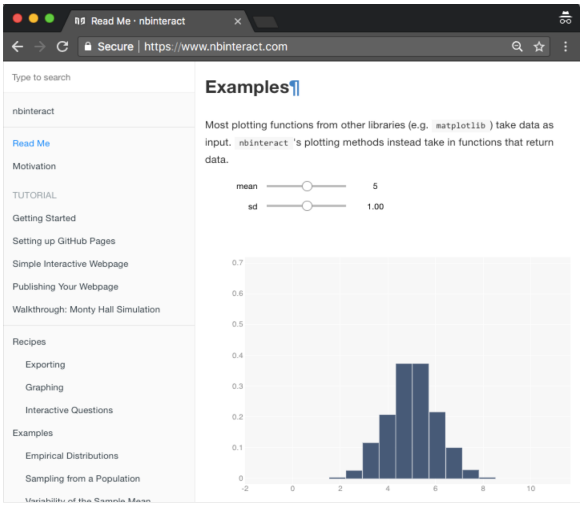
nbinteract: Generate Interactive Web Pages From Jupyter Notebooks

Samuel Lau

May 8, 2018



Succinctly create interactions in Jupyter



Publish standalone, interactive webpages

`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 textbook on data science.

Introduction

Jupyter notebooks provide a popular document format for authoring, executing, and publishing code alongside analysis [14]. 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

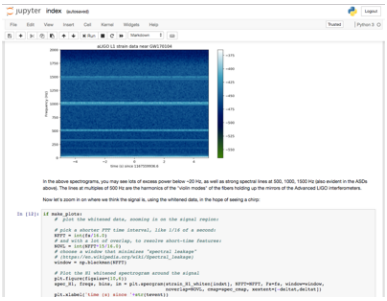


Figure 1: Jupyter notebooks combine code, text, and plots in a single document.

workflows in both academia and industry. These courses often use Jupyter notebooks as the preferred medium for homeworks, labs, projects, and lectures. As a prominent example, UC Berkeley's flagship data science courses serve thousands of students every year and use Jupyter for all of their 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 typically needed to create these interfaces [8]. This ease-of-use encourages instructors and researchers to create interactive explanations of their work.

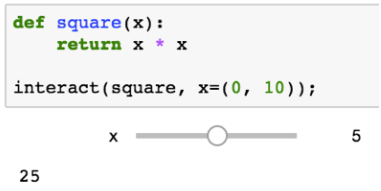


Figure 2: The `ipywidgets` library provides primitives for interaction in Jupyter notebooks.

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, standlone 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 package's features and design, its advantages and limitations compared to JavaScript, 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 [14].

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 [8]. 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 notebook formats. `nbconvert` also allows notebooks to be converted to static HTML pages [9]. 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].

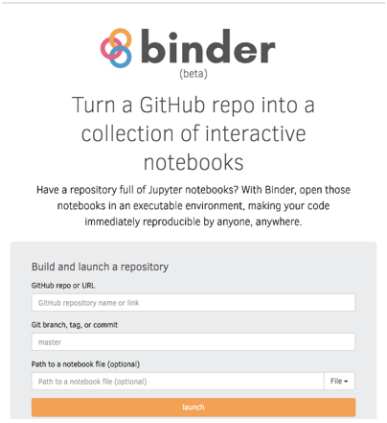


Figure 3: The free `Binder` service runs Jupyter servers for public use.

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].

A number of JavaScript libraries provide higher level abstractions for interaction creation, including 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 [13].

The Vega project provides a promising alternative to directly using Javascript for interaction data visualizations. By defining a grammar of visualization and interaction using JavaScript Object Notation, 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

In this section, we describe the installation steps and application programming interface (API) for `nbinteract`.

Installation

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

```
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 in a terminal shell.

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 list 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 documentation[1].

[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 service[2].

[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 shown in figure 4 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 fron 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 visualization. The complete plotting API is documented on `nbinteract`'s website: http://nbinteract.com/.

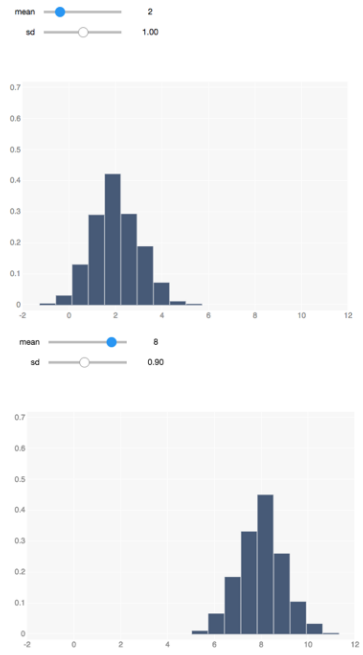


Figure 4: The `nbinteract` plotting functions create visualizations with interactive widgets. Here, two different histogram states are shown.

Future Python API for Complex Plots

As of this writing, `nbinteract`'s visualization methods only generate plots with a single mark (e.g. a line or a histogram). A forthcoming API will enable a declarative one-to-many mapping for widgets to functions and functions to plot marks. Although the these methods are still under testing, the API itself is largely stable. We include an example below to demonstrate plotting two lines that share data from a pair of interactive widgets. The resulting chart is shown in figure 5.

```
import numpy as np
import nbinteract as nbi

def x_intercept1(x_coord): return [x_coord, x_coord]
def x_intercept2(x_coord): return [x_coord + 5, x_coord + 5]
def y_points(y_coord): return [0, y_coord]

nbi.Figure(
    options = {'xlim': (0, 20), 'ylim': (0, 20)},
    widgets={'widget_x': (5, 0, 10), 'widget_y': (5, 0, 10)},
    functions={x_intercept1: ['widget_x'],
               x_intercept2: ['widget_x'],
               y_points:     ['widget_y']},
    marks={'line1': nbi.Line(x_intercept1, y_points),
           'line2': nbi.Line(x_intercept2, y_points)}
)
```

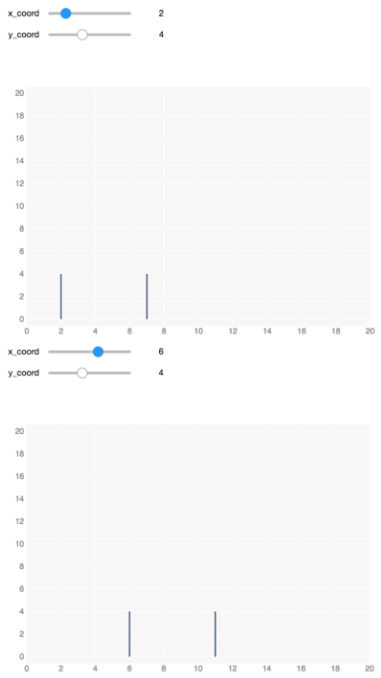


Figure 5: The two line marks in the resulting visualization share data input from the same pair of two widgets.

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 output[3].

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

[3] For example, the `ipywidgets` documentation has widgets embedded in the page that are detached from their original Python output.

kernel using 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 registry[4] 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.

[4] 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 usage[5] which we use to implement `nbinteract-core`.

[5] http://bit.ly/binder-api

The JupyterLab Services API contains methods to start and manage Jupyter kernels on a Jupyter server[6]. 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.

[6] 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 page[7] and sets up the necessary connection between widgets and kernel so that future interactions with the widgets will generate new output.

[7] https://github.com/jupyter-widgets/ipywidgets

Interactive Plotting Implementation

`nbinteract` combines the `ipywidgets` widget library and the `bqplot`[8] 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.

[8] 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 create 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 content[9], `nbinteract` is designed to easily fit into existing notebook publishing workflows.

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

UC Berkeley's upper division data science course Data 100 uses `nbinteract` to build its online textbook[10]. The textbook is written using Jupyter notebooks and receives over a 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 as shown in figure 6. The book also allows viewers to interactively change parameters and data of statistical models and displays updated visualizations in response.

[10] https://www.textbook.ds100.org/

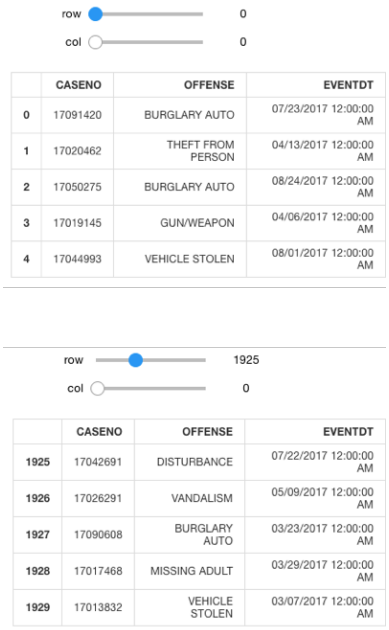


Figure 6: Use of `nbinteract` to embed large tables in the Data 100 Textbook.

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. We are actively implementing methods of reducing both startup and interaction latency—`nbinteract` caches previously-initialized Binder servers on the client and avoid unnecessary network requests when performing visualization updates.

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 knowledgable in Python but not JavaScript and HTML.

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 to increase utility for authors that use Jupyter notebooks.

As previously discussed, 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` library[11], 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`.

[11] Any of the widget libraries listed at `http://jupyter.org/widgets`, for example.

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. "D$\hat{3}$ 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://explorabl.es/.

[7] Jessica B. Hamrick and Jupyter Development Team. *2016 Jupyter Education Survey*. May 2016. DOI: 10.5281/zenodo.51701.

[8] *Jupyter-Widgets/Ipywidgets: Interactive Widgets for the Jupyter Notebook*. https://github.com/jupyter-widgets/ipywidgets.

[9] *Jupyter/Nbconvert: Jupyter Notebook Conversion*. https://github.com/jupyter/nbconvert.

[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] *The Most Popular Language For Machine Learning and Data Science Is ...* https://www.kdnuggets.com/2017/01/most-popular-language-machine-learning-data-science.html.

[14] 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.