



Get unlimited access

Open in app



Published in Plotly



plotly

Follow

Jun 21, 2017 · 12 min read · 🎧 Listen



Save



Introducing Dash



Create Reactive Web Apps in pure Python

Dash is a Open Source Python library for creating reactive, Web-based applications. Dash started as a public proof-of-concept on GitHub 2 years ago. We kept this prototype online, but subsequent work on Dash occurred behind closed doors. We used feedback from private trials at banks, labs, and data science teams to guide the product forward. **Today, we're excited to announce the first public release of Dash that is both enterprise-ready and a first-class member of Plotly's open-source tools.** Dash can be downloaded today from Python's package manager with `pip install dash` — it's entirely open-source and MIT licensed. You'll find a [getting started guide here](#) and the [Dash code on GitHub here](#).

Dash is a user interface library for creating analytical web applications. Those who use Python for data analysis, data exploration, visualization, modelling, instrument control, and reporting will find immediate use for Dash.

Dash makes it dead-simple to build a GUI around your data analysis code. Here's a 43-line example of a Dash App that ties a Dropdown to a D3.js Plotly Graph. As the user selects a value in the Dropdown, the application code dynamically exports data from Google Finance into a Pandas DataFrame. This app was written in just 43 lines of code ([view the source](#)). *Simple.*





Get unlimited access

Open in app



Hello World Dash app. For more examples, check out the [user guide](#).

Dash app code is declarative and reactive, which makes it easy to build complex apps that contain many interactive elements. Here's an example with 5 inputs, 3 outputs, and cross filtering. This app was composed in just 160 lines of code, all of which were Python.





Get unlimited access

Open in app

Dash app with cross filtering, multiple inputs, and multiple outputs. Built in around 163 lines of Python. [View the source](#)

Every aesthetic element of the app is customizable: The sizing, the positioning, the colors, the fonts. Dash apps are built and published in the Web, so the full power of CSS is available. Here's an example of a highly customized, interactive Dash report app, in the brand and style of a Goldman Sachs report.



A highly customized Dash app, styled just like a Goldman Sachs report. [View the source](#).

While Dash apps are viewed in the web browser, you don't have to write any Javascript or HTML. Dash provides a Python interface to a rich set of interactive web-based components.





Get unlimited access

Open in app

```
dcc.Slider(value=4, min=-10, max=20, step=0.5,
           labels={-5: '-5 Degrees', 0: '0', 10: '10 Degrees'})
```





An example of a simple Dash Slider component

Dash provides a simple reactive decorator for binding your custom data analysis code to your Dash user interface.

```
@dash_app.callback(Output('graph-id', 'figure'),
                    [Input('slider-id', 'value')])
def your_data_analysis_function(new_slider_value):
    new_figure = your_compute_figure_function(new_slider_value)
    return new_figure
```

When an input element changes (e.g. when you select an item in the dropdown or drag a slider), Dash's decorator provides your Python code with the new value of the input.

Your Python function can do anything that it wants with this input new value: It could filter a Pandas `DataFrame`, make a SQL query, run a simulation, perform a calculation, or start an experiment. Dash expects that your function will return a new property of some element in the UI, whether that's a new graph, a new table, or a new text element.

For example, here's a simple L  6.8K |  22 | ... as a text box as you interact with the `Graph` element. The application code filters data in a `Pandas DataFrame` based off of the currently selected point.



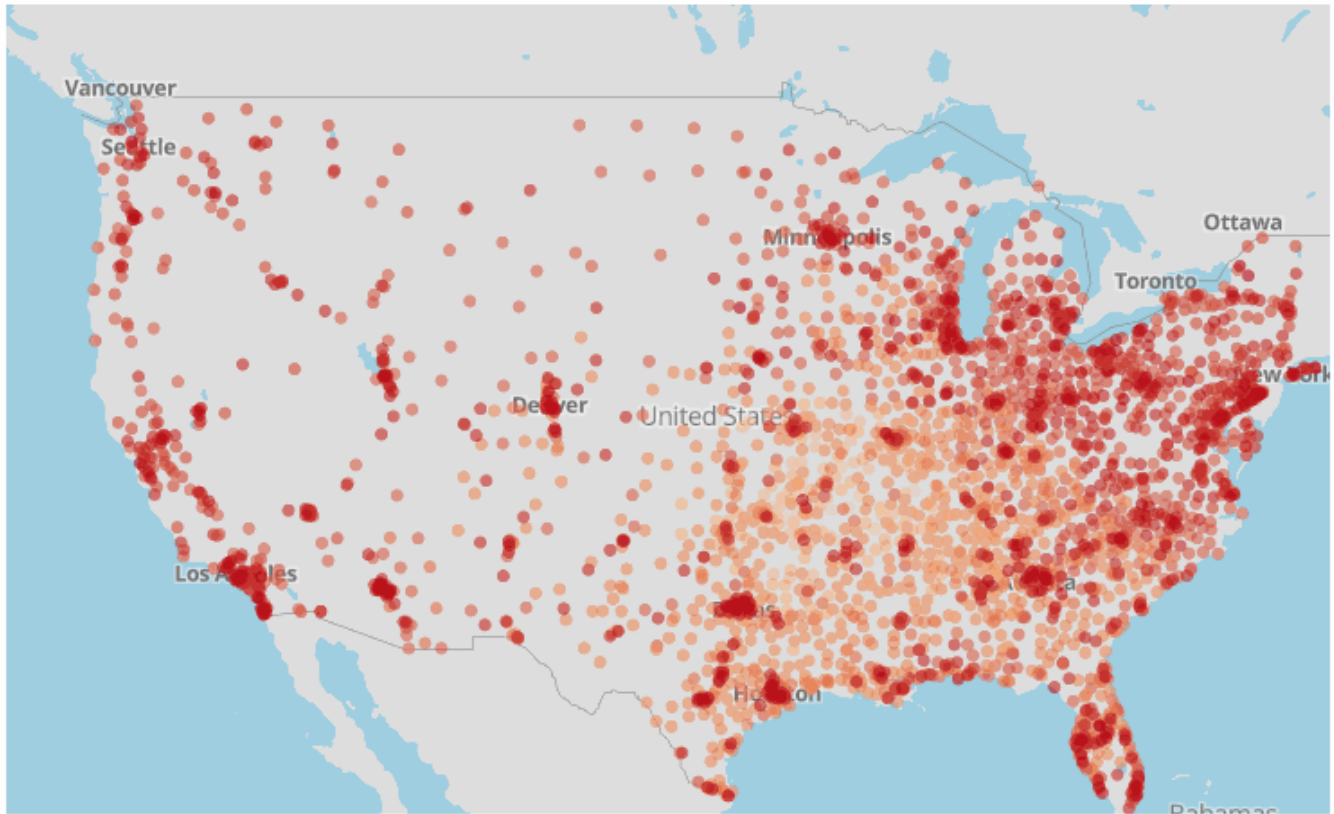


Get unlimited access

Open in app

Walmart Store Openings

The Richmond, MO Supercenter opened in 1980



Dash app that displays custom meta information as you hover over points by filtering a Pandas DataFrame. 60 lines of code. [View the source.](#)

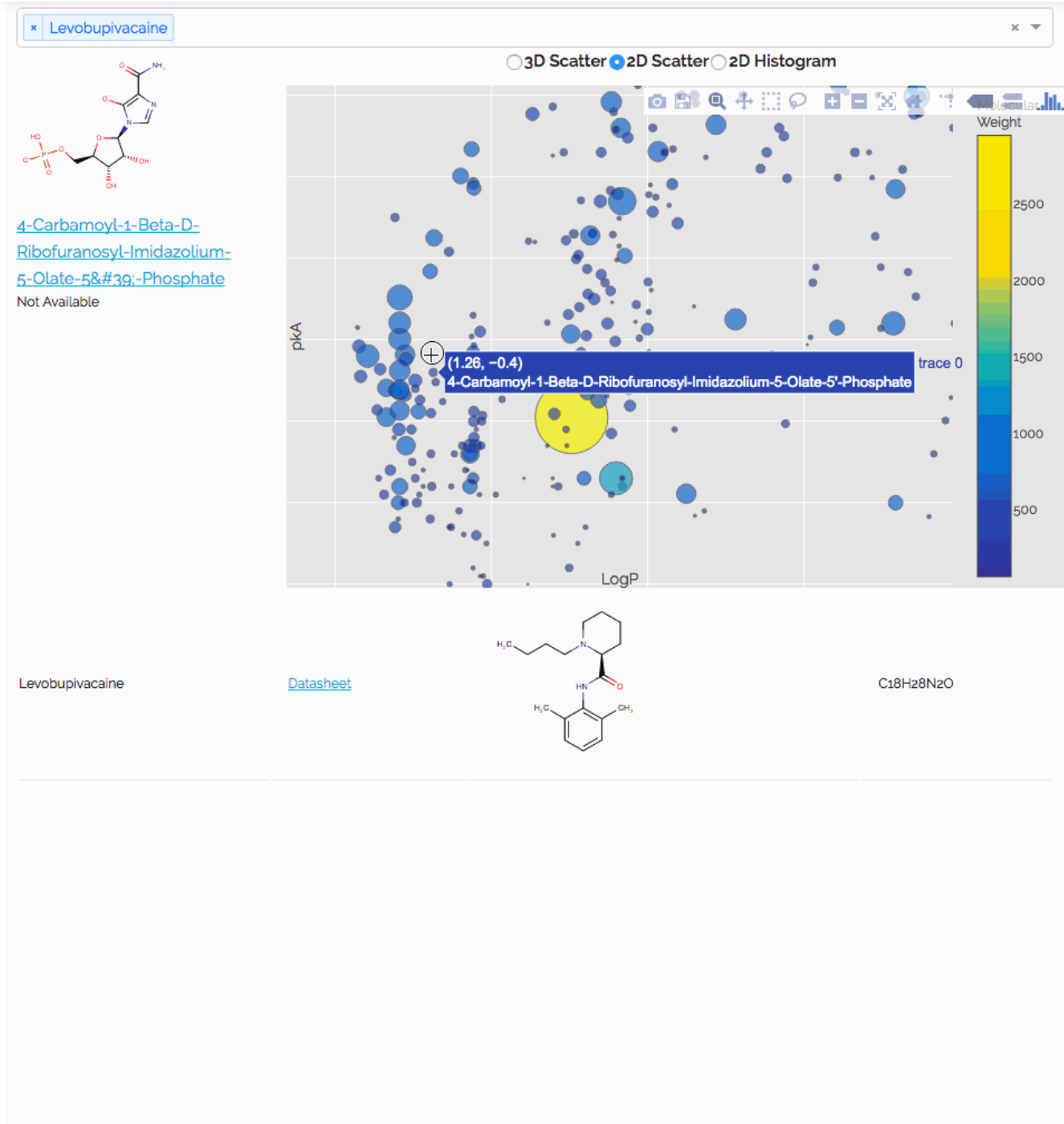
This Dash application displays meta information about drugs as you hover over points in the `Graph` component. The application code also appends rows to the `Table` component when elements are added to the multi `Dropdown` component.





Get unlimited access

Open in app



A Dash App for drug discovery. Hovering over points displays a description of the drug. Selecting drugs in the dropdown highlights their position in the chart and appends their symbol in the table below. Built in a few hundred lines of Python code.

Through these two abstractions — Python components and reactive functional decorators — Dash abstracts away all of the technologies and protocols that are required to build an interactive web-based application. Dash is simple enough that you can bind a user interface around your Python code in an afternoon.





Get unlimited access

Open in app

Dash applications are web servers running [Flask](#) and communicating JSON packets over HTTP requests. Dash's frontend renders components using React.js, the Javascript user-interface library written and maintained by Facebook.

Flask is great. It's widely adopted by the Python community and deployed in production environments everywhere. The underlying instance of Flask and all of its configurable properties is accessible to Dash app developers. For advanced developers, Dash apps can be extended through the rich set of [Flask Plugins](#) as well.

React is fantastic too. At Plotly, we've rewritten our entire web-platform and our [online chart editor](#) with React. One of the incredible things about React is how prolific and talented the community is. The open source React community has published thousands of high quality interactive components, from [Dropdowns](#) to [Sliders](#) to [Calendar Pickers](#) to [Interactive Tables](#).

Dash leverages the power of Flask and React, putting them to work for Python data scientists who may not be expert Web programmers.

From React.js to Python Dash Components

Dash components are Python classes that encode the properties and values of a specific React component and that serialize as JSON. Dash provides a [toolset](#) to easily package React components (written in Javascript) as components that can be used in Dash. This toolset uses dynamic programming to automatically generate standard Python classes from annotated React propTypes. The resulting Python classes that represent Dash components are user friendly: They come with automatic argument validation, docstrings, and more.

Here's an example of the dynamically generated argument validation:

```
>>> import dash_core_components as dcc
>>> dcc.Dropdown(valu=3)
Exception: Unexpected keyword argument `valu`
Allowed arguments: id, className, disabled, multi, options,
placeholder, value
```





Get unlimited access

Open in app

```
>>> help(dcc.DropDown)
class Dropdown(dash.development.base_component.Component)
| A Dropdown component.
| Dropdown is an interactive dropdown element for selecting one or
more
| items.
| The values and labels of the dropdown items are specified in the
`options`
| property and the selected item(s) are specified with the `value`
property.
|
| Use a dropdown when you have many options (more than 5) or when
you are
| constrained for space. Otherwise, you can use RadioItems or a
Checklist,
| which have the benefit of showing the users all of the items at
once.
|
| Keyword arguments:
| - id (string; optional)
| - className (string; optional)
| - disabled (boolean; optional): If true, the option is disabled
| - multi (boolean; optional): If true, the user can select
multiple values
| - options (list; optional)
| - placeholder (string; optional): The grey, default text shown
when no option is selected
| - value (string | list; optional): The value of the input. If
`multi` is false (the default)
| then value is just a string that corresponds to the values
| provided in the `options` property. If `multi` is true, then
| multiple values can be selected at once, and `value` is an
| array of items with values corresponding to those in the
| `options` prop.
|
| Available events: 'change'
```

The full set of HTML tags, like `<div/>`, ``, `<table/>` are also rendered dynamically with React and their Python classes are available through the `dash_html_component` library. A core set of interactive components like `Dropdown`, `Graph`, `Slider` will be maintained by the Dash core team through the `dash_core_components` library. Both of these libraries use the standard open-source React-to-Dash toolchain that you could use if you were to write your own component





Get unlimited access

Open in app

toolchain, you can easily write or port a React.js component into a Python class that can be used in your Dash application. Here's [the tutorial on building your own components](#). Or, the Dash core team can [build one for you](#).

Concurrency — Multi-User Applications

The state of a Dash application is stored in the front-end (i.e. the web browser). This allows Dash apps to be used in a multitenant setting: Multiple users can have independent sessions while interacting with a Dash app at the same time. Dash application code is functional: Your application code can read values from the global Python state but it can't modify them. This functional approach is easy to reason about and easy to test: It's just inputs and outputs with no side-effects or state.

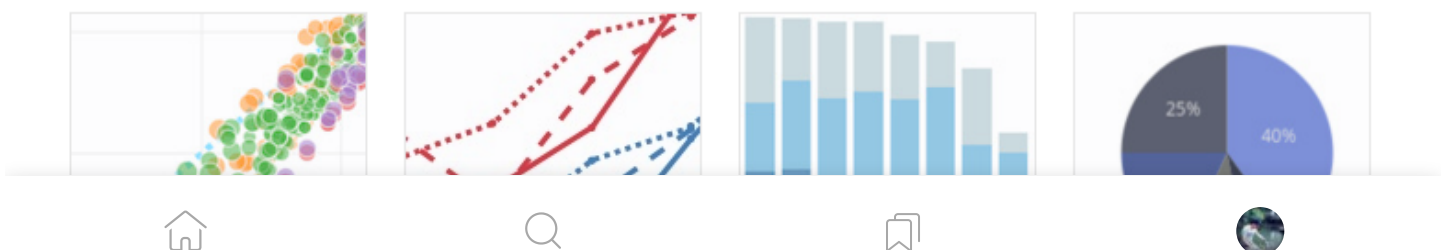
CSS and Default Styles

CSS and default styles are kept out of the core library for modularity, independent versioning, and to encourage Dash App developers to customize the look-and-feel of their apps. The Dash core team maintains a [core style guide here](#).

Data Visualization

Dash ships with a Graph component that renders charts with [plotly.js](#). Plotly.js is a great fit for Dash: it's declarative, open source, fast, and supports a complete range of scientific, financial, and business charts. Plotly.js is built on top of D3.js (for publication-quality, vectorized image export) and WebGL (for high performance visualization).

Dash's Graph element shares the same syntax as the open-source [plotly.py](#) library, so you can easily switch between the two. Dash's Graph component hooks into the plotly.js event system, allowing Dash app authors to write applications that respond to hovering, clicking, or selecting points on a Plotly graph.





Get unlimited access

Open in app



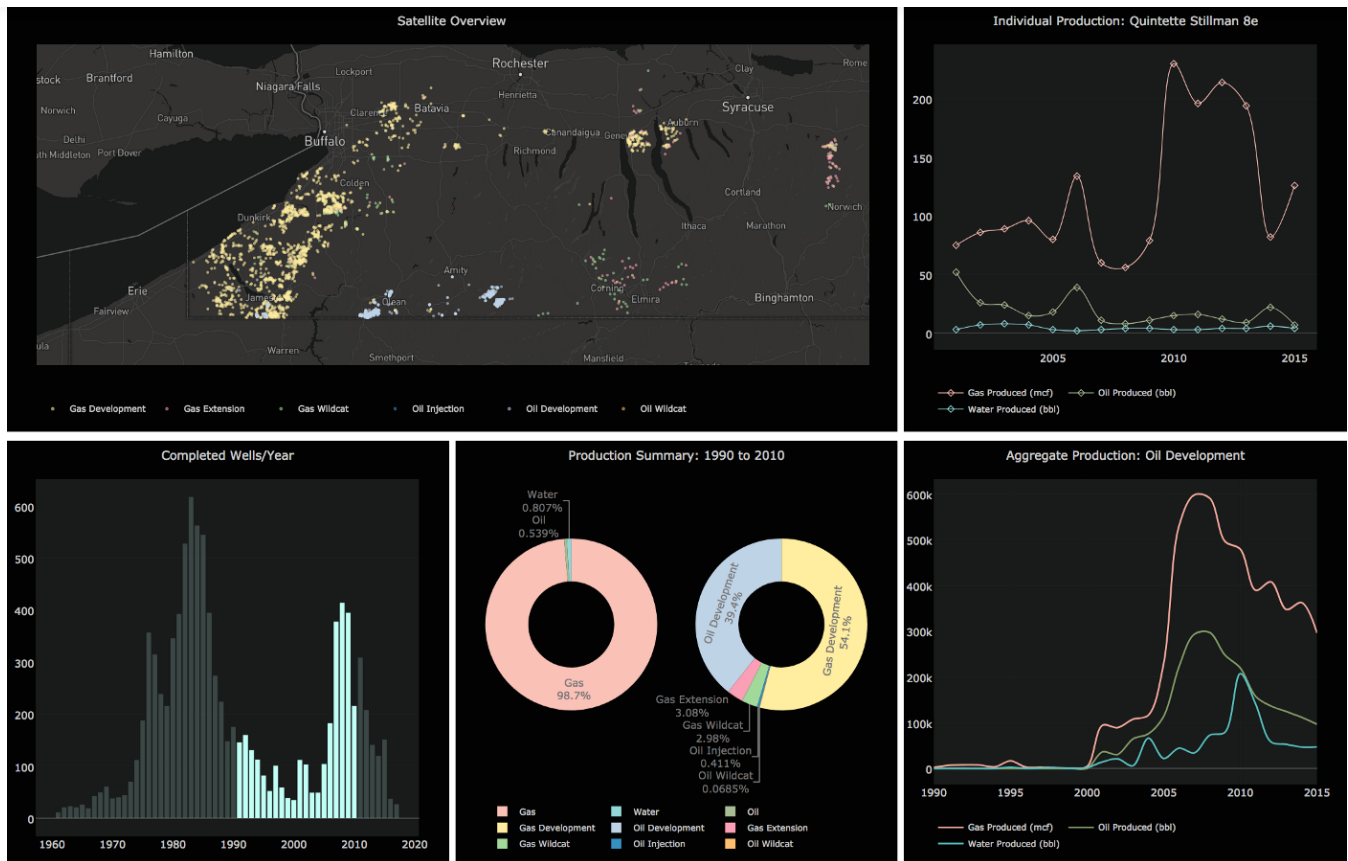
Some of the available chart types in Dash's Plotly.js Graph component. See more in the [plotly.py documentation](#).





Get unlimited access

Open in app



A Dash app with [Plotly.js charts](#) from the [Dash app gallery](#).

Open Source Repositories

You can check out the code yourself across a few repositories:

- Dash backend: <https://github.com/plotly/dash>
- Dash frontend: <https://github.com/plotly/dash-renderer>
- Dash core component library: <https://github.com/plotly/dash-core-components>
- Dash HTML component library: <https://github.com/plotly/dash-html-components>
- Dash component archetype (React-to-Dash toolchain): <https://github.com/plotly/dash-components-archetype>
- Dash docs and user guide: <https://github.com/plotly/dash-docs>, hosted at





Get unlimited access

Open in app

Prior Art

Dash is new in the Python ecosystem but the concepts and motivation behind Dash have existed for decades in a variety of different languages and applications.

If you're coming from **Excel**, then your head is in the right place. Both Dash and Excel use a “reactive” programming model. In Excel, output cells update automatically when input cells change. Any cell can be an output, an input, or both. Input cells aren't aware of which output cells depend on them, making it easy to add new output cells or chain together a series of cells. Here's an example Excel “application”:

Hours per Day		5
Rate per Hour	<input type="text" value="2"/>	
Amount		
Amount per Week		

There's an Excel analogy for Dash. Instead of cells, we have rich web based components like sliders, inputs, dropdowns, and graphs. Instead of writing Excel or VBA script, we're writing Python code. Here is that same spreadsheet application, rewritten in Dash:

```
app.layout = html.Div([
    html.Label('Hours per Day'),
    dcc.Slider(id='hours', value=5, min=0, max=24, step=1),

    html.Label('Rate'),
    dcc.Input(id='rate', value=2, type='number'),

    html.Label('Amount per Day'),
    html.Div(id='amount'),

    html.Label('Amount per Week'),
    html.Div(id='amount-per-week')
])

@app.callback(Output('amount', 'children'),
              [Input('hours', 'value'), Input('rate', 'value')])
def compute_amount(hours, rate):
```





Get unlimited access

Open in app

```
def compute_amount(amount):  
    return float(amount) * 7
```

Hours per Day

Rate

2

Amount per Day

10

Amount per Week

70

I like this example a lot because Excel still reigns supreme, even in technical computing and quantitative finance. I don't think that Excel's dominance is just a matter of technical ability. After all, there are legions of spreadsheet programmers who have learned the nuances of Excel, VBA, and even SQL.

It's more that Excel spreadsheets are frequently *easier to share* than Python programs, and Excel cells are easier to edit than command line arguments.

Yet modelling in Excel has well-known limits: These spreadsheets often outgrow themselves. They become too large or fragile to migrate into a production environment, peer review, test, and maintain. Remember the 2013 pro-austerity Excel typo?

I hope that Dash makes it easier for developers to use Python for their data projects. By sharing the same functional and reactive principles, it's almost as easy to write a Dash app as it is to write an analytical spreadsheet. It's certainly more powerful and presentable.

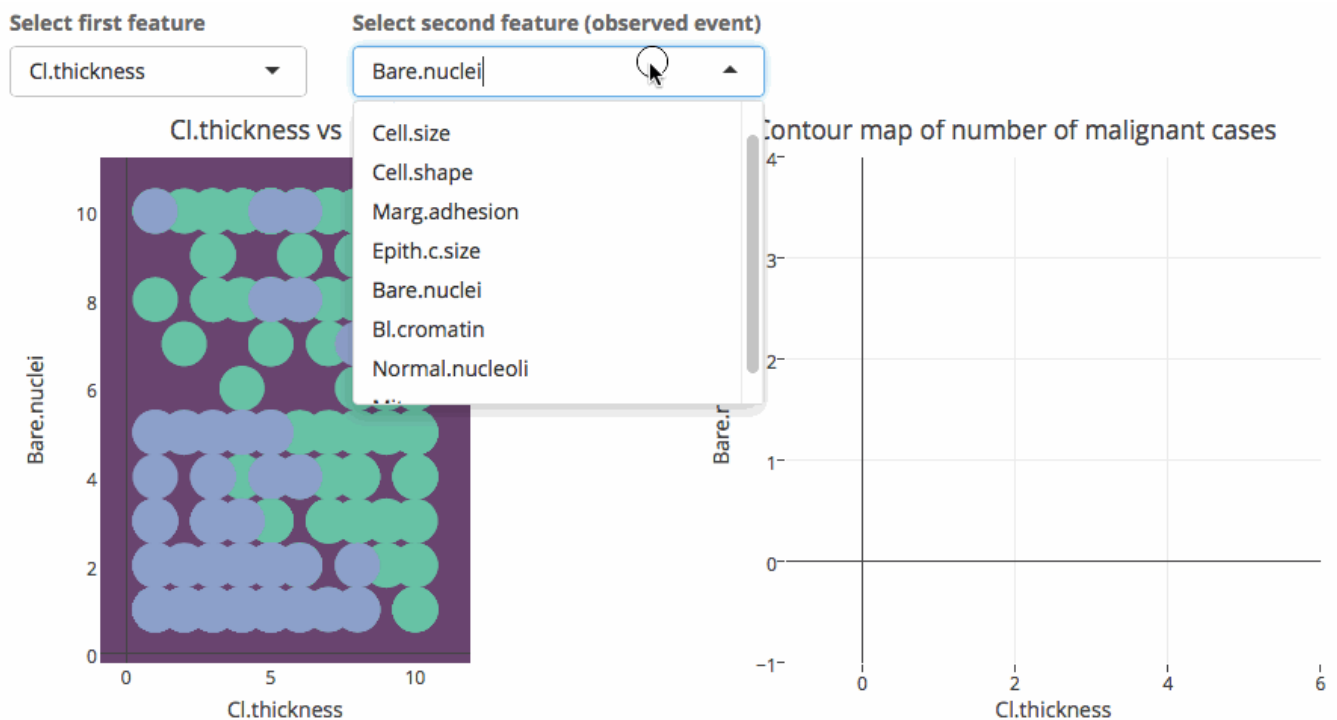




Get unlimited access

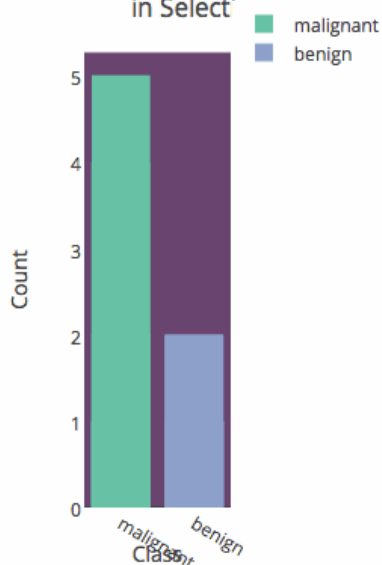
Open in app

even create interactive graphics with Shiny and Plotly's R library. Dash and Shiny are similar but Dash does not aim to be a replica of Shiny. The idioms and philosophies between Python and R are different enough to warrant a different syntax.

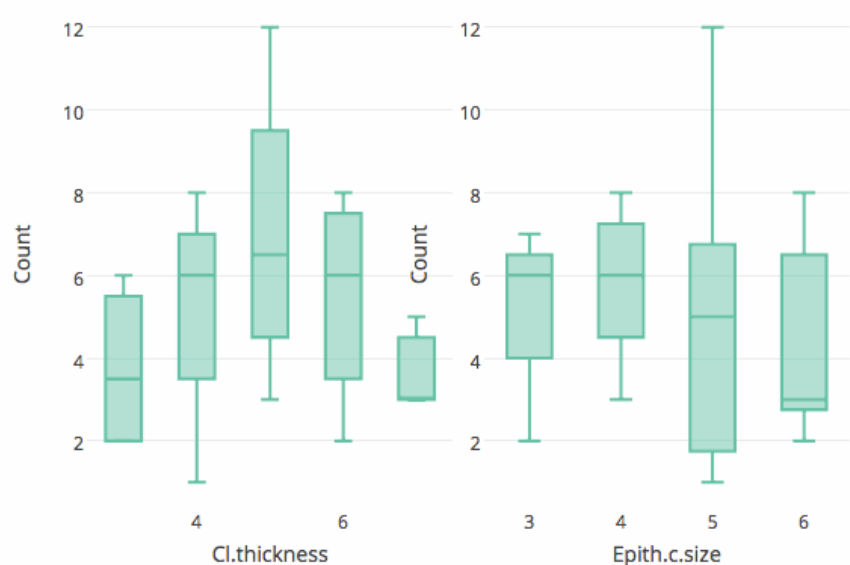


First drag a selection box in the scatter plot to populate the barchart. Then select one of the bars in the barchart to populate the boxplot

No. of Malignant and Benign cases in Select'



Box plot for Malignant cases



Interactive Web App made with Shiny in R

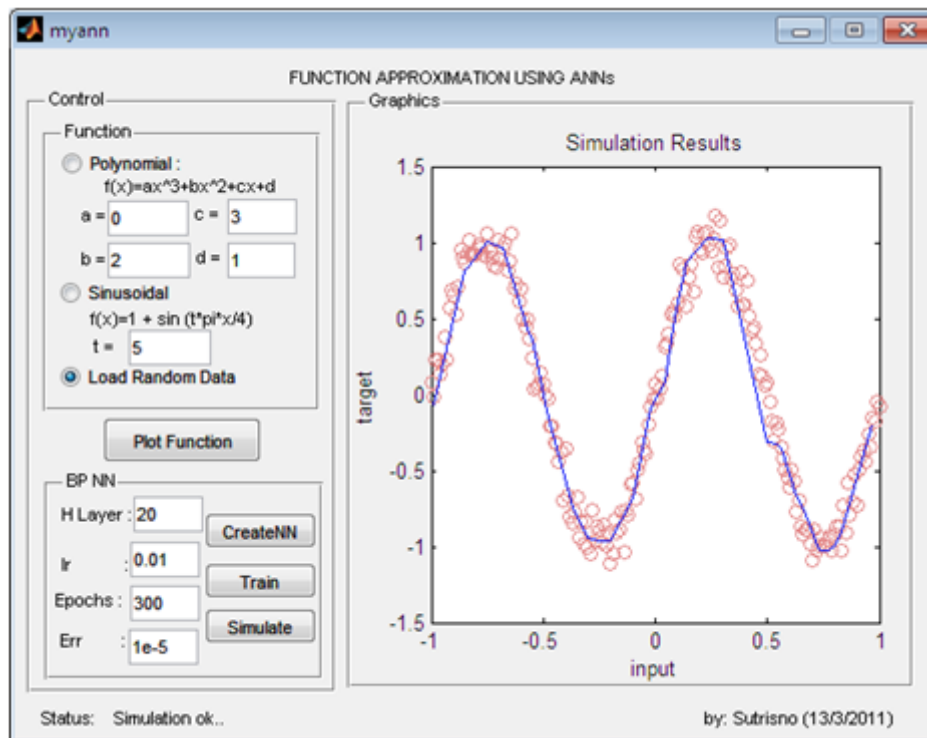




Get unlimited access

Open in app

If you program in **MATLAB** then you may be familiar with MATLAB's user interface library "GUIDE". Mathworks was one of the true original innovators in technical computing — GUIDE was written in 2004, 13 years ago!



GUIDE App built in MATLAB

If your data is structured in a database, then you may be using **Tableau** or one of the other BI tools. Tableau is incredible. They've set a new expectation in the industry that end-users should have the autonomy and the tools to be able to explore their organization's data. They've also helped popularize the concepts of "drilling down" and cross-filtering.





Get unlimited access

Open in app

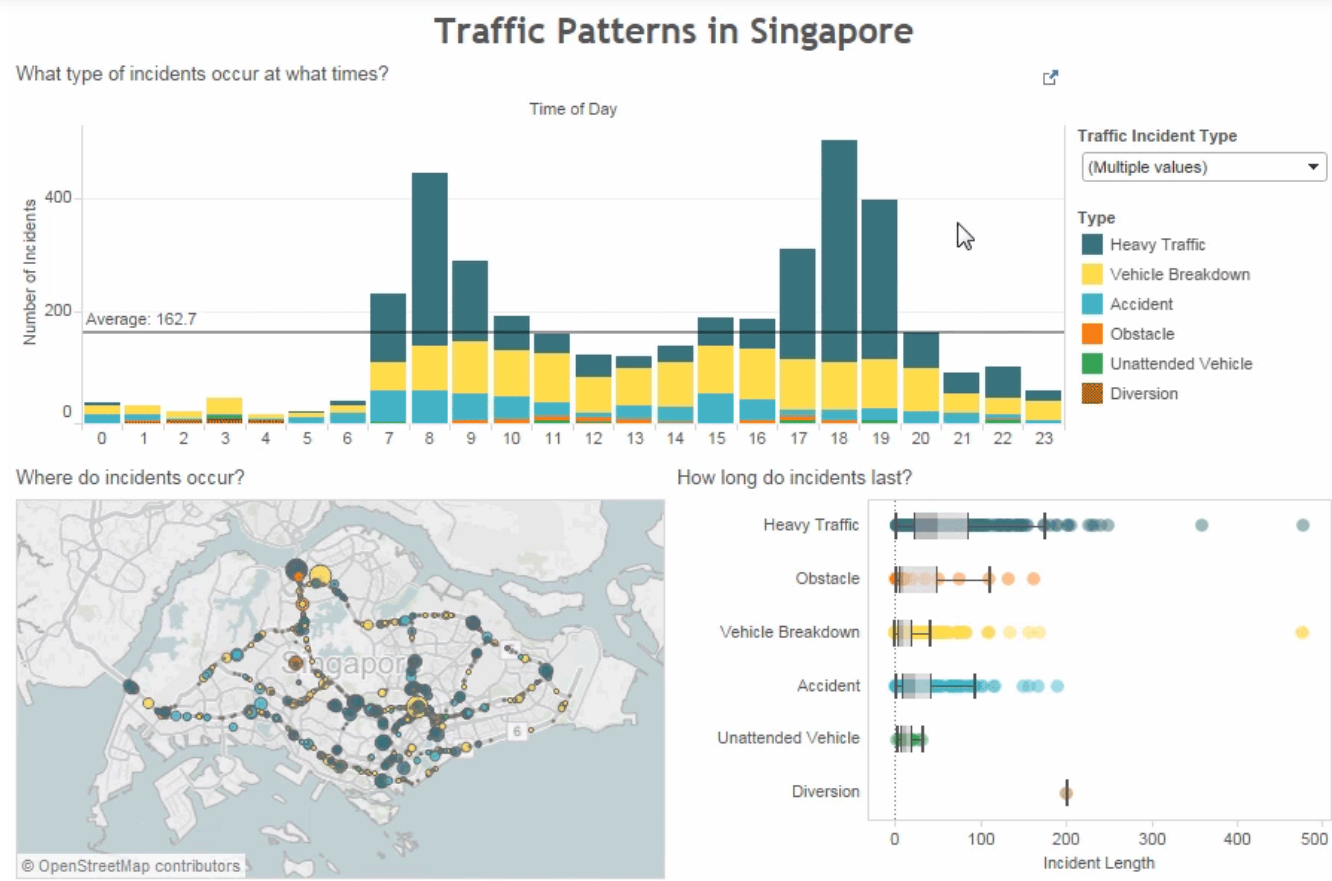


Tableau Cross-filtering

Dash is complementary to BI tools like these. These tools work great for structured data. But when it comes to data transformation and analytics, it's hard to beat the breadth and flexibility of programming languages and communities like Python. Dash abstracts away a lot of the complexities in building user interfaces, enabling you to build a beautiful front-end for your custom data analytics backend.

Finally, I'd like to give a shout out to **Jupyter widgets**. Jupyter provide a really nice widget framework inside their notebook interface. You can add sliders to your graphs in the Jupyter notebooks that you run locally.

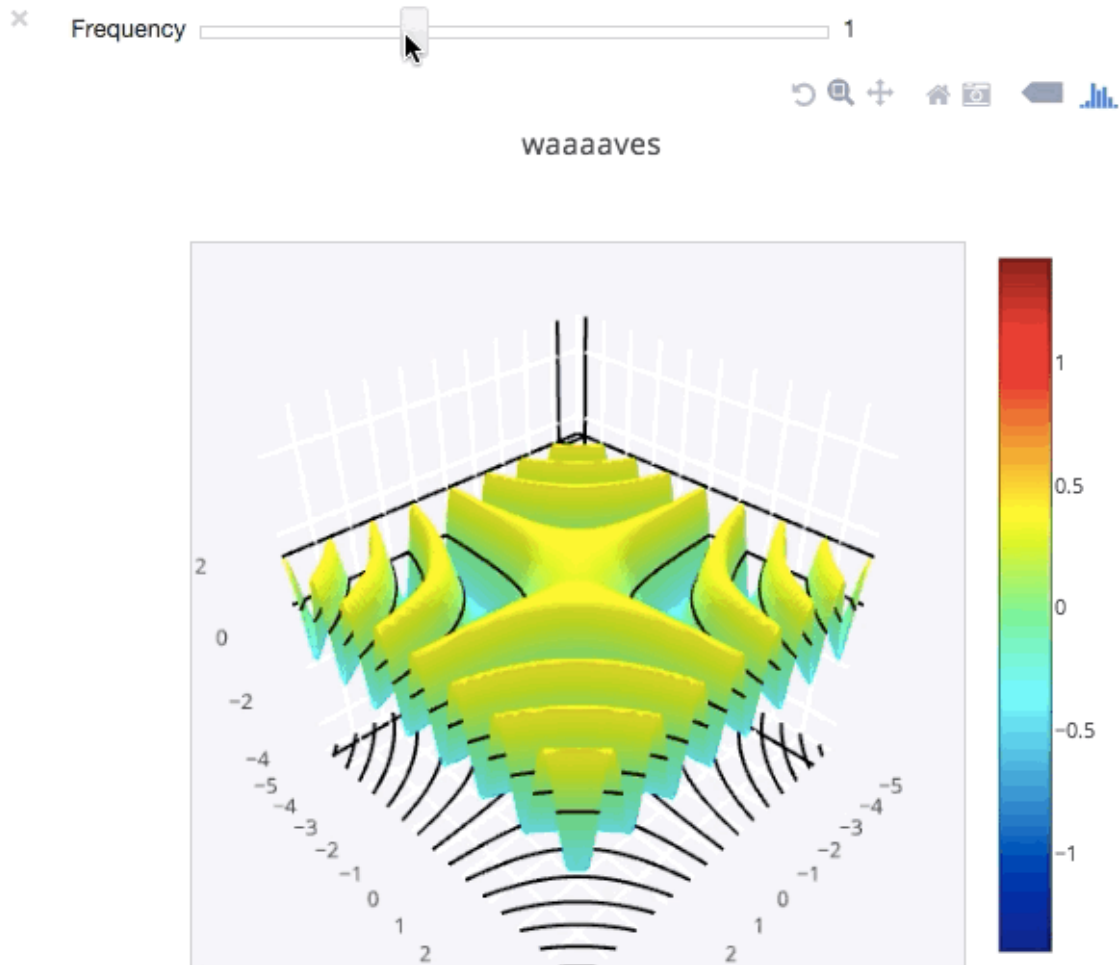




Get unlimited access

Open in app

```
In [4]: display(z_slider)
display(g)
```



The widgets in Dash are similar to the widgets in Jupyter. In Jupyter Notebooks, you can add widgets directly alongside your code. In Dash, your controls and application are kept separately from your code. Dash is aimed more towards sharable apps than it is to sharable code and notebooks. You can always mix-and-match the tools, and write your Dash apps in the Jupyter Notebook environment.

We're also big fans of the nteract project, which is really lowering the barrier to entry of Python and Jupyter Notebooks by wrapping up Jupyter Notebook as a desktop application.

Licensing and the Open Source Business Model

Plotly is a VC-backed startup. We founded in 2013 and we open sourced our core

technology in 2015 (MIT license). We maintain open source libraries in





Get unlimited access

Open in app

We provide subscriptions to our chart hosting and sharing platform, and to our chart editing and database querying app. This platform is available on the web (plot.ly) and [on-premise](#).

We're applying a similar model to Dash. Dash is MIT licensed. It's free to use and to modify. For companies, we're offering Dash Enterprise, a deployment server for easily publishing and provisioning Dash Apps behind your firewall.

Our goal with Dash Enterprise is to make sharing a Dash app internally as easy and secure as possible. No dev-ops required. Dash Enterprise handles the URL routing, the monitoring, the failure handling, the deployment, the versioning, and the package management. Dash Apps deployed with Dash Enterprise can be provisioned through your company's Active Directory or LDAP user accounts.

If you're using the open source version locally, there are no restrictions. You can manage deployment of Dash apps yourself through platforms like Heroku or Digital Ocean. If you have the resources, consider purchasing a [support plan](#) to get one-on-one help from a Plotly engineer. If you need more specialized help or would like to fund specific feature development, reach out to our [advanced development](#) program.

Open source is still a new idea for product companies, yet at the end of the day, we're able to dedicate more than half of our staff towards open source products. Huge thanks to everyone who has supported us so far ❤️

Thanks for checking out Dash. I'll be giving a talk about Dash at SciPy this summer in Austin and in next fall at Plotcon NYC. If you'll be at either of those events, please say hi! Otherwise, I'll see you on GitHub 🙌 📄

Further Resources and Footnotes

1. Our Dash documentation is hosted at <https://plot.ly/dash>
2. All of our open source work is in our GitHub organization at <https://github.com/plotly>
3. If you'd like to fund specialized features, reach out to our Advanced Development team: plot.ly/products/consulting-and-oem/





Get unlimited access

Open in app

5. If you're looking for inspiration in user interfaces for technical computing, I highly recommend Bret Victor's essay on [What Can A Technologist Do About Climate Change?](#) In particular, the sections on [Technical computing](#) and [Media for understanding situations](#)
6. Related, if you find the intersect between technical computing and interface interesting, you might like [Explorable Explanations](#)
7. You can reach out to me directly at chris@plot.ly or on twitter at [@chriddyp](https://twitter.com/chriddyp)

