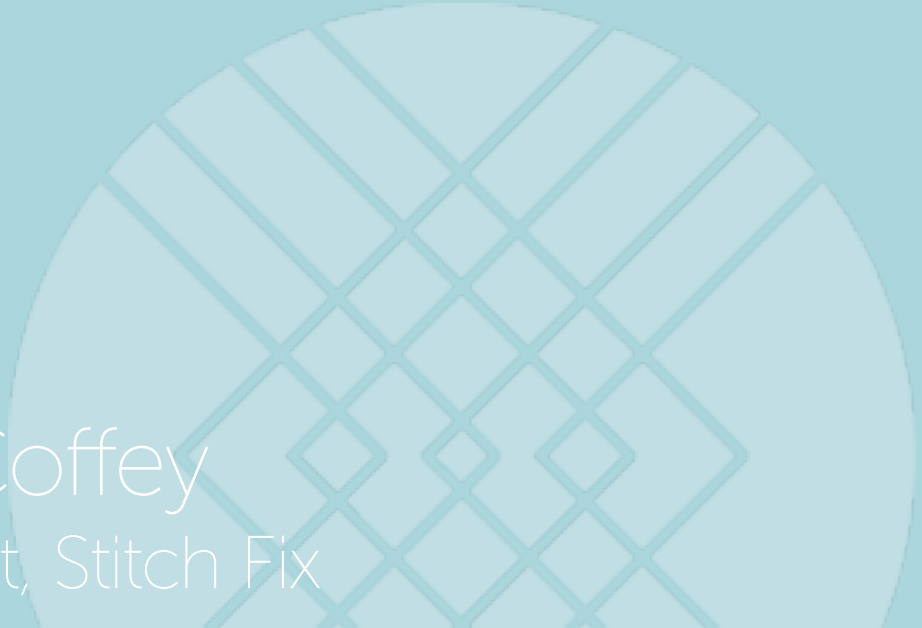
D3 in Jupyter

PyData NYC 2015

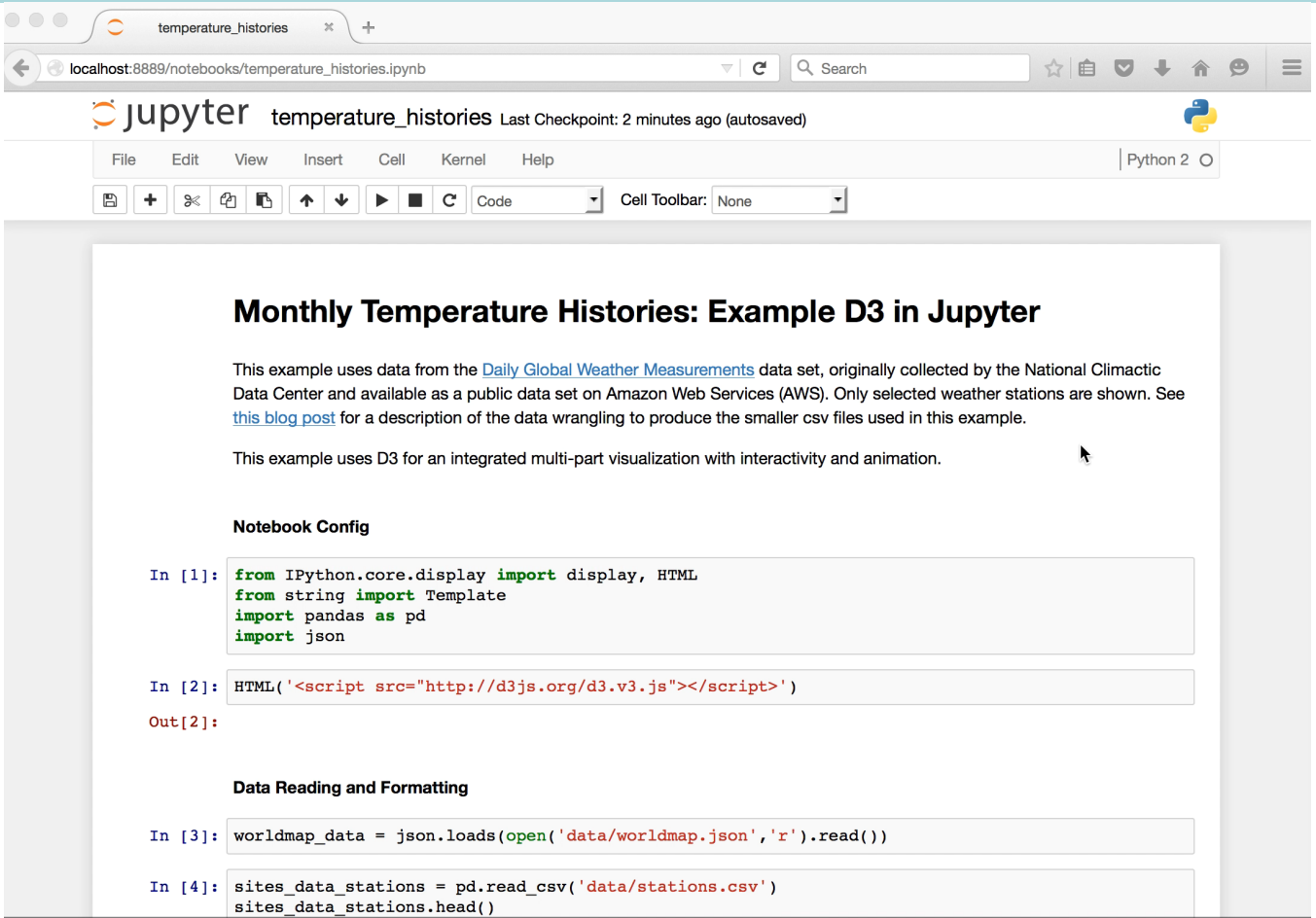


Brian Coffey

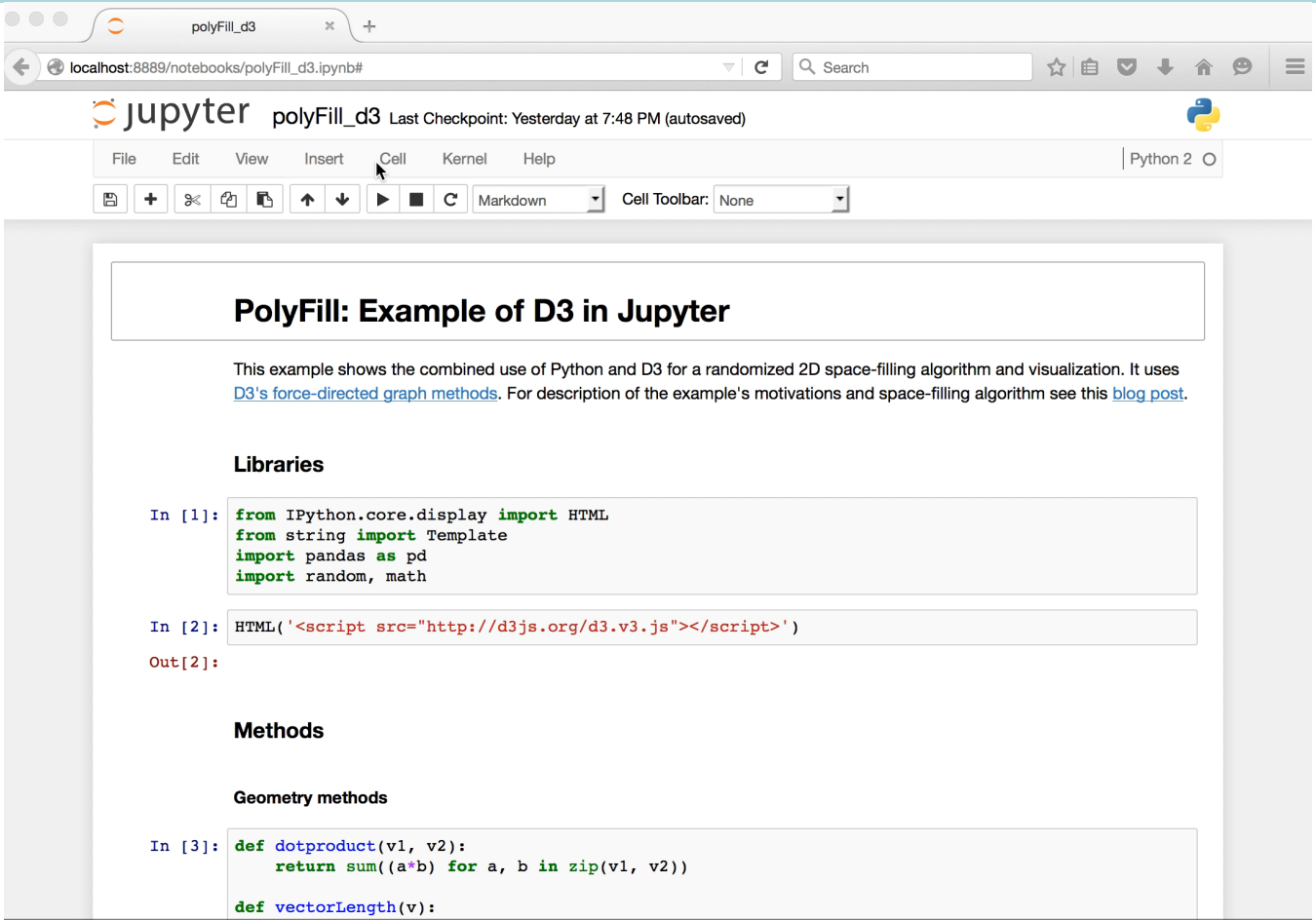
Data Scientist, Stitch Fix

https://github.com/stitchfix/d3-jupyter-tutorial

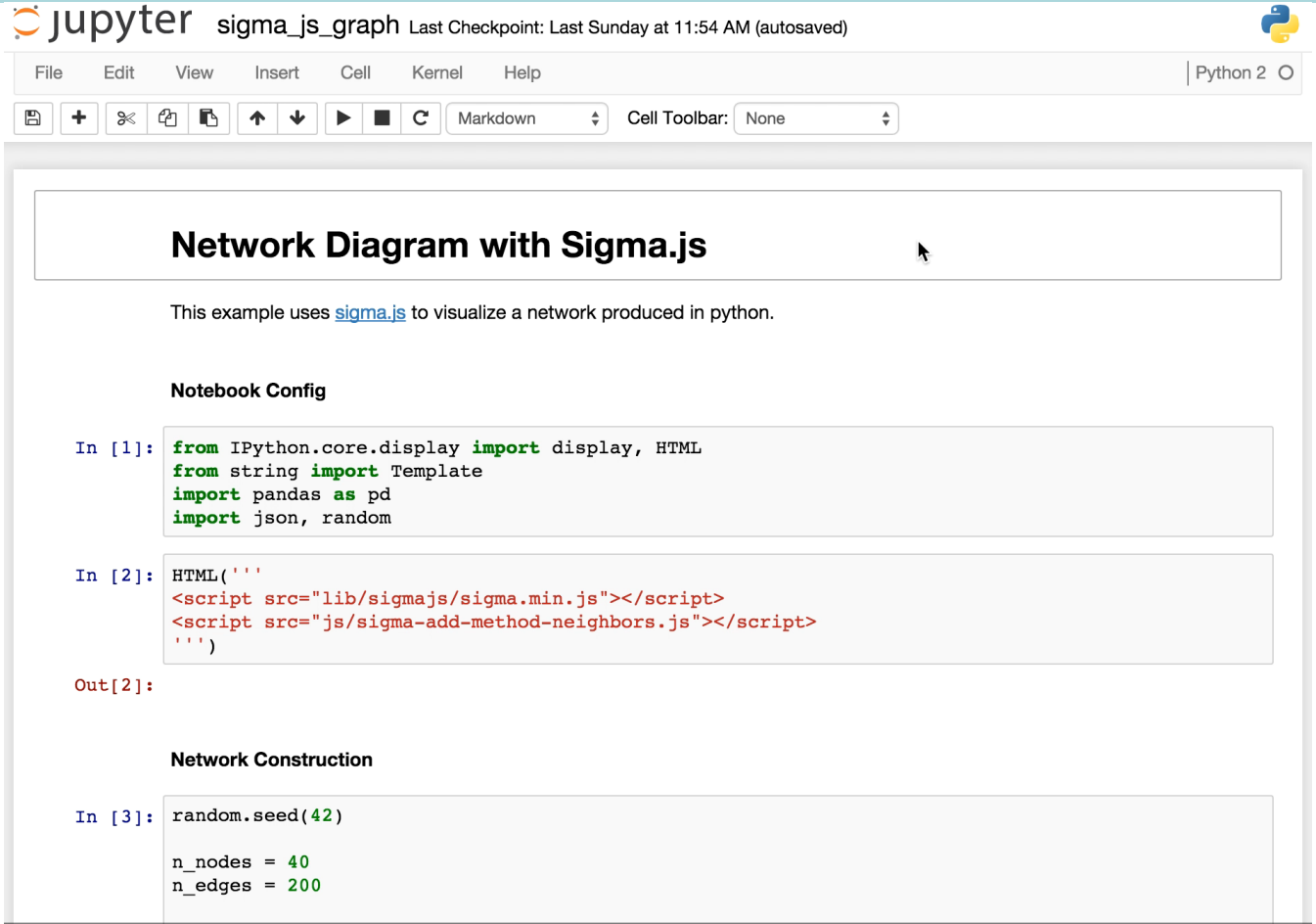
https://github.com/stitchfix/d3-jupyter-tutorial



https://github.com/stitchfix/d3-jupyter-tutorial



https://github.com/stitchfix/d3-jupyter-tutorial



Overview

1. Why you may occasionally wish to do this (but likely not all the time)
2. Hello DOM!
3. Meet bl.ocks.org, I think you'll be great friends
4. A simple D3 scatterplot in Jupyter
5. A bar chart with interactivity (hello again DOM!)
6. Networks, maps and more
7. D3_lib.py (if you want to)
8. Discussion: From sketch to notebook frame

1. Why you may occasionally wish to do this (but likely not all the time)

Why not all the time?

matplotlib, mpld3, bokeh, seaborn, toyplot, vispy, vincent, ...

1. Why you may occasionally wish to do this (but likely not all the time)

Why not all the time?

matplotlib, mpld3, bokeh, seaborn, toyplot, vispy, vincent, ...

Okay. Then why bother at all?

1. Maybe you just want the ultimate freedom in your data visualization, and you happen to have some extra time on your hands, and/or you happen to be a D3 wiz.
2. Maybe the perfect visualization for your analytical needs is not readily available with other graphing tools, but there is a D3 example out there that would be perfect with just some minor tweaks.
3. Maybe you are collaborating with a front-end person on a custom viz

Just a heads up before we dig in.

If you are not already familiar with html, css and javascript, your eyes may glaze over a bit during the first part of this tutorial.

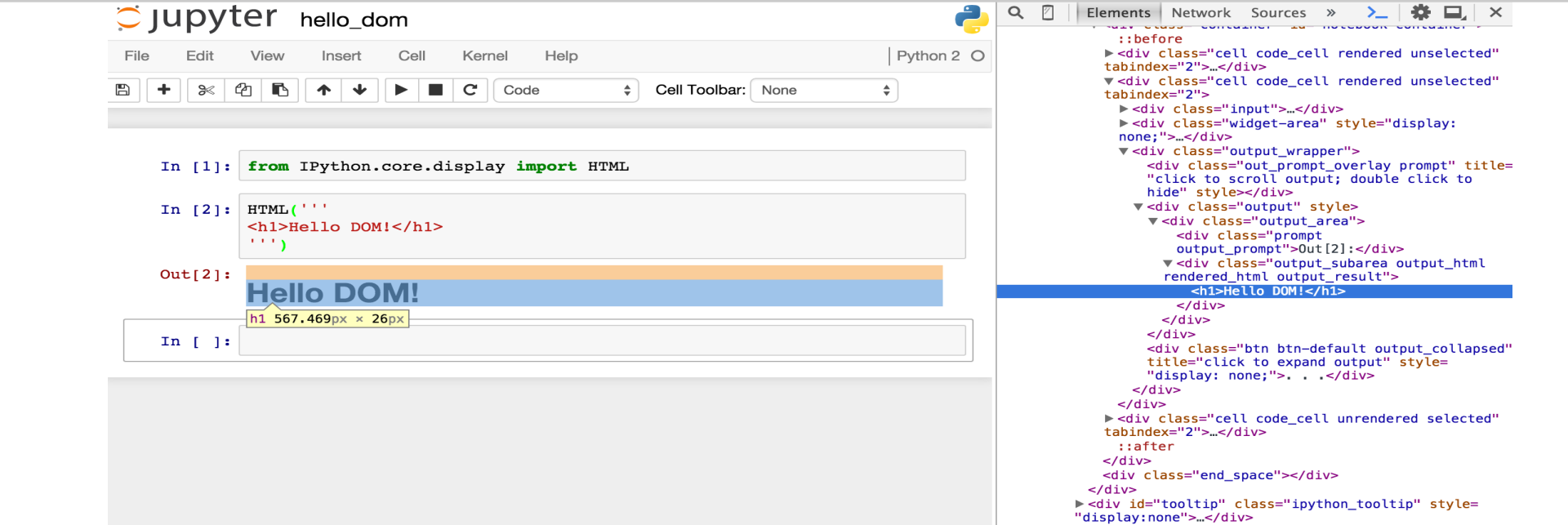
But get what you can out of it, as understanding how it works will help you to make the most of D3 in your Jupyter notebooks. I will try to be very clear on the essential pieces of information you need to function, and on the major gotchas for troubleshooting.

And if you get too lost, fear not. At the other end, there is a small helper library and a collection of examples that you can use straight out of the box.

notebook: **hello_dom.ipynb**

"Oh, you can just call me Document Object Model for short"

Check me out in your browser's development tools. Highlight a particular element using the magnifying glass.



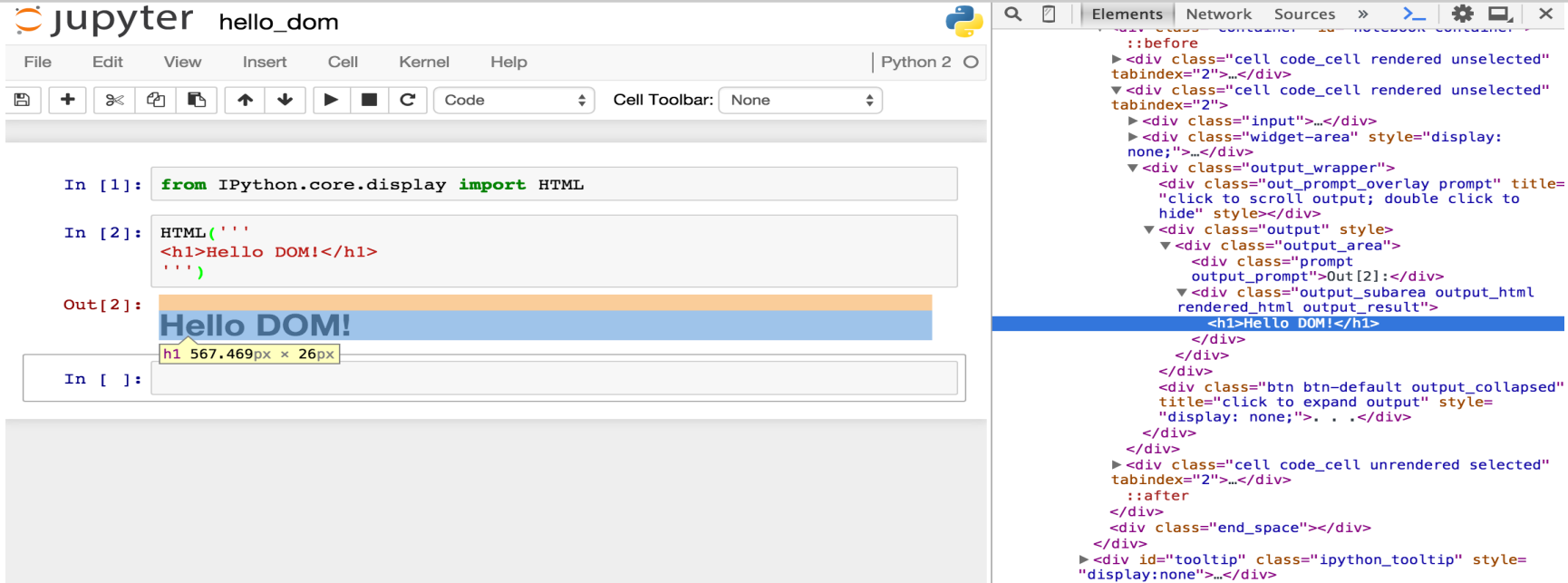
2. Hello DOM!

hello_dom.ipynb

```
from IPython.core.display import HTML
HTML( '<h1>Hello DOM!</h1>' )
```

in general ... HTML([insert a html string here and I will put it in the DOM])

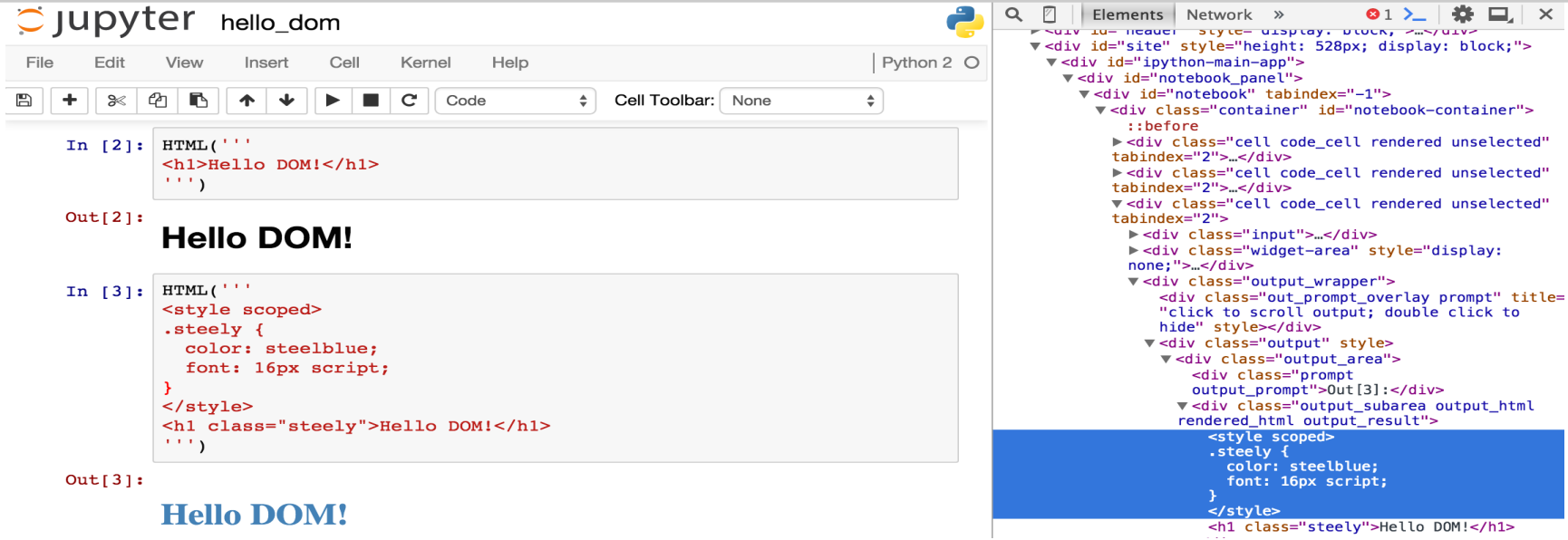
(I will put it in the div assigned to the output_result of the code block that I am in.)



hello_dom.ipynb

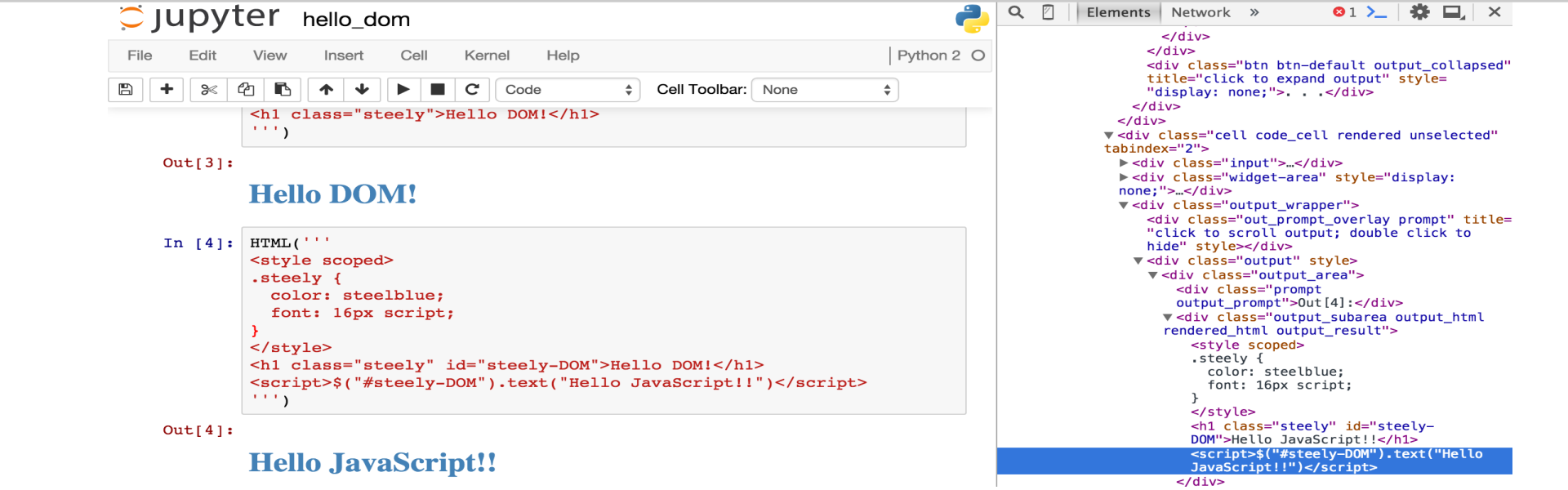
Let's add some style...

```
HTML('
<style> … </style>
<h1 class="steely">Hello DOM!</h1>')
```



And some javascript to modify the newly-added DOM element

```
HTML( '…
<h1 class="steely" id="steely-DOM">Hello DOM!</h1>
<script>$('steely-DOM').text("Hello JavaScript!!!")</script>')
```



Great. But how does D3 fit into this?

D3 is a JavaScript library. The Ds are for Data Driven Documents.
Think 'Data Driven DOM'.

A core aspect of using the D3 library is the following pattern:

1. Select some array of DOM elements
   Note that this may be an empty array
2. 'Attach' an array of data to that array of DOM elements
   The array lengths need not match. Extra data may 'enter', or DOM elements may 'exit'. D3 coordinates all of this with elegance.
3. Perform some actions for each element in the data array
   This will generally involve modifying the selected DOM elements, or removing or appending new DOM elements, based on the values of the data array.

Appending elements to the DOM with D3

jupyter hello_dom

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

In [5]:
```
HTML('<script src="lib/d3/d3.min.js"></script>')
```

Out[5]:

In [6]:
```
HTML('''
<style scoped>
.bedazzled {
  color: orange;
}
</style>
<div id="d3-div-1"></div>
<script>

var size_data = [10,20,30];

d3.select("#d3-div-1").selectAll('.bedazzled')
    .data(size_data)
    .enter().append('p')
      .attr("class","bedazzled")
      .style("font-size", function(d){ return "" + d + "px";})
      .text("Hello D3!");

</script>
''')
```

Out[6]: Hello D3!

Hello D3!

Hello D3!

```
tabindex="2">…</div>
▶ <div class="cell code_cell rendered unselected"
tabindex="2">…</div>
▼ <div class="cell code_cell rendered unselected"
tabindex="2">
  ▶ <div class="input">…</div>
  ▶ <div class="widget-area" style="display:
  none;">…</div>
  ▼ <div class="output_wrapper">
      <div class="out_prompt_overlay prompt" title=
      "click to scroll output; double click to
      hide" style></div>
    ▼ <div class="output" style>
      ▼ <div class="output_area">
          <div class="prompt
          output_prompt">Out[6]:</div>
        ▼ <div class="output_subarea output_html
        rendered_html output_result">
            <style scoped>
            .bedazzled {
              color: orange;
            }
            </style>
          ▼ <div id="d3-div-1">
              <p class="bedazzled" style="font-
              size: 10px;">Hello D3!</p>
              <p class="bedazzled" style="font-
              size: 20px;">Hello D3!</p>
              <p class="bedazzled" style="font-
              size: 30px;">Hello D3!</p>
            </div>
          ▶ <script>…</script>
          </div>
        </div>
      </div>
      <div class="btn btn-default output_collapsed"
      title="click to expand output" style=
      "display: none;">. . .</div>
    </div>

… div div div div div #d3-div-1 p.bedazzled

Styles Event Listeners DOM Breakpoints Properties

Console Search Emulation Rendering

<top frame> Preserve log
```

hello_dom.ipynb

Just one last step – writing data from Python into the JavaScript for use by D3

jupyter hello_dom

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

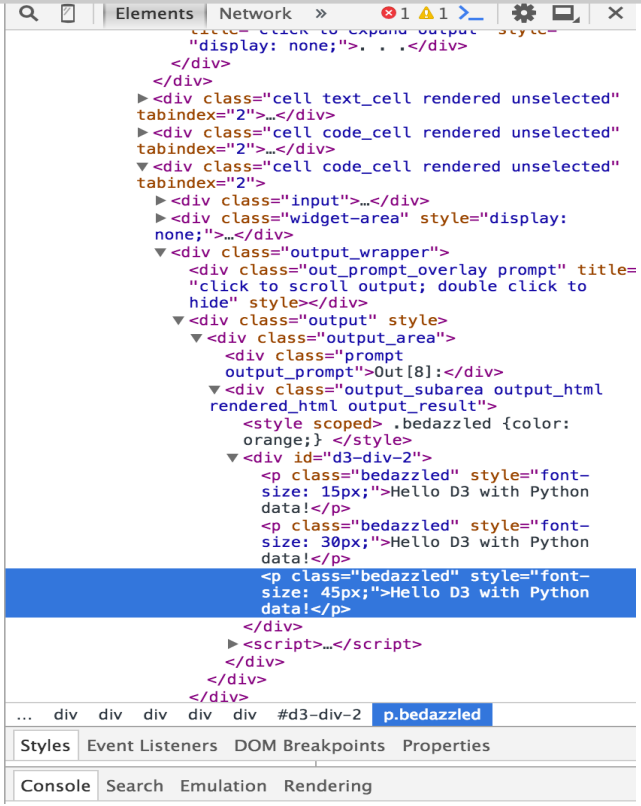
In [7]:
```
size_data = [15,30,45]
```

In [8]:
```
from string import Template
html_template = Template('''
<style scoped> .bedazzled {color: orange;} </style>
<div id="d3-div-2"></div>
<script>

var size_data = $size_data_python ;

d3.select("#d3-div-2").selectAll('.bedazzled')
    .data(size_data)
    .enter().append('p')
      .attr("class","bedazzled")
      .style("font-size", function(d){ return "" + d + "px";})
      .text("Hello D3 with Python data!");

</script>
''')
HTML(html_template.substitute({'size_data_python': str(size_data)}))
```

Out[8]: Hello D3 with Python data!

Hello D3 with Python data!

Hello D3 with Python data!



A Brief Review of Key Points:

IPython.core.display.HTML allows you to directly write new DOM elements

With that HTML function, we can add css (<style>) and javascript (<script>) elements, as well as elements (or <h1> elements, etc) for them to act on

D3 is a clever javascript library for DOM modification based on arrays of data

We can write python data directly into <script> element text so that it can be acted upon by D3.

for any late-joiners:

https://github.com/stitchfix/d3-jupyter-tutorial

A Brief Review of Key Points:

IPython.core.display.HTML allows you to directly write new DOM elements

With that HTML function, we can add css (<style>) and javascript (<script>) elements, as well as elements (or <h1> elements, etc) for them to act on

D3 is a clever javascript library for DOM modification based on arrays of data

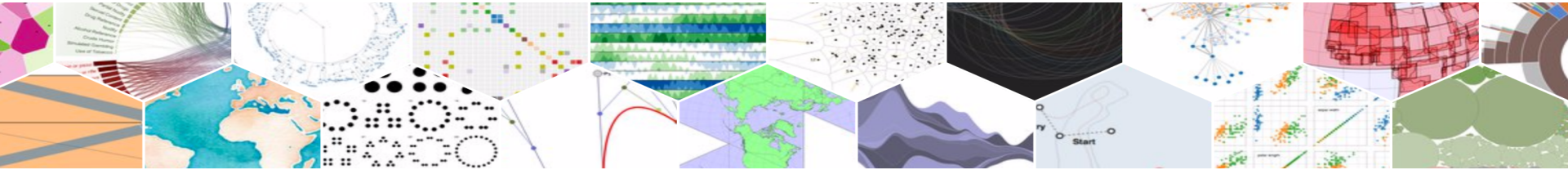
We can write python data directly into <script> element text so that it can be acted upon by D3.

Sure. Great. But I haven't seen any data visualizations yet...

3. Meet bl.ocks.org, I think you'll be great friends

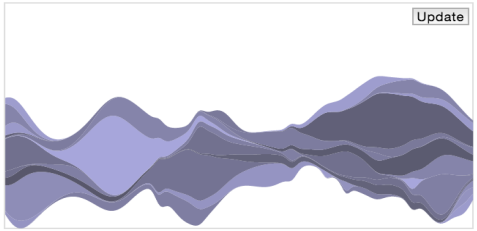
`d3js.org , bl.ocks.org/mbostock`

(And welcome to data visualization heaven)



`http://bl.ocks.org/mbostock/4060954`

Streamgraph



For continuous data such as time series, a streamgraph can be used in place of stacked bars. This example also demonstrates path transitions to interpolate between different layouts. Streamgraph algorithm, colors, and data generation inspired by Byron and Wattenberg.

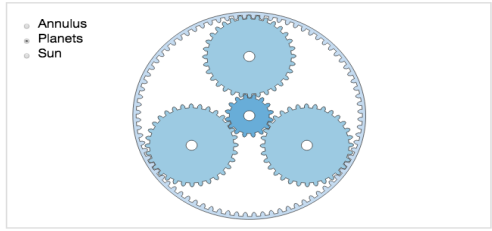
Open in a new window.

index.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>Streamgraph</title>
<style>

body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-ser
  margin: auto;
  position: relative;
  width: 960px;
```

`http://bl.ocks.org/mbostock/1353700`

Epicyclic Gearing



From Wikipedia:

> **Epicyclic gearing** or **planetary gearing** is a gear system consisting of one or more outer gears, or *planet* gears, revolving about a central, or *sun* gear. … Epicyclic gearing systems also incorporate the use of an outer ring gear or *annulus*, which meshes with the planet gears.

Use the menu in the top-left to change the frame of reference, fixing the specified gear in-place.

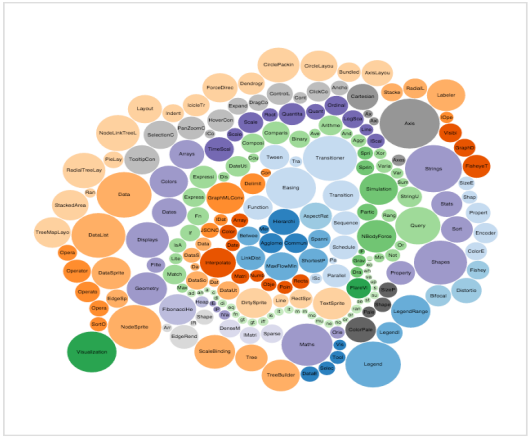
Open in a new window.

index.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>
```

`http://bl.ocks.org/mbostock/4063269`

Bubble Chart



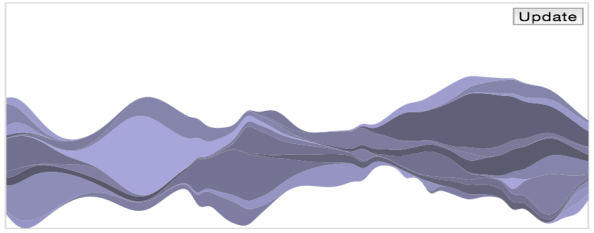
Bubble charts encode data in the area of circles. Although less perceptually-accurate than bar charts, they can pack hundreds of values into a small space. Implementation based on work by Jeff Heer. Data shows the Flare class

Open in a new window.

3. Meet bl.ocks.org, I think you'll be great friends

http://bl.ocks.org/mbostock/4060954

Streamgraph



For continuous data such as time series, a streamgraph can be used in place of stacked bars. This example also demonstrates path transitions to interpolate between different layouts. Streamgraph algorithm, colors, and data generation inspired by Byron and Wattenberg.

Open in a new window.

index.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>Streamgraph</title>
<style>

body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-ser
  margin: auto;
  position: relative;
  width: 960px;
}

button {
  position: absolute;
  right: 10px;
  top: 10px;
}

</style>
<button onclick="transition()">Update</button>
<script src="//d3js.org/d3.v3.min.js"></script>
<script>

var n = 20, // number of layers
    m = 200, // number of samples per layer
    stack = d3.layout.stack().offset("wiggle"),
    layers0 = stack(d3.range(n).map(function() { return bum
    layers1 = stack(d3.range(n).map(function() { return bum

var width = 960,
    height = 500;

var x = d3.scale.linear()
```

Anatomy of a bl.ock

Pretty graphic, often interactive

Short description

html, css and javascript that produces the graphic

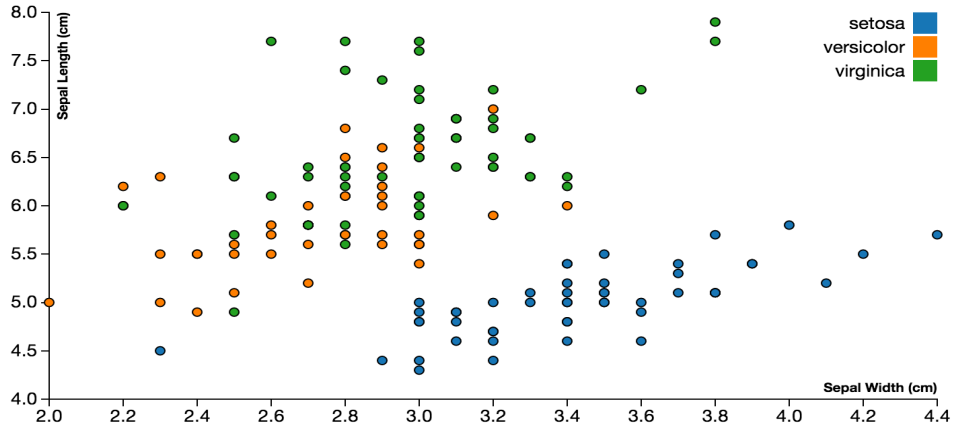
data that gets read by script, usually in csv or tsv form

Using this bl.ocks example: **http://bl.ocks.org/mbostock/3887118**

We will produce this in Jupyter:

In [7]:
```
html_template = Template('''
<style> $css_text </style>
<div id="graph-div"></div>
<script> $js_text </script>
''')
js_text = js_text_template.substitute({'python_data': json.dumps(iris_array_of_dicts),
                                       'graphdiv': 'graph-div'})
HTML(html_template.substitute({'css_text': css_text, 'js_text': js_text}))
```

Out[7]:



5. A bar chart with interactivity

bar_chart_with_update.ipynb

(hello again DOM!)

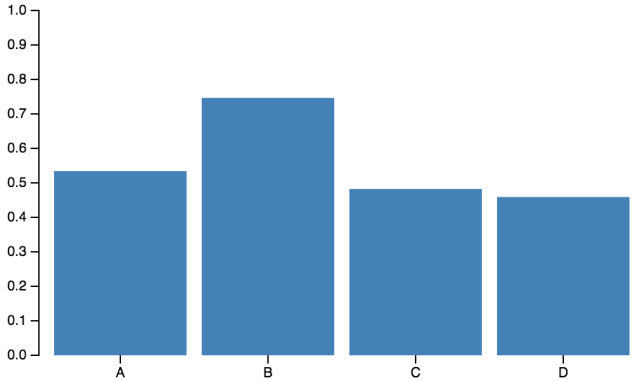
In [7]:
```
data = pd.DataFrame({'letter': ['A','B','C','D'], 'y': [1,1,1,1]})
data.head()
```

Out[7]:

	letter	y
0	A	1
1	B	1
2	C	1
3	D	1

In [8]:
```
js_text = js_text_template.substitute({'data': json.dumps(data.to_dict(orient='records'))})
HTML(html_template.substitute({'css_text': css_text, 'js_text': js_text}))
```

Out[8]:



watch out for the twist...

6. Networks, maps and more

sigma_js_graph.ipynb

```
In [5]: pd.DataFrame(graph_data['edges']).head()
```

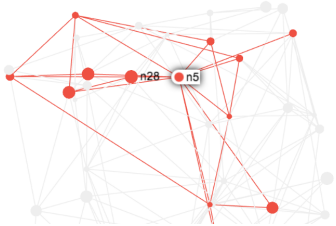
Out[5]:

	id	source	target
0	e0	n5	n12
1	e1	n25	n1
2	e2	n10	n20
3	e3	n6	n29
4	e4	n15	n17

Visualization

```
In [6]: js_text_template = Template(open('js/sigma-graph.js','r').read())

        js_text = js_text_template.substitute({'graph_data': json.dumps(graph_data),
                                               'container': 'graph-div'})

        html_template = Template('''
        <div id="graph-div" style="height:400px"></div>
        <script> $js_text </script>
        ''')
        HTML(html_template.substitute({'js_text': js_text}))
```

Out[6]:

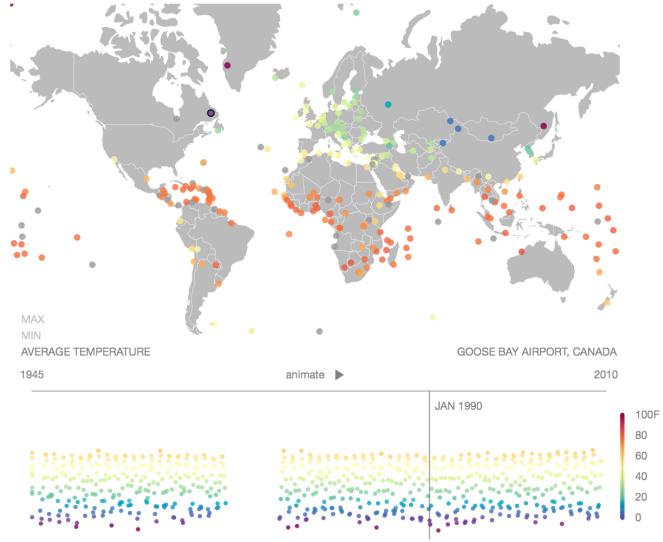


temperature_histories.ipynb

polyFill_d3.ipynb

3d_meshing.ipynb

```
In [13]: display(HTML(html_template.substitute({'css_text': css_text, 'js_text': js_text})))
```



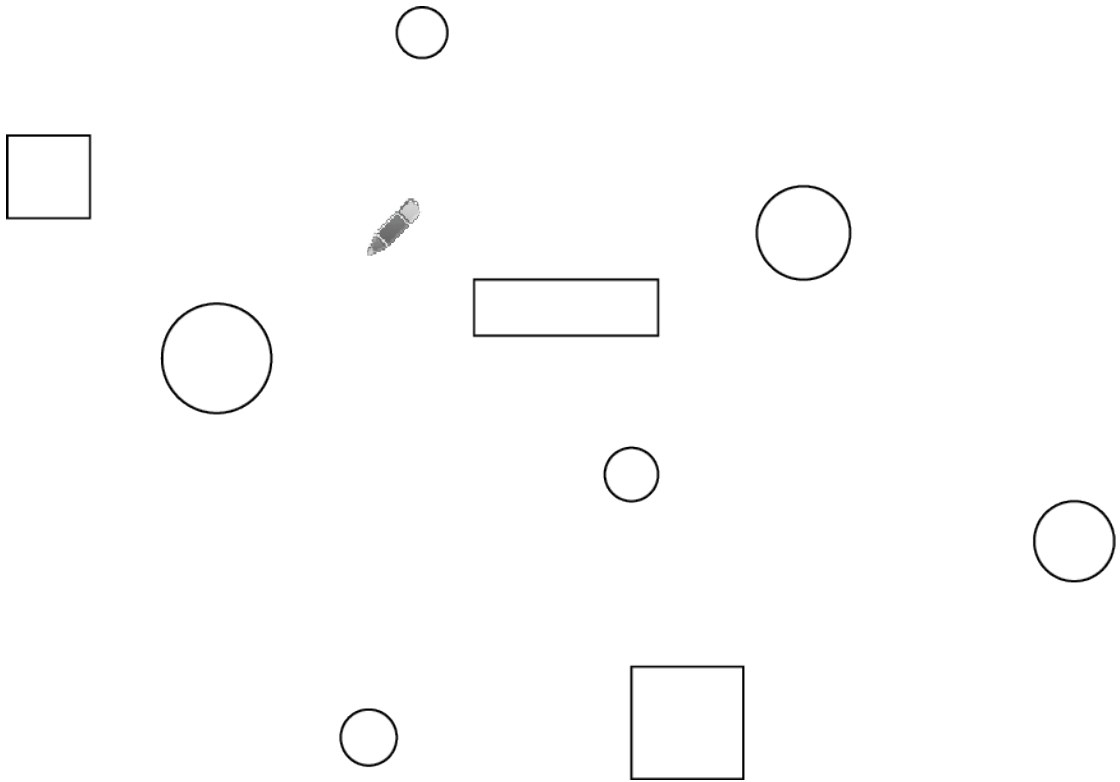
`multiple_simple_examples.ipynb`

We have encapsulated a lot of the boilerplate for this method into an external python file, which in turn calls external files containing your css and js templates.

This just helps to keep things visually clean.

Feel free to use it as is and/or modify it to suit your needs.

`randomized_sketch.ipynb`



Thank you!

Data Science @ Stitch Fix

bcoffey@stitchfix.com

multithreaded.stitchfix.com