

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

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

The screenshot shows a Jupyter Notebook interface with the title "temperature_histories". The notebook is running on "localhost:8889/notebooks/temperature_histories.ipynb" and has a Python 2 kernel. The main content area displays a section titled "Monthly Temperature Histories: Example D3 in Jupyter" which describes the use of the Daily Global Weather Measurements dataset. Below this, there is a "Notebook Config" section and two code cells. Cell [1] contains code to import IPython, string, pandas, and json modules. Cell [2] contains code to embed a D3.js script. The output of Cell [2] is shown as "Out[2]". Following this, there is a "Data Reading and Formatting" section with two more code cells. Cell [3] reads worldmap_data from a JSON file, and Cell [4] reads sites_data_stations from a CSV file.

Monthly Temperature Histories: Example D3 in Jupyter

This example uses data from the [Daily Global Weather Measurements](#) data set, originally collected by the National Climatic Data Center and available as a public data set on Amazon Web Services (AWS). Only selected weather stations are shown. See [this blog post](#) for a description of the data wrangling to produce the smaller csv files used in this example.

This example uses D3 for an integrated multi-part visualization with interactivity and animation.

Notebook Config

```
In [1]: from IPython.core.display import display, HTML
from string import Template
import pandas as pd
import json
```

```
In [2]: HTML('<script src="http://d3js.org/d3.v3.js"></script>')
```

Out[2]:

Data Reading and Formatting

```
In [3]: worldmap_data = json.loads(open('data/worldmap.json', 'r').read())
```

```
In [4]: sites_data_stations = pd.read_csv('data/stations.csv')
sites_data_stations.head()
```

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

The screenshot shows a Jupyter Notebook interface running in a web browser. The title bar indicates the notebook is titled "polyFill_d3" and the URL is "localhost:8889/notebooks/polyFill_d3.ipynb#". The toolbar includes standard options like File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 kernel selector. Below the toolbar is a cell toolbar with various icons for cell operations.

PolyFill: Example of D3 in Jupyter

This example shows the combined use of Python and D3 for a randomized 2D space-filling algorithm and visualization. It uses [D3's force-directed graph methods](#). For description of the example's motivations and space-filling algorithm see this [blog post](#).

Libraries

```
In [1]: from IPython.core.display import HTML
from string import Template
import pandas as pd
import random, math
```

```
In [2]: HTML('<script src="http://d3js.org/d3.v3.js"></script>')
```

```
Out[2]:
```

Methods

Geometry methods

```
In [3]: def dotproduct(v1, v2):
    return sum((a*b) for a, b in zip(v1, v2))

def vectorLength(v):
```

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

jupyter sigma_js_graph Last Checkpoint: Last Sunday at 11:54 AM (autosaved)



File Edit View Insert Cell Kernel Help

Python 2



Markdown

Cell Toolbar: None

Network Diagram with Sigma.js

This example uses [sigma.js](#) to visualize a network produced in python.

Notebook Config

```
In [1]: from IPython.core.display import display, HTML
from string import Template
import pandas as pd
import json, random
```

```
In [2]: HTML('''
<script src="lib/sigmajs/sigma.min.js"></script>
<script src="js/sigma-add-method-neighbors.js"></script>
''')
```

Out[2]:

Network Construction

```
In [3]: random.seed(42)

n_nodes = 40
n_edges = 200
```

Overview

1. Why you may occasionally wish to do this (but likely not all the time)
2. Hello DOM!
3. Meet blocks.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

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

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.

2. Hello DOM!

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.

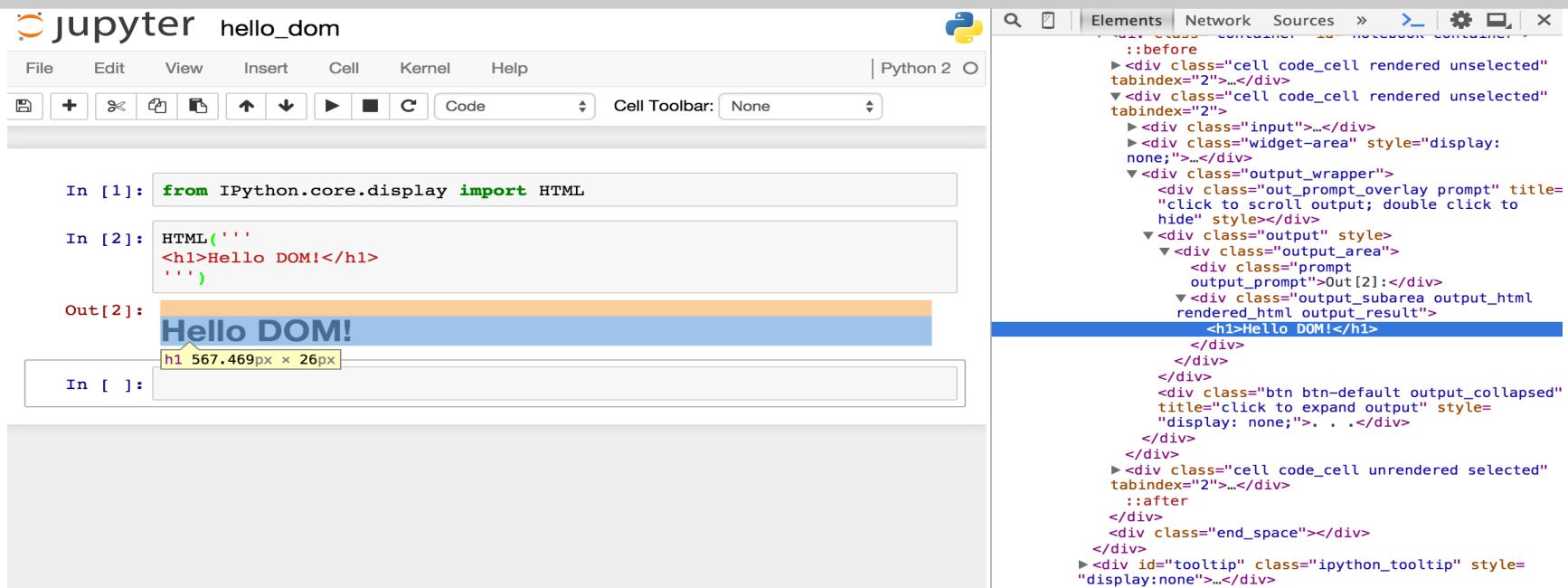
The screenshot shows a Jupyter Notebook interface with the following details:

- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Help, Python 2.
- In [1]:** `from IPython.core.display import HTML`
- In [2]:** `HTML('''<h1>Hello DOM!</h1>'''')`
- Out [2]:** A yellow-highlighted `<h1>Hello DOM!</h1>` element.
- Code Cell:** In []: (empty)
- Developer Tools:** An open developer tools panel shows the DOM structure of the highlighted element:
 - Element node: `<h1>Hello DOM!</h1>`
 - Style node: `h1 567.469px x 26px`
 - Parent elements: `<div class="cell code_cell rendered unselected" tabindex="2">...</div>`, `<div class="cell code_cell rendered unselected" tabindex="2">`, `<div class="output_wrapper">`, `<div class="output" style="display: none;">...</div>`, `<div class="output_area">`, `<div class="prompt" style="display: none;">...</div>`, `<div class="output_subarea output_html rendered_html output_result">`, `<div class="btn btn-default output_collapsed" style="display: none;">...</div>`.

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])



2. Hello DOM!

hello_dom.ipynb

Let's add some style...

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

Jupyter hello_dom

File Edit View Insert Cell Kernel Help | Python 2

In [2]:

```
HTML(''
<h1>Hello DOM!</h1>
'')
```

Out[2]:

Hello DOM!

In [3]:

```
HTML(''
<style scoped>
.steely {
    color: steelblue;
    font: 16px script;
}
</style>
<h1 class="steely">Hello DOM!</h1>
'')
```

Out[3]:

Hello DOM!

Elements Network

```
<div id="site" style="height: 528px; display: block;">
  <div id="ipython-main-app">
    <div id="notebook_panel">
      <div id="notebook" tabindex="-1">
        <div class="container" id="notebook-container">
          <div class="cell code_cell rendered unselected" tabindex="2">...</div>
          <div class="cell code_cell rendered unselected" tabindex="2">...</div>
          <div class="cell code_cell rendered unselected" tabindex="2">...</div>
        <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">...</div>
        <div class="output" style="border: 1px solid #ccc; border-radius: 5px; padding: 5px; margin-bottom: 10px;">
          <div class="output_area">
            <div class="prompt" style="margin-bottom: 10px;">Out[3]:</div>
            <div class="output_subarea output_html rendered_html output_result" style="border: 1px solid #ccc; border-radius: 5px; padding: 5px; background-color: #f9f9f9; min-height: 100px; height: 100px; width: 100%;">
              <style scoped>
                .steely {
                  color: steelblue;
                  font: 16px script;
                }
              </style>
              <h1 class="steely">Hello DOM!</h1>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

2. Hello DOM!

hello_dom.ipynb

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>' )
```

The screenshot shows a Jupyter Notebook environment with a Python 2 kernel. The top navigation bar includes File, Edit, View, Insert, Cell, Kernel, Help, and a toolbar with various icons. The main area has two code cells.

Cell 3:

```
<h1 class="steely">Hello DOM!</h1>
'''
```

Output 3:

Hello DOM!

Cell 4:

```
HTML('''
<style scoped>
.steely {
    color: steelblue;
    font: 16px script;
}
</style>
<h1 class="steely" id="steely-DOM">Hello DOM!</h1>
<script>$("#steely-DOM").text("Hello JavaScript!!!")</script>
'''')
```

Output 4:

Hello JavaScript!!

To the right, the browser's developer tools are open, specifically the Elements tab. It shows the DOM tree with the following structure:

```
</div>
</div>
<div class="btn btn-default output_collapsed" title="click to expand output" style="display: none;">. . .</div>
</div>
</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"></div>
        <div class="output" style="border: 1px solid #ccc; padding: 5px; border-radius: 5px; background-color: #f9f9f9; width: fit-content; margin: auto; font-family: monospace; font-size: 1em; white-space: pre; word-wrap: break-word; line-height: 1.4; margin-bottom: 10px; position: relative; height: 100px; overflow-y: scroll; user-select: none; -webkit-user-select: none; -ms-user-select: none; -moz-user-select: none; -o-user-select: none; -webkit-overflow-scrolling: touch; -ms-overflow-style: none; ">
            <div class="prompt">Out[4]:</div>
            <div class="output_subarea_output_html rendered_html output_result">
                <style scoped>
                    .steely {
                        color: steelblue;
                        font: 16px script;
                    }
                </style>
                <h1 class="steely" id="steely-DOM">Hello JavaScript!!</h1>
                <script>$("#steely-DOM").text("Hello JavaScript!!!")</script>
            </div>
        </div>
    </div>
</div>
```

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.

2. Hello DOM!

hello_dom.ipynb

Appending elements to the DOM with D3

The screenshot shows a Jupyter Notebook interface with a Python 2 kernel. In the code cell (In [5]), a script tag is included to load d3.js. The output cell (Out[5]) shows the result of the previous cell. In the code cell (In [6]), D3.js code is run to select an element, enter data, and append text. The output cell (Out[6]) shows the resulting text "Hello D3!". Below the notebook, two "Hello D3!" texts are displayed in orange. To the right, a browser's developer tools Elements tab is open, showing the DOM structure. A blue box highlights the newly appended text node under the id="d3-div-1". The bottom navigation bar of the developer tools includes tabs for Styles, Event Listeners, DOM Breakpoints, and Properties.

```
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">...</div>
    ><div class="output" style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">
      ><div class="output_area">
        ><div class="prompt">Out[6]:</div>
      ><div class="output_subarea output_html rendered_html output_result">
        ><pre><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 class="btn btn-default output_collapsed" title="click to expand output" style="display: none;">...</div>
</pre>
    ... 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

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!

```
Elements Network > x 1 ⚡ 1 > ⚙️ ... X  
title="Click to expand output" style="...  
    "display: none;". . .</div>  
    </div>  
  </div>  
  ▶<div class="cell text_cell rendered unselected" 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>  
      ▷<div class="output_area">  
        <div class="prompt output_prompt">Out[8]:</div>  
        ▷<div class="output_subarea output_html rendered_html output_result">  
          <style scoped> .bedazzled {color: orange;} </style>  
          ▷<div id="d3-div-2">  
            <p class="bedazzled" style="font-size: 15px;">Hello D3 with Python data!</p>  
            <p class="bedazzled" style="font-size: 30px;">Hello D3 with Python data!</p>  
            <p class="bedazzled" style="font-size: 45px;">Hello D3 with Python data!</p>  
          </div>  
          ▷<script>...</script>  
        </div>  
      </div>  
    </div>  
  </div>  
  ... div div div div div #d3-div-2 p.bedazzled  
  Styles Event Listeners DOM Breakpoints Properties  
Console Search Emulation Rendering
```

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 <div> 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 <div> 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...

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

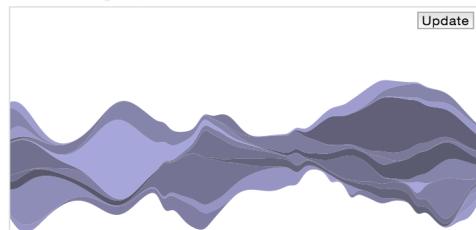
3. Meet blocks.org, I think you'll be great friends

d3js.org , b1.ocks.org/mbostock

(And welcome to data visualization heaven)

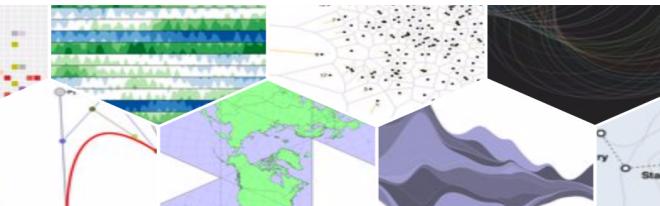


<http://b1.ocks.org/mbostock/4060954>
Streamgraph

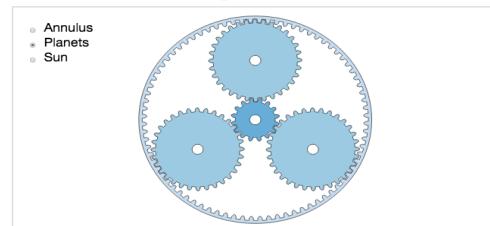


For continuous data such as time series, a streamgraph can be used in place of stacked bars. This example also demonstrates path transitions to interpolate between different layouts. Streamgraph algorithm, colors, and data generation inspired by [Byron and Wattenberg](#).

[Open in a new window.](#)



<http://b1.ocks.org/mbostock/1353700>
Epicyclic Gearing

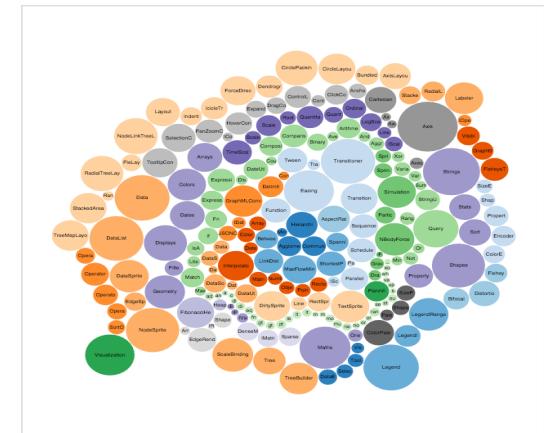


Epicyclic gearing or **planetary gearing** is a gear system consisting of one or more outer gears, or *planet* gears, revolving about a central, or *sun* gear. ... Epicyclic gearing systems also incorporate the use of an outer ring gear or *annulus*, which meshes with the planet gears.

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



<http://b1.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 hierarchy, also courtesy Jeff Heer.

[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-serif;
  margin: auto;
  position: relative;
  width: 960px;
}
```

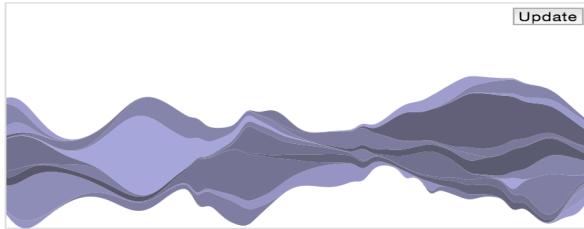
#index.html

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

3. Meet blocks.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](#).

```
#index.html

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

body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  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 block

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

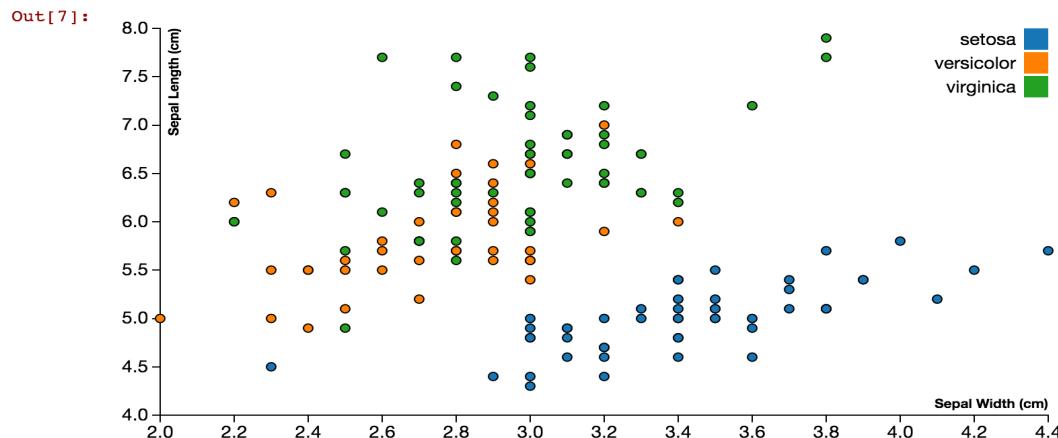
4. A simple D3 scatterplot in Jupyter

iris_scatterplot.ipynb

Using this blocks 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}))
```



5. A bar chart with interactivity

bar_chart_with_update.ipynb

(hello again DOM!)

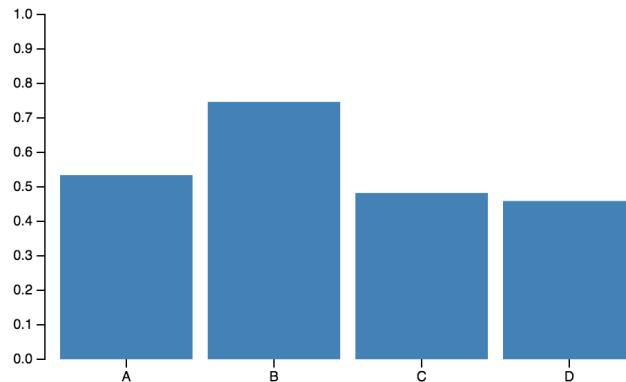
```
In [7]: data = pd.DataFrame({'letter': ['A','B','C','D'], 'y': [1,1,1,1]})  
data.head()
```

```
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()
```

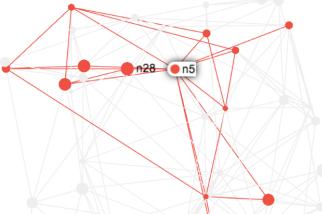
```
Out[5]:
```

	<code>id</code>	<code>source</code>	<code>target</code>
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}))
```

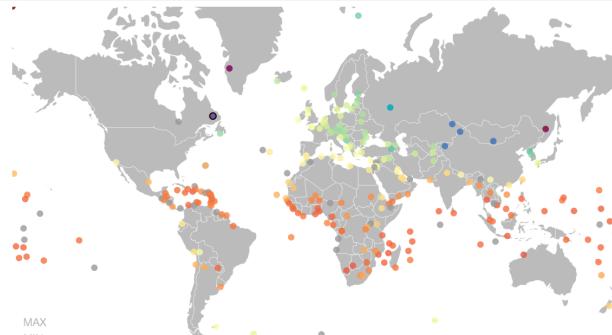
```
Out[6]:
```



`temperature_histories.ipynb`

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

`3d_meshing.ipynb`



MAX
MIN
AVERAGE TEMPERATURE
1945 animate ► 2010

JAN 1990

100F
80
60
40
20
0

`polyFill_d3.ipynb`

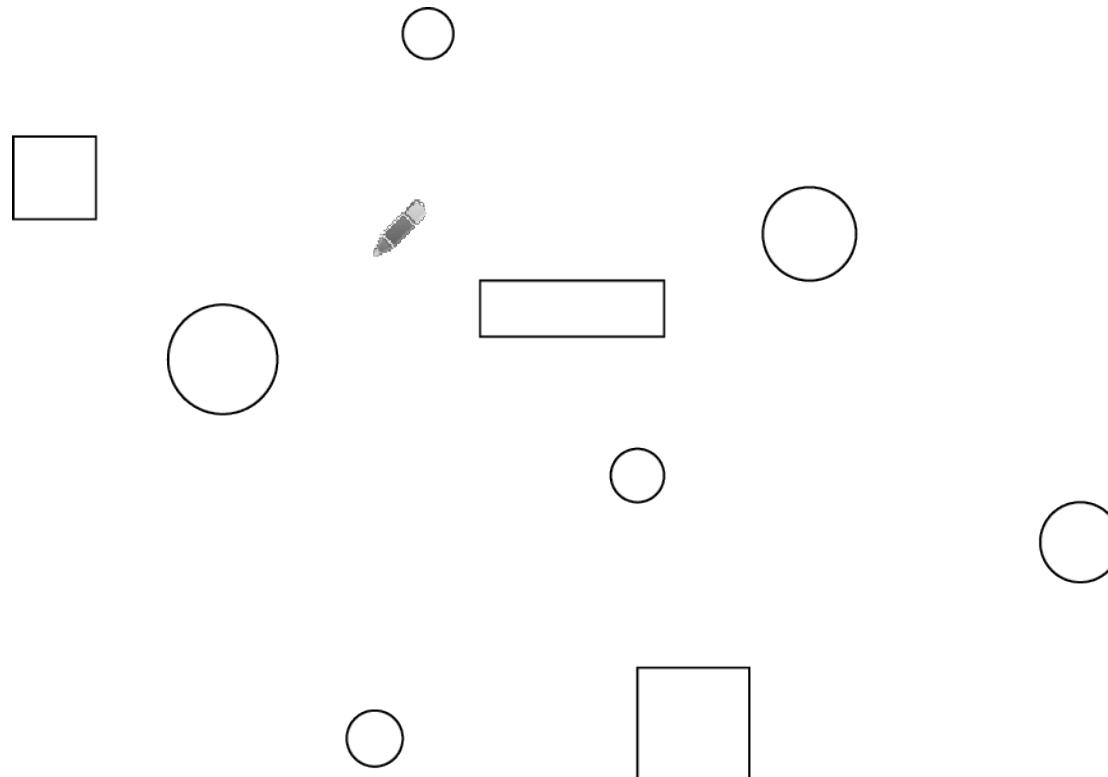
We have encapsulated a lot of the boilerplate for this method into an external python file, which in turn calls external files containing your css and js templates.

This just helps to keep things visually clean.

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

8. Discussion: From sketch to notebook frame

randomized_sketch.ipynb



Thank you!

bcoffey@stitchfix.com

Data Science @ Stitch Fix
multithreaded.stitchfix.com