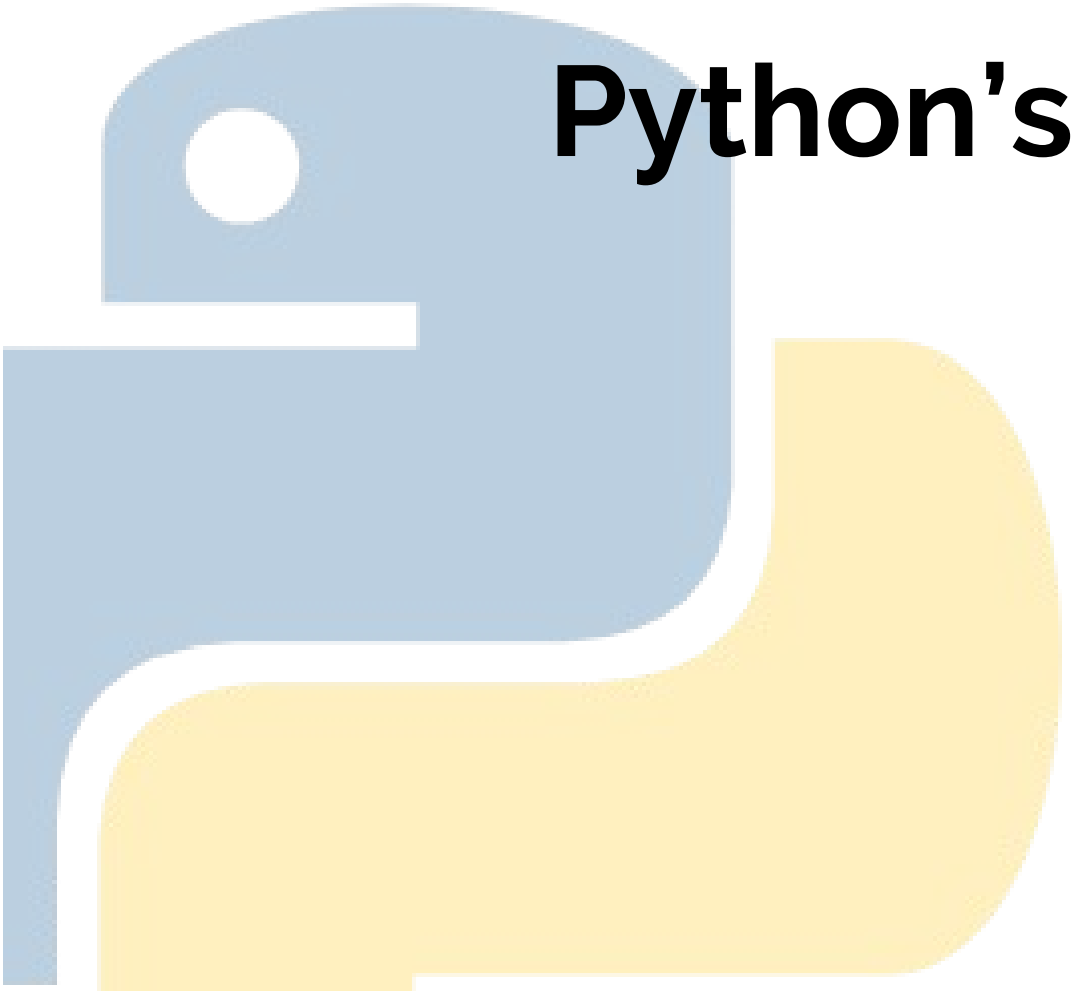
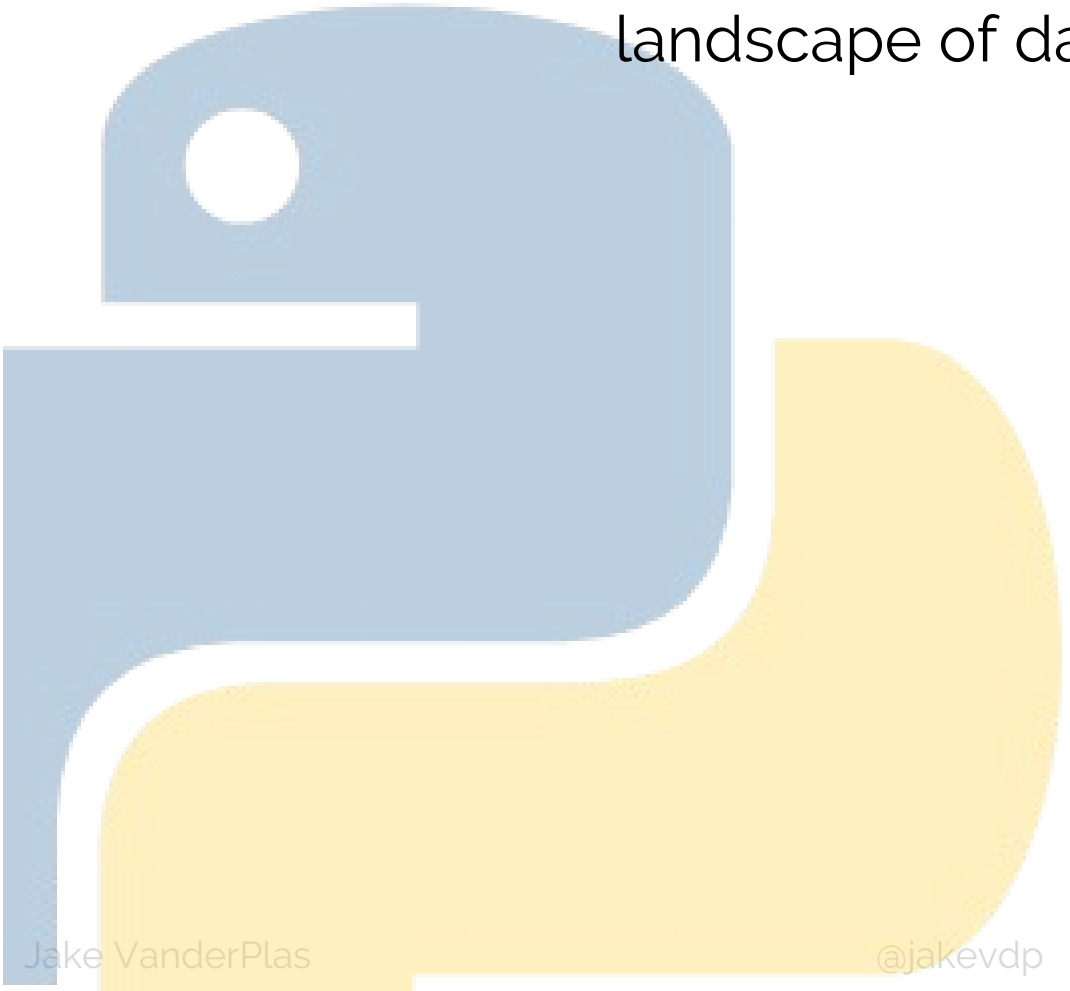


Python's Visualization Landscape

Jake VanderPlas
@jakevdp #PyCon2017

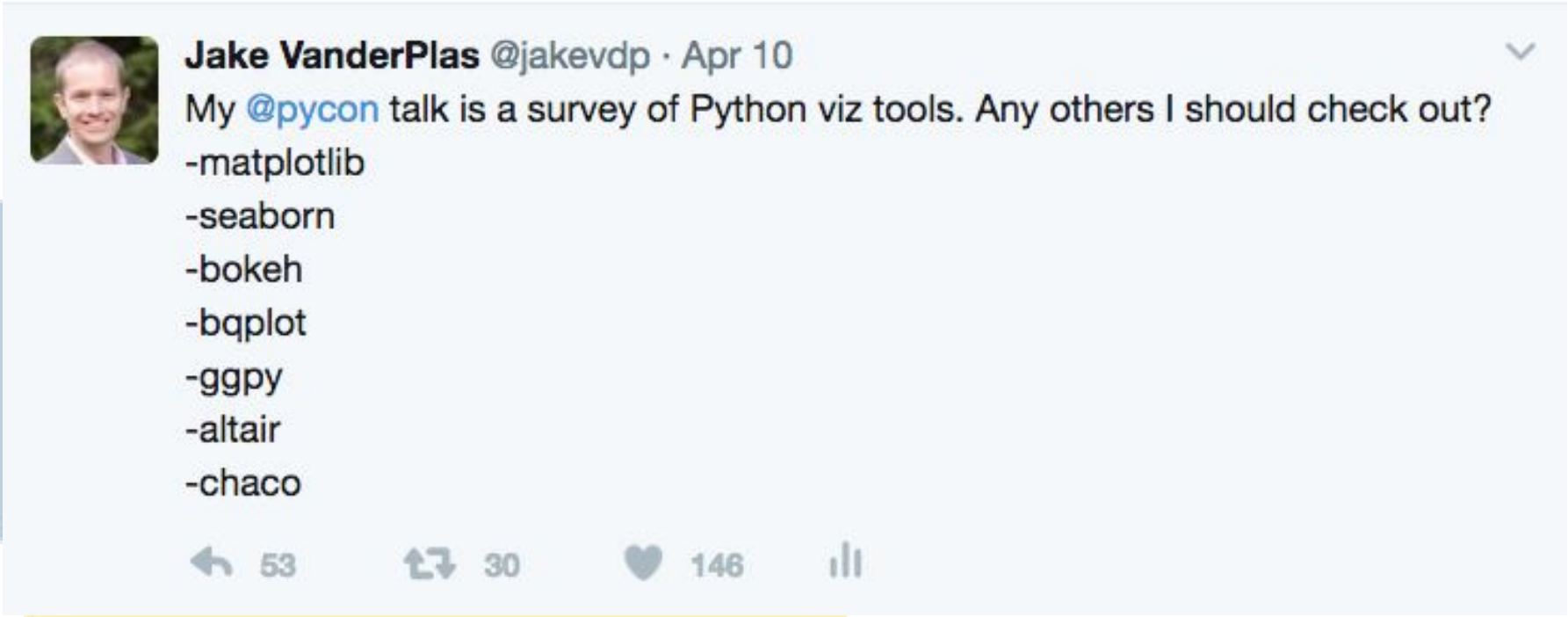
[Python's Visualization Landscape]

From the abstract:
"In this talk I'll give an overview of the landscape of dataviz tools in Python . . ."



[Python's Visualization Landscape]

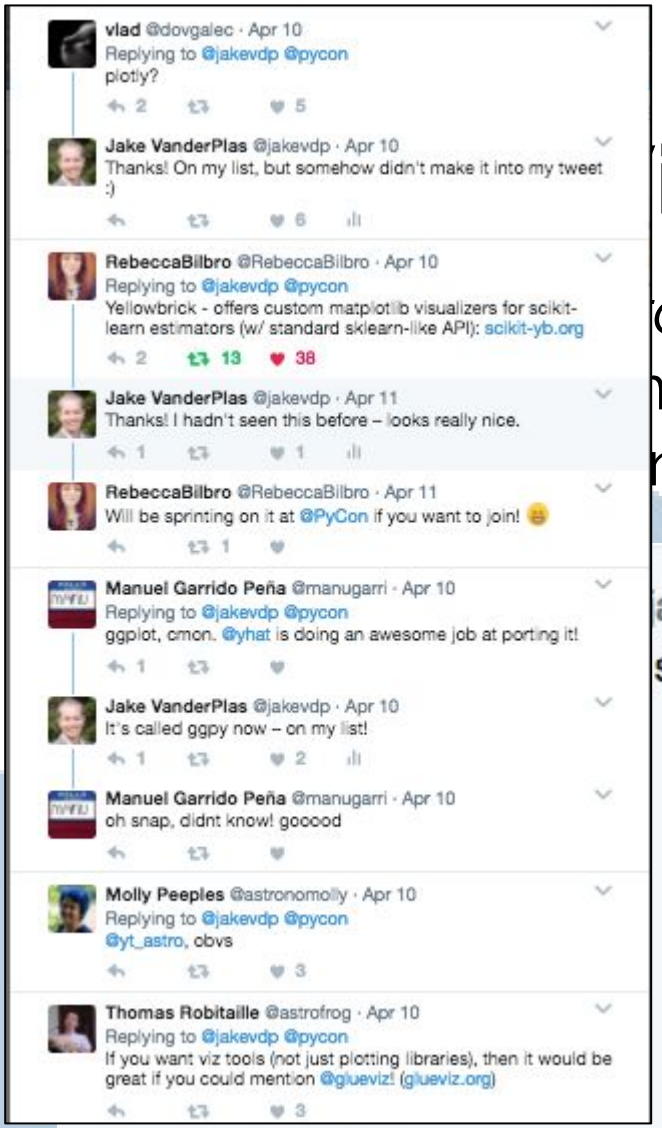
From the abstract:
"In this talk I'll give an overview of the landscape of dataviz tools in Python . . ."

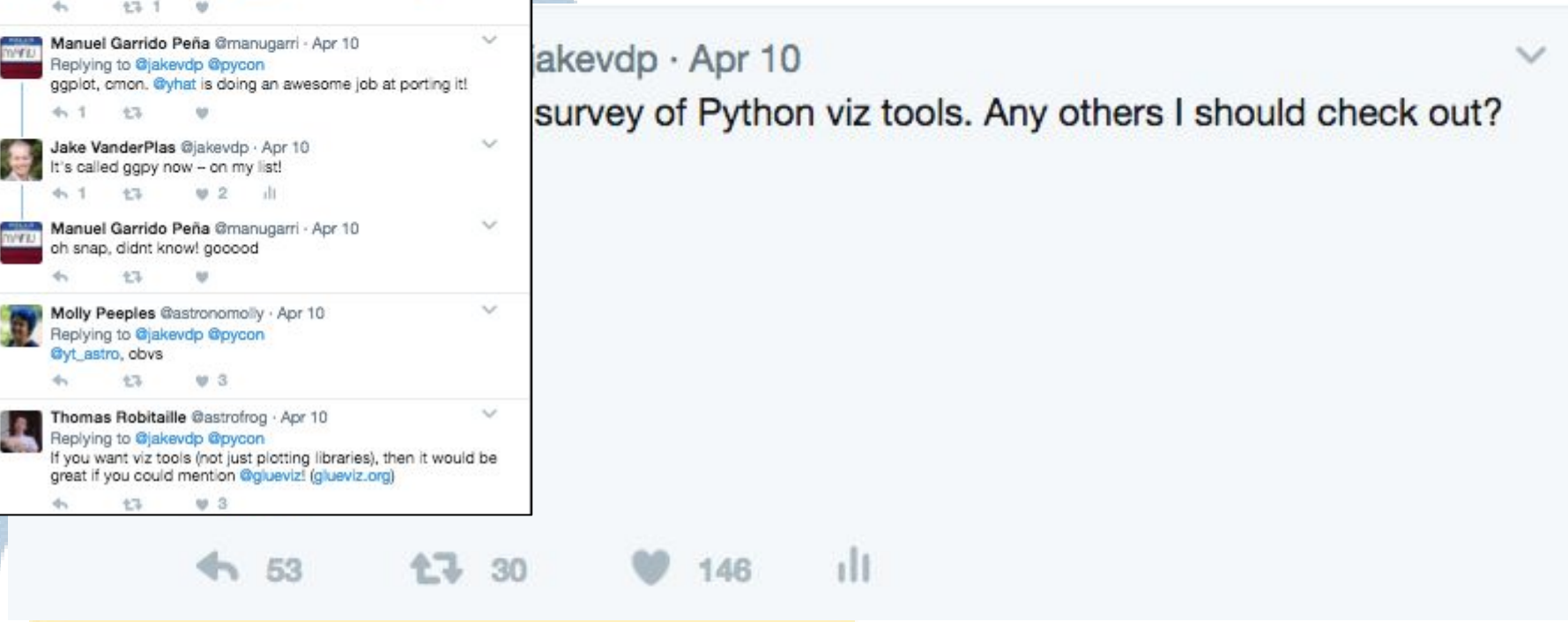


isualization Landscape]

om the abstract:

n this talk I'll give an overview of the
ndscape of dataviz tools in Python . . ."



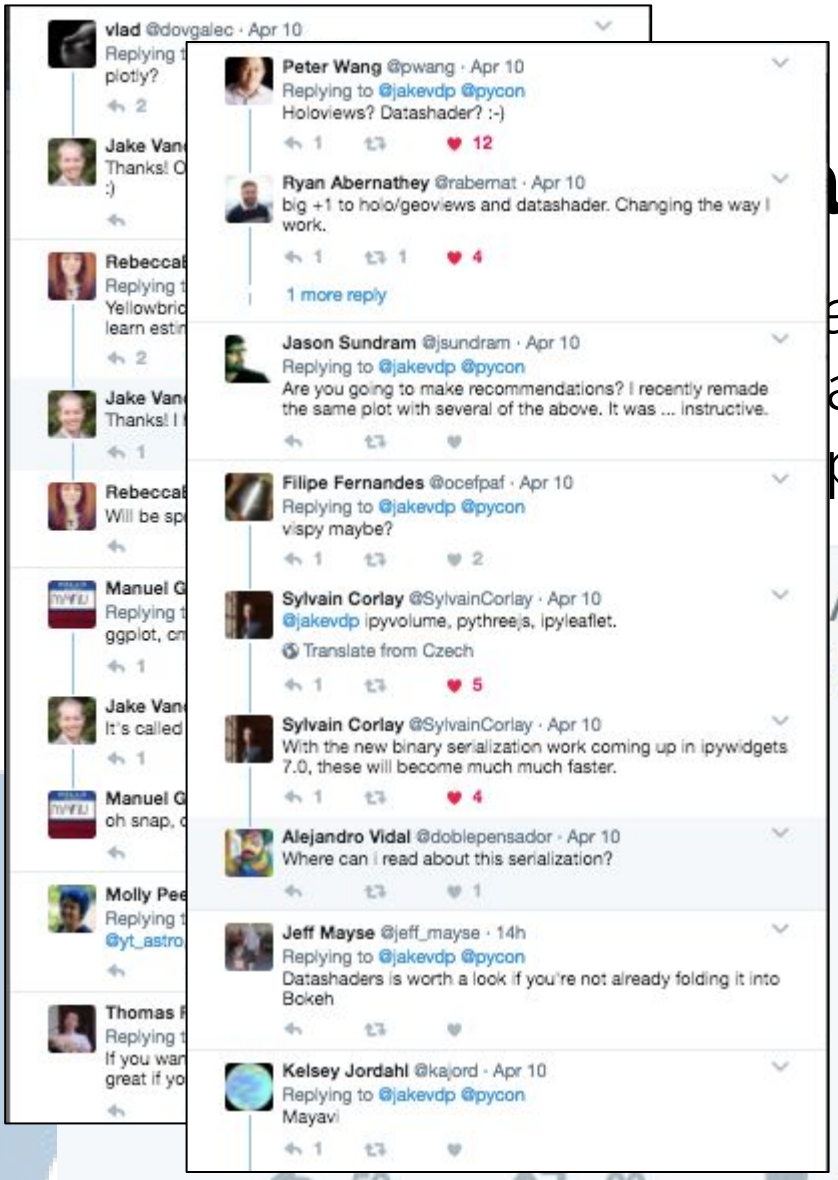


lization Landscape]

e abstract:

alk I'll give an overview of the

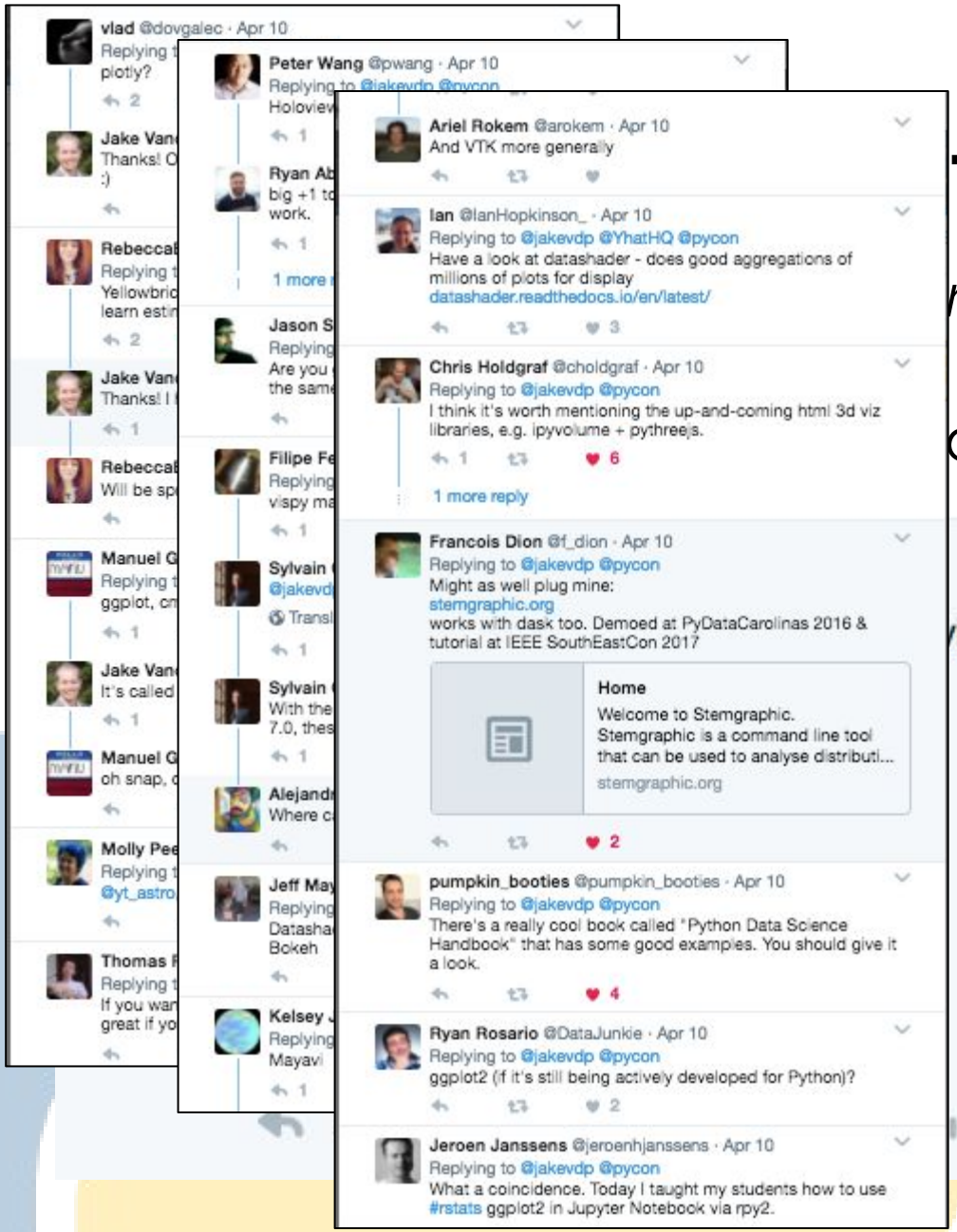
pe of dataviz tools in Python . . ."



tion Landscape]

ract:

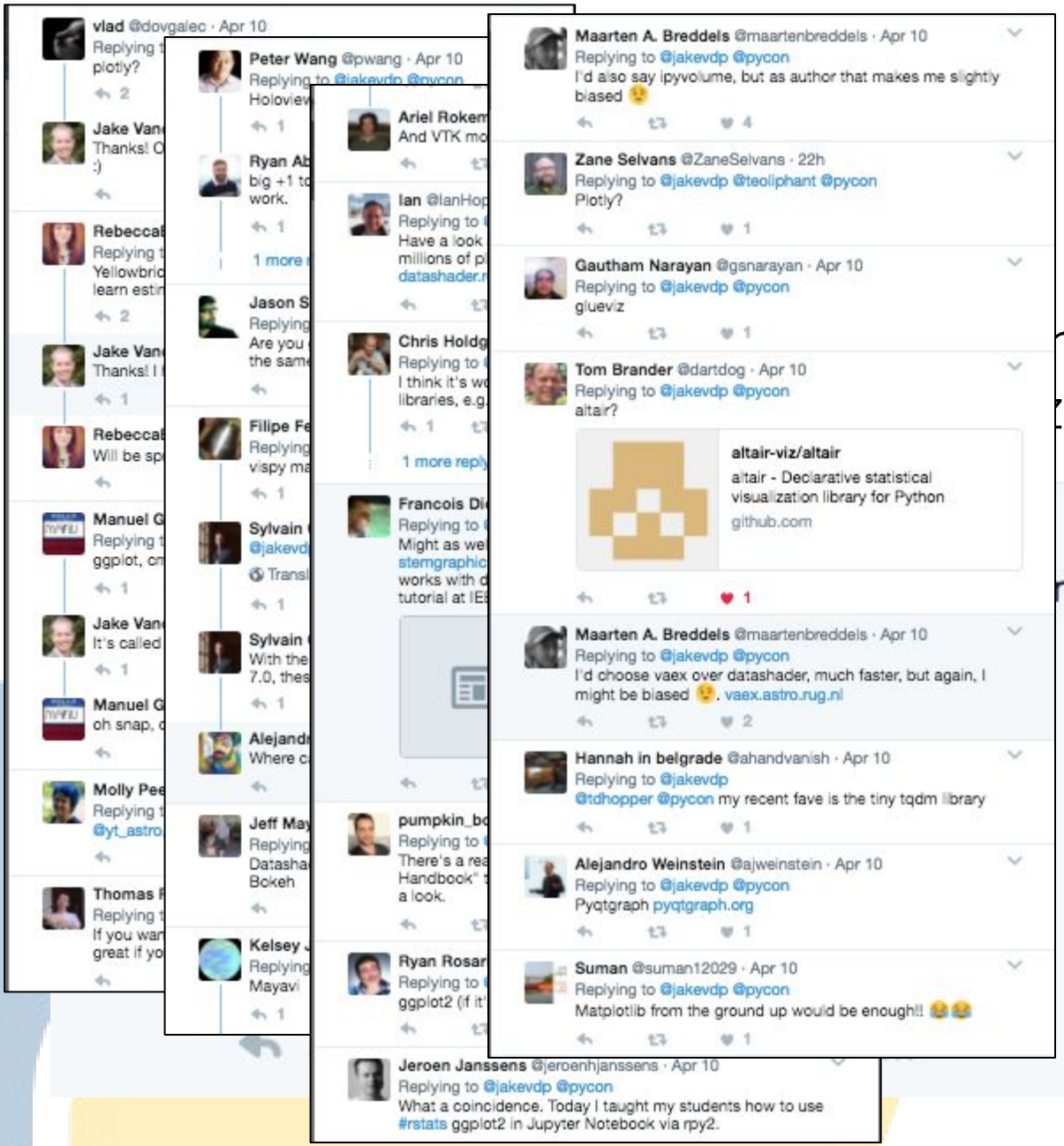
give an overview of the
dataviz tools in Python . . ."



Landscape]

n overview of the
z tools in Python . . ."

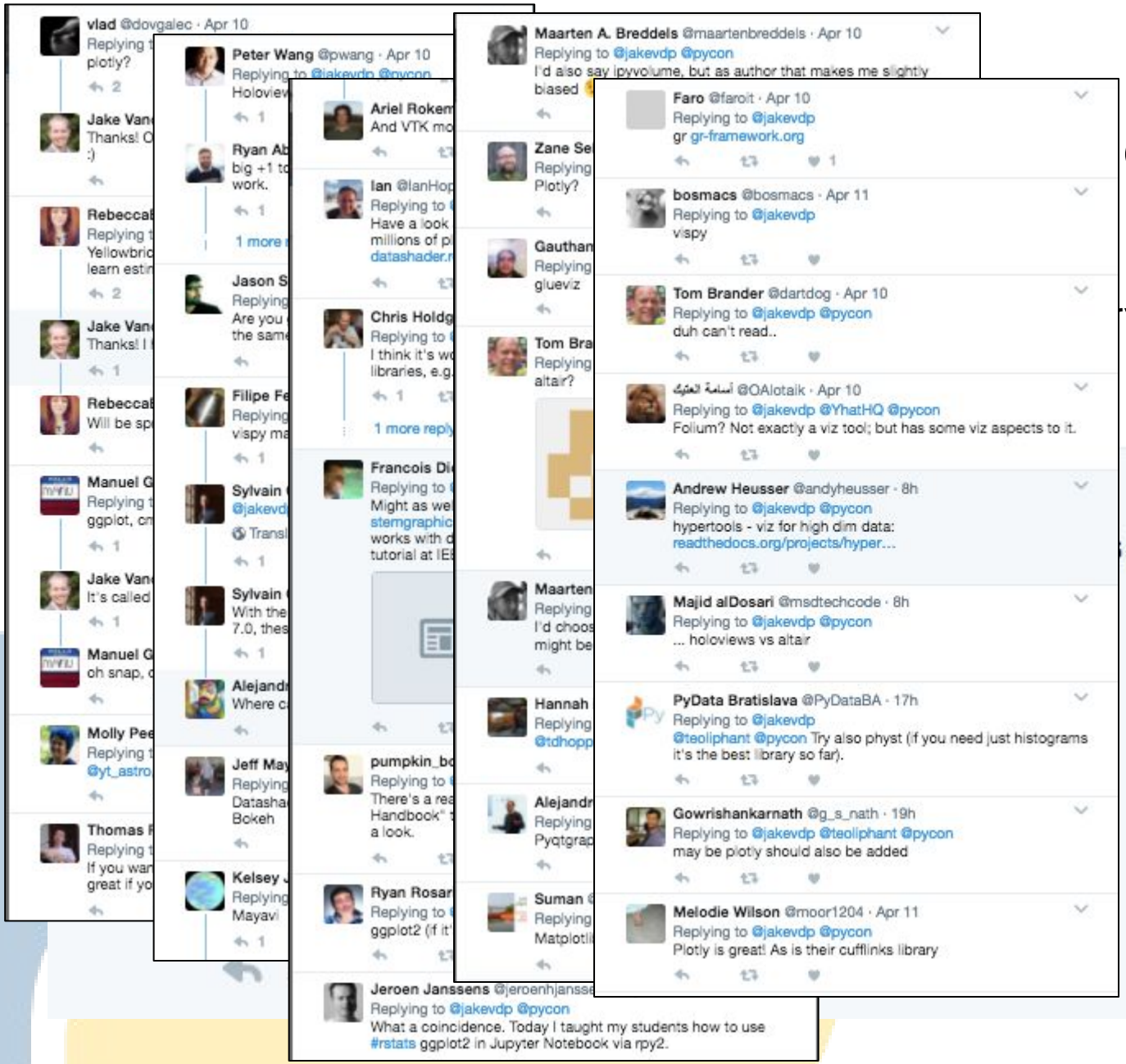
ny others I should check out?

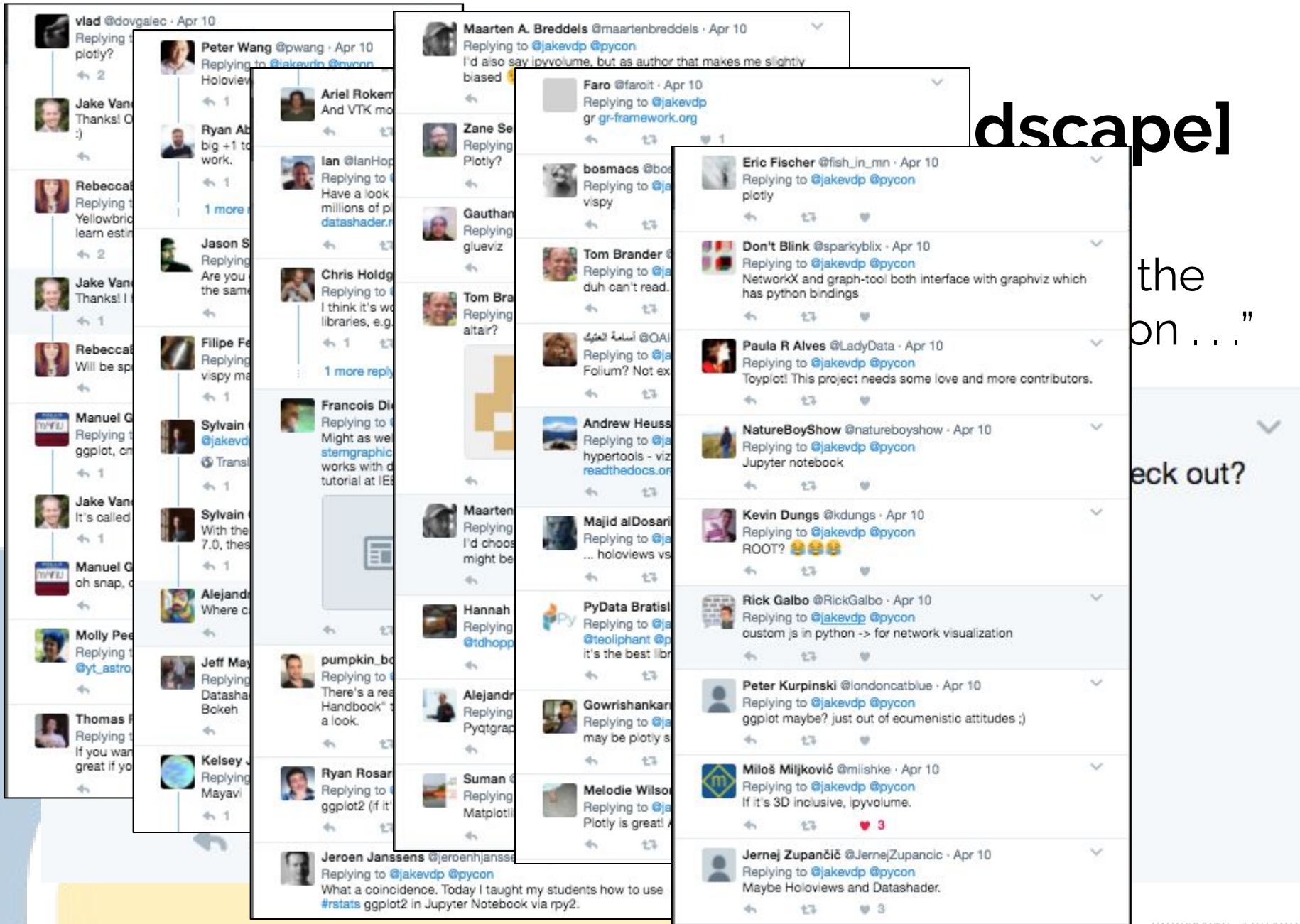


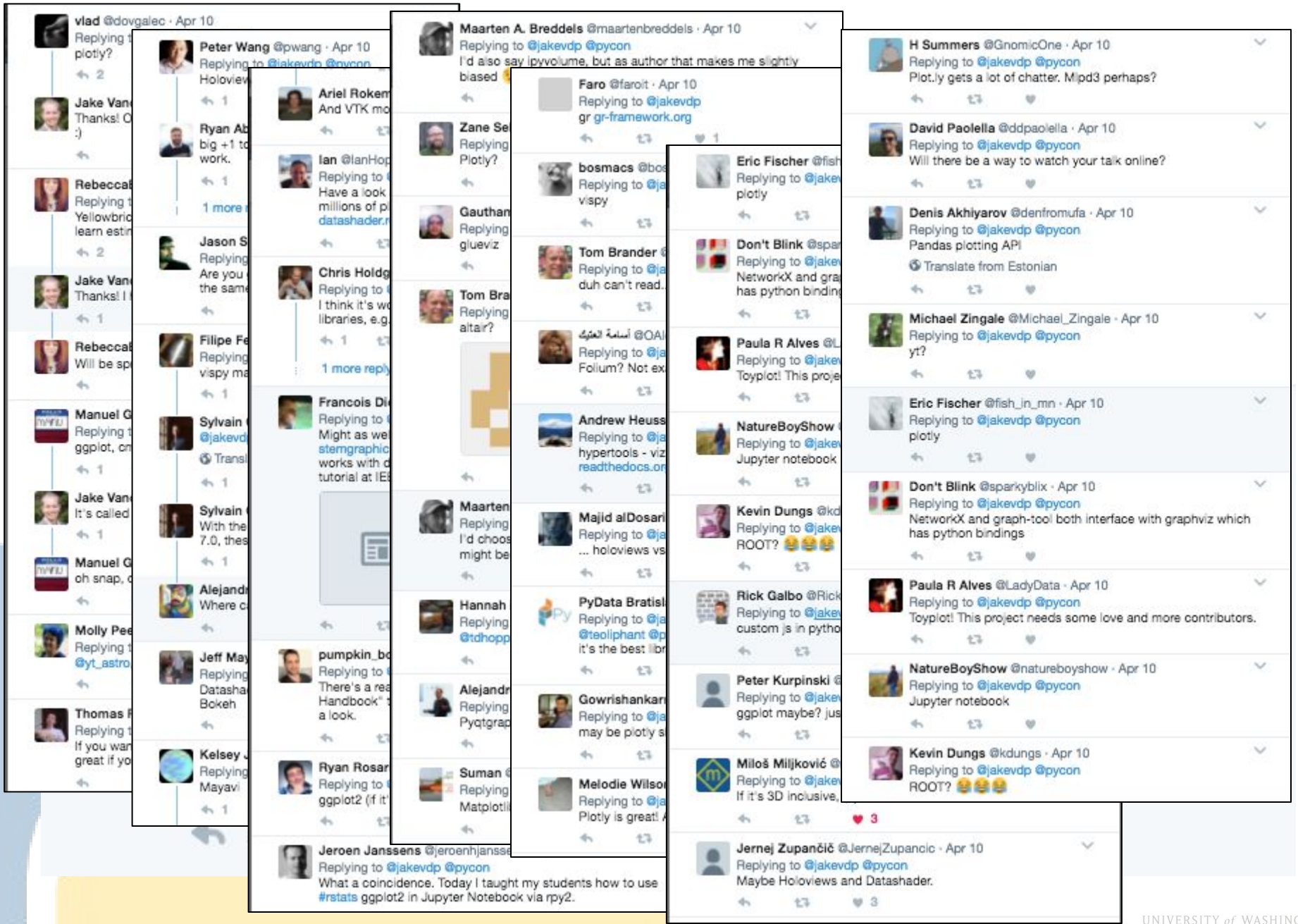
dscapel

rview of the
in Python . . ."

I should check out?

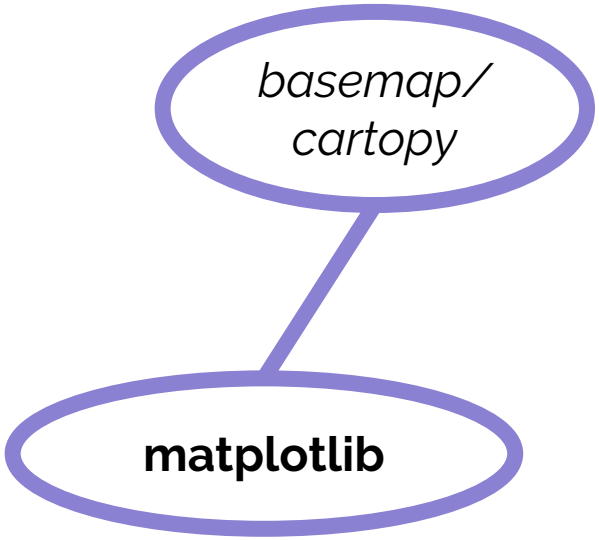


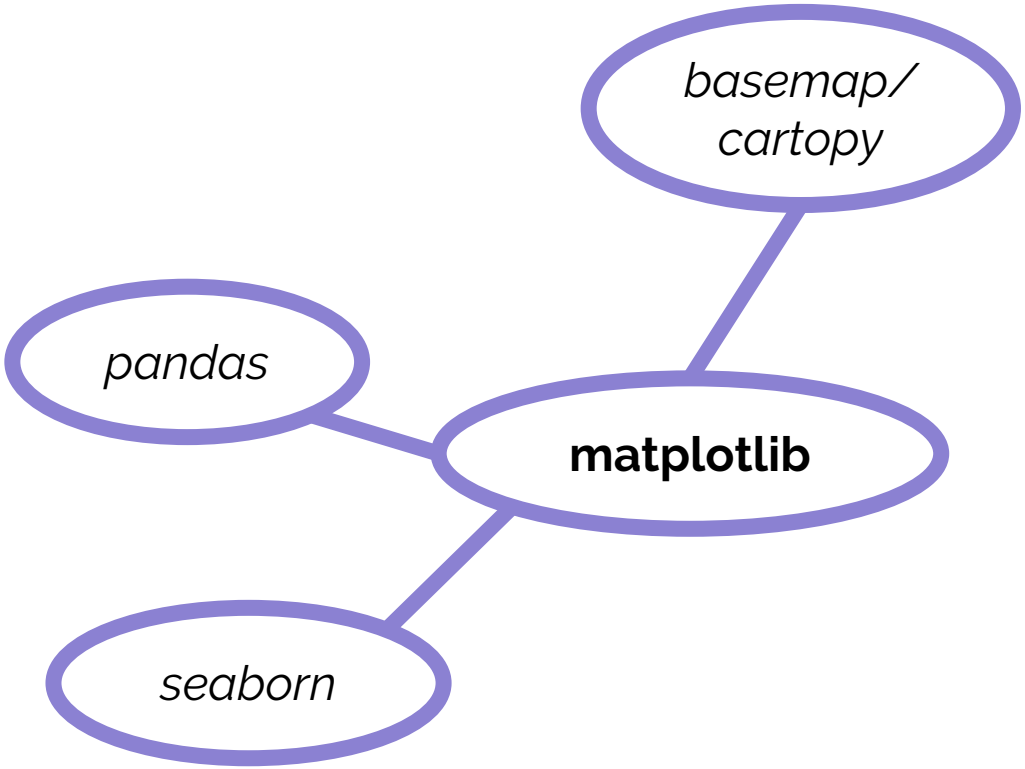


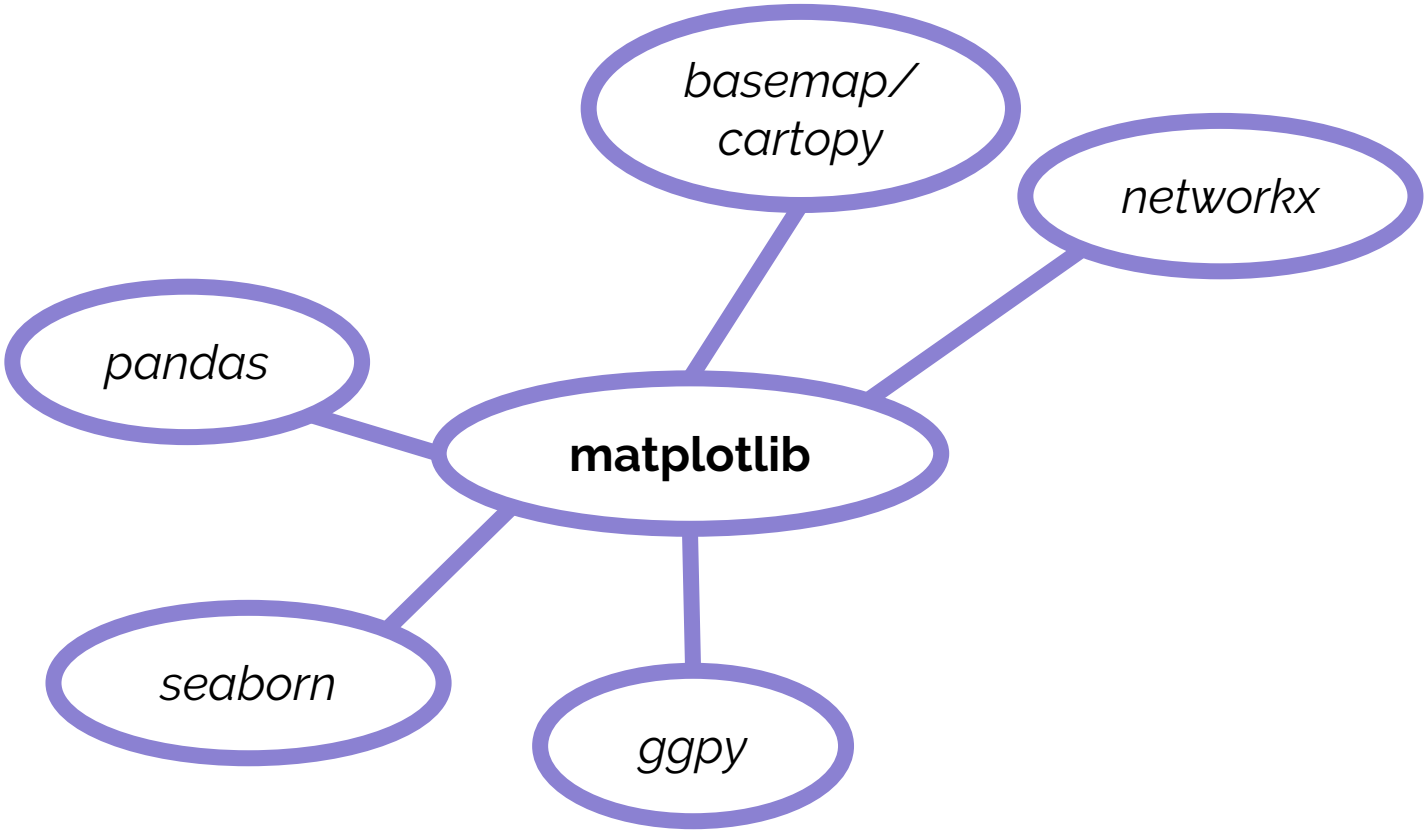


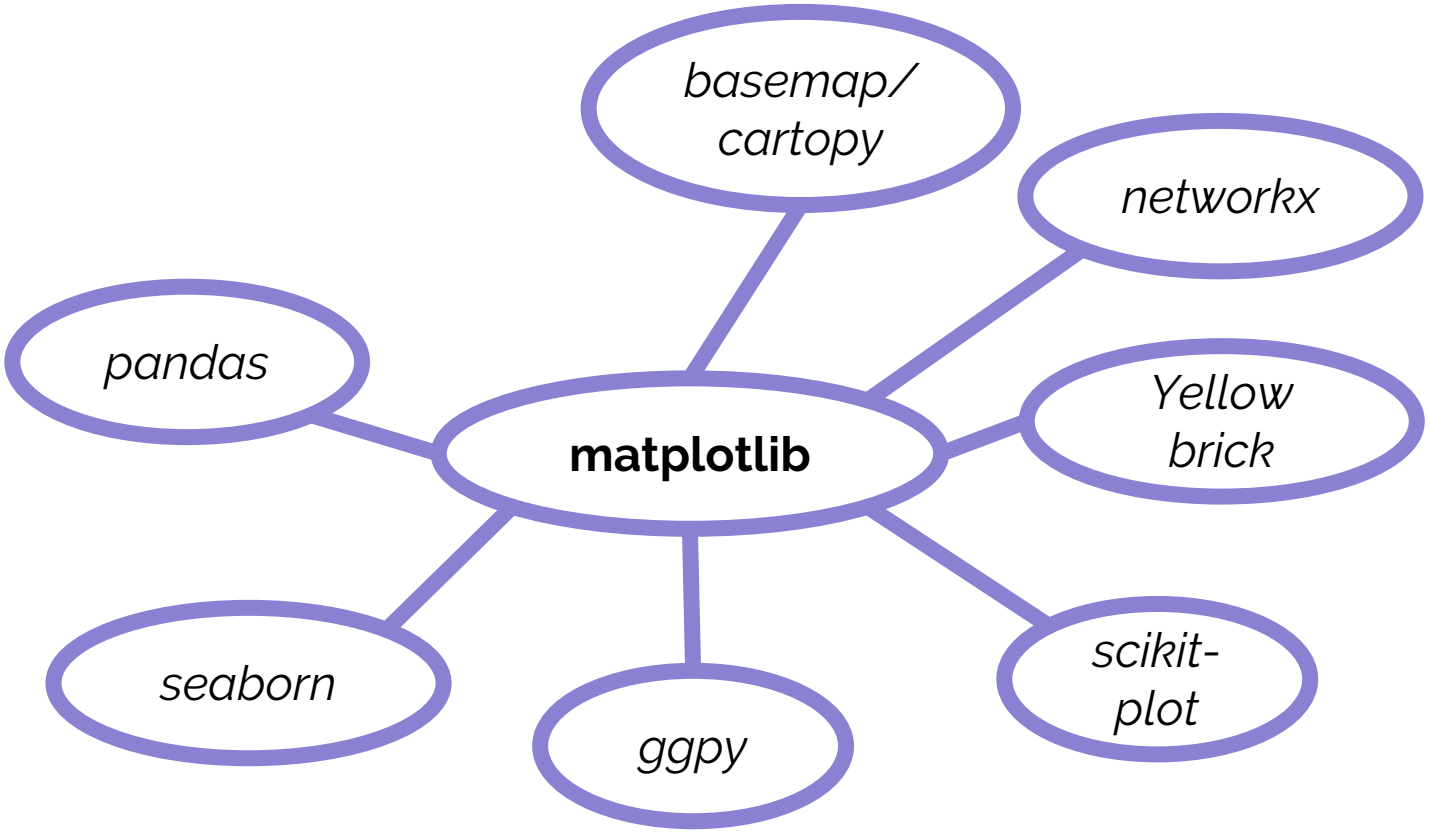
[Making Sense of the Deluge]

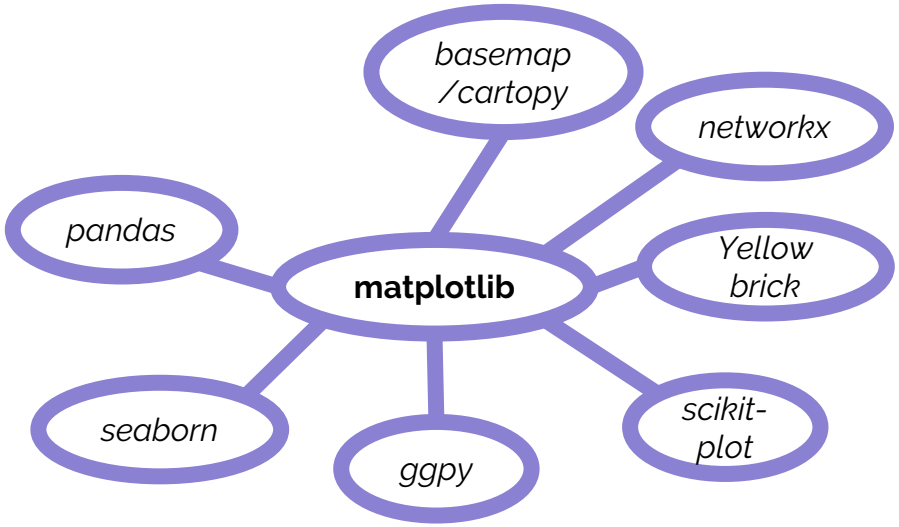

matplotlib

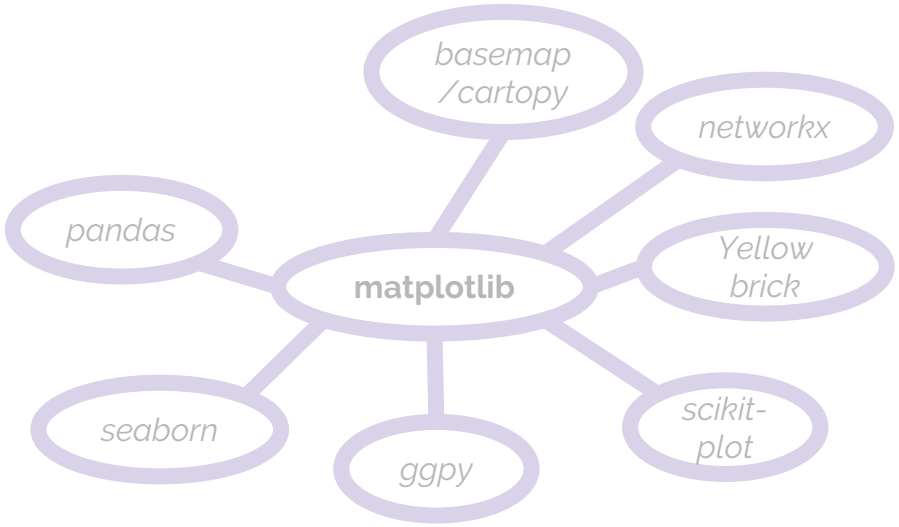

basemap/
cartopy
matplotlib

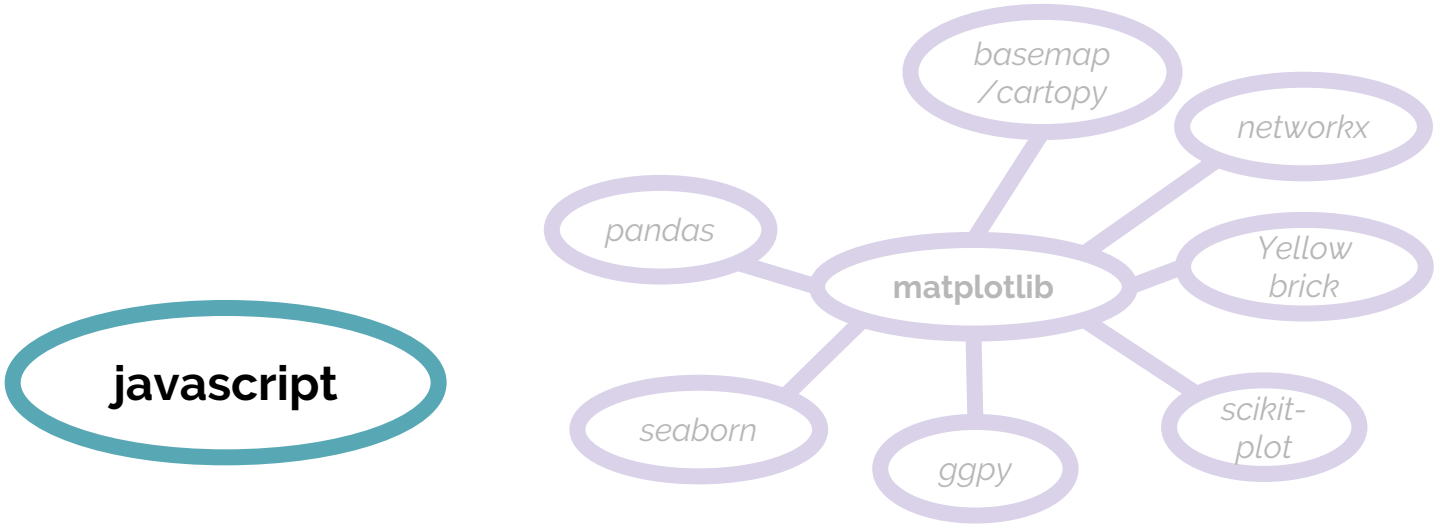

basemap/
cartopy
pandas
matplotlib
seaborn

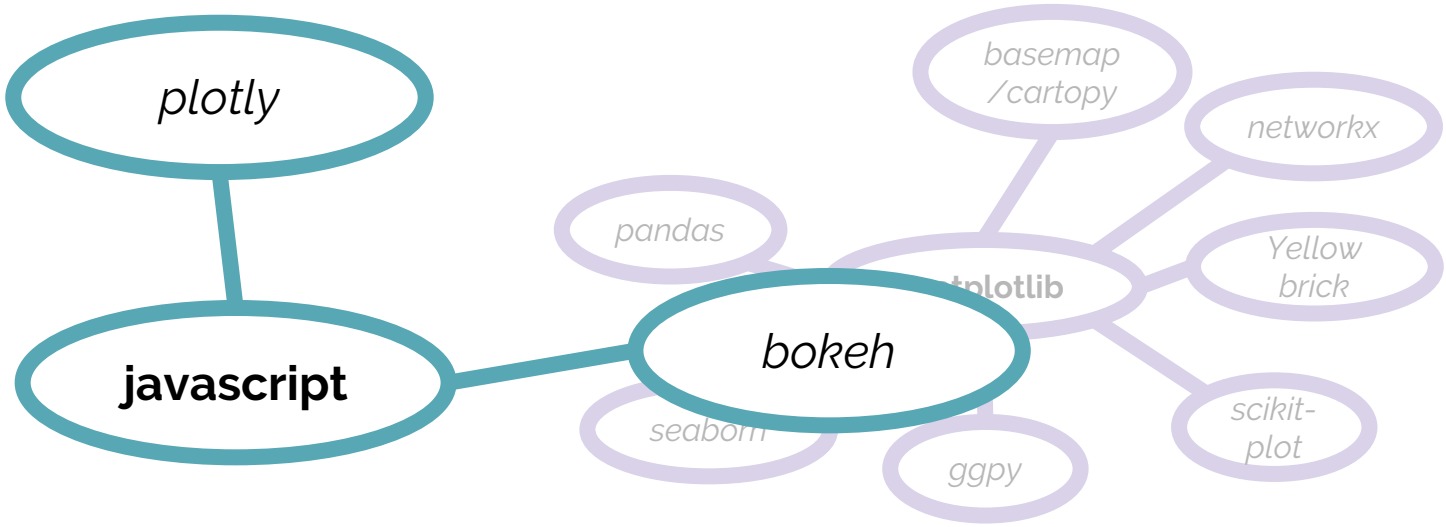

basemap/
cartopy
networkx
pandas
matplotlib
seaborn
ggpy

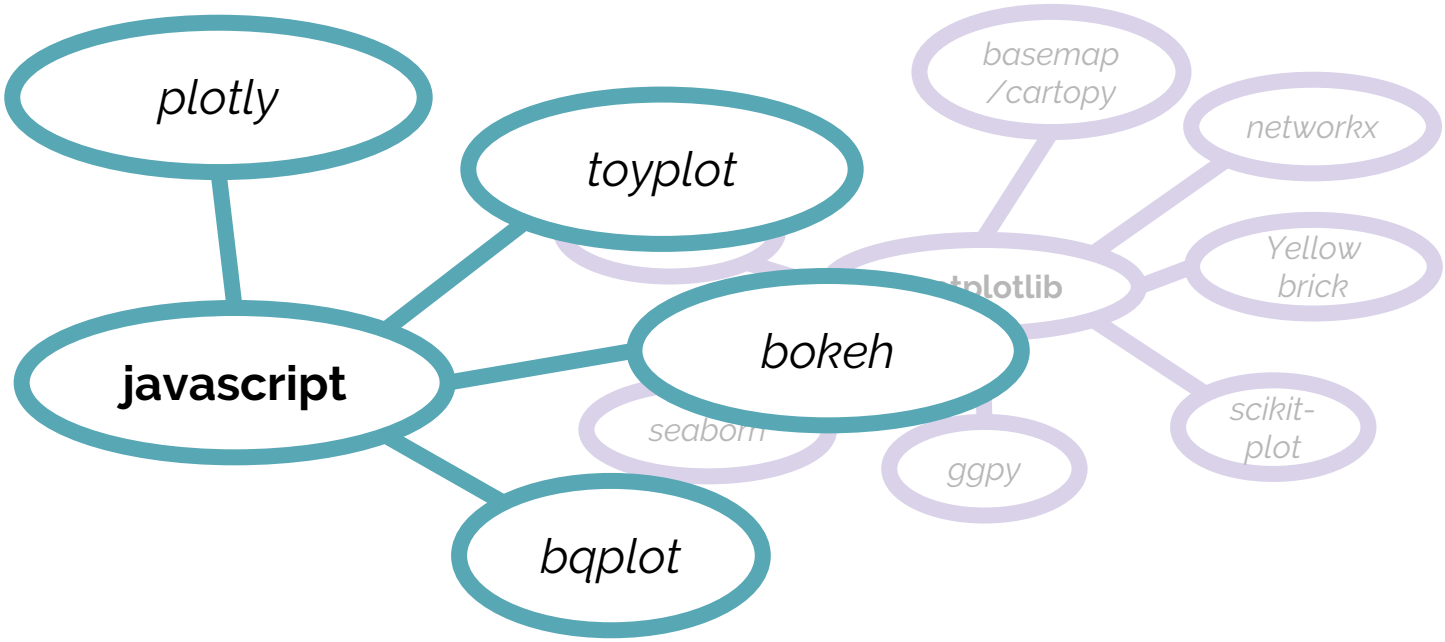

basemap/
cartopy
networkx
pandas
matplotlib
Yellow
brick
seaborn
ggpy
scikit-
plot

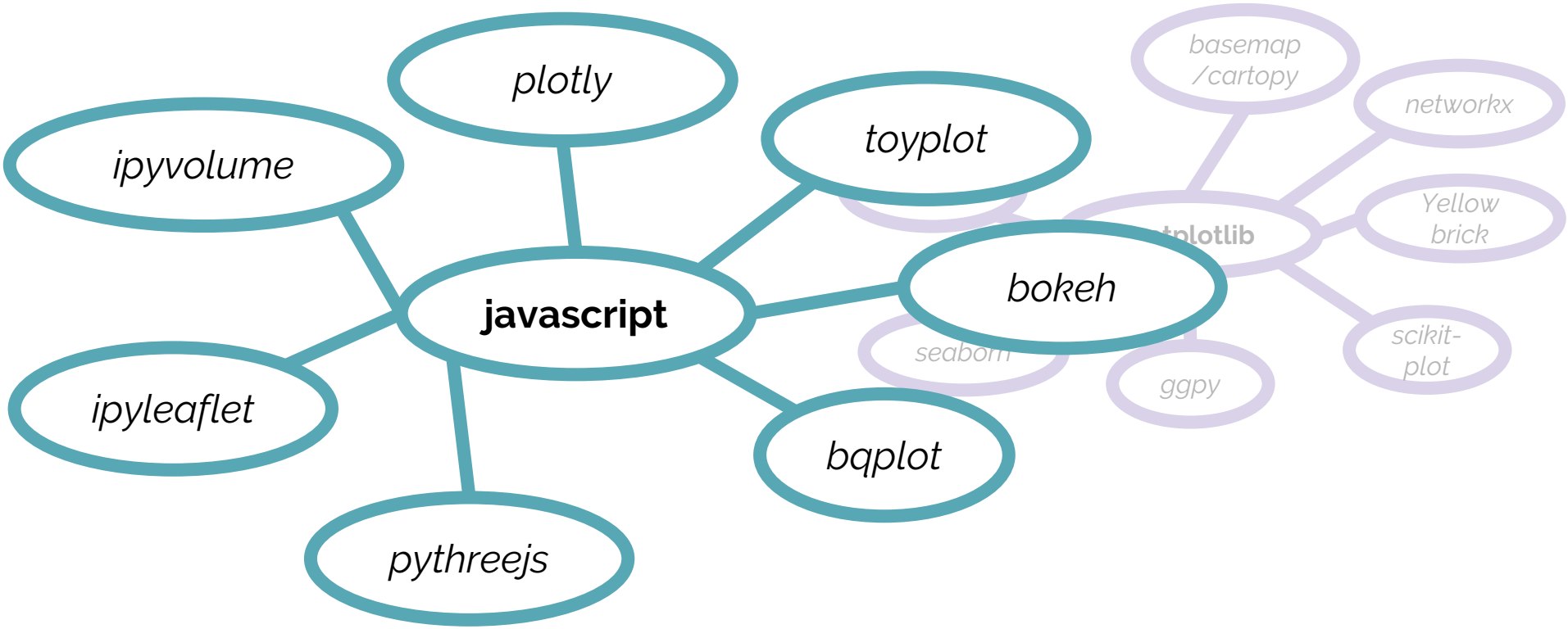

basemap
/cartopy
networkx
pandas
matplotlib
Yellow
brick
seaborn
ggpy
scikit-
plot

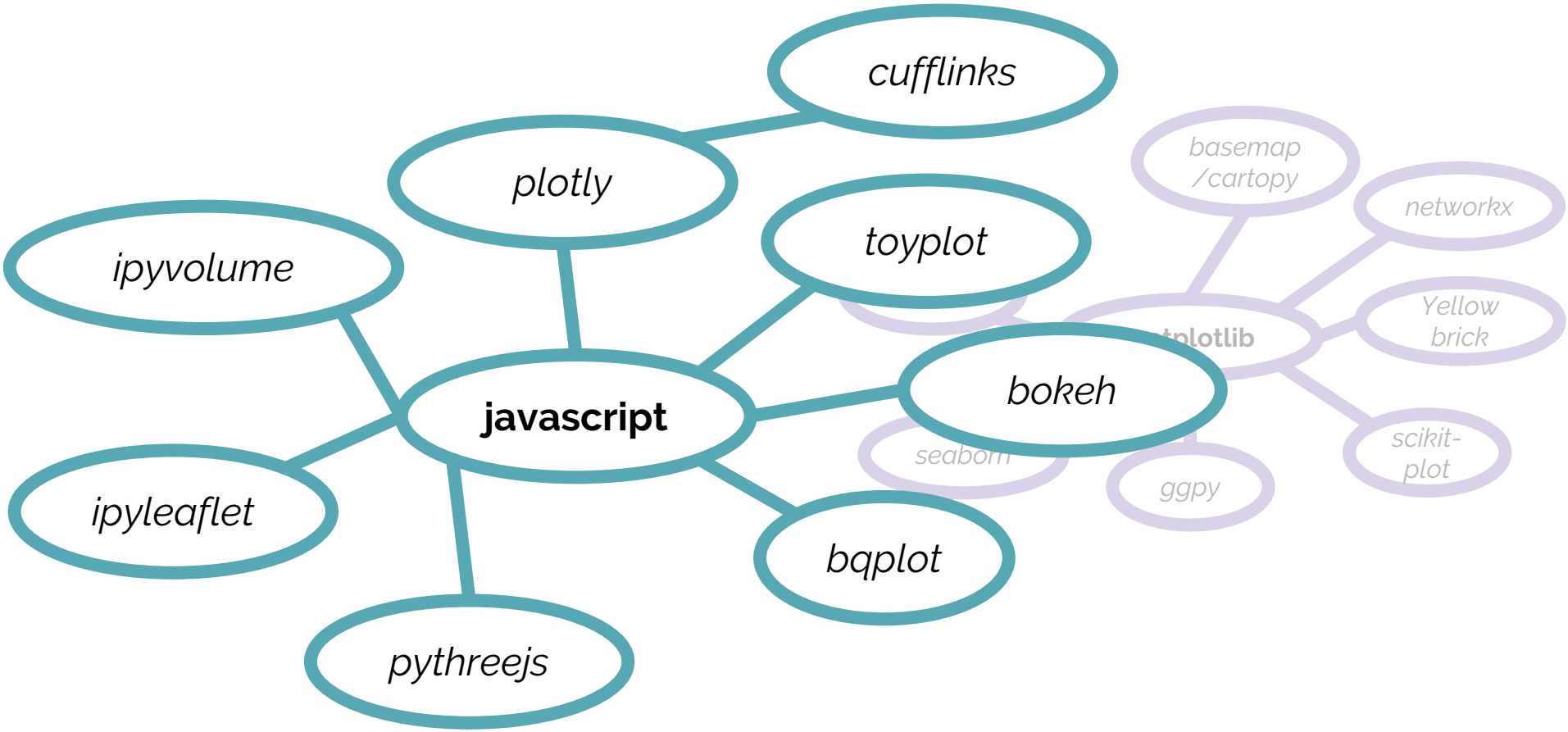

basemap
/cartopy
networkx
pandas
matplotlib
Yellow
brick
seaborn
ggpy
scikit-
plot

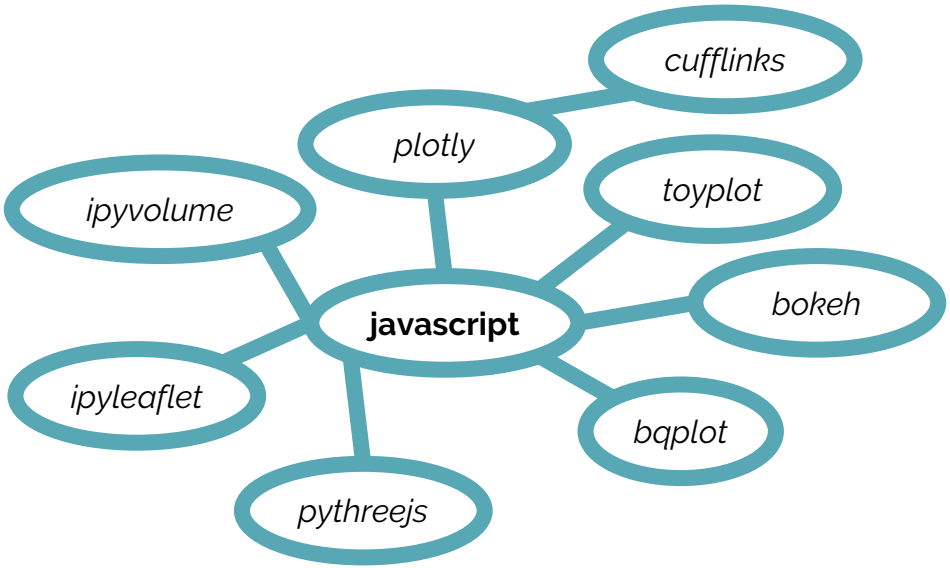

javascript
basemap
/cartopy
networkx
pandas
matplotlib
Yellow
brick
seaborn
ggpy
scikit-
plot

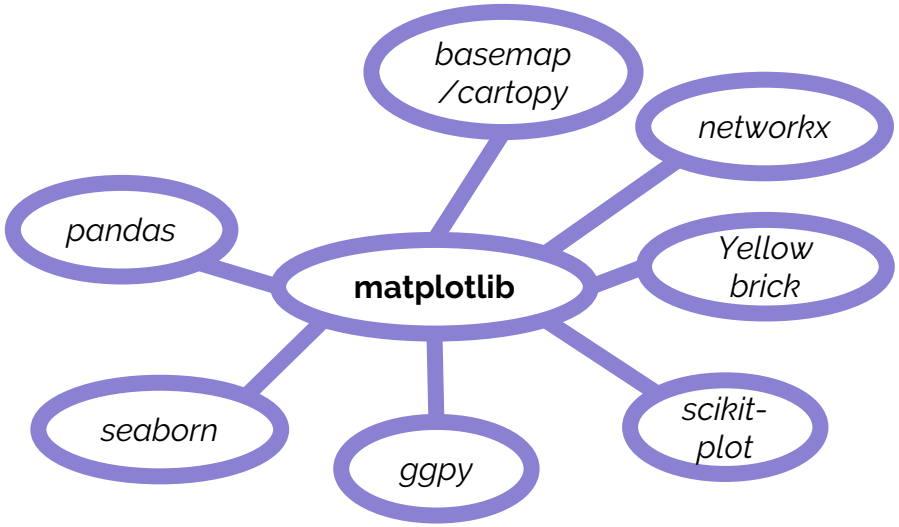

plotly
javascript
bokeh
basemap
/cartopy
networkx
pandas
Yellow
brick
tplotlib
scikit-
plot
seaborn
ggpy

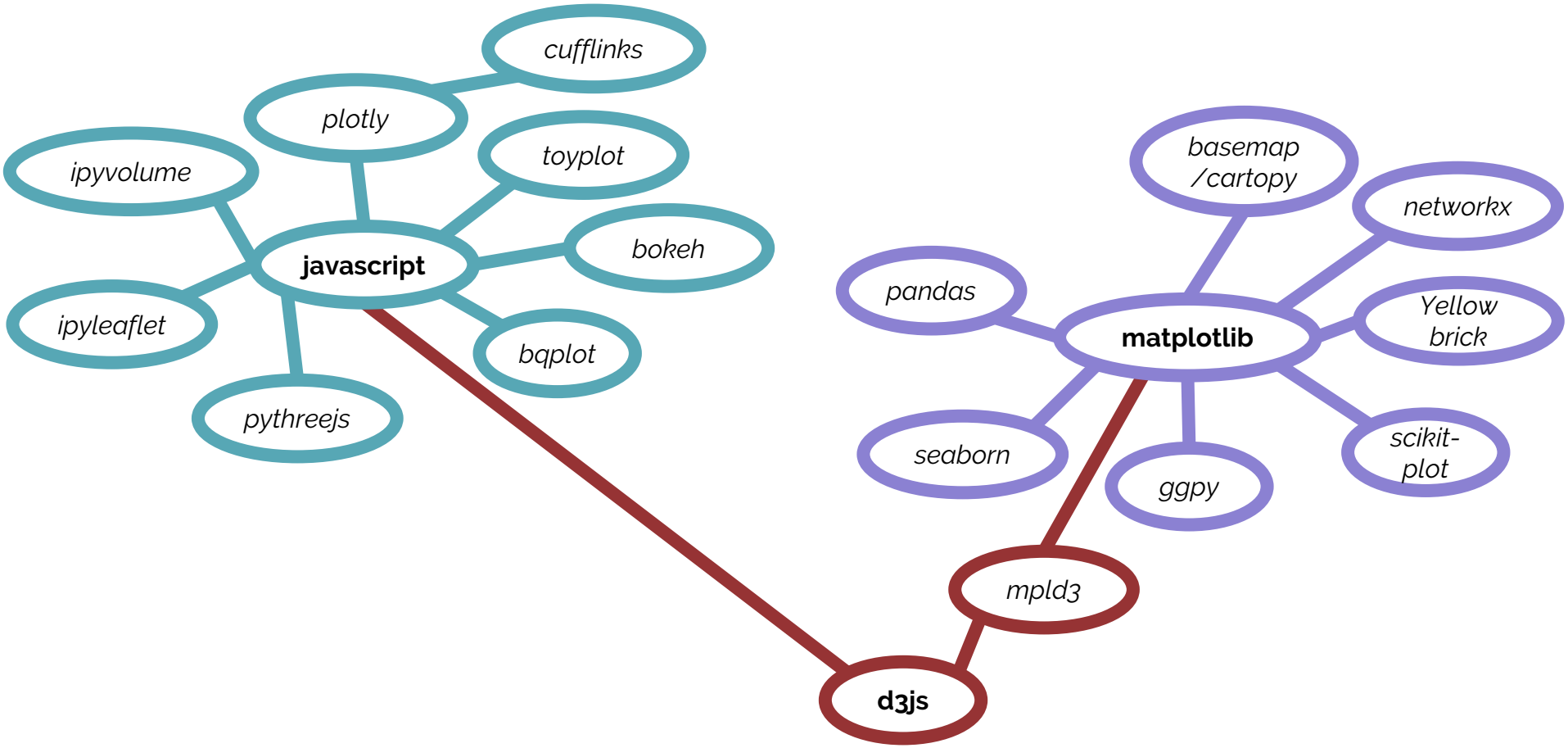

plotly
toyplot
javascript
bokeh
bqplot
basemap
/cartopy
networkx
Yellow
brick
tplotlib
scikit-
plot
seaborn
ggpy

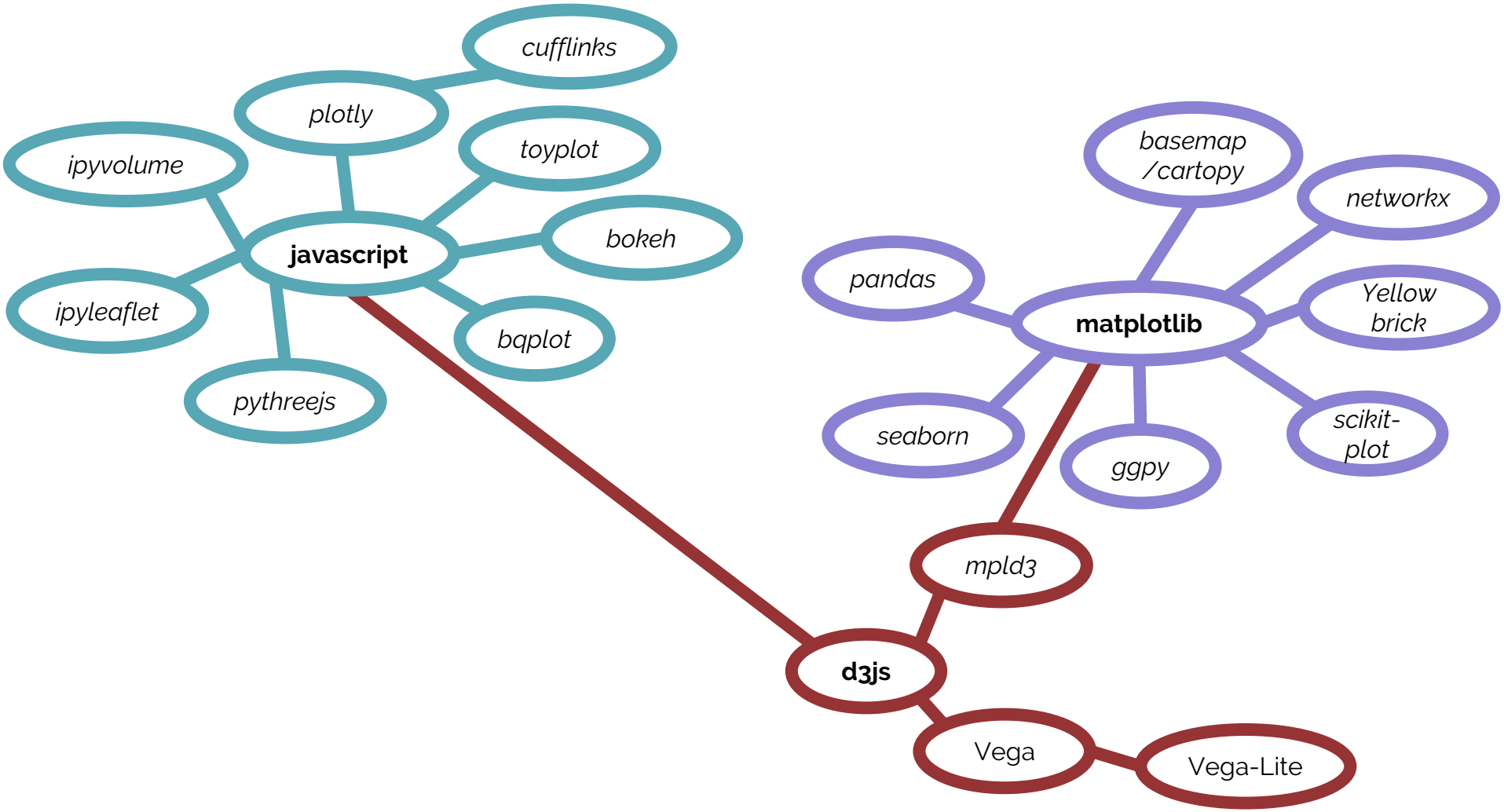

plotly
ipyvolume
toyplot
basemap
/cartopy
networkx
Yellow
brick
tplotlib
bokeh
javascript
seaborn
ggpy
scikit-
plot
ipyleaflet
bqplot
pythreejs

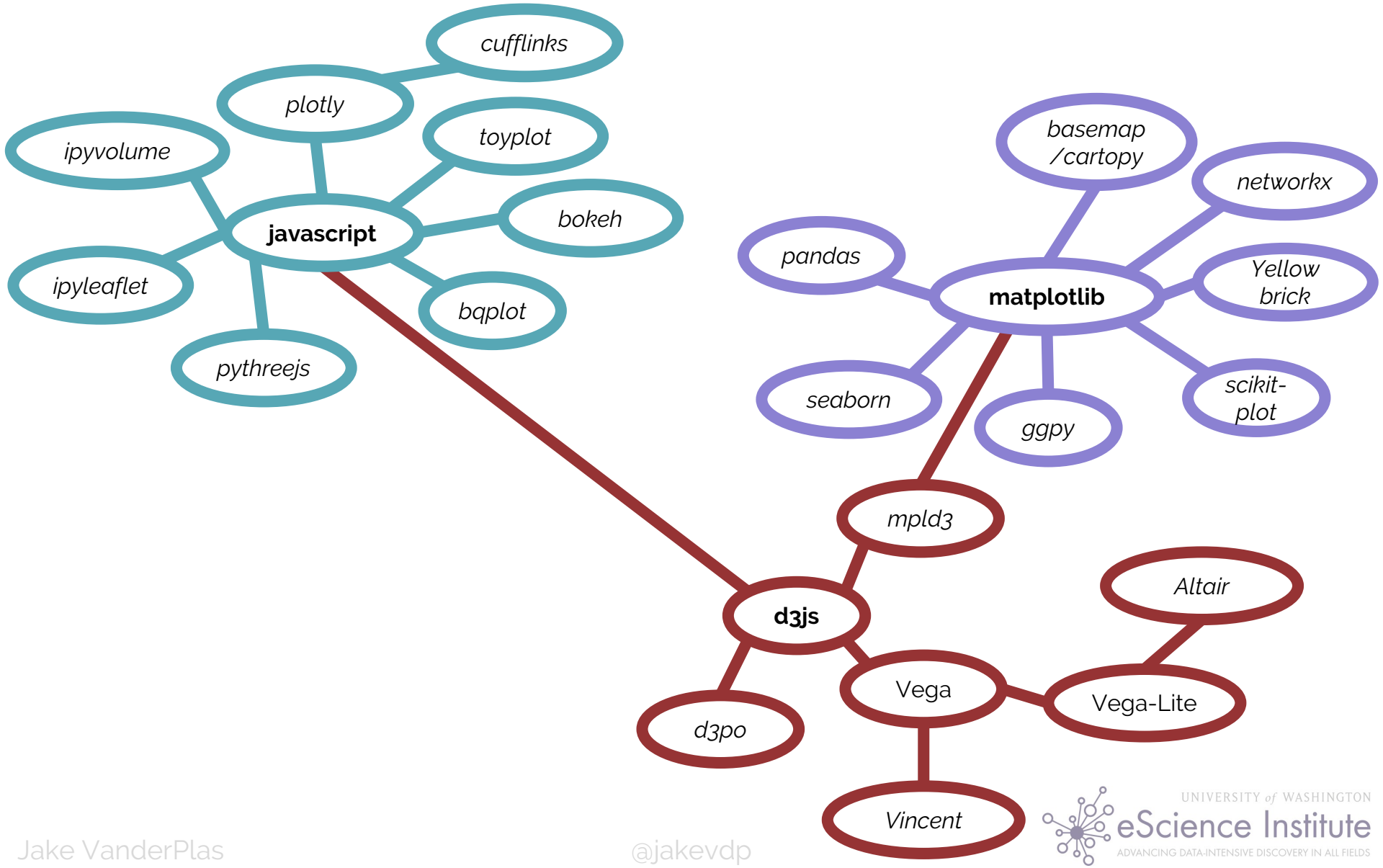

cufflinks
plotly
basemap
/cartopy
networkx
ipyvolume
toyplot
Yellow
brick
tplotlib
bokeh
javascript
scikit-
plot
seaborn
ggpy
ipyleaflet
bqplot
pythreejs

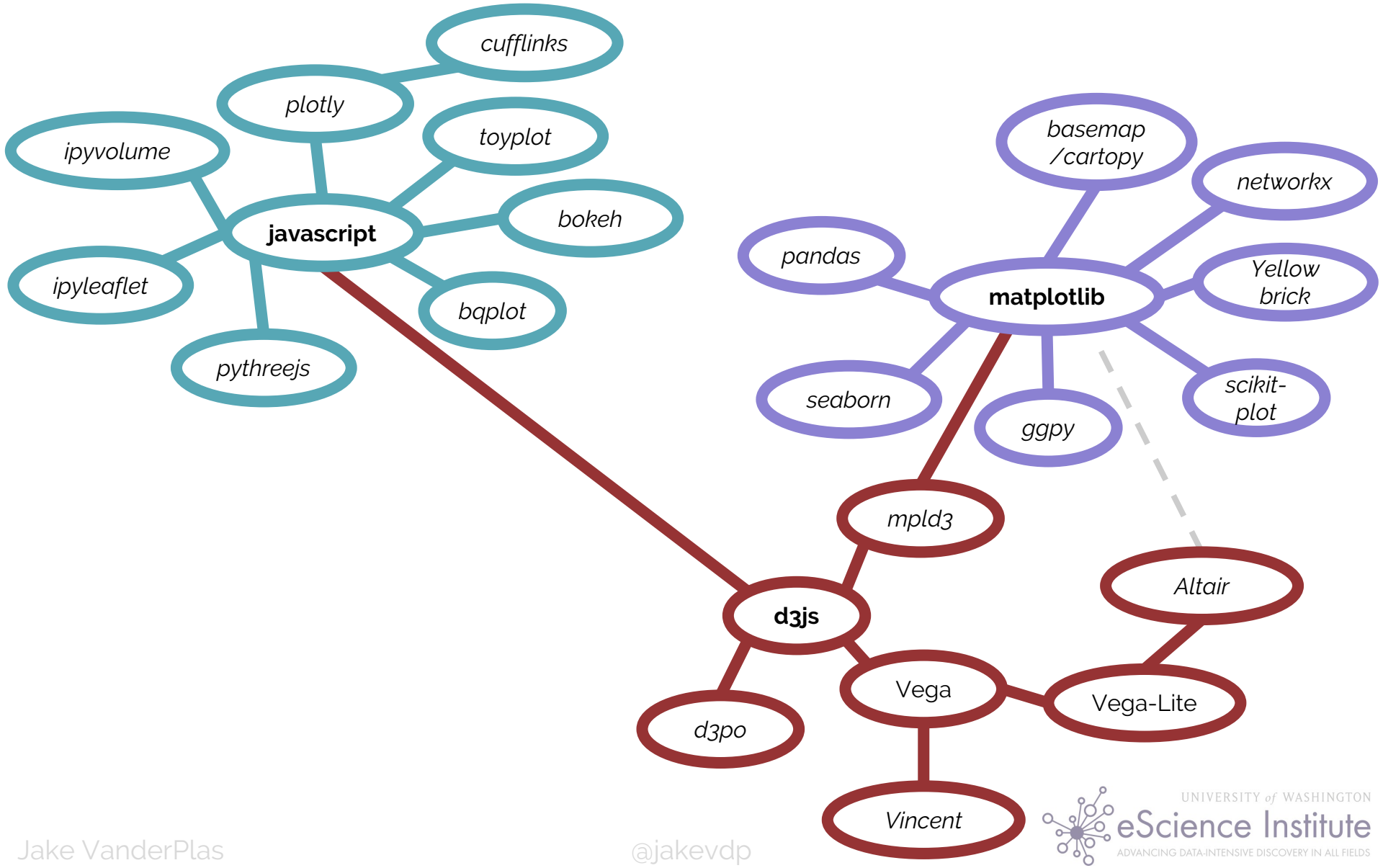

cufflinks
plotly
ipyvolume
toyplot
javascript
bokeh
ipyleaflet
bqplot
pythreejs

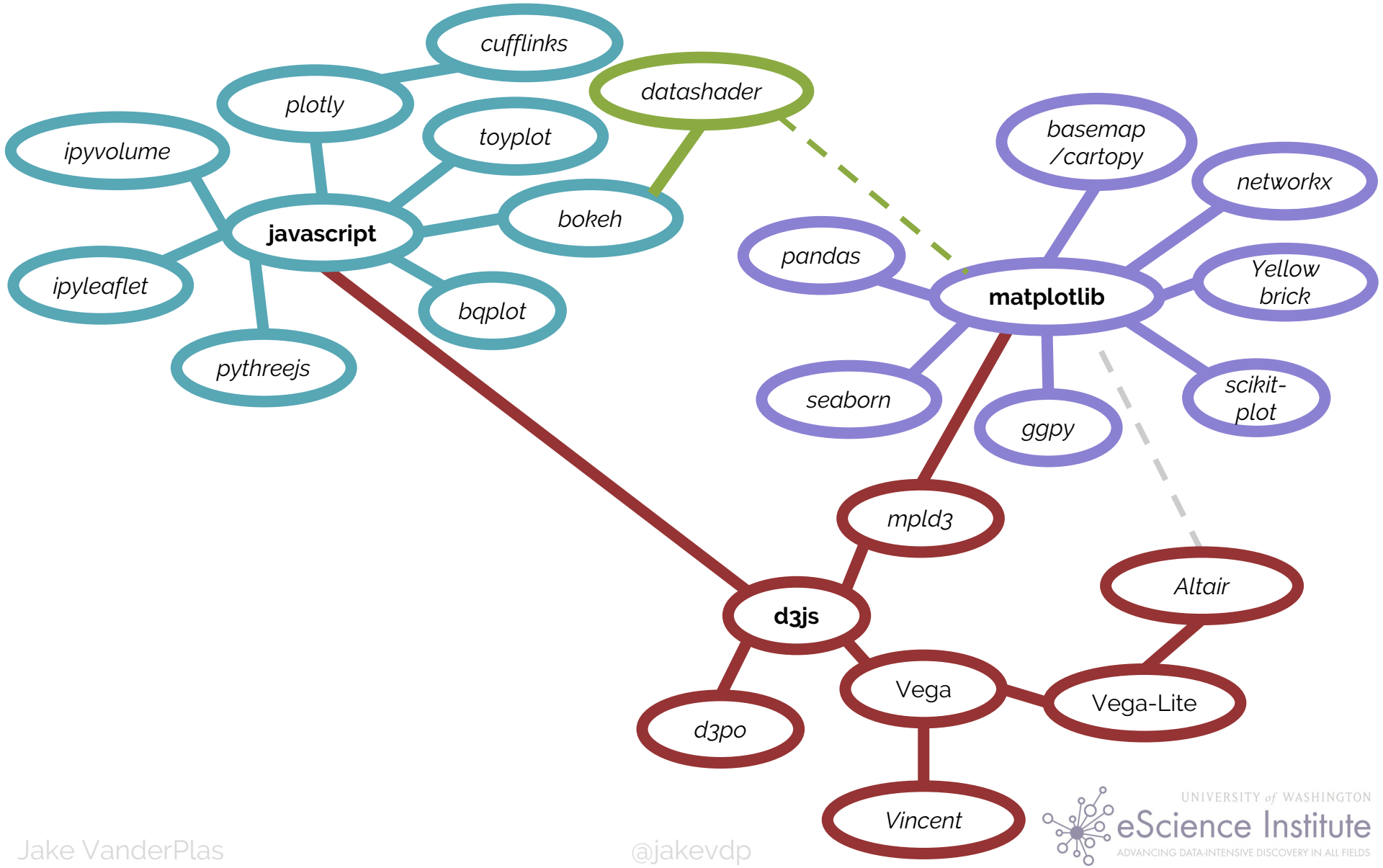

basemap
/cartopy
networkx
pandas
matplotlib
Yellow
brick
seaborn
ggpy
scikit-
plot

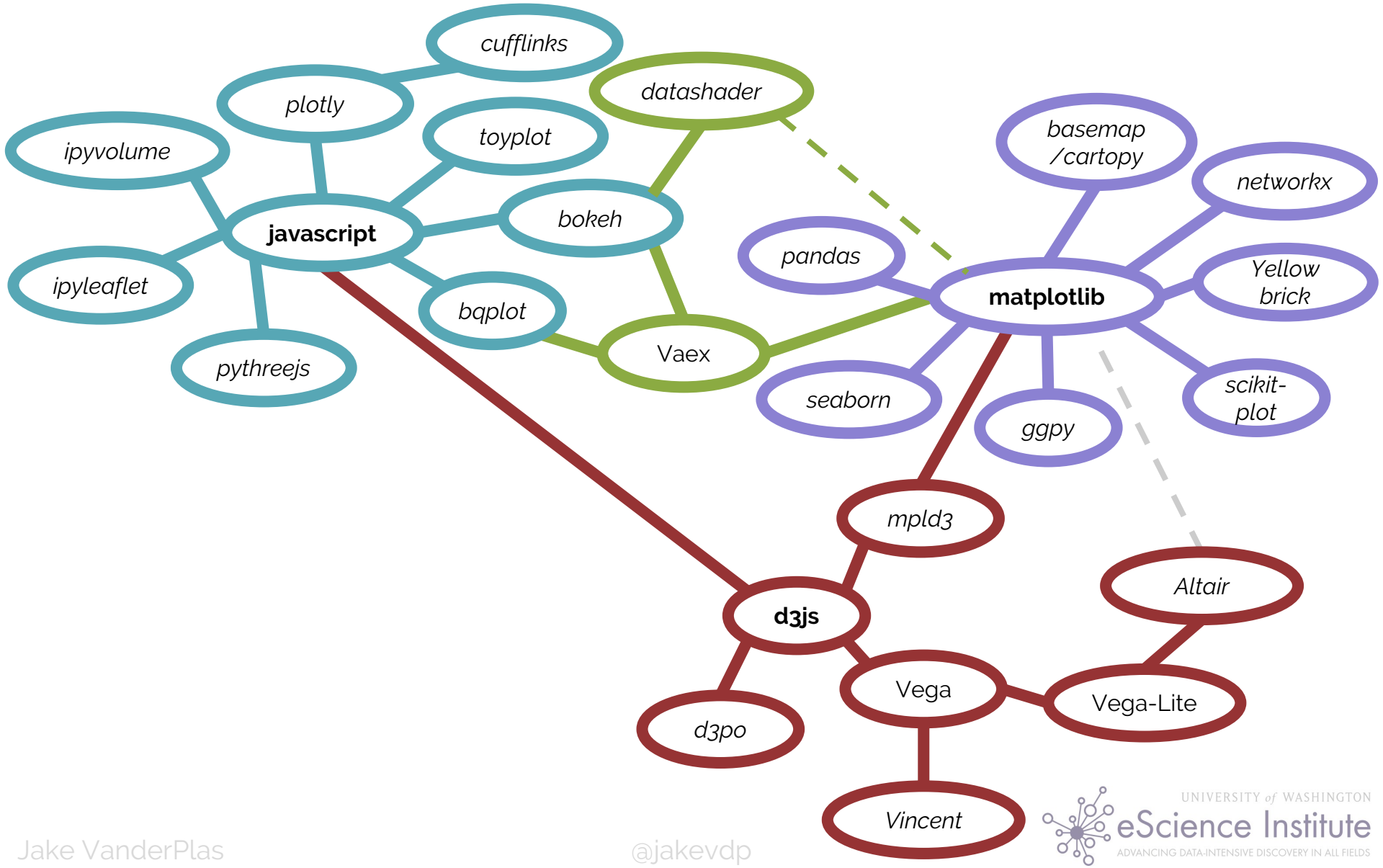


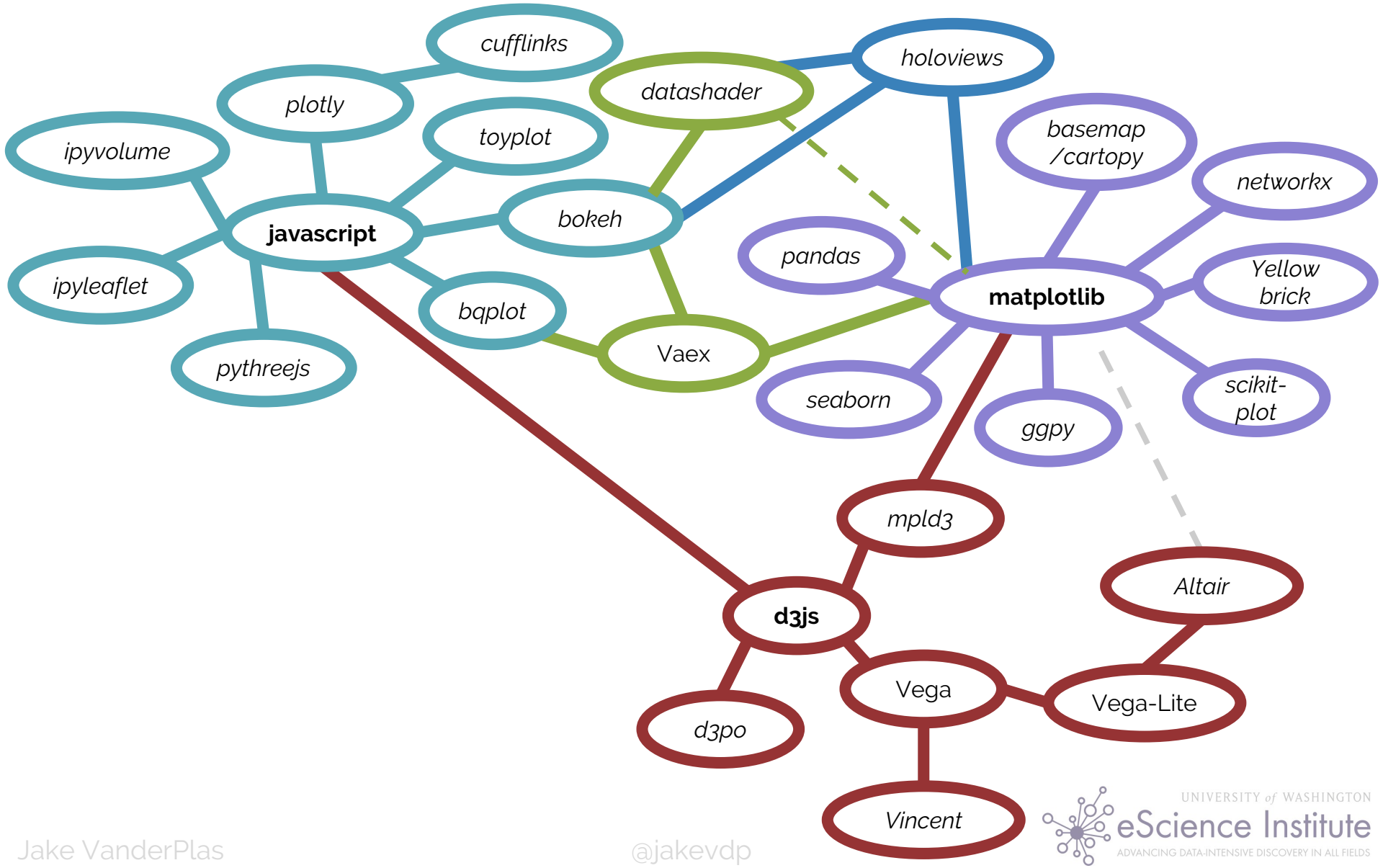


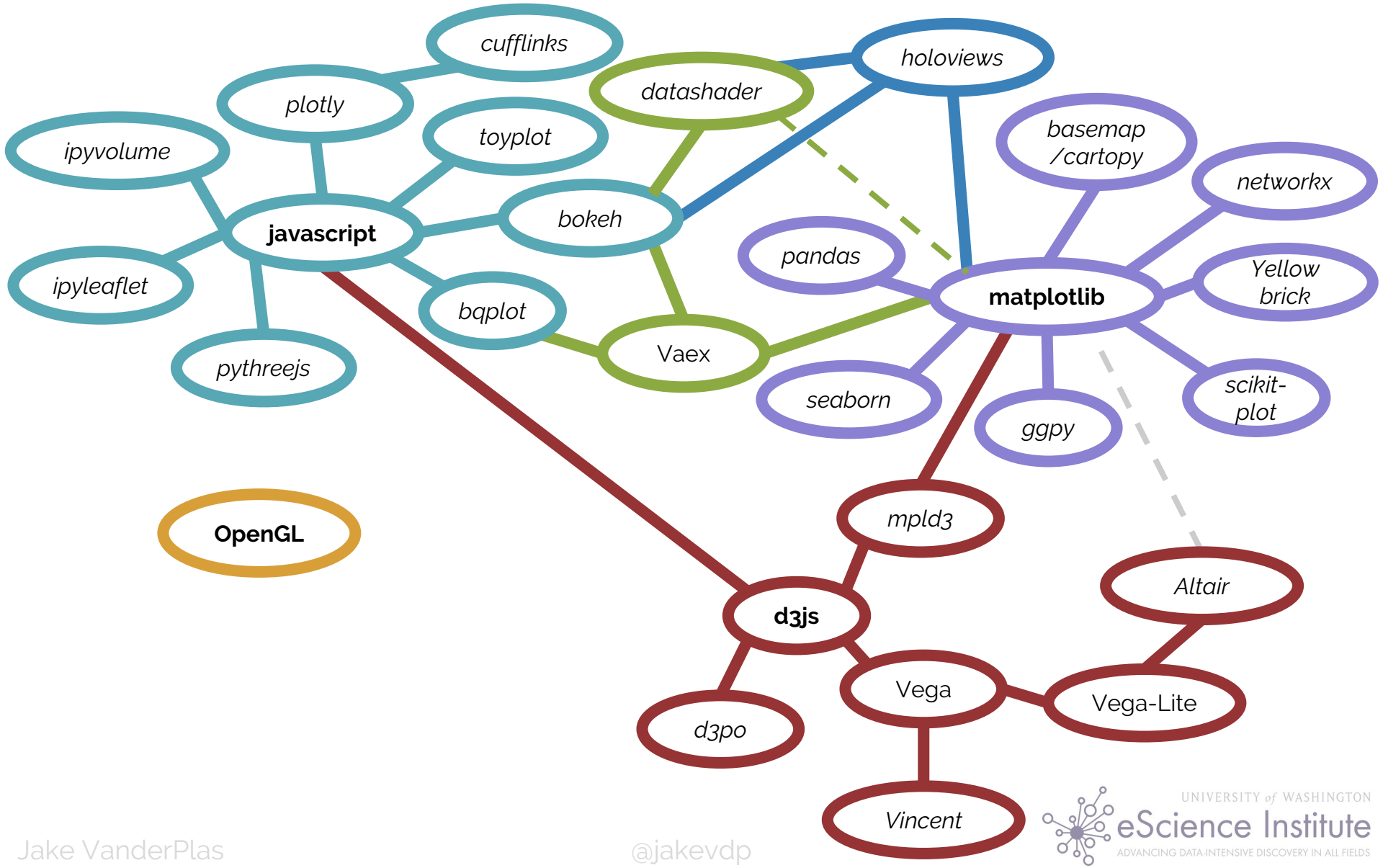

cufflinks
plotly
ipyvolume
toyplot
javascript
bokeh
ipyleaflet
bqplot
pythreejs
basemap
/cartopy
networkx
pandas
matplotlib
Yellow
brick
seaborn
ggpy
scikit-
plot
mpld3
Altair
d3js
Vega
Vega-Lite
d3po
Vincent

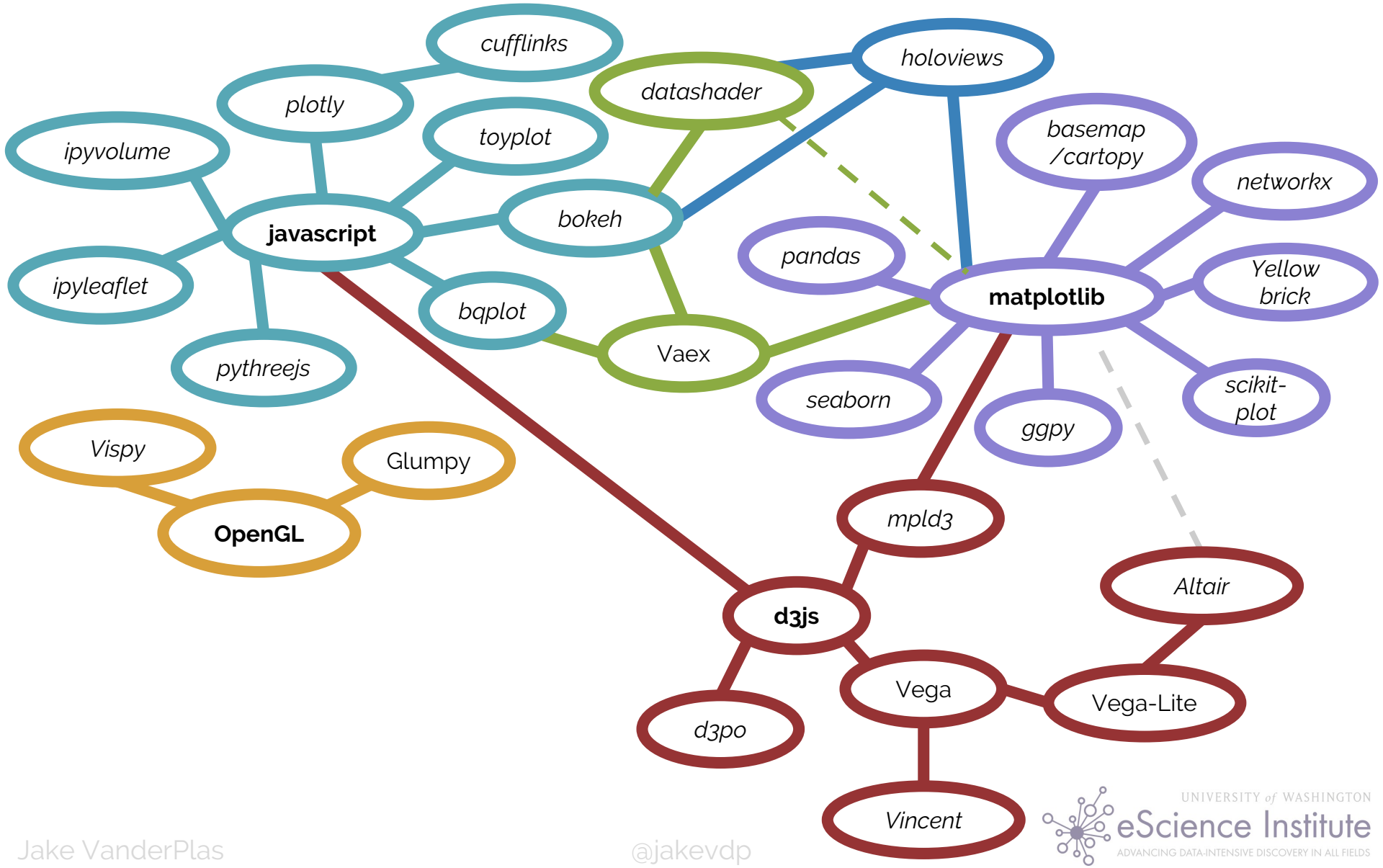

cufflinks
plotly
ipyvolume
toyplot
javascript
bokeh
ipyleaflet
bqplot
pythreejs
basemap
/cartopy
networkx
pandas
matplotlib
Yellow
brick
seaborn
ggpy
scikit-
plot
mpld3
Altair
d3js
Vega
Vega-Lite
d3po
Vincent

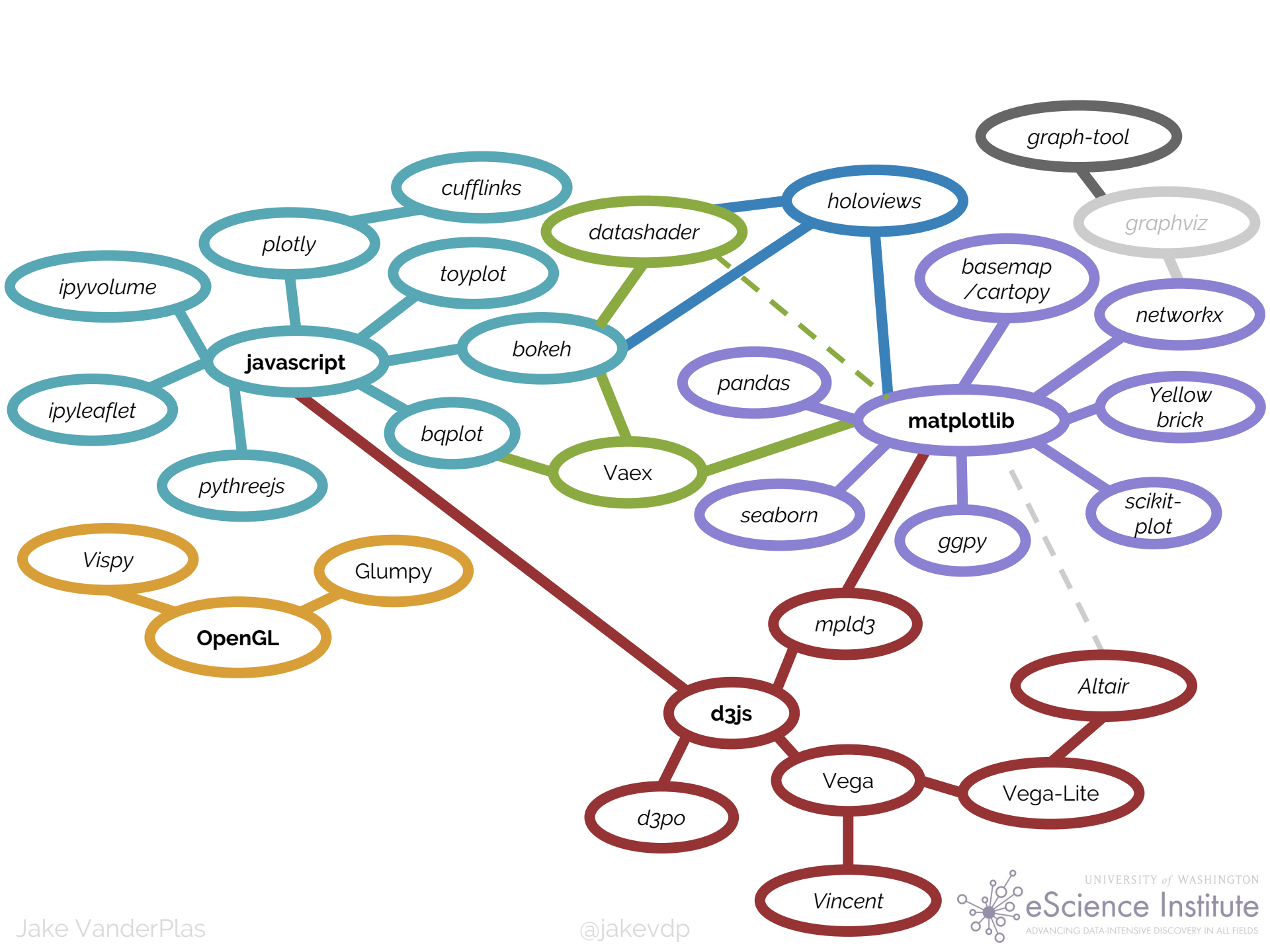





cufflinks
plotly
ipyvolume
toyplot
javascript
ipyleaflet
pythreejs
bokeh
bqplot
datashader
holoviews
Vaex
basemap
/cartopy
networkx
pandas
Yellow
brick
matplotlib
seaborn
ggpy
scikit-
plot
mpld3
Altair
d3js
Vega
Vega-Lite
d3po
Vincent

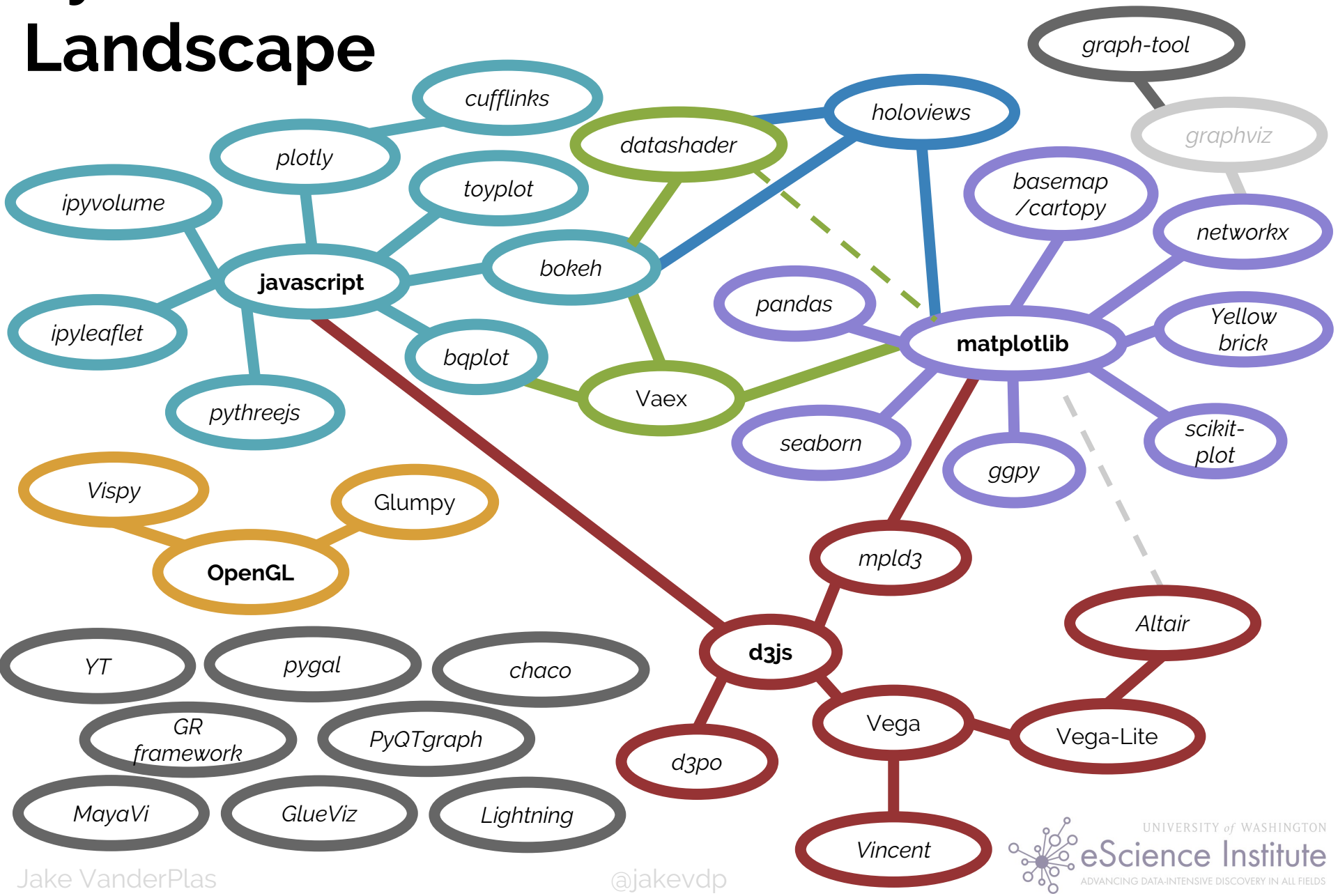

cufflinks
plotly
ipyvolume
toyplot
javascript
ipyleaflet
pythreejs
bokeh
bqplot
datashader
holoviews
basemap
/cartopy
networkx
pandas
Yellow
brick
matplotlib
Vaex
seaborn
ggpy
scikit-
plot
OpenGL
mpld3
Altair
d3js
Vega
Vega-Lite
d3po
Vincent

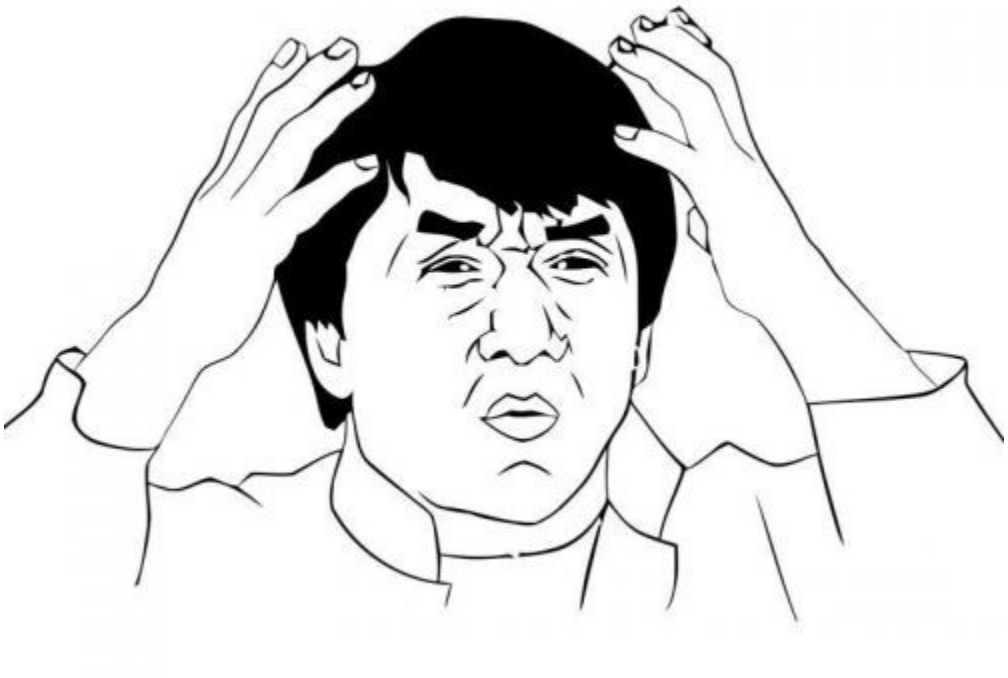



graph-tool
graphviz
cufflinks
holoviews
plotly
datashader
basemap
/cartopy
networkx
ipyvolume
toyplot
bokeh
javascript
pandas
matplotlib
Yellow
brick
ipyleaflet
bqplot
Vaex
pythreejs
seaborn
ggpy
scikit-
plot
Vispy
Glumpy
OpenGL
mpld3
Altair
d3js
Vega
Vega-Lite
d3po
Vincent
Jake VanderPlas
@jakevdp
UNIVERSITY of WASHINGTON
eScience Institute
ADVANCING DATA-INTENSIVE DISCOVERY IN ALL FIELDS


graph-tool
graphviz
cufflinks
holoviews
plotly
datashader
basemap
/cartopy
networkx
ipyvolume
toyplot
bokeh
javascript
pandas
Yellow
brick
ipyleaflet
bqplot
matplotlib
Vaex
pythreejs
seaborn
ggpy
scikit-
plot
Vispy
Glumpy
OpenGL
mpld3
Altair
YT
pygal
chaco
d3js
GR
framework
PyQTgraph
Vega
Vega-Lite
d3po
MayaVi
GlueViz
Lightning
Vincent
Jake VanderPlas
@jakevdp
UNIVERSITY of WASHINGTON
eScience Institute
ADVANCING DATA-INTENSIVE DISCOVERY IN ALL FIELDS

Python's Visualization Landscape





How did we get here?

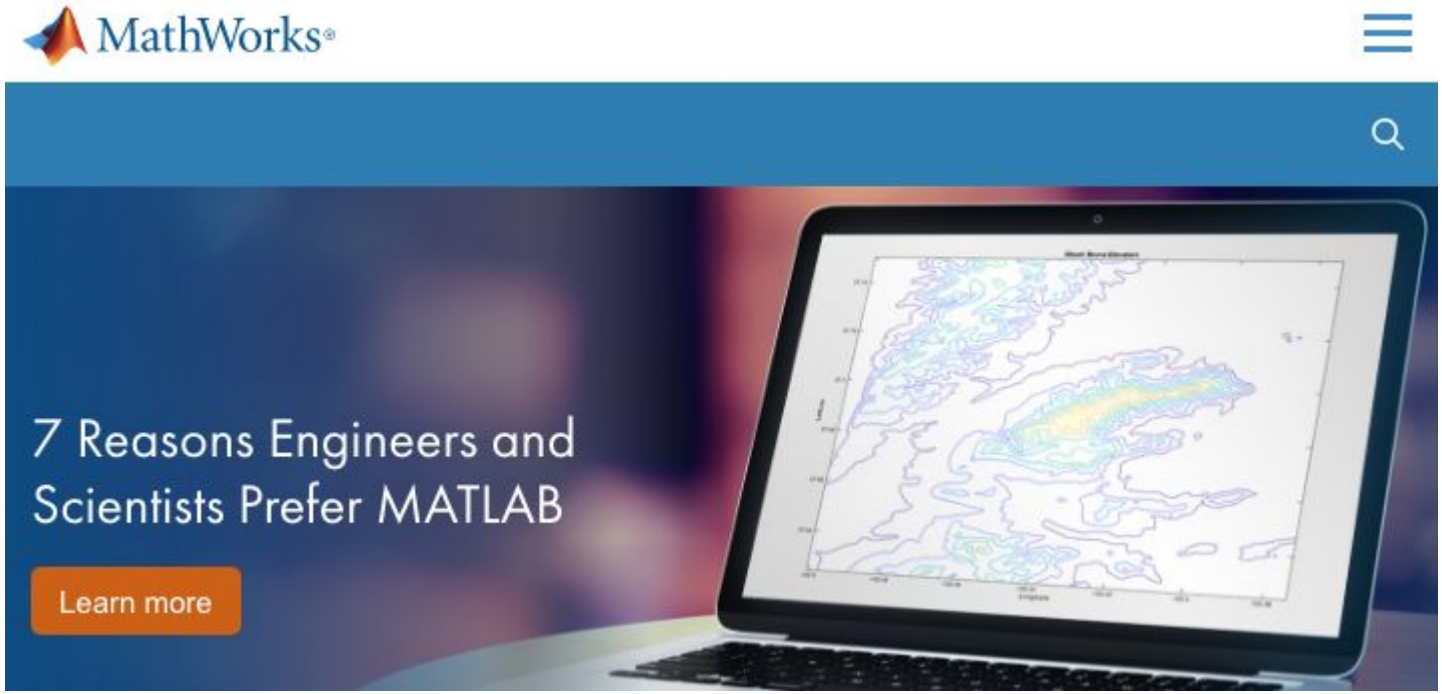
In the beginning was matplotlib*

* well, actually… Python visualization existed before matplotlib, but was not very mature.

Plotting with Matplotlib

Strengths:

- Designed like MatLab: switching was easy



Plotting with Matplotlib

Strengths:

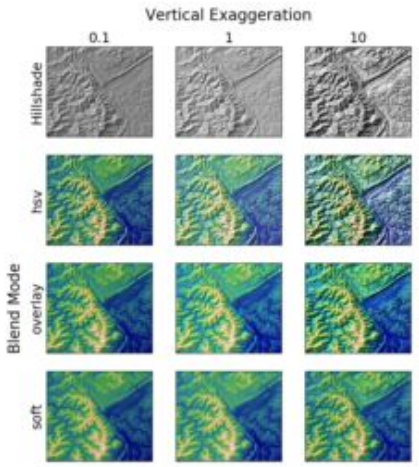
- Designed like MatLab: switching was easy
- Many rendering backends

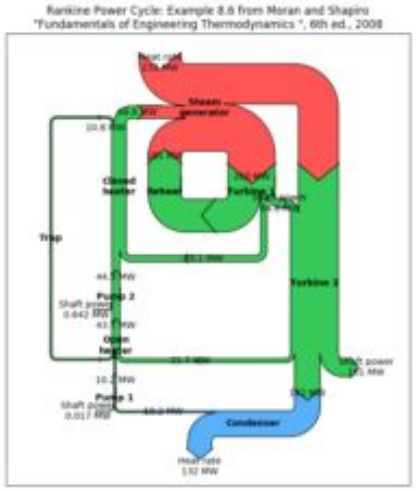
```
In [26]: from matplotlib import rcsetup
         rcsetup.all_backends

Out[26]: ['GTK',
          'GTKAgg',
          'GTKCairo',
          'MacOSX',
          'Qt4Agg',
          'Qt5Agg',
          'TkAgg',
          'WX',
          'WXAgg',
          'GTK3Cairo',
          'GTK3Agg',
          'WebAgg',
          'nbAgg',
          'agg',
          'cairo',
          'gdk',
          'pdf',
          'pgf',
          'ps',
          'svg',
          'template']
```

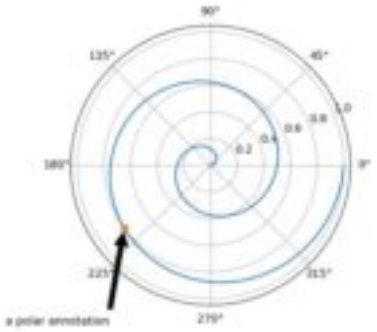
Plotting with Matplotlib

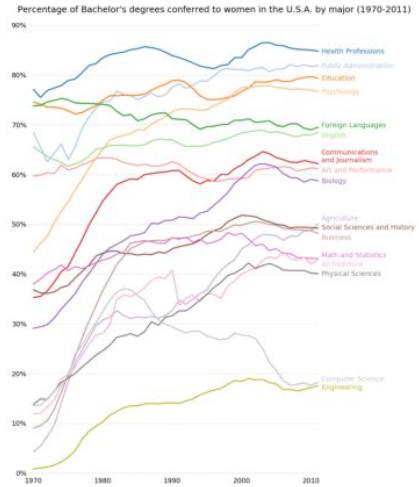
Strengths:

- Designed like MatLab: switching was easy
- Many rendering backends
- Can reproduce just about any plot (with a bit of effort)





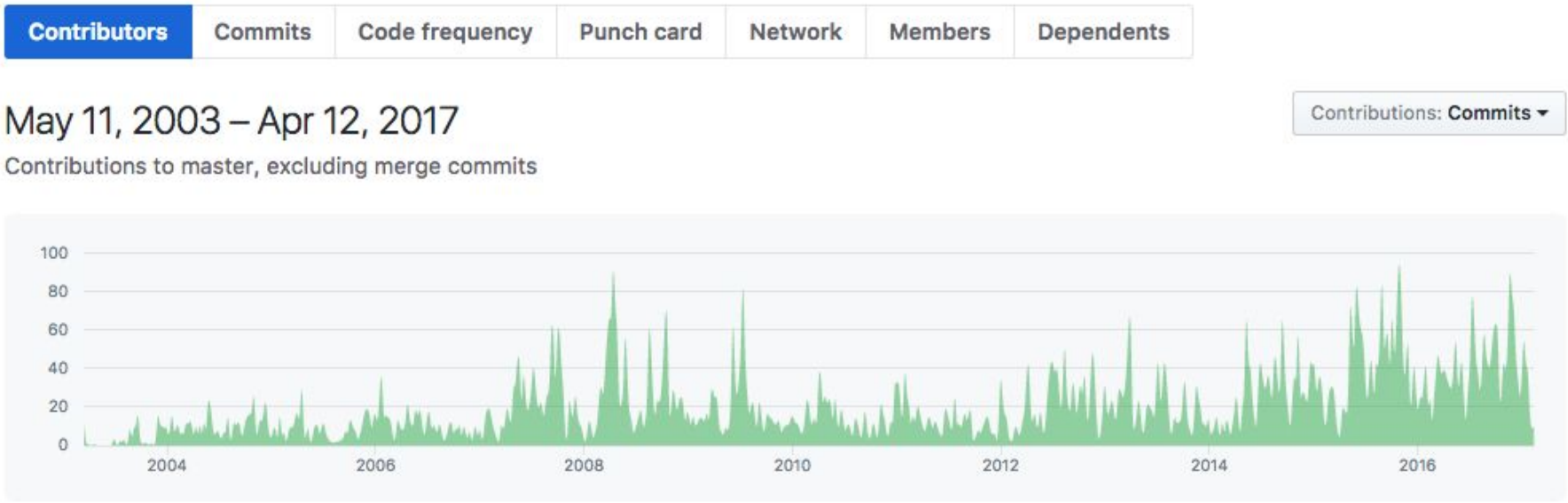




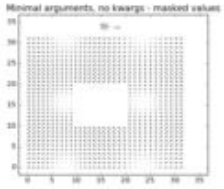
Plotting with Matplotlib

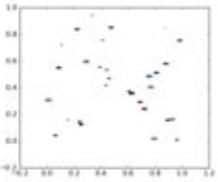
Strengths:

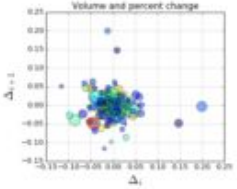
- Designed like MatLab: switching was easy
- Many rendering backends
- Can reproduce just about any plot (with a bit of effort)
- Well-tested, standard tool for over a decade



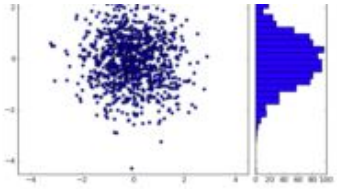
Matplotlib Gallery

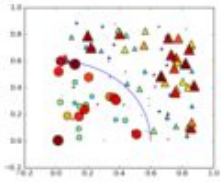

quiver_demo

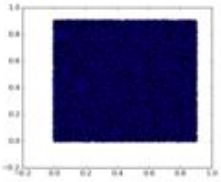

scatter_custom_symbol

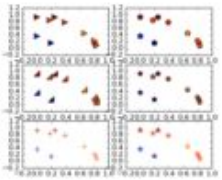


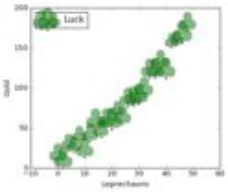
scatter_demo2


scatter_hist

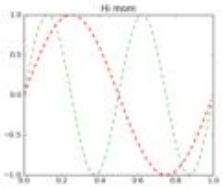

scatter_masked


scatter_profile

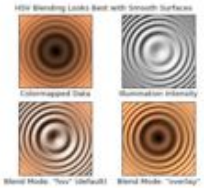

scatter_star_poly

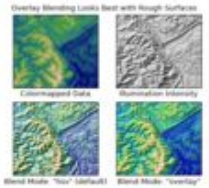


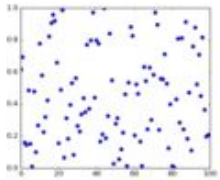
scatter_symbol

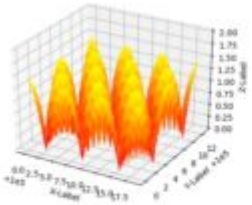


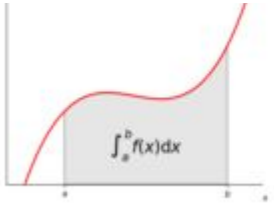
set_and_get


shading_example

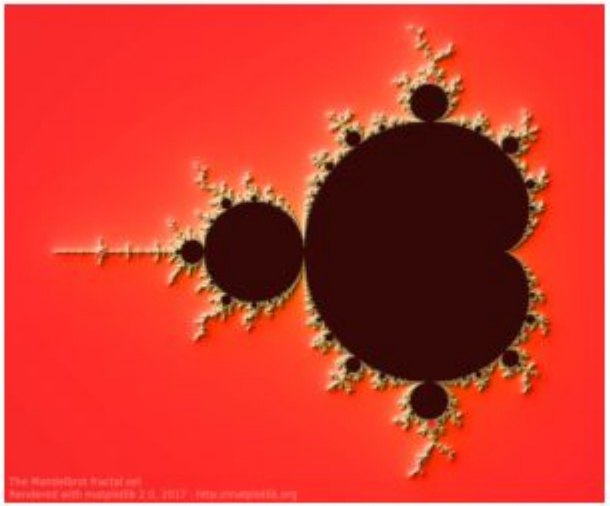

shading_example


shared_axis_across_figures

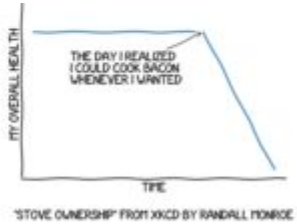

offset_demo



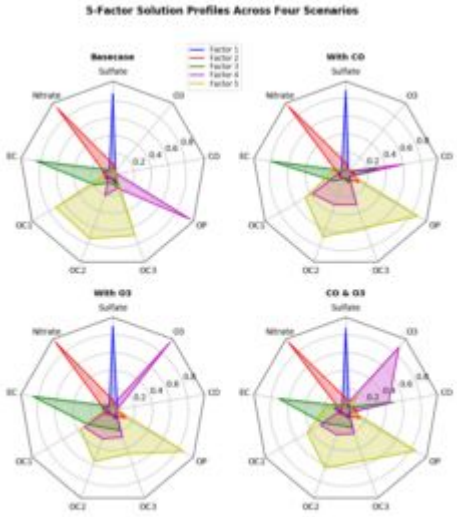
integral_demo


mandelbrot


svg_filter_pie



xkcd



radar_chart

Example: Statistical Data

```
import pandas as pd
iris = pd.read_csv('iris.csv')
iris.head()
```

	petalLength	petalWidth	sepalLength	sepalWidth	species
0	1.4	0.2	5.1	3.5	setosa
1	1.4	0.2	4.9	3.0	setosa
2	1.3	0.2	4.7	3.2	setosa
3	1.5	0.2	4.6	3.1	setosa
4	1.4	0.2	5.0	3.6	setosa

Tidy data: i.e. rows are samples, columns are features

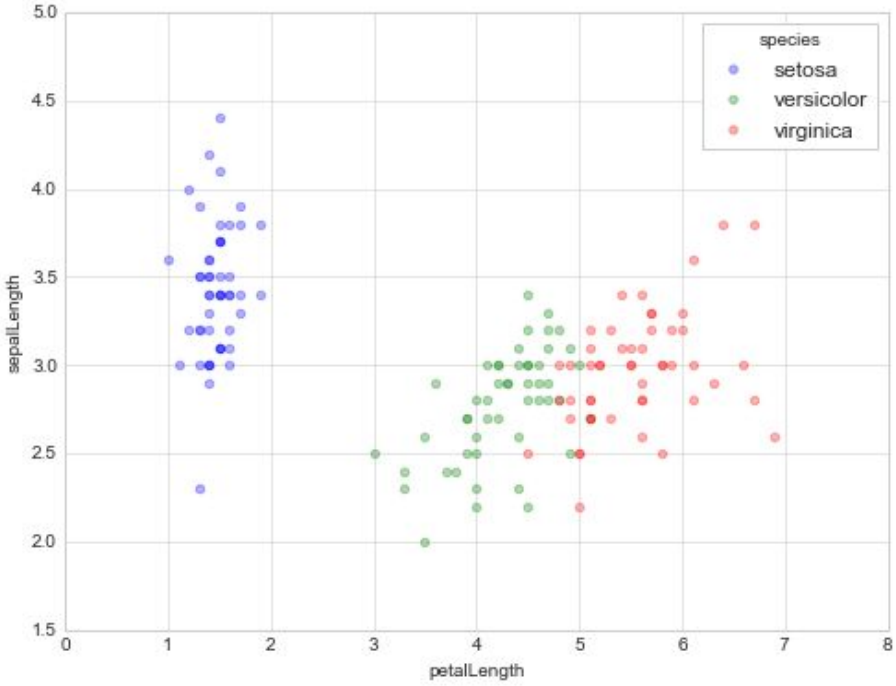
Just a simple visualization . . .

"I want to scatter petal length vs. sepal length, and color by species"

	petalLength	petalWidth	sepalLength	sepalWidth	species
0	1.4	0.2	5.1	3.5	setosa
1	1.4	0.2	4.9	3.0	setosa
2	1.3	0.2	4.7	3.2	setosa
3	1.5	0.2	4.6	3.1	setosa
4	1.4	0.2	5.0	3.6	setosa

Just a simple visualization . . .

```
color_map = dict(zip(iris.species.unique(),
                     ['blue', 'green', 'red']))

for species, group in iris.groupby('species'):
    plt.scatter(group['petalLength'], group['sepalLength'],
                color=color_map[species],
                alpha=0.3, edgecolor=None,
                label=species)

plt.legend(frameon=True, title='species')
plt.xlabel('petalLength')
plt.ylabel('sepalLength')
```



Plotting with Matplotlib

Strengths:

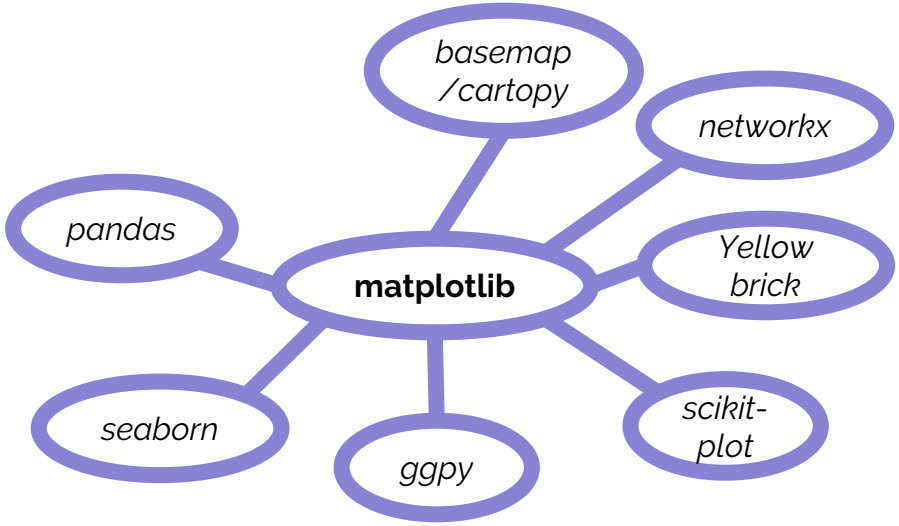
- Designed like MatLab: switching was easy
- Many rendering backends
- Can reproduce just about any plot with a bit of effort
- Well-tested, standard tool for over a decade

Weaknesses:

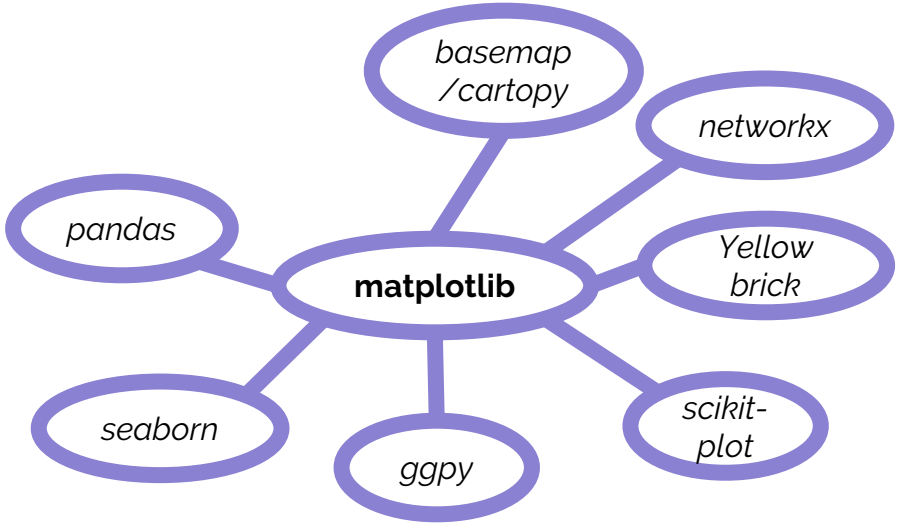
- API is imperative & often overly verbose
- Sometimes poor stylistic defaults
- Poor support for web/interactive graphs
- Often slow for large & complicated data

Everyone's Goal:
Improve on the weaknesses of matplotlib
(without sacrificing the strengths!)

Building on Matplotlib. . .

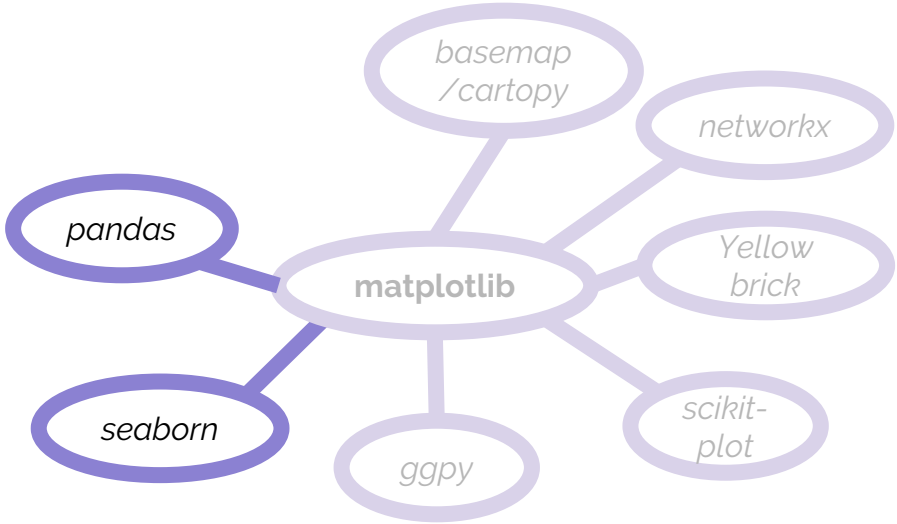


Building on Matplotlib. . .



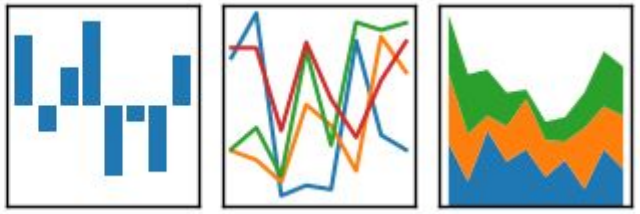
Common Idea: Keep matplotlib as a **versatile, well-tested backend**, and provide a new domain-specific API.

Building on Matplotlib. . .



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

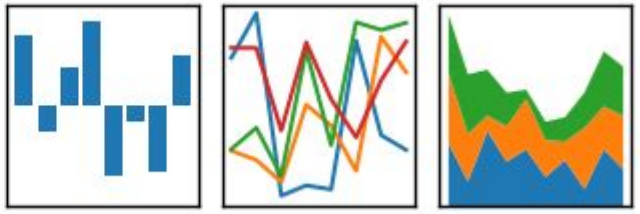


Key Features:

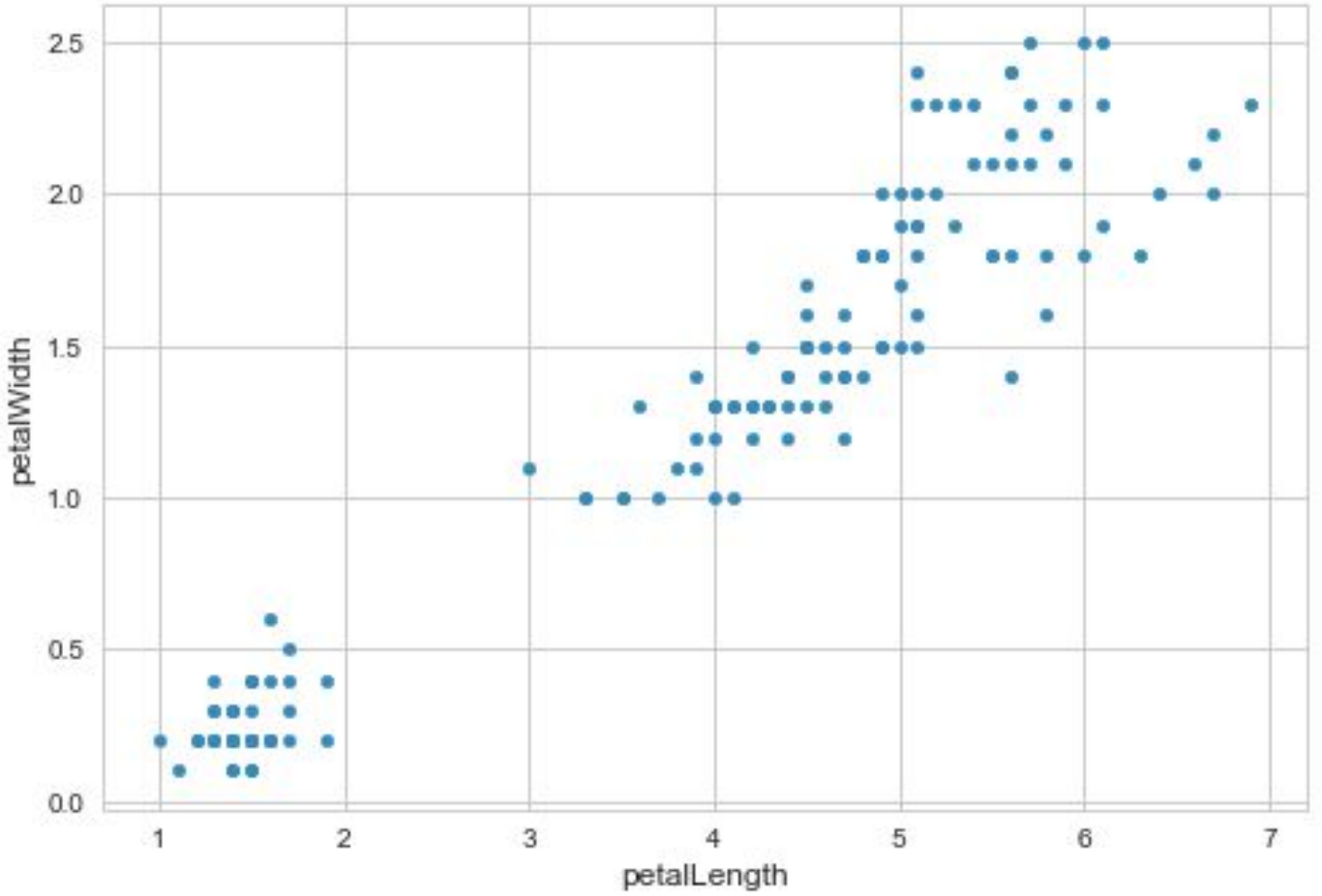
- Pandas provides a `DataFrame` object
- Also provides a simple API for plotting DataFrames

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

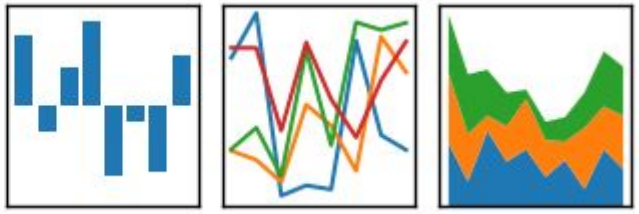


```
iris.plot.scatter('petalLength', 'petalWidth')
```



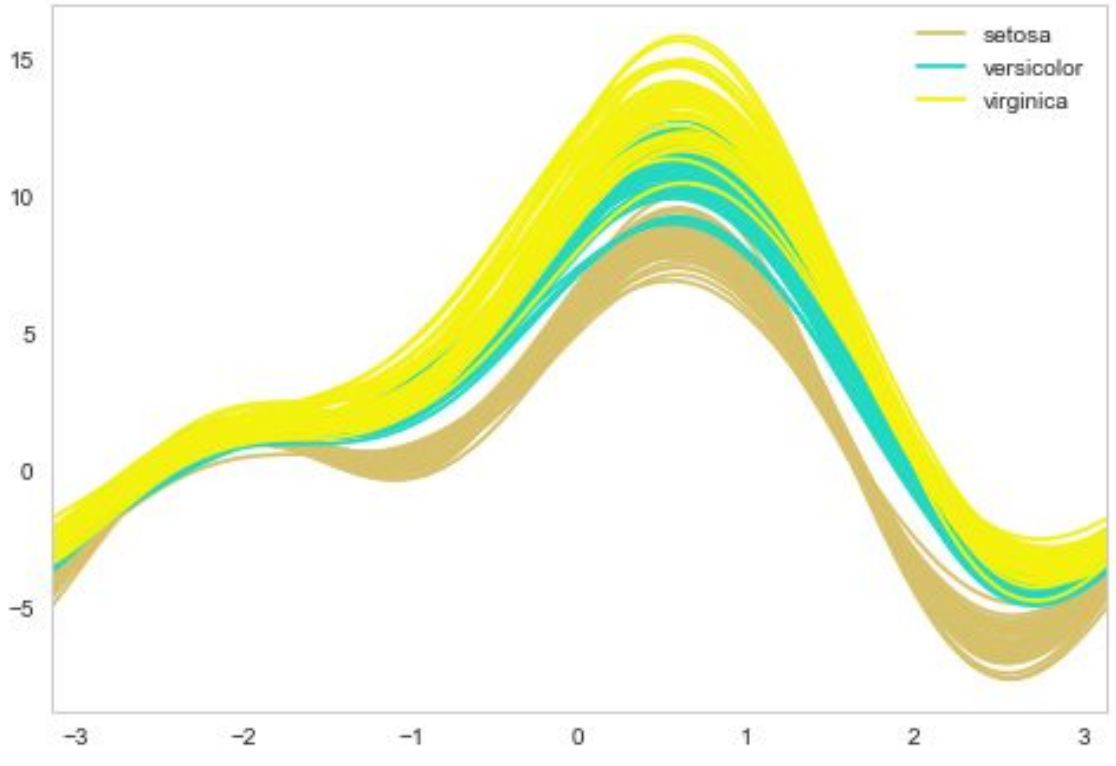
pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

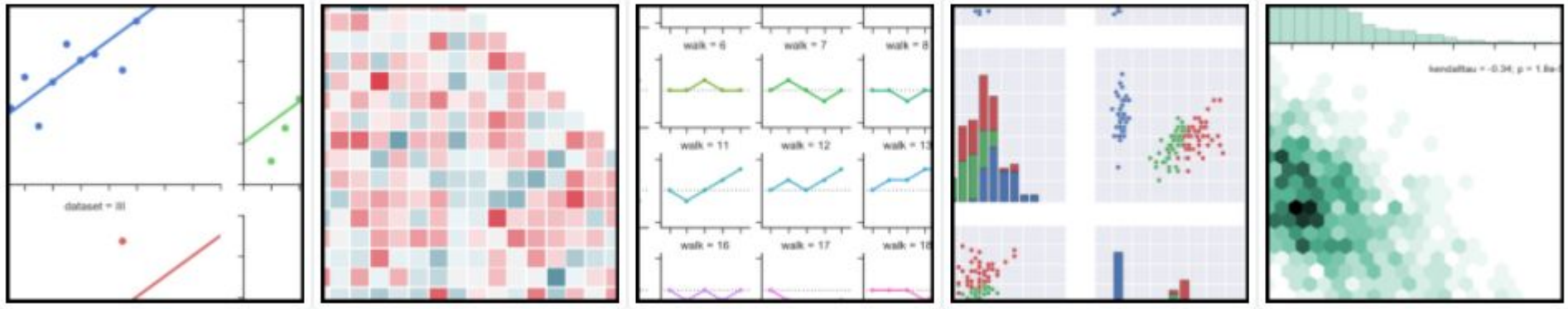


- More sophisticated statistical visualization tools have recently been added

```
from pandas.tools.plotting import andrews_curves
andrews_curves(iris, 'species')
```



Seaborn: statistical data visualization



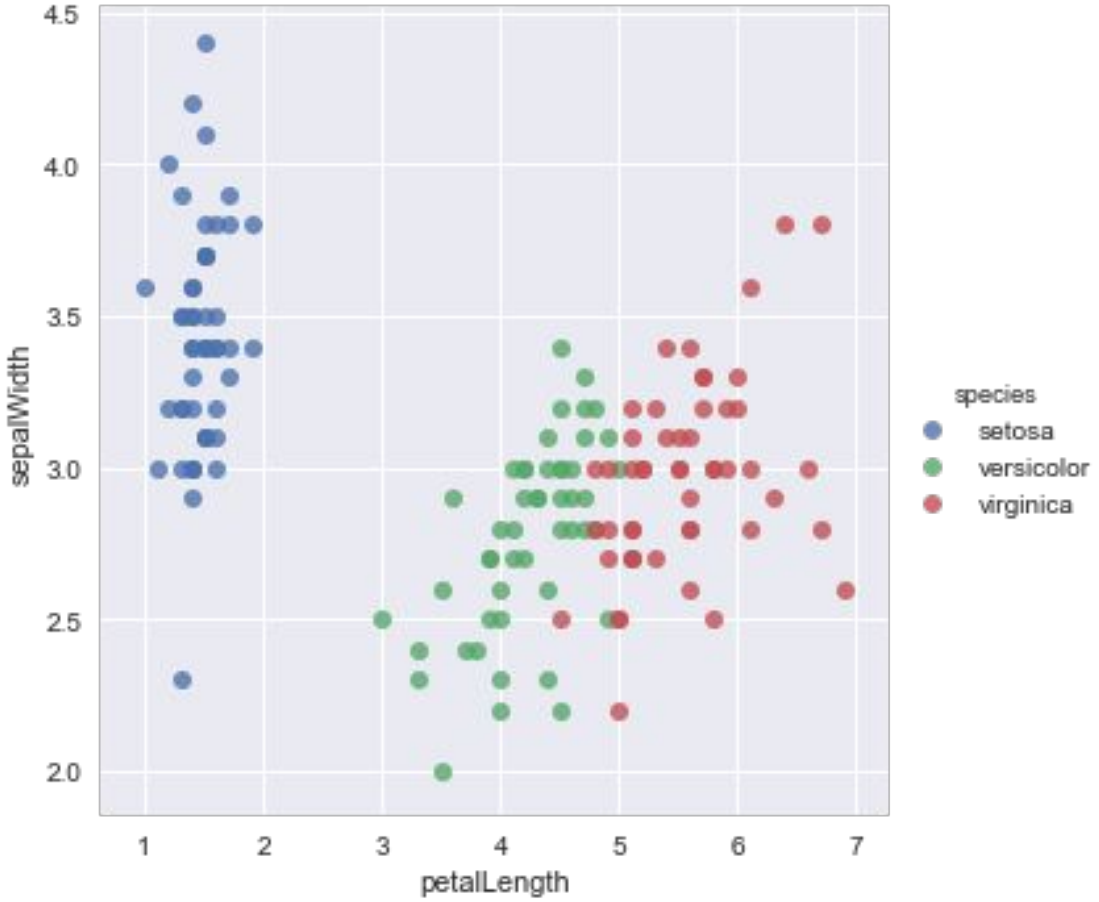
Key Features:

- Like Pandas, wraps matplotlib
- Nice set of color palettes & plot styles
- Focus on statistical visualization & modeling

http://seaborn.pydata.org

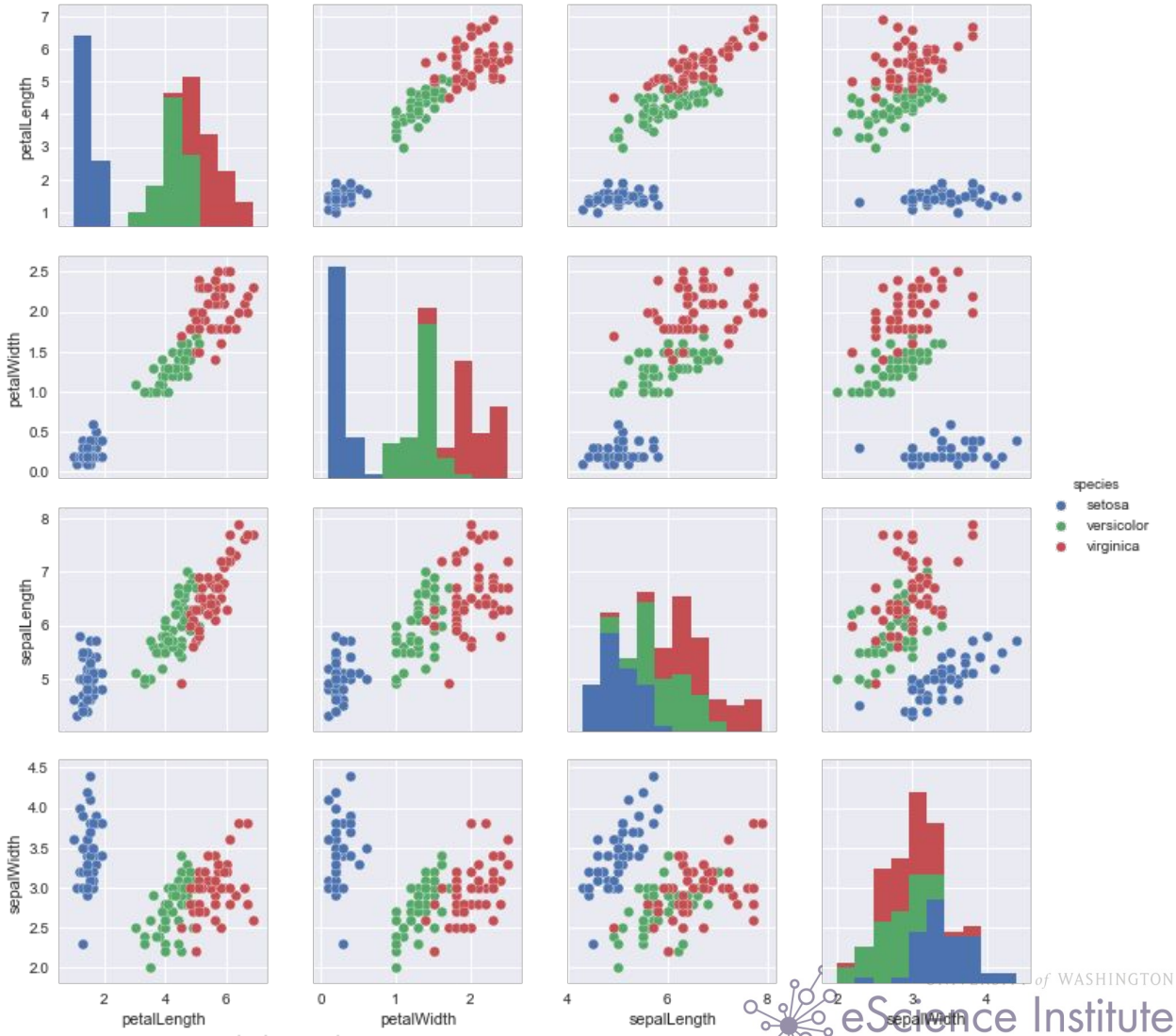
Seaborn examples

```
import seaborn as sns
sns.lmplot('petalLength', 'sepalWidth', iris,
           hue='species', fit_reg=False)
```

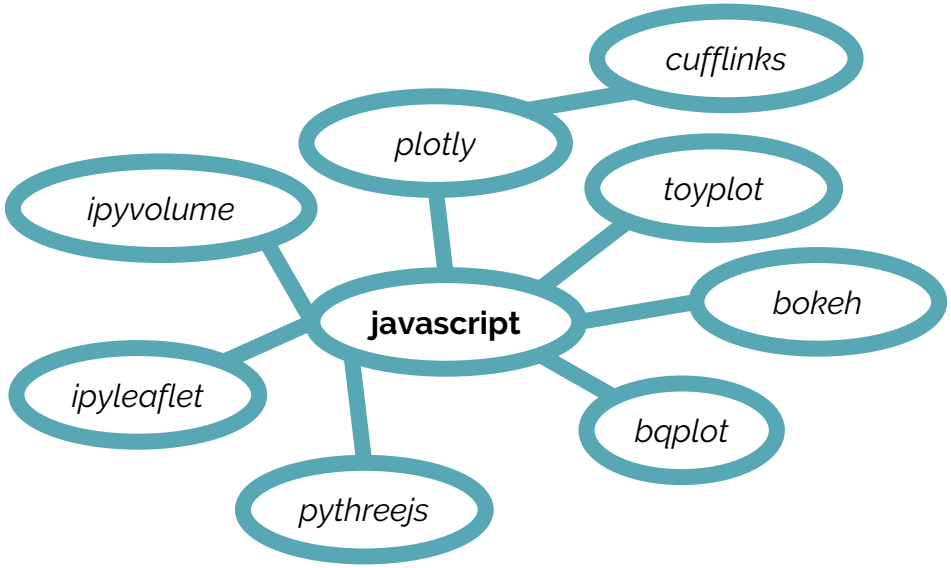


Seaborn examples

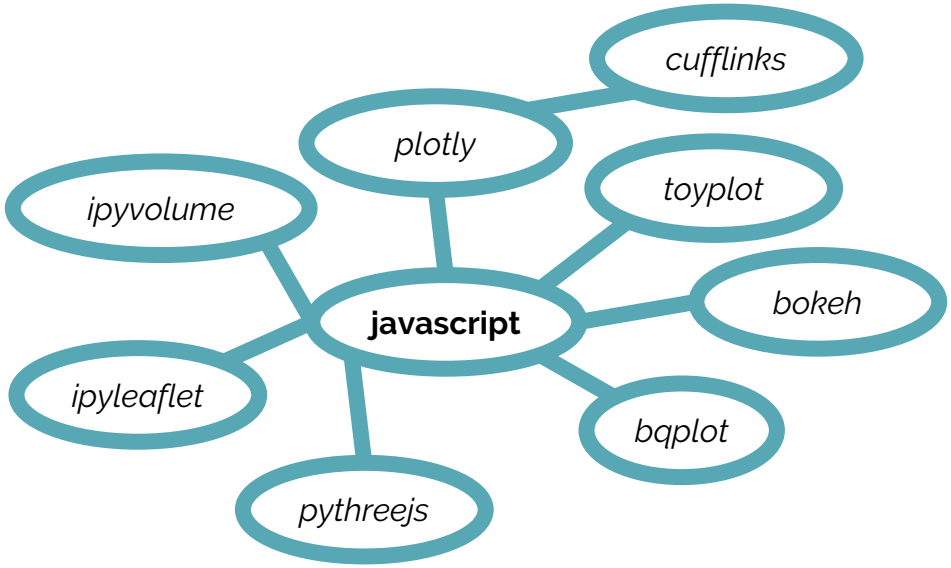
```
sns.pairplot(iris, hue='species')
```



Javascript-based Viz:

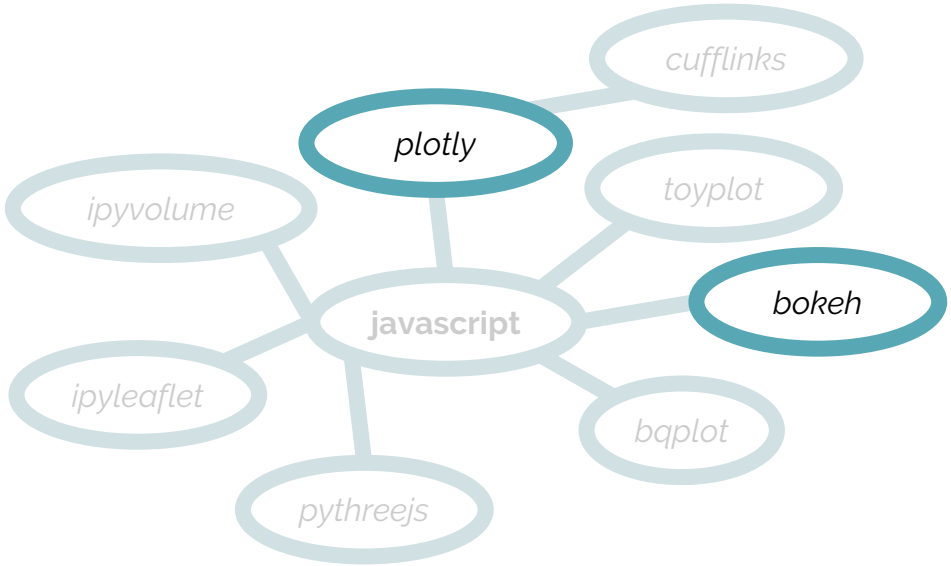


Javascript-based Viz:



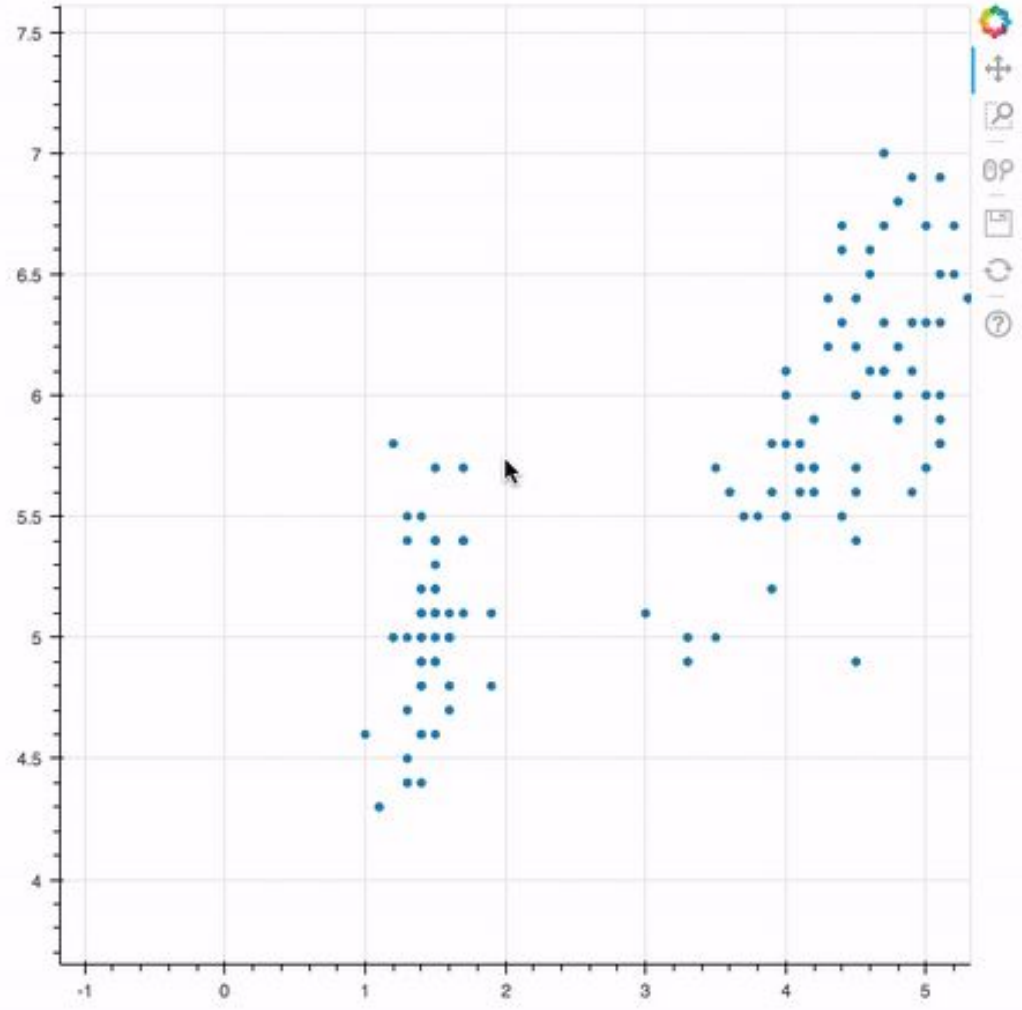
Common Idea: build a new API that produces a plot serialization (often JSON) that can be displayed in the browser (often in Jupyter notebooks)

Javascript-based Viz:

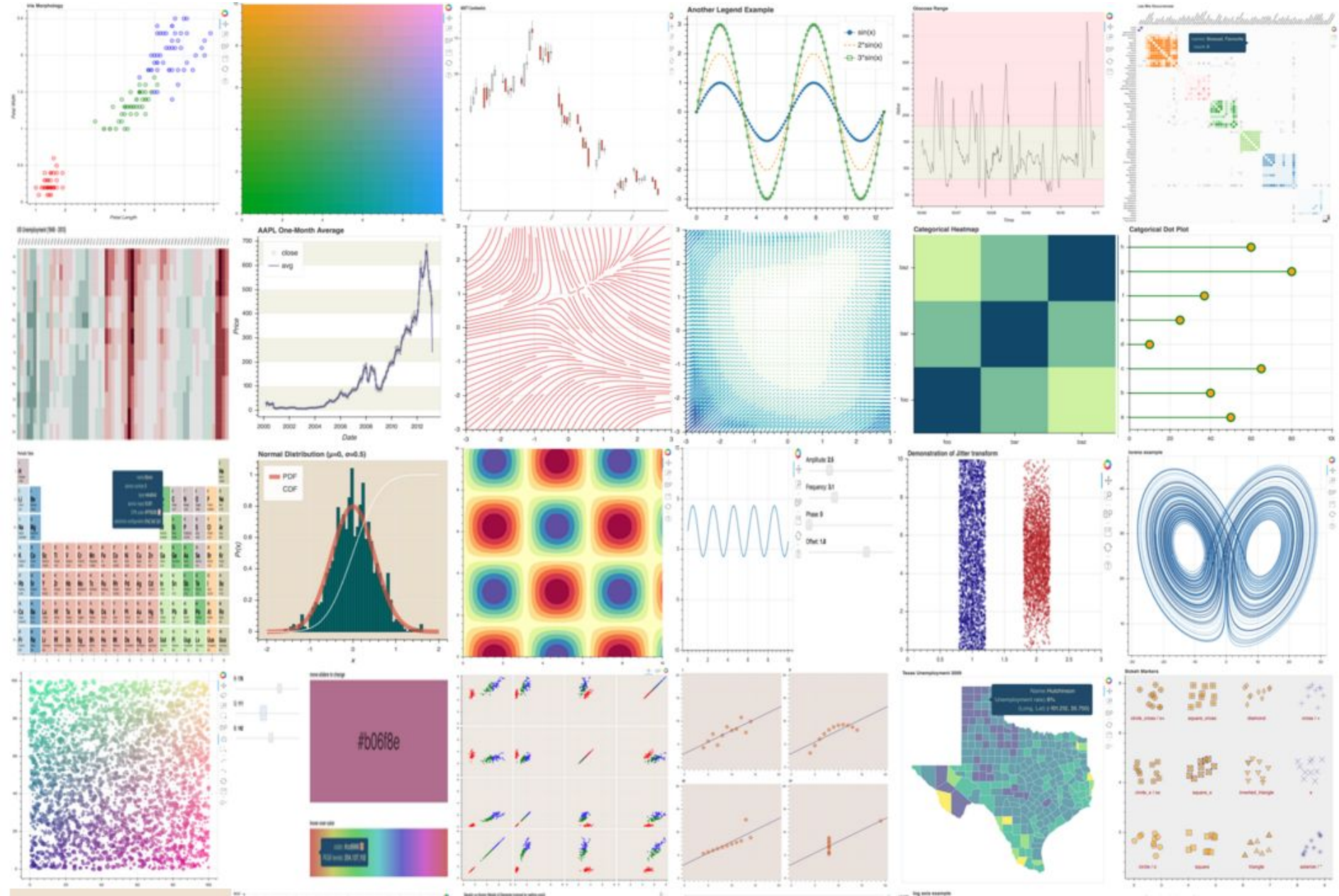


Plotting with Bokeh

```
In [10]: p = figure()
         p.circle(iris.petalLength, iris.sepalLength)
         show(p)
```



Bokeh Gallery



Plotting with Bokeh

Advantages:

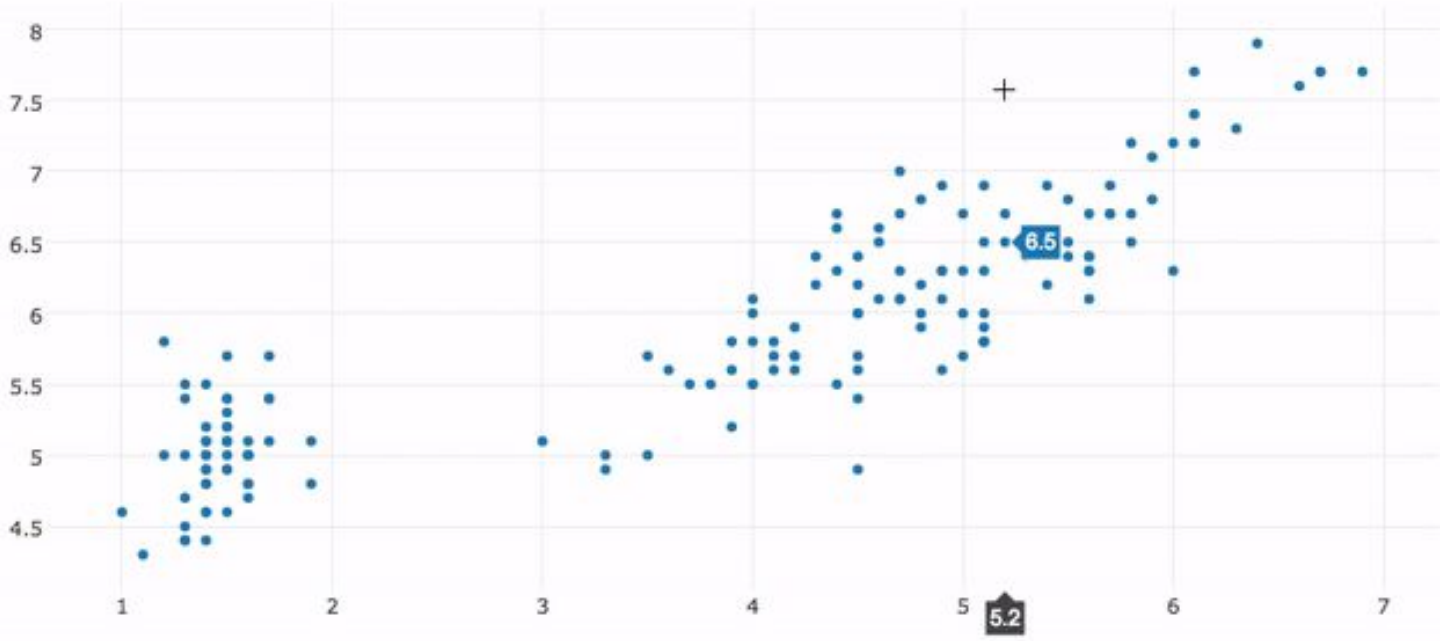
- Web view/interactivity
- Imperative and Declarative layer
- Handles large and/or streaming datasets
- Geographical visualization
- Fully open source

Disadvantages:

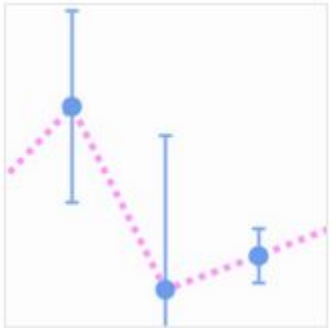
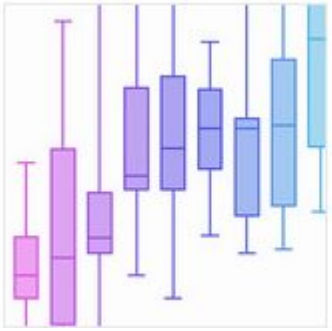
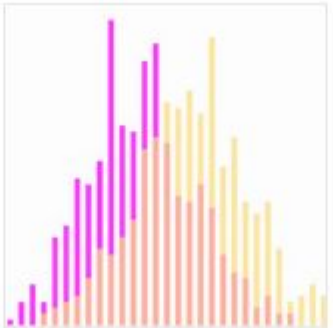
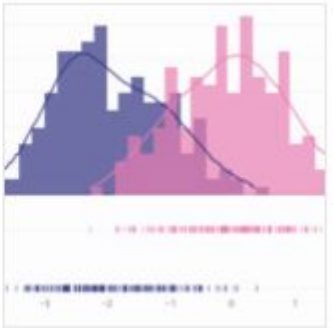
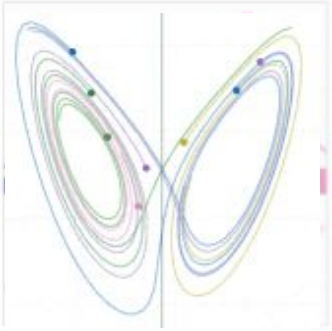
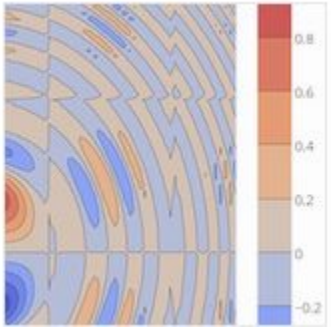
- No vector output (need PDF/EPS? Sorry)
- Newer tool with a smaller user-base than matplotlib

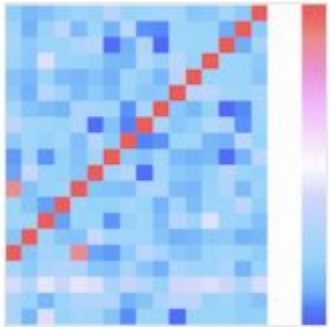
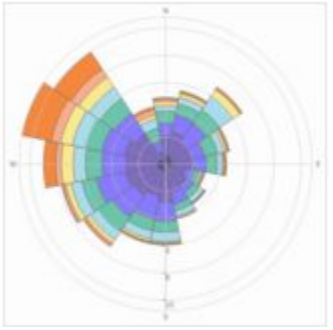
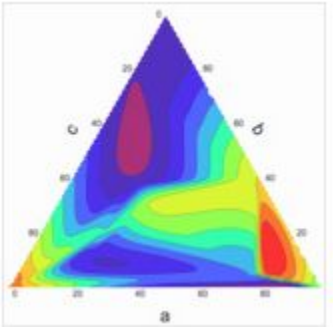
Basic Plotting with Plotly

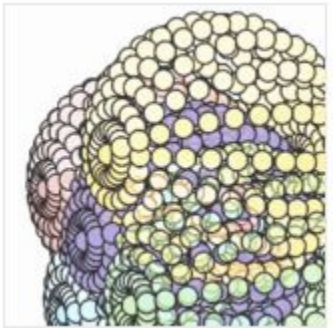
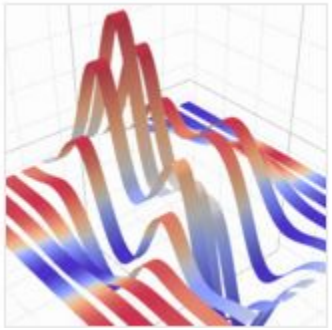
In [8]:
```
from plotly.graph_objs import Scatter
from plotly.offline import iplot

p = Scatter(x=iris.petalLength,
            y=iris.sepalLength,
            mode='markers')

iplot([p])
```

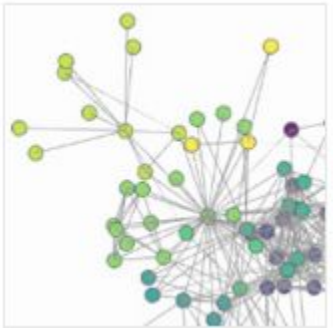


Plotly Gallery




Plotting with Plotly

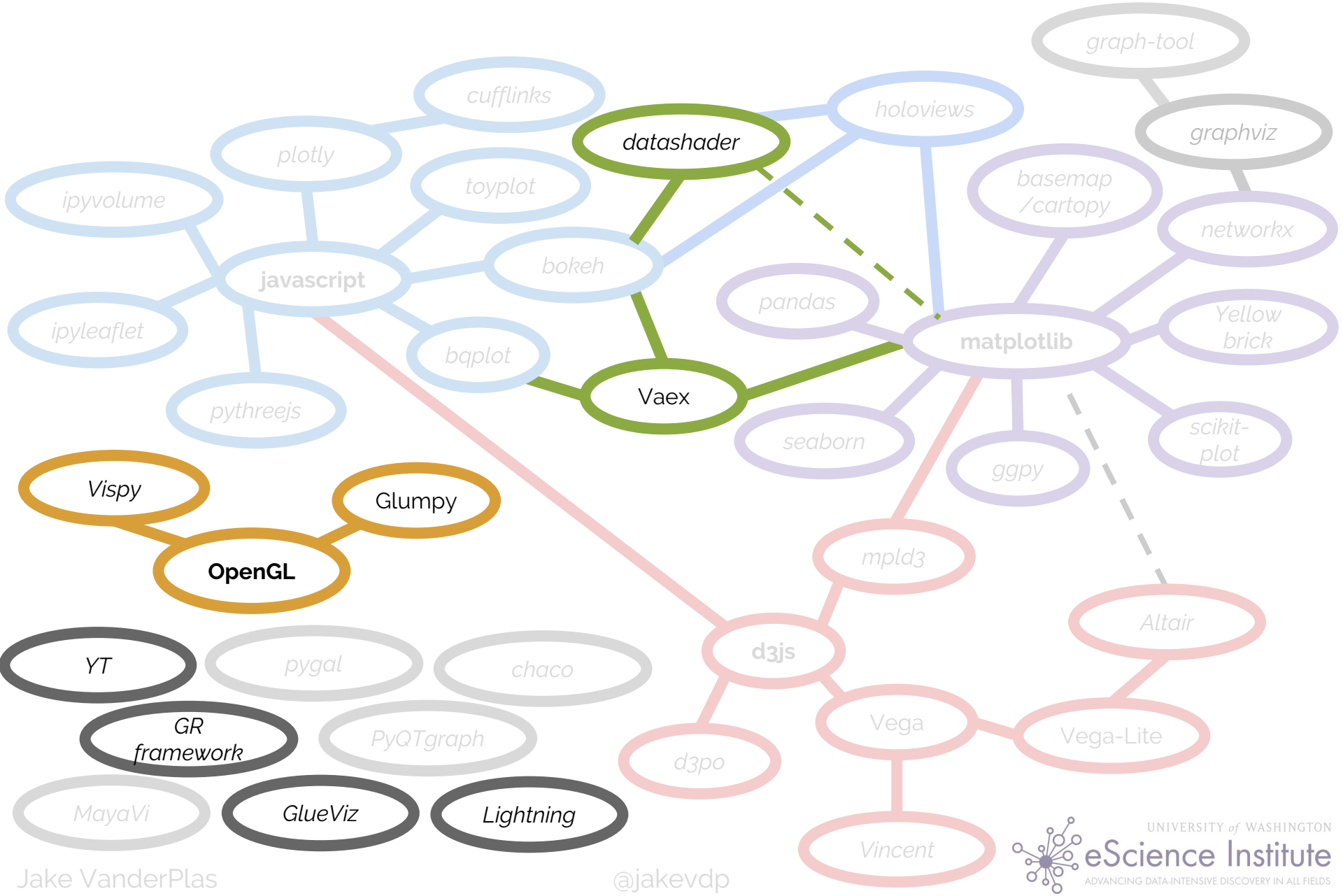
Advantages:

- Web view/interactivity
- Multi-language support
- 3D plotting capability
- Animation capability
- Geographical visualization

Disadvantages:

- Some features require a paid plan

Visualization for Larger Data . . .



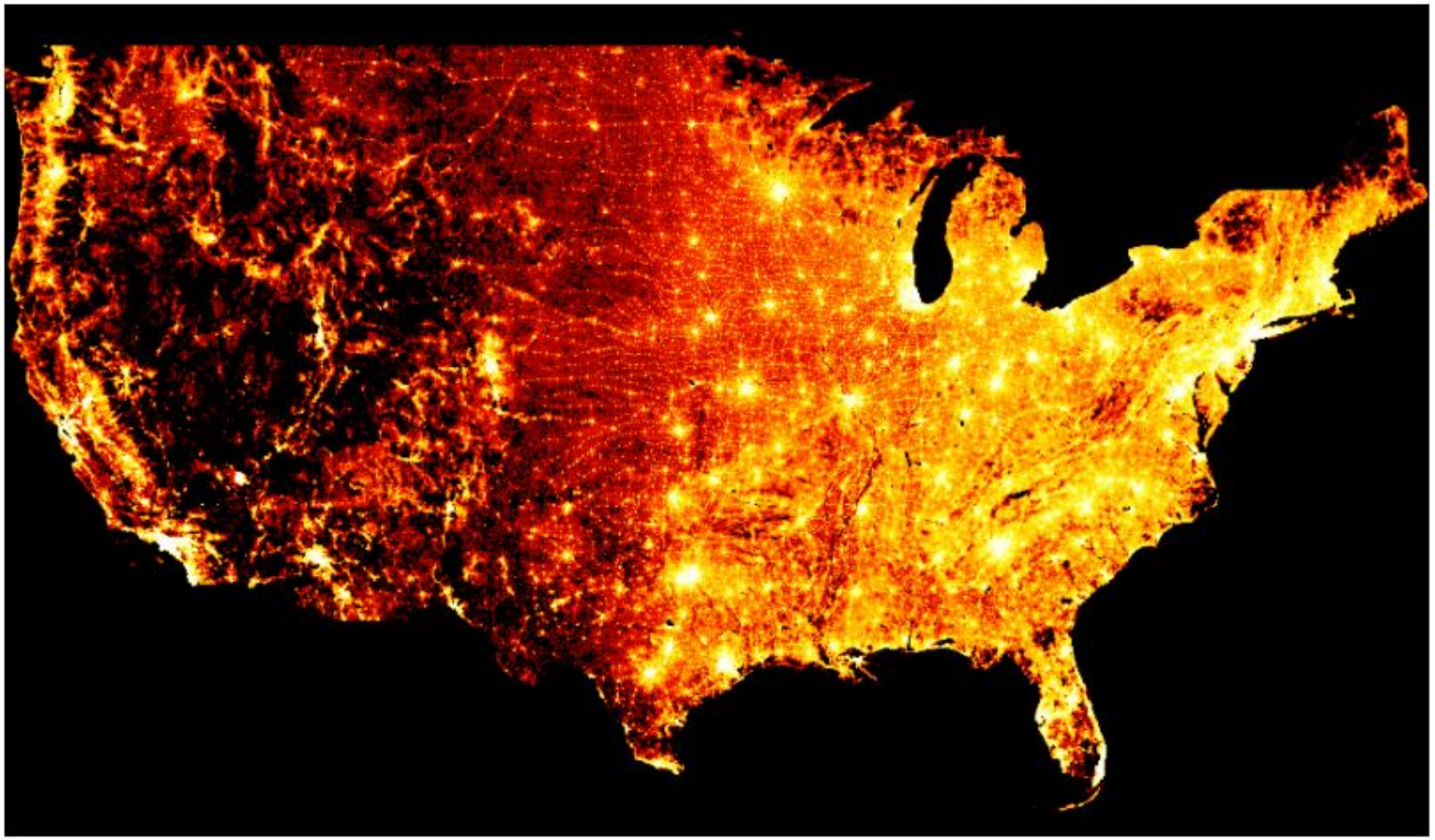
Visualization for Larger Data . . .

Datashader

Fast server-side engine for dynamic data aggregation

```
In [12]: from colorcet import fire
         export(tf.shade(agg, cmap = cm(fire,0.2), how='eq_hist'),"census_ds_fire_eq_hist")
```

Out[12]:

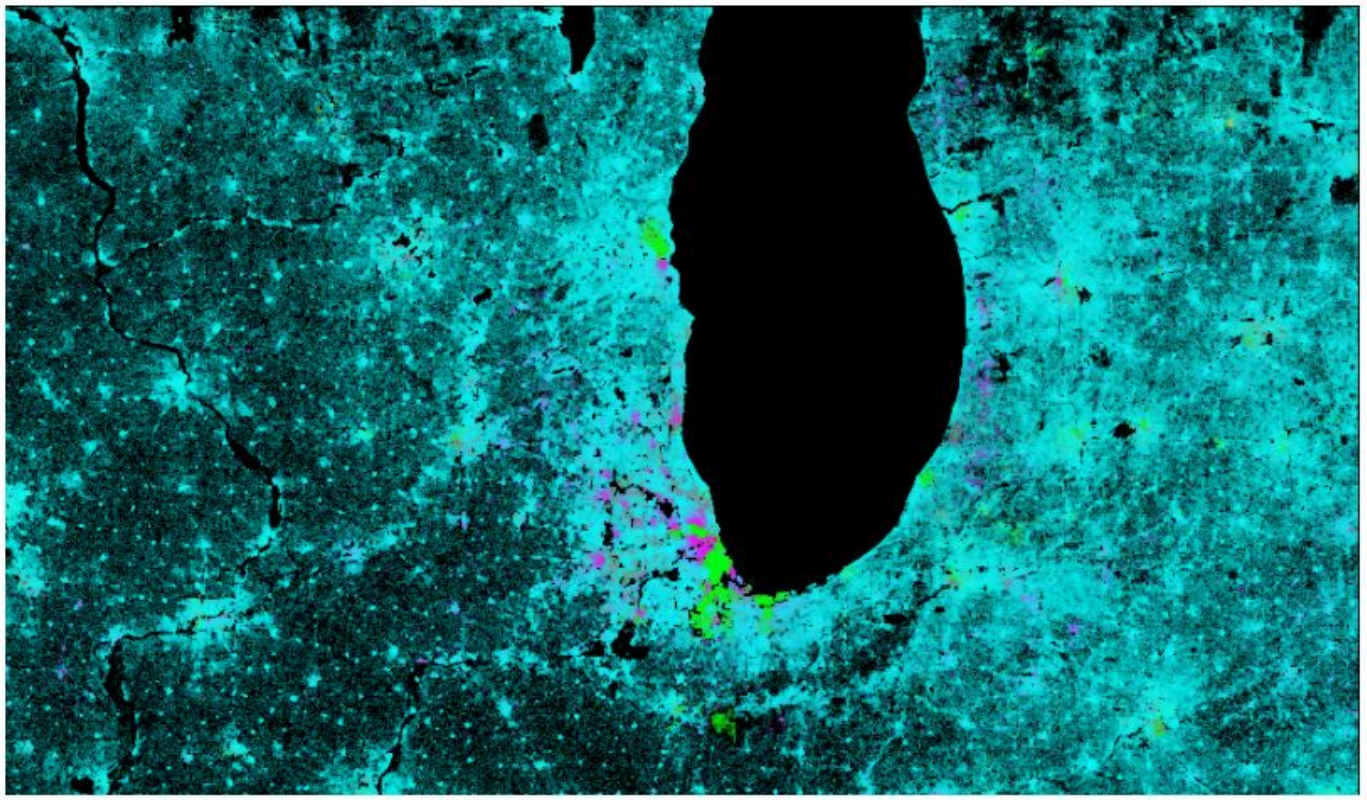


Datashader

- Compute layer that works with Bokeh
- Rather than sending *data* to the client, it aggregates data and sends *pixels*.
- Can handle interactive visualization of billions of rows.

```
In [19]: export(create_image(*LakeMichigan),"Zoom 1 - Lake Michigan")
```

Out[19]:

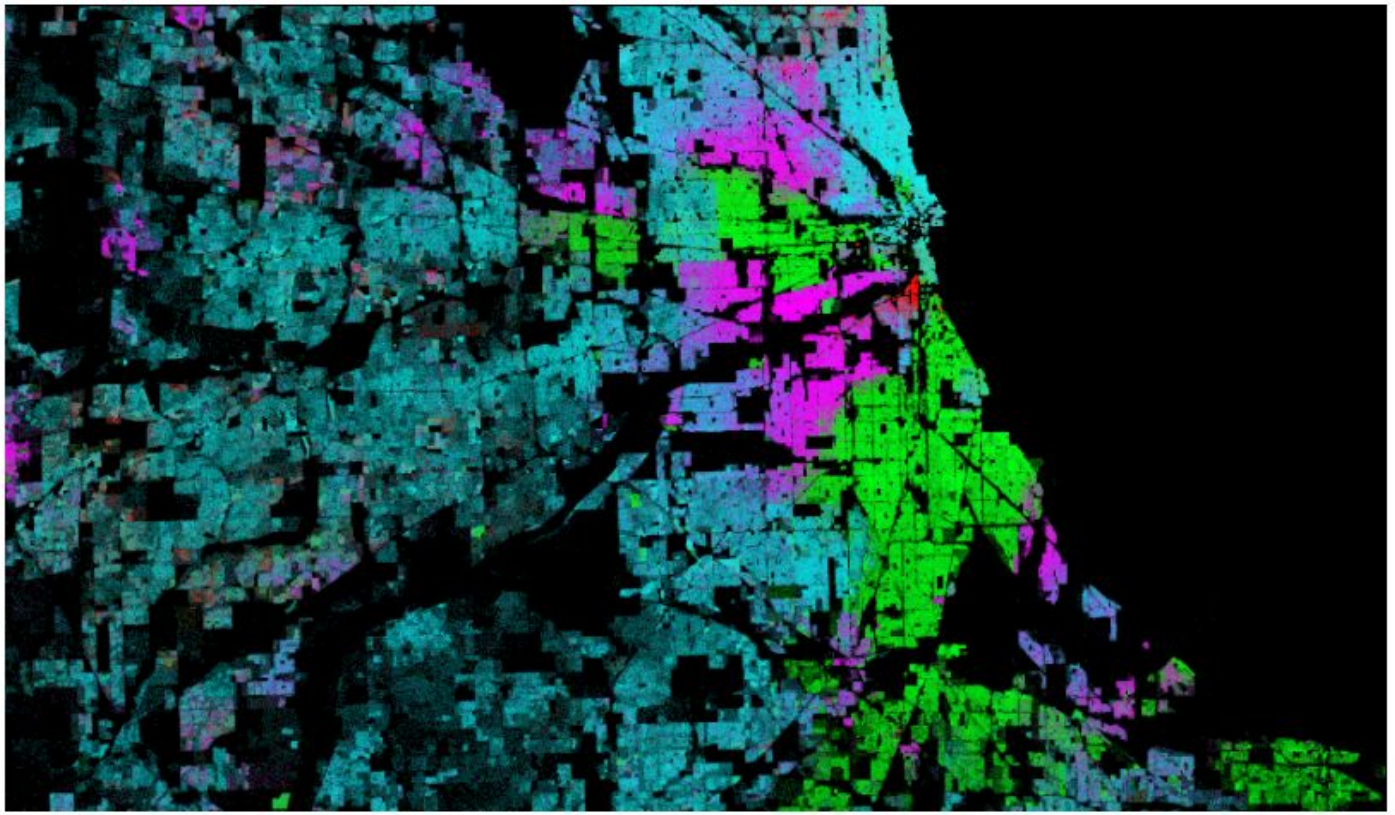


Datashader

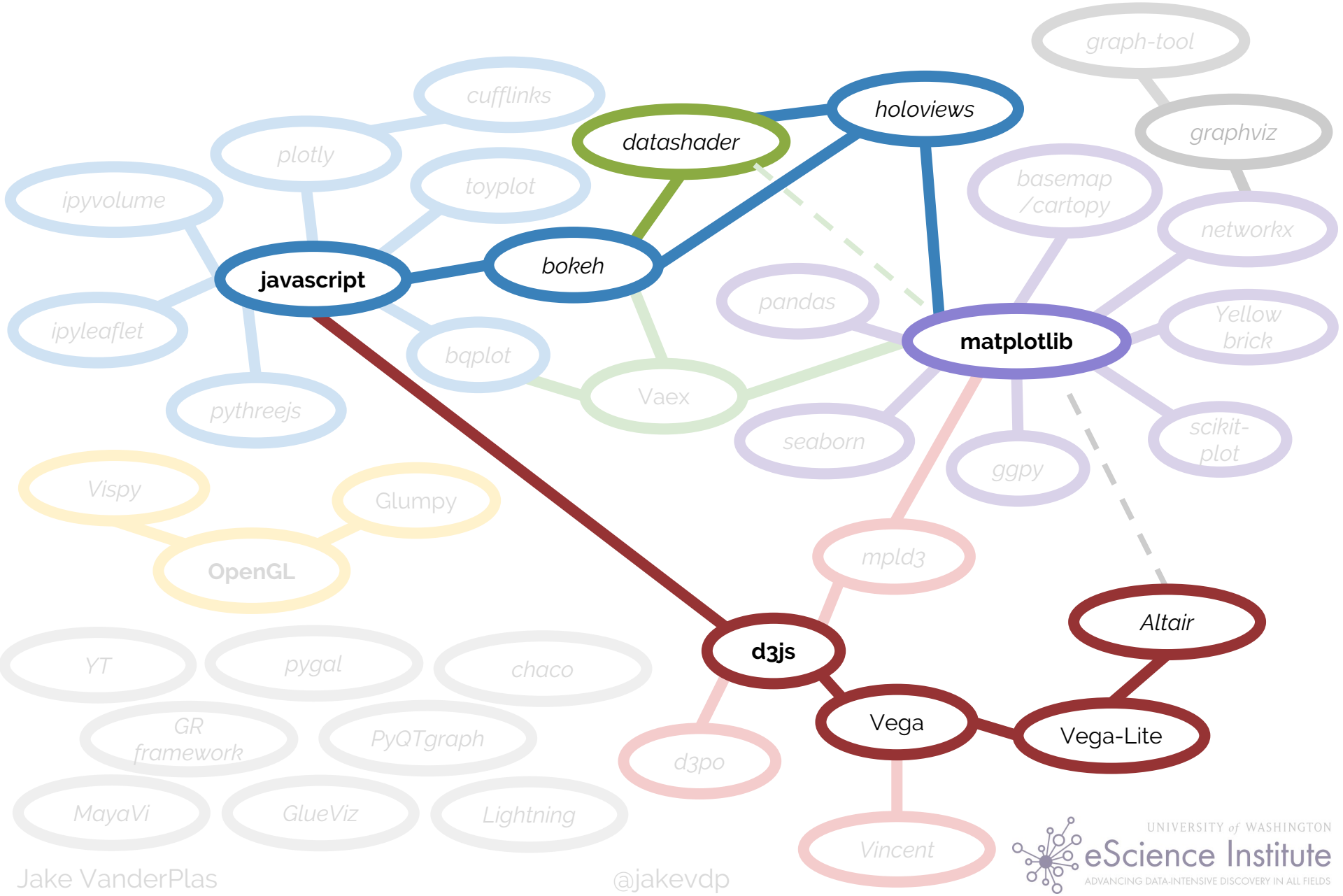
- Compute layer that works with Bokeh
- Rather than sending *data* to the client, it aggregates data and sends *pixels*.
- Can handle interactive visualization of billions of rows.

```
In [20]: export(create_image(*Chicago),"Zoom 2 - Chicago")
```

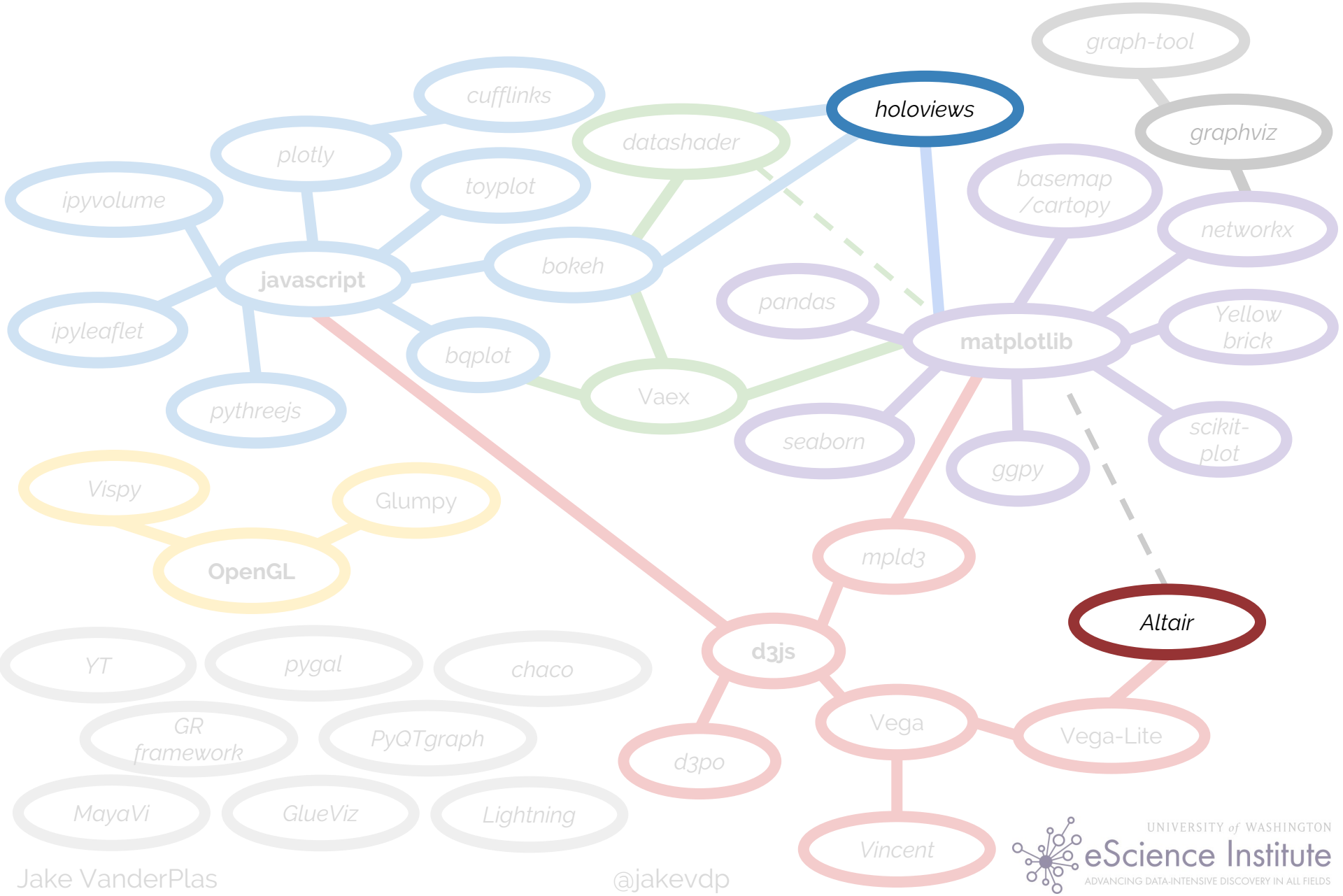
Out[20]:



Toward Declarative Visualization . . .



Toward Declarative Visualization . . .

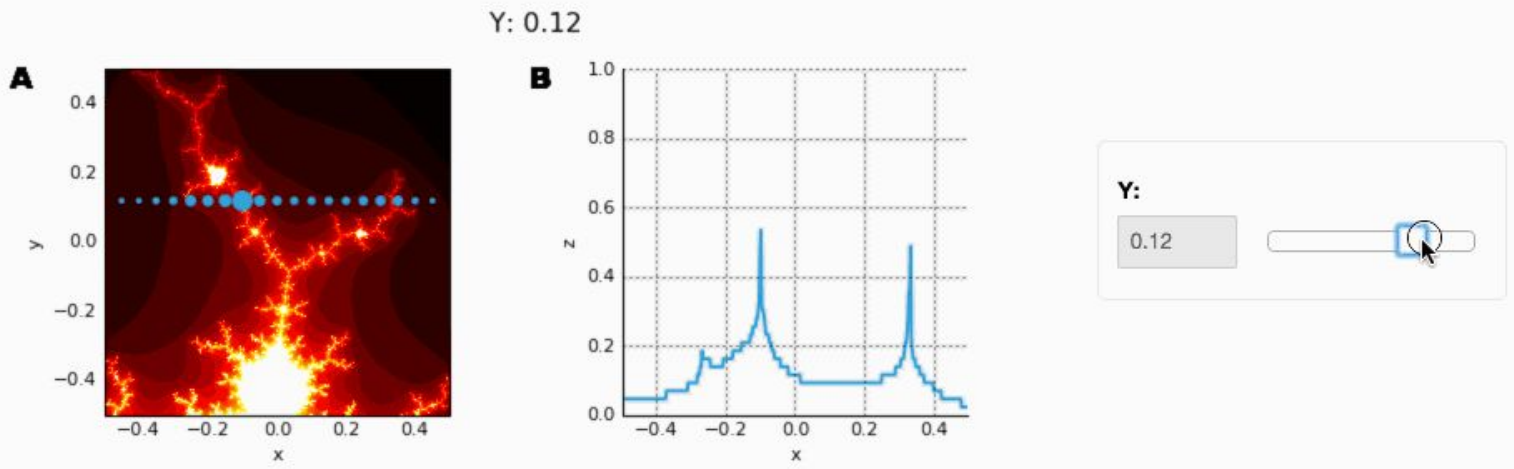


Holoviews

- Datasets themselves stored in objects that *automatically* produce intelligent visualizations
- Composition & Interactivity via operator overloading
- Renders to Bokeh, DataShader, and Matplotlib

```
In [3]: %%opts Points [scaling_factor=50] Contours (color='w')
        dots = np.linspace(-0.45, 0.45, 19)
        hv.HoloMap({y: (fractal * hv.Points(fractal.sample([(i,y) for i in dots])) + fractal.sample(y=y))
                    for y in np.linspace(-0.3, 0.3, 21)}, kdims=['Y']).collate().cols(2)
```

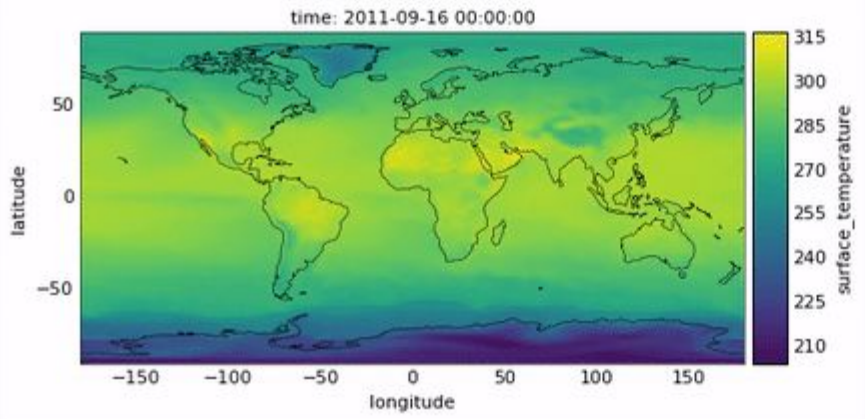
Out[3]:



Holoviews

- Also can handle geographic data & time-series

```
%%opts Image [colorbar=True] (cmap='viridis') Overlay [fig_size=200]
ensemble = xr.open_dataset('./sample-data/ensemble.nc')
dataset = gv.Dataset(ensemble, crs=crs.PlateCarree())
dataset.to(gv.Image, ['longitude', 'latitude'], ['surface_temperature'], ['time']) * gf.coastline()
```



Altair

What if instead of passing
around *pixels*, we pass around
visualization specifications plus *data?*

Altair

What if instead of passing
around *pixels*, we pass around
visualization specifications plus *data?*

“Declarative Visualization”

Altair

What if instead of passing
around *pixels*, we pass around
visualization specifications plus *data?*

“Declarative Visualization”

Declarative Visualization: Viz for data science

Imperative

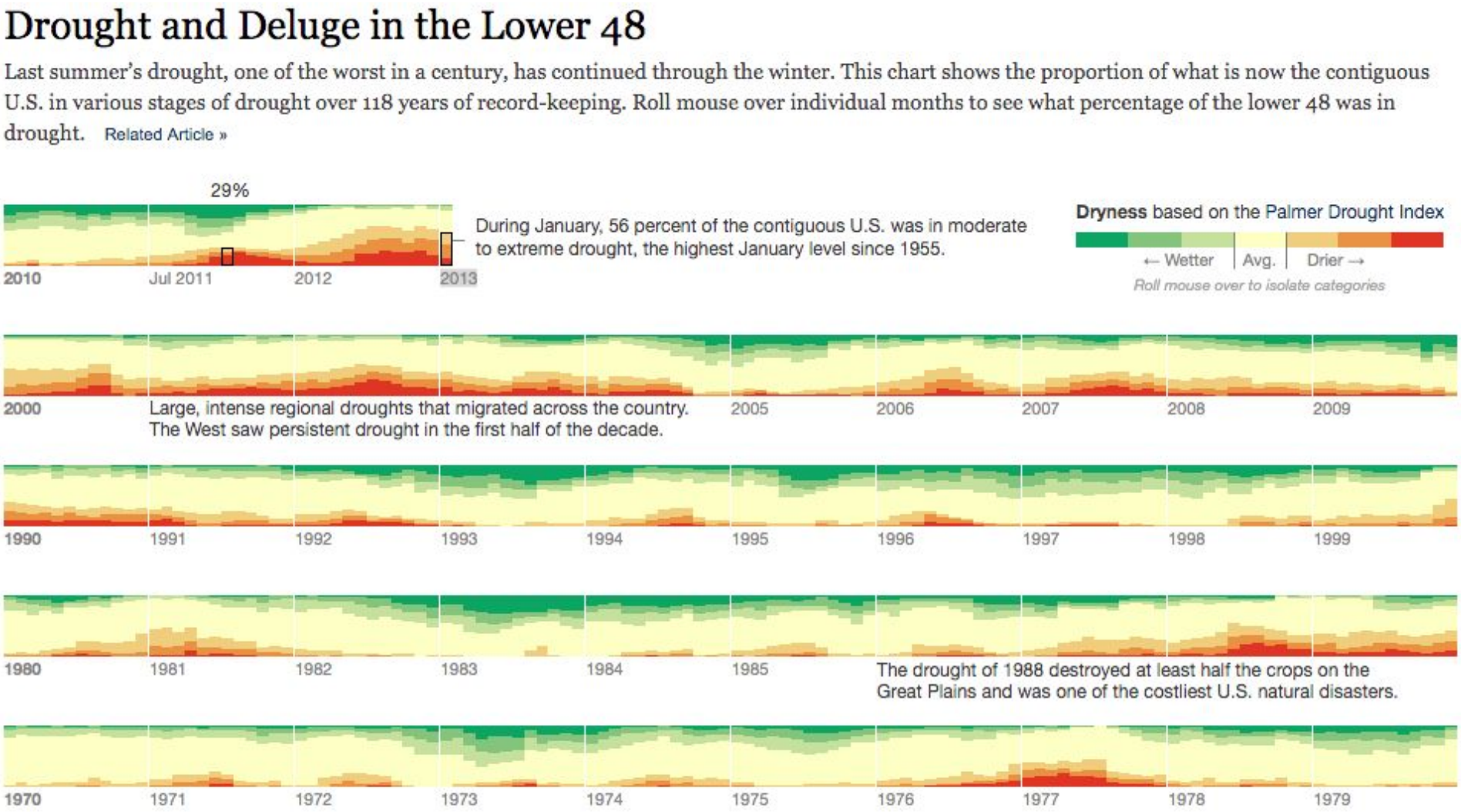
- Specify *How* something should be done.
- Must manually specify plotting steps
- Specification & Execution intertwined.

Declarative

- Specify *What* should be done
- Details determined automatically
- Separates Specification from Execution

Declarative visualization lets you think about **data** and **relationships**, rather than incidental details.

From D3 to Altair . . .

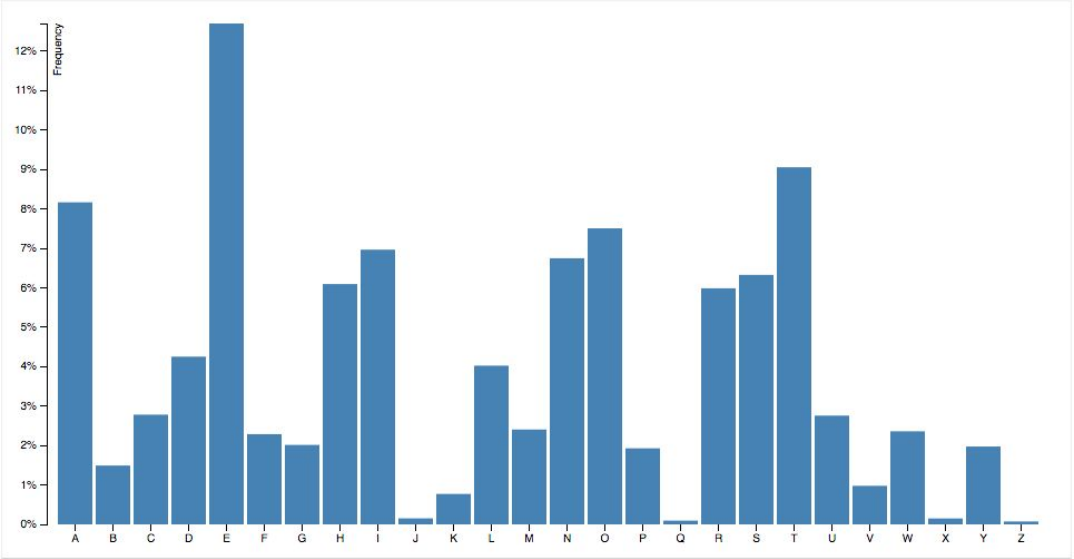


(link to live version)

But working in D3 can be challenging . . .

Bar Chart: d3

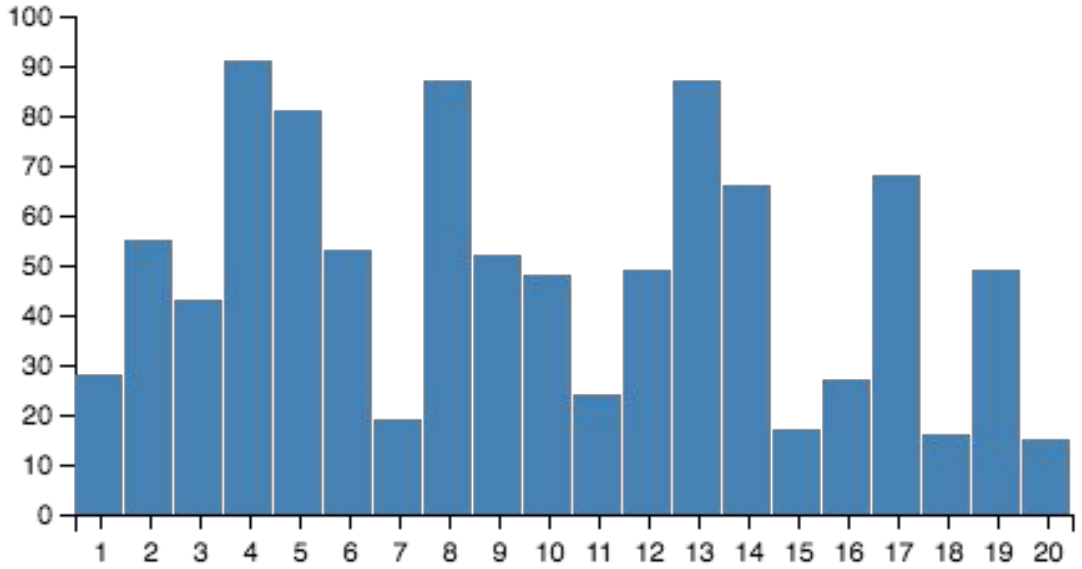
```
var margin = {top: 20, right: 20, bottom: 30, left: 40},
    width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;

var x = d3.scale.ordinal()
    .rangeRoundBands([0, width], .1);

var y = d3.scale.linear()
    .range([height, 0]);

var xAxis = d3.svg.axis()
    .scale(x)
    .orient("bottom");

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left")
    .ticks(10, "%");

var svg = d3.select("body").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

d3.tsv("data.tsv", type, function(error, data) {
  if (error) throw error;

  x.domain(data.map(function(d) { return d.letter; }));
  y.domain([0, d3.max(data, function(d) { return d.frequency; })]);

  svg.append("g")
      .attr("class", "x axis")
      .attr("transform", "translate(0," + height + ")")
      .call(xAxis);

  svg.append("g")
      .attr("class", "y axis")
      .call(yAxis)
    .append("text")
      .attr("transform", "rotate(-90)")
      .attr("y", 6)
      .attr("dy", ".71em")
      .style("text-anchor", "end")
      .text("Frequency");

  svg.selectAll(".bar")
      .data(data)
    .enter().append("rect")
      .attr("class", "bar")
      .attr("x", function(d) { return x(d.letter); })
      .attr("width", x.rangeBand())
      .attr("y", function(d) { return y(d.frequency); })
      .attr("height", function(d) { return height - y(d.frequency); });
});

function type(d) {
  d.frequency = +d.frequency;
  return d;
}
```



D3 is a Javascript package that streamlines manipulation of objects on a webpage.

Bar Chart: Vega

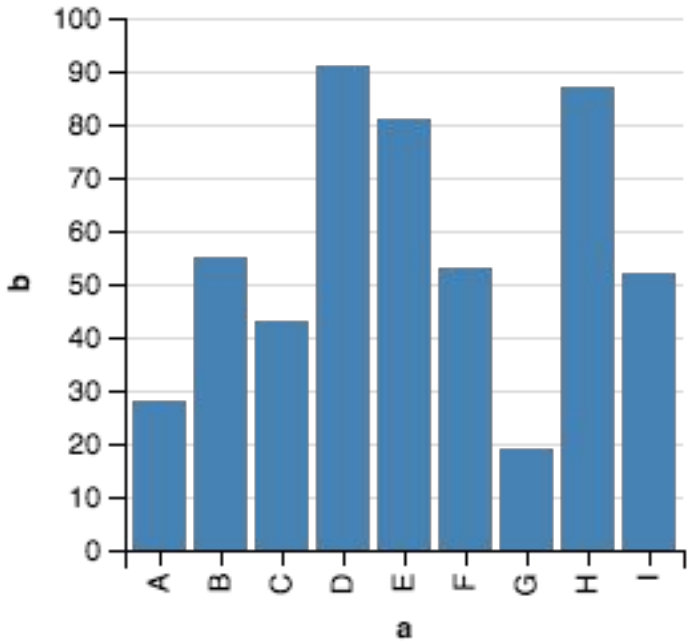
```
{
  "width": 400,
  "height": 200,
  "padding": {"top": 10, "left": 30, "bottom": 30, "right": 10},
  "data": [
    {
      "name": "table",
      "values": [
        {"x": 1,  "y": 28}, {"x": 2,  "y": 55},
        {"x": 3,  "y": 43}, {"x": 4,  "y": 91},
        {"x": 5,  "y": 81}, {"x": 6,  "y": 53},
        {"x": 7,  "y": 19}, {"x": 8,  "y": 87},
        {"x": 9,  "y": 52}, {"x": 10, "y": 48},
        {"x": 11, "y": 24}, {"x": 12, "y": 49},
        {"x": 13, "y": 87}, {"x": 14, "y": 66},
        {"x": 15, "y": 17}, {"x": 16, "y": 27},
        {"x": 17, "y": 68}, {"x": 18, "y": 16},
        {"x": 19, "y": 49}, {"x": 20, "y": 15}
      ]
    }
  ],
  "scales": [
    {
      "name": "x",
      "type": "ordinal",
      "range": "width",
      "domain": {"data": "table", "field": "x"}
    },
    {
      "name": "y",
      "type": "linear",
      "range": "height",
      "domain": {"data": "table", "field": "y"},
      "nice": true
    }
  ],
  "axes": [
    {"type": "x", "scale": "x"},
    {"type": "y", "scale": "y"}
  ],
  "marks": [
    {
      "type": "rect",
      "from": {"data": "table"},
      "properties": {
        "enter": {
          "x": {"scale": "x", "field": "x"},
          "width": {"scale": "x", "band": true, "offset": -1},
          "y": {"scale": "y", "field": "y"},
          "y2": {"scale": "y", "value": 0}
        },
        "update": {
          "fill": {"value": "steelblue"}
        }
```



Vega is a detailed declarative specification for visualizations, built on D3.

```
{
  "description": "A simple bar chart with embedded data.",
  "data": {
    "values": [
      {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
      {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
      {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
    ]
  },
  "mark": "bar",
  "encoding": {
    "x": {"field": "a", "type": "ordinal"},
    "y": {"field": "b", "type": "quantitative"}
  }
}
```

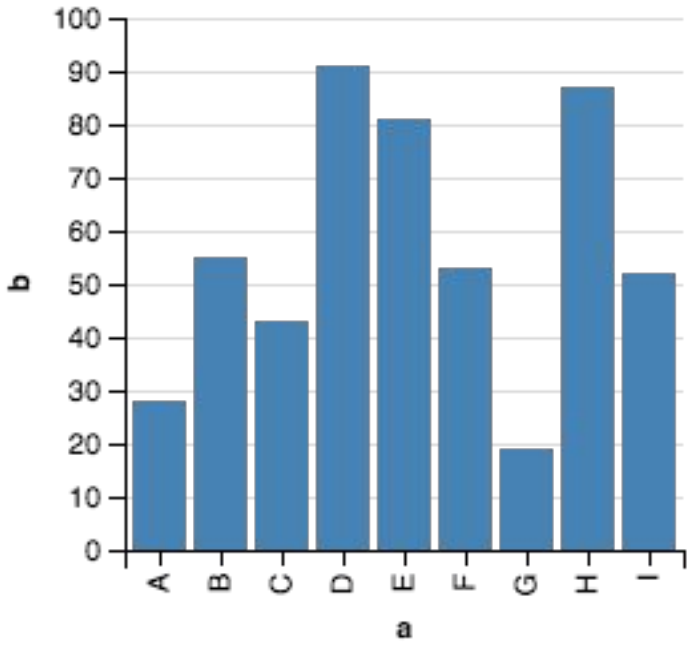
Bar Chart: Vega-Lite



Vega-Lite is a simpler declarative specification aimed at statistical visualization.

Bar Chart: Altair

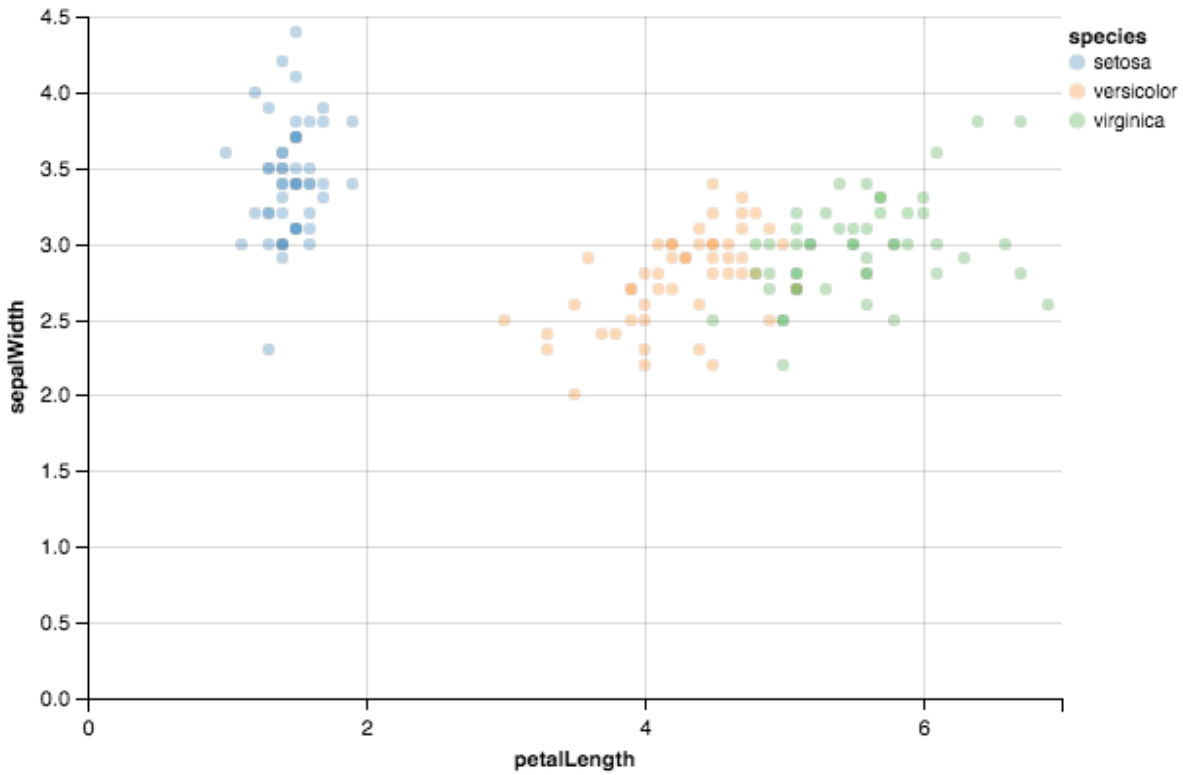
```
import pandas as pd
from altair import Chart

data = pd.DataFrame({'a': ['A', 'B', 'C', 'D', 'E',
                           'F', 'G', 'H', 'I'],
                     'b': [28, 55, 43, 91, 81,
                           53, 19, 87, 52]})

Chart(data).mark_bar().encode(
    x='a',
    y='b',
)
```



Altair is a Python API for creating Vega-Lite specifications.

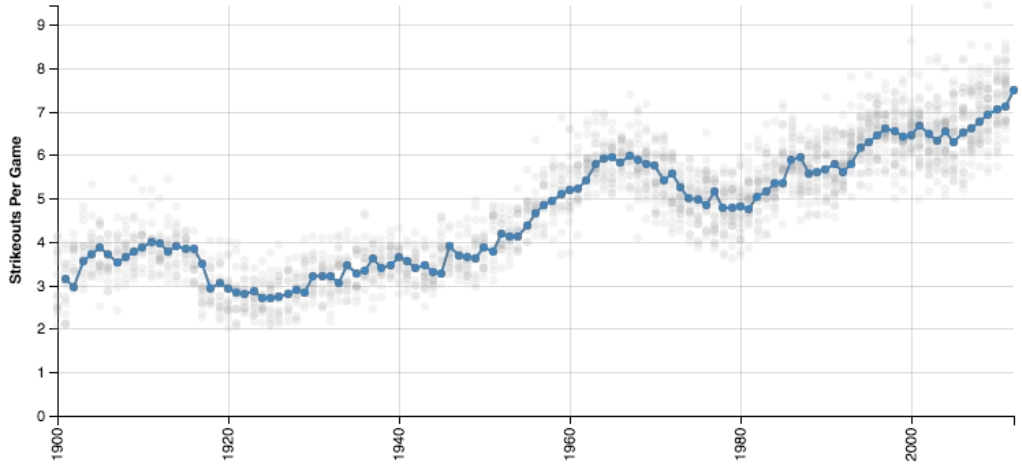
From Declarative *API* to declarative *Grammar*

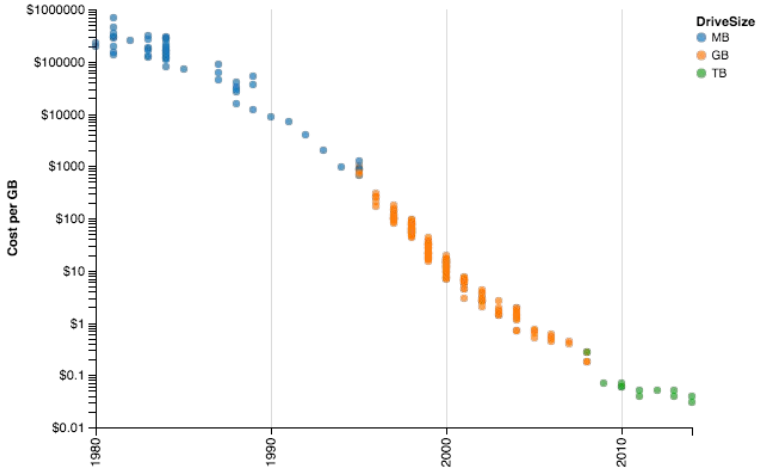
```
chart = Chart(data).mark_circle(
    opacity=0.3
).encode(
    x='petalLength:Q',
    y='sepalWidth:Q',
    color='species:N',
)
chart.display()
```

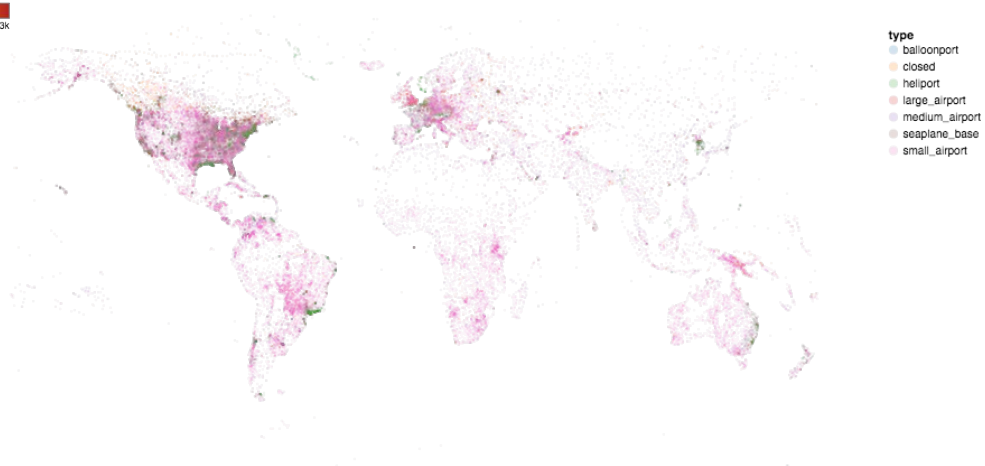


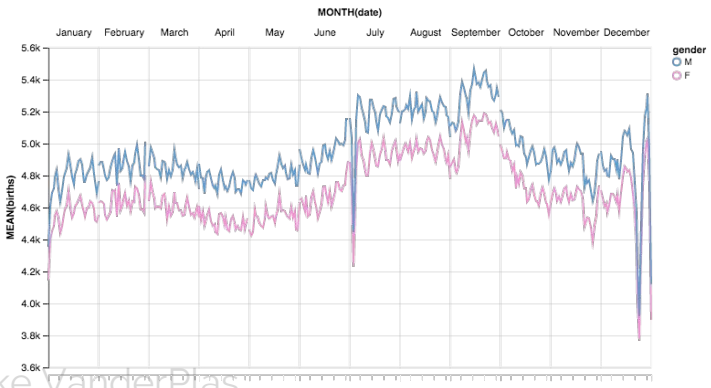
From Declarative *API* to declarative *Grammar*

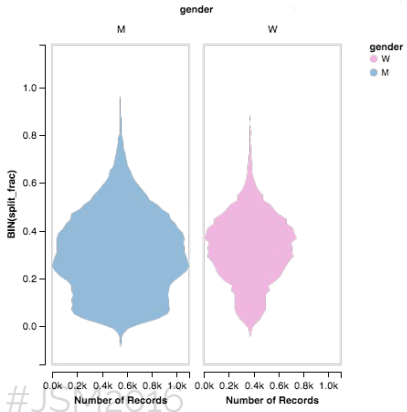
```
>>> chart.to_dict()

{'config': {'mark': {'opacity': 0.3}},
 'data':
   {'url': 'https://vega.github.io/vega-datasets/data/iris.json'},
 'encoding': {'color': {'field': 'species', 'type': 'nominal'},
              'x': {'field': 'petalLength', 'type': 'quantitative'},
              'y': {'field': 'sepalWidth', 'type': 'quantitative'}},
 'mark': 'circle'}
```







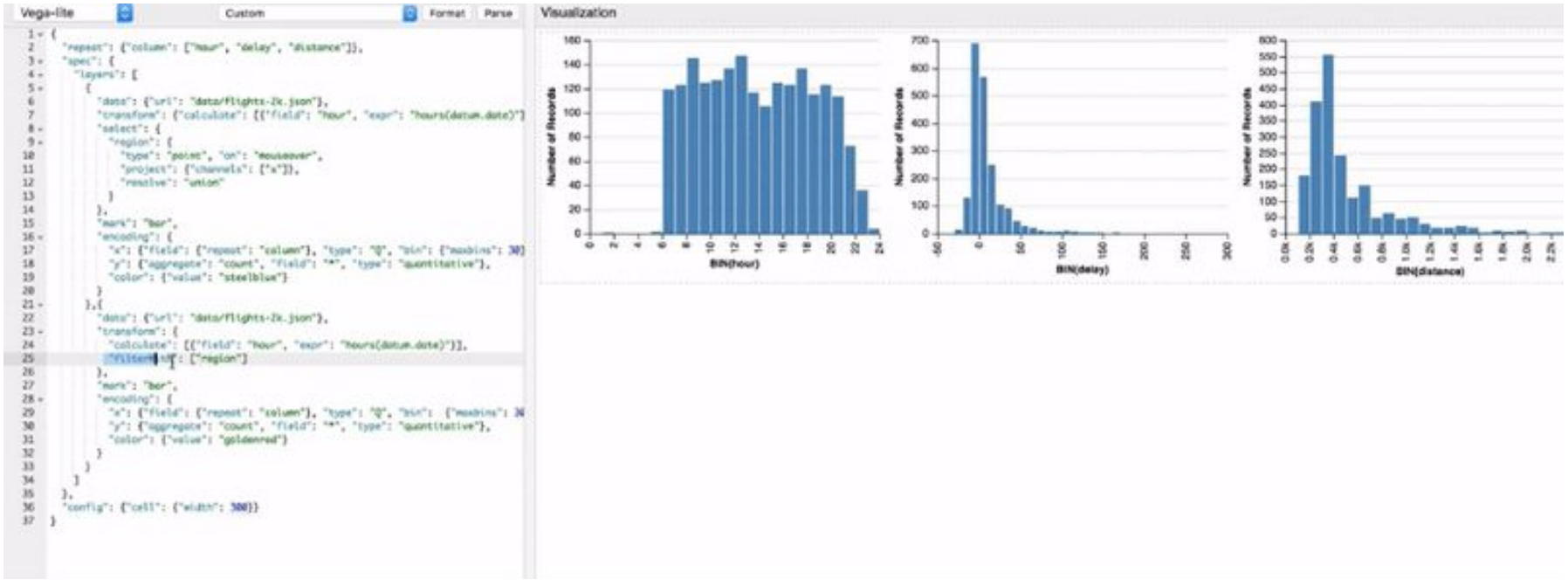




(Visualizations from jakevdp/altair-examples).

Coming Very Soon: Altair 2.0

- Includes a Grammar of Interaction



Try Altair:

```
$ conda install altair --channel conda-forge
```

or

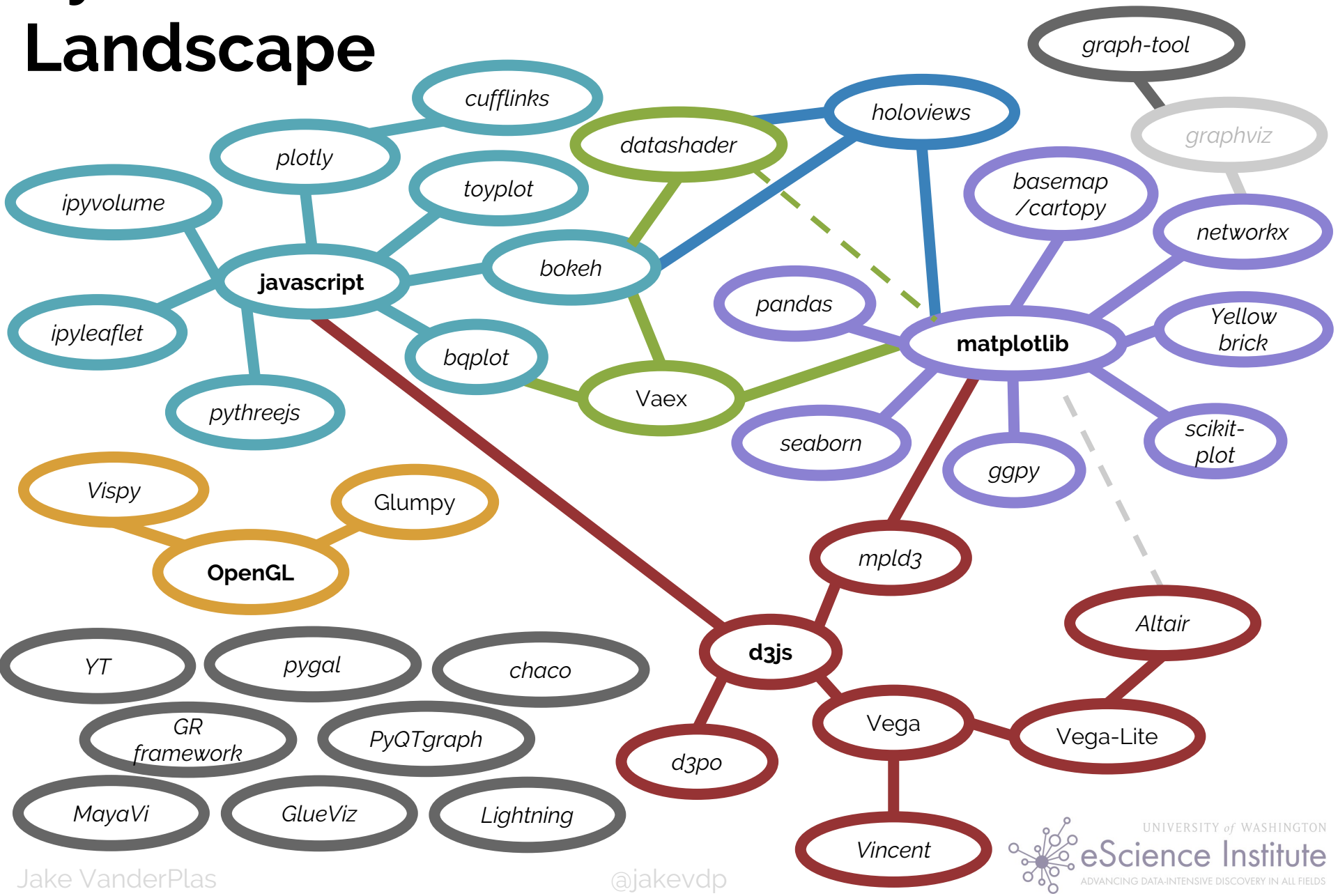
```
$ pip install altair
$ jupyter nbextension install --sys-prefix --py vega
```

For a Jupyter notebook tutorial, type

```
import altair
altair.tutorial()
```

http://github.com/ellisonbg/altair/

Python's Visualization Landscape



Thank You!

Email: jakevdp@uw.edu

Twitter: @jakevdp

Github: jakevdp

Web: http://vanderplas.com/

Blog: http://jakevdp.github.io/