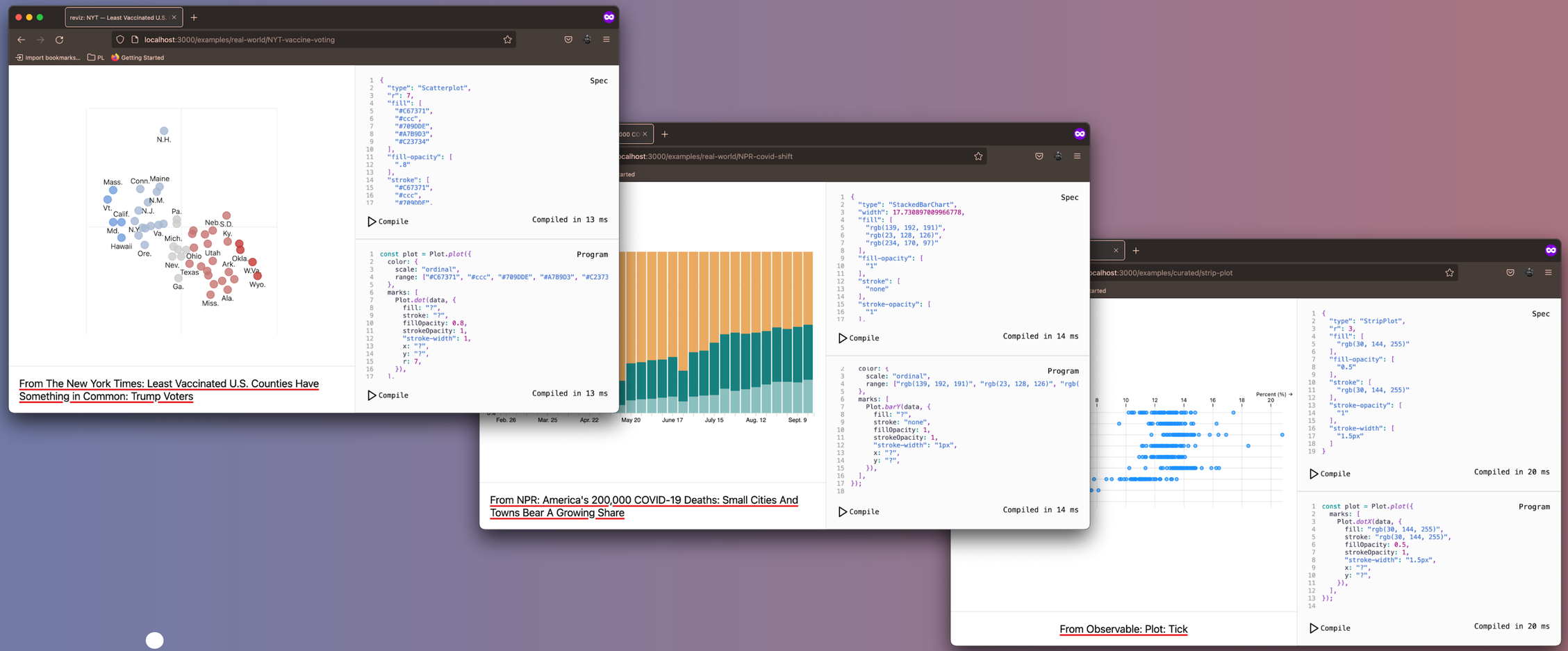


reviz

A lightweight engine for reverse engineering data visualizations from the DOM

Parker Ziegler // EPIC Meeting // March 9, 2022

What is this talk about?

A deep dive into a new

programming tool

for

data visualization authors

What is this talk (really) about?

The

power of examples

to act as

intent specification mechanisms

for data visualization authors

What is this talk (really) about?

How can

reverse engineering

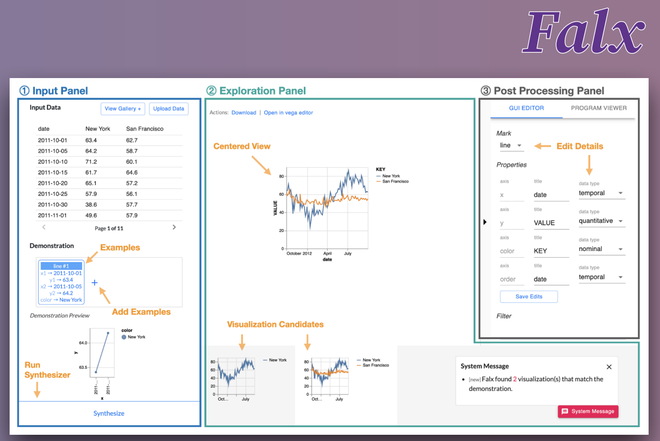
“in-the-wild” examples augment

existing techniques

authoring data visualizations?

What is this talk (really) about?

How can **reverse engineering** "in-the-wild" **examples** augment **existing techniques** for data visualization author?



LUX

reviz

Visualization recommendation

Data ····● **Visualization**

Visualization by demonstration

Data ····● **Visualization**

⋮

Demonstration

Visualization retargeting

Data ····● **Visualization**

⋮

"In-the-wild" Example

Roadmap

1.

The **promise** of (and **problem** with) examples in data visualization

2.

What is **reviz**? How does it address the **problem**?

3.

Questions and **curiosities** for future work

Roadmap

1.

The **promise** of
(and **problem** with)
examples in data
visualization

"Examples inspire by showing what's possible; examples demonstrate specific techniques; and ***examples are building blocks which help people get started****."*

Mike Bostock
"10 Years of Open Source Visualization"

“Developers use different strategies in learning APIs …
Most developers prefer the strategy of looking at examples [12, 21, 28, 33, 36, 38, 40, 41, 45, 51, 52].”

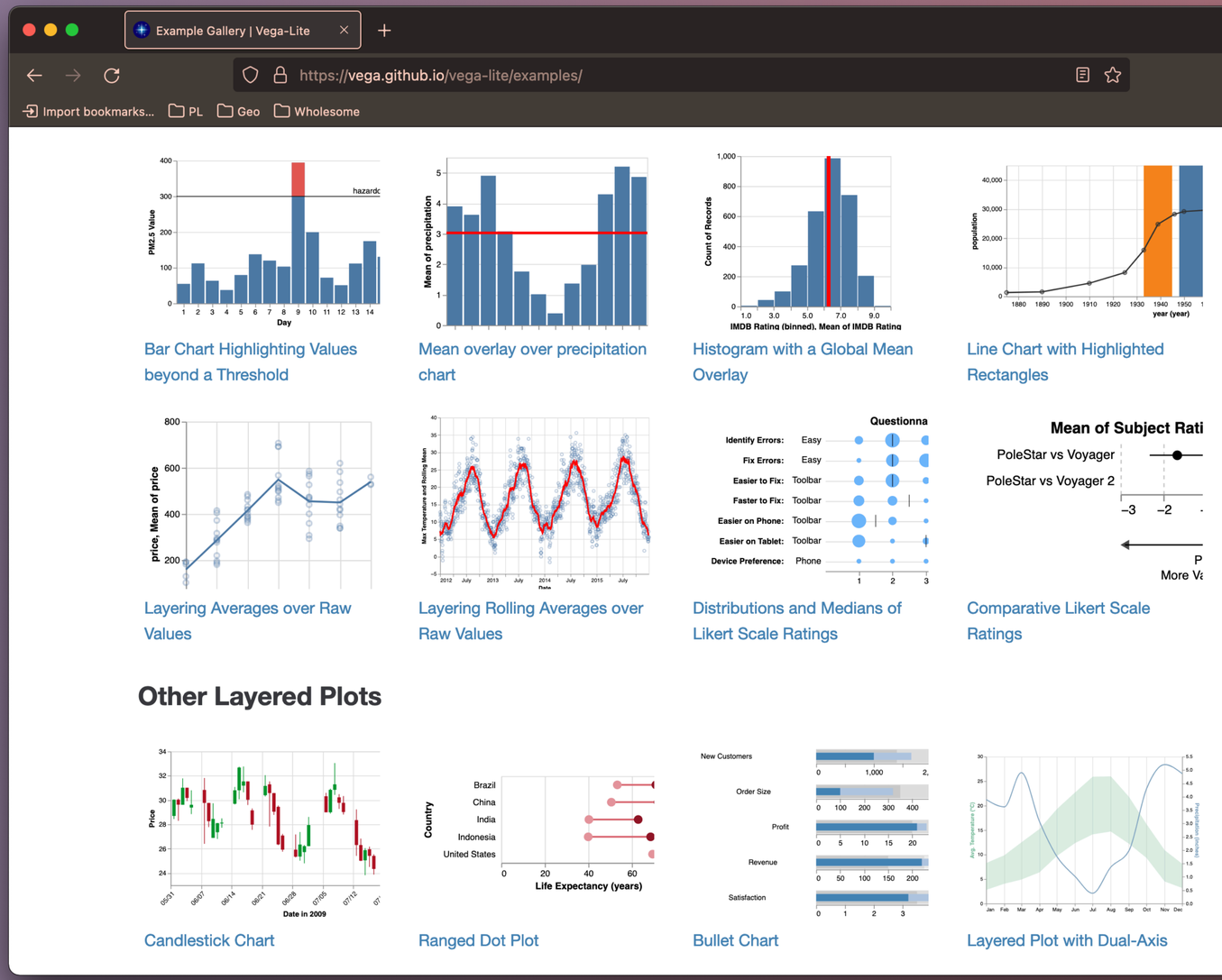
Kyle Thayer, Sarah E. Chasins, Amy J. Ko
“A Theory of Robust API Knowledge”
TOCE '21: ACM Transactions on Computing Education

Examples abound in the data visualization community

vega-lite has

189 examples

on its website

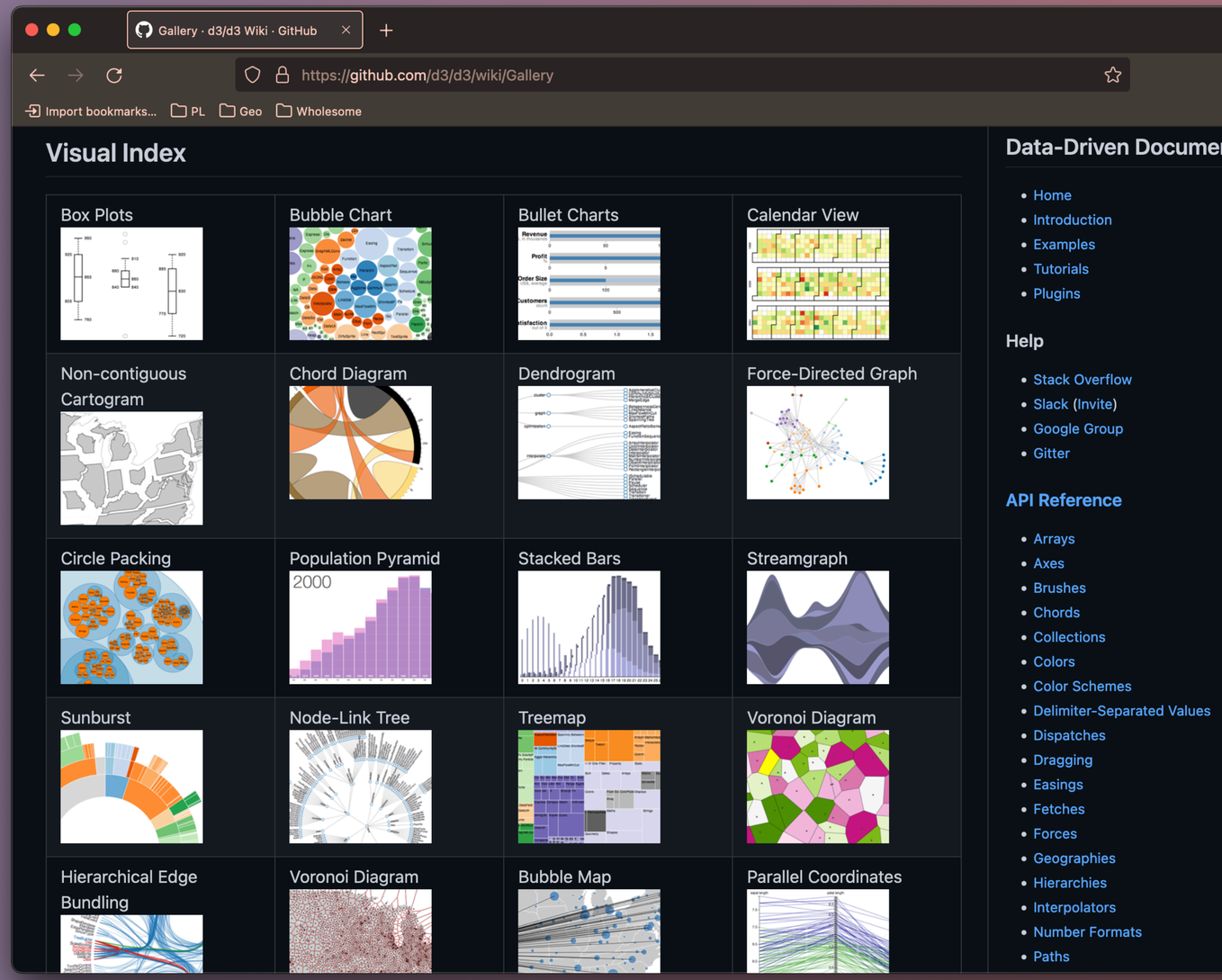


Examples abound in the data visualization community

D3 has

363 examples

on its GitHub Wiki

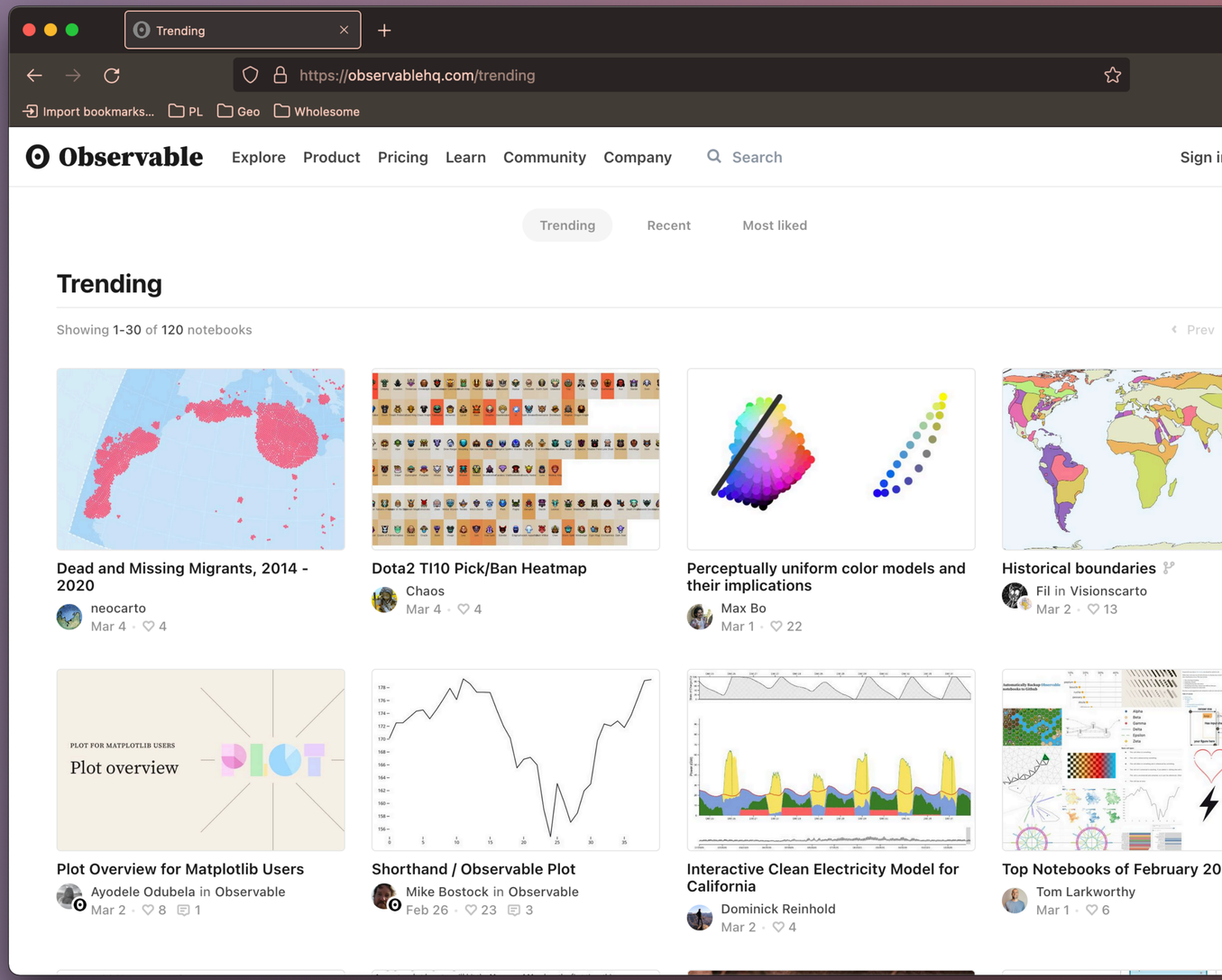


Examples abound in the data visualization community

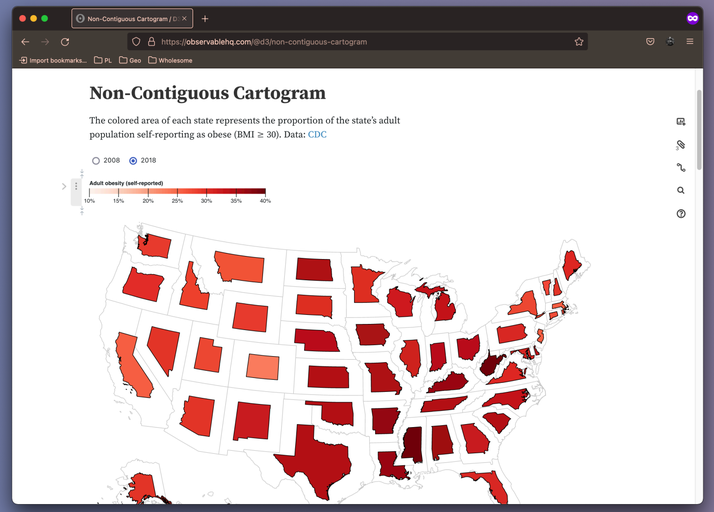
Observable has

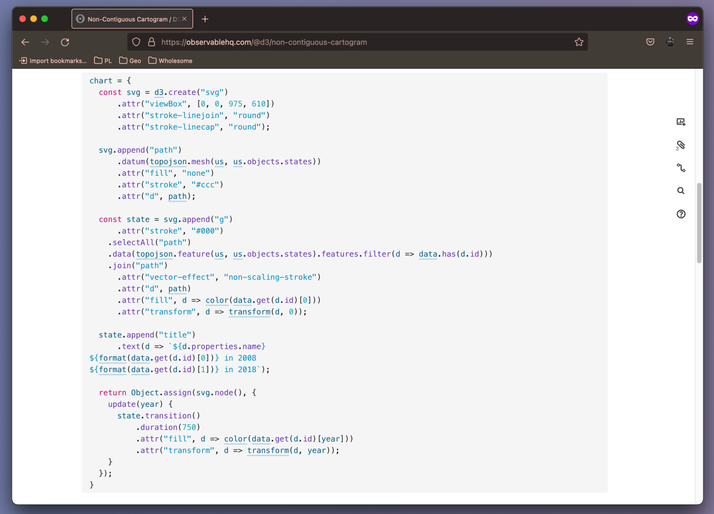
thousands

of user-built examples



Why are these examples effective?



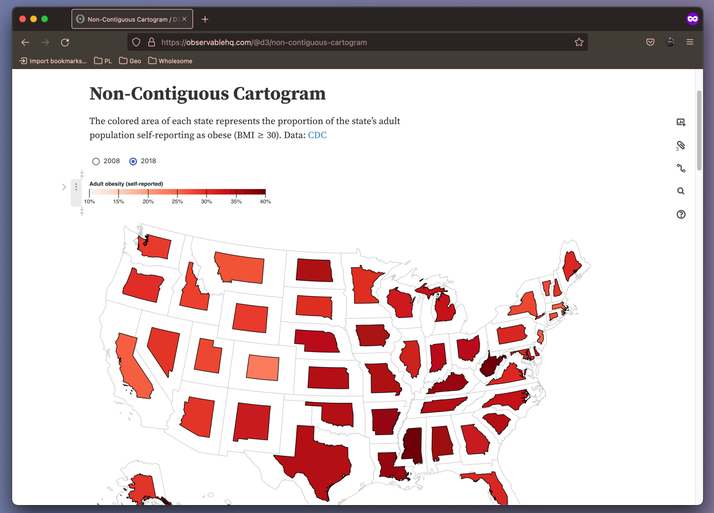


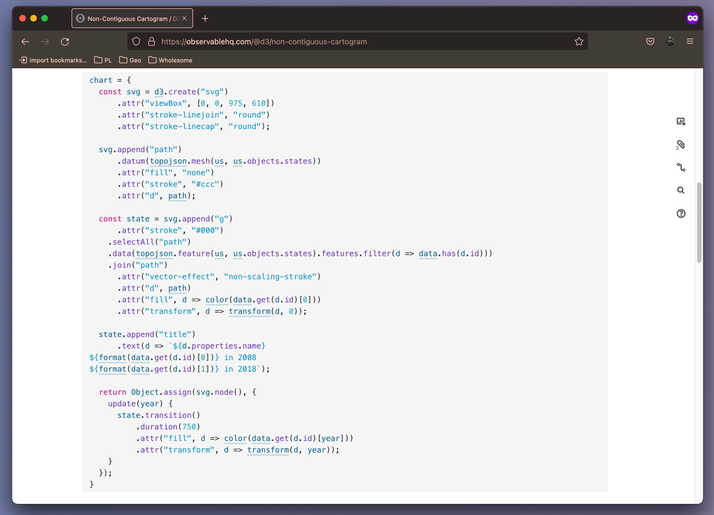
> "They're very lightweight, they're informal, they're something you can do in 15 minutes. They don't have the ceremony or the overhead of doing published graphics."

Mike Bostock

"For Example", *EYEO Festival 2013*

Why are these examples effective?





- **Colocation** of source code with the visualization
- **Immediate** interaction between source code and visualization
- Ability to see a visualization **applied to your own data**

Why are these examples effective?

Ability to see a visualization **applied to your own data**

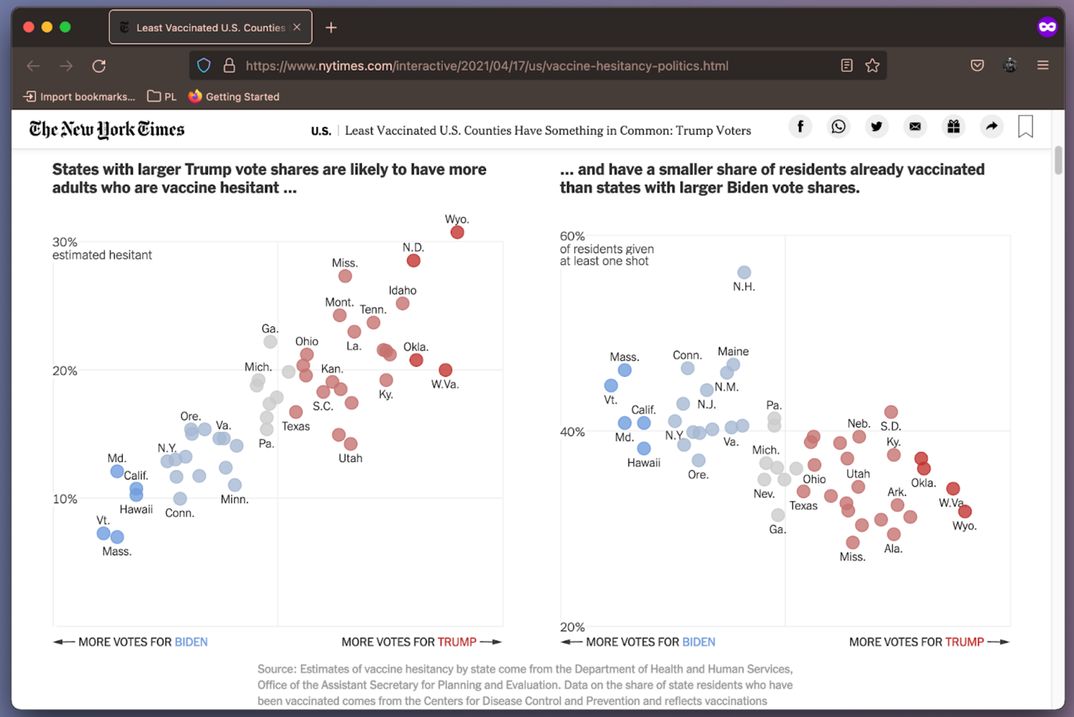
"Does this visualization encoding show me something **unexpected** about my data?"

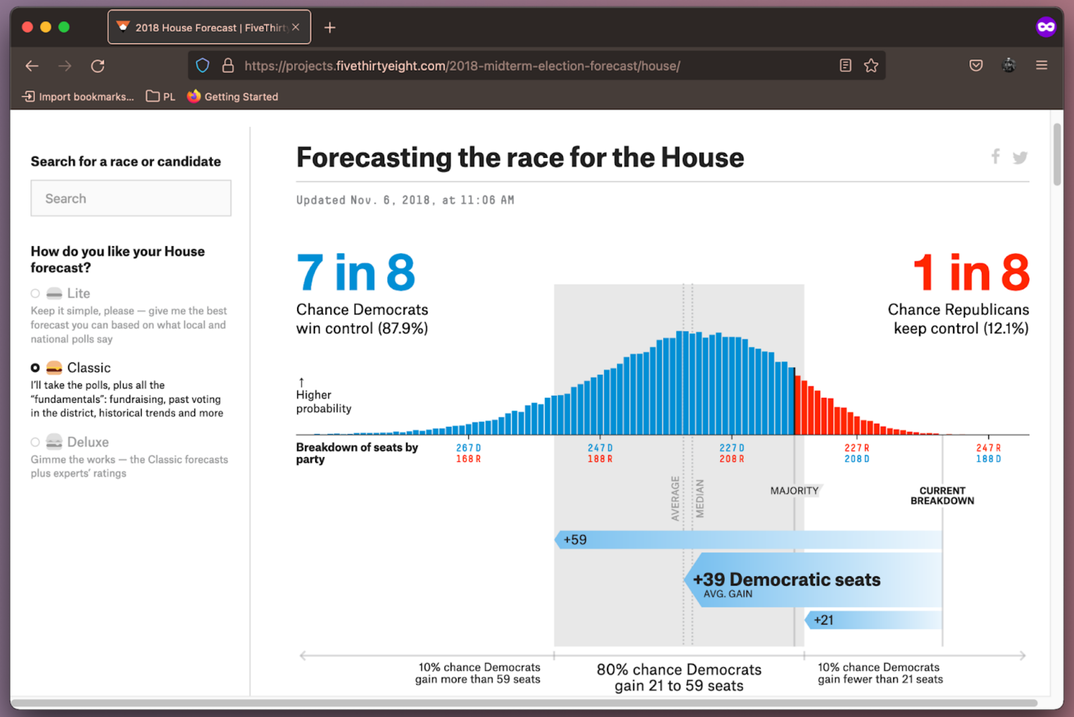
Exploratory

"Would an alternative encoding **better represent** what I want to show about my data?"

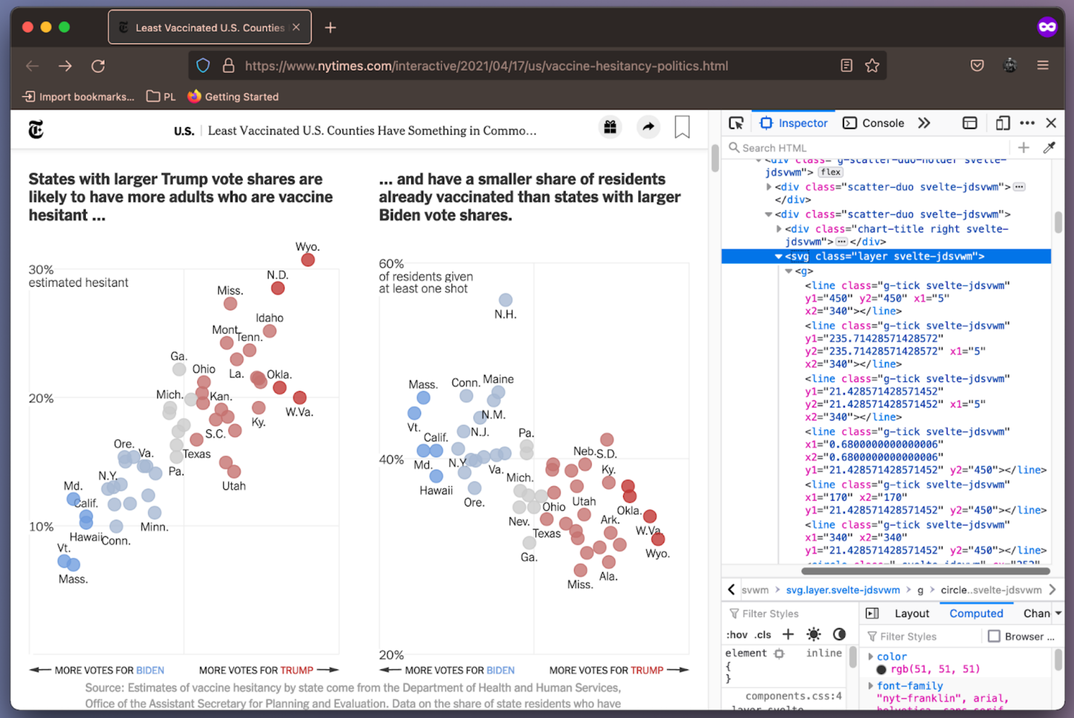
Explanatory

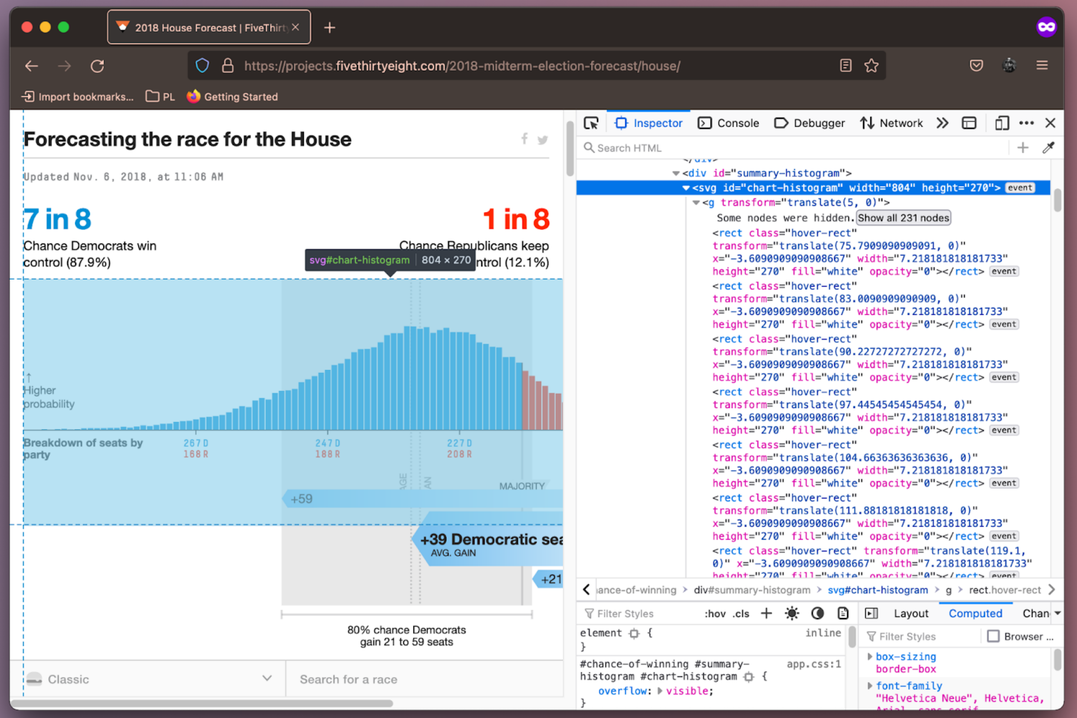
But what if the **code** powering an example **isn't visible**?





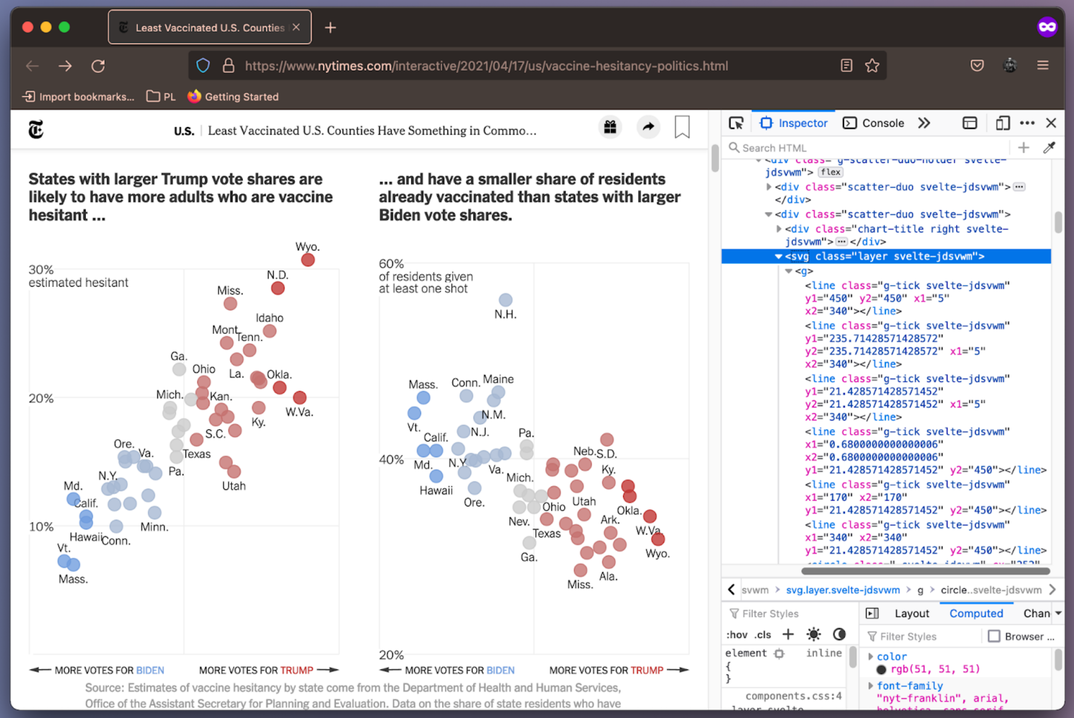
But what if the **code** powering examples isn't **visible**?

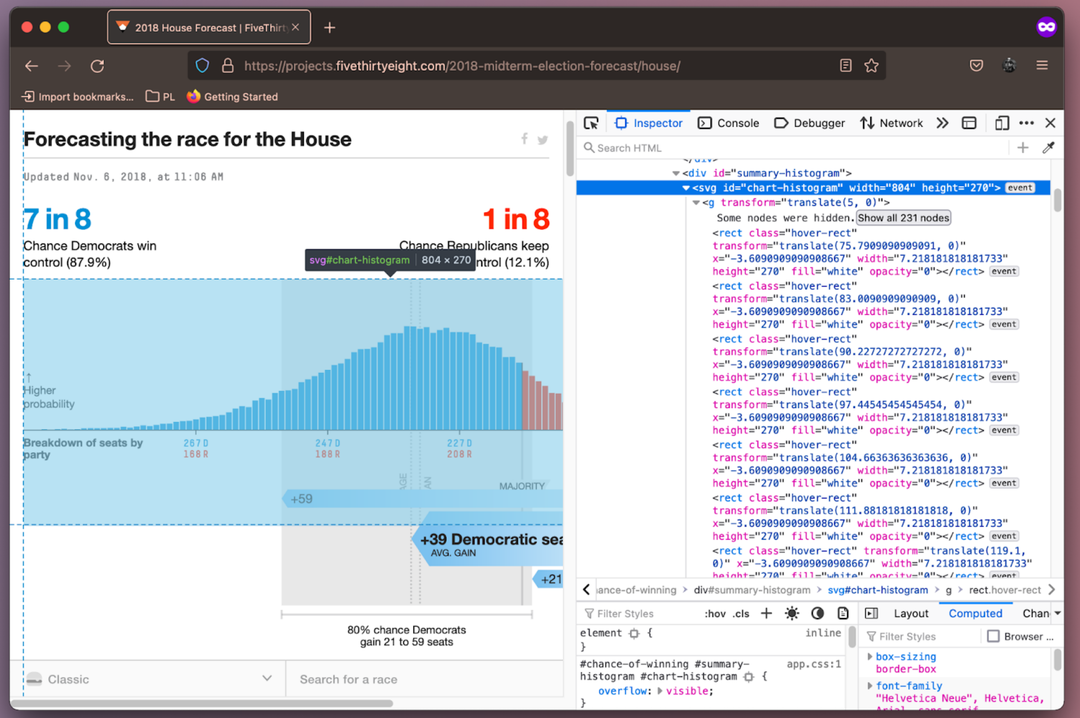




We *could* manually reverse engineer them.

But what if the **code** powering examples isn't **visible**?





Could we automatically

~~We *could* manually~~ reverse engineer them?

Roadmap

1.

The **promise** of
(and **problem** with)
examples in data
visualization

Roadmap

1.

The **promise** of (and **problem** with) examples in data visualization

2.

What is **reviz**? How does it address the **problem**?

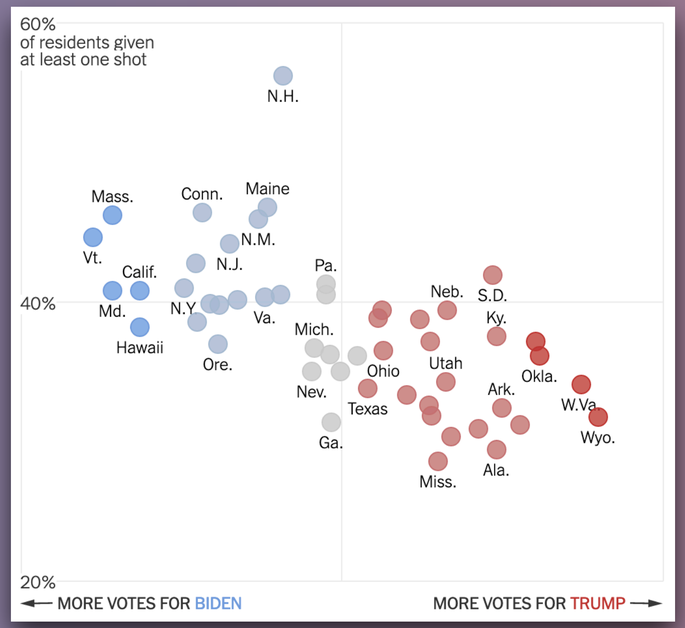
A “**compiler**” from **SVG subtrees** to **data visualizations**

A “**compiler**” from **SVG subtrees** to ~~**data visualizations**~~

partial programs in Observable Plot

How does it work?

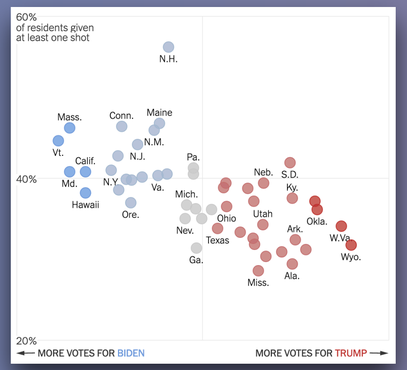
A user **identifies a visualization** they want to use for retargeting.



"Least Vaccinated U.S. Counties Have Something in Common: Trump Voters" from *The New York Times*
https://www.nytimes.com/interactive/2021/04/17/us/vaccine-hesitancy-politics.html

How does it work?

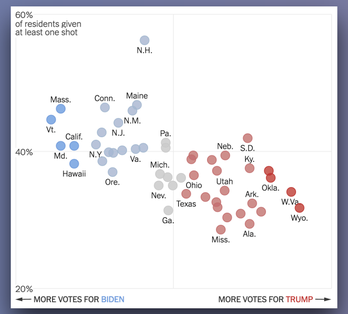
1.



They pass its `svg` **subtree** to **reviz.**

How does it work?

1.



2.

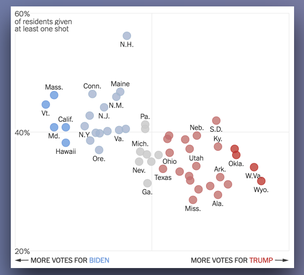
reviz generates a ***partial JavaScript program*** using Obervable's Plot library.

Program

```
const plot = Plot.plot({
  color: {
    type: "categorical",
    range: ["#C67371", "#ccc", "#709DDE", "#A7B9D3", "#C23734"],
  },
  marks: [
    Plot.dot(data, {
      x: "??",
      y: "??",
      r: 7,
      fill: "??",
      fillOpacity: 0.8,
      stroke: "??",
      strokeOpacity: 1,
      strokeWidth: 1,
    }),
  ],
});

```

How does it work?

1.

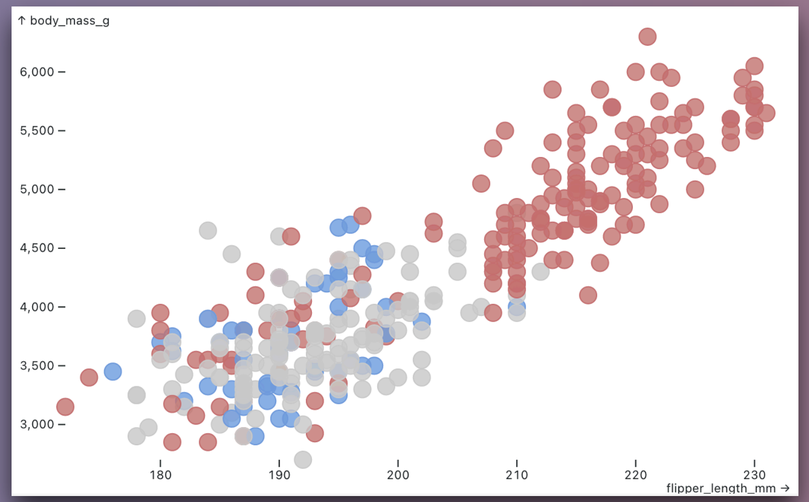


2.

3.

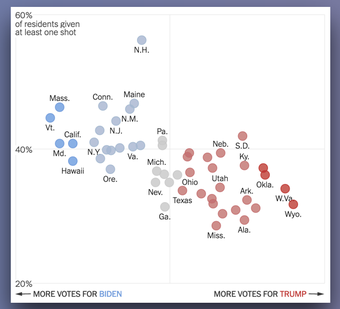
```
const plot = Plot.plot({                                    Program
  color: {
    type: "categorical",
    range: ["#C67371", "#ccc", "#709DDE", "#A7B9D3", "#C23734"],
  },
  marks: [
    Plot.dot(data, {
      x: "??",
      y: "??",
      r: 7,
      fill: "??",
      fillOpacity: 0.8,
      stroke: "??",
      strokeOpacity: 1,
      strokeWidth: 1,
    }),
  ],
});

```

The user **fills in the “holes”** in the partial program to get a retargeted visualization.



How does it work?

1.



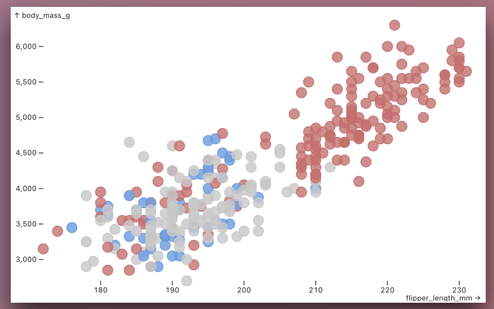
2.

reviz

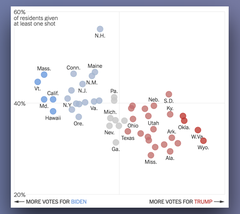
3.

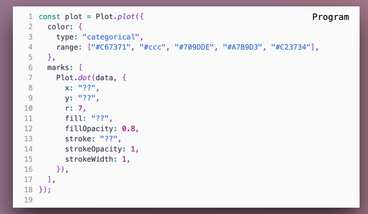
```
const plot = Plot.plot({                                    Program
  color: {
    type: "categorical",
    range: ["#C67371", "#ccc", "#709DDE", "#A7B9D3", "#C23734"],
  },
  marks: [
    Plot.dot(data, {
      x: "??",
      y: "??",
      r: 7,
      fill: "??",
      fillOpacity: 0.8,
      stroke: "??",
      strokeOpacity: 1,
      strokeWidth: 1,
    }),
  ],
});
```

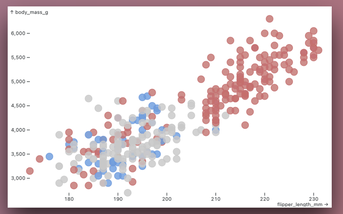
4.



How does it work?





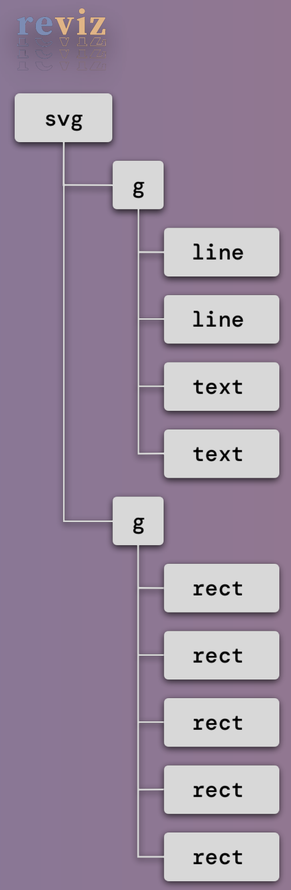


What **actually happens** in this step?

What actually happens in this step?

reviz

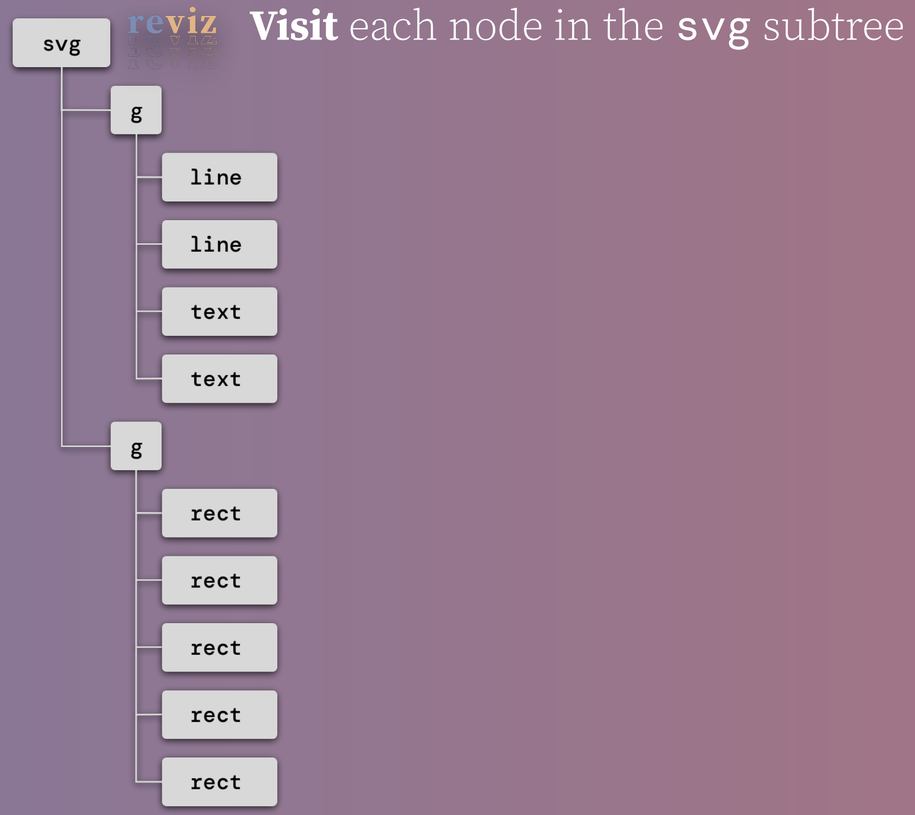
It all starts with
a **walk.**



What actually happens in this step?

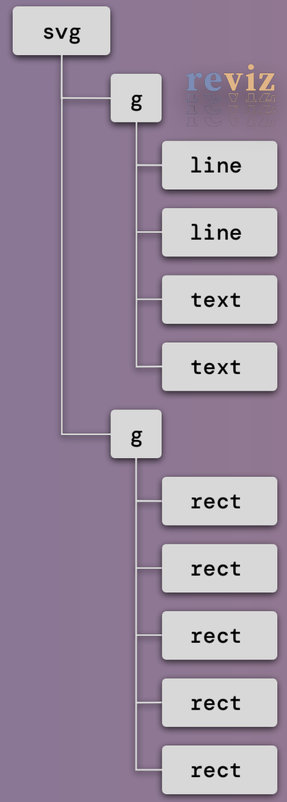
reviz

It all starts with
a **walk.**



What actually happens in this step?

It all starts with
a **walk.**

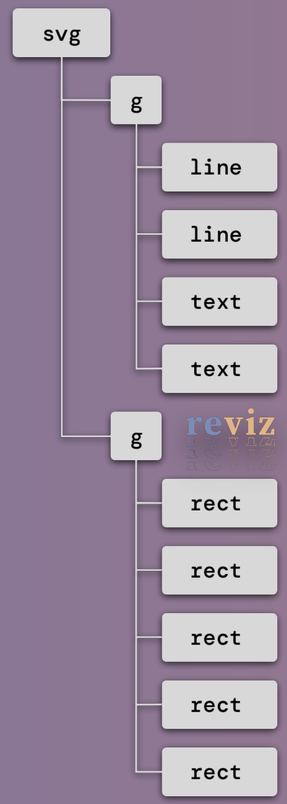


Visit each node in the `svg` subtree

What actually happens in this step?

1.

Visit each node in the `svg` subtree

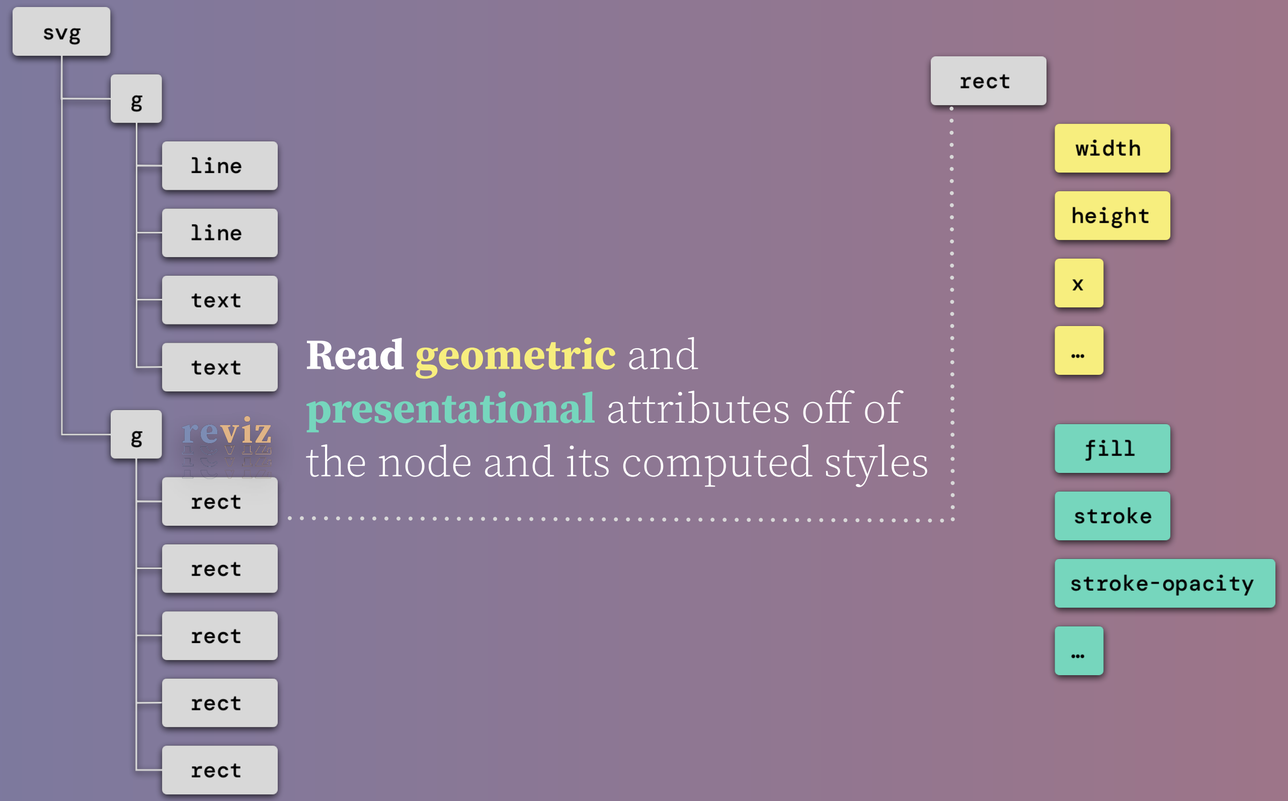


Read geometric and **presentational** attributes off of the node and its computed styles

What actually happens in this step?

1.

Visit each node in the `svg` subtree



What actually happens in this step?

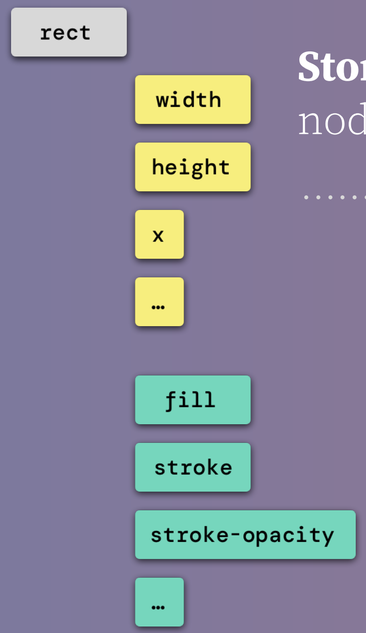
reviz

1.

Visit each node in the `svg` subtree

2.

Read geometric and **presentational** attributes off of the node and its computed styles



Store attributes of all nodes in **attribute sets**

```
{
 width: {'20'},
 height: {'10', '13', '27', '3', …},
 x: {'133', '353', '27', …}
 fill: {'#7b16ff'},
 stroke: {'none'},
 strokeOpacity: {'1'}
}
```

What actually happens in this step?

reviz

1.

Visit each node in the `svg` subtree

2.

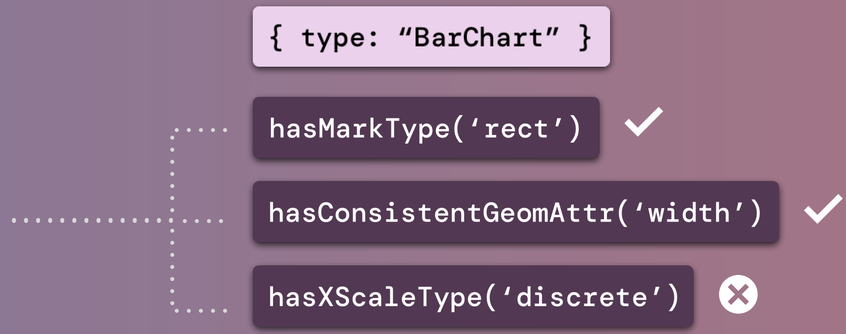
Read geometric and **presentational** attributes off of the node and its computed styles

3.

Store attributes of all nodes in **attribute sets**

Apply a set of **predicate functions** associated with a **visualization type**

```
{
 width: {'20'},
 height: {'10', '13', '27', '3', …},
 x: {'133', '353', '27', …}
 fill: {'#7b16ff'},
 stroke: {'none'},
 strokeOpacity: {'1'}
}
```



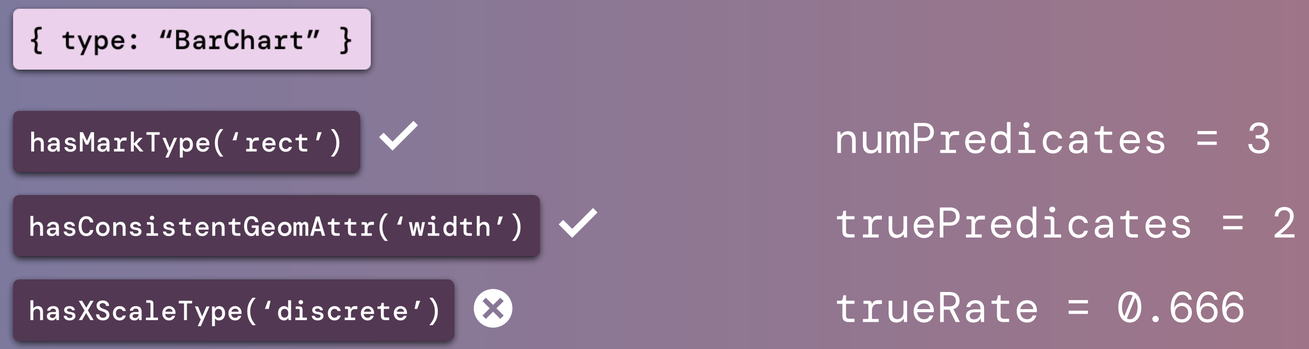
What actually happens in this step?

reviz

4.

Apply a set of **predicate functions** associated with a **visualization type**

Compute the **“true rate”** associated with a **visualization type**



What actually happens in this step?

reviz

4.

Apply a set of **predicate functions** associated with a **visualization type**

5.

Compute the **“true rate”** associated with a **visualization type**

What actually happens in this step?

reviz

4.

Apply a set of **predicate functions** associated with a **visualization type**

5.

Compute the **“true rate”** associated with a **visualization type**

`{ type: “Histogram” }`

`hasMarkType(‘rect’)` ✓

`hasConsistentGeomAttr(‘width’)` ✓

`hasXScaleType(‘continuous’)` ✓

`{ type: “BarChart” }`

`hasMarkType(‘rect’)` ✓

`hasConsistentGeomAttr(‘width’)` ✓

`hasXScaleType(‘discrete’)` ✗

`{ type: “Scatterplot” }`

`hasMarkType(‘circle’)` ✗

`hasConsistentGeomAttr(‘r’)` ✗

Select the **visualization type** with the highest **“true rate”**

What actually happens in this step?

4.

Apply a set of **predicate functions** associated with a **visualization type**

5.

Compute the **“true rate”** associated with a **visualization type**

6.

Select the **visualization type** with the highest **“true rate”**

Visualization type

```
{ type: “Histogram” }
```

Attribute sets

```
{
 width: {‘20’},
 height: {‘10’, ’13’, ’27’, ‘3’, …},
 x: {‘133’, ‘353’, ’27’, …}
 fill: {‘#7b16ff’},
 stroke: {‘none’},
 strokeOpacity: {‘1’}
}
```

Combine the **visualization type** and **attribute sets** into an IR

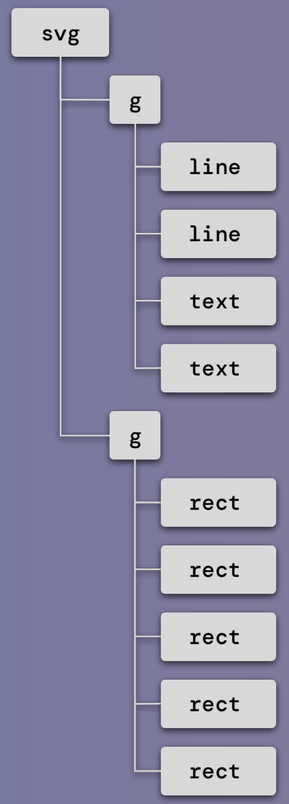
Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

Where are we now?

reviz

svg **subtree**



The ***frontend*** of the reviz compiler

Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

What actually happens in this step?

reviz

Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

The ***backend*** of the reviz compiler

Partial Observable Plot Program

Program

```
const plot = Plot.plot({
  marks: [
    Plot.barY(
      data,
      Plot.binX(
        { y: "count" },
        {
          x: "??",
          fill: "steelblue",
          fillOpacity: 1,
          stroke: "none",
          strokeOpacity: 1,
          strokeWidth: 1,
        }
      )
    ),
  ],
});

```

Where are we now?

reviz

Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

Apply **contextual semantics** to rewrite terms in the IR to the output program

```
`Plot.barY(data,
  Plot.binX(
    {
      y: ‘count’,
    },
    {
      x: ‘${PROGRAM_HOLE}’,
      ${EVAL_HOLE}
    }
  )
)`
```

```
`${camelCase(attrName)}: ${val}${!isLastAttr ? `, ${EVAL_HOLE}` : ''}`
```

Where are we now?

reviz

Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

Program

```
`${EVAL_HOLE}`
```

Start with an empty program, represented by a **“hole”**.

Where are we now?

reviz

Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

Apply the rewrite for the first key in the IR.

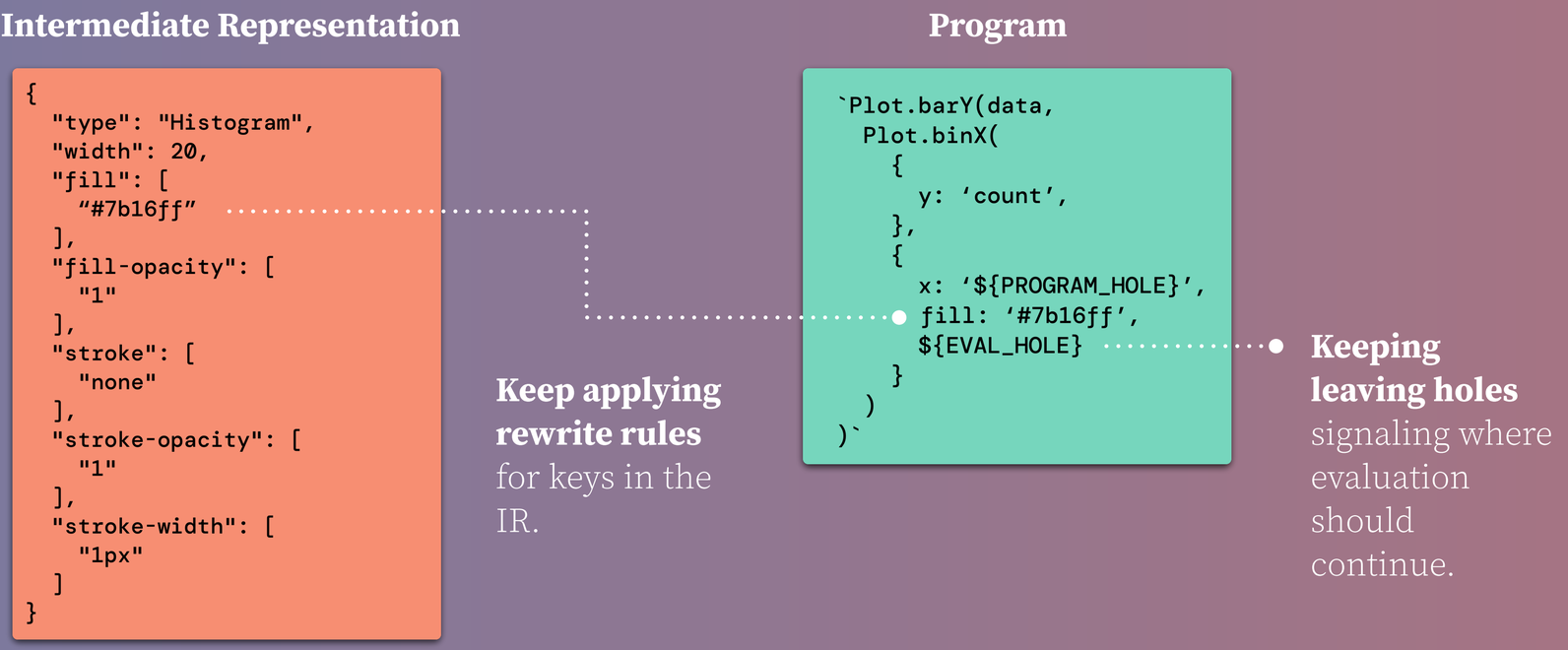
Program

```
`Plot.barY(data,
  Plot.binX(
    {
      y: ‘count’,
    },
    {
      x: ‘${PROGRAM_HOLE}’,
      ${EVAL_HOLE}
    }
  )
)`
```

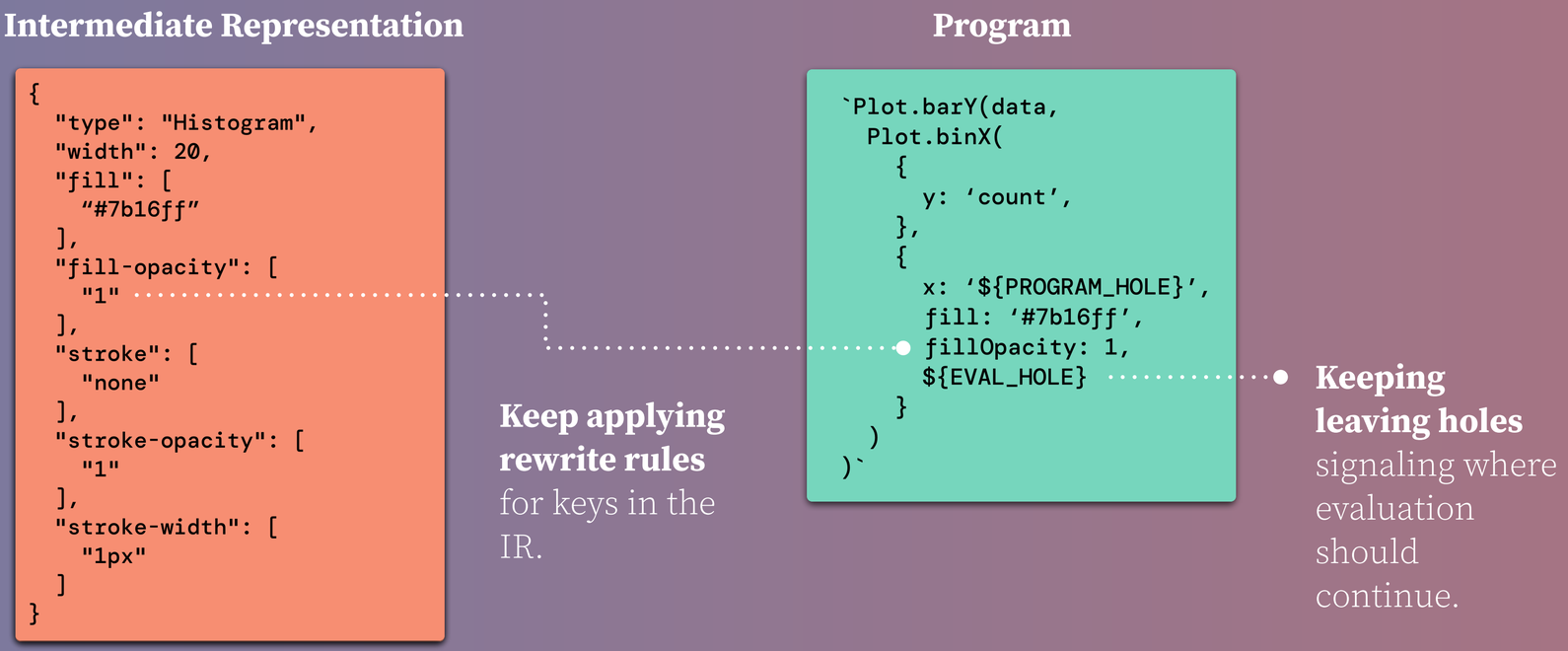
Leave a hole signaling where evaluation should continue.

Where are we now?

reviz



Where are we now?



Where are we now?

reviz

Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

Return the program once we’ve iterated through all keys in the IR.

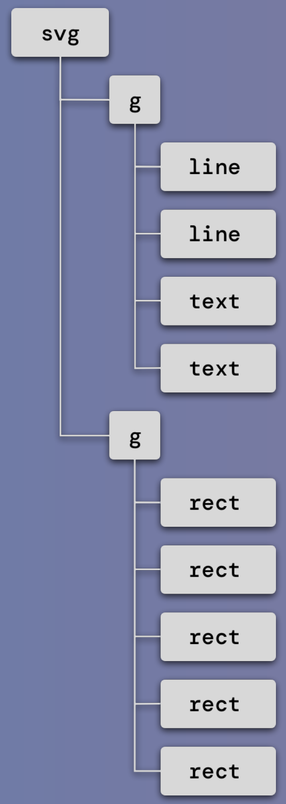
Program

```
`Plot.barY(data,
  Plot.binX(
    {
      y: ‘count’,
    },
    {
      x: ‘${PROGRAM_HOLE}’,
      fill: ‘#7b16ff’,
      fillOpacity: 1,
      stroke: ‘none’,
      strokeOpacity: 1,
      strokeWidth: 1
    }
  )
)`
```

Where are we now?

reviz

svg subtree



The ***frontend*** of the reviz compiler

Intermediate Representation

```
{
  "type": "Histogram",
  "width": 20,
  "fill": [
    “#7b16ff”
  ],
  "fill-opacity": [
    "1"
  ],
  "stroke": [
    "none"
  ],
  "stroke-opacity": [
    "1"
  ],
  "stroke-width": [
    "1px"
  ]
}
```

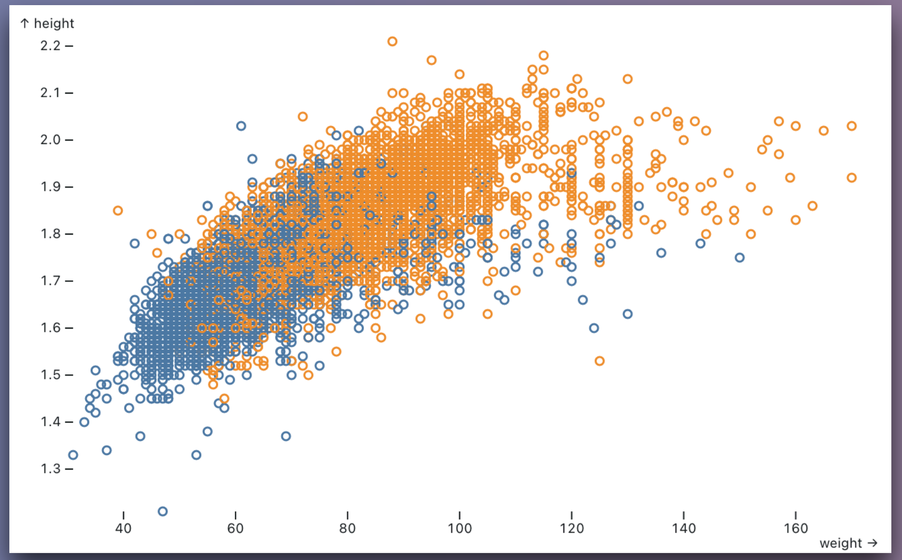
The ***backend*** of the reviz compiler

Partial Observable Plot Program

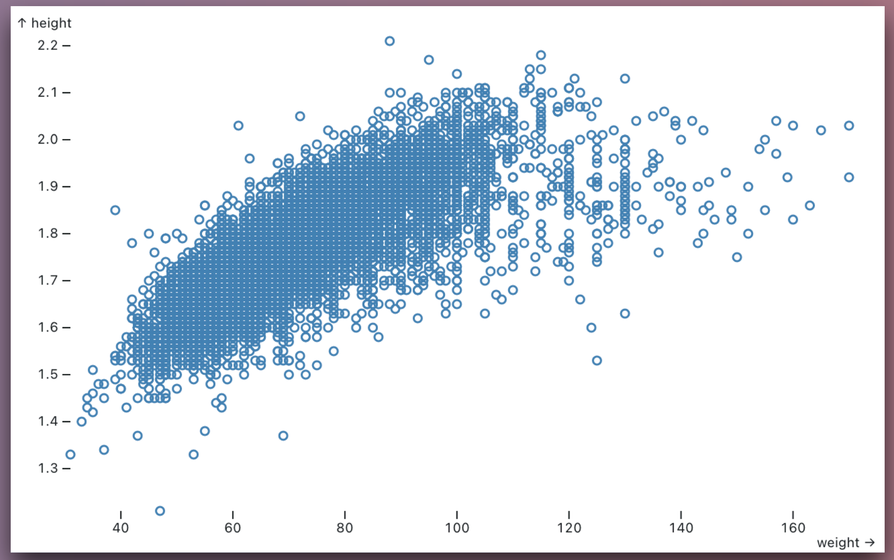
```
`Plot.barY(data,
  Plot.binX(
    {
      y: ‘count’,
    },
    {
      x: ‘${PROGRAM_HOLE}’,
      fill: ‘#7b16ff’,
      fillOpacity: 1,
      stroke: ‘none’,
      strokeOpacity: 1,
      strokeWidth: 1
    }
  )
)`
```

How do we decide where to leave holes?

`stroke: “??”`



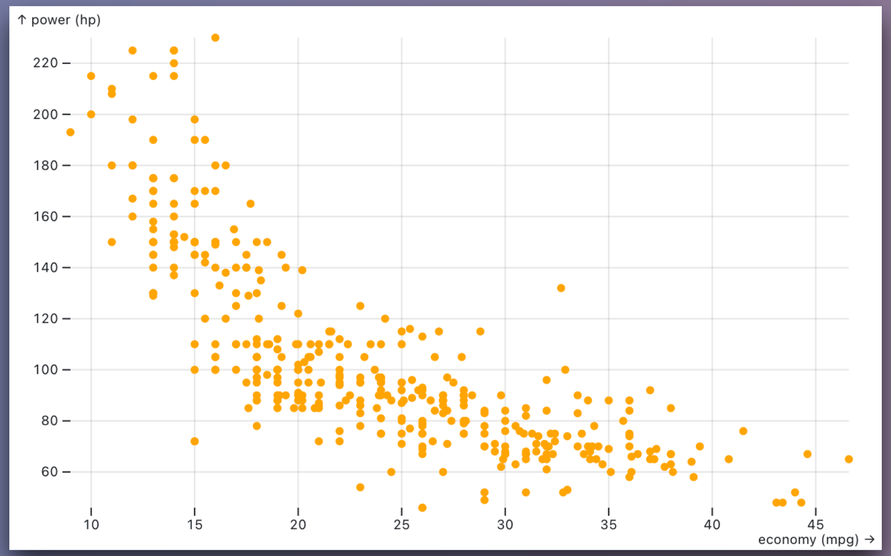
`stroke: “steelblue”`



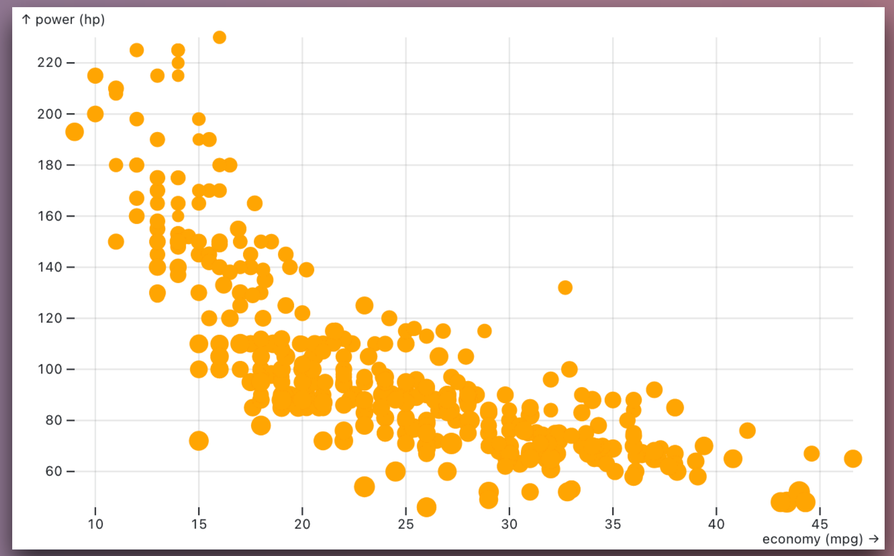
If there is variance of any kind in a visual variable, ***assume*** **it is because that variable is mapped to a column in the data**.

How do we decide where to leave holes?

`r: 3`



`range: [0, 8], r: “??”`



If there is variance of any kind in a visual variable, ***assume*** **it is because that variable is mapped to a column in the data**.

How do we decide where to leave holes?

The harder case (in my opinion) is actually about

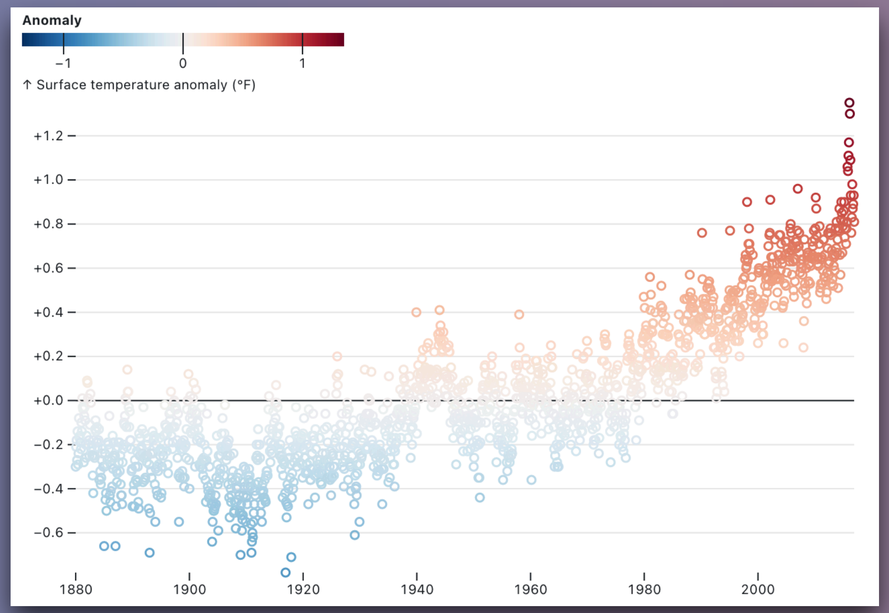
making semantic judgements

from

limited information!

How do we decide where to leave holes?

The harder case (in my opinion) is actually about **making semantic judgements** from **limited information**!




```
stroke: [
  'rgb(249, 200, 175)',
  'rgb(250, 213, 193)',
  …,
  'rgb(249, 232, 222)'
]
```

```
color: {
  type: "diverging",
  scheme: "BuRd",
  legend: true
},
stroke: "??"
```

How do we **infer** that this is a **diverging** color scheme?

Roadmap

1.

The **promise** of (and **problem** with) examples in data visualization

2.

What is **reviz**? How does it address the **problem**?

3.

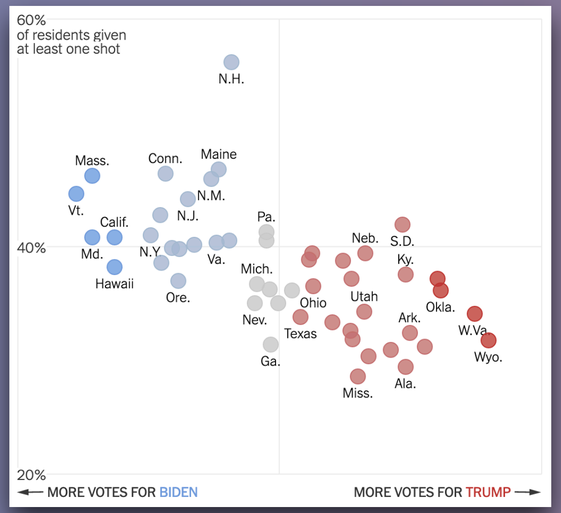
Questions and **curiosities** for future work

Roadmap

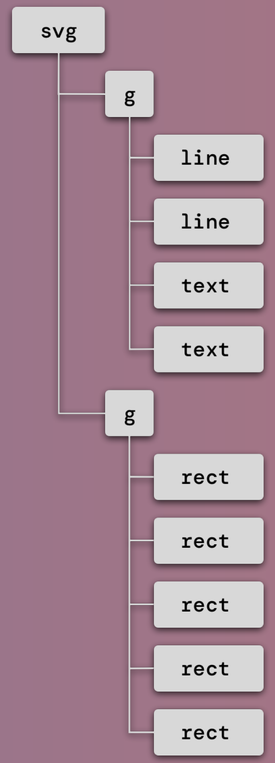
3.

Questions and
curiosities for
future work

What is the right interaction model?



How do we make **extracting the `svg` subtree** as simple as possible?

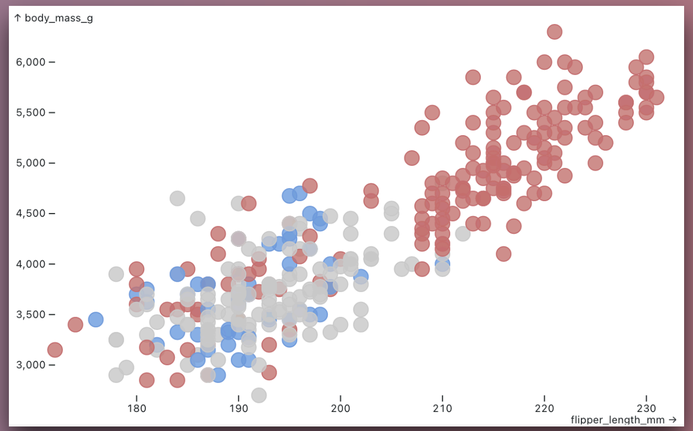


What is the right interaction model?

Program

```
const plot = Plot.plot({
  color: {
    type: "categorical",
    range: ["#C67371", "#ccc", "#709DDE", "#A7B9D3", "#C23734"],
  },
  marks: [
    Plot.dot(data, {
      x: "??",
      y: "??",
      r: 7,
      fill: "??",
      fillOpacity: 0.8,
      stroke: "??",
      strokeOpacity: 1,
      strokeWidth: 1,
    }),
  ],
});
```

How do we make **filling in holes** as simple as possible?



How can we scale support for more visualization types?

For each **new visualization type**, we need to define **new predicates**.

```
{ type: “BarChart” }
```

```
hasMarkType(‘rect’)
```

```
hasConsistentGeomAttr(‘width’)
```

```
hasXScaleType(‘discrete’)
```

How can we scale support for more visualization types?

For each **new visualization type**, we need to define **new predicates**.

```
{ type: “BarChart” }
```

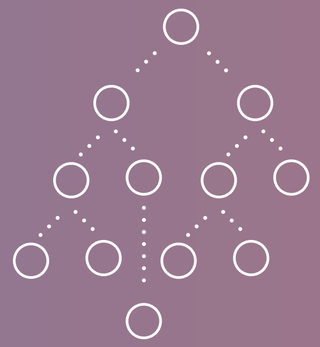
```
hasMarkType(‘rect’)
```

```
hasConsistentGeomAttr(‘width’)
```

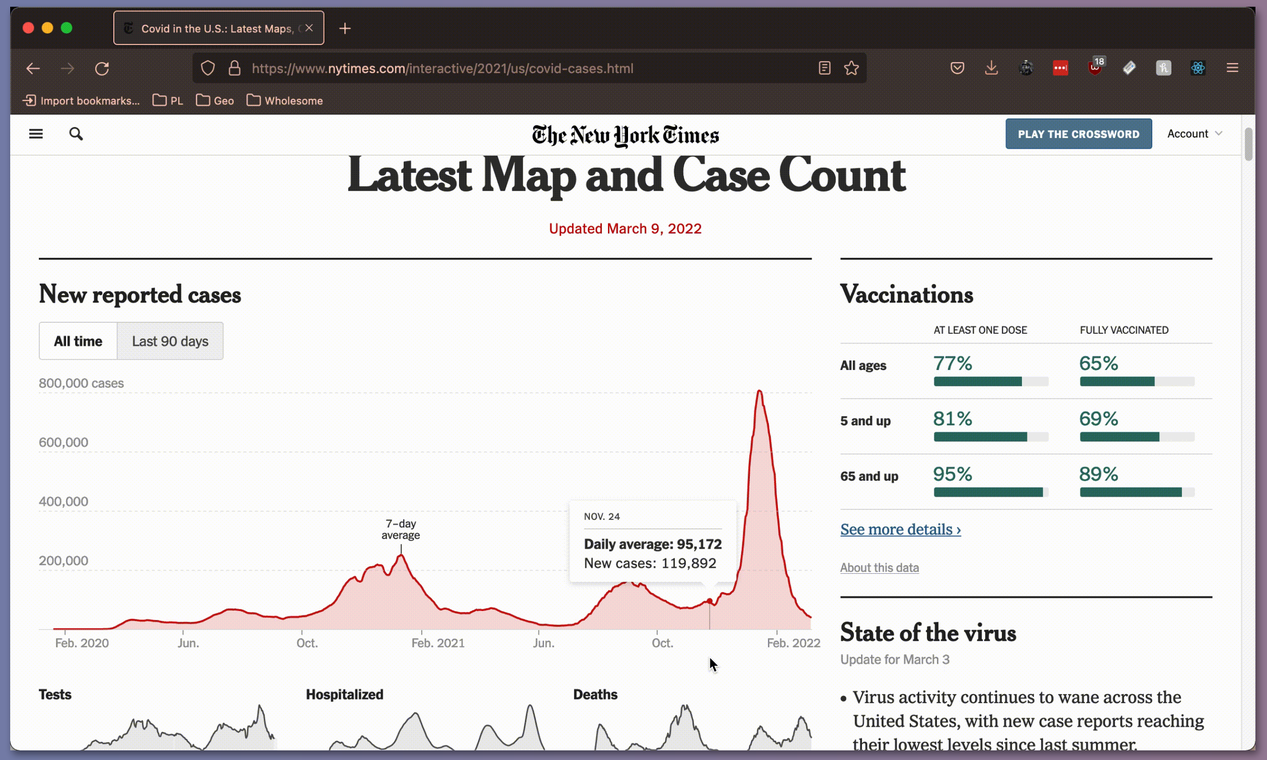
```
hasXScaleType(‘discrete’)
```

Could we instead **learn these predicates** by training a model on `svg` subtrees?

```
{ type: “BarChart” }
```



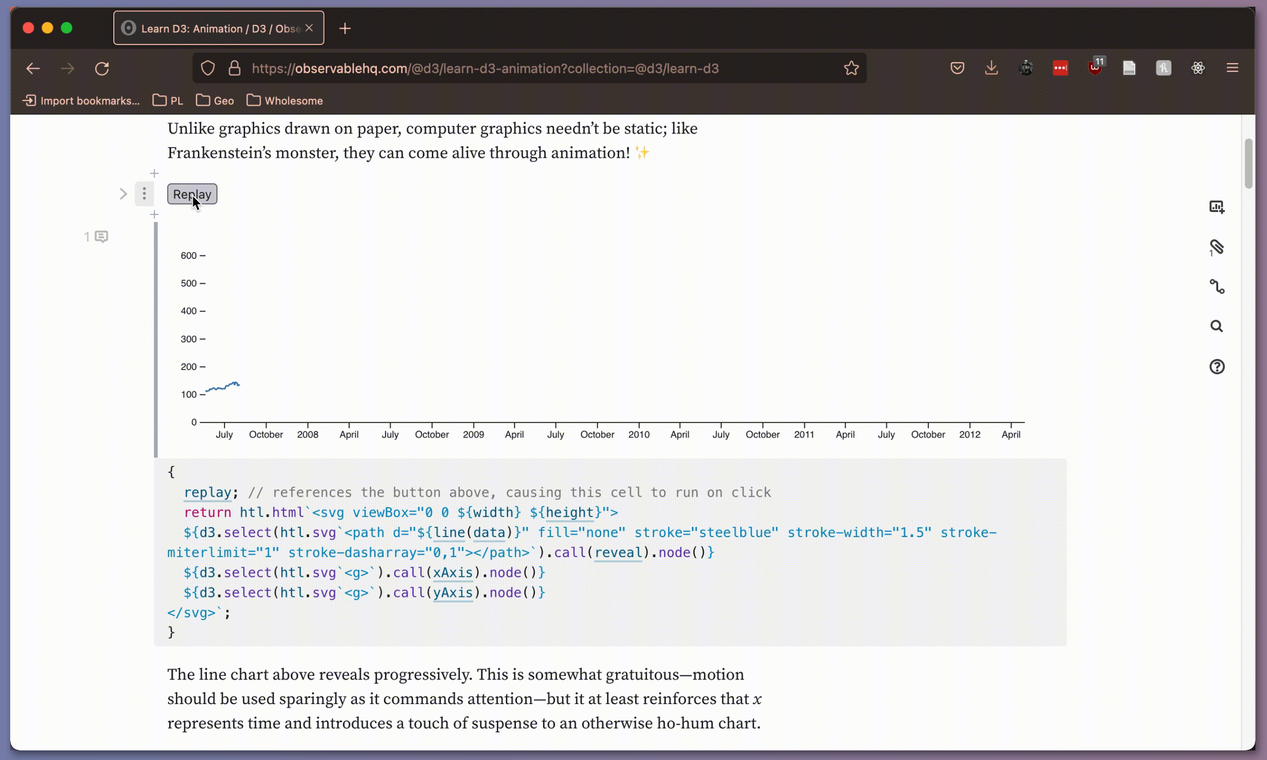
Can we reverse engineer interactivity?



Can we **detect interactive behavior** like tooltip rendering on hover?

If so, how might we **encode** this in our IR?

Can we reverse engineer animation?



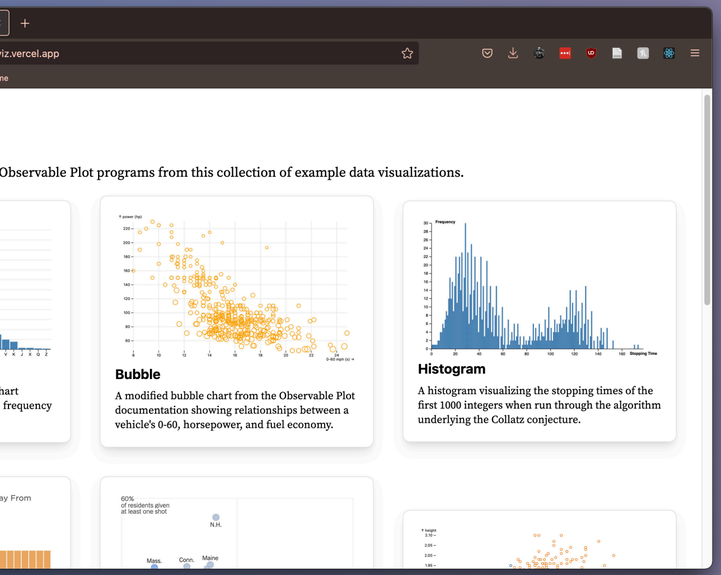
Can we **detect animation** like draw-on-render effects?

If so, how might we **encode** this in our IR?

Thank You!

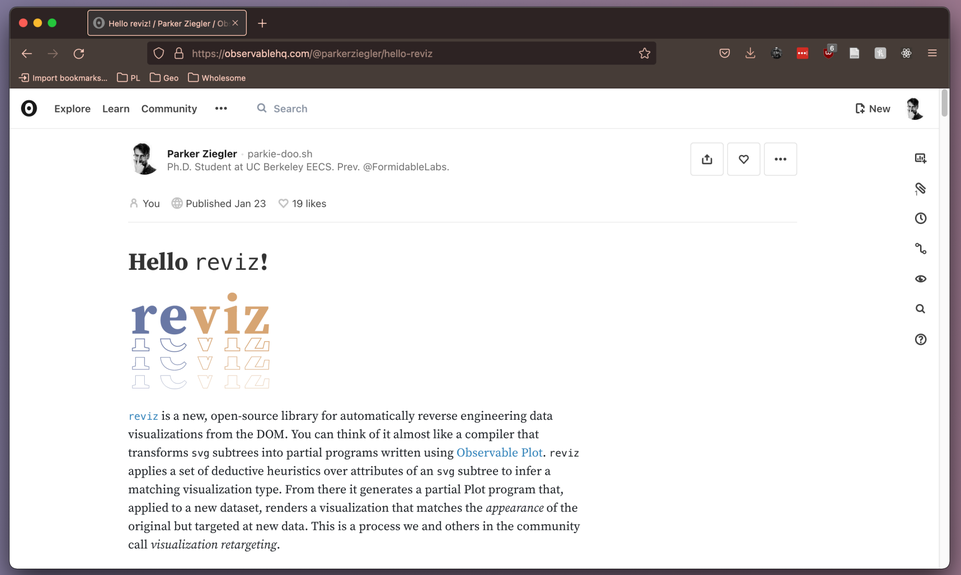
reviz

Examples



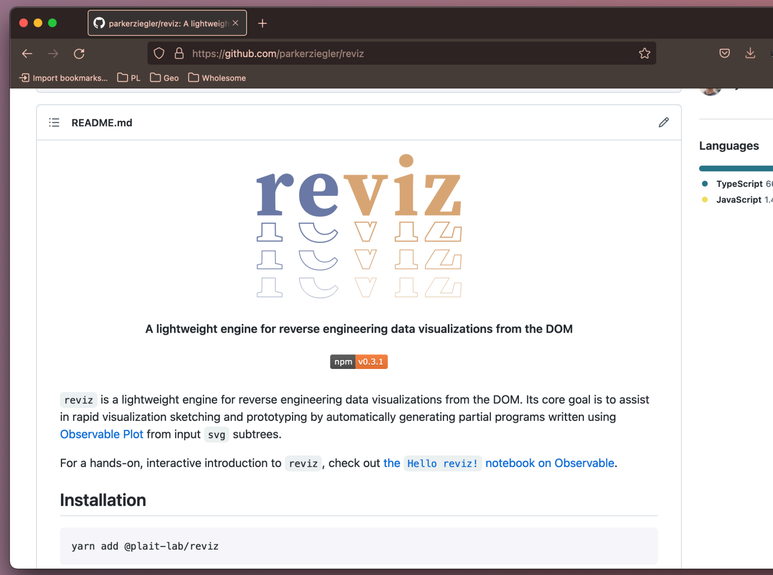
reviz.vercel.app

Notebook



observablehq.com/@parkerziegler/hello-reviz

Source



github.com/parkerziegler/reviz