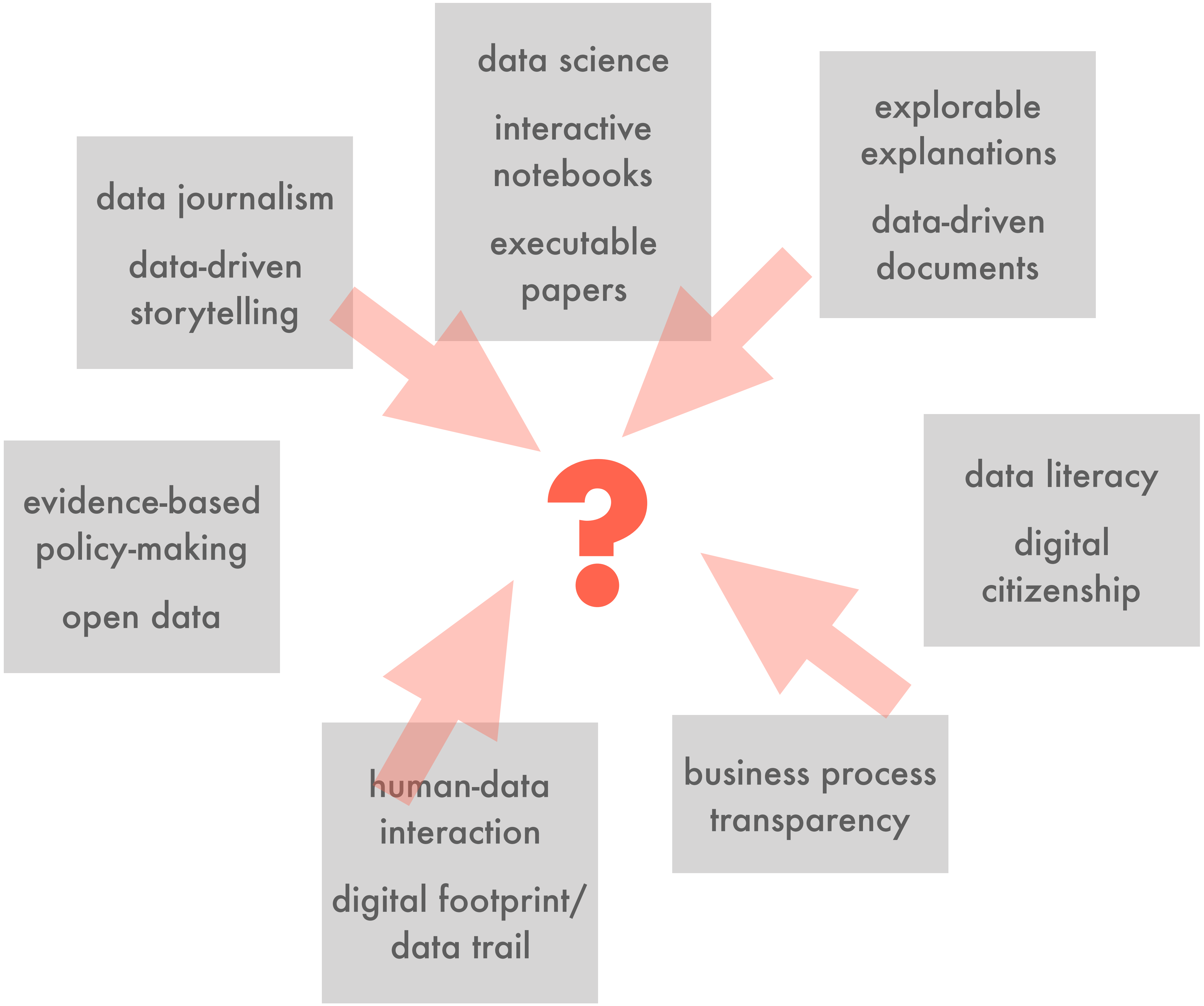
Self-Explaining Computation with Explicit Change

Roly Perera
University of Glasgow

trends towards a data-centric society



overview

computation growing into a new role

a literate medium for expressing arguments, narratives, workflows and ideas

overview

some desiderata of these new apps:

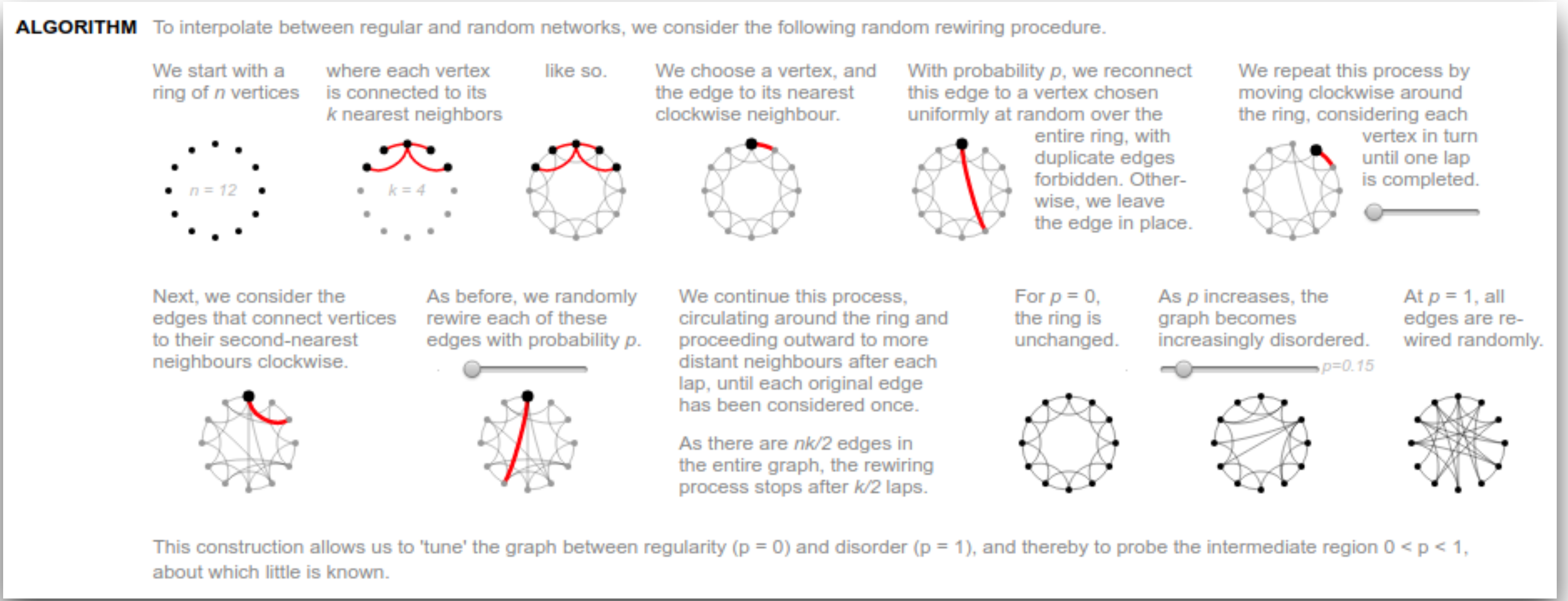
- human-readable
- transparent, reproducible
- access to *process*, not just outcome
- multiple stakeholders, different concerns
- end-user empowerment

(some tensions in there..)

plan for what follows

1. examples of explorable explanations, interactive notebooks, data journalism
2. some present limitations
3. proposal: *self-explaining computation with explicit change*
4. existing work we will build on

explorable explanations



ALGORITHM To interpolate between regular and random networks, we consider the following random rewiring procedure.

We start with a ring of *n* vertices

where each vertex is connected to its *k* nearest neighbors

like so.

We choose a vertex, and the edge to its nearest clockwise neighbour.

With probability *p*, we reconnect this edge to a vertex chosen uniformly at random over the entire ring, with duplicate edges forbidden. Otherwise, we leave the edge in place.

We repeat this process by moving clockwise around the ring, considering each vertex in turn until one lap is completed.

Next, we consider the edges that connect vertices to their second-nearest neighbours clockwise.

As before, we randomly rewire each of these edges with probability *p*.

We continue this process, circulating around the ring and proceeding outward to more distant neighbours after each lap, until each original edge has been considered once.

As there are *nk*/2 edges in the entire graph, the rewiring process stops after *k*/2 laps.

For *p* = 0, the ring is unchanged.

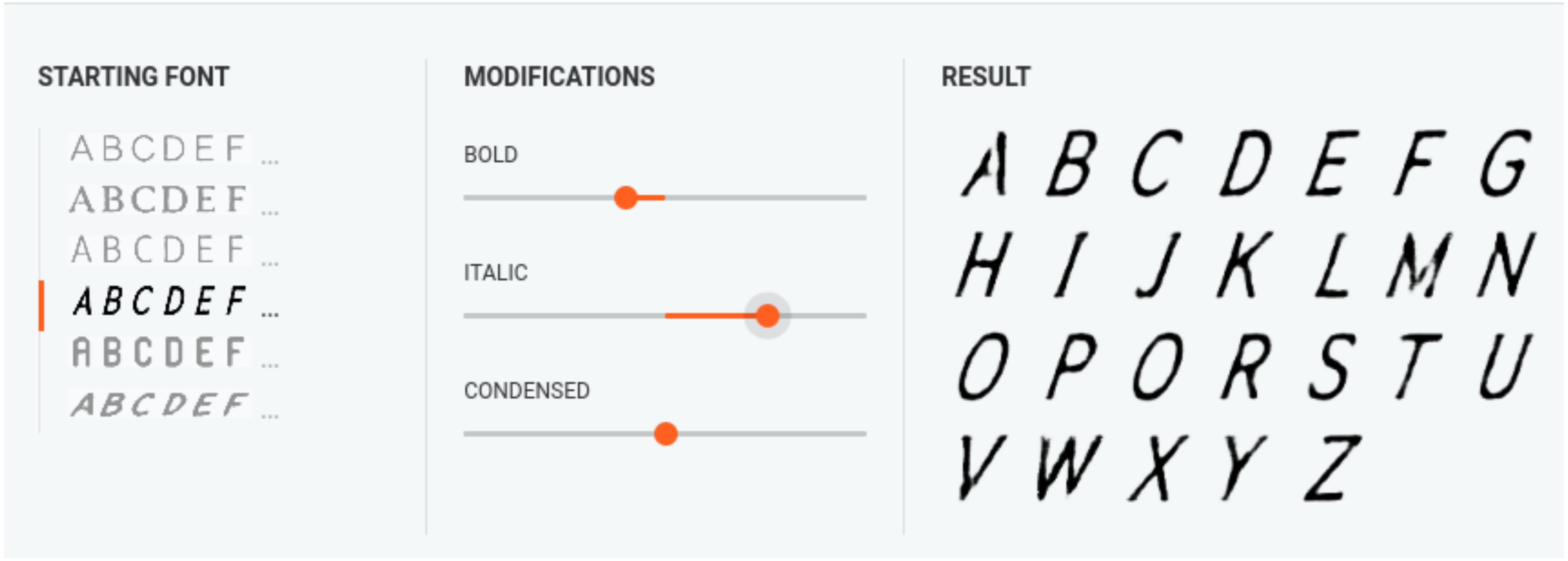
As *p* increases, the graph becomes increasingly disordered.

At *p* = 1, all edges are rewired randomly.

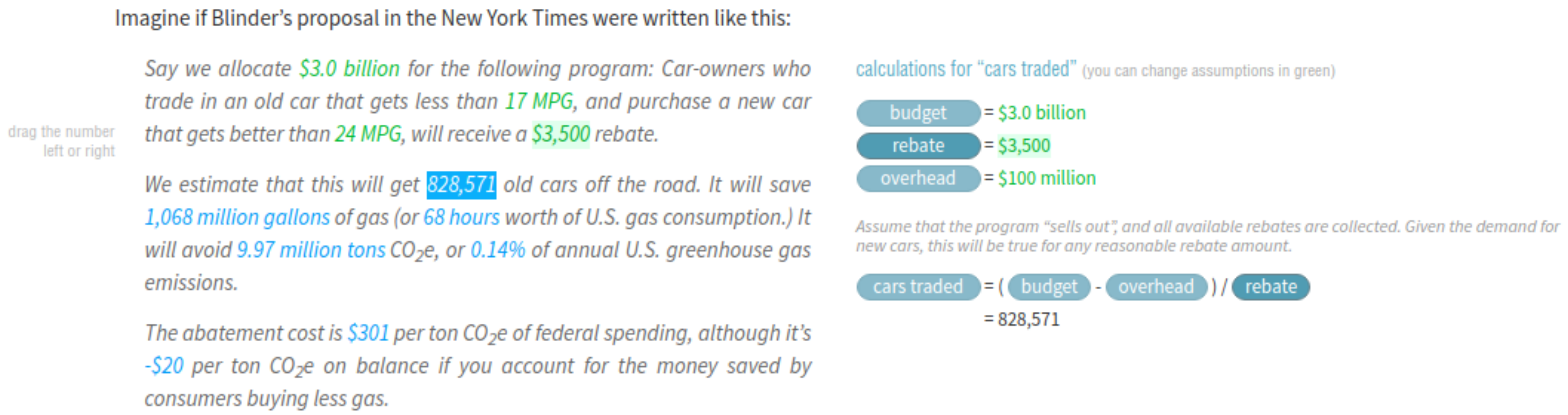
This construction allows us to 'tune' the graph between regularity (p = 0) and disorder (p = 1), and thereby to probe the intermediate region 0 < p < 1, about which little is known.

explorable explanations

Let's look at an example where a machine learning model makes a new type of interface possible. To understand the interface, imagine you're a type designer, working on creating a new font [1] . After sketching some initial designs, you wish to experiment with bold, italic, and condensed variations. Let's examine a tool to generate and explore such variations, from any initial design. For reasons that will soon be explained the quality of results is quite crude; please bear with us.



explorable explanations



Imagine if Blinder's proposal in the New York Times were written like this:

drag the number left or right

Say we allocate $3.0 billion for the following program: Car-owners who trade in an old car that gets less than 17 MPG, and purchase a new car that gets better than 24 MPG, will receive a $3,500 rebate.

We estimate that this will get 828,571 old cars off the road. It will save 1,068 million gallons of gas (or 68 hours worth of U.S. gas consumption.) It will avoid 9.97 million tons CO_2e, or 0.14% of annual U.S. greenhouse gas emissions.

The abatement cost is $301 per ton CO_2e of federal spending, although it's -$20 per ton CO_2e on balance if you account for the money saved by consumers buying less gas.

calculations for "cars traded" (you can change assumptions in green)

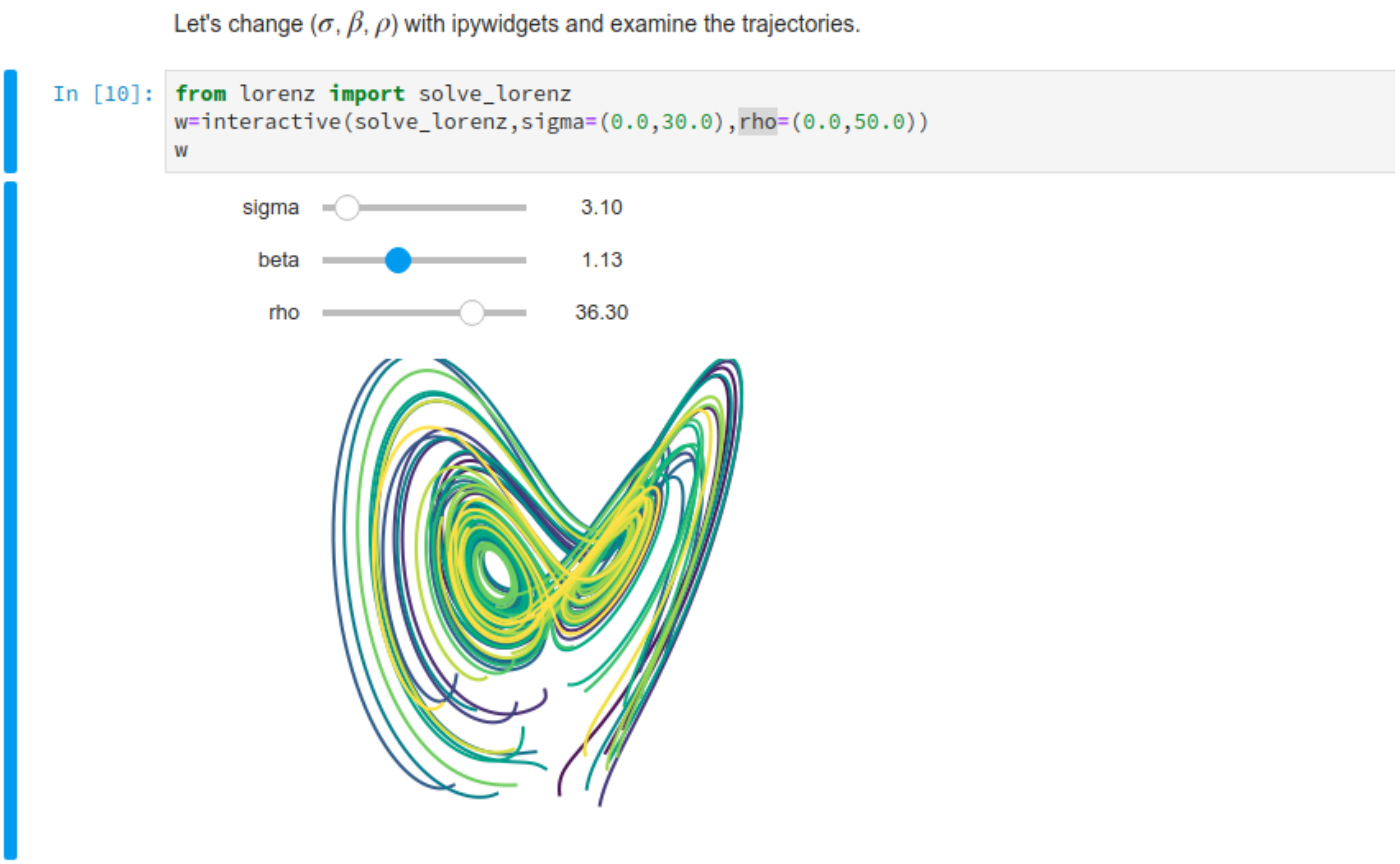
budget	= $3.0 billion
rebate	= $3,500
overhead	= $100 million

Assume that the program "sells out", and all available rebates are collected. Given the demand for new cars, this will be true for any reasonable rebate amount.

cars traded = (budget - overhead) / rebate

= 828,571

interactive notebooks

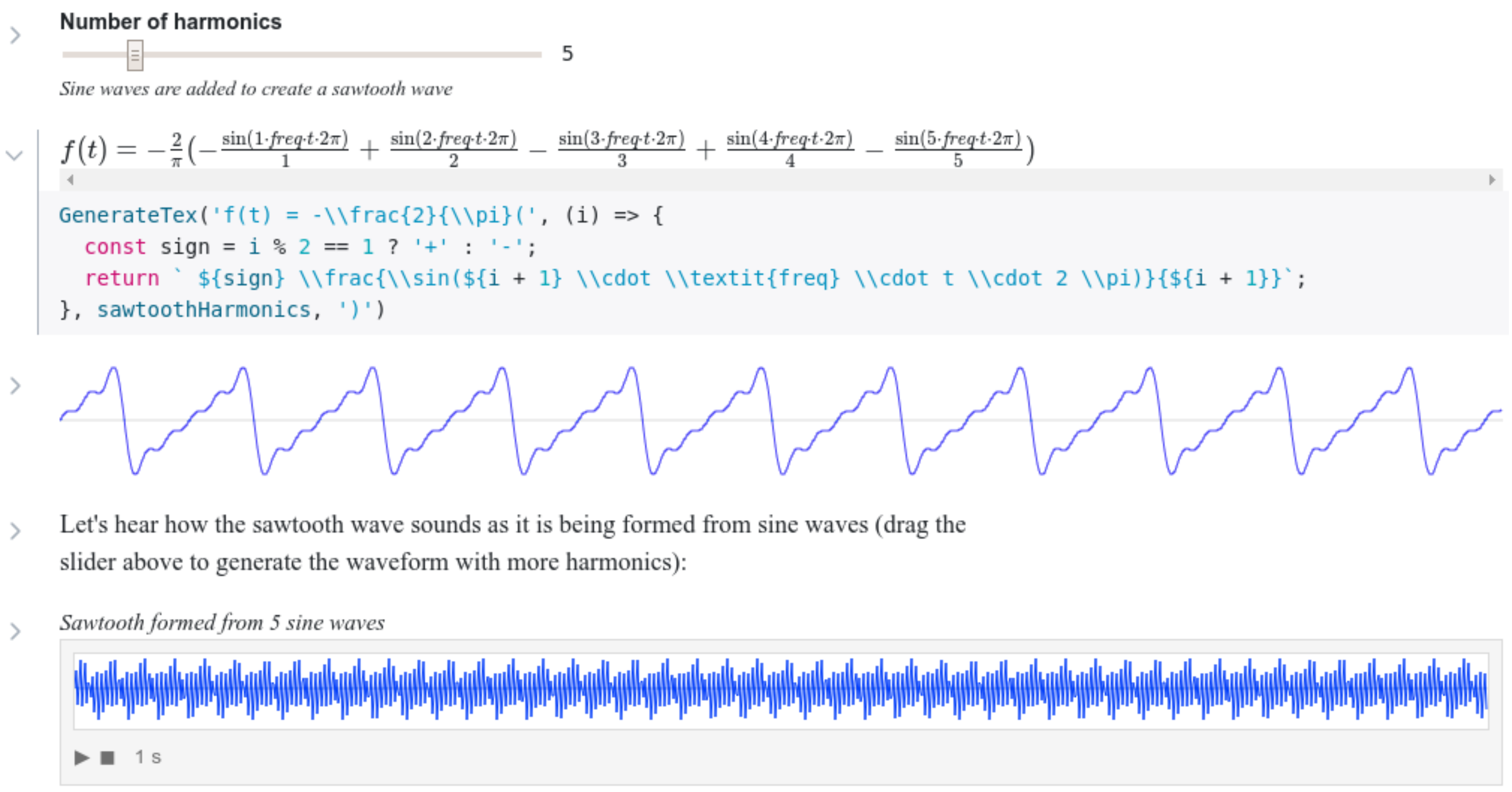


Let's change (σ, β, ρ) with ipywidgets and examine the trajectories.

```
In [10]: from lorenz import solve_lorenz
         w=interactive(solve_lorenz,sigma=(0.0,30.0),rho=(0.0,50.0))
         w
```

```
sigma   3.10
beta    1.13
rho     36.30
```

interactive notebooks



Number of harmonics

5

Sine waves are added to create a sawtooth wave

$$f(t) = -\frac{2}{\pi}(-\frac{\sin(1 \cdot \textit{freq} \cdot t \cdot 2\pi)}{1} + \frac{\sin(2 \cdot \textit{freq} \cdot t \cdot 2\pi)}{2} - \frac{\sin(3 \cdot \textit{freq} \cdot t \cdot 2\pi)}{3} + \frac{\sin(4 \cdot \textit{freq} \cdot t \cdot 2\pi)}{4} - \frac{\sin(5 \cdot \textit{freq} \cdot t \cdot 2\pi)}{5})$$

```
GenerateTex('f(t) = -\\frac{2}{\\pi}(', (i) => {
  const sign = i % 2 == 1 ? '+' : '-';
  return ` ${sign} \\frac{\\sin(${i + 1} \\cdot \\textit{freq} \\cdot t \\cdot 2 \\pi)}{${i + 1}}`;
}, sawtoothHarmonics, ')')
```

Let's hear how the sawtooth wave sounds as it is being formed from sine waves (drag the slider above to generate the waveform with more harmonics):

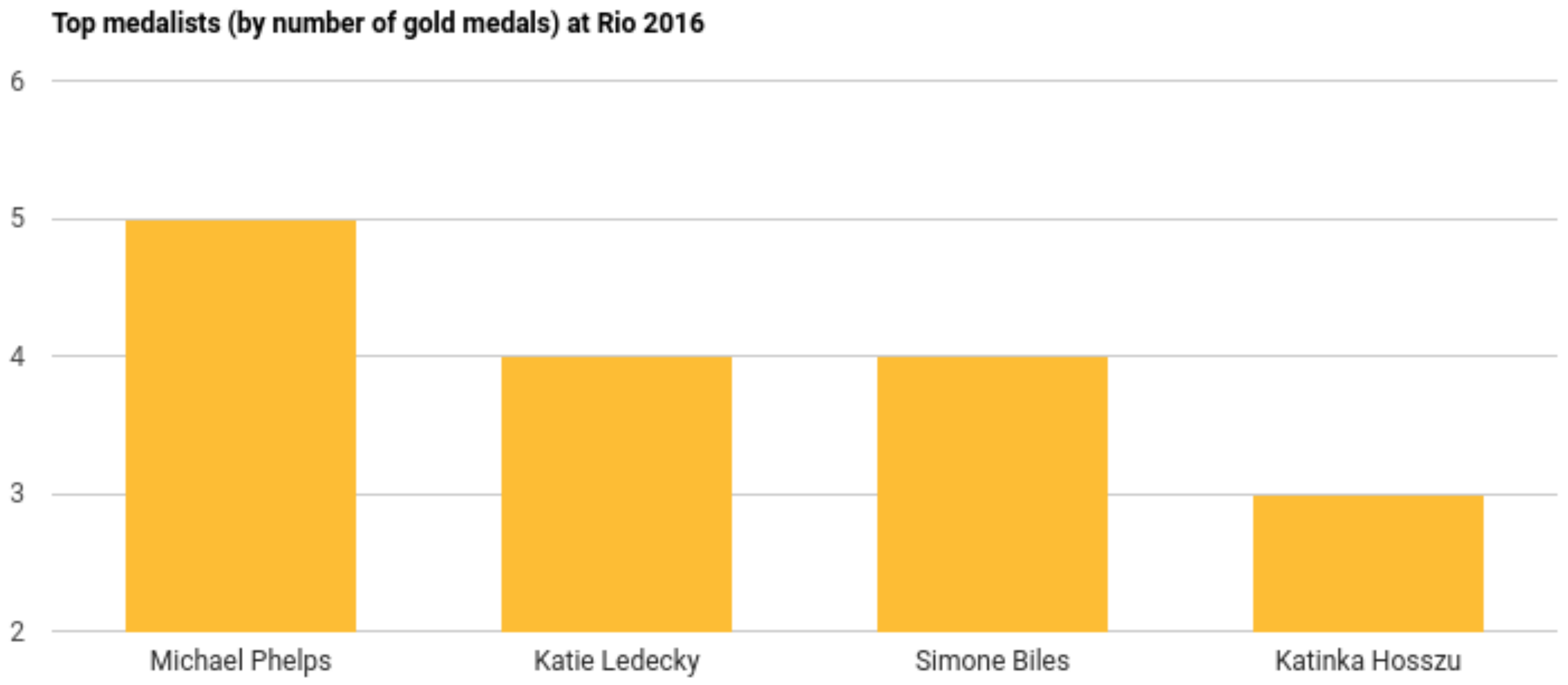
Sawtooth formed from 5 sine waves

data-driven storytelling

```
let data =
  olympics
    .'filter data'.'Games is'.'Rio (2016)'.then.'group data'.'by Athlete'.'sum Gold'.then
    .'sort data'.'by Gold descending'.then.paging.take(4)
    .'get series'.'with key Athlete'.'and value Gold'

chart.column(data).legend(position="none")
  .set(fontName="Roboto", fontSize=12, colors=["#F4C300"], title="Top medalists (by number of gold medals) at Rio 2016")
```

preview



some limitations

An exciting new direction in content creation, dissemination and pedagogy

But still quite *ad hoc*:

- transparency only partial or pre-set
- coarse-grained data/view relationship

some limitations

Distill-style essays and explorable explanations typically <u>fix in advance</u> what can be explored

- not fully transparent/open

some limitations

Notebooks are more open: code is inline in the document

- not easy to see all intermediate results
- not clear how code relates to data/views

proposed research

Self-Explaining Computation with Explicit Change

build on techniques from <u>self-explaining</u> and <u>incremental</u> computation to make explicit:

– how *parts* relate to *other parts*

– how *changes* cause *other changes*

proposed research

Three main use-cases:

#1

how?

easily view or hide any subcomputation or intermediate result

#2

whence?

understand how views relate to data in a fine-grained way

#3

what if?

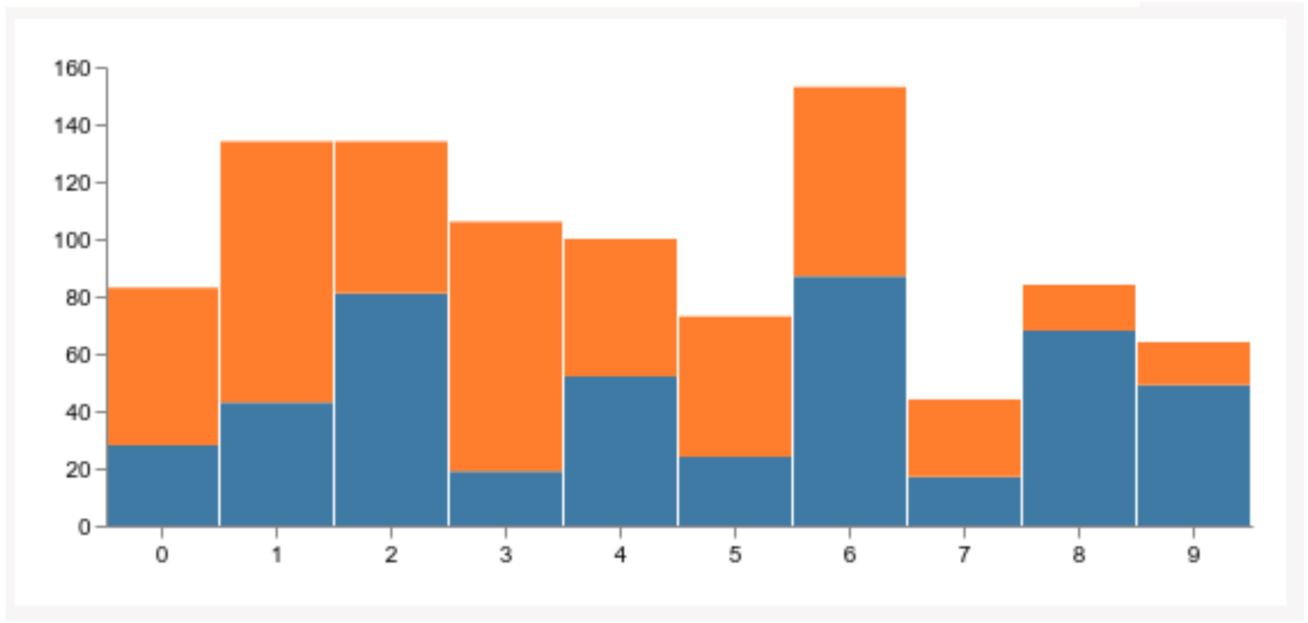
change data or code and see fine-grained consequences

use cases

#2

whence?

```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y": 27, "c":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```

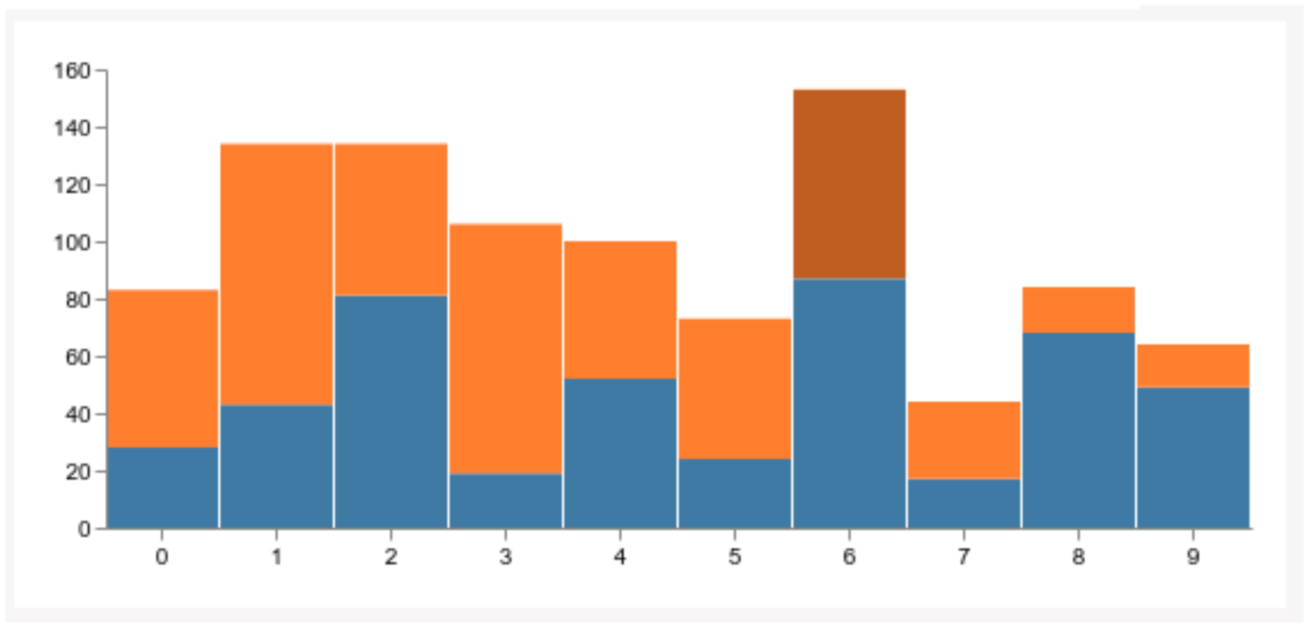


use cases

#2

whence?

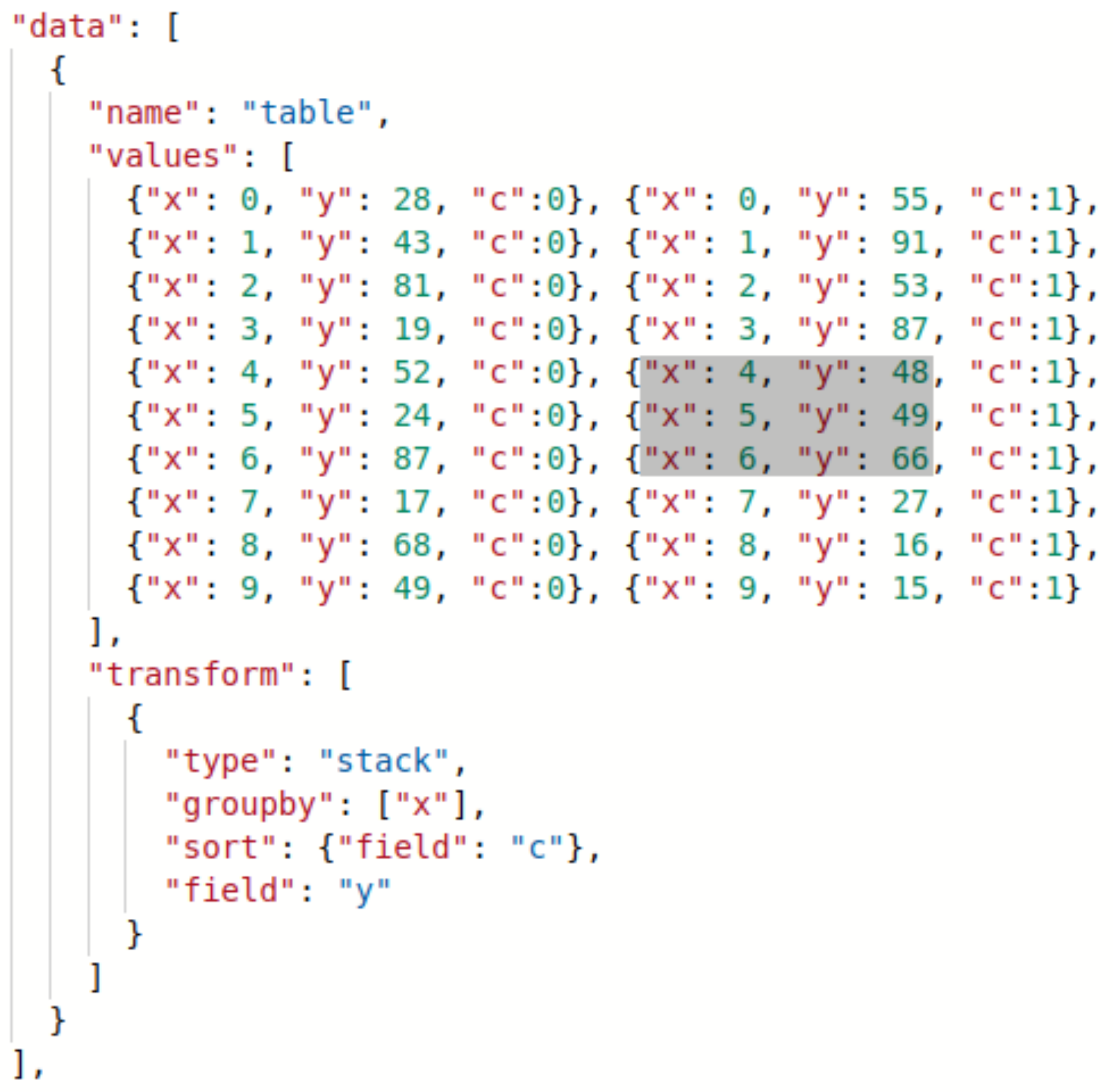
```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y": 27, "c":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```



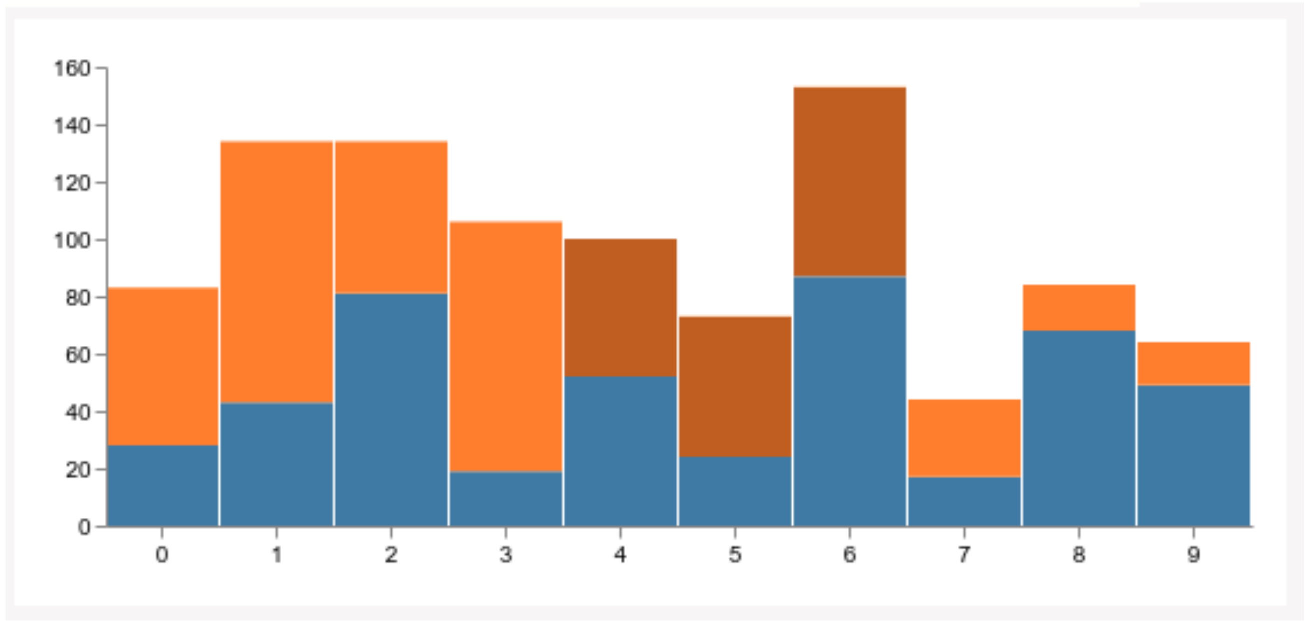
use cases

#2

whence?



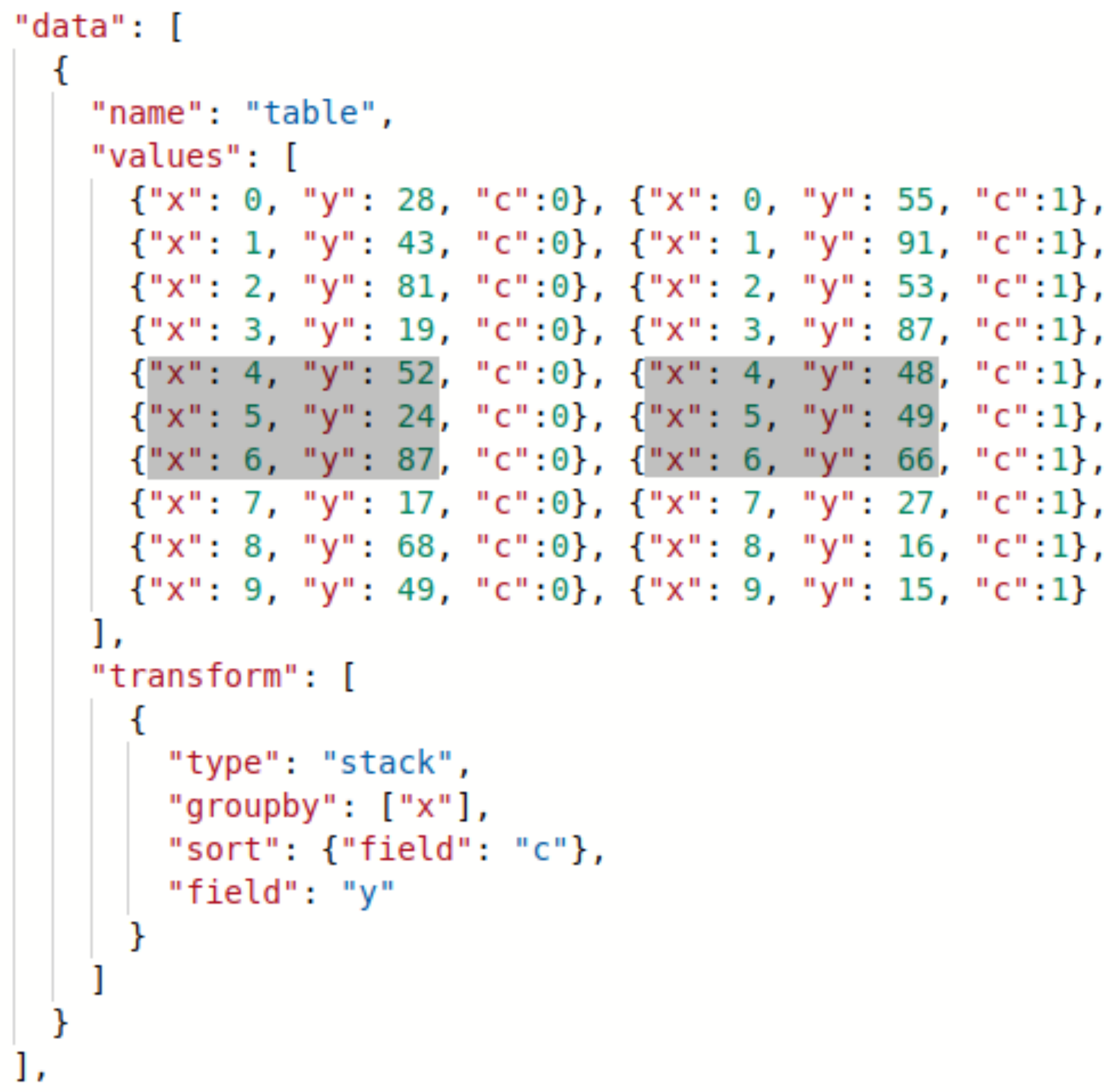
```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y": 27, "c":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```



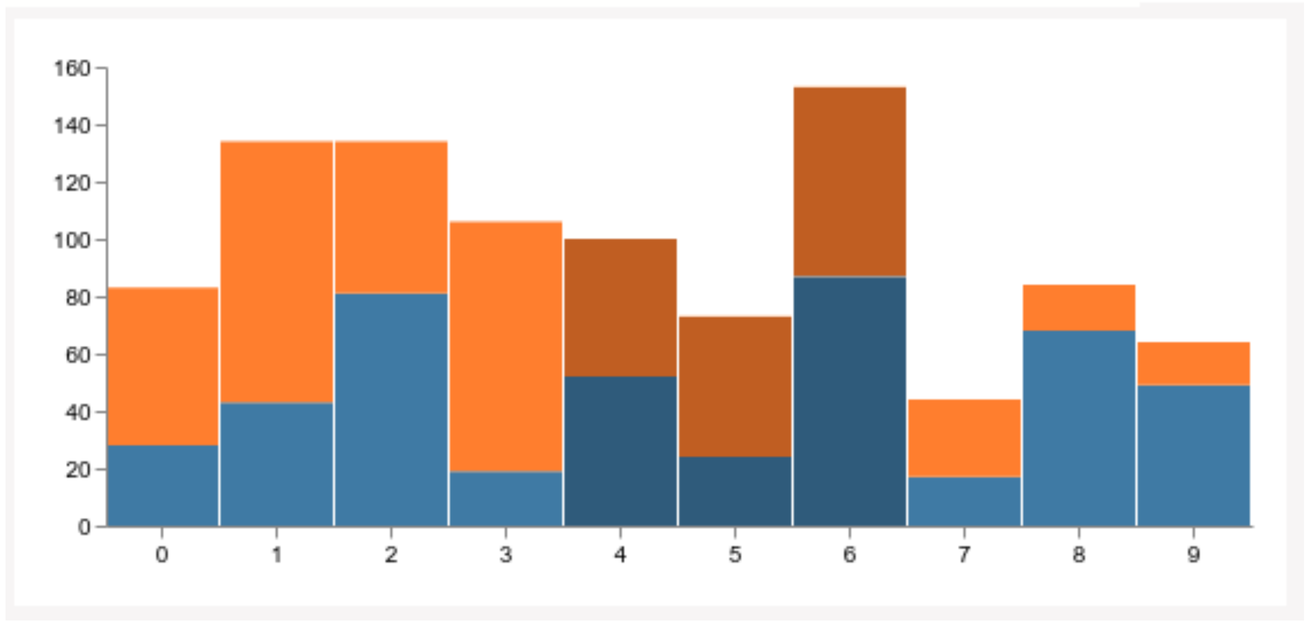
use cases

#2

whence?



```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y": 27, "c":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```

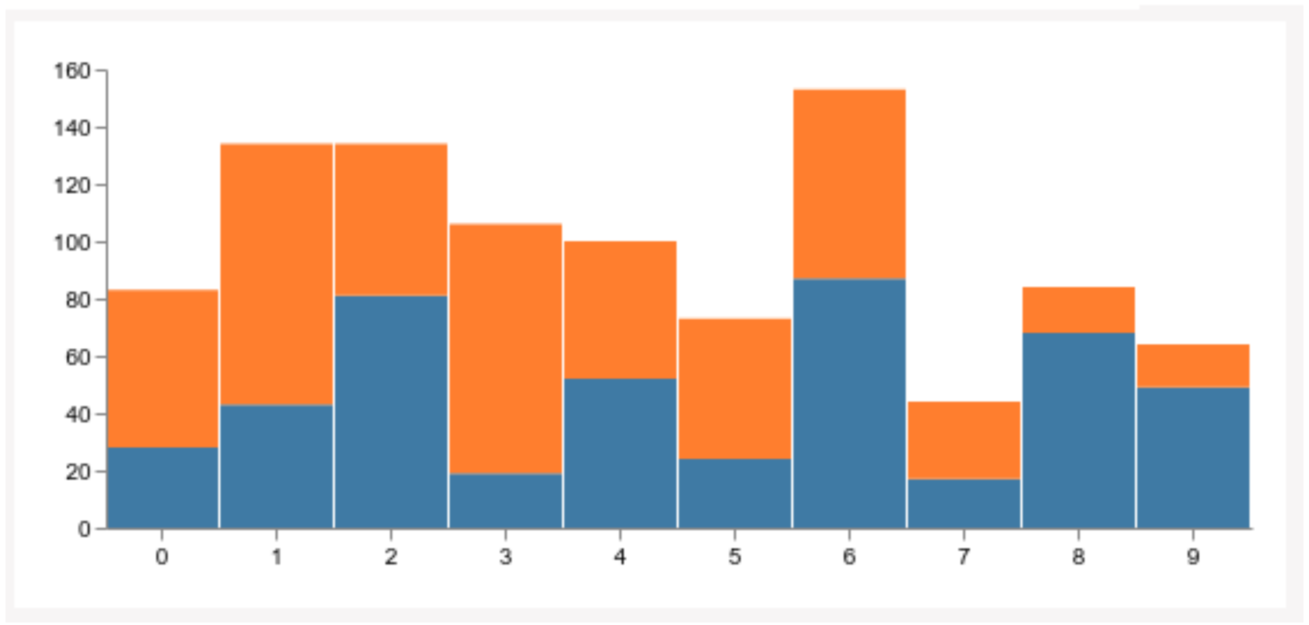


use cases

#2

whence?

```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y": 27, "c":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```

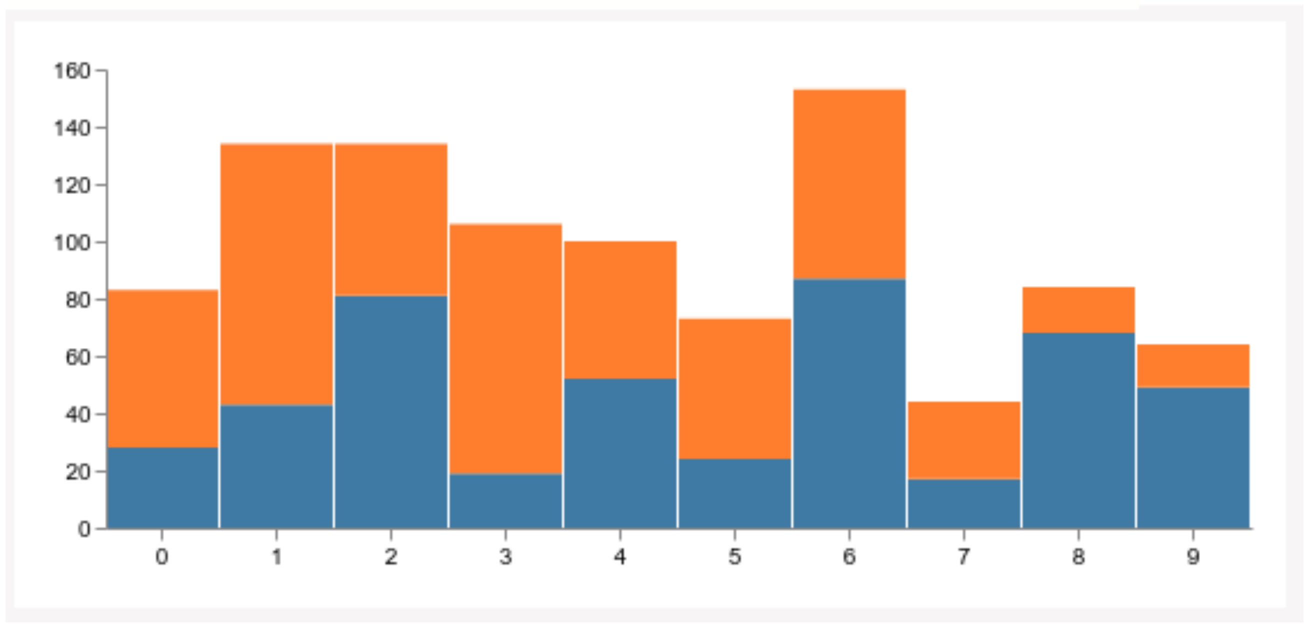


use cases

#3

what if?

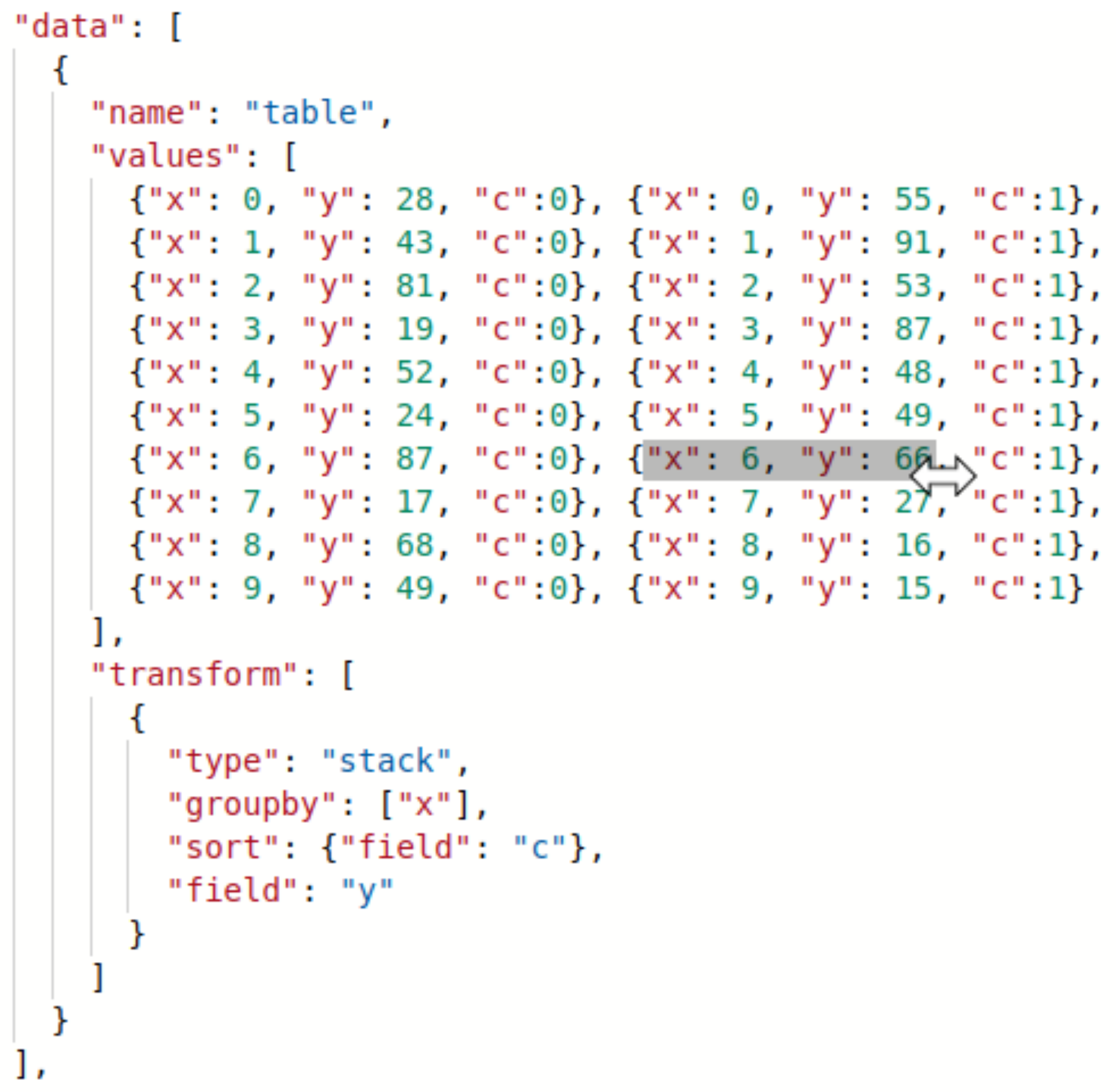
```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y": 27, "c":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```



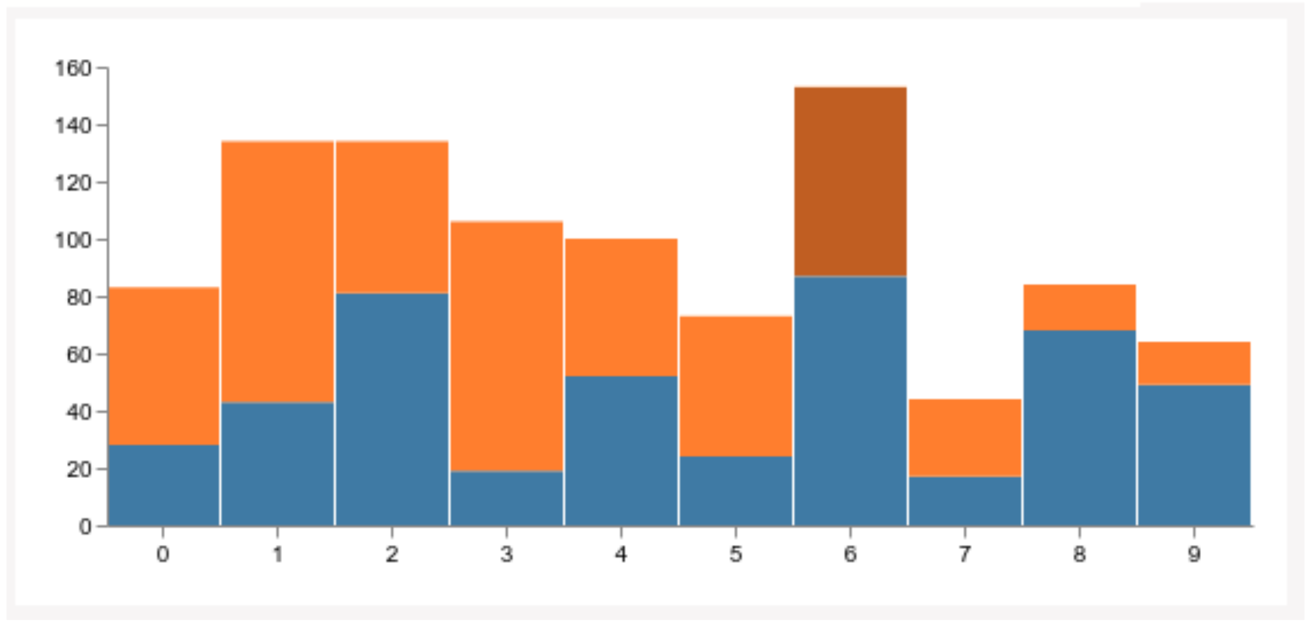
use cases

#3

what if?



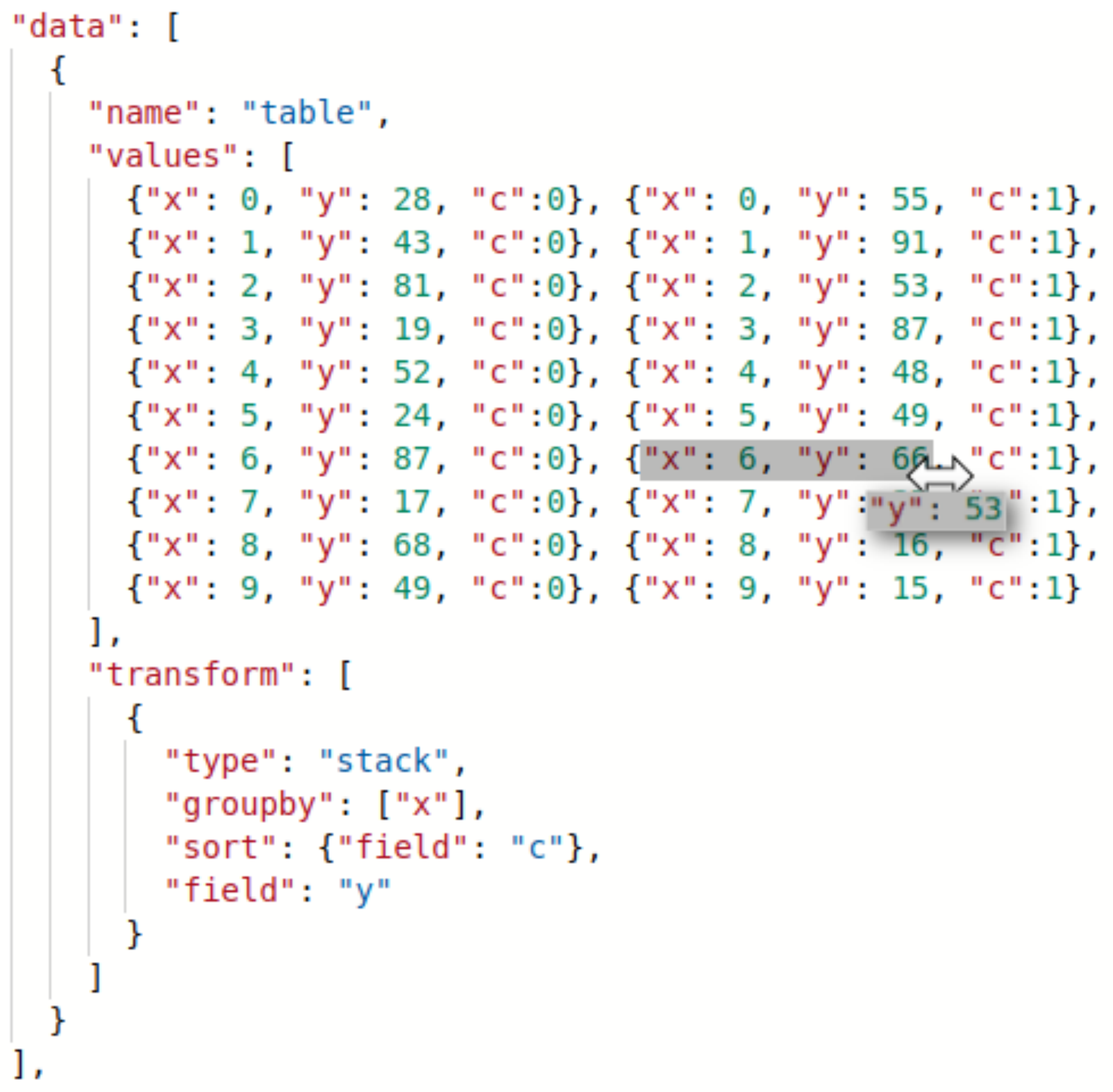
```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y": 27, "c":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```



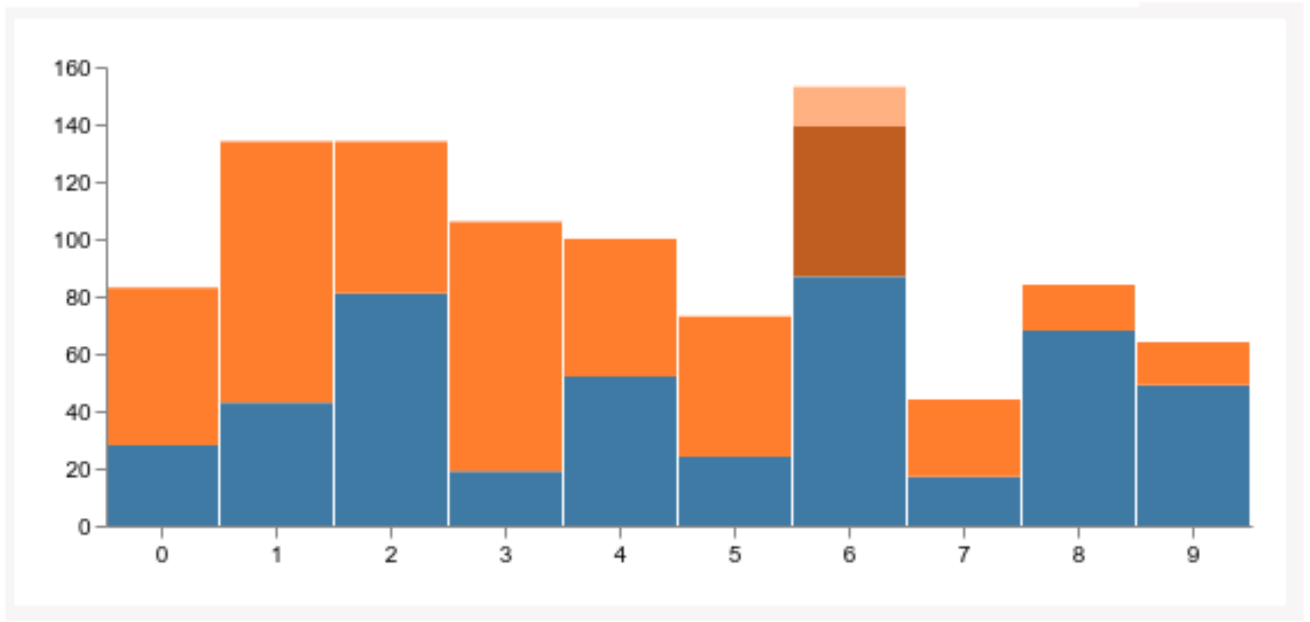
use cases

#3

what if?



```
"data": [
  {
    "name": "table",
    "values": [
      {"x": 0, "y": 28, "c":0}, {"x": 0, "y": 55, "c":1},
      {"x": 1, "y": 43, "c":0}, {"x": 1, "y": 91, "c":1},
      {"x": 2, "y": 81, "c":0}, {"x": 2, "y": 53, "c":1},
      {"x": 3, "y": 19, "c":0}, {"x": 3, "y": 87, "c":1},
      {"x": 4, "y": 52, "c":0}, {"x": 4, "y": 48, "c":1},
      {"x": 5, "y": 24, "c":0}, {"x": 5, "y": 49, "c":1},
      {"x": 6, "y": 87, "c":0}, {"x": 6, "y": 66, "c":1},
      {"x": 7, "y": 17, "c":0}, {"x": 7, "y":"y": 53 ":1},
      {"x": 8, "y": 68, "c":0}, {"x": 8, "y": 16, "c":1},
      {"x": 9, "y": 49, "c":0}, {"x": 9, "y": 15, "c":1}
    ],
    "transform": [
      {
        "type": "stack",
        "groupby": ["x"],
        "sort": {"field": "c"},
        "field": "y"
      }
    ]
  }
],
```



technical goals/innovations

Theoretical

extend work on self-explaining computation to accommodate code and data changes

parts of explanations explain *parts* of data

↓

changes to explanations explain *changes* to data

Practical

extend these partial and incremental computation ideas to user interfaces

viz components which support slicing and delta-visualisation

make explanations accessible directly from data views

prior work

incremental computation

incremental relational algebra — Griffin, Libkin, Trickey (1997); Horn, Perera, Cheney (2018)

self-adjusting computation — Acar (2005); Hammer (2012)

incremental lambda calculus — Cai *et al* (2013)

provenance & program slicing

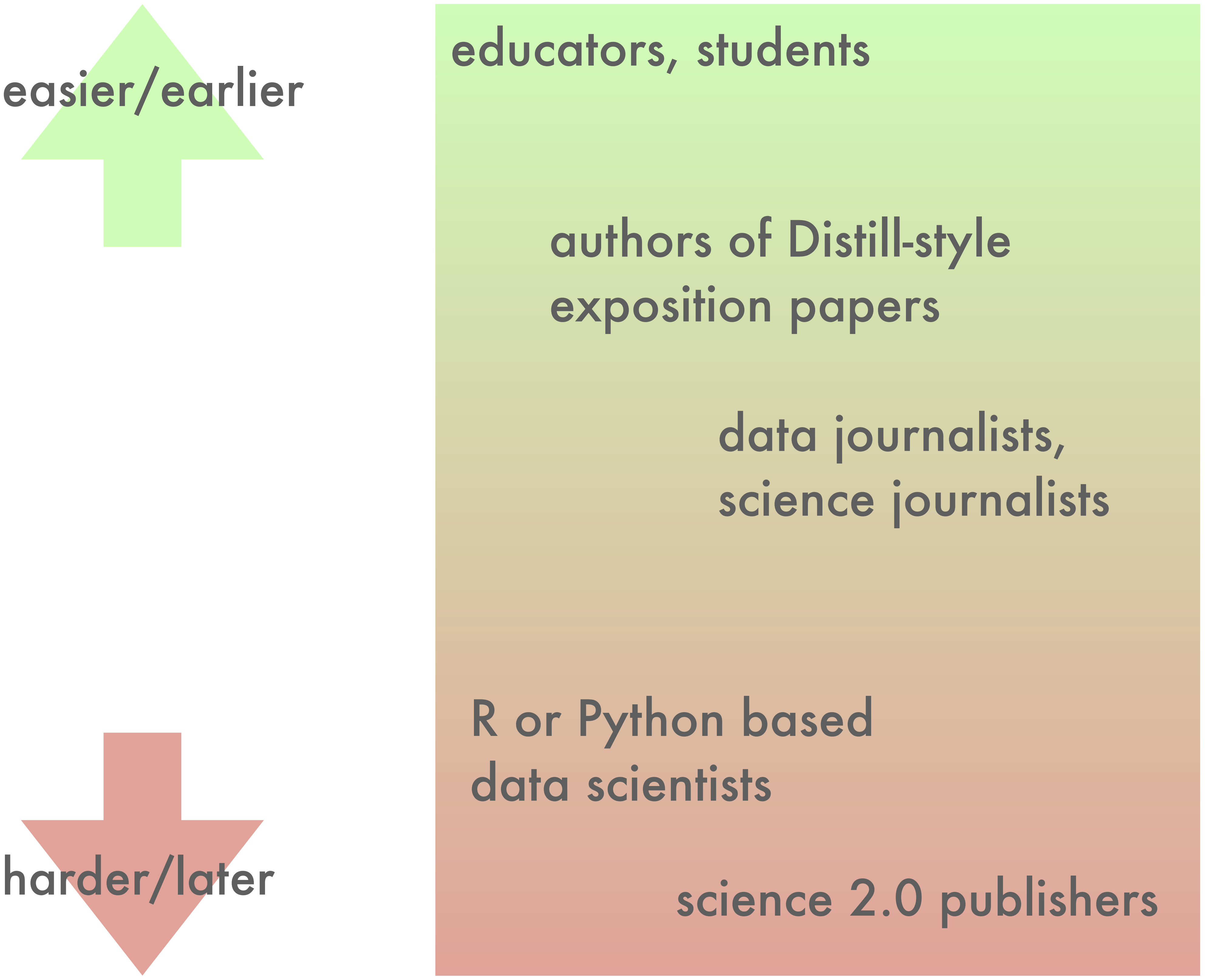
data provenance — Buneman, Khanna, Tan (2001); Cheney, Chiticariu, Tan (2009)

self-explaining computation — Perera *et al* (2012); Cheney, Acar, Perera (2013); Cheney, Ahmed, Acar (2014)

notebooks & data-driven storytelling

Wrattler, The Gamma — Petricek (2017); Petricek, Geddes, Sutton (2018)

target audience


easier/earlier
harder/later
educators, students
authors of Distill-style exposition papers
data journalists, science journalists
R or Python based data scientists
science 2.0 publishers

summary

emerging new role of computation

a literate medium for expressing arguments, narratives, workflows and ideas

summary

static documents superceded by interactive views of real-world processes

James Somers. *The Scientific Paper Is Obsolete*. The Atlantic, April, 2018

for computations to *explain* it needs to be clear how <u>parts</u> relate to <u>other parts</u>

and how <u>changes</u> cause <u>other changes</u>