IN4MATX 133: User Interface Software

Lecture:
Software & Visualization Tools

Announcements

- A2 has been posted to the course website
- We will be covering materials necessary to complete A2 over the next two weeks.
- Discussion classes start today!
    - Topic: Typescript basics

Goals for this Lecture

By the end of this lecture, you should be able to…

- Describe the concepts of threshold and ceiling in software tools and what tool designers should be striving to create
- Explain the relative threshold and ceilings of visualization tools like Protovis, D3, and Vega-Lite
- Describe common visualization primitives like marks, axes, and scales
- Implement simple visualizations with Vega-Lite

Today is a *very* narrow slice
of visualization

If you want more, take IN4MATX 143

Sequential programs (command line)

- Program takes control,
  prompts for input
- Person waits on the program
- Program says when it is ready
  for more input, which the person then provides

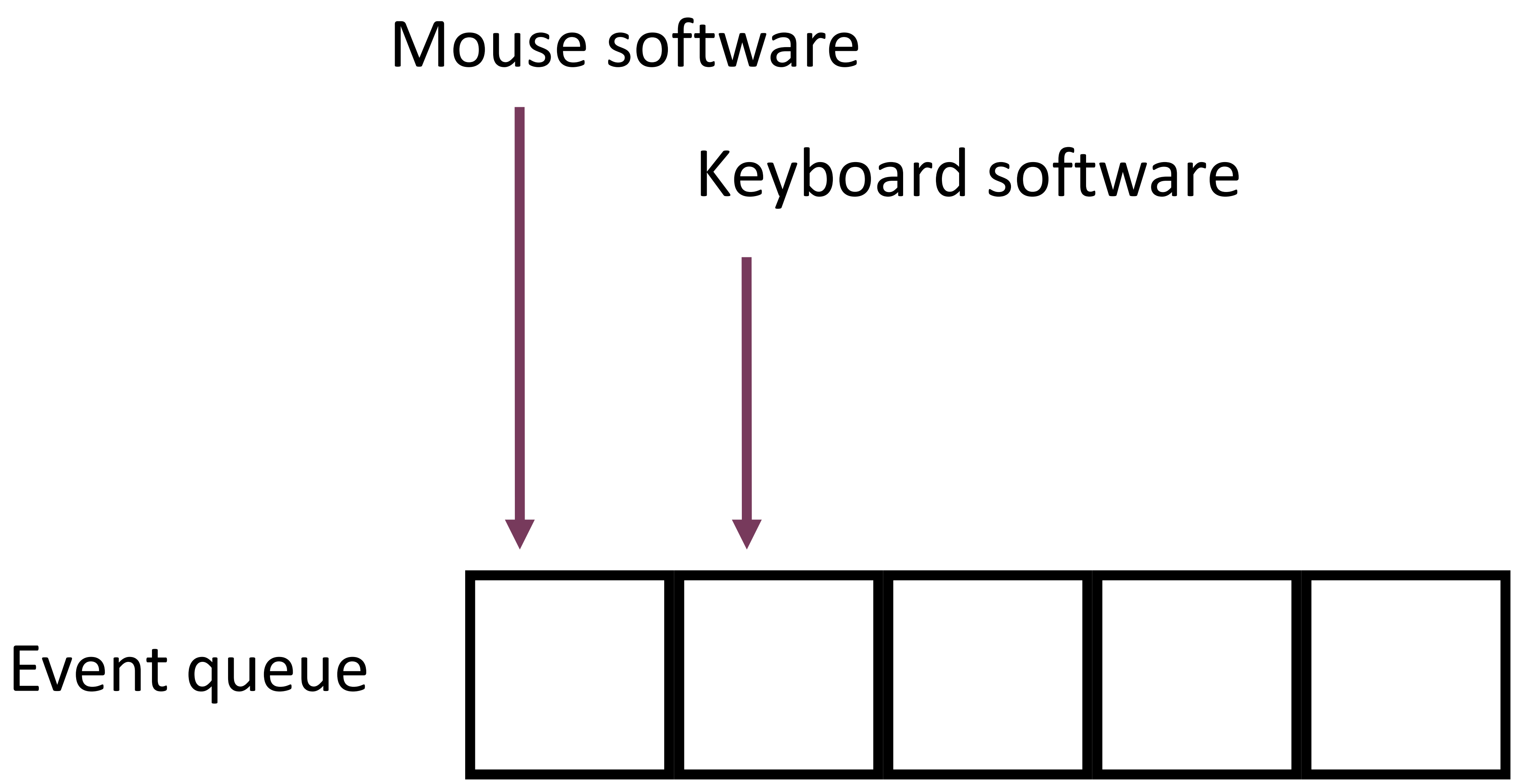
Sequential programs (command line)

```
while true {
  print “Prompt for Input”
  input = read_line_of_text()
  output = do_work()
  print output
}
```

- Person is literally modeled as a file

Event-driven programming

- A program waits for a person to provide input
- All communication is done via events
    - Mouse down, item drag, key up
- All events go in a queue
    - Ensures events are handled in order
    - Hides specifics from applications



Basic interactive software loop

- All interactive software has this loop somewhere

```
do {
  e = read_event();        ⇐ Input
  dispatch_event(e);       ⇐ Processing
  if (damage_exists())
     update_display();     ⇐ Output
} while (e.type != WM_QUIT);
```

Basic interactive software loop

- Maybe if you've made a game, you've built this loop
- But imagine you had to write this loop
  every time you wanted to write a webpage,
  desktop app, or mobile app
- Instead, we rely on tools to handle common operations

```
do {
  e = read_event();
  dispatch_event(e);
  if (damage_exists())
     update_display();
} while (e.type != WM_QUIT);
```

Example: a button

- What's behind a button?
  - Set X and Y boundaries
  - Check if mouse down is within those boundaries
  - Check if mouse up is *also* within those boundaries
  - If so, then fire an event
- What if you had to program this sequence
  every time you wanted to add a button to your website?

Understanding tools

What is a user interface tool?

- Software or libraries which help you build a user interface
  - Bootstrap is a user interface tool, designed to help make interfaces responsive
  - Angular, React, etc. are all user interface tools

Understanding tools

We use tools because they…

- Identify common or important practices
- Package those practices in a framework
- Make it easy to follow those practices
- Make it easier to focus on the application we're building

Understanding tools

Tools enable…

- Faster and more iterative design
- Better implementation than without the tool
- Consistency across applications using the same tool

Understanding tools

Why is designing tools difficult?

- Need to understand the core practices and problems
- Those are often evolving with technology and design
- The tasks people are trying to solve change quickly, so tools struggle to keep up

Brad Myers, Scott E. Hudson, and Randy Pausch. TOCHI, 2000. Past, present, and future of user interface software tools. https://doi.org/10.1145/344949.344959

Understanding tools

Key terms

- Threshold: How hard to get started
- Ceiling: How much can be achieved
- Path of least resistance: Tools influence what interfaces are created
- Moving targets: Changing needs make tools obsolete

Brad Myers, Scott E. Hudson, and Randy Pausch. TOCHI, 2000. Past, present, and future of user interface software tools. https://doi.org/10.1145/344949.344959

Threshold

How hard to get started

- Some tools are harder to pick up
- Depends on what a person knows already
  - A new programming language adds to the threshold
  - If a tool borrows concepts from another popular tool, it will be easier for many people to pick up

Brad Myers, Scott E. Hudson, and Randy Pausch. TOCHI, 2000. Past, present, and future of user interface software tools. https://doi.org/10.1145/344949.344959

Ceiling

How much can be achieved

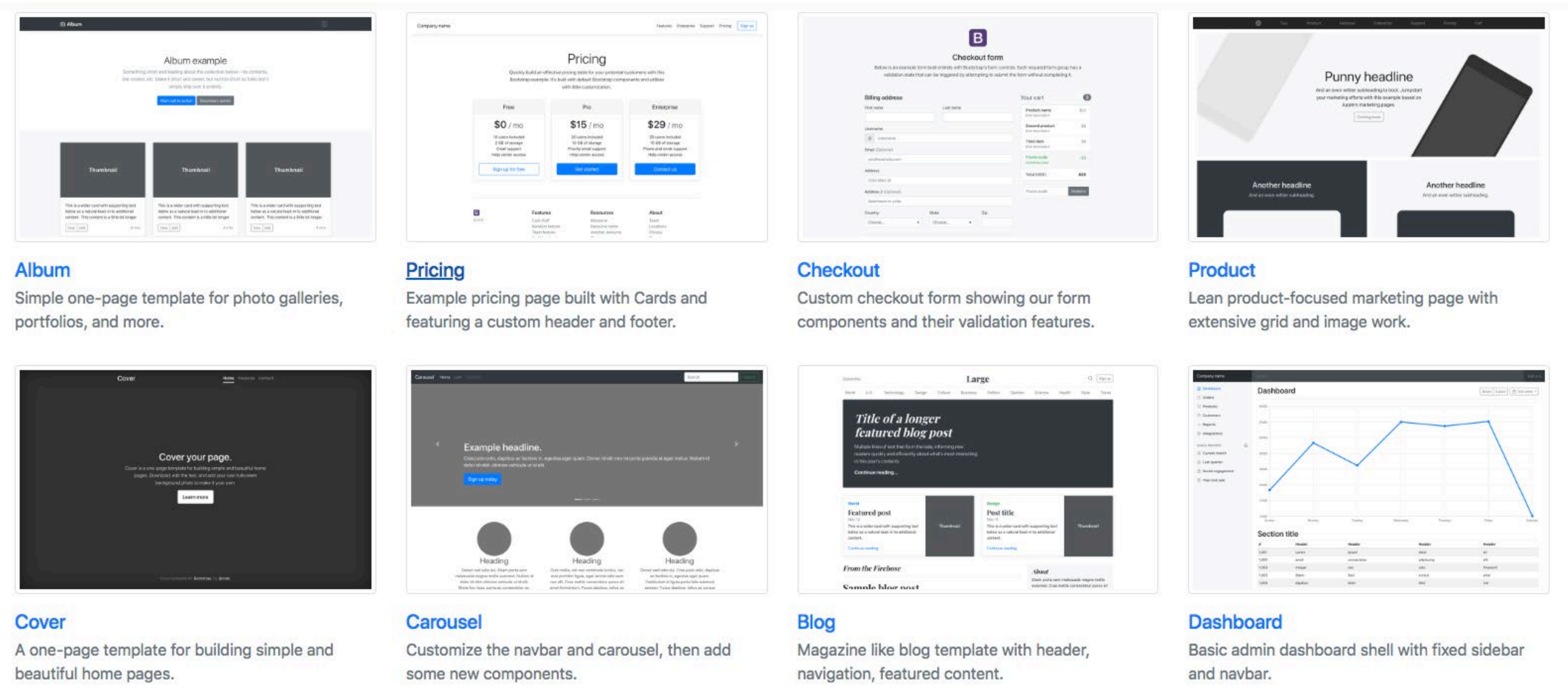
- Tools restrict what’s possible
    - Your program could do much more if it had direct access to the bits on your computing device

Brad Myers, Scott E. Hudson, and Randy Pausch. TOCHI, 2000. Past, present, and future of user interface software tools. https://doi.org/10.1145/344949.344959

Path of least resistance

Tools influence what interfaces are created

- Remember the concern that all Bootstrap pages look similar?
- Linguistic Relativity
  - Or…the Sapir-Whorf Hypothesis
  - Roughly, some thoughts in one language cannot be expressed or understood in another language
- In UI, our tools frame how we think about interaction and design



Brad Myers, Scott E. Hudson, and Randy Pausch. TOCHI, 2000. Past, present, and future of user interface software tools. https://doi.org/10.1145/344949.344959

Moving targets

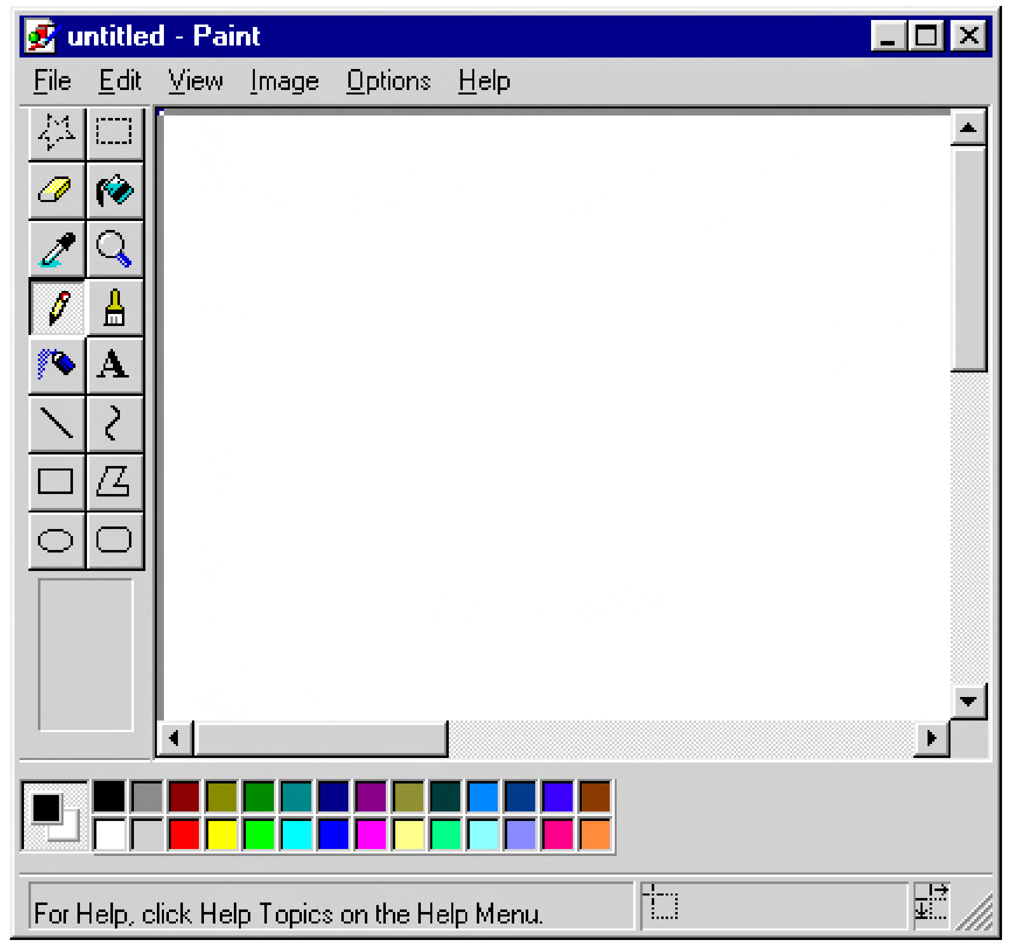
Changing needs make tools obsolete

- Codification eventually constrains design
  - Our understanding of how people interact with technology improves
  - New technology comes along to change the needs of tools
  - Example: Virtual reality has wildly different interactions and tool needs

Brad Myers, Scott E. Hudson, and Randy Pausch. TOCHI, 2000. Past, present, and future of user interface software tools. https://doi.org/10.1145/344949.344959

Threshold and ceiling

- It's all relative; no absolute measure
- Tools should be *low threshold*
  - Easy to pick up
- But tools should also be *high ceiling*
  - Can do a lot
- The best tools are both
  - Photoshop introduces tutorials, etc. to lower the threshold



Ok, so what does any of this have to do with visualization?

Scalable Vector Graphics (SVG)

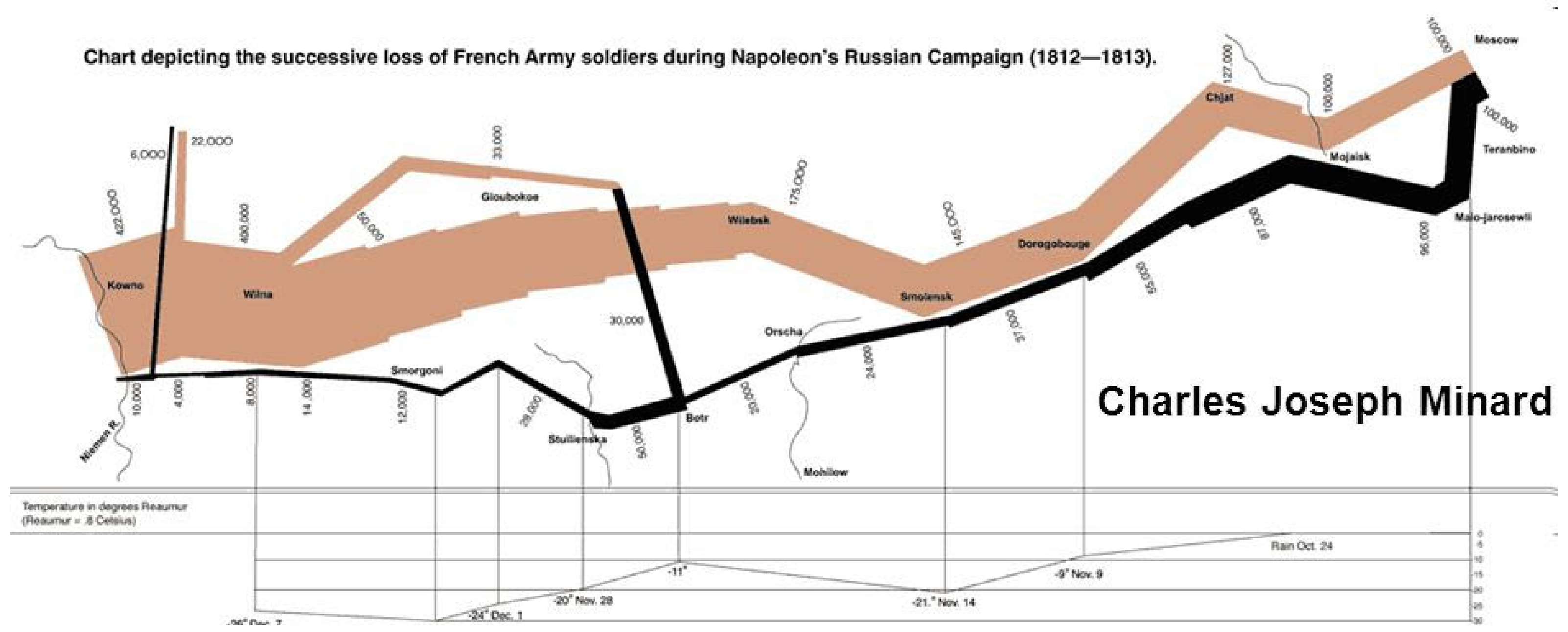
- XML format for specifying graphics
  - Looks somewhat like HTML
  - Most browsers can render them
- Composed of lines, circles, rectangles, etc.

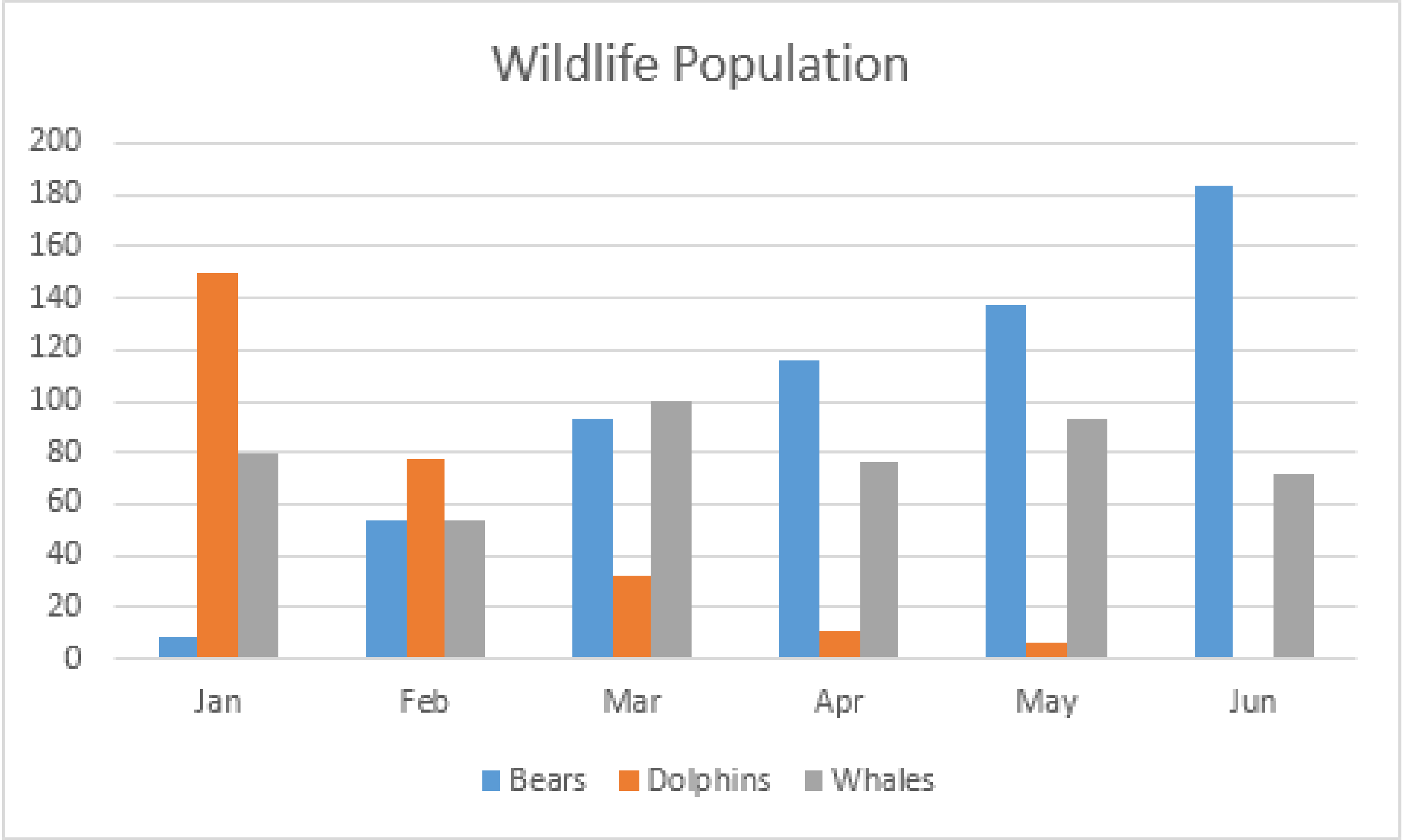
```
<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-
width="4" fill="yellow" />
</svg>
```

Visualization tools

- Are governed
  by the same principles
- Scaleable vector graphics (svg)
  - High ceiling, but high threshold
- Microsoft excel
  - Low threshold, but low ceiling

Successive Loss of French Army During Napoleon's Russian Campaign





https://www.edwardtufte.com/tufte/posters

Lower threshold

Higher ceiling

Charting tools

Chart Typologies
Excel, Google Charts

Declarative languages

Visual Analysis Grammars
ggplot, VizQL

Visualization Grammars
Protovis, D3.js, Vega-Lite, Draco

Programming languages

Graphics APIs
SVG, Processing , OpenGL

Declarative languages

- Programming by describing *what*, not *how*
- Separate specification (*what you want*)
  from execution (*how it should be computed*)
- Contrasts to **imperative** programming, where you must give explicit steps

Declarative languages

HTML

Markup
language

CSS

Styling
language

JS

Programming
language

Declarative languages

HTML

Declarative
language

CSS

Declarative
language

JS

Imperative
language

Declarative languages

HTML

```html
<main class="container">
  <div class="row">
    <div class="col-3">A</div>
    <div class="col-6">B</div>
    <div class=“col-4">C</div>
    <div class=“col">D</div>
    <div class="col">E</div>
  </div>
</main>
```

What should be rendered, but not how

JS

```javascript
var array = ['1', 'fish', 2, 'blue'];
array[5] = 'dog';
array.push('2');
array[2] = array[array.length - 1] - 4;
array[0] = typeof array[2];
array[4] = array.indexOf('blue');

console.log(array.join('*'));
```

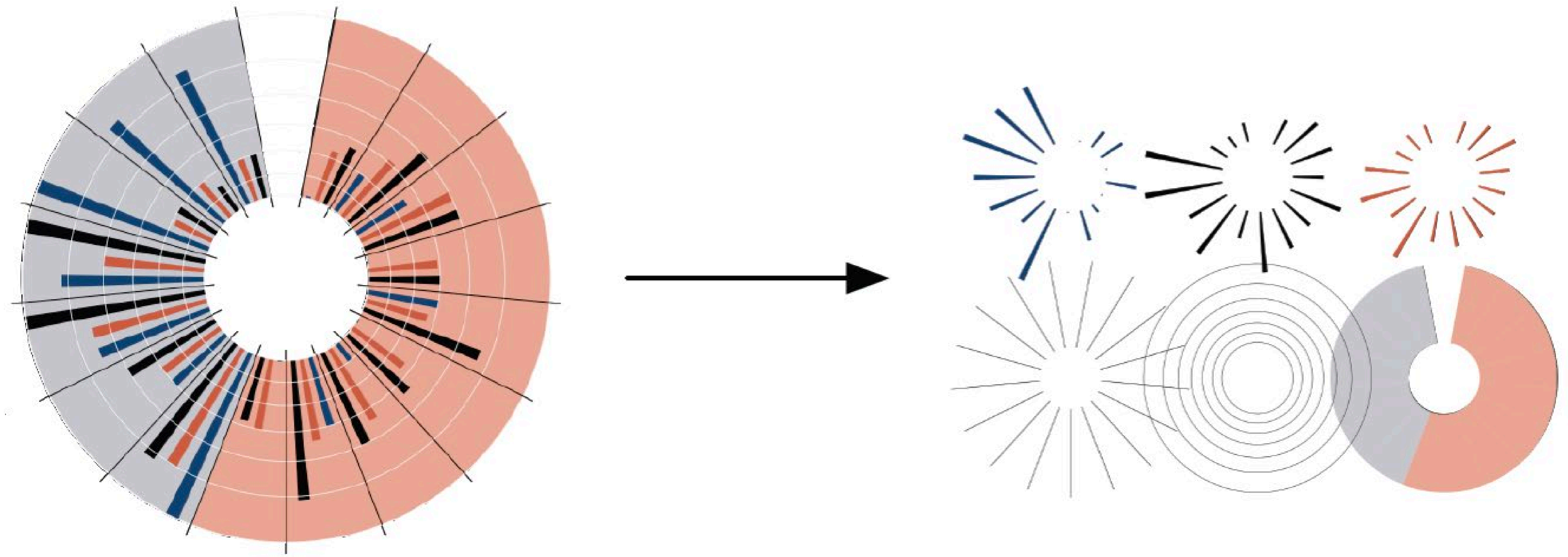
Step-by-step

Why declarative languages?

- Faster iteration, less code, lower threshold
- Can be generated programmatically
- Generally considered easier to learn than programming/imperative languages like JavaScript

Protovis

- Initial grammar for visualization
- A composition of data-representative marks
    - Self-contained JavaScript model (doesn't export to SVG or anything else)

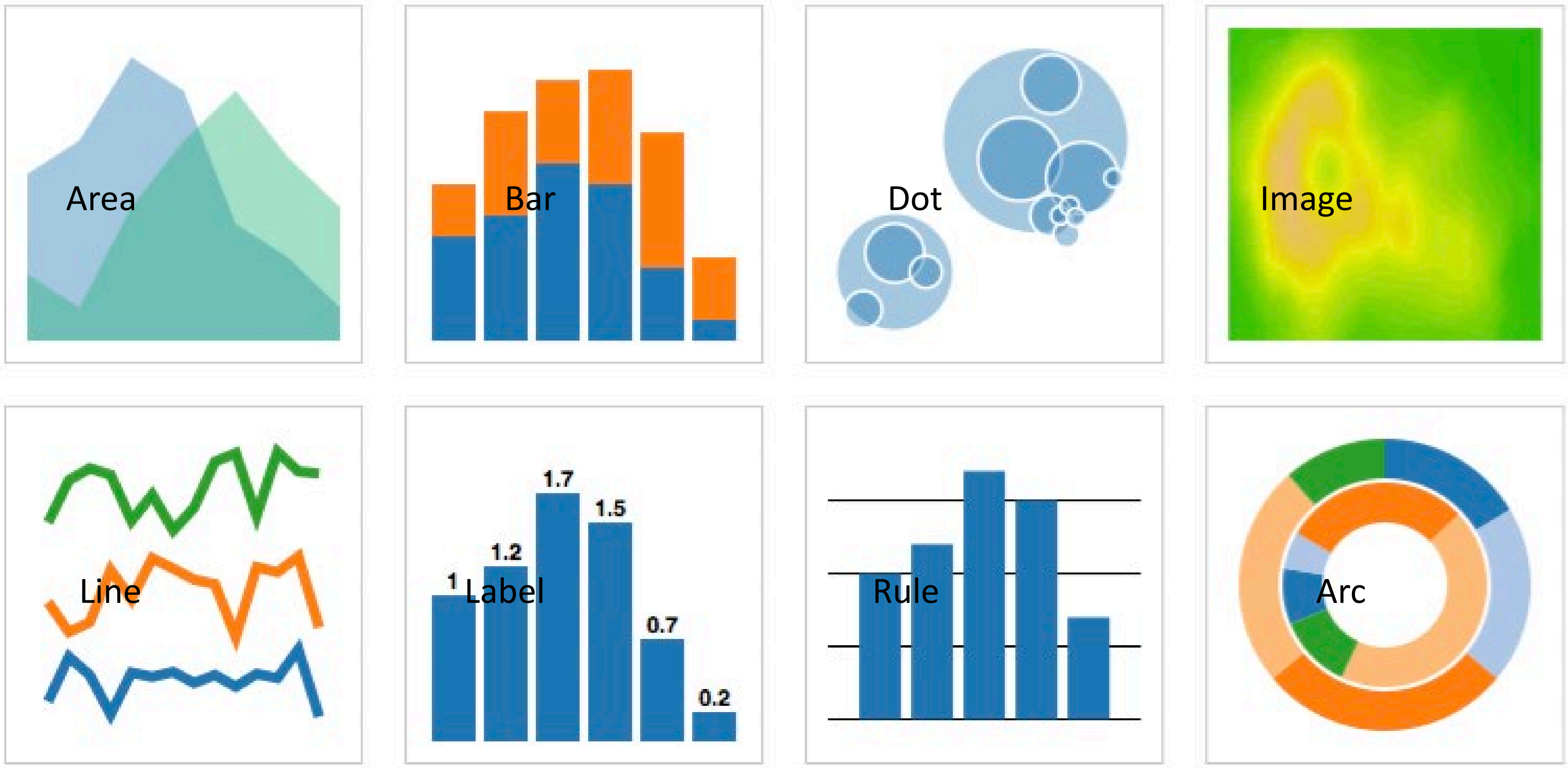


Michael Bostock, Jeffrey Heer. IEEE Vis, 2009. Protovis: A Graphical Toolkit for Visualization.

https://doi.org/10.1109/TVCG.2009.174

Protovis

- Marks: graphical primitives
  - Marks specify how content should be rendered



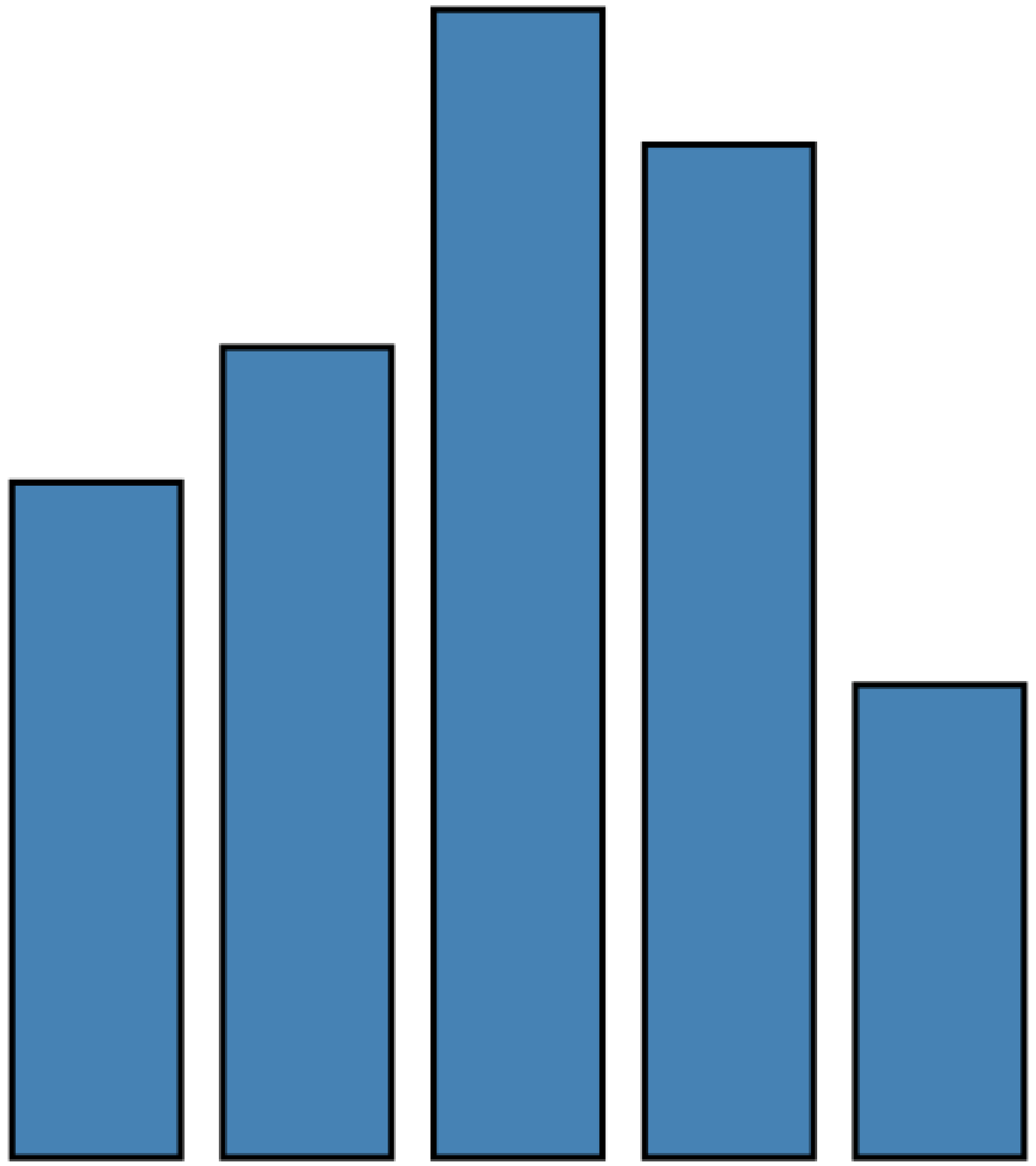
Michael Bostock, Jeffrey Heer. IEEE Vis, 2009. Protovis: A Graphical Toolkit for Visualization.
https://doi.org/10.1109/TVCG.2009.174

Protovis

```
var vis = new pv.Panel();
vis.add(pv.Bar)
.data([1, 1.2, 1.7, 1.5,
0.7])
.visible(true)
.left((d) => this.index * 25)
.bottom(0)
.width(20)
.height((d) => d * 80)
.fillStyle("blue")
.strokeStyle("black")
.lineWidth(1.5);
vis.render();
```

Mark

Literally specifies which pixel each bar should start at and how many pixels tall it should be

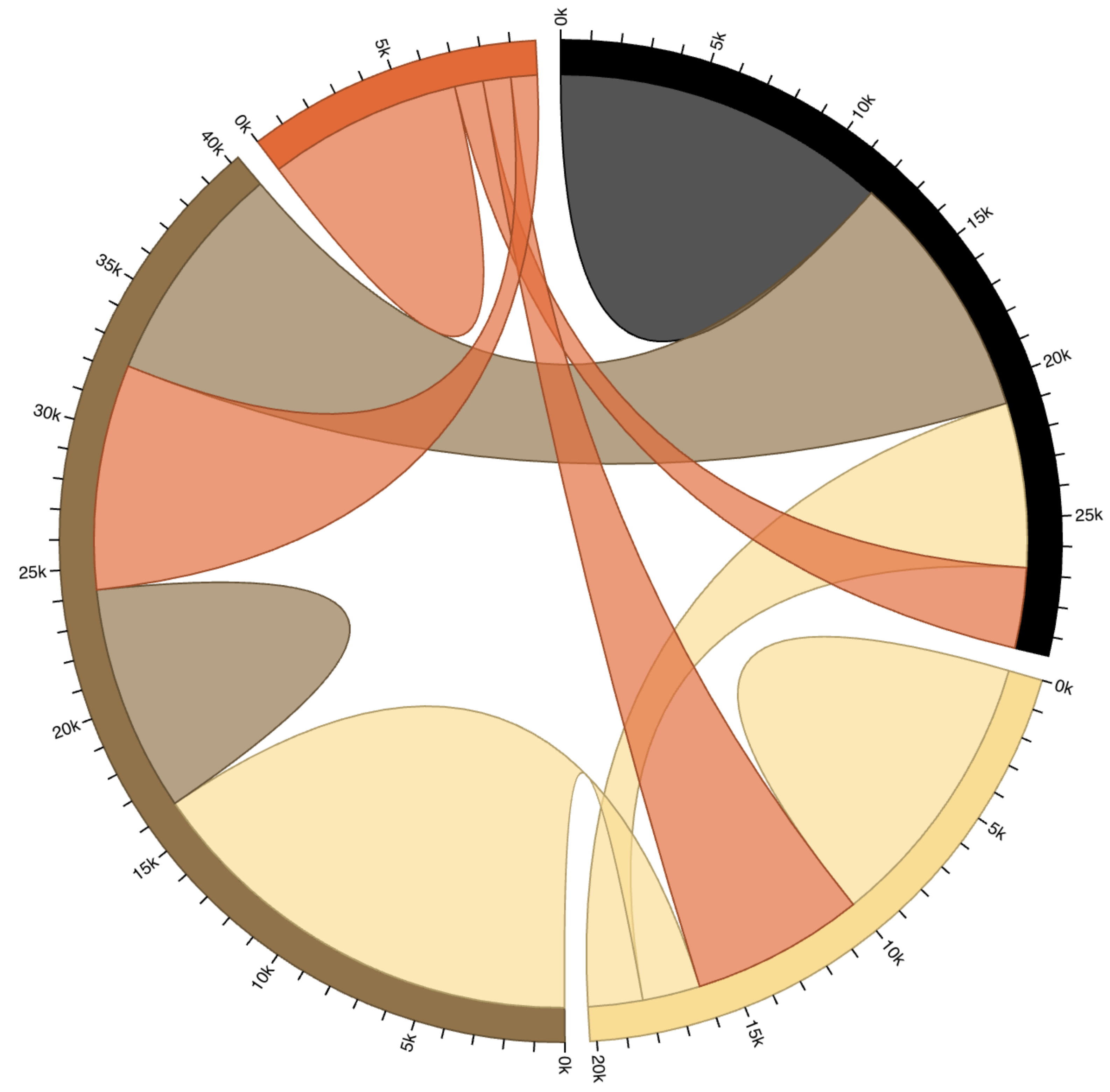


Michael Bostock, Jeffrey Heer. IEEE Vis, 2009. Protovis: A Graphical Toolkit for Visualization.

https://doi.org/10.1109/TVCG.2009.174

D3

- Binds data directly to a web page's DOM by editing a SVG
  - More expressive! Can make anything an SVG can make
  - Enables interactivity, can access mouse & keyboard events through the same tools as a browser
  - Much more complex…



Michael Bostock, Vadim Ogievetsky, Jeffrey Heer. IEEE Vis, 2011. D3: Data Driven Documents.

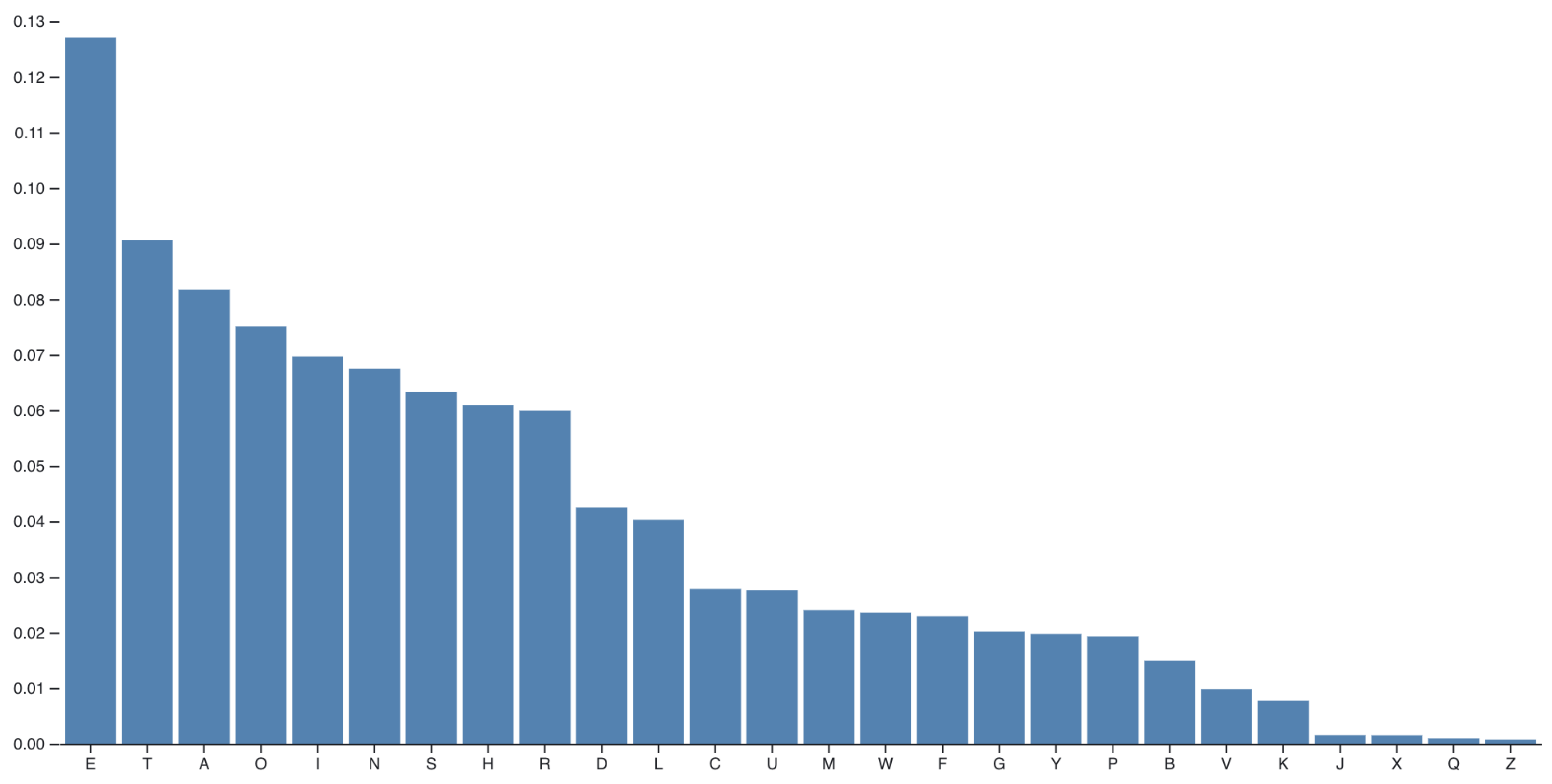
http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.185

D3

```
var svg = d3.select(DOM.svg(width,
height));

svg.append("g")
   .attr("fill", "steelblue")
.selectAll("rect").data(data).enter()
.append("rect")
   .attr("x", d => x(d.name))
   .attr("y", d => y(d.value))
   .attr("height", d => y(0) -
y(d.value))
   .attr("width", x.bandwidth());

svg.append("g")
   .call(xAxis);

svg.append("g")
   .call(yAxis);
```

Find SVG in the DOM

No more mention of marks!

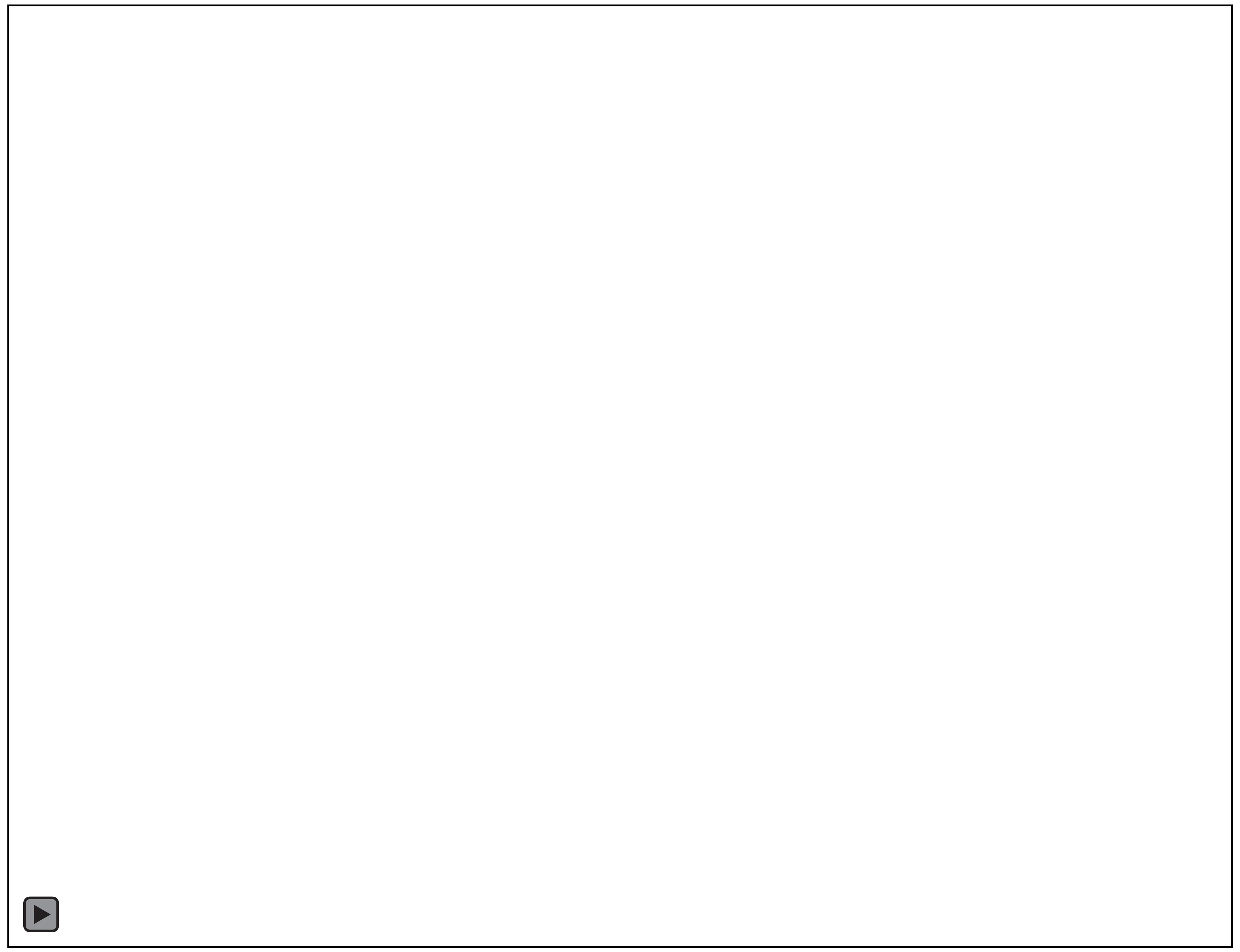


Michael Bostock, Vadim Ogievetsky, Jeffrey Heer. IEEE Vis, 2011. D3: Data Driven Documents.
http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.185

D3

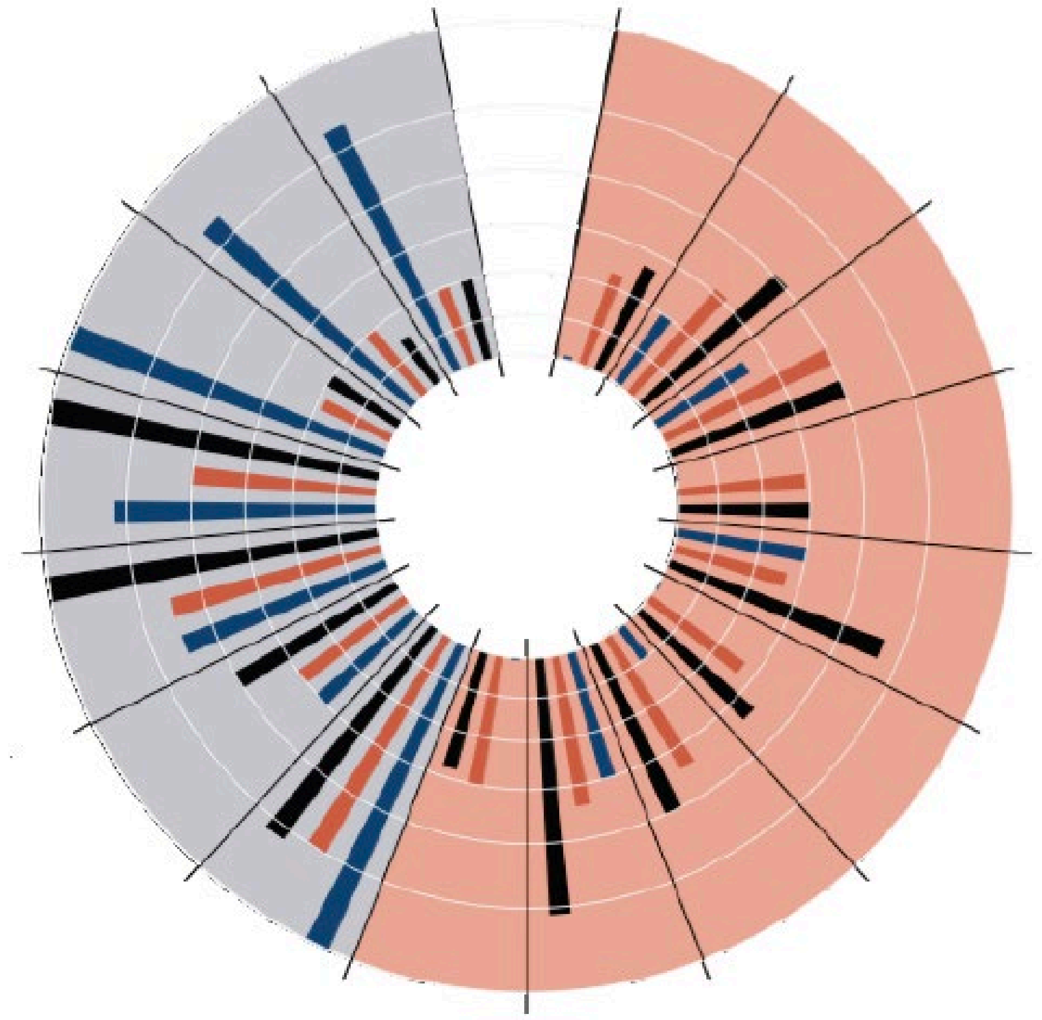
```
var svg = d3.select("body").append("svg")
      .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
      .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

d3.tsv("VotingInformation.tsv", function(error, data){

      // filter year
      var data = data.filter(function(d){return d.Year == '2012';});
      // Get every column value
      var elements = Object.keys(data[0])
            .filter(function(d){
                  return ((d != "Year") & (d != "State"));
            });
      var selection = elements[0];

      var y = d3.scale.linear()
                  .domain([0, d3.max(data, function(d){
                        return +d[selection];
                  })])
                  .range([height, 0]);

      var x = d3.scale.ordinal()
                  .domain(data.map(function(d){ return d.State;}))
                  .rangeBands([0, width]);


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

      var yAxis = d3.svg.axis()
            .scale(y)
          .orient("left");

      svg.append("g")
      .attr("class", "x axis")
      .attr("transform", "translate(0," + height + ")")
      .call(xAxis)
      .selectAll("text")
      .style("font-size", "8px")
      .style("text-anchor", "end")
      .attr("dx", "-.8em")
      .attr("dy", "-.55em")
      .attr("transform", "rotate(-90)" );


      svg.append("g")
      .attr("class", "y axis")
      .call(yAxis);

      svg.selectAll("rectangle")
            .data(data)
            .enter()
            .append("rect")
            .attr("class","rectangle")
            .attr("width", width/data.length)
            .attr("height", function(d){
                  return height - y(+d[selection]);
            })
            .attr("x", function(d, i){
                  return (width / data.length) * i ;
            })
            .attr("y", function(d){
                  return y(+d[selection]);
            })
            .append("title")
            .text(function(d){
                  return d.State + " : " + d[selection];
            });

      var selector = d3.select("#drop")
      .append("select")
      .attr("id","dropdown")
      .on("change", function(d){
            selection = document.getElementById("dropdown");

            y.domain([0, d3.max(data, function(d){
                        return +d[selection.value];})]);

            yAxis.scale(y);

            d3.selectAll(".rectangle")
                  .transition()
                  .attr("height", function(d){
                              return height - y(+d[selection.value]);
                        })
                        .attr("x", function(d, i){
                              return (width / data.length) * i ;
                        })
                        .attr("y", function(d){
                              return y(+d[selection.value]);
                        })
                  .ease("linear")
                  .select("title")
                  .text(function(d){
                        return d.State + " : " + d[selection.value];
                  });

            d3.selectAll("g.y.axis")
                  .transition()
                  .call(yAxis);

         });

    selector.selectAll("option")
      .data(elements)
      .enter().append("option")
      .attr("value", function(d){
        return d;
      })
      .text(function(d){
        return d;
      })


});
```

~118 lines of code,
plus data in a separate file

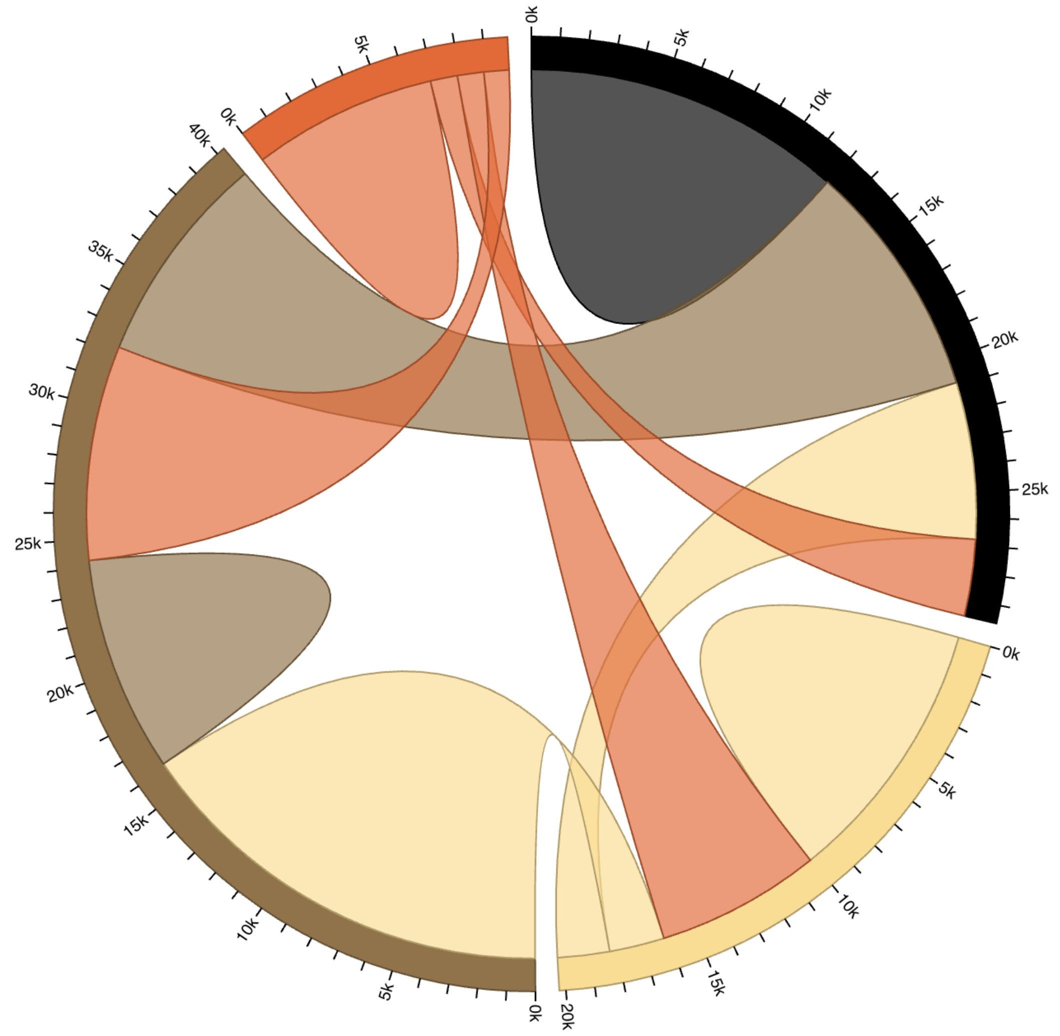


Michael Bostock, Vadim Ogievetsky, Jeffrey Heer. IEEE Vis, 2011. D3: Data Driven Documents.

http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.185



D3



Protovis
Low(er) ceiling

D3
High(er) ceiling

Compared to excel, etc., both have a high ceiling

But both have a pretty high threshold!

Vega-Lite: lowering the threshold

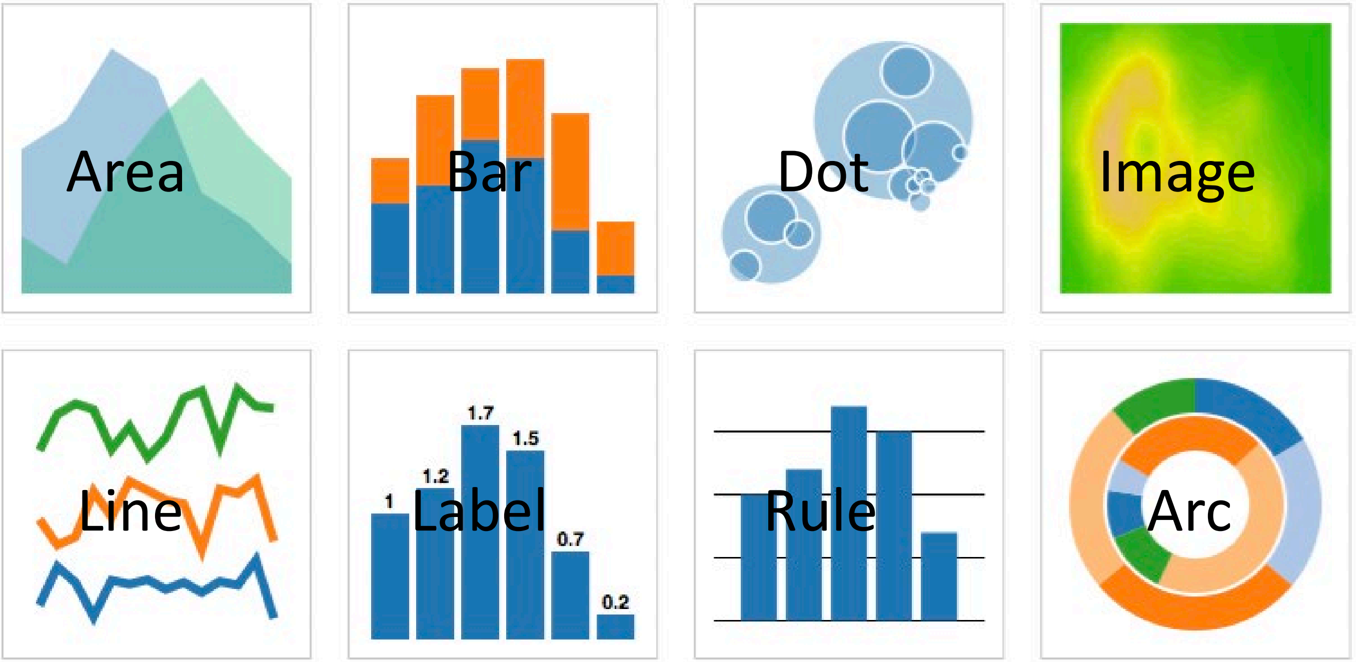
Lowering the threshold

- Goal: “create an *expressive* (high ceiling) yet *concise* (low threshold) declarative language for specifying visualizations”

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

https://doi.org/10.1109/TVCG.2016.2599030

Vega-Lite

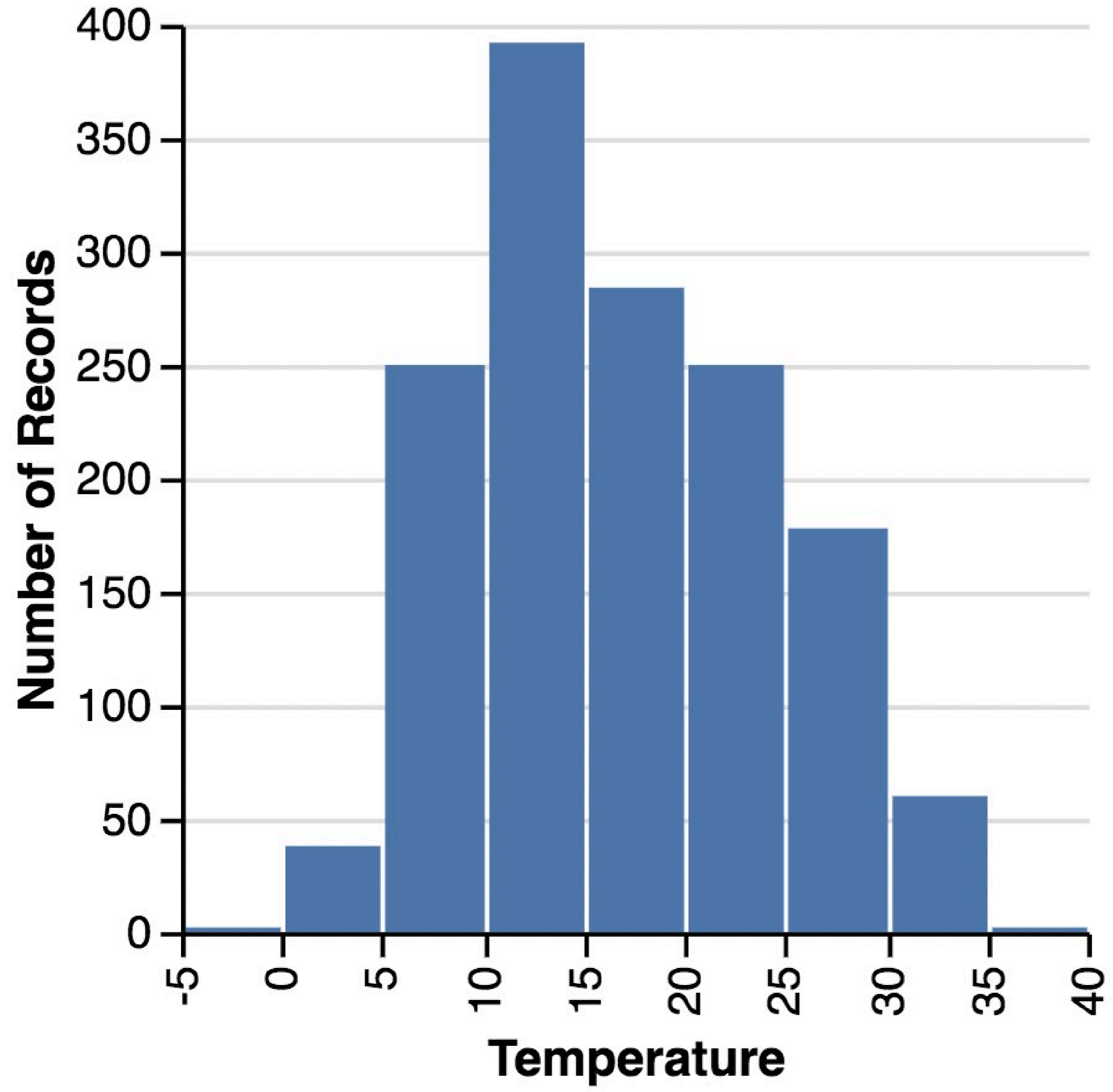


- Grammar of graphics
  - Data: input data to visualize
  - Mark: Data-representative graphics
  - Transform: whether to filter, aggregate, bin, etc.
  - Encoding: mapping between data and mark properties
  - Scale: map between data values and visual values
  - Guides: axes & legends that visualize scales

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics
https://doi.org/10.1109/TVCG.2016.2599030

Vega-lite

Making a histogram



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

https://doi.org/10.1109/TVCG.2016.2599030

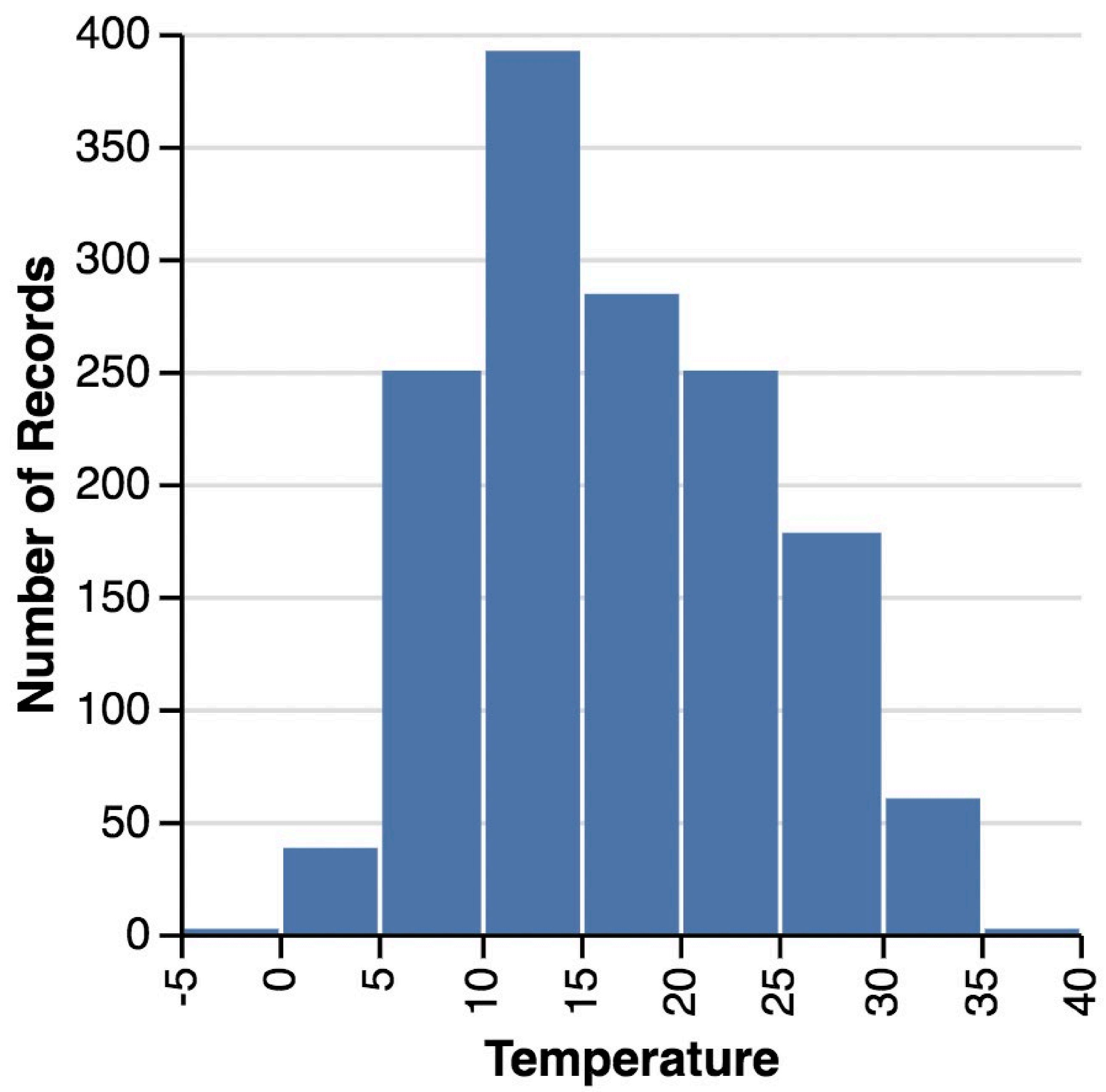
JSON file

```
[
    {
        "date": "2015/01/01",
        "weather": "sun",
        "temperature": 1.1999999999999997
    },
    {
        "date": "2015/01/02",
        "weather": "fog",
        "temperature": 2.8
    },
    {
        "date": "2015/01/03",
        "weather": "fog",
        "temperature": 3.35
    },
    {
        "date": "2015/01/04",
        "weather": "fog",
        "temperature": 6.949999999999999
    },
    {
        "date": "2015/01/05",
        "weather": "fog",
        "temperature": 10.8
    },
...
```

Vega-lite

Histogram = (Bar with x=binned field, y=count)

- Bin records by their temperature
- Count how many records fall into each bin
- Render those bins as vertical bars



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

https://doi.org/10.1109/TVCG.2016.2599030

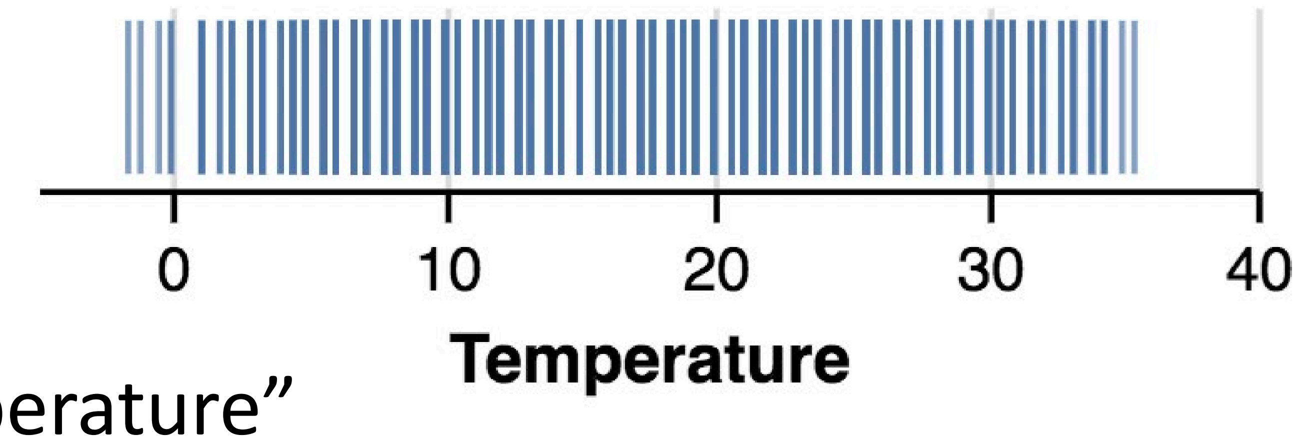
Vega-lite

Histogram = (Bar with x=binned field, y=count)

```
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative"
    }
}
```

Set mark as a tick

Encode x according to the "temperature" field



Four types:
quantitative (numerical)
temporal (time)
ordinal (ordered)
nominal (categorical)

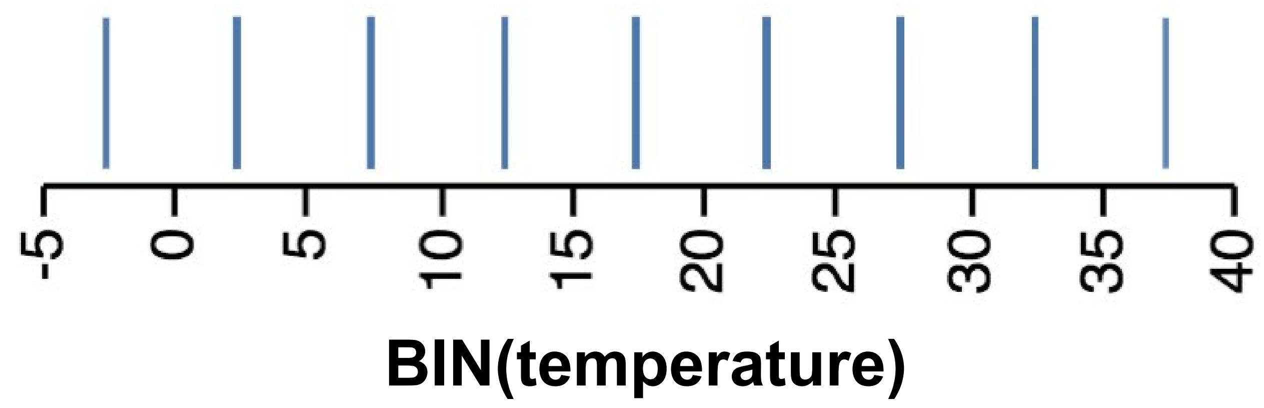
Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics
https://doi.org/10.1109/TVCG.2016.2599030

Vega-lite

Histogram = (Bar with x=binned field, y=count)

```
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    }
}
```

Bin values by x dimension



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

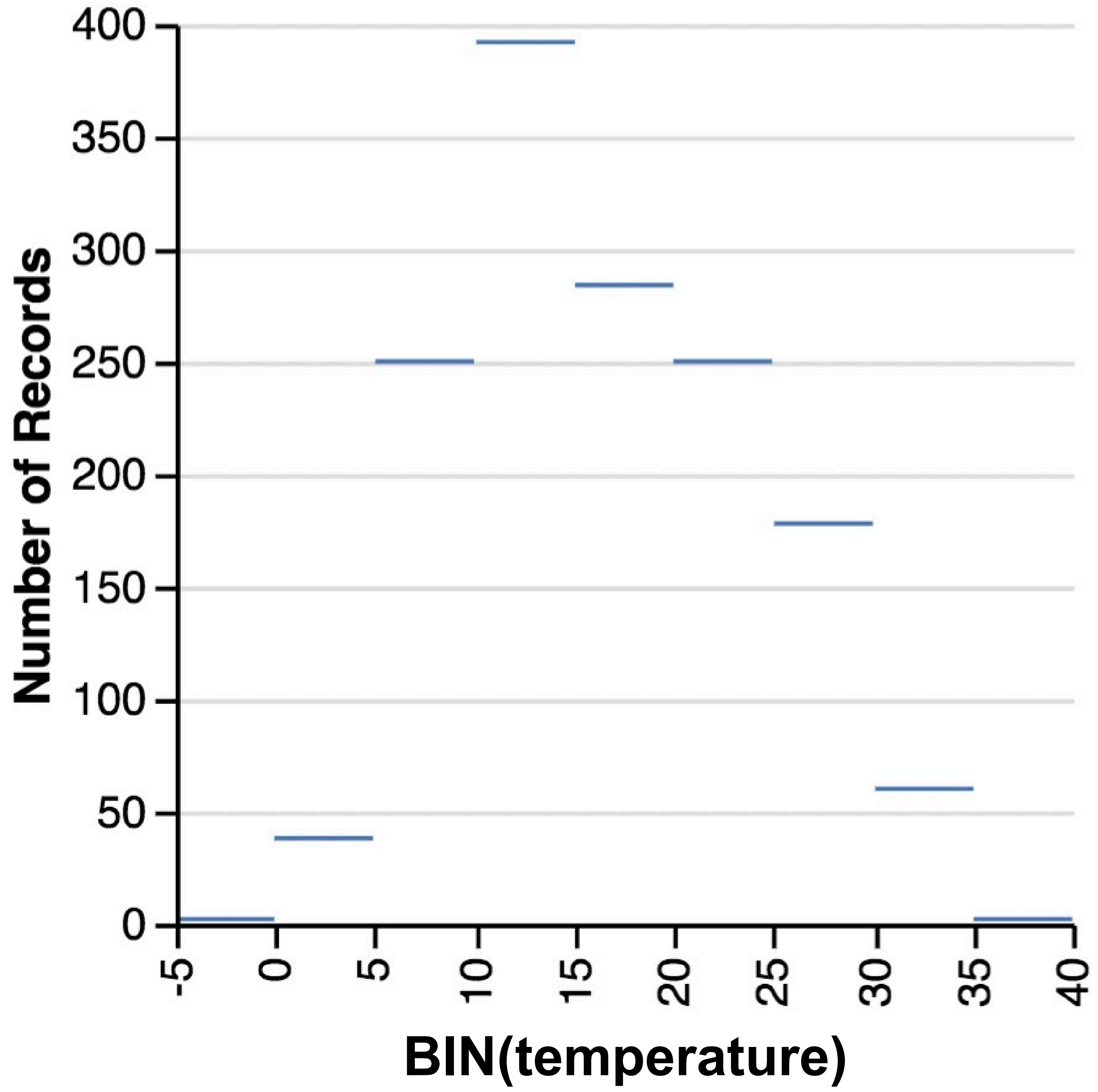
https://doi.org/10.1109/TVCG.2016.2599030

Vega-lite

Histogram = (Bar with x=binned field, y=count)

```
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
}
```

y should aggregate the bins by counting how many values are in them



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

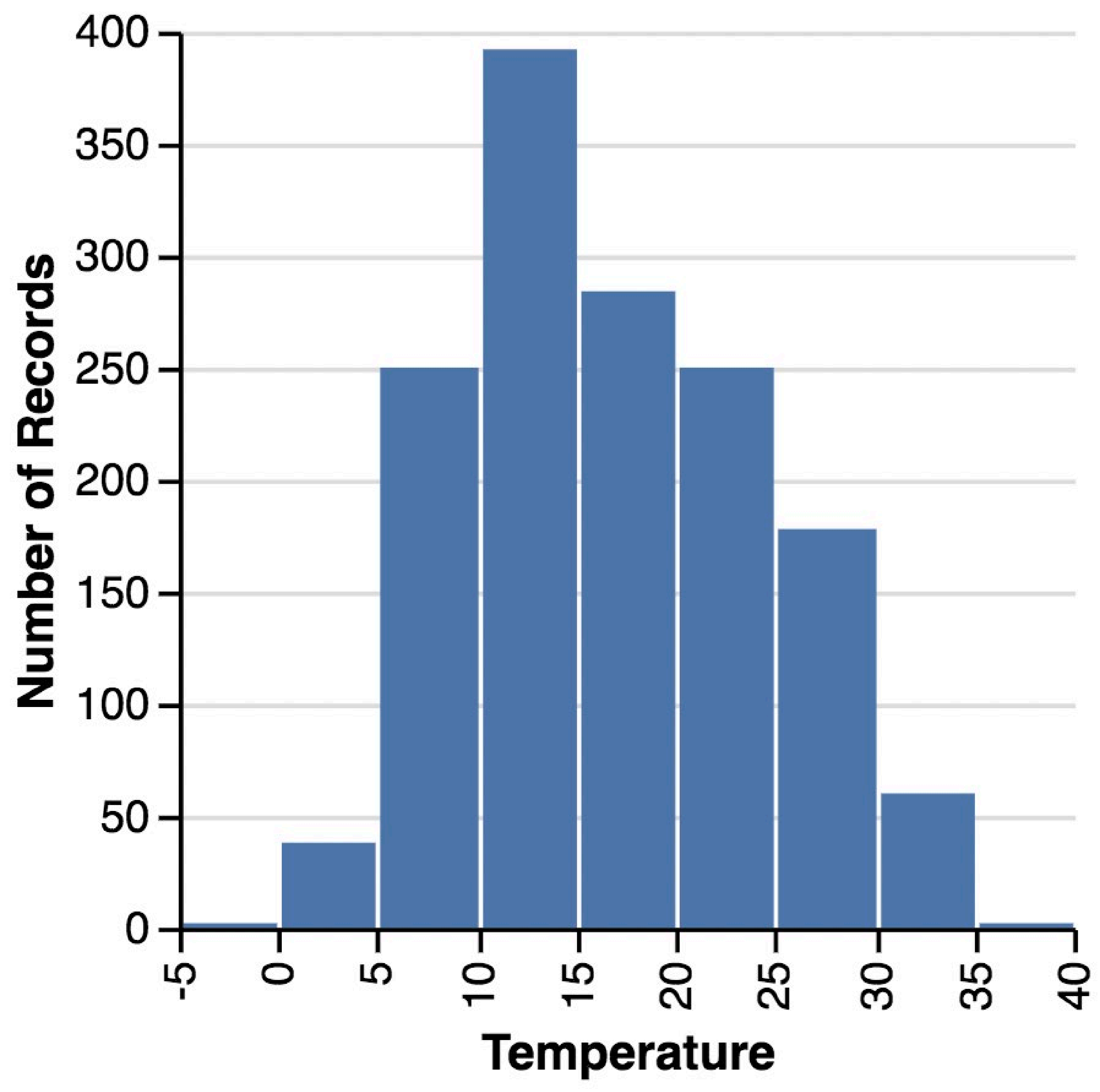
https://doi.org/10.1109/TVCG.2016.2599030

Vega-lite

Histogram = (Bar with x=binned field, y=count)

```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```

Change the mark to a bar



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

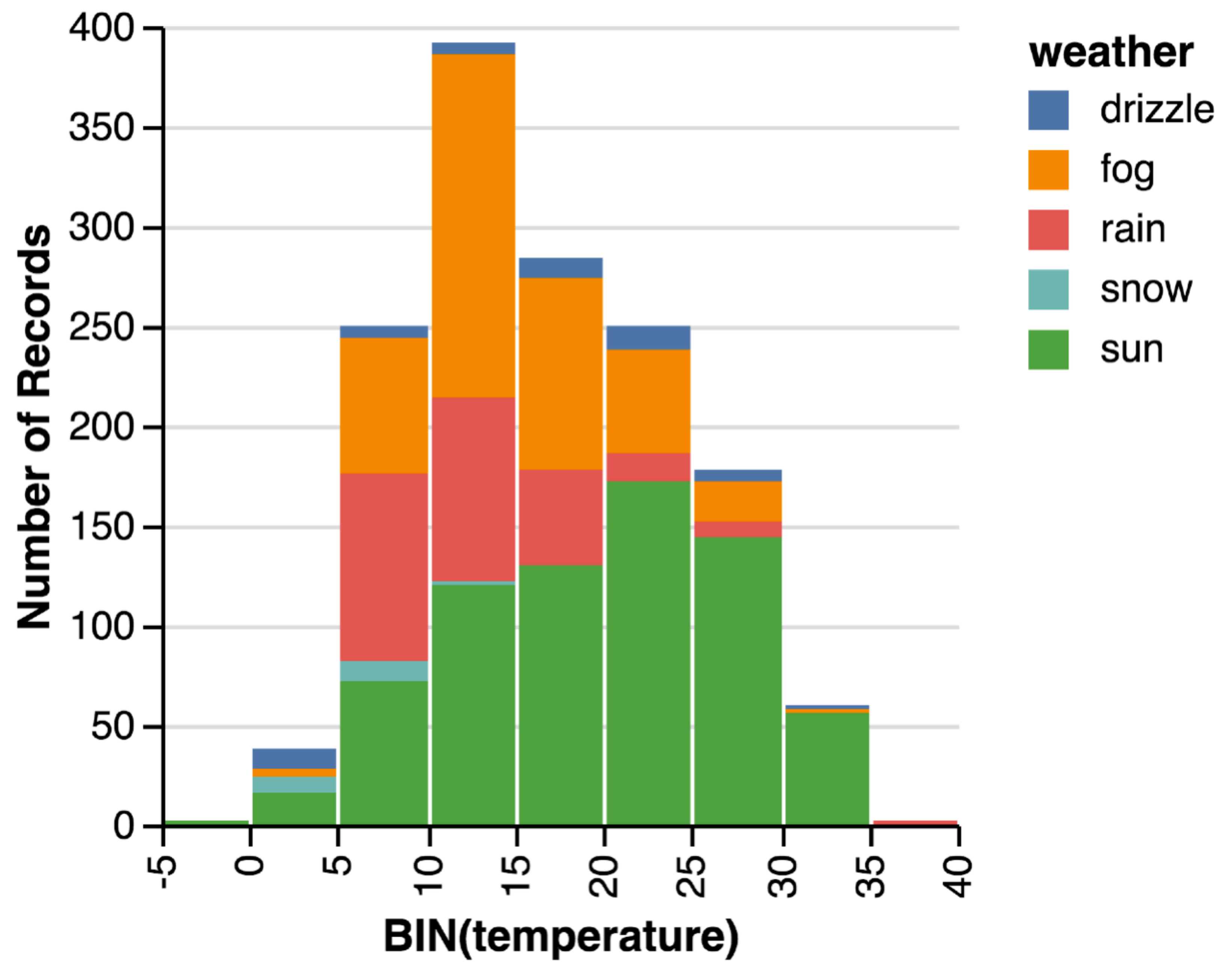
https://doi.org/10.1109/TVCG.2016.2599030

Vega-lite

Histogram + Color

```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    },
    color: {
     field: "weather",
     type: "nominal"
    }
}
```

Set the color to follow the weather field



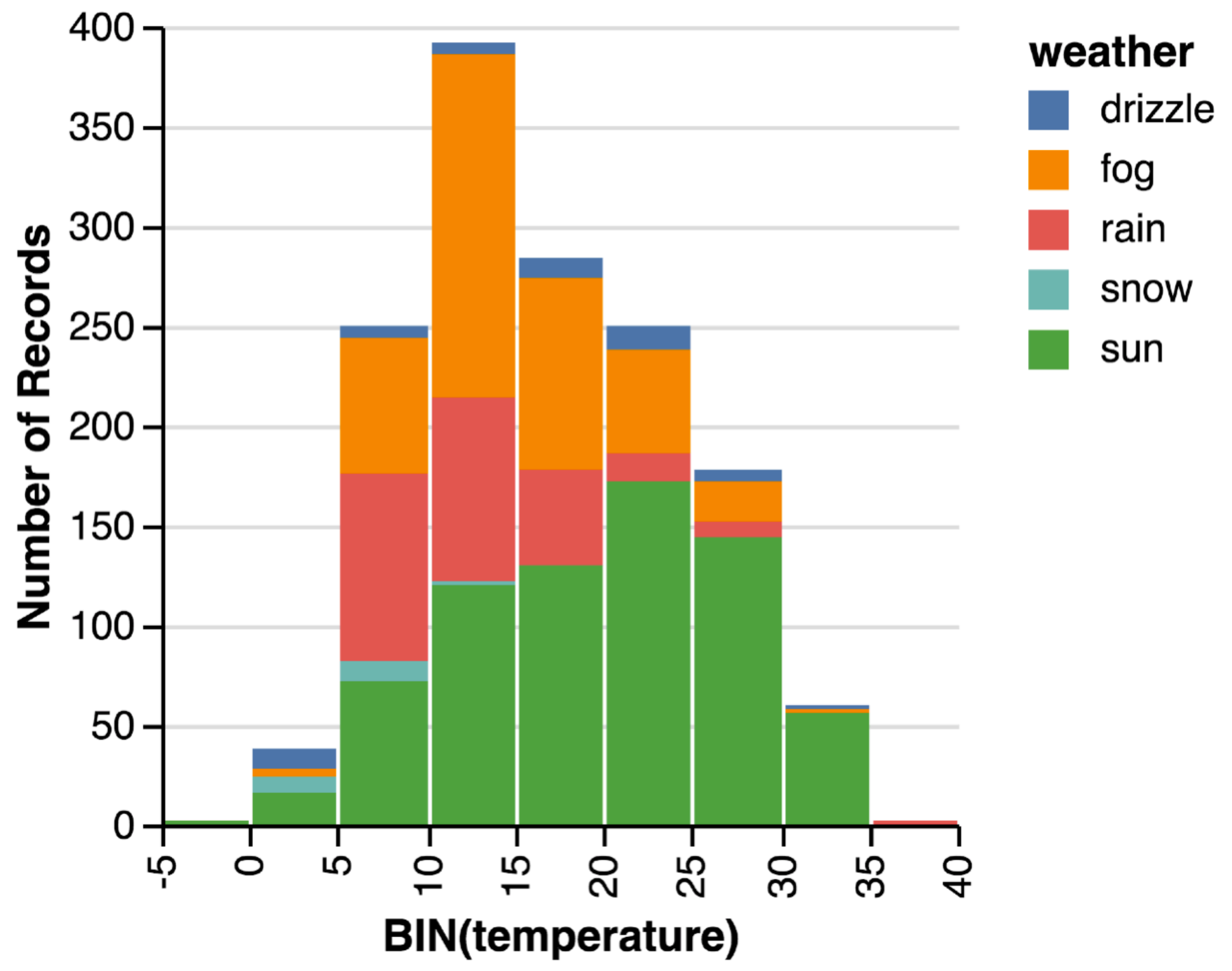
Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

https://doi.org/10.1109/TVCG.2016.2599030

Vega-lite

“Sensible defaults”

- The field chose reasonable defaults for presenting the data
  - We didn’t specify what colors to use
  - Or how wide bins should be
  - Or how to label the axes
  - Or that the bars should be stacked
  - …



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

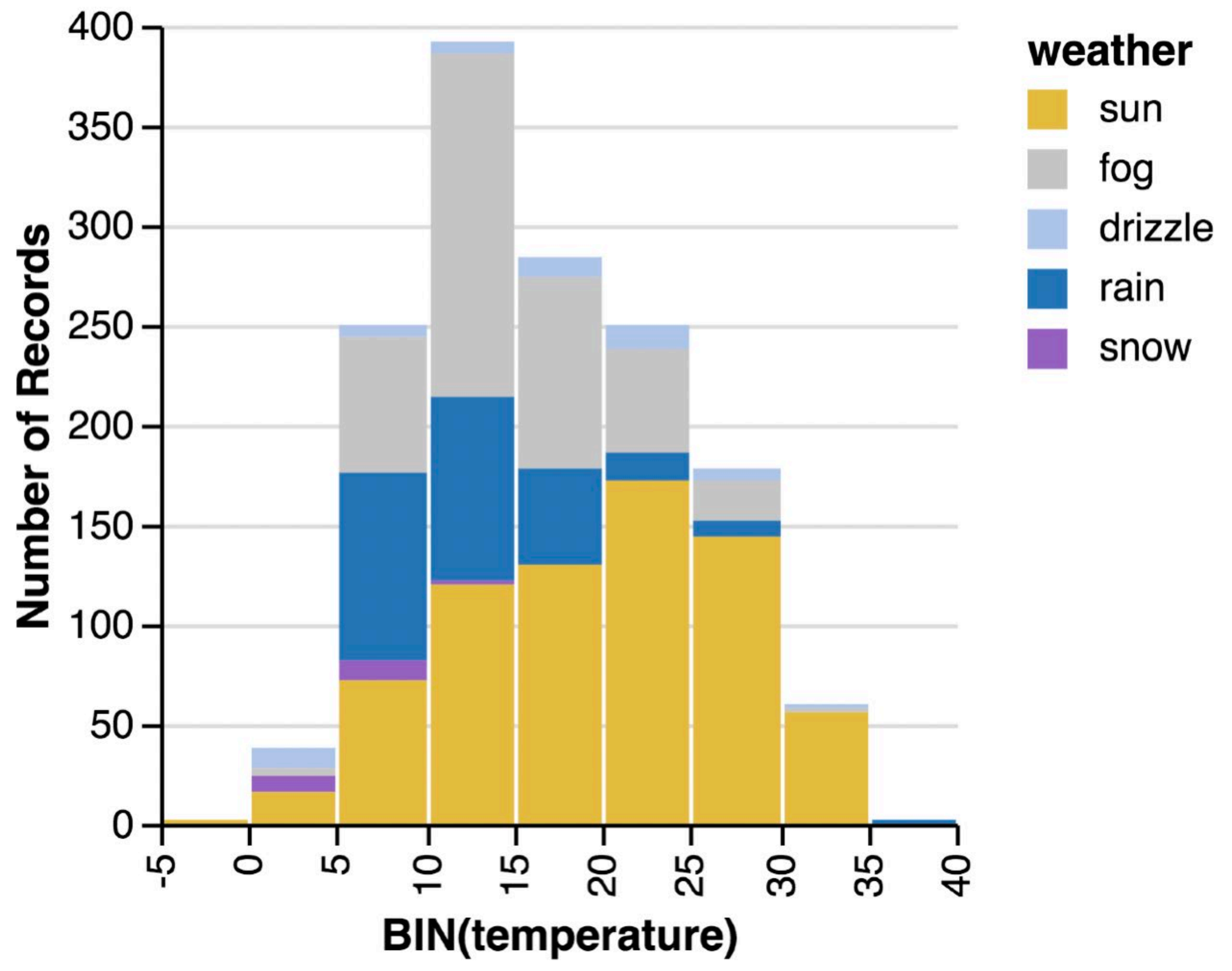
https://doi.org/10.1109/TVCG.2016.2599030

Vega-lite

Overriding the sensible defaults

```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    },
    color: {
     field: "weather",
     type: "nominal"
    },
    scale: {
        domain: ["sun","fog","drizzle", "rain","snow"],
        range: ["#e7ba52","#c7c7c7","#aec7e8",
             "#1f77b4","#9467bd"]
  }
}
```

Set our own color scale



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

https://doi.org/10.1109/TVCG.2016.2599030

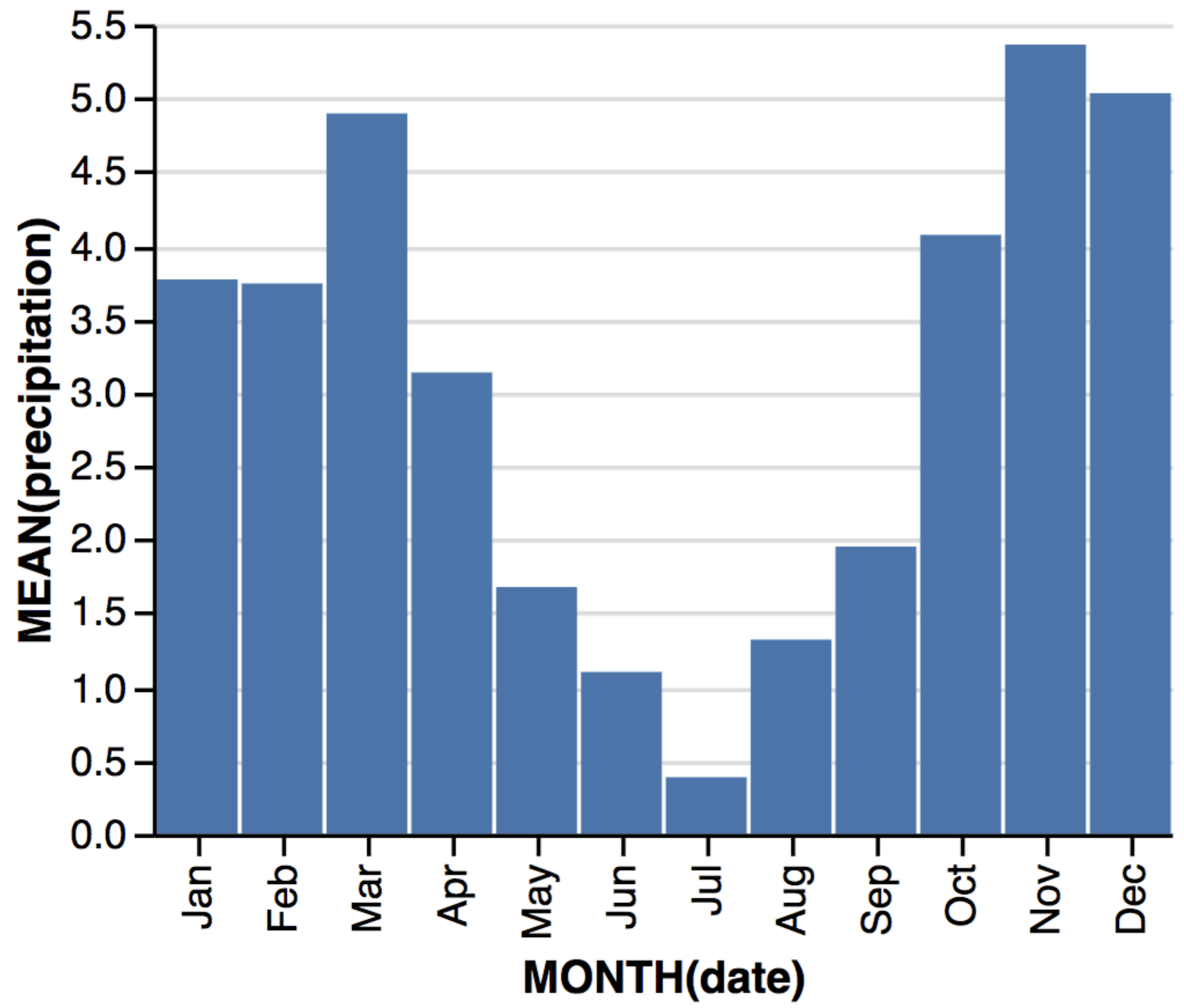
Vega-lite

Monthly precipitation

```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      timeUnit: "month", field: "date",
      type: "quantitative"
    },
    y: {
      aggregate: "mean",
      field: "precipitation",
      type: "quantitative"
    }
  }
}
```

Field is a date string (“2018-10-17”), but display and bin it as a month

Aggregate by the mean



Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. IEEE Vis, 2017. Vega-Lite: A Grammar of Interactive Graphics

https://doi.org/10.1109/TVCG.2016.2599030

Sensible defaults: Vega-lite's secret

- Threshold is lower
  - More concise definitions, less to understand up front
- Ceiling remains the same
  - The sensible defaults can be overridden
- A downside: visualizations made with Vega-Lite look similar
  - The path of least resistance Vega-Lite provides influences what visualizations people make and what they look like

Downside: path of least resistance

- The path of least resistance: tools influence what is created
- Sensible defaults make Vega-Lite visualizations look similar to one another
  - These defaults *can* be overwritten, but are they in practice?
- Similar concern to the widespread adoption of grid frameworks

Goals for this Lecture

By the end of this lecture, you should be able to…

- Describe the concepts of threshold and ceiling in software tools and what tool designers should be striving to create
- Explain the relative threshold and ceilings of visualization tools like Protovis, D3, and Vega-Lite
- Describe common visualization primitives like marks, axes, and scales
- Implement simple visualizations with Vega-Lite