DEPARTMENT OF COMPUTER SCIENCE

Manipulable Semantic Components: A Computational Representation of Data Visualization Scenes

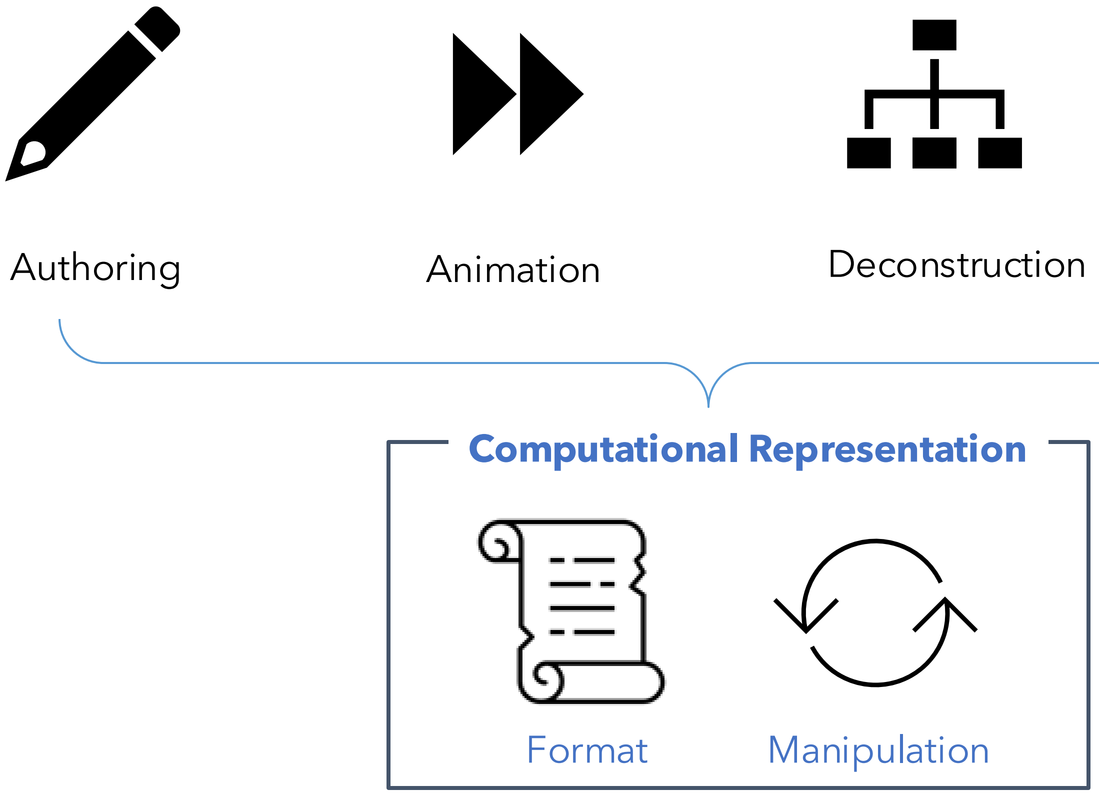
Zhicheng Liu

Chen Chen

John Hooker

Motivation

Many visualization tasks requires **a suitable vocabulary** that describes the semantic structure of visualization, i.e., representations, and how the visualization shall be manipulated, i.e., manipulations.



Authoring

Animation

Deconstruction

Navigation

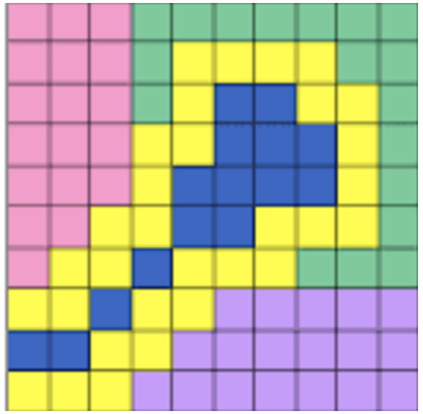
Computational Representation

Format

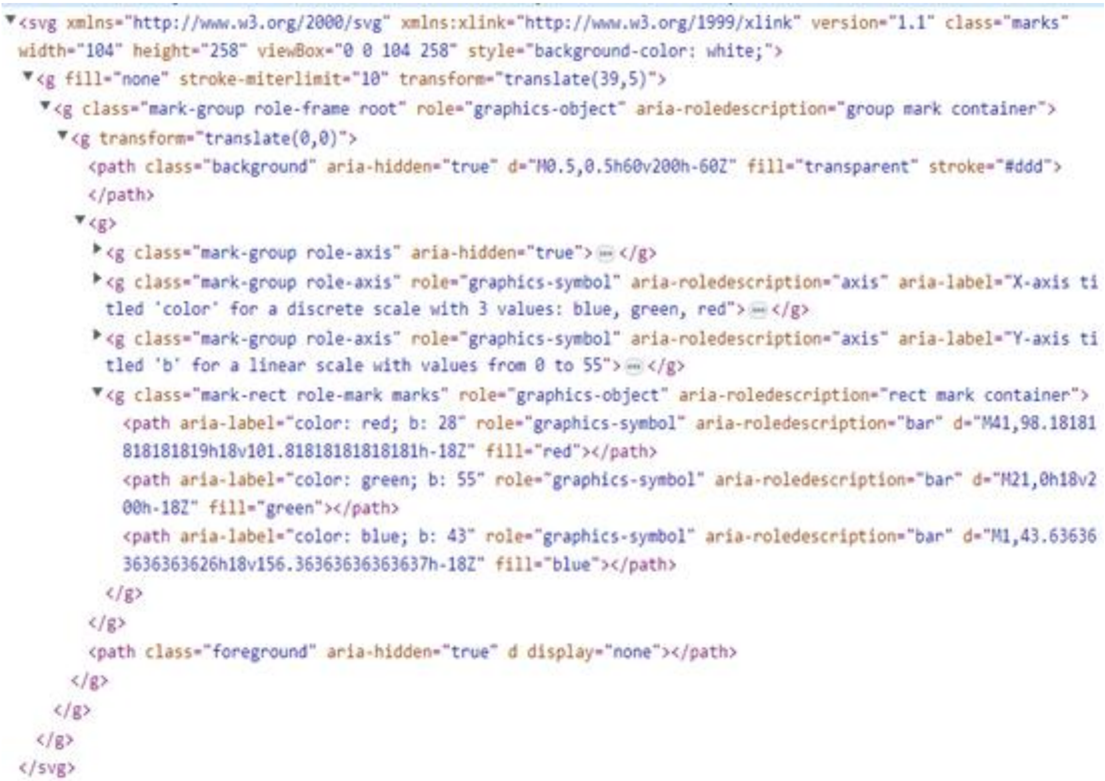
Manipulation

Existing Computational Representations for Data Visualization

Bitmap

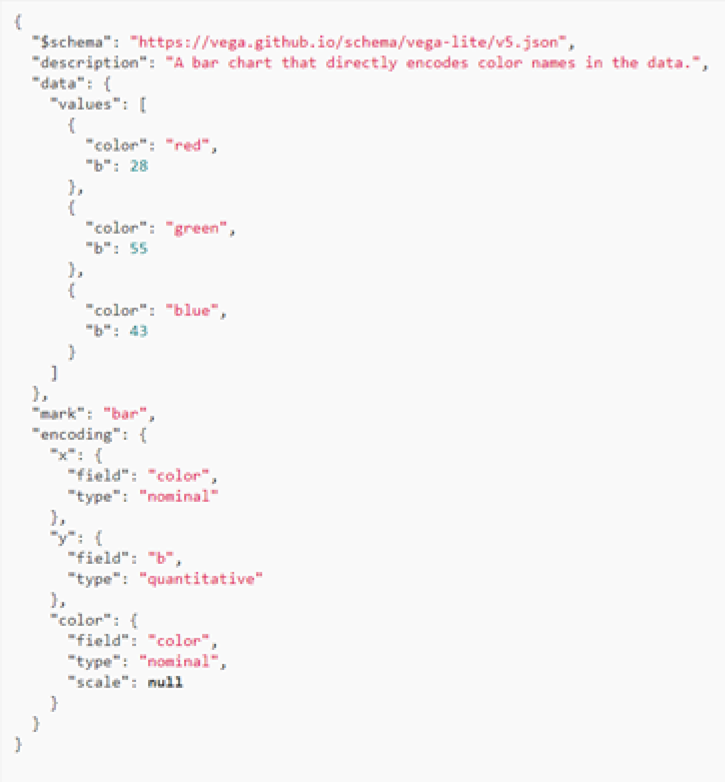


Vector Graphics



```
▼<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1" class="marks"
 width="104" height="258" viewBox="0 0 104 258" style="background-color: white;">
  ▼<g fill="none" stroke-miterlimit="10" transform="translate(39,5)">
    ▼<g class="mark-group role-frame root" role="graphics-object" aria-roledescription="group mark container">
      ▼<g transform="translate(0,0)">
         <path class="background" aria-hidden="true" d="M0.5,0.5h60v200h-60Z" fill="transparent" stroke="#ddd">
         </path>
        ▼<g>
          ▶<g class="mark-group role-axis" aria-hidden="true">⋯</g>
          ▶<g class="mark-group role-axis" role="graphics-symbol" aria-roledescription="axis" aria-label="X-axis ti
           tled 'color' for a discrete scale with 3 values: blue, green, red">⋯</g>
          ▶<g class="mark-group role-axis" role="graphics-symbol" aria-roledescription="axis" aria-label="Y-axis ti
           tled 'b' for a linear scale with values from 0 to 55">⋯</g>
          ▼<g class="mark-rect role-mark marks" role="graphics-object" aria-roledescription="rect mark container">
             <path aria-label="color: red; b: 28" role="graphics-symbol" aria-roledescription="bar" d="M41,98.18181
             818181819h18v101.81818181818181h-18Z" fill="red"></path>
             <path aria-label="color: green; b: 55" role="graphics-symbol" aria-roledescription="bar" d="M21,0h18v2
             00h-18Z" fill="green"></path>
             <path aria-label="color: blue; b: 43" role="graphics-symbol" aria-roledescription="bar" d="M1,43.63636
             3636363626h18v156.36363636363637h-18Z" fill="blue"></path>
           </g>
         </g>
         <path class="foreground" aria-hidden="true" d display="none"></path>
       </g>
     </g>
   </g>
 </svg>
```

Program



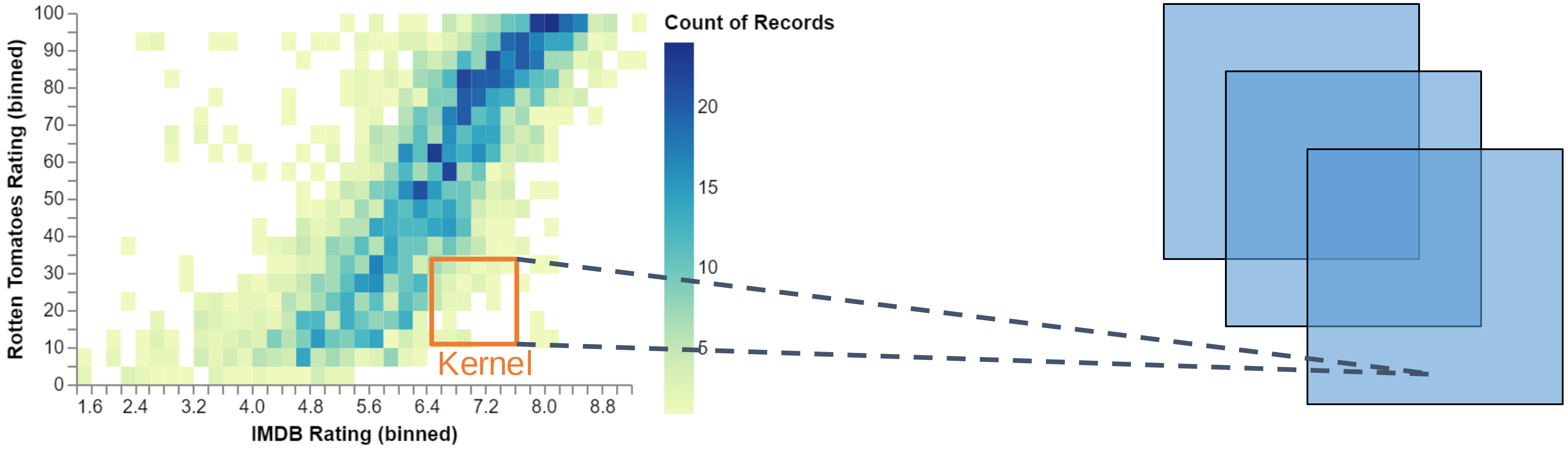
```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "description": "A bar chart that directly encodes color names in the data.",
  "data": {
    "values": [
      {
        "color": "red",
        "b": 28
      },
      {
        "color": "green",
        "b": 55
      },
      {
        "color": "blue",
        "b": 43
      }
    ]
  },
  "mark": "bar",
  "encoding": {
    "x": {
      "field": "color",
      "type": "nominal"
    },
    "y": {
      "field": "b",
      "type": "quantitative"
    },
    "color": {
      "field": "color",
      "type": "nominal",
      "scale": null
    }
  }
}
```

Motivation

Bitmap representation + modern CNN architectures

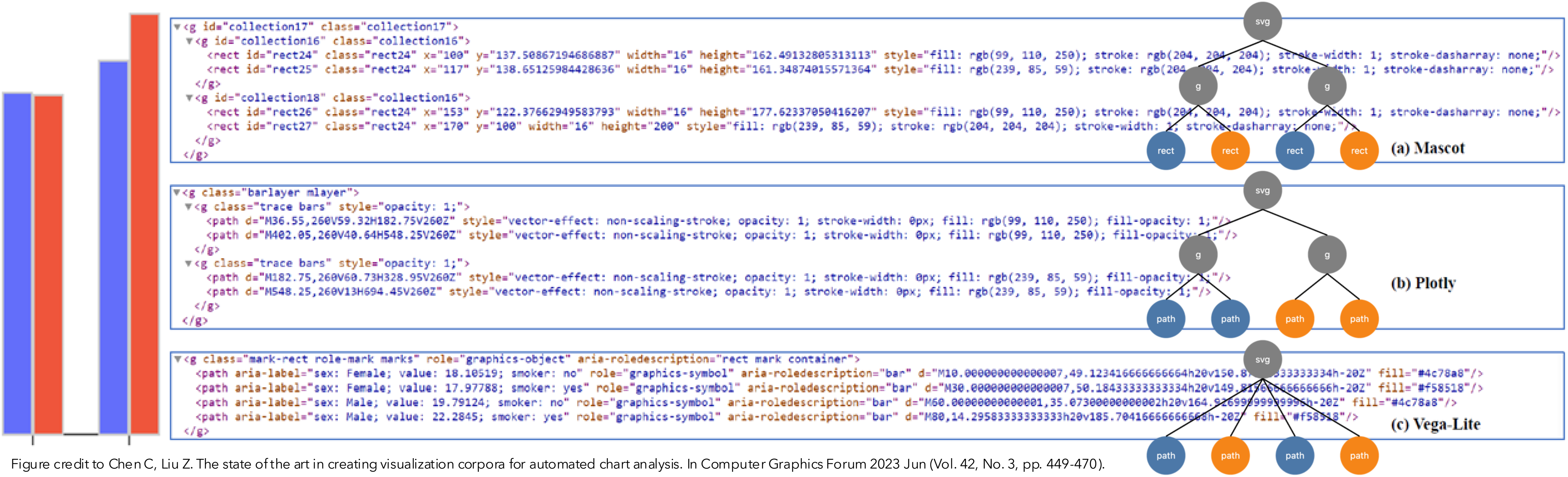
Format: Bitmap

Method: Convolution



Motivation

SVG: Semantic information such as element type & grouping unreliable



```
▼<g id="collection17" class="collection17">
  ▼<g id="collection16" class="collection16">
      <rect id="rect24" class="rect24" x="100" y="137.50867194686887" width="16" height="162.49132805313113" style="fill: rgb(99, 110, 250); stroke: rgb(204, 204, 204); stroke-width: 1; stroke-dasharray: none;"/>
      <rect id="rect25" class="rect24" x="117" y="138.65125984428636" width="16" height="161.34874015571364" style="fill: rgb(239, 85, 59); stroke: rgb(204, 204, 204); stroke-width: 1; stroke-dasharray: none;"/>
    </g>
  ▼<g id="collection18" class="collection16">
      <rect id="rect26" class="rect24" x="153" y="122.37662949583793" width="16" height="177.62337050416207" style="fill: rgb(99, 110, 250); stroke: rgb(204, 204, 204); stroke-width: 1; stroke-dasharray: none;"/>
      <rect id="rect27" class="rect24" x="170" y="100" width="16" height="200" style="fill: rgb(239, 85, 59); stroke: rgb(204, 204, 204); stroke-width: 1; stroke-dasharray: none;"/>
    </g>
 </g>
```

(a) Mascot

```
▼<g class="barlayer mlayer">
  ▼<g class="trace bars" style="opacity: 1;">
      <path d="M36.55,260V59.32H182.75V260Z" style="vector-effect: non-scaling-stroke; opacity: 1; stroke-width: 0px; fill: rgb(99, 110, 250); fill-opacity: 1;"/>
      <path d="M402.05,260V40.64H548.25V260Z" style="vector-effect: non-scaling-stroke; opacity: 1; stroke-width: 0px; fill: rgb(99, 110, 250); fill-opacity: 1;"/>
    </g>
  ▼<g class="trace bars" style="opacity: 1;">
      <path d="M182.75,260V60.73H328.95V260Z" style="vector-effect: non-scaling-stroke; opacity: 1; stroke-width: 0px; fill: rgb(239, 85, 59); fill-opacity: 1;"/>
      <path d="M548.25,260V13H694.45V260Z" style="vector-effect: non-scaling-stroke; opacity: 1; stroke-width: 0px; fill: rgb(239, 85, 59); fill-opacity: 1;"/>
    </g>
 </g>
```

(b) Plotly

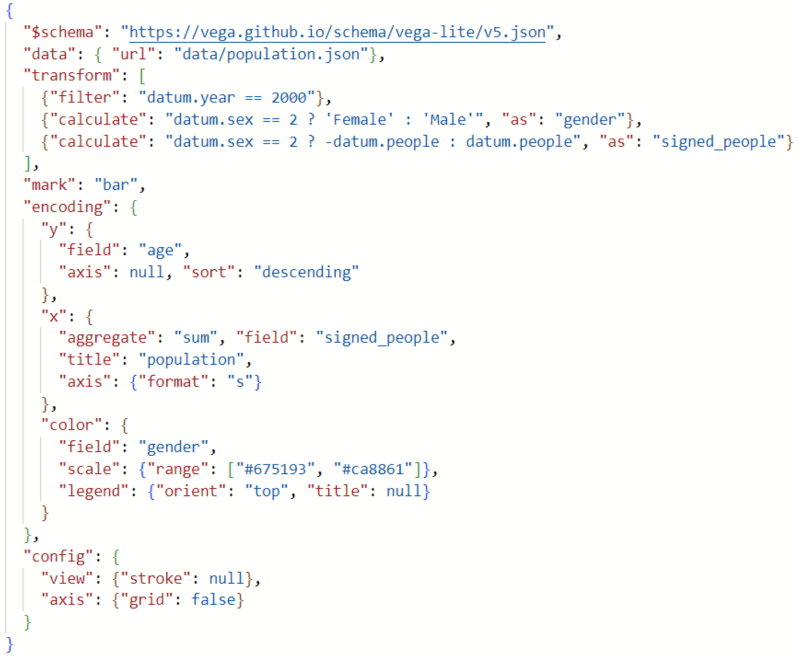
```
▼<g class="mark-rect role-mark marks" role="graphics-object" aria-roledescription="rect mark container">
    <path aria-label="sex: Female; value: 18.10519; smoker: no" role="graphics-symbol" aria-roledescription="bar" d="M10.000000000000007,49.123416666666664h20v150.8[illegible]33333334h-20Z" fill="#4c78a8"/>
    <path aria-label="sex: Female; value: 17.97788; smoker: yes" role="graphics-symbol" aria-roledescription="bar" d="M30.000000000000007,50.184333333333334h20v149.81566666666666h-20Z" fill="#f58518"/>
    <path aria-label="sex: Male; value: 19.79124; smoker: no" role="graphics-symbol" aria-roledescription="bar" d="M60.00000000000001,35.07300000000002h20v164.92699999999996h-20Z" fill="#4c78a8"/>
    <path aria-label="sex: Male; value: 22.2845; smoker: yes" role="graphics-symbol" aria-roledescription="bar" d="M80,14.295833333333333h20v185.70416666666668h-20Z" fill="#f58518"/>
 </g>
```

(c) Vega-Lite

Figure credit to Chen C, Liu Z. The state of the art in creating visualization corpora for automated chart analysis. In Computer Graphics Forum 2023 Jun (Vol. 42, No. 3, pp. 449-470).

Motivation

Program

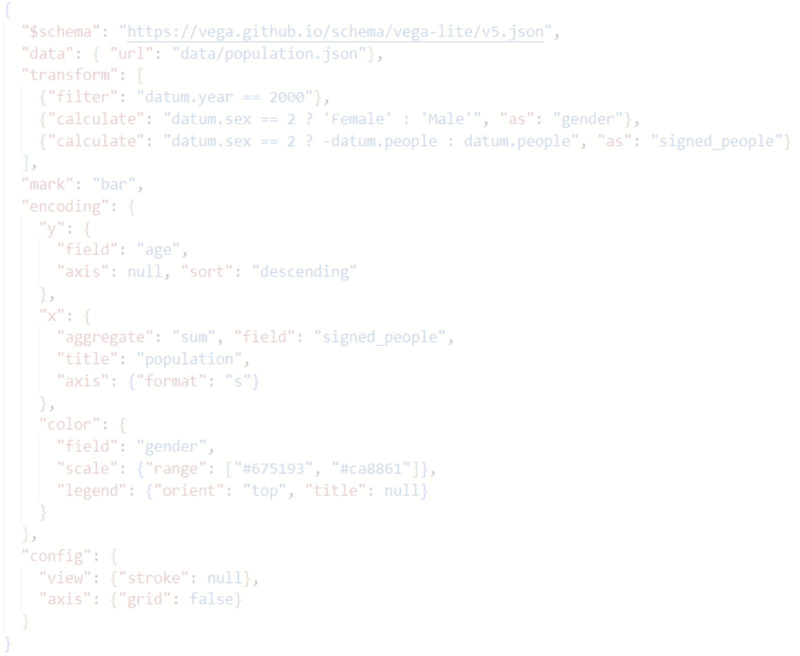


```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "data": { "url": "data/population.json"},
  "transform": [
    {"filter": "datum.year == 2000"},
    {"calculate": "datum.sex == 2 ? 'Female' : 'Male'", "as": "gender"},
    {"calculate": "datum.sex == 2 ? -datum.people : datum.people", "as": "signed_people"}
  ],
  "mark": "bar",
  "encoding": {
    "y": {
      "field": "age",
      "axis": null, "sort": "descending"
    },
    "x": {
      "aggregate": "sum", "field": "signed_people",
      "title": "population",
      "axis": {"format": "s"}
    },
    "color": {
      "field": "gender",
      "scale": {"range": ["#675193", "#ca8861"]},
      "legend": {"orient": "top", "title": null}
    }
  },
  "config": {
    "view": {"stroke": null},
    "axis": {"grid": false}
  }
}
```

Declarative languages hide the details of the semantic structure

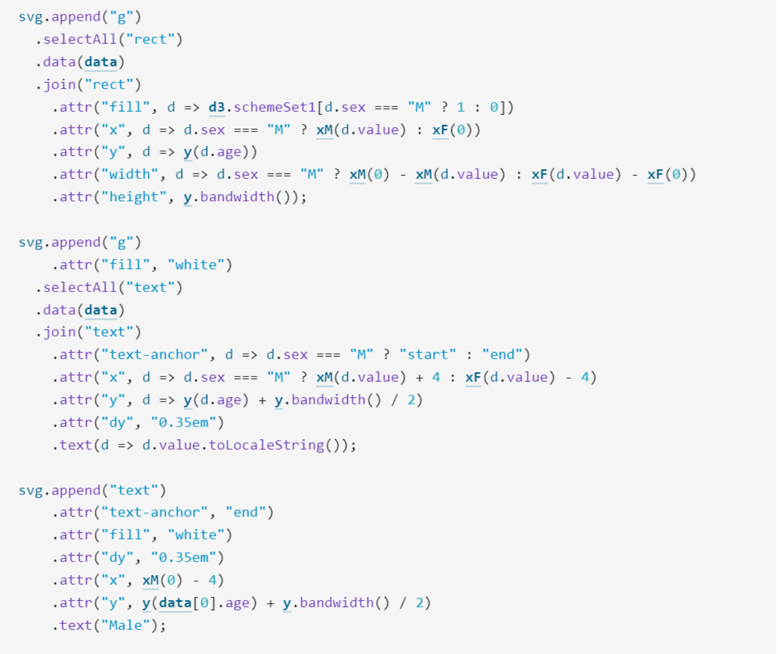
Motivation

Program



```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "data": { "url": "data/population.json"},
  "transform": [
    {"filter": "datum.year == 2000"},
    {"calculate": "datum.sex == 2 ? 'Female' : 'Male'", "as": "gender"},
    {"calculate": "datum.sex == 2 ? -datum.people : datum.people", "as": "signed_people"}
  ],
  "mark": "bar",
  "encoding": {
    "y": {
      "field": "age",
      "axis": null, "sort": "descending"
    },
    "x": {
      "aggregate": "sum", "field": "signed_people",
      "title": "population",
      "axis": {"format": "s"}
    },
    "color": {
      "field": "gender",
      "scale": {"range": ["#675193", "#ca8861"]},
      "legend": {"orient": "top", "title": null}
    }
  },
  "config": {
    "view": {"stroke": null},
    "axis": {"grid": false}
  }
}
```

Declarative languages hide the details of the semantic structure

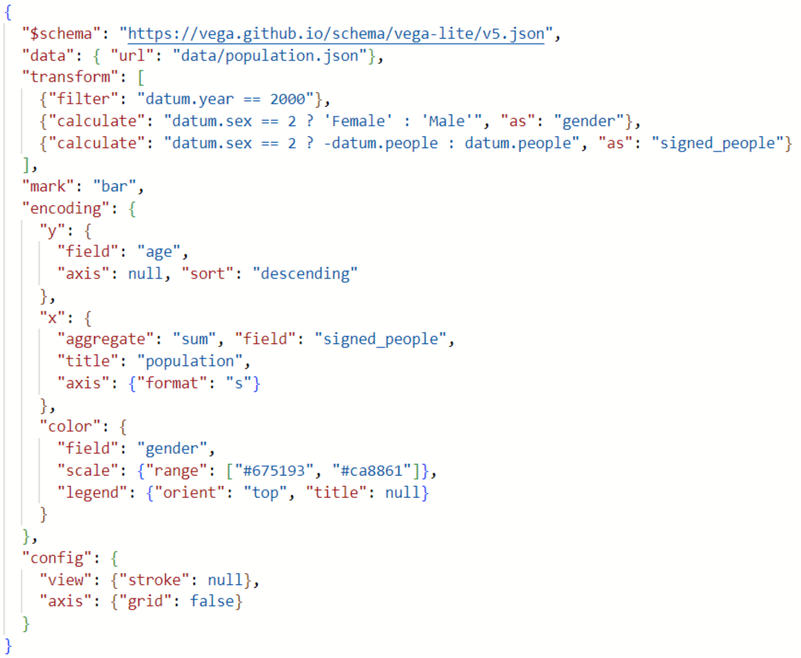


```
svg.append("g")
  .selectAll("rect")
  .data(data)
  .join("rect")
    .attr("fill", d => d3.schemeSet1[d.sex === "M" ? 1 : 0])
    .attr("x", d => d.sex === "M" ? xM(d.value) : xF(0))
    .attr("y", d => y(d.age))
    .attr("width", d => d.sex === "M" ? xM(0) - xM(d.value) : xF(d.value) - xF(0))
    .attr("height", y.bandwidth());

svg.append("g")
    .attr("fill", "white")
  .selectAll("text")
  .data(data)
  .join("text")
    .attr("text-anchor", d => d.sex === "M" ? "start" : "end")
    .attr("x", d => d.sex === "M" ? xM(d.value) + 4 : xF(d.value) - 4)
    .attr("y", d => y(d.age) + y.bandwidth() / 2)
    .attr("dy", "0.35em")
    .text(d => d.value.toLocaleString());

svg.append("text")
    .attr("text-anchor", "end")
    .attr("fill", "white")
    .attr("dy", "0.35em")
    .attr("x", xM(0) - 4)
    .attr("y", y(data[0].age) + y.bandwidth() / 2)
    .text("Male");
```

Scene assembly languages lack high-level semantic abstractions

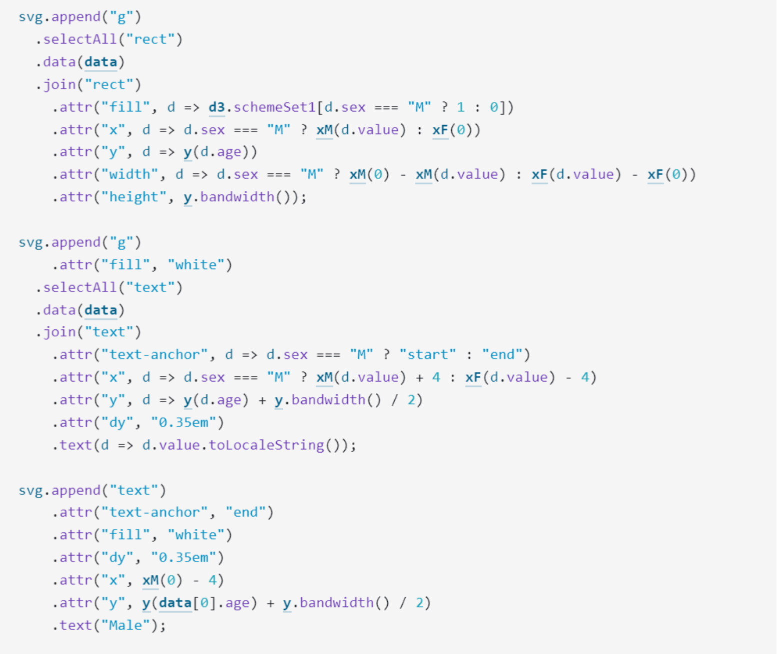
Motivation

Program: difficult to generalize to diverse libraries and languages



```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "data": { "url": "data/population.json"},
  "transform": [
    {"filter": "datum.year == 2000"},
    {"calculate": "datum.sex == 2 ? 'Female' : 'Male'", "as": "gender"},
    {"calculate": "datum.sex == 2 ? -datum.people : datum.people", "as": "signed_people"}
  ],
  "mark": "bar",
  "encoding": {
    "y": {
      "field": "age",
      "axis": null, "sort": "descending"
    },
    "x": {
      "aggregate": "sum", "field": "signed_people",
      "title": "population",
      "axis": {"format": "s"}
    },
    "color": {
      "field": "gender",
      "scale": {"range": ["#675193", "#ca8861"]},
      "legend": {"orient": "top", "title": null}
    }
  },
  "config": {
    "view": {"stroke": null},
    "axis": {"grid": false}
  }
}
```

Declarative languages hide the details of the semantic structure



```
svg.append("g")
  .selectAll("rect")
  .data(data)
  .join("rect")
    .attr("fill", d => d3.schemeSet1[d.sex === "M" ? 1 : 0])
    .attr("x", d => d.sex === "M" ? xM(d.value) : xF(0))
    .attr("y", d => y(d.age))
    .attr("width", d => d.sex === "M" ? xM(0) - xM(d.value) : xF(d.value) - xF(0))
    .attr("height", y.bandwidth());

svg.append("g")
    .attr("fill", "white")
  .selectAll("text")
  .data(data)
  .join("text")
    .attr("text-anchor", d => d.sex === "M" ? "start" : "end")
    .attr("x", d => d.sex === "M" ? xM(d.value) + 4 : xF(d.value) - 4)
    .attr("y", d => y(d.age) + y.bandwidth() / 2)
    .attr("dy", "0.35em")
    .text(d => d.value.toLocaleString());

svg.append("text")
    .attr("text-anchor", "end")
    .attr("fill", "white")
    .attr("dy", "0.35em")
    .attr("x", xM(0) - 4)
    .attr("y", y(data[0].age) + y.bandwidth() / 2)
    .text("Male");
```

Scene assembly languages lack high-level semantic abstractions

Motivation

Researchers have been proposing new computation representations…

But they are mostly task-orientated, **limiting the generalizability**…

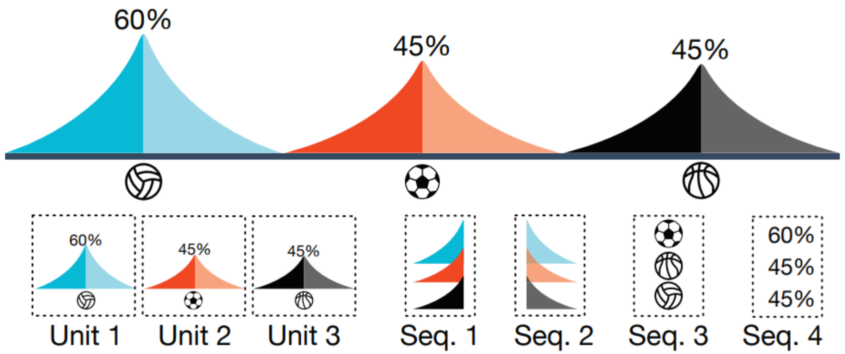


Figure 6. Example of unit and sequence detection. From the top example, Three units and four sequences are identified. Please note that the horizontal line is a chart-level embellishment and excluded.

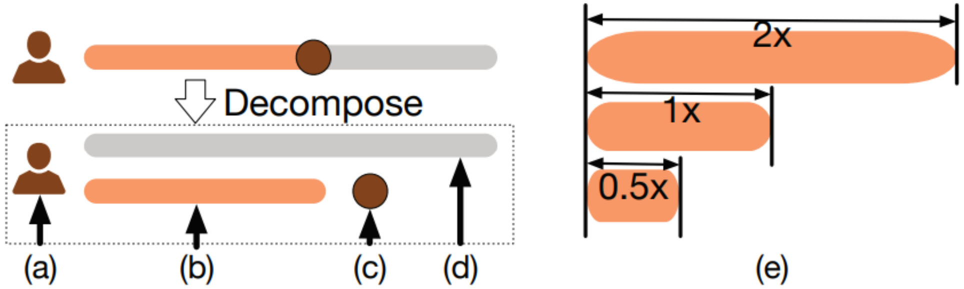


Figure 5. Update types for different elements. (a) *Fix*: invariant to underlying value (but needs to be replaced if the data semantic is changed). (b) *Morph*: need to morph if the underlying value is changed. (c) *Move*: need to move if the underlying value is changed. (d) *Fix*: invariant to underlying data. (e) A rounded rectangle scaled by different factors.

ChartReuse, TVCG 2021

Motivation

Researchers have been proposing new computation representations…

While they are mostly task-orientated, **limiting the generalizability**…

A unified and expressive model of data visualization scenes for a variety of applications has been missing…

We propose a new computational representation named Manipulable Semantic Components (MSC) to support scene understanding and augmentation.

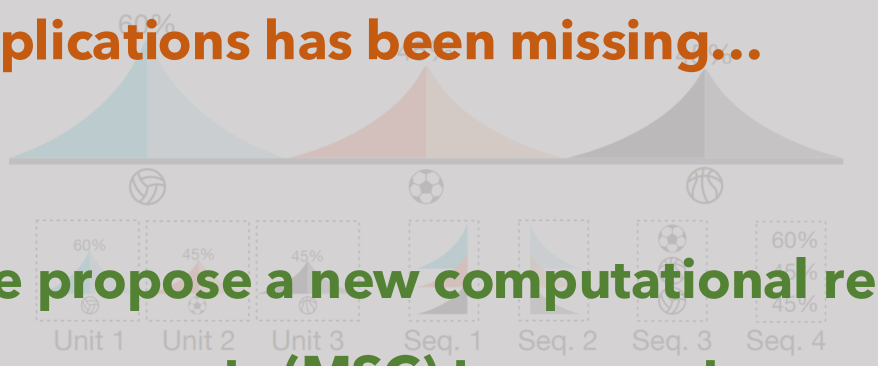


Figure 6. Example of unit and sequence detection. From the top example, Three units and four sequences are identified. Please note that the horizontal line is a chart-level embellishment and excluded.

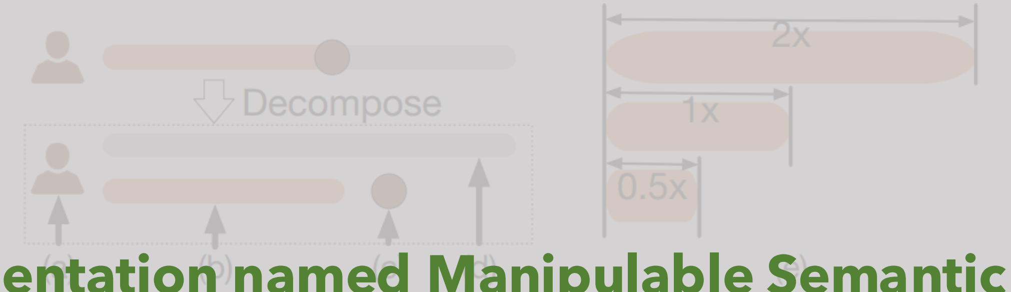


Figure 5. Update types for different elements. (a) *Fix*: invariant to under-[illegible]tic is changed). (b) *Morph*: need to morph if the underlying value is changed. (c)*Move*: need to move if the underlying value is changed. (d) *Fix*: invariant to underlying data. (e) A rounded rectangle scaled by different factors.

ChartReuse, TVCG 2021

Manipulable Semantic Components

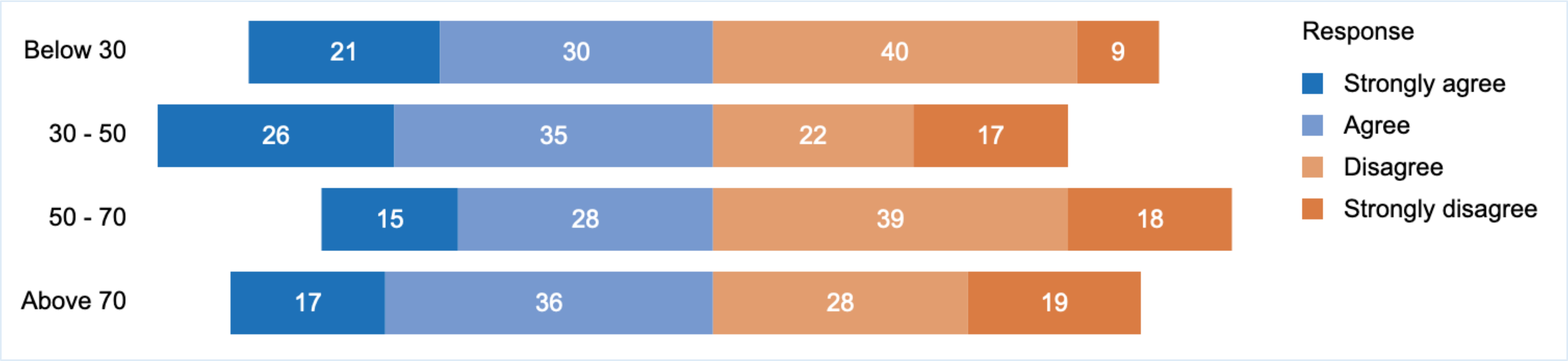
Overview

Manipulable Semantic Components (MSC) is a computational representation of data visualization scenes, to support applications in scene understanding and augmentation.

- MSC is the result of a continuous effort since Fall 2020, led by Professor Zhicheng Liu.
- Taking a **graphics-centric** approach and focusing on how graphical objects can be created, modified and joined with data to generate visualizations
- MSC contains (1) a unified object model describing the visualization scene structure in terms of semantic components and (2) an operation set for modifying the scene components.

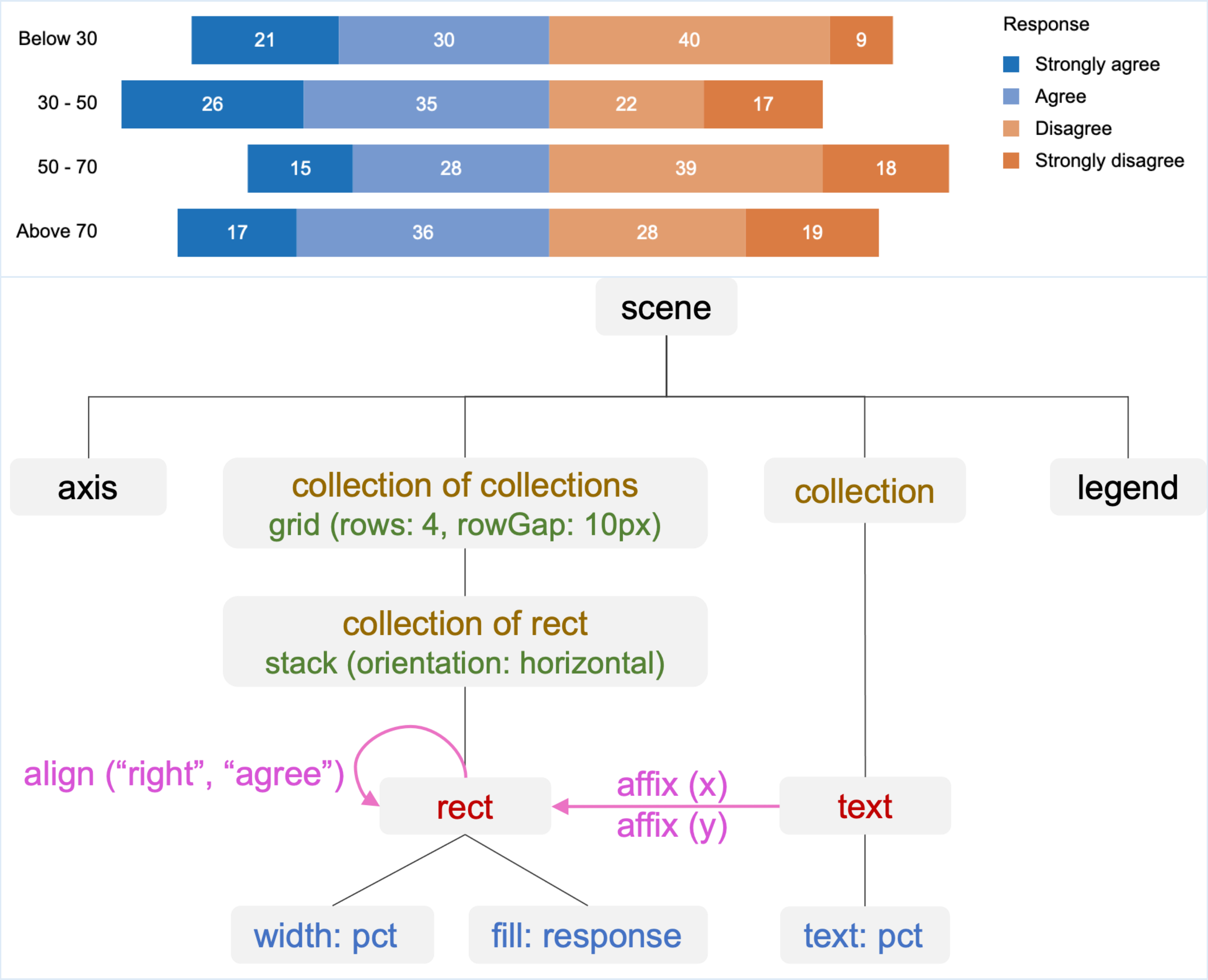
An Example Illustration

Stacked bar chart



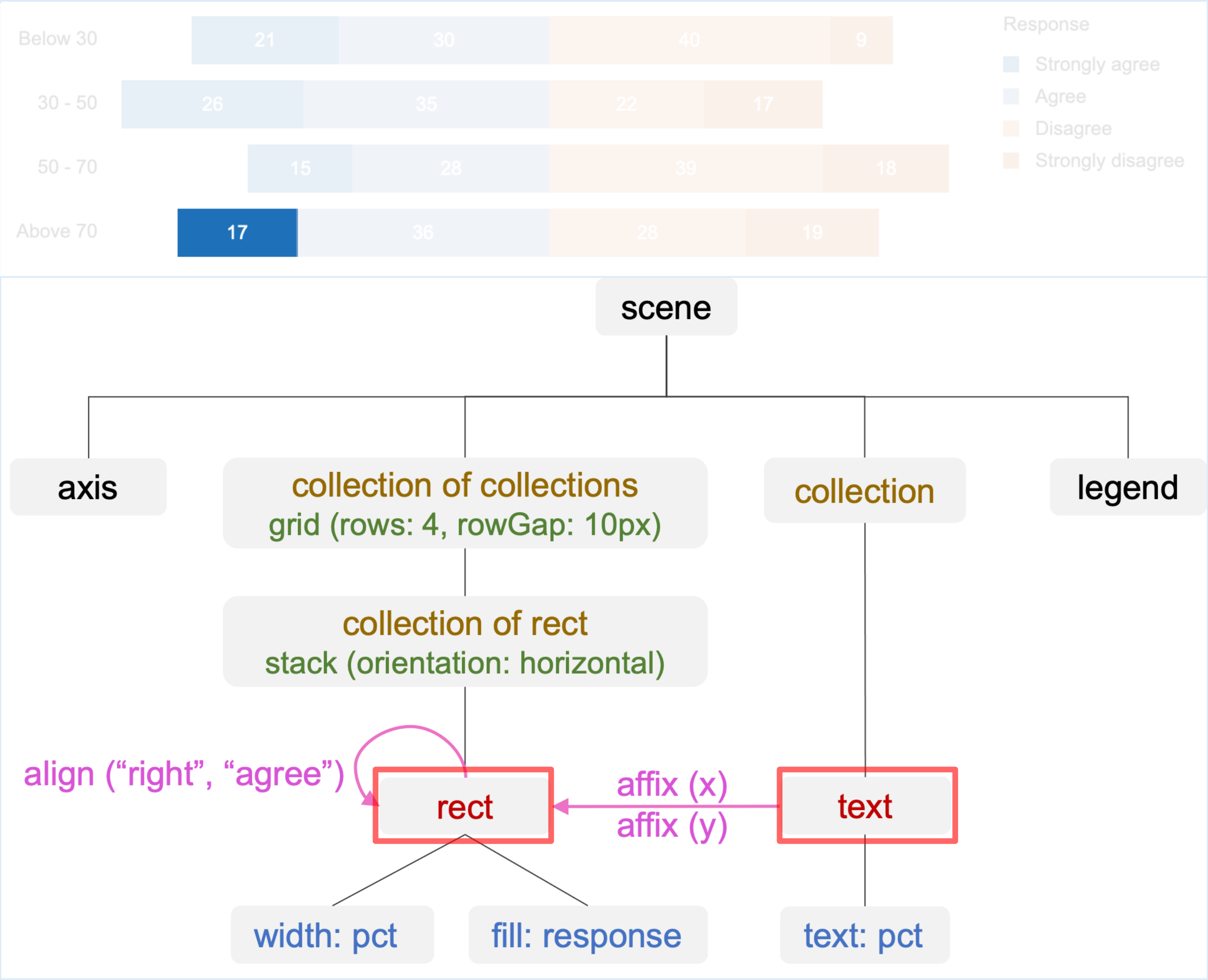
Scene Structure

Overview



Scene Structure: Semantic Components

Marks



scene

axis

collection of collections
grid (rows: 4, rowGap: 10px)

collection

legend

collection of rect
stack (orientation: horizontal)

align (“right”, “agree”)

rect

affix (x)
affix (y)

text

width: pct

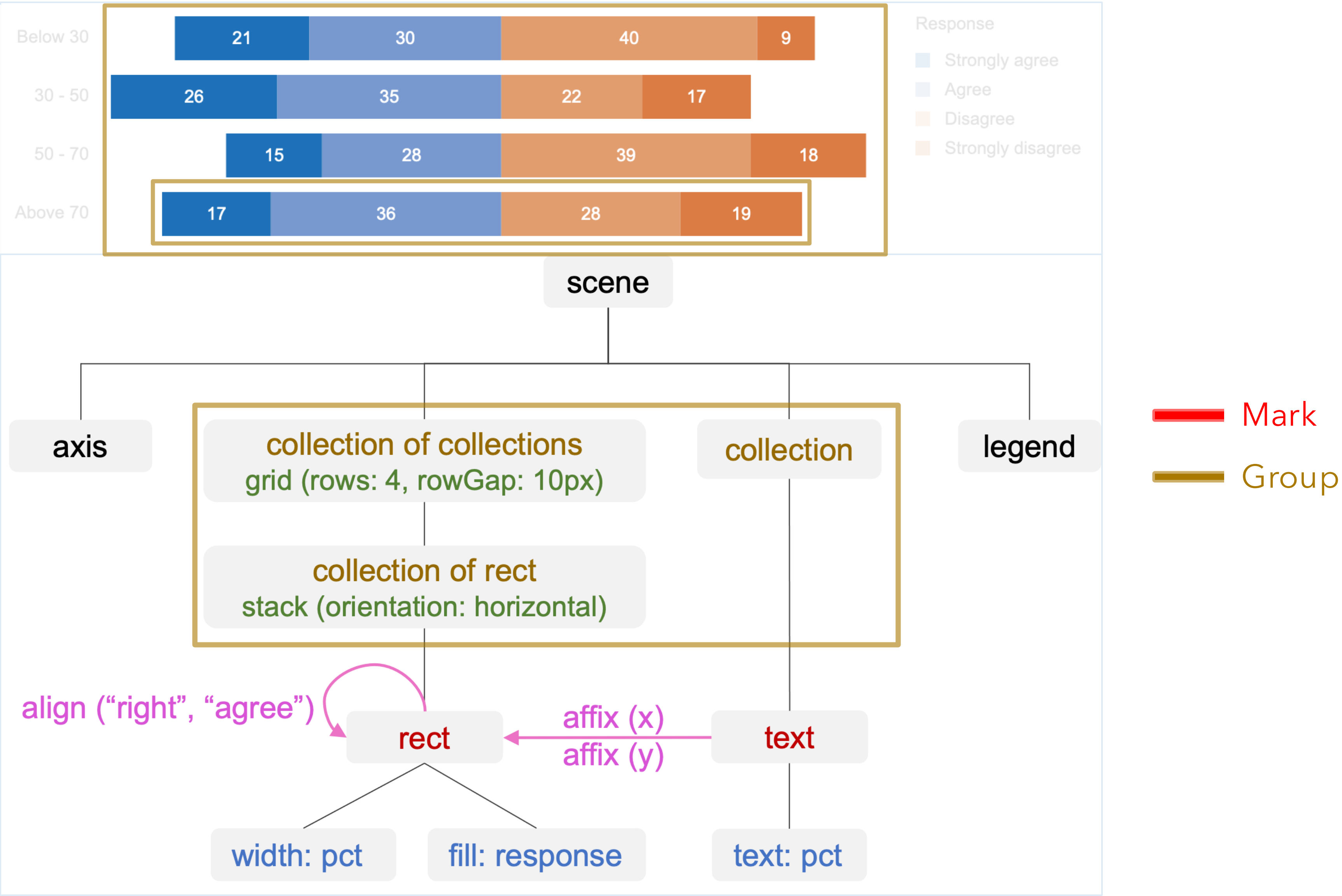
fill: response

text: pct

🟥 Mark

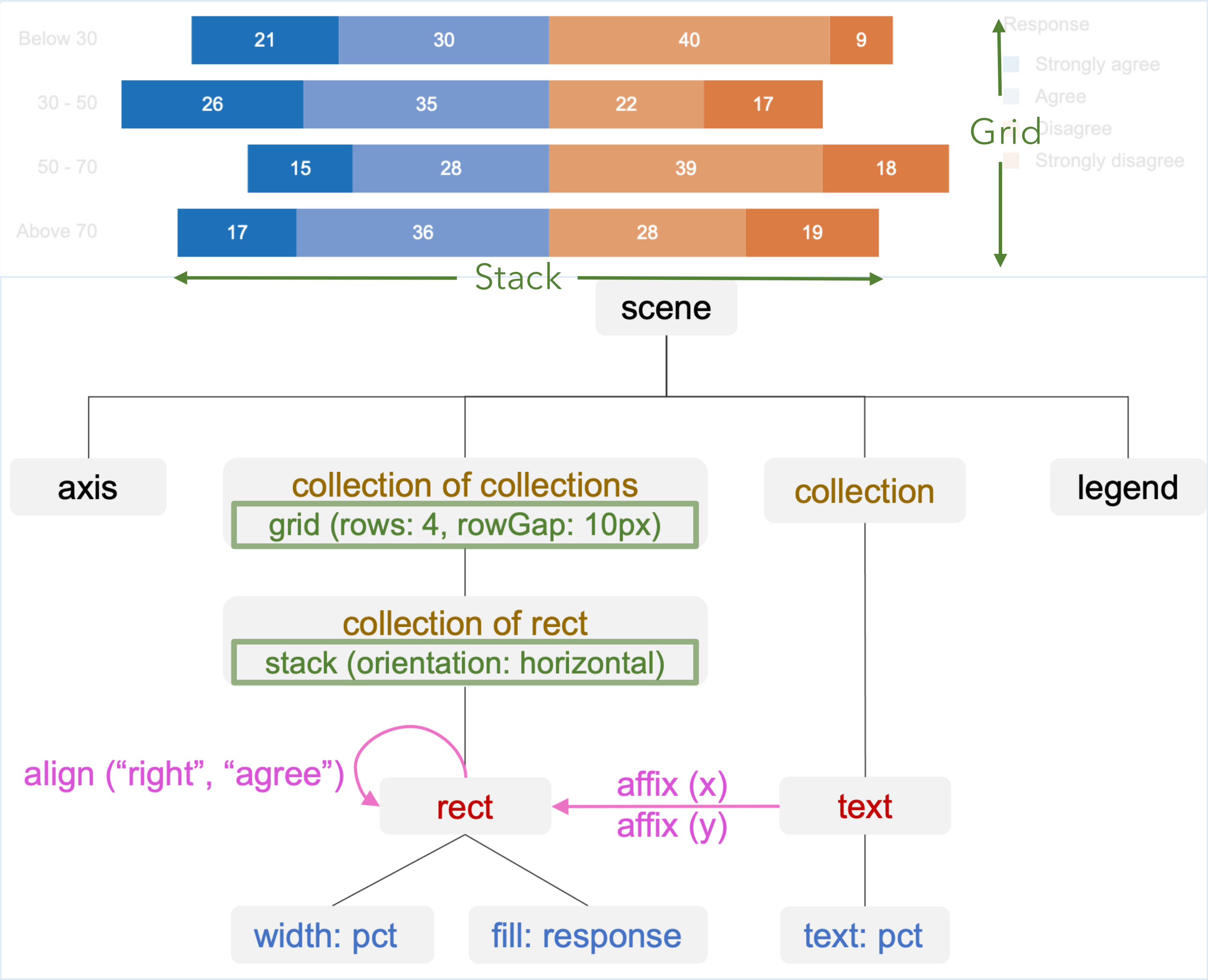
Scene Structure: Semantic Components

Groups



Scene Structure: Semantic Components

Layouts



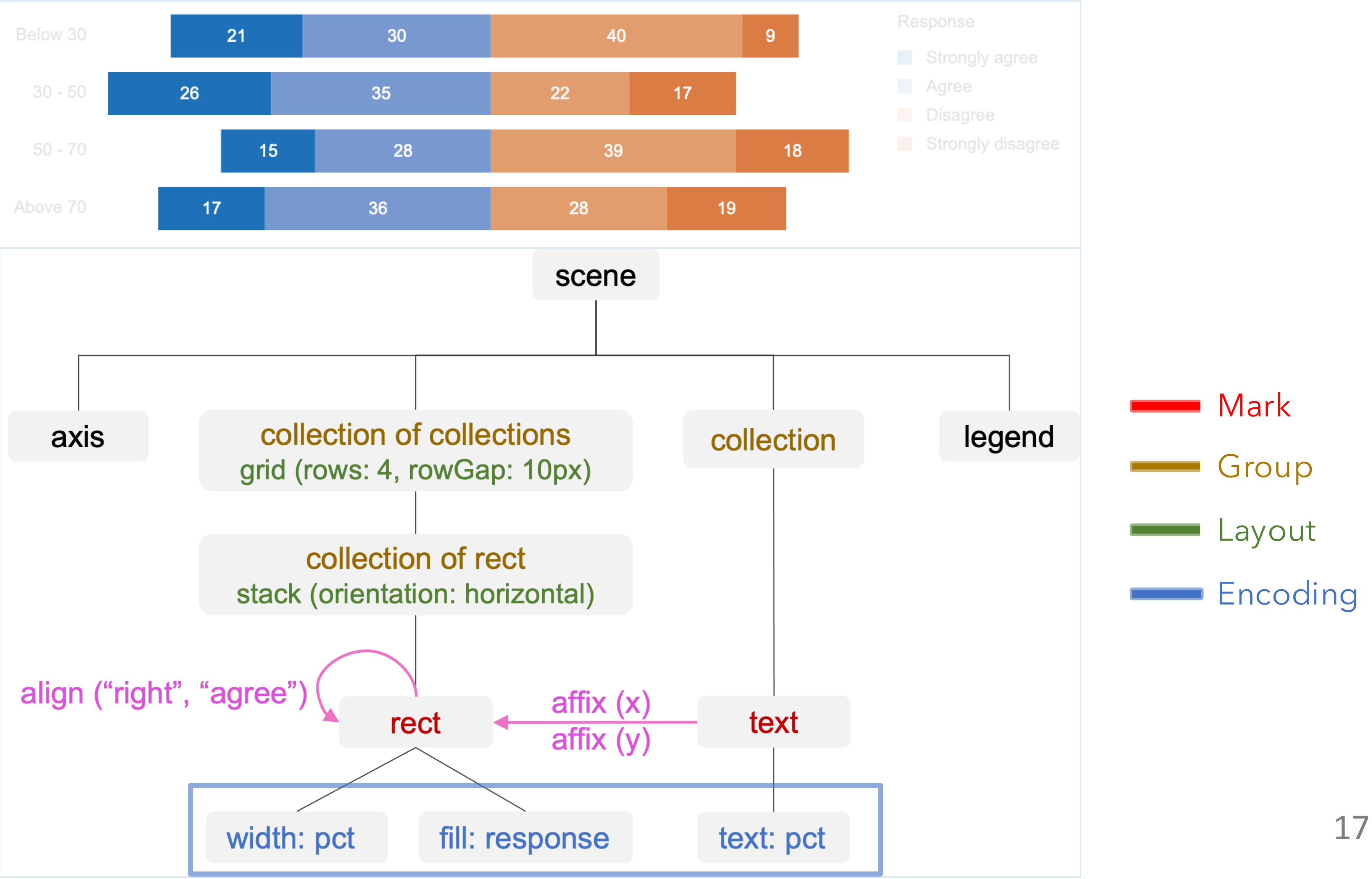
🟥 Mark

🟨 Group

🟩 Layout

Scene Structure: Semantic Components

Encodings



scene

axis

collection of collections
grid (rows: 4, rowGap: 10px)

collection of rect
stack (orientation: horizontal)

collection

legend

align (“right”, “agree”)

rect

affix (x)
affix (y)

text

width: pct

fill: response

text: pct

Mark

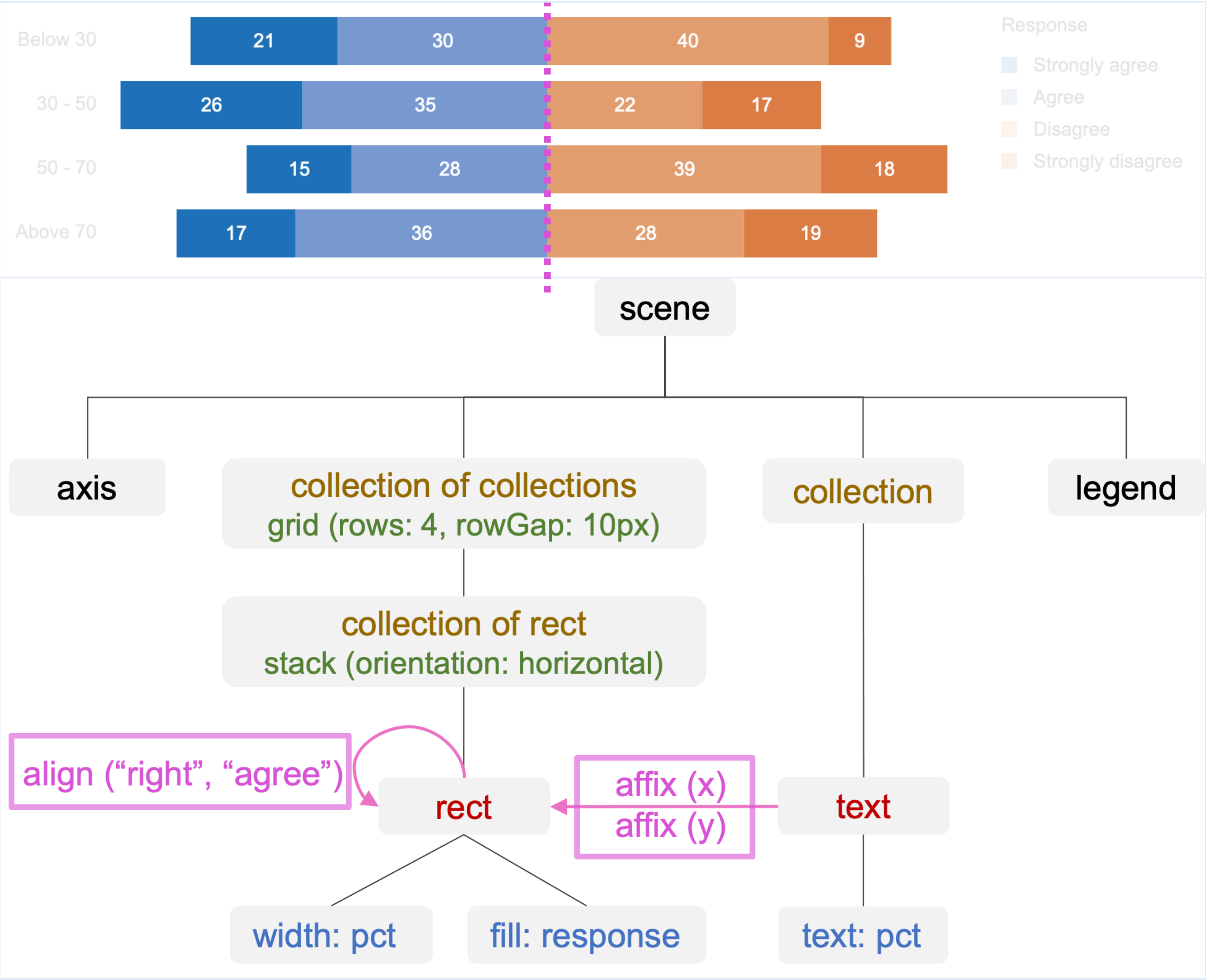
Group

Layout

Encoding

Scene Structure: Semantic Components

Constraints



🟥 Mark

🟨 Group

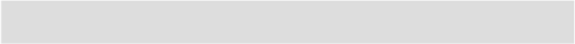
🟩 Layout

🟦 Encoding

🟪 Constraint

Scene Manipulation: Operations

Create elements

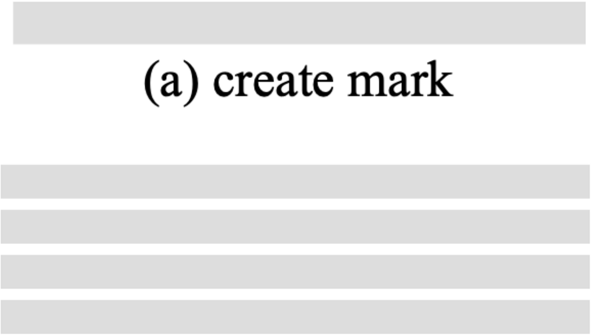


(a) create mark

(a) **create** a rectangle mark;

Scene Manipulation: Operations

Repeat elements with data



(a) create mark

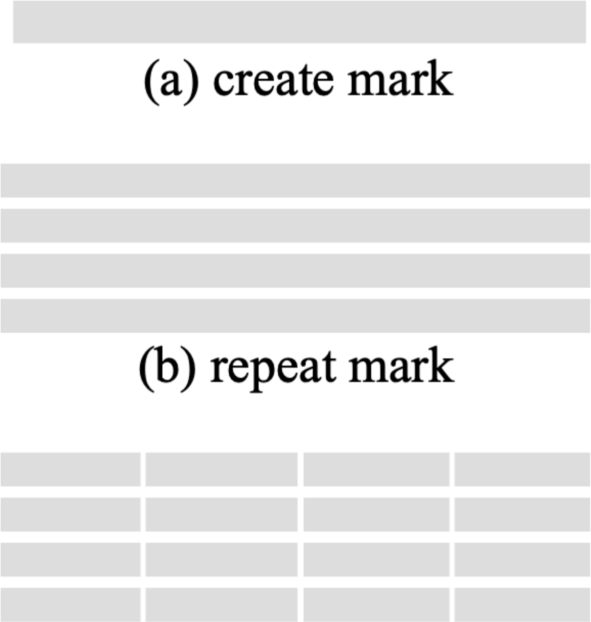
(b) repeat mark

(a) **create** a rectangle mark;

(b) **repeat** the rectangle by age;

Scene Manipulation: Operations

Divide elements with data



(a) create mark

(b) repeat mark

(c) divide mark

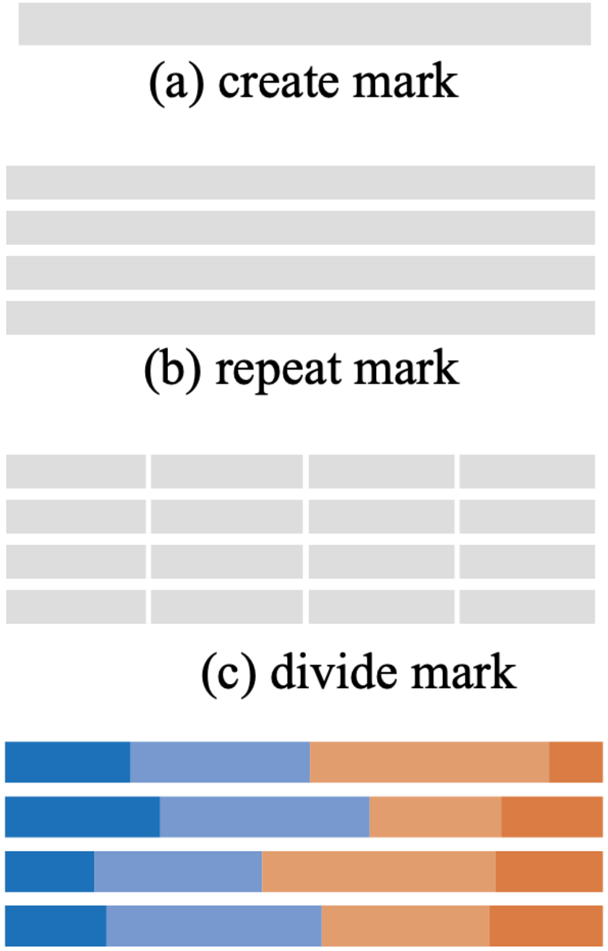
(a) **create** a rectangle mark;

(b) **repeat** the rectangle by age;

(c) **divide** the bars by response;

Scene Manipulation: Operations

Encode visual channels with data



(a) create mark

(b) repeat mark

(c) divide mark

(d) encode mark with data

(a) **create** a rectangle mark;

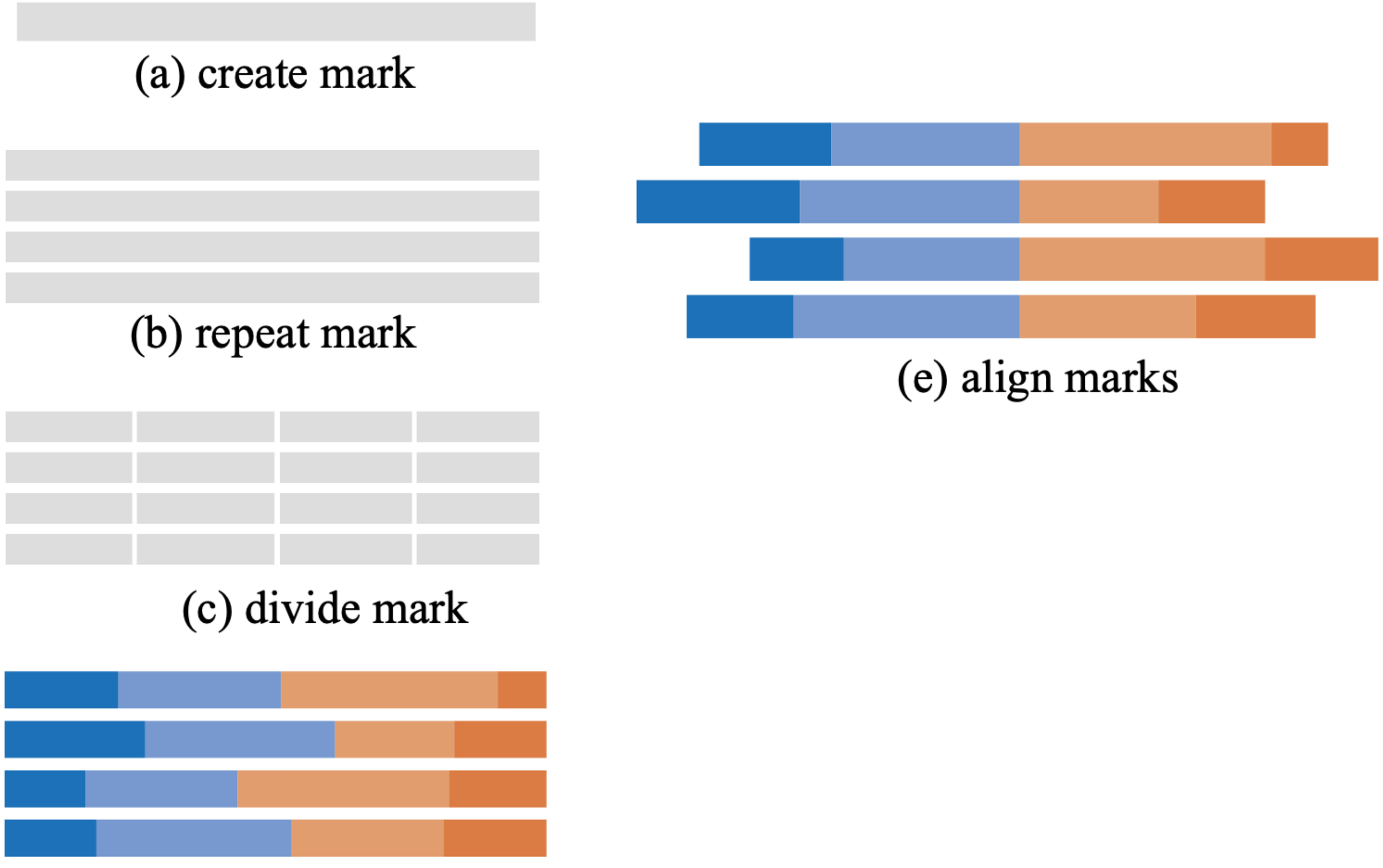
(b) **repeat** the rectangle by age;

(c) **divide** the bars by response;

(d) **encode** the rectangles' width by response and fill color by pct;

Scene Manipulation: Operations

Align elements according to data



(a) create mark

(b) repeat mark

(c) divide mark

(d) encode mark with data

(e) align marks

(a) **create** a rectangle mark;

(b) **repeat** the rectangle by age;

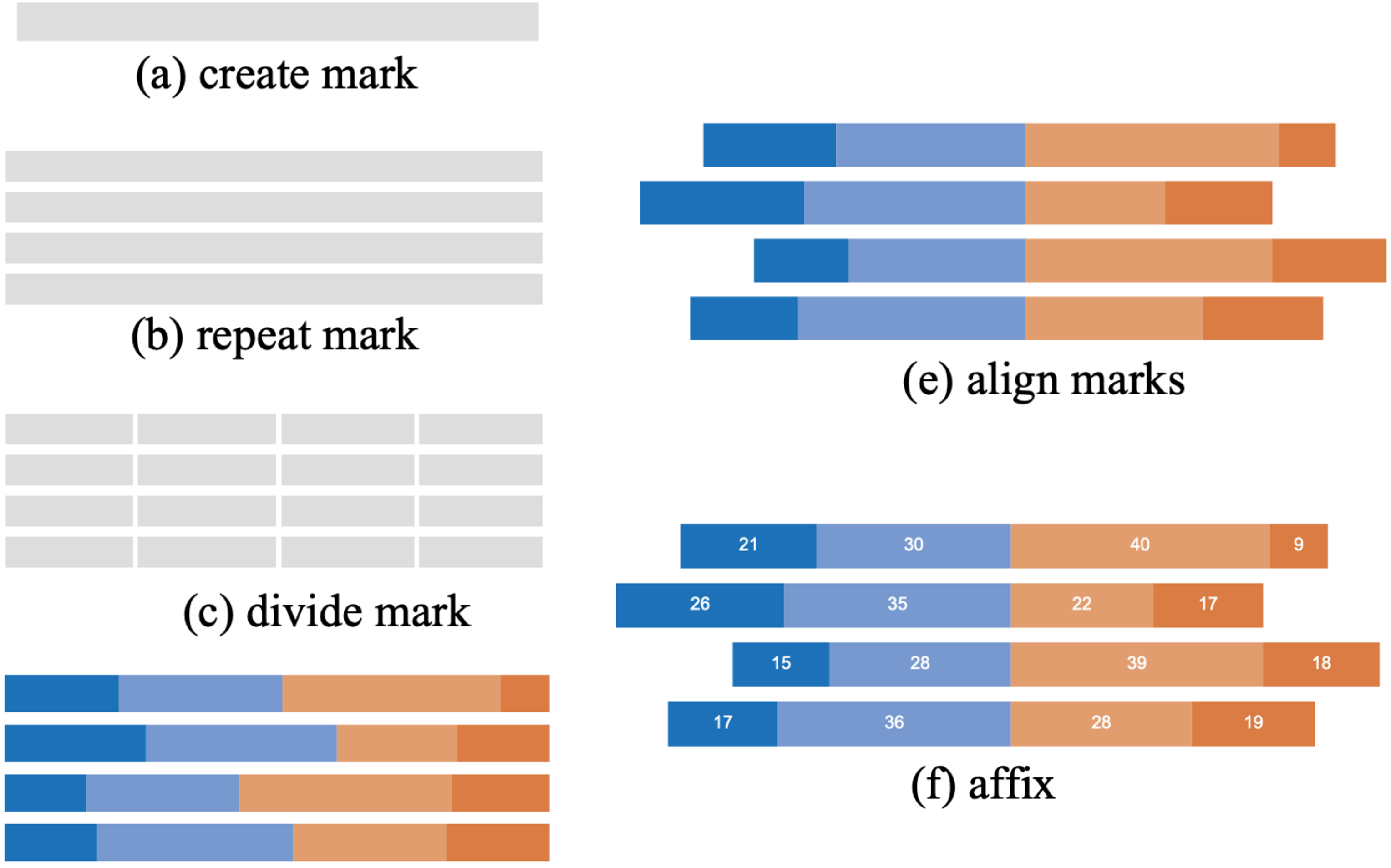
(c) **divide** the bars by response;

(d) **encode** the rectangles’ width by response and fill color by pct;

(e) **align** the light blue rectangles to the right to show the divergence;

Scene Manipulation: Operations

Generate and Affix texts with bars



(a) create mark

(b) repeat mark

(c) divide mark

(d) encode mark with data

(e) align marks

(f) affix

(a) **create** a rectangle mark;

(b) **repeat** the rectangle by age;

(c) **divide** the bars by response;

(d) **encode** the rectangles' width by response and fill color by pct;

(e) **align** the light blue rectangles to the right to show the divergence;

(f) **repeat** an initial text item by pct and **affix** them to the center of corresponding rectangles.

MSC: Components and Operations

Components

Visual Elements

mark, glyph, collection, reference element

Data Scope

Algorithmic Layouts

grid, stack, packing, …

Encodings & Scales

Relational Constraints

alignment, affixation, …

View Configuration

Operations

Generative

repeat
densify
repopulate
divide
classify
stratify

Modificative

apply/remove encoding
customize scale
set channel values
apply/remove layout
update layout parameter
apply/remove constraint
configure view

MSC: Components and Operations

Components

Visual Elements

mark, glyph, collection, reference element

Data Scope

Algorithmic Layouts

grid, stack, packing, …

Encodings & Scales

Relational Constraints

alignment, affixation, …

View Configuration

Operations

Generative

repeat
densify
repopulate

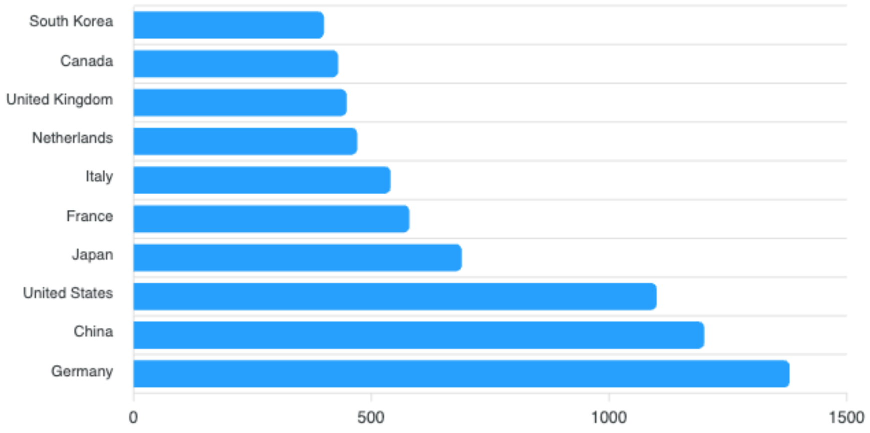
divide
classify
stratify

Modificative

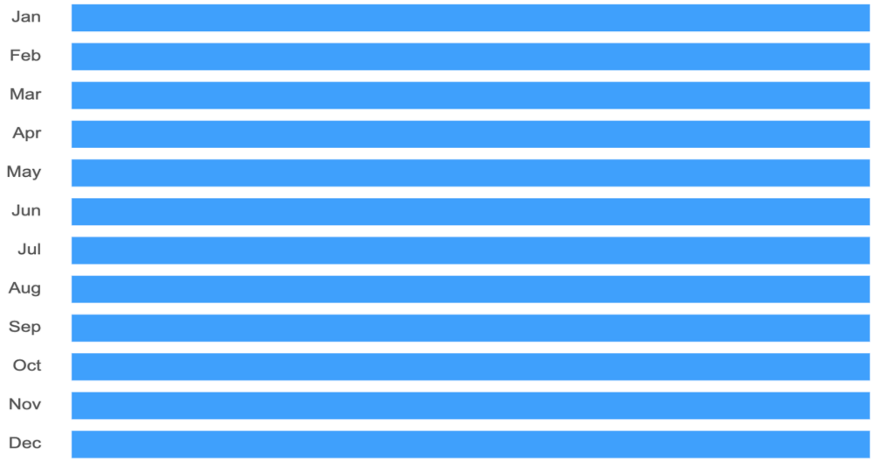
apply/remove encoding
customize scale
set channel values
apply/remove layout
update layout parameter
apply/remove constraint
configure view

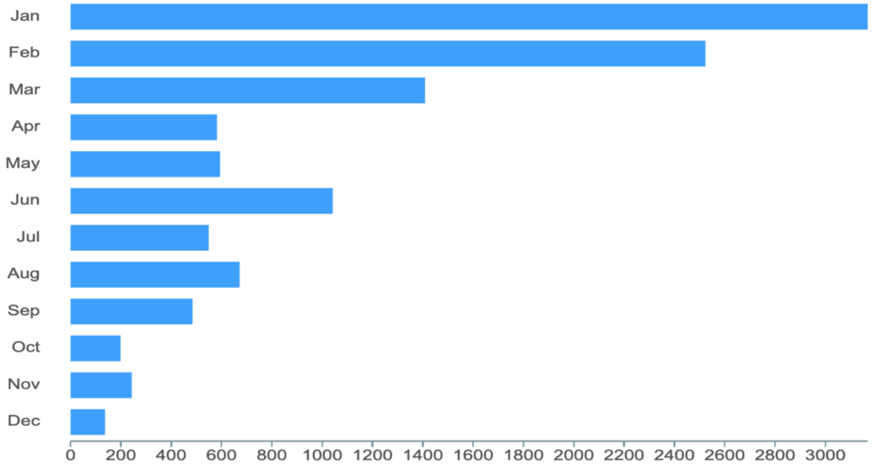
More on the Operations

Repopulate & Apply Encoding



Repopulate: Country -> Month





Apply Encoding: *Width* -> Death

More on the Operations

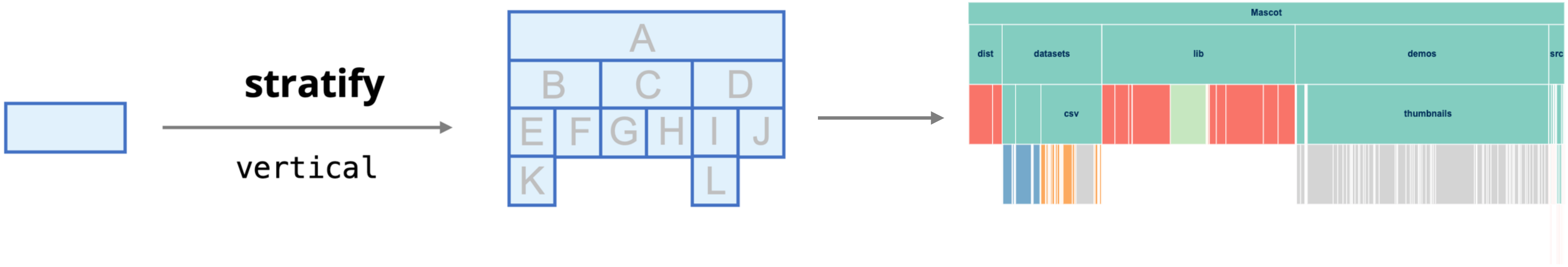
Stratify & Apply Encoding

items

A, B, C, D, E, F
G, H, I, J, K, L

links

A → B, A → C, A → D, B → E,
B → F, C → G, C → H, D → I,
D → J, E → K, I → L

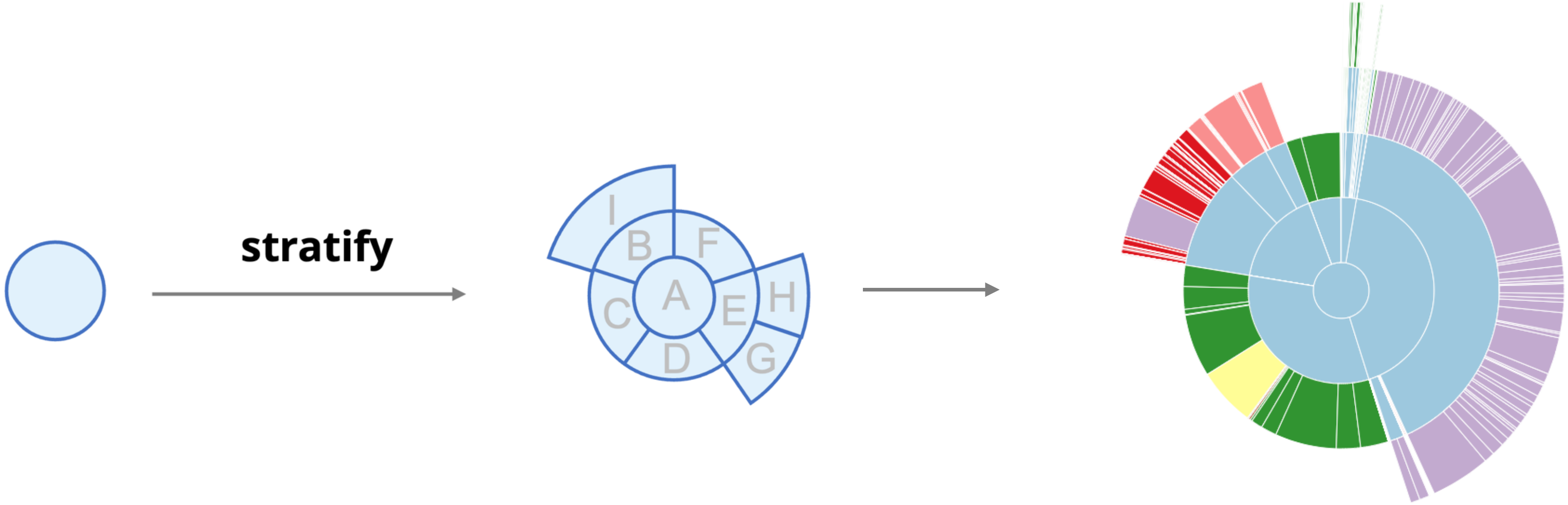


items

A, B, C, D, E, F, G, H, I

links

A → B, A → C, A → D, A → E,
A → F, E → G, E → H, B → I,

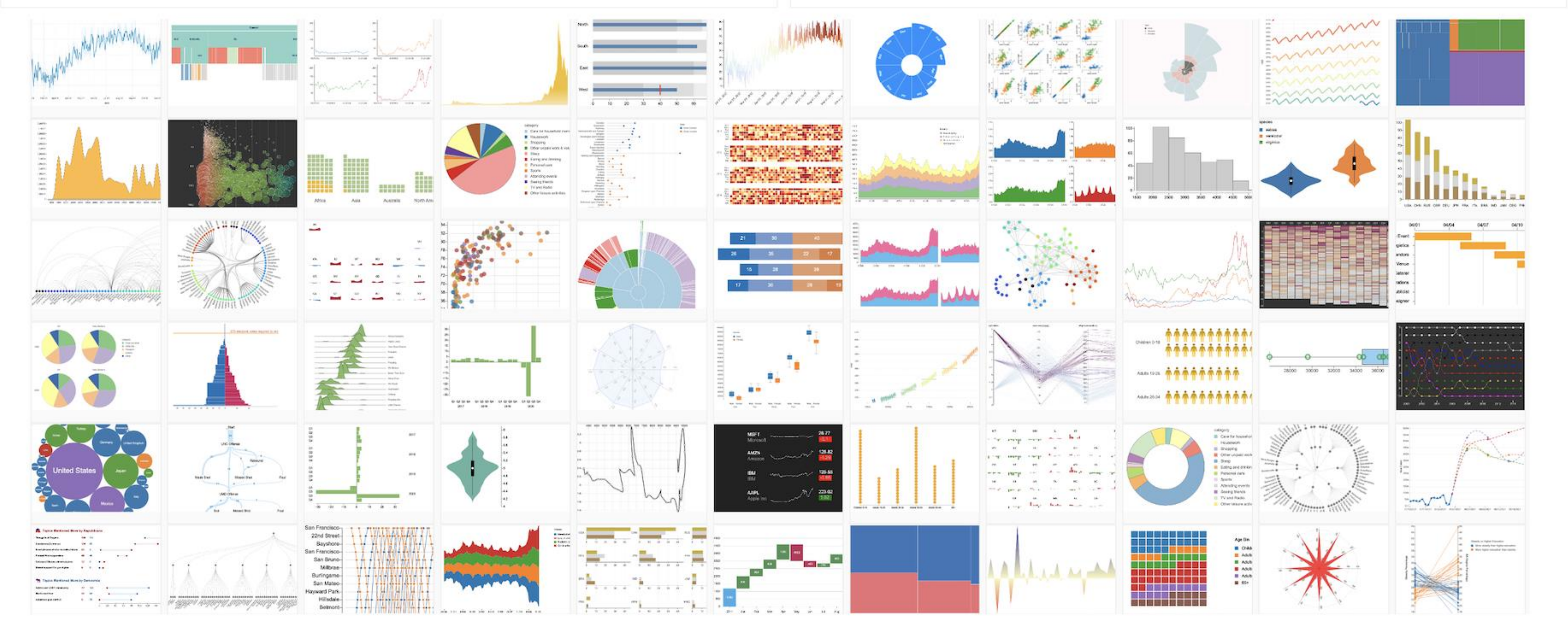


Implementation

Mascot.js

Tutorials, documentation and examples available at:

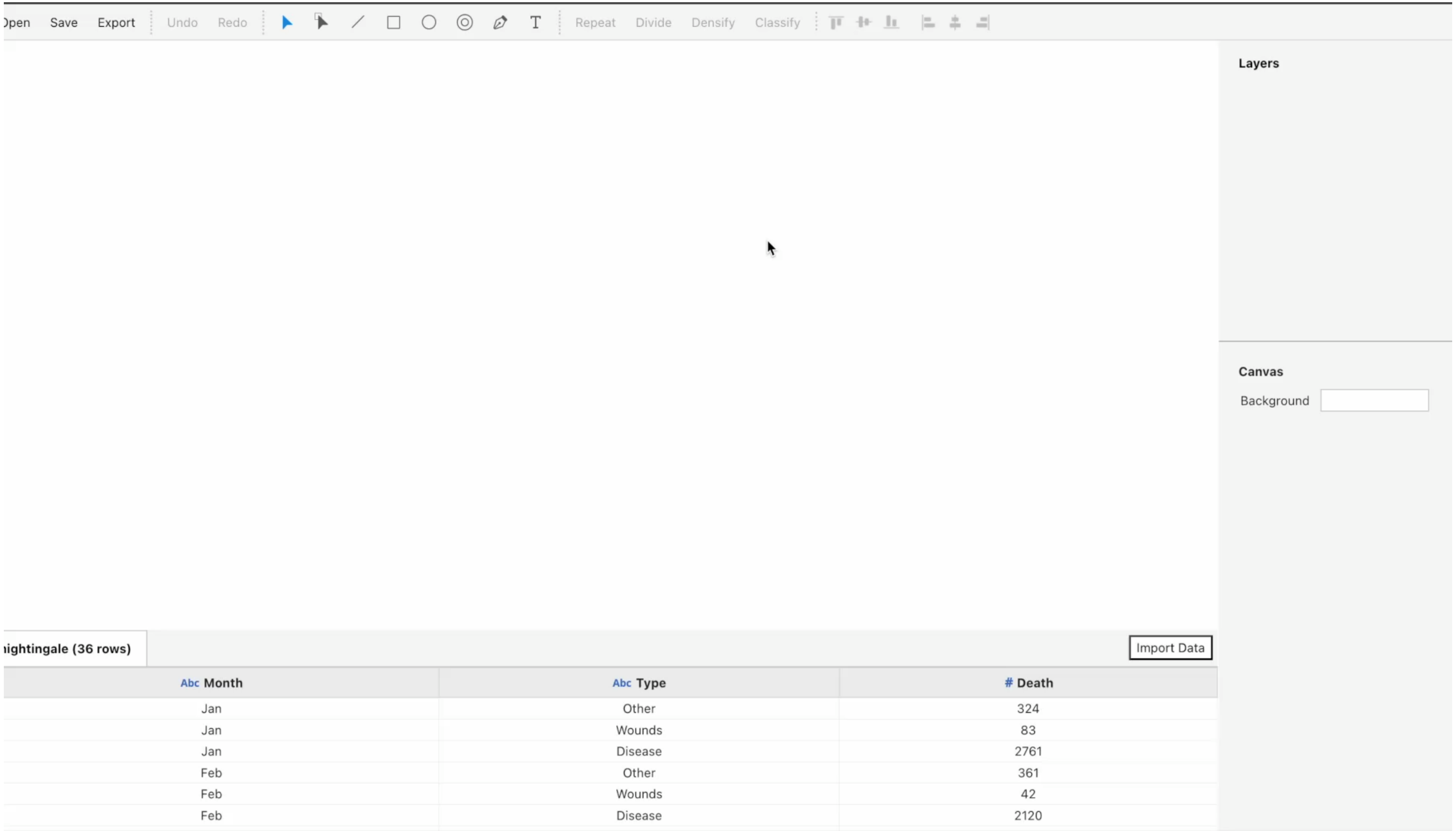
https://mascot-vis.github.io/



Applications 1

Interactive visualization authoring

A new version of the Data Illustrator: https://data-illustrateur.github.io/



ightingale (36 rows)

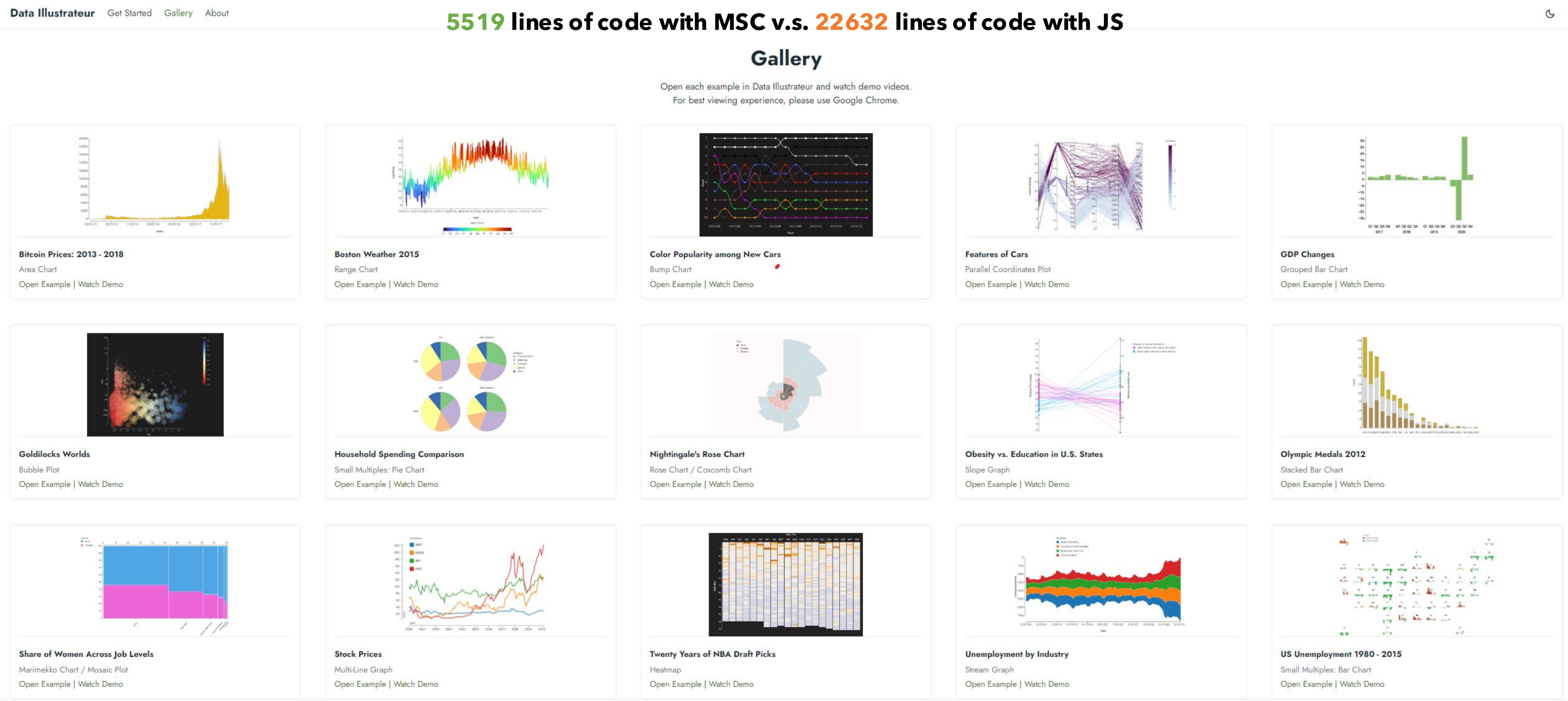
Abc Month	Abc Type	# Death
Jan	Other	324
Jan	Wounds	83
Jan	Disease	2761
Feb	Other	361
Feb	Wounds	42
Feb	Disease	2120

Applications 1

Interactive visualization authoring

A new version of the Data Illustrator: https://data-illustrateur.github.io/

5519 lines of code with MSC v.s. 22632 lines of code with JS



Data Illustrateur Get Started Gallery About

Gallery

Open each example in Data Illustrateur and watch demo videos.
For best viewing experience, please use Google Chrome.

Bitcoin Prices: 2013 - 2018
Area Chart
Open Example | Watch Demo

Boston Weather 2015
Range Chart
Open Example | Watch Demo

Color Popularity among New Cars
Bump Chart
Open Example | Watch Demo

Features of Cars
Parallel Coordinates Plot
Open Example | Watch Demo

GDP Changes
Grouped Bar Chart
Open Example | Watch Demo

Goldilocks Worlds
Bubble Plot
Open Example | Watch Demo

Household Spending Comparison
Small Multiples: Pie Chart
Open Example | Watch Demo

Nightingale's Rose Chart
Rose Chart / Coxcomb Chart
Open Example | Watch Demo

Obesity vs. Education in U.S. States
Slope Graph
Open Example | Watch Demo

Olympic Medals 2012
Stacked Bar Chart
Open Example | Watch Demo

Share of Women Across Job Levels
Marimekko Chart / Mosaic Plot
Open Example | Watch Demo

Stock Prices
Multi-Line Graph
Open Example | Watch Demo

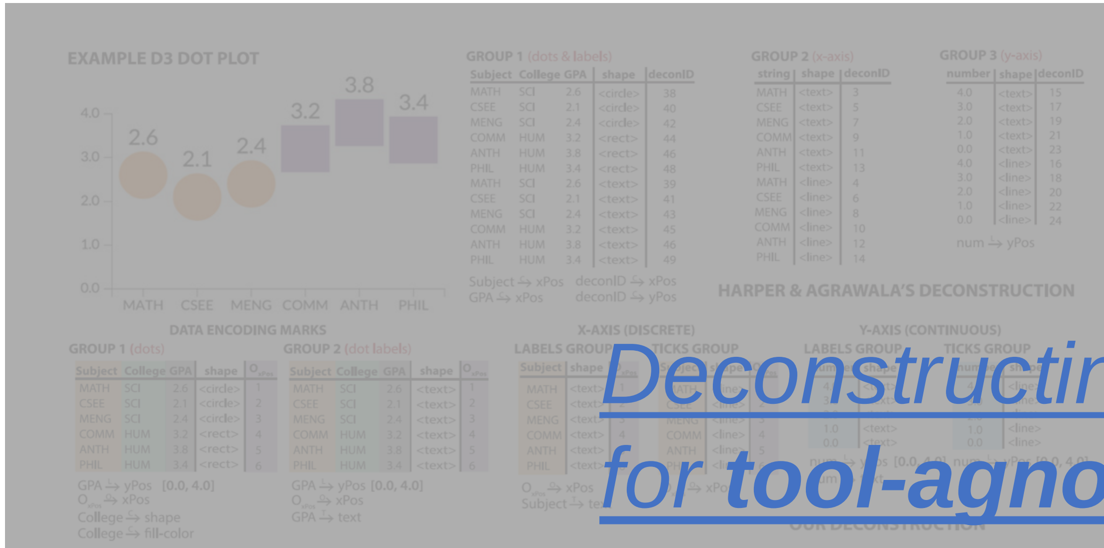
Twenty Years of NBA Draft Picks
Heatmap
Open Example | Watch Demo

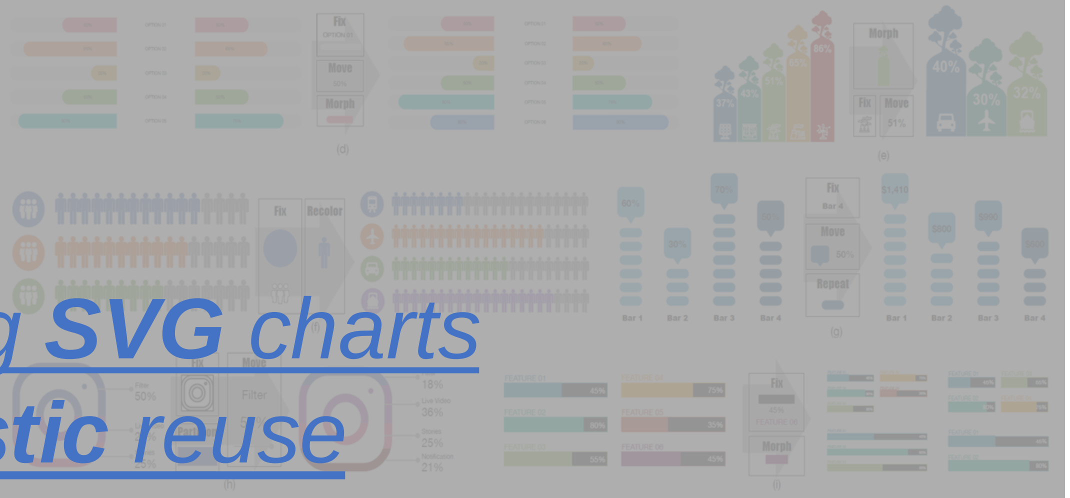
Unemployment by Industry
Stream Graph
Open Example | Watch Demo

US Unemployment 1980 - 2015
Small Multiples: Bar Chart
Open Example | Watch Demo

Applications 2

Interactive chart repurposing





[Deconstructing **SVG charts** for **tool-agnostic** reuse]

D3 Deconstructor, TVCG 2018

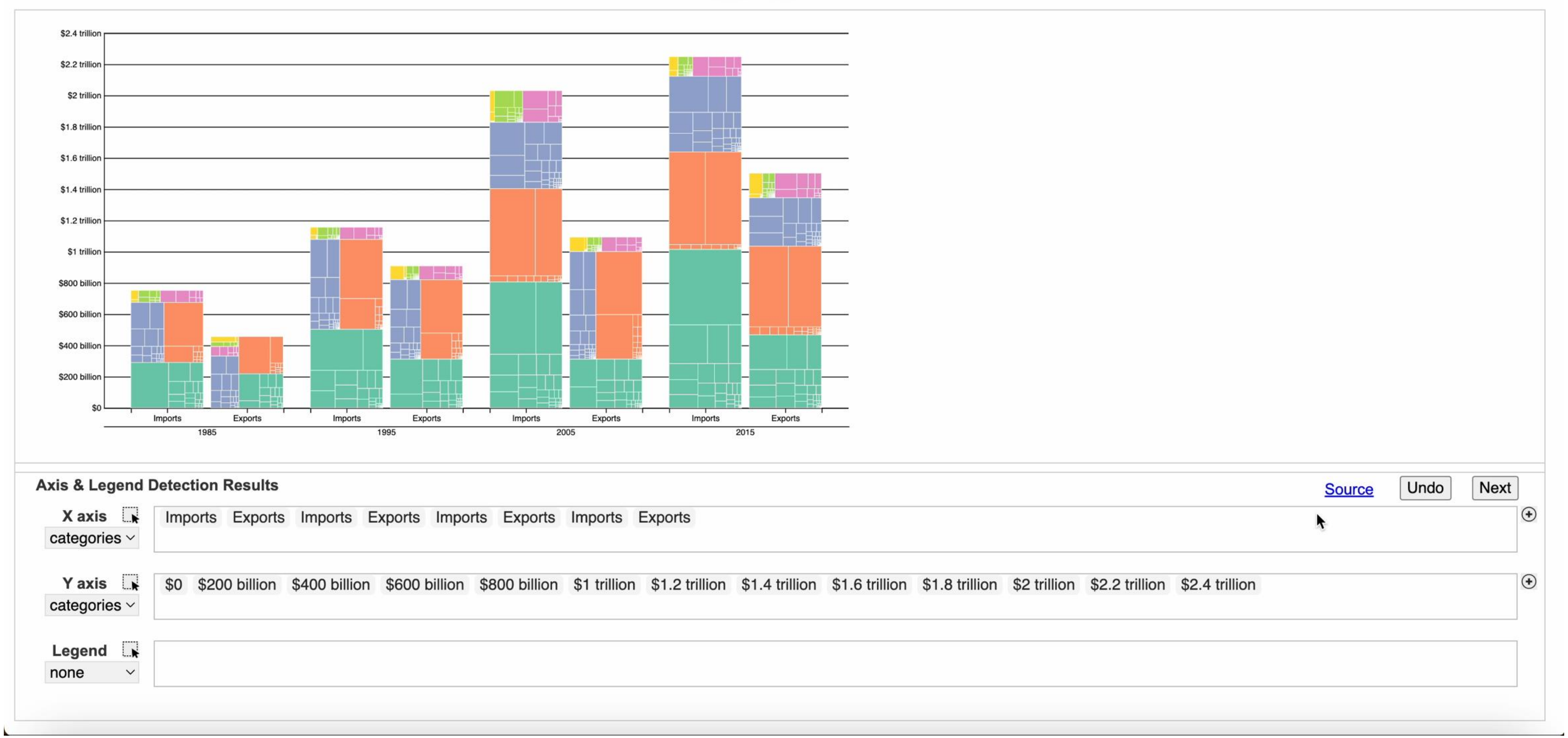
ChartReuse, TVCG 2022

Basic chart types in D3

Glyph-based bar charts in PPT

Applications 2

Interactive chart repurposing



Axis & Legend Detection Results

X axis: categories — Imports, Exports, Imports, Exports, Imports, Exports, Imports, Exports

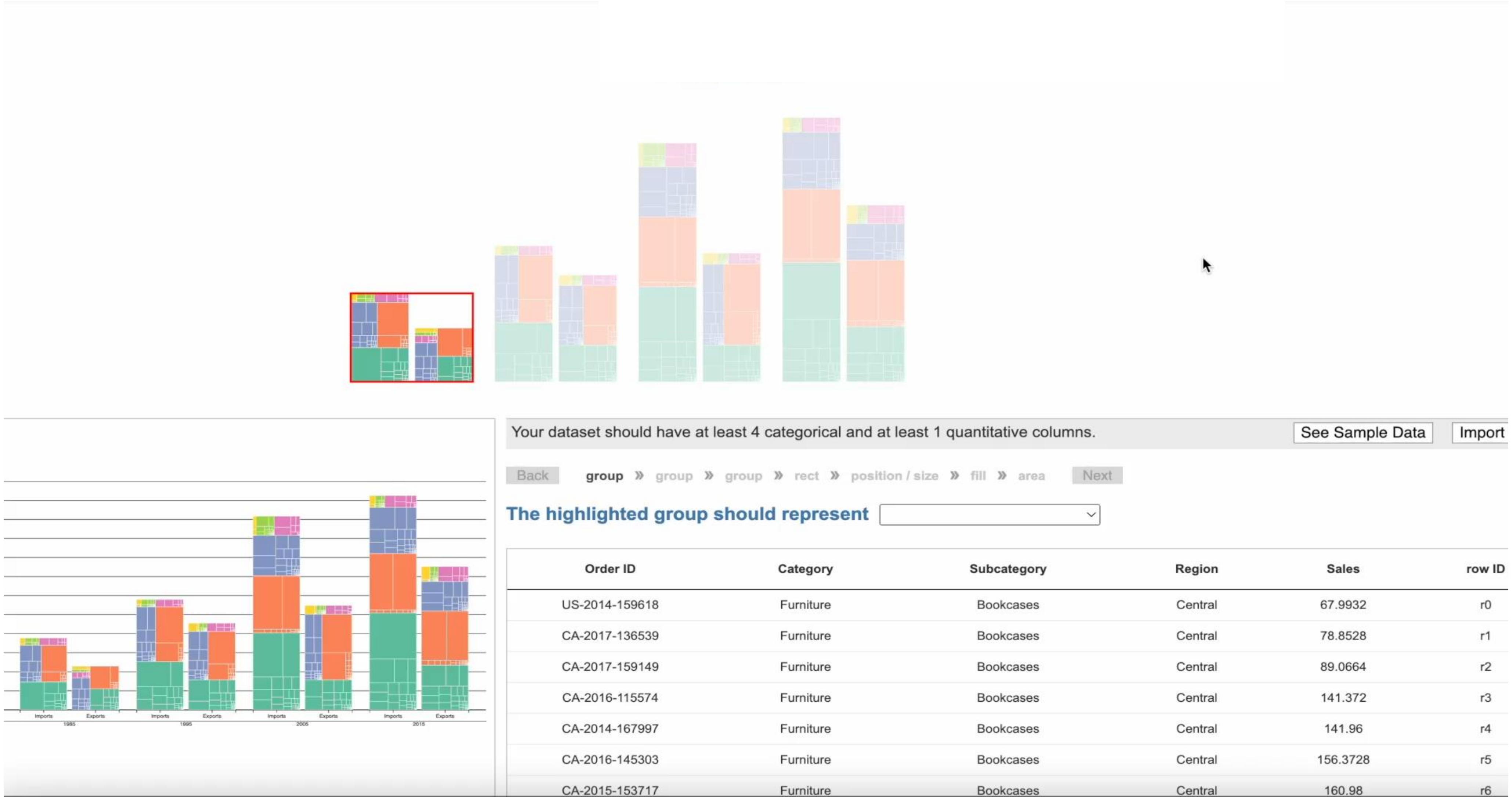
Y axis: categories — $0, $200 billion, $400 billion, $600 billion, $800 billion, $1 trillion, $1.2 trillion, $1.4 trillion, $1.6 trillion, $1.8 trillion, $2 trillion, $2.2 trillion, $2.4 trillion

Legend: none

Chen C, Lee B, Wang Y, Chang Y, Liu Z. Mystique: Deconstructing SVG Charts for Layout Reuse. IEEE Transactions on Visualizati on and Computer Graphics. 2023 Oct 26.

Applications 2

Interactive chart repurposing



See Sample Data
Import
Back
group » group » group » rect » position / size » fill » area
Next
Imports
Exports
1985
1995
2005
2015

Your dataset should have at least 4 categorical and at least 1 quantitative columns.

The highlighted group should represent

Order ID	Category	Subcategory	Region	Sales	row ID
US-2014-159618	Furniture	Bookcases	Central	67.9932	r0
CA-2017-136539	Furniture	Bookcases	Central	78.8528	r1
CA-2017-159149	Furniture	Bookcases	Central	89.0664	r2
CA-2016-115574	Furniture	Bookcases	Central	141.372	r3
CA-2014-167997	Furniture	Bookcases	Central	141.96	r4
CA-2016-145303	Furniture	Bookcases	Central	156.3728	r5
CA-2015-153717	Furniture	Bookcases	Central	160.98	r6

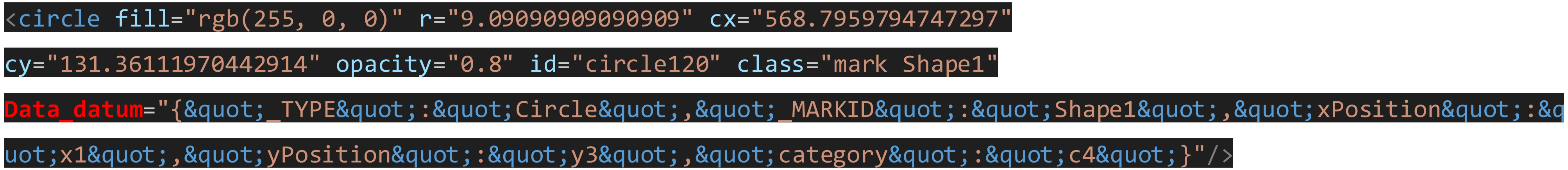
Chen C, Lee B, Wang Y, Chang Y, Liu Z. Mystique: Deconstructing SVG Charts for Layout Reuse. IEEE Transactions on Visualization and Computer Graphics. 2023 Oct 26.

Applications 3

animating static visualizations

In this use case, we explore how MSC supports augmentation tasks like animating static visualizations.

- We use CAST as the animation tool, which requires the input format called data-enrich SVG (dSVG)



```
<circle fill="rgb(255, 0, 0)" r="9.090909090909" cx="568.7959794747297"
cy="131.36111970442914" opacity="0.8" id="circle120" class="mark Shape1"
Data_datum="{&quot;_TYPE&quot;:&quot;Circle&quot;,&quot;_MARKID&quot;:&quot;Shape1&quot;,&quot;xPosition&quot;:&q
uot;x1&quot;,&quot;yPosition&quot;:&quot;y3&quot;,&quot;category&quot;:&quot;c4&quot;}"/>
```

- We achieved a unified script to turn SVGs with the MSC representations into the dSVG format

Applications 3

animating static visualizations

Summary

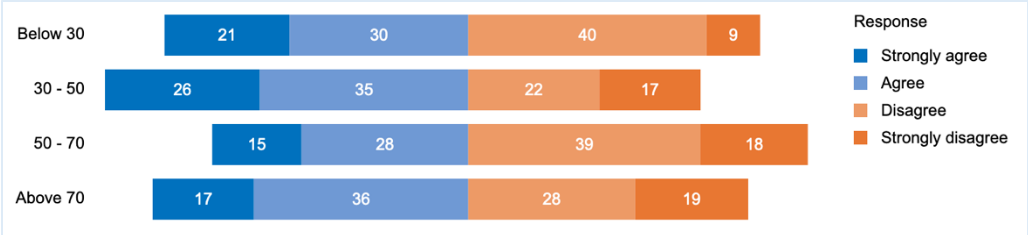
Manipulable Semantic Components (MSC)

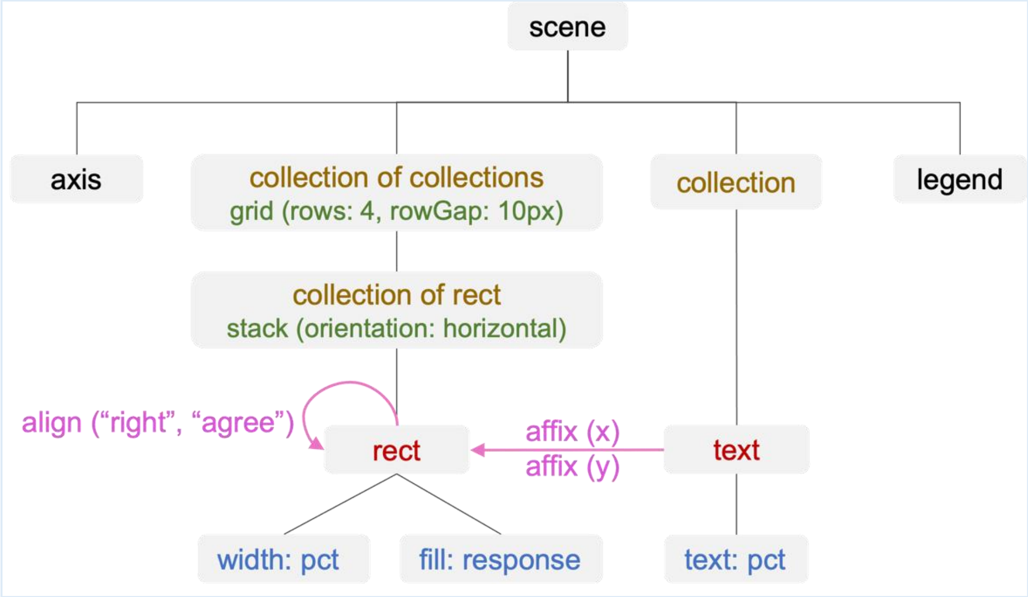
- MSC is a computational data visualization scene representation.
- It contains (1) a unified object model and (2) an operation set.
- We show its applications in interactive visualization authoring, chart deconstruction and reuse , and animating static visualizations.

Project Page: **https://mascot-vis.github.io/**

Contact: cchen24@umd.edu

SCAN ME





scene

axis

collection of collections
grid (rows: 4, rowGap: 10px)

collection of rect
stack (orientation: horizontal)

align (“right”, “agree”)

rect

affix (x)
affix (y)

collection

text

legend

width: pct

fill: response

text: pct