Web Crawler Practice

Web Design IV - JavaScript Advance

Dr. Chun-Hsiang Chan

Department of Geography,
National Taiwan Normal University

Outline

- What's JSON?
- JSON I/O
- What's Chart.js?
- What's D3.js?

Source: https://miro.medium.com/v2/resize:fit:1358/1*Sj10emlsj2VUZE8CmJjTPg.png

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation.
- JSON is a **text format** for storing and transporting data.
- JSON is "**self-describing**" and easy to understand.

*The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

JSON Data Structure

```
// Data Structure
{
"ISO_Data":[
  {"country_name":"Taiwan","population":"23,568,378"},
  {"country_name":"Japan", "population":"122,797,230"},
  {"country_name":"Korea", "population":"51,752,676"}
]
}
```

JSON Data Type

- In JSON, values must be one of the following data types:
  - a **string**
  - a **number**
  - an **object** (JSON object)
  - an **array**
  - a **boolean**
  - ***null***

JSON Demo

Demo: w0401_json_demo.html

```
<p id="demo"></p>
<script>
let text = '{"ISO_Data":[{"country_name":"Taiwan","population":"23,568,378"},{"country_name":"Jap
an","population":"122,797,230"}, {"country_name":"Korea","population":"51,752,676"}]}';

const obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
    obj.ISO_Data[1].country_name + " " + obj.ISO_Data[1].population;
</script>
```

In the browser, …

Japan 122,797,230

JSON Demo

- **Lab Practice:**

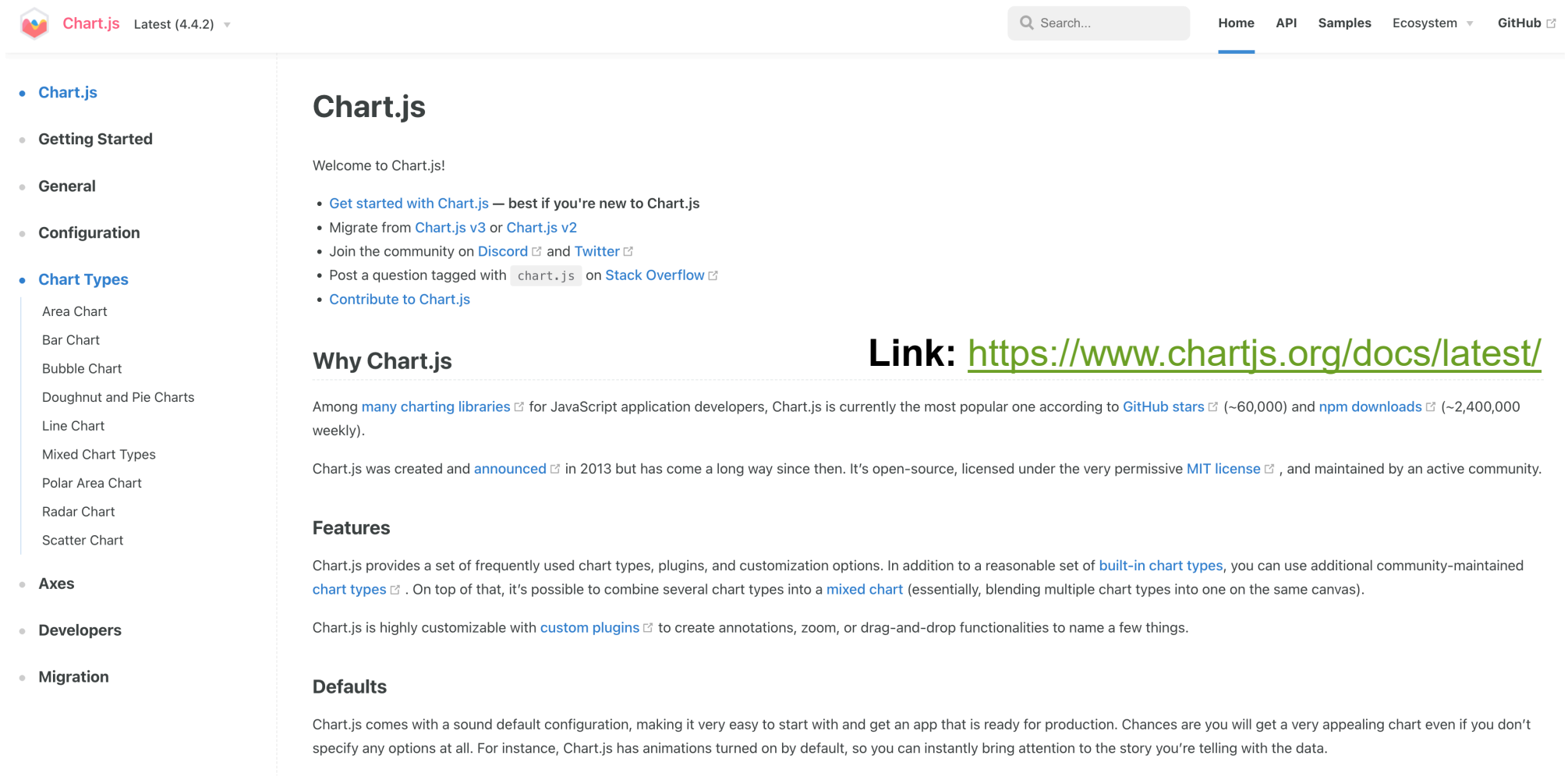
Using JavaScript to print all data one by one, as follows.

1 Taiwan 23,568,378

2 Japan 122,797,230

3 Korea 51,752,676

Get to Know Chart.js



Link: https://www.chartjs.org/docs/latest/

Source: https://www.chartjs.org/docs/latest/

Get to Know Chart.js

Why Chart.js?

- Among many charting libraries for JavaScript application developers, Chart.js is currently the most popular one according to GitHub stars (~60,000) and npm downloads (~2,400,000 weekly).

Features

- Chart.js provides a set of frequently used chart types, plugins, and customization options. In addition to a reasonable set of built-in chart types, you can use additional community-maintained chart types . On top of that, it's possible to combine several chart types into a mixed chart (essentially, blending multiple chart types into one on the same canvas).

Source: https://www.chartjs.org/docs/latest/

Get to Know Chart.js

Integrations

- Chart.js comes with built-in TypeScript typings and is compatible with all popular JavaScript frameworks including React , Vue , Svelte , and Angular . You can use Chart.js directly or leverage well-maintained wrapper packages that allow for a more native integration with your frameworks of choice.

Source: https://www.chartjs.org/docs/latest/

Get to Know Chart.js

Canvas rendering

- Chart.js renders chart elements on an HTML5 canvas unlike several others, mostly D3.js-based, charting libraries that render as SVG. Canvas rendering makes Chart.js very performant, especially for large datasets and complex visualizations that would otherwise require thousands of SVG nodes in the DOM tree. At the same time, canvas rendering disallows CSS styling, so you will have to use built-in options for that or create a custom plugin or chart type to render everything to your liking.

Source: https://www.chartjs.org/docs/latest/

Chart.js ➔ Installation

- Follow a step-by-step guide to get up to speed with Chart.js
- Install Chart.js from **npm** or a **CDN**
  - First, installing **`Nodejs`**
  - Second, installing **`npm install -g npm`**
  - Third, **`npm install chart.js`**
- Integrate Chart.js with bundlers, loaders, and front-end frameworks

Source: https://www.chartjs.org/docs/latest/

npm install –g npm

Getting started / Configuring / Downloading and installing Node.js and npm

Downloading and installing Node.js and npm

To publish and install packages to and from the public npm registry or a private npm registry, you must install Node.js and the npm command line interface using either a Node version manager or a Node installer. **We strongly recommend using a Node version manager like nvm to install Node.js and npm.** We do not recommend using a Node installer, since the Node installation process installs npm in a directory with local permissions and can cause permissions errors when you run npm packages globally.

Note: to download the latest version of npm, on the command line, run the following command:

```
npm install –g npm
```


Source: https://docs.npmjs.com/downloading-and-installing-node-js-and-npm

Install Dependencies

- Open the terminal, … install npm and the rest of the dependencies
  - `npm install -g npm`
  - `npm install chart.js`
  - `npm install parcel`
  - …

Demonstrate the Webpage

```
(base) toodou@TooDous-MBP:~/Desktop$ npx http-server
Starting up http-server, serving ./

http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://127.0.0.1:8080
  http://192.168.50.140:8080
Hit CTRL-C to stop the server
```

1. Change the directory to your webpage.
2. Add **`npx http-server`**
3. Open a browser
4. Set **url** with **`localhost:8080`**
5. To terminate the local server with **`CTRL+C`**

Chart.js Barplot

```html
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <title>ChartJS Bar</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>

<body>
    <div>
        <canvas id="myChart"></canvas>
    </div>
    <script>
    const ctx = document.getElementById('myChart');

    new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple',
                'Orange'],
            datasets: [{
                label: '# of Votes',
                data: [12, 19, 3, 5, 2, 3],
                borderWidth: 1
            }]
        },
        options: {
            scales: {
                y: {
                    beginAtZero: true
                }
            }
        }
    });
    </script>
</body>
```

import chart.js from online source

Add canvas for plotting the Chart.js figure

Get the element for plotting the Chart.js figure

Chart.js settings

- Element variable
- Chart type
- Data
- Options
  - Scale…
  - …

Chart.js Barplot

```html
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <title>ChartJS Bar</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>

<body>
    <div>
        <canvas id="myChart"></canvas>
    </div>
    <script>
    const ctx = document.getElementById('myChart');

    new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple',
                'Orange'],
            datasets: [{
                label: '# of Votes',
                data: [12, 19, 3, 5, 2, 3],
                borderWidth: 1
            }]
        },
        options: {
            scales: {
                y: {
                    beginAtZero: true
                }
            }
        }
    });
    </script>
</body>
```

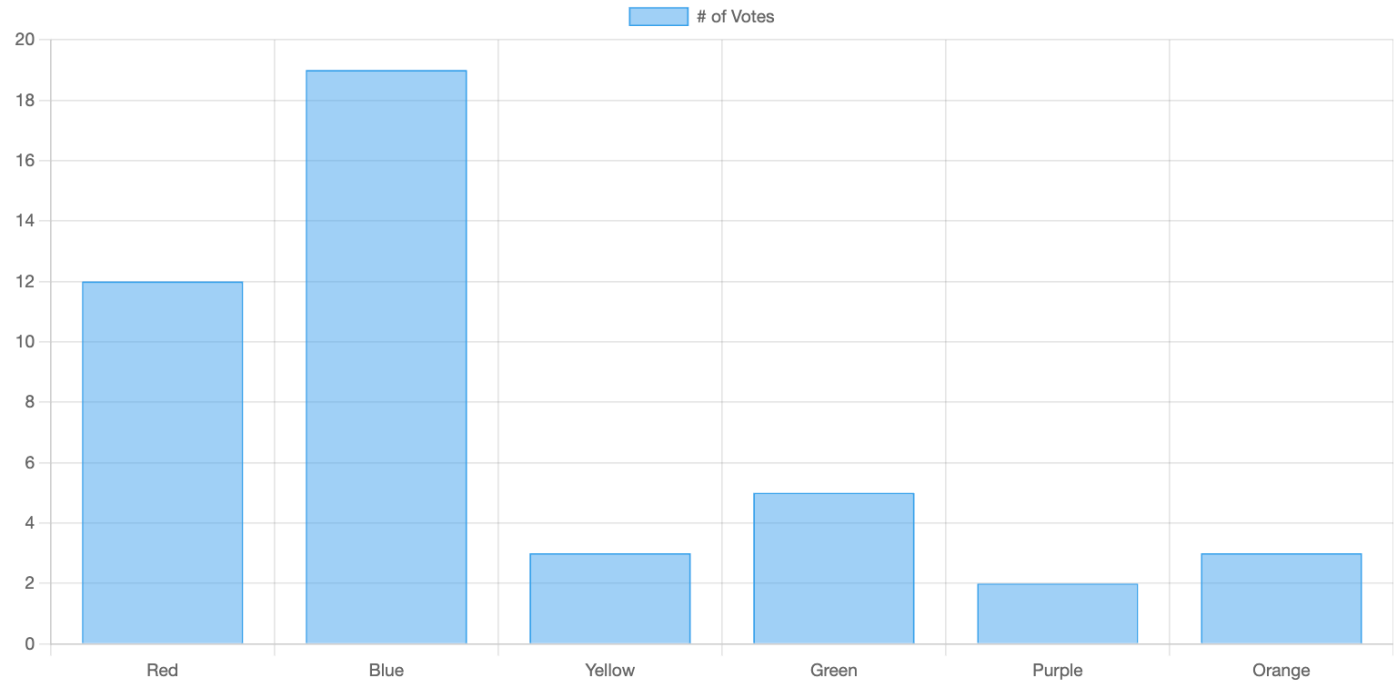


Chart.js Bubble

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <title>ChartJS Bubble</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
```

```
<div style="width: 500px;"><canvas id="myChart"></canvas></div><br />
<script type="text/javascript">
const data = {
    datasets: [{
        label: 'First Dataset',
        data: [{
            x: 20,
            y: 30,
            r: 1.5
        }, {
            x: 40,
            y: 10,
            r: 12
        }, {
            x: 20,
            y: 34,
            r: 2.9
        }, {
            x: 56,
            y: 72,
            r: 3.7
        }, {
            x: 34,
            y: 62,
            r: 4.5
        }, {
            x: 20,
            y: 17,
            r: 7.8
        }, {
            x: 13,
            y: 19,
            r: 10
        }],
        backgroundColor: 'rgb(255, 99, 132)'
    }, {
        label: 'Second Dataset',
        data: [{
            x: 20,
            y: 20,
            r: 1.5
        }, {
            x: 40,
            y: 40,
            r: 12
        }, {
            x: 20,
            y: 14,
            r: 2.9
        }, {
            x: 56,
            y: 22,
            r: 3.7
        }, {
            x: 34,
            y: 12,
            r: 4.5
        }, {
            x: 10,
            y: 27,
            r: 7.8
        }, {
            x: 23,
            y: 49,
            r: 10
        }],
        backgroundColor: 'rgb(25, 29, 252)'
    }]
};
```

Chart.js Bubble

```
const ctx = document.getElementById('myChart');

new Chart(ctx, {
    type: 'bubble',
    data: data,
    options: {
        scales: {
            y: {
                beginAtZero: true
            }
        }
    }
});
</script>
```

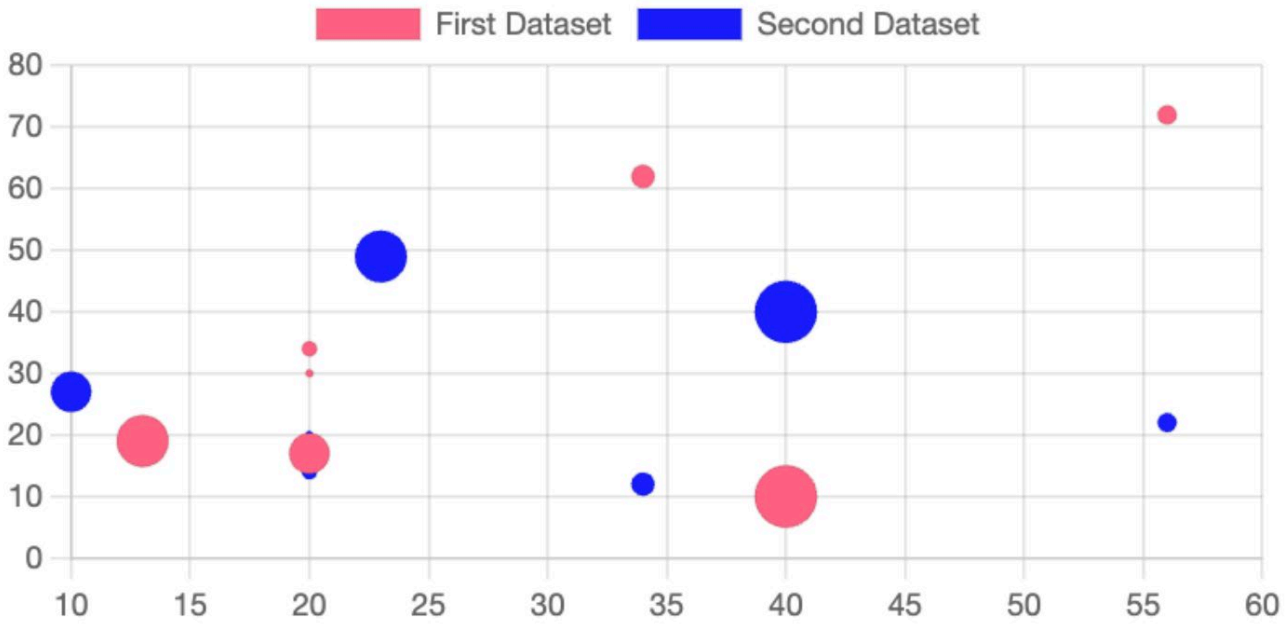


Chart.js Doughnut

```html
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>ChartJS Doughnut</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="Chart.js/docs/scripts/utils.js"></script>
</head>
```

```html
<div style="width: 500px;"><canvas id="myChart"></canvas></div>
<script>
const data = {
    labels: [
        'Red',
        'Blue',
        'Yellow'
    ],
    datasets: [{
        label: 'My First Dataset',
        data: [300, 50, 100],
        backgroundColor: [
            'rgb(255, 99, 132)',
            'rgb(54, 162, 235)',
            'rgb(255, 205, 86)'
        ],
        hoverOffset: 4
    }]
};

const ctx = document.getElementById('myChart');

new Chart(ctx, {
    type: 'doughnut',
    data,
    options: {}
});
</script>
```

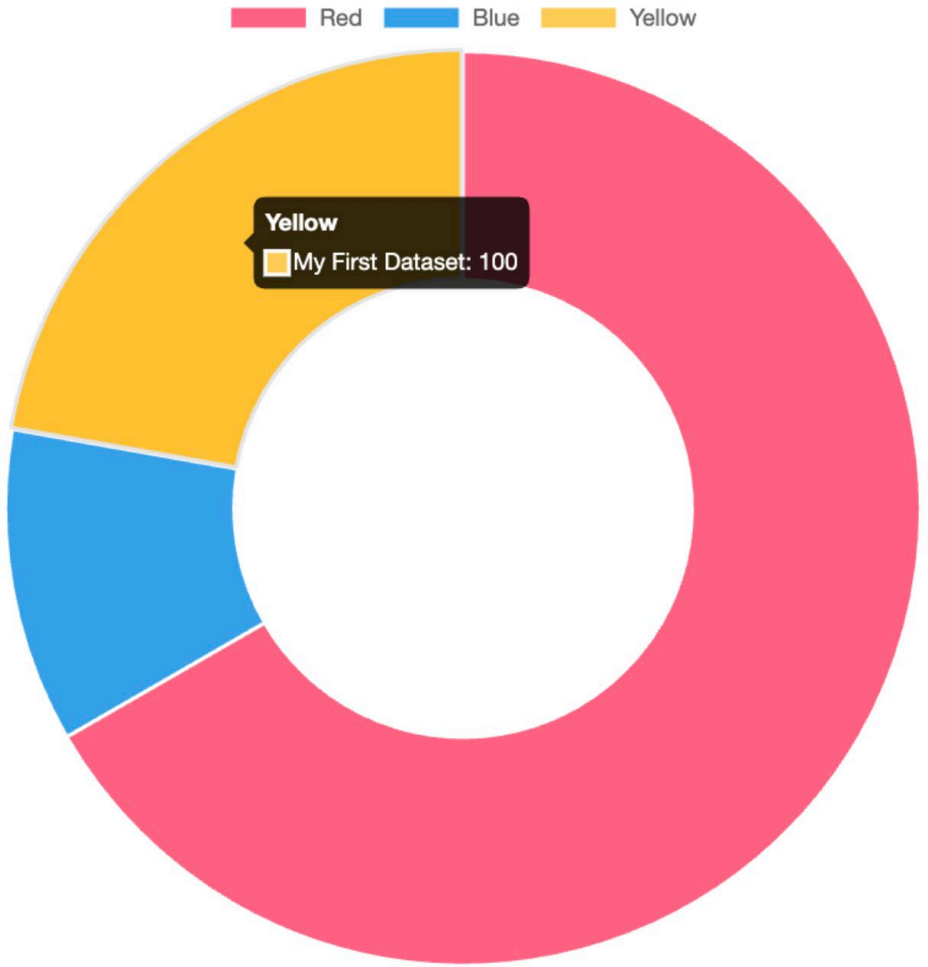


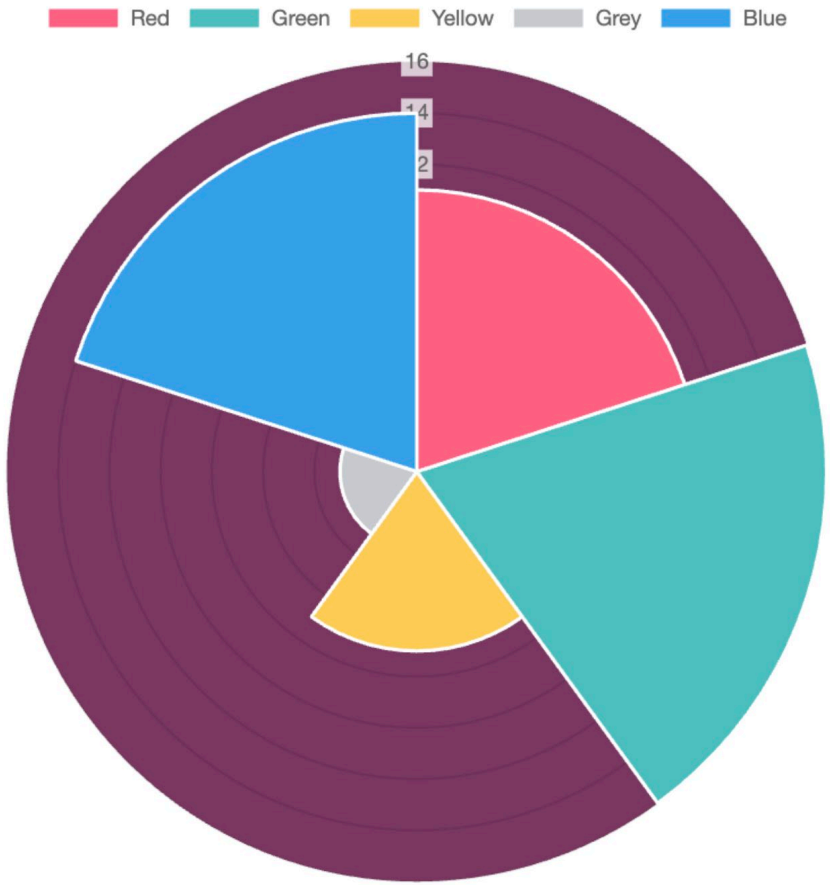
Chart.js Polar Area

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>ChartJS Doughnut</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
```

```
<div style="width: 500px;"><canvas id="myChart"></canvas></div>
<script>
const data = {
    labels: [
        'Red',
        'Green',
        'Yellow',
        'Grey',
        'Blue'
    ],
    datasets: [{
        label: 'My First Dataset',
        data: [11, 16, 7, 3, 14],
        backgroundColor: [
            'rgb(255, 99, 132)',
            'rgb(75, 192, 192)',
            'rgb(255, 205, 86)',
            'rgb(201, 203, 207)',
            'rgb(54, 162, 235)'
        ]
    }]
};

const ctx = document.getElementById('myChart');

new Chart(ctx, {
    type: 'polarArea',
    data,
    options: {
        scale: { backgroundColor: 'rgb(123, 56, 98)' }
    }
});
</script>
```



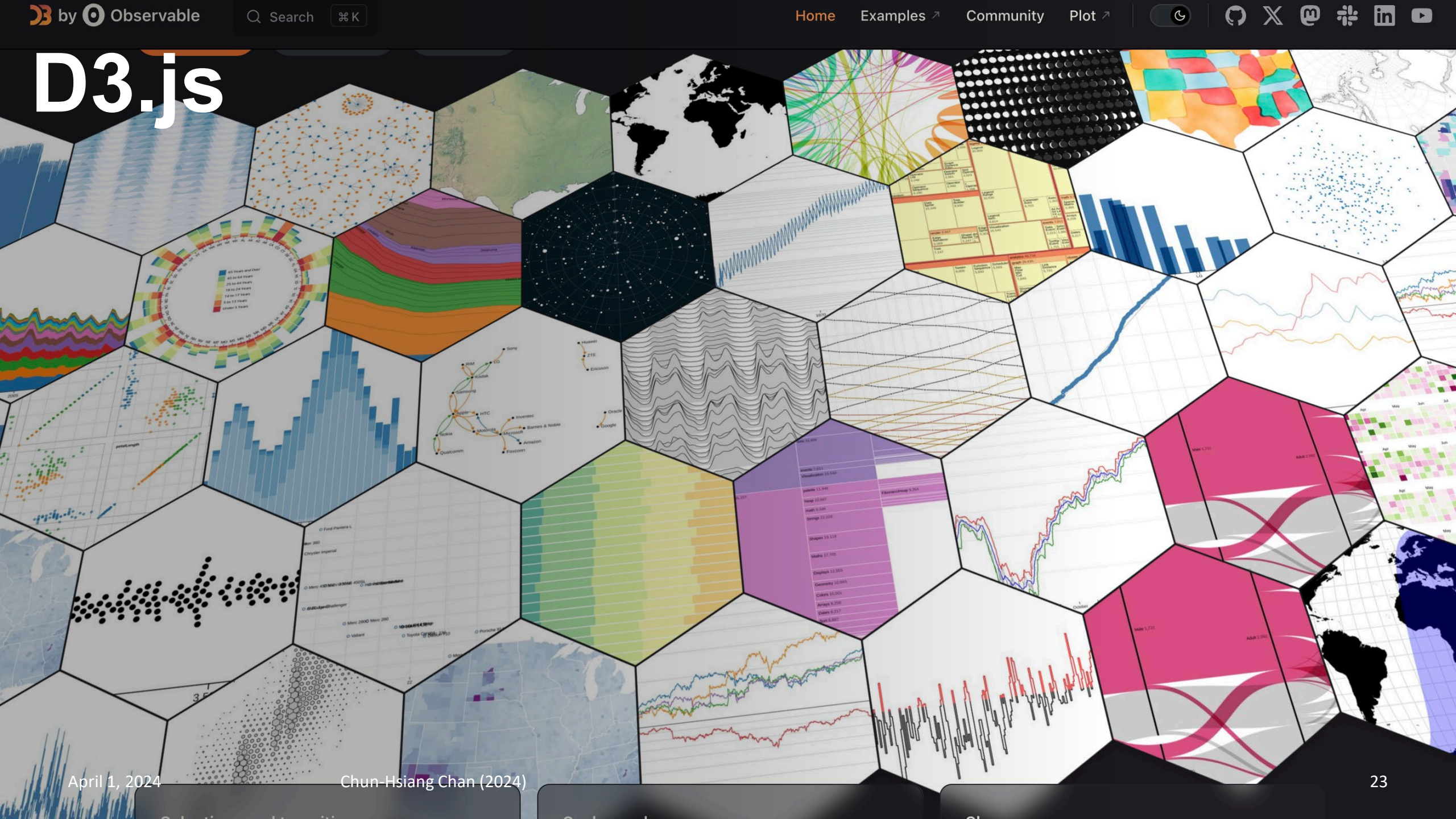
Lab Practice

- Download real-world data and plot it with Chart.js!
- The official webpage also has some examples and demos.

What's D3.js?

- **D3** (or **D3.js**) is a free, open-source JavaScript library for visualizing data. Its low-level approach built on web standards offers unparalleled flexibility in authoring dynamic, data-driven graphics. For more than a decade, D3 has powered groundbreaking and award-winning visualizations, become a foundational building block of higher-level chart libraries, and fostered a vibrant community of data practitioners worldwide.

Source: https://d3js.org/what-is-d3


by Observable
Search ⌘K
Home
Examples
Community
Plot
D3.js

D3.js

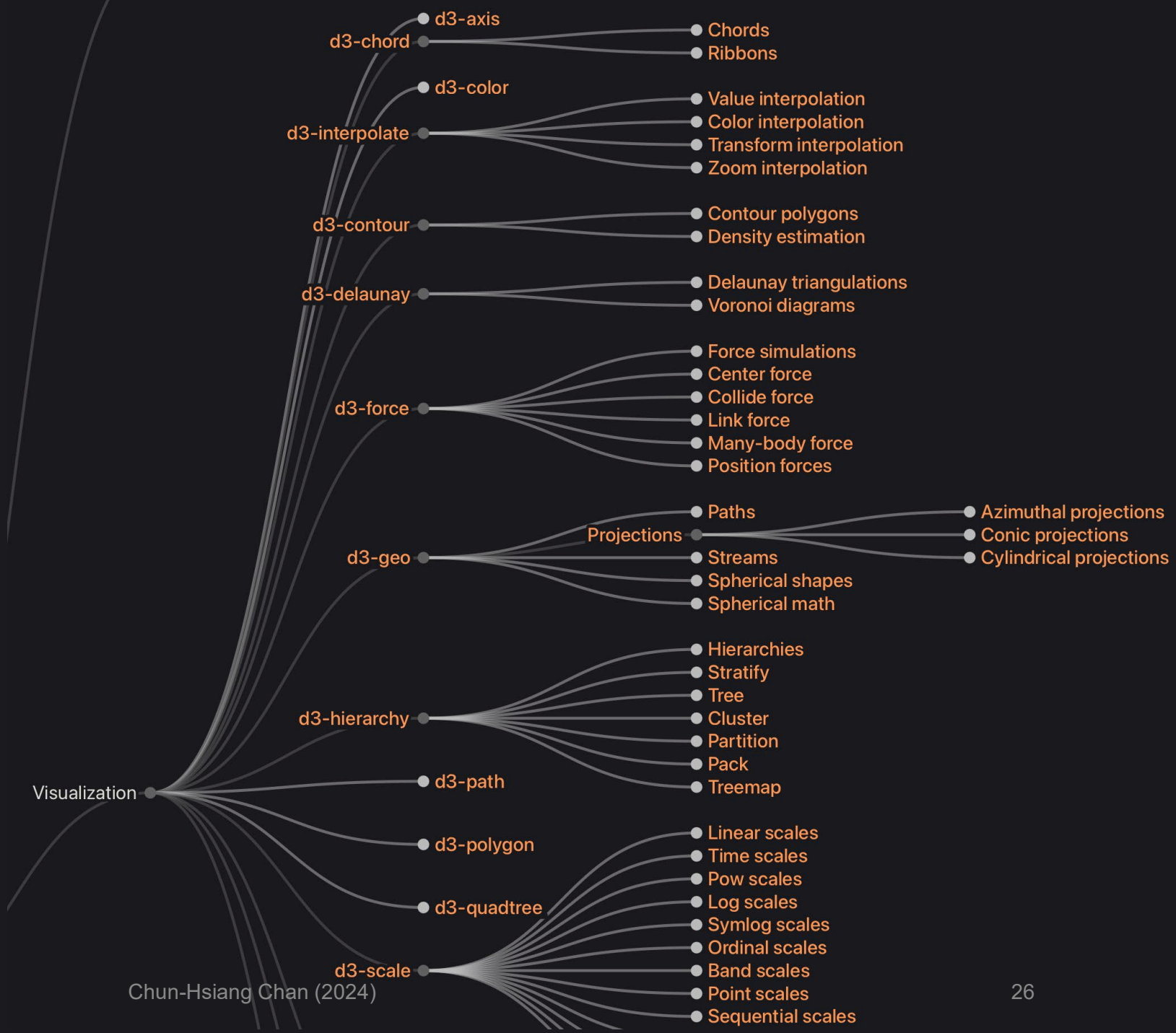
- **D3 is a low-level toolbox.**
- D3 is not a charting library in the traditional sense. It has no concept of “charts”. When you visualize data with D3, you compose a variety of primitives.
- To make a stacked area chart, you might use
  - a **CSV** parser to load data,
  - a time scale for a horizontal position (*x*),
  - a linear scale for a vertical position (*y*),
  - an ordinal scale and categorical scheme for color,
  - a stack layout for arranging values,
  - an area shape with a linear curve for generating SVG path data,
  - axes for documenting the position encodings, and
  - selections for creating SVG elements.

Source: https://d3js.org/what-is-d3

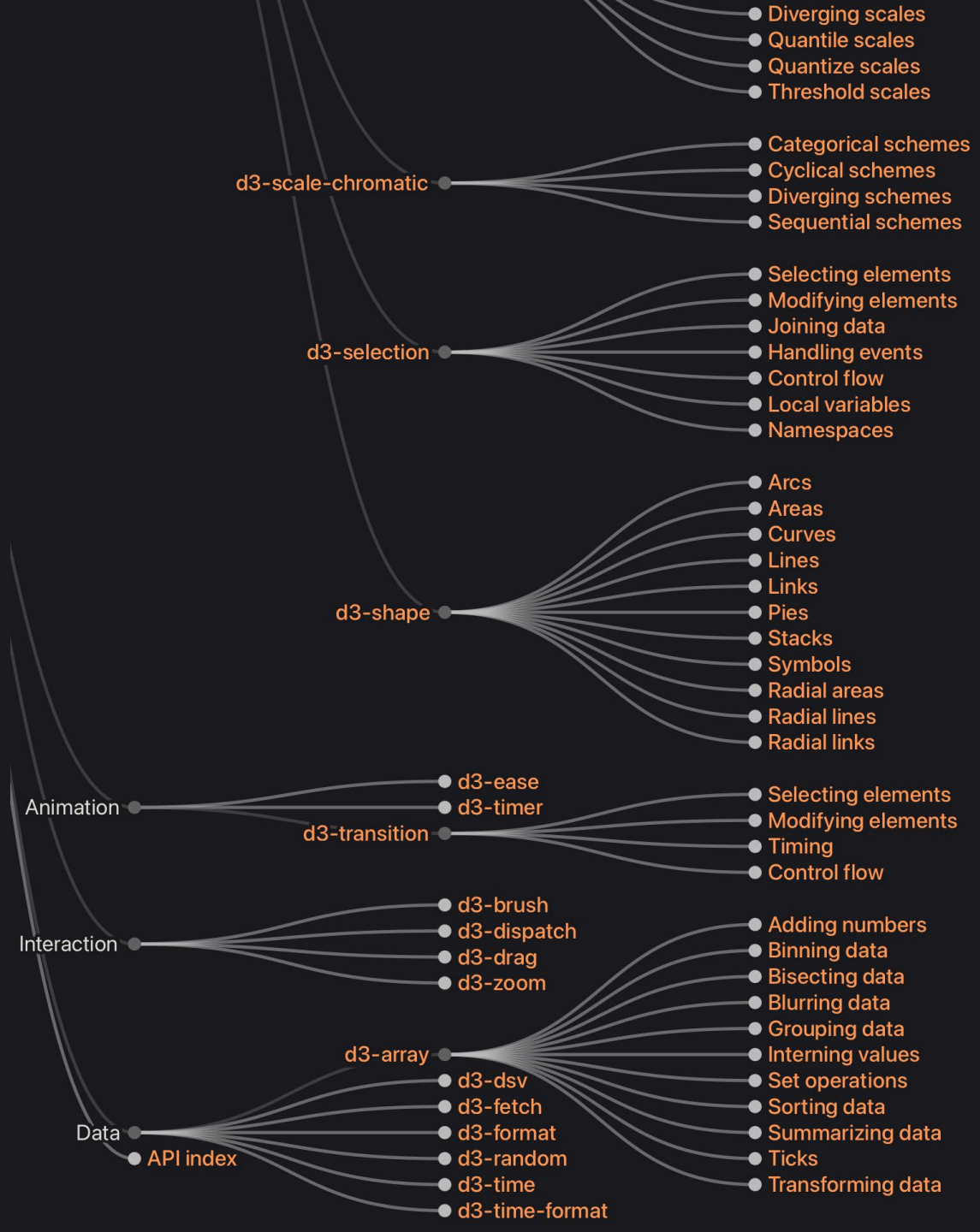
Differences between D3.js and Chart.js

- D3.js requires dynamic demonstration; therefore, we cannot use D3.js illustrations on the GitHub platform or in the university webpage space.
- Different from D3.js, Chart.js does not require dynamic demonstration; as a result, it could be illustrated on the GitHub platform or in the university webpage space.

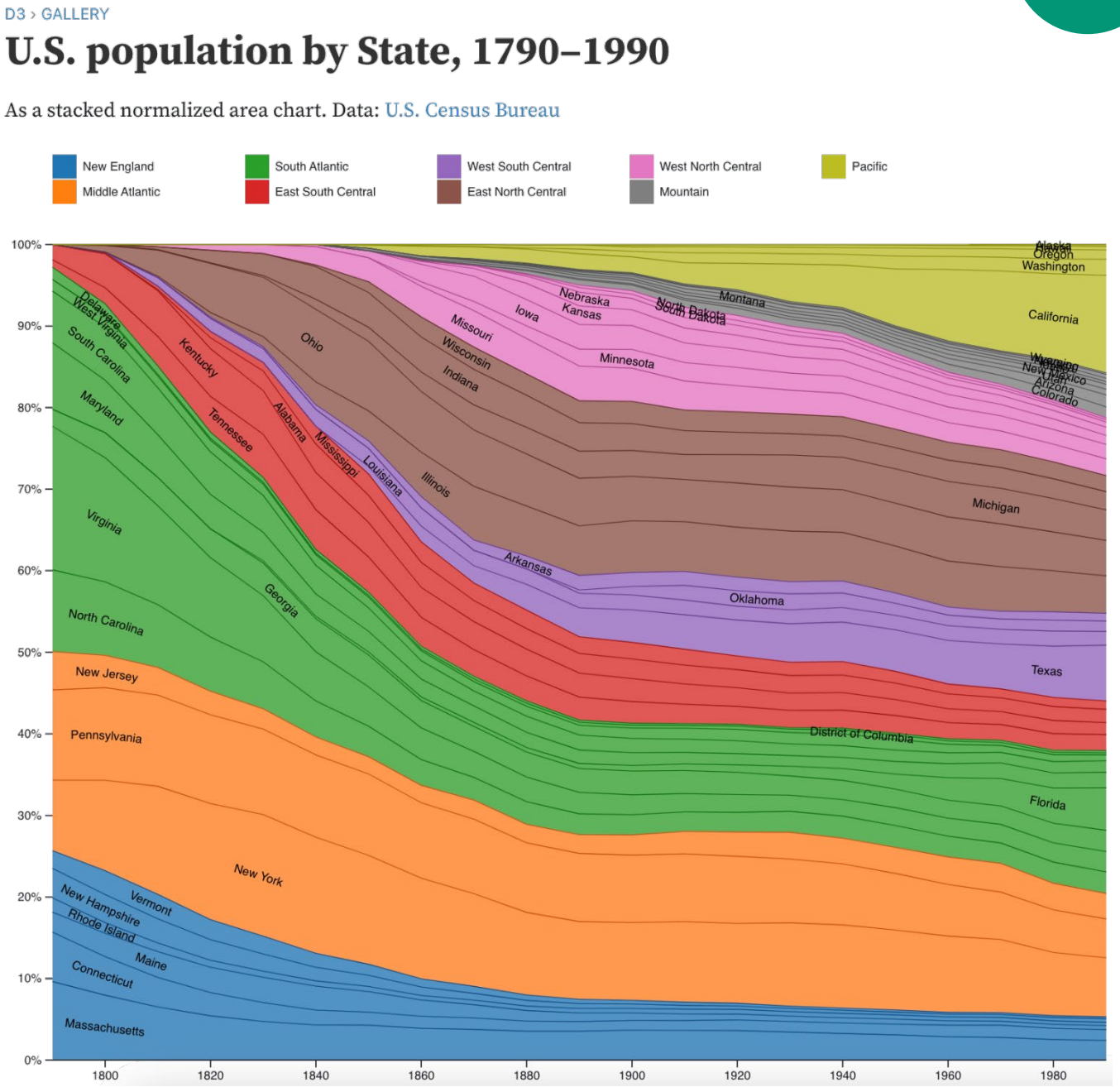
Functions


Visualization
d3-axis
d3-chord
Chords
Ribbons
d3-color
d3-interpolate
Value interpolation
Color interpolation
Transform interpolation
Zoom interpolation
d3-contour
Contour polygons
Density estimation
d3-delaunay
Delaunay triangulations
Voronoi diagrams
d3-force
Force simulations
Center force
Collide force
Link force
Many-body force
Position forces
d3-geo
Paths
Projections
Azimuthal projections
Conic projections
Cylindrical projections
Streams
Spherical shapes
Spherical math
d3-hierarchy
Hierarchies
Stratify
Tree
Cluster
Partition
Pack
Treemap
d3-path
d3-polygon
d3-quadtree
d3-scale
Linear scales
Time scales
Pow scales
Log scales
Symlog scales
Ordinal scales
Band scales
Point scales
Sequential scales

Functions


Diverging scales
Quantile scales
Quantize scales
Threshold scales
d3-scale-chromatic
Categorical schemes
Cyclical schemes
Diverging schemes
Sequential schemes
d3-selection
Selecting elements
Modifying elements
Joining data
Handling events
Control flow
Local variables
Namespaces
d3-shape
Arcs
Areas
Curves
Lines
Links
Pies
Stacks
Symbols
Radial areas
Radial lines
Radial links
Animation
d3-ease
d3-timer
d3-transition
Selecting elements
Modifying elements
Timing
Control flow
Interaction
d3-brush
d3-dispatch
d3-drag
d3-zoom
d3-array
Adding numbers
Binning data
Bisecting data
Blurring data
Grouping data
Interning values
Set operations
Sorting data
Summarizing data
Ticks
Transforming data
Data
API index
d3-dsv
d3-fetch
d3-format
d3-random
d3-time
d3-time-format

D3.js Demo 1



Source: https://observablehq.com/@d3/u-s-population-by-state-1790-1990?intent=fork

D3.js Demo 1

```
chart = {

  // Declare the chart dimensions and margins.
  const width = 928;
  const height = 720;
  const marginTop = 10;
  const marginRight = 10;
  const marginBottom = 30;
  const marginLeft = 40;

  // Declare the scales.
  const x = d3.scaleUtc()
      .domain(d3.extent(data, d => d.date))
      .range([marginLeft, width - marginRight]);

  const y = d3.scaleLinear()
      .range([height - marginBottom, marginTop]);

  const color = d3.scaleOrdinal()
      .domain(regionRank)
      .range(d3.schemeCategory10)
      .unknown("gray");

  // Create the SVG container.
  const svg = d3.create("svg")
      .attr("width", width)
      .attr("height", height)
      .attr("viewBox", [0, 0, width, height])
      .attr("style", "max-width: 100%; height: auto;");
```

```
  // Create the areas
  const area = d3.area()
      .x(d => x(d.data.date))
      .y0(d => y(d[0]))
      .y1(d => y(d[1]));
  svg.append("g")
      .attr("fill-opacity", 0.8)
    .selectAll("path")
    .data(series)
    .join("path")
      .attr("fill", ({key}) => color(regionByState.get(key)))
      .attr("d", area)
    .append("title")
      .text(({key}) => key);

  // Create the lines.
  const midline = d3.line()
      .curve(d3.curveBasis)
      .x(d => x(d.data.date))
      .y(d => y((d[0] + d[1]) / 2));

  svg.append("g")
      .attr("fill", "none")
      .attr("stroke-width", 0.75)
    .selectAll("path")
    .data(series)
    .join("path")
      .attr("stroke", ({key}) => d3.lab(color(regionByState.get(key))).darker())
      .attr("d", area.lineY1());

  // Append a path for text labels
  svg.append("defs")
    .selectAll("path")
    .data(series)
    .join("path")
      .attr("id", d => (d.id = DOM.uid("state")).id)
      .attr("d", midline);
```


Source: https://observablehq.com/@d3/u-s-population-by-state-1790-1990?intent=fork

D3.js Demo 1

```
  // Append a path for text labels
  svg.append("defs")
    .selectAll("path")
    .data(series)
    .join("path")
      .attr("id", d => (d.id = DOM.uid("state")).id)
      .attr("d", midline);

  // Append the labels.
  svg.append("g")
      .style("font", "10px sans-serif")
      .attr("text-anchor", "middle")
    .selectAll("text")
    .data(series)
    .join("text")
      .attr("dy", "0.35em")
    .append("textPath")
      .attr("href", d => d.id.href)
      .attr("startOffset", d => `${Math.max(0.05, Math.min(0.95, d.offset = d3.maxIndex(d, d => d[1] - d[0]) / (d.length - 1))) * 100}%`)
      .text(d => d.key);

  // Append the axes.
  svg.append("g")
      .attr("transform", `translate(0,${height - marginBottom})`)
      .call(d3.axisBottom(x).ticks(width / 80).tickSizeOuter(0));

  svg.append("g")
      .attr("transform", `translate(${marginLeft},0)`)
      .call(d3.axisLeft(y).ticks(10, "%"))
      .call(g => g.select(".domain").remove());

  return Object.assign(svg.node(), {scales: {color}});
}
```


Source: https://observablehq.com/@d3/u-s-population-by-state-1790-1990?intent=fork

D3.js Demo 1

```
data = ▼Array(21) [
  0: ▸Object {date: 1790-01-01, Massachusetts: 378787, Connecticut: 237946, Maine: 96540, Rhode Island: 68825, New Hampshire: 141885, Ve
  1: ▸Object {date: 1800-01-01, Massachusetts: 422845, Connecticut: 251002, Maine: 151719, Rhode Island: 69122, New Hampshire: 183858, V
  2: ▸Object {date: 1810-01-01, Massachusetts: 472040, Connecticut: 261942, Maine: 228705, Rhode Island: 76931, New Hampshire: 214460, V
  3: ▸Object {date: 1820-01-01, Massachusetts: 523287, Connecticut: 275248, Maine: 298335, Rhode Island: 83059, New Hampshire: 244161, V
  4: ▸Object {date: 1830-01-01, Massachusetts: 610408, Connecticut: 297675, Maine: 399455, Rhode Island: 97199, New Hampshire: 269328, V
  5: ▸Object {date: 1840-01-01, Massachusetts: 737699, Connecticut: 309978, Maine: 501793, Rhode Island: 108830, New Hampshire: 284574,
  6: ▸Object {date: 1850-01-01, Massachusetts: 994514, Connecticut: 370792, Maine: 583169, Rhode Island: 147545, New Hampshire: 317976,
  7: ▸Object {date: 1860-01-01, Massachusetts: 1231066, Connecticut: 460147, Maine: 628279, Rhode Island: 174620, New Hampshire: 326073,
  8: ▸Object {date: 1870-01-01, Massachusetts: 1457351, Connecticut: 537454, Maine: 626915, Rhode Island: 217353, New Hampshire: 318300,
  9: ▸Object {date: 1880-01-01, Massachusetts: 1783085, Connecticut: 622700, Maine: 648936, Rhode Island: 276531, New Hampshire: 346991,
  10: ▸Object {date: 1890-01-01, Massachusetts: 2238947, Connecticut: 746258, Maine: 661086, Rhode Island: 345506, New Hampshire: 376530
  11: ▸Object {date: 1900-01-01, Massachusetts: 2805346, Connecticut: 908420, Maine: 694466, Rhode Island: 428556, New Hampshire: 411588
  12: ▸Object {date: 1910-01-01, Massachusetts: 3366416, Connecticut: 1114756, Maine: 742371, Rhode Island: 542610, New Hampshire: 43057
  13: ▸Object {date: 1920-01-01, Massachusetts: 3852356, Connecticut: 1380631, Maine: 768014, Rhode Island: 604397, New Hampshire: 44308
  14: ▸Object {date: 1930-01-01, Massachusetts: 4249614, Connecticut: 1606903, Maine: 797423, Rhode Island: 687497, New Hampshire: 46529
  15: ▸Object {date: 1940-01-01, Massachusetts: 4316721, Connecticut: 1709242, Maine: 847226, Rhode Island: 713346, New Hampshire: 49152
  16: ▸Object {date: 1950-01-01, Massachusetts: 4690514, Connecticut: 2007280, Maine: 913774, Rhode Island: 791896, New Hampshire: 53324
  17: ▸Object {date: 1960-01-01, Massachusetts: 5148578, Connecticut: 2535234, Maine: 969265, Rhode Island: 859488, New Hampshire: 60692
  18: ▸Object {date: 1970-01-01, Massachusetts: 5689170, Connecticut: 3031709, Maine: 992048, Rhode Island: 946725, New Hampshire: 73768
  19: ▸Object {date: 1980-01-01, Massachusetts: 5737037, Connecticut: 3107576, Maine: 1124660, Rhode Island: 947154, New Hampshire: 9206
  … more
]
```

```
data = {
  const years = d3.range(1790, 2000, 10);
  const states = d3.tsvParse(await FileAttachment("population.tsv").text(), (d, i) => i === 0 ? null : ({name: d[""], values: years.map(y
=> +d[y].replace(/,/g, "") || 0)}));
  states.sort((a, b) => d3.ascending(regionRank.indexOf(regionByState.get(a.name)), regionRank.indexOf(regionByState.get(b.name))) ||
d3.descending(d3.sum(a.values), d3.sum(b.values)));
  return Object.assign(years.map((y, i) => Object.fromEntries([["date", new Date(Date.UTC(y, 0, 1))]].concat(states.map(s => [s.name,
s.values[i]])))), {columns: ["date", ...states.map(s => s.name)]});
}
```

```
data = ▼Array(21) [
  0: ▼Object {
    date: 1790-01-01
    Massachusetts: 378787
    Connecticut: 237946
    Maine: 96540
    Rhode Island: 68825
    New Hampshire: 141885
    Vermont: 85425
    New York: 340120
    Pennsylvania: 434373
    New Jersey: 184139
    North Carolina: 393751
    Virginia: 691737
    Georgia: 82548
    Florida: 0
    Maryland: 319728
    South Carolina: 249073
    West Virginia: 55873
    District of Columbia: 0
    Delaware: 59096
    Tennessee: 35691
    … more
  }
```


Source: https://observablehq.com/@d3/u-s-population-by-state-1790-1990?intent=fork

D3.js Demo 1

```
series = ▼Array(51) [
  0: ▼Array(21) [
    0: ▼Array(2) [
      0: 0
      1: 0.09640274110801804
      data: ▼Object {
        date: 1790-01-01
        Massachusetts: 378787
        Connecticut: 237946
        Maine: 96540
        Rhode Island: 68825
        New Hampshire: 141885
        Vermont: 85425
        New York: 340120
        Pennsylvania: 434373
        New Jersey: 184139
        North Carolina: 393751
        Virginia: 691737
        Georgia: 82548
        Florida: 0
        Maryland: 319728
        South Carolina: 249073
        West Virginia: 55873
        District of Columbia: 0
        Delaware: 59096
        Tennessee: 35691
        … more
      }
    ]
    1: ▸Array(2) [0, 0.07965458305131616, data: Object]
    2: ▸Array(2) [0, 0.06519996668453529, data: Object]
    3: ▸Array(2) [0, 0.05429159637962648, data: Object]
    4: ▸Array(2) [0, 0.04746303895386115, data: Object]
```

```
series = d3.stack()
    .keys(data.columns.slice(1))
    .offset(d3.stackOffsetExpand)
  (data)
```

```
regionRank = ▼Array(9) [
  0: "New England"
  1: "Middle Atlantic"
  2: "South Atlantic"
  3: "East South Central"
  4: "West South Central"
  5: "East North Central"
  6: "West North Central"
  7: "Mountain"
  8: "Pacific"
]
```

```
regionRank = [
  "New England",
  "Middle Atlantic",
  "South Atlantic",
  "East South Central",
  "West South Central",
  "East North Central",
  "West North Central",
  "Mountain",
  "Pacific"
]
```

```
regionByState = ▼Map(51) {
  "Alaska" => "Pacific"
Focus cell
ma" => "East South Central"
  "Arkansas" => "West South Central"
  "Arizona" => "Mountain"
  "California" => "Pacific"
  "Colorado" => "Mountain"
  "Connecticut" => "New England"
  "District of Columbia" => "South Atlantic"
  "Delaware" => "South Atlantic"
  "Florida" => "South Atlantic"
  "Georgia" => "South Atlantic"
  "Hawaii" => "Pacific"
  "Iowa" => "West North Central"
  "Idaho" => "Mountain"
  "Illinois" => "East North Central"
  "Indiana" => "East North Central"
  "Kansas" => "West North Central"
  "Kentucky" => "East South Central"
  "Louisiana" => "West South Central"
  "Massachusetts" => "New England"
  … more
}
```

```
regionByState = {
  const regions = await d3.csv("https://raw.githubusercontent.com/cphalpert/census-regions/7bdc6aa1cb0892361e90ce9ad54983041c2ad015/us%20census%20bureau%20regions%20and%20divisions.csv");
  return new Map(regions.map(d => [d.State, d.Division]));
}
```

Source: https://observablehq.com/@d3/u-s-population-by-state-1790-1990?intent=fork

JavaScript Visualization Tools

What kind of charts do I want to build? Pie charts, maps, lines, bars?

- Some libraries support only a handful of types. Make sure you know which ones you need first.

How large is the dataset?

- Libraries based on SVG are usually better for smaller to medium datasets, as each element is a unique node and exists in the DOM tree. This also means that they offer a lot more flexibility by allowing direct access. Although you could make them work with large data sets with the help of some data aggregation algorithms, smart memory management, and other fancy tricks, going with Canvas-based tools for large datasets is the more reliable option here. Canvas is really fast.

Source: https://www.monterail.com/blog/javascript-libraries-data-visualization/#sixteen

JavaScript Visualization Tools

Is the app used for Web, mobile, or both?

- Some libraries are better at responsiveness, while a few others have their own React Native versions like Victory.

What's the browser support for a given library?

- Check your browser market share to figure this out.

Which JavaScript framework do you use?

- Make sure your data viz library will go well with it. Using React? Going with a React-specific library might be a better option than using wrappers.

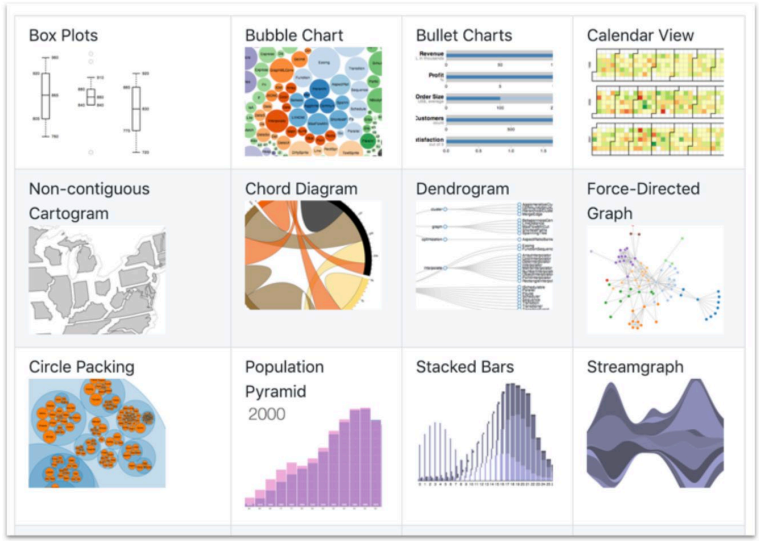
What kind of customization of the look and feel do you need?

- If you need some advanced animations, for example, you should take that into consideration, too.

Source: https://www.monterail.com/blog/javascript-libraries-data-visualization/#sixteen

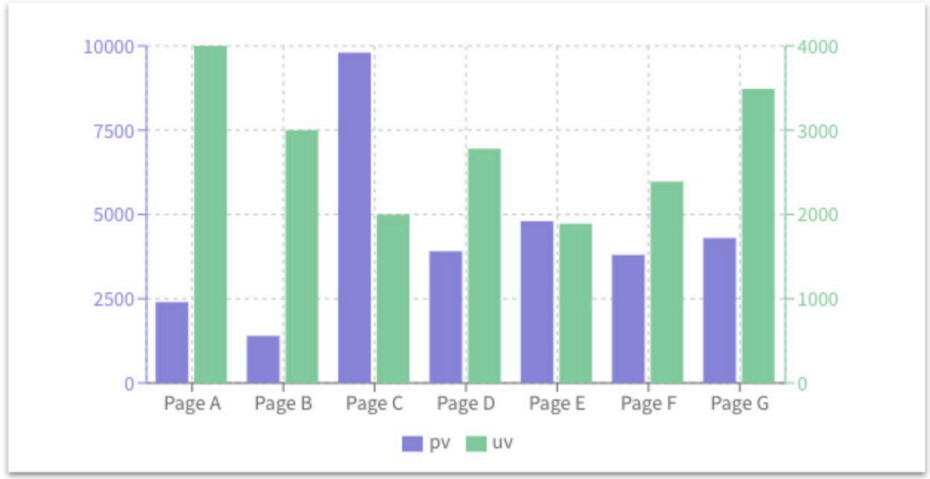
1. D3.js

GitHub Stars: 100,000



2. Recharts

GitHub Stars: 17,700

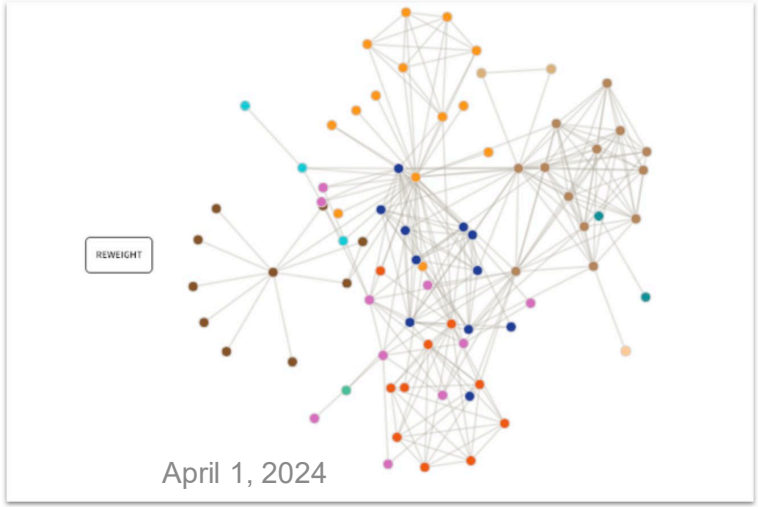


3. Victory

GitHub Stars: 9,300

4. React-vis

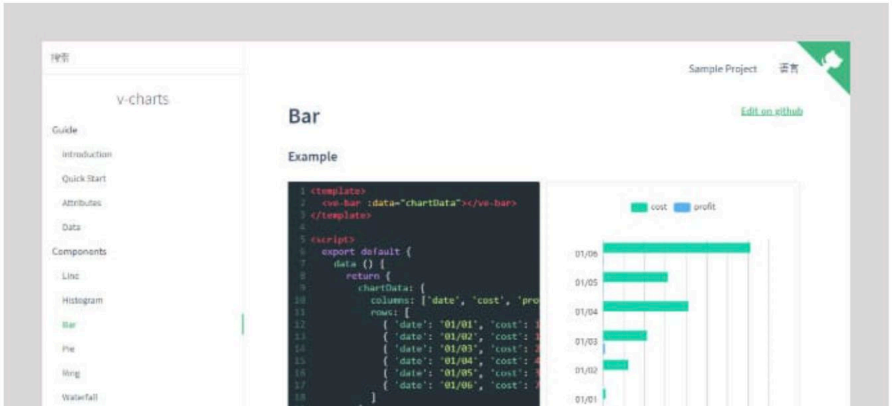
GitHub Stars: 8,100



5. V Charts

GitHub Stars: 6,500

It's a good all-around tool for creating common charts with simple data configuration. It is based on Vue2.x and echarts.



6. Trading Vuejs

GitHub Stars: 1,500

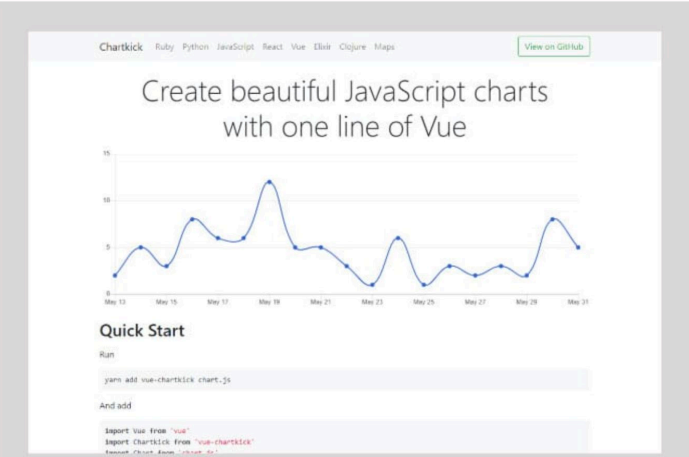
It's an advanced, comprehensive charting system for traders.

Works with: Vue.js

7. Chartkick

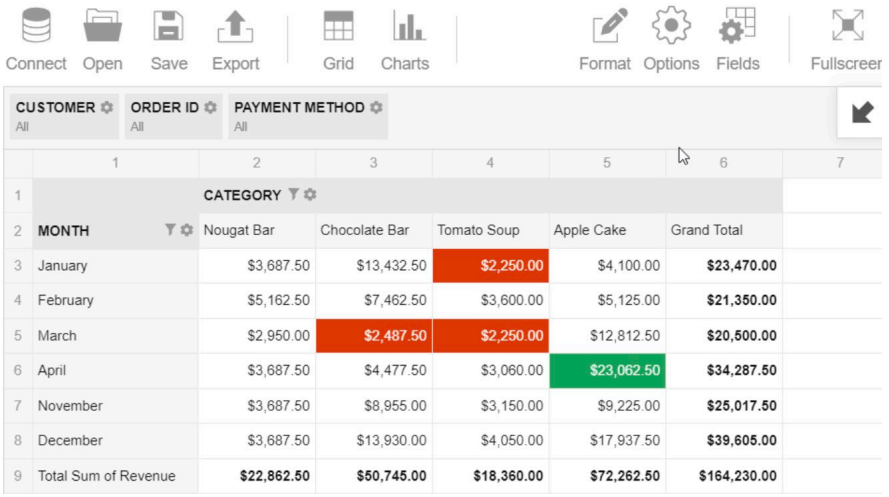
GitHub Stars: 5,800

Chartkick is a great tool not only for Vue. It enables you to generate charts that are functional and aesthetic.



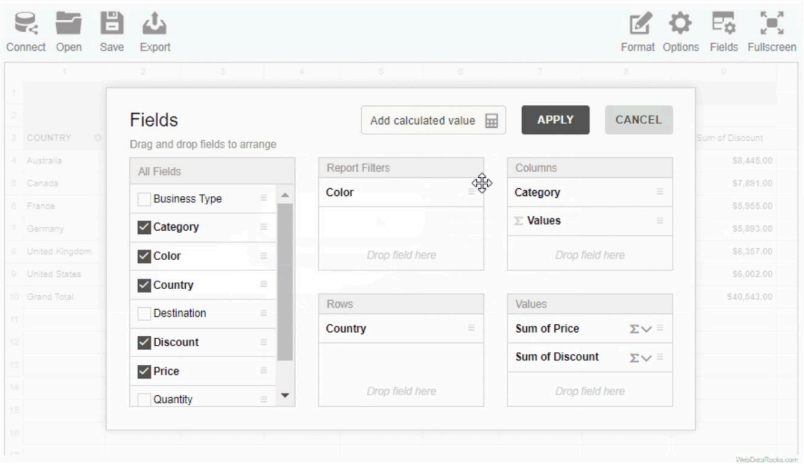
8. Flexmonster

It's a pivot table component for React Native. Great for visualizing business data.



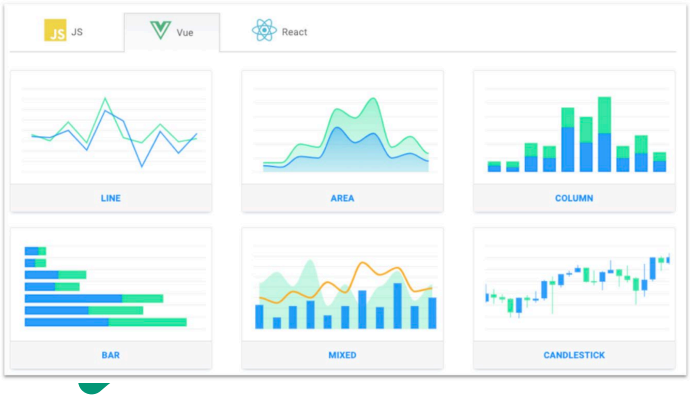
9. Webdatarocks

A JavaScript pivot table component well compatible with React and other frameworks. It's great for data reporting with its aggregating, sorting, and filtering features. It's free to use in your Web browser.



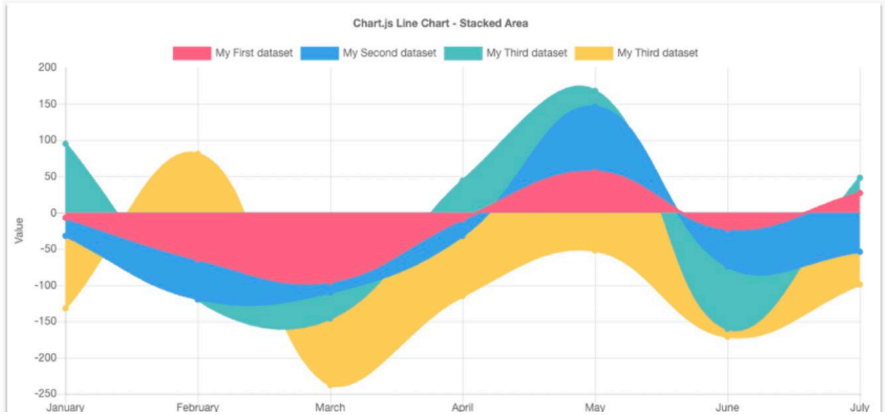
10. ApexCharts

GitHub Stars: 11,100



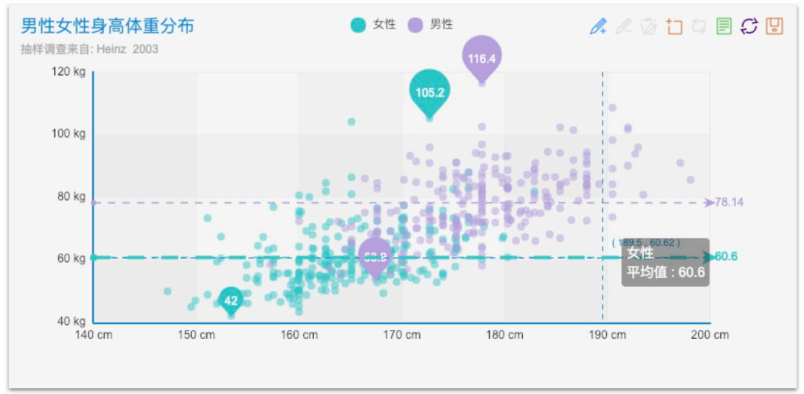
11. Chart.js

GitHub Stars: 56,100



12. Echarts

GitHub Stars: 49,600



Midterm Report

- **Team Size**: 1~3 people
- **Report Duration**: Max. 8 min
- **Contents:**
  - Motivation
  - Target Webpage
  - Target Information
  - Target Statistics
  - Anticipated Results

Source: https://www.crummy.com/software/BeautifulSoup/10.1.jpg

The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw
Web: toodou.github.io

SCAN ME

SCAN ME