データ可視化

情報学科　李新肖

今日のゴール

- データ可視化の位置づけを知る
- Webで可視化とD3.jsの基本を理解する
- 演習でデータ可視化を実践する
- 生成AIでのデータ可視化を学ぶ

データ可視化の位置づけ
UI → 可視化 → データ可視化

- UI (User Interface): 人とコンピュータの接点
- 可視化 (Visualization): 目に見えないものを見えるようにする全般
- データ可視化 (Information Visualization):データを対象とした可視化。主に、大量データを構造的に理解する

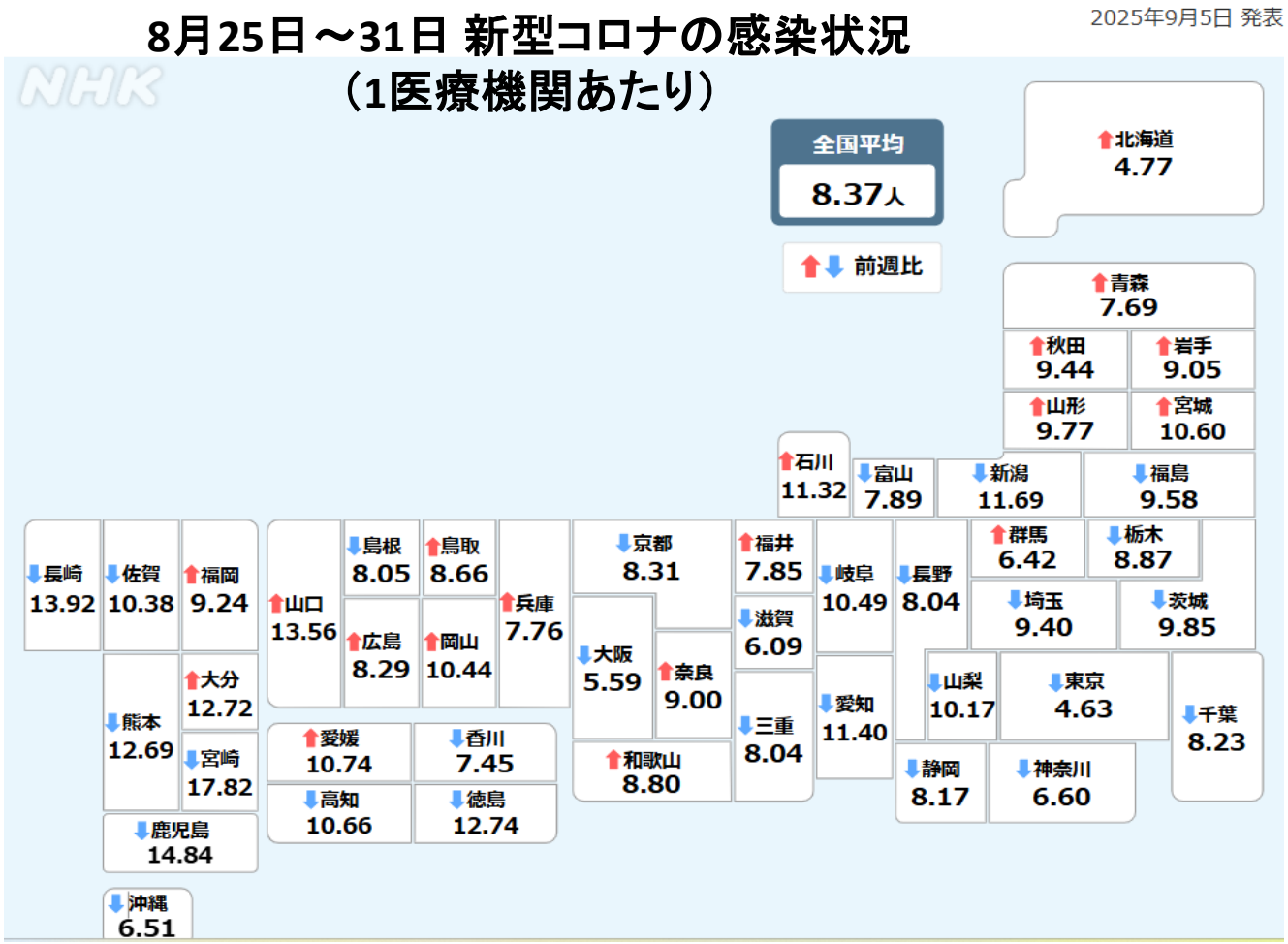
データ可視化の位置づけ

なぜ可視化するのか？

- データの理解を促進する
- 意思決定を支援する
- コミュニケーションを円滑にする

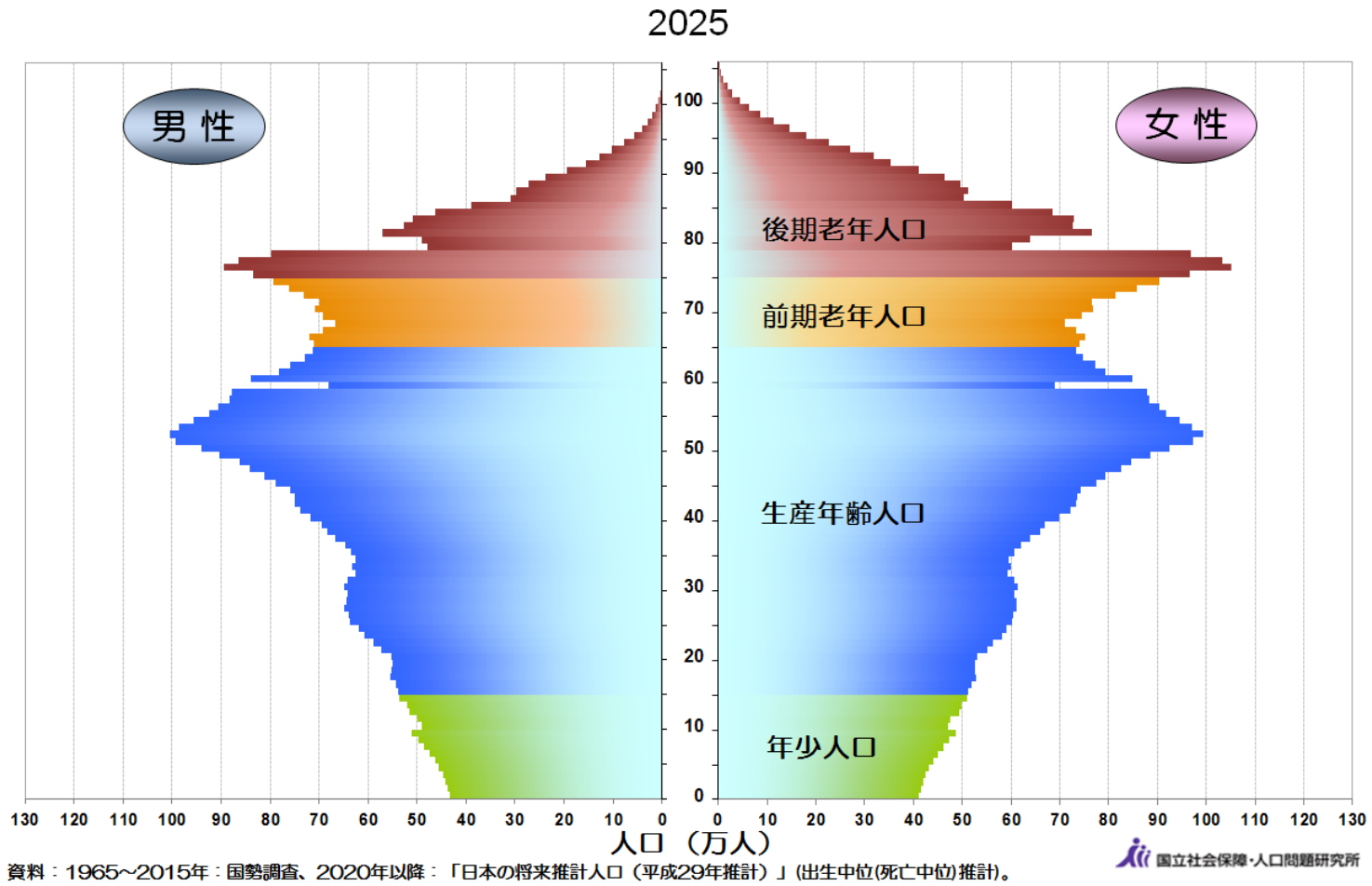
データ可視化の例：地図＋数値

感染者数マップ



https://www3.nhk.or.jp/news/special/coronavirus/data/

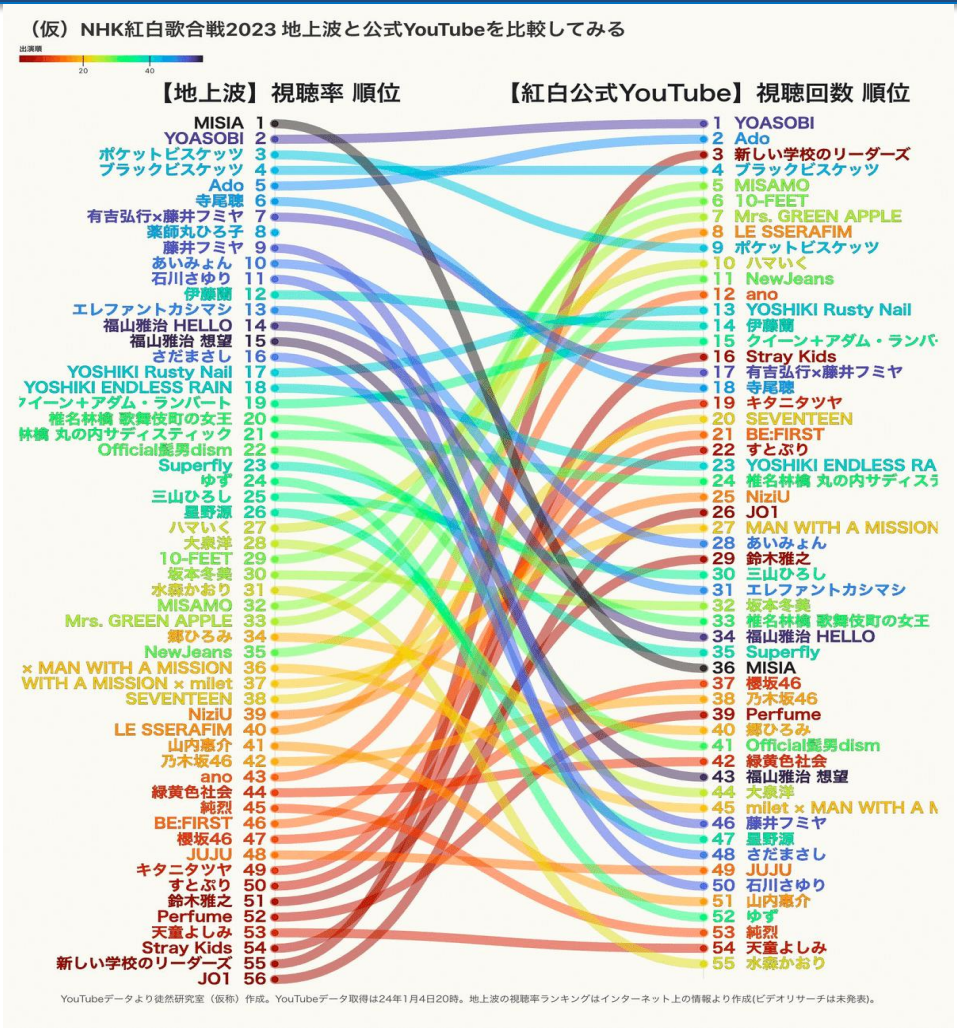
データ可視化の例：棒グラフ
人口ピラミッド



https://www.ipss.go.jp/site-ad/TopPageData/PopPyramid2017_J.html

データ可視化の例：サンキー・ダイアグラム

大規模化するYouTube視聴



https://note.com/tsurezure_cat/n/n4126c4ecca36

Webでデータ可視化

なぜ、Webで可視化する？

- アクセス性と共有性が高い
  - **URLで簡単に共有**
- インタラクティブな表現が可能
  - **動的で操作可能な可視化**
- リアルタイムデータとの連携が容易
  - 最新のデータを即座に可視化

Webでデータ可視化の基本

- JavaScriptでDOM操作
- JavaScriptライブラリで開発と実行を効率化
- D3.jsライブラリを使ってデータ可視化

Webでデータ可視化の基本:
JacascriptでDOM操作（復習）

- (vanilla) JavaScriptでの基本操作:
  - document.querySelector('div')
  - element.textContent = 'Hello'
  - element.style.color = 'red'

  - newElement = document.createElement('p')
  - document.body.appendChild(newElement)

→ ページに動的に要素を追加できる

D3.jsデータ可視化入門

- D3.jsとは
- SVGとCanvasの違い
- D3.jsの基本構造

D3.jsデータ可視化入門

D3.jsとは？

- DOM操作をデータに基づいて自動化
- 複雑なグラフやアニメーションも容易
- 柔軟性が高く研究用途にも使える

  https://d3js.org/ の例を触ってみて

D3.jsデータ可視化入門

D3.jsの基本構造

最小の例（円を並べる）

circle.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">  <!-- 文字エンコーディング -->
    <title>D3 Circle Visualization with Scale</title> <!-- ページタイトル -->
    <script src="https://d3js.org/d3.v7.min.js"></script> <!-- D3.jsの読み込み -->
</head>
<body>
<svg width="400" height="150"></svg>
    <script>
        const radii = [5, 10, 15]; //データ：半径の値
        const svg = d3.select("svg");  // SVG要素を選択
        svg.selectAll("circle") // circle要素を選択
            .data(radii) // データを結びつける
            .enter() // データの数だけ要素を追加
            .append("circle") // circle要素を追加
            .attr("cx", (d, i) => 50 + i * 100)  // x位置をずらして並べる
            .attr("cy", 75)                      // y位置を中央に設定
            .attr("r", d => d)                   // 半径をデータに基づいて設定
            .attr("fill", "steelblue"); // 塗りつぶしの色を設定
    </script>
</body>
</html>
```

→ データを要素に結びつける仕組み

スケールの導入

```
9   <svg width="400" height="150"></svg> <!-- SVG要素を用意 -->
10  <script>
11      const radii = [5, 10, 15]; //データ：半径の値
12      const svg = d3.select("svg");  // SVG要素を選択
13
14      /*半径データの範囲 → 画面上の描画範囲にスケーリング*/
15      const rScale = d3.scaleLinear()
16                       .domain([0, 15])   // データの範囲（例: 0〜100点満点の値など）
17                       .range([0, 45]);    // 画面上の半径は 0〜50px に変換
18
19      svg.selectAll("circle") // circle要素を選択
20          .data(radii) // データを結びつける
21          .enter() // データの数だけ要素を追加
22          .append("circle") // circle要素を追加
23          .attr("cx", (d, i) => 50 + i * 100)  // x位置をずらして並べる
24          .attr("cy", 75)                      // y位置を中央に設定
25          .attr("r", d => rScale(d))           // 半径をスケールで変換
26          .attr("fill", "steelblue"); // 塗りつぶしの色を設定
27
28  </script>
```

座標系と<g>要素

- <g>: グループ化 → グループをまとめて移動する

```
<script>
  const radii = [5, 10, 15]; //データ：半径の値
  const svg = d3.select("svg");  // SVG要素を選択

  /*半径データの範囲 → 画面上の描画範囲にスケーリング*/
  const rScale = d3.scaleLinear()
                   .domain([0, 15])   // データの範囲（例: 0〜100点満点の値など）
                   .range([0, 45]);    // 画面上の半径は 0〜50px に変換

  /*<g> グループを作成（軸と円をまとめる）*/
  const chartGroup = svg.append("g") // g要素を追加
                        .attr("transform", "translate(150,50)"); // グループ全体を右と下にずらす

  chartGroup.selectAll("circle") // circle要素を選択
      .data(radii) // データを結びつける
      .enter() // データの数だけ要素を追加
      .append("circle") // circle要素を追加
      .attr("cx", (d, i) => 50 + i * 100)  // x位置をずらして並べる
      .attr("cy", 75)                      // y位置を中央に設定
      .attr("r", d => rScale(d))           // 半径をスケールで変換
      .attr("fill", "steelblue"); // 塗りつぶしの色を設定
</script>
```

D3.jsの基本図形描画関数一覧（SVG）

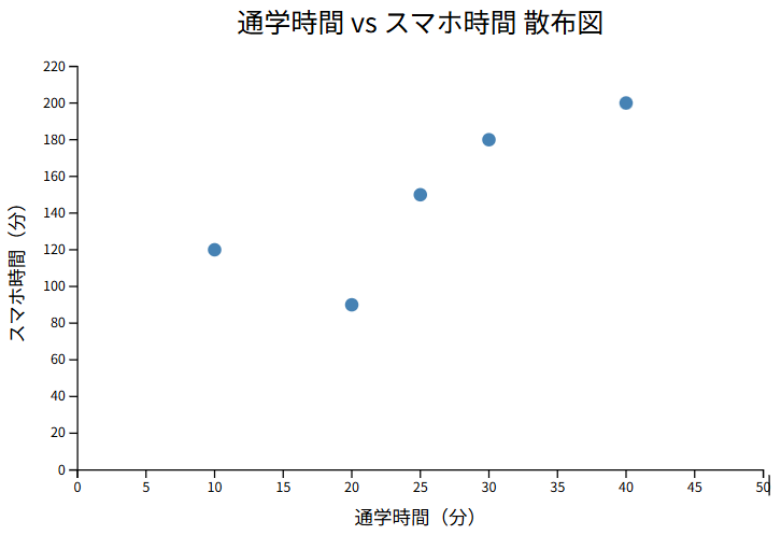
図形	SVG要素名	D3.jsでの主な属性	説明
円	circle	cx, cy, r, fill, stroke	中心座標と半径で円を描画
矩形	rect	x, y, width, height, fill	左上座標とサイズで四角形を描画
線	line	x1, y1, x2, y2, stroke	始点と終点を指定して直線を描画
楕円	ellipse	cx, cy, rx, ry, fill	中心と半径（x方向・y方向）で楕円を描画
多角形	polygon	points, fill, stroke	頂点座標を列挙して多角形を描画
折れ線	polyline	points, fill, stroke	線でつながった複数の点を描画
パス	path	d, fill, stroke	複雑な形状を自由に描画（曲線など）
テキスト	text	x, y, font-size, fill	指定位置に文字列を表示

簡単な散布図の骨組み

```
26        // データ: 通学時間（分） × スマホ時間（分）
27        const data = [
28            { commute: 10, smartphone: 120 },
29            { commute: 30, smartphone: 180 },
30            { commute: 20, smartphone: 90 },
31            { commute: 40, smartphone: 200 },
32            { commute: 25, smartphone: 150 }
33        ];
34        // x軸のスケール
35        const x = d3.scaleLinear()
36            .domain([0, d3.max(data, d => d.commute) + 10])
37            .range([0, width]);
38        // y軸のスケール
39        const y = d3.scaleLinear()
40            .domain([0, d3.max(data, d => d.smartphone) + 20])
41            .range([height, 0]);
42        //散布図点の描画
43        chart.selectAll("circle")
44            .data(data)
45            .enter()
46            .append("circle")
47            .attr("cx", d => x(d.commute))
48            .attr("cy", d => y(d.smartphone))
49            .attr("r", 5);
```

scatterplot.html

- データ: 通学時間 × スマホ時間
- X軸=通学時間,
- Y軸=スマホ時間
- “circle”図形を使用

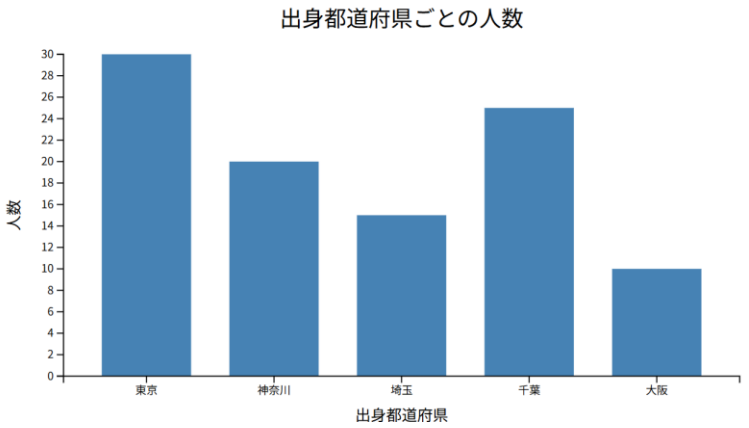


棒グラフの骨組み

```
26      // データ: 出身都道府県と人数
27      const data = [
28          { prefecture: "東京", count: 30 },
29          { prefecture: "神奈川", count: 20 },
30          { prefecture: "埼玉", count: 15 },
31          { prefecture: "千葉", count: 25 },
32          { prefecture: "大阪", count: 10 }
33      ];
34      // x軸のスケール
35      const x = d3.scaleBand()
36          .domain(data.map(d => d.prefecture))
37          .range([0, width])
38          .padding(0.3);// 棒の幅を調整 // 0〜1の値で指定
39
40      // y軸のスケール
41      const y = d3.scaleLinear()
42          .domain([0, d3.max(data, d => d.count)]) // データの最大値まで
43          .nice()// きりの良い数値に調整
44          .range([height, 0]); // yは上が0、下がheightになるので注意
45
46       // 棒（rect要素）の描画
47      chart.selectAll(".bar")
48          .data(data)
49          .enter()
50          .append("rect") // rect要素を追加
51          .attr("class", "bar") // CSSクラスを設定
52          .attr("x", d => x(d.prefecture)) // x位置
53          .attr("y", d => y(d.count)) // y位置
54          .attr("width", x.bandwidth()) // 棒の幅
55          .attr("height", d => height - y(d.count)); // 棒の高さ
```

barchart.html

- データ: 出身都道府県の人数
- X軸=都道府県,
- Y軸=人数
- “rect”図形を使用する



演習：散布図と棒グラフを作成

student_data.csvをダウンロードして使う。

1. 配布テンプレートを開く
2. 自分のデータを入力
3. ブラウザで更新 → グラフ反映
4. 気づいた点をメモ

可視化からの発見を共有

- 隣の人に説明:
  - どんなデータを使ったか？
  - グラフから何がわかったか？（メモ）
- 数人を発表
  - 「図」と「わかったこと」。
- Moodleへ提出（Wordでまとめる）。

生成AI活用で可視化生成

- student_data.csvをアップロード
- 散布図を生成する
  - プロンプト：　通学時間 x スマホ時間の散布図を生成して。
- 散布図のデータ傾向を発見
  - プロンプト：　この散布図からわかることを説明して。

- 棒グラフも同じような手順で結果をAIで生成。

典型的なWeb可視化ライブラリ

項目	D3.js	Chart.js	Plotly.js
柔軟性	◎ 非常に高い（自由に設計可能）	△ 基本的なグラフのみ	○ 多機能でカスタマイズ可能
インタラクション	◎ 自由に実装可能	○ ツールチップなど基本対応	◎ 標準で豊富なインタラクション
学習コスト	高い（低レベルAPI）	低い（設定中心）	中程度（機能は多いが使いやすい）
グラフ種類	自由に定義可能	限定的（棒・折れ線・円など）	多彩（2D・3D・統計・地図など）
描画方式	SVG / Canvas（選択可能）	Canvas（内部的に使用）	SVG / WebGL（自動選択）
用途例	研究・教育・高度な可視化	業務アプリ・教育用ダッシュボード	データ分析・科学技術・Webレポート

D3.jsデータ可視化入門
SVGとCanvasの違い

SVG: ベクター形式、要素ごとに操作可能

```svg
<svg width="300" height="100"></svg>

<script>
const svg = d3.select("svg");

svg.append("circle")
    .attr("cx", 50)
    .attr("cy", 50)
    .attr("r", 30)
    .attr("fill", "steelblue");
</script>

```

Canvas: ピクセル描画、高速だが操作が複雑

```canvas
<canvas width="300" height="100"></canvas>

<script>
const canvas = d3.select("canvas").node();
const ctx = canvas.getContext("2d");

ctx.beginPath();
ctx.arc(50, 50, 30, 0, 2 * Math.PI);
ctx.fillStyle = "steelblue";
ctx.fill();
</script>

```

D3.jsデータ可視化入門

SVGとCanvasの違い

下記データで棒グラフ（バーチャート）を作成して、各ライブラリの違いを比較

```
const categories =
{‘A’: 30, ‘B’: 50, ‘C’:80, ‘D’:120, ‘E’:200, ‘F’:150, ‘G’:80, ‘H’:60, ‘I’:100];
```

- barchart-d3.html
- barchart-plotly.html
- barchart-Chartjs.html

全体共有とまとめ

- データ可視化の意義
- D3.jsでデータ可視化の基本
- 生成AIでのデータ可視化

課題

- students_data.csvを使って、
  勉強・睡眠・スマホ時間・得点を同時に表示するパラレル座標プロットを生成して、そのトレンドをメモして。（生成AI利用可）

プロンプト:

1. 勉強・睡眠・スマホ利用時間・得点を同時に表示するパラレル座標を生成して。
2. パラレル座標のストリーテリングを生成して。

生成例: