

# D3.js で始めるデータビジュアライゼーション

30 のサンプルで基礎から応用まで

河合勝彦

名古屋市立大学大学院経済学研究科

[kkawai@econ.nagoya-cu.ac.jp](mailto:kkawai@econ.nagoya-cu.ac.jp)

2026 年 1 月 9 日



# 目次

<b>第Ⅰ部 D3.js の紹介</b>	5
<b>第1章 D3.js とは</b>	7
1.1 概要 . . . . .	7
1.2 D3.js の読み込み . . . . .	8
1.3 実行環境の準備 . . . . .	8
1.4 基本的な考え方 . . . . .	9
1.5 SVG の基礎 . . . . .	10
1.6 外部データの読み込み . . . . .	11
<b>第Ⅱ部 基礎編</b>	15
<b>第2章 Hello D3.js</b>	17
2.1 学習目標 . . . . .	17
2.2 新しく学ぶ関数・メソッド . . . . .	18
2.3 コード例 . . . . .	18
2.4 ポイント . . . . .	18
2.5 練習問題 . . . . .	19
<b>第3章 SVG の基礎</b>	21
3.1 学習目標 . . . . .	21
3.2 新しく学ぶ関数・メソッド . . . . .	22
3.3 SVG 座標系 . . . . .	22
3.4 練習問題 . . . . .	23
<b>第4章 データバインディング</b>	25
4.1 学習目標 . . . . .	25
4.2 新しく学ぶ関数・メソッド . . . . .	26
4.3 Enter-Update-Exit パターン . . . . .	26
4.4 練習問題 . . . . .	27
<b>第5章 スケール</b>	29

---

5.1	学習目標	29
5.2	新しく学ぶ関数・メソッド	30
5.3	スケールの種類	30
5.4	練習問題	31
<b>第 6 章</b>	<b>軸 (Axes)</b>	33
6.1	学習目標	33
6.2	新しく学ぶ関数・メソッド	34
6.3	練習問題	35
<b>第 7 章</b>	<b>棒グラフ</b>	37
7.1	学習目標	37
7.2	棒グラフの構成要素	37
7.3	練習問題	39
<b>第 8 章</b>	<b>折れ線グラフ</b>	41
8.1	学習目標	41
8.2	新しく学ぶ関数・メソッド	42
8.3	練習問題	43
<b>第 9 章</b>	<b>円グラフ</b>	45
9.1	学習目標	45
9.2	新しく学ぶ関数・メソッド	46
9.3	練習問題	47
<b>第 10 章</b>	<b>トランジション</b>	49
10.1	学習目標	49
10.2	新しく学ぶ関数・メソッド	50
10.3	練習問題	51
<b>第 11 章</b>	<b>イベント処理</b>	53
11.1	学習目標	53
11.2	新しく学ぶ関数・メソッド	54
11.3	主なイベントタイプ	54
11.4	練習問題	55
<b>第 III 部</b>	<b>中級編</b>	57
<b>第 12 章</b>	<b>散布図</b>	59
12.1	学習目標	59
12.2	散布図の特徴	59
12.3	練習問題	60

<b>第 13 章 エリアチャート</b>	63
13.1 学習目標	63
13.2 新しく学ぶ関数・メソッド	64
13.3 練習問題	64
<b>第 14 章 複数折れ線グラフ</b>	67
14.1 学習目標	67
14.2 複数系列の扱い方	67
14.3 練習問題	68
<b>第 15 章 積み上げ棒グラフ</b>	71
15.1 学習目標	71
15.2 新しく学ぶ関数・メソッド	72
15.3 練習問題	72
<b>第 16 章 ドーナツグラフ</b>	75
16.1 学習目標	75
16.2 実装のポイント	75
16.3 練習問題	76
<b>第 17 章 ツールチップ</b>	77
17.1 学習目標	77
17.2 練習問題	77
<b>第 18 章 レスポンシブチャート</b>	79
18.1 学習目標	79
18.2 練習問題	79
<b>第 19 章 データ更新</b>	81
19.1 学習目標	81
19.2 練習問題	81
<b>第 20 章 ズーム機能</b>	83
20.1 学習目標	83
20.2 新しく学ぶ関数・メソッド	84
20.3 練習問題	84
<b>第 21 章 ブラッシング</b>	87
21.1 学習目標	87
21.2 新しく学ぶ関数・メソッド	88
21.3 練習問題	88

---

<b>第Ⅳ部 上級編</b>	91
<b>第22章 フォースグラフ</b>	93
22.1 学習目標 .....	93
22.2 新しく学ぶ関数・メソッド .....	94
22.3 練習問題 .....	94
<b>第23章 ツリーマップ</b>	97
23.1 学習目標 .....	97
23.2 練習問題 .....	97
<b>第24章 階層ツリー</b>	99
24.1 学習目標 .....	99
24.2 練習問題 .....	99
<b>第25章 サークルパッキング</b>	101
25.1 学習目標 .....	101
25.2 練習問題 .....	101
<b>第26章 サンバースト図</b>	103
26.1 学習目標 .....	103
26.2 練習問題 .....	103
<b>第27章 コードダイアグラム</b>	105
27.1 学習目標 .....	105
27.2 練習問題 .....	105
<b>第28章 世界地図</b>	107
28.1 学習目標 .....	107
28.2 練習問題 .....	107
<b>第29章 ヒートマップ</b>	109
29.1 学習目標 .....	109
29.2 練習問題 .....	109
<b>第30章 レーダーチャート</b>	111
30.1 学習目標 .....	111
30.2 練習問題 .....	111
<b>第31章 インタラクティブダッシュボード</b>	113
31.1 学習目標 .....	113
31.2 ダッシュボードの構成要素 .....	113
31.3 練習問題 .....	114

---

<b>第Ⅴ部 LLM を活用した学習</b>	<b>115</b>
<b>第32章 最短で上達するプロンプトセット 30本</b>	<b>117</b>
32.1 ★1:D3の書き方に慣れる（#01～#06）	119
32.2 ★2:実用の入口（#07～#12）	126
32.3 ★3:更新設計を覚える（#13～#18）	133
32.4 ★4:高度な操作と複合（#19～#24）	140
32.5 ★5:実務品質（#25～#30）	147
32.6 プロンプト活用のコツ	154
<b>参考文献・参考サイト</b>	<b>157</b>



## 第Ⅰ部

### D3.js の紹介



# 第1章

## D3.js とは

### 1.1 概要

D3.js (Data-Driven Documents) は、データに基づいてドキュメントを操作するための JavaScript ライブラリです。Mike Bostock によって開発され、Web ブラウザ上でインタラクティブなデータビジュアライゼーションを作成するための強力なツールを提供します。

#### D3.js の特徴

- **データ駆動**：データと DOM 要素を結びつける強力なバインディング機能
- **差分更新**：enter/update/exit（または join）パターンでデータの変化に応じた DOM 操作を組み立てられる
- **トランジション**：.transition() で滑らかなアニメーションを付けられる
- **柔軟性**：SVG、Canvas、HTML を使用した自由なビジュアライゼーション
- **豊富な機能**：スケール、軸、レイアウト、地理投影など多数の機能を提供
- **Web 標準**：標準的な Web 技術 (HTML、CSS、SVG) を活用

#### 本書のサンプルコード

本書で解説するサンプルコード（30 個の HTML ファイル）は、以下の GitHub リポジトリから入手できます：

<https://github.com/kkawailab/kklab-D3-samples>

各サンプルはブラウザで直接開いて動作を確認できます。コードを読みながら実際に動かすことで、理解が深まります。

**サンプル番号と章番号について**：本章（第1章）は導入のためサンプルがありません。そのため、サンプル 01 は第2章、サンプル 02 は第3章…というように、サンプル番号は章番号より 1 つ小さい値になります。各章の図のキャプションにサンプル番号を記載していますので、参照してください。

## 1.2 D3.js の読み込み

D3.js は CDN (Content Delivery Network) から簡単に読み込むことができます。

Listing 1.1 D3.js v7 の読み込み

```
1 <script src="https://d3js.org/d3.v7.min.js"></script>
```

## 1.3 実行環境の準備

D3.js を使った開発を始める前に、実行環境を整えましょう。

### 1.3.1 推奨環境

- エディタ：Visual Studio Code (Live Server 拡張機能が便利)
- ブラウザ：Chrome、Firefox、Safari、Edge (開発者ツール付き)
- ローカルサーバー：外部データ読み込み時に必要

Listing 1.2 ローカルサーバーの起動方法

```
1 # Node.js を使う場合
2 npx serve .
3
4 # Python を使う場合
5 python -m http.server 8000
```

### 1.3.2 基本テンプレート

以後の章では、以下のテンプレートを前提としてコードを記述します。このテンプレートをベースにすることで、各章のコード例をそのまま試すことができます。

Listing 1.3 共通テンプレート (index.html)

```
1 <!DOCTYPE html>
2 <html lang="ja">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>D3.js サンプル</title>
7     <style>
8       #chart { max-width: 800px; margin: 20px auto; }
9     </style>
10  </head>
```

```
11 <body>
12   <div id="chart"></div>
13   <script src="https://d3js.org/d3.v7.min.js"></script>
14   <script>
15     // マージン設定（軸用のスペースを確保）
16     const margin = {top: 30, right: 20, bottom: 40, left: 50};
17     const width = 600 - margin.left - margin.right;
18     const height = 400 - margin.top - margin.bottom;
19
20     // SVGを作成
21     const svg = d3.select("#chart")
22       .append("svg")
23       .attr("width", width + margin.left + margin.right)
24       .attr("height", height + margin.top + margin.bottom);
25
26     // 描画用グループ（マージンを適用）
27     const g = svg.append("g")
28       .attr("transform", `translate(${margin.left},${margin.top})`);
29
30     // ここにチャートを描画するコードを記述
31   </script>
32 </body>
33 </html>
```

### コード例について

本書のコード例は、紙面の都合上、上記テンプレートの// ここにチャートを描画の部分のみを抜粋して掲載しています。svg、g、width、height、marginなどの変数は、すでに上記テンプレートで定義されているものとして扱います。

viewport メタタグはスマートフォンや縮小表示時の挙動を安定させます。レスポンシブ対応（第18章）を学ぶ際に重要になります。

## 1.4 基本的な考え方

D3.jsは以下の3つの基本概念に基づいています。

### 1.4.1 セレクション (Selection)

DOM要素を選択し、操作するための仕組みです。jQueryに似ていますが、データバインディングに特化しています。

```

1 d3.select("body")
2
3 // 複数要素の選択
4 d3.selectAll("div")

```

### 1.4.2 データバインディング (Data Binding)

データを DOM 要素に結びつける機能です。Enter-Update-Exit パターンによって、データの変更に応じた要素の追加・更新・削除を管理します。

```

1 .data(dataset)      // データをバインド
2 .enter()            // 新しい要素用
3 .append("rect")     // 要素を追加

```

### 1.4.3 メソッドチェーン (Method Chaining)

複数のメソッドを連続して呼び出すことで、簡潔なコードを記述できます。

```

4 .append("circle")
5 .attr("cx", 50)
6 .attr("cy", 50)
7 .attr("r", 25)
8 .style("fill", "blue");

```

## 1.5 SVG の基礎

D3.js では主に SVG (Scalable Vector Graphics) を使用してグラフを描画します。

表 1.1 SVG の基本図形

要素	説明	主な属性
<rect>	長方形	x, y, width, height, rx, ry
<circle>	円	cx, cy, r
<ellipse>	楕円	cx, cy, rx, ry
<line>	線	x1, y1, x2, y2
<path>	パス	d
<text>	テキスト	x, y, text-anchor
<g>	グループ	transform

## 1.6 外部データの読み込み

実務ではデータをコード内に直接書くのではなく、外部ファイルから読み込むのが一般的です。D3.js は CSV や JSON ファイルを簡単に読み込むための関数を提供しています。

### データ読み込み関数

```
d3.csv(url)
    CSV ファイルを読み込み、オブジェクトの配列として返します。
d3.json(url)
    JSON ファイルを読み込みます。
d3.text(url)
    プレーンテキストとして読み込みます。
d3.dsv(delimiter, url)
    任意の区切り文字 (TSV など) で読み込みます。
```

### 1.6.1 CSV ファイルの読み込み

Listing 1.4 CSV ファイルの読み込み

```
1 // data.csv の内容 :
2 // name,value
3 // A,30
4 // B,45
5 // C,25
6
7 d3.csv("data.csv").then(data => {
8     console.log(data);
9     // [{name: "A", value: "30"}, {name: "B", value: "45"}, ...]
10
11    // 注意： 数値も文字列として読み込まれる
12    // 数値に変換するには以下のようにする
13    data.forEach(d => {
14        d.value = +d.value; // "30" -> 30
15    });
16
17    // ここでチャートを描画
18    drawChart(data);
19});
```

### 1.6.2 自動型変換 (autoType)

Listing 1.5 autoType による自動型変換

```

1 // 数値や日付を自動的に変換
2 d3.csv("data.csv", d3.autoType).then(data => {
3   console.log(data);
4   // [{name: "A", value: 30}, ...] // valueが数値になる
5 });

```

### 1.6.3 日付のパース

日付データを扱う場合は、`d3.timeParse` を使用して日付オブジェクトに変換します。

Listing 1.6 日付のパース

```

1 // 日付フォーマットを指定してパーサを作成
2 const parseDate = d3.timeParse("%Y-%m-%d");
3
4 d3.csv("sales.csv").then(data => {
5   data.forEach(d => {
6     d.date = parseDate(d.date); // "2024-01-15" -> Date object
7     d.sales = +d.sales;
8   });
9 });

```

#### 日付フォーマット指定子

%Y	4桁の年 (2024)
%m	2桁の月 (01-12)
%d	2桁の日 (01-31)
%H	時 (00-23)
%M	分 (00-59)
%S	秒 (00-59)

### 1.6.4 エラーハンドリング

Listing 1.7 エラーハンドリング

```

1 d3.csv("data.csv")
2   .then(data => {
3     // 成功時の処理
4     drawChart(data);
5   })
6   .catch(error => {
7     // エラー時の処理
8   });

```

```
8     console.error("データの読み込みに失敗:", error);
9 );
```

### 注意：ローカルサーバーが必要

外部ファイルの読み込みはセキュリティ上の制約（CORS）があるため、HTML ファイルを直接ブラウザで開いても動作しません。必ずローカルサーバー（`npx serve .` や `python -m http.server`）を使用してください。



第 II 部

基礎編



## 第2章

# Hello D3.js

### 2.1 学習目標

この章では、D3.js の最も基本的な使い方を学びます。DOM 要素の選択と操作の基礎を理解しましょう。

The screenshot shows a web application interface. At the top, a header reads "01 - Hello D3.js". Below it, a box contains the text "学習ポイント: D3.jsの基本的なDOM操作 - 要素の選択、追加、テキスト設定、スタイル適用". The main content area has a title "D3.jsへようこそ！" followed by the text "D3.js (Data-Driven Documents) は、データに基づいてDOMを操作するJavaScriptライブラリです。". Below this, there is a list of greetings in various languages:

こんにちは
Hello
Bonjour
你好
مرحبا

図 2.1 サンプル 01 の実行結果

## 2.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
d3.select(selector)
```

CSS セレクタに一致する最初の要素を選択します。

```
d3.selectAll(selector)
```

CSS セレクタに一致するすべての要素を選択します。

```
selection.text(value)
```

要素のテキスト内容を設定または取得します。

```
selection.style(name, value)
```

要素の CSS スタイルを設定します。

```
selection.append(type)
```

選択した要素の子として新しい要素を追加します。

## 2.3 コード例

Listing 2.1 基本的な DOM 操作

```
1 // 要素の選択とテキスト変更
2 d3.select("#message")
3   .text("Hello, D3.js!")
4   .style("color", "blue")
5   .style("font-size", "24px");
6
7 // 新しい要素の追加
8 d3.select("body")
9   .append("p")
10  .text("D3.jsで追加されました");
```

## 2.4 ポイント

- d3.select() は jQuery の \$() に似た機能
- メソッドチェーンで複数の操作を連続実行
- 戻り値は選択されたオブジェクト（セレクション）

## 2.5 練習問題

### 練習問題

**問題 1：**ID が title の要素を選択し、テキストを「D3.js へようこそ」に変更し、文字色を赤、フォントサイズを 32px に設定するコードを書いてください。

**問題 2：**body 要素に新しい <h2> 要素を追加し、テキストを「サブタイトル」、背景色を黄色に設定するコードを書いてください。

**問題 3：**クラス名が item のすべての要素を選択し、文字色を緑に変更するコードを書いてください。

### 模範解答

**問題 1 の解答：**

```
9   .text("D3.js へようこそ")
10  .style("color", "red")
11  .style("font-size", "32px");
```

**問題 2 の解答：**

```
12  .append("h2")
13  .text("サブタイトル")
14  .style("background-color", "yellow");
```

**問題 3 の解答：**

```
15  .style("color", "green");
```



## 第3章

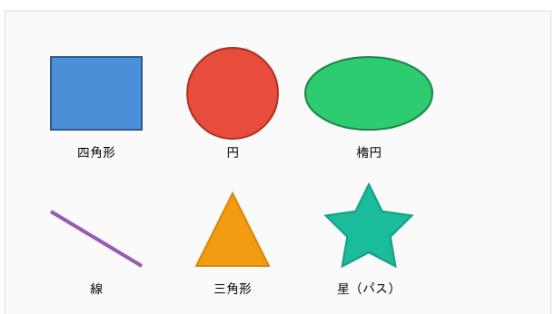
# SVG の基礎

### 3.1 学習目標

SVG 要素を使った基本的な図形の描画方法を学びます。

### 02 - SVGの基本

学習ポイント: D3.jsでSVG要素（円、四角形、線、テキスト）を描画する方法



The image displays six basic SVG shapes: a blue square labeled '四角形' (square), a red circle labeled '円' (circle), a green ellipse labeled '楕円' (elliptical), a purple line segment labeled '線' (line), an orange triangle labeled '三角形' (triangle), and a teal five-pointed star labeled '星 (パス)' (star (path)). Below these shapes is the text 'SVG基本图形' (Basic SVG shapes).

図 3.1 サンプル 02 の実行結果

## 3.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
selection.attr(name, value)
```

要素の属性を設定または取得します。SVG 要素のサイズや位置の設定に使用。

```
selectionclassed(names, value)
```

CSS クラスの追加・削除を行います。

## 3.3 SVG 座標系

SVG の座標系は左上が原点 (0,0) で、右方向が X 軸の正、下方向が Y 軸の正となります。

Listing 3.1 SVG 要素の作成と図形の描画

```
1 // SVGコンテナの作成
2 const svg = d3.select("#chart")
3   .append("svg")
4   .attr("width", 400)
5   .attr("height", 300);
6
7 // 長方形の描画
8 svg.append("rect")
9   .attr("x", 50)
10  .attr("y", 50)
11  .attr("width", 100)
12  .attr("height", 80)
13  .attr("fill", "#3498db");
14
15 // 円の描画
16 svg.append("circle")
17  .attr("cx", 250)
18  .attr("cy", 100)
19  .attr("r", 40)
20  .attr("fill", "#e74c3c");
```

## 3.4 練習問題

### 練習問題

**問題 1：**幅 500px、高さ 400px の SVG を作成し、その中に以下の図形を描画してください：

- 位置 (100, 100) に半径 50px の青い円
- 位置 (250, 50) から幅 150px、高さ 100px の緑の長方形

**問題 2：**楕円 (ellipse) を作成してください。中心座標 (200, 200)、X 方向の半径 80px、Y 方向の半径 40px、色はオレンジ。

**問題 3：**線 (line) を描画してください。始点 (50, 50) から終点 (350, 250) まで、線の太さ 3px、色は紫。

## 模範解答

### 問題1の解答：

```
1  .append("svg")
2  .attr("width", 500)
3  .attr("height", 400);
4
5  svg.append("circle")
6  .attr("cx", 100)
7  .attr("cy", 100)
8  .attr("r", 50)
9  .attr("fill", "blue");
10
11 svg.append("rect")
12 .attr("x", 250)
13 .attr("y", 50)
14 .attr("width", 150)
15 .attr("height", 100)
16 .attr("fill", "green");
```

### 問題2の解答：

```
1  .attr("cx", 200)
2  .attr("cy", 200)
3  .attr("rx", 80)
4  .attr("ry", 40)
5  .attr("fill", "orange");
```

### 問題3の解答：

```
6  .attr("x1", 50)
7  .attr("y1", 50)
8  .attr("x2", 350)
9  .attr("y2", 250)
10 .attr("stroke", "purple")
11 .attr("stroke-width", 3);
```

## 第4章

# データバインディング

### 4.1 学習目標

D3.js の核心機能であるデータバインディングを学びます。配列データを DOM 要素に結びつける方法を理解しましょう。

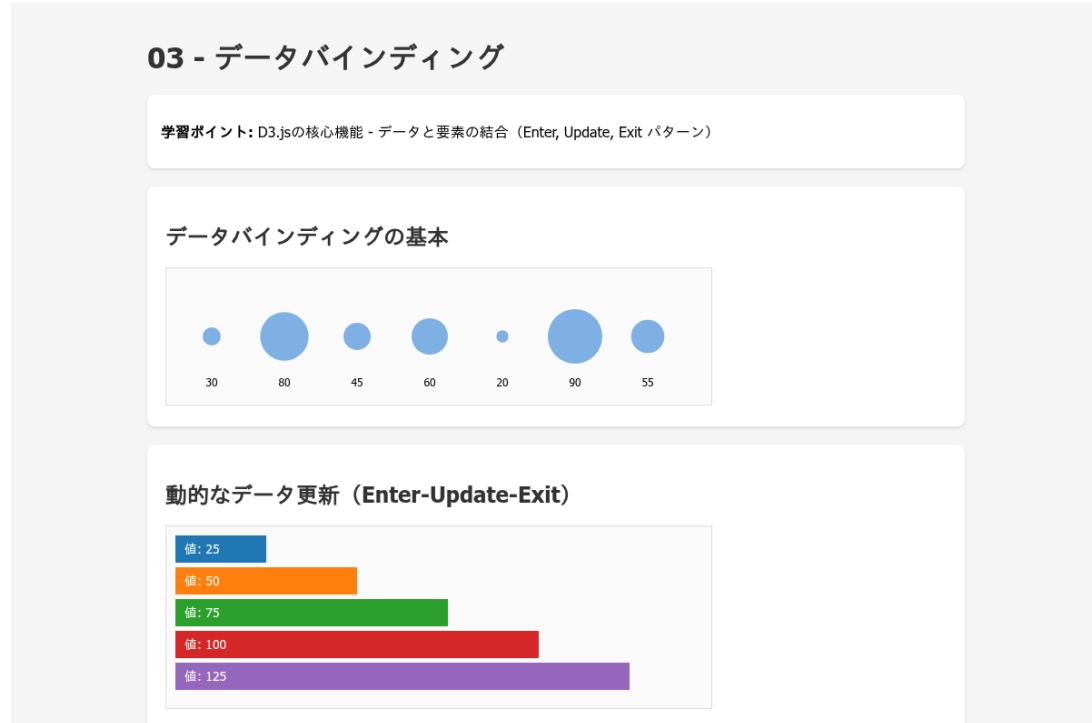


図 4.1 サンプル 03 の実行結果

## 4.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
selection.data(array)
    配列データを選択した要素にバインドします。
selection.enter()
    データに対応する要素がない場合のプレースホルダを返します。
selection.exit()
    データに対応しなくなった要素を返します。
selection.join(enter, update, exit)
    Enter-Update-Exit パターンを簡潔に記述します (D3 v5 以降)。
```

## 4.3 Enter-Update-Exit パターン

データの変更に応じて要素を管理する3つのフェーズ：

1. **Enter**：新しいデータに対して要素を追加
2. **Update**：既存の要素を更新
3. **Exit**：不要になった要素を削除

Listing 4.1 データバインディングの基本

```
1 const data = [30, 50, 80, 40, 60];
2
3 // データをバインドして棒を作成
4 svg.selectAll("rect")
5   .data(data)
6   .enter()
7   .append("rect")
8   .attr("x", (d, i) => i * 60) // d: データ値, i: インデックス
9   .attr("y", d => 100 - d)
10  .attr("width", 50)
11  .attr("height", d => d)
12  .attr("fill", "#3498db");
```

## 4.4 練習問題

### 練習問題

**問題 1：**配列 [20, 45, 70, 35, 90] をバインドして、5つの円を横一列に描画してください。各円の半径はデータ値の半分 ( $d/2$ ) とします。

**問題 2：**配列 ["赤", "青", "緑", "黄"] をバインドして、4つの長方形を描画し、それぞれの色を配列の値に設定してください。

**問題 3：**データの値に応じて高さが変わる縦棒グラフを作成してください。データは [100, 150, 80, 200, 120] です。

## 模範解答

### 問題1の解答：

```
17
18 svg.selectAll("circle")
19   .data(data)
20   .enter()
21   .append("circle")
22   .attr("cx", (d, i) => 50 + i * 80)
23   .attr("cy", 100)
24   .attr("r", d => d / 2)
25   .attr("fill", "#3498db");
```

### 問題2の解答：

```
26
27 svg.selectAll("rect")
28   .data(colors)
29   .enter()
30   .append("rect")
31   .attr("x", (d, i) => i * 60)
32   .attr("y", 50)
33   .attr("width", 50)
34   .attr("height", 50)
35   .attr("fill", d => d);
```

### 問題3の解答：

```
36 const height = 250;
37
38 svg.selectAll("rect")
39   .data(data)
40   .enter()
41   .append("rect")
42   .attr("x", (d, i) => i * 60)
43   .attr("y", d => height - d)
44   .attr("width", 50)
45   .attr("height", d => d)
46   .attr("fill", "#e74c3c");
```

# 第5章

## スケール

### 5.1 学習目標

データの値をピクセル座標に変換するスケール関数の使い方を学びます。

### 04 - スケール (Scale)

学習ポイント: D3.jsのスケール機能 - データ値を視覚的な値（位置、サイズ、色）に変換

#### 1. Linear Scale (線形スケール)

```
d3.scaleLinear().domain([0, 100]).range([0, 500])  
データ値 0~100 をピクセル 0~500 に変換
```

#### 2. Band Scale (バンドスケール)

```
d3.scaleBand().domain(['りんご', 'バナナ', 'オレンジ', 'ぶどう', 'いちご']).range([0, 500])  
データ名をピクセル位置に変換
```

図 5.1 サンプル 04 の実行結果

## 5.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
d3.scaleLinear()
    連続的な数値データ用の線形スケールを作成します。
scale.domain([min, max])
    入力データの範囲（定義域）を設定します。
scale.range([min, max])
    出力値の範囲（値域）を設定します。
d3.scaleBand()
    カテゴリデータ用のバンドスケールを作成します。棒グラフに最適。
d3.scaleOrdinal()
    離散値から離散値へのマッピングを行う序数スケール。
d3.max(array, accessor)
    配列の最大値を返します。
d3.min(array, accessor)
    配列の最小値を返します。
```

## 5.3 スケールの種類

表 5.1 主なスケールの種類

スケール	用途	入力→出力
scaleLinear	連続数値	連続→連続
scaleBand	カテゴリ（棒グラフ）	離散→連続
scaleOrdinal	カテゴリ（色など）	離散→離散
scaleTime	日時データ	日時→連続
scaleLog	対数スケール	連続→連続

Listing 5.1 スケールの使用例

```
1 // 線形スケール
2 const xScale = d3.scaleLinear()
3   .domain([0, 100])      // データの範囲: 0~100
4   .range([0, 400]);     // ピクセルの範囲: 0~400px
5
6 // バンドスケール
7 const yScale = d3.scaleBand()
8   .domain(["A", "B", "C", "D"])
9   .range([0, 300])
```

```
10   .padding(0.1);           // 棒の間隔
11
12 // 使用例
13 const x = xScale(50);    // 200を返す
14 const y = yScale("B");   // 対応するy座標を返す
```

## 5.4 練習問題

### 練習問題

**問題 1：**0～1000 のデータ範囲を 0～500px のピクセル範囲に変換する線形スケールを作成し、値 750 を変換した結果を求めてください。

**問題 2：**カテゴリ ["春", "夏", "秋", "冬"] に対して、幅 400px の領域で等間隔に配置するバンドスケールを作成し、padding を 0.2 に設定してください。

**問題 3：**配列 [15, 82, 47, 93, 28] の最大値と最小値を取得し、それを domain に設定した線形スケールを作成してください。range は [0, 300] とします。

### 模範解答

**問題 1 の解答：**

```
47   .domain([0, 1000])
48   .range([0, 500]);
49
50 const result = scale(750); // 375を返す
```

**問題 2 の解答：**

```
51   .domain(["春", "夏", "秋", "冬"])
52   .range([0, 400])
53   .padding(0.2);
```

**問題 3 の解答：**

```
54
55 const scale = d3.scaleLinear()
56   .domain([d3.min(data), d3.max(data)]) // [15, 93]
57   .range([0, 300]);
```



# 第6章

## 軸 (Axes)

### 6.1 学習目標

グラフに軸を追加する方法を学びます。軸はデータの読み取りに不可欠な要素です。

### 05 - 軸 (Axes)

学習ポイント: D3.jsで軸を作成・カスタマイズする方法 - axisBottom, axisLeft, axisRight, axisTop

#### 基本的な軸の種類

#### カスタマイズされた軸 (グリッド線付き)

図 6.1 サンプル 05 の実行結果

## 6.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

`d3.axisBottom(scale)`

下向きの水平軸を作成します。

`d3.axisLeft(scale)`

左向きの垂直軸を作成します。

`d3.axisTop(scale)`

上向きの水平軸を作成します。

`d3.axisRight(scale)`

右向きの垂直軸を作成します。

`axis.ticks(count)`

目盛りの数を設定します。

`axis.tickFormat(format)`

目盛りラベルのフォーマットを設定します。

`selection.call(function)`

セレクションに対して関数を呼び出します。軸の描画に使用。

Listing 6.1 軸の作成と描画

```
1 // スケールを定義
2 const xScale = d3.scaleLinear()
3   .domain([0, 100])
4   .range([0, width]);
5
6 // 軸ジェネレータを作成
7 const xAxis = d3.axisBottom(xScale)
8   .ticks(10)
9   .tickFormat(d => d + "%");
10
11 // 軸をSVGに描画
12 svg.append("g")
13   .attr("transform", `translate(0, ${height})`)
14   .call(xAxis);
```

### 6.3 練習問題

#### 練習問題

**問題 1：**0～500 の範囲を持つ線形スケールを使って、下向きの X 軸を作成してください。目盛りは 5 つとし、グラフ下端 ( $y=300$ ) に配置してください。

**問題 2：**0～100 の範囲を持つ Y 軸を作成し、目盛りラベルに「点」という単位を付けてください。(例：「80 点」)

**問題 3：**X 軸と Y 軸の両方を持つ座標系を作成してください。マージンは上下左右 50px とします。

## 模範解答

### 問題1の解答：

```
58   .domain([0, 500])
59   .range([0, 400]);
60
61 const xAxis = d3.axisBottom(xScale)
62   .ticks(5);
63
64 svg.append("g")
65   .attr("transform", "translate(50, 300)")
66   .call(xAxis);
```

### 問題2の解答：

```
67   .domain([0, 100])
68   .range([250, 0]);
69
70 const yAxis = d3.axisLeft(yScale)
71   .tickFormat(d => d + "点");
72
73 svg.append("g")
74   .attr("transform", "translate(50, 50)")
75   .call(yAxis);
```

### 問題3の解答：

```
76 const width = 400, height = 300;
77
78 const xScale = d3.scaleLinear()
79   .domain([0, 100]).range([0, width]);
80 const yScale = d3.scaleLinear()
81   .domain([0, 100]).range([height, 0]);
82
83 const xAxisY = margin.top + height;
84 svg.append("g")
85   .attr("transform", `translate(${margin.left}, ${xAxisY})`)
86   .call(d3.axisBottom(xScale));
87
88 svg.append("g")
89   .attr("transform", `translate(${margin.left}, ${margin.top})`)
90   .call(d3.axisLeft(yScale));
```

## 第7章

# 棒グラフ

### 7.1 学習目標

これまでの知識を活用して、実用的な棒グラフを作成します。

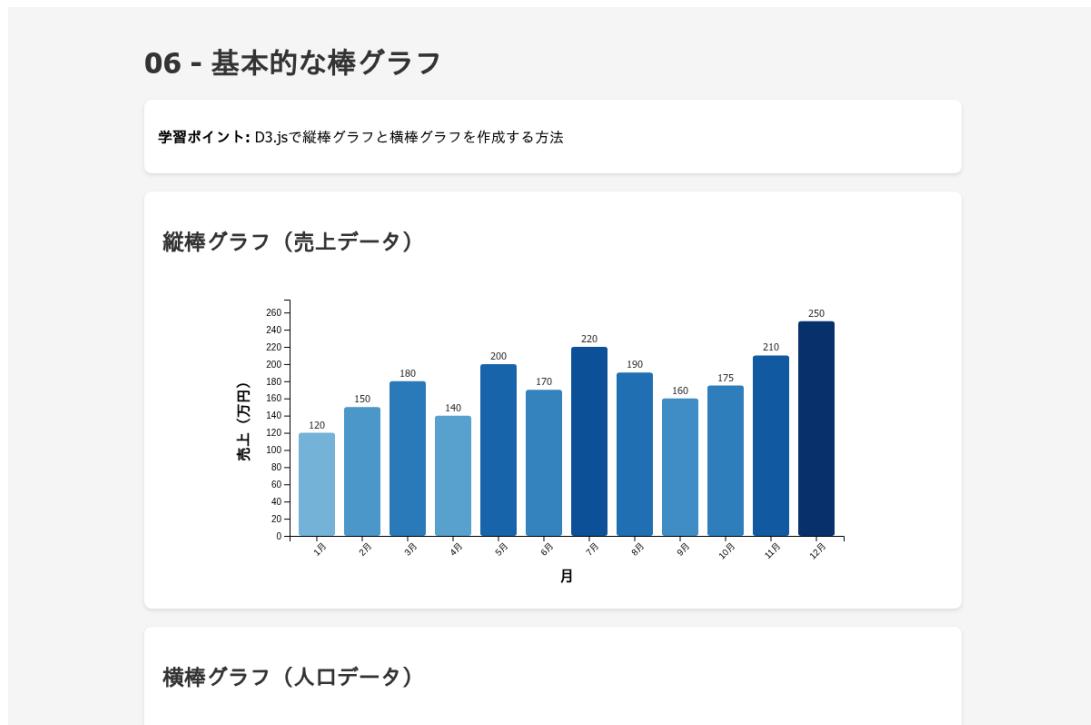


図 7.1 サンプル 06 の実行結果

### 7.2 棒グラフの構成要素

棒グラフは以下の要素で構成されます：

- **棒 (rect 要素)**：データ値を高さで表現
- **X 軸**：カテゴリラベル
- **Y 軸**：数値目盛り
- **ラベル**：各棒の値表示

Listing 7.1 棒グラフの実装

```
1 const data = [
2   {name: "A", value: 30},
3   {name: "B", value: 50},
4   {name: "C", value: 80}
5 ];
6
7 // スケール設定
8 const xScale = d3.scaleBand()
9   .domain(data.map(d => d.name))
10  .range([0, width])
11  .padding(0.2);
12
13 const yScale = d3.scaleLinear()
14  .domain([0, d3.max(data, d => d.value)])
15  .range([height, 0]);
16
17 // 棒を描画 (classを付けて識別しやすくする)
18 svg.selectAll("rect.bar")
19  .data(data)
20  .enter()
21  .append("rect")
22  .attr("class", "bar")
23  .attr("x", d => xScale(d.name))
24  .attr("y", d => yScale(d.value))
25  .attr("width", xScale.bandwidth())
26  .attr("height", d => height - yScale(d.value))
27  .attr("fill", "#4a90d9");
```

### ベストプラクティス：クラスセレクタの使用

`selectAll("rect")` のような広いセレクタは、複数のチャートを同じ SVG に描画する場合に問題を引き起します。必ず `class` 属性を付けて、`selectAll("rect.bar")` のように特定のクラスで選択する習慣をつけましょう。

```
5  svg.selectAll("rect.bar")
6    .data(data)
7    .enter()
8    .append("rect")
9    .attr("class", "bar");
10
11 // 後で更新する際も同じセレクタを使用
12 svg.selectAll("rect.bar")
13   .data(newData)
14   .transition()
15   .attr("height", d => height - yScale(d.value));
```

## 7.3 練習問題

### 練習問題

**問題 1：**以下のデータで棒グラフを作成してください：

```
91 {month: "1月", sales: 120},
92 {month: "2月", sales: 85},
93 {month: "3月", sales: 200},
94 {month: "4月", sales: 150}
95 ];
```

**問題 2：**各棒の上に売上の値をテキストとして表示してください。

**問題 3：**棒の色を売上に応じて変化させてください（低いほど青、高いほど赤）。

## 模範解答

### 問題1の解答：

```
96   .domain(data.map(d => d.month))
97   .range([0, width])
98   .padding(0.2);
99
100 const yScale = d3.scaleLinear()
101   .domain([0, d3.max(data, d => d.sales)])
102   .range([height, 0]);
103
104 svg.selectAll("rect")
105   .data(data)
106   .enter()
107   .append("rect")
108   .attr("x", d => xScale(d.month))
109   .attr("y", d => yScale(d.sales))
110   .attr("width", xScale.bandwidth())
111   .attr("height", d => height - yScale(d.sales))
112   .attr("fill", "#4a90d9");
```

### 問題2の解答：

```
12   .data(data)
13   .enter()
14   .append("text")
15   .attr("class", "label")
16   .attr("x", d => xScale(d.month) + xScale.bandwidth() / 2)
17   .attr("y", d => yScale(d.sales) - 5)
18   .attr("text-anchor", "middle")
19   .text(d => d.sales);
```

### 問題3の解答：

```
113 const maxSales = d3.max(data, d => d.sales);
114 const colorScale = d3.scaleLinear()
115   .domain([minSales, maxSales])
116   .range(["blue", "red"]);
117
118 svg.selectAll("rect")
119   .attr("fill", d => colorScale(d.sales));
```

## 第8章

# 折れ線グラフ

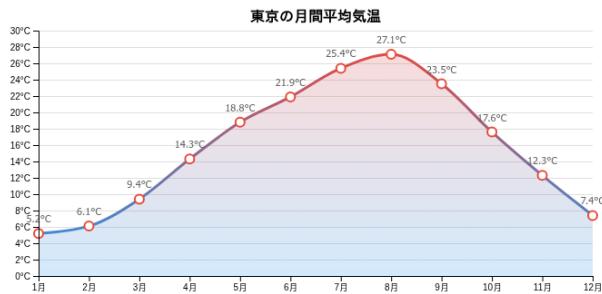
### 8.1 学習目標

連続データを表現する折れ線グラフの作成方法を学びます。

#### 07 - 基本的な折れ線グラフ

学習ポイント: D3.jsで折れ線グラフを作成する方法 - d3.line(), カーブタイプ, データポイント

##### 基本的な折れ線グラフ（気温データ）



##### カーブタイプの比較

図 8.1 サンプル 07 の実行結果

## 8.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
d3.line()  
    線ジェネレータを作成します。データ配列から SVG パス文字列を生成。  
line.x(accessor)  
    各データポイントの X 座標を返す関数を設定。  
line.y(accessor)  
    各データポイントの Y 座標を返す関数を設定。  
line.curve(curveType)  
    線の補間方法を設定（直線、曲線など）。  
d3.curveMonotoneX  
    単調な X 方向の曲線補間。
```

Listing 8.1 折れ線グラフの実装

```
1 // 線 ジ ェ ネ レ タ を 作 成  
2 const line = d3.line()  
3   .x(d => xScale(d.date))  
4   .y(d => yScale(d.value))  
5   .curve(d3.curveMonotoneX);  
6  
7 // パス を 描 画  
8 svg.append("path")  
9   .datum(data)           // 単一のパスに全データをバインド  
10  .attr("fill", "none")  
11  .attr("stroke", "#e74c3c")  
12  .attr("stroke-width", 2)  
13  .attr("d", line);     // パス 文字列を 生成
```

## 8.3 練習問題

### 練習問題

問題 1：以下の気温データで折れ線グラフを作成してください：

```
120 {day: 1, temp: 15}, {day: 2, temp: 18},  
121 {day: 3, temp: 22}, {day: 4, temp: 19},  
122 {day: 5, temp: 25}, {day: 6, temp: 23}  
123 ];
```

問題 2：各データポイントに小さな円を追加してください。

問題 3：曲線補間を d3.curveCatmullRom に変更してください。

## 模範解答

### 問題1の解答：

```
124     .domain([1, 6]).range([0, width]);
125     const yScale = d3.scaleLinear()
126         .domain([10, 30]).range([height, 0]);
127
128     const line = d3.line()
129         .x(d => xScale(d.day))
130         .y(d => yScale(d.temp));
131
132     svg.append("path")
133         .datum(data)
134         .attr("fill", "none")
135         .attr("stroke", "#e74c3c")
136         .attr("stroke-width", 2)
137         .attr("d", line);
```

### 問題2の解答：

```
20     .data(data)
21     .enter()
22     .append("circle")
23     .attr("cx", d => xScale(d.day))
24     .attr("cy", d => yScale(d.temp))
25     .attr("r", 5)
26     .attr("fill", "#e74c3c");
```

### 問題3の解答：

```
138     .x(d => xScale(d.day))
139     .y(d => yScale(d.temp))
140     .curve(d3.curveCatmullRom);
```

## 第9章

# 円グラフ

### 9.1 学習目標

円グラフ（パイチャート）の作成方法を学びます。

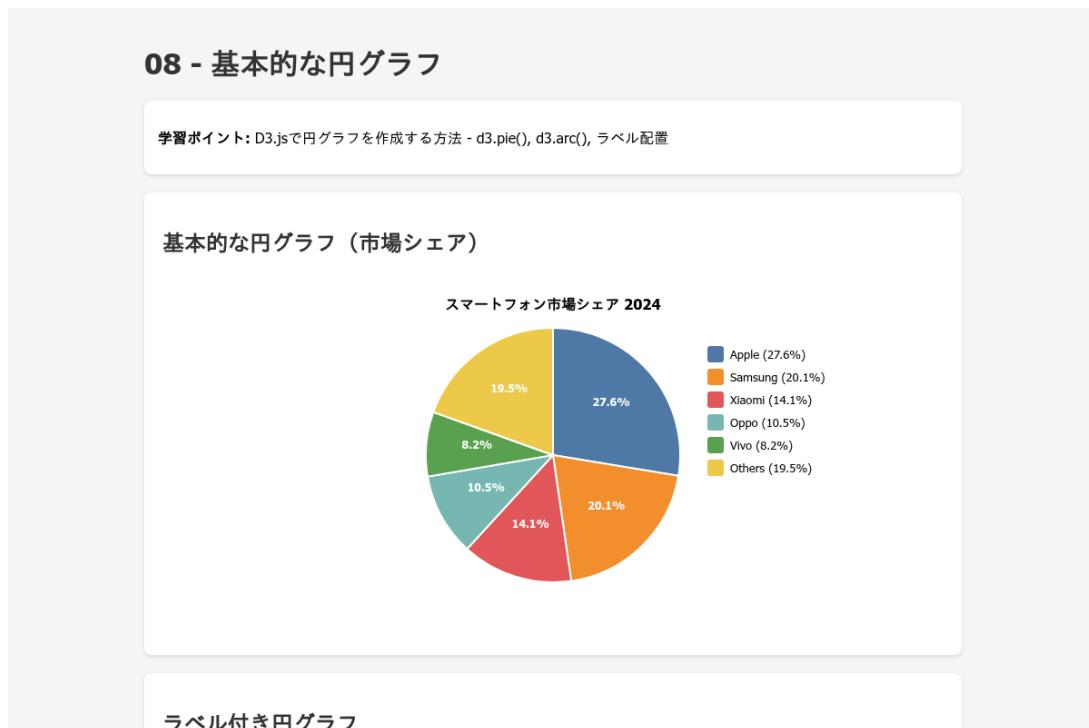


図 9.1 サンプル 08 の実行結果

## 9.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
d3.pie()  
    データを円グラフ用の角度データに変換するレイアウト関数。  
  
d3.arc()  
    円弧（扇形）のSVGパスを生成するジェネレータ。  
  
arc.innerRadius(radius)  
    内側の半径を設定（0で円グラフ、正の値でドーナツ）。  
  
arc.outerRadius(radius)  
    外側の半径を設定。  
  
pie.value(accessor)  
    各データから値を取得する関数を設定。
```

Listing 9.1 円グラフの実装

```
1 // パイレイアウトを作成  
2 const pie = d3.pie()  
3   .value(d => d.value);  
4  
5 // アークジェネレータを作成  
6 const arc = d3.arc()  
7   .innerRadius(0)  
8   .outerRadius(radius);  
9  
10 // 各セグメントを描画  
11 svg.selectAll("path")  
12   .data(pie(data))  
13   .enter()  
14   .append("path")  
15   .attr("d", arc)  
16   .attr("fill", (d, i) => colorScale(i));
```

### 可視化のベストプラクティス

円グラフは見た目に魅力的ですが、以下の点に注意が必要です：

- **比較が難しい**：人間は角度や面積の比較が苦手です。正確な比較が必要な場合は棒グラフを検討してください。
- **カテゴリ数は少なく**：5~6個程度が限度。それ以上は「その他」にまとめるか、別のグラフを使用します。
- **3Dは避ける**：3D円グラフは奥の要素が小さく見えるため、誤解を招きます。
- **合計が100%**：円グラフは全体に対する割合を表すため、合計が100%になるデータに限定します。

データの傾向を伝えるなら円グラフ、正確な値を比較するなら棒グラフが適しています。

## 9.3 練習問題

### 練習問題

**問題1**：以下の売上データで円グラフを作成してください：

```
141 {product: "製品A", sales: 300},  
142 {product: "製品B", sales: 150},  
143 {product: "製品C", sales: 250},  
144 {product: "製品D", sales: 100}  
145 ];
```

**問題2**：各セグメントに製品名のラベルを追加してください。

**問題3**：円グラフをドーナツグラフに変更してください（内側の半径を外側の半径の50%に設定）。

## 模範解答

### 問題1の解答：

```
146 const arc = d3.arc().innerRadius(0).outerRadius(150);
147 const color = d3.scaleOrdinal(d3.schemeCategory10);
148
149 svg.selectAll("path")
150   .data(pie(data))
151   .enter()
152   .append("path")
153   .attr("d", arc)
154   .attr("fill", (d, i) => color(i))
155   .attr("transform", "translate(200, 200)");
```

### 問題2の解答：

```
156   .innerRadius(100).outerRadius(100);
157
158 svg.selectAll("text")
159   .data(pie(data))
160   .enter()
161   .append("text")
162   .attr("transform", d => `translate(${labelArc.centroid(d)})`)
163   .attr("text-anchor", "middle")
164   .text(d => d.data.product);
```

### 問題3の解答：

```
165   .innerRadius(75) // 外側半径の50%
166   .outerRadius(150);
```

## 第 10 章

# トランジション

### 10.1 学習目標

アニメーション効果を追加するトランジションの使い方を学びます。

### 09 - トランジション（アニメーション）

学習ポイント: D3.jsのトランジション機能 - アニメーション、イージング関数、遅延

#### イージング関数の比較

easeLinear

easeQuad

easeCubic

easeSin

easeExp

easeCircle

easeElastic

easeBounce

アニメーション再生

#### 棒グラフのアニメーション

図 10.1 サンプル 09 の実行結果

## 10.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

`selection.transition()`

要素のトランジション（アニメーション）を開始します。

`transition.duration(ms)`

トランジションの継続時間をミリ秒で設定。

`transition.delay(ms)`

トランジション開始までの遅延時間を設定。

`transition.ease(easing)`

イージング関数を設定（加減速のパターン）。

`d3.easeLinear`

一定速度のイージング。

`d3.easeBounce`

バウンドするようなイージング。

`d3.easeElastic`

弾性的なイージング。

Listing 10.1 トランジションの使用例

```
1 // 基本的なトランジション
2 svg.selectAll("rect")
3   .data(data)
4   .enter()
5   .append("rect")
6   .attr("y", height)           // 初期位置（下端）
7   .attr("height", 0)          // 初期高さ（0）
8   .transition()               // トランジション開始
9   .duration(1000)             // 1秒間
10  .delay((d, i) => i * 100) // 順番に遅延
11  .ease(d3.easeElastic)      // 弾性イージング
12  .attr("y", d => yScale(d.value))
13  .attr("height", d => height - yScale(d.value));
```

## 10.3 練習問題

### 練習問題

問題 1：円が左端から右端まで 2 秒かけて移動するアニメーションを作成してください。

問題 2：5 つの棒グラフが順番に（0.2 秒間隔で）下から上に伸びるアニメーションを作成してください。

問題 3：円が縮小してから拡大するバウンドエフェクトを作成してください（イージング: d3.easeBounce）。

## 模範解答

### 問題1の解答：

```
27   .attr("cx", 50)
28   .attr("cy", 100)
29   .attr("r", 25)
30   .attr("fill", "blue")
31   .transition()
32   .duration(2000)
33   .attr("cx", 450);
```

### 問題2の解答：

```
34   .data([60, 80, 40, 100, 70])
35   .enter()
36   .append("rect")
37   .attr("x", (d, i) => i * 60)
38   .attr("y", height)
39   .attr("width", 50)
40   .attr("height", 0)
41   .attr("fill", "#3498db")
42   .transition()
43   .duration(800)
44   .delay((d, i) => i * 200)
45   .attr("y", d => height - d)
46   .attr("height", d => d);
```

### 問題3の解答：

```
47   .attr("cx", 200)
48   .attr("cy", 150)
49   .attr("r", 50)
50   .attr("fill", "red")
51   .transition()
52   .duration(1500)
53   .ease(d3.easeBounce)
54   .attr("r", 80);
```

## 第 11 章

# イベント処理

### 11.1 学習目標

マウスイベントなどのユーザーインターフェイクションの処理方法を学びます。

**10 - イベントハンドリング**

学習ポイント: D3.jsでのマウスイベント、クリック、ホバー、ドラッグの処理

マウスイベント（クリック、ホバー）

円をクリック・ダブルクリック・ホバーしてみてください

100% 95%

インタラクティブな棒グラフ

図 11.1 サンプル 10 の実行結果

## 11.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
selection.on(type, listener)
    イベントリスナーを登録します。
d3.pointer(event)
    イベントから座標を取得します。
```

## 11.3 主なイベントタイプ

表 11.1 よく使用するイベント

イベント	発生タイミング
click	クリック時
mouseenter	マウスが要素に入った時
mouseleave	マウスが要素から出た時
mousemove	マウス移動時
mousedown	マウスボタン押下時
mouseup	マウスボタン離した時

Listing 11.1 イベント処理の例

```
1 svg.selectAll("rect")
2   .on("mouseenter", function(event, d) {
3     // this: イベントが発生した要素
4     // event: イベントオブジェクト
5     // d: バインドされたデータ
6     d3.select(this)
7       .attr("fill", "red");
8   })
9   .on("mouseleave", function() {
10    d3.select(this)
11      .attr("fill", "blue");
12  });

```

### 注意：アロー関数と this

アロー関数 (`() => {}`) を使うと `this` が異なる値を指すため、要素を参照できません。`this` を使用する場合は、必ず通常の `function` を使用してください。

```
16 .on("click", (event, d) => {
17   d3.select(this); // thisはDOM要素を指さない
18 })
19
20 // OK: 通常の function
21 .on("click", function(event, d) {
22   d3.select(this); // thisはイベントが発生した要素
23 })
24
25 // OK: アロー関数で event.targetを使う
26 .on("click", (event, d) => {
27   d3.select(event.target); // event.targetで要素を取得
28 })
```

## 11.4 練習問題

### 練習問題

**問題 1：**円をクリックするたびに色がランダムに変わるようにしてください。

**問題 2：**長方形にマウスを乗せると拡大し、離すと元のサイズに戻るインタラクションを実装してください。

**問題 3：**マウスの位置を追跡して、その座標をテキスト要素に表示してください。

## 模範解答

### 問題 1 の解答：

```
55     .attr("cx", 200).attr("cy", 150).attr("r", 50)
56     .attr("fill", "blue")
57     .on("click", function() {
58       const rand = Math.floor(Math.random() * 16777215);
59       const randomColor = "#" + rand.toString(16);
60       d3.select(this).attr("fill", randomColor);
61     });

```

### 問題 2 の解答：

```
62     .attr("x", 100).attr("y", 100)
63     .attr("width", 80).attr("height", 60)
64     .attr("fill", "green")
65     .on("mouseenter", function() {
66       d3.select(this)
67         .transition().duration(200)
68         .attr("width", 100).attr("height", 80);
69     })
70     .on("mouseleave", function() {
71       d3.select(this)
72         .transition().duration(200)
73         .attr("width", 80).attr("height", 60);
74   });

```

### 問題 3 の解答：

```
167     .attr("x", 10).attr("y", 30);
168
169 svg.on("mousemove", function(event) {
170   const [x, y] = d3.pointer(event);
171   text.text(`X: ${Math.round(x)}, Y: ${Math.round(y)} `);
172 });

```

第 III 部

中級編



## 第 12 章

# 散布図

### 12.1 学習目標

2 変数の関係を表現する散布図の作成方法を学びます。

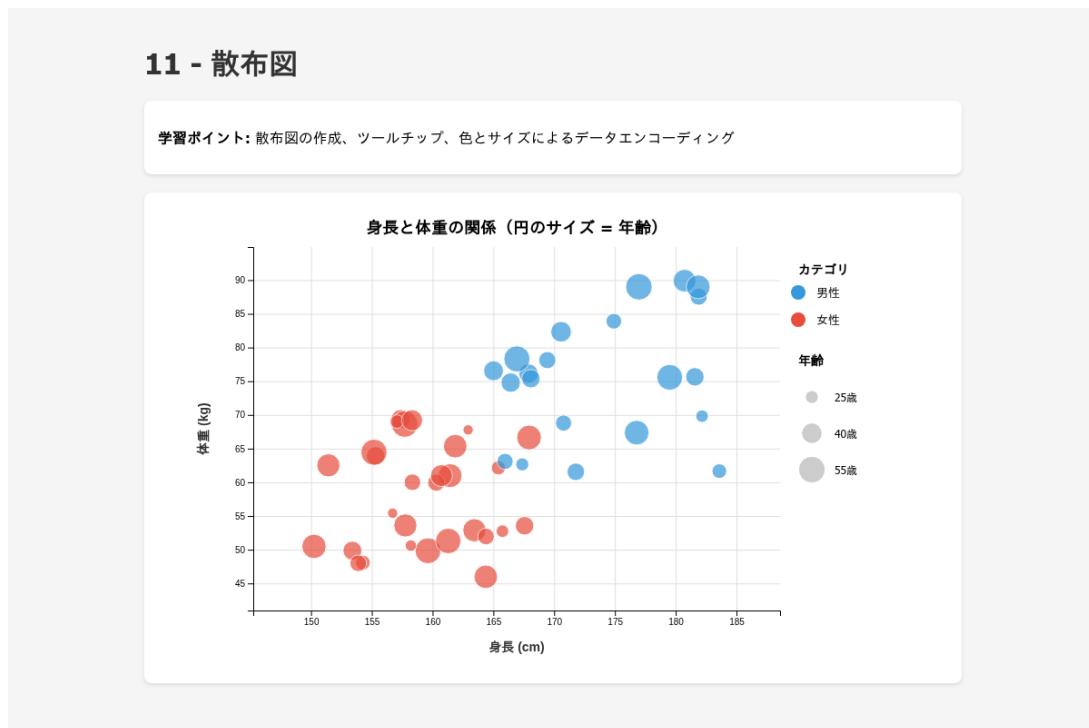


図 12.1 サンプル 11 の実行結果

### 12.2 散布図の特徴

散布図は 2 つの変量の関係（相関）を可視化するのに適しています。各データポイントを円で表現し、X 軸と Y 軸にそれぞれの変量をマッピングします。

Listing 12.1 散布図の実装

```
1 // データポイントを描画
2 svg.selectAll("circle")
3   .data(data)
4   .enter()
5   .append("circle")
6   .attr("cx", d => xScale(d.x))
7   .attr("cy", d => yScale(d.y))
8   .attr("r", 5)
9   .attr("fill", "#3498db")
10  .attr("opacity", 0.7);
```

## 12.3 練習問題

### 練習問題

問題 1：身長と体重のデータで散布図を作成し、BMI に応じて円の色を変えてください。

問題 2：各点のサイズを第 3 の変数（例：年齢）に応じて変化させてください。

問題 3：回帰直線を散布図に追加してください。

## 模範解答

### 問題 1 の解答：

```
173   .domain([18, 25, 30])
174   .range(["green", "yellow", "red"]);
175
176 svg.selectAll("circle")
177   .data(data)
178   .enter()
179   .append("circle")
180   .attr("cx", d => xScale(d.height))
181   .attr("cy", d => yScale(d.weight))
182   .attr("r", 6)
183   .attr("fill", d => {
184     const bmi = d.weight / (d.height/100) ** 2;
185     return colorScale(bmi);
186   });
187
```

### 問題 2 の解答：

```
187   .domain([20, 60]).range([3, 15]);
188
189 svg.selectAll("circle")
190   .attr("r", d => sizeScale(d.age));
191
```

### 問題 3 の解答：

```
29 const xMean = d3.mean(data, d => d.x);
30 const yMean = d3.mean(data, d => d.y);
31
32 // 傾きと切片を計算
33 const num = d3.sum(data, d => (d.x - xMean) * (d.y - yMean));
34 const den = d3.sum(data, d => (d.x - xMean) ** 2);
35 const slope = num / den;
36 const intercept = yMean - slope * xMean;
37
38 // 描画範囲（データのxの最小～最大）で直線を引く
39 const [xMin, xMax] = d3.extent(data, d => d.x);
40
41 svg.append("line")
42   .attr("x1", xScale(xMin))
43   .attr("y1", yScale(slope * xMin + intercept))
44   .attr("x2", xScale(xMax))
45   .attr("y2", yScale(slope * xMax + intercept))
46   .attr("stroke", "red")
47   .attr("stroke-width", 2);
```



## 第 13 章

# エリアチャート

### 13.1 学習目標

折れ線グラフの下部を塗りつぶしたエリアチャートを作成します。



図 13.1 サンプル 12 の実行結果

## 13.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
d3.area()  
    エリア（塗りつぶし領域）のSVGパスを生成するジェネレータ。  
area.y0(accessor)  
    エリアの下端のY座標を設定。  
area.y1(accessor)  
    エリアの上端のY座標を設定。
```

Listing 13.1 エリアチャートの実装

```
1 const area = d3.area()  
2   .x(d => xScale(d.date))  
3   .y0(height)           // 下端（ベースライン）  
4   .y1(d => yScale(d.value)) // 上端（データ値）  
5   .curve(d3.curveMonotoneX);  
6  
7 svg.append("path")  
8   .datum(data)  
9   .attr("fill", "#3498db")  
10  .attr("fill-opacity", 0.3)  
11  .attr("d", area);
```

## 13.3 練習問題

### 練習問題

問題1：月間売上データでエリアチャートを作成し、折れ線も重ねて表示してください。

問題2：グラデーション塗りつぶしを適用してください。

問題3：2つのデータ系列の間を塗りつぶすエリアチャートを作成してください。

## 模範解答

### 問題 1 の解答：

```
48 svg.append("path")
49   .datum(data)
50   .attr("fill", "#3498db")
51   .attr("fill-opacity", 0.3)
52   .attr("d", area);
53
54 // 折れ線
55 const line = d3.line()
56   .x(d => xScale(d.date))
57   .y(d => yScale(d.value));
58
59 svg.append("path")
60   .datum(data)
61   .attr("fill", "none")
62   .attr("stroke", "#3498db")
63   .attr("stroke-width", 2)
64   .attr("d", line);
```

### 問題 2 の解答：

```
191 .append("linearGradient")
192   .attr("id", "areaGradient")
193   .attr("x1", "0%").attr("y1", "0%")
194   .attr("x2", "0%").attr("y2", "100%");
195
196 gradient.append("stop")
197   .attr("offset", "0%").attr("stop-color", "#3498db");
198 gradient.append("stop")
199   .attr("offset", "100%").attr("stop-color", "white");
200
201 svg.append("path")
202   .datum(data)
203   .attr("fill", "url(#areaGradient)")
204   .attr("d", area);
```

### 問題 3 の解答：

```
205   .x(d => xScale(d.date))
206   .y0(d => yScale(d.min)) // 下のデータ
207   .y1(d => yScale(d.max)); // 上のデータ
```



## 第 14 章

# 複数折れ線グラフ

### 14.1 学習目標

複数のデータ系列を 1 つのグラフに表示する方法を学びます。

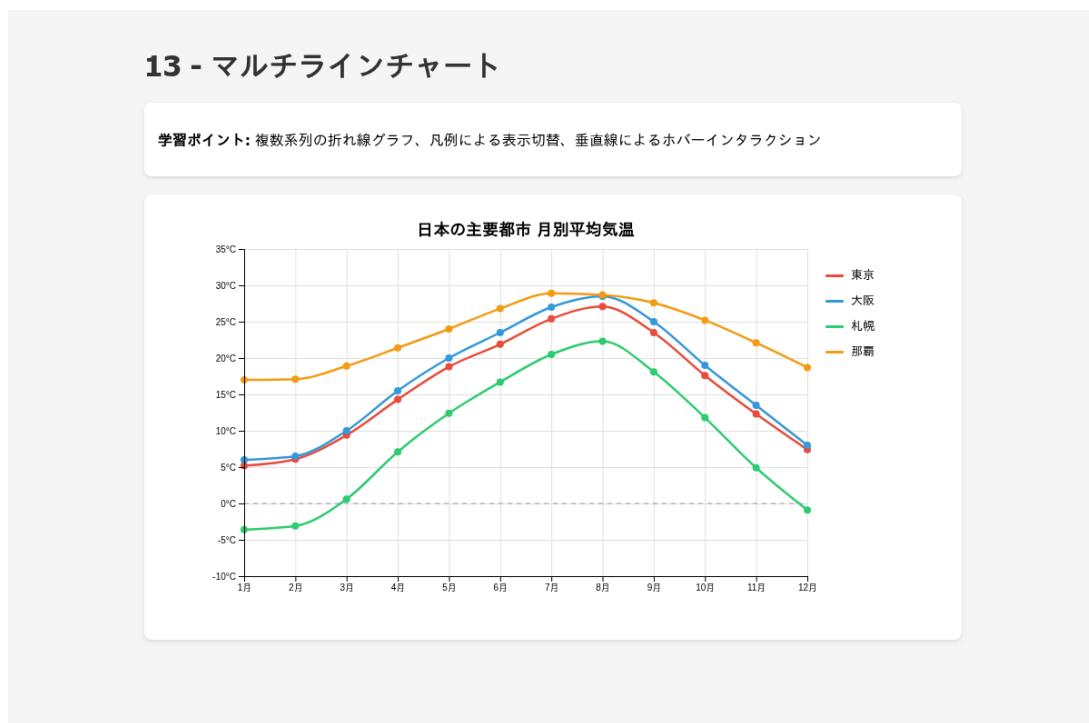


図 14.1 サンプル 13 の実行結果

### 14.2 複数系列の扱い方

データを系列ごとにグループ化し、それぞれに対してパスを描画します。

Listing 14.1 複数折れ線グラフ

```
1 // 系列ごとにパスを描画
2 const lines = svg.selectAll(".line")
3   .data(series)
4   .enter()
5   .append("path")
6   .attr("class", "line")
7   .attr("fill", "none")
8   .attr("stroke", d => colorScale(d.name))
9   .attr("d", d => line(d.values));
```

### 14.3 練習問題

#### 練習問題

問題1：3つの製品の月別売上推移を複数折れ線グラフで表示してください。

問題2：凡例を追加してください。

問題3：ホバー時に対応する線をハイライトしてください。

## 模範解答

### 問題 1 の解答：

```
208 {name: "製品A", values: /* データ */},  
209 {name: "製品B", values: /* データ */},  
210 {name: "製品C", values: /* データ */}  
211 ];  
212  
213 svg.selectAll(".line")  
214 .data(series)  
215 .enter()  
216 .append("path")  
217 .attr("fill", "none")  
218 .attr("stroke", d => colorScale(d.name))  
219 .attr("d", d => line(d.values));
```

### 問題 2 の解答：

```
220 .data(series)  
221 .enter()  
222 .append("g")  
223 .attr("transform", (d, i) => {  
  return `translate(${width + 10}, ${i * 20})`;  
});  
224  
225  
226  
227 legend.append("rect")  
228 .attr("width", 15).attr("height", 15)  
229 .attr("fill", d => colorScale(d.name));  
230  
231 legend.append("text")  
232 .attr("x", 20).attr("y", 12)  
233 .text(d => d.name);
```

### 問題 3 の解答：

```
75 .on("mouseenter", function() {  
  svg.selectAll(".line").attr("opacity", 0.2);  
  d3.select(this)  
    .attr("opacity", 1).attr("stroke-width", 3);  
})  
76  
77  
78  
79  
80 .on("mouseleave", function() {  
  svg.selectAll(".line")  
    .attr("opacity", 1).attr("stroke-width", 2);  
});  
81  
82  
83
```



## 第 15 章

# 積み上げ棒グラフ

### 15.1 学習目標

複数カテゴリを積み上げた棒グラフを作成します。



図 15.1 サンプル 14 の実行結果

## 15.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

`d3.stack()`

積み上げレイアウトを作成します。データを積み上げ形式に変換。

`stack.keys(keys)`

積み上げるデータのキー（列名）を設定。

`d3.sum(array)`

配列の合計値を計算。

## 15.3 練習問題

### 練習問題

**問題1：**四半期ごとの部門別売上を積み上げ棒グラフで表示してください。

**問題2：**100% 積み上げ棒グラフ（正規化）に変更してください。

**問題3：**グループ化棒グラフとの切り替え機能を実装してください。

## 模範解答

### 問題 1 の解答：

```
234   .keys(["部門A", "部門B", "部門C"]);  
235  
236   const series = stack(data);  
237  
238   svg.selectAll("g.layer")  
239     .data(series)  
240     .enter()  
241     .append("g")  
242     .attr("fill", d => colorScale(d.key))  
243     .selectAll("rect")  
244     .data(d => d)  
245     .enter()  
246     .append("rect")  
247     .attr("x", (d, i) => xScale(data[i].quarter))  
248     .attr("y", d => yScale(d[1]))  
249     .attr("height", d => yScale(d[0]) - yScale(d[1]))  
250     .attr("width", xScale.bandwidth());
```

### 問題 2 の解答：

```
251   .keys(["部門A", "部門B", "部門C"])  
252   .offset(d3.stackOffsetExpand); // 正規化
```

### 問題 3 の解答：

```
1  if (type === "stacked") {  
2    // 積み上げ表示  
3  } else {  
4    // グループ化表示  
5    const xSubScale = d3.scaleBand()  
6      .domain(keys).range([0, xScale.bandwidth()]);  
7    // 各グループ内で横に並べる  
8  }  
9 }
```



# 第 16 章

## ドーナツグラフ

### 16.1 学習目標

円グラフの発展形であるドーナツグラフを作成します。

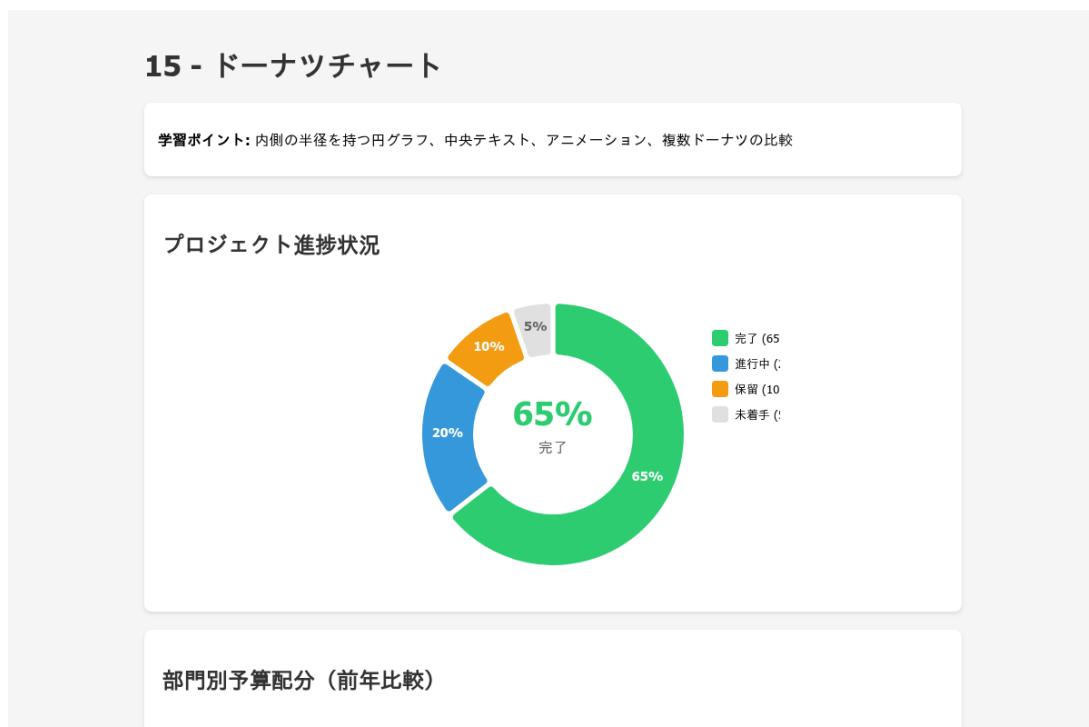


図 16.1 サンプル 15 の実行結果

### 16.2 実装のポイント

ドーナツグラフは円グラフの `innerRadius` を 0 より大きく設定することで作成できます。

### 16.3 練習問題

#### 練習問題

問題1：中央に合計値を表示するドーナツグラフを作成してください。

問題2：セグメントをクリックすると拡大するインタラクションを追加してください。

#### 模範解答

問題1の解答：

```
253     .innerRadius(radius * 0.6)
254     .outerRadius(radius);
255
256 // 中央テキスト
257 svg.append("text")
258   .attr("text-anchor", "middle")
259   .attr("dy", "0.35em")
260   .attr("font-size", "24px")
261   .text(d3.sum(data, d => d.value));
```

問題2の解答：

```
262     .innerRadius(radius * 0.6)
263     .outerRadius(radius * 1.1);
264
265 svg.selectAll("path")
266   .on("click", function() {
267     svg.selectAll("path").attr("d", arc);
268     d3.select(this).attr("d", arcExpanded);
269   });
270
```

## 第 17 章

# ツールチップ

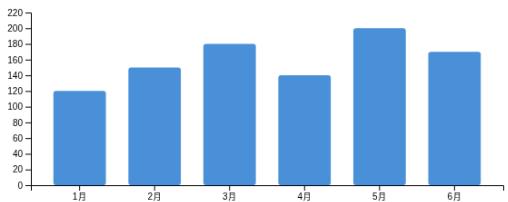
### 17.1 学習目標

マウスホバー時に詳細情報を表示するツールチップを実装します。

**16 - ツールチップ**

学習ポイント: 基本的なツールチップ、リッチなツールチップ、SVG内ツールチップ、位置調整

**基本的なツールチップ**



月	値
1月	120
2月	145
3月	175
4月	140
5月	195
6月	165

**リッチツールチップ（詳細情報付き）**



図 17.1 サンプル 16 の実行結果

### 17.2 練習問題

#### 練習問題

**問題 1：**複数行の情報を含むリッチなツールチップを作成してください。

**問題 2：**ツールチップが画面右端で切れないように位置を調整してください。

## 模範解答

### 問題1の解答：

```
1 <strong>${d.name}</strong><br>
2 売上： ${d.sales}万円<br>
3 前年比： ${d.growth}%
4 `);
```

### 問題2の解答：

```
1 let left = event.pageX + 10;
2 let top = event.pageY - 20;
3
4 // 右端チェック
5 if (left + 150 > window.innerWidth) {
6     left = event.pageX - 160;
7 }
8
9 tooltip.style("left", left + "px").style("top", top + "px");
10});
```

**発展：**上下左右すべての端を考慮する場合は、`tooltip.node().getBoundingClientRect()`でツールチップの実寸を取得し、各方向ではみ出すかを判定して位置を反転させます。詳細はLLM章（#09）の「よくある誤り」も参考にしてください。

**座標の補足：**`pageX/pageY`はドキュメント全体を基準とした座標で、スクロール量を含みます。一方、`clientX/clientY`はビューポート（表示領域）を基準とした座標です。ページがスクロールしない場合は同じ値になりますが、スクロールがある場合は異なります。ツールチップを`position: fixed`で配置する場合は`clientX/clientY`、`position: absolute`で配置する場合は`pageX/pageY`を使うのが一般的です。

## 第 18 章

# レスポンシブチャート

### 18.1 学習目標

ウィンドウサイズに応じて自動的にサイズ調整されるチャートを作成します。

#### 17 - レスponsivチャート

学習ポイント: viewBoxを使ったレスポンシブデザイン、リサイズ対応、アスペクト比の維持  
ブラウザのウィンドウサイズを変更すると、チャートが自動的にリサイズされます。

##### viewBoxによるレスポンシブチャート

月別売上データ (viewBoxレスポンシブ)

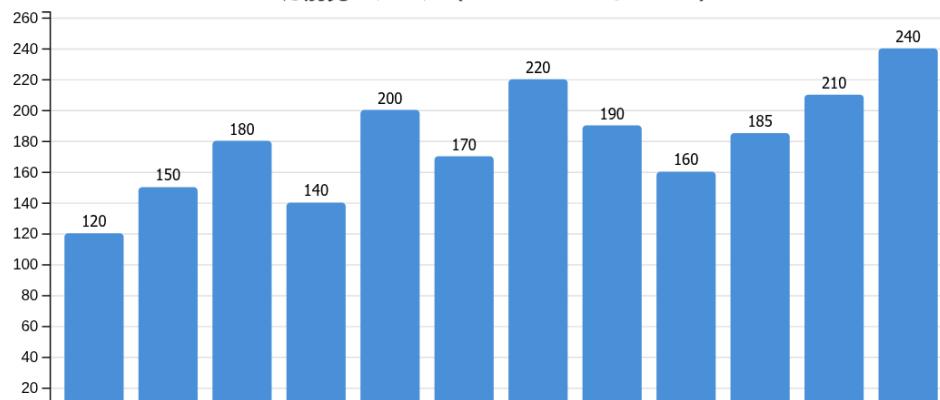


図 18.1 サンプル 17 の実行結果

### 18.2 練習問題

#### 練習問題

問題 1：viewBox を使ってレスポンシブな棒グラフを作成してください。

問題 2：ウィンドウリサイズ時に再描画するチャートを作成してください。

## 模範解答

### 問題1の解答：

```
270     .append("svg")
271     .attr("viewBox", "0 0 600 400")
272     .attr("preserveAspectRatio", "xMidYMid meet")
273     .style("width", "100%")
274     .style("max-width", "600px");
```

### 問題2の解答：

```
65 function debounce(fn, delay = 150) {
66   let timeoutId;
67   return (...args) => {
68     clearTimeout(timeoutId);
69     timeoutId = setTimeout(() => fn(...args), delay);
70   };
71 }
72
73 // コンテナ要素の参照
74 const container = d3.select("#chart");
75 let svg;
76
77 function draw() {
78   // 既存のSVGを削除
79   container.select("svg").remove();
80
81   // コンテナの幅を取得
82   const width = container.node().getBoundingClientRect().width;
83   const height = width * 0.6; // アスペクト比を維持
84
85   // 新しいSVGを作成
86   svg = container.append("svg")
87     .attr("width", width)
88     .attr("height", height);
89
90   // widthとheightに基づいてスケールを計算
91   const xScale = d3.scaleBand()
92     .domain(data.map(d => d.name))
93     .range([margin.left, width - margin.right])
94     .padding(0.2);
95
96   // ... チャートを描画
97 }
98
99 // リサイズ時にdebounceを適用して再描画
100 window.addEventListener("resize", debounce(draw, 150));
101
102 // 初回描画
103 draw();
```

## 第 19 章

# データ更新

### 19.1 学習目標

データが変更された際にチャートを更新する方法を学びます。



図 19.1 サンプル 18 の実行結果

### 19.2 練習問題

#### 練習問題

- 問題 1：ボタンクリックでランダムなデータに更新される棒グラフを作成してください。
- 問題 2：データの追加・削除に対応したアニメーション付き更新を実装してください。

## 模範解答

### 問題1の解答：

```
10 const newData = d3.range(5).map(() => Math.random() * 100);
11
12 svg.selectAll("rect")
13   .data(newData)
14   .transition()
15   .duration(500)
16   .attr("y", d => yScale(d))
17   .attr("height", d => height - yScale(d));
18 }
19
20 d3.select("#updateBtn").on("click", update);
```

### 問題2の解答：

```
21 const bars = svg.selectAll("rect").data(data, d => d.id);
22
23 bars.exit()
24   .transition().duration(300)
25   .attr("width", 0).remove();
26
27 bars.enter()
28   .append("rect")
29   .attr("width", 0)
30   .merge(bars)
31   .transition().duration(500)
32   .attr("x", d => xScale(d.name))
33   .attr("width", xScale.bandwidth());
34 }
```

## 第 20 章

# ズーム機能

### 20.1 学習目標

マウスホイールやドラッグでグラフをズーム・パンする機能を実装します。



図 20.1 サンプル 19 の実行結果

## 20.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
d3.zoom()  
    ズーム動作を作成します。  
zoom.scaleExtent([min, max])  
    ズーム倍率の範囲を設定。  
zoom.on("zoom", handler)  
    ズームイベントのハンドラを設定。  
transform.rescaleX(scale)  
    スケールにズーム変換を適用。  
d3.zoomIdentity  
    初期状態のズーム変換。
```

### 実装の落とし穴

ズームやブラッシングを実装する際の注意点：

- **ズームのリセット機能**：ユーザーが迷子にならないよう、必ずリセットボタンを用意しましょう。
- **scaleExtent の設定**：最小・最大倍率を設定しないと、無限にズームできてしまいます。
- **translateExtent の設定**：パン（移動）の範囲も制限しておくと、データが画面外に行くのを防げます。
- **軸の更新**：ズーム時は軸も一緒に更新する必要があります（rescaleX/Y を使用）。

## 20.3 練習問題

### 練習問題

**問題 1**：散布図にズーム機能を追加してください（1～10 倍の範囲）。

**問題 2**：ズームをリセットするボタンを追加してください。

**問題 3**：X 軸方向のみズームできるようにしてください。

## 模範解答

### 問題 1 の解答：

```
275     .scaleExtent([1, 10])
276     .on("zoom", (event) => {
277       const t = event.transform;
278       svg.selectAll("circle")
279         .attr("transform", t);
280     });
281
282 svg.call(zoom);
```

### 問題 2 の解答：

```
16   svg.transition().duration(500)
17     .call(zoom.transform, d3.zoomIdentity);
18 };
```

### 問題 3 の解答：

```
35   const newXScale = event.transform.rescaleX(xScale);
36   xAxis.call(d3.axisBottom(newXScale));
37   circles.attr("cx", d => newXScale(d.x));
38   // Y座標は変更しない
39 }
```

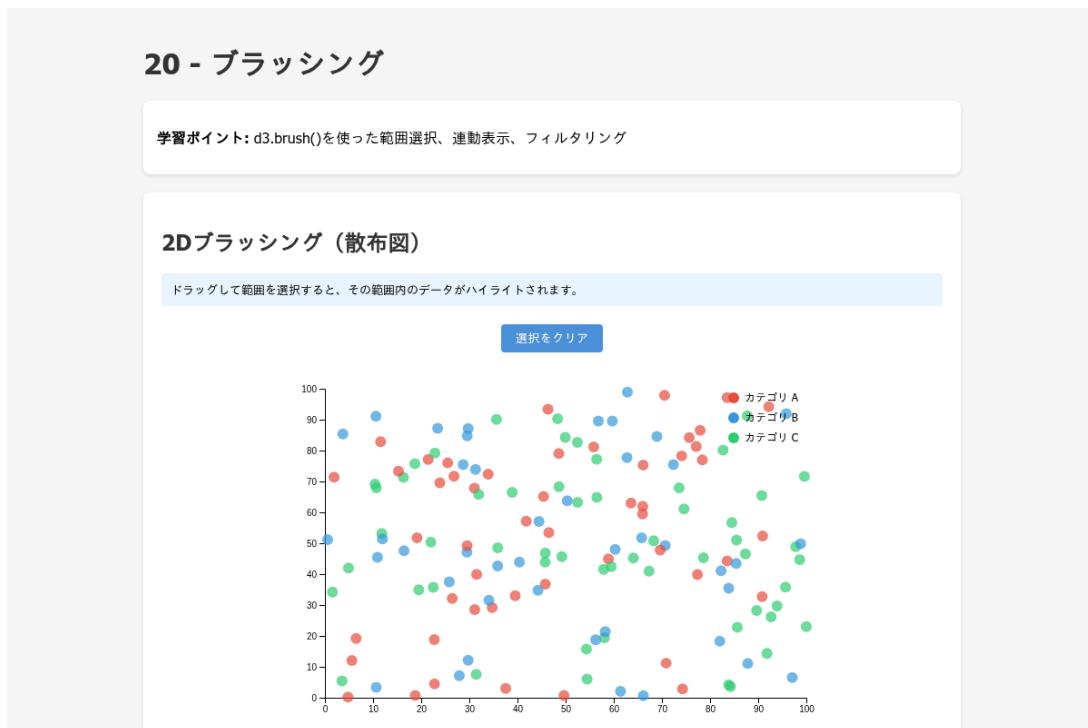


## 第 21 章

# ブラッシング

### 21.1 学習目標

マウスドラッグで範囲を選択するブラッシング機能を実装します。



## 21.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

`d3.brush()`

2D ブラッシングを作成。

`d3.brushX()`

X 軸方向のみのブラッシング。

`d3.brushY()`

Y 軸方向のみのブラッシング。

`brush.extent([[x0,y0],[x1,y1]])`

ブラッシング可能な領域を設定。

## 21.3 練習問題

### 練習問題

**問題 1：**散布図で選択範囲内の点だけをハイライトしてください。

**問題 2：**選択された点の数を表示してください。

**問題 3：**2 つの連携したチャート（概要と詳細）を作成してください。

## 模範解答

### 問題 1 の解答：

```
283     .extent([[0, 0], [width, height]])
284     .on("brush", brushed);
285
286     function brushed(event) {
287       const [[x0, y0], [x1, y1]] = event.selection;
288       circles.attr("fill", d => {
289         const cx = xScale(d.x), cy = yScale(d.y);
290         return (cx >= x0 && cx <= x1 && cy >= y0 && cy <= y1)
291           ? "red" : "gray";
292     });
293 }
```

### 問題 2 の解答：

```
40 // event.selectionがnullの場合（クリア時）を考慮
41 if (!event.selection) {
42   d3.select("#count").text("選択：0件");
43   return;
44 }
45
46 const [[x0, y0], [x1, y1]] = event.selection;
47
48 const selected = data.filter(d => {
49   const cx = xScale(d.x);
50   const cy = yScale(d.y);
51   return x0 <= cx && cx <= x1 && y0 <= cy && cy <= y1;
52 });
53
54 d3.select("#count").text(`選択：${selected.length}件`);
55 }
```

### 問題 3 の解答：

```
104 const brushX = d3.brushX()
105   .extent([[0, 0], [overviewWidth, overviewHeight]])
106   .on("brush end", brushed);
107
108 overviewSvg.append("g")
109   .attr("class", "brush")
110   .call(brushX);
111
112 function brushed(event) {
113   if (!event.selection) return;
114   const [x0, x1] = event.selection;
115
116   // 概要のスケールを反転して詳細のドメインを更新
117   detailXScale.domain([overviewXScale.invert(x0),
118                         overviewXScale.invert(x1)]);
119   updateDetailChart();
```



第 IV 部

上級編



## 第 22 章

# フォースグラフ

### 22.1 学習目標

ノードとリンクで構成されるネットワーク図を作成します。

#### 21 - 力指向グラフ

学習ポイント: d3.forceSimulation()を使ったネットワーク可視化、ノードのドラッグ、力のカスタマイズ  
ノードをドラッグして移動できます。

シミュレーション再開 ラベル表示切替 反発力 + 反発力 -

```
graph TD; CEO((CEO)) --- CFO((CFO)); CFO --- CTO((CTO)); MD[マーケティング部長] --- MA((マーケティングA)); MD --- MB((マーケティングB)); MD --- MC((マーケティングC)); HD[人事部長] --- HA((人事A)); HD --- HB((人事B));
```

図 22.1 サンプル 21 の実行結果

## 22.2 新しく学ぶ関数・メソッド

### 主要な関数・メソッド

```
d3.forceSimulation(nodes)
    物理シミュレーションを作成。
d3.forceLink(links)
    リンク間の距離を制御する力。
d3.forceManyBody()
    ノード間の引力/斥力。
d3.forceCenter(x, y)
    中心に向かう力。
simulation.on("tick", handler)
    シミュレーションの各ステップで呼び出される。
```

## 22.3 練習問題

### 練習問題

**問題 1：**10 個のノードと 15 本のリンクを持つフォースグラフを作成してください。

**問題 2：**ノードをドラッグで移動できるようにしてください。

**問題 3：**ノードのサイズを接続数に応じて変化させてください。

## 模範解答

### 問題 1 の解答：

```
294   .force("link", d3.forceLink(links).id(d => d.id))
295   .force("charge", d3.forceManyBody().strength(-200))
296   .force("center", d3.forceCenter(width/2, height/2))
297   .on("tick", ticked);
```

### 問題 2 の解答：

```
298   .on("start", (event, d) => {
299     if (!event.active) simulation.alphaTarget(0.3).restart();
300     d.fx = d.x; d.fy = d.y;
301   })
302   .on("drag", (event, d) => {
303     d.fx = event.x; d.fy = event.y;
304   })
305   .on("end", (event, d) => {
306     if (!event.active) simulation.alphaTarget(0);
307     d.fx = null; d.fy = null;
308   });
309
310 node.call(drag);
```

### 問題 3 の解答：

```
121 nodes.forEach(n => n.linkCount = 0);
122 links.forEach(l => {
123   nodes[l.source].linkCount++;
124   nodes[l.target].linkCount++;
125 });
126
127 node.attr("r", d => 5 + d.linkCount * 2);
```



## 第 23 章

# ツリーマップ

### 23.1 学習目標

階層データを入れ子の長方形で可視化するツリーマップを作成します。



図 23.1 サンプル 22 の実行結果

### 23.2 練習問題

#### 練習問題

問題 1：ファイルサイズを可視化するツリーマップを作成してください。

問題 2：クリックでドリルダウンできるようにしてください。

## 模範解答

### 問題1の解答：

```

311   .size([width, height])
312   .padding(2);
313
314 const root = d3.hierarchy(data)
315   .sum(d => d.size)
316   .sort((a, b) => b.value - a.value);
317
318 treemap(root);
319
320 svg.selectAll("rect")
321   .data(root.leaves())
322   .enter()
323   .append("rect")
324   .attr("x", d => d.x0)
325   .attr("y", d => d.y0)
326   .attr("width", d => d.x1 - d.x0)
327   .attr("height", d => d.y1 - d.y0);

```

### 問題2の解答：

```

56 // クリックしたノードの範囲を全体に拡大するスケール
57 const x = d3.scaleLinear()
58   .domain([node.x0, node.x1])
59   .range([0, width]);
60 const y = d3.scaleLinear()
61   .domain([node.y0, node.y1])
62   .range([0, height]);
63
64 // 全ての矩形の位置とサイズを更新
65 svg.selectAll("rect")
66   .transition()
67   .duration(750)
68   .attr("x", d => x(d.x0))
69   .attr("y", d => y(d.y0))
70   .attr("width", d => x(d.x1) - x(d.x0))
71   .attr("height", d => y(d.y1) - y(d.y0));
72
73 // テキストラベルも更新
74 svg.selectAll("text")
75   .transition()
76   .duration(750)
77   .attr("x", d => x(d.x0) + 5)
78   .attr("y", d => y(d.y0) + 15);
79 }
80
81 svg.selectAll("rect").on("click", (event, d) => {
82   // 親ノードがあればそこにズーム（戻る機能）
83   // なければ選択したノードにズーム
84   zoom(d.parent || d);
85 });

```

## 第 24 章

# 階層ツリー

### 24.1 学習目標

組織図のような階層ツリーを作成します。



図 24.1 サンプル 23 の実行結果

### 24.2 練習問題

#### 練習問題

問題 1：展開/折りたたみ可能な組織図を作成してください。

問題 2：水平レイアウトと垂直レイアウトを切り替えられるようにしてください。

## 模範解答

### 問題 1 の解答：

```
86  if (d.children) {  
87      d._children = d.children;  
88      d.children = null;  
89  } else {  
90      d.children = d._children;  
91      d._children = null;  
92  }  
93  update(d);  
94 }
```

### 問題 2 の解答：

```
95  const tree = d3.tree();  
96  if (horizontal) {  
97      tree.size([height, width]);  
98      // リンクは linkHorizontal  
99  } else {  
100     tree.size([width, height]);  
101     // リンクは linkVertical  
102  }  
103  update(root);  
104 }
```

## 第 25 章

# サークルパッキング

### 25.1 学習目標

階層データを入れ子の円で表現するサークルパッキングを作成します。

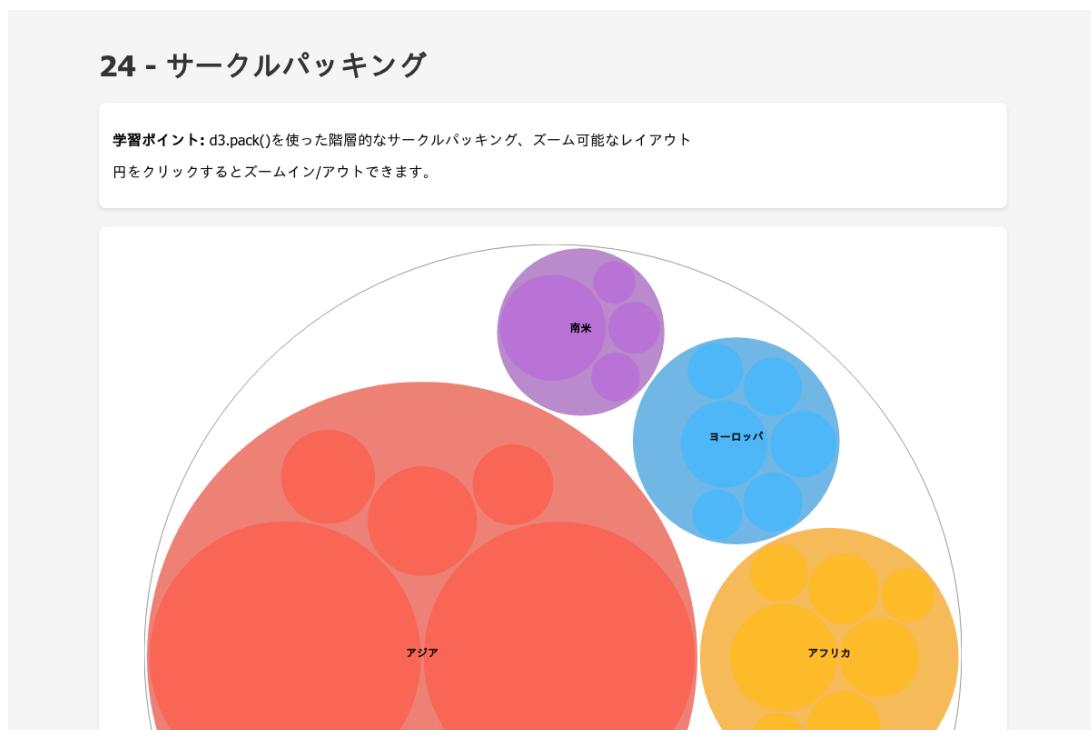


図 25.1 サンプル 24 の実行結果

### 25.2 練習問題

#### 練習問題

問題 1：世界の人口データでサークルパッキングを作成してください。

問題 2：クリックでズームイン/アウトできるようにしてください。

## 模範解答

### 問題1の解答：

```
328     .size([width, height])
329     .padding(3);
330
331 const root = d3.hierarchy(data)
332   .sum(d => d.population);
333
334 pack(root);
```

### 問題2の解答：

```
105 const transition = svg.transition().duration(750);
106
107 svg.selectAll("circle")
108   .transition(transition)
109   .attr("transform", node => {
110     const k = width / (d.r * 2);
111     const x = (node.x - d.x) * k;
112     const y = (node.y - d.y) * k;
113     return `translate(${x + width/2}, ${y + height/2})`;
114   })
115   .attr("r", node => node.r * (width / (d.r * 2)));
116 }
```

## 第 26 章

# サンバースト図

### 26.1 学習目標

階層データを同心円状のセグメントで表現するサンバースト図を作成します。

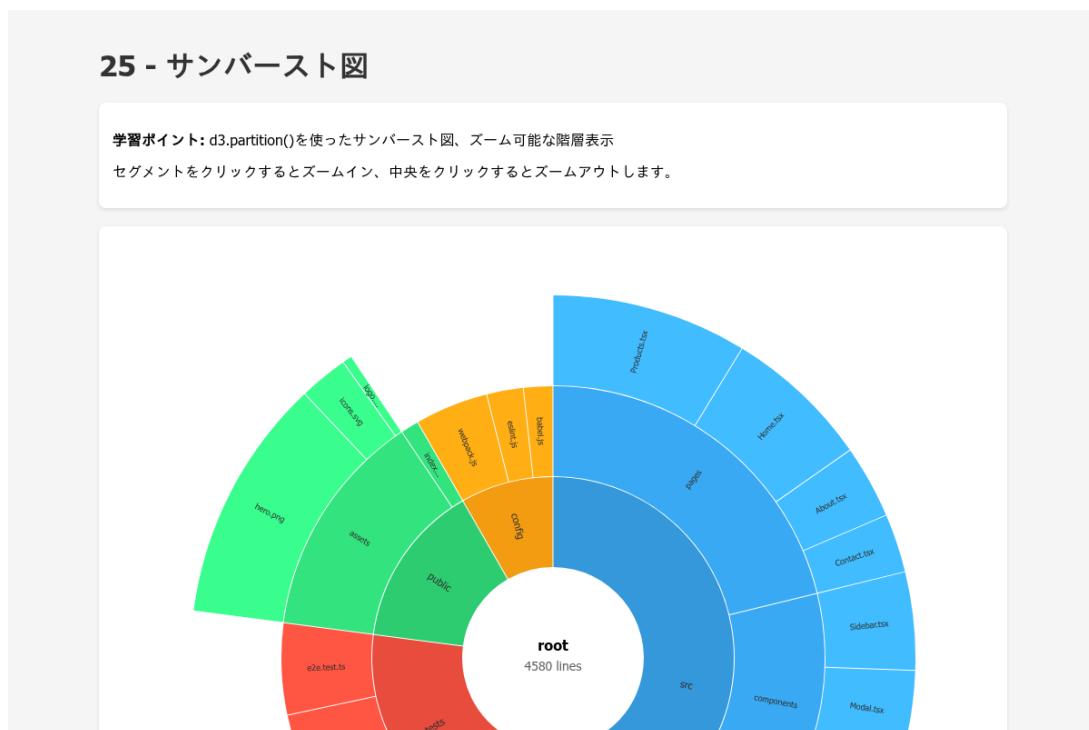


図 26.1 サンプル 25 の実行結果

### 26.2 練習問題

#### 練習問題

問題 1：ファイルシステム構造をサンバースト図で可視化してください。

問題 2：セグメントクリックでズームできるようにしてください。

## 模範解答

### 問題1の解答：

```
335     .size([2 * Math.PI, radius]);  
336  
337     const arc = d3.arc()  
338         .startAngle(d => d.x0)  
339         .endAngle(d => d.x1)  
340         .innerRadius(d => d.y0)  
341         .outerRadius(d => d.y1);
```

### 問題2の解答：

```
117     svg.transition().duration(750).tween("scale", () => {  
118         const xd = d3.interpolate(x.domain(), [p.x0, p.x1]);  
119         const yd = d3.interpolate(y.domain(), [p.y0, 1]);  
120         return t => { x.domain(xd(t)); y.domain(yd(t)); };  
121     });  
122 }
```

## 第 27 章

# コードダイアグラム

### 27.1 学習目標

関係性やフローを表現するコードダイアグラムを作成します。

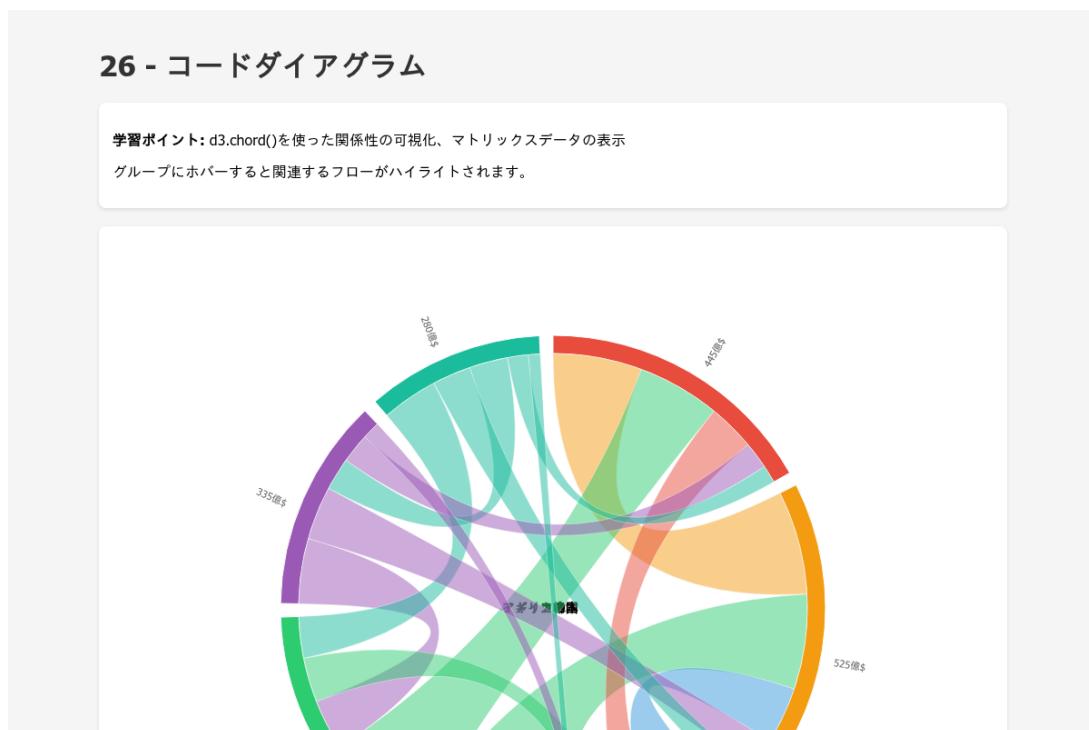


図 27.1 サンプル 26 の実行結果

### 27.2 練習問題

#### 練習問題

問題 1：国間の貿易フローをコードダイアグラムで表示してください。

問題 2：ホバー時に関連するフローをハイライトしてください。

## 模範解答

### 問題1の解答：

```
342     .padAngle(0.05)
343     .sortSubgroups(d3.descending);
344
345 const chords = chord(matrix);
346
347 const ribbon = d3.ribbon().radius(innerRadius);
348
349 svg.selectAll("path.ribbon")
350   .data(chords)
351   .enter()
352   .append("path")
353   .attr("d", ribbon);
```

### 問題2の解答：

```
84   .on("mouseenter", (event, d) => {
85     svg.selectAll(".ribbon")
86       .style("opacity", r =>
87         r.source.index === d.index || r.target.index === d.index
88           ? 1 : 0.1
89       );
90   });
91 );
```

## 第 28 章

# 世界地図

### 28.1 学習目標

地理データを使った世界地図とコロプレスマップを作成します。



図 28.1 サンプル 27 の実行結果

### 28.2 練習問題

#### 練習問題

問題 1：人口密度に応じた色分け地図を作成してください。

問題 2：ドラッグで地球を回転できるようにしてください。

## 模範解答

### 問題1の解答：

```
354     .domain([0, d3.max(data, d => d.density)]);  
355  
356 svg.selectAll("path")  
357   .data(countries.features)  
358   .enter()  
359   .append("path")  
360   .attr("d", path)  
361   .attr("fill", d => colorScale(d.properties.density));
```

### 問題2の解答：

```
362   .on("drag", (event) => {  
363     const rotate = projection.rotate();  
364     const newRotate = [  
365       rotate[0] + event.dx * 0.5,  
366       rotate[1] - event.dy * 0.5  
367     ];  
368     projection.rotate(newRotate);  
369     svg.selectAll("path").attr("d", path);  
370   });  
371  
372 svg.call(drag);
```

## 第 29 章

# ヒートマップ

### 29.1 学習目標

2 次元データを色で可視化するヒートマップを作成します。

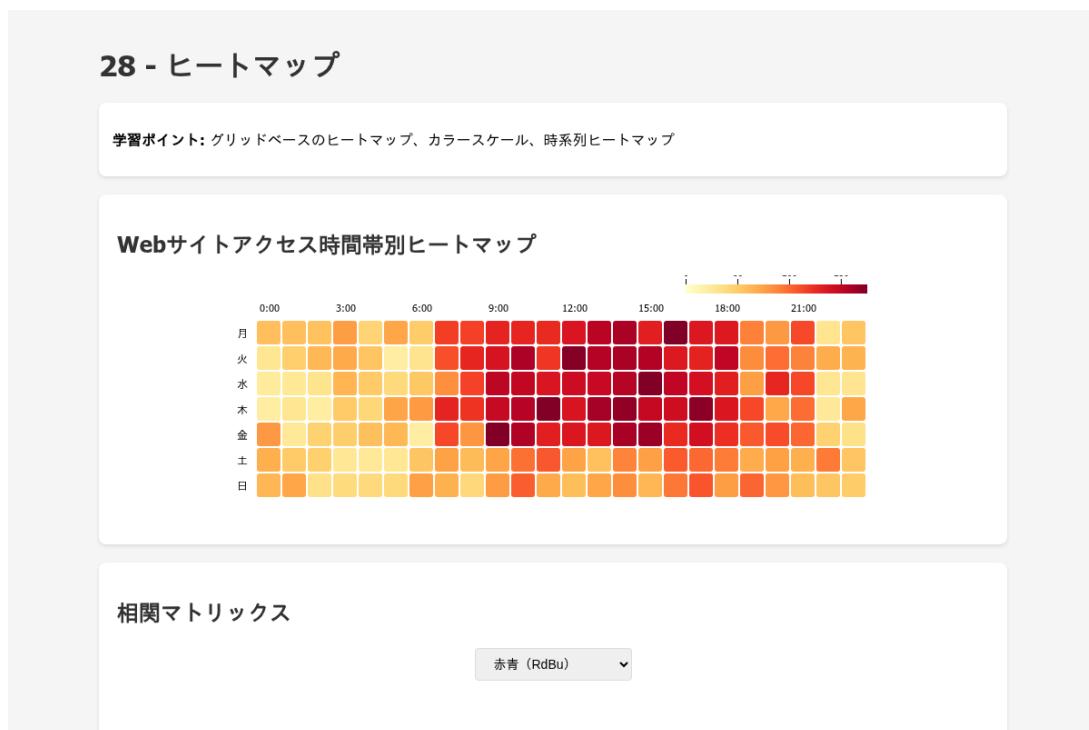


図 29.1 サンプル 28 の実行結果

### 29.2 練習問題

#### 練習問題

問題 1：時間帯 × 曜日のアクセス数ヒートマップを作成してください。

問題 2：カラースケールを変更する UI を追加してください。

## 模範解答

### 問題1の解答：

```
373 const colorScale = d3.scaleSequential(d3.interpolateYlOrRd)
374   .domain([0, d3.max(data, d => d.value)]);
375
376 svg.selectAll("rect")
377   .data(data)
378   .enter()
379   .append("rect")
380   .attr("x", d => d.hour * cellSize)
381   .attr("y", d => d.day * cellSize)
382   .attr("width", cellSize - 2)
383   .attr("height", cellSize - 2)
384   .attr("fill", d => colorScale(d.value));
```

### 問題2の解答：

```
19 const scheme = d3[this.value];
20 colorScale.interpolator(scheme);
21 svg.selectAll("rect")
22   .transition()
23   .attr("fill", d => colorScale(d.value));
24 });
```

## 第 30 章

# レーダーチャート

### 30.1 学習目標

複数の評価軸を持つデータを多角形で可視化するレーダーチャートを作成します。

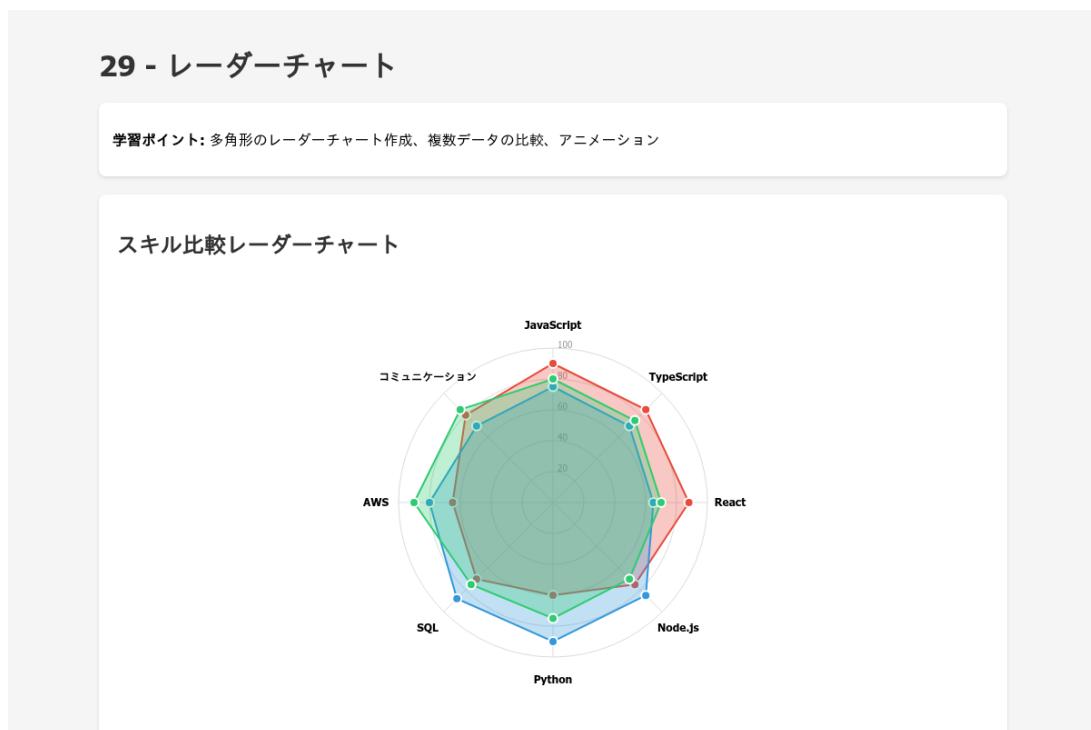


図 30.1 サンプル 29 の実行結果

### 30.2 練習問題

#### 練習問題

問題 1：スキル評価のレーダーチャートを作成してください。

問題 2：複数人のデータを重ねて表示してください。

## 模範解答

### 問題1の解答：

```
385
386 function getPath(values) {
387   return values.map((v, i) => {
388     const angle = angleSlice * i - Math.PI / 2;
389     const r = (v / 100) * radius;
390     return [r * Math.cos(angle), r * Math.sin(angle)];
391   });
392 }
393
394 svg.append("polygon")
395   .attr("points", getPath(data).map(p => p.join(",")).join(" "))
396   .attr("fill", "rgba(52, 152, 219, 0.3)")
397   .attr("stroke", "#3498db");
```

### 問題2の解答：

```
1  svg.append("polygon")
2    .attr("points", getPath(person.values))
3    .attr("fill", colors[i])
4    .attr("fill-opacity", 0.3)
5    .attr("stroke", colors[i]);
6});
```

## 第 31 章

# インタラクティブダッシュボード

### 31.1 学習目標

複数のチャートを連携させたインタラクティブなダッシュボードを作成します。



図 31.1 サンプル 30 の実行結果

### 31.2 ダッシュボードの構成要素

- 複数のチャート（棒グラフ、折れ線グラフ、円グラフなど）
- フィルタ機能
- チャート間の連携（一方を操作すると他方も更新）
- ツールチップとハイライト

### 31.3 練習問題

#### 練習問題

**問題1：**売上ダッシュボードを作成してください。棒グラフ、折れ線グラフ、円グラフを含め、フィルタで連携させてください。

**問題2：**月を選択すると全てのチャートが更新されるようにしてください。

#### 模範解答

**問題1 の解答：**

```
1
2 function updateAll(filter) {
3   currentFilter = filter;
4   const filteredData = filter
5     ? data.filter(d => d.category === filter)
6     : data;
7
8   updateBarChart(filteredData);
9   updateLineChart(filteredData);
10  updatePieChart(filteredData);
11 }
12
13 // 各チャートからフィルタを呼び出し
14 svg.selectAll("rect").on("click", (event, d) => {
15   updateAll(d.category);
16 });
```

**問題2 の解答：**

```
25 const month = this.value;
26 const monthData = data.filter(d => d.month === month);
27
28 updateBarChart(monthData);
29 updateLineChart(data, month); // ハイライト付き
30 updatePieChart(monthData);
31 updateKPIs(monthData);
32});
```

第 V 部

LLM を活用した学習



## 第 32 章

# 最短で上達するプロンプトセット 30 本

この章では、ChatGPT や Claude などの大規模言語モデル（LLM）を活用して D3.js を効率的に学習するためのプロンプトセットを紹介します。各プロンプトは「身につく概念」「よくある誤り」「到達目標」を含み、体系的な学習を支援します。

### この章の使い方

1. 各プロンプトを LLM（ChatGPT、Claude 等）に入力する
2. 生成されたコードを実際に動かして確認する
3. 「よくある誤り」を参考にデバッグ力を養う
4. 「到達目標」でセルフチェックを行う
5. 改造課題に取り組んで応用力を身につける

### LLM 活用の心構え

LLM は答えを得る道具ではなく、試行錯誤を早回しする道具です。

生成されたコードは必ず読んで、「なぜこう書くのか」を説明できる状態にしてください。「動いた」だけで次に進むと、応用が効かなくなります。

### 30 本共通のチェックポイント

すべてのプロンプトに共通するルールです。迷ったときはここに戻ってください。

- **D3.js v7 前提**：生成コードは D3.js v7 を前提とします
- **単一 HTML の場合**：`<script src="https://d3js.org/d3.v7.min.js"></script>` を使用
- **外部データを読む場合**：ローカルサーバが必要です（第 1 章「実行環境の準備」参照）
- **迷ったら**：「到達目標」を先に読んでゴールを確認する
- **動かないとき**：「よくある誤り」をデバッグ観点として活用する

### 用語の定義

**LLM** Large Language Model（大規模言語モデル）の略。ChatGPT、Claude、Geminiなど  
の総称。本章では「LLM」で統一します。

**A11y** Accessibility（アクセシビリティ）の略記。「A」と「y」の間に11文字あることから。  
キーボード操作対応やスクリーンリーダー対応など、障害の有無にかかわらず利用でき  
る設計を指します。

**D3.js v7** 本書のプロンプトは D3.js v7 を前提としています。v6 以前とは API が異なる部分  
があります。

### 本文との対応表

各プロンプトが本書のどの章に対応するかを示します。復習や先取り学習の参考にしてください。

プロンプト	テーマ	対応する本文
#01	Hello D3 / SVG 導入	第2章・第3章
#02	margin convention	第5章・第6章
#03	データ結合 (data join)	第4章
#04	スケールと軸	第5章・第6章
#05	遷移とイージング	第10章
#06	棒グラフ基本	第7章
#07	外部データ読み込み・型変換	第1章
#08	レスポンシブ対応	第18章
#09	ツールチップ	第17章
#10	折れ線グラフ + エリア	第8章・第13章
#11	散布図 + 回帰直線	第12章
#12	積み上げ棒グラフ	第15章
#13	Enter-Update-Exit	第4章・第19章
#14	ブラシ選択	第21章
#15	ズーム&パン	第20章
#16	階層データ (Treemap)	第23章・第24章
#17	力学シミュレーション	第22章
#18	地図描画 (GeoJSON)	第28章
#19	ダッシュボード統合	第31章
#20	ヒートマップ + カラースケール	第29章
#21	レーダーチャート	第30章
#22	Sankey Diagram	応用トピック
#23	アニメーション連鎖	第10章 (応用)
#24	マルチシリーズ折れ線	第14章
#25	Canvas 性能最適化	上級トピック
#26	アクセシビリティ対応	上級トピック
#27	React 統合	上級トピック
#28	TypeScript 型定義	上級トピック
#29	成果物出力 (PNG/SVG)	上級トピック
#30	コードレビュー観点	総仕上げ

## 推奨の進め方

以下は目安となる学習プランです。ご自身のペースに合わせて調整してください。

### 学習プランの例

#### 【集中プラン】2週間で一巡

- 1週目：★1～★2 (#01～#12) — 基礎固め。1日2本ペース
- 2週目：★3～★4 (#13～#24) — 応用力養成。1日2本ペース
- 仕上げ：★5 (#25～#30) — 実務品質。時間をかけて取り組む

#### 【じっくりプラン】4週間で定着

- 1週目：★1 (#01～#06) — D3 の書き方に慣れる
- 2週目：★2 (#07～#12) — 実用の入口
- 3週目：★3 (#13～#18) — 更新設計を覚える
- 4週目：★4～★5 (#19～#30) — 高度な操作と実務品質

ポイント：各プロンプトの「改造課題」まで取り組むと、定着度が大きく上がります。

## 32.1 ★ 1:D3 の書き方に慣れる (#01～#06)

基本的な D3.js の構文と SVG の扱い方を身につけます。

### 32.1.1 【#01】最小の「Hello D3」テンプレを作る

#### プロンプト

1 あなたはD3.js v7の講師です。初学者向けに、次を満たす  
2 「单一HTML」を作ってください。  
3  
4 要件：  
5 - <script src="https://d3js.org/d3.v7.min.js"></script> を使う  
6 - 画面中央に "Hello D3" をSVGテキストで描画  
7 - 背景は薄いグリッド（線でも点でも良い）を描いて  
8 「SVG座標」を意識できるようにする  
9 - コメントを多めに（初学者がコピペして理解できるように）  
10  
11 出力：  
12 1) 5行以内の超短い解説（何が起きているか）  
13 2) 完全に動く单一HTML（コピペ実行できる）  
14 3) 動作確認チェックリスト（3項目）  
15 4) 改造課題（3つ）

### 身につく概念（用語ミニ辞書）

SVG ベクタ図形を描くための XML。D3 の基本キャンバス。

Selection（選択）`d3.select()` で DOM 要素をつかむ仕組み。

`append` 要素の追加（例：`svg.append("text")`）。

`attr / style` 属性・スタイルを設定する操作（位置、サイズ、色など）。

座標系 SVG の (x,y) の基準（左上が (0,0)）。

### よくある誤りと直し方

- 何も表示されない

- 原因：`<script>` の読み込み順（D3 より先に描画コードが走る）

- 対策：D3 読み込み後に描画コードを書く／`defer` を使う

- 文字が見えない

- 原因：`fill` が背景と同色、座標が画面外

- 対策：`attr("fill", "black")`、x/y を中央に再計算

- SVG サイズが 0

- 原因：CSS で潰れている、width/height 未指定

- 対策：`width/height` を明示、または `viewBox` を設定

### 到達目標（チェック基準）

- ブラウザでエラーなく表示される（コンソールが赤くない）
- SVG 上に ”Hello D3” が中央付近に表示される
- グリッドが描画され、座標感覚がつかめる
- `d3.select → append → attr` の流れがコードで確認できる

#### 32.1.2 【#02】margin convention（余白設計）を身体で覚える

##### プロンプト

```
1 D3 v7 で、margin convention の基本形を「テンプレ」として
2 作ってください（単一HTML）。
3
4 要件：
5 - outer svg と inner g (translate(margin.left, margin.top)) を作る
6 - inner 領域の四隅に小さな丸を置いて
7 「ここが描画領域」と分かるようにする
8 - width/height/margin は冒頭の定数にまとめる
9 - コメント多め
10
11 出力：
12 1) なぜ margin convention が必要か（初学者向けに）
13 2) 動く単一HTML
14 3) 改造課題（軸を付ける等、3つ）
```

### 身につく概念（用語ミニ辞書）

Margin convention 軸やラベル用の余白を確保し、描画領域を明確化する定石。

outer svg / inner g 外枠（全体）と内側の描画領域（余白を除いた部分）。

translate g を平行移動して余白分をずらす。

innerWidth/innerHeight 実際にデータを描く領域の幅/高さ。

### よくある誤りと直し方

- 軸やラベルが切れる

- 原因：margin を取らずにそのまま描いている
- 対策：必ず inner = svg.append("g").attr("transform", ...)

- 座標がズレる

- 原因：outer 基準の (x,y) で inner に描いている
- 対策：図形は inner 座標で描く (= margin 分を意識しないで済む)

- innerWidth を使ってない

- 原因：range が outer 幅のまま
- 対策：スケールの range([0, innerWidth]) に統一

### 到達目標（チェック基準）

- outer/inner の 2 層構造になっている
- inner 領域の四隅マーカーが正しい位置に描かれる
- 定数 (width/height/margin) が冒頭にまとまっている
- inner 基準で座標が扱われている (margin を足し引きしない)

### 32.1.3 【#03】データ結合（data join）の最小：円を並べる

#### プロンプト

```

1 D3 v7で「配列データを円として描画」する最小例を、
2 初学者向けに作ってください（単一HTML）。
3
4 データ（コード内に直書き）：
5 const data = [4, 8, 15, 16, 23, 42];
6
7 要件：
8 - data join (selectAll → data → join) を必ず使う
9 - 円のx位置はインデックスで等間隔、半径は値に比例
10 - 値ラベル（text）も描く（円の上に数値表示）
11
12 出力：
13 - 解説（data joinとは何か）
14 - 動く単一HTML
15 - チェックリスト
16 - 改造課題（3つ）

```

#### 身につく概念（用語ミニ辞書）

Data join データ配列と DOM 要素を結びつける D3 の中核概念。

selectAll まだ存在しない要素も”選ぶ”ことで、後から生成できる。

join enter/update/exit をまとめて扱う現代的な書き方。

Index 配列の順番（0,1,2…）。位置決めの最小材料。

#### よくある誤りと直し方

- 円が増え続ける（重複する）
  - 原因：append だけで描いている（joinしていない）
  - 対策：必ず selection.data(data).join("circle")
- text が円とズレる
  - 原因：同じ x/y 計算を共有していない
  - 対策：位置計算を関数化／同じスケールで計算
- 半径が大きすぎて重なる
  - 原因：値をそのまま r している
  - 対策：d3.scaleLinear で r 用スケールを作る

#### 到達目標（チェック基準）

- selectAll → data → join の形がコードにある

- データ数と同じ数の円が描画される
- 半径が値に応じて変化している
- 数値ラベルが各円に対応して表示される

### 32.1.4 【#04】スケールを理解：linear / band を見比べる

#### プロンプト

```
1 D3 v7で、linear scale と band scale の違いが一目で分かる
2 教材を作ってください（単一HTML）。
3
4 要件：
5 - 上段：linear scale (0～100 → 0～innerWidth) で点を配置
6 - 下段：band scale (カテゴリA～E → 0～innerWidth) で矩形を配置
7 - それぞれ「入力値」「出力座標」をconsole.logでも表示
8 - 初学者向けコメント多め
9
10 出力：
11 1) スケールの役割をたとえ話で説明
12 2) 動く単一HTML
13 3) 改造課題 (domain/rangeを変える等)
```

#### 身につく概念（用語ミニ辞書）

Scale（スケール）データの値を画面座標へ変換する「変換器」。

domain 入力（データ側）の範囲や集合。

range 出力（画面側）の範囲。

scaleLinear 連続値向け（数値・時間など）。

scaleBand カテゴリ向け（棒グラフの軸など）。bandwidth() が重要。

#### よくある誤りと直し方

- 点が全部左に固まる
  - 原因：domain が合っていない（例：0～1なのに 0～100を入れる）
  - 対策：domain をデータに合わせて設定、d3.extent 利用
- 棒の幅が不明
  - 原因：bandwidth を使っていない
  - 対策：x.bandwidth() を rect の width に使う
- range を outer 幅で設定
  - 原因：margin 分を考慮していない
  - 対策：[0, innerWidth] にする

### 到達目標（チェック基準）

- 上段に linear で配置された点が見える
- 下段に band で配置された矩形が見える
- console に入力値→出力座標が表示される
- domain/range がコード上で明確に分かる

#### 32.1.5 【#05】軸（axis）の最小：tick とフォーマット

##### プロンプト

```

1 D3 v7で、x軸とy軸を表示する最小の教材を作ってください
2 (单一HTML)。
3
4 要件：
5 - x: band scale (カテゴリ)
6 - y: linear scale (値)
7 - tickの数を調整し、値は桁区切り (例: 1,234) にフォーマット
8 - グリッド線 (y方向) も付ける
9 - margin conventionを使う
10
11 出力：
12 - 解説 (axisBottom/axisLeftが何をするか)
13 - 動く单一HTML
14 - 改造課題 (単位表示、tick間隔など)

```

### 身につく概念（用語ミニ辞書）

Axis generator スケールをもとに目盛りを描く道具 (d3.axisLeft 等)。

tick 目盛り線とラベル。

tickFormat ラベル文字列の整形 (桁区切り、% など)。

Gridlines 補助線。読み取りを助けるが多すぎると邪魔。

### よくある誤りと直し方

- 軸が表示されない
  - 原因：軸を call() していない
  - 対策：g.call(d3.axisLeft(y))
- グリッドがズレる
  - 原因：軸用 g の transform が間違い
  - 対策：x 軸は下へ translate、y 軸は左へ配置
- tick が多すぎ / 少なすぎ
  - 原因：デフォルト任せ

- 対策: ticks(n)、時間なら d3.timeMonth.every(1) など

### 到達目標（チェック基準）

- x 軸と y 軸が描画されている
- y 軸ラベルが桁区切りフォーマットになっている
- y 方向のグリッド線が表示されている
- margin convention が使われている

#### 32.1.6 【#06】棒グラフ（静的）を完成させる

##### プロンプト

```
1 D3 v7で、初学者向けの縦棒グラフを単一HTMLで作ってください。
2
3 データ（直書き）：
4 const data = [
5   {name: "A", value: 12},
6   {name: "B", value: 5},
7   {name: "C", value: 18},
8   {name: "D", value: 9},
9   {name: "E", value: 15}
10 ];
11
12 要件：
13 - 軸、グリッド、バー、バー上の値ラベル
14 - 0のベースラインを意識（y=0）
15 - 見た目はシンプルで良いが、コメントで読み解けるように
16
17 出力：
18 - 動く単一HTML
19 - チェックリスト（軸/スケール/マージン）
20 - 改造課題（横棒にする等）
```

### 身につく概念（用語ミニ辞書）

Bar chart カテゴリ比較の基本図。

scaleBand + scaleLinear 棒の位置と高さの定番組み合わせ。

baseline（基準線）y=0 の位置。負の値対応の基礎。

rect 棒（四角形）を描くSVG要素。

### よくある誤りと直し方

- 棒が上下逆

- 原因: SVG は上が 0 で下が増える、y 計算を誤る
- 対策: `y = yScale(value)`、`height = yScale(0) - yScale(value)`
- 棒がはみ出す
  - 原因: range が outer サイズ
  - 対策: inner サイズに統一
- 値ラベルが棒の中に埋もれる
  - 原因: y 位置が同じ
  - 対策: `yScale(value) - 4` など少し上へ

### 到達目標（チェック基準）

- バー・軸・グリッド・値ラベルが揃っている
- 0 基準が正しく（少なくとも暗黙に）成立している
- 値が変わったときも破綻しにくい（スケール使用）
- コードに「データ→スケール→描画」の流れが読み取れる

## 32.2 ★ 2：実用の入口（#07～#12）

外部データ読み込み、レスポンシブ対応、インタラクションの基礎を学びます。

### 32.2.1 【#07】 CSV 読み込み+型変換+欠損除外

#### プロンプト

```

1 D3 v7でCSVを読み込む教材を作ってください（单一HTML）。
2
3 要件：
4 - d3.csv() を使う（URL指定と、サンプルCSVをBlobで読む2パターン）
5 - 列: date,value,category を想定
6 - dateはDate型、valueはnumberに変換
7 - 変換失敗や欠損行は除外し、除外件数をconsoleに出す
8 - 変換後データをconsole.tableで確認
9
10 出力：
11 1) 型変換が必要な理由
12 2) 動く单一HTML（BlobのサンプルCSV内蔵で動くように）
13 3) 改造課題（列追加、欠損補完など）

```

### 身につく概念（用語ミニ辞書）

d3.csv CSV を読み、配列（オブジェクト）にする。

Row accessor 読み込み時に各行を変換する関数（型変換の定石）。

型変換（parse）文字列→数値/日付へ（可視化の前提）。

欠損値 (missing) null/空文字/NaN などの扱い。

console.table 整形表示でデータを検査する方法。

### よくある誤りと直し方

- 数値が文字列のまま
  - 症状: d3.extent が変、スケールが崩れる
  - 対策: +d.value や Number(d.value) を徹底
- Date に変換できない
  - 原因: フォーマット違い (YYYY/MM/DD など)
  - 対策: d3.timeParse("%Y-%m-%d") 等で明示
- 除外行が多すぎる
  - 原因: 前処理条件が厳しすぎる/空白トリム不足
  - 対策: String(v).trim()、除外理由をログ

### 到達目標（チェック基準）

- Blob 内蔵 CSV でも動作する（外部ファイル不要で再現）
- date が Date 型、value が number になっている（console で確認）
- 欠損・変換失敗行が除外され、件数がログに出る
- ”可視化前のデータ検査” の重要性が説明できる

## 32.2.2 【#08】 レスポンシブ (ResizeObserver) で追従する棒グラフ

プロンプト

```
1 D3 v7で、ResizeObserverを使い「親要素の幅に追従する棒グラフ」
2 を作ってください（単一HTML）。
3
4 要件：
5 - コンテナdivの幅に合わせてSVGを再計算
6 - リサイズ時にスケール・軸・バーを再描画（update関数化）
7 - 高さは固定でも可
8 - コメント多め（どこを触ると何が変わるか）
9
10 出力：
11 - 解説（なぜviewBoxだけでは足りないケースがあるか）
12 - 動く単一HTML
13 - 改造課題（高さも追従、余白自動調整など）
```

### 身につく概念（用語ミニ辞書）

ResizeObserver 要素サイズ変化を検知する API。

レスポンシブ 画面/親要素サイズに合わせて再描画する設計。

update 関数 サイズやデータ変更に応じて描画を更新する核。

再計算ポイント innerWidth、スケール range、軸、rect 位置/サイズ。

### よくある誤りと直し方

- リサイズで棒がズレる

- 原因：range は更新しているが axis/rect が更新されていない
- 対策：update 内で「スケール→軸→図形」を再実行

- ResizeObserver が無限ループ

- 原因：observer 内で DOM サイズを変えてしまう
- 対策：サイズを変える処理を避ける（SVG 属性更新は OK だが注意）

- 毎回 append して増殖

- 原因：join を使わず都度 append
- 対策：join で更新、固定要素は 1 回だけ生成

### 到達目標（チェック基準）

- ウィンドウ幅変更でグラフが追従する
- update 関数に更新処理が集約されている
- リサイズ後も軸・バー・ラベルが正しい位置
- 要素が増殖しない（DOM が増え続けない）

### 32.2.3 【#09】ツールチップ（はみ出し防止）を付ける

前提：#06「棒グラフ基本」を完了していること。#06で作成した棒グラフをベースに進めます（第8章参照）。

#### プロンプト

```
1 #06の棒グラフに、divツールチップを追加してください
2 (単一HTML、D3 v7)。
3
4 要件：
5 - hoverでツールチップ表示 (name/value)
6 - マウス追従
7 - 画面端でははみ出さない (左右上下で位置を反転するロジック)
8 - キーボード操作: Tabでバーにフォーカス、Enterでツールチップ固定/解除
9 - aria-labelも付ける (簡易でOK)
10
11 出力：
12 - 実装方針 (イベント、座標計算)
13 - 完全なHTML
14 - 改造課題 (ツールチップに小さなミニチャート等)
```

#### 身につく概念（用語ミニ辞書）

Tooltip ホバー/フォーカスで詳細を出すUI。

pointer 座標 マウス位置 (D3ではd3.pointer(event)等)。

イベント mouseover/mousemove/mouseout、focus/blur/keydown。

アクセシビリティ (A11y) キーボードや読み上げへの配慮。

aria-label 要素の意味をスクリーンリーダーに伝える属性。

#### よくある誤りと直し方

- ツールチップがズレる
  - 原因：ページ座標とSVG座標を混同
  - 対策：div tooltip は clientX/clientY 基準で配置するのが簡単
- 端で見切れる
  - 原因：反転ロジックなし
  - 対策：tooltip の getBoundingClientRect() で端判定して反転
- キーボードで表示できない
  - 原因：バーに tabindex がない
  - 対策：attr("tabindex", 0) + focuskeydown 対応

### 到達目標（チェック基準）

- マウスホバーで詳細が表示される
- 画面端でもツールチップが見切れない
- Tab でバーにフォーカス移動できる
- Enter で固定/解除できる（最低限どちらか）
- aria-label 等の説明属性が付いている

#### 32.2.4 【#10】ソートボタン（値降順）+アニメーション

##### プロンプト

```

1 棒グラフに「値降順に並べ替え」ボタンを付けてください
2 (D3 v7、単一HTML)。
3
4 要件：
5 - ボタン押下でxスケールのdomainを並べ替え
6 - バーが滑らかに移動するトランジション
7 - 軸も更新
8 - 初学者向けに「どこが更新ポイントか」コメントで説明
9
10 出力：
11 - 動く単一HTML
12 - チェックリスト (domain更新、transition対象)
13 - 改造課題 (昇順/元に戻す、複数条件など)

```

### 身につく概念（用語ミニ辞書）

domain 更新 並び替え = band scale の domain 順序変更。

transition 属性変化を時間で補間してアニメ化。

再描画 軸とバー位置の両方を更新する必要がある。

状態 (state) 現在の並び順（降順/昇順/元）を保持する考え方。

### よくある誤りと直し方

- 軸だけ並び替わる/バーだけ並び替わる
  - 原因：更新対象が片方だけ
  - 対策：axisG.transition().call(...) と bars.transition().attr("x", ...)
- 並び替えでバーがワープ
  - 原因：transition 開始前に属性が確定していない
  - 対策：transition を同じ duration で揃え、更新順を整理
- ラベルが追従しない
  - 原因：text 位置更新が漏れている

- 対策：バーと同様に text も attr("x", ...) 更新

### 到達目標（チェック基準）

- ボタンで降順並び替えができる
- バーが滑らかに移動する（transition）
- 軸ラベルも正しく更新される
- 値ラベル（ある場合）も棒に追従する

## 32.2.5 【#11】時系列折れ線：欠損で線を切る + 移動平均

### プロンプト

```

1 D3 v7で時系列の折れ線グラフ教材を作ってください（単一HTML）。
2
3 データ（直書き、欠損あり）：
4 const data = [
5   {date:"2025-01-01", value: 10},
6   {date:"2025-01-02", value: 12},
7   {date:"2025-01-03", value: null},
8   {date:"2025-01-04", value: 18},
9   {date:"2025-01-05", value: 17}
10 ];
11
12 要件：
13 - dateをDateに変換
14 - valueがnullの点は線を切る（line.defined）
15 - 3点移動平均の線を別で重ねる（欠損考慮）
16 - ツールチップ（近い点を探す簡易でOK）
17
18 出力：
19 - 解説（defined、time scale、移動平均の考え方）
20 - 動く単一HTML
21 - 改造課題（7日平均、面グラフ追加など）

```

### 身につく概念（用語ミニ辞書）

d3.line 折れ線を生成する関数（x/y アクセサで定義）。

defined 欠損点を線から除外し「切れ目」を作る。

scaleTime 時間軸用スケール。

移動平均 ノイズをならす集計（窓幅、欠損への配慮が必要）。

近傍点探索 ホバー位置に近いデータ点を探す（bisector 等の入口）。

## よくある誤りと直し方

- 欠損でも線がつながる
  - 原因: defined 未使用/条件が誤り
  - 対策: line.defined(d => d.value != null && !isNaN(d.value))
- 日付がうまく並ばない
  - 原因: date が文字列のまま
  - 対策: 必ず Date に変換し、必要なら sort
- 平均線が変
  - 原因: 欠損を 0 扱いしている
  - 対策: 窓内の有効値だけで平均し、データ不足なら null

## 到達目標（チェック基準）

- 欠損箇所で線が切れている
- 移動平均線が別スタイルで重ね描きされている
- date が Date 型になっている（console 確認）
- 軸が time/linear で適切に表示される

### 32.2.6 【#12】散布図：色分け＋ホバーで強調

#### プロンプト

```

1 D3 v7で散布図を作ってください（单一HTML、初学者向け）。
2
3 データ（直書き）：
4 const data = [
5   {x: 5, y: 20, group:"A", name:"p1"}, 
6   {x: 10, y: 35, group:"A", name:"p2"}, 
7   {x: 8, y: 10, group:"B", name:"p3"}, 
8   {x: 15, y: 30, group:"B", name:"p4"}, 
9   {x: 20, y: 25, group:"C", name:"p5"} 
10 ];
11
12 要件：
13 - x,yはlinear scale
14 - groupで色分け（ordinal scale）
15 - hoverした点だけ強調（半径UP・不透明UP）、他は薄く
16 - 凡例を表示（最小でOK）
17
18 出力：
19 - 動く单一HTML
20 - チェックリスト（スケール、色、凡例）
21 - 改造課題（回帰直線、バブル化など）

```

### 身につく概念（用語ミニ辞書）

Scatter plot 相関・クラスタを見る基本図。

scaleOrdinal カテゴリ→色などの対応表。

強調（highlight）注目対象だけ視覚的優先度を上げる。

Legend（凡例）色/形の意味を説明する部品。

### よくある誤りと直し方

- 点が画面外
  - 原因：domain 設定ミス（extent 未使用）
  - 対策：d3.extent(data, d=>d.x) で domain を作る
- 凡例と色が一致しない
  - 原因：color scale の domain が不安定
  - 対策：domain([...new Set(data.map(d=>d.group))]) を明示
- ホバーでガクガク
  - 原因：毎回全点の style を重く変更
  - 対策：class 切替や opacity 変更を最小に

### 到達目標（チェック基準）

- group で色分けがされ、凡例が説明している
- ホバーで対象点が強調される
- 非対象点が薄くなり、視線誘導ができる
- 軸・スケールが妥当に設定されている

## 32.3 ★ 3：更新設計を覚える (#13～#18)

データの動的更新とアニメーションの設計パターンを学びます。

### 32.3.1 【#13】enter/update/exit を明示した「更新できる棒グラフ」

#### プロンプト

```

1 D3 v7で「更新できる棒グラフ」を作ってください（単一HTML）。
2
3 要件：
4 - createChart(container) と update(data) を分ける
5 - データ更新ボタンを2つ用意し、データ件数が増減するケースを作る
6 - joinのkey関数を必ず指定（nameをキー）
7 - enter/update/exit が分かるように、追加=緑、更新=青、削除=赤など説明
8 - 初学者向けに、なぜkeyが必要かを丁寧に
9

```

```
10 出力：  
11 - 解説（joinとkey、exitの意味）  
12 - 完全なHTML  
13 - 改造課題（ソート、フィルタ、アニメなど）
```

### 身につく概念（用語ミニ辞書）

enter/update/exit 追加・更新・削除の3状態。

key 関数 データの”同一性”を決める（DOM要素の対応を安定化）。

create/update 分離 初期構築と更新処理を分ける設計（実務の基本）。

安定した更新 並び替え・追加削除しても”同じ棒”が同じデータに紐づく。

### よくある誤りと直し方

- 更新のたびに棒が増殖
  - 原因：join せず append
  - 対策：selection.data(data, d=>d.name).join(...)
- 棒が別データに入れ替わる
  - 原因：key がない（index で対応される）
  - 対策：必ず d=>d.name 等の一意キー
- update で軸が更新されない
  - 原因：軸 call が create 側に固定
  - 対策：update 内で axisG.call(...) を実行

### 到達目標（チェック基準）

- createChart と update が分離されている
- 2種類以上のデータに切替でき、増減に対応する
- key 関数が指定されている
- enter/update/exit の挙動が目視できる（説明がある）

### 32.3.2 【#14】更新トランジション（数値・位置・軸）を整える

前提：#13「Enter-Update-Exit」を完了していること。#13で作成した更新棒グラフをベースに進めます。

#### プロンプト

1 #13の更新棒グラフを、トランジション込みで完成度を上げてください。

2

3 要件：

4 - 追加バーは下から伸びる

5 - 更新バーは高さが滑らかに変わる

6 - 削除バーは縮んで消える

7 - 軸もトランジションで更新

8 - トランジションが衝突しないように設計 (interrupt等の説明も軽く)

9

10 出力：

11 - 動く単一HTML

12 - よくある失敗例（ガタつき、軸だけ遅れる等）と対処

13 - 改造課題（イージング、遅延など）

### 身につく概念（用語ミニ辞書）

transition 更新の差分を視覚的に伝える。

duration/ease アニメ時間と補間カーブ。

interrupt 連打などで進行中 transition を止める/上書きする。

同期更新 軸と図形の更新を同じタイミングで揃える。

### よくある誤りと直し方

- 軸が遅れてズレる

- 原因：軸とバーの duration が異なる
- 対策：同じ duration を共有（変数にする）

- 追加時に棒が一瞬上に出る

- 原因：enter 初期属性が最終値
- 対策：enter で `y=y(0), height=0` から開始

- 連打でガタつく

- 原因：transition が積み重なる
- 対策：`selection.interrupt()`、ボタン連打を抑制

### 到達目標（チェック基準）

- 追加・更新・削除がそれぞれ滑らかにアニメ化
- 軸も同時に更新される
- 連打しても破綻しにくい（最低限の対策がある）
- enter 初期値→遷移→最終値がコードで明確

### 32.3.3 【#15】複数系列折れ線：凡例クリックでON/OFF

#### プロンプト

```

1 D3 v7で「複数系列の折れ線」を作ってください（単一HTML）。
2
3 データ（直書き）：
4 const data = [
5   {date:"2025-01-01", A:10, B:5, C:7},
6   {date:"2025-01-02", A:12, B:6, C:9},
7   {date:"2025-01-03", A:8, B:7, C:6},
8   {date:"2025-01-04", A:15, B:10, C:11}
9 ];
10
11 要件：
12 - 列A/B/Cは「系列」として動的に抽出（将来列が増えてもOK）
13 - 凡例を自動生成し、クリックで系列表示ON/OFF
14 - yスケールは表示中系列の最大最小に追従（更新で再計算）
15 - 初学者向けにデータ変換（wide→series配列）の説明
16
17 出力：
18 - 動く単一HTML
19 - チェックリスト（系列抽出、凡例、再計算）
20 - 改造課題（片対数、ツールチップ強化など）

```

#### 身につく概念（用語ミニ辞書）

wide → long 変換（系列化）列ごとの値を「系列の配列」に変形。

domain 再計算 表示中系列だけでy範囲を取り直す。

凡例の自動生成 color.domain() からUIを作る発想。

可変系列対応 系列追加/削除でも壊れない設計。

#### よくある誤りと直し方

- 系列がハードコード
  - 原因：A/B/Cを固定で書いている
  - 対策：Object.keys(data[0]).filter(k => k!="date")
- y軸が追従しない
  - 原因：y domain 更新後に軸 callしていない
  - 対策：ON/OFF のたびに y.domain(...) → axisG.call(...)
- ON/OFF で線が増殖
  - 原因：更新時にappendし直し
  - 対策：系列パスもjoinで管理

### 到達目標（チェック基準）

- 系列数が増えても凡例・線が自動生成される
- 凡例クリックで ON/OFF できる
- y 軸が表示中系列に追従する
- 更新で要素が増殖しない

#### 32.3.4 【#16】ヒストグラム：d3.bin と bin 設計

##### プロンプト

```
1 D3 v7でヒストグラム教材を作ってください（単一HTML）。
2
3 要件：
4 - ランダムデータを生成（正規分布っぽくしてOK）
5 - d3.binでビニングし、棒として描画
6 - ビン数をスライダーで変えられる（10～60）
7 - ビン数で見え方が変わることを解説
8
9 出力：
10 - 動く単一HTML
11 - 初学者向け解説（ビン幅/ビン数のトレードオフ）
12 - 改造課題（密度曲線、外れ値表示など）
```

### 身につく概念（用語ミニ辞書）

d3.bin 連続値を区間（ビン）に分ける。

thresholds ビン数や区切りの指定。

分布（distribution）データのばらつき・形状を見る視点。

トレードオフ ビン多い=細かいがノイズ、少ない=滑らかだが粗い。

### よくある誤りと直し方

- ビンの幅がズレる
  - 原因：domain をデータ範囲と合わせてない
  - 対策：bin.domain(x.domain())
- 棒が途切れる
  - 原因：band スケールの扱いを間違う
  - 対策：ビンは連続区間なので x は linear で、rect は  $x(x1)-x(x0)$  で幅計算
- スライダー変更で増殖
  - 原因：更新で append し直し
  - 対策：join で rect 更新

### 到達目標（チェック基準）

- スライダーで BIN 数が変わり再描画される
- d3.bin の結果を使って棒が描けている
- BIN 数の違いで見え方が変わる説明が付いている
- 更新で要素が増殖しない

### 32.3.5 【#17】 積み上げ棒 & 100% 積み上げ（切替）

#### プロンプト

```

1 D3 v7 で 「積み上げ棒」と「100% 積み上げ棒」を切替できる
2 教材を作ってください（単一 HTML）。
3
4 データ（直書き）：
5 const data = [
6   {name:"A", x:30, y:20, z:10},
7   {name:"B", x:10, y:25, z:15},
8   {name:"C", x:20, y:10, z:30}
9 ];
10
11 要件：
12 - d3.stack を使う
13 - トグルボタンで「通常」 ⇄ 「100%」を切替
14 - ツールチップで「値」と「割合」を両方表示
15 - 初学者向けに、stack が返す構造を説明（console.log も可）
16
17 出力：
18 - 動く単一 HTML
19 - 改造課題（凡例 ON/OFF、並べ替え等）

```

#### 身につく概念（用語ミニ辞書）

d3.stack 系列を積み上げ用の [y0,y1] 区間に変換。

正規化（normalize）合計=1 (=100%) にする前処理。

系列（series）x,y,z などの内訳カテゴリ。

tooltip で値/割合 絶対値と相対値を同時に扱う実務力。

#### よくある誤りと直し方

- 100% で合計が 100 にならない
  - 原因：正規化の分母がカテゴリ合計でない
  - 対策：各 name ごとに合計を計算し各系列を割る
- stack の構造が分からぬ

- 原因：stack 結果を見ていらない
- 対策：console.log(stacked) で確認、説明コメントを付ける
- 軸の最大値が固定で見切れる
  - 原因：y domain 更新漏れ
  - 対策：通常は最大合計、100% は [0,1] に切替

### 到達目標（チェック基準）

- 通常/100% をトグルで切替できる
- ツールチップで値と割合が確認できる
- stack を使っている（自前積み上げではない）
- y 軸スケールが切替に追従する

### 32.3.6 【#18】スモールマルチプル（小さな複数グラフ）

#### プロンプト

```

1 D3 v7でスモールマルチプル（小さな折れ線をカテゴリごとに並べる）
2 を作ってください（単一HTML）。
3
4 要件：
5 - サンプルデータを生成（カテゴリ3～6、各カテゴリに時系列10点）
6 - グリッド状に配置
7 - スケールは「共通」と「カテゴリ別」を切替できる
8 - 小さくても読みやすい工夫（軸の間引き、ラベル位置など）を解説
9
10 出力：
11 - 動く単一HTML
12 - 初学者向けにfacetの考え方説明
13 - 改造課題（並び替え、ハイライトなど）

```

#### 身につく概念（用語ミニ辞書）

Facet（ファセット）同じ形式のグラフを条件別に並べて比較する。

共通スケール/個別スケール 比較の公平性 vs 各系列の見やすさ。

レイアウト 小さなチャートを格子状に配置する設計。

情報密度 小さい領域に何を残し何を捨てるかの判断。

#### よくある誤りと直し方

- 軸が多すぎて読めない
  - 原因：各小グラフに完全な軸
  - 対策：外側だけ軸、または tick を間引く

- 共通/個別の切替が中途半端
  - 原因：y domainだけ切替、線はそのまま等
  - 対策：切替時にスケール→軸→線の順で更新
- カテゴリ順がバラバラ
  - 原因：Set順序依存
  - 対策：カテゴリをソートして固定順に

#### 到達目標（チェック基準）

- カテゴリごとに小さな折れ線が並ぶ
- 共通/個別スケールを切替できる
- 読みやすさの工夫（軸間引き等）が説明されている
- 小グラフ間で比較がしやすい

### 32.4 ★ 4：高度な操作と複合（#19～#24）

ブラッシング、ズーム、連動、階層データなど高度なテクニックを学びます。

#### 32.4.1 【#19】ブラッシング：散布図で範囲選択＋選択リスト表示

プロンプト

```
1 D3 v7で散布図にブラッシングを付け、選択された点を
2 下のHTMLテーブルに表示してください（単一HTML）。
3
4 要件：
5 - d3.brushで矩形選択
6 - 選択中は点を強調、非選択は薄く
7 - 選択解除ボタンを用意
8 - 選択結果をテーブル（name, x, y, group）で表示
9
10 出力：
11 - 動く单一HTML
12 - brushの座標系（inner座標）を初学者向けに解説
13 - 改造課題（複数回選択の蓄積など）
```

### 身につく概念（用語ミニ辞書）

d3.brush 範囲選択 UI（矩形、xのみ、yのみ等）。

extent ブラシ可能領域（inner 座標で指定）。

選択状態 選ばれた点集合を保持し、見た目と表に反映。

ビューとデータの連携 図の操作が別 UI（表）に反映される。

### よくある誤りと直し方

- 選択がずれる

- 原因：outer 座標で判定している
- 対策：inner 座標に統一、必要なら margin を考慮

- 選択解除ができない

- 原因：brush のクリア方法不明
- 対策：brushG.call(brush.move, null) やボタンから呼ぶ

- 表が更新されない

- 原因：selection イベントで DOM 更新していない
- 対策：brush.on("brush end", ({selection}) => ...) で更新

### 到達目標（チェック基準）

- 矩形選択で点が絞り込まれる
- 選択結果が HTML テーブルに反映される
- 選択解除ボタンが機能する
- 座標系（inner 基準）の説明がある

### 32.4.2 【#20】ズーム＆パン：時系列の拡大縮小

#### プロンプト

```

1 D3 v7で時系列折れ線にズーム＆パンを実装してください（単一HTML）。
2
3 要件：
4 - d3.zoomを使用
5 - ズームに合わせてx軸tickを更新
6 - 線もズームに追従（transformを当てるだけでなく、再計算方式も説明）
7 - クリップ（clipPath）で描画領域外を隠す
8
9 出力：
10 - 動く单一HTML
11 - 初学者向け解説（zoomTransformとrescaleX）
12 - 改造課題（yもズーム、ダブルクリックリセット等）

```

#### 身につく概念（用語ミニ辞書）

d3.zoom 拡大縮小・移動の操作を扱う。

zoomTransform 現在の拡大率・平行移動量。

rescaleX 元のスケールをズーム変換後のスケールに変換。

clipPath 描画領域外の線や点を隠す仕組み。

#### よくある誤りと直し方

- 軸だけ動く／線だけ動く
  - 原因：rescale と描画更新の片方だけ
  - 対策：イベントで「新 xScale → 軸更新 → 線再描画」を統一
- 線が領域外に見える
  - 原因：clipPath 未設定
  - 対策：clipPath を作り線/点に適用
- ズーム操作が効かない
  - 原因：zoom を svg/g のどこに適用するかミス
  - 対策：ズーム対象のレイヤ（透明 rect など）を用意して適用

#### 到達目標（チェック基準）

- マウス操作でズーム＆パンできる
- x 軸がズームに追従して更新される
- 線もズームに追従し、破綻しない
- クリップで描画領域外が隠れている

### 32.4.3 【#21】連動ダッシュボード：ヒストグラム選択 → 散布図フィルタ

#### プロンプト

```
1 D3 v7で、上にヒストグラム、下に散布図を置き、  
2 ヒストグラムのブラシ範囲で散布図の点をフィルタしてください  
3 （单一HTML）。  
4  
5 要件：  
6 - 同じ元データから両方を描く  
7 - ヒストグラムで選んだvalue範囲に入る点だけ散布図で強調  
8 - UI状態（選択範囲）を1つのstateとして管理する設計にする  
9 - 初学者向けに「状態管理の考え方」を説明  
10  
11 出力：  
12 - 動く单一HTML  
13 - 改造課題（逆方向：散布図選択→ヒスト更新等）
```

#### 身につく概念（用語ミニ辞書）

Linked views 複数の可視化を連動させる設計。

State（状態）選択範囲など、UI の”現在値”。

単一情報源（Single source of truth）状態は1箇所に置き、各図が参照。

フィルタ/ハイライト 選択に応じて表示を変える2方式。

#### よくある誤りと直し方

- 片方だけ更新される
  - 原因：状態更新後に片方しか再描画していない
  - 対策：state 更新→両チャート update を呼ぶ
- state が散らばる
  - 原因：各チャート内に選択範囲を保持
  - 対策：外側に state オブジェクト、更新関数に渡す
- フィルタで点が消えすぎる
  - 原因：厳密フィルタのみで情報がなくなる
  - 対策：消すのではなく”薄くする”ハイライト方式を検討

#### 到達目標（チェック基準）

- 同一データから2つの図が生成される
- ヒストグラム選択で散布図が変化する
- state が1箇所にまとまっている
- 更新フロー（state → update）が説明されている

### 32.4.4 【#22】ツリーマップ：階層データ+ズーム

#### プロンプト

```
1 D3 v7でツリーマップを作ってください（単一HTML）。
2
3 要件：
4 - サンプル階層データ (root→category→item) をコード内に用意
5 - d3.hierarchy と d3.treemap を使う
6 - クリックでズーム（下位へ）&戻る
7 - ツールチップでパス（例：root > A > item1）と値を表示
8
9 出力：
10 - 動く単一HTML
11 - hierarchy/treemapの概念を初学者向けに説明
12 - 改造課題（色分け、パンくず表示など）
```

#### 身につく概念（用語ミニ辞書）

Hierarchy 親子関係をもつデータ構造。

d3.hierarchy 階層データをD3レイアウトに渡す形へ変換。

sum 各ノードの重み（面積）を決める集計。

Treemap 面積で比率を表現する階層可視化。

パンくず（breadcrumb）現在位置（階層パス）を示すUI。

#### よくある誤りと直し方

- 全部同じ大きさ
  - 原因：hierarchy.sum(...)をしていない
  - 対策：葉のvalueをsumで設定
- ズーム後にラベルがぐちゃぐちゃ
  - 原因：レイアウト更新とテキスト更新が不整合
  - 対策：ズーム時にrect/textを同時に更新、必要なら省略
- ツールチップがパスを出せない
  - 原因：祖先取得の方法が不明
  - 対策：node.ancestors()でパスを組み立てる

#### 到達目標（チェック基準）

- 階層データがツリーマップとして表示される
- クリックで下位へズーム、戻る操作がある
- ツールチップでパスと値が表示される

- d3.hierarchy と treemap が使われている

### 32.4.5 【#23】フォース有向グラフ：ドラッグ+隣接強調

#### プロンプト

```
1 D3 v7でフォース有向グラフ教材を作ってください（単一HTML）。
2
3 要件：
4 - サンプル nodes/links を直書き（10ノード程度）
5 - d3.forceSimulationでレイアウト
6 - ノードをドラッグできる
7 - ホバーしたノードの隣接ノード/リンクだけ強調し、他は薄く
8 - 初学者向けに「forceの各力」の役割を説明
9
10 出力：
11 - 動く単一HTML
12 - よくある落とし穴（暴れる、重なる）と対策
13 - 改造課題（重み、ラベル、固定等）
```

#### 身につく概念（用語ミニ辞書）

forceSimulation 物理シミュレーションで配置を決める。

forceLink / forceManyBody / forceCenter リンク、斥力、中心寄せ。

drag behavior ドラッグ中に位置を固定/更新する仕組み (fx/fy)。

隣接強調 ネットワーク理解の基本 (ego network 表示)。

#### よくある誤りと直し方

- ノードが暴れる
  - 原因：力のパラメータが強すぎる/弱すぎる
  - 対策：charge 強度や link 距離を調整、alphaDecay も検討
- リンクがつながらない
  - 原因：links の source/target が id と一致していない
  - 対策：forceLink().id(d=>d.id) を設定
- ドラッグ後に戻ってしまう
  - 原因：fx/fy を固定していない
  - 対策：drag 中に fx/fy 設定、終了時の扱いを設計

#### 到達目標（チェック基準）

- シミュレーションでノードが配置される
- ドラッグでノードを動かせる

- ホバーで隣接だけ強調できる
- 力の役割説明がある（最低3つ）

### 32.4.6 【#24】地図（GeoJSON）：fitSize + 簡易コロプレス

#### プロンプト

```
1 D3 v7で「GeoJSONを描画」し、値で塗り分ける簡易コロプレス
2 を作ってください（単一HTML）。
3
4 要件：
5 - サンプルGeoJSONをコード内に直書き（簡単な四角形ポリゴン2~4個でOK）
6 - d3.geoPath と d3.geoMercator（または適切な投影）を使う
7 - projection.fitSizeで自動フィット
8 - regionごとにvalueを結合して色分け（欠損はグレー）
9 - ツールチップと凡例を付ける
10
11 出力：
12 - 動く単一HTML（外部ファイルなしで動くミニGeoJSONで）
13 - 初学者向け解説（投影/geoPath/fitSize）
14 - 改造課題（実GeoJSON差し替え手順など）
```

#### 身につく概念（用語ミニ辞書）

GeoJSON 地理形状（ポリゴン等）を表すJSON形式。

projection（投影）地球→平面への写像（歪みが出る）。

geoPath 投影済みパス（SVGのd属性）を生成。

fitSize 指定サイズに収まるよう投影を自動調整。

Choropleth 値で地域を塗り分ける地図。

#### よくある誤りと直し方

- 地図が表示されない
  - 原因：projection/pathの指定漏れ
  - 対策：const path = d3.geoPath(projection)
- 画面外に飛ぶ
  - 原因：fitSize未使用 or GeoJSON座標が想定外
  - 対策：projection.fitSize([w,h], geojson)
- 値結合ができない
  - 原因：キー（regionコード）が一致していない
  - 対策：結合キーを統一、欠損時の扱いを明示

### 到達目標（チェック基準）

- ミニ GeoJSON が描画される
- fitSize でサイズに収まる
- 値で塗り分け、欠損は別色になる
- 凡例とツールチップがある

## 32.5 ★ 5：実務品質 (#25～#30)

性能最適化、アクセシビリティ、React/TypeScript 統合など実務レベルのスキルを学びます。

### 32.5.1 【#25】性能：大量散布図を Canvas 化+近傍検索でツールチップ

#### プロンプト

1 D3 v7で「大量データの散布図」を実務的に作ってください。

2

3 要件：

- 4 - 描画はCanvas（点が多い前提）
- 5 - スケール・軸はSVGでOK（ハイブリッド構成）
- 6 - データはランダム生成で1万点
- 7 - ツールチップは「マウス近傍の点」を探して表示  
（簡易で良いが高速化の工夫も説明）
- 8 - requestAnimationFrame等の描画最適化を解説

9

10

11 出力：

12 - 動く単一HTML

13 - どこが遅くなるか（ボトルネック）を初学者向けに説明

14 - 改造課題（LOD、ズーム対応など）

### 身につく概念（用語ミニ辞書）

Canvas 大量描画に強い（DOM 要素が増えない）。

SVG×Canvas ハイブリッド 軸は SVG、点は Canvas の実務定番。

近傍探索 マウス位置に近い点を探す（単純探索→高速化の入口）。

requestAnimationFrame 描画タイミングをブラウザに合わせる最適化。

devicePixelRatio 高 DPI でぼけない Canvas 設定に必要。

### よくある誤りと直し方

- Canvas がぼやける

- 原因：DPR 未対応

- 対策：canvas.width = cssW\*dpr; ctx.scale(dpr,dpr) 等

- フレームごとに残像が出る
  - 原因: clear しない
  - 対策: 每回 ctx.clearRect(0, 0, w, h) して再描画
- ツールチップが重い
  - 原因: 每mousemove で 1 万点全探索
  - 対策: 間引き (throttle)、簡易グリッド分割、d3.quadtree 等を検討

### 到達目標（チェック基準）

- 1 万点が実用的に描画できる（極端に重くない）
- 軸は SVG で表示される
- ホバーで近い点の情報が出る（簡易で OK）
- DPR 対応で点がシャープに見える
- ポトルネックの説明がある

#### 32.5.2 【#26】アクセシビリティ：SVG の説明・フォーカス・キー操作

##### プロンプト

```

1 D3 v7 の棒グラフをアクセシブルに改善してください（単一HTML）。
2
3 要件：
4 - svg に <title> と <desc> を付与（何を示すグラフか説明）
5 - 各バーに aria-label（カテゴリ名と値）
6 - Tab でバーにフォーカスできる
7 - Enter でツールチップ固定/解除
8 - 色だけに頼らない（強調は太さ/下線/枠線なども併用できる設計を説明）
9
10 出力：
11 - 動く単一HTML
12 - 初学者向けにA11yの要点（最低限ここだけ）を箇条書き
13 - 改造課題（スクリーンリーダー想定の改善など）

```

##### 身につく概念（用語ミニ辞書）

A11y（アクセシビリティ）多様な利用者が情報にアクセスできる設計。

title/desc SVG 全体の説明（読み上げにも役立つ）。

tabindex キーボードでフォーカス可能にする。

aria-label 各々の要素の意味を説明する。

色以外の符号化 色弱や印刷でも伝わる工夫。

## よくある誤りと直し方

- Tab で移動できない
  - 原因: tabindex 未設定
  - 対策: インタラクティブ要素に tabindex=0
- 読み上げが不親切
  - 原因: aria-label が短すぎる/ない
  - 対策: 「カテゴリ A、値 12」など具体化
- 色だけで区別
  - 原因: 強調が色変更のみ
  - 対策: 枠線、太さ、パターン等を併用

## 到達目標（チェック基準）

- SVG に title/desc がある
- 各バーに aria-label が付与されている
- キーボード操作 (Tab/Enter) が機能する
- 色以外でも状態が分かる工夫がある
- A11y 要点が箇条書きで説明されている

### 32.5.3 【#27】成果物：SVG ダウンロード + PNG 書き出し

#### プロンプト

1 D3 v7 の SVG チャートに「SVG ダウンロード」ボタンを付けてください  
2 (単一 HTML)。  
3  
4 追加要件:  
5 - 可能なら PNG エクスポートも提案し、  
6 ブラウザだけで実現する方法の雛形も示す  
7 - 制約 (外部フォント、外部画像、CORS など) を初学者向けに説明  
8 - 外部ライセンスなし  
9  
10 出力:  
11 - 動く単一 HTML (SVG ダウンロードは必須)  
12 - PNG は「動く最小例」か「動かすための注意点込み雛形」  
13 - 改造課題 (メタ情報、ファイル名規則など)

## 身につく概念（用語ミニ辞書）

SVG シリアライズ SVG 要素を文字列 (XML) にする。

Blob / URL.createObjectURL ブラウザ内でファイル生成して DL する仕組み。

PNG エクスポート SVG → 画像化 (Canvas 経由が一般的)。

CORS 外部画像などが混ざると画像化が失敗する要因。

### よくある誤りと直し方

- ダウンロードはできるが開くと崩れる
  - 原因: xmlns 不足、スタイルが外部依存
  - 対策: xmlns="http://www.w3.org/2000/svg"付与、スタイルを SVG 内に
- PNG が真っ白
  - 原因: SVG → Image 変換のタイミング/サイズ指定ミス
  - 対策: onload で描画、width/height を明示
- 外部フォントで見た目が変わる
  - 原因: 環境依存
  - 対策: フォントを指定、必要なら埋め込みを検討

### 到達目標（チェック基準）

- ボタンで SVG をダウンロードできる
- ダウンロードした SVG を開いても崩れにくい
- PNG について方法と制約の説明がある
- 外部依存が少ない（再現性が高い）

#### 32.5.4 【#28】React 統合：useRef + useEffect で D3 を安全に使う

##### プロンプト

```
1 React (関数コンポーネント) でD3 v7 の棒グラフを実装する例を
2 作ってください。
3
4 要件：
5 - useRefでSVGコンテナ参照
6 - useEffectで描画、data変更でupdate
7 - クリーンアップを実装 (イベント解除など)
8 - D3はDOM操作担当、Reactはデータ/状態担当、という役割分担を
9 初学者向けに説明
10 - コードは1コンポーネントで提示 (必要なら補助関数もOK)
11
12 出力：
13 - 完全なコンポーネント例
14 - ありがちなバグ (再描画で二重に積まれる等) と対策
15 - 改造課題 (凡例、ツールチップなど)
```

### 身につく概念（用語ミニ辞書）

useRef DOM 参照を保持する React フック。

useEffect 副作用（DOM 描画）を実行する場所。  
クリーンアップ イベント解除や observer 解除（メモリリーク防止）。  
責務分離 React は状態、D3 は描画（混ぜすぎない）。

### よくある誤りと直し方

- 再レンダリングで要素が増殖
  - 原因：毎回 append し続ける
  - 対策：描画ルートを 1 つに固定、join/update 設計に
- 依存配列のミスで更新されない
  - 原因：useEffect(()=>[], []) 固定
  - 対策：[data] など必要な依存を入れる
- イベントが多重登録
  - 原因：cleanup なし
  - 対策：return で解除処理を実装

### 到達目標（チェック基準）

- 1 つの React コンポーネントとして動く
- data 変更で更新される
- 要素が増殖しない
- cleanup が実装されている
- 役割分担（React/D3）が説明されている

## 32.5.5 【#29】TypeScript 化：型定義＋パース関数＋安全なアクセサ

### プロンプト

```
1 D3 v7 を TypeScript で使う前提で、データ型定義とパース関数を
2 作ってください。
3
4 想定 CSV 行：
5 - date (YYYY-MM-DD)
6 - value (数値)
7 - category (文字列)
8
9 要件：
10 - interface 定義
11 - パース関数 (Date/number 変換、失敗行の除外、理由ログ)
12 - アクセサ関数を用意し、D3 の scale に渡すまでの例を示す
13 - 初学者向けに「TS で何が嬉しいか」を説明
14
15 出力：
16 - TS コード (型定義 + パース + 使用例)
```

**17 - 改造課題 (zod等なしでどこまで守れるか、など)****身につく概念（用語ミニ辞書）**

interface データ形状の契約。

型ガード 実行時に型を確認する関数（TS の弱点補強）。

ペース関数 入力→内部表現へ（失敗を扱う設計）。

アクセサ `d => d.value` など、D3 に渡す取り出し関数。

型の恩恵 補完、事故防止、リファクタ容易性。

**よくある誤りと直し方**

- Date/number が any になる
  - 原因：ペース結果の型が曖昧
  - 対策：戻り値型を明示し、失敗ケースを除外して返す
- d3.csv の戻りが string 前提で破綻
  - 原因：生 CSV は文字列
  - 対策：row accessor で型変換し、型に落とす
- null 混入でスケールが壊れる
  - 原因：欠損をそのまま通す
  - 対策：value === null を除外、または defined で扱う

**到達目標（チェック基準）**

- interface が定義されている
- ペース関数があり、失敗行を除外できる
- D3 スケールに渡す例まで示されている
- TS の利点が初学者向けに説明されている

**32.5.6 【#30】総仕上げ：コードレビュー→リファクタ（create/update/destroy）  
→チェック項目****コードを貼る前に確認**

自分のコードを LLM に貼る際は、以下の情報を必ずマスク（ダミーに置換）してください：

- API キー・トークン（例：`sk-xxxx → YOUR_API_KEY`）
- 内部 URL・エンドポイント（例：`https://internal.example.com`）
- 社内名・プロジェクト名（例：`acme-corp → my-company`）
- 顧客データ・個人情報（実データ → サンプルデータに差し替え）

## プロンプト

```
1 あなたはD3 v7のシニアエンジニア兼レビュアです。
2 次のD3コード（私が貼る）を対象に、実務品質へ引き上げてください。
3
4 手順（必ずこの順）：
5 1) 問題点のレビュー（可読性/拡張性/更新設計/性能/A11y）を優先度順に
6 2) createChart(container, options) / update(data) / destroy() に
7     リファクタした完成コードを提示
8 3) 動作確認チェックリスト（10項目）
9 4) 改造課題（3つ）
10
11 制約：
12 - 外部ライブラリはD3のみ
13 - 単一HTML（または単一JS）として提示
14 - 初学者にも分かるように、重要ポイントにはコメントを入れる
15
16 対象コード：
17 {ここに自分のコードを貼る}
```

## 身につく概念（用語ミニ辞書）

コードレビュー観点 可読性・拡張性・性能・A11y・バグ耐性。

コンポーネント化 create/update/destroy で”使い回せる部品”にする。

責務分離 初期化と更新と破棄を分ける（不具合が減る）。

チェックリスト思考 再現性のある品質保証の入口。

## よくある誤りと直し方

- 一枚岩スクリプトで直せない
  - 原因：関数分割なし、状態が散らばる
  - 対策：create で DOM 確保、update でデータ反映、destroy で解除に分割
- 更新で壊れる
  - 原因：join/key がない、スケール再計算が漏れる
  - 対策：更新フローを明文化（スケール→軸→図形）
- イベント/observer が残る
  - 原因：destroy 未実装
  - 対策：listener 解除、ResizeObserver disconnect 等を実装

## 到達目標（チェック基準）

- レビューが優先度順で具体的（なぜ/どう直す）
- create/update/destroy の 3 関数が揃う

- 更新に耐える（データ差し替えで破綻しにくい）
- 10 項目以上のチェックリストがある
- 初学者が追えるコメント/説明が付いている

## 32.6 プロンプト活用のコツ

### 効果的な LLM 活用のポイント

1. **段階的に進める**：★ 1 から順番に取り組み、基礎を固めてから応用へ
2. **実際に動かす**：生成されたコードは必ずブラウザで実行して確認
3. **エラーを読む**：コンソールのエラーメッセージを LLM に貼って質問
4. **改造課題に挑戦**：各プロンプトの改造課題で応用力を養う
5. **コードを読む**：生成されたコードを理解してから次へ進む

### 注意事項

- LLM の出力は必ず検証してください。古い API や誤ったコードが含まれる可能性があります
- D3.js v7 を前提としています。v6 以前とは API が異なる部分があります
- 生成されたコードをそのまま本番環境で使用する前に、セキュリティとパフォーマンスを確認してください
- **機密情報を貼らない**：社内データ、個人情報、API キーなどの機密情報をプロンプトに含めないでください
- **ライセンスに注意**：既存コードを貼って質問する場合（#30 など）、そのコードのライセンスや著作権に留意してください。生成されたコードの利用条件も確認しましょう
- 「動いた」だけで採用しない：生成コードは必ずレビュー・テストを行い、理解してから使用してください

### デバッグ相談のテンプレート

エラーが出たときは、以下の情報を LLM に伝えると効率的に解決できます。

```
1 【目的】何を作っているか
2 (例：棒グラフにツールチップを追加しようとしている)
3
4 【期待】どう動くべきか
5 (例：バーにホバーすると値が表示される)
6
7 【現状】何が起きているか
8 (例：ツールチップが左上に固定されて動かない)
9
10 【エラー】コンソールのエラーメッセージ（あれば）
11 (例：Uncaught TypeError: Cannot read property 'x' of undefined)
12
13 【コード】問題を再現する最小限のコード
14 (例：関連する部分だけ抜粋して貼る)
15
16 【環境】OS / ブラウザ / D3の読み込み方法 / ローカルサーバの有無
17 (例：macOS / Chrome 120 / CDN / npx serve で起動)
18
19 【再現手順】問題が起きるまでの操作（1～3ステップ）
20 (例：1) ページを開く → 2) バーにホバー → 3) エラー発生)
```

このテンプレートを使うことで、「丸投げ」ではなく「デバッグ学習」として LLM を活用できます。環境や再現手順を書くことで、CORS・パス・ブラウザ差異・キャッシュなど「自分の環境だけで起きる問題」の切り分けにも役立ちます。



# 参考文献・参考サイト

## 公式ドキュメント・リファレンス

### D3.js 公式サイト

<https://d3js.org/>

D3.js の公式ウェブサイト。API リファレンス、チュートリアル、サンプルが掲載されています。

### D3.js GitHub リポジトリ

<https://github.com/d3/d3>

D3.js のソースコード、Issue、ディスカッションを確認できます。

### D3.js API Reference

<https://d3js.org/d3-selection>

各モジュールの詳細な API ドキュメント。selection、scale、axis、shape 等のモジュール別に整理されています。

## チュートリアル・学習リソース

### Observable - D3 Gallery

<https://observablehq.com/@d3/gallery>

Mike Bostock 氏（D3.js 作者）によるインタラクティブなサンプル集。コードを直接編集して動作を確認できます。

### Observable - Learn D3

<https://observablehq.com/collection/@d3/learn-d3>

D3.js の基礎から応用までを学べる公式チュートリアルコレクション。

### MDN Web Docs - SVG Tutorial

<https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial>

SVG の基礎を学ぶための Mozilla による包括的なチュートリアル。

### MDN Web Docs - JavaScript Guide

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>

JavaScript の基礎文法からオブジェクト指向、非同期処理まで網羅したガイド。

## 書籍

Interactive Data Visualization for the Web, 2nd Edition

Scott Murray 著, O'Reilly Media, 2017

D3.js の入門書として定評のある一冊。基礎から実践的なビジュアライゼーションまでを丁寧に解説。

D3.js in Action, 3rd Edition

Elijah Meeks, Anne-Marie Dufour 著, Manning Publications, 2024

より高度な D3.js の使い方を学べる実践的な書籍。複雑なビジュアライゼーションの構築方法を解説。

Data Visualization with Python and JavaScript, 2nd Edition

Kyran Dale 著, O'Reilly Media, 2022

Python と JavaScript (D3.js) を組み合わせたデータビジュアライゼーションの手法を解説。

Visualization Analysis and Design

Tamara Munzner 著, A K Peters/CRC Press, 2014

データビジュアライゼーションの理論と設計原則を体系的に学べる教科書。

## データソース

Natural Earth

<https://www.naturalearthdata.com/>

世界地図データ (GeoJSON、TopoJSON 形式) を無料で提供。地理データビジュアライゼーションに必須。

TopoJSON

<https://github.com/topojson/topojson>

GeoJSON を圧縮した形式。D3.js の地図描画で頻繁に使用されます。

World Bank Open Data

<https://data.worldbank.org/>

世界各国の経済・社会統計データを提供。ダッシュボードやチャートのサンプルデータとして活用できます。

## コミュニティ・フォーラム

Stack Overflow - D3.js

<https://stackoverflow.com/questions/tagged/d3.js>

D3.js に関する質問と回答が集まる Q&A サイト。実装時の問題解決に役立ちます。

D3.js Slack

<https://d3js.slack.com/>

D3.js ユーザーのコミュニティ。リアルタイムで質問や議論ができます。

Observable Forum

<https://talk.observablehq.com/>

Observable と D3.js に関するディスカッションフォーラム。

## 関連ライブラリ・ツール

Vega / Vega-Lite

<https://vega.github.io/vega/>

宣言的なビジュアライゼーション文法。D3.js をベースに構築されており、より簡潔にチャートを作成できます。

Observable Plot

<https://observablehq.com/plot/>

D3.js を基盤とした高レベルのチャートライブラリ。少ないコードで美しいチャートを作成できます。

Chart.js

<https://www.chartjs.org/>

シンプルで使いやすいチャートライブラリ。D3.js ほどの柔軟性はありませんが、基本的なチャートを素早く作成できます。

## LLM プロンプト設計の参考資料

本書第 32 章で紹介した LLM 活用学習の背景にある、プロンプト設計の公式ガイドです。URL は将来変更される可能性があります。

Anthropic Prompt Engineering Guide

<https://docs.anthropic.com/claude/docs/prompt-engineering>

Claude (Anthropic 社) のプロンプト設計ガイド。用途：指示の明確化、出力フォーマットの指定方法を学ぶ。（参照日：2026-01-06）

OpenAI Prompt Engineering Guide

<https://platform.openai.com/docs/guides/prompt-engineering>

ChatGPT/GPT-4 (OpenAI 社) のプロンプト設計ガイド。用途：プロンプト戦略の全体像とベストプラクティスを把握する。（参照日：2026-01-06）

Google AI Prompting Guide

<https://ai.google.dev/gemini-api/docs/prompting-intro>

Gemini (Google 社) のプロンプト設計入門。用途：構造化プロンプトや few-shot 学習の感覚をつかむ。（参照日：2026-01-06）