Highcharts Data Visualization

readbytes

Highcharts Data Visualization

Interactive, beautiful charts

readbytes.github.io

2025-07-20

This page is intentionally left blank.

Contents

Chapter 1.

Introduction

1. What Is Highcharts?
2. Licensing: Free for Non-Commercial, Commercial Options Explained
3. Why Choose Highcharts?
4. Setting Up Your Environment (CDN, NPM, and Modules)
5. Overview of Charting Concepts in Highcharts

1 Introduction

1.1 What Is Highcharts?

Highcharts is a powerful, flexible JavaScript library designed to create interactive and visually appealing charts for the web. It enables developers to embed a wide variety of charts into websites and applications with minimal effort, while providing rich features such as animation, tooltips, and responsiveness out of the box.

1.1.1 Highcharts in the Data Visualization Ecosystem

In the vast ecosystem of data visualization tools and libraries, Highcharts stands out as a mature, well-supported, and user-friendly option focused primarily on ease of use and rapid development. It targets web developers and data analysts who want to add professional-quality charts without spending excessive time on configuration or deep customization.

Highcharts leverages SVG (Scalable Vector Graphics) and HTML5 technologies to render charts that work seamlessly across modern browsers and devices. Its design balances simplicity for beginners with advanced capabilities for experienced developers, making it suitable for a wide range of use cases—from dashboards and reports to interactive data exploration.

1.1.2 Types of Charts Supported

Highcharts supports a comprehensive set of chart types, including:

- **Line charts** and **area charts** for trend visualization
- **Bar charts** and **column charts** for categorical comparisons
- **Pie charts** and **donut charts** for part-to-whole relationships
- **Scatter plots** and **bubble charts** for correlation and distribution analysis
- **Stock charts** with built-in indicators for financial data
- **Heatmaps** and **treemaps** for hierarchical or density data representation
- **Gauge charts** and **polar charts** for specialized visualizations

This versatility allows Highcharts to meet the needs of diverse industries, such as finance, marketing, healthcare, and more.

1.1.3 How Highcharts Compares with Other Libraries

When considering Highcharts, it helps to understand how it fits relative to popular alternatives like **Chart.js** and **D3.js**:

- **Chart.js**: Chart.js is an open-source library known for its simplicity and lightweight nature. It is excellent for basic charts and smaller projects. However, it offers fewer built-in features and less flexibility compared to Highcharts, especially for complex or interactive visualizations.
- **D3.js**: D3 (Data-Driven Documents) is a highly flexible and powerful library that provides fine-grained control over every aspect of a chart's design and behavior. It is a developer's toolkit rather than a ready-to-use charting solution. While D3 can create virtually any visualization, it requires more coding and a deeper understanding of SVG and JavaScript.

Highcharts strikes a balance between these two by providing an extensive set of chart types and interactive features with a simpler API than D3, but more capabilities and polish than Chart.js. Additionally, Highcharts includes commercial support and thorough documentation, which can be crucial for enterprise applications.

1.2 Licensing: Free for Non-Commercial, Commercial Options Explained

Understanding the licensing model of Highcharts is essential before integrating it into your projects. Highcharts offers a flexible licensing structure designed to accommodate both personal and commercial use, but it is important to know the distinctions to ensure compliance.

1.2.1 Free for Personal and Non-Commercial Use

Highcharts is **free to use** for personal projects, school assignments, and other **non-commercial** purposes. This means you can explore, learn, and build interactive charts without any cost, provided that your usage does not generate direct revenue or serve commercial interests.

This free license makes Highcharts an excellent choice for hobbyists, students, educators, and nonprofit organizations who want to create rich visualizations without worrying about licensing fees.

1.2.2 Licensing for Commercial Applications

If you intend to use Highcharts in a **commercial** setting—such as a business website, product, or service that generates revenue—you are required to purchase a commercial license. This license grants you legal permission to deploy Highcharts in commercial software or websites, and it also provides access to professional support and additional services.

Commercial licenses come with different pricing tiers depending on your needs, such as the number of developers using Highcharts, deployment scale, and additional modules or features you may require.

1.2.3 Where to Find Up-to-Date Licensing Details

Highcharts' licensing terms and pricing can evolve over time. To ensure you have the most accurate and current information, always refer to the official Highcharts website's licensing page:

https://www.highcharts.com/license

Here you will find detailed explanations of the license types, pricing information, and FAQs that address common questions about usage rights and restrictions.

1.3 Why Choose Highcharts?

Choosing the right charting library is crucial to creating effective, engaging, and maintainable data visualizations. Highcharts is a popular choice for many developers and organizations because it offers a strong combination of features that address the common challenges of web-based charting.

Here are some of the key strengths that make Highcharts stand out:

1.3.1 Ease of Use

Highcharts is designed to be beginner-friendly without sacrificing power. Its clean, well-documented API allows developers to create beautiful charts quickly with just a few lines of code. This ease of setup and use means you can focus more on your data and insights rather than struggling with complex configurations.

1.3.2 Wide Browser and Device Support

Highcharts uses modern web standards like SVG and HTML5, which ensures your charts render consistently across all major browsers—including Chrome, Firefox, Safari, Edge, and even older versions like Internet Explorer 9 and up. It also supports responsive design, so your charts adapt seamlessly to different screen sizes, from desktop monitors to mobile phones and tablets.

1.3.3 Built-In Accessibility Features

Accessibility is increasingly important in modern web applications. Highcharts includes built-in support for screen readers, keyboard navigation, and ARIA attributes, making your charts usable by people with disabilities. This helps you meet legal and ethical accessibility standards with minimal additional effort.

1.3.4 Rich Interactivity

Interactivity is a hallmark of Highcharts. It offers features such as tooltips, zooming, panning, clickable legends, drilldowns, and real-time updates. These capabilities allow users to explore data dynamically, uncover patterns, and gain insights through direct manipulation of charts, enhancing the overall user experience.

1.3.5 Extensibility and Customization

While Highcharts works well out of the box, it is also highly extensible. You can customize the look and feel extensively via themes, colors, and styles, or extend functionality with plugins and custom event handlers. Highcharts supports modular usage, so you can load only the chart types and features you need, optimizing performance.

1.3.6 Ideal Use Cases for Highcharts

Highcharts excels in scenarios where interactive, polished, and reliable charts are required quickly and with minimal fuss. Typical applications include:

- **Dashboards**: Real-time or regularly updated dashboards that require dynamic charts with filtering and zooming.
- **Reporting**: Business intelligence reports and data summaries that need professional

styling and export options.
- **Enterprise Applications**: Internal and customer-facing software where cross-browser support, accessibility, and commercial-grade reliability are critical.

1.4 Setting Up Your Environment (CDN, NPM, and Modules)

Before you start creating charts with Highcharts, you need to set up your development environment. Highcharts offers flexible ways to include the library in your projects, whether you're building a simple webpage or a complex application with modern JavaScript tooling.

This section covers three common setup methods:

- Using a CDN for quick, simple inclusion
- Installing via NPM for projects using bundlers like Webpack or Rollup
- Importing individual Highcharts modules for optimized builds

1.4.1 Including Highcharts via CDN

The easiest way to get started is to include Highcharts directly from a Content Delivery Network (CDN). This method requires no installation and works well for small projects or demos.

Simply add the following `<script>` tag inside the `<head>` or before the closing `<body>` tag of your HTML file:

```
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <div id="container" style="width: 600px; height: 400px;"></div>
  <script>
    Highcharts.chart('container', {
      title: { text: 'Sample Chart' },
      series: [{
        data: [1, 3, 2, 4]
      }]
    });
  </script>
```

This loads the full Highcharts library from the official CDN and lets you create charts immediately.

Full runnable code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Highcharts CDN Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
```

```
</head>
<body>
  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      title: { text: 'Sample Chart' },
      series: [{
        data: [1, 3, 2, 4]
      }]
    });
  </script>
</body>
</html>
```

1.4.2 Installing Highcharts via NPM

For larger projects using JavaScript bundlers such as Webpack, Rollup, or Parcel, it's best to install Highcharts as a dependency via NPM. This approach offers better control over your dependencies and enables tree shaking and module optimization.

Run this command in your project directory:

```
npm install highcharts
```

Then, in your JavaScript file, import Highcharts and initialize your chart:

```
import Highcharts from 'highcharts';

Highcharts.chart('container', {
  title: { text: 'Sample Chart' },
  series: [{
    data: [5, 7, 3, 8]
  }]
});
```

Make sure your HTML file includes a container element with the corresponding ID:

```
<div id="container" style="width: 600px; height: 400px;"></div>
```

1.4.3 Importing Individual Modules

Highcharts is modular, allowing you to import only the parts you need to keep your bundle size small and improve performance. This is especially useful if you want to use specific chart types or additional features.

For example, to use the stock charts module, you would install the main library and import the stock module like this:

```
npm install highcharts highcharts/modules/stock
```

Then in your code:

```
import Highcharts from 'highcharts';
import StockModule from 'highcharts/modules/stock';

// Initialize the module
StockModule(Highcharts);

Highcharts.stockChart('container', {
  series: [{
    data: [ /* stock data array */ ]
  }]
});
```

Similarly, you can import other modules such as exporting, accessibility, or specific chart types following the same pattern.

By choosing the setup method that best fits your project's scale and tooling, you can quickly get Highcharts up and running. The CDN method is perfect for rapid prototyping, while NPM and modular imports give you full flexibility and optimization for production applications.

1.5 Overview of Charting Concepts in Highcharts

Before diving into creating your first charts, it's important to understand the core concepts that form the foundation of Highcharts. These concepts help you organize and configure your visualizations effectively and will be referenced throughout this book.

1.5.1 The `chart` Object

At the heart of every Highcharts visualization is the `chart` object. This object represents the complete chart instance rendered on the page. When you call `Highcharts.chart()`, you create a new chart attached to a specified HTML container.

The `chart` object manages the overall layout, rendering, and behavior of your chart, and provides methods for interacting with it programmatically, such as updating data or exporting the chart.

1.5.2 Series: The Data Backbone

A **series** in Highcharts is a collection of data points that represent a particular dataset or measurement in your chart. Each series corresponds to one visual representation, such as a line, bar, or slice in a pie chart.

Charts can contain one or multiple series, allowing you to compare datasets or display related information side by side. For example, a line chart might show two series representing sales over two years.

1.5.3 Axes: Measuring and Scaling Data

Axes provide the framework for measuring and positioning your data points. Most charts use one or two axes:

- **X-axis** (horizontal): Typically represents categories or time.
- **Y-axis** (vertical): Typically represents numeric values.

Highcharts supports multiple axes, allowing you to plot different series on separate scales or dimensions.

1.5.4 Options: Declarative Configuration

Highcharts uses a **declarative configuration** approach, meaning you define your chart's appearance and behavior by passing a structured JavaScript object—called `options`—to the chart constructor.

This `options` object describes everything from chart type and colors to data series, axes, tooltips, and interactivity features.

Here is a simple example showing the structure of a basic options object:

```
Highcharts.chart('container', {
  chart: { type: 'line' },
  title: { text: 'Monthly Sales' },
  xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
  yAxis: { title: { text: 'Revenue' } },
  series: [{
    name: '2024',
    data: [150, 200, 180, 220]
  }]
});
```

This approach allows you to clearly see and control every aspect of your chart without writing complex imperative code.

Full runnable code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Line Chart Example</title>
  <!-- Load Highcharts from CDN -->
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <div id="container" style="width: 600px; height: 400px; margin: 0 auto"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'line' },
      title: { text: 'Monthly Sales' },
      xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
      yAxis: { title: { text: 'Revenue' } },
      series: [{
        name: '2024',
        data: [150, 200, 180, 220]
      }]
    });
  </script>
</body>
</html>
```

1.5.5 What to Expect Next

In the following chapters, you will learn how to use these concepts to build a wide variety of charts, customize their look and feel, add interactivity, and integrate Highcharts into real-world applications. With a solid understanding of the chart object, series, axes, and options, you're well-prepared to start your journey into Highcharts data visualization.

Chapter 2.

Getting Started with Highcharts

1. Installing and Including Highcharts
2. Your First Chart: A Simple Line Chart
3. Understanding the Chart Object and Configuration
4. Basic Chart Anatomy: Container, Series, Axes

2 Getting Started with Highcharts

2.1 Installing and Including Highcharts

Before you can start creating charts with Highcharts, you need to add the library to your project. There are two common ways to include Highcharts depending on the type and scale of your project:

- Including Highcharts via a `<script>` tag for quick and simple use
- Installing Highcharts using package managers like NPM or Yarn for modern development workflows

2.1.1 Including Highcharts Using a `script` Tag

If you are working on a small project, prototype, or a simple HTML page without build tools, the easiest way to get started is by including Highcharts directly from a Content Delivery Network (CDN).

2.1.2 How to do it:

1. Add the following `<script>` tag inside the `<head>` or just before the closing `</body>` tag of your HTML file:

```
<script src="https://code.highcharts.com/highcharts.js"></script>
```

2. Create a container element where the chart will be rendered:

```
<div id="chart-container" style="width: 600px; height: 400px;"></div>
```

3. Initialize a chart using inline JavaScript:

```
<script>
  Highcharts.chart('chart-container', {
    title: { text: 'My First Chart' },
    series: [{
      data: [1, 3, 2, 4]
    }]
  });
</script>
```

This method requires no installation or build process and is ideal for quick demos, learning, or simple static sites.

2.1.3 Installing Highcharts via NPM or Yarn

For larger projects, especially those using modern JavaScript frameworks (React, Angular, Vue) or bundlers like Webpack, Rollup, or Parcel, installing Highcharts as a dependency using NPM or Yarn is the recommended approach. This method allows better dependency management, easier upgrades, and supports tree shaking for optimized bundles.

2.1.4 How to install:

Using **NPM**:

```
npm install highcharts
```

Or using **Yarn**:

```
yarn add highcharts
```

2.1.5 How to include in your JavaScript code:

After installation, import Highcharts in your JavaScript or TypeScript files:

```
import Highcharts from 'highcharts';

Highcharts.chart('chart-container', {
  title: { text: 'My First Chart' },
  series: [{
    data: [5, 6, 3, 8]
  }]
});
```

2.1.6 HTML container remains the same:

```
<div id="chart-container" style="width: 600px; height: 400px;"></div>
```

2.1.7 When to Use Each Method

Use Case	Recommended Method
Quick demos, prototypes, or learning	`<script>` tag via CDN
Simple static sites without build tooling	`<script>` tag via CDN

Use Case	Recommended Method
Modern apps with bundlers or frameworks	NPM/Yarn package installation
Projects needing modular imports or optimization	NPM/Yarn with modular imports

By choosing the right installation method for your project, you ensure a smooth development experience with Highcharts. The next sections will build on this setup to help you create your first interactive chart.

2.2 Your First Chart: A Simple Line Chart

Now that you have Highcharts included in your project, let's create your very first chart—a simple line chart. This example will walk you through the complete HTML and JavaScript needed to render the chart, while explaining the key parts of the configuration object.

2.2.1 Complete Example: Basic Line Chart

```
<!-- Container where the chart will be rendered -->
<div id="line-chart-container" style="width: 600px; height: 400px;"></div>

<script>
  // Initialize the chart
  Highcharts.chart('line-chart-container', {
    chart: {
      type: 'line'  // Specifies a line chart
    },
    title: {
      text: 'Monthly Sales Data'  // Chart title
    },
    xAxis: {
      categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']  // X-axis labels
    },
    yAxis: {
      title: {
        text: 'Sales (in units)'  // Y-axis label
      }
    },
    series: [{
      name: '2024',               // Series name shown in the legend
      data: [29, 71, 45, 50, 65, 80]  // Data points for each month
    }]
  });
</script>
```

Full runnable code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple Line Chart with Highcharts</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <!-- Container where the chart will be rendered -->
  <div id="line-chart-container" style="width: 600px; height: 400px;"></div>

  <script>
    // Initialize the chart
    Highcharts.chart('line-chart-container', {
      chart: {
        type: 'line'  // Specifies a line chart
      },
      title: {
        text: 'Monthly Sales Data'  // Chart title
      },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']  // X-axis labels
      },
      yAxis: {
        title: {
          text: 'Sales (in units)'  // Y-axis label
        }
      },
      series: [{
        name: '2024',               // Series name shown in the legend
        data: [29, 71, 45, 50, 65, 80]  // Data points for each month
      }]
    });
  </script>
</body>
</html>
```

2.2.2 Breaking Down the Configuration Object

When calling `Highcharts.chart()`, you pass two arguments:

- The **container ID** where the chart will render (`'line-chart-container'`)
- The **configuration object** that defines the chart's appearance and behavior

2.2.3 Key Sections in the Configuration Object

Section	Purpose
`chart`	Defines chart-level properties, including chart type (`'line'` here)

Section	Purpose
`title`	Sets the main title of the chart
`xAxis`	Configures the horizontal axis, including categories or values
`yAxis`	Configures the vertical axis and its label
`series`	An array of one or more data series; each series represents a dataset to be visualized

2.2.4 What Happens When You Run This Code?

- Highcharts selects the `<div>` with ID `line-chart-container`.
- It renders a line chart with the title "Monthly Sales Data."
- The X-axis displays the months January through June.
- The Y-axis is labeled to indicate sales units.
- A single data series named "2024" plots the corresponding sales figures for each month.
- A legend is automatically added for the series name.

This simple example is the foundation for all charts you'll build with Highcharts. In upcoming chapters, you'll learn how to customize these elements, add interactivity, and create more complex visualizations.

2.3 Understanding the Chart Object and Configuration

At the core of every Highcharts chart is the **configuration object**—a structured JavaScript object that defines everything about how your chart looks and behaves. When you call `Highcharts.chart()`, this object tells Highcharts how to render your visualization.

Let's break down the most common parts of the configuration object, using a simple example for context:

```
Highcharts.chart('container', {
  chart: { type: 'line' },
  title: { text: 'Monthly Sales' },
  xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
  yAxis: {
    title: { text: 'Units Sold' }
  },
  series: [{
    name: '2024',
    data: [29, 71, 45, 50]
  }]
});
```

2.3.1 Key Configuration Sections

`chart`

The `chart` property controls general chart settings.

- **`type`**: Defines the chart type (e.g., `'line'`, `'bar'`, `'pie'`).
- Other options include dimensions (`width`, `height`), background, animation, and more.

Effect: Changes how the data will be visualized — for example, switching from a line to a bar chart.

`title`

The `title` property sets the main heading of the chart.

- **`text`**: The string displayed as the chart's title.

Effect: Provides context to users by clearly labeling what the chart represents.

`xAxis`

The `xAxis` configures the horizontal axis.

- **`categories`**: An array of labels for each data point along the X-axis (like months, days, or categories).
- You can also configure axis titles, grid lines, tick intervals, and more.

Effect: Determines how data points are spaced and labeled horizontally.

`yAxis`

The `yAxis` configures the vertical axis.

- **`title`**: Sets the label for the Y-axis.
- Additional properties control min/max values, grid lines, and formatting.

Effect: Defines the scale and units for measuring your data vertically.

`series`

The `series` array holds the actual data to plot.

- Each object in the array represents one dataset or "series."
- Common properties:
  - **`name`**: Label used in the legend and tooltips.
  - **`data`**: An array of numeric values or objects representing data points.
  - **`type`**: Optional — can override the global chart type per series.

Effect: Provides the core data that drives the chart's visualization.

2.3.2 How These Pieces Work Together

When Highcharts renders your chart:

- It uses the `chart.type` to determine the overall visual style.
- It places the `title.text` prominently to explain the chart's subject.
- It lays out the `xAxis.categories` horizontally, aligning each data point.
- It scales the vertical `yAxis` according to your data and labels it.
- It plots each `series` dataset according to the chosen chart type and configuration.

This declarative setup lets you control every aspect of your chart by simply modifying the configuration object.

2.3.3 Summary

- **`chart`** defines chart type and layout.
- **`title`** labels your chart.
- **`xAxis`** and **`yAxis`** set the chart's coordinate system and labels.
- **`series`** holds your data to be visualized.

Understanding these core properties is essential to customizing and building complex charts with Highcharts. As you progress, you'll explore many more options and configurations that expand these basics.

2.4 Basic Chart Anatomy: Container, Series, Axes

To build effective charts with Highcharts, it's important to understand how three fundamental parts work together:

- The **container** where the chart is rendered
- The **series** data that drives what is displayed
- The **axes** that provide structure and scale

These elements form the basic anatomy of every Highcharts visualization.

2.4.1 The Container: Where Your Chart Lives

Highcharts renders charts inside an HTML container element, usually a `<div>`. This container acts as a placeholder in your webpage.

Example container:

```
<div id="chart-container" style="width: 600px; height: 400px;"></div>
```

- The **id** attribute (**chart-container**) links this element to your JavaScript configuration.
- CSS styles like **width** and **height** control the size of your chart.

Highcharts takes this container and draws the entire chart inside it using SVG, HTML, or Canvas, depending on the chart type.

2.4.2 Series: Your Data Visualized

The **series** section of the configuration holds the actual data you want to display.

Example series configuration:

```
series: [{
  name: 'Sales 2024',
  data: [30, 40, 35, 50, 49, 60]
}]
```

- Each series is one dataset plotted on the chart.
- The **name** appears in the legend and tooltips.
- The **data** array contains numeric values corresponding to the X-axis categories.

You can include multiple series to compare datasets side by side.

2.4.3 Axes: The Framework of the Chart

Axes give context and scale to your data.

2.4.4 X-Axis (Horizontal)

Defines the categories or points along the horizontal line.

Example:

```
xAxis: {
  categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
}
```

- Labels correspond to each data point in the series.
- You can customize axis titles, grid lines, and more.

2.4.5 Y-Axis (Vertical)

Defines the scale for measuring your data values.

Example:

```
yAxis: {
  title: { text: 'Units Sold' }
}
```

- Sets the label for the vertical axis.
- Automatically adjusts the range based on data, or you can set fixed limits.

2.4.6 Bringing It All Together

Here's how these parts connect visually and in code:

```
<div id="chart-container" style="width: 600px; height: 400px;"></div>

<script>
Highcharts.chart('chart-container', {
  chart: { type: 'line' },

  title: { text: 'Monthly Sales' },

  xAxis: {
    categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']  // <-- X-Axis labels
  },

  yAxis: {
    title: { text: 'Units Sold' }  // <-- Y-Axis label
  },

  series: [{
    name: 'Sales 2024',              // <-- Series name shown in legend
    data: [30, 40, 35, 50, 49, 60]  // <-- Series data points
  }]
});
</script>
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Example: Monthly Sales</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Bringing It All Together</h2>

  <div id="chart-container" style="width: 600px; height: 400px;"></div>
```

```
  <script>
    Highcharts.chart('chart-container', {
      chart: { type: 'line' },

      title: { text: 'Monthly Sales' },

      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']  // <-- X-Axis labels
      },

      yAxis: {
        title: { text: 'Units Sold' }  // <-- Y-Axis label
      },

      series: [{
        name: 'Sales 2024',              // <-- Series name shown in legend
        data: [30, 40, 35, 50, 49, 60]  // <-- Series data points
      }]
    });
  </script>
</body>
</html>
```

By mastering the relationship between container, series, and axes, you gain the foundation needed to create meaningful and well-structured Highcharts visualizations. In the next chapters, you'll learn how to customize each part further and add advanced features.

Chapter 3.

Core Chart Types

1. Line and Spline Charts
2. Column and Bar Charts
3. Pie and Donut Charts
4. Area and Area Spline Charts
5. Specialized Chart Types (Gauge, Heatmap, TreeMap)

3 Core Chart Types

3.1 Line and Spline Charts

Line and spline charts are among the most common ways to visualize data trends over time or ordered categories. Both chart types connect data points with lines, but they differ in how those lines are drawn.

3.1.1 Difference Between Line and Spline Charts

- **Line Chart**: Connects data points with straight line segments. It emphasizes the exact values and changes between points with crisp, angular transitions.
- **Spline Chart**: Connects data points with smooth, curved lines (also called splines). This results in a softer, more flowing appearance that can make trends easier to follow visually, especially when data is continuous.

Both are ideal for showing how values evolve over time, such as stock prices, temperature changes, or monthly sales.

3.1.2 Use Cases

Chart Type	Best For
Line	Precise values and exact changes between points; ideal for discrete data with clear jumps or drops
Spline	Smooth trends and gradual changes; better for continuous data or when you want a visually appealing curve

3.1.3 Basic Example: Line vs. Spline Chart

Here is a simple example showing how to create a line chart and switch it to a spline chart by changing the `type` property.

```
<div id="chart-container" style="width: 600px; height: 400px;"></div>

<script>
  // Change 'line' to 'spline' to switch chart type
  Highcharts.chart('chart-container', {
    chart: {
```

```
        type: 'line'  // Change to 'spline' for smooth curves
      },
      title: {
        text: 'Monthly Average Temperature'
      },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
      },
      yAxis: {
        title: { text: 'Temperature (°C)' }
      },
      series: [{
        name: 'Tokyo',
        data: [7.0, 6.9, 9.5, 14.5, 18.4, 21.5]
      }]
    });
  </script>
```

Full runnable code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Line and Spline Charts</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <div id="chart-container" style="width: 600px; height: 400px;"></div>

  <script>
    // Change 'line' to 'spline' to switch chart type
    Highcharts.chart('chart-container', {
      chart: {
        type: 'line'  // Change to 'spline' for smooth curves
      },
      title: {
        text: 'Monthly Average Temperature'
      },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
      },
      yAxis: {
        title: { text: 'Temperature (°C)' }
      },
      series: [{
        name: 'Tokyo',
        data: [7.0, 6.9, 9.5, 14.5, 18.4, 21.5]
      }]
    });
  </script>
</body>
</html>
```

3.1.4 How to Switch Between Line and Spline

In the configuration's `chart` section:

```
chart: {
  type: 'line'   // or 'spline'
}
```

Simply update the `type` property to `'spline'` for smooth curves, or `'line'` for straight segments.

Both line and spline charts provide clear and effective visualizations of data trends. Choosing between them depends on the nature of your data and the style you want to present.

3.2 Column and Bar Charts

Column and bar charts are powerful tools for comparing categories and groups. Although they display similar information, their orientation distinguishes them:

- **Column charts** display data with vertical bars.
- **Bar charts** display data with horizontal bars.

Both are ideal for categorical comparisons, but the choice between them often depends on label length, space, and readability.

3.2.1 Vertical Columns (`column`)

- Bars extend **vertically** from the X-axis.
- Categories are typically shown along the **X-axis** at the bottom.
- Suitable when category labels are short and fit comfortably beneath each column.
- Commonly used for time-based data or ranked categories.

3.2.2 Example: Vertical Column Chart

```
Highcharts.chart('container', {
  chart: { type: 'column' },
  title: { text: 'Fruit Consumption' },
  xAxis: {
    categories: ['Apples', 'Bananas', 'Oranges'],
    title: { text: 'Fruit Type' }
  },
  yAxis: {
    title: { text: 'Quantity' }
```

```
  },
  series: [{
    name: 'John',
    data: [5, 3, 4]
  }, {
    name: 'Jane',
    data: [2, 2, 3]
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Column Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Example: Vertical Column Chart</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Fruit Consumption' },
      xAxis: {
        categories: ['Apples', 'Bananas', 'Oranges'],
        title: { text: 'Fruit Type' }
      },
      yAxis: {
        title: { text: 'Quantity' }
      },
      series: [{
        name: 'John',
        data: [5, 3, 4]
      }, {
        name: 'Jane',
        data: [2, 2, 3]
      }]
    });
  </script>
</body>
</html>
```

3.2.3 Horizontal Bars (`bar`)

- Bars extend **horizontally** from the Y-axis.
- Categories are shown along the **Y-axis** on the left.
- Ideal when category labels are long or need more space to be readable.

- Works well in dashboards or reports where vertical space is limited.

3.2.4 Example: Horizontal Bar Chart

```
Highcharts.chart('container', {
  chart: { type: 'bar' },
  title: { text: 'Fruit Consumption' },
  xAxis: {
    title: { text: 'Quantity' }
  },
  yAxis: {
    categories: ['Apples', 'Bananas', 'Oranges'],
    title: { text: 'Fruit Type' }
  },
  series: [{
    name: 'John',
    data: [5, 3, 4]
  }, {
    name: 'Jane',
    data: [2, 2, 3]
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Bar Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Example: Horizontal Bar Chart</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'bar' },
      title: { text: 'Fruit Consumption' },
      xAxis: {
        title: { text: 'Quantity' }
      },
      yAxis: {
        categories: ['Apples', 'Bananas', 'Oranges'],
        title: { text: 'Fruit Type' }
      },
      series: [{
        name: 'John',
        data: [5, 3, 4]
      }, {
        name: 'Jane',
```

```
        data: [2, 2, 3]
      }]
    });
  </script>
</body>
</html>
```

3.2.5 Label Styling

- **Rotate X-axis labels** in column charts if category names are long:

```
xAxis: {
  labels: {
    rotation: -45,
    style: { fontSize: '12px', fontFamily: 'Arial, sans-serif' }
  }
}
```

- Use readable fonts and sufficient font size to improve clarity.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Column Chart with Rotated Labels</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Column Chart with Rotated X-Axis Labels</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Website Traffic by Source' },
      xAxis: {
        categories: [
          'Direct Visitors',
          'Referral Traffic',
          'Social Media Links',
          'Email Campaigns',
          'Paid Advertisements'
        ],
        labels: {
          rotation: -45,
          style: {
            fontSize: '12px',
            fontFamily: 'Arial, sans-serif'
          }
        },
```

```
        title: { text: 'Traffic Source' }
      },
      yAxis: {
        title: { text: 'Visits' }
      },
      series: [{
        name: '2024',
        data: [120, 90, 150, 70, 100]
      }]
    });
  </script>
</body>
</html>
```

3.2.6 Adjust Bar/Column Width

- Control the width using `pointWidth` for fixed size bars:

```
plotOptions: {
  column: {
    pointWidth: 30
  },
  bar: {
    pointWidth: 20
  }
}
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Bar Chart with Fixed Width Bars</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Bar Chart (Fixed Width)</h2>

  <div id="bar-container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('bar-container', {
      chart: { type: 'bar' },
      title: { text: 'Support Tickets by Type' },
      yAxis: {
        categories: ['Bug', 'Feature Request', 'Question'],
        title: { text: 'Ticket Type' }
      },
      xAxis: {
        title: { text: 'Tickets' }
      },
      plotOptions: {
```

```
        bar: {
          pointWidth: 20 // Narrower bars
        }
      },
      series: [{
        name: 'Count',
        data: [45, 30, 60]
      }]
    });
  </script>
</body>
</html>
```

- Or use `pointPadding` and `groupPadding` to adjust spacing between bars:

```
plotOptions: {
  column: {
    pointPadding: 0.1,
    groupPadding: 0.2
  }
}
```

3.2.7 When to Use Which?

Chart Type	When to Use
Column	Short category labels, time series, vertical space ample
Bar	Long category names, horizontal layout preferred, limited vertical space

By understanding the difference between column and bar charts and applying simple styling tweaks, you can create clear, attractive comparisons tailored to your data and layout needs.

3.3 Pie and Donut Charts

Pie charts are widely used to show proportions or percentages of a whole, making it easy to visualize how different parts contribute to a total. Donut charts are a variation of pie charts with a hollow center, which can improve readability and provide space for additional labels or content.

3.3.1 Configuring a Pie Chart

A pie chart in Highcharts is created by setting the chart type to `'pie'` and supplying data as an array of objects or values representing each slice.

3.3.2 Basic Pie Chart Example

```
<div id="pie-container" style="width: 600px; height: 400px;"></div>

<script src="https://code.highcharts.com/highcharts.js"></script>
<script>
Highcharts.chart('pie-container', {
  chart: {
    type: 'pie'
  },
  title: {
    text: 'Browser Market Shares'
  },
  series: [{
    name: 'Share',
    data: [
      { name: 'Chrome', y: 61.41 },
      { name: 'Firefox', y: 11.84 },
      { name: 'Edge', y: 10.85 },
      { name: 'Safari', y: 4.67 },
      { name: 'Others', y: 11.23 }
    ]
  }]
});
</script>
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Basic Pie Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Basic Pie Chart Example</h2>

  <div id="pie-container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('pie-container', {
      chart: {
        type: 'pie'
      },
      title: {
        text: 'Browser Market Shares'
      },
```

```
      tooltip: {
        pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'
      },
      accessibility: {
        point: {
          valueSuffix: '%'
        }
      },
      plotOptions: {
        pie: {
          allowPointSelect: true,
          cursor: 'pointer',
          dataLabels: {
            enabled: true,
            format: '<b>{point.name}</b>: {point.percentage:.1f} %'
          }
        }
      },
      series: [{
        name: 'Share',
        colorByPoint: true,
        data: [
          { name: 'Chrome', y: 61.41 },
          { name: 'Firefox', y: 11.84 },
          { name: 'Edge', y: 10.85 },
          { name: 'Safari', y: 4.67 },
          { name: 'Others', y: 11.23 }
        ]
      }]
    });
  </script>
</body>
</html>
```

3.3.3 Modifying a Pie Chart to a Donut Chart

To convert a pie chart into a donut chart, use the `innerSize` option inside the series configuration. This defines the radius of the hollow center.

3.3.4 Donut Chart Example

```
series: [{
  name: 'Share',
  innerSize: '50%',  // Creates the donut hole
  data: [
    { name: 'Chrome', y: 61.41 },
    { name: 'Firefox', y: 11.84 },
    { name: 'Edge', y: 10.85 },
    { name: 'Safari', y: 4.67 },
```

```
    { name: 'Others', y: 11.23 }
  ]
}]
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Donut Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Donut Chart Example</h2>

  <div id="donut-container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('donut-container', {
      chart: {
        type: 'pie'
      },
      title: {
        text: 'Browser Market Shares'
      },
      tooltip: {
        pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'
      },
      accessibility: {
        point: {
          valueSuffix: '%'
        }
      },
      plotOptions: {
        pie: {
          innerSize: '50%', // Creates the donut hole
          allowPointSelect: true,
          cursor: 'pointer',
          dataLabels: {
            enabled: true,
            format: '<b>{point.name}</b>: {point.percentage:.1f} %'
          }
        }
      },
      series: [{
        name: 'Share',
        colorByPoint: true,
        data: [
          { name: 'Chrome', y: 61.41 },
          { name: 'Firefox', y: 11.84 },
          { name: 'Edge', y: 10.85 },
          { name: 'Safari', y: 4.67 },
          { name: 'Others', y: 11.23 }
        ]
      }]
    });
```

```
  </script>
</body>
</html>
```

3.3.5 Labeling Slices and Adding Legends

- **Data Labels:** By default, slices show labels with the category name and percentage. You can customize them using `dataLabels`:

```
plotOptions: {
  pie: {
    dataLabels: {
      enabled: true,
      format: '{point.name}: {point.percentage:.1f} %'
    }
  }
}
```

- **Legends:** To display a legend alongside the chart, set `legend.enabled` to `true` (enabled by default):

```
legend: {
  enabled: true,
  layout: 'vertical',
  align: 'right',
  verticalAlign: 'middle'
}
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Donut Chart with Labels and Legend</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Donut Chart with Custom Labels and Legend</h2>

  <div id="donut-container" style="width: 700px; height: 400px;"></div>

  <script>
    Highcharts.chart('donut-container', {
      chart: {
        type: 'pie'
      },
      title: {
        text: 'Browser Market Shares'
      },
      legend: {
        enabled: true,
        layout: 'vertical',
```

```
        align: 'right',
        verticalAlign: 'middle'
      },
      tooltip: {
        pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'
      },
      plotOptions: {
        pie: {
          innerSize: '50%',
          allowPointSelect: true,
          cursor: 'pointer',
          dataLabels: {
            enabled: true,
            format: '{point.name}: {point.percentage:.1f} %'
          }
        }
      },
      series: [{
        name: 'Share',
        colorByPoint: true,
        data: [
          { name: 'Chrome', y: 61.41 },
          { name: 'Firefox', y: 11.84 },
          { name: 'Edge', y: 10.85 },
          { name: 'Safari', y: 4.67 },
          { name: 'Others', y: 11.23 }
        ]
      }]
    });
  </script>
</body>
</html>
```

3.3.6 Adding Interactivity: Responding to Click Events

You can make your pie or donut chart interactive by listening to click events on individual slices.

3.3.7 Example: Alert slice name and value on click

```
plotOptions: {
  pie: {
    cursor: 'pointer',
    events: {
      click: function (event) {
        alert('You clicked: ' + event.point.name + ' with value ' + event.point.y);
      }
    }
  }
```

```
}
```

3.3.8 Summary

- Use `type: 'pie'` to create pie charts.
- Add `innerSize` in the series to convert to a donut chart.
- Customize slice labels with `dataLabels`.
- Enable and configure legends for better readability.
- Add event handlers on slices to make charts interactive and responsive to user actions.

Pie and donut charts provide a visually intuitive way to communicate parts of a whole, and Highcharts makes it easy to customize and enhance them for your needs.

3.4 Area and Area Spline Charts

Area charts are a powerful way to visualize cumulative totals and show how values evolve over time. By filling the area beneath the line, these charts emphasize the volume or magnitude of data, making trends and comparisons more visually impactful.

3.4.1 Area vs. Area Spline Charts

- **Area Chart (`area`)**: Connects data points with straight line segments, then fills the area beneath the line. It clearly shows discrete changes in values.
- **Area Spline Chart (`areaspline`)**: Connects data points with smooth, curved lines, producing a flowing, rounded shape. This is useful for continuous data or when you want a more visually appealing smooth trend.

Both types are useful for emphasizing accumulated quantities, such as total sales, revenue, or resource usage over time.

3.4.2 Unstacked Area Chart

```
Highcharts.chart('container', {
  chart: { type: 'area' },
  title: { text: 'Unstacked Area Chart' },
  xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
```

```
  yAxis: { title: { text: 'Units Sold' } },
  series: [{
    name: 'Product A',
    data: [3, 4, 3, 5]
  }, {
    name: 'Product B',
    data: [2, 3, 5, 4]
  }]
});
```

This example shows multiple series as overlapping filled areas, each representing individual values without stacking.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Unstacked Area Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Basic Examples</h2>
  <h3>Unstacked Area Chart</h3>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'area' },
      title: { text: 'Unstacked Area Chart' },
      xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
      yAxis: { title: { text: 'Units Sold' } },
      plotOptions: {
        area: {
          stacking: null // explicitly unstacked
        }
      },
      series: [{
        name: 'Product A',
        data: [3, 4, 3, 5]
      }, {
        name: 'Product B',
        data: [2, 3, 5, 4]
      }]
    });
  </script>
</body>
</html>
```

3.4.3 Stacked Area Chart

Stacked area charts build on the area chart by stacking values on top of each other, emphasizing the total cumulative value at each point.

```
Highcharts.chart('container', {
  chart: { type: 'area' },
  title: { text: 'Stacked Area Chart' },
  xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
  yAxis: { title: { text: 'Units Sold' } },
  plotOptions: {
    area: {
      stacking: 'normal'  // Enable stacking
    }
  },
  series: [{
    name: 'Product A',
    data: [3, 4, 3, 5]
  }, {
    name: 'Product B',
    data: [2, 3, 5, 4]
  }]
});
```

Stacking highlights how each category contributes to the total across time or categories.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Stacked Area Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Stacked Area Chart</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'area' },
      title: { text: 'Stacked Area Chart' },
      xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
      yAxis: { title: { text: 'Units Sold' } },
      plotOptions: {
        area: {
          stacking: 'normal'  // Enable stacking
        }
      },
      series: [{
        name: 'Product A',
        data: [3, 4, 3, 5]
      }, {
        name: 'Product B',
        data: [2, 3, 5, 4]
```

```
      }]
    });
  </script>
</body>
</html>
```

3.4.4 Switching to Area Spline

To use smooth curves instead of straight lines, change the `chart.type` to `'areaspline'`. The rest of the configuration remains the same.

```
chart: { type: 'areaspline' }
```

This will produce a visually softer, rounded chart while keeping the filled area under the curve.

3.4.5 When to Use Area and Area Spline Charts

- Use **area charts** when you want a clear view of discrete changes and exact values.
- Use **area spline charts** when you want to emphasize smooth trends and continuity.
- Use **stacked variants** to emphasize total amounts and how components contribute over time.

Area and area spline charts are ideal for illustrating cumulative data and can make your visualizations more expressive and insightful. As you work through examples, try switching between these types and stacking options to see how they change your data story.

3.5 Specialized Chart Types (Gauge, Heatmap, TreeMap)

Highcharts offers several specialized chart types designed to visualize specific kinds of data beyond traditional charts. This section introduces three popular types: **gauge charts**, **heatmaps**, and **treemaps**—each suited to different data visualization challenges.

3.5.1 Gauge Charts: Radial Visualizations for Metrics and Thresholds

Gauge charts display a single value on a circular dial, similar to a speedometer or fuel gauge.

- The **radial layout** uses a needle or pointer to indicate the current value on a scale.

- Useful for showing performance metrics, KPIs, or sensor readings.
- You can set **thresholds or zones** (color-coded ranges) to indicate safe, warning, or critical levels.

Example use cases:

- CPU usage monitoring
- Speed measurement
- Completion percentage

The gauge's dial visually conveys if a metric is within acceptable limits or requires attention.

3.5.2 Heatmaps: Mapping Values to Colors on a Grid

Heatmaps represent data values with colors, arranged on a grid defined by two categorical axes.

- Each cell's color intensity or hue corresponds to the data value at that point.
- Useful for visualizing correlations, activity levels, or density.

Highcharts automatically maps data values to a color gradient or discrete color stops.

Example use cases:

- Website click activity by page and time
- Correlation matrices
- Resource utilization across departments and days

Heatmaps make it easy to spot hotspots, clusters, and anomalies through color variations.

3.5.3 Treemaps: Visualizing Hierarchical Data as Nested Rectangles

Treemaps display hierarchical data using nested rectangles, where:

- The **size** of each rectangle represents a quantitative value.
- The **color** can represent another dimension, like category or performance.
- The hierarchy is **flattened** visually by nesting rectangles within parent rectangles.

Example use cases:

- Disk usage breakdown by folder and file size
- Market share by company divisions and products
- Budget allocation by departments and projects

Treemaps give a compact overview of complex hierarchies and help identify large contributors or outliers.

3.5.4 Summary

Chart Type	Key Feature	Typical Use Case
Gauge	Radial dial with thresholds	Metrics monitoring, performance dashboards
Heatmap	Color-coded grid cells	Correlations, activity levels, density maps
Treemap	Nested rectangles showing hierarchy	Hierarchical data breakdown and comparison

These specialized charts enhance Highcharts' versatility, empowering you to visualize a broad range of data types in ways that suit your analysis needs.

Chapter 4.

Chart Configuration Deep Dive

1. Series Data and Data Formats
2. Axes Configuration and Customization
3. Titles, Subtitles, and Legends
4. Tooltips and Data Labels
5. Colors, Gradients, and Themes

4 Chart Configuration Deep Dive

4.1 Series Data and Data Formats

A core part of creating charts with Highcharts is supplying the **series data**—the numerical or categorical information you want to visualize. Highcharts is flexible in how it accepts data, allowing you to use different formats depending on your data type and chart requirements.

4.1.1 Common Data Formats Accepted by Highcharts

Array of Values

The simplest format is an array of numeric values. Highcharts assigns these values automatically along the X-axis, starting at zero or using defined categories.

```
series: [{
  name: 'Sales',
  data: [29, 71, 45, 50, 65, 80]
}]
```

- Suitable for charts with implicit X-axis values, like basic line or column charts.
- X-axis points are automatically numbered (0, 1, 2, …) or labeled using categories.

Array of `[x, y]` Pairs

For more control, provide explicit X and Y values as pairs:

```
series: [{
  name: 'Revenue',
  data: [
    [0, 29],
    [1, 71],
    [2, 45],
    [3, 50]
  ]
}]
```

- This format works with numeric or datetime X-axes.
- Allows plotting data points at specific X values rather than sequential indexes.

You can also use JavaScript `Date.UTC()` for time series:

```
series: [{
  data: [
    [Date.UTC(2024, 0, 1), 29],
    [Date.UTC(2024, 1, 1), 71],
    [Date.UTC(2024, 2, 1), 45]
  ]
}]
```

Array of Objects

This format is the most flexible and supports additional properties per point, like names, colors, or custom data.

```
series: [{
  data: [
    { x: 0, y: 29, name: 'Jan' },
    { x: 1, y: 71, name: 'Feb' },
    { x: 2, y: 45, name: 'Mar' }
  ]
}]
```

- Use this when you want to add metadata or override styling for individual points.
- Especially useful for pie charts, scatter plots, or bubble charts.

Using Categories and Date Strings on Axes

- **Categories**: Define explicit labels for the X-axis, turning numeric indexes into meaningful categories.

```
xAxis: {
  categories: ['Jan', 'Feb', 'Mar', 'Apr']
}
```

- When categories are set, a simple array of Y-values corresponds to those labels.
- **Date Strings**: Highcharts supports dates as milliseconds since epoch (`Date.UTC()`), enabling precise time-based plotting.

```
xAxis: {
  type: 'datetime'
}
```

- You can pass dates as timestamps or use JavaScript `Date` objects converted to milliseconds.

4.1.2 How Different Chart Types Interpret Series Data

- **Line, Area, Column, Bar**: Accept simple arrays or `[x, y]` pairs; categories or datetime axes often used.
- **Pie and Donut**: Require an array of objects with `name` and `y` properties to label slices.
- **Scatter and Bubble**: Use arrays of objects with `x`, `y`, and optionally `z` (for bubble size).
- **Treemaps and Heatmaps**: Expect hierarchical or matrix-like structured data using objects with keys specifying position, size, or color.

4.1.3 Summary

Highcharts' flexible data formats let you adapt to your dataset and chart type easily:

Format	Usage	Example
Array of values	Simple sequential numeric data	`[29, 71, 45, 50]`
Array of `[x, y]`	Numeric or datetime X-axis	`[[0, 29], [1, 71]]`
Array of objects	Metadata per point (labels, color)	`[{x:0, y:29, name:'Jan'}]`

Understanding how to structure your series data will help you leverage Highcharts' full power to build rich, accurate visualizations.

4.2 Axes Configuration and Customization

Axes are the backbone of any chart, providing the scale and context that help users interpret the data visually. Highcharts offers powerful and flexible options to configure and customize both the X and Y axes to fit a variety of charting needs.

4.2.1 Basic Axis Configuration

Each axis can be customized through its configuration object, typically under `xAxis` or `yAxis`. Common options include:

4.2.2 `title`

Sets a descriptive label for the axis.

```
xAxis: {
  title: { text: 'Months' }
},
yAxis: {
  title: { text: 'Sales (units)' }
}
```

4.2.3 `tickInterval`

Controls the spacing between ticks on the axis.

```
xAxis: {
  tickInterval: 1  // Show a tick every category (for categorical axes)
},
yAxis: {
  tickInterval: 10 // Y-axis ticks every 10 units
}
```

4.2.4 `min` and `max`

Manually set the minimum and maximum values of an axis to control the visible range.

```
yAxis: {
  min: 0,
  max: 100
}
```

This is useful to maintain consistent scale across multiple charts or focus on a specific data range.

4.2.5 Custom Formatting (`labels.formatter` or `labels.format`)

Format axis labels to show units, dates, or custom strings.

```
xAxis: {
  labels: {
    formatter: function () {
      return this.value + ' units';
    }
  }
}
```

Or using a format string (available in newer versions):

```
yAxis: {
  labels: {
    format: '{value}°C'
  }
}
```

4.2.6 Multiple Axes and Dual-Scale Charts

Highcharts supports **multiple axes** on the same chart, enabling you to visualize datasets with different scales or units.

4.2.7 Adding Multiple Y-Axes

You can specify an array of Y-axes, each with its own configuration:

```
yAxis: [{
  title: { text: 'Temperature (°C)' },
  opposite: false // Left side (default)
}, {
  title: { text: 'Rainfall (mm)' },
  opposite: true  // Right side
}]
```

- The `opposite` property places the axis on the right side (true) or left side (false).
- Use this to create dual-scale charts where one series corresponds to the left Y-axis and another to the right.

4.2.8 Aligning Series to Axes

Each series can be linked to a specific Y-axis using the `yAxis` index:

```
series: [{
  name: 'Temperature',
  data: [15, 18, 21, 20],
  yAxis: 0   // Left axis
}, {
  name: 'Rainfall',
  data: [40, 60, 55, 70],
  yAxis: 1   // Right axis
}]
```

This enables clear comparison between different units or scales on the same chart.

4.2.9 Example: Dual-Scale Chart Configuration

```
Highcharts.chart('container', {
  xAxis: {
    categories: ['Jan', 'Feb', 'Mar', 'Apr']
  },
  yAxis: [{
    title: { text: 'Temperature (°C)' }
```

```
  }, {
    title: { text: 'Rainfall (mm)' },
    opposite: true
  }],
  series: [{
    name: 'Temperature',
    data: [7, 6.9, 9.5, 14.5],
    yAxis: 0
  }, {
    name: 'Rainfall',
    data: [49.9, 71.5, 106.4, 129.2],
    yAxis: 1
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Dual-Scale Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Dual-Scale Chart: Temperature and Rainfall</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr']
      },
      yAxis: [{
        title: { text: 'Temperature (°C)' }
      }, {
        title: { text: 'Rainfall (mm)' },
        opposite: true
      }],
      series: [{
        name: 'Temperature',
        data: [7, 6.9, 9.5, 14.5],
        yAxis: 0,
        type: 'line'
      }, {
        name: 'Rainfall',
        data: [49.9, 71.5, 106.4, 129.2],
        yAxis: 1,
        type: 'column'
      }]
    });
  </script>
</body>
</html>
```

4.2.10 Summary

- Use `title`, `tickInterval`, `min`, and `max` to control axis labeling, spacing, and scale.
- Customize labels with `formatter` or `format` to improve readability.
- Add multiple axes to handle data with different units or scales.
- Link series to specific axes with the `yAxis` property for dual-scale visualizations.

Mastering axis configuration ensures your charts communicate data clearly and accurately across various scenarios.

4.3 Titles, Subtitles, and Legends

Titles, subtitles, and legends are essential components that help users quickly interpret a chart. Highcharts provides full control over how these elements are displayed and styled, allowing you to tailor the user experience and visual clarity of your data.

4.3.1 Setting and Styling Titles and Subtitles

Every chart can include a **main title** and an optional **subtitle**, defined under the `title` and `subtitle` properties of the chart configuration.

4.3.2 Example: Basic Title and Subtitle

```
title: {
  text: 'Monthly Sales Performance'
},
subtitle: {
  text: 'Year 2024'
}
```

4.3.3 Styling Titles

Both `title` and `subtitle` support rich styling via the `style` property:

```
title: {
  text: 'Monthly Sales',
  style: {
    fontSize: '18px',
    fontWeight: 'bold',
```

```
    color: '#333'
  },
  align: 'center'  // Options: 'left', 'center', 'right'
},
subtitle: {
  text: 'Source: Internal Sales Database',
  style: {
    fontSize: '14px',
    color: '#666'
  }
}
```

You can also position titles using `x`, `y`, and `verticalAlign` properties for custom layouts.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Styled Titles Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Chart with Styled Title and Subtitle</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'line' },
      title: {
        text: 'Monthly Sales',
        style: {
          fontSize: '18px',
          fontWeight: 'bold',
          color: '#333'
        },
        align: 'center'  // 'left', 'center', or 'right'
      },
      subtitle: {
        text: 'Source: Internal Sales Database',
        style: {
          fontSize: '14px',
          color: '#666'
        }
      },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr']
      },
      yAxis: {
        title: {
          text: 'Revenue'
        }
      },
      series: [{
        name: '2024',
```

```
        data: [150, 200, 180, 220]
      }]
    });
  </script>
</body>
</html>
```

4.3.4 Controlling the Legend

The **legend** helps users identify which series correspond to which visual elements (lines, bars, slices, etc.). Highcharts allows full control over the legend's position, layout, and behavior.

4.3.5 Positioning the Legend

Use the `legend` object to control placement:

```
legend: {
  layout: 'vertical',       // or 'horizontal'
  align: 'right',           // 'left', 'center', or 'right'
  verticalAlign: 'middle',  // 'top', 'middle', or 'bottom'
  floating: false
}
```

4.3.6 Styling Legend Items

Customize font, spacing, and padding:

```
legend: {
  itemStyle: {
    fontWeight: 'normal',
    color: '#333'
  },
  itemHoverStyle: {
    color: '#000'
  },
  itemMarginTop: 5
}
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Legend Control Example</title>
```

```
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Chart with Customized Legend Position and Style</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'line' },
      title: { text: 'Monthly Sales with Custom Legend' },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr']
      },
      yAxis: {
        title: { text: 'Revenue' }
      },
      legend: {
        layout: 'vertical',       // vertical or horizontal
        align: 'right',           // left, center, or right
        verticalAlign: 'middle',  // top, middle, bottom
        floating: false,
        itemStyle: {
          fontWeight: 'normal',
          color: '#333'
        },
        itemHoverStyle: {
          color: '#000'
        },
        itemMarginTop: 5
      },
      series: [{
        name: '2024',
        data: [150, 200, 180, 220]
      }, {
        name: '2023',
        data: [130, 180, 150, 210]
      }]
    });
  </script>
</body>
</html>
```

4.3.7 Toggling Series Visibility with Legend

By default, clicking a legend item will **toggle the visibility** of its corresponding series. This interactive feature is especially useful in charts with many data series, allowing users to focus on what's most relevant.

4.3.8 Example: UX Improvement with Toggle

In a line chart with multiple product lines, clicking on a legend item can hide or show that product's data:

```
series: [{
  name: 'Product A',
  data: [5, 7, 3, 6]
}, {
  name: 'Product B',
  data: [3, 6, 4, 2]
}]
```

Toggling legend items is helpful when:

- Comparing individual series in a crowded chart
- Letting users control the focus
- Exploring combinations of data without redrawing the chart

You can disable legend toggling per series:

```
series: [{
  name: 'Fixed Series',
  data: [1, 2, 3],
  showInLegend: true,
  enableMouseTracking: false,
  visible: true,
  legendIndex: 0
}]
```

4.3.9 Summary

Feature	Key Properties
Title	`title.text`, `title.style`, `title.align`
Subtitle	`subtitle.text`, `subtitle.style`
Legend	`legend.layout`, `align`, `verticalAlign`, `itemStyle`
Interactivity	Click to toggle series visibility in the legend

Titles, subtitles, and legends provide structure, context, and usability to your Highcharts visualizations. Customizing these elements ensures your charts are not only functional, but also polished and user-friendly.

4.4 Tooltips and Data Labels

Highcharts provides two important mechanisms for displaying data directly to users: **tooltips**, which appear on hover, and **data labels**, which are drawn directly on chart elements. Both enhance readability and interactivity when used effectively.

4.4.1 Tooltips: Interactive Hover Information

Tooltips provide contextual data when a user hovers over a chart point. They are enabled by default and highly customizable.

4.4.2 Basic Tooltip Configuration

```
tooltip: {
  enabled: true,
  shared: false,
  valueSuffix: ' units'
}
```

- `enabled`: Turns tooltips on or off.
- `shared`: When `true`, displays all series values for the same X value in one tooltip (useful for multi-series charts).
- `valueSuffix`: Appends text to the value (e.g., "units", "°C", "%").

4.4.3 Using Format Strings

For quick formatting:

```
tooltip: {
  pointFormat: '{series.name}: <b>{point.y}</b><br/>'
}
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Tooltip Format String Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Tooltip Using Format Strings</h2>
```

```
  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'line' },
      title: { text: 'Monthly Revenue' },
      xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
      yAxis: { title: { text: 'Revenue ($)' } },
      tooltip: {
        pointFormat: '{series.name}: <b>{point.y}</b><br/>'
      },
      series: [{
        name: '2024',
        data: [150, 200, 180, 220]
      }, {
        name: '2023',
        data: [130, 180, 160, 210]
      }]
    });
  </script>
</body>
</html>
```

4.4.4 Custom Tooltip Formatter Function

For full control, use `formatter`:

```
tooltip: {
  formatter: function () {
    return `<b>${this.series.name}</b><br/>` +
           `X: ${this.x}<br/>Y: ${this.y}`;
  }
}
```

You can also format dates or categories conditionally:

```
tooltip: {
  formatter: function () {
    return `Date: ${Highcharts.dateFormat('%b %e, %Y', this.x)}<br/>` +
           `Value: <b>${this.y}</b>`;
  }
}
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Custom Tooltip Formatter</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
```

```
  <h2>Custom Tooltip Formatter Example</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'line' },
      title: { text: 'Sales Over Time' },
      xAxis: {
        type: 'datetime',
        title: { text: 'Date' }
      },
      yAxis: {
        title: { text: 'Sales ($)' }
      },
      tooltip: {
        formatter: function () {
          return `Date: <b>${Highcharts.dateFormat('%b %e, %Y', this.x)}</b><br/>` +
                 `${this.series.name}: <b>${this.y}</b>`;
        }
      },
      series: [{
        name: '2024 Sales',
        data: [
          [Date.UTC(2024, 0, 1), 150],
          [Date.UTC(2024, 1, 1), 200],
          [Date.UTC(2024, 2, 1), 180],
          [Date.UTC(2024, 3, 1), 220]
        ]
      }]
    });
  </script>
</body>
</html>
```

4.4.5 Data Labels: Showing Values Inside the Chart

Data labels display values **directly on points**, bars, or slices. These are useful for immediate insight without hovering, especially in static or printed charts.

4.4.6 Enabling Data Labels

```
plotOptions: {
  series: {
    dataLabels: {
      enabled: true,
      format: '{y}'  // Display the Y value
    }
  }
```

```
}
```

This enables labels for all series. You can also enable or style them per series:

```
series: [{
  name: 'Sales',
  data: [29, 71, 45],
  dataLabels: {
    enabled: true,
    color: '#000',
    style: { fontWeight: 'bold' }
  }
}]
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Data Labels Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Chart with Data Labels Showing Values</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Monthly Sales' },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar']
      },
      yAxis: {
        title: { text: 'Units Sold' }
      },
      plotOptions: {
        series: {
          dataLabels: {
            enabled: true,
            format: '{y}',
            style: {
              fontWeight: 'bold',
              color: '#000'
            }
          }
        }
      },
      series: [{
        name: 'Sales',
        data: [29, 71, 45]
      }]
    });
  </script>
</body>
</html>
```

4.4.7 Data Labels with Custom Formatters

```
dataLabels: {
  enabled: true,
  formatter: function () {
    return `${this.point.name}: ${this.y}`;
  }
}
```

4.4.8 Best Practices for Tooltips and Data Labels

Feature	Best Practice
Tooltips	Use for interactivity—keep them concise and consistent
Data Labels	Show only when space allows—avoid overcrowding the chart
Shared Tooltips	Use `shared: true` in multi-series line charts for comparison
Formatting	Format numbers, units, and dates for readability
Clarity	Avoid overlapping labels by enabling smart layout options

4.4.9 Avoiding Clutter

For charts with many points (e.g. scatter plots or large bar charts), excessive data labels can clutter the view. You can disable them selectively:

```
plotOptions: {
  series: {
    dataLabels: {
      enabled: false
    }
  }
}
```

Or conditionally show them only for certain values:

```
dataLabels: {
  enabled: true,
  formatter: function () {
    return this.y > 50 ? this.y : null;
  }
}
```

4.4.10 Summary

- **Tooltips** provide interactive, hover-based insight. Customize with format strings or functions.
- **Data labels** show values directly in the chart. Great for static contexts or when hovering isn't practical.
- Use both wisely: tooltips for interactivity, data labels for immediate visibility—without cluttering the chart.

By customizing tooltips and data labels, you can significantly improve how users perceive and understand your data visualizations.

4.5 Colors, Gradients, and Themes

Visual styling is key to building engaging, readable, and consistent charts. Highcharts provides a flexible system for applying colors, gradients, and themes to match your design needs—from simple palettes to full brand integration.

4.5.1 Setting Colors for Series and Points

By default, Highcharts cycles through a set of predefined colors, but you can override this behavior to apply specific colors to individual series or data points.

4.5.2 Set a Color for a Series

```
series: [{
  name: 'Sales',
  color: '#2f7ed8',
  data: [29, 71, 45, 60]
}]
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Series Color Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Chart with Custom Series Color</h2>
```

```
  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'line' },
      title: { text: 'Monthly Sales' },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr']
      },
      yAxis: {
        title: { text: 'Units Sold' }
      },
      series: [{
        name: 'Sales',
        color: '#2f7ed8', // Custom blue color
        data: [29, 71, 45, 60]
      }]
    });
  </script>
</body>
</html>
```

4.5.3 Set Colors for Individual Points

```
series: [{
  name: 'Revenue',
  data: [
    { y: 10, color: '#90ed7d' },
    { y: 20, color: '#f45b5b' },
    { y: 30, color: '#8085e9' }
  ]
}]
```

This is especially useful for pie charts or column/bar charts where each point should stand out individually.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Individual Point Colors</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Chart with Individual Point Colors</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
```

```
      chart: { type: 'column' },
      title: { text: 'Revenue by Category' },
      xAxis: {
        categories: ['Q1', 'Q2', 'Q3']
      },
      yAxis: {
        title: { text: 'Revenue (in millions)' }
      },
      series: [{
        name: 'Revenue',
        data: [
          { y: 10, color: '#90ed7d' },  // light green
          { y: 20, color: '#f45b5b' },  // red
          { y: 30, color: '#8085e9' }   // purple
        ]
      }]
    });
  </script>
</body>
</html>
```

4.5.4 Using Gradients for Visual Depth

Gradients can make your charts more visually dynamic by adding shading and depth. Highcharts supports SVG-based linear and radial gradients.

4.5.5 Example: Linear Gradient for a Column Chart

```
color: {
  linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
  stops: [
    [0, '#7cb5ec'],      // Start color
    [1, '#434348']       // End color
  ]
}
```

Apply this to a series or individual point:

```
series: [{
  type: 'column',
  data: [{
    y: 30,
    color: {
      linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
      stops: [
        [0, '#f7a35c'],
        [1, '#8085e9']
      ]
    }
```

```
  }]
}]
```

This technique is great for highlighting key bars, giving a polished and modern appearance.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Column Chart with Gradient Colors</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Column Chart with Linear Gradient Colors</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Sales with Gradient Colors' },
      xAxis: {
        categories: ['Product A', 'Product B', 'Product C']
      },
      yAxis: {
        title: { text: 'Units Sold' }
      },
      series: [{
        name: 'Sales',
        data: [
          {
            y: 30,
            color: {
              linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
              stops: [
                [0, '#f7a35c'], // top color
                [1, '#8085e9']  // bottom color
              ]
            }
          },
          {
            y: 45,
            color: {
              linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
              stops: [
                [0, '#7cb5ec'],
                [1, '#434348']
              ]
            }
          },
          {
            y: 25,
            color: {
              linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
              stops: [
```

```
                [0, '#90ed7d'],
                [1, '#1f7700']
              ]
            }
          }
        ]
      }]
    });
  </script>
</body>
</html>
```

4.5.6 Applying and Modifying Themes

Highcharts comes with several **built-in themes** (such as `dark-unica`, `grid-light`, `sand-signika`) that can be included via script tags. You can also **create a custom theme** to match your branding or app design.

4.5.7 Applying a Built-in Theme

1. Include a theme via CDN:

```
<script src="https://code.highcharts.com/themes/grid-light.js"></script>
```

2. This will globally affect all charts rendered after the theme is loaded.

4.5.8 Creating a Custom Theme with `Highcharts.setOptions`

To define your own theme, use `Highcharts.setOptions()` before rendering any charts.

4.5.9 Example: Custom Theme

```
Highcharts.setOptions({
  colors: ['#1abc9c', '#3498db', '#9b59b6', '#e74c3c'],
  chart: {
    backgroundColor: '#f4f4f4',
    style: {
      fontFamily: 'Arial, sans-serif'
    }
  },
  title: {
```

```
    style: {
      color: '#333',
      fontSize: '20px'
    }
  },
  legend: {
    itemStyle: {
      color: '#666'
    }
  }
});
```

This theme will apply to all subsequent charts on the page unless overridden locally.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Custom Theme Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Highcharts Chart with Custom Theme</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    // Define the custom theme before creating any chart
    Highcharts.setOptions({
      colors: ['#1abc9c', '#3498db', '#9b59b6', '#e74c3c'],
      chart: {
        backgroundColor: '#f4f4f4',
        style: {
          fontFamily: 'Arial, sans-serif'
        }
      },
      title: {
        style: {
          color: '#333',
          fontSize: '20px'
        }
      },
      legend: {
        itemStyle: {
          color: '#666'
        }
      }
    });

    // Now create the chart using the theme
    Highcharts.chart('container', {
      chart: {
        type: 'line'
      },
      title: {
```

```
        text: 'Monthly Sales with Custom Theme'
      },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr']
      },
      yAxis: {
        title: {
          text: 'Units Sold'
        }
      },
      series: [{
        name: 'Sales 2024',
        data: [29, 71, 45, 60]
      }, {
        name: 'Sales 2023',
        data: [20, 50, 40, 55]
      }]
    });
  </script>
</body>
</html>
```

4.5.10 Best Practices

Style Element	Recommendation
Series colors	Use consistent colors across charts for the same metric
Gradients	Use sparingly for emphasis or visual appeal
Themes	Centralize style control for brand consistency
Accessibility	Ensure sufficient contrast for readability

4.5.11 Summary

- **Color customization**: Directly style series or individual points with solid colors or gradients.
- **Gradients**: Add polish and depth using SVG-based color transitions.
- **Themes**: Apply built-in or custom themes to unify styling across charts using `Highcharts.setOptions`.

Styling charts isn't just cosmetic—when used well, color and design choices improve clarity, brand consistency, and user experience.

Chapter 5.

Advanced Features

1. Multiple Axes and Dual Axis Charts
2. Stacking and Grouping Series
3. Drilldown and Hierarchical Data
4. Exporting Charts (Images, PDF, SVG)
5. Accessibility Features

5 Advanced Features

5.1 Multiple Axes and Dual Axis Charts

In many real-world scenarios, you'll need to compare two or more datasets that have **different units or scales**—such as temperature vs. rainfall, or revenue vs. conversion rate. Using **multiple Y-axes**, especially **dual-axis charts**, helps visualize such comparisons clearly without distorting the scale of either dataset.

Highcharts makes it straightforward to add and configure multiple Y-axes, and to assign series to the appropriate one.

5.1.1 When to Use Multiple Y-Axes

Use multiple Y-axes when:

- Datasets use **different units** (e.g., °C vs. mm, dollars vs. percentage).
- One series has values **orders of magnitude different** from another.
- You want to visualize **multiple perspectives** on the same X-axis (e.g., time-based KPIs).

  WARNING Avoid using too many Y-axes, as it may confuse the viewer. Dual-axis charts are ideal for 2 datasets.

5.1.2 Creating a Dual Y-Axis Chart

To define two Y-axes in Highcharts, use an **array of axis configurations** under the `yAxis` property.

5.1.3 Step-by-Step: Basic Dual-Axis Chart

```
<div id="dual-axis-chart" style="width: 600px; height: 400px;"></div>

<script src="https://code.highcharts.com/highcharts.js"></script>
<script>
Highcharts.chart('dual-axis-chart', {
  chart: {
    zoomType: 'xy'
  },
  title: {
    text: 'Temperature and Rainfall'
  },
```

```
  xAxis: [{
    categories: ['Jan', 'Feb', 'Mar', 'Apr']
  }],
  yAxis: [{
    // Primary Y-axis (left)
    title: {
      text: 'Temperature (°C)'
    },
    labels: {
      format: '{value}°C'
    }
  }, {
    // Secondary Y-axis (right)
    title: {
      text: 'Rainfall (mm)'
    },
    labels: {
      format: '{value} mm'
    },
    opposite: true  // Puts axis on the right
  }],
  tooltip: {
    shared: true
  },
  series: [{
    name: 'Temperature',
    type: 'column',
    data: [7.0, 6.9, 9.5, 14.5],
    yAxis: 0  // Link to left axis
  }, {
    name: 'Rainfall',
    type: 'spline',
    data: [49.9, 71.5, 106.4, 129.2],
    yAxis: 1  // Link to right axis
  }]
});
</script>
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Dual-Axis Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Temperature and Rainfall Dual-Axis Chart</h2>

  <div id="dual-axis-chart" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('dual-axis-chart', {
      chart: {
        zoomType: 'xy'
      },
```

```
      title: {
        text: 'Temperature and Rainfall'
      },
      xAxis: [{
        categories: ['Jan', 'Feb', 'Mar', 'Apr']
      }],
      yAxis: [{
        title: {
          text: 'Temperature (°C)'
        },
        labels: {
          format: '{value}°C'
        }
      }, {
        title: {
          text: 'Rainfall (mm)'
        },
        labels: {
          format: '{value} mm'
        },
        opposite: true
      }],
      tooltip: {
        shared: true
      },
      series: [{
        name: 'Temperature',
        type: 'column',
        data: [7.0, 6.9, 9.5, 14.5],
        yAxis: 0
      }, {
        name: 'Rainfall',
        type: 'spline',
        data: [49.9, 71.5, 106.4, 129.2],
        yAxis: 1
      }]
    });
  </script>
</body>
</html>
```

5.1.4 Key Configuration Points

- **`yAxis: [{...}, {...}]`**: Define two axes, primary (left) and secondary (right).
- **`opposite: true`**: Puts the axis on the right-hand side.
- **`series[n].yAxis`**: Assign each series to an axis by index (`0` = left, `1` = right).
- **Tooltip**: Use `shared: true` for cleaner cross-reference on hover.

5.1.5 Customizing Axes Independently

Each axis can be styled or scaled separately:

```
yAxis: [{
  min: 0,
  max: 30,
  gridLineWidth: 1,
  title: { text: 'Temperature (°C)' }
}, {
  min: 0,
  max: 150,
  gridLineWidth: 0,
  title: { text: 'Rainfall (mm)' },
  opposite: true
}]
```

You can also add different tick intervals, formatting, or grid lines per axis.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Dual-Axis Chart with Custom Y-Axis Settings</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Dual-Axis Chart with Custom Axis Scaling and Styling</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { zoomType: 'xy' },
      title: { text: 'Temperature and Rainfall' },
      xAxis: [{ categories: ['Jan', 'Feb', 'Mar', 'Apr'] }],
      yAxis: [{
        min: 0,
        max: 30,
        gridLineWidth: 1,
        title: { text: 'Temperature (°C)' },
        labels: { format: '{value}°C' }
      }, {
        min: 0,
        max: 150,
        gridLineWidth: 0,
        title: { text: 'Rainfall (mm)' },
        labels: { format: '{value} mm' },
        opposite: true
      }],
      tooltip: { shared: true },
      series: [{
        name: 'Temperature',
        type: 'column',
        data: [7.0, 6.9, 9.5, 14.5],
        yAxis: 0
```

```
      }, {
        name: 'Rainfall',
        type: 'spline',
        data: [49.9, 71.5, 106.4, 129.2],
        yAxis: 1
      }]
    });
  </script>
</body>
</html>
```

5.1.6 Use Case Examples

Use Case	Left Y-Axis	Right Y-Axis
Weather report	Temperature (°C)	Rainfall (mm)
Business dashboard	Revenue ($)	Conversion rate (%)
Financial chart	Stock price ($)	Volume traded

5.1.7 Best Practices

- Use clear and distinct **axis labels** and **colors** to avoid confusion.
- Don't overload the chart—**two Y-axes is usually the maximum** for good readability.
- Consider color-coding series and their corresponding axis titles for better UX.

5.1.8 Summary

- **Multiple Y-axes** allow side-by-side comparison of unrelated units on a shared timeline or category axis.
- **Dual-axis charts** are ideal for 2 datasets with different units or scales.
- Use `yAxis` array and `series[n].yAxis` to configure and assign axes.
- Customize axis ranges, tick intervals, and label formats independently.

Dual-axis charts are a powerful feature that helps you present complex comparisons clearly without compromising visual clarity or accuracy.

5.2 Stacking and Grouping Series

Highcharts supports **stacked** and **grouped** charts, both of which are powerful tools for comparing multiple series across categories. While grouped charts emphasize **individual comparisons**, stacked charts are ideal for showing **cumulative totals**.

This section explains when to use each layout, and how to configure and label them effectively.

5.2.1 Grouped Charts: Comparing Series Side by Side

Grouped charts (the default behavior) display series side by side within each category. This layout makes it easy to compare values directly across series for a given category.

5.2.2 Example: Grouped Column Chart

```
Highcharts.chart('container', {
  chart: { type: 'column' },
  title: { text: 'Sales by Region' },
  xAxis: {
    categories: ['Q1', 'Q2', 'Q3', 'Q4']
  },
  yAxis: {
    min: 0,
    title: { text: 'Sales (k USD)' }
  },
  series: [{
    name: 'North',
    data: [30, 50, 40, 60]
  }, {
    name: 'South',
    data: [20, 40, 35, 55]
  }]
});
```

- Each series is displayed next to the others for every category (quarter).
- This format is best when the focus is on **individual contributions** rather than the total.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Grouped Column Chart Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
```

```
<body>
  <h2>Sales by Region (Grouped Column Chart)</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Sales by Region' },
      xAxis: {
        categories: ['Q1', 'Q2', 'Q3', 'Q4']
      },
      yAxis: {
        min: 0,
        title: { text: 'Sales (k USD)' }
      },
      series: [{
        name: 'North',
        data: [30, 50, 40, 60]
      }, {
        name: 'South',
        data: [20, 40, 35, 55]
      }]
    });
  </script>
</body>
</html>
```

5.2.3 Stacked Charts: Showing Cumulative Totals

Stacked charts place series on top of each other. This allows you to show how each part contributes to the total across a category.

5.2.4 Basic Stacked Column Chart

```
plotOptions: {
  column: {
    stacking: 'normal'  // Options: 'normal', 'percent'
  }
}
```

5.2.5 Full Example

```
Highcharts.chart('container', {
  chart: { type: 'column' },
```

```
  title: { text: 'Quarterly Sales by Region' },
  xAxis: {
    categories: ['Q1', 'Q2', 'Q3', 'Q4']
  },
  yAxis: {
    min: 0,
    title: { text: 'Total Sales (k USD)' },
    stackLabels: {
      enabled: true,
      style: {
        fontWeight: 'bold',
        color: 'gray'
      }
    }
  },
  plotOptions: {
    column: {
      stacking: 'normal',
      dataLabels: {
        enabled: true
      }
    }
  },
  series: [{
    name: 'North',
    data: [30, 50, 40, 60]
  }, {
    name: 'South',
    data: [20, 40, 35, 55]
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Stacked Column Chart with Stack Labels</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Quarterly Sales by Region (Stacked Column Chart)</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Quarterly Sales by Region' },
      xAxis: {
        categories: ['Q1', 'Q2', 'Q3', 'Q4']
      },
      yAxis: {
        min: 0,
        title: { text: 'Total Sales (k USD)' },
        stackLabels: {
```

```
          enabled: true,
          style: {
            fontWeight: 'bold',
            color: 'gray'
          }
        }
      },
      plotOptions: {
        column: {
          stacking: 'normal',
          dataLabels: {
            enabled: true
          }
        }
      },
      series: [{
        name: 'North',
        data: [30, 50, 40, 60]
      }, {
        name: 'South',
        data: [20, 40, 35, 55]
      }]
    });
  </script>
</body>
</html>
```

5.2.6 Key Configuration Options

- `stacking: 'normal'`: Standard vertical stacking.
- `stacking: 'percent'`: Stacks all series to total 100%, good for showing proportional contributions.
- `stackLabels`: Shows the total at the top of each stack.
- `dataLabels`: Shows individual values on each segment.

5.2.7 Grouped Stacked (Advanced)

Highcharts doesn't support grouped + stacked charts out of the box, but with custom configuration and `stack` identifiers, you can simulate grouped stacks.

```
series: [{
  name: 'Product A - 2023',
  data: [10, 20, 30],
  stack: '2023'
}, {
  name: 'Product B - 2023',
  data: [15, 25, 35],
  stack: '2023'
}, {
```

```
  name: 'Product A - 2024',
  data: [20, 30, 40],
  stack: '2024'
}, {
  name: 'Product B - 2024',
  data: [10, 15, 25],
  stack: '2024'
}]
```

Each `stack` group will be displayed adjacent to the others.

5.2.8 Best Practices

Layout	Use When…
Grouped	Comparing individual series across categories
Stacked	Showing cumulative values (e.g., total sales)
Percent	Highlighting relative proportions per category

- Always label stacks clearly using `stackLabels` or `dataLabels`.
- Use consistent colors for series across groups to enhance readability.
- Percent stacks work best with the `tooltip.shared` option enabled.

5.2.9 Summary

- Use **grouped charts** to compare values across series.
- Use **stacked charts** to show totals and part-to-whole relationships.
- Use `plotOptions.series.stacking = 'normal' | 'percent'` to control behavior.
- Label totals with `stackLabels` for quick insights into cumulative values.

Stacked and grouped series enhance your ability to present multi-dimensional data in a clean and informative way, helping your audience draw the right conclusions quickly.

5.3 Drilldown and Hierarchical Data

In many data visualizations, a **single chart is not enough** to convey layered or hierarchical information. This is where **drilldown charts** come in. Highcharts' drilldown module enables you to create interactive charts that let users click on a data point to explore deeper levels—perfect for dashboards, reports, and exploratory analytics.

5.3.1 What Is a Drilldown Chart?

A **drilldown chart** allows users to click on a chart element (e.g., a column, pie slice, or point) and "drill into" more specific data related to that element. This interaction supports **hierarchical exploration**, helping to:

- Break down high-level categories into subcategories
- Explore regions → countries → cities
- Navigate from product categories → individual SKUs
- Maintain a clean interface while supporting deeper insight

To use drilldown features, make sure to include the Highcharts Drilldown module.

5.3.2 Loading the Drilldown Module

Add the module after the core Highcharts library:

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/drilldown.js"></script>
```

5.3.3 Example: Regions Cities Drilldown

Here's a working example of a column chart where clicking on a region shows sales by city.

```
<div id="drilldown-container"></div>

<script>
Highcharts.chart('drilldown-container', {
  chart: {
    type: 'column'
  },
  title: {
    text: 'Regional Sales'
  },
  subtitle: {
    text: 'Click a region to view city-level data'
  },
  xAxis: {
    type: 'category'
  },
  legend: {
    enabled: false
  },
  plotOptions: {
    series: {
      borderWidth: 0,
      dataLabels: { enabled: true }
    }
  },
```

```
  series: [{
    name: 'Regions',
    colorByPoint: true,
    data: [{
      name: 'North',
      y: 150,
      drilldown: 'north-cities'
    }, {
      name: 'South',
      y: 120,
      drilldown: 'south-cities'
    }]
  }],
  drilldown: {
    series: [{
      id: 'north-cities',
      name: 'North Region Cities',
      data: [
        ['City A', 60],
        ['City B', 50],
        ['City C', 40]
      ]
    }, {
      id: 'south-cities',
      name: 'South Region Cities',
      data: [
        ['City D', 70],
        ['City E', 30],
        ['City F', 20]
      ]
    }]
  }
});
</script>
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Drilldown Column Chart</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <script src="https://code.highcharts.com/modules/drilldown.js"></script>
</head>
<body>
  <h2>Regional Sales Drilldown Chart</h2>
  <div id="drilldown-container" style="width: 700px; height: 400px;"></div>

  <script>
    Highcharts.chart('drilldown-container', {
      chart: { type: 'column' },
      title: { text: 'Regional Sales' },
      subtitle: { text: 'Click a region to view city-level data' },
      xAxis: { type: 'category' },
      legend: { enabled: false },
      plotOptions: {
```

```
        series: {
          borderWidth: 0,
          dataLabels: { enabled: true }
        }
      },
      series: [{
        name: 'Regions',
        colorByPoint: true,
        data: [
          { name: 'North', y: 150, drilldown: 'north-cities' },
          { name: 'South', y: 120, drilldown: 'south-cities' }
        ]
      }],
      drilldown: {
        series: [
          {
            id: 'north-cities',
            name: 'North Region Cities',
            data: [
              ['City A', 60],
              ['City B', 50],
              ['City C', 40]
            ]
          },
          {
            id: 'south-cities',
            name: 'South Region Cities',
            data: [
              ['City D', 70],
              ['City E', 30],
              ['City F', 20]
            ]
          }
        ]
      }
    });
  </script>
</body>
</html>
```

5.3.4 How It Works

- The top-level `series` defines the first-level categories (e.g., regions).
- Each point can optionally specify a `drilldown` ID that links to a series in the `drilldown.series` array.
- When a user clicks a column, Highcharts replaces the current series with the corresponding drilldown series.

5.3.5 Drilldown Configuration Keys

Key	Purpose
`drilldown`	Enables the drilldown module
`point.drilldown`	Links a point to a `drilldown.series.id`
`drilldown.series[]`	Holds the subcategory data, indexed by `id`
`chart.events.drilldown`	(Optional) Hook to load data dynamically

5.3.6 Dynamic Drilldowns (Optional)

You can load drilldown data **on demand** by listening to the `drilldown` event:

```
chart: {
  type: 'column',
  events: {
    drilldown: function (e) {
      if (!e.seriesOptions) {
        // Fetch data from server or compute dynamically
        this.addSeriesAsDrilldown(e.point, {
          name: 'Dynamic Series',
          data: [['Item A', 10], ['Item B', 20]]
        });
      }
    }
  }
}
```

This is useful for large datasets or remote data sources.

5.3.7 Best Practices

- **Limit the number of drill levels** to avoid overwhelming users.
- **Provide breadcrumbs or clear navigation** (Highcharts includes a back button by default).
- Use **tooltips** and **legends** to provide context at each level.
- Keep color schemes consistent across levels when possible.

5.3.8 Summary

- **Drilldown charts** allow users to interactively explore hierarchical data.
- Use the `drilldown` module and define `drilldown.series` to link subcategories.

- Customize behavior with events like `drilldown` and `drillup` for advanced workflows.
- Ideal for dashboards, business intelligence, and multi-level data summaries.

By using drilldowns, you empower users to interact with your charts and discover insights hidden behind summary-level data—without cluttering the main view.

5.4 Exporting Charts (Images, PDF, SVG)

Exporting is a key feature in data visualization, especially for sharing, reporting, or archiving charts. Highcharts includes an **exporting module** that allows users to save charts as PNG, JPEG, PDF, or SVG with just a click.

In this section, you'll learn how to enable and customize exporting options using the `exporting` module.

5.4.1 Including the Exporting Module

To enable export functionality, you must load the `exporting.js` module **after** Highcharts:

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
```

Optionally, add `export-data.js` to allow CSV/XLS download options:

```
<script src="https://code.highcharts.com/modules/export-data.js"></script>
```

5.4.2 Default Exporting Options

Once the exporting module is included, a small menu appears in the top-right corner of the chart. This gives users options to:

- Download as **PNG**, **JPEG**, **PDF**, or **SVG**
- View or download the chart data as **CSV** or **XLS** (if `export-data.js` is loaded)
- Print the chart

5.4.3 Basic Exporting Example

```
Highcharts.chart('container', {
  title: { text: 'Monthly Sales' },
```

```
  series: [{
    name: 'Sales',
    data: [10, 20, 30, 25]
  }],
  exporting: {
    enabled: true
  }
});
```

This will render the export menu button with default options.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highcharts Exporting Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <script src="https://code.highcharts.com/modules/exporting.js"></script>
</head>
<body>
  <h2>Monthly Sales Chart with Exporting</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      title: { text: 'Monthly Sales' },
      series: [{
        name: 'Sales',
        data: [10, 20, 30, 25]
      }],
      exporting: {
        enabled: true
      }
    });
  </script>
</body>
</html>
```

5.4.4 Customizing the Export Menu

You can tailor the export button's appearance and behavior using the `exporting.buttons` object.

5.4.5 Example: Customize the Export Button

```
exporting: {
  buttons: {
    contextButton: {
      text: 'Export',
      symbol: 'menu',
      menuItems: ['downloadPNG', 'downloadPDF', 'downloadSVG']
    }
  },
  filename: 'monthly-sales-report'
}
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Custom Export Button Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <script src="https://code.highcharts.com/modules/exporting.js"></script>
</head>
<body>
  <h2>Monthly Sales Chart with Custom Export Button</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      title: { text: 'Monthly Sales' },
      series: [{
        name: 'Sales',
        data: [10, 20, 30, 25]
      }],
      exporting: {
        filename: 'monthly-sales-report',
        buttons: {
          contextButton: {
            text: 'Export',
            symbol: 'menu',
            menuItems: ['downloadPNG', 'downloadPDF', 'downloadSVG']
          }
        }
      }
    });
  </script>
</body>
</html>
```

5.4.6 Menu Item Options

You can choose from:

- downloadPNG
- downloadJPEG
- downloadPDF
- downloadSVG
- downloadCSV
- downloadXLS
- viewData
- printChart

5.4.7 Customizing the Exported File

Set a Custom Filename

```
exporting: {
  filename: 'my-custom-chart'
}
```

The exported file will be named `my-custom-chart.png`, `my-custom-chart.pdf`, etc.

Customize Export Dimensions

```
exporting: {
  sourceWidth: 800,
  sourceHeight: 600,
  scale: 2  // Multiplies dimensions for higher resolution
}
```

Export Programmatically

You can trigger exports via code (e.g. a custom button outside the chart):

```
document.getElementById('exportBtn').addEventListener('click', function () {
  chart.exportChart({
    type: 'image/png',
    filename: 'custom-export'
  });
});
```

```
<button id="exportBtn">Export Chart</button>
```

5.4.8 Best Practices

Goal	How to Achieve It
Export with custom branding	Use `chart.backgroundColor`, `title`, and `credits`
High-res export	Increase `scale`, `sourceWidth`, and `sourceHeight`

Goal	How to Achieve It
Limited export options	Set `menuItems` to only include desired formats
Export with data	Include `export-data.js` for CSV/XLS options

5.4.9 Summary

- Include `exporting.js` to enable chart download options (PNG, PDF, SVG, etc.).
- Customize the export menu, filename, and dimensions using the `exporting` config.
- Use `exportChart()` to trigger downloads programmatically.
- For data exporting, add `export-data.js` to support CSV and XLS formats.

Exporting makes your Highcharts visualizations shareable and portable—perfect for reports, publications, and stakeholder communications.

5.5 Accessibility Features

Highcharts prioritizes accessibility to ensure that data visualizations are usable by all individuals, including people with disabilities. Whether you're building a public-facing dashboard or an internal analytics tool, making your charts accessible is a critical part of inclusive design.

This section explores the built-in accessibility features provided by Highcharts and offers best practices to improve chart readability for all users.

5.5.1 Built-in Accessibility Module

Highcharts provides a dedicated **accessibility module** (`accessibility.js`) that enhances charts with semantic structure, keyboard navigation, and screen reader support.

5.5.2 How to Include It

Add the module after Highcharts:

```
<script src="https://code.highcharts.com/modules/accessibility.js"></script>
```

Once included, accessibility enhancements are applied automatically to your charts.

5.5.3 Screen Reader Support

With the accessibility module enabled, Highcharts automatically:

- Adds **ARIA roles** and **labels** to the chart container.
- Generates **long descriptions** of the chart and its contents.
- Allows assistive technologies (like screen readers) to interpret chart elements meaningfully.

You can provide custom descriptions using:

```
accessibility: {
  description: 'This chart compares revenue and profit across four quarters.',
  enabled: true
}
```

5.5.4 Keyboard Navigation

Users can explore charts using a keyboard:

- **Arrow keys** to move between series and data points.
- **Enter/Space** to activate tooltips or drilldowns.
- **Tab** to move into and out of the chart.

Enable or customize keyboard navigation:

```
accessibility: {
  keyboardNavigation: {
    enabled: true
  }
}
```

This is especially useful for users who can't use a mouse or touch interface.

5.5.5 ARIA Labels and Semantics

Highcharts automatically adds:

- `role="img"` on the chart container.
- `aria-label` with a text description of the chart content.
- Accessible elements for chart headings, axes, and data points.

You can customize these with:

```
accessibility: {
  screenReaderSection: {
    beforeChartFormat:
      '<h5>{chartTitle}</h5><div>{typeDescription}</div><div>{chartSubtitle}</div>'
```

```
  }
}
```

5.5.6 Best Practices for Accessible Charts

Consideration	Recommendation
Color contrast	Use high-contrast colors that meet WCAG standards (minimum 4.5:1)
Color-independent	Never rely on color alone—use patterns, labels, or shapes for distinction
Data labels	Enable `dataLabels` on points for context without hover
Keyboard navigation	Always enable for users without a mouse
Chart description	Add a meaningful `accessibility.description`
Legend visibility	Ensure all series are labeled clearly in the legend

5.5.7 Example: High-contrast and Labelled Chart

```
Highcharts.chart('container', {
  chart: { type: 'bar' },
  title: { text: 'Top Products' },
  series: [{
    name: 'Sales',
    data: [50, 70, 30],
    dataLabels: { enabled: true }
  }],
  accessibility: {
    description: 'A bar chart showing sales of top three products. Product A had the highest sales.',
    enabled: true
  }
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>High-Contrast Accessible Bar Chart</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Top Products Sales (Accessible Bar Chart)</h2>
```

```
  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'bar' },
      title: { text: 'Top Products' },
      series: [{
        name: 'Sales',
        data: [50, 70, 30],
        dataLabels: { enabled: true }
      }],
      accessibility: {
        description: 'A bar chart showing sales of top three products. Product A had the highest sales.
        enabled: true
      }
    });
  </script>
</body>
</html>
```

5.5.8 Summary

- Highcharts' **accessibility module** adds ARIA support, screen reader descriptions, and keyboard navigation out of the box.
- Customize descriptions and structure to give assistive technologies more context.
- Follow design best practices: high contrast, labeled data, and keyboard-friendly controls.
- Accessibility isn't just a feature—it's a responsibility for inclusive visualization design.

By building accessible charts, you ensure that your data is not only powerful but also usable by everyone.

Chapter 6.

Interactivity and Animation

1. Mouse and Touch Events
2. Dynamic Updates and Live Data
3. Zooming, Panning, and Scrollbar
4. Custom Animation and Transitions

6 Interactivity and Animation

6.1 Mouse and Touch Events

Highcharts provides robust support for **mouse and touch interactions**, allowing you to attach custom behaviors to chart elements like points, series, and axes. These events make your charts more engaging and interactive—whether you're building dashboards, analytics apps, or dynamic visual storytelling experiences.

In this section, you'll learn how to use `click`, `mouseOver`, and `touch` events to add interactive features like opening modals, highlighting related elements, or triggering external actions.

6.1.1 Event Handling in Highcharts

Highcharts lets you attach event handlers to:

- **Chart-wide events** (e.g., `click` on the background)
- **Individual data points** (e.g., bar or pie slice clicks)
- **Series** (e.g., hovering over an entire line)
- **Legend items**, **axes**, and more

These events are specified within the `plotOptions` or `chart.events` configuration.

6.1.2 Example 1: Clicking a Column to Open a Modal

You can attach a `click` event to individual points like this:

```
Highcharts.chart('container', {
  chart: { type: 'column' },
  title: { text: 'Revenue by Product' },
  xAxis: { categories: ['Product A', 'Product B', 'Product C'] },
  series: [{
    name: 'Revenue',
    data: [100, 200, 150],
    point: {
      events: {
        click: function () {
          alert('You clicked on ' + this.category + ' with value ' + this.y);
          // You could also open a modal or fetch more details here
        }
      }
    }
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Column Click Event Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Revenue by Product (Click Column)</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Revenue by Product' },
      xAxis: { categories: ['Product A', 'Product B', 'Product C'] },
      series: [{
        name: 'Revenue',
        data: [100, 200, 150],
        point: {
          events: {
            click: function () {
              alert('You clicked on ' + this.category + ' with value ' + this.y);
              // Replace alert with modal logic if desired
            }
          }
        }
      }]
    });
  </script>
</body>
</html>
```

6.1.3 What's Happening?

- `point.events.click` is triggered when a user clicks on a bar.
- `this` refers to the point object (with `category`, `y`, and more).

You can replace the `alert` with custom functions like opening a modal or navigating to a detail page.

6.1.4 Example 2: Highlighting Related Points on Hover

You can change the visual appearance of related data when hovering over a point:

```
plotOptions: {
  series: {
    point: {
```

```
      events: {
        mouseOver: function () {
          this.update({ color: 'orange' });
        },
        mouseOut: function () {
          this.update({ color: undefined }); // reset
        }
      }
    }
  }
}
```

This example changes the color of a bar or point on hover, then resets it on mouse out.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Highlight Points on Hover Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Hover to Highlight Points</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Sales by Product' },
      xAxis: { categories: ['Product A', 'Product B', 'Product C'] },
      series: [{
        name: 'Sales',
        data: [100, 200, 150]
      }],
      plotOptions: {
        series: {
          point: {
            events: {
              mouseOver: function () {
                this.update({ color: 'orange' });
              },
              mouseOut: function () {
                this.update({ color: undefined }); // reset to default
              }
            }
          }
        }
      }
    });
  </script>
</body>
</html>
```

6.1.5 Example 3: Handling Touch Events

Highcharts supports touch events on mobile devices natively. The same `click` and `hover` events work on touchscreens, but you can also detect device type and customize the experience:

```
plotOptions: {
  series: {
    point: {
      events: {
        click: function () {
          if ('ontouchstart' in window) {
            console.log('Touch tap on ' + this.name);
          }
        }
      }
    }
  }
}
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Touch Event Handling Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Touch Event Detection on Data Points</h2>

  <div id="container" style="width: 600px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Sales by Product' },
      xAxis: { categories: ['Product A', 'Product B', 'Product C'] },
      series: [{
        name: 'Sales',
        data: [100, 200, 150]
      }],
      plotOptions: {
        series: {
          point: {
            events: {
              click: function () {
                if ('ontouchstart' in window) {
                  console.log('Touch tap on ' + this.category);
                  alert('Touch tap on ' + this.category);
                } else {
                  alert('Clicked on ' + this.category);
                }
              }
            }
          }
        }
      }
```

```
      }
    });
  </script>
</body>
</html>
```

6.1.6 Chart-Level Event Example

To respond to clicks anywhere in the chart background:

```
chart: {
  events: {
    click: function (event) {
      console.log('Clicked at x:', event.xAxis[0].value);
    }
  }
}
```

You can use this for drawing reference lines, capturing user input, or defining interactive zones.

6.1.7 Best Practices

Interaction Type	Use Case Example
`point.click`	Drilldown to more data, show modal or tooltip
`point.mouseOver`	Highlight series, show related points dynamically
`chart.click`	Add custom annotations or guide interactions
`touch`	Create mobile-friendly gestures and tap handlers

- Keep interactions **discoverable** (e.g., cursor changes, tooltips).
- Avoid overloading interactions—**one or two per chart element is plenty**.
- Use `tooltip.shared` and `crosshairs` for multi-series interactions.

6.1.8 Summary

Highcharts enables interactive charts by supporting mouse and touch events at multiple levels—from individual points to the entire chart. By handling events like `click` and `mouseOver`, you can turn static data visualizations into responsive, data-driven interfaces.

In the next sections, we'll expand this interactivity further with **real-time updates**, **zooming and panning**, and **custom animation effects**.

6.2 Dynamic Updates and Live Data

Highcharts makes it easy to create **dynamic, real-time charts** that update with new data—perfect for dashboards, monitoring tools, or any application where the data changes frequently.

In this section, you'll learn how to:

- Add new points using `series.addPoint()`
- Replace entire datasets with `series.setData()`
- Smoothly animate frequent updates

6.2.1 Why Use Dynamic Updates?

Dynamic charts let you:

- Monitor stock prices, sensor data, or server metrics in real time
- Reflect user input or external events
- Keep visualizations fresh without reloading the page

Highcharts provides a simple API to update existing charts without reinitializing them.

6.2.2 Method 1: Adding Points with `series.addPoint()`

Use this when you want to **append new data** to an existing chart—ideal for time-series data like live traffic, CPU usage, or stock tickers.

6.2.3 Example: Add a Point Every Second

```
<div id="container"></div>
<script>
const chart = Highcharts.chart('container', {
  chart: {
    type: 'line'
  },
  title: { text: 'Live Data Stream' },
  series: [{
    name: 'Random Data',
    data: []
  }]
});

// Simulate live data every second
```

```
setInterval(() => {
  const x = (new Date()).getTime();
  const y = Math.floor(Math.random() * 100);
  chart.series[0].addPoint([x, y], true, chart.series[0].data.length >= 20);
}, 1000);
</script>
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Live Data Stream Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Live Data Stream (Appending Points)</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    const chart = Highcharts.chart('container', {
      chart: {
        type: 'line',
        animation: Highcharts.svg, // use SVG animation
        marginRight: 10,
        events: {
          load: function () {
            // Set up the updating of the chart each second
            const series = this.series[0];
            setInterval(() => {
              const x = (new Date()).getTime(); // current time
              const y = Math.floor(Math.random() * 100);
              series.addPoint([x, y], true, series.data.length >= 20);
            }, 1000);
          }
        }
      },
      title: { text: 'Live Data Stream' },
      xAxis: {
        type: 'datetime',
        tickPixelInterval: 150
      },
      yAxis: {
        title: { text: 'Random value' },
        max: 100,
        min: 0
      },
      series: [{
        name: 'Random Data',
        data: (function () {
          // Generate initial empty data array or seed with initial points if you like
          return [];
        })()
      }]
    });
```

```
  </script>
</body>
</html>
```

6.2.4 Notes

- `addPoint([x, y], redraw, shift)`:
  - `redraw = true`: Update the chart immediately
  - `shift = true`: Remove the oldest point to maintain a fixed-length series

6.2.5 Method 2: Replacing Data with `series.setData()`

Use this when:

- You want to **replace the entire dataset**
- You're syncing with a new API response
- You're switching between different views (e.g., daily vs monthly)

6.2.6 Example: Replace Data on Button Click

```
document.getElementById('update').addEventListener('click', () => {
  const newData = Array.from({ length: 10 }, () => Math.floor(Math.random() * 50));
  chart.series[0].setData(newData, true);
});
```

This is ideal for dashboard controls or filters that change the dataset.

6.2.7 Performance Tips for Frequent Updates

When updating charts frequently (e.g. several times per second):

Tip	Why It Helps
Use `addPoint(..., false)`	Avoids redrawing too often; call `chart.redraw()` manually
Limit data points with `shift`	Prevents memory bloat and clutter
Disable animation if needed	Reduces visual lag during rapid updates

6.2.8 Example: Batched Updates Without Animation

```
chart.series[0].addPoint([x, y], false); // Don't redraw yet
chart.redraw();                          // Redraw once per batch
```

You can also turn off animation temporarily:

```
chart.update({
  plotOptions: {
    series: {
      animation: false
    }
  }
});
```

6.2.9 Updating Other Chart Properties

You can update more than just data. Try changing:

- **Title**
  ```
  chart.setTitle({ text: 'Updated Title' });
  ```
- **Axis extremes**
  ```
  chart.xAxis[0].setExtremes(min, max);
  ```
- **Color or style**
  ```
  chart.series[0].update({ color: 'red' });
  ```

6.2.10 Summary

- Use `series.addPoint()` to append new data in real time.
- Use `series.setData()` to replace entire datasets.
- Optimize performance with redraw batching, data limits, and optional animation.
- Highcharts' live update capabilities make it ideal for real-time apps and dashboards.

In the next section, we'll explore how to let users **zoom**, **pan**, and **scroll** through these data-rich visualizations.

6.3 Zooming, Panning, and Scrollbar

When working with large datasets or detailed time series, it's often impractical to display all the data at once. Highcharts offers interactive tools like **zooming**, **panning**, and **scrollbars** to let users focus on specific sections of a chart without overwhelming them with information.

In this section, you'll learn how to:

- Enable zooming with `chart.zoomType`
- Allow panning with `panning` and `panKey`
- Add scrollbars to long axes

6.3.1 Enabling Zooming

Zooming lets users **click and drag** to zoom in on specific regions of a chart. This works especially well for:

- Time series
- Financial/stock charts
- Sensor or log data

6.3.2 Example: Enable X-Axis Zooming

```
Highcharts.chart('container', {
  chart: {
    zoomType: 'x' // can also be 'y' or 'xy'
  },
  title: { text: 'Website Traffic Over Time' },
  xAxis: {
    type: 'datetime'
  },
  series: [{
    name: 'Visitors',
    data: generateTimeSeries() // your function to return timestamped data
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>X-Axis Zoom Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
```

```
  <h2>Website Traffic Over Time (Zoomable X-Axis)</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    // Generate sample timestamped data (last 30 days, 1 point per day)
    function generateTimeSeries() {
      const data = [];
      const now = Date.now();
      const day = 24 * 3600 * 1000;

      for (let i = 30; i >= 0; i--) {
        data.push([
          now - i * day,
          Math.floor(50 + Math.random() * 100)
        ]);
      }
      return data;
    }

    Highcharts.chart('container', {
      chart: {
        zoomType: 'x'
      },
      title: { text: 'Website Traffic Over Time' },
      xAxis: {
        type: 'datetime'
      },
      yAxis: {
        title: { text: 'Visitors' }
      },
      series: [{
        name: 'Visitors',
        data: generateTimeSeries()
      }]
    });
  </script>
</body>
</html>
```

6.3.3 zoomType Options

Value	Behavior
`'x'`	Zoom along X-axis
`'y'`	Zoom along Y-axis
`'xy'`	Zoom in both directions

6.3.4 Enabling Panning

Panning allows users to **click and drag** the chart area (with a modifier key, like Shift) to navigate after zooming in.

6.3.5 Example: Enable Panning with Shift Key

```
chart: {
  zoomType: 'x',
  panning: {
    enabled: true,
    type: 'x'
  },
  panKey: 'shift' // Hold Shift + drag to pan
}
```

This allows intuitive exploration: users zoom in to see details, then pan left or right without resetting the zoom.

6.3.6 Adding a Scrollbar

Scrollbars provide another navigation option, particularly for **large datasets or long category axes**. Highcharts can display scrollbars automatically when you set a `min`/`max` window that is smaller than the full dataset.

6.3.7 Example: Horizontal Scrollbar for Category Axis

```
Highcharts.chart('container', {
  chart: {
    type: 'column'
  },
  title: { text: 'Product Sales by Region' },
  xAxis: {
    categories: ['North', 'South', 'East', 'West', 'Central', 'Asia', 'Europe', 'Africa'],
    min: 0,
    max: 3,
    scrollbar: {
      enabled: true
    }
  },
  series: [{
    name: 'Sales',
    data: [100, 200, 150, 300, 120, 180, 90, 60]
  }]
```

```
});
```

Use `min` and `max` to define the initial viewable range. The scrollbar will appear if the full axis has more items than shown.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Horizontal Scrollbar on Category Axis</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <script src="https://code.highcharts.com/modules/scrollbar.js"></script>
</head>
<body>
  <h2>Product Sales by Region with Horizontal Scrollbar</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Product Sales by Region' },
      xAxis: {
        categories: ['North', 'South', 'East', 'West', 'Central', 'Asia', 'Europe', 'Africa'],
        min: 0,
        max: 3, // Show only first 4 categories initially
        scrollbar: { enabled: true },
        // Enable navigator for a nicer experience (optional)
        // navigator: { enabled: true }
      },
      series: [{
        name: 'Sales',
        data: [100, 200, 150, 300, 120, 180, 90, 60]
      }]
    });
  </script>
</body>
</html>
```

6.3.8 Use Case: Stock and Financial Charts

Interactive navigation features are especially valuable in stock and financial charts:

- Zoom in to a specific date range
- Scroll across years of data
- Pan through hourly/daily intervals

Highcharts Stock (a specialized version of Highcharts) includes even more zoom/pan controls and built-in range selectors.

6.3.9 Summary

Feature	Enables Users To…	Key Configs
Zooming	Focus on a specific area	`chart.zoomType`
Panning	Navigate horizontally or vertically	`chart.panning`, `chart.panKey`
Scrollbar	Scroll across long axes or categories	`xAxis.scrollbar`, `min`, `max`

By combining these features, you create a powerful and intuitive data exploration interface—essential for users who need to dive deep into dense or long-running datasets.

In the next section, we'll explore **custom animations and transitions** to add polish and motion to your charts.

6.4 Custom Animation and Transitions

Highcharts brings charts to life with built-in animation support, making transitions smooth and engaging. Whether you're drawing a chart for the first time, updating data dynamically, or interacting with elements, animations help users understand what's changing and where to focus.

In this section, you'll learn how to:

- Customize animation duration and easing
- Control animation when adding or updating data
- Animate redraws for smoother updates

6.4.1 Animating Initial Chart Rendering

By default, Highcharts animates the initial rendering of series. You can configure the animation duration and easing using the `plotOptions.series.animation` object.

6.4.2 Example: Custom Initial Animation

```
Highcharts.chart('container', {
  chart: { type: 'column' },
  title: { text: 'Quarterly Revenue' },
  plotOptions: {
    series: {
      animation: {
```

```
        duration: 1500,
        easing: 'easeOutBounce'
      }
    }
  },
  series: [{
    name: 'Revenue',
    data: [120, 180, 140, 200]
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Custom Initial Animation Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Quarterly Revenue with Custom Animation</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Quarterly Revenue' },
      plotOptions: {
        series: {
          animation: {
            duration: 1500,
            easing: 'easeOutBounce'
          }
        }
      },
      series: [{
        name: 'Revenue',
        data: [120, 180, 140, 200]
      }]
    });
  </script>
</body>
</html>
```

6.4.3 Supported Easing Types

Highcharts supports common CSS-like easing values:

- `'linear'`
- `'easeInQuad'`

- `'easeOutQuad'`
- `'easeInOutQuad'`
- `'easeOutBounce'` (fun for playful UIs!)

  For full control, you can supply a custom easing function via Highcharts' internal animation engine.

6.4.4 Animating Data Updates

When a chart is updated dynamically using `setData()`, `addPoint()`, or `series.update()`, Highcharts animates the transition if animation is enabled.

6.4.5 Example: Animate Updated Data

```
const chart = Highcharts.chart('container', {
  chart: { type: 'line' },
  series: [{
    name: 'Live Score',
    data: [10, 20, 30, 40]
  }]
});

// Simulate data update with animation
setTimeout(() => {
  chart.series[0].setData([15, 25, 35, 45], true, {
    duration: 1000,
    easing: 'easeOutBounce'
  });
}, 2000);
```

The third parameter of `setData()` is an animation config object.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Animate Updated Data Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
  <h2>Animate Updated Data on Line Chart</h2>

  <div id="container" style="width: 700px; height: 400px;"></div>

  <script>
    const chart = Highcharts.chart('container', {
```

```
      chart: { type: 'line' },
      title: { text: 'Live Score' },
      series: [{
        name: 'Live Score',
        data: [10, 20, 30, 40]
      }]
    });

    // Update data after 2 seconds with animation
    setTimeout(() => {
      chart.series[0].setData([15, 25, 35, 45], true, {
        duration: 1000,
        easing: 'easeOutBounce'
      });
    }, 2000);
  </script>
</body>
</html>
```

6.4.6 Redrawing with Animation

When you make multiple changes—like axis extremes, title updates, or color changes—you can use `chart.redraw()` with animation.

6.4.7 Example: Animated Redraw

```
chart.series[0].addPoint(50, false); // No redraw yet
chart.series[0].addPoint(60, false); // Still batching
chart.redraw({ duration: 800 });     // Now animate the update
```

This technique ensures that multiple updates are processed smoothly without triggering several redraws.

6.4.8 Animating Axis Transitions

Axes can also animate when you change their extremes using `setExtremes()`.

```
chart.xAxis[0].setExtremes(2, 6, true, { duration: 600, easing: 'easeInOutQuad' });
```

This is particularly helpful in zoomable time-series charts where axes move frequently.

6.4.9 Turning Off Animation

For performance or UX reasons, you might want to disable animations:

```
plotOptions: {
  series: {
    animation: false
  }
}
```

Or selectively disable animation on a method call:

```
series.setData(newData, true, false); // Third param: animation = false
```

6.4.10 Best Practices

Use Case	Recommendation
Initial load	Use moderate duration (800–1500ms)
Real-time updates	Keep animations subtle and fast
Axis or series changes	Use easing like `'easeInOutQuad'`
Performance-critical dashboards	Disable or limit animation

6.4.11 Summary

- Use `plotOptions.series.animation` to control how charts render.
- Animate updates with `setData()`, `addPoint()`, or `setExtremes()` using duration/easing.
- Use `redraw()` batching for smoother composite updates.
- Balance motion with usability—keep transitions clean, helpful, and quick.

With the right animations, your charts won't just *look* great—they'll communicate change clearly and intuitively.

Chapter 7.

Responsive Design and Theming

1. Responsive Configuration Options
2. Creating and Applying Themes
3. Dark Mode and Custom Styling
4. Mobile Optimization Tips

7 Responsive Design and Theming

7.1 Responsive Configuration Options

Creating charts that look great on all screen sizes—from large desktop monitors to compact mobile devices—is essential for a modern data visualization experience. Highcharts supports responsive design through a built-in `responsive` configuration block, enabling you to adapt your chart's layout, styling, and features based on the available screen width.

In this section, you'll learn how to:

- Use the `responsive.rules` array
- Change layout settings like legend position and font sizes
- Apply conditional styles based on screen size

7.1.1 The `responsive` Block: An Overview

The `responsive` configuration lets you define a set of **rules**, each with a condition and a corresponding set of chart options that should be applied when the condition is met.

7.1.2 Basic Structure

```
responsive: {
  rules: [{
    condition: {
      maxWidth: 500
    },
    chartOptions: {
      legend: {
        layout: 'vertical',
        align: 'center',
        verticalAlign: 'bottom'
      }
    }
  }]
}
```

Each rule has:

- `condition`: Defines when the rule applies (e.g., based on `maxWidth`, `minHeight`, etc.)
- `chartOptions`: The Highcharts options to override when the condition is met

7.1.3 Example: Switch to Vertical Legend on Small Screens

This is a common use case—placing the legend below the chart on mobile instead of to the right.

```
Highcharts.chart('container', {
  chart: { type: 'column' },
  title: { text: 'Sales by Region' },
  legend: {
    layout: 'horizontal',
    align: 'right',
    verticalAlign: 'middle'
  },
  series: [{ name: 'Q1', data: [100, 120, 140] }],
  responsive: {
    rules: [{
      condition: {
        maxWidth: 600
      },
      chartOptions: {
        legend: {
          layout: 'vertical',
          align: 'center',
          verticalAlign: 'bottom'
        }
      }
    }]
  }
});
```

When the screen width is below 600 pixels, the legend repositions to the bottom in a vertical format.

Full runnable code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Responsive Highcharts Legend</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    #container {
      max-width: 800px;
      margin: 0 auto;
    }
  </style>
</head>
<body>
  <div id="container"></div>

  <script>
    Highcharts.chart('container', {
      chart: {
        type: 'column'
      },
```

```
      title: {
        text: 'Sales by Region'
      },
      xAxis: {
        categories: ['North', 'South', 'East']
      },
      yAxis: {
        title: {
          text: 'Sales (k USD)'
        }
      },
      legend: {
        layout: 'horizontal',
        align: 'right',
        verticalAlign: 'middle'
      },
      series: [{
        name: 'Q1',
        data: [100, 120, 140]
      }, {
        name: 'Q2',
        data: [90, 110, 130]
      }],
      responsive: {
        rules: [{
          condition: {
            maxWidth: 600
          },
          chartOptions: {
            legend: {
              layout: 'vertical',
              align: 'center',
              verticalAlign: 'bottom'
            }
          }
        }]
      }
    });
  </script>
</body>
</html>
```

7.1.4 Adapting Fonts and Spacing

You can also adjust font sizes and spacing to keep charts readable on small screens.

7.1.5 Example: Smaller Fonts for Mobile

```
responsive: {
  rules: [{
    condition: { maxWidth: 500 },
    chartOptions: {
      title: {
        style: {
          fontSize: '14px'
        }
      },
      xAxis: {
        labels: {
          style: {
            fontSize: '10px'
          }
        }
      }
    }
  }]
}
```

7.1.6 Multiple Rules for Different Breakpoints

You can define several responsive rules to fine-tune your charts across screen sizes:

```
responsive: {
  rules: [
    {
      condition: { maxWidth: 800 },
      chartOptions: {
        chart: { marginRight: 50 },
        legend: { enabled: true }
      }
    },
    {
      condition: { maxWidth: 500 },
      chartOptions: {
        legend: {
          layout: 'vertical',
          align: 'center',
          verticalAlign: 'bottom'
        },
        title: {
          style: { fontSize: '12px' }
        }
      }
    }
  ]
}
```

7.1.7 When to Use Responsive Configs

Chart Element	Why It Matters on Small Screens
Legend placement	Prevents overlap and saves horizontal space
Font size	Improves readability on mobile
Margins/padding	Helps avoid cutoff labels or content
Axis labels	Reduces clutter when space is tight

7.1.8 Summary

- The `responsive` block allows Highcharts to **adapt chart layouts based on screen size**.
- Use `condition.maxWidth` to target mobile devices and adjust layout or styles.
- Responsive design enhances readability, usability, and mobile-friendliness.
- You can stack multiple rules to match different breakpoints—just like in CSS media queries.

In the next section, we'll explore how to create and apply **custom themes** to maintain consistent design across all charts.

7.2 Creating and Applying Themes

In any data visualization project, **consistent visual style** is key to professionalism and usability. Highcharts makes it easy to apply consistent colors, fonts, backgrounds, and other stylistic settings across multiple charts using **themes**.

In this section, you'll learn how to:

- Define a theme using `Highcharts.setOptions()`
- Customize chart colors, fonts, borders, and backgrounds
- Apply themes globally across charts for design consistency

7.2.1 What Is a Highcharts Theme?

A theme is simply a JavaScript object passed to `Highcharts.setOptions()` that modifies the default styling options for all charts created after it's applied.

Themes allow you to:

- Use consistent colors across your dashboard
- Match corporate branding (fonts, color palette)

- Define dark/light modes
- Reduce code duplication across chart configs

7.2.2 Creating a Custom Theme

To create a theme, define a plain JavaScript object with chart-wide styling properties, and pass it to `Highcharts.setOptions()` **before** creating any charts.

7.2.3 Example: A Minimal Custom Theme

```
Highcharts.setOptions({
  colors: ['#007acc', '#e91e63', '#ffc107', '#4caf50'],
  chart: {
    backgroundColor: '#f4f4f4',
    style: {
      fontFamily: 'Segoe UI, sans-serif'
    }
  },
  title: {
    style: {
      color: '#333333',
      fontSize: '16px'
    }
  },
  legend: {
    itemStyle: {
      fontWeight: 'normal',
      color: '#444'
    }
  }
});
```

Now, all subsequent charts will automatically use these styles.

Full runnable code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Highcharts Minimal Custom Theme</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      margin: 0;
      padding: 2em;
      background: #fafafa;
    }
    #container {
```

```
      max-width: 800px;
      margin: 0 auto;
    }
  </style>
</head>
<body>
  <div id="container"></div>

  <script>
    // Apply the custom theme
    Highcharts.setOptions({
      colors: ['#007acc', '#e91e63', '#ffc107', '#4caf50'],
      chart: {
        backgroundColor: '#f4f4f4',
        style: {
          fontFamily: 'Segoe UI, sans-serif'
        }
      },
      title: {
        style: {
          color: '#333333',
          fontSize: '16px'
        }
      },
      legend: {
        itemStyle: {
          fontWeight: 'normal',
          color: '#444'
        }
      }
    });

    // Create the chart
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: 'Monthly Revenue' },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr']
      },
      yAxis: {
        title: { text: 'Revenue (k USD)' }
      },
      series: [{
        name: 'Product A',
        data: [29, 71, 85, 42]
      }, {
        name: 'Product B',
        data: [40, 55, 60, 80]
      }]
    });
  </script>
</body>
</html>
```

7.2.4 Reusable Theme File

To keep things clean, you can place your theme in a separate file (e.g., `my-theme.js`):

```
(function(H) {
  H.setOptions({
    colors: ['#1abc9c', '#2ecc71', '#3498db', '#e67e22'],
    chart: {
      backgroundColor: '#ffffff',
      style: { fontFamily: 'Roboto, sans-serif' }
    },
    title: {
      style: { color: '#2c3e50' }
    }
    // ...more customizations
  });
}(Highcharts));
```

Then include it in your HTML or module build before creating charts:

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="my-theme.js"></script>
<script>
  Highcharts.chart('container', {
    // Theme will be applied automatically
    series: [{ data: [1, 2, 3, 4] }]
  });
</script>
```

7.2.5 Common Theme Options to Customize

Option	Description
`colors`	Array of series colors
`chart.backgroundColor`	Page/chart background color
`chart.style.fontFamily`	Global font for chart text
`title.style`	Title color, font size, etc.
`tooltip.style`	Font styling for tooltips
`legend.itemStyle`	Legend text style
`xAxis/yAxis.labels.style`	Axis label font and color

7.2.6 Tip: Resetting the Theme

If you need to create charts with different themes in the same session, you can reset options by storing and restoring the default settings, or isolate theme effects using different Highcharts instances in separate scopes.

7.2.7 Summary

- Use `Highcharts.setOptions()` to define a **custom global theme**
- Themes help maintain **design consistency** across all charts
- Encapsulate theme logic in a reusable `.js` file for clean organization
- Customize colors, fonts, backgrounds, and more

In the next section, we'll look at implementing **dark mode support** and applying **custom CSS styling** for seamless UI integration.

7.3 Dark Mode and Custom Styling

With dark mode becoming a standard feature in modern applications, it's important to ensure your charts look great against both light and dark backgrounds. Highcharts gives you full control over styling, making it easy to switch between light and dark modes through **themes** or **class-based toggling**.

In this section, you'll learn how to:

- Apply dark mode using a custom theme
- Dynamically toggle between light and dark styles
- Adjust specific visual elements like axes, grid lines, tooltips, and labels

7.3.1 Creating a Dark Mode Theme

To implement a dark theme, use `Highcharts.setOptions()` with colors and styles that complement a dark background. This includes updating text, grid lines, tooltips, and background colors.

7.3.2 Example: Basic Dark Theme

```
Highcharts.setOptions({
  chart: {
    backgroundColor: '#1f1f1f',
    style: {
      fontFamily: 'Segoe UI, sans-serif',
      color: '#FFFFFF'
    }
  },
  title: {
    style: {
      color: '#FFFFFF'
```

```
    }
  },
  xAxis: {
    gridLineColor: '#444444',
    labels: {
      style: { color: '#CCCCCC' }
    },
    lineColor: '#666666',
    tickColor: '#666666'
  },
  yAxis: {
    gridLineColor: '#444444',
    labels: {
      style: { color: '#CCCCCC' }
    },
    title: {
      style: { color: '#CCCCCC' }
    }
  },
  legend: {
    itemStyle: {
      color: '#CCCCCC'
    }
  },
  tooltip: {
    backgroundColor: '#2a2a2a',
    style: { color: '#FFFFFF' }
  },
  colors: ['#00d1b2', '#ff6b6b', '#ffc107', '#7f8c8d']
});
```

Apply this theme before creating any charts, and all visuals will inherit the dark styling.

Full runnable code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Highcharts Dark Theme Example</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      margin: 0;
      padding: 2em;
      background-color: #121212;
      color: #fff;
      font-family: Segoe UI, sans-serif;
    }
    #container {
      max-width: 800px;
      margin: 0 auto;
    }
  </style>
</head>
<body>
  <div id="container"></div>
```

```
<script>
  // Apply Dark Mode Theme
  Highcharts.setOptions({
    chart: {
      backgroundColor: '#1f1f1f',
      style: {
        fontFamily: 'Segoe UI, sans-serif',
        color: '#FFFFFF'
      }
    },
    title: {
      style: {
        color: '#FFFFFF'
      }
    },
    xAxis: {
      gridLineColor: '#444444',
      labels: {
        style: { color: '#CCCCCC' }
      },
      lineColor: '#666666',
      tickColor: '#666666'
    },
    yAxis: {
      gridLineColor: '#444444',
      labels: {
        style: { color: '#CCCCCC' }
      },
      title: {
        style: { color: '#CCCCCC' }
      }
    },
    legend: {
      itemStyle: {
        color: '#CCCCCC'
      }
    },
    tooltip: {
      backgroundColor: '#2a2a2a',
      style: { color: '#FFFFFF' }
    },
    colors: ['#00d1b2', '#ff6b6b', '#ffc107', '#7f8c8d']
  });

  // Create a sample chart
  Highcharts.chart('container', {
    chart: {
      type: 'line'
    },
    title: {
      text: 'Monthly Active Users'
    },
    xAxis: {
      categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May']
    },
    yAxis: {
      title: {
        text: 'Users (in thousands)'
```

```
      }
    },
    series: [{
      name: 'Website A',
      data: [29, 35, 40, 55, 65]
    }, {
      name: 'Website B',
      data: [20, 30, 45, 50, 70]
    }]
  });
</script>
</body>
</html>
```

7.3.3 Toggling Themes with JavaScript

If your application supports theme switching (light/dark toggle), you can switch chart styles dynamically by storing both themes and calling `Highcharts.setOptions()` followed by a redraw.

7.3.4 Example: Toggle Theme on Button Click

```
const darkTheme = {
  chart: { backgroundColor: '#1f1f1f' },
  title: { style: { color: '#fff' } },
  // ...rest of dark theme
};

const lightTheme = {
  chart: { backgroundColor: '#ffffff' },
  title: { style: { color: '#000' } },
  // ...rest of light theme
};

let currentTheme = 'light';

document.getElementById('toggleTheme').addEventListener('click', () => {
  currentTheme = currentTheme === 'light' ? 'dark' : 'light';
  Highcharts.setOptions(currentTheme === 'dark' ? darkTheme : lightTheme);
  chart.update({}, true); // redraw chart with new theme
});
```

> Use `chart.update({}, true)` to reapply the new theme without altering existing data.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Highcharts Theme Toggle</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      font-family: Segoe UI, sans-serif;
      background: #ffffff;
      color: #000000;
      padding: 2em;
      transition: background 0.3s, color 0.3s;
    }
    #container {
      max-width: 800px;
      margin: 1em auto;
    }
    button {
      padding: 0.6em 1em;
      font-size: 1em;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <button id="toggleTheme">Toggle Theme</button>
  <div id="container"></div>

  <script>
    const darkTheme = {
      chart: {
        backgroundColor: '#1f1f1f',
        style: { color: '#ffffff' }
      },
      title: {
        style: { color: '#ffffff' }
      },
      xAxis: {
        gridLineColor: '#444',
        labels: { style: { color: '#cccccc' } },
        lineColor: '#666',
        tickColor: '#666'
      },
      yAxis: {
        gridLineColor: '#444',
        labels: { style: { color: '#cccccc' } },
        title: { style: { color: '#cccccc' } }
      },
      legend: {
        itemStyle: { color: '#cccccc' }
      },
      tooltip: {
        backgroundColor: '#2a2a2a',
        style: { color: '#ffffff' }
      },
      colors: ['#00d1b2', '#ff6b6b', '#ffc107', '#7f8c8d']
    };
```

```
  const lightTheme = {
    chart: {
      backgroundColor: '#ffffff',
      style: { color: '#000000' }
    },
    title: {
      style: { color: '#000000' }
    },
    xAxis: {
      gridLineColor: '#e6e6e6',
      labels: { style: { color: '#333333' } },
      lineColor: '#999999',
      tickColor: '#999999'
    },
    yAxis: {
      gridLineColor: '#e6e6e6',
      labels: { style: { color: '#333333' } },
      title: { style: { color: '#333333' } }
    },
    legend: {
      itemStyle: { color: '#333333' }
    },
    tooltip: {
      backgroundColor: '#ffffff',
      style: { color: '#000000' }
    },
    colors: ['#007acc', '#e91e63', '#ffc107', '#4caf50']
  };

  let currentTheme = 'light';
  Highcharts.setOptions(lightTheme);

  const chart = Highcharts.chart('container', {
    chart: { type: 'column' },
    title: { text: 'Monthly Sales' },
    xAxis: { categories: ['Jan', 'Feb', 'Mar', 'Apr'] },
    yAxis: {
      title: { text: 'Sales (k USD)' }
    },
    series: [{
      name: 'Product A',
      data: [29, 71, 55, 90]
    }, {
      name: 'Product B',
      data: [45, 60, 80, 70]
    }]
  });

  document.getElementById('toggleTheme').addEventListener('click', () => {
    currentTheme = currentTheme === 'light' ? 'dark' : 'light';
    Highcharts.setOptions(currentTheme === 'dark' ? darkTheme : lightTheme);
    document.body.style.backgroundColor = currentTheme === 'dark' ? '#121212' : '#ffffff';
    document.body.style.color = currentTheme === 'dark' ? '#ffffff' : '#000000';

    // Re-create chart to apply new theme
    chart.update({}, true, false);
  });
</script>
```

```
</body>
</html>
```

7.3.5 Styling Tooltips and Grid Lines for Dark Backgrounds

Dark themes need special attention for high-contrast readability.

7.3.6 Key Adjustments:

Element	Recommended Color (Dark Mode)
Background	`#1f1f1f` or `#2a2a2a`
Text	`#FFFFFF` or light gray
Grid lines	`#444444` or semi-transparent
Axis lines	`#666666`

7.3.7 Example: Tooltip Customization

```
tooltip: {
  backgroundColor: '#2a2a2a',
  borderColor: '#888',
  style: {
    color: '#ffffff'
  }
}
```

7.3.8 Using CSS and Class Toggles

Although most Highcharts styling is handled via JavaScript configuration, you can integrate with your site's CSS class toggles (e.g., adding `dark-mode` to `<body>`) by defining separate Highcharts options and switching them accordingly.

7.3.9 Integration Tip:

```
if (document.body.classList.contains('dark-mode')) {
  Highcharts.setOptions(darkTheme);
} else {
  Highcharts.setOptions(lightTheme);
}
```

7.3.10 Summary

- Dark mode is implemented in Highcharts via `Highcharts.setOptions()` with custom color and style settings.
- You can dynamically toggle between light and dark themes using JavaScript.
- For best results, style the background, text, grid lines, axis labels, and tooltips to ensure contrast and readability.
- Combine with CSS class detection or media queries for seamless integration into your site's theme system.

Next, we'll dive into **mobile optimization tips** to ensure your charts are not only stylish—but also usable—on touch devices and small screens.

7.4 Mobile Optimization Tips

Designing charts for mobile devices requires special attention. Smaller screens, touch input, and limited interaction space all call for **clear, uncluttered, and responsive charts**. Fortunately, Highcharts provides several tools and settings to help optimize the mobile experience without sacrificing interactivity or clarity.

In this section, you'll learn practical strategies for:

- Improving touch usability
- Simplifying UI elements like legends and labels
- Enhancing tooltip behavior on touch devices
- Adapting charts dynamically for smaller screens

7.4.1 Increase Touch Target Sizes

Touch inputs are less precise than mouse clicks, so it's important to make interactive elements easy to tap. Highcharts allows you to enlarge clickable areas without affecting the visual size of the data point.

7.4.2 Example: Increase Marker Hit Area

```
plotOptions: {
  series: {
    marker: {
      enabled: true,
      radius: 4,
      states: {
        hover: {
          enabled: true,
          radiusPlus: 2
        }
      }
    },
    point: {
      events: {
        click: function () {
          alert('You clicked ' + this.category);
        }
      }
    }
  }
}
```

To further improve touchability, you can use the `turboThreshold` and `cropThreshold` settings to optimize performance for large datasets.

7.4.3 Simplify Legends for Small Screens

Legends can take up a lot of space on mobile. Consider:

- Using **vertical layout at the bottom**
- Hiding the legend entirely for simple charts
- Reducing the number of series or using short names

7.4.4 Example: Hide or Reposition Legend

```
responsive: {
  rules: [{
    condition: {
      maxWidth: 500
    },
    chartOptions: {
      legend: {
        layout: 'horizontal',
        align: 'center',
        verticalAlign: 'bottom',
        itemStyle: {
```

```
          fontSize: '10px'
        }
      }
    }
  }]
}
```

For charts with only one series, you can disable the legend entirely:

```
legend: { enabled: false }
```

7.4.5 Enable Sticky Tooltips for Touch

Tooltips that disappear on touch can be frustrating. Highcharts provides a way to **keep the tooltip visible** after tapping a point.

7.4.6 Example: Persistent Tooltips on Touch

```
tooltip: {
  stickOnContact: true,
  useHTML: true
}
```

This setting improves the experience on touch devices by letting users read values without holding their finger over the chart.

7.4.7 Minimize Axis Clutter

Tick labels and grid lines can become crowded on small screens. You can reduce clutter by:

- Using fewer ticks (`tickInterval`, `tickAmount`)
- Rotating or hiding labels
- Using abbreviated or formatted dates

7.4.8 Example: Reduce X-Axis Labels

```
xAxis: {
  tickInterval: 2, // Show every second label
  labels: {
    rotation: -45,
```

```
    style: {
      fontSize: '10px'
    }
  }
}
```

Also consider setting `max` and `min` values dynamically to focus on the most relevant data range.

7.4.9 Use Responsive Rules to Adapt Layout

The `responsive` configuration is your best friend on mobile. Use it to change chart structure and styling based on screen width.

7.4.10 Example: Adjust for Narrow Screens

```
responsive: {
  rules: [{
    condition: {
      maxWidth: 400
    },
    chartOptions: {
      chart: {
        spacingLeft: 10,
        spacingRight: 10
      },
      yAxis: {
        labels: {
          enabled: false
        }
      },
      tooltip: {
        shared: true
      }
    }
  }]
}
```

7.4.11 Summary: Mobile Optimization Checklist

Recommendation	Benefit
Increase marker/touch size	Easier tap interaction
Simplify/hide legends	Saves vertical and horizontal space

Recommendation	Benefit
Enable `stickOnContact` tooltip	Improves tooltip usability on touch
Reduce ticks and label rotation	Prevents visual crowding
Use `responsive.rules`	Dynamic layout adaptation

By applying these mobile-specific optimizations, your Highcharts visualizations will be just as **interactive and readable on smartphones** as they are on desktops. In the next chapter, we'll explore integrating Highcharts with real-world applications and frameworks.

Chapter 8.

Working with Data

1. Loading Data from External Sources (CSV, JSON, APIs)
2. Data Parsing and Preprocessing
3. Handling Large Datasets Efficiently
4. Data Grouping and Aggregation

8 Working with Data

8.1 Loading Data from External Sources (CSV, JSON, APIs)

Highcharts supports multiple ways to load external data, making it easy to visualize information stored outside your HTML or JavaScript. Whether you're working with a CSV file, a JSON dataset, or a live REST API, you can pull that data in and format it for use with your charts.

In this section, we'll cover:

- Loading data from a CSV file
- Using JSON objects (static or dynamic)
- Fetching data from a REST API using `fetch()` or `axios`
- Preprocessing the data before passing it to Highcharts

8.1.1 Loading Data from a CSV File

Highcharts includes a built-in `data` module that lets you load and parse CSV files directly.

8.1.2 Example: Using a CSV File

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/data.js"></script>

<div id="container"></div>

<script>
Highcharts.chart('container', {
  data: {
    csvURL: 'https://example.com/data.csv',
    enablePolling: true,
    dataRefreshRate: 5
  },
  title: {
    text: 'CSV Data Loaded into Highcharts'
  },
  yAxis: {
    title: {
      text: 'Value'
    }
  },
  xAxis: {
    title: {
      text: 'Category'
    }
  },
```

```
  chart: {
    type: 'line'
  }
});
</script>
```

Note: Your CSV should have a structure like:

```
Month,Sales
January,100
February,120
March,140
```

8.1.3 Loading Static JSON Data

JSON is one of the most common formats for data interchange. If you already have a JSON object in your code, you can pass it directly to the `series` configuration.

8.1.4 Example: JSON from JavaScript

```
const data = [29, 71, 106, 129, 144];

Highcharts.chart('container', {
  chart: {
    type: 'column'
  },
  title: {
    text: 'Static JSON Data'
  },
  xAxis: {
    categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May']
  },
  series: [{
    name: 'Revenue',
    data: data
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Highcharts with Static JSON Data</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
```

```
    body {
      font-family: Segoe UI, sans-serif;
      padding: 2em;
      background: #f9f9f9;
    }
    #container {
      max-width: 800px;
      margin: auto;
    }
  </style>
</head>
<body>

  <h2>Static JSON Data Example</h2>
  <div id="container"></div>

  <script>
    const data = [29, 71, 106, 129, 144]; // static JSON-like array

    Highcharts.chart('container', {
      chart: {
        type: 'column'
      },
      title: {
        text: 'Static JSON Data'
      },
      xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May']
      },
      yAxis: {
        title: {
          text: 'Revenue (in $1,000s)'
        }
      },
      series: [{
        name: 'Revenue',
        data: data
      }]
    });
  </script>

</body>
</html>
```

8.1.5 Fetching JSON from an API Using `fetch()`

To load data from a RESTful API, use JavaScript's native `fetch()` function and convert the response into the format Highcharts expects.

8.1.6 Example: Using `fetch()`

```
fetch('https://api.example.com/sales')
  .then(response => response.json())
  .then(json => {
    const categories = json.map(item => item.month);
    const values = json.map(item => item.value);

    Highcharts.chart('container', {
      chart: {
        type: 'line'
      },
      title: {
        text: 'Sales Over Time'
      },
      xAxis: {
        categories: categories
      },
      series: [{
        name: 'Sales',
        data: values
      }]
    });
  })
  .catch(error => console.error('Error loading data:', error));
```

Assumes the API returns something like:

```
[
  { "month": "Jan", "value": 100 },
  { "month": "Feb", "value": 150 },
  { "month": "Mar", "value": 200 }
]
```

8.1.7 Using Axios for Data Fetching

If you're using a framework like React or Vue, you may prefer `axios` for its convenience and broader browser support.

8.1.8 Example: Axios Integration

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

<script>
axios.get('https://api.example.com/data')
  .then(response => {
    const rawData = response.data;

    const categories = rawData.map(row => row.label);
```

```
    const values = rawData.map(row => row.amount);

    Highcharts.chart('container', {
      chart: { type: 'bar' },
      title: { text: 'Data via Axios' },
      xAxis: { categories },
      series: [{ name: 'Amount', data: values }]
    });
  })
  .catch(err => console.error('Axios load failed:', err));
</script>
```

8.1.9 Preprocessing Tips

Often, the raw data you fetch isn't quite ready for Highcharts. Here are a few common tasks:

Task	Example
Convert strings to numbers	`Number(row.value)`
Format dates	`new Date(row.timestamp)`
Filter or sort data	`data.filter(...).sort(...)`
Map to `[x, y]` pairs	`data.map(row => [row.x, row.y])`

8.1.10 Example: Mapping `[x, y]` Data

```
const seriesData = apiData.map(row => [Date.parse(row.date), row.value]);

Highcharts.chart('container', {
  chart: { type: 'line' },
  xAxis: { type: 'datetime' },
  series: [{ name: 'Trend', data: seriesData }]
});
```

8.1.11 Summary

- Highcharts can load data from **CSV, JSON, or APIs** using built-in tools or JavaScript functions.
- Use the `data` module for simple CSV loading.
- Fetch JSON data using `fetch()` or `axios`, and preprocess it to match Highcharts' format.
- Clean and map your data as needed before passing it into `series` or `categories`.

In the next section, you'll learn how to **parse and preprocess data** more effectively, especially when working with complex or messy datasets.

8.2 Data Parsing and Preprocessing

Raw data from external sources often needs some transformation before it fits neatly into Highcharts' expected structure. Proper parsing and preprocessing ensure your charts render correctly and efficiently.

This section covers common techniques such as:

- Converting timestamp strings to JavaScript `Date` objects or numeric timestamps
- Flattening nested JSON objects into arrays suitable for series data
- Filtering, sorting, and formatting data points

8.2.1 Why Preprocessing Matters

Highcharts expects data in particular formats, typically:

- Arrays of numeric values for simple series
- Arrays of `[x, y]` pairs where `x` is a number or timestamp and `y` is numeric
- Arrays of objects with keys like `x`, `y`, `name`, or `color`

If your data doesn't match this structure, the chart might not render properly, or the interactivity (like tooltips and zooming) might behave unexpectedly.

8.2.2 Converting Timestamp Strings

APIs often deliver dates and times as strings (e.g., `"2025-07-15T14:30:00Z"`). Highcharts requires numeric timestamps for datetime axes, so convert them using `Date.parse()` or the `Date` constructor.

8.2.3 Example: Parsing ISO Date Strings

```
const rawData = [
  { timestamp: "2025-07-15T14:30:00Z", value: 42 },
  { timestamp: "2025-07-16T14:30:00Z", value: 55 },
  { timestamp: "2025-07-17T14:30:00Z", value: 48 }
```

```
];

// Convert to [timestamp, value] pairs
const seriesData = rawData.map(item => [
  Date.parse(item.timestamp), // x value as timestamp
  item.value                  // y value
]);

Highcharts.chart('container', {
  xAxis: { type: 'datetime' },
  series: [{ data: seriesData }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Highcharts ISO Date Parsing</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      font-family: Segoe UI, sans-serif;
      padding: 2em;
      background: #f9f9f9;
    }
    #container {
      max-width: 800px;
      margin: auto;
    }
  </style>
</head>
<body>

  <h2>Time Series from ISO Dates</h2>
  <div id="container"></div>

  <script>
    const rawData = [
      { timestamp: "2025-07-15T14:30:00Z", value: 42 },
      { timestamp: "2025-07-16T14:30:00Z", value: 55 },
      { timestamp: "2025-07-17T14:30:00Z", value: 48 }
    ];

    const seriesData = rawData.map(item => [
      Date.parse(item.timestamp), // convert ISO to timestamp
      item.value
    ]);

    Highcharts.chart('container', {
      chart: { type: 'line' },
      title: { text: 'Parsed ISO Date Data' },
      xAxis: {
        type: 'datetime',
        title: { text: 'Date' }
      },
```

```
      yAxis: {
        title: { text: 'Value' }
      },
      tooltip: {
        xDateFormat: '%Y-%m-%d %H:%M:%S',
        shared: true
      },
      series: [{
        name: 'Sample Data',
        data: seriesData
      }]
    });
  </script>

</body>
</html>
```

8.2.4 Flattening Nested JSON Responses

Sometimes JSON data is nested within multiple layers or arrays. You need to extract and flatten this data before use.

8.2.5 Example: Flattening Nested JSON

```
const nestedData = {
  region: 'North',
  sales: [
    { month: 'Jan', amount: 100 },
    { month: 'Feb', amount: 120 },
    { month: 'Mar', amount: 140 }
  ]
};

// Extract sales array for Highcharts
const categories = nestedData.sales.map(item => item.month);
const data = nestedData.sales.map(item => item.amount);

Highcharts.chart('container', {
  xAxis: { categories },
  series: [{ data }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
```

```
  <title>Highcharts Nested JSON Flattening</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      font-family: Segoe UI, sans-serif;
      padding: 2em;
      background-color: #f0f0f0;
    }
    #container {
      width: 800px;
      height: 400px;
      margin: auto;
    }
  </style>
</head>
<body>

  <h2>Sales by Month (Flattened from Nested JSON)</h2>
  <div id="container"></div>

  <script>
    // Nested JSON structure
    const nestedData = {
      region: 'North',
      sales: [
        { month: 'Jan', amount: 100 },
        { month: 'Feb', amount: 120 },
        { month: 'Mar', amount: 140 }
      ]
    };

    // Flatten the structure for Highcharts
    const categories = nestedData.sales.map(item => item.month);
    const data = nestedData.sales.map(item => item.amount);

    // Render the chart
    Highcharts.chart('container', {
      chart: { type: 'column' },
      title: { text: `Monthly Sales - ${nestedData.region} Region` },
      xAxis: {
        categories: categories,
        title: { text: 'Month' }
      },
      yAxis: {
        title: { text: 'Sales Amount' }
      },
      series: [{
        name: 'Sales',
        data: data
      }]
    });
  </script>

</body>
</html>
```

8.2.6 Filtering and Sorting Data

Before visualizing, it's often useful to:

- Remove irrelevant or incomplete entries
- Sort data chronologically or by value

8.2.7 Example: Filtering Invalid Values

```
const rawData = [
  { x: 1, y: 10 },
  { x: 2, y: null },
  { x: 3, y: 30 }
];

const filteredData = rawData.filter(point => point.y !== null);

Highcharts.chart('container', {
  series: [{ data: filteredData }]
});
```

8.2.8 Formatting Data Labels and Values

Sometimes you need to convert raw numeric values to a more readable format before displaying.

8.2.9 Example: Formatting Percentages

```
tooltip: {
  pointFormatter: function () {
    return `<b>${(this.y * 100).toFixed(1)}%</b>`;
  }
}
```

8.2.10 Summary

- Preprocessing aligns raw data with Highcharts' expected formats.
- Convert date strings to timestamps for datetime axes.
- Flatten nested JSON structures into arrays for categories and series.
- Filter and sort data to improve chart clarity.

- Format data labels and tooltips for better readability.

Mastering these preprocessing techniques will make your charts more reliable and visually consistent. In the next section, we will discuss **handling large datasets efficiently** to maintain performance.

8.3 Handling Large Datasets Efficiently

When working with thousands—or even tens of thousands—of data points, rendering performance and responsiveness become critical. Highcharts offers several strategies and configuration options to ensure your charts remain fast and smooth, even with large datasets.

This section covers:

- Turbo mode for optimized parsing
- Disabling animations to speed up rendering
- Reducing data density and aggregation
- Using the Boost module for GPU acceleration

8.3.1 Turbo Mode: Fast Parsing for Large Data

Highcharts has a built-in **turbo mode** that dramatically speeds up rendering when you supply large arrays of simple numeric data points.

- Enabled by default when series data exceeds 1,000 points
- Works by limiting data formats to simple arrays (e.g., `[y1, y2, y3, ...]` or `[x, y]` pairs)
- Avoids overhead of complex point objects for performance gain

8.3.2 How to Use Turbo Mode

Simply pass large datasets as arrays of numbers or pairs. Turbo mode activates automatically.

```
series: [{
  data: [29, 71, 106, 129, 144, /* ... thousands more ... */]
}]
```

If your data is too complex or uses objects, you can enable turbo mode explicitly or simplify your data.

8.3.3 Disable Animation for Faster Rendering

Animations add polish but cost performance, especially for big data.

- Turn off animations during initial rendering or updates
- Improves speed and responsiveness

8.3.4 Example: Disable Animation

```
chart: {
  animation: false
}
```

You can also disable animation for specific series:

```
series: [{
  animation: false,
  data: largeDataset
}]
```

8.3.5 Limit Data Density

Plotting every single data point isn't always necessary or helpful:

- Consider aggregating or sampling data before plotting
- Use server-side preprocessing to reduce dataset size
- Aggregate by time intervals (e.g., hourly averages instead of raw minutes)

This approach reduces rendering load and improves clarity.

8.3.6 Using the Boost Module for GPU Acceleration

Highcharts includes a **Boost module** that leverages WebGL for hardware-accelerated rendering of large datasets.

8.3.7 Key Features:

- Supports hundreds of thousands of points smoothly
- Automatically switches to GPU rendering when thresholds are crossed
- Works best with simple series types like line, scatter, and area

8.3.8 How to Enable the Boost Module

Include the Boost module script:

```
<script src="https://code.highcharts.com/modules/boost.js"></script>
```

Then, enable boost in your chart:

```
boost: {
  useGPUTranslations: true,
  usePreAllocated: true
},
series: [{
  data: largeDataset,
  boostThreshold: 5000 // Enable boost above 5000 points
}]
```

8.3.9 Summary of Best Practices

Strategy	Effect
Turbo mode	Fast parsing with simple arrays
Disable animations	Speed up rendering
Reduce data density	Avoid plotting unnecessary points
Boost module	Use GPU acceleration for huge datasets

By applying these techniques, you can maintain high performance and a smooth user experience, even when visualizing very large datasets with Highcharts.

In the next section, we will explore **data grouping and aggregation** to further enhance clarity and efficiency.

8.4 Data Grouping and Aggregation

When working with dense time series data—such as daily stock prices or minute-by-minute sensor readings—displaying every single data point can overwhelm both the chart and the viewer. Data grouping and aggregation help simplify these datasets by summarizing data points over larger intervals like weeks or months, improving readability and performance.

Highcharts provides built-in support for data grouping, especially useful for time series charts.

8.4.1 What Is Data Grouping?

Data grouping consolidates multiple data points within a time interval into a single aggregated value, such as an average, sum, or range. Instead of showing every raw data point, the chart shows a summary for each group, making trends easier to spot and reducing clutter.

8.4.2 Enabling Data Grouping in Highcharts

The `dataGrouping` option is available on series, primarily for series types like `line`, `area`, and `column`. It lets you specify the grouping interval and the aggregation method.

8.4.3 Example: Aggregating Daily Data into Weekly Averages

```
Highcharts.chart('container', {
  chart: { type: 'line' },
  title: { text: 'Daily Sales Aggregated Weekly' },
  xAxis: { type: 'datetime' },
  series: [{
    name: 'Daily Sales',
    data: [
      // [timestamp, value]
      [Date.UTC(2025, 0, 1), 120],
      [Date.UTC(2025, 0, 2), 135],
      // ... daily points for several months
    ],
    dataGrouping: {
      enabled: true,
      approximation: 'average',  // Aggregate by average
      forced: true,              // Force grouping regardless of zoom
      units: [['week', [1]]]     // Group data points by week
    }
  }]
});
```

In this example:

- `approximation: 'average'` means grouped points represent the average of the included values.
- `units: [['week', [1]]]` specifies grouping intervals of 1 week.
- `forced: true` ensures grouping applies even when zoomed in.

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
```

```
  <title>Highcharts Weekly Aggregation</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <script src="https://code.highcharts.com/modules/data.js"></script>
  <script src="https://code.highcharts.com/modules/exporting.js"></script>
  <script src="https://code.highcharts.com/modules/export-data.js"></script>
  <script src="https://code.highcharts.com/modules/accessibility.js"></script>
  <style>
    body {
      font-family: Segoe UI, sans-serif;
      padding: 2em;
      background: #f0f0f0;
    }
    #container {
      max-width: 900px;
      margin: auto;
      height: 500px;
    }
  </style>
</head>
<body>

  <h2>Daily Sales Aggregated into Weekly Averages</h2>
  <div id="container"></div>

  <script>
    // Generate fake daily data for 60 days
    const data = [];
    const start = Date.UTC(2025, 0, 1); // Jan 1, 2025
    for (let i = 0; i < 60; i++) {
      const timestamp = start + i * 24 * 3600 * 1000;
      const value = Math.round(100 + Math.random() * 50); // 100-150
      data.push([timestamp, value]);
    }

    // Enable data grouping with Highstock compatibility
    Highcharts.chart('container', {
      chart: { type: 'line', zoomType: 'x' },
      title: { text: 'Daily Sales Aggregated Weekly' },
      xAxis: { type: 'datetime' },
      yAxis: {
        title: { text: 'Sales' }
      },
      tooltip: {
        xDateFormat: '%A, %b %e, %Y',
        shared: true
      },
      plotOptions: {
        series: {
          dataGrouping: {
            enabled: true,
            approximation: 'average',
            forced: true,
            units: [['week', [1]]]
          }
        }
      },
      series: [{
        name: 'Daily Sales',
```

```
        data: data
      }]
    });
  </script>

</body>
</html>
```

8.4.4 Supported Aggregation Methods

Common aggregation options for `approximation` include:

Method	Description
`average`	Mean value of points in the group
`sum`	Total sum of grouped points
`open`	First value in the group
`high`	Maximum value in the group
`low`	Minimum value in the group
`close`	Last value in the group

8.4.5 When to Use Aggregation

- **Long time spans:** Aggregating daily data into weekly or monthly summaries helps reveal overall trends without getting lost in noise.
- **Performance:** Fewer points means faster rendering and smoother interaction.
- **User experience:** Simplifies charts for non-technical audiences or mobile screens.

8.4.6 Combining Multiple Grouping Units

You can define multiple units for dynamic grouping based on zoom level:

```
dataGrouping: {
  units: [
    ['month', [1, 3, 6]], // Group by 1, 3, or 6 months
    ['week', [1, 2]],     // Or by 1 or 2 weeks
    ['day', [1]]          // Or daily
  ]
}
```

Highcharts will automatically adjust grouping depending on how much of the dataset is visible.

8.4.7 Summary

- Data grouping consolidates many points into fewer, aggregated values.
- Use the `dataGrouping` option to enable grouping with custom intervals and aggregation methods.
- Grouping improves readability and performance, especially for dense time series.
- Multiple grouping units allow smooth zoom-based transitions between granularities.

Mastering data grouping lets you build cleaner, more performant time series charts that communicate insights effectively, even with large datasets.

Chapter 9.

Extending Highcharts

1. Custom Series and Renderer API
2. Plugins and Extensions
3. Creating Custom Export Formats

9 Extending Highcharts

9.1 Custom Series and Renderer API

10 Custom Series and Renderer API

Highcharts is designed to be highly extensible, allowing you to create **custom series types** and leverage the powerful **renderer API** to draw bespoke SVG elements. This capability is invaluable when built-in chart types don't fully meet your needs or when you want to visualize data in unique ways.

In this section, we'll explore how to create custom series using the `Highcharts.seriesType()` method and use the `renderer` API for manual SVG drawing. We'll walk through a practical example—building a simple thermometer chart—and explain how it integrates into the existing Highcharts system.

10.0.1 Creating Custom Series with `seriesType()`

The `Highcharts.seriesType()` function enables you to define a new series type by extending an existing one, inheriting its properties and behaviors, and overriding or adding features.

10.0.2 Basic Syntax

```
Highcharts.seriesType(
  'customType',          // new series type name
  'baseType',            // existing series to extend, e.g., 'line'
  { /* default options */ },
  { /* prototype members - methods and properties */ },
  { /* optional: custom point members */ }
);
```

10.0.3 Why Use Custom Series?

- Implement new drawing logic.
- Add or override animation or interaction.
- Customize tooltip or legend behavior.
- Support unique data formats or visual metaphors.

10.0.4 The Renderer API: Drawing Custom SVG Elements

The `renderer` is a low-level API that lets you create and manipulate SVG elements like paths, circles, rectangles, text, and groups directly within the chart container.

Access the renderer via:

```
chart.renderer
```

10.0.5 Common Renderer Methods

- `rect(x, y, width, height, borderRadius)`: Draw a rectangle.
- `circle(x, y, radius)`: Draw a circle.
- `path(arrayOfCommands)`: Draw an SVG path.
- `text(string, x, y)`: Add text labels.
- `.attr()`: Set attributes like fill, stroke, opacity.
- `.add()`: Append to the SVG root or group.

10.0.6 Example: Building a Thermometer Chart

A thermometer chart visually represents a single value relative to a scale—ideal for dashboards or KPI indicators.

10.0.7 Step 1: Define a Custom Series Type

We'll extend the `column` series for simple bar behavior but override the drawing to create a vertical thermometer.

```
Highcharts.seriesType(
  'thermometer',
  'column',
  {
    // default options specific to thermometer
    color: '#FF4136',       // red fill
    borderColor: '#990000',
    borderWidth: 2,
    dataLabels: { enabled: true }
  },
  {
    // Override drawPoints method to customize drawing
    drawPoints: function () {
      const series = this;
      const chart = series.chart;
      series.points.forEach(point => {
```

```
        if (!point.graphic) {
          // Draw custom thermometer shape
          point.graphic = chart.renderer.g('thermometer-group').add(series.group);

          // Thermometer tube (rounded rectangle)
          point.tube = chart.renderer.rect(
            point.plotX + series.xAxis.left - 10,  // x
            series.yAxis.top,                       // y (top of axis)
            20,                                    // width
            series.yAxis.len,                       // height
            10                                     // borderRadius
          ).attr({
            fill: '#eee',
            stroke: '#999',
            'stroke-width': 2
          }).add(point.graphic);

          // Mercury level (height based on value)
          const mercuryHeight = series.yAxis.toPixels(series.yAxis.min) - series.yAxis.toPixels(point.y
          point.mercury = chart.renderer.rect(
            point.plotX + series.xAxis.left - 8,
            series.yAxis.top + series.yAxis.len - mercuryHeight,
            16,
            mercuryHeight,
            8
          ).attr({
            fill: series.options.color
          }).add(point.graphic);

          // Data label
          if (series.options.dataLabels.enabled) {
            point.dataLabel = chart.renderer.text(
              point.y + '°C',
              point.plotX + series.xAxis.left + 15,
              series.yAxis.top + series.yAxis.len - mercuryHeight - 5
            ).css({
              color: '#333',
              fontWeight: 'bold'
            }).add(point.graphic);
          }
        } else {
          // Update mercury height on redraw
          const mercuryHeight = series.yAxis.toPixels(series.yAxis.min) - series.yAxis.toPixels(point.y
          point.mercury.attr({
            y: series.yAxis.top + series.yAxis.len - mercuryHeight,
            height: mercuryHeight
          });
          point.dataLabel.attr({
            y: series.yAxis.top + series.yAxis.len - mercuryHeight - 5,
            text: point.y + '°C'
          });
        }
      });
    }
  }
);
```

10.0.8 Step 2: Use Your Custom Series in a Chart

```
Highcharts.chart('container', {
  chart: { type: 'thermometer' },
  title: { text: 'Thermometer Chart Example' },
  yAxis: {
    min: 0,
    max: 100,
    title: { text: 'Temperature' }
  },
  series: [{
    name: 'Temperature',
    data: [65]
  }]
});
```

Full runnable code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Thermometer Chart</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      font-family: Segoe UI, sans-serif;
      background: #f5f5f5;
      padding: 2em;
    }
    #container {
      width: 600px;
      height: 500px;
      margin: auto;
    }
  </style>
</head>
<body>

  <h2 style="text-align: center;">Custom Thermometer Chart</h2>
  <div id="container"></div>

  <script>
    // Step 1: Define custom series type "thermometer"
    Highcharts.seriesType(
      'thermometer',
      'column',
      {
        color: '#FF4136',
        borderColor: '#990000',
        borderWidth: 2,
        dataLabels: { enabled: true }
      },
      {
        drawPoints: function () {
          const series = this;
          const chart = series.chart;
```

```
        series.points.forEach(point => {
          const x = point.plotX + series.xAxis.left;
          const axisTop = series.yAxis.top;
          const axisLen = series.yAxis.len;
          const mercuryHeight = series.yAxis.toPixels(series.yAxis.min) - series.yAxis.toPixels(point

          if (!point.graphic) {
            point.graphic = chart.renderer.g('thermometer-group').add(series.group);

            point.tube = chart.renderer.rect(
              x - 10, axisTop, 20, axisLen, 10
            ).attr({
              fill: '#eee',
              stroke: '#999',
              'stroke-width': 2
            }).add(point.graphic);

            point.mercury = chart.renderer.rect(
              x - 8, axisTop + axisLen - mercuryHeight, 16, mercuryHeight, 8
            ).attr({
              fill: series.options.color
            }).add(point.graphic);

            if (series.options.dataLabels.enabled) {
              point.dataLabel = chart.renderer.text(
                point.y + '°C', x + 15, axisTop + axisLen - mercuryHeight - 5
              ).css({
                color: '#333',
                fontWeight: 'bold'
              }).add(point.graphic);
            }
          } else {
            point.mercury.attr({
              y: axisTop + axisLen - mercuryHeight,
              height: mercuryHeight
            });
            point.dataLabel.attr({
              y: axisTop + axisLen - mercuryHeight - 5,
              text: point.y + '°C'
            });
          }
        });
      }
    }
  );

  // Step 2: Use the custom thermometer series
  Highcharts.chart('container', {
    chart: { type: 'thermometer' },
    title: { text: 'Thermometer Chart Example' },
    yAxis: {
      min: 0,
      max: 100,
      title: { text: 'Temperature (°C)' }
    },
    xAxis: {
      categories: ['Sensor 1'],
      visible: false
```

```
      },
      series: [{
        name: 'Temperature',
        data: [65]
      }]
    });
  </script>

</body>
</html>
```

10.0.9 How It Ties Into the Highcharts System

- The custom series extends an existing series type to reuse core functionality.
- The `drawPoints()` method is overridden to create bespoke SVG elements instead of default bars.
- The renderer places the thermometer tube, mercury level, and data label inside the chart's SVG container.
- The series still supports Highcharts features like axes, tooltips, and exporting seamlessly.

10.0.10 Additional Uses of the Renderer API

- Draw **sparklines** as tiny inline charts by combining lines, circles, and text.
- Create **custom annotations** or shapes overlaid on charts.
- Build **interactive controls** or icons embedded in charts.

10.0.11 Summary

- Use `Highcharts.seriesType()` to define new series types by extending existing ones.
- Override methods such as `drawPoints` to customize rendering logic.
- Leverage the `renderer` API to manually draw SVG shapes, text, and groups.
- Custom series integrate fully with Highcharts features like axes, tooltips, and animation.
- Practical examples like thermometer or sparkline charts demonstrate how to visualize data uniquely.

Mastering custom series and the renderer unlocks powerful possibilities to tailor Highcharts exactly to your visualization needs.

10.1 Plugins and Extensions

Highcharts' modular architecture allows you to **extend its functionality** beyond the core features by writing or using **plugins and extensions**. Plugins can add new behaviors such as annotations, custom legends, export enhancements, or even new chart interaction modes.

This section explores how to create and integrate plugins, highlights common use cases, and provides best practices for organizing and safely injecting plugin code into your Highcharts projects.

10.1.1 What Are Highcharts Plugins?

Plugins are self-contained pieces of code that augment Highcharts by:

- Adding new chart components (e.g., custom legends, annotations).
- Enhancing existing features (e.g., advanced export options).
- Introducing new APIs or utility functions.
- Modifying or extending the chart lifecycle or event handling.

They typically hook into Highcharts by extending prototypes, wrapping functions, or listening to chart events.

10.1.2 Using Existing Plugins

Highcharts offers several official and community plugins, such as:

- **Annotations module:** Adds rich annotation support (shapes, labels, connectors).
- **Export-data module:** Enhances exporting capabilities to include CSV, XLS, or raw data.
- **Accessibility module:** Improves screen reader support and keyboard navigation.
- **Boost module:** Optimizes performance for large datasets.

These can be included via script tags, NPM packages, or ES modules:

```
<script src="https://code.highcharts.com/modules/annotations.js"></script>
```

Or

```
npm install highcharts
```

Then import and initialize:

```
import Highcharts from 'highcharts';
import AnnotationsModule from 'highcharts/modules/annotations';

AnnotationsModule(Highcharts);
```

10.1.3 Writing Your Own Plugin

To write a custom plugin, follow these steps:

10.1.4 Encapsulate Your Code

Wrap your plugin in a function to avoid polluting the global namespace:

```
(function(H) {
  // Plugin code here, using H as the Highcharts reference

  // Example: Add a method to all chart instances
  H.Chart.prototype.myCustomMethod = function() {
    console.log('Custom method on chart:', this);
  };

  // Or hook into existing events
  H.addEvent(H.Chart, 'load', function() {
    console.log('Chart loaded:', this);
  });

})(Highcharts);
```

10.1.5 Extend Prototypes or Use Events

You can extend core classes like `Chart`, `Series`, or `Point` by adding methods or overriding behavior. Use Highcharts' `addEvent` utility to hook into lifecycle events safely.

10.1.6 Register New Components or Options

If you're adding UI elements (e.g., custom legend), create new SVG elements using the renderer and add them to the chart's container.

10.1.7 Example: Simple Annotation Plugin

```
(function(H) {
  H.Chart.prototype.addSimpleAnnotation = function(text, x, y) {
    const renderer = this.renderer;

    this.simpleAnnotation = renderer.text(text, x, y)
      .css({ color: 'red', fontWeight: 'bold' })
```

```
      .add();
  };

  H.addEvent(H.Chart, 'load', function() {
    if (this.options.customAnnotation) {
      this.addSimpleAnnotation(
        this.options.customAnnotation.text,
        this.options.customAnnotation.x,
        this.options.customAnnotation.y
      );
    }
  });
})(Highcharts);
```

Usage:

```
Highcharts.chart('container', {
  // chart options
  customAnnotation: { text: 'Hello!', x: 100, y: 50 }
});
```

10.1.8 Organizing Plugin Code

- **Modularize:** Keep plugin code separate from app logic.
- **Namespace carefully:** Use unique names to avoid conflicts.
- **Document your API:** Clearly describe plugin options and methods.
- **Ensure backward compatibility:** Avoid overriding core methods unless necessary.
- **Test with various Highcharts versions** if supporting multiple.

10.1.9 Injecting Plugins Safely

- Load plugin scripts **after** the core Highcharts library.
- Initialize modules or plugins by passing the Highcharts object.
- When using bundlers, import plugins explicitly and invoke them with Highcharts.
- Avoid global side effects by encapsulating plugin logic.
- Use Highcharts' event system to integrate without modifying core source.

10.1.10 Summary

- Highcharts plugins extend or enhance core capabilities like annotations, export, or interactivity.
- Use official modules or write your own by extending prototypes and hooking into events.

- Encapsulate code and namespace to avoid conflicts.
- Organize plugins separately and initialize them after loading Highcharts.
- Plugins allow you to tailor Highcharts to complex, custom visualization needs safely and cleanly.

Mastering plugins and extensions empowers you to build sophisticated, customized charts beyond the out-of-the-box experience.

10.2 Creating Custom Export Formats

Highcharts comes with built-in export options for common image formats like PNG, PDF, and SVG. However, many applications require exporting data or visualizations in other formats—such as CSV, Excel spreadsheets, or even custom-styled HTML reports. This section guides you through extending Highcharts' exporting capabilities by adding **custom export formats**, converting chart data into different outputs, and hooking into export events to tailor the process.

10.2.1 Extending Export Options

The Highcharts exporting module exposes an API to customize and add new export formats. You can define additional menu items that perform custom export logic, allowing users to download chart data or visuals in your desired formats.

10.2.2 How Exporting Works

When the user clicks an export button, Highcharts triggers an event and runs the corresponding export function. By adding new menu items with custom callbacks, you can implement your own export behaviors.

10.2.3 Example: Adding CSV Export to Export Menu

While Highcharts has an export-data module for CSV export, here's how you can add a custom CSV export menu item manually, giving you full control over the export format:

```
Highcharts.chart('container', {
  exporting: {
    buttons: {
      contextButton: {
```

```
        menuItems: [
          'printChart',
          'separator',
          'downloadPNG',
          'downloadPDF',
          {
            text: 'Download CSV',
            onclick: function () {
              const chart = this;
              let csv = 'Category,Value\n';
              chart.series[0].data.forEach(point => {
                csv += `${point.category || point.x},${point.y}\n`;
              });
              const blob = new Blob([csv], { type: 'text/csv;charset=utf-8;' });
              const url = URL.createObjectURL(blob);

              const a = document.createElement('a');
              a.href = url;
              a.download = 'chart-data.csv';
              document.body.appendChild(a);
              a.click();
              document.body.removeChild(a);
              URL.revokeObjectURL(url);
            }
          }
        ]
      }
    }
  },
  // chart options...
});
```

10.2.4 Exporting to Excel

To export chart data as an Excel `.xlsx` file, you can combine Highcharts data retrieval with a JavaScript library like SheetJS (xlsx). The process involves:

1. Extracting data from the chart's series.
2. Formatting the data as worksheets.
3. Using SheetJS to generate and trigger the Excel file download.

This method lets you produce richly formatted Excel reports directly from your chart data.

10.2.5 Exporting Custom-Styled HTML

Sometimes you want to export the chart and additional context as a full HTML report, including custom styles and annotations.

You can hook into the `exporting` process by:

- Capturing chart SVG and surrounding HTML.
- Combining it with CSS styles.
- Triggering a download of a `.html` file.

Example outline:

```
exporting: {
  buttons: {
    contextButton: {
      menuItems: [
        {
          text: 'Download HTML Report',
          onclick: function () {
            const chart = this;
            const svg = chart.getSVG();
            const htmlContent = `
              <html>
                <head>
                  <style>
                    body { font-family: Arial, sans-serif; }
                    .chart-container { max-width: 800px; margin: auto; }
                  </style>
                </head>
                <body>
                  <h1>${chart.title.textStr}</h1>
                  <div class="chart-container">${svg}</div>
                  <p>Generated on: ${new Date().toLocaleString()}</p>
                </body>
              </html>
            `;
            const blob = new Blob([htmlContent], { type: 'text/html' });
            const url = URL.createObjectURL(blob);
            const a = document.createElement('a');
            a.href = url;
            a.download = 'chart-report.html';
            document.body.appendChild(a);
            a.click();
            document.body.removeChild(a);
            URL.revokeObjectURL(url);
          }
        }
      ]
    }
  }
}
```

10.2.6 Hooking Into Export Events

Highcharts fires events during exporting to customize or preprocess data:

- `exporting.exportChart`
- `exporting.beforeExport`

- `exporting.afterExport`

Use these events to modify the chart or data before export:

```
Highcharts.addEvent(Highcharts.Chart, 'beforeExport', function () {
  console.log('Preparing chart for export:', this);
  // Modify chart options or data here if needed
});
```

10.2.7 Summary

- Extend Highcharts export menu by adding custom items with your export logic.
- Export chart data to CSV or Excel by extracting series data and formatting appropriately.
- Generate custom HTML reports combining SVG and styled content.
- Use export-related events to intercept and customize export behavior.
- Combine Highcharts' API with external libraries (like SheetJS) for rich export formats.

By creating custom export formats, you empower your users to get meaningful outputs from your charts tailored to their workflows and tools.

Chapter 10.

Troubleshooting and Best Practices

1. Debugging Common Issues
2. Performance Optimization Tips
3. Security Considerations
4. Accessibility Compliance

11 Troubleshooting and Best Practices

11.1 Debugging Common Issues

Even though Highcharts is designed for ease of use, you may occasionally encounter problems such as charts not rendering, incorrect data display, or misaligned tooltips. This section outlines common issues, how to diagnose them, and practical tips for troubleshooting effectively.

11.1.1 Common Issues and How to Diagnose Them

Chart Does Not Render or Shows a Blank Container

Possible causes:

- Missing or incorrect container ID in the `Highcharts.chart()` call.
- The container `<div>` has zero width or height.
- Highcharts library or modules not properly loaded.
- JavaScript errors preventing execution.

How to diagnose:

- Check the browser's Developer Console (F12) for JavaScript errors.
- Verify that the container exists and has a size:

```
const container = document.getElementById('container');
console.log(container.offsetWidth, container.offsetHeight);
```

- Confirm the Highcharts scripts are correctly loaded by inspecting network requests.
- Use minimal code to isolate the issue, e.g., try rendering a simple line chart with hardcoded data.

Data Not Displayed or Incorrectly Plotted

Possible causes:

- Malformed data format (e.g., wrong structure or missing `x`/`y` values).
- Misaligned axis types (e.g., using categories on a numeric axis).
- Series data not matching axis configuration.

How to diagnose:

- Log your data before passing it to Highcharts to ensure the structure is correct.
- Use the Highcharts debug flag:

```
Highcharts.setOptions({ debug: true });
```

- Check the `series` and `axes` config for consistency.
- Temporarily replace dynamic data with static sample data to check if the problem persists.

Tooltip or Legend Misalignment

Possible causes:

- CSS affecting tooltip or legend container positioning.
- Chart container resizing without calling `chart.reflow()`.
- Custom formatter functions returning incorrect HTML or values.

How to diagnose:

- Inspect tooltip elements in browser DevTools to check positioning and styles.
- Call `chart.reflow()` after container size changes.
- Temporarily disable custom tooltip formatters to see if alignment improves.

Slow Performance or Laggy Interactions

Possible causes:

- Large datasets without using boost or data grouping.
- Animations enabled on complex or frequent updates.
- Excessive DOM or SVG elements.

How to diagnose:

- Use browser performance profiling tools.
- Test disabling animation and see if performance improves:

```
plotOptions: {
  series: {
    animation: false
  }
}
```

- Enable the Boost module for large datasets.

11.1.2 Tips for Effective Troubleshooting

Use Minimal Test Cases

Strip your chart config down to the simplest possible version that reproduces the issue. This helps isolate whether the problem is with data, configuration, or environment.

Validate Configuration Options

Consult the official Highcharts API documentation to verify you are using supported configuration keys and valid value types.

Leverage Browser Developer Tools

Use:

- **Console** for JavaScript errors and warnings.
- **Network tab** to verify scripts and data files load correctly.
- **Elements inspector** to check chart container and tooltip DOM elements.

Check Highcharts Debug Logs

Enable verbose debugging by:

```
Highcharts.setOptions({ debug: true });
```

This outputs additional internal logs to the console that can help identify issues.

11.1.3 Summary

- Verify container presence and size if the chart doesn't render.
- Confirm data formats align with chart and axis types.
- Inspect tooltip and legend DOM elements for CSS conflicts.
- Disable animations and enable Boost module to improve performance.
- Use minimal code examples to isolate problems.
- Consult official docs and enable debug logs for insight.

Following these steps will help you quickly identify and fix common Highcharts issues, ensuring your charts render correctly and perform optimally.

11.2 Performance Optimization Tips

When working with Highcharts, especially with large datasets or multiple charts on a page, performance can become a bottleneck. Slow rendering, laggy interactions, or delayed updates impact user experience. This section shares practical techniques to optimize performance, helping your charts stay smooth and responsive even under demanding conditions.

11.2.1 Enable Turbo Mode

Highcharts' **turbo mode** drastically speeds up rendering by skipping complex data parsing when series data exceeds 1,000 points (by default). It expects simple arrays of numbers or `[x, y]` pairs for efficient plotting.

```
series: [{
  turboThreshold: 1000, // default is 1000
  data: largeDataArray
}]
```

Before: Chart freezes or lags rendering 10,000 points. **After:** Smooth rendering with turbo mode enabled.

11.2.2 Disable Animation for Large or Frequent Updates

Animations look nice but can be costly for large datasets or rapid data changes. Disable animation for initial render or dynamic updates:

```
plotOptions: {
  series: {
    animation: false
  }
}
```

Or disable animation selectively during live updates:

```
chart.series[0].setData(newData, true, false); // false disables animation on update
```

11.2.3 Use the Boost Module for GPU Acceleration

Highcharts provides a **Boost module** that offloads rendering of large datasets to the GPU using WebGL, greatly improving performance.

- To enable:

```
boost: {
  enabled: true,
  useGPUTranslations: true
},
series: [{
  boostThreshold: 2000,
  data: largeDataArray
}]
```

- Works best for charts with thousands of points (e.g., scatter, line charts).

11.2.4 Throttle Redraws and Updates

Avoid calling redraw or update functions excessively during frequent data changes or UI interactions:

- Batch updates and call `chart.redraw()` once after multiple data changes.
- Use debouncing or throttling to limit update frequency in real-time scenarios.

Example:

```
let redrawTimeout;
function updateData(newData) {
  chart.series[0].setData(newData, false); // update without redraw
  clearTimeout(redrawTimeout);
  redrawTimeout = setTimeout(() => chart.redraw(), 100); // redraw once after updates
}
```

11.2.5 Optimize Series and Axis Configuration

- Reduce the number of series or points when possible.
- Limit axis ticks and labels to reduce rendering workload (`tickInterval`).
- Disable unnecessary chart features like shadows or markers.

```
plotOptions: {
  series: {
    marker: {
      enabled: false // disables markers on line charts for large data
    },
    shadow: false
  }
}
```

11.2.6 Before and After Example: FPS and Render Times

Scenario	Without Optimization	With Optimization
10,000-point line chart	Initial render: 3–5 seconds	Initial render: ~500 ms
Live update with 100 points/sec	Noticeable lag and freeze	Smooth updates, < 100 ms latency
Multiple charts on dashboard	UI becomes sluggish	Responsive and fluid interaction

11.2.7 Summary

- Use **turbo mode** for large data arrays to speed parsing.
- Disable animation on heavy or frequent updates.
- Enable the **Boost module** for GPU-accelerated rendering.
- Throttle redraws to batch multiple updates into one.
- Simplify series and axis configurations to reduce workload.

By applying these strategies, you ensure your Highcharts visualizations remain performant and user-friendly, even when handling complex data or high interactivity demands.

11.3 Security Considerations

When deploying Highcharts in production environments—especially those involving dynamic or user-submitted data—it's critical to follow security best practices. Improper handling can expose your applications to vulnerabilities such as cross-site scripting (XSS), injection attacks, or data leakage. This section highlights key security concerns and recommendations to keep your Highcharts implementations safe.

11.3.1 Avoid Unsafe Code Execution

Highcharts itself does not use `eval()` or similar JavaScript code execution functions, but custom configurations or formatter functions **can** introduce risks if they include dynamically generated code or user input.

- **Never** insert user-generated content directly into formatter functions without sanitization.
- Avoid dynamic construction of functions or scripts based on untrusted data.
- Use strict content policies to prevent injection of malicious code.

11.3.2 Beware of XSS Risks in Tooltips and Labels

Tooltips, data labels, and legends often contain HTML or formatted text. If user data is included here, attackers could inject malicious scripts.

Best practices:

- Sanitize or escape any user input displayed inside tooltips or labels.
- Prefer using Highcharts' built-in text rendering rather than raw HTML when possible.
- Use libraries like DOMPurify to sanitize any HTML strings before injecting.

Example risk:

```
tooltip: {
  formatter: function() {
    // Unsafe if this.point.name comes from user input
    return '<b>' + this.point.name + '</b>: ' + this.y;
  }
}
```

Safer approach: sanitize `this.point.name` before usage or avoid HTML tags.

11.3.3 Secure Exporting Endpoints

When using Highcharts' exporting feature, especially with custom or server-side export servers:

- Ensure export servers are secured and do not allow arbitrary code execution.
- Use HTTPS to protect data sent to and from export endpoints.
- Limit export file sizes and types to prevent abuse or denial-of-service attacks.

11.3.4 Sandbox and Content Security Policy (CSP)

To further enhance security:

- Use **iframe sandboxing** if embedding charts from untrusted sources.
- Implement strong **Content Security Policies** (CSP) to restrict scripts, styles, and data sources.
- Avoid inline scripts or styles when possible.

11.3.5 Summary of Security Best Practices

Recommendation	Reason
Sanitize all user-submitted data before display	Prevent XSS via tooltips, labels
Avoid dynamic code execution in formatters	Prevent injection and arbitrary script
Secure exporting endpoints with HTTPS and auth	Protect data and prevent unauthorized use
Apply CSP headers and sandbox if embedding	Limit attack surface in browsers

By adhering to these guidelines, you protect your Highcharts-powered applications from common web security threats, maintaining trust and integrity for your users.

11.4 Accessibility Compliance

Making data visualizations accessible ensures that all users—including those with disabilities—can interact with and understand your charts. Highcharts comes with robust built-in accessibility features that help developers meet web accessibility standards such as WCAG and ARIA guidelines. This section explains how to leverage these features and test for accessibility compliance.

11.4.1 ARIA Roles and Semantic Markup

Highcharts automatically adds **ARIA roles and properties** to chart elements, helping assistive technologies understand the structure and purpose of your charts. This includes roles for the chart container, series, points, and legends.

- Roles like `img` or `listitem` describe visual elements.
- Descriptions and labels clarify chart content.

11.4.2 Keyboard Navigation Support

Users who cannot use a mouse rely on keyboard navigation. Highcharts supports:

- Navigating between series and points using keyboard arrows and tab keys.
- Activating tooltips and drilldowns with the keyboard.
- Accessible focus indicators for current selection.

To enable keyboard navigation, make sure the accessibility module is loaded and active:

```
accessibility: {
  enabled: true,
  keyboardNavigation: {
    enabled: true
  }
}
```

11.4.3 Screen Reader Compatibility

Highcharts provides **screen reader-friendly descriptions** for chart types, axes, and data points. When enabled, the library emits meaningful text updates that screen readers like **NVDA**, **JAWS**, or **VoiceOver** can read.

- Descriptions include chart summaries, data values, and interaction instructions.
- Live regions announce updates dynamically.

11.4.4 High-Contrast and Customizable Themes

For users with low vision or color blindness, Highcharts supports:

- Customizable color palettes with sufficient contrast.
- Integration with high-contrast themes by overriding colors and styles.
- Accessibility options to control focus outlines and font sizes.

Example to set a high-contrast color scheme:

```
Highcharts.setOptions({
  accessibility: {
    highContrastTheme: true
  },
  colors: ['#000000', '#FFFFFF', '#FFD700']
});
```

11.4.5 Configuring the Accessibility Module

To activate and customize accessibility features, include the module and configure options:

```
Highcharts.chart('container', {
  accessibility: {
    enabled: true,
    description: 'A simple line chart showing sales over time',
    keyboardNavigation: { enabled: true },
    point: {
      valueDescriptionFormat: '{index}. {xDescription}, {value}.'
    }
  },
  // other chart options...
});
```

11.4.6 Testing Your Charts for Accessibility

- Use screen readers like **NVDA** (Windows) or **VoiceOver** (Mac) to listen to how charts are described.
- Test keyboard navigation by tabbing through chart elements.
- Employ accessibility evaluation tools such as **axe**, **WAVE**, or browser devtools audits.
- Check color contrast ratios with tools like **Color Contrast Analyzer**.

11.4.7 Summary

Feature	Benefit	How to Enable
ARIA roles	Semantic markup for assistive tech	Enabled by default with accessibility module
Keyboard navigation	Usable without mouse	`accessibility.keyboardNavigation.enabled = true`
Screen reader descriptions	Clear verbal data summaries	`accessibility.enabled = true`
High-contrast themes	Improved visibility	Customize colors and enable theme

By thoughtfully enabling and testing Highcharts accessibility features, you make your data visualizations inclusive and usable for a broader audience, ensuring compliance with accessibility standards and enhancing user experience for everyone.