Data Visualization With Chart.js

Alireza Mohammadi

Fall 2017

Introduction To Data Visualization

- You can tell powerful stories with data.
- If your website or application is **data-intensive**, then you will need to find a way to **make that data easy to visualize**.
- Humans, after all, are not wonderful at understanding long lists of raw numbers.
- Charts and graphs can make **complicated statistical relationships obvious** and intuitive.
- Using charts when it’s beneficial, will make your website **easier to understand** and visually more appealing.

Why Chart.js

- It can be learned and leveraged **quickly**.
- It's designed with **simplicity** in mind, yet is extremely **customizable**.
- Chart.js is one of the **quickest** and **easiest** libraries to learn that **doesn't** heavily **limit** your options.

Why Chart.js (Cont.)

- It comes with **eight different chart types** that will cover almost all of your data visualization needs.

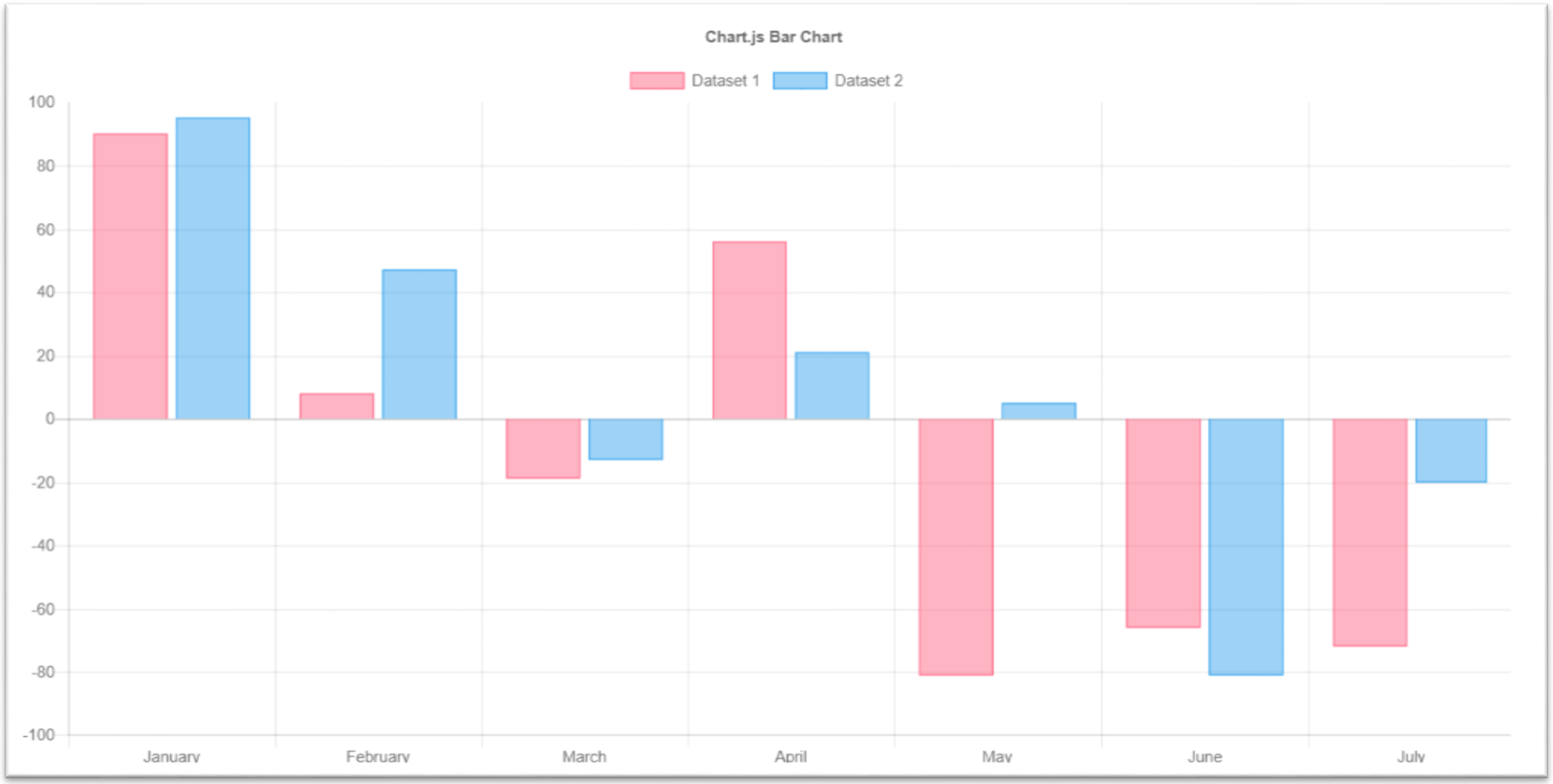
- Chart.js is **actively** maintained to a **high standard** by the **open source community**.

- Chart.js provides **responsive** charts that displayed correctly in any **device** with any **display size** and resolution.

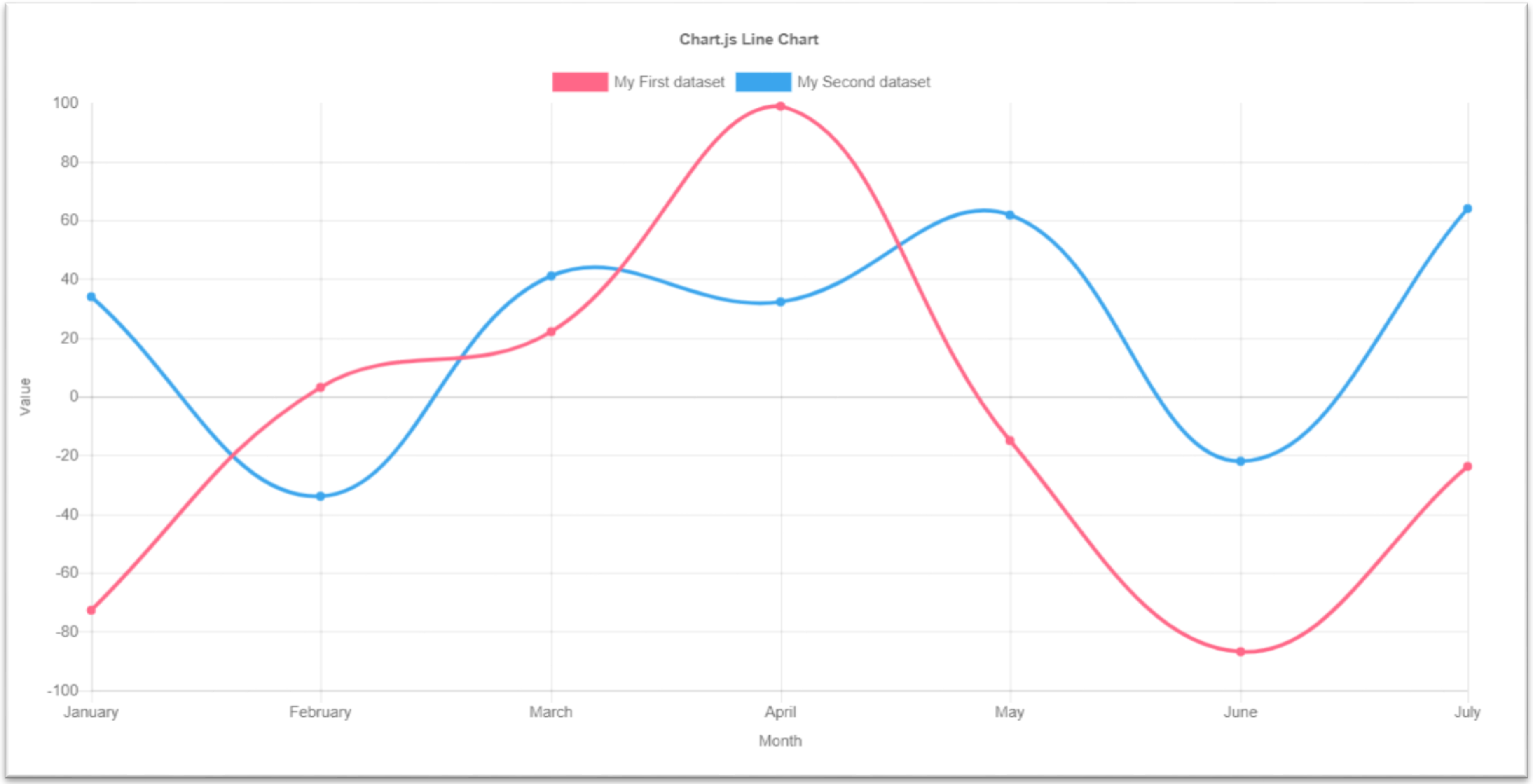
What you’ll need

- Basic knowledge from **HTML**, **CSS** & **JavaScript**.
- A text editor like **Atom**, **Visual Studio Code** or even window **notepad**.
- Some experience in **web API** and **JS** will help you grasp the nuances of what’s going on.

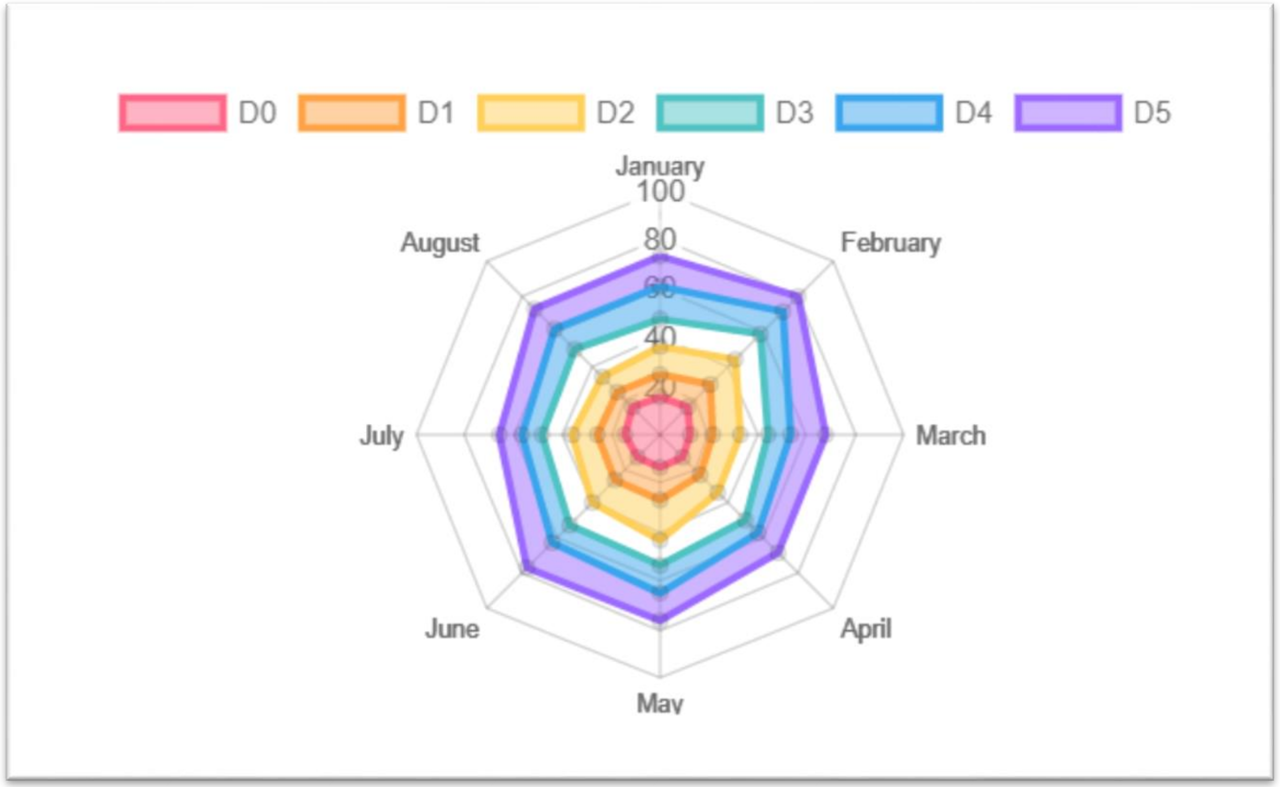
Example 1: Bar Chart (Vertical)



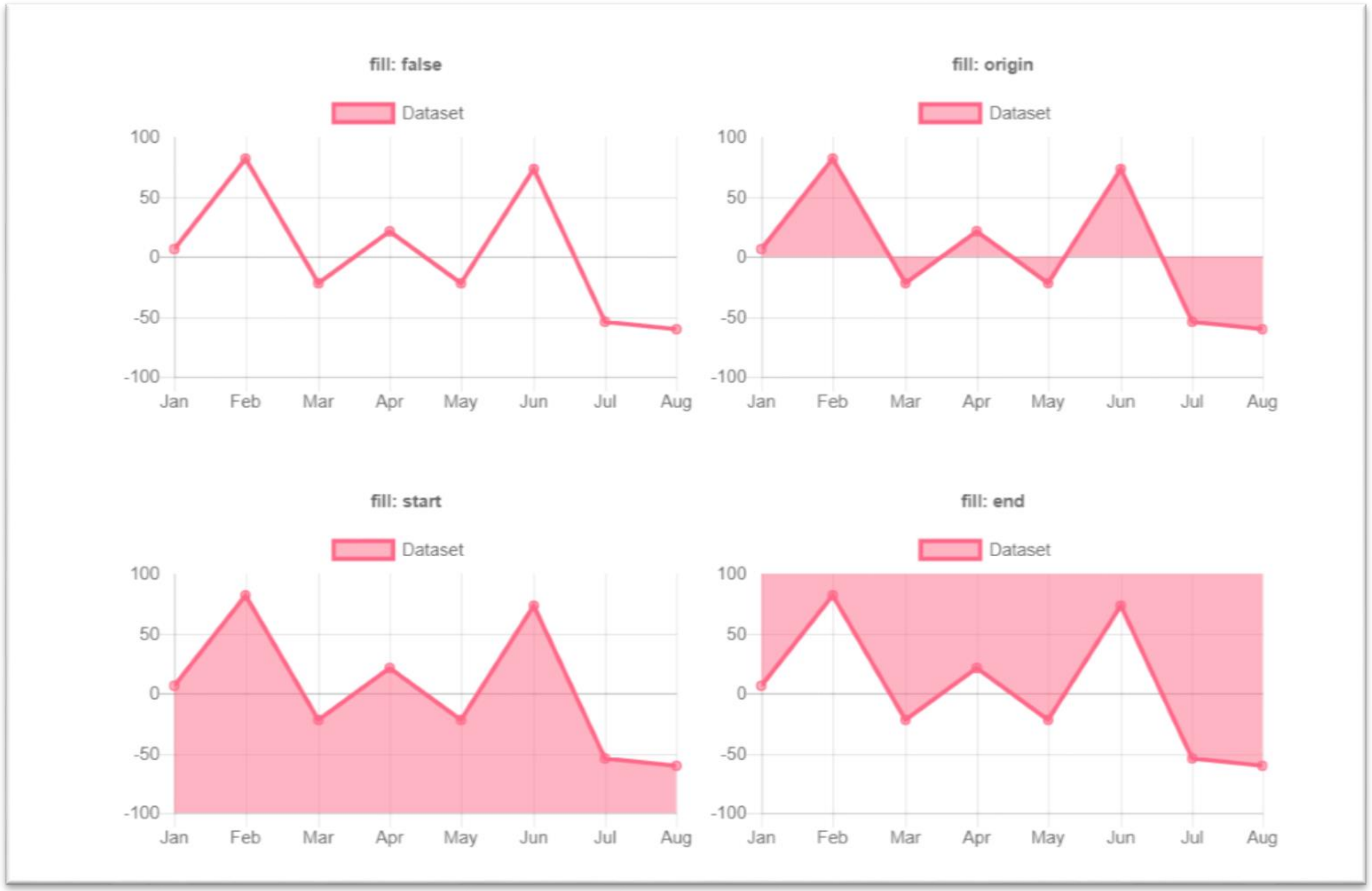
Example 2: Line Chart (Basic)



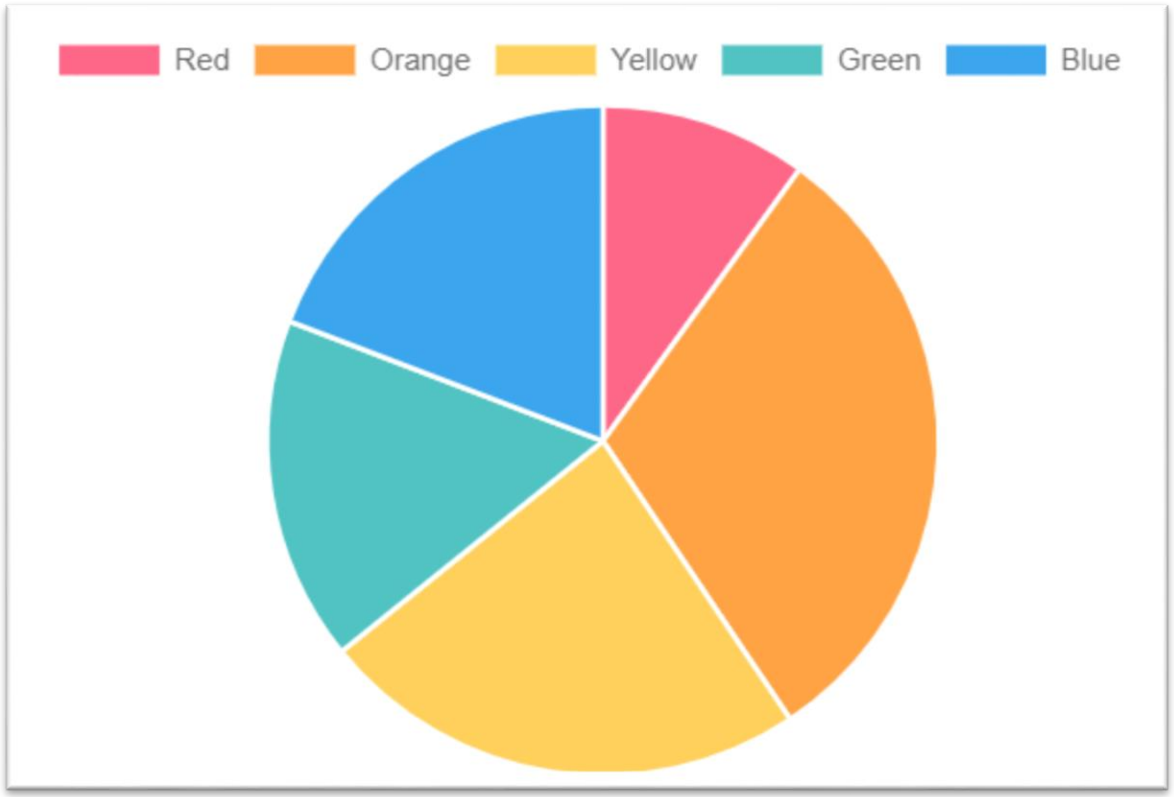
Example 3: Area Chart (Radar)



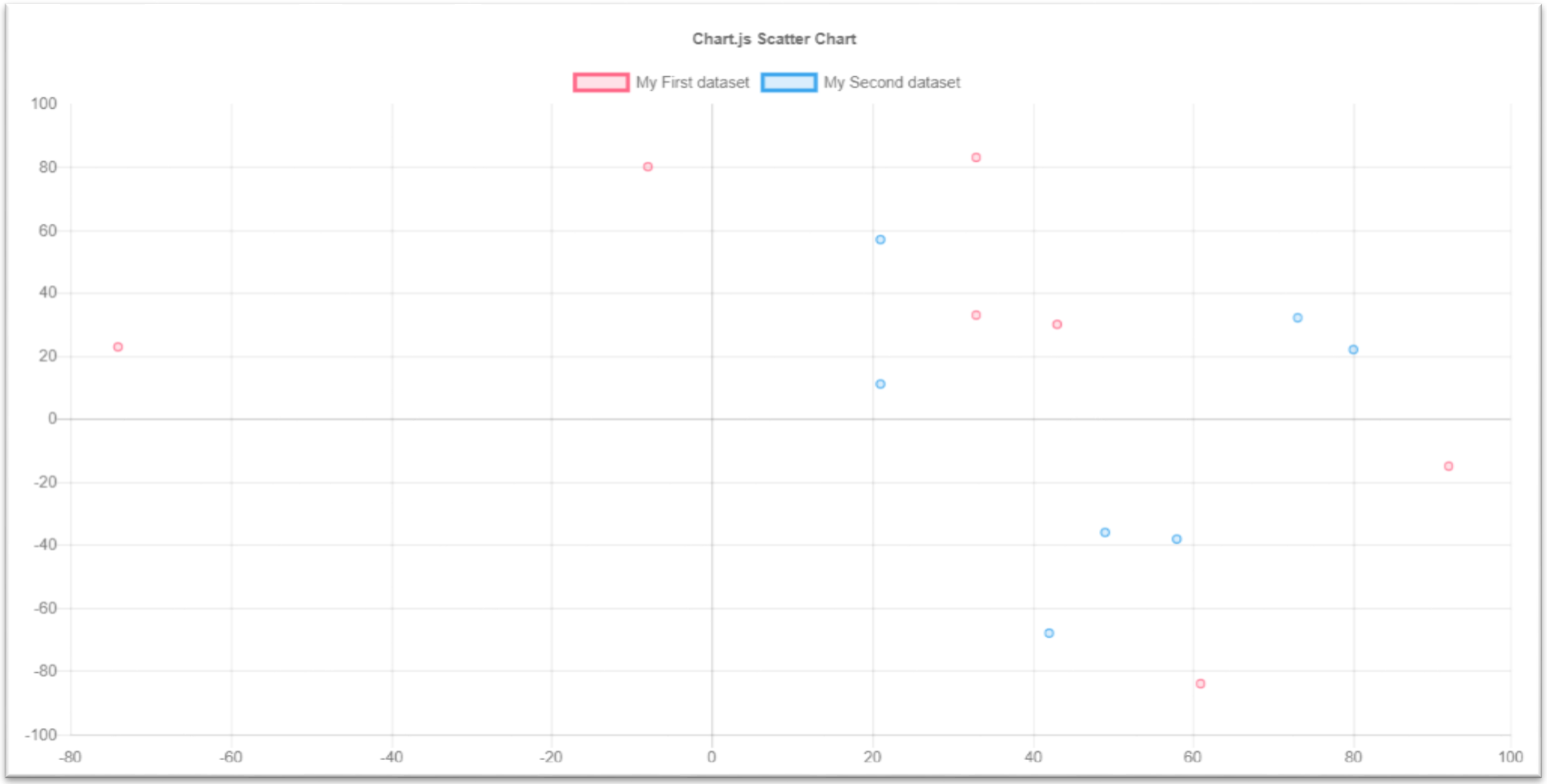
Example 4: Area Chart (Boundaries)



Example 5: Pie Chart



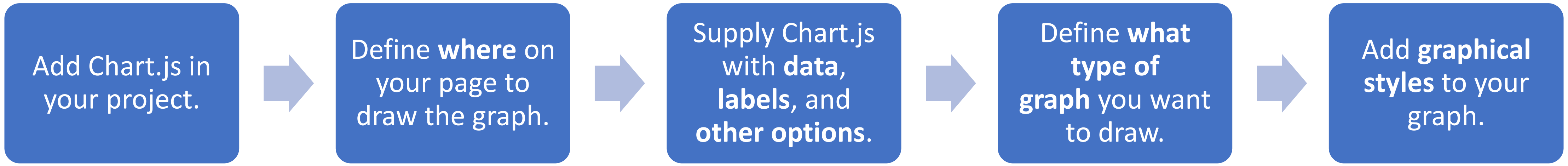
Example 6: Scatter Chart



Installing Chart.js

- Installing using a **package manager** like **bower** as bellow:
  1. Create a bower project in console or terminal with `bower init` command.
  2. Download and install Chart.js with `bower install --save chart.js` command.
  3. Create an empty `.html` file and link `.css` and `.js` files to it.
- You can download source from [GitHub]() and link required files in your project.
- You can also use [CDN]() and link files into your project without saving them.

Steps To Draw A Chart



Add Chart.js in your project.

Define **where** on your page to draw the graph.

Supply Chart.js with **data**, **labels**, and **other options**.

Define **what type of graph** you want to draw.

Add **graphical styles** to your graph.

Step 1: Add Chart.js

- Create an empty `.html` file, `.js` file and `.css` file if needed and link them in your `.html` page.
- If you don’t want to use CDN, install Chart.js using one of the methods described in previews slides.
- Link `chart.js` (or `chart.min.js`) file in your html page like below:

```
<script src="[route prefix]/chart.min.js"></script>
```

Step 2: Prepare a place in your HTML to render the chart

- The last thing we need to prepare before we can start visualizing our data is to define an area in our HTML where we want to draw the graph.
- For Chart.js you do this by adding a `canvas` element, and setting `width` and `height` to define the proportions of your graph.

```
<canvas id="myChart" width="1600" height="900"></canvas>
```

- Notice that we've added an `id` (`myChart`) to the `canvas` element that we can later use to reference our designated graph element in JavaScript or CSS.

Step 3: Prepare the data

- Here's the **raw data** that we'll be using:

World historical and predicted populations (in millions)

Country	1500	1600	1700	1750	1800	1850	1900	1950	1999	2050
Africa	86	114	106	106	107	111	133	221	783	2478
Asia	282	350	411	502	635	809	947	1402	3700	5267
Europe	168	170	178	190	203	276	408	547	675	734
Latin America	40	20	10	16	24	38	74	167	508	784
North America	6	3	2	2	7	26	82	172	312	433

Step 3: Prepare the data (Cont.)

- Chart.js expects the data to be passed in the **form of a set of arrays**.
- The table in previous slide, reformatted to arrays, looks like so:

```
// Our labels along the x-axis
var years = [1500, 1600, 1700, 1750, 1800, 1850, 1900, 1950, 1999, 2050];
// For drawing the lines
var africa = [86, 114, 106, 106, 107, 111, 133, 221, 783, 2478];
var asia = [282, 350, 411, 502, 635, 809, 947, 1402, 3700, 5267];
var europe = [168, 170, 178, 190, 203, 276, 408, 547, 675, 734];
var latinAmerica = [40, 20, 10, 16, 24, 38, 74, 167, 508, 784];
var northAmerica = [6, 3, 2, 2, 7, 26, 82, 172, 312, 433];
```

Step 4: Draw a line!

- All we need to do is **define what graph we want to draw**, and pass in the data that we want to visualize.
- Let’s start by drawing one single line to see if we can get it to work:

```
var ctx = document.getElementById("myChart");
var myChart = new Chart(ctx, {
    type: 'line’,
    data: {
        labels: years,
        datasets: [
            {
                data: africa
            }
        ]
    }
});
```

Step 4: Draw a line! (Cont.)

- What's happening in this bit of code?

- First, we locate the `canvas` element that we added earlier to our `index.html` file (notice `"myChart"`):

```
var ctx = document.getElementById("myChart");
```

Step 4: Draw a line! (Cont.)

- Then, using that `canvas` element, we create a line chart (`type: 'line'`), and pass along some of our data.
- `labels: years` sets our array of `years` (that we created earlier) for the labels along the x-axis, and `data: africa` uses our `africa` variable to draw the line.
- You may have noticed that our line is missing a label (it says *`undefined`* at the top of the graph), and it's not very colorful. Boo! Let's make it!

Step 5: Style the line

- Start out by giving our first line a name. After `data: africa`, add a comma (hey! I’m serious about the comma (remember the comma!), miss it and everything breaks), create a new row, and add `label: "Africa"`:

```
{
    data: africa,
    label: "Africa"
}
```

Step 5: Style the line (Cont.)

- To set the border color and remove the big gray area below the graph, add another comma after label: "Africa" and add these two lines:

```
borderColor: "#3e95cd",
fill: false
```

- refresh and you should see a blue line named Africa!

Step 6: Add the rest of the data

- All we need to do now is copy the code for Africa and paste it another four times, adding a comma after every }.

```
{
    data: africa,
    label: "Africa",
    borderColor: "#3e95cd",
    fill: false
},
{
    data: asia,
    label: "Asia",
    borderColor: "#3e95cd",
    fill: false
},
{
    data: europe,
    label: "Europe",
    borderColor: "#3e95cd",
    fill: false
},
...
```

Thank You!