How to use Chart.js

André Gardi
Oct 8, 2018 · 6 min read

Chart.js is a popular open source library that helps us to plot data in web applications. It is highly customizable, but configuring all of its options remains a challenge for some people. Let's explore it, starting from a simple example and building upon it.

Basic implementation:
To keep it simple, I am going to use the CDN version of the library. However, if you are planning on using it with React or Angular, I recommend that you use the npm package and install it via `npm install chart.js --save`.

We are also going to use jQuery to make the ajax requests.
Our basic implementation will plot the revenues data of a restaurant for every day of the week.

Let's start with the **index.html**

```
1  <!DOCTYPE html>
2  <html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Exploring Chart.js</title>
</head>
<style>
    .container {
        width: 50%;
        height: 50%;
    }
</style>
<body>

    <button id="renderBtn">
        Render
    </button>
    <div class="container">
        <canvas id="myChart"></canvas>
    </div>

</body>

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="./myChart.js"></script>

</html>
```

And **myChart.js**

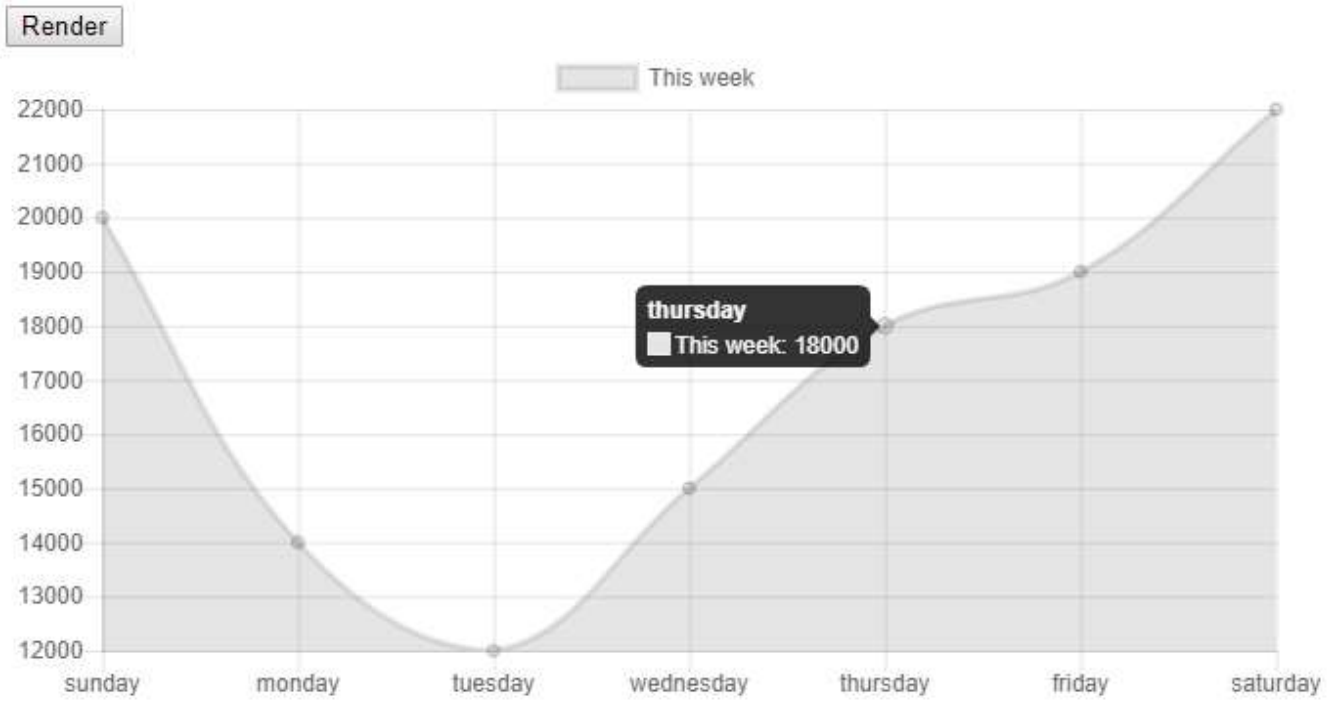
```
function renderChart(data, labels) {
    var ctx = document.getElementById("myChart").getContext('2d');
    var myChart = new Chart(ctx, {
        type: 'line',
        data: {
            labels: labels,
            datasets: [{
                label: 'This week',
                data: data,
            }]
        },
    });
}

```

```
$("#renderBtn").click(
    function () {
        data = [20000, 14000, 12000, 15000, 18000, 19000, 22000];
        labels =  ["sunday", "monday", "tuesday", "wednesday", "thursday", "friday", "saturday"
        renderChart(data, labels);
    }
);
```

Setting the `type` variable, we could change the line chart into a bar chart, or even a pie chart. All of the different types of charts can be seen here.

As you can see, `datasets` is an array. That means that we can plot two or more data series in the same chart. We will get back to this possibility later on in the article.

The result of our project so far is:



What I don't like about it:

- The Y axis doesn't start at 0. It starts at 12000, the lowest value in the series;
- It is gray and I want other colors;
- It doesn't show data as currency: 20000 -> U$ 20,000.00

So, I made a few modifications:

```
function float2dollar(value){
```

```
2       return "U$ "+(value).toFixed(2).replace(/\d(?=(\d{3})+\.)/g, '$&,');
3   }
4
5   function renderChart(data, labels) {
6       var ctx = document.getElementById("myChart").getContext('2d');
7       var myChart = new Chart(ctx, {
8           type: 'line',
9           data: {
10              labels: labels,
11              datasets: [{
12                  label: 'This week',
13                  data: data,
14                  borderColor: 'rgba(75, 192, 192, 1)',
15                  backgroundColor: 'rgba(75, 192, 192, 0.2)',
16              }]
17          },
18          options: {
19              scales: {
20                  yAxes: [{
21                      ticks: {
22                          beginAtZero: true,
23                          callback: function(value, index, values) {
24                              return float2dollar(value);
25                          }
26                      }
27                  }]
28              }
29          },
30      });
31  }
```

line 1:

`float2dollar` is a function which formats a float into a currency string:

18000 -> U$ 18,000.00

line 14:

`boderColor` and `backgroundColor` are properties which change the line and area colors.

line 22:

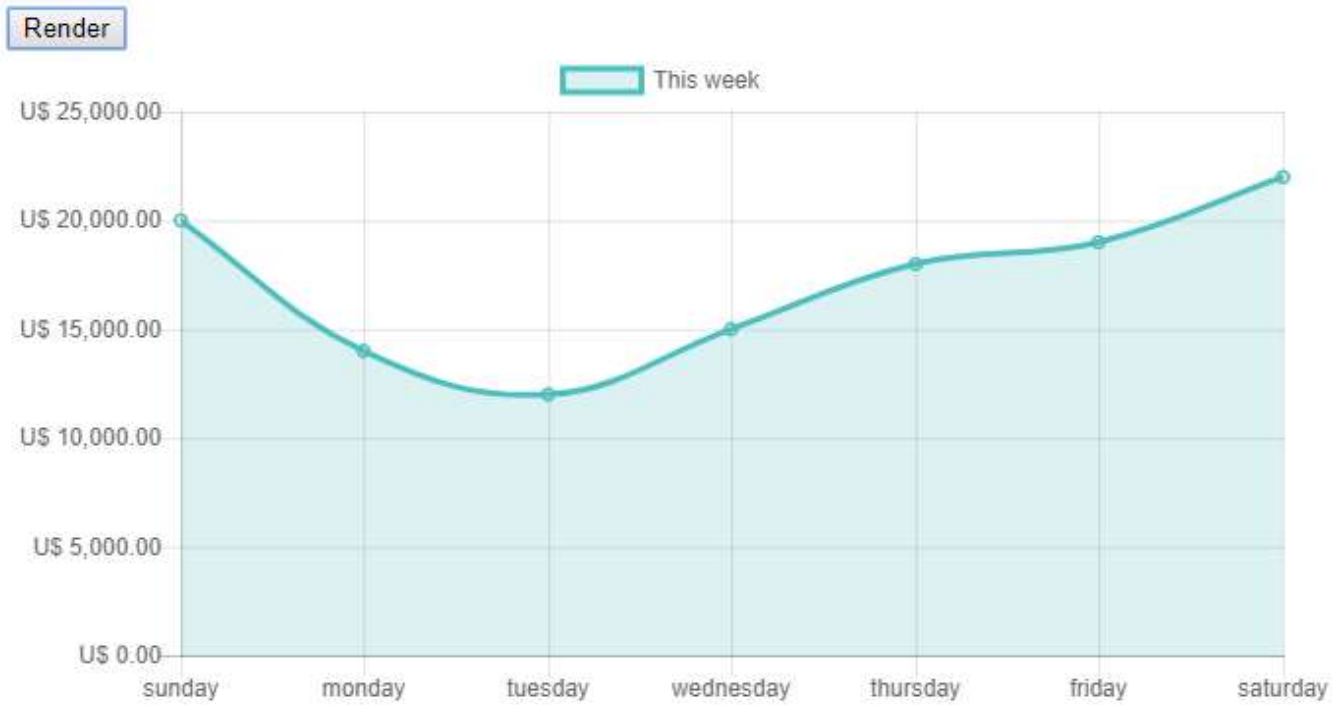
`beginAtZero: true` forces the axes to start at 0 and not at 12000.

line 23:

`callback` of the `ticks` of the axes is called every time that a tick is rendered. The

`value` variable is the default value displayed. When we override this callback, we can change the text which is displayed on the axis. Here, we call our `float2dollar` function.

We can already see the changes:



Load data from an Ajax request:

To have a backend which allows us to make a GET request without much work, we will use the package json-server. It can simulate a REST server from a JSON file.

```
npm install json-server -save
```

On our node package.json, we will set the npm start script to run the json-server with a 2 seconds delay. Because the data is all local, we will have immediate responses. The artificial delay will help us to simulate a real internet connection.

```
"start": "json-server data.json --delay 2000",
```

When we run the npm start script, the service will be available on port 3000.

For that to work, we need to create our data.json file.

```
{
    "chartData": {
        "labels": [
            "sunday",
            "monday",
            "tuesday",
            "wednesday",
            "thursday",
            "friday",
            "saturday"
        ],
        "thisWeek": [
            20000,
            14000,
            12000,
            15000,
            18000,
            19000,
            22000
        ]
    }
}
```

data.json hosted with ♡ by GitHub view raw

We will also require a new div on the html to show a “Loading” gif and an error message.

```
<style>
    .container {
        width: 50%;
        height: 50%;
        position: relative;
    }
    #loadingMessage{
        position: absolute;
        left: 50%;
        top: 50%;
        transform: translate(-50%, -50%);
    }
</style>
<body>

    <button id="renderBtn">
        Render
```

```
        Render
    </button>
    <div class="container">
        <div id="loadingMessage"></div>
        <canvas id="myChart"></canvas>
    </div>

</body>
```

index.html hosted with ♡ by GitHub view raw

To make things more organized, we will create a function `getChartData()` to separate the ajax request from the button click.

```
function getChartData() {
    $("#loadingMessage").html('<img src="./giphy.gif" alt="" srcset="">');
    $.ajax({
        url: "http://localhost:3000/chartdata",
        success: function (result) {
            $("#loadingMessage").html("");
            var data = [];
            data.push(result.thisWeek);
            data.push(result.lastWeek);
            var labels = result.labels;
            renderChart(data, labels);
        },
        error: function (err) {
            $("#loadingMessage").html("Error");
        }
    });
}

$("#renderBtn").click(
    function () {
        getChartData();
    }
);
```

myChart.js hosted with ♡ by GitHub view raw

After that we have our loading gif.

Now, let's try a chart with two data series:

Suppose we want to compare the revenues of this week with those from last week. Let's keep the current week's series in blue and we'll add last week's series in gray.

The first thing we need to update is our data.json

```
{
    "chartData": {
        "labels": [
            "sunday",
            "monday",
            "tuesday",
            "wednesday",
            "thursday",
            "friday",
            "saturday"
        ],
        "thisWeek": [
            20000,
            14000,
            12000,
            15000,
            18000,
            19000,
            22000
        ],
        "lastWeek": [
            19000,
            10000,
            14000,
            14000,
            15000,
            22000,
            24000
        ]
    }
}
```

Then, we need to change the myChart.js file:

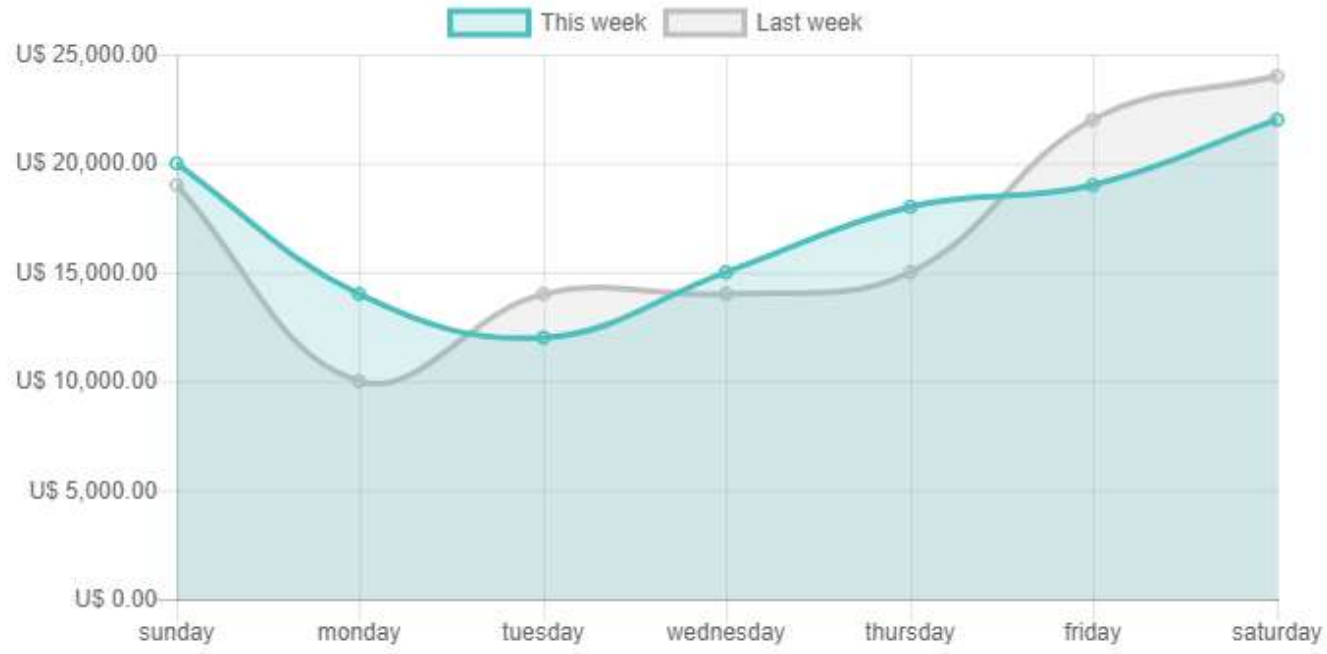
The data that we send to the renderChart function will be an array of two arrays. The first (`data[0]`) will be the data from this week's revenues and the second (`data[1]`) will be the data from the last week.

Finally, the datasets from the chart will have a second object: the series from last week.

```
1
2  function float2dollar(value) {
3      return "U$ " + (value).toFixed(2).replace(/\d(?=(\d{3})+\.)/g, '$&,');
4  }
5
6  function renderChart(data, labels) {
7      var ctx = document.getElementById("myChart").getContext('2d');
8      var myChart = new Chart(ctx, {
9          type: 'line',
10         data: {
11             labels: labels,
12             datasets: [
13                 {
14                     label: 'This week',
15                     data: data[0],
16                     borderColor: 'rgba(75, 192, 192, 1)',
17                     backgroundColor: 'rgba(75, 192, 192, 0.2)',
18                 },
19                 {
20                     label: 'Last week',
21                     data: data[1],
22                     borderColor: 'rgba(192, 192, 192, 1)',
23                     backgroundColor: 'rgba(192, 192, 192, 0.2)',
24                 }
25             ]
26         },
27         options: {
28             scales: {
29                 yAxes: [{
30                     ticks: {
31                         beginAtZero: true,
32                         callback: function (value, index, values) {
33                             return float2dollar(value);
34                         }
35                     }
36                 }]
37             },
38         }
39     });
40 }
```

```
41
42  function getChartData() {
43      $("#loadingMessage").html('<img src="./giphy.gif" alt="" srcset="">');
44      $.ajax({
45          url: "http://localhost:3000/chartdata",
46          success: function (result) {
47              $("#loadingMessage").html("");
48              var data = [];
49              data.push(result.thisWeek);
50              data.push(result.lastWeek);
51              var labels = result.labels;
52              renderChart(data, labels);
53          },
54          error: function (err) {
55              $("#loadingMessage").html("Error");
56          }
57      });
58  }
59
60  $("#renderBtn").click(
61      function () {
62          getChartData();
63      }
```

Render



Other types of charts:

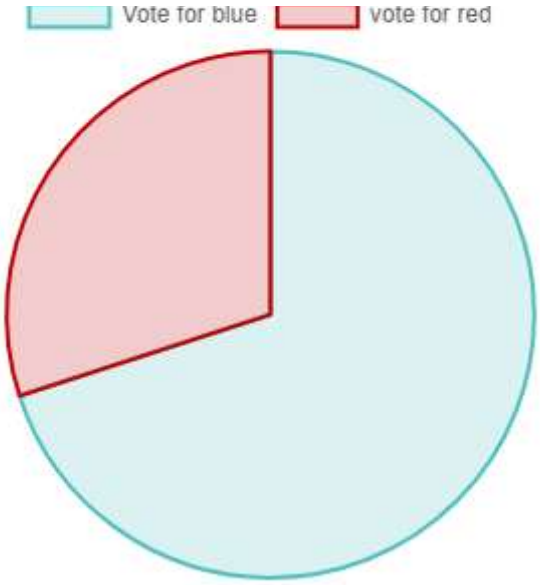
As mentioned earlier, Chart.js provides a wide variety of chart types.

The properties we study here are shared between different types of graphics, with some differences. In the pie chart, for example, instead of passing only one color per series, we may pass an array of colors, since different slices of the pie should have different colors.

Let's also add a title to the chart using the `text` property of the `title`.

```
var labels = [
    "Vote for blue",
    "vote for red",
];
var data = [
    70,
    30,
];
var pie = document.getElementById("pieChart").getContext('2d');
var myChart = new Chart(pie, {
    type: 'pie',
    data: {
        labels: labels,
        datasets: [
            {
                data: data,
                borderColor: ['rgba(75, 192, 192, 1)', 'rgba(192, 0, 0, 1)'],
                backgroundColor: ['rgba(75, 192, 192, 0.2)', 'rgba(192, 0, 0, 0.2)'],
            }
        ]
    },
    options: {
        title: {
            display: true,
            text: "Colors election"
        }
    }
});
```

The result:



Mixing chart types

Chart.js allows us to mix two types of chart on the same canvas. Let's go back to the restaurant example. Now we want to analyze how many clients we had every day, but still see the revenues.

In this example, we can use a bar chart for the clients mixed with a line chart for the revenues.

It is important to point out that, in this case, we will need to use a secondary Y axis. This is because the revenues will be numerically larger than the customers and therefore the bars would be very small on the revenues scale.

For that we will create another file, **mixChart.js**

```
1
2   var labels = [
3       "sunday",
4       "monday",
5       "tuesday",
6       "wednesday",
7       "thursday",
8       "friday",
9       "saturday"
10  ];
11  var revenues = [
12      20000,
13      14000,
14      12000,
15      15000,
16      18000,
```

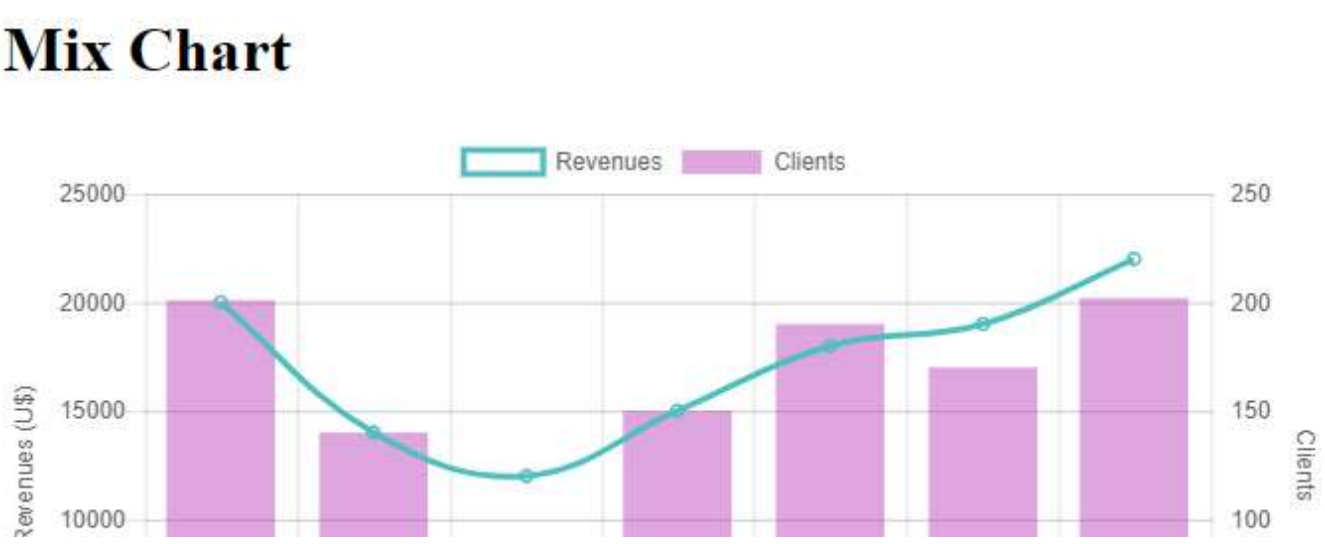
```
    19000,
    22000
];
var clients = [
    201,
    140,
    80,
    150,
    190,
    170,
    202
];
var mix = document.getElementById("mixChart").getContext('2d');
var mixChart = new Chart(mix, {
    type: 'bar',
    data: {
        labels: labels,
        datasets: [
            {
                type: 'line',
                label: "Revenues",
                data: revenues,
                borderColor: 'rgba(75, 192, 192, 1)',
                backgroundColor: 'rgba(0, 0, 0, 0)',
                yAxisID: 'revenues',
            },
            {
                label: "Clients",
                data: clients,
                borderColor: 'rgba(0, 0, 0, 0)',
                backgroundColor: 'rgba(192, 75, 192, 0.5)',
                yAxisID: 'clients',
            }
        ]
    },
    options: {
        scales: {
            yAxes: [
                {
                    id: "revenues",
                    ticks: {
                        beginAtZero: true,
                    },
                    scaleLabel: {
                        display: true,
                        labelString: 'Revenues (U$)'
                      }
```

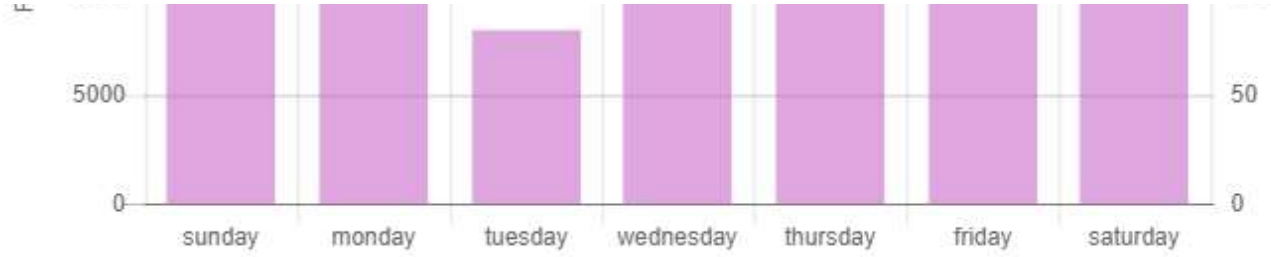
```
                },
                {
                    id: "clients",
                    position: 'right',
                    ticks: {
                        beginAtZero: true,
                    },
                    scaleLabel: {
                        display: true,
                        labelString: 'Clients'
                      }
                },
            ]
        },
    }
```

want to change, you have to redefine the variable `type` inside the individual dataset. For our example, we set the main type as `type: "bar"` but, inside the first dataset, we redefine `type: "line"`.

`yAxes` is a property from `scales`, which is an object of `options`. This `yAxes` is actually an array and allows us to use multiple scales for different series. In this example we use two Y axes. For this reason, we need to give an `id` for each. The first Y axis has `id:"revenues"` and the second `id:"clients"`.

We've used `position: 'right'` on the client axis to show its ticks on the right side of the chart. To make things more readable, we also add a `labelString` to each axis, so people may know that the left axis is the revenues, and the right one is the number of clients.





Now what?

Actually a lot! As you can see, Chart.js is very customizable and we would still have to see many topics to cover all possibilities.

Some examples are:
- Show stacked data;
- Customize legends;
- Customize the tooltips;
- Two dimensional data types;
- Interactions events (e.g., click);
- Logarithmic scale;
And more…

Also, there are plugins which improve the use of Chart.js. I think **chartjs-plugin-datalabels** is particularly useful and would recommend you to take a look at it.

You can check the files used on this article in our GitHub repository.

Let us know in the comments if you would like to read more about chart.js in the future.

JavaScript Chartjs Charts Front End Development Data Science