ChartJS Tutorial For Beginners

https://www.codewall.co.uk/

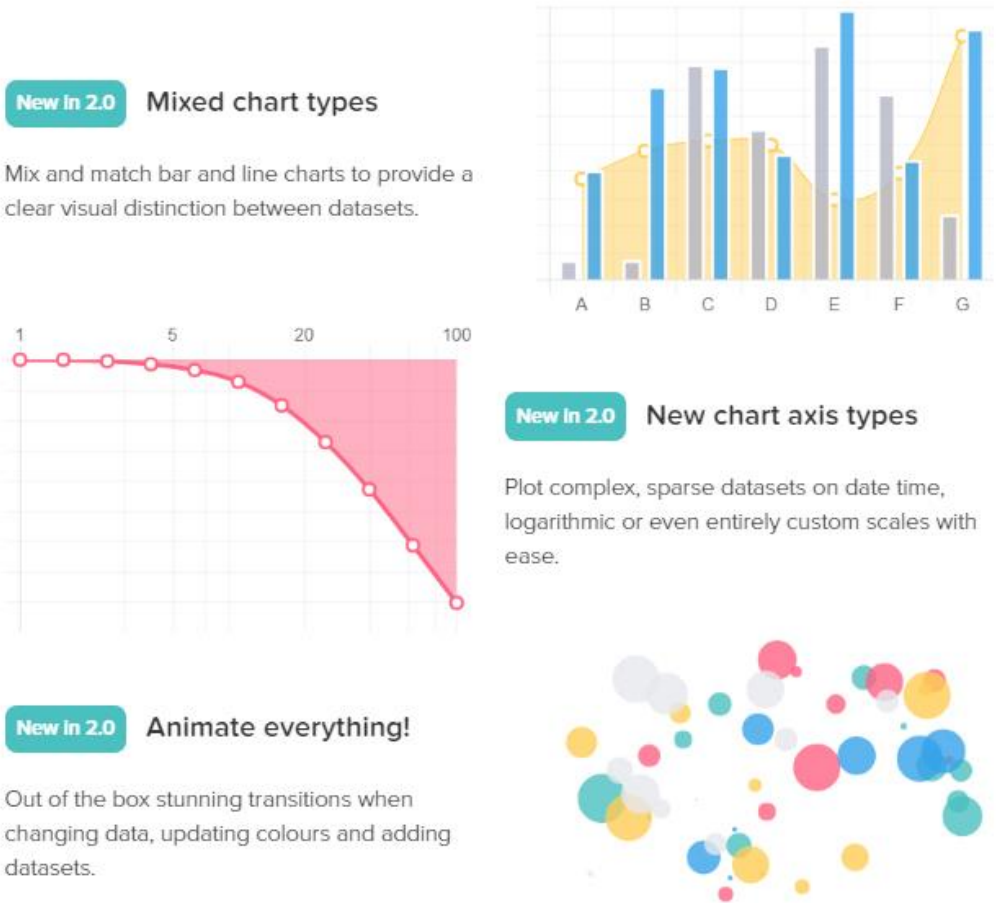
@codewallblog

Contents

Welcome To The ChartJS Tutorial

This is the starting point to becoming competent with the ChartJS library. Hopefully, around 3 lessons will be released per week from the day this article is published. You can sign up to the newsletter at the end of the article for updates on new lessons too. We will be going through working with the following graphs -

- Bar
- Radar
- Line
- Donut
- Pie
- Scatter
- Polar Area
- Area
- Mixed



Within the series of tutorials, we will see how each can be configured and used in real-life environments. Testing data sourcing from a range of API's and even some databases. This series will put ChartJS through its paces also, delving deep into its features, events and styling. By the end of the series, you will feel comfortable enough to construct your very own data visualization dashboards and informative website-graphs.

Managing & Using Data

In general, JavaScript libraries are not always the same when it comes to how they require consuming data. Libraries can require pure JSON, arrays, Strings, even single integers. This is why it's important to get to grips with how the library you are working with can consume data, and hopefully, this is in a variety of ways too.

So how does ChartJS require data?

ChartJS likes to consume its data mainly by using the `Array` type, whether that be an array of numeric or string values alike. This is pretty great because JavaScript arrays are really simple to setup and work with. The only exception to this default behaviour is Scatter Charts which, require x and y coordinates in a JSON format.

The Data Property

`Data` is a property of the master JSON object that the main `Chart` function requires as a parameter. It exposes a sub-set of properties that can be populated. Here are the most important properties that need to be understood.

- `labels` - An array of x-axis Labels in either Numeric or String form.
- `dataSets`, which exposes another sub-set of properties
  - `data` - An array of data-point values that are parallel to the labels specified.
  - `backgroundColor` and `borderColor` A pair of String Arrays that consist of Hexadecimal or RGBA color values to color the bars and borders or other various visualizations.
  - `borderWidth` which takes a single `Integer` or `Float` value, this of course, specifies border widths across the data visualizations.

Understanding these properties will enable fluent construction of charts with any data, whilst knowing in most cases, the data will sit in an array and can be filled in a range of ways. Let's see this in the flesh -

Example taken from the ChartJS Documentation

```
data: {
        labels: ["Red", "Blue", "Yellow", "Green", "Purple", "Orange"],
        datasets: [{
                label: '# of Votes',
                data: [12, 19, 3, 5, 2, 3],
                backgroundColor: [
                        'rgba(255, 99, 132, 0.2)',
                        'rgba(54, 162, 235, 0.2)',
                        'rgba(255, 206, 86, 0.2)',
                        'rgba(75, 192, 192, 0.2)',
                        'rgba(153, 102, 255, 0.2)',
```

```
                        'rgba(255, 159, 64, 0.2)'
                ],
                borderColor: [
                        'rgba(255,99,132,1)',
                        'rgba(54, 162, 235, 1)',
                        'rgba(255, 206, 86, 1)',
                        'rgba(75, 192, 192, 1)',
                        'rgba(153, 102, 255, 1)',
                        'rgba(255, 159, 64, 1)'
                ],
                borderWidth: 1
        }]
}
```

The DataSets Property

By now, you will have realized that `datasets` is pluralized, meaning it is capable of having more than one **data set**. This is when some JSON knowledge is helpful, because the `datasets` property will need nested objects for multiple series of data. Here's an example of a two date series nested inside the `datasets` property.

```
data: {
        labels: ["Tokyo", "Mumbai", "Mexico City", "Shanghai", "Sao Paulo",
"New York", "Karachi", "Buenos Aires", "Delhi", "Moscow"],
        datasets: [{
                        label: 'Series 1',
                        data: [500, 50, 2424, 14040, 14141, 4111, 4544, 47,
5555, 6811],
                        backgroundColor: [
                                'rgba(255, 99, 132, 0.2)',
                                'rgba(54, 162, 235, 0.2)',
                                'rgba(255, 206, 86, 0.2)',
                                'rgba(75, 192, 192, 0.2)',
                                'rgba(153, 102, 255, 0.2)',
                                'rgba(255, 159, 64, 0.2)'
                        ],
                        borderColor: [
                                'rgba(255,99,132,1)',
                                'rgba(54, 162, 235, 1)',
                                'rgba(255, 206, 86, 1)',
                                'rgba(75, 192, 192, 1)',
                                'rgba(153, 102, 255, 1)',
                                'rgba(255, 159, 64, 1)'
                        ],
                        borderWidth: 1
                },
                {
                        label: 'Series 2',
                        data: [1288, 88942, 44545, 7588, 99, 242, 1417,
5504, 75, 457],
                        backgroundColor: [
                                'rgba(255, 99, 132, 0.2)',
                                'rgba(54, 162, 235, 0.2)',
                                'rgba(255, 206, 86, 0.2)',
                                'rgba(75, 192, 192, 0.2)',
                                'rgba(153, 102, 255, 0.2)',
                                'rgba(255, 159, 64, 0.2)'
                        ],
                        borderColor: [
```

```
                                'rgba(255,99,132,1)',
                                'rgba(54, 162, 235, 1)',
                                'rgba(255, 206, 86, 1)',
                                'rgba(75, 192, 192, 1)',
                                'rgba(153, 102, 255, 1)',
                                'rgba(255, 159, 64, 1)'
                        ],
                        borderWidth: 1
                }
        ]
}
```

This type of nesting can be multiplied many times, allowing you to create large multi-series data visualizations with ease. Throughout this series, there will be examples of both single and multi-series charts to play with and use in your own projects.

Hopefully this article will bring more clarity to how ChartJS manages data and in addition, help you become more content with working with these important properties too. If you have any queries, please leave a comment and I will get back to you as soon as I can.

Setting Up With ChartJS

To get started, ChartJS have already made many scenarios effortless to get up and running. Let's go through each way of using ChartJS in your own project.

1. Source Download

Depending if you want to play with the many samples provided by ChartJS, you could download the full source code. Similarly, you may want to edit the source code to suit certain needs within your project too, in this case, downloading the source would be best.

Personally, I like to get the full source code so that examples and documentation are easily to access, even when offline.

To get the full source code, follow these steps.

1. Go-to the latest repository on GitHub here (https://github.com/chartjs/Chart.js/releases/tag/v2.7.2).
2. Download the link named 'Source code (zip)'
3. Extract the contents of the folder somewhere safe.
4. Copy the Chart.js file from the dist/ folder to your project.
5. Finally, reference the Chart.js file in your HTML code.

Walla, you are ready to start coding!

2. Content Delivery Network

ChartJS is on the CDNJS website and therefore can be referenced directly from your project as-long as you have an **internet connection**. This is by far the quickest to get going, but obviously it has the drawback of always having to be on-line. To setup with CDN, see the following steps

1. Go-to https://cdnjs.com/libraries/Chart.js
2. Copy the link source you want to use, for example 'https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.js'.
3. Add this link within a script tag into your HTML code.

3. Package Managers

If 1 and 2 aren't how you like to add libraries to your project, there are a few alternatives. Namely, NPM & Bower. See the following commands to download the ChartJS package into your project. Remember to reference the scripts within your HTML code, though.

NPM

```
npm install chart.js --save
```

Bower

```
bower install chart.js --save
```

Integration

ChartJS website provides details on integrating their library with your current environment, this includes integrating with ES6 and common JS. Check it out here. (http://www.chartjs.org/docs/latest/getting-started/integration.html)

Create A Bar Chart

Let's start seeing some magic happen with working data visualizations, and what better way to start than a simple bar chart? I have already gathered some public data for the use within the tutorial so that you can copy it and easily add it to your own code-playground. In addition, I will create all demonstrations in a CodePen that you can fork, copy and save. For this PDF Version, images from the CodePen's will replace the actual working examples. Visit the article online to use the CodePens. In this tutorial, there is no need for any back-end programming as such, just pure HTML, CSS and JavaScript.

Let's get started

Each chart of the ChartJS library requires a HTML5 Canvas placeholder, solely required to render beautiful charts onto. If you haven't seen the canvas tag already, it looks like the following -

```
<canvas>
```

This moves us onto the very first part of the tutorial, adding our HTML element. Add this element to your HTML code -

HTML

```
<canvas id="myChart"></canvas>
```

The canvas ID is very important when using this library and if you can help it, use something more descriptive rather than 'myChart', especially if you have more than one chart on the page.

The ID will be used to instruct the charting library where to render the fully configured chart. With this in mind, let's start to add some script by creating a canvas context by specifying the ID we used in the HTML.

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');
```

Now we need to specify some data values & labels, as I mentioned before, I've already got some great public data prepared, see the following JavaScript -

```
// Define the data
var data = [22006299,  15834918,      14919501,      14797756,
        14433147,      13524139,      11877109,      11862073,
        11779606, 10452000]; // Add data values to array

var labels = ["Tokyo", "Mumbai",      "Mexico City", "Shanghai",    "Sao
Paulo", "New York",    "Karachi","Buenos Aires",     "Delhi","Moscow"]; //
Add labels to array
// End Defining data
```

Now let's setup the ChartJS object, passing in the canvas context we created earlier and pass in the arrays of data -

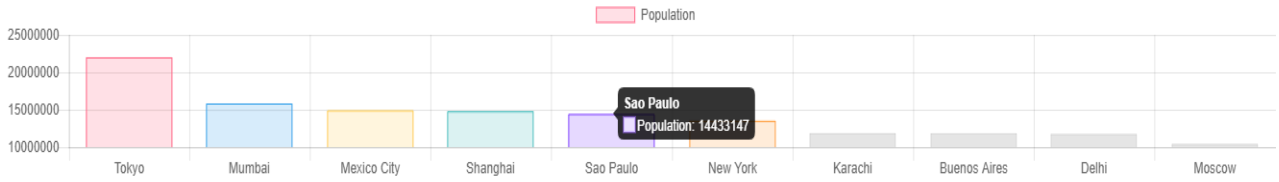
```
var myChart = new Chart(ctx, {
    type: 'bar',
    data: {
        labels: labels,
        datasets: [{
            label: 'Population', // Name the series
            data: data, // Specify the data values array
            backgroundColor: [ // Specify custom colors
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
            ],
            borderColor: [ // Add custom color borders
                'rgba(255,99,132,1)',
                'rgba(54, 162, 235, 1)',
                'rgba(255, 206, 86, 1)',
                'rgba(75, 192, 192, 1)',
                'rgba(153, 102, 255, 1)',
                'rgba(255, 159, 64, 1)'
            ],
            borderWidth: 1 // Specify bar border width
        }]
    },
    options: {
      responsive: true, // Instruct chart js to respond nicely.
```

```
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
    }
});
```

Some notable points from above -

- ChartJS requires a type, in this case, bar for bar chart.
- Passing in data is relatively easy, specifying the series, labels and values.
- Style each bar precisely how you want them to look with background, border colors and even border width.
- Set responsive to true so that the chart responds to the screen-width, leaving pixel perfect visualizations across different screen-sizes.
- Lastly, Make the Maintain Aspect Ration option false, so that it doesn't render as the full screen width and height. - You can remove this option to see exactly what I mean.

Let's see how this all looks!



Create A Line Chart

You will be happy to hear that creating a Line Chart takes minimal effort really. A few small option changes, slightly more data, for this example and a Line Chart can be rendered successfully. Within this example, we will use the same HTML canvas element and context variable. Obviously, if there was going to be multiple charts on one page, this wouldn't be feasible, but for this example, it's perfectly OK.

HTML

```
<canvas id="myChart"></canvas>
```

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');
```

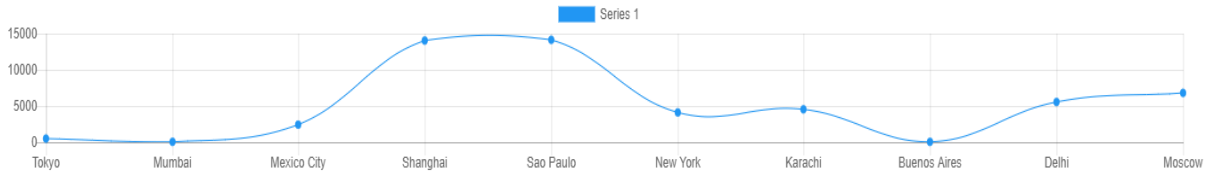
The following JavaScript chart configuration has been coded slightly different, rather than having the data in separate variables, I've coded them directly into the Chart object. Why? Just to showcase the different ways you can do it, ultimately, it's up to which way you prefer.

JavaScript

```
var myChart = new Chart(ctx, {
    type: 'line',
    data: {
```

```
        labels: ["Tokyo",      "Mumbai",       "Mexico City", "Shanghai",
        "Sao Paulo",    "New York",     "Karachi","Buenos Aires",
        "Delhi","Moscow"],
        datasets: [{
            label: 'Series 1', // Name the series
            data: [500,        50,     2424,   14040,  14141,  4111,   4544,
        47,     5555, 6811], // Specify the data values array
            fill: false,
            borderColor: '#2196f3', // Add custom color border (Line)
            backgroundColor: '#2196f3', // Add custom color background
(Points and Fill)
            borderWidth: 1 // Specify bar border width
        }]},
    options: {
      responsive: true, // Instruct chart js to respond nicely.
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
    }
});
```

Notice the only real change is the `type` option, this has been specified as 'line'. That simple change, will instruct the library to render you a line chart. Let's see the demo -



Multiple Series Line Chart

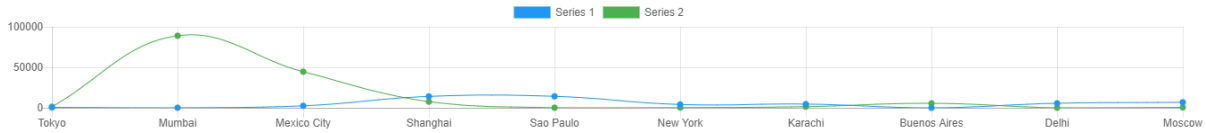
Whilst we're on the Line Chart section, you may as well know how easy it is to have multiple series on your graph. All is required is to multiply your data set and add a new label for that series. In reality, this can be done as many times as you have data-sets, as long as the indexes match, your good to go.

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

var myChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: ["Tokyo",      "Mumbai",       "Mexico City", "Shanghai",
        "Sao Paulo",    "New York",     "Karachi","Buenos Aires",
        "Delhi","Moscow"],
        datasets: [{
            label: 'Series 1', // Name the series
            data: [500,        50,     2424,   14040,  14141,  4111,   4544,
        47,     5555, 6811], // Specify the data values array
            fill: false,
            borderColor: '#2196f3', // Add custom color border (Line)
            backgroundColor: '#2196f3', // Add custom color background
(Points and Fill)
            borderWidth: 1 // Specify bar border width
        },
                {
```

```
            label: 'Series 2', // Name the series
            data: [1288,       88942,  44545,  7588,   99,     242,    1417,
        5504,   75, 457], // Specify the data values array
            fill: false,
            borderColor: '#4CAF50', // Add custom color border (Line)
            backgroundColor: '#4CAF50', // Add custom color background
(Points and Fill)
            borderWidth: 1 // Specify bar border width
        }]
    },
    options: {
      responsive: true, // Instruct chart js to respond nicely.
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
    }
});
```

As you can see, the only change here is the additional data-set, labelled Series 2. You could have these data-sets in separate variables and pass them in, or do it like I have above, it's completely up to you. Here's the multi-series line chart in action –



Create A Radar Chart

Radar charts usually only come in handy for extra-visual understanding of data, but nevertheless, they are beautiful and extremely good at how they visualize data. In this following example, we will store the Chart objects options and data-sets in different variables, passing them into the object call when required.

From the example, you will see that just like other charts within this tutorial, there isn't much difference in terms of code, just a couple of options that need to be adjusted.

HTML

Note the small addition of the `width` inline-style, this is important as the default render of the radar chart is barely understandable due to its small size.

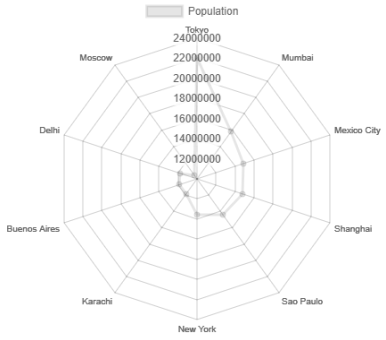
```
<canvas width="750" id="myChart"></canvas>
```

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

// Define the data
var data = [22006299, 15834918, 14919501, 14797756, 14433147, 13524139,
11877109, 11862073, 11779606, 10452000]; // Add data values to array

var labels = ["Tokyo", "Mumbai", "Mexico City", "Shanghai", "Sao Paulo",
"New York", "Karachi", "Buenos Aires", "Delhi", "Moscow"]; // Add labels to
array
```

```
// End Defining data
var options = {responsive: true, // Instruct chart js to respond nicely.
    maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
};

// End Defining data
var myChart = new Chart(ctx, {
    type: 'radar',
    data: {
        labels: labels,
        datasets: [{
                label: 'Population', // Name the series
                data: data, // Specify the data values array
          fill: false
            }]
    },
    options: options
});
```

Note the only change here is the `type` option which is specified as 'radar'. See the image below to see the result of the above code.



Create An Area Chart

Area charts are **not a chart-type of their own**, they are achieved by using the `fill` option. Two chart types support this option, the Line chart and the Radar chart. This may seem some sort of cheat, but it works resiliently to be fair. This is good news for the amount of knowledge needed to create them though, we've already been through radar and line charts, so it's just simply a case of adding extra options.

So, with this in mind, what are the particular options?

There are two options to note, one is the `fill` option which is part of the `datasets` object. The other is the `propagate` Boolean option that is placed within the options object.

If you haven't already noticed yet, `fill` has already been used in a few of the examples in this tutorial, namely the line and radar chart examples. This option was set to **false** within the code to stop the 'area' filling.

Let's use the code in the Multi Series Line Chart example again and slightly change it.

HTML

```
<canvas id="myChart"></canvas>
```

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

var myChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: ["Tokyo",      "Mumbai",       "Mexico City", "Shanghai",
        "Sao Paulo",    "New York",     "Karachi","Buenos Aires",
        "Delhi","Moscow"],
        datasets: [{
            label: 'Series 1', // Name the series
            data: [500,         50,     2424,   14040,  14141,  4111,   4544,
        47,     5555, 6811], // Specify the data values array
            fill: true,
            borderColor: '#2196f3', // Add custom color border (Line)
            backgroundColor: '#2196f3', // Add custom color background
(Points and Fill)
            borderWidth: 1 // Specify bar border width
        },
                {
            label: 'Series 2', // Name the series
            data: [1288,        88942,  44545,  7588,   99,     242,    1417,
        5504,   75, 457], // Specify the data values array
            fill: true,
            borderColor: '#4CAF50', // Add custom color border (Line)
            backgroundColor: '#4CAF50', // Add custom color background
(Points and Fill)
            borderWidth: 1 // Specify bar border width
        }]
    },
    options: {
      responsive: true, // Instruct chart js to respond nicely.
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
    }
});
```

The only change in the above JavaScript that was made was the `fill` option switched to **true**. It's also worth noting here that if you completely remove the `fill` option, the chart will default to **true**, and will fill the area on render.

This gives us the following result, a fully functional Area Chart -

Create A Polar Area Chart

I've never had the need to use a Polar Area chart in a production environment before, but I have to admit they're pretty cool. Somewhat similar to a radar chart with a slight difference with the scaling. This particular chart has some nice animation effects too, which are showcased in the working example. I've purposely reduced the sample data so that the legend fits better to the demo, just in-case you were wondering why it was missing.

HTML

```
<canvas id="myChart"></canvas>
```

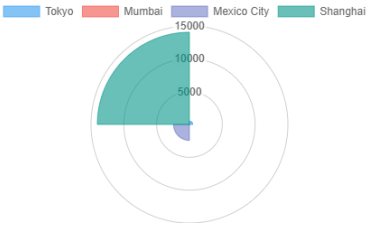
JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

var myChart = new Chart(ctx, {
    type: 'polarArea',
    data: {
        labels: ["Tokyo",      "Mumbai",       "Mexico City", "Shanghai"],
        datasets: [{
            data: [500,        50,     2424,   14040], // Specify the data
values array

            borderColor: ['#2196f38c', '#f443368c', '#3f51b570',
'#00968896'], // Add custom color border (Line)
            backgroundColor: ['#2196f38c', '#f443368c', '#3f51b570',
'#00968896'], // Add custom color background (Points and Fill)
            borderWidth: 1 // Specify bar border width
        }]},
    options: {
      responsive: true, // Instruct chart js to respond nicely.
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
      animation: {
                                      animateRotate: true,
                                      animateScale: true
                                  }
    }
});
```

Some notable points for this chart type -

- Specifying individual border and background colors is important for visual management.
- Giving these colors some opacity is also pretty important if you want to visualize the full chart scale and labels.
- Using the `animation` option with its sub-options of `animateRotate` and `animateScale` which are both Boolean types. These will instruct the chart to animate on render.

Here is the above code rendered



Create A Pie Chart

Another simple chart to set-up is the ever-popular Pie Chart, the simplicity of this chart is beautiful, as-long as you have the right data you can switch seamlessly from another chart by changing the `type` to **pie**. So, let's use the same code from the Polar Area chart and change the type to 'pie' and removing the animation options, as the Pie Chart renders with a nice animation anyways.

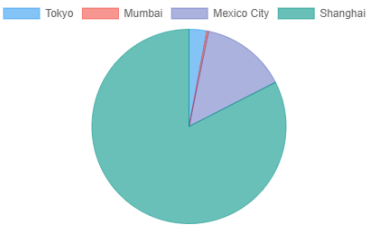
HTML

```
<canvas width="500" id="myChart"></canvas>
```

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

var myChart = new Chart(ctx, {
    type: 'pie',
    data: {
        labels: ["Tokyo",      "Mumbai",       "Mexico City", "Shanghai"],
        datasets: [{
            data: [500,        50,     2424,   14040], // Specify the data
values array

            borderColor: ['#2196f38c', '#f443368c', '#3f51b570',
'#00968896'], // Add custom color border
            backgroundColor: ['#2196f38c', '#f443368c', '#3f51b570',
'#00968896'], // Add custom color background (Points and Fill)
            borderWidth: 1 // Specify bar border width
        }]},
    options: {
      responsive: true, // Instruct chart js to respond nicely.
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
    }
});
```

The small changes made to the Polar Area chart as above will render the following Pie Chart perfectly –



Create A Doughnut Chart

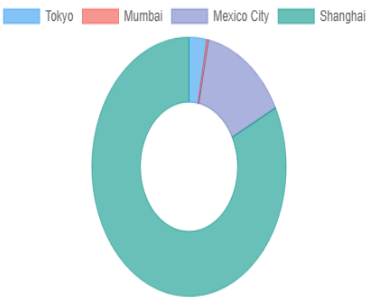
This example is almost identical to above, with the exception of the `type` being 'doughnut'. There literally isn't anything more to do to set it up in terms of code, which is pretty sweet. Here's the code and the demo.

HTML

```
<canvas width="500" id="myChart"></canvas>
```

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

var myChart = new Chart(ctx, {
    type: 'doughnut',
    data: {
        labels: ["Tokyo",      "Mumbai",       "Mexico City", "Shanghai"],
        datasets: [{
            data: [500,        50,     2424,   14040], // Specify the data
values array

            borderColor: ['#2196f38c', '#f443368c', '#3f51b570',
'#00968896'], // Add custom color border
            backgroundColor: ['#2196f38c', '#f443368c', '#3f51b570',
'#00968896'], // Add custom color background (Points and Fill)
            borderWidth: 1 // Specify bar border width
        }]},
    options: {
      responsive: true, // Instruct chart js to respond nicely.
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
    }
});
```



Create A Bubble & Scatter

I've purposely grouped these chart-types together due to the data-format requirement. The data format is in 'x' and 'y' coordinate style. Meaning X would be one numeric value and Y would be another numeric value. See the following JSON data example -

```
data: [{
                x: 5,
                y: 4
            }, {
                x: 2,
                y: 14
            }, {
                x: -2,
                y: 12

        }]
```

As you can see, there isn't nothing too complicated about it, you will just need to instruct the chart where you want the scattered data points to be placed. If you've already used scatter or bubble charts before, then you will already understand what I am talking about. If not, It will become clear in the chart demo's how these coordinates pan out.

Scatter Chart

Let's start with the Scatter chart, using the same canvas HTML element as all other examples.

HTML

```
<canvas width="500" id="myChart"></canvas>
```

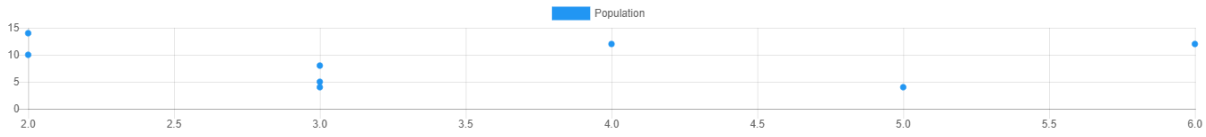
JavaScript

Note that the data within the data-set is randomly picked for this example -

```
var ctx = document.getElementById("myChart").getContext('2d');

// Define the data
var data = [{
                x: 5,
                y: 4
            }, {
                x: 2,
                y: 14
            },
            {
                x: 4,
                y: 12
            },
            {
                x: 2,
                y: 10
            },
```

```
            {
                x: 3,
                y: 4
            },
            {
                x: 3,
                y: 5
            },
            {
                x: 3,
                y: 8
            },
            {
                x: 6,
                y: 12

         }]; // Add data values to array
// End Defining data
var options = {responsive: true, // Instruct chart js to respond nicely.
    maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
};

// End Defining data
var myChart = new Chart(ctx, {
    type: 'scatter',
    data: {
        datasets: [{
                label: 'Population', // Name the series
                data: data, // Specify the data values array
          borderColor: '#2196f3', // Add custom color border
          backgroundColor: '#2196f3', // Add custom color background
(Points and Fill)
            }]
    },
    options: options
});
```

So, in reality, nothing changes much again apart from the `type` , the data format being X & Y coordinates and the complete removal of the `labels` array. See this code in action with the following image -



Bubble Chart

Bubble charts are essentially identical to a scatter chart, there is a small addition to the data required, though. The bubble radius which is identified as **r** within the JSON data. The **r** bubble radius value converts to pixels, which in turn grows or shrinks each data point as specified.

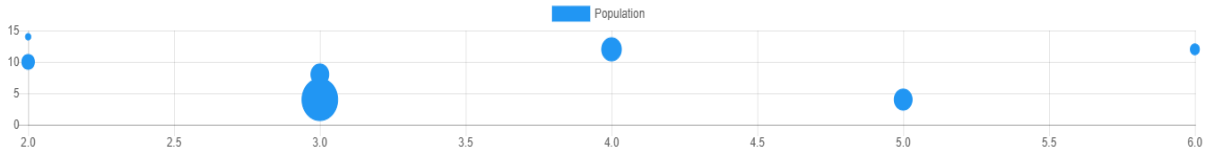
Important Note: You must specify the bubble `type` for ChartJS to consume the new **r** value within the data. The **r** value has no effect on a scatter chart.

JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

// Define the data
var data = [{
                x: 5,
                y: 4,
                r: 10
            }, {
                x: 2,
                y: 14,
                r: 3
            },
            {
                x: 4,
                y: 12,
                r: 11
            },
            {
                x: 2,
                y: 10,
                r: 7
            },
            {
                x: 3,
                y: 4,
                r: 20
            },
            {
                x: 3,
                y: 5,
                r: 2
            },
            {
                x: 3,
                y: 8,
                r: 10
            },
            {
                x: 6,
                y: 12,
                r: 5
         }]; // Add data values to array
// End Defining data
var options = {responsive: true, // Instruct chart js to respond nicely.
    maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
};

// End Defining data
var myChart = new Chart(ctx, {
    type: 'bubble',
    data: {
        datasets: [{
                label: 'Population', // Name the series
                data: data, // Specify the data values array
          borderColor: '#2196f3', // Add custom color border
          backgroundColor: '#2196f3', // Add custom color background
(Points and Fill)
            }]
```

```
    },
    options: options
});
```

Notice that the **data now has x, y and r**. Here's the code rendered -



Creating An Advanced Mixed Chart

So, we've went through each chart separately, but what if we want a line and bar chart in one? This type of chart allows that kind of functionality. The magic that defined the sub-chart-types is all defined with the `datasets` property. So, for instance if we have a bar chart, we specify that as the main type and subsequently define a line as one of the data's properties.

For the following example, I've made up some random data to have in each of the example lines that are rendered onto the bar chart.

HTML

```
<canvas width="500" id="myChart"></canvas>
```

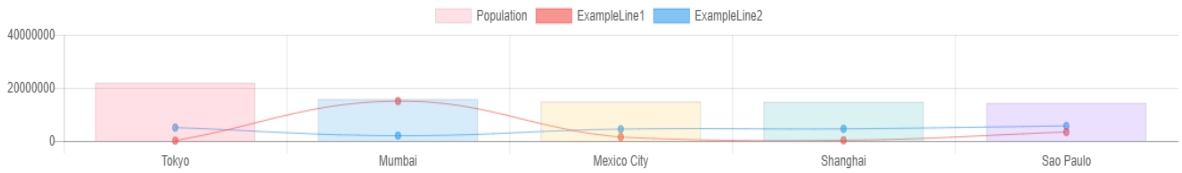
JavaScript

```
var ctx = document.getElementById("myChart").getContext('2d');

// Define the data
var barTotalPopulationData = [22006299,       15834918,       14919501,
        14797756,       14433147]; // Add data values to array
var lineExample1 = [120000, 15000000, 1454210, 240124, 3358452];
var lineExample2 = [5024554, 2001424, 4454201, 4565420, 5659888];

var labels = ["Tokyo", "Mumbai",       "Mexico City", "Shanghai",     "Sao
Paulo"]; // Add labels to array
// End Defining data

// End Defining data
var myChart = new Chart(ctx, {
    type: 'bar',
    data: {
        labels: labels,
        datasets: [{
            label: 'Population', // Name the series
            data: barTotalPopulationData, // Specify the data values array
            backgroundColor: [ // Specify custom colors
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)'
            ],
```

```
            borderWidth: 1 // Specify bar border width
        },
                {
            label: 'ExampleLine1', // Name the series
            data: lineExample1, // Specify the data values array
            backgroundColor: '#f443368c',
            borderColor: '#f443368c',

            borderWidth: 1, // Specify bar border width
            type: 'line', // Set this data to a line chart
            fill: false
        },
                {
            label: 'ExampleLine2', // Name the series
            data: lineExample2, // Specify the data values array
            backgroundColor: '#2196f38c',
            borderColor: '#2196f38c',

            borderWidth: 1, // Specify bar border width
            type: 'line', // Set this data to a line chart
            fill: false
        }]
    },
    options: {
      responsive: true, // Instruct chart js to respond nicely.
      maintainAspectRatio: false, // Add to prevent default behaviour of
full-width/height
    }
});
```

Notice the `type` and `fill` options have been used in each of the line example's data. These instruct the library to render lines instead of bars. Yes, it's as simple as that!

Now let's see this graph rendered -



Summary

This super-streamlined library shows its capability of quickly switching chart-types seamlessly. If you haven't already thought of what I have, then this library is the perfect interface tool for users to be able to switch charts on-demand. With the exception of Scatters & Bubble charts due to the nature of the data requirement.

Within the tutorial, basic examples have been shown, don't get me wrong though, there is so much more to this library to be discovered and used. I may go through these in an 'advanced' tutorial in the future. For now, though, this article gives you the know-how to get started straight away. Hope it helps!