8 Playing with Google Charts

In this chapter we will cover:

- Getting started with a pie chart
- Creating charts using the ChartWrapper
- Changing data source to Google Spreadsheet
- Customizing chart properties with an options object
- Adding a dashboard to charts

Introduction

In this chapter, we will explore the Google visualization API task by task. We will look at the steps involved in creating a chart and integrating it with the charting API.

To work with the Google APIs, you must comply with the Google terms of use and policies that can be located at `https://google-developers.appspot.com/readme/terms`.

Getting started with a pie chart

In this first recipe, we will start with Google Charts, covering the basic steps that you need to understand when working with Google Charts through an interactive dataset that is based on the CDC death rates in the USA (LCWK)—deaths, percent of total deaths, and death rates for the 15 leading causes of death in five-year age groups, by race and sex in the United States in 2008.

Getting ready

We will start from scratch with an empty HTML file and an empty JavaScript file named `08.01.getting-started.html` and `08.01.getting-started.js`.

How to do it...

Let's list the steps required to complete the task starting with the HTML file:

1. Let's start by creating a `head` and linking it to the Google `jsapi` and our local JavaScript file:

   ```
   <!DOCTYPE html>
   <html>
     <head>
       <title>Google Charts Getting Started</title>
       <meta charset="utf-8" />
       <script src="https://www.google.com/jsapi"></script>
       <script src="./08.01.getting-started.js"></script>
     </head>
   ```

2. Then create an empty `div` with `id chart`:

   ```
     <body style="background:#fafafa">
       <div id="chart"></div>
     </body>
   </html>
   ```

   Now, it's time to move into the `08.01.getting-started.js` file.

3. Lets request the visualization API from the Google `jsapi`:

   ```
   google.load('visualization', '1.0', {'packages':['corechart']});
   ```

4. We want to add a `callback` that will be triggered when the library is ready:

   ```
   google.setOnLoadCallback(init);
   ```

5. Create an `init` function as follows:

   ```
   function init(){
   ..

   }
   ```

   From now on we will break down the code added within the `init` function:

6. Create a new Google data object and provide data sources as shown in the following code snippet:

```
data.addColumn('string', 'Type of Death');
data.addColumn('number', 'Deaths');
data.addRows([
        ['Diseases of heart', 616828],
        ['Malignant neoplasms', 565469],
        ['Chronic lower respiratory diseases', 141090],
        ['Cerebrovascular diseases', 134148],
        ['Accidents', 121902],
        ['Alzheimer\'s disease', 82435],
        ['Diabetes mellitus', 70553],
        ['Influenza and pneumonia', 56284],
        ['Suicide', 36035],
        ['Septicemia', 35927],
        ['Chronic liver disease and cirrhosis', 29963],
        ['Essential hypertension and hypertensive renal
        disease', 25742],
        ['Parkinson\'s disease', 20483],
        ['Homicide', 17826],
        ['All other causes', 469062]

]);
```

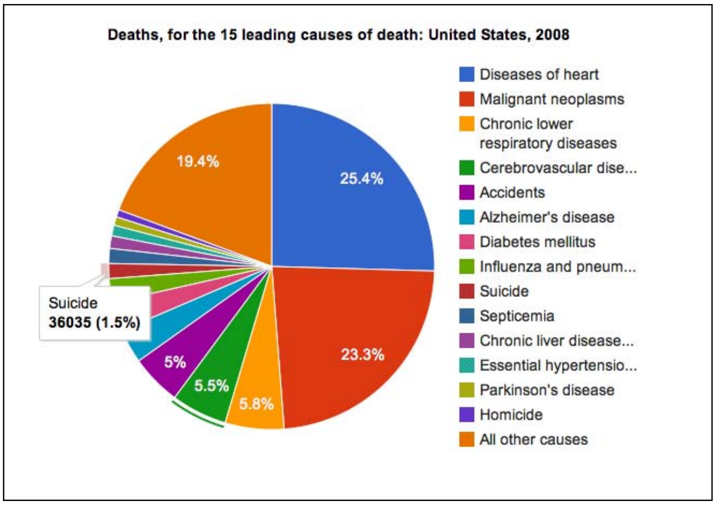
7. Create an `options` object for the chart:

```
var options = {'title':'Deaths, for the 15 leading causes of
death: United States, 2008',
                     'width':800,
                     'height':600};
```

8. Create and draw the chart by using the following code snippet:

```
var chart = new google.visualization.PieChart(document.
getElementById('chart'));
    chart.draw(data, options);
```

Load the HTML file. You will find a working, interactive chart as shown in the following screenshot:



How it works...

Let's explore the steps involved in working with Google Charts. The first step we establish when working with the Google API's is adding Google's API link into our HTML file:

```
<script src="https://www.google.com/jsapi"></script>
```

Now that the Google API is loaded into our application, we can request the library we wish to work with. In our case, we want to work with the visualization API and the `corechart` package:

```
google.load('visualization', '1.0', {'packages':['corechart']});
```

Notice that we are requesting version 1.0; this might be confusing but we are actually asking for the production chart, 1.0 is always the current production version. As such if you wanted to lock into a build, you would need to discover what its code version is and send it instead of the 1.0 stable build.

The `corechart` library in the example defines most basic charts. For charts that are not included, you would need to pass in the extra packages needed, such as the table chart:

```
google.load('visualization', '1.0', {'packages':['corechart','tab
le']});
```

This covers the basics of how to load the API. But before we can finish our loading process, we need a way to have a callback so that we can know when the library is available for us to manipulate:

```
google.setOnLoadCallback(init);
```

We are asking the Google API to let us know when the package has loaded in a similar way to how we added a callback to our document. When the API is loaded, it is time for us to start interacting with the charting API.

There are three components that you will probably want to explore in each Google Chart:

- Creating the data source
- Adding options to your chart
- Creating the chart

Let's explore all these options.

All Google Charts need a data source. The data source format is based on an internal object created through the charting API:

```
var data = new google.visualization.DataTable();
```

Data tables are 2D arrays (or tables). They have columns and rows just like databases. Our next step will be to define the data columns:

```
data.addColumn('string', 'Type of Death');
data.addColumn('number', 'Deaths');
```

In our case, as we are working with a pie chart, only two rows are needed—one to name our elements and the other to provide them with a value. There is only one mandatory parameter to the `addColumn` method to define the datatype . The datatype can be one of the following:

- `string`
- `number`
- `boolean`
- `date`
- `datetime`
- `timeofday`

The second parameter is an optional descriptor of the type of data and it is used for visualization such as in our case `10 Deaths`. There are other parameters too, but as long as we provide the elements in an ordered list, we do not need to explore them.

Last but not least, we will call the `addRows` method. We can call the `addRows` method and send a one-dimensional array (again in the same order of data as we set our `addColumn`). In our case, we are using the `addRows` method that expects a two-dimensional array:

```
data.addRows([
        ['Diseases of heart', 616828],
….
]);
```

This covers our datasets. As long as we set our columns in the order of our data and send our information via arrays, we are set and don't need to dig any deeper into the data API.

The `options` object enables us to create and modify the elements of our chart. The elements we control in our application are width, height, and our title.

After creating the data sources and setting the options for our array, it's time for the easy part. To create the chart our first step is to pick our chart type and define where it will be created. Then we render it with the data source and options:

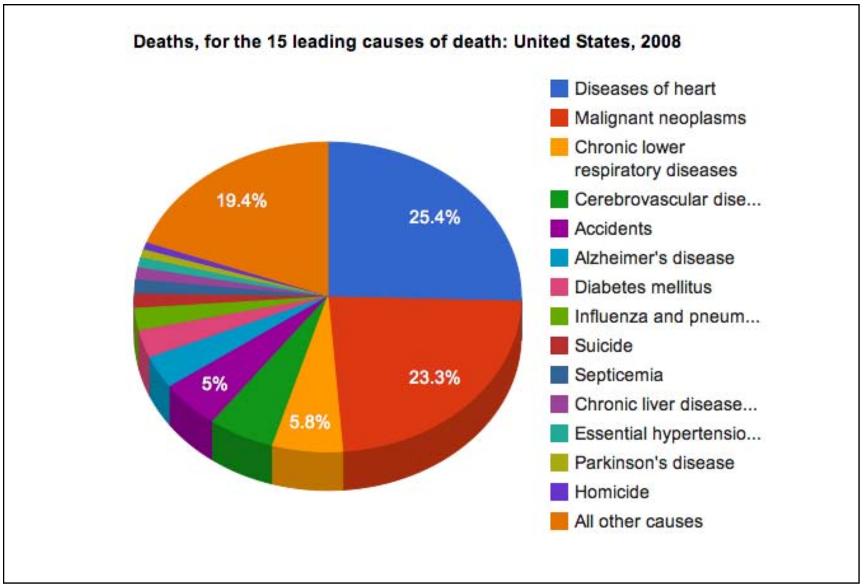
```
var chart = new google.visualization.PieChart(document.
getElementById('chart'));
chart.draw(data, options);
```

There's more...

Let's explore a few more tips, tricks, and advanced features of Google Charts. Using the option `Objectto create 3D chartsTo`, we can turn our chart into 3D. We can very quickly and simply add a new parameter into the options object:

```
var options = {'title':'Deaths, for the 15 leading causes of death:
United States, 2008',
                     'width':800,
                     'height':600,
                     "is3D": true};
```

The outcome would be a chart that is tilted in a 3D space.

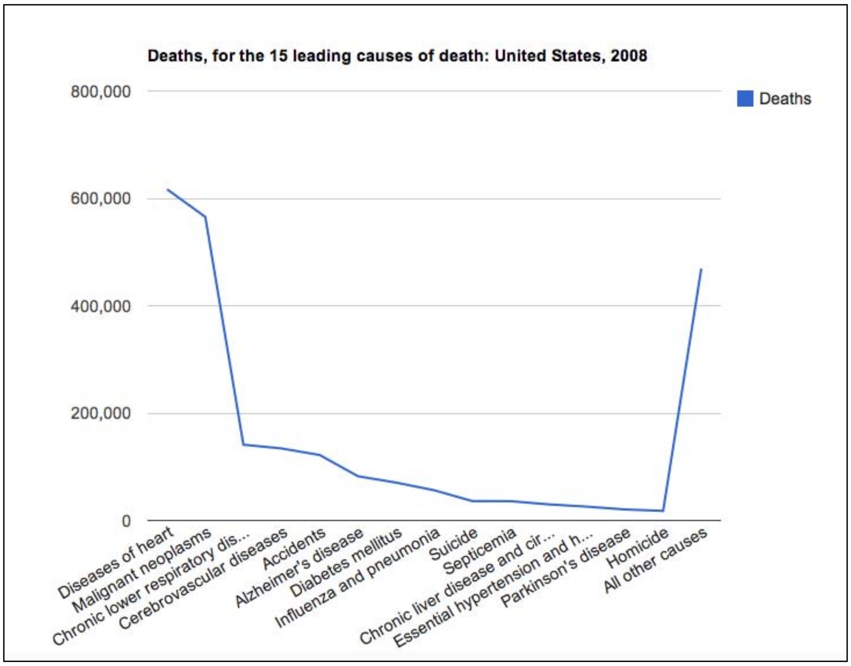


Changing the chart type

Changing a chart type isn't something complicated. As long as the chart types share the same number of data entries the change is usually one word from the actual constructor object of the chart. For example, we can very quickly switch our chart type by changing the method in the visualization library that is called:

```
var chart = new google.visualization.LineChart(document.
getElementById('chart'));
    chart.draw(data, options);
```

That would take the same data only rendered into a line chart (the `LineChart` object).



Creating charts using the ChartWrapper

There are two ways to create charts with Google Charts. One is the way we did it in the recipe *Getting started with a pie chart* and the second will be covered in this recipe. The goal of the ChartWrapper object is to enable you to cut down the amount of code needed to create a chart.

Its main advantages are less code and more flexibility of data sources. Its disadvantage is less control over the steps of graph creation.

Getting ready

Grab the HTML file from the last recipe (*Getting started with pie charts*). We will only modify the file path of the external JavaScript file and the rest of the code will remain the same.

How to do it...

After changing the path of the HTML file source to the JavaScript file, it's time to go into the JavaScript file and start over:

1. Load Google API (you do not need to mention what you want to load any more) and add a callback:

   ```
   google.load('visualization', '1.0');
   google.setOnLoadCallback(init);
   ```

2. Create the `init` function:

   ```
   function init(){
   …
   }
   ```

3. Build a 2D array with the data source:

   ```
   var dataTable = [
           ['Type of Death','Deaths'],
           ['Diseases of heart', 616828],
           ['Malignant neoplasms', 565469],
           ['Chronic lower respiratory diseases', 141090],
           ['Cerebrovascular diseases', 134148],
           ['Accidents ', 121902],
           ['Alzheimer\'s disease ', 82435],
           ['Diabetes mellitus', 70553],
           ['Influenza and pneumonia', 56284],
           ['Suicide', 36035],
           ['Septicemia', 35927],
           ['Chronic liver disease and cirrhosis', 29963],
           ['Essential hypertension and hypertensive renal
           disease', 25742],
           ['Parkinson\'s disease', 20483],
           ['Homicide', 17826],
           ['All other causes', 469062]
         ];
   ```

4. Create the `options` object:

   ```
   var options = {'title':'Deaths, for the 15 leading causes of
   death: United States, 2008',
                        'width':800,
                        'height':600,
                        "is3D": true};
   ```

5. Build and render the chart:

   ```
   var chart = new google.visualization.ChartWrapper({
     chartType:'PieChart',
     dataTable:dataTable,
     options:options,
     containerId:'chart'

   });
   chart.draw();
   ```

You've completed the creation of this chart type. Refresh your screen and you will see the same chart as in the last example, only using less code.

How it works...

The nice thing about this example is you don't need to know much more about how it works. The `ChartWrapper` function itself deals with all the information that you've had to deal with in the last recipe. With that said, it doesn't mean this way is always the better way—if you need more control over the steps, the last example would work better.

There's more...

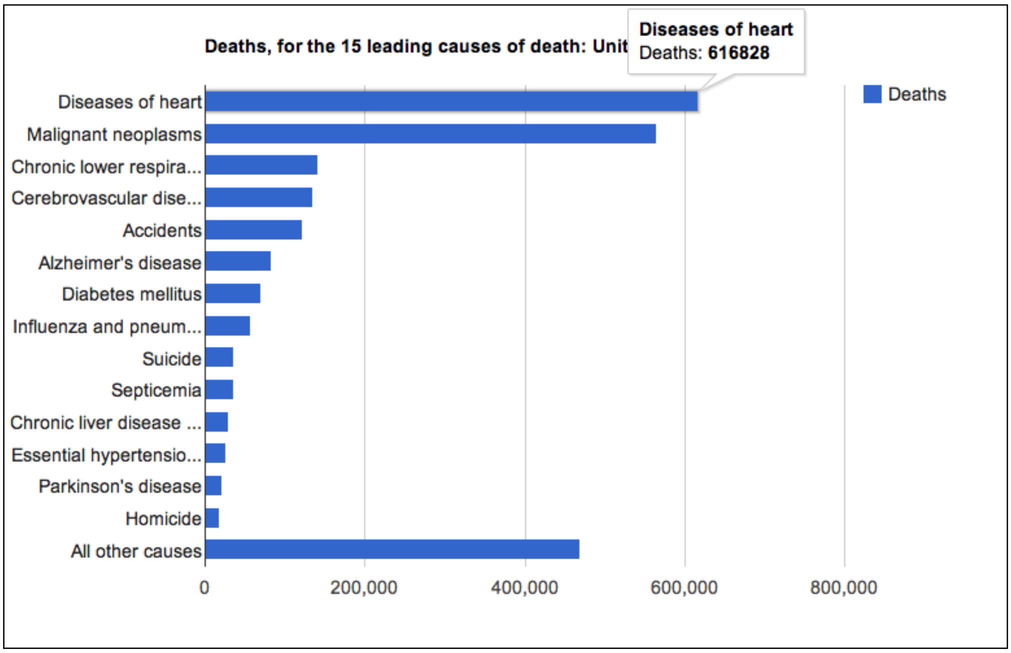
As this recipe was very easy, let's add an extra pointer.

Changing a chart in one line

It's really easy changing between the types of views of the Google Chart API. All you need to do is switch the type. Let's change our chart to a `BarChart`:

```
var chart = new google.visualization.ChartWrapper({
  chartType:'BarChart',
  dataTable:dataTable,
  options:options,
  containerId:'chart'

});
```

Refresh your window and you will find a bar chart.



Changing data source to Google Spreadsheet

One of the powerful features of working with the Google API is the deep relationship between the product lines. In this recipe, based on the last recipe, we will create a Google Spreadsheet and then integrate it into our application.

Getting ready

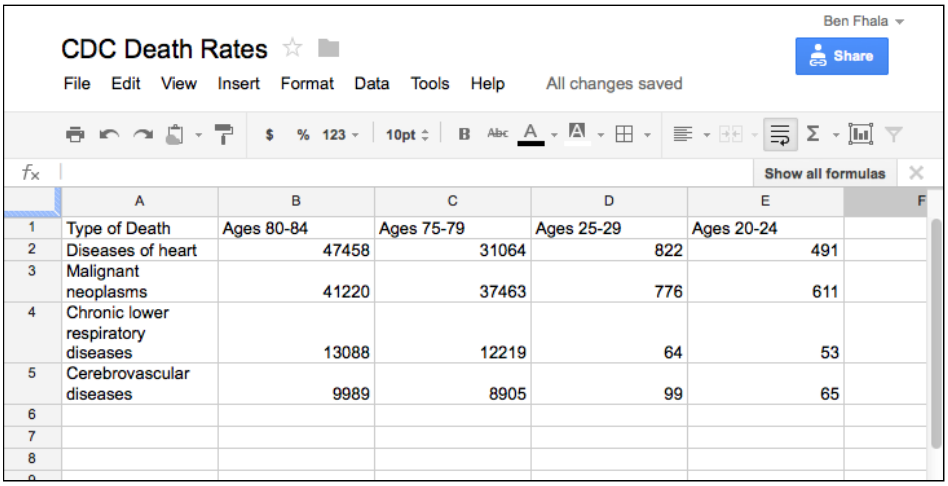
Have a copy around you of the source files from the last recipe (*Creating charts using the ChartWrapper*).

How to do it...

The steps involved with creating a new Google document are simple, but are needed to be able to integrate our work; as such we will run through them quickly.

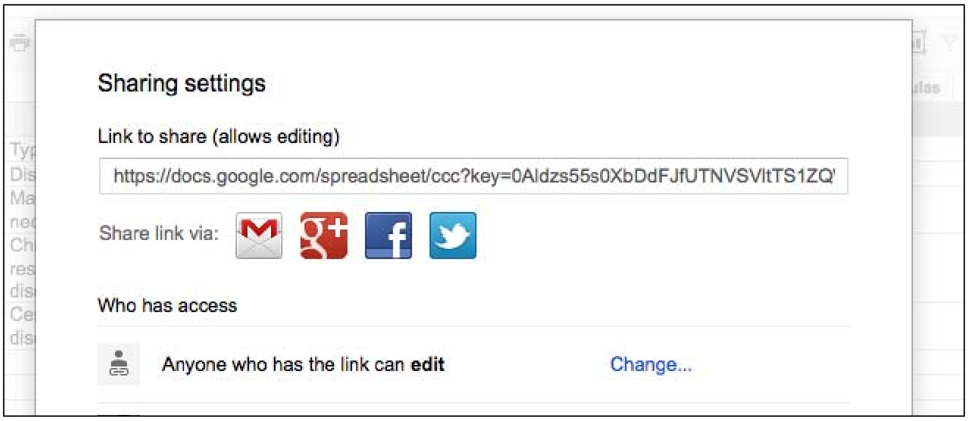
1. Go to `http://drive.google.com/` (formally known as Google Docs) and register/login.

2. Create a new spreadsheet.
3. Add data to the spreadsheet.



	A	B	C	D	E
1	Type of Death	Ages 80-84	Ages 75-79	Ages 25-29	Ages 20-24
2	Diseases of heart	47458	31064	822	491
3	Malignant neoplasms	41220	37463	776	611
4	Chronic lower respiratory diseases	13088	12219	64	53
5	Cerebrovascular diseases	9989	8905	99	65
6					
7					
8					

4. Click on the **Share** button and set the view to public:


Sharing settings

Link to share (allows editing)

https://docs.google.com/spreadsheet/ccc?key=0Aldzs55s0XbDdFJfUTNVSVltTS1ZQ

Share link via:

Who has access

Anyone who has the link can **edit** Change...

5. Create an API URL based on the document ID:
   - **Document link**:

     `https://docs.google.com/spreadsheet/ccc?key=0Aldzs55s0XbDdFJfUTNVSVltTS1ZQWQ0bWNsX2xSbVE`

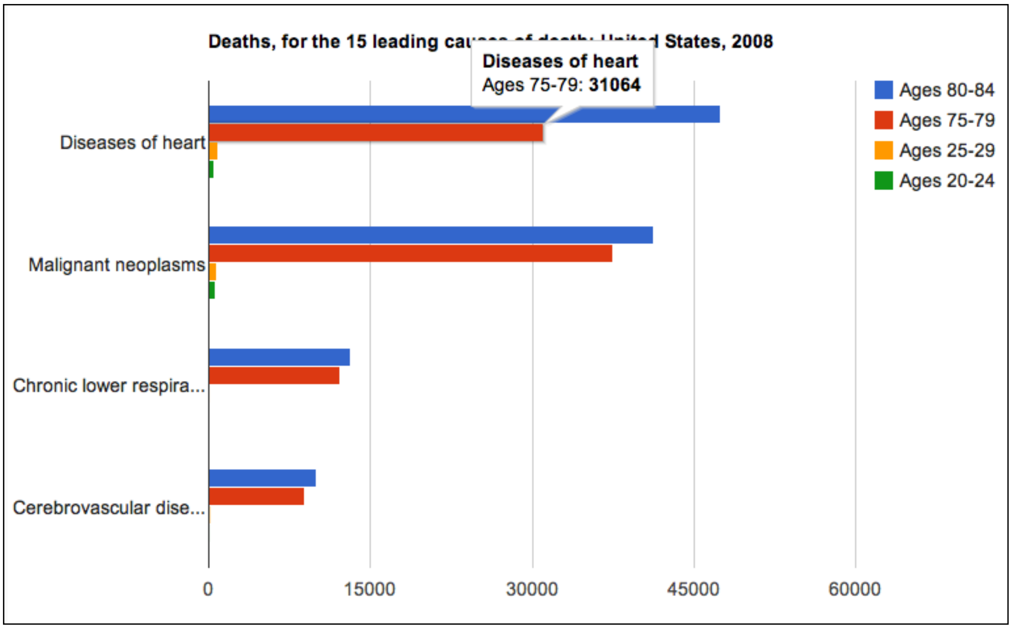
   - **API link**:

     `https://spreadsheets.google.com/tq?key=0Aldzs55s0XbDdFJfUTNVSVltTS1ZQWQ0bWNsX2xSbVE`

6. Now, it's time to get into our JavaScript file, and delete the current data source and replace it with a URL feed:

```
google.load('visualization', '1.0');

google.setOnLoadCallback(init);

function init(){
  var options = {'title':'Deaths, for the 15 leading
  causes of death: United States, 2008',
                   'width':800,
                   'height':600};
  var chart = new google.visualization.ChartWrapper({
    chartType:'BarChart',
    dataSourceUrl:"https://spreadsheets.google.com/
    tq?key=0Aldzs55s0XbDdFJfUTNVSVltTS1ZQWQ0bWNsX2xSbVE",
    options:options,
    containerId:'chart'

  });
  chart.draw();
}
```

Amazing! See how little code we needed to create a rich and fully interactive chart:



How it works...

This is really the amazing part about it. You just don't need to understand how it works, all you need to do is create your chart and use the steps provided in the preceding section, and you can convert any of your own spreadsheets into a Google Spreadsheet.

The most important step in the preceding steps is step 4. Notice that the URL that is generated through the Google Documents (Google Drive) is not the same as the URL that we need to hit when working in code. This is because the first URL is intended to be rendered as a visual page, while the second link generates a new Google data object. Don't forget that every page has its own unique ID.

There's more...

If you have a bit of a background with working with databases, you can send simple SQL queries into the data source and only get the items that you want to view. Let's say in our example we want to get the items in a different order, exclude column B, and sort based on column D (by age):

```
SELECT A,E,D,C ORDER BY D
```

Our `Select` statement is listing out what we want to select. The `ORDER BY` statement is self-explanatory. Let's add it to our code:

```
var chart = new google.visualization.ChartWrapper({
  chartType:'BarChart',
  dataSourceUrl:"https://spreadsheets.google.com/
  tq?key=0Aldzs55s0XbDdFJfUTNVSVltTS1ZQWQ0bWNsX2xSbVE",
  query: 'SELECT A,E,D,C ORDER BY D',
  options:options,
  containerId:'chart'

});
```

When you refresh your code, column B will be missing and the data will be organized based on column D.

Last but not least, add this to your code:

```
var chart = new google.visualization.ChartWrapper({
  chartType:'BarChart',
  dataSourceUrl:"https://spreadsheets.google.com/
  tq?key=0Aldzs55s0XbDdFJfUTNVSVltTS1ZQWQ0bWNsX2xSbVE",
  query: 'SELECT A,E,D,C ORDER BY D',
  refreshInterval: 1,
  options:options,
  containerId:'chart'

});
chart.draw();
```

Now go back to the public chart and change the data in it. You will see that it will automatically update the chart.

Customizing the chart properties with an options object

In this recipe, we will create a new chart with Google Charts API—a candlestick chart—and we will incorporate a variety of configurations into it.

Getting ready

We will start with a clean slate by creating a fresh new JavaScript and an HTML file.

How to do it...

Most of the steps will look almost identical to the past recipes in this chapter. Our main focus will be on our `options` parameters:

1. Create an HTML file and link it to a JavaScript file (in our case `08.04.candlestick.js`):

   ```
   <!DOCTYPE html>
   <html>
     <head>
       <title>Google Charts Getting Started</title>
       <meta charset="utf-8" />
       <script src="https://www.google.com/jsapi"></script>
       <script src="./08.04.candlestick.js"></script>
     </head>
     <body style="background:#fafafa">
       <div id="chart"></div>
     </body>
   </html>
   ```

2. In the `08.04.candlestick.js` file, add the API `load` and `callback` functions:

   ```
   google.load('visualization', '1', {packages: ['corechart']});
   google.setOnLoadCallback(init);

   function init(){
   ```

3. In the `init` function (from now until the end of this recipe we will remain in the `init` function), create a new `DataTable` object by using the `google.visualization.arrayToDataTable` method:

   ```
   var data = google.visualization.arrayToDataTable([
     ['Mon', 10, 24, 18, 21],
     ['Tue', 31, 38, 55, 74],
     ['Wed', 50, 55, 20, 103],
     ['Thu', 77, 77, 77, 77],
     ['Fri', 68, 66, 22, 15]
   ], true);
   ```

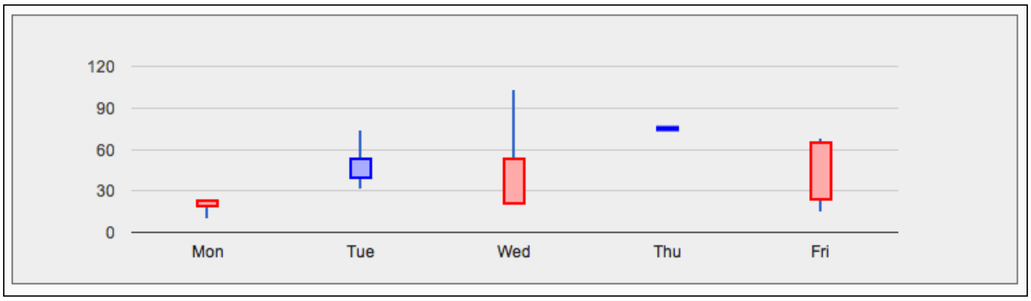
4. Create an `options` object (a configuration object) for the chart:

   ```
   var options = {
     legend:'none',
     backgroundColor:{fill:'#eeeeee',strokeWidth:2},
     bar:{groupWidth:17},
     candlestick:{hollowIsRising:true,
       fallingColor:{stroke:'red',fill:'#ffaaaa'},
       risingColor: {stroke:'blue',fill:'#aaaaff'}
     },
     enableInteractivity:false

   };
   ```

5. Draw the chart by using the following code snippet:

   ```
   var chart = new google.visualization.CandlestickChart
   (document.getElementById('chart'));
   chart.draw(data, options);

   }
   ```

When you load the HTML file, you will discover a customized candlestick chart, as shown in the following screenshot:



How it works...

This is the first time that we have used the method `google.visualization.arrayToDataTable`. This method takes in an array and returns a data table. When the second parameter of this method is set to `true`, it will treat the first row in the array as part of the data; and otherwise it will be treated as header data.

There are many options and for a full list of them, review Google Charts documentation. We will focus on the items that we have picked to modify our view. The Google charts enable you to send an object with parameters. Each chart type has a different set of options. In our case, we have many options that enable us to control the details of how our chart looks. Most of the options are style related:

```
backgroundColor:{fill:'#eeeeee',strokeWidth:2},
  bar:{groupWidth:17},
  candlestick:{hollowIsRising:true,
   fallingColor:{stroke:'red',fill:'#ffaaaa'},
  risingColor: {stroke:'blue',fill:'#aaaaff'}
  },
```

Some options directly relate to the function such as disabling the legend:

```
legend:'none',
```

Or disabling interactive elements:

```
enableInteractivity:false
```

There's more...

The main goal of highlighting this element is not because it's difficult, but because it's easy, and it is the main place where you would find yourself making changes to the charts. One point to note is that it is really important to check that you can do what you need by using Google Charts before working with them, as contrary to other chart systems, you can't go into their source files and change them, as we did in the recipes in *Chapter 7*, *Depending on the Open Source Sphere*.

Adding a dashboard to charts

In this last recipe of this chapter we will add live controllers that will enable the users to change the filtering of data to see less or more information.

Getting ready

We will start from scratch so nothing to worry about.

How to do it...

The following are the steps needed to create a basic dashboard controller:

1. Create an HTML file and link it to an external JavaScript file (in our case we will use the file `08.05.slider.js`):

   ```
   <!DOCTYPE html>
   <html>
     <head>
       <title>Google Charts DASHBOARD</title>
       <meta charset="utf-8" />
       <script src="https://www.google.com/jsapi"></script>
       <script src="./08.05.slider.js"></script>
     </head>
     <body style="background:#fafafa">
       <div id="chart"></div>
       <div id="dashboard"></div>
       <div id="filter"></div>
     </body>
   </html>
   ```

2. Now, it's time to get into `08.05.slider.js` and to load the Google Visualization API. This time around we will load in the controller package:

   ```
   google.load('visualization', '1', {packages: ['controls']});
   ```

3. Now, it's time to add a callback:

   ```
   google.setOnLoadCallback(init);
   function init(){
   ```

4. Let's create our data source. We will base it on CDC death rates for 2008:

   ```
   var data = google.visualization.arrayToDataTable([
       ['Age (+- 2 Years)', 'Deaths'],
           [2, 4730],
           [7, 2502],
           [12, 3149],
           [17, 12407],
           [22, 19791],
           [27,20786],
           [32,21489],
           [37,29864],
           [42,46506],
           [47,77417],
   ```

```
            [52, 109125],
            [57,134708],
            [62,161474],
            [67,183450],
            [72,218129],
            [77,287370],
            [82,366190],
            [87,372552],
            [92,251381],
             [100,20892],
        ]);
```

5. Then create a new dashboard:

```
var dashboard = new google.visualization.Dashboard(document.
getElementById('dashboard'));
```

6. Let's create a slider and provide it with the information it needs to connect to the data source:

```
  var slider = new google.visualization.ControlWrapper({
    containerId: 'filter',
    controlType: 'NumberRangeFilter',
    options: {
    filterColumnLabel: 'Age (+- 2 Years)'
  }
});
```

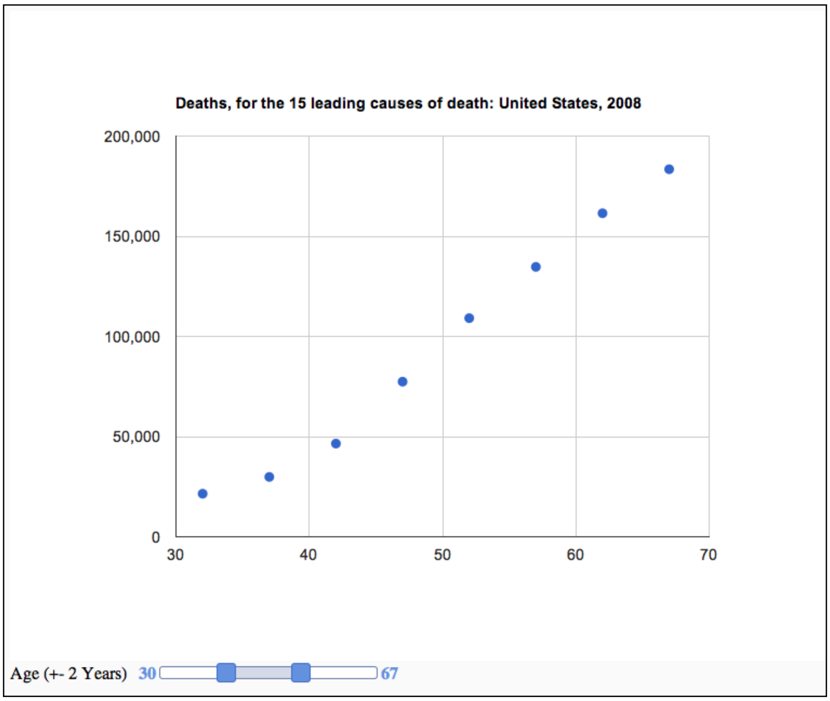
7. Create a chart:

```
var chart = new google.visualization.ChartWrapper({
  chartType: 'ScatterChart',
  containerId: 'chart',
  options: {
    legend: 'left',
    title:'Deaths, for the 15 leading causes of death:
    United States, 2008',
    width: 800,
    height: 600

  }
});
```

8. Last but not least, it's time to bind and draw our controller:

   ```
   dashboard.bind(slider, chart).draw(data);
   }
   ```

Load the HTML file and you will discover a scatter chart with a controller that enables you to select the age range that you want to dig deeper into.


Deaths, for the 15 leading causes of death: United States, 2008
200,000
150,000
100,000
50,000
0
30
40
50
60
70
Age (+- 2 Years)
30
67

How it works...

This is probably one of the smoothest parts of working with the Google Charting API. So let's break down and figure out the steps involved in creating controllers for your chart. We will showcase one controller, but the same logic flow would work for all components.

First in our HTML file, we need to have a `div` layer with an ID associated for our dashboard and a `div` for each following controller. To add controllers we assign them to the dashboard. We start with creating a dashboard:

```
var dashboard = new google.visualization.Dashboard(document.
getElementById('dashboard'));
```

This dashboard is now going to be our hub where we connect all of our controllers (in our case, one controller). Then, we will create the next controller; in our case, we want to use a slider:

```
var slider = new google.visualization.ControlWrapper({
  containerId: 'filter',
  controlType: 'NumberRangeFilter',
  options: {
    filterColumnLabel: 'Age (+- 2 Years)'
  }
});
```

Notice that we are adding a control type to get our range slider and we are linking it to a column by giving it the column ID (the label in the first row).

We continue and create a chart in the same way as before. In this case we picked a scatter chart. The order here isn't important, but the most important part left is to link between our controller and the chart. We do that by using the `dashboard.bind` method:

```
dashboard.bind(slider, chart);
```

Then, we draw our element as our dashboard returns itself when a `bind` function is created:

```
dashboard.bind(slider, chart).draw(data);
```

If we want, we can split this into separate lines as follows:

```
dashboard.bind(slider, chart);
dashboard.draw(data);
```

And there you go! Now you know how to work with dashboards. These steps are critical, but you can now add any controller. The rest of the documentation for this product is self-explanatory.