Data Processing Assignments Weeks 4-7

Thijs Coenen

September 10, 2015

Contents

Introduction

This reader contains all assignments for Data Processing that are about JavaScript. JavaScript is the only programming language that is supported by all modern web browsers and it is the least intrusive way of adding interactive elements to a web page as it requires no plug-ins or other special extensions. We will be using JavaScript to implement a number of interactive visualizations.

For this course we will be using the book *Eloquent JavaScript* by Marijn Haverbeke. This book assumes very little programming background but goes into sufficient detail for our purposes. It is furthermore available freely from the publisher's web site.

Modern web browsers support two ways of creating graphics, the `canvas`-element provides raster graphics and Scalable Vector Graphics (SVG) provide vector graphics. Raster graphics are specified as a grid of colored image elements and vector graphics are specified as a collection of shapes. Raster graphics become blurry (or show large pixels) when they are enlarged while vector graphics remain sharp. We will use both techniques to create visualizations.

JavaScript

The assignments of week 4 and 5 are to create an interactive graph of the temperature at the De Bilt weather station. The data for these assignments can be freely downloaded from the KNMI (the Royal Dutch Meteorological Institute) web pages.

We will implement this plot using JavaScript and the HTML 5 `canvas`-element. The `canvas`-element provides an API with which graphics can be created using JavaScript. You will furthermore need a little bit of Python to convert the downloaded data to CSV (Comma Separated Format) file.

The assignment is split into two parts to be handed in over two weeks. The first week we will acquire the raw data, convert it to a usable CSV format (using Python), include it in a web page and write the JavaScript to show a simple line graph. The second week we will extend the graph to support interactivity.

2.1 Reading Assignment and Resources

Read chapters 1 through 5 and chapters 12 and 13 of *Eloquent JavaScript*. This book by Marijn Haverbeke is freely available from the book's web site[1] The first few chapters should be a quick read and the later chapters explain how JavaScript integrates in the browser and how the Document Object Model (DOM) is used from JavaScript. The Mozilla Developer Network[2] (MDN) contains up-to-date information on client-side web technologies (i.e. about HTML, CSS and JavaScript) and can serve as a reference.

2.1.1 Questions

Answer the following questions in your own words. This assignment will only be graded *pass* or *fail*.

1. Explain the difference between the `==` operator and the `===` operator.
2. Explain what a closure is. (Note that JavaScript programs use closures very often.)
3. Explain what higher order functions are.
4. Explain what a query selector is and give an example line of JavaScript that uses a query selector.

2.1.2 Acquiring the Weather Data

Visit KNMI webpage[3] that allows you to download raw weather station data in CSV format. Select the De Bilt weather station, select the maximum temperature records and choose a year for which you want to create a temperature graph. Download the data and verify that you have in fact one full year's worth of data (hint: CSV files can be viewed in text editors). The downloaded CSV file starts with a so-called *header* that contains meta-data. This meta-data allows you to correctly interpret the raw data in the rest of the file.

[1] `http://eloquentjavascript.net/`

[2] `https://developer.mozilla.org`

[3] `http://www.knmi.nl/climatology/daily_data/selection.cgi`

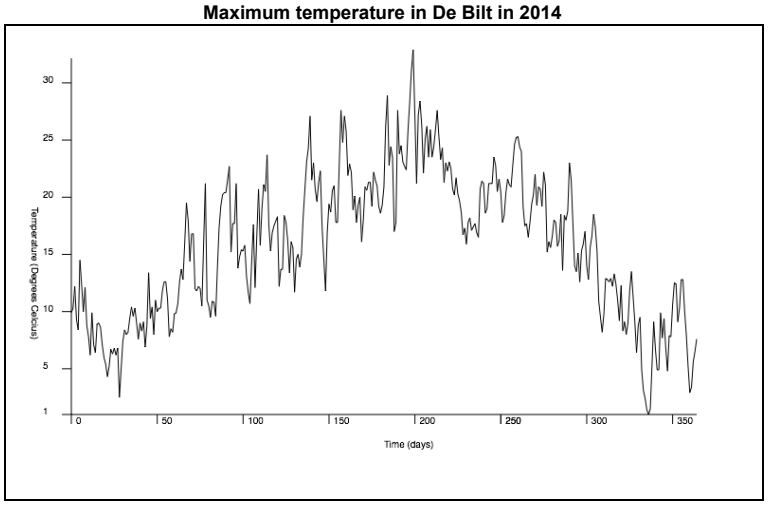

Figure 2.1: An example temperature graph as created by Jenny Hasenack.

2.1.3 Reformatting and Loading the Data

For this assignment it is ok to add any JavaScript you write inside a script tag at the very end of the HTML body. Handing in your code as a separate script is also fine, but make sure that all the needed files are included.

To keep the complexity of loading the data low, we will not load it from an external file but embed it directly in the web page. Create an empty web page and add a `textarea`-element to it. Normally these text areas are used in forms to get input from a user, but we will use it to embed our data. Reformat the CSV file such that it only contains the dates and the maximum temperatures. This will result in something like the following snippet (without the ellipsis of course).

```
<textarea id="rawdata">
#date,maxtemp
2014/01/01,   99
2014/01/02,  103
2014/01/03,  122
2014/01/04,   93
2014/01/05,   84
2014/01/06,  145
...
</textarea>
```

You can now load the data by writing some JavaScript that selects the text area (`document.getElementById(...)`) and then accesses its content. The content should by split into lines and then further split into two chunks, one for the date and one for the maximum temperature. Now create an array of data points and use `console.log(...)` to see whether the data was in fact loaded. Before moving on make sure that the dates are in fact JavaScript dates and the numbers JavaScript numbers. To convert date strings to JavaScript dates use the `Date` function: `new Date('2014/01/01')` (make sure that the date string matches the example's formatting).

2.1.4 The `canvas`-element

The HTML 5 canvas element allows JavaScript to draw pictures, the Mozilla Developer Network has a small tutorial[4] on the `canvas`-element that you should read. For this assignment you will not need to know all the details of the canvas but you should be able to at least draw lines, rectangles, circles and text (and how to rotate text).

2.1.5 Transforming the data to screen coordinates

The `canvas`-element provides its own coordinate system, you will have to transform you raw data to these coordinates to draw the graph. Figure 2.1 shows you a very simple version of the plot you have to create for this week. The position encodings for this graph only need linear transforms, one for the x-axis and one for the y-axis, of the following form:

$$x_{\mathsf{SCREEN}} = f(x_{\mathsf{DATA}}) = \alpha \cdot x_{\mathsf{DATA}} + \beta.$$

Because finding the two constants α and β is a bit tedious we will create a function that can do it for us. We will use JavaScript's support for *closures* to create a function `createTransform` that calculates α and β and returns a transformation function. The following snippet of code demonstrates this technique, but you will have to implement the actual calculation yourself.

```
function createTransform(domain, range){
    \\ domain is a two-element array of the domain's bounds
    \\ range is a two-element array of the range's bounds
    \\ implement the actual calculation here
    var alpha = ...;
    var beta = ...;

    return function(x){
        return alpha * x + beta;
    };
}

\\ to use this for instance:
var tranform = createTransform([10, 20], [10, 20]);
console.log(transform(15));  // should log 15
```

To test this function you can make a transformation that transforms the domain $[10, 20]$ to the range $[10, 20]$ and see whether points are transformed to themselves. This function will work directly on your temperature data, but for the dates along the x-axis there is an extra complication. An example of a transformation (for either the x or y coordinate) is given in Figure 2.2).

The x-transform needs to deal with dates, and any calculations involving calendars tend to get complicated quickly. For this assignment it is ok to use the `Date.getTime()` method to change all date to milliseconds since January 1st 1970. These milliseconds can then be transformed into days since the start of your data (so the x-axis would run from 1 through 365 or 366 days depending on the year you chose).

2.2 Extra credit

- Create an x-axis that uses calendar dates (in stead of days since the first date in the data set.
- Loading the data from file (you will need `XMLHTTPRequest`).
- Nice graphical presentation will be credited.

[4] `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial`

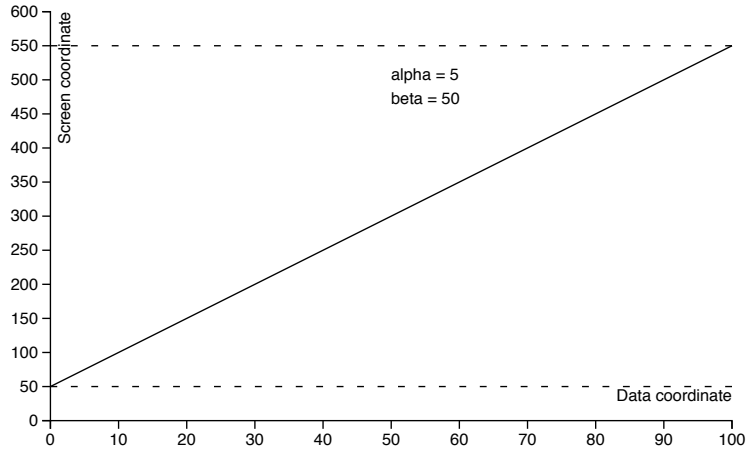

Figure 2.2: An example of a linear transform of the shape $f(x) = \alpha x + \beta$, that can be used to transform between data coordinates and screen coordinates.

2.2.1 Checks before submitting

- See Chapter A for general guidelines that submissions need to follow.
- Does your submission contain a PDF with the answers to the questions in Section 2.1, and the complete implementation of the temperature graph?

JavaScript 2

3.1 Reading Assignment and Resources

This week's assignment is to add interactivity to last week's visualization and to load the data formatted as JSON (JavaScript Object Notation) instead of CSV. To prepare read chapters 6 (about JavaScript objects and inheritance), 8 (error handling), 10 (modules) and 14 (event handling) of *Eloquent JavaScript*. JSON itself is not explained in a lot of detail this book, but the Wikipedia page on JSON and the `http://www.json.org` should provide you with enough background on JSON. If you are interested in more advanced background reading on JavaScript the series of books *You don't know JavaScript*[1] and the *JavaScript Garden* website[2] are great resources.

3.1.1 Questions

Answer the following questions in your own words. This assignment will only be graded *pass* or *fail*.

1. Why is it useful to encapsulate JavaScript code in a module as described in chapter 10 of *Eloquent JavaScript*?.
2. Describe what asynchronous programming is and how it differs from the style of programming you used in C and Python.
3. Is JSON valid JavaScript? Describe the differences between JavaScript and JSON.

3.1.2 Reformatting the data to JSON

Since JSON is commonly used to transfer data between JavaScript programs and web services that provide data[3]. We will transform our CSV data to JSON using Python, and then load that JSON data using the same trick as last week. In Python the `json` module provides a very easy way to save Python data structures as JSON. You should take the data of last week, load it into Python and create a list of lists of the data. This can then be saved as JSON using the `json` module's `dump` function. Take your submission of last week, remove the CSV data from the text area and replace it with the JSON data (JSON is very close to JavaScript in appearance, and like JavaScript it is just text).

3.1.3 Loading the data using JavaScript

As before you can access the JSON data as a string by selecting the `textarea`-element and grabbing its value. Use the built-in `JSON.parse` function to load the data (never use `eval` as it can lead to security issues). See what your data looks like with `console.log` before moving on. If you see a `null` value for the data you loaded most likely was not formatted correctly.

As you can see the JSON data that was loaded consists of an array of arrays containing the data, much like the data structure you created last week. You should therefore be able to re-use much of the JavaScript

[1] Released by O'Reilly and available on the web at `https://github.com/getify/You-Dont-Know-JS`.
[2] `http://bonsaiden.github.io/JavaScript-Garden/`
[3] Such as `data.overheid.nl`.

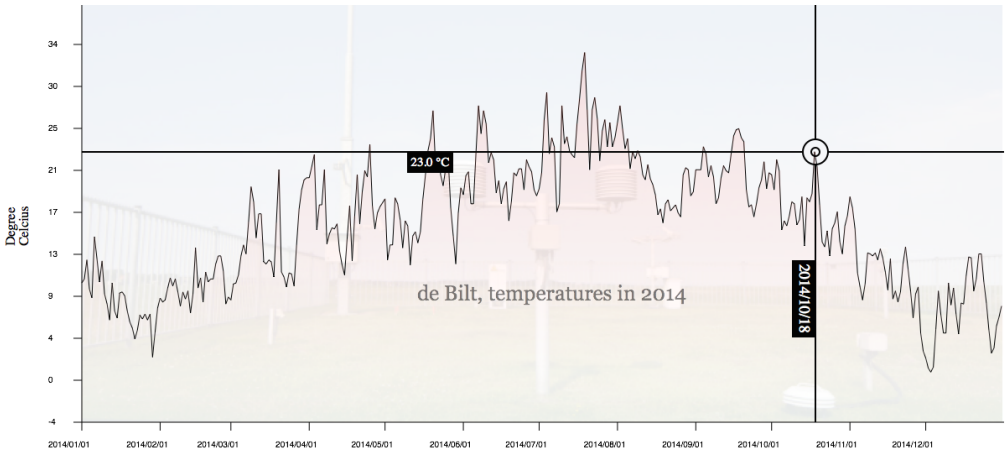

Figure 3.1: Temperature graph with interactive features as implemented by Tein de Vries.

code you used last week. Make sure your graph, based on the JSON data, looks the same as the graph based on the CSV data.

3.1.4 Interactivity

The interactive features we will implement this week are a tooltip that shows the exact temperature for a given day and an cross-hair that follows the X position of the mouse cursor and the Y position of the graph. The tooltip will be implemented as a absolutely positioned `div`-element and the cross-hair will be drawn on a `canvas`-element.

3.1.5 The cross-hair and event listeners

First create a second `canvas`-element that we will use to draw the cross-hair on. This `canvas`-element must be positioned *over* the graph and be the same size. By drawing all the interactive features on this second canvas you will not have to redraw the whole graph when you update the cross-hair. You can use a trick to erase the canvas, overwrite its `width` or `height` properties with their previous values.

To draw the cross-hair in the correct position you need the position of the mouse (relative to the canvas). This position can be determined by listening for mouse move events and implementing an appropriate callback. Register an event listener on the top `canvas`-element that listens for `mousemove`-events. In the callback access the `clientX` and `clientY` properties of the event that was fired, these properties have the mouse position relative to the whole document (not in "`canvas`" coordinates). Use `getBoundingClientRect()` of the `canvas`-element to correct the mouse position. As always you should use `console.log` to see whether what the actual coordinates are.

Now that you have access to the x-coordinate of the mouse relative to the canvas element you can find the height of the graph at that position. You do this by transforming the x-coordinate relative to the canvas back to "data" coordinates (use last week's `createTransform`) and looking up the nearest temperature value. That temperature value can then be used to find the y-position relative to the canvas. You now have the coordinates where the horizontal and vertical lines of the cross-hair should cross. Draw the cross hair.

3.1.6 The tooltip and `setTimeout`

The second interactive feature we are going to implement is a tooltip that appears several seconds after the mouse stopped moving and shows the actual, precise, data value under the cursor. The tooltip should be a `div`-element that you position absolutely on top of the two `canvas`-elements and slightly offset from the mouse pointer. First create this `div` make sure it shows the temperature and date of the point that is under

the cross-hair. Make the tooltip move with the mouse (that is update is position when a `mousemove`-event is fired.

The tooltip should not be showing the entire time and thus we will implement a delay to the moment that it shows up. Make sure that the drawing and updating of the tooltip is encapsulated in a function. Use the `setTimeout` function to delay the drawing of the tooltip (so make it call your tooltip drawing function as a callback). Make sure you make the tooltip invisible when the mouse starts moving again.

Figure 3.1 shows an example of the full graph.

3.1.7 Checks before submitting

- See Chapter A for general guidelines that submissions need to follow.
- Does your submission also contain the Python script you used to create the JSON data?

Scalable Vector Graphics

4.1 Introduction

In the previous weeks' exercises the `canvas`-element was used to create graphics, this week we will switch to Scalable Vector Graphics (SVG). Graphics created using the `canvas`-element are specified in terms of a raster of pixels, while graphics created with SVG are specified in terms of vectors shapes. Unlike raster graphics that are blurry at high magnification vector graphics remain sharp.

SVG as a language to specify graphics in looks much like HTML, modern browsers allow you to mix HTML and SVG. The SVG elements show up in the Document Object Model just like HTML elements do. Below is a simple example of an HTML document containing SVG that draws a red square.

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Small SVG example</title>
</head>
<body>
    <svg width="400" height="400">
        <rect x="100" y="100" width="200" height="200" fill="red">
    </svg>
</body>
</html>
```

4.2 Reading

A nice introduction to SVG can be found on the Mozilla Developer Network, while the World Wide Web Consortium has the official specifications for SVG.

- Mozilla Developer Network SVG tutorial:
  `https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial`.
- Mozilla Developer Network SVG resources:
  `https://developer.mozilla.org/nl/docs/Web/SVG`.
- SVG standards:
  `http://www.w3.org/TR/SVG/`.

Further reading about visualization in general:

- Rainbow Color Map still considered harmful:
  `http://cdn.mprog.nl/dataviz/excerpts/w4/Borland_Rainbow_Color_Map.pdf`
- A Tour through the Visualization Zoo:
  `http://idl.cs.washington.edu/papers/visualization-zoo`

- D3 Data-Driven Documents:
  `http://idl.cs.washington.edu/papers/d3`

4.2.1 Questions

Answer the questions below in your own words and you are however allowed to support your reasoning using quotes. Quotes must be clearly indicated and the sources properly cited. The answers to these questions are to be handed in as a A4-sized PDF file.

1. Explain why a rainbow color scale is inappropriate. Refer to the article *Considering Visual Variables as a Basis for Information Visualisation* (Carpendale, 2003) in your answer.
2. Find an example of an example of a inappropriate rainbow color scale. Explain what the problems are with that colors scale *and* suggest how the visualization can be improved. *Do not use any of the following:*
   - `https://eagereyes.org/basics/rainbow-color-map`
   - `http://visual.ly/high-resolution-global-topographic-map-moon`
   - `http://root.cern.ch/drupal/content/rainbow-color-map`
3. Compare and contrast two of the network visualizations shown in the *A Tour through the Visualization Zoo* paper. Furthermore explain which of these visualizations you find best.

4.3 Assignment

Hand in problems 1 and 2 as specified on `http://data.mprog.nl/homework/week-6-svg-and-maps` along with the answers to the questions above. *Skip the questions in problem 4 on the linked page.*

4.4 Checklist

- Are all files present and in the correct relative paths?
- Does your HTML pass the validation checks of `http://validator.w3.org/`?
- Does your CSS pass the validation checks of `https://jigsaw.w3.org/css-validator/`? Make sure you have two CSS files, one for the CSS rules that apply to the HTML elements and one for the CSS rules that apply to the SVG elements (otherwise your CSS will not validate).
- Does your script use strict mode (i.e. does it start with `"use strict";` and still work)?
- Do you have enough comments and are your variable names appropriate?
- Are your answers on a A4-sized PDF? Did you properly cite your sources? Did you indicated which text are quotes (by e.g. italicizing text and using quotation marks)?

D3 and SVG

5.1 Reading

The previous weeks we created visualizations using no JavaScript libraries, this week we will start using the Data-Driven Documents (D3) library [1]. This library works directly with the web standards and the debugging tools in your browser, and it provides building blocks for things like scales, transitions, geographic projections etc. If you have experience with a library like for instance Matplotlib[1] you will find that D3 is still quite low level — it provides no pre-built charts, only building blocks to create them.

D3 is very well documented with an API reference on its and a plethora of working examples with code available from its website[2]. The examples range from quite simple, such as a scatter plot, to quite advanced, like an interactive scatter plot matrix. When reading up on D3 pay particular attention to its concept of selections and the way data is *bound* to the DOM.

Please read the following (there are some questions).

- The scientific paper that describes D3 [1].
- Mike Bostock's blog post that describes D3 selections:
  `http://bost.ocks.org/mike/selection/` (accessed 2015-03-16).
- The example that demonstrates how a scatter plot is implemented in D3:
  `http://bl.ocks.org/mbostock/3887118` (accessed 2015-03-16).
- The example that demonstrates how a line chart is implemented in D3:
  `http://bl.ocks.org/mbostock/3883245` (accessed 2015-03-16).

Further useful documentation

- *Interactive Data Visualization* [2] a book on D3 by Scott Murray which is available on the web:
  `http://alignedleft.com/tutorials/d3/`.
- D3 tutorials on the D3 wiki pages:
  `https://github.com/mbostock/d3/wiki/Tutorials`.
- *D3 for mere mortals* a tutorial on some of the D3 concepts:
  `http://www.recursion.org/d3-for-mere-mortals/` (accessed 2015-03-16).

5.2 Questions regarding the readings

Answer the questions below in your own words and you are however allowed to support your reasoning using quotes. Quotes must be clearly indicated and the sources properly cited. The answers to these questions are to be handed in as a A4-sized PDF file.

- Answer the questions of problem 5 on `http://data.mprog.nl/homework/week-7-svg-and-d3`.

[1] `http://matplotlib.org/`
[2] `http://d3js.org/`

5.3 Assignment: re-creating the temperature graph

For this assignment you are to create a line plot based on the KNMI data used previously, but this time using D3 and SVG. Create a new HTML file that loads the D3 script, make sure you have a local copy of that file. Create a separate JavaScript file for your program. Use two CSS files, one for the HTML styles and one for the SVG styles (for now the CSS validator of the World Wide Web Consortium does not support mixing SVG and HTML properties). For this week also include a separate JavaScript file for your program.

As D3 provides functions that load CSV and JSON files, you will this week put the temperature data in a separate file. Whether you use JSON or CSV is up to you, as long as you load them from a separate file using the proper D3 functions. Note that these loading functions take callbacks and that the file loading is done asynchronously, so the data will only be available in the scope of the callback. As an aside: if you want to load several files that all need to be available before your program can run look into `queue.js`[3].

Now create a line plot using the D3 functionality, make sure that you properly label the axes, that the plot has a title and that the data source is explained. You are, of course, allowed to look at the examples on `d3js.org` but do not copy and paste code from there. In particular there is an example (referenced above) that explains how to create a line chart. Make sure that the date axis uses JavaScript dates, something that is quite easy using `d3.time.format` function.

D3 also supports interactivity, in particular the `selection.on` function allows you to bind event listeners to DOM elements. Recreate the moving cross-hair from week 5's assignment by binding an event listener to the appropriate DOM element. Note that D3 also makes dealing with mouse coordinates easier, see `d3.mouse`, and that the D3 scales have an `invert` function. Recreate the pop-up of week 5 as well, here you can use D3 to manipulate the DOM, but the use of `setTimeout` is no different from that of week 5.

5.4 Mapping using D3

In this assignment you will create a map of recent asteroid strikes. The data is available from NASA/JPL's Near Earth Object Program website[4]. The map you are going to create is inspired by the map of *Bolide Events 1994-2013*[5]. A simple, and therefore incomplete, example of what the map may look like.

For this exercise the following web pages are useful:

- *Let's Make a Map* a blog post by Mike Bostock where he explains how a map can be made using D3 and TopoJSON data. Part of the blog post is about the actual generation of the TopoJSON data, that part you can skip:
  `http://bost.ocks.org/mike/map/`
- The example world map with an exotic, Lambert Azimuthal Equal-Area, projection (we will use the map data of this example):
  `http://bl.ocks.org/mbostock/3757101`

5.4.1 Collecting the data

Write a small Python program to scrape the data and create a JSON file with at least the latitude, longitude and explosive power of the fireballs on the NASA web page. The web page only lists recent fireballs, so the original map cannot be recreated exactly. Furthermore download the `world-50m.json` data file used in the example map.

[3] `https://github.com/mbostock/queue`

[4] `http://neo.jpl.nasa.gov/fireball/`

[5] `http://www.jpl.nasa.gov/news/news.php?release=2014-397`

5.4.2 Loading the data

Choose a projection that you would like to use for your map, the built-in projections of D3 are listed in the documentation [6]. Create a new HTML file, add D3, TopoJSON[7] and `queue.js`[8] scripts.

First load the two data files using `d3.json` and `queue.js` as follows:

```
var q = queue(1);
q.defer(d3.json, 'world-50m.json');
q.defer(d3.json, 'asteroids.json');
q.awaitAll(drawMap);
```

In this snippet `drawMap` is a function that accepts two parameters, the first parameter is for any errors that may occur and the second parameter is passed an array with the two objects created from the JSON files. Any map drawing, both the base layer and the thematic layer, will be done in the scope of the callback function.

5.4.3 Drawing the base layer

Define the function `drawMap` and give it two parameters as described above. Following the blog post try to show the base layer of the map. You should draw each country's borders, need the `topojson.feature` function as described in *Let's Make a Map*, except in our case the TopoJSON data has no `subunits` but it has `countries`. You do not need custom colors for each of the countries.

5.4.4 Drawing the thematic layer

Now that you have the base layer, you can use the same geographic projection to add all the fireballs to the map. As an example we will add Amsterdam to the map. This example assumes you are using the Mercator projection and have a selected the SVG element using D3 and called it `svg`.

```
var lat = 52,
    lon = 5,
    proj = d3.geo.Mercator(),
    svgCoordinates;

var svgCoordinates = proj([lon, lat]);

svg.append('circle')
    .attr('cx', svgCoordinates[0])
    .attr('cy', svgCoordinates[1])
    .attr('r', 5)
    .attr('fill', 'red')
    .attr('stroke', 'black');
```

Now create a `g`-element (a group-element), give it a `id` of `thematic`. For each fireball add a `circle`-element to the group and style it appropriately. Use the proper D3 techniques here. No `for`-loop over the array of fireballs, in stead bind the data to the DOM.

5.4.5 Extra credit

You can get extra credit for the following:

- Change the size of the circles for each fireball based on its explosive power, you will probably need a logarithmic mapping from explosive power to radius.
- A top-tip showing more information about the event (so when it happened for instance).

[6] https://github.com/mbostock/d3/wiki/Geo-Projections
[7] `https://github.com/mbostock/topojson`
[8] `https://github.com/mbostock/queue`

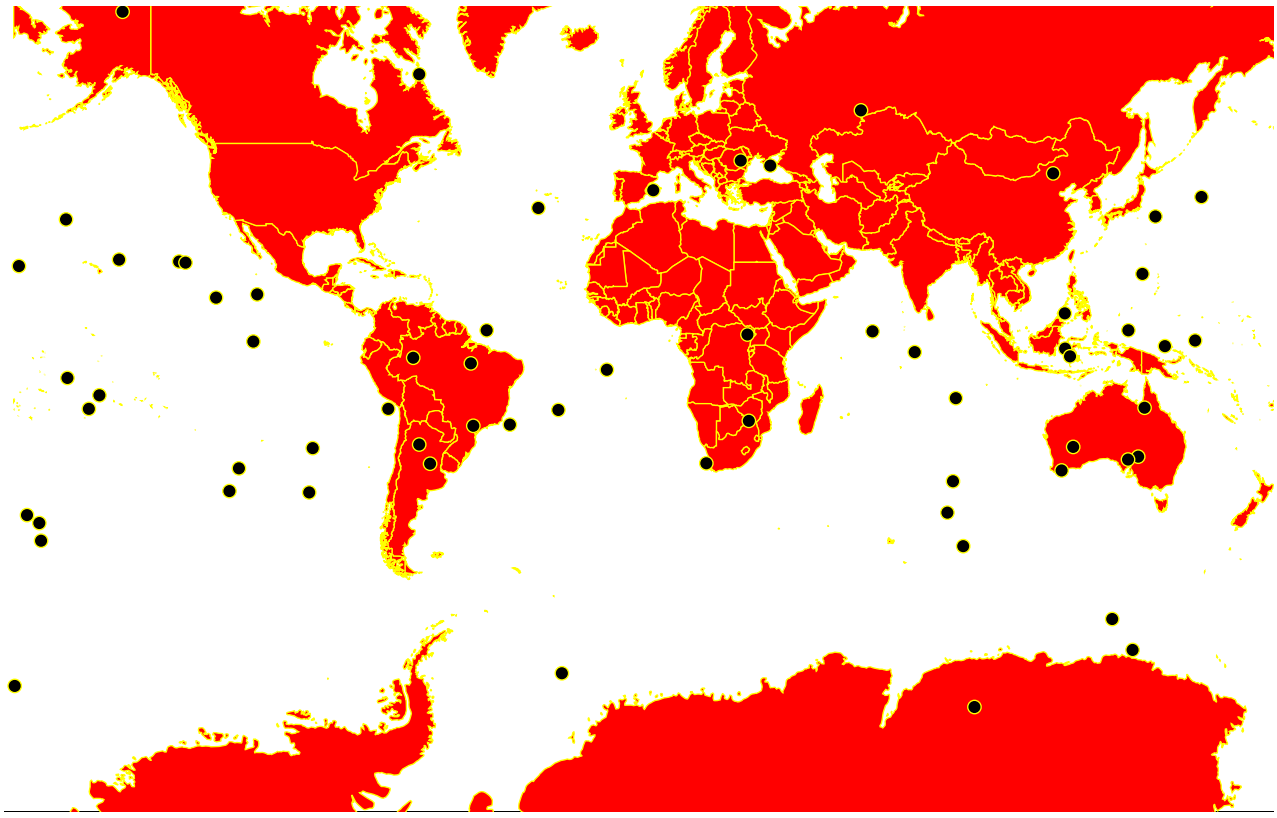

Figure 5.1: An example showing the basics of the thematic map of asteroids.

5.5 Checks before handing in

A non-exhaustive list of checks:

- Do your HTML and CSS validate?
- Does your JavaScript work in strict mode?
- Have you included all the needed scripts (so no linking to scripts on the Internet).
- Are your visualizations properly labeled? Are your data sources described in the page.
- Is your Python scraping script included?
- Do you have enough comments and are your variable names appropriate?

Bibliography

[1] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[2] Scott Murray. *Interactive Data Visualization for the Web*. O'Reilly, 2013.

Checklist for Submissions

Below you will find a small, non-exhaustive, checklist for your submissions.

A.1 Readings

- Your name, student number and the assignment should be indicated on your submissions.
- Give answers in your own words. If a you have a source that you want to quote you must clearly indicate the quote (different typeface, in quotes and with a citation to the source).
- Clearly indicate which of the questions you answer (so number your answers at the very least, or better, repeat the question and then provide your answer).
- Answers must be provided as A4-sized PDF files. If you use a program like Microsoft Word you can print the document to a PDF file.

A.2 Code

A.2.1 General

- Your name, student number and the assignment should be indicated in the source files.
- Remove commented out code.
- Variables, functions and modules should properly named.
- No magic numbers, if you use some constants, name them appropriately.
- Code should be properly commented and documented.
- Try to minimize the number of variables in the global/outer scope.
- No code copied from other sources and if your code is based on example code available on the Internet indicate so using a comment with a link to the example.

A.2.2 Python

- Follow the Python style guide (PEP 8) for Python code.
- Structure your code such that it can both be imported as a module and run as a program.

A.2.3 Web pages and JavaScript

- HTML5 should be used (not XHTML, nor HTML4 and older).
- HTML should pass the validation checks of `http://validator.w3.org/?`
- CSS should pass the validation checks of `https://jigsaw.w3.org/css-validator/?`
- No jQuery, just pure Javascript unless otherwise indicated.
- Does your script use strict mode (i.e. does it start with `"use strict";` and still work)?
- All CSS and Javascript files should be included in a submission, no links to external resources or files provided through Content Dilevery Networks (CDNs).
- All files should be present and in the correct *relative* paths.

Design critiques

This chapter contains a list of points to consider when critiquing a design, consider them conversation starters.

- What is the problem domain or context of the visualization under consideration?
- Which tasks can be achieved with this visualization?
- Tufte's principles of graphical integrity:
  - Are the scales appropriately labeled?
  - Is the *Lie factor* high?
  - Does the visualization show data variation and not design variation?
- Tufte's visualization design principles, are they adhered to?
  - Maximize the *data-ink* ratio.
  - Avoid chart junk.
  - Increase data density.
  - Layer information.
- Graphic design principles:
  - How is *contrast* used? What kind of contrast is used?
  - How is *repetition* used?
  - How is *alignment* used?
  - How is *proximity* used?
- Comment on the visual encodings that are used.
  - Which visual encodings are used?
  - Are the visual encodings appropriate?
- Comment on subjective dimensions such as *aesthetics*, *style*, *playfulness* and *vividness*.
- What is the intended goal of the visualization and is that goal achieved?
- Are there any things you would do differently, and why?