Treemap visualisations

This exercise aims to be a getting started guide for building interactive treemap visualisations using the D3 JavaScript library. While data visualisation has existed for many years and the theory is well established, recent improvements in technology and browser performance allow interactive visualisations of huge datasets. Treemaps, a rich visualisation for representing numeric data, are especially suited for analysing distribution of resources in large organisations.

Introduction

This exercise is built around web technologies that, when put together, build powerful and interactive HTML5 web pages. A core part of HTML5 is “action”, specified using the Javascript language. While Javascript is commonly used for manipulating elements on a web page, more advanced features include graphics and visualisation libraries. D3, one of these libraries, is built especially for multi-dimensional, interactive visualisation of large datasets.

The Datasets

1. UK Home Office Flight Data

This dataset contains data about the flights taken by members of the Home Office during 2011. This dataset includes the following:

- Destination region
- Flight operator (e.g. British Airways)
- Ticket class
- Ticket price

A treemap visualisation is ideal for this dataset. It is likely that each of these items, except price, will many duplicates in the dataset, so grouping the data is beneficial. For example, there are only three different destination regions in the entire dataset.

A tool such as OpenRefine is recommended to explore and clean the dataset, to ensure that the data is ready for visualisation and does not contain any inconsistencies.

2. Tanzania education data

This dataset contains data about performance of schools across Tanzania. This dataset lists items including:

- Name of school
- District of school
- Band of school (1-10)
- Pass band of school (red, yellow, green)
- Number of registered students
- Number of students who passed

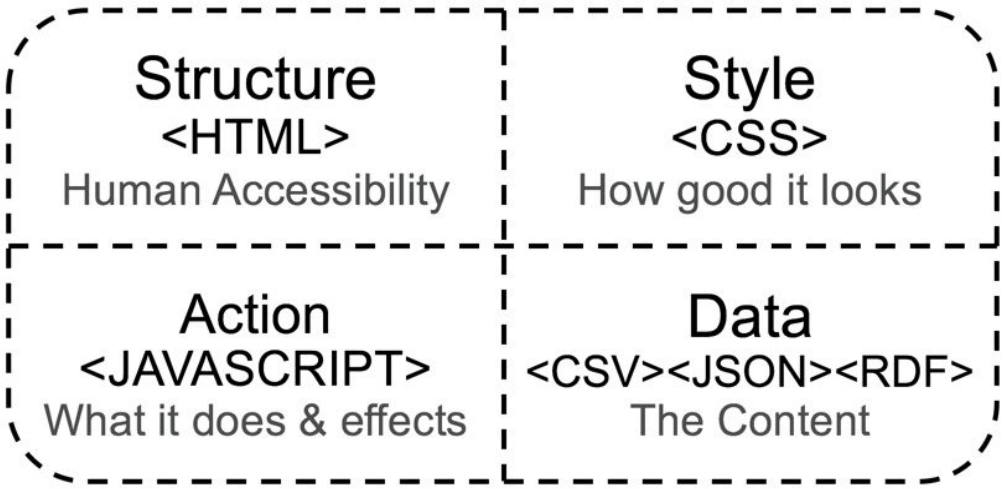
A treemap visualisation is ideal for this dataset as each of these items, except name of school, will likely contain many duplicates in the dataset, thus the data can be grouped. For example there are only six areas and thirty different districts in Tanzania.

In order to explore and clean the dataset it is recommended that a tool such as OpenRefine be used to ensure that the data is ready for visualisation and does not contain any inconsistencies.

Building blocks of the web

In this exercise we shall be using codepen.io to build an interface to explore complex data formats.

codepen.io provides a safe environment to experiment with building interactive web pages using modern HTML5 components outlined below. This exercise focuses on the Action block to load data and generate an advanced crossfilter visualisation.



Structure: The structure of a web page. Designed to enable human consumption of the content no matter the type of user or device. Accessibility is key and most pages misuse structure in order to add style, making content very hard to access for disabled the and visually impaired. When was the last time you accessed your website using a screen reader?

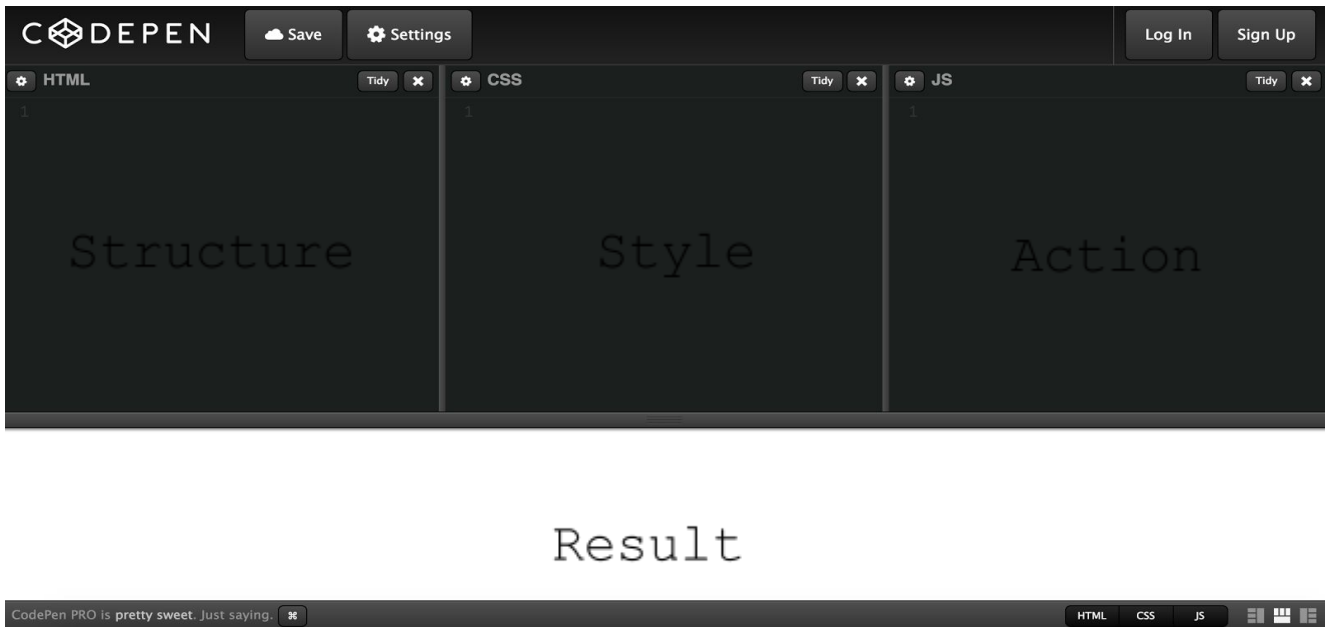
Style: How the web page looks. Including where the elements are on the page, what colours and borders things have and what is hidden from view. Using style sheets you can define multiple presentations of your content for different media and users, e.g. desktop, mobile and print mediums.

Action: What the page does. Using action you can define interaction with the content. Simple examples include dynamically loading content based on user choices, controlling embedded video and multimedia, and rotating banner news articles to promote content.

Data: Your content. Data is machine readable, e.g. in a database or provided by a data provider, e.g. calendar and news/blog feed system. Data is the content which gets embedded in your web site for presentation to a user. Traditionally, web servers obtain the data from an internal system and ONLY present it on a web page, designed only for human consumption. Equally, you can serialise your data into other formats in addition to your HTML and expose the data in a machine readable fashion.

codepen.io

codepen (shown below) is an online sandbox which allows you freely to experiment with the building blocks of the web. It has three main panels that allow you to define structure, style and action related to your web page. The fourth panel (at the bottom) shows the result of your editing; this panel automatically updates as you enter your code.



In this exercise we shall be loading data dynamically to produce our treemap visualisation.

Step 1 – Getting started.

From the course website you should find a link to a codepen template for the dataset you are interested in.

To obtain your own copy of the template, simply click the **fork** button.

Once you have forked a copy, create an account with codepen.io so you can easily access your work later.

Step 2 – Exploring codepen

Before we begin, let's look at the content of the **HTML**, **CSS** and **JavaScript** blocks.

The **HTML** block defined one thing only, the element in which the treemap is put (called chart).

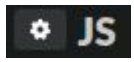
```
<div id="chart"></div>
```

The **CSS** block is a little busier and contains the styling for the treemap. Key to this style is the setting of various opacity levels which allow you to see deep within the treemap.

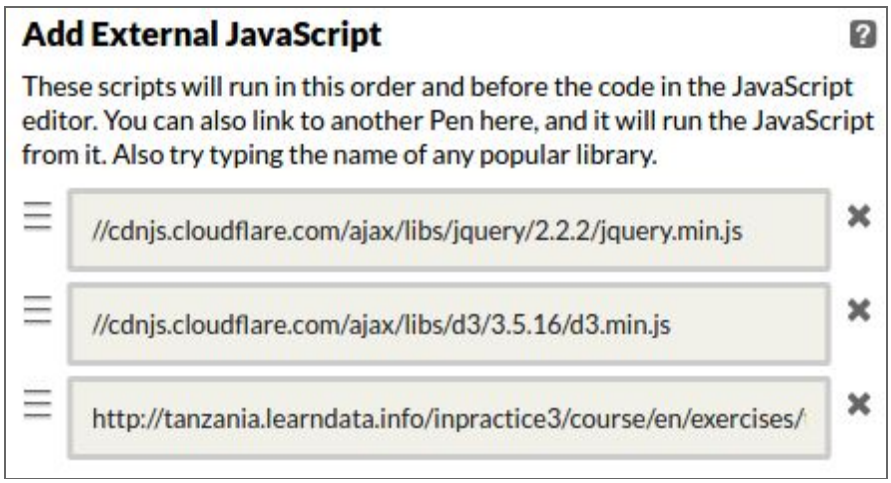
The **Javascript** block is the most complicated, as this is where the data is loaded and the treemap built.

In addition to the short amount of javascript that is contained within this panel, there are a number of helper libraries loaded that are also doing a lot of heavy lifting.

The two main libraries are the jQuery library, a shortcuts library for common javascript functions, and the D3js library, which produces the visualisation.

To see which libraries are loaded click the **settings cog** next to 

JS.



Here you can see that jQuery and D3 are loaded and there is also an additional helper library which abstracts the common treemap functions. If you are digging deeper into D3, then you will probably need to customise this last library for youself. You can open the link to view the contents of this library if interested.

Returning to the main interface of codepen, you will be able to see that the rest of the code in the javascript block has been commented line by line (comment lines start with "//"). Hopefully the comments are straightforward to follow.

Step 3 – Making the treemap

In order to make the treemap we need to understand the dataset. From the course website you should be able to download a copy of the dataset and explore it in a desktop tool like Excel.

Take a look through the CSV file and note below the order of the columns that you wish to display on your treemap at the various parent child levels. An example for the two datasets in given below.

Air travel dataset	Tanzania dataset

Destination | Supplier_name | Ticket_class_description	AREA NAME | REGION NAME | COLOR

In this flight example the top level of the tree will split the data by destination (3 options) then by carrier (supplier) and finally by class of ticket. The final element is the numerical paid fare for each ticket; here each ticket represents a leaf of the tree.

More on hierarchical data, parent-child relationships and treemaps can be found at the end of this exercise.

Step 4 – Editing the javascript

5a – Load some data

Your first line of code in our control file calls the *d3.csv* function. This function loads a file and returns its contents as a csv data object (*res*).

The examples have already been set up to load a defined dataset, however you can choose to change this later.

```
// Convert a csv file (tabular data) into a json hierarchical structure
d3.csv("data/data_file.csv", function(csv_data){
```

Once the data is loaded, the function returns res which can then be manipulated in terms of columns and rows. d3.csv essentially loads a tabular csv file into a tabular data structure ready to be manipulated.

5b – Change data from tabular to hierarchical

The second stage involves changing our data structure from tabular to hierarchical using the structure that you defined in **Step 3**. To do this we can use the d3.nest() function which will give us back our hierarchical *nested_data*.

```
var data = d3.nest()
  .key(function(d) { return d["Grand parent"]; })
  .key(function(d) { return d["Parent"]; })
  .key(function(d) { return d["Child"]; })
  .entries(res);
```

In order to define the order of the nesting, change *d[“Grand parent”]*, *d[“Parent”]* and *d[“Child”]* to the **exact** column titles at each level. The follow example corresponds with that given in **Step 4** for the flights dataset.

```
var data = d3.nest()
        .key(function(d) { return d["Destination"]; })
        .key(function(d) { return d["Ticket_class_description"]; })
        .key(function(d) { return d["Supplier_name"]; })
        .entries(res);
```

NOTE: Do not define the last element by which the summation of the treemap is done (e.g. Paid_fare), this is not required when the column is called value.

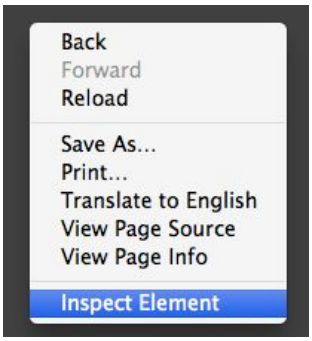
If you need more levels of nesting, add another *.key(..* row above the final *.entries(..* line.

5c – Give you treemap a name and top level item.

On the last line of the file you can choose to give your treemap and name and a top level Grand parent title for displaying inside the treemap.

Step 7 – Debugging

If you managed to show a map, fantastic! If not then this section is for you.



Many web browsers come with useful tools to help debug errors on a web site, for the purposes of this exercise we are going to use the **Web Inspector** in **Google Chrome**. To access this first ensure you have your broken web page displayed and then right click anywhere on the page itself and click **Inspect Element**.

The web inspector is a very powerful tool and allows you to view the source of a web page, view its performance as well as browse locally stored data. The feature we are interested in is the console, where errors in our scripts will be displayed.

In the case of this exercise some of the errors you might see may include:

- Failed to load resource: the server responded with a status of 404 (Not Found)
  - Probably an error in your config or the data file is missing
- Failed to parse, unexpected …
  - You forgot to close a bracket or have too many/few commas and semicolons in your config file.
- Uncaught typeError
  - The data file may not be a CSV

Step 8 – Extension exercise

A simple exercise is to change and extend the hierarchy.

A more advanced extension involves getting a different dataset working to create a treemap.

Tabular data and hierarchical data

In order to draw a treemap, we need our data to in a hierarchical form. Such data has a clear parent-child relationship between every element; the deeper you go, the more children you have and thus the greater the number of grandparents.

As CSV is a tabular format, we need to translate our data from one format to the other. Helpfully we can get a computer to do this for us.

Additionally, the D3 library works best when data is in the JavaScript Object Notation (JSON) format. Below is an example of a hierarchical dataset in JSON format:

```
                                                                    Grandparent
{
   "name": "Grandparent",
  "children": [                                                          Parent
     {
        "name": "Parent1",
        "children": [
                    { "name": "Child_1", "value": 15 },                      Children
                    { "name": "Child_2", "value": 24 },
                    { "name": "Child_3", "value": 28 },
        ]
     },
     {
        "name": "Parent2",
        "children": [
                    { "name": "Child_4", "value": 7 },
                    { "name": "Child_2", "value": 8 },
        ]
    }
  ]
}
```

While we could translate CSV into this format by hand, we used the d3 *nest* function to do this.