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 has enabled the provision of interactive visualisations for 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 Dataset – UK Home Office Flight Data

This dataset contains data about flights taken by members of the home office during 2011. This dataset lists items including:

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

A treemap visualisation is ideal for this dataset as each of these items, except price, will likely contain many duplicates in the dataset, thus the data can be grouped. For example there are only three different destination regions in the entire dataset.

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

GitHub.com

To effectively prototype our visualisation, we are going to use GitHub, an online version control tool. One major advantage of using GitHub is that this service provides you with free online web hosting for your project once completed.

NOTE: You will need a GitHub account with a **verified** email address in order to complete this exercise.

Step 1 – Exploring the Repository

For the purposes of this exercise, an example GitHub repository has been created at the following location:

https://github.com/ukodi-training/Infographics

Opening a web browser at this location should show you the main GitHub screen with the “Code” tab highlighted on the right hand side.

The repository contains a website, to view the website simply remix the URL to point at a GitHub.io address:

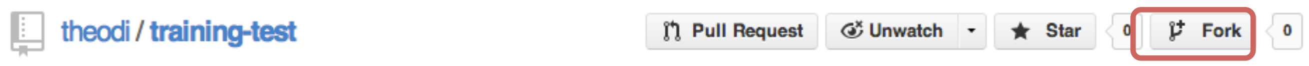
http://ukodi-training.github.io/Infographics/

Feel free to browse around this website. It contains a multitude of infographics templates that are used in different exercises. For these exercise we are interested in completing the treemap visualisation.

Step 2 – Fork the Repository

This repository belongs to the ukodi-training user. In order to edit it you will need your own copy; a process known as forking. This process can be carried out from the main github.com repository site (the first url above).

To fork the repository, ensure you are logged in and then click the fork button:



Once you have forked the repository you will be re-directed to a location from which you can make changes.

Step 3 – Exploring the repository

Before we can edit the repository, lets have a look at the different parts of the treemap folder.

Inside this folder you will find 3 directories and a file called index.html.

1. The index.html page is your web page and defines the **structure** of the page and locations of related **style** and **action** files.
2. The css directory contains cascading **style** sheets that define the appearance of the web page (colours and font size etc)
3. The js directory contains javascript **action** files that define how the page interacts with the user. It is in here that we control both the infographic and its interactivity.
4. The **data** directory contains your data files (csv, json, xml etc) that are loaded by the action scripts and displayed on the web page.

Essentially any treemap can be drawn by uploading some data into the data directory and then editing the **js/control.js** file to define what to do with this data.

Step 4 – Making the treemap

http://*your-username*.github.io/Infographics/

Before we begin, make sure you can see your website but opening the URL above in a new tab alongside your GitHub code repository. You will need the treemap directory for this exercise in both cases.

From the GitHub code repository, browse into the data directory note the presence of a file called "*Home_Office_Air_Travel_Data_2011.csv*", click on this file and take a look through the data. In order to create a treemap from this data, we must first decide how to group the data in order to create a hierarchical dataset.

Take a lot 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 air travel dataset is given.

Air Travel Dataset

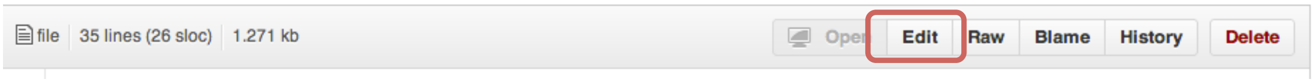
```
Destination
    |Supplier_name
        |Ticket_class_description
          |  Paid_fare
```

In this 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 5 – Editing the control file (js/control.js)

In order to edit the control file we need to navigate to the js directory, view the control file and then click the edit button.

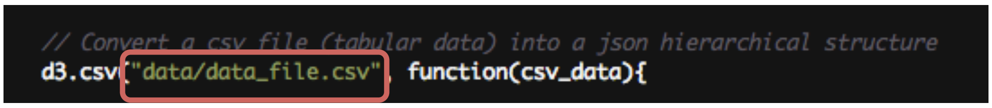


The control defines a single function that executes when the rest of the web page (document) has loaded (ready). This means that the client's web browser has loaded the structure and style files and is ready to execute actions; you cannot perform actions on something that is not loaded. The action in this case is to load some data, translate it into hierarchical form and then draw a treemap.

5a – Load some data

You 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 (*csv_data*).

Change this function call to point to the data file in the data direction you want to use.

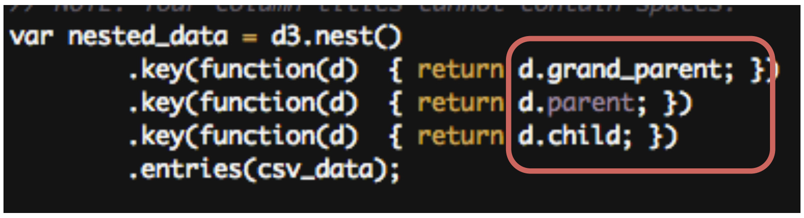


```
// Convert a csv file (tabular data) into a json hierarchical structure
d3.csv("data/data_file.csv", function(csv_data){
```

Once the data is loaded, the function returns csv_data 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.

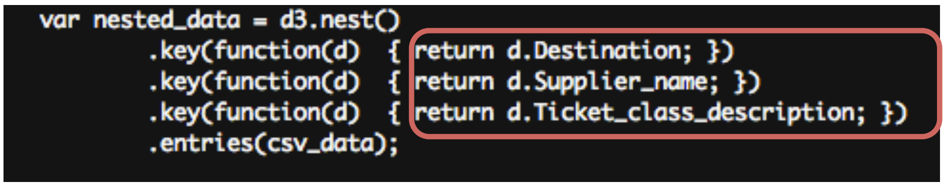
5b – Change data from tabular to hierarchical

The second stage involves changing our data structure from tabular to hierarchical using the structure that you defined in **Step 4**. To do this we can use the d3.nest() function which will give us back our hierarchical *nested_data*.



```
var nested_data = d3.nest()
        .key(function(d)  { return d.grand_parent; })
        .key(function(d)  { return d.parent; })
        .key(function(d)  { return d.child; })
        .entries(csv_data);
```

In order to define the order of the nesting, change *d.grand_parent*, *d.parent* and *d.child* to the column titles at each level. The follow example corresponds with that given in **Step 4**.



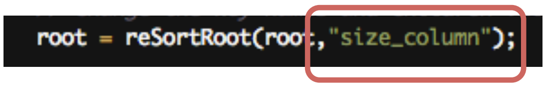
```
var nested_data = d3.nest()
        .key(function(d)  { return d.Destination; })
        .key(function(d)  { return d.Supplier_name; })
        .key(function(d)  { return d.Ticket_class_description; })
        .entries(csv_data);
```

NOTE: Do not define the last element by which the summation of the treemap is done (e.g. Paid_fare), this is done in the next step.

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

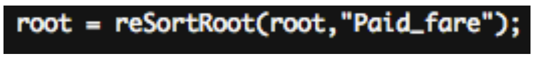
5c – Define the column that contains the values that represent the size of each treemap square

The final stage of preparing the control file is to define which column contains the numeric values that will be used to define the size of each area of the treemap. This can be done by editing the column name called in the *reSortRoot()* function.



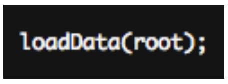
```
root = reSortRoot(root,"size_column");
```

The reSortRoot function parses our new hierarchical dataset and changed the column title of your defined column to the pre-defined “values” title required for using the data with off-the-shelf treemap implementations in D3. Although we could change the column title in the data itself, this would lose the context of what data the column contains and is considered bad practice.



```
root = reSortRoot(root,"Paid_fare");
```

With the data now transformed and translated into the correct form to create a treemap we can now simply hand the data (called *root*) to the function that draws the treemap (*loadData*).

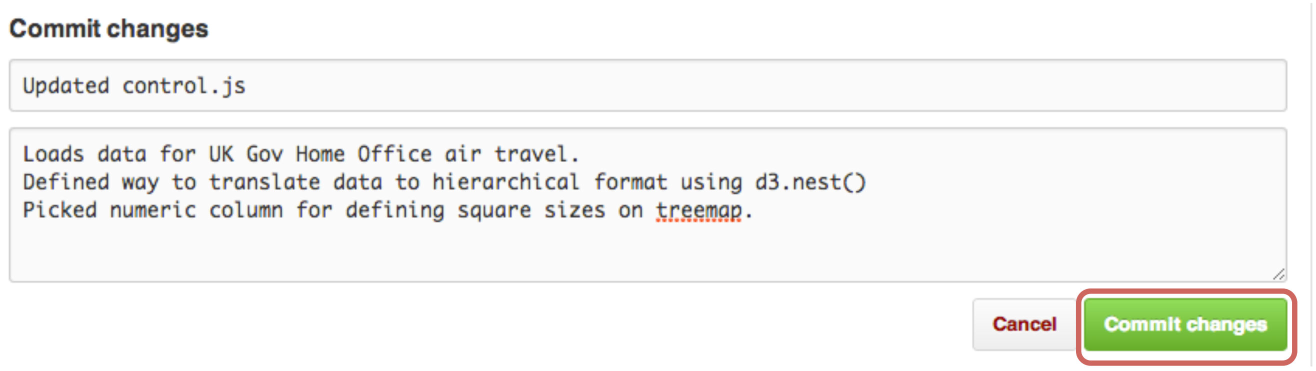


```
loadData(root);
```

In the example this is already done for us.

5d – Commit and reload

In order to save your control file, scroll to the bottom of the page and fill in some comments about your changes and the click commit changes.



Commit changes

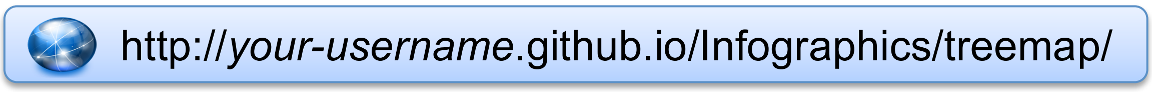
```
Updated control.js
```

```
Loads data for UK Gov Home Office air travel.
Defined way to translate data to hierarchical format using d3.nest()
Picked numeric column for defining square sizes on treemap.
```

As this is a version control system, it is polite to let people know what changes have been made and why at each stage. Additionally you might need to come back later to fix any bugs and typos.

Step 6 – Viewing the result

Simply browse to your treemap example web page in your browser and see if it worked.

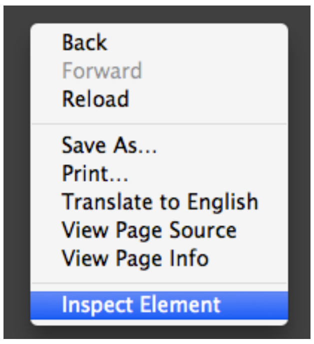


http://*your-username*.github.io/Infographics/treemap/

If it worked, well done, proceed to **step 8**. If it didn’t proceed to **step 7**.

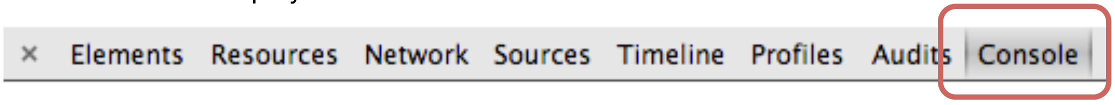
Step 7 – Debugging

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



Many web browsers come with nifty 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 semi-colons in your config file.
- Uncaught typeError
  - The data file is not a csv?

Step 8 – Extension exercise

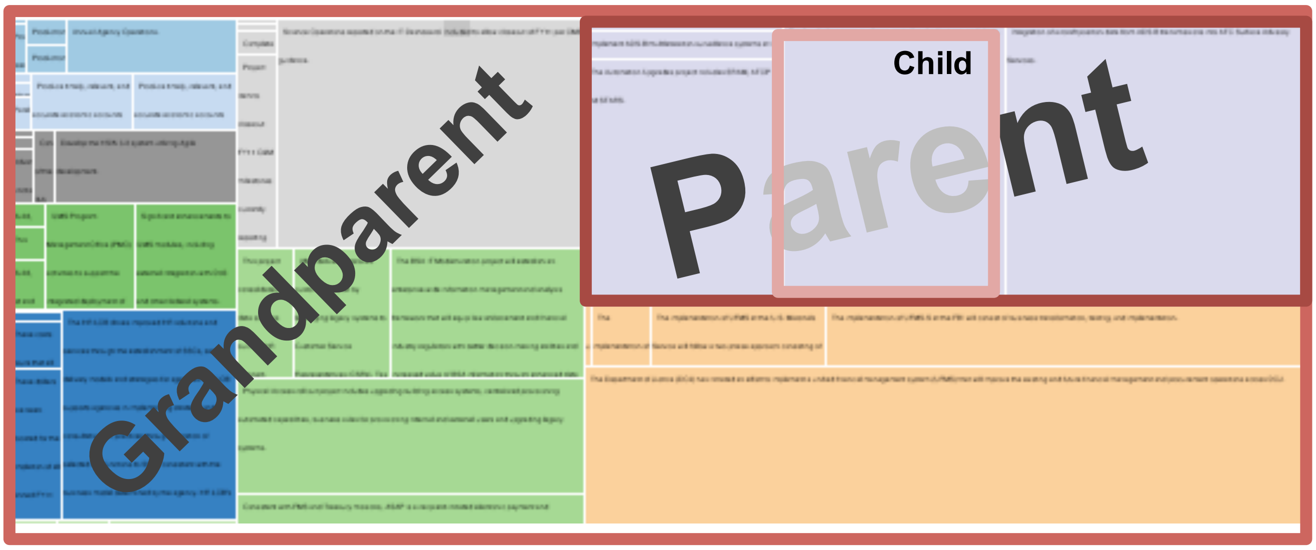
The challenge here is to redo the exercise with your own dataset, if you haven't done so already.

Here is the recipe to get you started:

1. Upload your data to the data directory (you can create a new file and then copy/paste the data)
2. Change the config.js file to point at the new dataset
3. Add the hierarchical structure to the config.js file.
4. Define the column name that dictates the size of each element of the tree
5. Save and view.

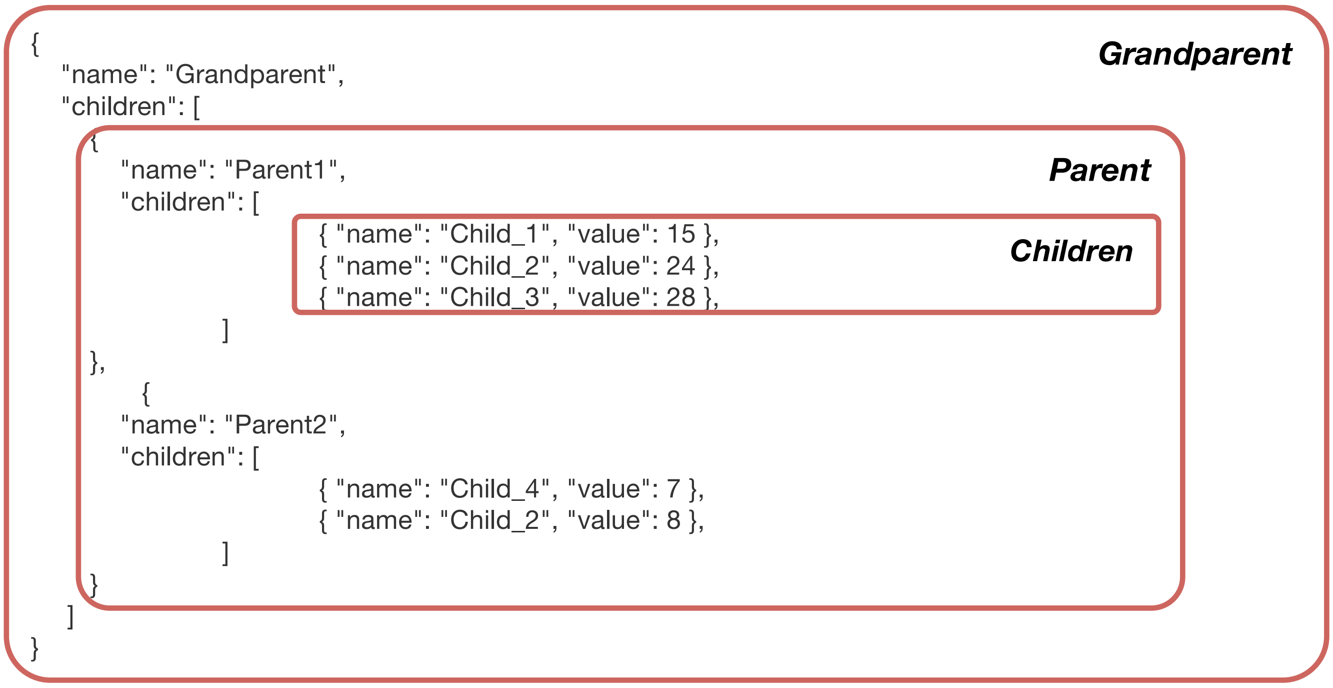
Tabular data and hierarchical data

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 tablular format, we are going to 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:



```
{
  "name": "Grandparent",
  "children": [
    {
      "name": "Parent1",
      "children": [
                     { "name": "Child_1", "value": 15 },
                     { "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 are going to use the d3 *nest* function.