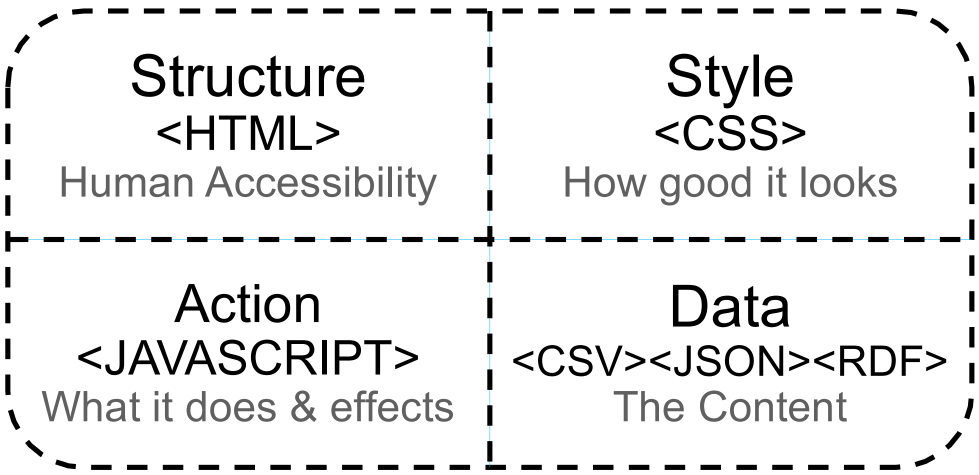
Exploring complex datasets

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 focusses on the **Action** block to load data and generate an advanced crossfilter visualisation.

Building Blocks



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 web site 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.

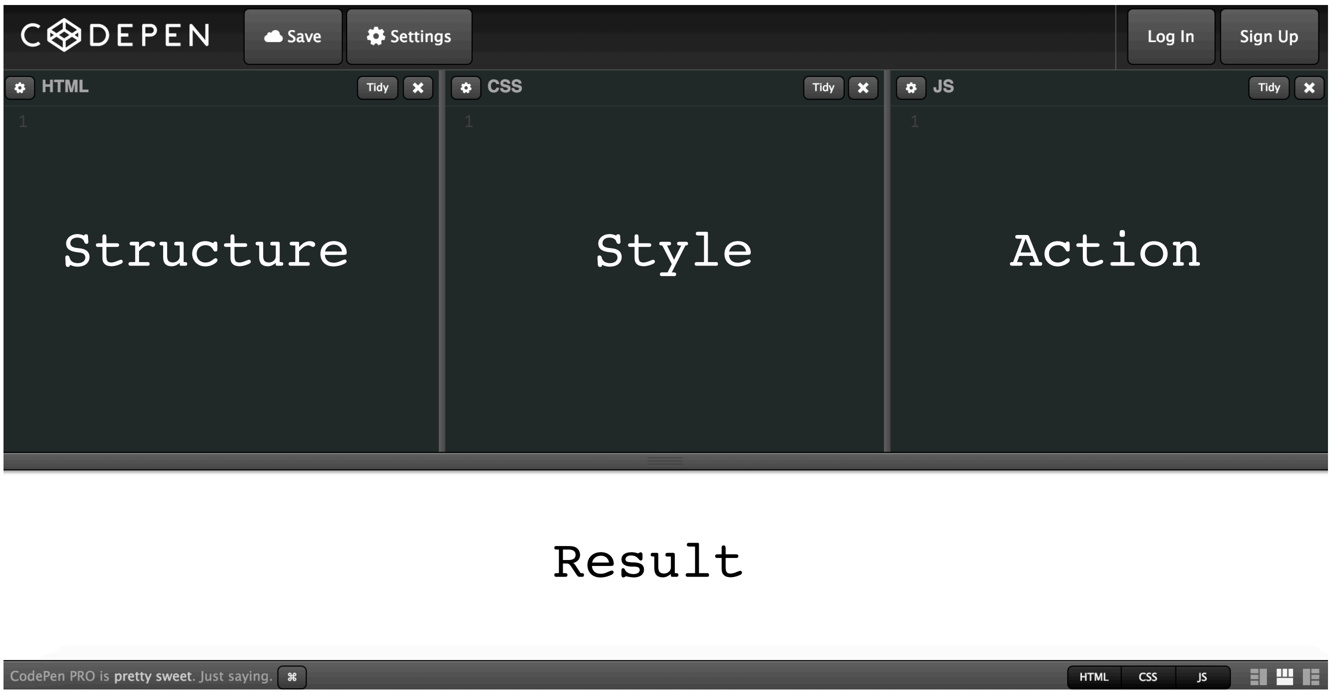
Exercise

The idea of this exercise is to extend beyond simple HTML5 pages and look at how Javascript can be used to generate interactive graphics from raw data

In order to complete this exercise we are going to use the codepen.io tool.

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 and the aim is to produce a live, automatically updating, dashboard view of data on the web.

Starting point

In order to speed up this exercise a basic template has been created.

http://codepen.io/davetaz/pen/NGprqg

This template has already defined the basic HTML elements as well as core JavaScript functions that will draw a basic graph of some data.

Load this template and fork it into you own account for editing.

Note that the **D3**, **bootstrap**, **crossfilter** and **dc** libraries have already been configured and loaded. These libraries make our job of making an interactive visualising much easier.

1. The framework

Before we begin, lets look at the content of the HTML and JavaScript blocks.

```
<div id="elevation-chart"></div>
<div id="bedroom-chart"></div>
<div id="location-chart"></div>
```

The HTML block defines 3 elements (divs) that are going to contain one chart each. With the data being about houses, we are going to build a simple cross filter of the elevation, number of bedrooms and location of each property.

```
// Define some variables to control our three charts.
//var elevationChart = dc.barChart('#elevation-chart');
//var bedroomsChart = dc.rowChart('#bedroom-chart');
//var locationChart = dc.pieChart('#location-chart');

// Load data from the web into a variable called data.
d3.csv('http://training.theodi.org/StatsML/data/training-data.csv',
function (data) {
      // Your code goes in here
});
```

The JavaScript block (shown above) only contains two lines of code that are execute. These two lines load in the data and perform a blank function with it. All other lines are commented out with "`//`".

As we add charts you need to ensure that the variables at the top of this block are uncommented so they become usable.

All other code added in this exercise goes in the function block, marked "`Your code goes in here`".

2. Setting up the cross filter

The cross filter libraries are pre loaded so we just need to load the data into them ready to manipulate and visualise.

Add the following to the BEGINNING of your code block (Your code goes in here)

```
var ndx = crossfilter(data);
var all = ndx.groupAll();
```

These two variables create a cross filter called `ndx` and a cross reference variable call `all`. We will need these to draw graphs. If we draw the graphs with `data`, the cross filter would not work.

Add the following to the END of your code block.

```
dc.renderAll();
```

This function call tells our charts to render on the page. So far nothing should happen as we have no charts!

3. Add a location pie chart

Adding graphics involves two main steps:

1) Define the data.
2) Define the chart and load the data into it.

Add the following to the MIDDLE of your code block to define the data required for the pie chart.

```
var location = ndx.dimension(function(d) {
      if (d.target == 1) { return "San Francisco"; }
      if (d.target == 0) { return "New York"; }
});

var locationGroup = location.group();
```

Here an `ndx dimension` is a row in the table. This row is loaded into a variable called `d`. We select the column called target from this row and return a human readable representation of the target which is wither 0 (for New York) or 1 (for San Francisco). The last line adds this chart to the cross filter grouping.

Add the following below the data definition to create a pie chart for the data. Don't forget to uncomment the location chart at the top of the JavaScript block.

```
locationChart
        .width(180)
        .height(180)
        .radius(80)
        .dimension(location)
        .group(locationGroup)
        .ordinalColors(['blue', 'green'])
        .label(function (d) {
            var label = d.key;
            return label;
        });
```

Feel free to change the height, width, radius and colours of the pie chart to suit your own preferences.

Once done you should have rendered a pie chart on your page which shows 139 properties in San Francisco and 111 in New York.

This is a simple visualisation, but not a great cross filter as we only have one chart. So let's plot another.

4. Adding a second chart

In this section we are going to add a second chart to our cross filter. In particular we are going to add a row chart looking at the number of bedrooms in a property. Again we have to load the data and then display the chart.

Add the following below your pie chart code to load the bedroom data.

```
var bedrooms =  ndx.dimension(function (d) {
            return Math.round(d.beds);
    });
var bedroomGroup = bedrooms.group();
```

Note that we have to round the number of bedrooms to a whole number as some properties amazingly manage to have half a bedroom which doesn't make much sense. Feel free to remove the `Math.round` and see what happens to the data when rendered.

Add the following below your data load to add the chart to your cross filter. Remember to uncomment the bedroom chart at the top in order to display it!

```
bedroomsChart
        .width(480)
        .height(400)
        .group(bedroomGroup)
        .dimension(bedrooms)
        .ordinalColors(['#3182bd', '#6baed6', '#9ecae1',
'#c6dbef', '#dadaeb','#3182bd', '#6baed6', '#9ecae1'])
        .label(function (d) {
            return d.key;
        })
        .title(function (d) {
            return d.value;
        })
        .elasticX(true)
        .xAxis().ticks(4);
```

Again you can adjust the dimensions of the chart. Colours are listed at html colour codes and you can change them but using an online colour palette (e.g. http://www.w3schools.com/tags/ref_colorpicker.asp)

With this chart rendered you now have your first cross filter. To use it select an option from any chart to filter all other charts.

With the line chart you can also change axis properties. Try changing the elasticX property to false and see what the difference is when you select New York or San Francisco from the pie chart.

5. Adding a third chart (less guidance)

The last chart to add is the elevation chart.

First create the data following the same method as in (3) and (4). Make sure you select and call your variable elevation and not bedroom or location. Note that you do not need to perform any rounding or changing of the elevation data.

With the data created now add the elevation chart. Don’t forget the thing to uncomment at the top!

```
elevationChart
    .width(800)
    .height(300)
    .dimension(elevation)
    .group(elevationGroup)
    .elasticY(true)
    .centerBar(true)
    .gap(1)
    .round(dc.round.floor)
    .alwaysUseRounding(true)
    .x(d3.scale.linear().domain([0, 230]))
    .renderHorizontalGridLines(true);

elevationChart.xAxis().tickFormat(
    function (v) { return v + 'ft'; });
elevationChart.yAxis().ticks(5);
```

This last chart is a filter chare where you can drag across to filter a selection of the data.

PTO.

6. View tabular data

As you are filtering you can also view the raw data in a table. Adding this is very slightly more complex as it is not defined anywhere in the HTML or JavaScript yet.

To define it in the HTML block (not the JavaScript block, add the the following)

```
<table id="dc-data-table" class="table table-hover"></table>
```

Next add the variable to the top of the JavaScript block where all the others are.

```
var dataTable = dc.dataTable('#dc-data-table');
```

Now we can add the code to add the table to the page. We do not need to select the data as we can use any dimension to access the current filtered data. In this case we are going to use the location dimension.

Add the following code to where the rest of the charts are made in order to draw the table.

```
dataTable
    .dimension(location)
    .group(function (d) {
        return d.target;
    })
    .size(10)
    .columns([
         {
            label: 'Location',
            format: function(d) {
              if (d.target == 1) { return "San Francisco"; }
              if (d.target == 0) { return "New York"; }
            }
          },
          'elevation',
          'beds',
          'bath',
          'year_built',
          'price_per_sqft'
    ]);
```

Now you can filter the data based upon many conditions and also see the raw data in a table. Feel free to change the columns and number of rows that display in the table.

The completed example is available from the following URL:

http://codepen.io/davetaz/pen/PPpGGe

7. Extension exercise ideas (in increasing order of challenge)

a) Change number of bedrooms to number of bathrooms

b) Add a new chart for price or price per sq ft.

c) Try adding a new type of chart or even a map?

Documentation and examples can be found at

https://github.com/dc-js/dc.js/blob/master/web/docs/api-1.6.0.md