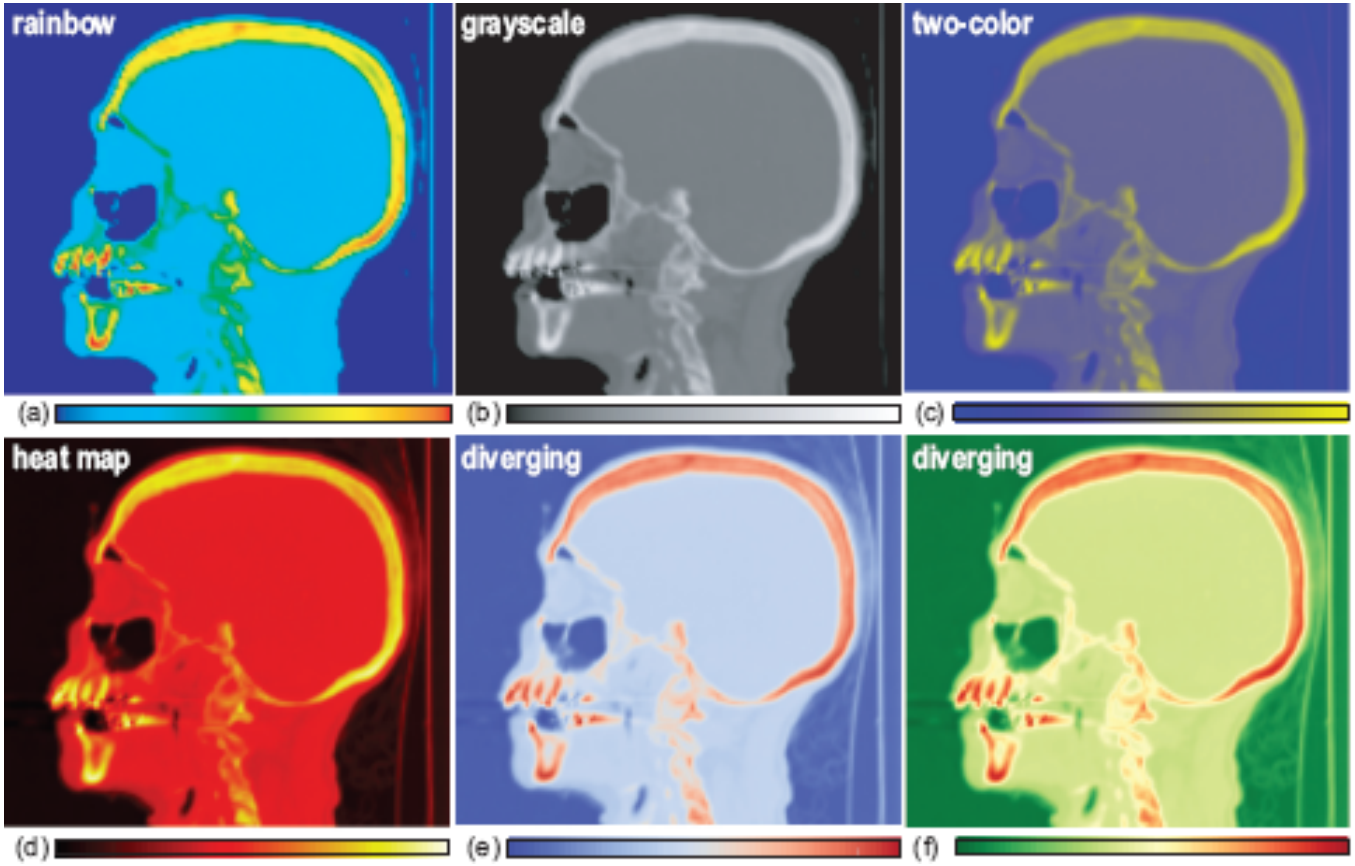
Visualization

- Communicate information graphically
- Python: the `matplotlib` library comprised the primary visualization pipeline that we studied
  - Set Marker and Line Style
  - Set Range of axes
  - Provide appropriate labels

Subdisciplines in Visualization

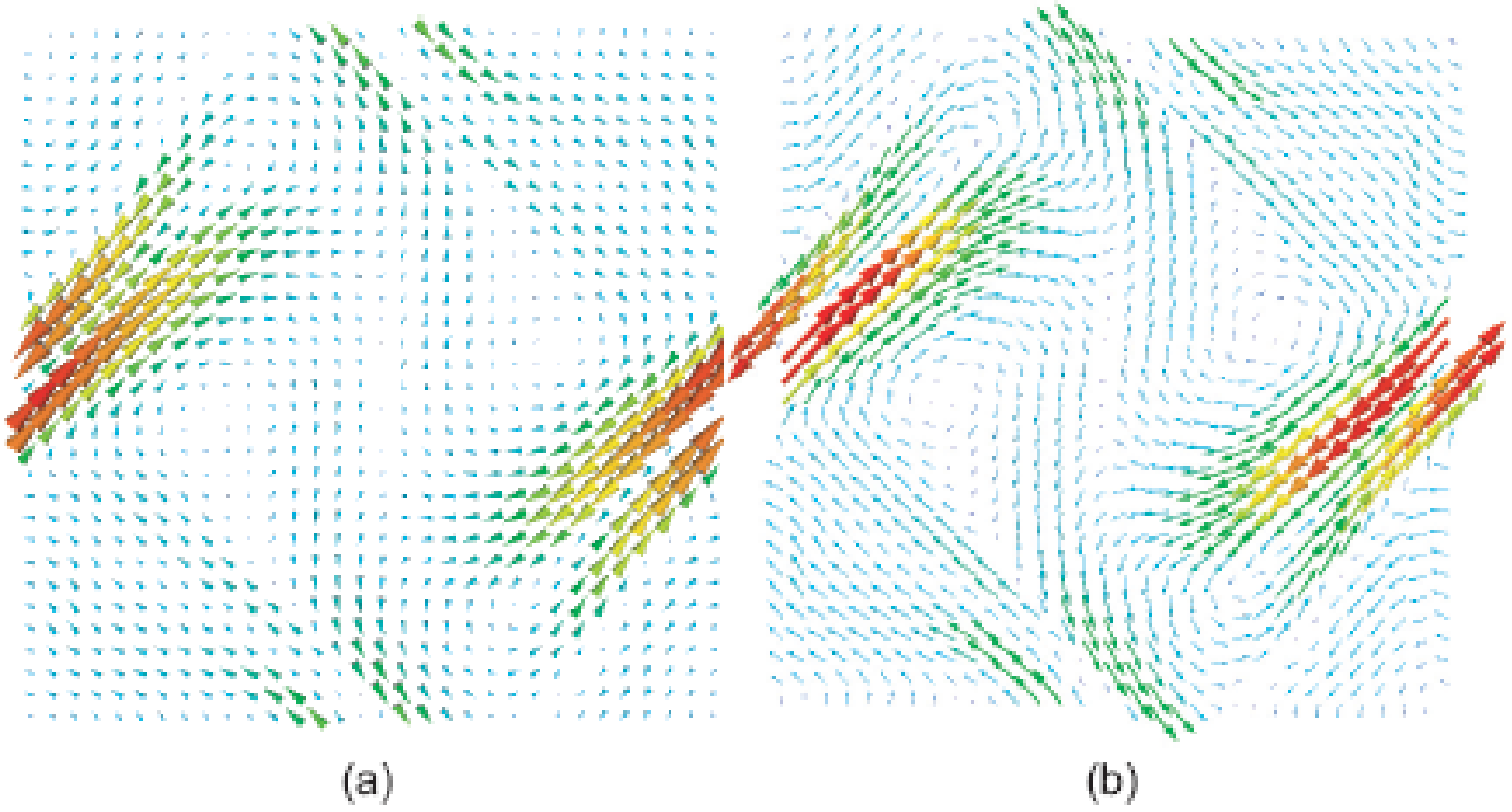
- *Scalar and Vector Visualization*: scalar functions, streamlines, etc.
- *Scientific Visualization* (scivis): datasets contain samples of continuous functions over subsets of Euclidean Space
- *Information Visualization* (infovis): visual representation of more abstract data such as graphs, text

Scalar Field Visualization



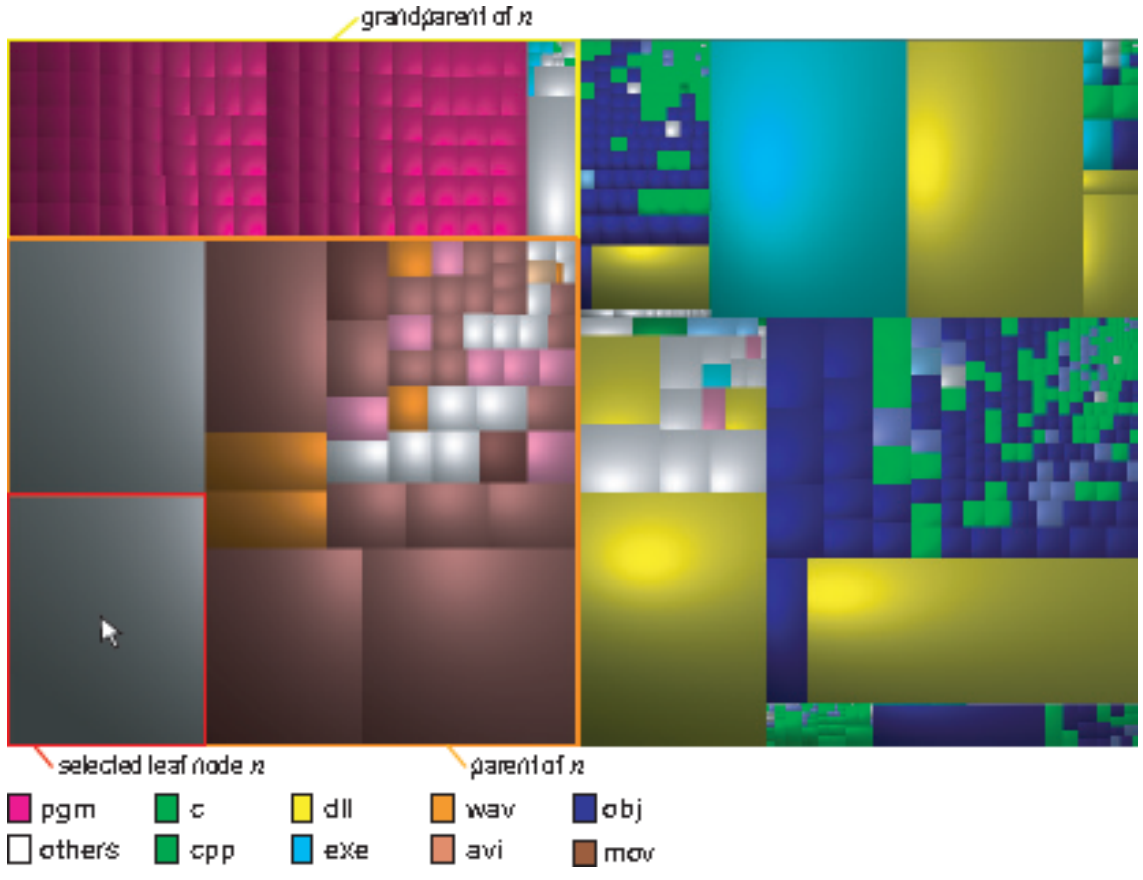
- Visualize a two-dimensional scalar field with different colormaps

Vector Field Visualization



- Render a velocity field with arrows or *glyphs*

Treemap Visulization



- Representation of tree data with a *tree map*

Data Visualizations with Javascript

- Industry standard for creating attractive visualizations with web content
- **SVG + D3**
- SVG: Scalable Vector Graphics
- D3: Data Driven Documents
  - Javascript Library created by Mike Bostock maintained at `d3js.org`
  - Enables developer to set the attributes of SVG graphical elements

Scalable Vector Graphics

- Two types of graphical format: *raster* and *vector*
- Raster: Image made of a fixed number of square pixels
- Vector: Specifies graphical primitives like circles, lines, rectangles
- SVG is a vector format that is also a web standard

Drawing Circles in a web page with SVG

- Create an `svg` in the body of the document, sets aside space for the graphics
- Specify width and height attributes and add `circle`

```
<svg width="600px" height="600px">

    <circle cx="100" cy="50" r="20"/>

</svg>
```

SVG Circle Attributes

- In addition to positioning the `circle` also has attributes for colors and other aspects of appearance
- *fill*: interior color of the circle
- *stroke*: line around edge of shape
- *stroke-width*: width of border line
- SVG element can contain multiple shapes

```
<svg width="600px" height="600px">
  <circle cx="100" cy="50" r="20" fill="blue"
        stroke="black" stroke-width="5"/>
  <circle cx="200" cy="50" r="20"/>

</svg>
```

Other SVG shapes

- Example of some different types of styling for SVG circles (from "Visual Storytelling with D3" by R. King 2014)

```
<svg width="500px" height="500px">
 <circle cx="50" cy="50" r="20" fill="BlanchedAlmond"/>
 <circle cx="100" cy="50" r="20" stroke="#0000FF" stroke-width="5" fill="none"/>
 <circle cx="150" cy="50" r="20" fill="rgb(139,0,139)"/>
</svg>
```

D3 Selections

- D3: Javascript Library, large text file containing javascript code
- Interact with the document through *selections*
- Make selection using the `d3` function and then manipulating the selections
- Select the first `p` element in the document and change its text

```
\\ inside a script tag

d3.select("p").text("Replace with this text!");
```

Chaining and Attributes

- Selections can also be chained (for example to select the first `p` tag within a `div` element)
- The attribute method `attr` can be used to change the attributes of a selection

```
var firstCircle = d3.select("circle");
firstCircle.attr("fill","blue");
```

HTML Template for Experimenting with D3

- Use the template below

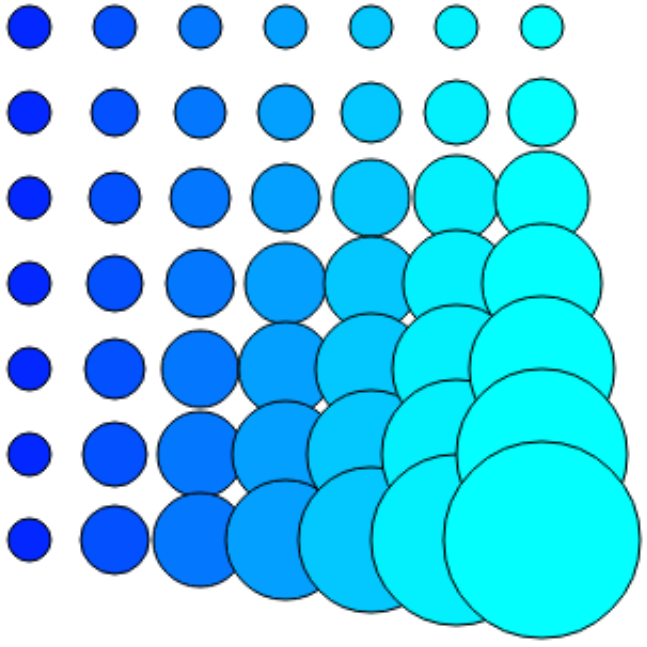
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3 Page Template</title>
        <script type="text/javascript" src="d3/d3.js"></script>
    </head>
    <body>

    <svg width="600px" height="600px"></svg>

        <script type="text/javascript">

            // javascript code

        </script>



    </body>
</html>
```

Simple D3 Example

- Simple D3 example that produces a grid of circles

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3 Page Template</title>
        <script type="text/javascript" src="d3/d3.js"></script>
    </head>
    <body>

    <svg width="600px" height="600px"></svg>

        <script type="text/javascript">
            var dataset = [40,80,120,160,200,240,280];

            for(var i=0;i < dataset.length;i++){
                for(var j=0;j < dataset.length;j++) {
                    d3.select("svg").append("circle")
                    .attr("r",((i * j)+10))
                    .attr("cx",dataset[i])
                    .attr("cy",dataset[j])
                    .attr("fill","rgb(0," + dataset[i].toString() + ",255)")
                    .attr("stroke","black");
                }
            }
        </script>
    </body>
</html>
```

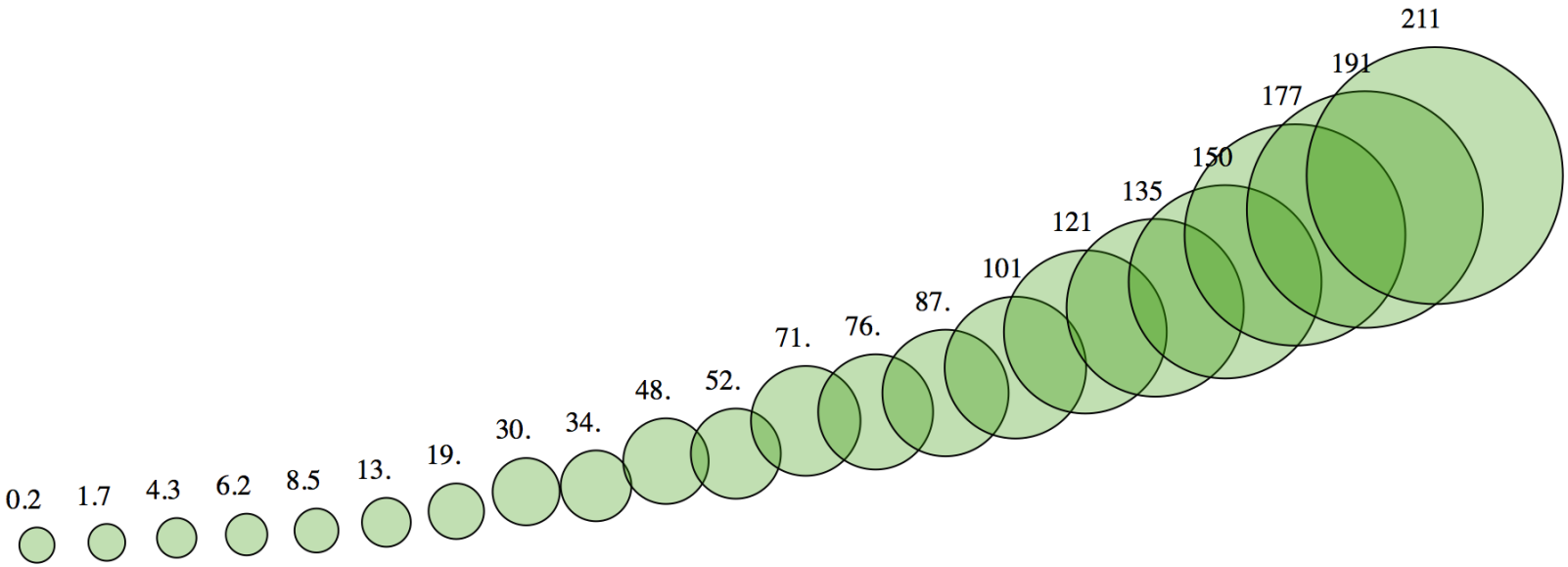
Iterating with for loops



- When we created this image we iterated over all of the elements and changed their attributes as needed
- D3 offers an alternative paradigms by introducing a new concept of *data-joins*

d3 Task

- Can render our dataset in multiple ways in same visualization



Example using a data join

- Visualization using data joins

```
<script type="text/javascript">

            d3.text("d3sample.csv", function(data) {
               var dataset = data.split(",");

               d3.select("svg").selectAll("circle")
                .data(dataset)
                .enter()
                .append("circle")
                .attr("fill","rgba(50,150,0,0.3)")
                .attr("stroke","rgb(0,0,0)")
                .attr("cx",function(d,i){ return 30 + 40 * i;})
                .attr("cy",function(d){ return 500 - (30 + 1 * d);})
                .attr("r",function(d){ return 10 + 0.3 * d;});


                d3.select("svg").selectAll("text")
                .data(dataset)
                .enter()
                .append("text")
                .attr("x",function(d,i){ return 30 + 40 * i;})
                .attr("y", function(d){return 500 - (30 + 1 * d) - (30 + 10 + 0.3 * d);}
                .attr("dx", -15/2)
                .attr("dy", "1.2em")
                .attr("text-anchor", "middle")
                .text(function(d) { return d.toString().slice(0,3);})
                .attr("fill", "black");
            });
        </script>
```

Data-Joins

- *Data-Joins* join some data to the elements of a webpage
- Usual elements such as `divs` or the SVG elements we discussed previously
- Data points are *bound* to elements of a webpage using a combination of the d3 methods `.data()` and `.enter()`

Example of a data-join

- Create an svg in the body of the document, sets aside space for the graphics

```
<script type="text/javascript">

  dataset = // array of javascript objects

  d3.select("svg").selectAll("circle")
  .data(dataset)
  .enter()
  .append("circle")
  .attr("fill","rgba(140,0,50,0.3)")
  .attr("stroke","rgb(0,0,0)")
  .attr("cx",function(d){ return 100 * d.x;})
  .attr("cy",function(d){ return 100 * d.y;})
  .attr("r",function(d){ return 10 * d.x * d.y;})

</script>
```

Steps of a data-join

- D3 chains data selection and manipulation operations together in a declarative way
- First select our root element with `d3.select("svg")`
- Use `.selectAll()` to select all of the children elements of a given type, even if they don't exist
- `.data()` and `.enter()` bind data to the placeholder elements created above
- If we call `.append()` next in the chain d3 create a new element of that type and with the data bound to it

Manipulating data with anonymous functions

- After calling the `.append()` method we can manipulate the DOM elements with `.attr`
- Give .attr an anonymous function as a parameter
- d3 will pass the data-point bound to the element to the anonymous function

```
.attr("cx",function(d){ return 100 * d.x;})
.attr("cy",function(d){ return 100 * d.y;})
```

Loading Data from an external file

- In D3 we can also load external data using either `d3.text()` or `d3.csv()`
- Simple case below we call our csv function and put our code in the callback
- For security purposes this may not work locally

```
d3.text("d3sample.csv", function(data) {
                    var dataset = data.split(",");

                    // rest of code
                });
```

Data-Joins

- *Data-Joins* join some data to the elements of a webpage
- Usual elements such as `divs` or the SVG elements we discussed previously
- Data points are *bound* to elements of a webpage using a combination of the d3 methods `.data()` and `.enter()`

Example of a data-join

- Create an svg in the body of the document, sets aside space for the graphics

```
<script type="text/javascript">

  dataset = // array of javascript objects

  d3.select("svg").selectAll("circle")
  .data(dataset)
  .enter()
  .append("circle")
  .attr("fill","rgba(140,0,50,0.3)")
  .attr("stroke","rgb(0,0,0)")
  .attr("cx",function(d){ return 100 * d.x;})
  .attr("cy",function(d){ return 100 * d.y;})
  .attr("r",function(d){ return 10 * d.x * d.y;})

</script>
```

`.enter()` method

- The d3 method `.enter()` followed by `.append()` is good when we want to create a static visualization
- However it might be desirable or required to create an *interactive* visualization
- In d3 DOM elements can be dynamically reconfigured using `update`, `.exit()`, and `.remove()`

Example: Datasets with different sizes

- Imagine you want to use d3 to bind to some data sets of different sizes, in this case, three different arrays
- Initially we just bind our dataset using `data` and `enter`

```
var datasetone=[10,20,30];
var datasettwo=[40,50,60,70];
var datasetthree=[80,90];

d3.select("svg").selectAll("text")
  .data(datasetone)
  .enter()
  .append("text");
```

Example: Updating

- Calling `.data` on previously existing DOM elements is update
- Bind data to existing elements
- `.enter()` and `.append` afterwards create a new element that has the extra datapoint bound to it

```
d3.select("svg").selectAll("text")
  .data(datasettwo)
  .enter().append("text");

// redraw
```

Example: `.exit()` and `.remove()`

- What happens when we perform an update and have some DOM elements left-over in our selection
- `.exit()`: Selects any element that has no data-point bound to it
- `.remove()`: Will remove the element from the document

```
// previously bound datasettwo

d3.select("svg").selectAll("text")
  .data(datasetthree)
   .exit().remove();

// redraw
```

Adding interactivity with `.on`

- The `.on()` method allows us to control the behavior when a specific element is clicked
- Give anonymous function that gets executed for the given event
- Set additional attributes of the element to indicate interactivity to the user

```
.attr("cursor","pointer")
.on("click",function(d,i) {
      console.log(i.toString() + " i");
      // additional javascript code
      });
```

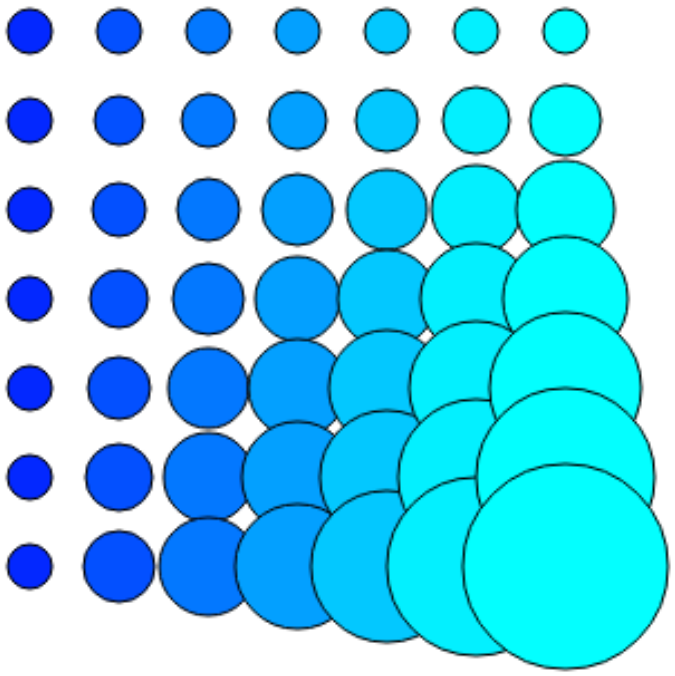
Referencing elements with identifiers

- d3 can also select specific elements on the page
- In this case:
  `d3.select("#firstsvg").selectAll("text")`

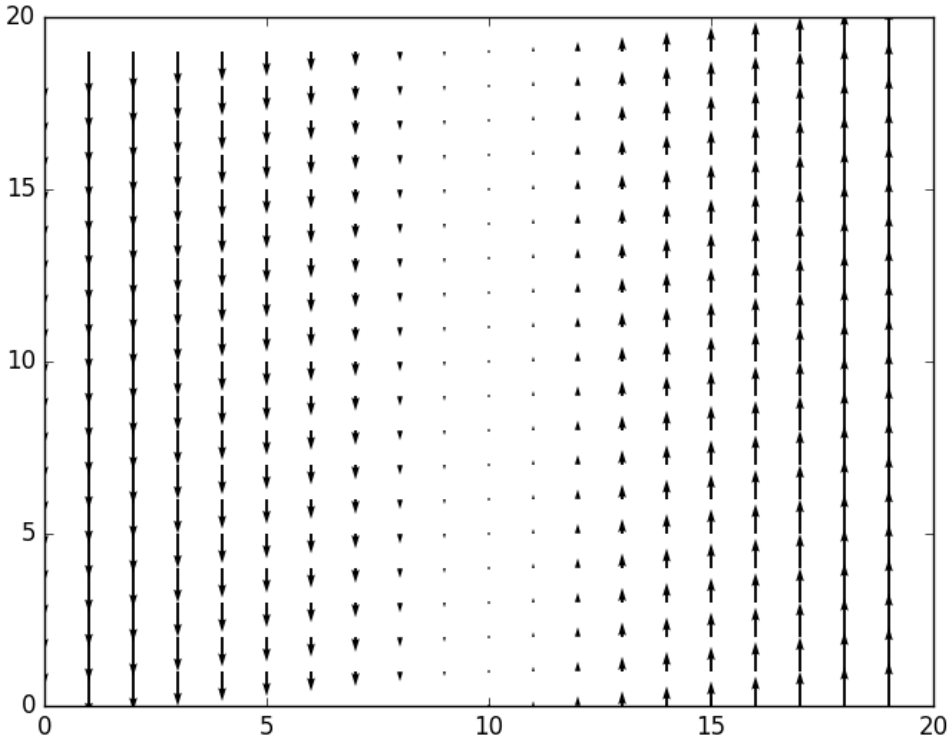
```
<svg id="firstsvg" width="200px" height="600px">
</svg>

<svg id="secondsvg" width="600px" height="600px">
</svg>
```

d3 Homework Task

- See if you can recreate the example below using javascript with d3 and svg
- Should not require anything more than a browser, d3 library (d3js.org), and a text editor
- Note that you can launch a simple web server using python with the command `python -m http.server 8000`



Scientific Visualization Example (Vector Field)



- Each point in the two-dimensional space has a vector attached to it. We can visualize it at discrete points.

$$f : (x, y) \rightarrow (dx, dy) \tag{1}$$

Showing a quiver plot in python

- Quiver plot is another name for a vector field visualization using arrows
- Each arrow comprises 4 pieces of data, (x, y, u, v)

```
import matplotlib.pyplot as plt
import numpy as np
from numpy import ma

X, Y = np.meshgrid(np.arange(-1,1, .1), np.arange(-1, 1, .1))
U = (0*Y)
V = X

plt.figure()
plt.quiver(U,V)
plt.show()
```

In-Class Exercise

- Visualize a particular vector field using python
- What vector field do we get in the particular cases of $(x, y, -y, x)$ and $(x, y, y, -x)$
- First two coordinates represent the base point and the last two coordinates are the components of the vector