D3 Hands on Session

Code4Lib 2016 Measuring your Metadata Preconference
Matt Miller (@thisismmiller) from @nypl_labs

Clone or download as zip this repository:

https://github.com/saverkamp/measure-metadata-workshop

Workshop Format

- We will walk through examples, first looking at the data side and then the visualization side.

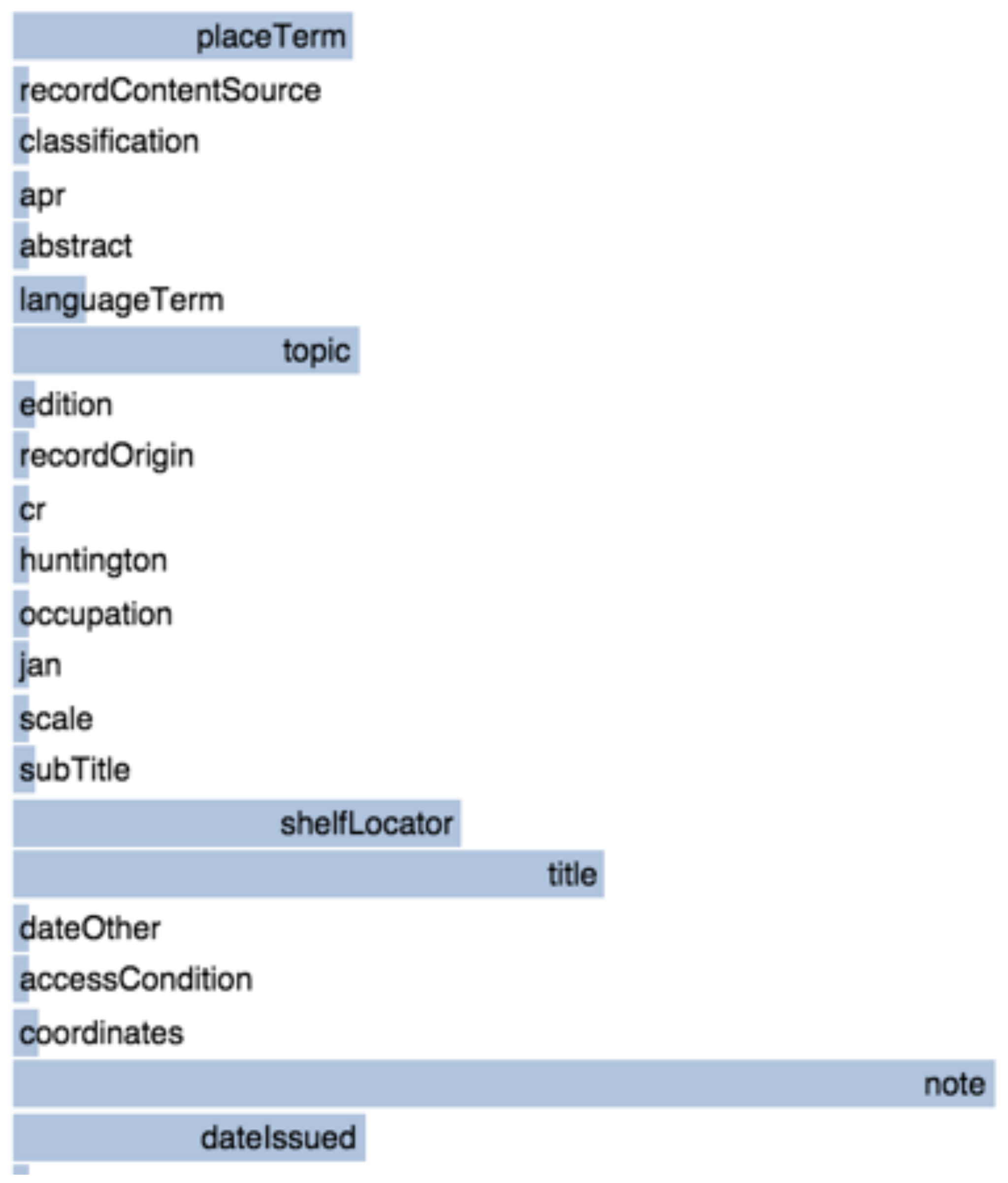
- Small work session after each example with a challenge. (gives us breathing room to work through problems or get things working)

Workshop Format

- Level of participation up to you!
- Work along with each example and try out the code or just follow along and get the higher level concepts
- Team up with another or a group to trouble shoot problems quicker
- Help each other with system problems or questions
- Please interrupt me if I’m not being clear.

Element Count: Simple Bar Chart

Out of 96877 elements.



Will built a file called: xml_field_count.json

```
[
	{
		"count": 3989,
		"tag": "placeTerm"
	},
	{
		"count": 5,
		"tag": "recordContentSource"
	},
	{
		"count": 20,
		"tag": “classification”
…
```

Element Count: Simple Bar Chart | VIZ

- Need a way to take that array of objects and draw something with them.

- d3.js a javascript library that allows you to bind data to HTML, SVG and CSS. Ties data to the DOM

- /d3_viz/bar_chart_simple/index.html

Make sure you have your local server running.

- Open a terminal/command line in the d3_viz directory and run:
  - python -m SimpleHTTPServer 8000
  - check http://localhost:8000 and make sure it is loading

/d3_viz/bar_chart_simple/index.html

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>Barchart simple</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <script src="../libraries/d3.min.js"></script>


        <!-- The styles used in the d3 SVG/HTML -->
        <style>

            .chart div {
                font: 20px sans-serif;
                background-color: lightsteelblue;
                text-align: right;
                padding: 5px;
                margin: 3px;
                color: black;
                overflow: visible;
                white-space: nowrap;

            }
        </style>
```

Traditional HTML opening, load the d3.js library define some styles

/d3_viz/bar_chart_simple/index.html

```
<!-- The javascript to build the viz -->
<script>

    // load the external data
    d3.json("../../data/pre_baked_data/xml_field_count.json", function(error, data) {

        if (error){
            alert("Error loading json file");
        }
```

d3 has a bunch of data loading methods, we use the .json() method to load the json file generated and pass it to the is function as data

/d3_viz/bar_chart_simple/index.html

```
//we want to know the total number of fields so we can set the domain the scale
//so loop through the data and add up the total number of fields
var totalFields = 0;

for (var aField in data){
    totalFields = totalFields + data[aField]['count'];
}
```

We want to add up some totals, so loop through the data and count

/d3_viz/bar_chart_simple/index.html

```
//now we can set the domain and range of the bar charts
//the domain is min max of our data (count of fields)
//the range is the translation into a linear value
//between 0 and x (here we are using 5000)
var linearScaleFunction = d3.scale.linear()
    .domain([0, totalFields])
    .range([0, 5000]);
```

d3 has a bunch of tools to help us work with data like the scale methods, convert our real numbers into an arbitrary scale to use in controlling the size/look of the visualization

/d3_viz/bar_chart_simple/index.html

```
d3.select("h1").text("Out of " + totalFields + " elements.")
```

similar to jquery you select based on tag/class/id name. Here we are just updating an existing h1 element with the total number of fields

/d3_viz/bar_chart_simple/index.html

```
d3.select(".chart")
    .selectAll("div")
    .data(data)
    .enter().append("div")
    .style("width", function(d) { return linearScaleFunction(d.count) + "px"; })
    .text(function(d) { return d.tag; });

});
```

Where the magic happens. D3 does a lot of method chaining.
Let’s walk through each method.

/d3_viz/bar_chart_simple/index.html

```
d3.select(".chart")
```

Select an exiting DOM element to work with.

/d3_viz/bar_chart_simple/index.html

```
d3.select(".chart")
    .selectAll("div")
```

Our DOM elements we are going to bind to the data. Div elements

/d3_viz/bar_chart_simple/index.html

```
d3.select(".chart")
    .selectAll("div")
    .data(data)
```

Tell D3 this is the data we want to bind

/d3_viz/bar_chart_simple/index.html

```
d3.select(".chart")
    .selectAll("div")
    .data(data)
    .enter().append("div")
```

The enter method is special. If there is more data than elements it then adds in those data items as new elements. This enter method chain defines how each one should be handled. In this case when there is more data than elements, add a div element.

/d3_viz/bar_chart_simple/index.html

```
d3.select(".chart")
    .selectAll("div")
    .data(data)
    .enter().append("div")
    .style("width", function(d) { return linearScaleFunction(d.count) + "px"; })
```

And set the width. The value is a function which is passed “d” which is the data for that element. In our case d is an object that has a .count and .tag property. The function returns the element count translated though the d3 scale. In this case the count is turned into the approximate pixel width.

/d3_viz/bar_chart_simple/index.html

```
d3.select(".chart")
    .selectAll("div")
    .data(data)
    .enter().append("div")
    .style("width", function(d) { return linearScaleFunction(d.count) + "px"; })
    .text(function(d) { return d.tag; });
```

And set the text of the div to the .tag value

/d3_viz/bar_chart_simple/index.html

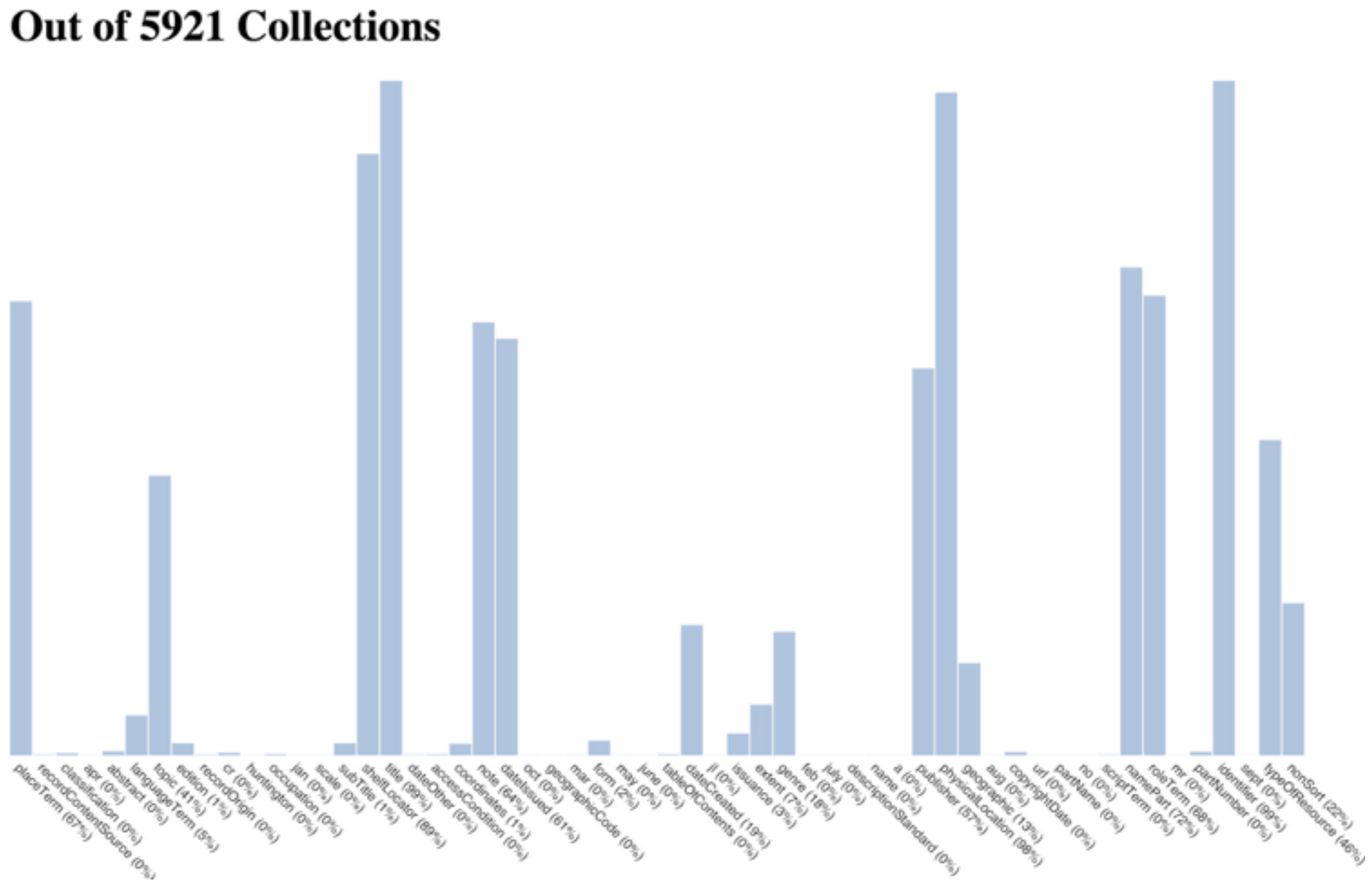
```
d3.select(".chart")
     .selectAll("div")
     .data(data)
     .enter().append("div")
     .style("width", function(d) { return linearScaleFunction(d.count) + "px"; })
     .text(function(d) { return d.tag; });

});
```

This is key to understanding d3. If this makes sense then you should be set!

/d3_viz/bar_chart_simple/index.html

Challenge: Edit the script so it includes the name of the element in the div and the total percentage that element represents out of all of the elements

Element Count: SVG Bar Chart



Element Count: SVG Bar Chart

- Now we are working with SVG not HTML
- SVG: Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation.
- Drawing shapes (eg, rect), paths, and other vectors

Element Count: SVG Bar Chart

- SVG Gotchas:
  - Coordinate system: 0,0 is the upper left.
  - Styling is close, but not the same.
    - In HTML CSS:
      - bacground-color: #000;
      - border: 1px solid #000;
    - In SVG Styling:
      - fill: #000;
      - stroke: #000;
      - stroke-width: 1;

Element Count: SVG Bar Chart

- SVG Gotchas:
  - Lots of layering. For example the <g> element (https://developer.mozilla.org/en-US/docs/Web/SVG/Element/g).
  - Some elements can be X/Y positioned but others like <g> can only be transformed.

/d3_viz/bar_chart_svg/index.html

```
var bar = chart.selectAll("rect")
    .data(data)
    .enter().append("rect")
    .attr("x", function(d,i) { return (barWidth * i); })
    .attr("y", function(d) { return height - linearScaleFunction(d.count); })
    .attr("height", function(d) { return linearScaleFunction(d.count); })
    .attr("title",function(d){return d.count})
    .attr("width", barWidth - 1)
```

Same as before but now we are adding SVG elements and not HTML DIVs. We can now use SVG styles and positioning

/d3_viz/bar_chart_svg/index.html

```
//draw the labels
 var labels = chart.selectAll(".labels")
      .data(data)
      .enter()
      .append("g")
          .attr("class","labels")
          .attr("transform", function(d,i) { return "translate("+((barWidth * i) + 3)+","+ (height+ 10) +"),rotate(45)"})
          .append("text")
              .text(function(d) { return d.tag + " (" + Math.floor(d.count/totalCollection*100) + "%)"; });
```

We can also use groups to put other elements into them and transform them, such as rotating

/d3_viz/bar_chart_svg/index.html

Challenge: The 0% bars are distracting and makes everything harder to read, how would we have d3 not render them?

More complex is the judgment arising from comparing the pair in Figure 3.

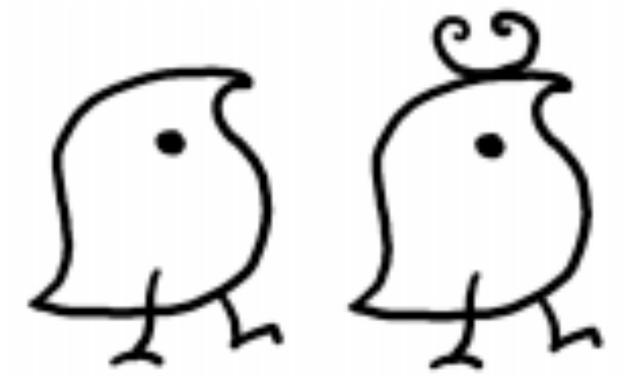


FIGURE 3. Elaboration.

MODS Quality Donuts

- Using the data generated by the MODS tools by Shawn & Sara.
- Quick at a glance check of the fields present in the MODS records for our divisions.

/d3_viz/mods_qual_donut/index.html

```
<!-- The javascript to build the viz -->
<script>

    // load the external data
    d3.csv("../pre_baked_data/sample_mods_scores.csv", function(error, data) {

        if (error){
            alert("Error loading css file");
        }
```

This time we are using the .csv() method to load the data. It will be transformed into an array of objects

/d3_viz/mods_qual_donut/index.html

```
//lets look at the data by divsion so we need to build a new object by division
var allDivsions = {};
for (var x in data){
    var aItem = data[x];
    //the division name will be our key
    //do we have this division in our data yet?
    if (!allDivsions[aItem.division]){
        allDivsions[aItem.division] = { total: 0, hasDate: 0 };
    }
    //see if this one has a date and update the object
    if (aItem.date == 1){
        allDivsions[aItem.division].hasDate++
    }
    //++ the total items for this division
    allDivsions[aItem.division].total++
}
```

Since this data is coming from a tool and not custom made for this viz we can manipulate it a bit in javascript to get it ready.

/d3_viz/mods_qual_donut/index.html

```
//a donut normally has more than one data point,
//so we need to add the number of things that do not have dates
allDivsions[division].noDate = allDivsions[division].total - allDivsions[division].hasDate

//these are the data points we are going to render
var dataPoints = [
    { label: "Has Date", count: allDivsions[division].hasDate},
    { label: "No Date", count: allDivsions[division].noDate}
]
```

And then for each donut we can make our data points, the two parts of the donut that we want to render, has date or has no date.

/d3_viz/mods_qual_donut/index.html

```
//a donut is just two arcs
var arc = d3.svg.arc()
    .outerRadius(radius - 20)
    .innerRadius(radius - 120);


var pie = d3.layout.pie()
    .value(function(d) { return d.count; });
```

We need to define some d3 tools to help build our donuts. The arc is just a SVG generator, you pass it some info and it will return all the SVG need to make an arc. The pie is a layout it is kind of like the scale methods we used. You pass it your data and it will convert it into a layout you can render.

/d3_viz/mods_qual_donut/index.html

```
                        //add in the arcs, use the pie layout to figure
out how big each data point should be
                        var g = svg.selectAll(".arc")
                             .data(pie(dataPoints))
                             .enter().append("g")
                             .attr("class", "arc");
```

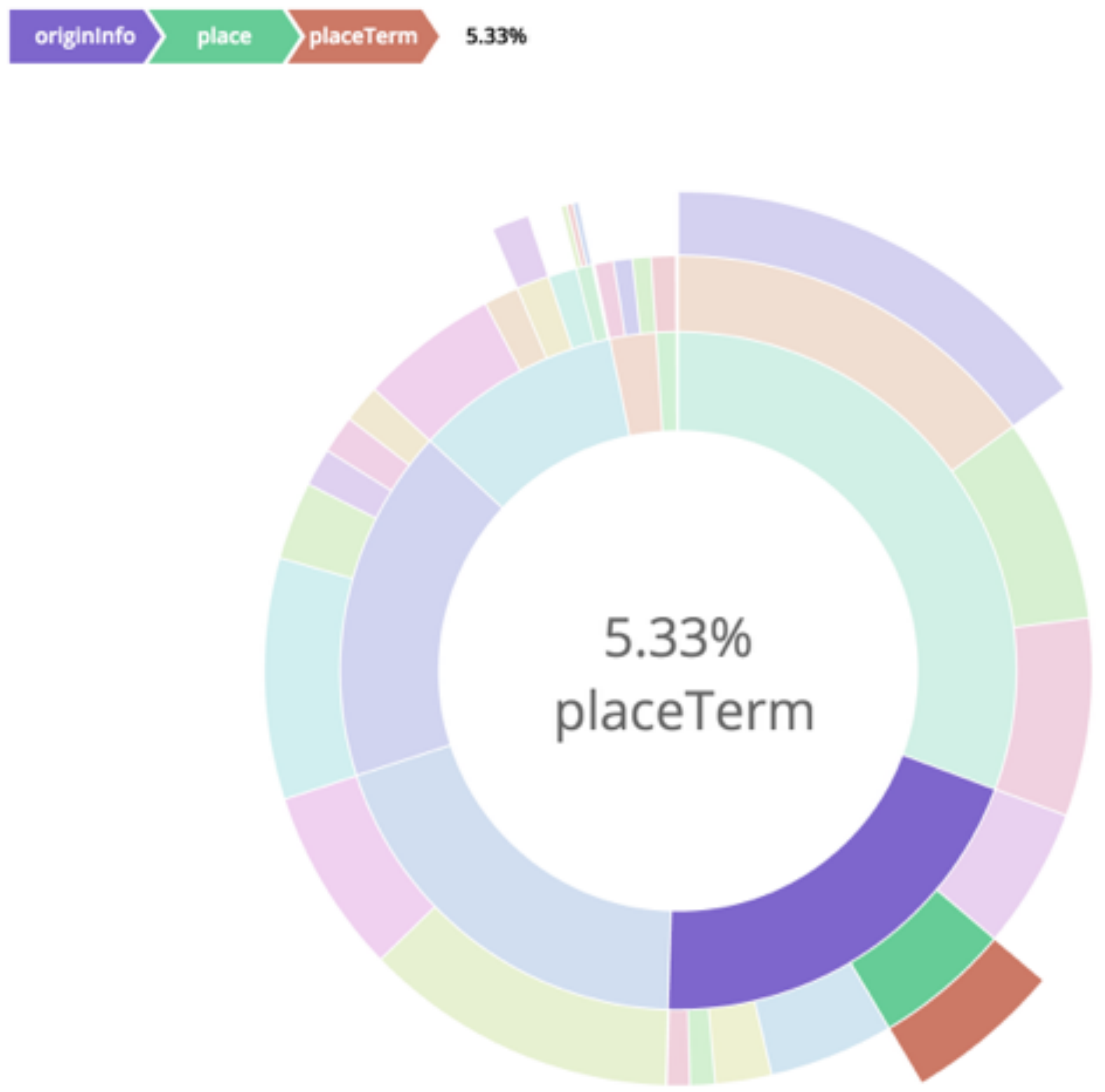
We “enter” the data but through the layout, which does the hard part of figuring out to layout everything.

/d3_viz/mods_qual_donut/index.html

```
//add in the arcs, use the pie layout to
//figure out how big each data point should be
var g = svg.selectAll(".arc")
    .data(pie(dataPoints))
    .enter().append("g")
    .attr("class", "arc");
```

The arc method also has some nice tools, like finding where the middle of the arc x/y is located.

Repurposing d3.js examples



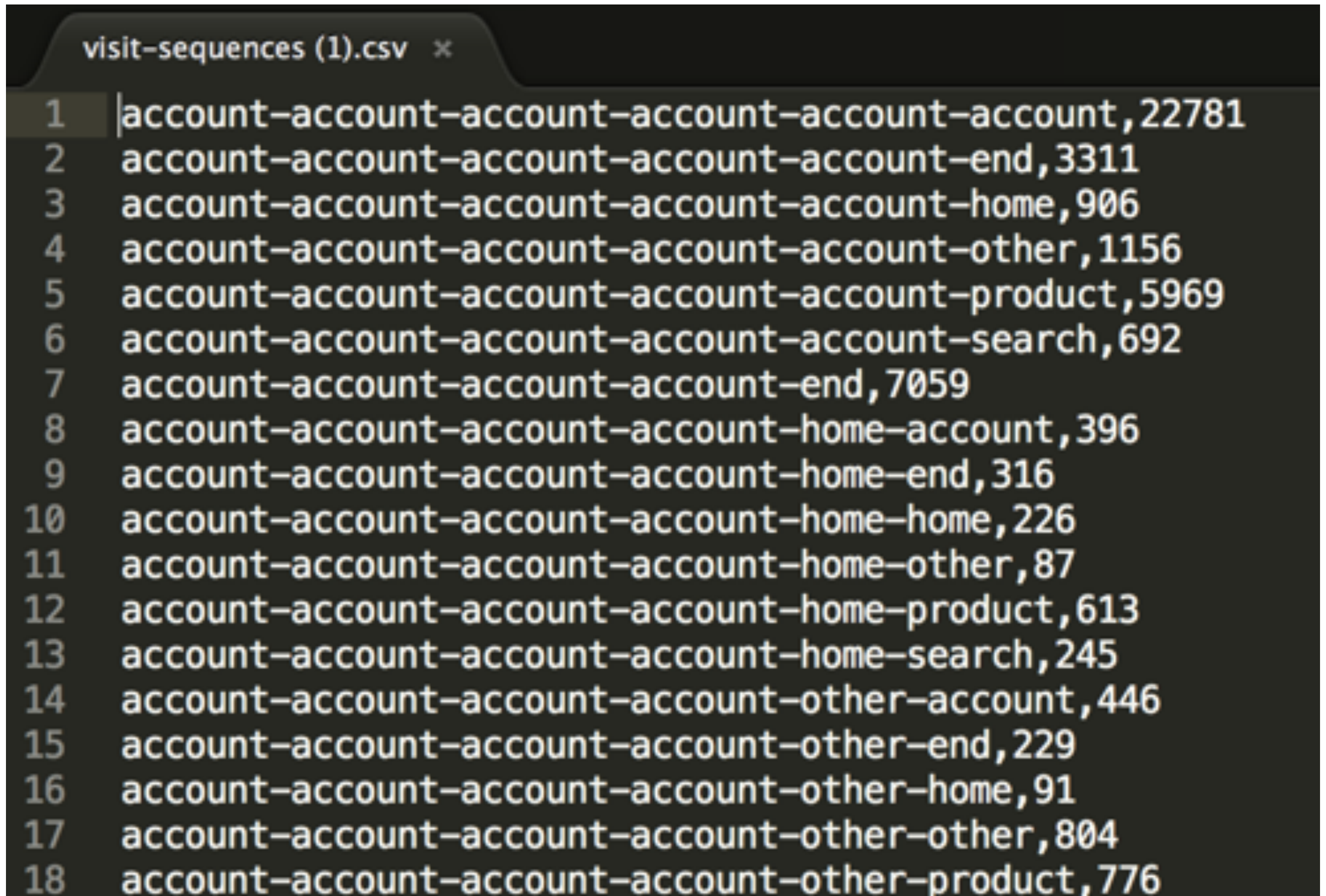
Repurposing d3.js examples

- Lots of good examples of d3.js layouts around the net. https://github.com/mbostock/d3/wiki/Gallery
- Let’s find one the better represents the hierarchy of XML elements
- http://bl.ocks.org/kerryrodden/7090426

Repurposing d3.js examples

Find where the data is being loaded

```
// Use d3.text and d3.csv.parseRows so that we do not need to have a header
// row, and can receive the csv as an array of arrays.
d3.text("visit-sequences.csv", function(text) {
  var csv = d3.csv.parseRows(text);
  var json = buildHierarchy(csv);
  createVisualization(json);
});
```



visit-sequences (1).csv ×

```
account-account-account-account-account-account,22781
account-account-account-account-account-end,3311
account-account-account-account-account-home,906
account-account-account-account-account-other,1156
account-account-account-account-account-product,5969
account-account-account-account-account-search,692
account-account-account-account-end,7059
account-account-account-account-home-account,396
account-account-account-account-home-end,316
account-account-account-account-home-home,226
account-account-account-account-home-other,87
account-account-account-account-home-product,613
account-account-account-account-home-search,245
account-account-account-account-other-account,446
account-account-account-account-other-end,229
account-account-account-account-other-home,91
account-account-account-account-other-other,804
account-account-account-account-other-product,776
```

So we need to make our elements into a dash delimited string

/d3_viz/network/index.html

- Same principle as before. Enter in new DOM elements for each item of data.
- But now define a layout that updates and repositions the SVG DOM based on the force layout.

/d3_viz/network/index.html

Define the layout and add in our nodes and links

```
var force = d3.layout.force()
    .charge(-120)
    .linkDistance(30)
    .gravity(0.05)
    .distance(25)
    .linkStrength(0.2)
    .size([width, height]);
```

```
force
    .nodes(graph.nodes)
    .links(graph.links)
    .start();
```

/d3_viz/network/index.html

Enter in the data and add the DOM elements

```
var node = svg.selectAll(".node")
    .data(graph.nodes)
  .enter().append("circle")
    .attr("class", "node")
    .attr("r", 5)
    .style("fill", function(d) { return color(d.group); })
    .call(force.drag);
```

/d3_viz/network/index.html

Tell it what to do on every “tick” or update of positions

```
force.on("tick", function() {
  link.attr("x1", function(d) { return d.source.x; })
      .attr("y1", function(d) { return d.source.y; })
      .attr("x2", function(d) { return d.target.x; })
      .attr("y2", function(d) { return d.target.y; });

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

/d3_viz/network/index.html

Challenge: Try to add something else besides a circle & Play with the force layout settings