Visualizing Library Data

Code4Lib 2015 Preconference
Matt Miller (@thisismmiller) from @nypl_labs

WIFI: PSAV Meeting Room
Password: osu2015

bit.ly/viz4lib2015

Clone or download as zip this repository:

https://github.com/thisismattmiller/viz4lib-2015

The Plan

- Intro
    - Why?
    - What tech skills are needed?
    - Workshop Format
    - The Data
- Let’s start simple.
    - Using d3 to bar chart XML element usage
        - HTML Bar Chart w/ d3.js
        - SVG Bar Chart w/ d3.js
        - Sunburst visualization, AKA repurposing a d3.js visualization
    - Break
    - Using d3.js to make networks!
        - Some elaborations you can do
    - Making Gephi networks.
    - Making BIG Gephi networks.
    - Hack Session/Show and tell

Why?

- A result of my experience last year presenting some visualization work I was doing at NYPL (bit.ly/nyplnetwork) and other projects (http://linkedjazz.org/network) and the reactions I got, especially from within my institution.

- Getting people excited about our data and how we can manipulate and use it.

Goals

- Three hours is very long yet incredibly short.
- We are not going to learn d3.js today.
- Get some basics down.
- Equip you with the basic data processing scripts and some rough templates to start working with your own data

Data?

- Data wrangling is as equally important in the visualization process
- Not circulation stats (although very important)
- Bibliographic metadata. MARC/MODS/EAD records
- Access to data?

Technical Skills Needed

- Okay with the command line
- Have python 2.7 installed (default on OSX) (if you want to run the example scripts)
- Okay with looking at/tweaking source code

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 or add a question to the ether pad in the current section, which I will be monitoring

Workshop Format

- Some Python scripts likely will not work out of the box for you. Libraries used:
    - lxml
    - BeautifulSoup
    - pymarc
- Let's not spend a ton of time sys-admining. We can get stuff working on your machine later. The data is already pre-baked so you do not need to run the scripts to follow along.

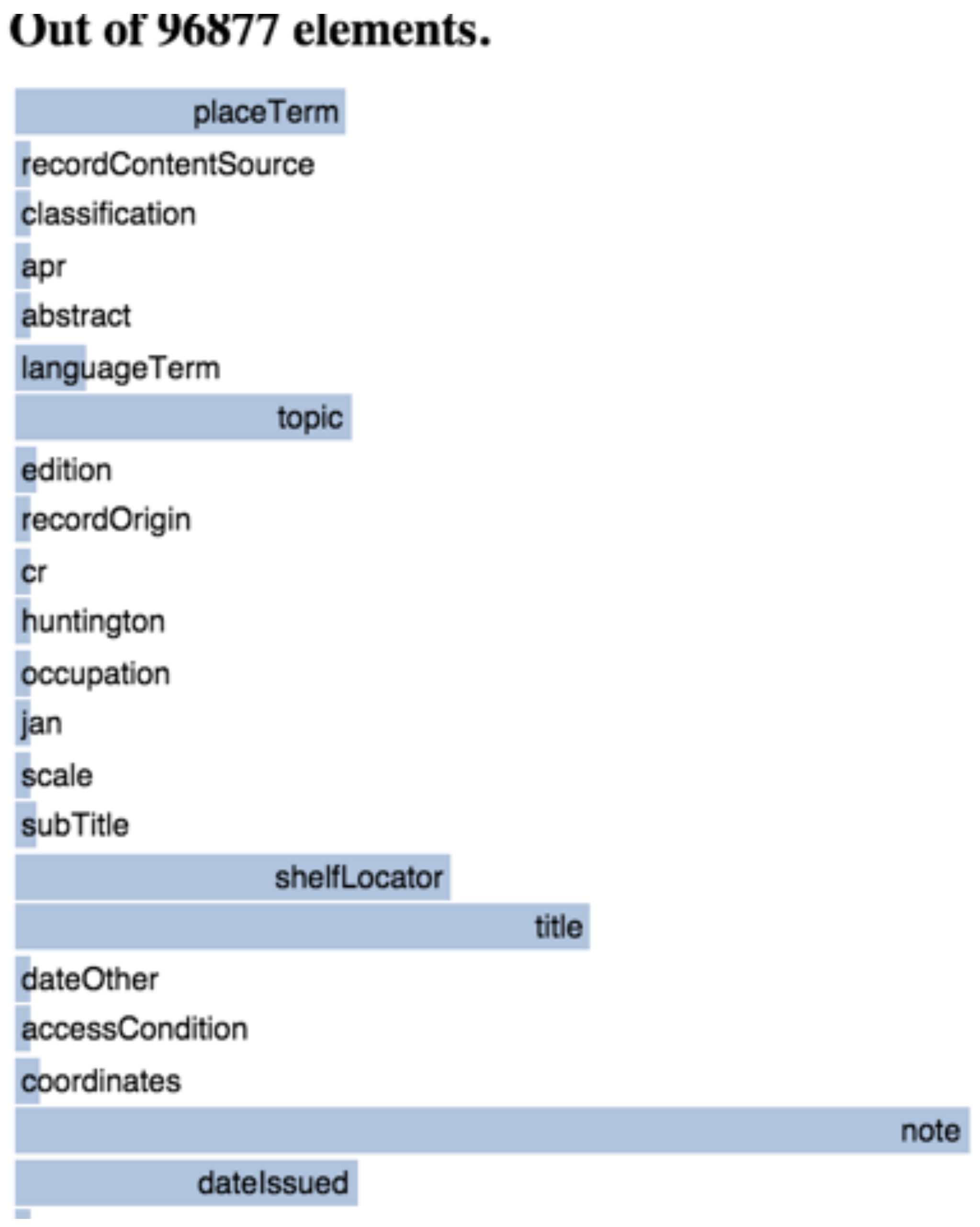
viz4lib2015 Repo

- d3_viz - holds the web-based visualization examples
- data - the data used in these examples
  - ead - 100 NYPL EADS files
  - mods - 100 NYPL MODS files
  - pre_baked_data - result files created using these scripts but on bigger datasets (1000s of EADs/MODs/MARCs)
- data_scripts - python scripts that process the data files
- gephi_render - a java jar library for large network visualizations

Optional Data Care Package

- To keep the viz4lib2015 repo size small there is only small datasets in the data directory
- You can replace this directory with this data folder to have larger datasets to work with.
- Grab it from:
    - http://bit.ly/viz4lib2015data (link also in either pad)
    - Flash drives up front

Element Count: Simple Bar Chart



Element Count: Simple Bar Chart

- What elements do we use in our metadata? MODS/EAD?
- We need to process some XML, parse out what elements are in use and compile the results.

Element Count: Simple Bar Chart | DATA

- Open a bunch of XML files
- Count the elements
- Write it out to a file that the visualization will use
- data_scripts/build_xml_field_count.py

build_xml_field_count.py

To run the script pass a directory full of XML files to it

```
matt$ python build_xml_field_count.py ../data/mods/
```

Let's look at the code.

```
for el in root.iter():
  if el != root:
    self.processElement(el,root)
```

build_xml_field_count.py

Will build 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 viz4lib2015 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 the there is more data than elements then elements need to enter and this method chain defines how that 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

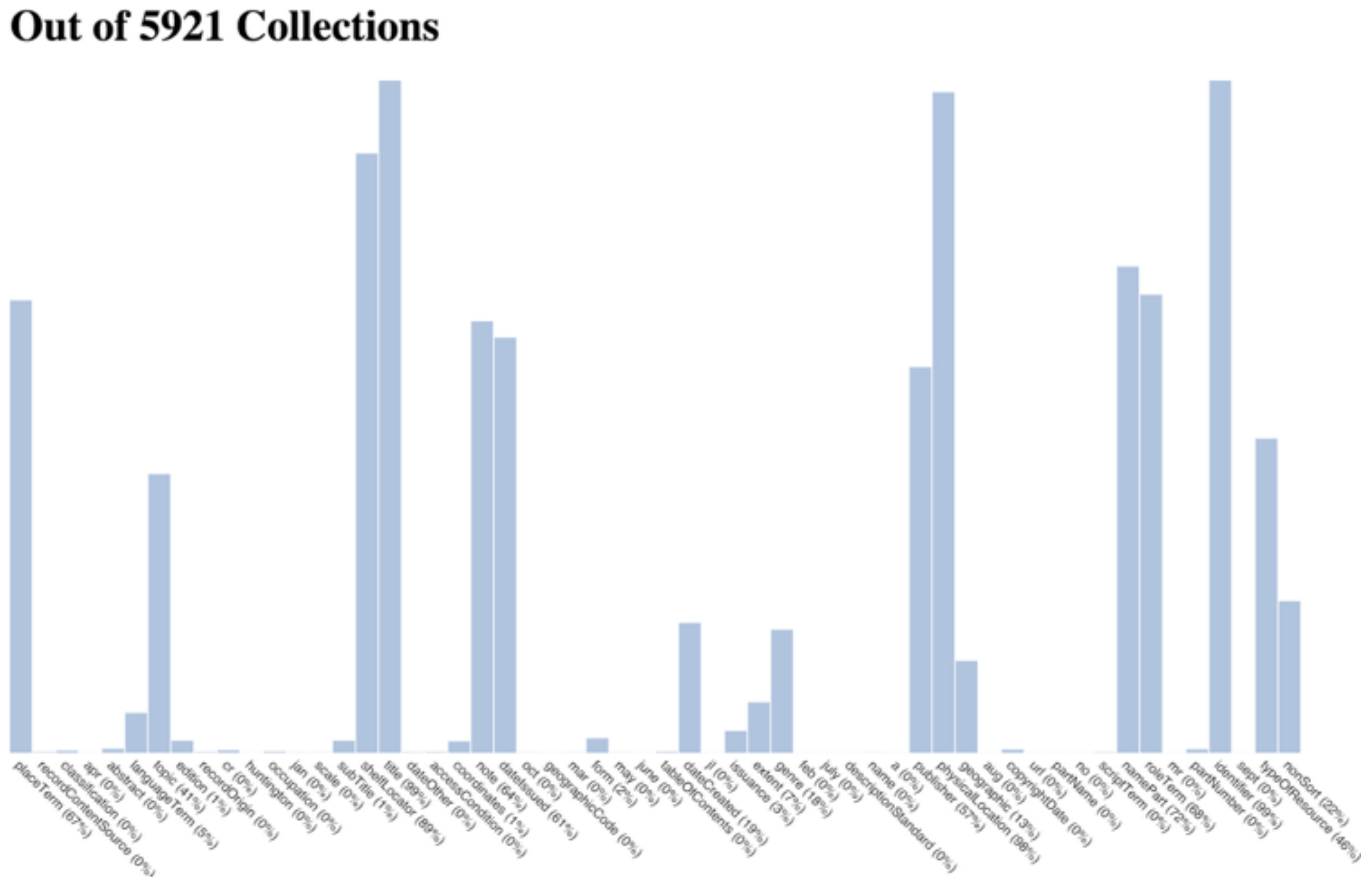
```
d3.select(".chart")
    .selectAll("div")
    .data(data)
    .enter().append("div")
    .style("width", function(d) { return linearScaleFunction(d.count) + "px"; })
    .text(function(d) { return d.tag; });

});
```

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.

Element Count: SVG Bar Chart | Data

- Very similar to the last script, but now we want to only count each occurrence of a element per file. So we know across our collections how often an element is being used.
- data_scripts/build_xml_field_count_per_file.py

build_xml_field_count_per_file.py

To run the script pass a directory full of XML files to it

```
matt$ python build_xml_field_count_per_file.py ../data/mods/
```

```
#store the basic count per file, so we need to check if we incremented it for this file or not
if tag not in self.elementUsePerFileFlag:
    self.elementUsePerFileFlag[tag] = False


#we have not yet set this file for having this element
if self.elementUsePerFileFlag[tag] == False:
    if tag not in self.elementUsePerFile:
        self.elementUsePerFile[tag] = {"tag" : tag, "count": 1}
    else:
        self.elementUsePerFile[tag]['count']+=1

    #don't count this one again for this file
    self.elementUsePerFileFlag[tag] = True
```

The only difference is that we keep track per file if we have counted an element.

/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

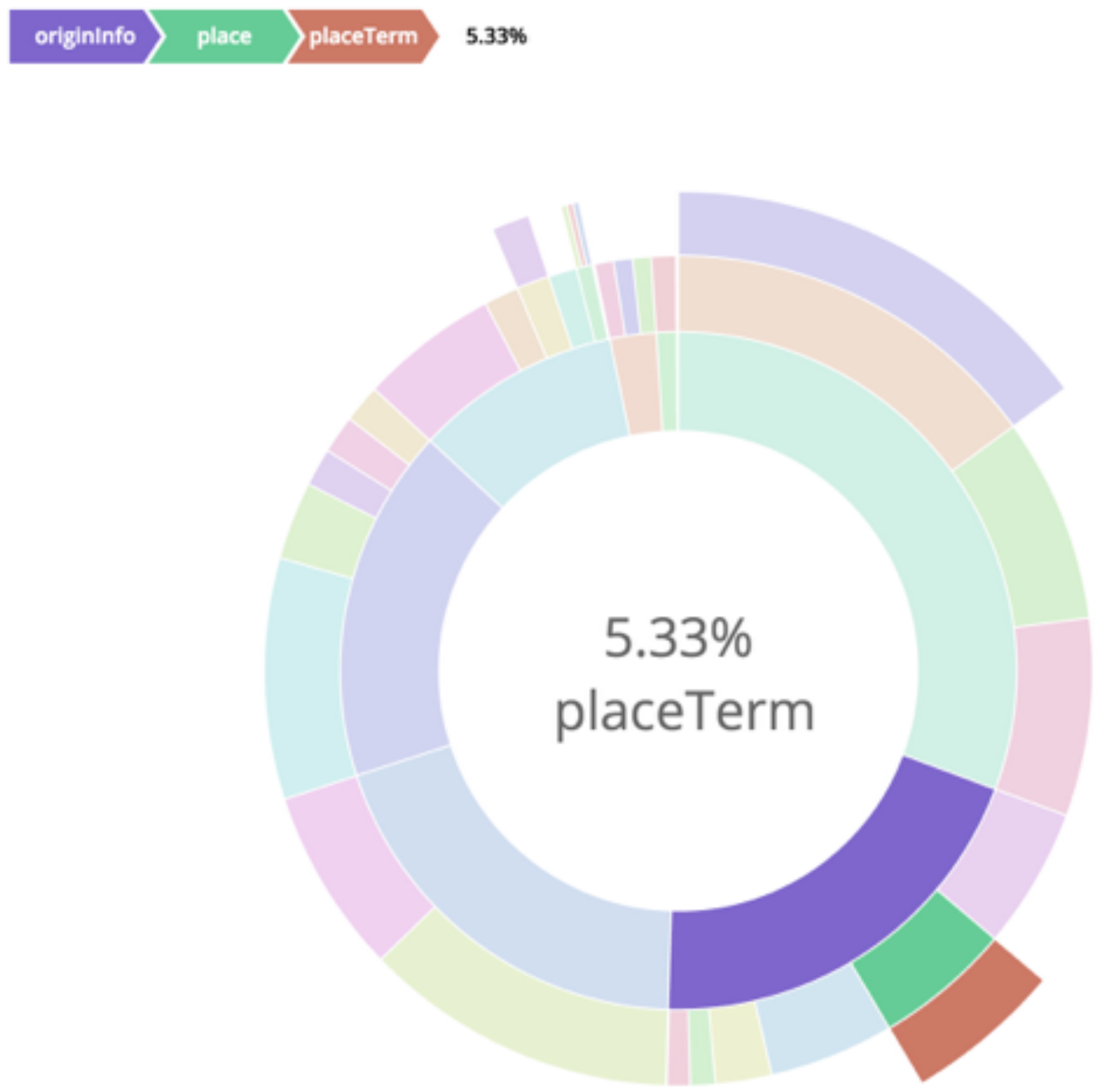
```
//draw the labels
 var labels = chart.selectAll(".labels")
     .data(data)
     .enter()
     .append("g")
         .attr("class","labels")
         .attr("transform", function(d,i) { return "translate("+((barWidth * i) + 3)+","+ (height+ 10) +"),rotate(45)"})
         .append("text")
         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

Repurposing d3.js examples



Repurposing d3.js examples

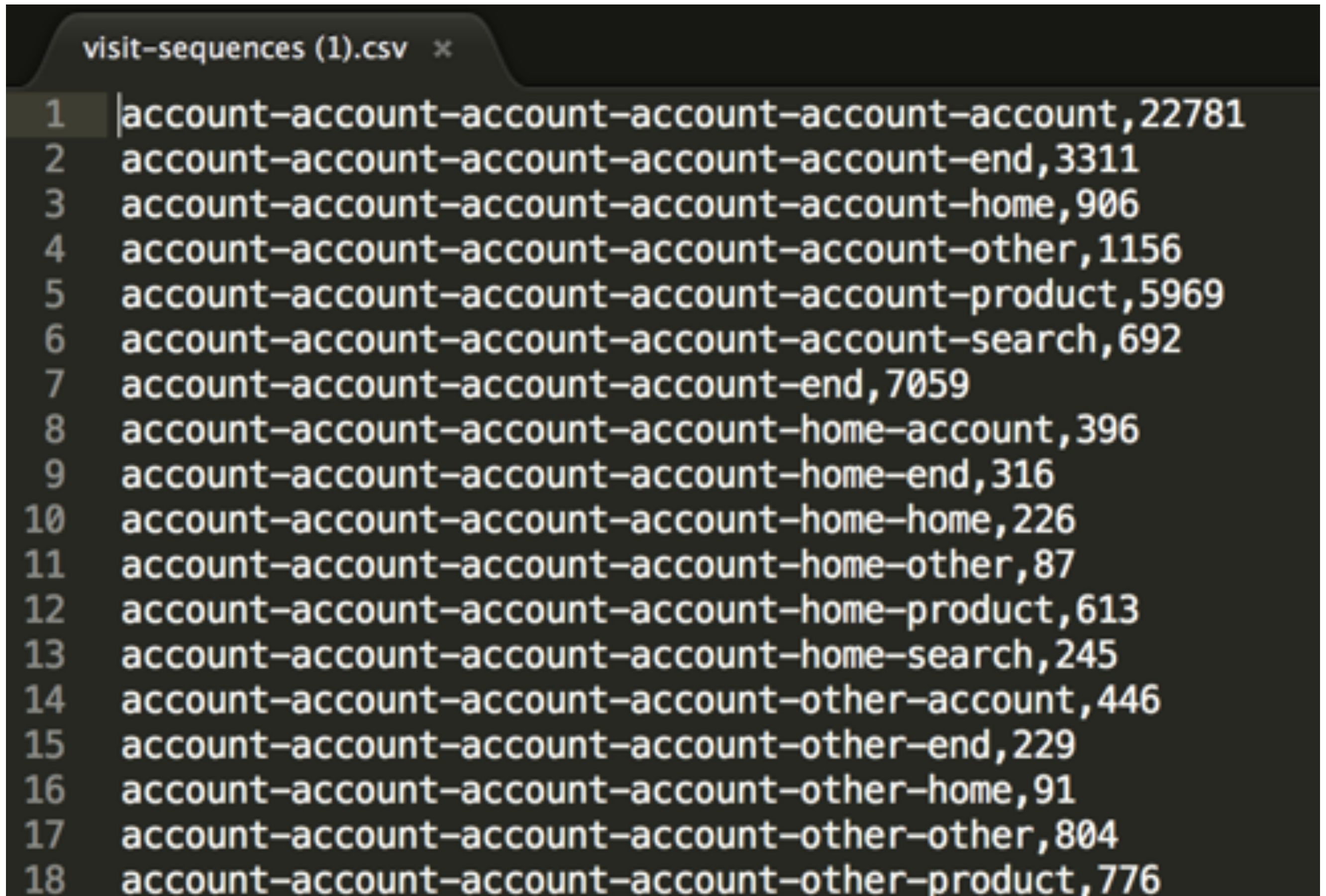
- Lots of good examples of d3.js layouts around the net. https://github.com/mbostock/d3/wiki/Gallery
- Let’s find one the better represents the hierarchy of XML elements
- http://bl.ocks.org/kerryrodden/7090426

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);
});
```

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



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
```

build_xml_field_hierarchy.py

To run the script pass a directory full of XML files to it

```
matt$ python build_xml_field_hierarchy.py ../data/mods/
```

```python
if parent != root and len(list(el.iter())) == 1:

            #loop back up through the parents storing the names
            while parent != root:
                hierarchy.insert(0, self.deNamespace(parent))
                parent = parent.getparent()

            #add the current level at the end
            hierarchy.append(tag)

            #make the string path so it looks something like: "originInfo-place-placeTerm"
            path = "-".join(hierarchy)

            #record it
            if path not in self.elementHierarchy:
                self.elementHierarchy[path] = {"path" : path, "count": 1}
            else:
                self.elementHierarchy[path]['count']+=1
```

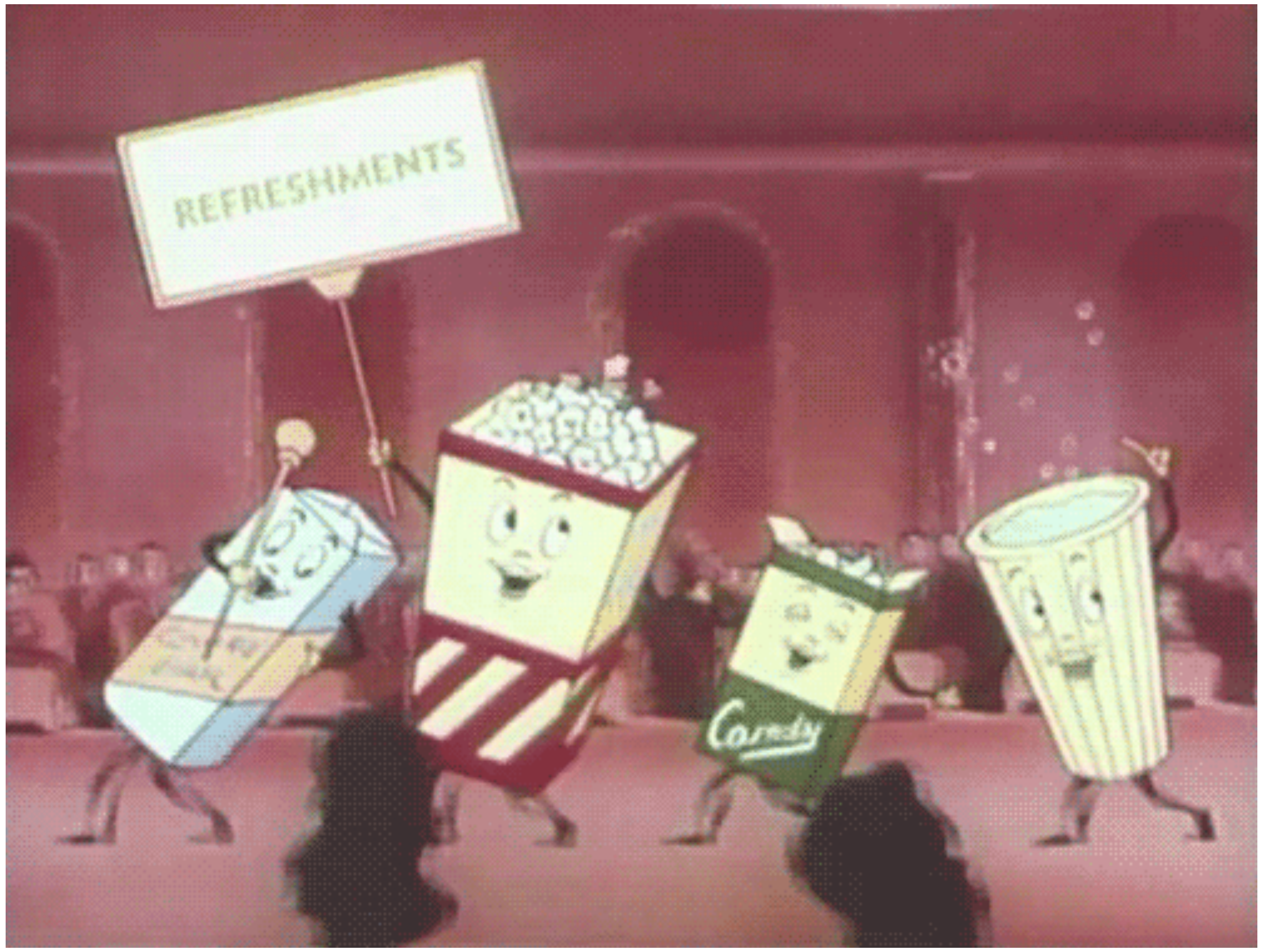
We now need to loop through the parent nodes and keep track of the entire hierarchy

Repurposing d3.js examples

5000 MODs:
/d3_viz/sunburst/index.html

10,000 EADs:
/d3_viz/sunburst_ead/index.html

10-15 Min Break



Try to install Gephi: http://gephi.github.io/

Networks

- Why?
- d3.js Networks
- Gephi Networks
- BIG Gephi Networks

d3.js Networks | Data

- Build the nodes and links(edges) for related items. In this case if the same subject is mentioned in two items there is a link between them.

- data_scripts/xml_to_d3_network.py

data_scripts/xml_to_d3_network.py

```
matt$ python xml_to_d3_network.py
```

- Open each XML file.
- Look for the designated element (persname)
- Build a dictionary of related elements
- Config:
    - You need to set the path to the XML (dataPath)
    - You can define what elements to look for with elements
    - edgeWeightLimit & subjectCountLimit controls the requirements for a node to make it into the graph (has to occur subjectCountLimit number of times with edgeWeightLimit connections)

/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

/d3_viz/network/index.html

- http://linkedjazz.org/network
- http://www.writingstudiestree.org/live/network

Gephi Networks

- Gephi is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs.

- If we build .gexf files (http://gexf.net/format/) we can use Gephi and its rendering engine to build large to xx-large networks. Beyond what d3 can handle.

data_scripts/marc_to_gexf.py

```
matt$ python marc_to_gexf.py
```

- Open each MARC file.
- Look for the designated fields and sub-fields
- Build a dictionary of related elements
- Config:
  - You need to set the path to the MARC (dataPath)
  - You can define what fields to look for with fieldMap
  - edgeWeightLimit & subjectCountLimit controls the requirements for a node to make it into the graph (has to occur subjectCountLimit number of times with edgeWeightLimit connections)

data_scripts/xml_to_gexf.py

```
matt$ python xml_to_gexf.py
```

- Open each XML file.
- Look for the designated element
- Build a dictionary of related elements
- Config:
    - You need to set the path to the XML (dataPath)
    - You can define what elements to look for with elements
    - edgeWeightLimit & subjectCountLimit controls the requirements for a node to make it into the graph (has to occur subjectCountLimit number of times with edgeWeightLimit connections)

Loding GEFX into Gephi

- Try with:
  - ead_network.gexf - NYPL EAD Subject network
  - harvard_small.gexf - Small subset of Harvard's Public MARC files
- Able to export as Sigma web presentation.

Large Gehpi

- The GUI cannot 100,000s of nodes. A group built a standalone library called gephi-toolkit that allows you to use the java classes without the GUI.

- catalog-network.jar is a wrapper for the tool-kit, specifically the Force Atlas 2 layout. You can feed it a GEFX file it will render out from the command line.

gephi_render/catalog-network.jar

```
matt$ java -jar catalog-network.jar -file ../path/to/file.gexf
```

More info on the settings:
https://github.com/thisismattmiller/catalog-network-java

Turning large GEXF into Images

- Once we have the final GEXF file it contains the X/Y, color and size of the nodes. We can write that out to large image, tile it and server it on the web. One approach to displaying a graph with 100,000s of nodes.