http://rubenhm.org/blog/

Research, Writings + Code

Coding Annoyances

FOMC dotplot with D3 and Python

03 January 2017 • Rubén Hernández-Murillo •

The FOMC's ***dotplot*** is part of the Summary of Economic Projections (SEPs) released 4 times per year along with the policy decision statement on the 2nd, 4th, 6th, and 8th meetings of the FOMC. It shows the views of each of the FOMC's participants regarding the end-of-year level of the fed funds rate over the next few years and in the longer run.

See page 3 of the projection materials (https://www.federalreserve.gov/monetarypolicy/files/fomcprojtabl20161214.pdf) from the December 2016 meeting.

Recreating this chart in Excel or some other program, such as Matlab or STATA, is not particularly straightforward. I wanted to recreate the plot using D3 (https://d3js.org/) but it wasn't clear how to prepare the source data to use D3's data-binding capabilities.

The html version (https://www.federalreserve.gov/monetarypolicy/fomcprojtabl20161214.htm#figure2) of the projection materials provides the following source data (also from the December 2016 meeting).

Midpoint of target range	*2016*	*2017*	*2018*	*2019*	*Longer run*
0.125					
0.250					

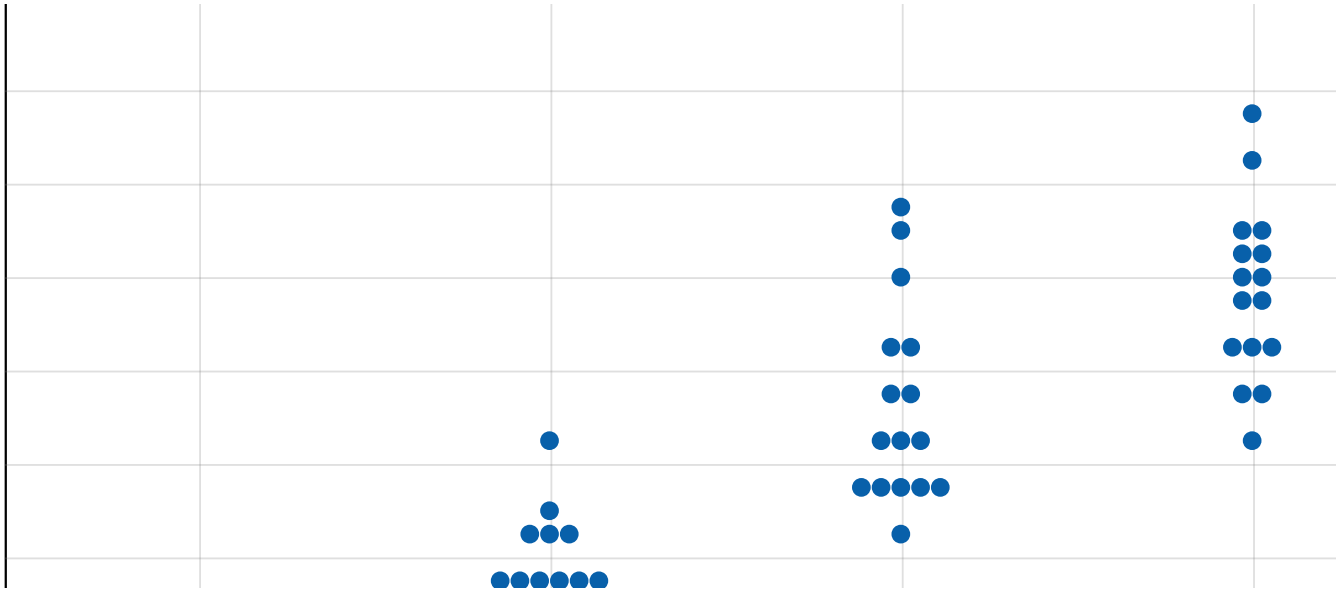
Midpoint of target range	*2016*	*2017*	*2018*	*2019*	*Longer run*
0.375					
0.500					
0.625	17				
0.750					
0.875		2	1	1	
1.000					
1.125		4			
1.250					
1.375		6			
1.500					
1.625		3	1		
1.750		1			
1.875			5		
2.000					
2.125		1	3	1	
2.250					
2.375			2	2	

Midpoint of target range	*2016*	*2017*	*2018*	*2019*	*Longer run*
2.500					1
2.625			2	3	
2.750					6
2.875				2	
3.000			1	2	7
3.125				2	
3.250			1	2	
3.375			1		
3.500					1
3.625				1	
3.750					1
3.875				1	

The dotplot is a scatterplot

This post from Len Kiefer's blog (http://lenkiefer.com/2016/06/22/Make-a-dotplot) made me realize I could treat the plot as a simple scatterplot. The key, then is to expand the count of participants at each rate level provided in the source table to be able to identify each of them with a circle.

Here is the final result using D3:



Open (/downloads/blog/2017-01-03-fomc-dotplot-with-d3-and-python/dotplot.html)

Collect and prepare the source data with Python

The simplest approach is to scrap the html code for the table using python and export the data to JSON format.

```
# Download data for constructing the dotplot.
# The source URL has the following structure:
# https://www.federalreserve.gov/monetarypolicy/fomcprojtabl20161214.htm
# (the file name seems to reference the date of the release)


import pandas as pd
import bs4
import requests


url = u'https://www.federalreserve.gov/monetarypolicy/fomcprojtabl20161214.htm'


# Download file
res = requests.get(url)
# Check for errors
try:
    res.raise_for_status()
except Exception as exc:
    print('There was a problem: %s' % (exc))

# Save file to disk
projectionFile = open('fomcprojtabl.htm','wb')
for chunk in res.iter_content(100000):
    projectionFile.write(chunk)
projectionFile.close()


# Make the soup
soup = bs4.BeautifulSoup(res.text,'lxml')

# Find the public tables
tables = soup.select('table[class="pubtables"]')


# Parse table to generate a pandas DataFrame
def parse_table(table):
    # Parse rows
    bdata = []
    rows = table.find_all('tr')
    for row in rows:
        # find first column header
        cols0 = row.find_all('th')
        cols0 = [ele.text.strip() for ele in cols0]
        cols = row.find_all('td')
        cols = [ele.text.strip() for ele in cols]
        cols = [ele for ele in cols0]+[ele for ele in cols]
        bdata.append(cols)
```

```
    # Convert to DataFrame
    bdata[0][0] = "MidpointTargetRange"
    bdata[0][-1]= "Longer Run"
    df = pd.DataFrame(bdata[1:], columns=bdata[0])
    df.set_index("MidpointTargetRange", inplace=True)
    return   df


# Parse the last table only
df = parse_table(tables[-1])

# Expand count of participants to an array to identify
# each dot individually

dfsize = df.shape
for rows in range(0,dfsize[0]-1):
    for cols in range (0,dfsize[1]):
        # print (df.ix[rows][cols])
        count = df.ix[rows][cols]
        if count:
            array = range(1,int(count)+1)
        else:
            array = [0]
        df.ix[rows][cols] = array

# Write data to json file resetting the index, per the following stackoverflow
 question:
# http://stackoverflow.com/questions/28590663/pandas-dataframe-to-json-without-
index

df.transpose().reset_index().to_json("dotplot.json",orient='records')
```

Here is the resulting JSON file (/downloads/blog/2017-01-03-fomc-dotplot-with-d3-and-python/dotplot.json).

The javascript code

Embedding the javascript in an html file, we have the following dotplot.html (/downloads/blog/2017-01-03-fomc-dotplot-with-d3-and-python/dotplot.html) file.

What's great is that when a new set of SEPs comes out, you need only to regenerate the updated json data file. The html file below need not be changed and the new dotplot will be updated.

The trickiest part was to figure out I had to use two scales for the x-axis: first an ordinal ***band scale*** for the different periods, and within each period, a ***point scale*** to organize the dots.

I also had to figure out how to center the dots in each period using the appropriate `translate` operation calculating where to start placing the dots along the point scale.

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>
circle.dots {
  fill: #0860aa;
  opacity: 1.0
}
g.tick line {
  stroke: black;
  opacity: 0.5;
}

g.x.axis text {
  font-size: 18px;
  fill: black;
  font-family: Helvetica;
}

g.y.axis text {
  font-size: 18px;
  fill: black;
  font-family: Helvetica;
}

g.grid line {
  stroke: grey;
  stroke-opacity: 0.5;
  shape-rendering: crispEdges;
}

g.grid text.y.label{
  font-size: 18px;
  fill: black;
  font-family: Helvetica;
}

g.grid path {
  stroke-width: 1px;
}
</style>
<body>
<div id="dotplot">

</div>
<script src="//d3js.org/d3.v4.min.js"></script>
<script>
// Reads data from a JSON file with the annual SEPs projections

// Auxiliary function
```

```
50. // Create array [1,2,...,N]
51. // http://stackoverflow.com/questions/3746725/create-a-javascript-array-containing-1-n
52. function nArray (N) {
53.     var dotArray = [];
54.     for (var i = 1; i <= N; i++) {
55.       dotArray.push(i);
56.     }
57.     return dotArray;
58. }
59.
60. function rArray (a,N) {
61.     var newArray = [];
62.     for (var i = 1; i <= N; i++) {
63.         newArray.push(a);
64.     }
65.     return newArray;
66. }
67.
68. var margin = {top: 50, right: 200, bottom: 50, left: 50};
69. var outerWidth = 1210 ;
70. var outerHeight = 600;
71. var innerWidth = outerWidth - margin.left - margin.right;
72. var innerHeight = outerHeight - margin.top - margin.bottom;
73. var topYAxisPadding = 10;
74.
75. var plotDotPlot = {};
76. plotDotPlot.svg = null;
77.
78. // Define SVG
79. plotDotPlot.svg = d3.select("#dotplot")
80.       .append("svg")
81.       .attr("width", outerWidth)
82.       .attr("height", outerHeight);
83.
84. // Read data
85. d3.json('dotplot.json', function (error, data) {
86.
87.
88.       if (error) {
89.         return pt.displayError(error);
90.       }
91.
92.       // Examine data
93.       console.log(data)
94.
95.       // Read years
96.       var years = []
97.       for (var i=0;i<data.length; i++) {
```

```
        years.push(data[i].index)
      }
      console.log(years)

      // Read rates
      // https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys
      var dataCols = Object.keys(data[0])
      // Remove index
      dataCols.splice(0,1)
      // Interest Rate Scale range provided in SEPs
      // make numeric
      var numScale = dataCols.map(function (d) {
        return +d;
      })

      console.log(dataCols)
      console.log(numScale)

      // Calculate max number of dots for the same rate
      var maxDots = d3.max(data, function (d) {
       |   'use strict';
          // Generate array of values
          // console.log(d)
          var dvalues = [];
          var ic;
          for (ic in dataCols) {
              var col = dataCols[ic]
              // if (col != "Period") {
                dvalues.splice(0,0,d[col].length)
              // }
          }
          // console.log(dvalues)
          // Spread operator (...) to calculate max of an array:
          // https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/max
          // console.log(Math.max(...dvalues))
          return Math.max(...dvalues);
        });

      // console.log(maxDots)

      // Generate array of serials
      // dotArray = [1, 2, ..., N]
      var dotArray = nArray(maxDots)
      console.log(dotArray)

      // Scales and Axes
      // Ordinal Band scale for X. Keep as string (don't use +d.index)
```

```
145.       plotDotPlot.x0 = d3.scaleBand()
146.              .domain(data.map(function (d) {return d.index; }))
147.              .rangeRound([0,innerWidth])
148.              .padding(0.1);
149.
150.       // Ordinal point scale to arrange dots in each band
151.       // domain is the largest number of dots in any given band
152.       // range is the bandwidth of the band
153.       plotDotPlot.x1 = d3.scalePoint()
154.               .domain(dotArray)
155.               .range([0, plotDotPlot.x0.bandwidth()])
156.
157.       // Y linear scale with hardcoded bounds
158.       plotDotPlot.y = d3.scaleLinear()
159.               // .domain(d3.extent(numScale)) // automatic
160.               .domain([0,5])                // harcoded
161.               .range([innerHeight,0]);
162.
163.       // Axes generators
164.       plotDotPlot.xAxis = d3.axisBottom(plotDotPlot.x0);
165.       plotDotPlot.yAxis = d3.axisRight(plotDotPlot.y).ticks(10);
166.
167.       // Gridlines: https://bl.ocks.org/d3noob/c506ac45617cf9ed39337f99f8511218
168.       // gridlines in x axis function
169.       function make_x_gridlines() {
170.           return d3.axisBottom(plotDotPlot.x0)
171.               .ticks(10)
172.       }
173.
174.       // gridlines in y axis function
175.       function make_y_gridlines() {
176.           return d3.axisLeft(plotDotPlot.y)
177.               .ticks(10)
178.       }
179.
180.       // Add svg
181.       plotDotPlot.chartGroup = plotDotPlot.svg.append("g")
182.               .attr("transform","translate("+margin.left+","+margin.top+")");
183.
184.       plotDotPlot.chartGroup.append("g").attr("class","x axis")
185.                .attr("transform", "translate(0,"+innerHeight+")")
186.                .call(plotDotPlot.xAxis);
187.
188.
189.       plotDotPlot.chartGroup.append("g")
190.                 .attr("class","grid")
191.                 .attr("transform", "translate(0," + innerHeight + ")")
192.                 .call(make_x_gridlines()
193.                   .tickSize(-innerHeight)
```

```
                  .tickFormat("")
                );

      plotDotPlot.chartGroup.append("g").attr("class","y axis")
               .attr("transform", "translate("+innerWidth+",0)")
               .call(plotDotPlot.yAxis);

      plotDotPlot.chartGroup.append("g")
               .attr("class","grid")
               .call(make_y_gridlines()
                  .tickSize(-innerWidth)
                  .tickFormat("")
                );

      // Add labels to axes
      plotDotPlot.chartGroup.append("text")
              .attr("class", "y label")
              .attr("text-anchor","end")
              .attr("x", innerWidth)
              .attr("y", -topYAxisPadding)
              .text("Percent, end of year")


      // Add circles
      for (var rows = 0; rows < years.length; rows++)  {

          var rateData = data[rows]
          plotDotPlot.chartGroup.selectAll("fomc "+"M"+years[rows])
                .data(rateData)
                .enter().append("g")
                .attr("class","fomc "+"M"+years[rows])

          for (var cols = 0; cols< dataCols.length; cols++) {
               // Calculate length of dot array
               var lenDotArray = rateData[dataCols[cols]].length
               var mclass = "dots "+"M"+ rows+" N"+cols
               plotDotPlot.chartGroup.selectAll(mclass)
                     .data(function (d) {
                        return d=rateData[dataCols[cols]];
                     })
                     .enter()
                     .append("circle")
                     .attr("class",mclass)
                     .attr("transform", function(d) {
                        // Center dots in each band:
                        // Caculate translate distance as half of diff with ban
    dwidth
                        var transDim = plotDotPlot.x0(years[rows])+(plotDotPlot.
    .bandwidth()-plotDotPlot.x1(lenDotArray))/2;
```

```
                        return "translate("+transDim+",0)";
                     })
                    .attr("cx", function (d) {
                        return plotDotPlot.x1(d);
                    })
                    .attr("cy", function (d) {
                          return  plotDotPlot.y(+dataCols[cols]); })
                    .attr("r", function (d) {
                           if (d >0){
                             return 5;
                           } else {
                             return 0;
                           }
                     })
            }
       }

})
</script>
```

Follow for updates.

(http://rubenhm.org/feed.xml) @rubenhm_org