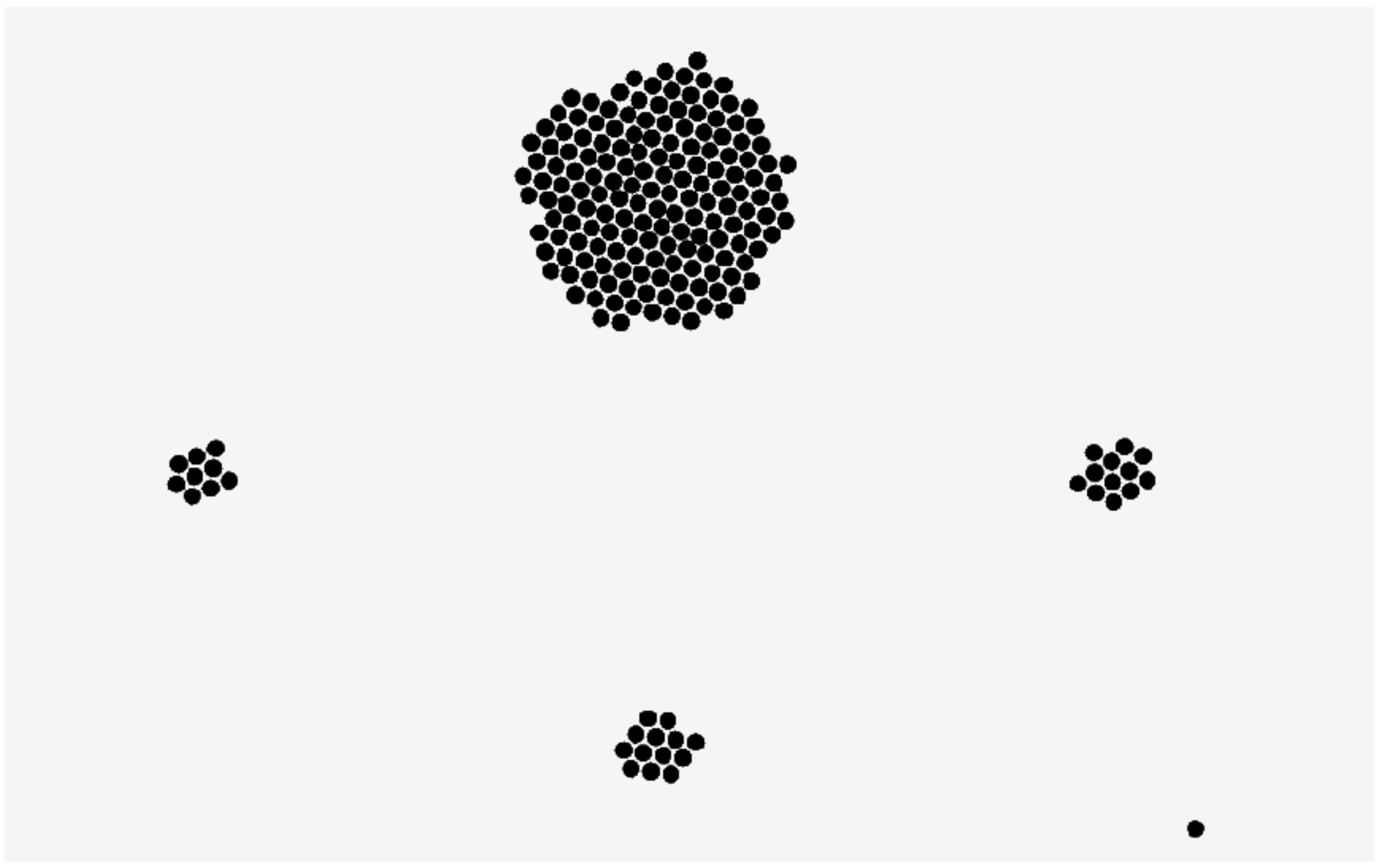
Make a Moving Bubbles Chart to Show Clustering and Distributions

By Nathan Yau

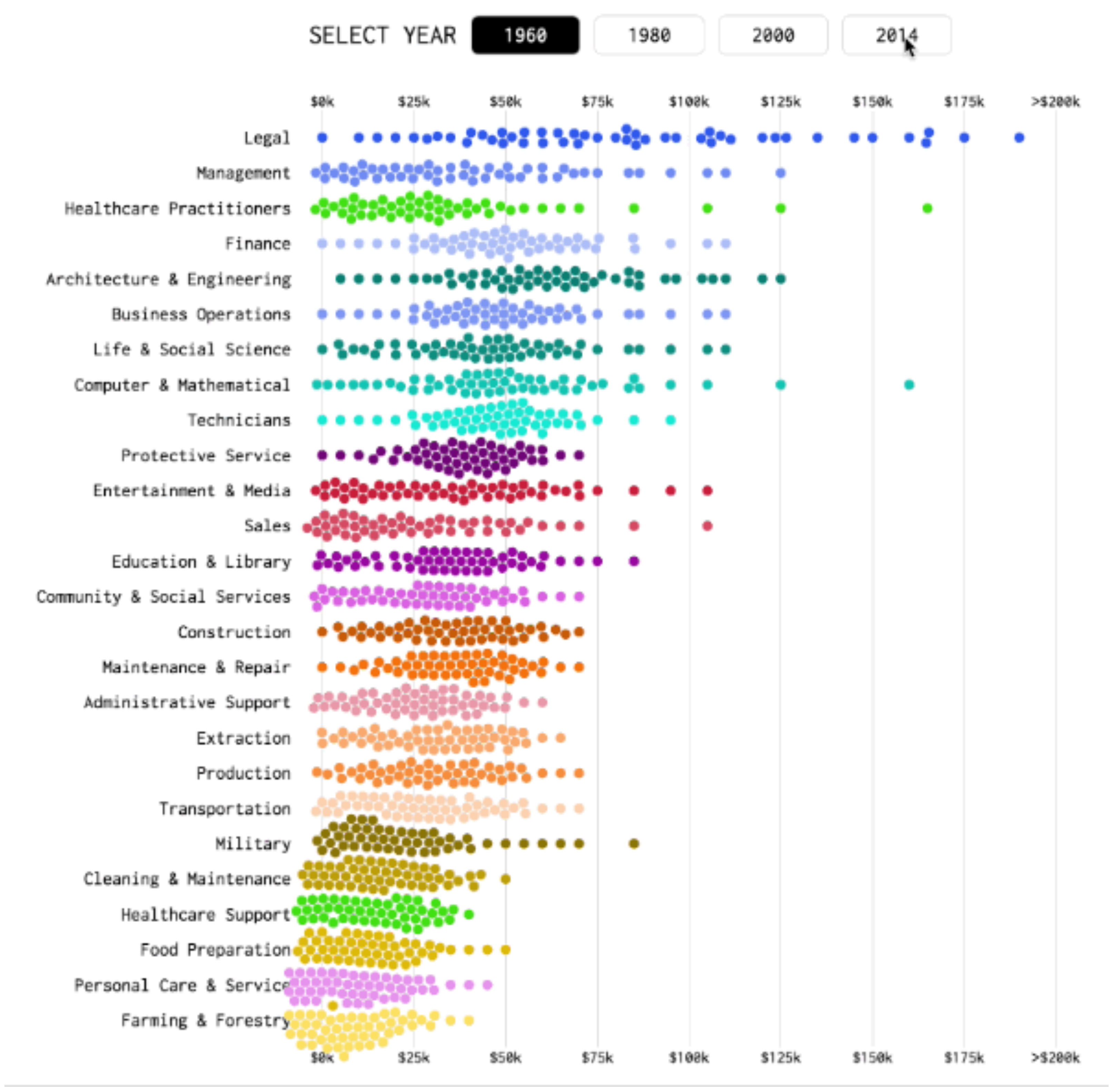
Use a force-directed graph to form a collection of bubbles and move them around based on data.

Demo
Download Source

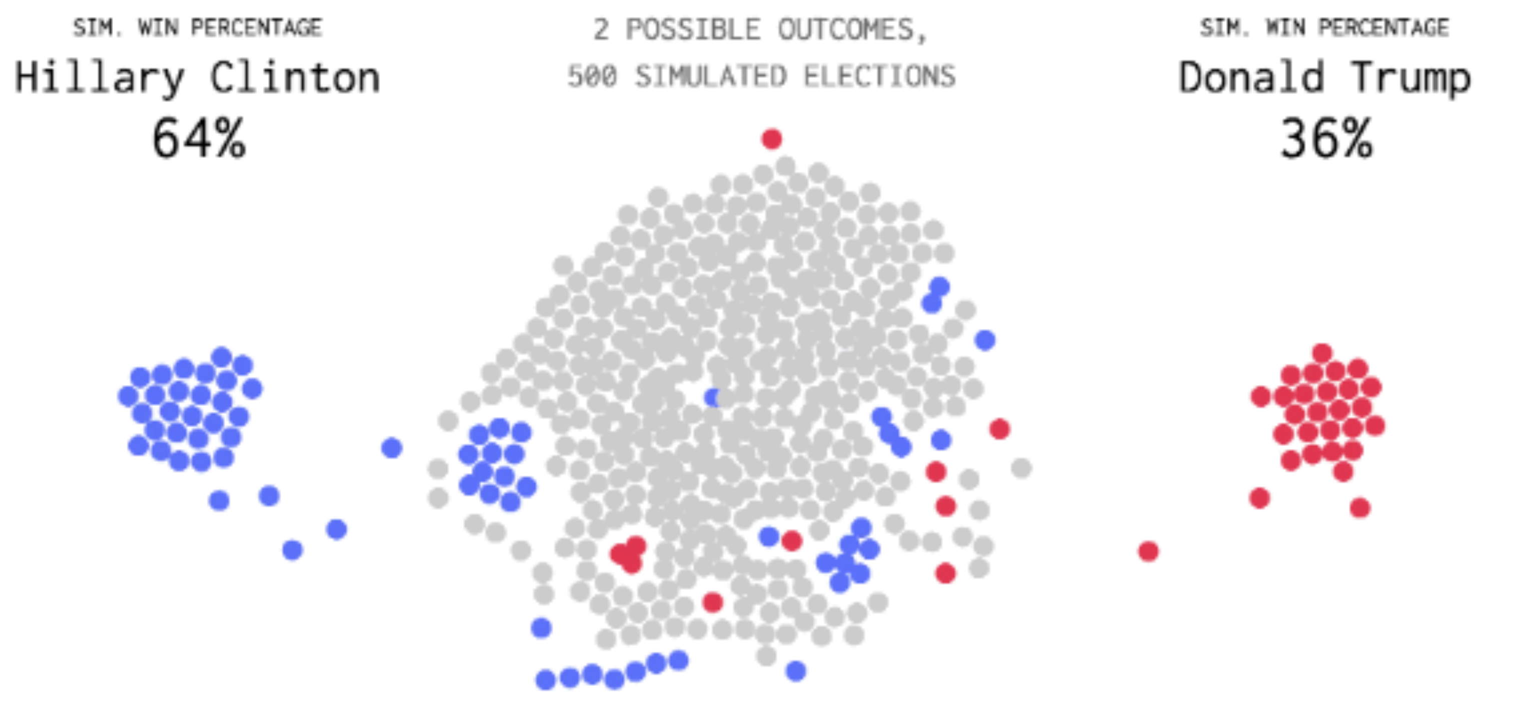


I've been playing around with moving bubbles lately. While they are perhaps not the most perceptually accurate way to show data, they do seem to help a lot of people grab on to the concept of distributions and how individual items, things, and events can add up to a bigger picture.

I used it show the simulated days of 1,000 people, household types in America, and even used them in lieu of histograms to show income distributions.



I also used the method to simulate election probabilities.



In this tutorial, you'll make a variant of this one.

Setup

This tutorial uses d3.js, or Data-Driven Documents, created by Mike Bostock. The software recently went up a version to v4, but this one uses v3. The JavaScript library is included in the js folder of the tutorial download, so you don't have to download anything extra.

To develop on your computer, you need access to a localhost or equivalent on your browser. If you don't have this yet, this tutorial for Windows PC users might help, and this one for OS X users should do the trick.

Finally, if you're new to code and/or JavaScript, I highly encourage you to work through Mike's introduction tutorials before getting into this one. Otherwise, the logic behind the code that follows can seem completely foreign. The getting started tutorial is good, and follow that with the basic bar chart.

Unlike R, which gets you a chart or many charts with a single line of code, JavaScript development is a bit more involved. I created several index.html files to make it easier to follow along with the code.

All that said, let's get started on those moving bubbles.

Start the Page

I have a project template folder that I typically start from.Most d3.js projects start in the same way with some basic HTML, loading the library file, and setting some margins. Start with the page header, which will look familiar if you've ever made an HTML page.

```
<!DOCTYPE html>
<head>
    <title>Moving Bubble Tutorial</title>
    <link rel="stylesheet" href="style/style.css" type="text/css" media="screen"
/>
    <meta charset="utf-8">
</head>
```

Then add a body and a div holder where you will place the chart.

```
<body>
<div id="main-wrapper">
    <div id="chart"></div>
</div><!-- @end #main-wrapper -->
```

Right now it's empty, and you won't see anything appear in your browser window when you load index00.html, but the div with the chart id is where you place all your shapes. You do this with d3.js, which you can load as shown below.

```
<script src="js/d3-3-5-5.min.js"></script>
```

Now for some JavaScript, which you write inside a `script` tag. Define the dimensions of the chart, along with the margins, which follows Mike Bostock's margin convention.

```
var margin = {top: 16, right: 0, bottom: 0, left: 0},
    width = 950 - margin.left - margin.right,
    height = 700 - margin.top - margin.bottom;
```

Then create the SVG object.

```
var svg = d3.select("#chart").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

Again, you'll do this or something similar for just about every project using d3.js.

Clustered Force Layout

Open **index01.html**.Like I said earlier, there are so many d3.js online examples, especially by the creator Mike Bostock, that you rarely start a project from scratch. Usually the challenge is to break down a visualization into components so that you can work on mini-tasks. Find examples that match your components (or are at least similar) and work on piecing things together.

In this case, I started with this clustered force layout example by Bostock.

This doesn't give us everything we want for the final chart, but it gets us about half way there. There's collision detection so that nodes don't overlap, nodes can be clustered into different groups, and you can make each group converge around a certain point.

So it's a good starting point.

I'm not going to go into much detail for this section, because most of it is from Bostock's example, but I'll cover the parts that need changing to shift from example to final result.

First, set a few variables, as shown below.

```
var node_radius = 5,
    padding = 1,
    cluster_padding = 10,
    num_nodes = 200;
```

The `node_radius` variable is, as you'd expect, the radius of each node. The `padding` is the space between each node within a cluster, in pixels. `cluster_padding` refers to the minimum space between nodes that belong to different clusters. Finally, `num_nodes` is the total number of nodes to create for this visualization.

Then, we want four clusters, each in a different position. Also, each cluster will be a different color. We set the x-y positions and colors like so.

```
// Foci
var foci = {
    "toppos": { x: 475, y: 150, color: "#cc5efa" },
    "leftpos": { x: 225, y: 300, color: "#29bf10" },
    "rightpos": { x: 725, y: 300, color: "#23cdc7" },
    "bottompos": { x: 475, y: 450, color: "#eb494f" },
};
```

Initialize `num_nodes` node objects all in the top position. The added `Math.random()` to the x and y coordinates is to add some randomness to the initial positions, so that the force layout doesn't get mixed up.

```
// Create node objects
var nodes = d3.range(0, num_nodes).map(function(o, i) {
    return {
        id: "node" + i,
        x: foci.toppos.x + Math.random(),
        y: foci.toppos.y + Math.random(),
        radius: node_radius,
        choice: "toppos",
    }
});
```

Open **index02.html**.Create the force layout, which uses the node objects in its initialization.

```
// Force-directed layout
var force = d3.layout.force()
    .nodes(nodes)
    .size([width, height])
    .gravity(0)
    .charge(0)
    .friction(.91)
    .on("tick", tick)
    .start();
```

For more details on the force layout, consult the documentation, which Mike spends a lot of time making things clear.Notice the values for gravity and charge both set to zero. This is so that nodes don't all gravitate towards a point on the screen and don't repel each other when near each other, respectively. Feel free to experiment with these values in index02.html to see what happens.

Okay, now add some actual circles to the SVG object. Placement and color is based on `nodes`.

```
// Draw circle for each node.
var circle = svg.selectAll("circle")
    .data(nodes)
  .enter().append("circle")
    .attr("id", function(d) { return d.id; })
    .attr("class", "node")
    .style("fill", function(d) { return foci[d.choice].color; });
```

Here's what you get without the transition.For a smooth load, instead of a random, bouncy one, use a transition for the circles as they come onto the screen. This is straight from Mike's example.

```
// For smoother initial transition to settling spots.
circle.transition()
    .duration(900)
    .delay(function(d,i) { return i * 5; })
    .attrTween("r", function(d) {
        var i = d3.interpolate(0, d.radius);
        return function(t) { return d.radius = i(t); };
    });
```

The next three functions, `tick()`, `gravity()`, and `collide()`, also come from the original example. The only differences are that color and position come from the `foci` array you made earlier. Also, in `collide()`, nodes are defined to belong to different clusters based on the `choice` attribute of the node objects instead of their color attribute.

In any case, here's `tick()`. It runs continuously on loop, changing the position and color of circles as necessary.

```
function tick(e) {
  circle
    .each(gravity(.051 * e.alpha))
    .each(collide(.5))
    .style("fill", function(d) { return foci[d.choice].color; })
    .attr("cx", function(d) { return d.x; })
    .attr("cy", function(d) { return d.y; });
}
```

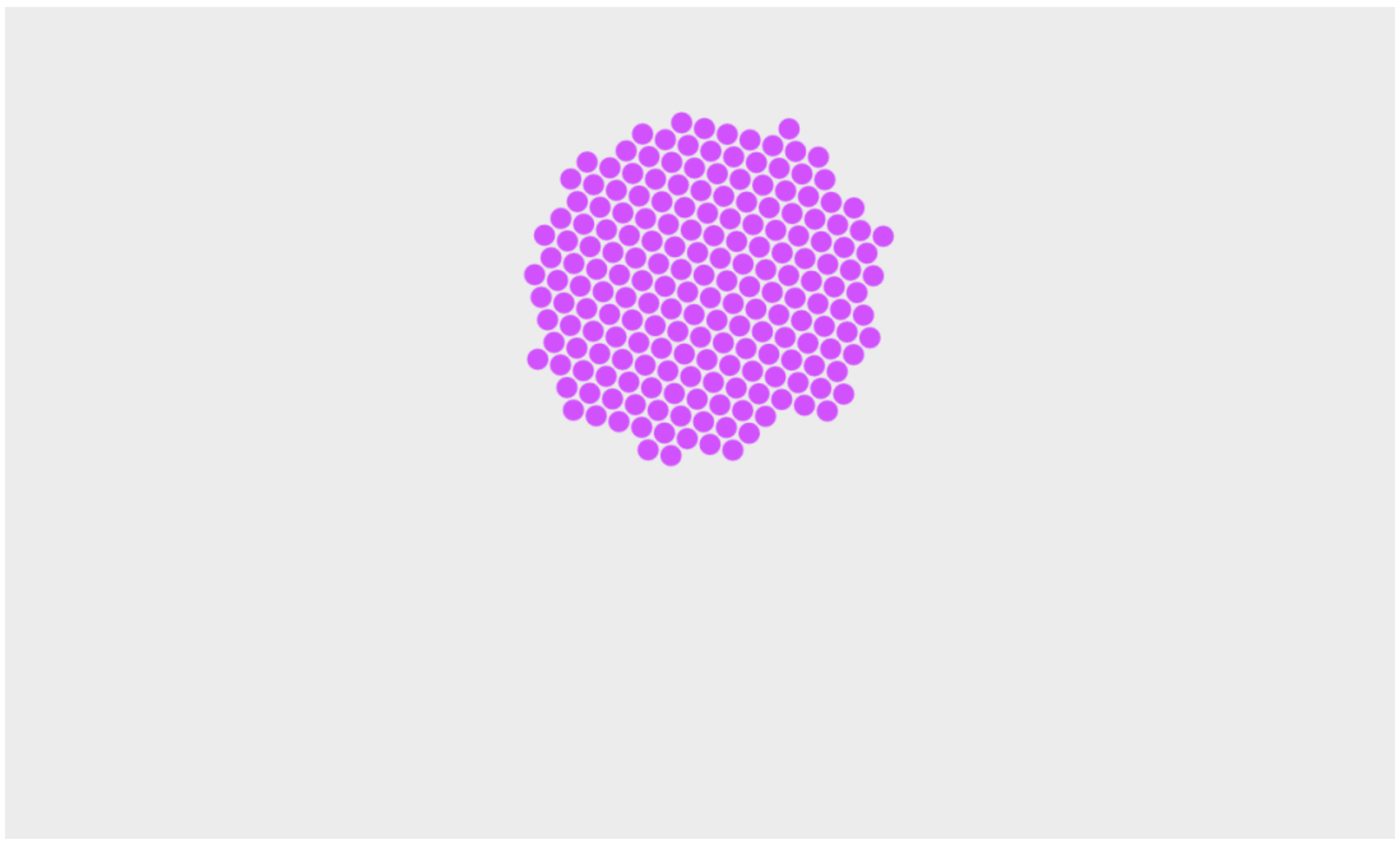
The `gravity()` function moves circles towards their defined foci.

```
// Move nodes toward cluster focus.
function gravity(alpha) {
  return function(d) {
    d.y += (foci[d.choice].y - d.y) * alpha;
    d.x += (foci[d.choice].x - d.x) * alpha;
  };
}
```

Then the `collide()` provides a force that ensures circles don't overlap. They bounce against each other instead.

```
// Resolve collisions between nodes.
function collide(alpha) {
  var quadtree = d3.geom.quadtree(nodes);
  return function(d) {
      var r = d.radius + node_radius + Math.max(padding, cluster_padding),
        nx1 = d.x - r,
        nx2 = d.x + r,
        ny1 = d.y - r,
        ny2 = d.y + r;
    quadtree.visit(function(quad, x1, y1, x2, y2) {
      if (quad.point && (quad.point !== d)) {
        var x = d.x - quad.point.x,
            y = d.y - quad.point.y,
            l = Math.sqrt(x * x + y * y),
            r = d.radius + quad.point.radius + (d.choice === quad.point.choice ?
padding : cluster_padding);
        if (l < r) {
          l = (l - r) / l * alpha;
          d.x -= x *= l;
          d.y -= y *= l;
          quad.point.x += x;
          quad.point.y += y;
        }
      }
      return x1 > nx2 || x2 < nx1 || y1 > ny2 || y2 < ny1;
    });
  };
}
```

This gets us here:



The circles converge to the top position and don't go anywhere after that. We handle the state changes in the next section.

Movement

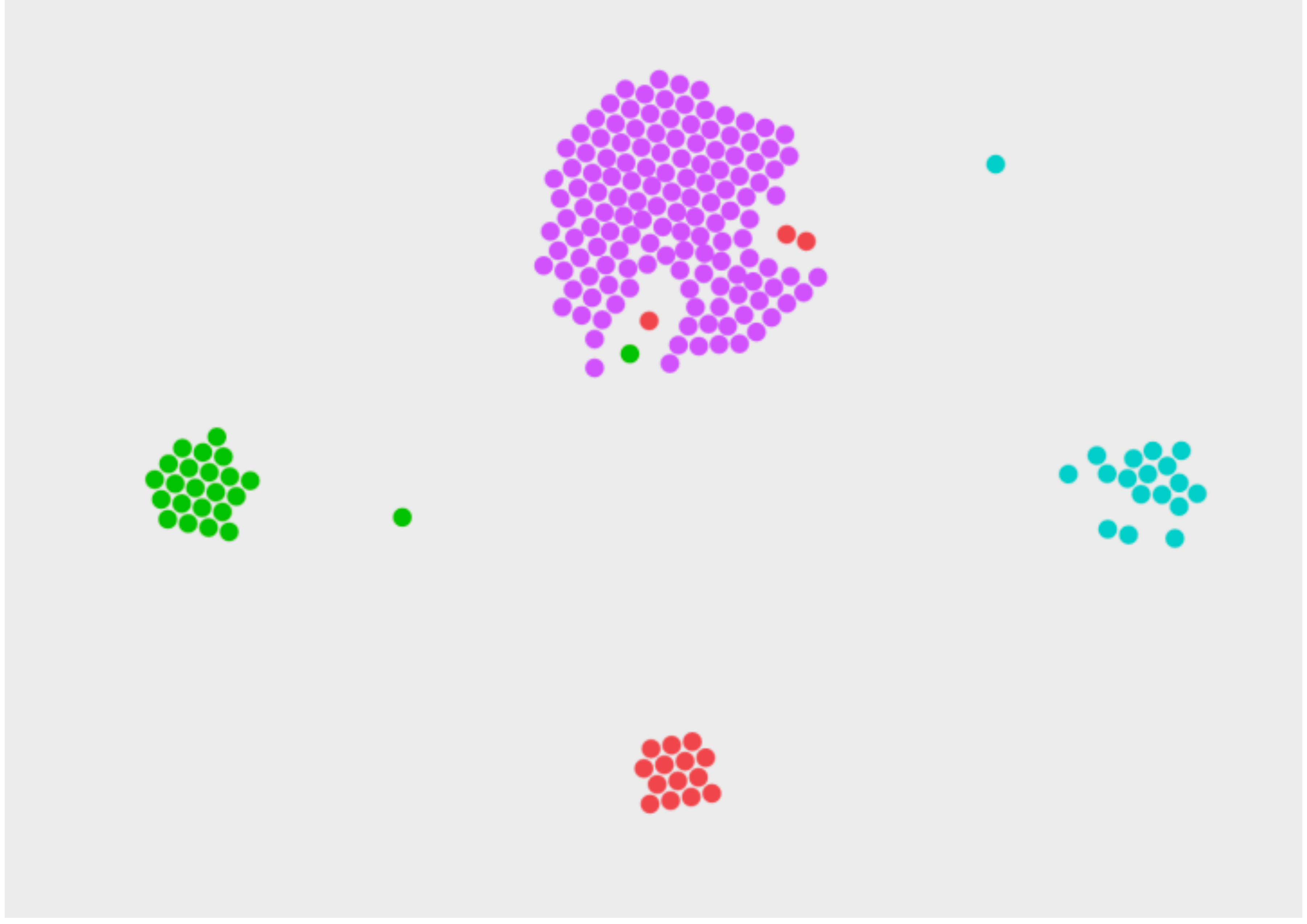
Open **index03.html**.Remember that the position of a circle is based on its `choice` attribute. The `gravity()` function, which is called repeatedly with `tick()`, moves circles towards the positions that correspond to a given choice. Change a node object's `choice` (which circles base their appearance on), and the force layout will move things accordingly.

In short, to make circles move on your screen, you change the `choice` attributes. Write a function that does this, and then use `setTimeout()` to call the function at set intervals.

In the function below, `timer()`, a choice is randomly selected and then you change a random node to that choice. You `resume()` the force to set the circles moving again (they eventually settle in a spot otherwise). At the end of the function, `setTimeout()` is called again to schedule the next call.

```
// Run function periodically to make things move.
var timeout;
function timer() {

    // Random place for a node to go
    var choices = d3.keys(foci);
    var foci_index = Math.floor( Math.random() * choices.length );
    var choice = d3.keys(foci)[foci_index];

    // Update random node
    var random_index = Math.floor( Math.random() * nodes.length );
    nodes[random_index].cx = foci[choice].x;
    nodes[random_index].cy = foci[choice].y;
    nodes[random_index].choice = choice;

    force.resume();

    // Run it again in a few seconds.
    timeout = setTimeout(timer, 400);
}

timeout = setTimeout(timer, 400);
```

See **index.html** for the finished version.Here is what you get:



Wrapping Up

Once you figure out the minutiae of moving around a bunch of small circles, you'll see that the method can be fairly versatile. They're like little building blocks that you put together to show groups and clustering. The key, as with most d3.js projects, is to breakdown the visualization to its elements and work bit-by-bit.

About the Author

Nathan Yau is a statistician who works primarily with visualization. He earned his PhD in statistics from UCLA, is the author of two best-selling books — *Data Points* and *Visualize This* — and runs FlowingData. Introvert. Likes food. Likes beer.

---

19 Comments (Add Yours)

- *Hendrik Henze* — November 17, 2016 at 5:15 am

  I love this! How can I change the random part into something more data based?

  Reply
  - *Nathan Yau* — November 29, 2016 at 11:11 am

    In the `timer()` function. In this example, movements are decided by `Math.random()`.

    Reply
- *Tim Ryan* — January 20, 2017 at 9:49 am

  Thanks, this is great. With your "simulated days of 1,000 people" chart, how do you insert the "days-simulated-v2" data file in the code itself? I've tried
  var data =
  ["0,270,5,32,10,73,16,25,5,165,2,35,4,300,1,53,10,117,0,370",
  …];
  But it doesn't seem to work. Thanks again!

  Reply
  - *Abhishek Nambiar* — June 26, 2018 at 2:27 am

    Did you find a solution to this?

    Reply
  - *Nathan Yau* — June 26, 2018 at 2:31 pm

    You'd have to create an object that matches the structure of the loaded CSV file. Make use of `console.log()` to see what the data looks like.

    Reply
- *Kirshnee Moodley* — January 3, 2018 at 4:04 am

  Hi Nathan, Thanks for the great tutorial! How do I direct the nodes from the top one to the other two nodes in specific proportions? I am trying to create something similar to the election probabilities example you showed. Thanks!

  Reply
  - *Nathan Yau* — January 3, 2018 at 1:16 pm

    That would happen at this spot:

    ```
    // Random place for a node to go
    ```

```
var choices = d3.keys(foci);
var foci_index = Math.floor( Math.random() * choices.length );
var choice = d3.keys(foci)[foci_index];
```

Right now the `choice` is random. So at some point (maybe after a mouseclick event), you want to make a certain proportion of the nodes with the corresponding choice.

Reply

- *Kirshnee Moodley* — January 4, 2018 at 5:35 am

  Thanks Nathan! Is it possible to show a sample of the data, as I am struggling to set up my data frame (using R) to the right format needed for this visualization.

- *Nathan Yau* — January 10, 2018 at 10:18 am

  Not sure how useful this is, but here's the data I used for the time use simulation:

  https://flowingdata.com/projects/2015/timeuse-simulation/data/days-simulated-v2.tsv

  Each row represents one person. The encoding is activity code first, and then how many minutes. Then next activity code, and then how many minutes. This is specific to that visualization though and will likely be handled in different ways for different datasets.

  So I think the main thing you should be after is encoding data for each observation (person, thing, circle, etc).

- *Josh Goldberg* — February 16, 2018 at 9:49 am

  Hi Nathan,

  How easy would it be to modify the time scale to days, months, years? Same data format as yours (activity code, duration, activity code, duration…), but the duration unit of measure would be one of the previous mentioned. I imagine you would modify this code to get it to work? How can we use months as an example?

  // Minutes to time of day. Data is minutes from 4am.
  function minutesToTime(m) {
  var minutes = (m + 4*60) % 1440;
  var hh = Math.floor(minutes / 60);
  var ampm;
  if (hh > 12) {
  hh = hh – 12;
  ampm = "pm";
  } else if (hh == 12) {
  ampm = "pm";
  } else if (hh == 0) {
  hh = 12;
  ampm = "am";
  } else {
  ampm = "am";
  }
  var mm = minutes % 60;
  if (mm < 10) {
  mm = "0" + mm;
  }

  return hh + ":" + mm + ampm
  }

  Reply

  - *Nathan Yau* — February 18, 2018 at 9:13 am

    Yeah, this could be adapted to show other time scales. The `minutesToTime()` function is a helper function to display time, so that would only help in converting an integer to readable time.

    The area you'd be most concentrated on is the `timer()` function. That function is called each iteration (i.e. time segment).

    Reply

- *Priya Ramakrishnan* — April 28, 2018 at 3:28 am

  Hi Nathan, a tutorial on how to do this in v4 will be very helpful as the syntax for force layout seems to have changed substantially in v4 compared to v3?

  Reply

- *Jamie Wood* — June 17, 2018 at 6:23 am

  Great work! I've spent months learning CSS, HTML and d3 and finally have something similar to the American visual.

  I have one problem and i can't figure it out. My data is the number of calls per minute into an imaginary call centre. I have the same amount of "circles" yet a lot cluster to one area and the circles struggle to escape the other circles around it in time to move to its next location. Any ideas to ensure the dots set off around the minute they are supposed to and not just get stuck by the others?

  Thanks, and again, brilliant!

  Reply

  - *Nathan Yau* — June 19, 2018 at 11:01 pm

Try experimenting with the force parameters (the snippet that starts with `d3.layout.force()`), namely friction and gravity. Those control nodes being held back and moving towards places.

Reply

- *Matthias Schlesinger* — June 22, 2018 at 12:32 am

  Hi,
  great tutorial. I love the Day in the life of Americans visualisation and was wondering if you could share the underlying data structure that you pass to your chart. Do you have 1000 nodes with an id, where for each time stamp they have a group assigned to them? And then you iterate those time stamps and update the position of the node in the timer() function based on the position of the group?

  Thanks for your help.

  Reply

  - *Matthias Schlesinger* — June 22, 2018 at 12:34 am

    Ah I just saw your response to a similar question. You encode it with group and period of stay. That's significantly less data. Nice solution

    Reply

- *Abhishek Nambiar* — July 2, 2018 at 3:35 am

  Hi, amazing tutorial.
  I have a few queries though.
  Can we have the nodes move slowly throughout the 'hour', but major data shifts only happen only on an hourly basis (e.g at 3,4,5 and so on)
  Conversely, can we have the data move on a percentage basis only? (out of a total base of 100, 20 moves from centre to A, 15 to B, 17 to C and so on, changing with every iteration)?

  Reply

  - *Nathan Yau* — July 3, 2018 at 10:33 am

    Yeah, I would mess around with the logic in `timer()`. That's where you decide what switches states and when. You might also want to look at `friction()` in the force to change how quickly circles move from one spot to another.

    Reply

    - *Abhishek Nambiar* — July 4, 2018 at 11:29 pm

      Whenever I tweak around with timer(), something horribly goes wrong.
      Can you show a few examples so I can catch a better hang of it?

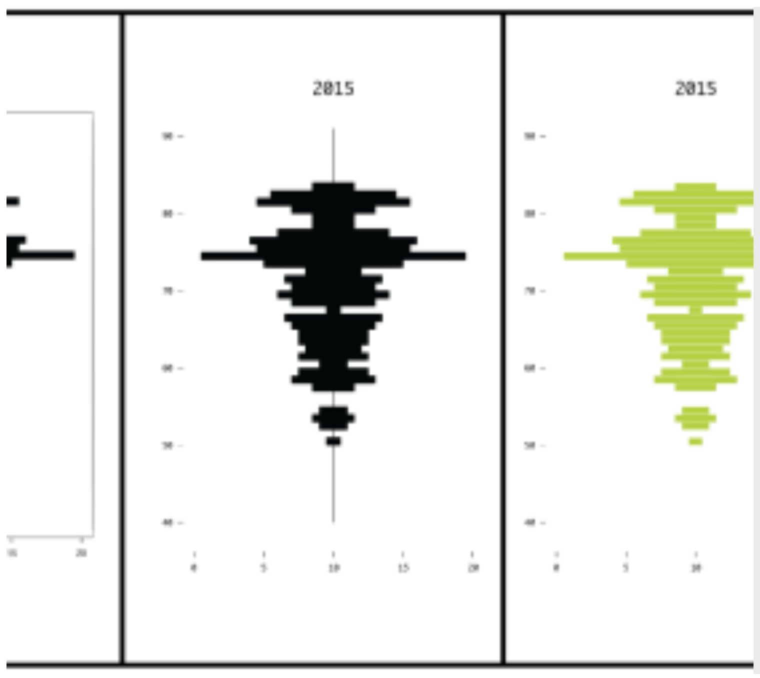
Add Comment

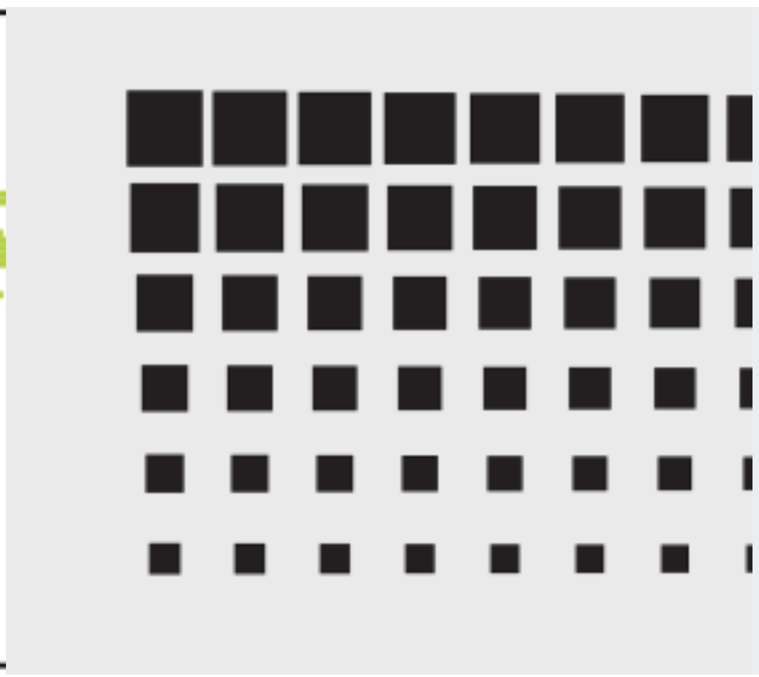
Logged in as Hugo Kevenaar. Logout »

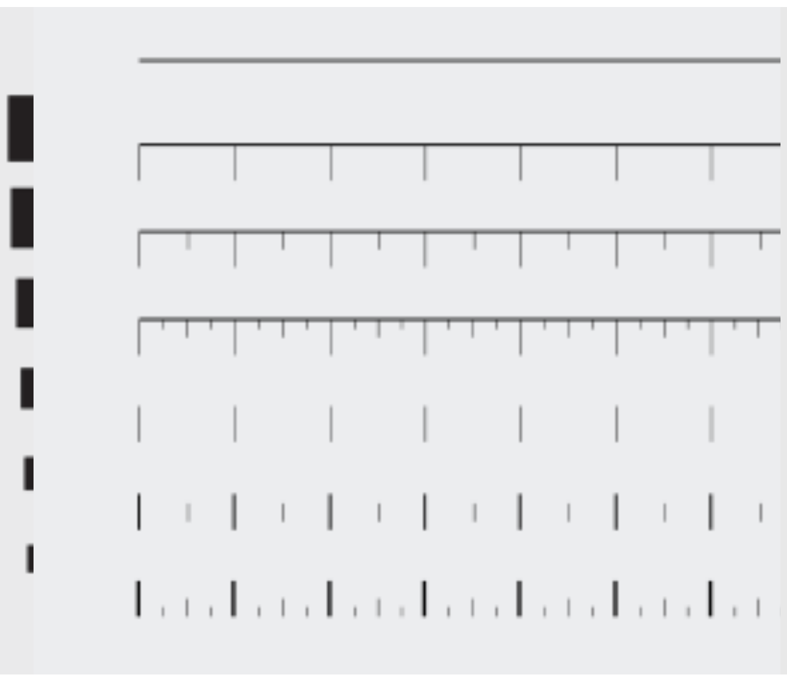
Comment

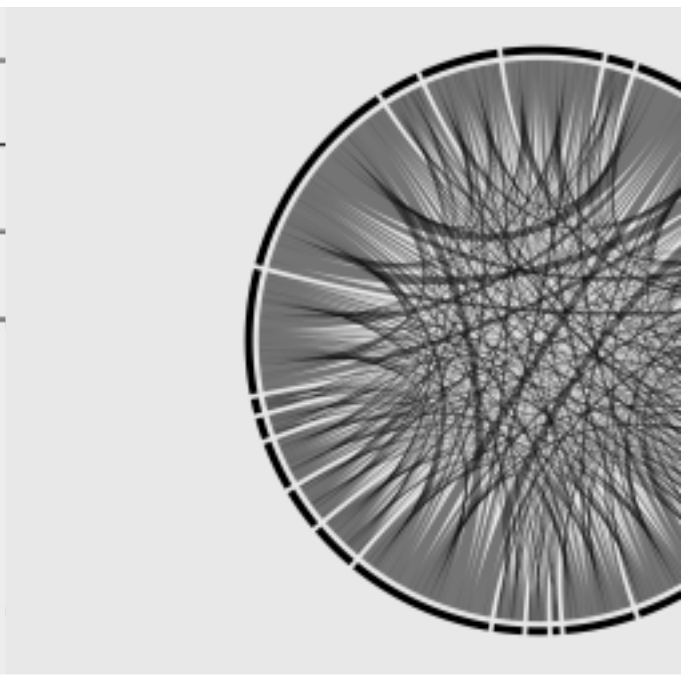
Submit

More Tutorials See All →









How to Edit R Charts in Adobe Illustrator

A detailed guide for R users who want to polish their charts in the popular graphic design app for readability and aesthetics.

Drawing Squares and Rectangles in R

R makes it easy to add squares and rectangles to your plots, but it gets a little tricky when you have a bunch to draw at once. The key is to break it down to the elements.

How to Customize Axes in R

For presentation purposes, it can be useful to adjust the style of your axes and reference lines for readability. It's all about the details.

How to Make Chord Diagrams in R

Show connections in the circular layout for a more compact presentation.

Copyright © 2007-Present FlowingData. All rights reserved.