Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 9

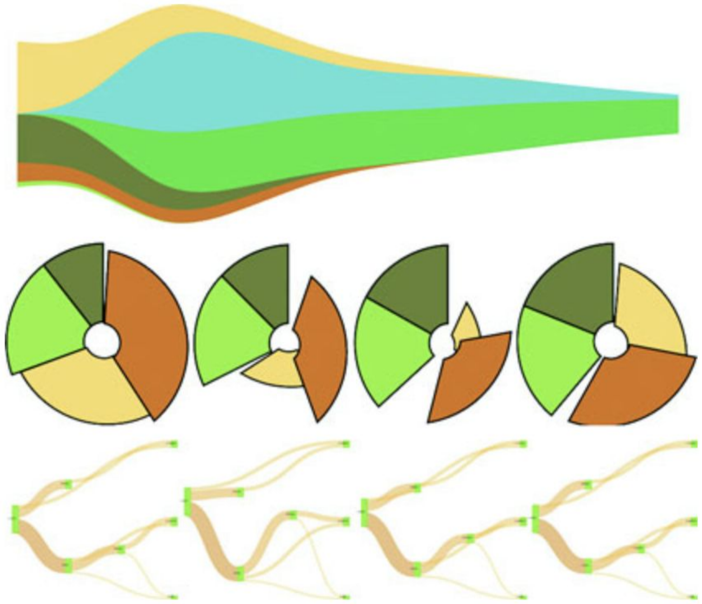
Layouts

Overview

- Understanding histogram and pie chart layouts
- Learning about simple tweening
- Working with stack layouts
- Using Sankey diagrams and word clouds

Overview

Multiple layouts are demonstrated in this chapter, including the circle pack, tree, stack, and Sankey, as well as tweening to properly animate shapes like the arcs in pie charts.



Histograms

Histogram code

```
d3.json("tweets.json", function(error, data) { histogram(data.tweets)
})
function histogram(tweetsData) {
   var xScale = d3.scaleLinear().domain([ 0, 5 ]).range([ 0, 500 ]);
   var yScale = d3.scaleLinear().domain([ 0, 10 ]).range([ 400, 0 ]);
   var xAxis = d3.axisBottom().scale(xScale).ticks(5)
   var histoChart = d3.histogram();                              1

   histoChart
    .domain([ 0, 5 ])
    .thresholds([ 0, 1, 2, 3, 4, 5 ])
    .value(d => d.favorites.length)                             2

histoData = histoChart(tweetsData);                               3
```

1 Creates a new layout function
2 Determines the values the histogram bins for
3 Formats the data

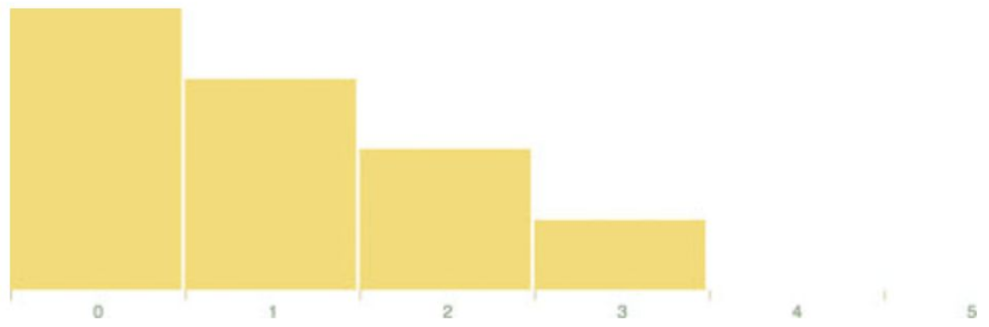
Histograms

```
d3.select("svg")
    .selectAll("rect")
    .data(histoData).enter()
    .append("rect")
    .attr("x", d => xScale(d.x0))
    .attr("y", d => yScale(d.length))
    .attr("width", d => xScale(d.x1 - d.x0) - 2)
    .attr("height", d => 400 - yScale(d.length))            4
    .style("fill", "#FCD88B")

  d3.select("svg").append("g").attr("class", "x axis")
    .attr("transform", "translate(0,400)").call(xAxis);
  d3.select("g.axis").selectAll("text").attr("dx", 50);      5
}
```

4 Formatted data is used to draw the bars
5 Centers the axis labels under the bars

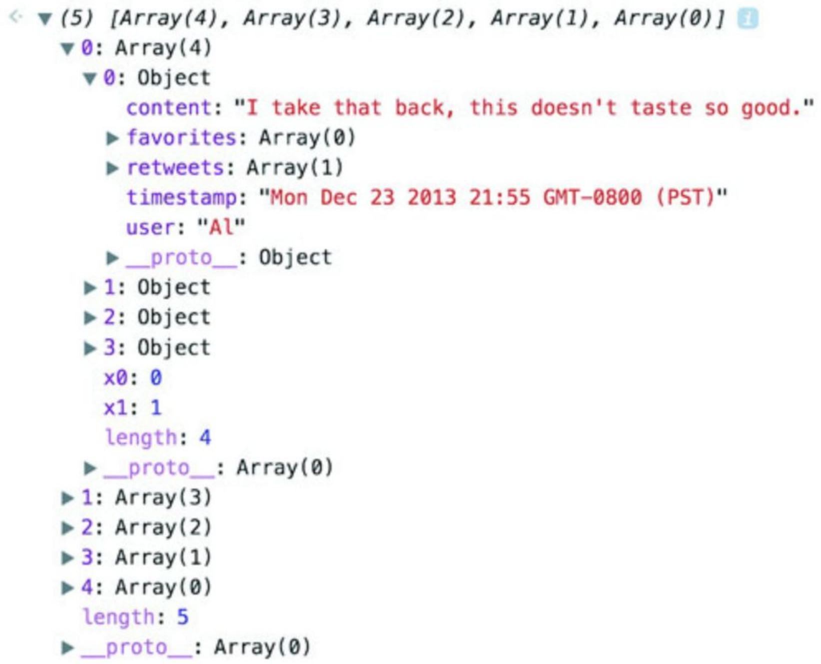
Histograms

The histogram in its initial state before we change the measure from favorites to retweets by clicking on one of the bars



Histograms

The processed data from d3.histogram returns an array where each array item also has an x0 and x1 field.



```
▼ (5) [Array(4), Array(3), Array(2), Array(1), Array(0)]
  ▼ 0: Array(4)
    ▼ 0: Object
        content: "I take that back, this doesn't taste so good."
      ▶ favorites: Array(0)
      ▶ retweets: Array(1)
        timestamp: "Mon Dec 23 2013 21:55 GMT-0800 (PST)"
        user: "Al"
      ▶ __proto__: Object
    ▶ 1: Object
    ▶ 2: Object
    ▶ 3: Object
      x0: 0
      x1: 1
      length: 4
    ▶ __proto__: Array(0)
  ▶ 1: Array(3)
  ▶ 2: Array(2)
  ▶ 3: Array(1)
  ▶ 4: Array(0)
    length: 5
  ▶ __proto__: Array(0)
```

Histograms

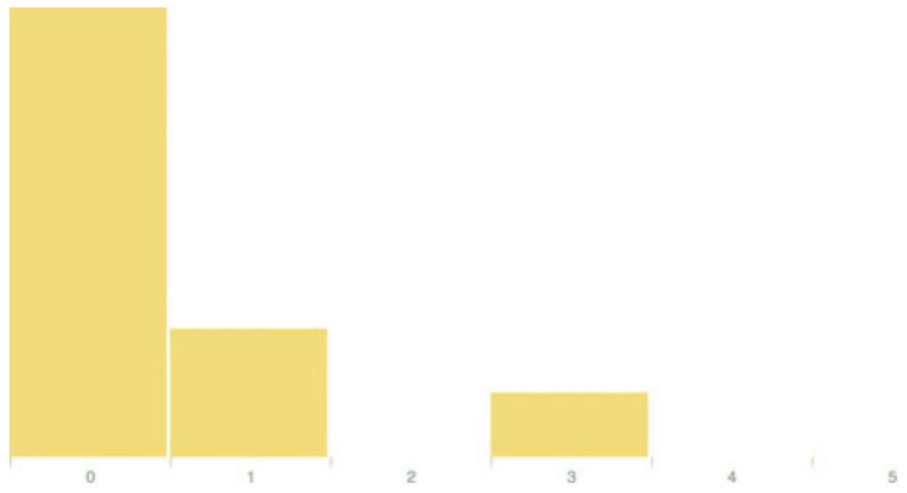
Histogram interactivity

```
...
          .attr("height", d => 400 - yScale(d.length))
          .on("click", retweets);

  function retweets() {
    histoChart.value(d => d.retweets.length)                    1
    histoData = histoChart(tweetsData);
    d3.selectAll("rect").data(histoData)                    2
    .transition().duration(500).attr("x", d => xScale(d.x0))
    .attr("y", d => yScale(d.length))
      .attr("height", d => 400 - yScale(d.length))
 };
```

1 Changes the value being measured
2 Binds and redraws the new data

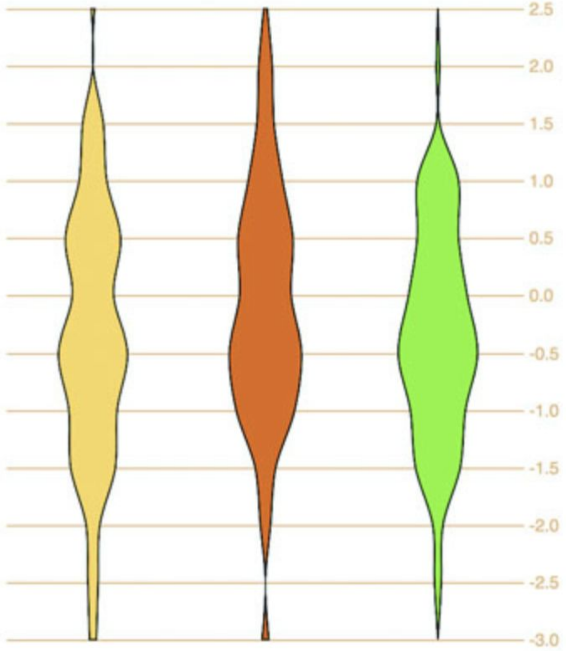
Histograms

The histogram chart we've built will make an animated transition to display tweets binned by the number of retweets instead of the number of favorites.



Histograms

Three violin plots based on the data produced by d3.histogram



Histograms

Generating violin plots with d3.histogram

```
var fillScale = d3.scaleOrdinal().range(["#fcd88a", "#cf7c1c", "#93c464"])

  var normal = d3.randomNormal()
  var sampleData1 = d3.range(100).map(d => normal())
  var sampleData2 = d3.range(100).map(d => normal())
  var sampleData3 = d3.range(100).map(d => normal())          1
  var histoChart = d3.histogram();

  histoChart
   .domain([ -3, 3 ])
   .thresholds([ -3, -2.5, -2, -1.5, -1,
          -0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3 ])                  2
   .value(d => d)

   var yScale = d3.scaleLinear().domain([ -3, 3 ]).range([ 400, 0 ]);
   var yAxis = d3.axisRight().scale(yScale)
    .tickSize(300)
   d3.select("svg").append("g").call(yAxis)
```

1 Generate three sample distributions
2 The more thresholds, the smoother any distribution chart will look

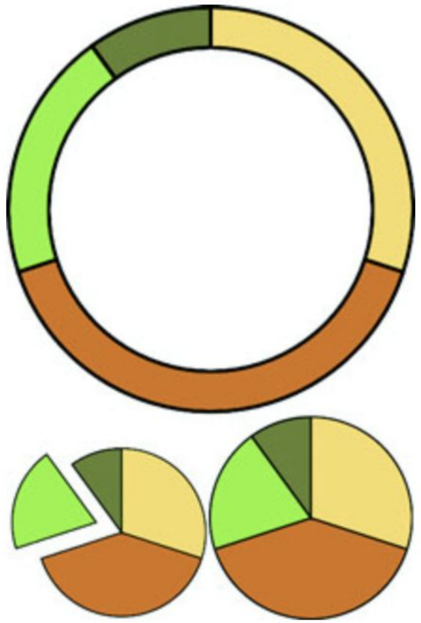
Histograms

```
 var area = d3.area()
    .x0(d => -d.length)
    .x1(d => d.length)                                   3
    .y(d => yScale(d.x0))
    .curve(d3.curveCatmullRom)                               4

d3.select("svg")
  .selectAll("g.violin")
  .data([sampleData1, sampleData2, sampleData3])
  .enter()
  .append("g")
  .attr("class", "violin")
  .attr("transform", (d,i) => `translate(${50 + i * 100},0)`)
  .append("path")
  .style("stroke", "black")
  .style("fill", (d,i) => fillScale(i))
  .attr("d", d => area(histoChart(d)))                    5
```

3 Unlike in the last chapter, we’ll draw these vertically
4 Use a Catmull–Rom spline interpolation for the area generator
5 We’re going to generate the area based on the data transformed by the histogram function

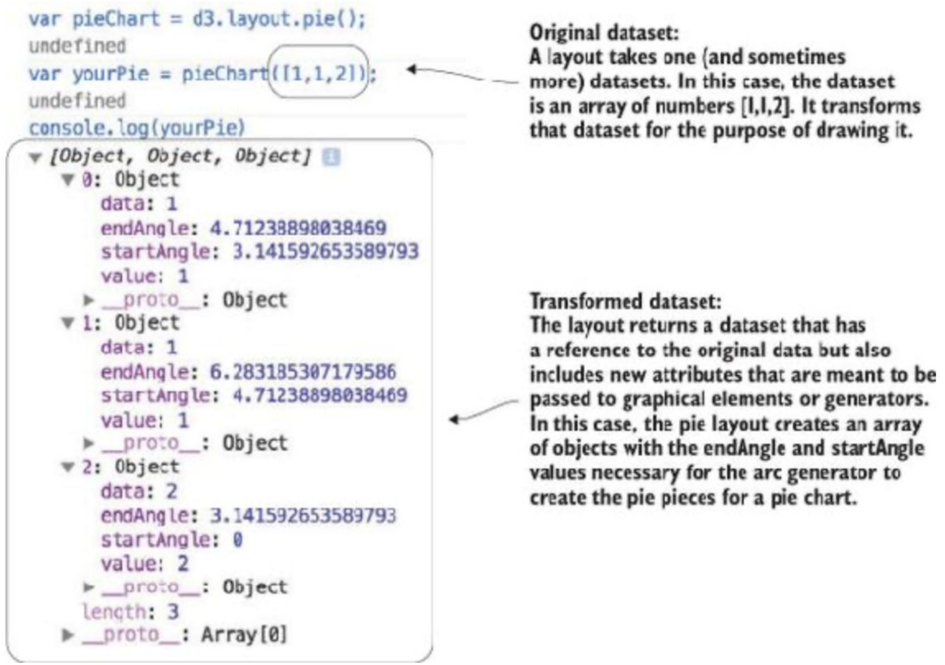
Pie charts

The traditional pie chart (bottom right) represents proportion as an angled slice of a circle. With slight modification, it can be turned into a donut or ring chart (top) or an exploded pie chart (bottom left).



Pie charts

A pie layout applied to an array of [1,1,2] shows objects created with a start angle, end angle, and value attribute corresponding to the dataset, as well as the original data, which in this case is a number.



```
var pieChart = d3.layout.pie();
undefined
var yourPie = pieChart([1,1,2]);
undefined
console.log(yourPie)
▼ [Object, Object, Object]
  ▼ 0: Object
      data: 1
      endAngle: 4.71238898038469
      startAngle: 3.141592653589793
      value: 1
    ▶ __proto__: Object
  ▼ 1: Object
      data: 1
      endAngle: 6.283185307179586
      startAngle: 4.71238898038469
      value: 1
    ▶ __proto__: Object
  ▼ 2: Object
      data: 2
      endAngle: 3.141592653589793
      startAngle: 0
      value: 2
    ▶ __proto__: Object
    length: 3
  ▶ __proto__: Array[0]
```

Original dataset:
A layout takes one (and sometimes more) datasets. In this case, the dataset is an array of numbers [1,1,2]. It transforms that dataset for the purpose of drawing it.

Transformed dataset:
The layout returns a dataset that has a reference to the original data but also includes new attributes that are meant to be passed to graphical elements or generators. In this case, the pie layout creates an array of objects with the endAngle and startAngle values necessary for the arc generator to create the pie pieces for a pie chart.

Pie charts

```
var newArc = d3.arc();
newArc.innerRadius(0)
   .outerRadius(100)                    1
console.log(newArc(yourPie[0]));         2
```

1 Gives our arcs and resulting pie chart a radius of 100 px
2 Returns the d attribute necessary to draw this arc as a <path> element: “M6.123031769111886e-15,100A100,100 0 0,1 -100,1.2246063538223773e-14L0,0Z”

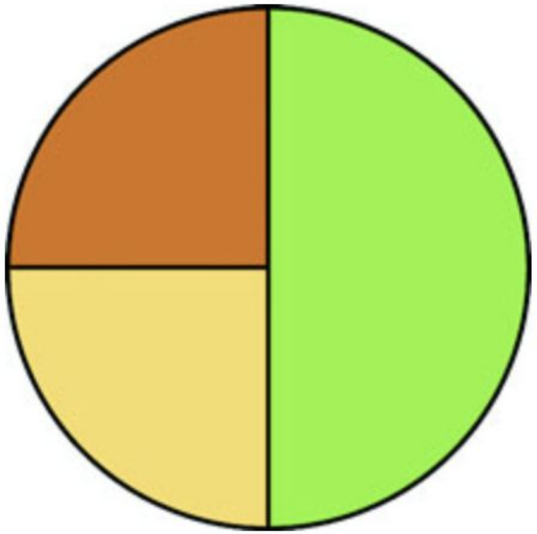
Pie charts

```
var fillScale = d3.scaleOrdinal()
   .range(["#fcd88a", "#cf7c1c", "#93c464",
"#75734F"])
d3.select("svg")
  .append("g")                                    1
  .attr("transform","translate(250,250)")
  .selectAll("path")
  .data(yourPie)                                  2
  .enter()
  .append("path")
  .attr("d", newArc)                              3
  .style("fill", (d,i) => fillScale(i))
  .style("stroke", "black")
  .style("stroke-width", "2px");
```

1 Appends a new <g> and moves it to the middle of the canvas so that it'll be easier to see the results
2 Binds the array that was created using the pie layout, not our original array or the pie layout itself
3 Each path drawn based on that array needs to pass through the newArc function, which sees the startAngle and endAngle attributes of the objects and produces the commensurate SVG drawing code

Pie charts

A pie chart showing three pie pieces that subdivide the circle between the values in the array [1,1,2]



Pie charts

```
d3.json("tweets.json", pieChart)
function pieChart(data) {
   var nestedTweets = d3.nest()
      .key(d => d.user)
      .entries(data.tweets);
   nestedTweets.forEach(d => {
      d.numTweets = d.values.length;
      d.numFavorites = d3.sum(d.values, p => p.favorites.length)        1
      d.numRetweets = d3.sum(d.values, p => p.retweets.length)          2
   });
}
```

1 Gives the total number of favorites by summing the favorites array length of all the tweets
2 Gives the total number of retweets by doing the same for the retweets array length

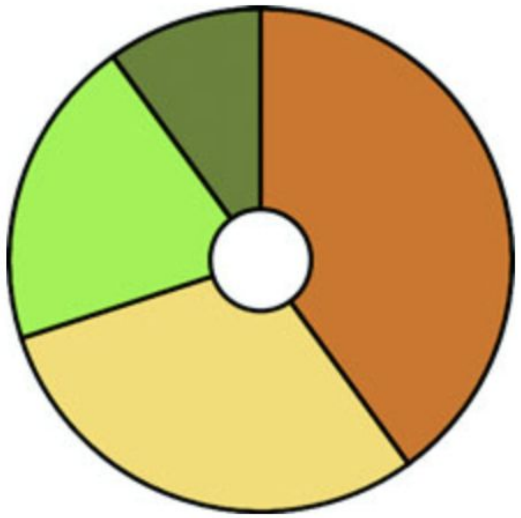
Pie charts

Creating a ring chart

```
pieChart.value(d => d.numTweets);
newArc.innerRadius(20)
var yourPie = pieChart(nestedTweets);
```

Pie charts

A donut chart showing the number of tweets from our four users represented in the nestedTweets dataset.

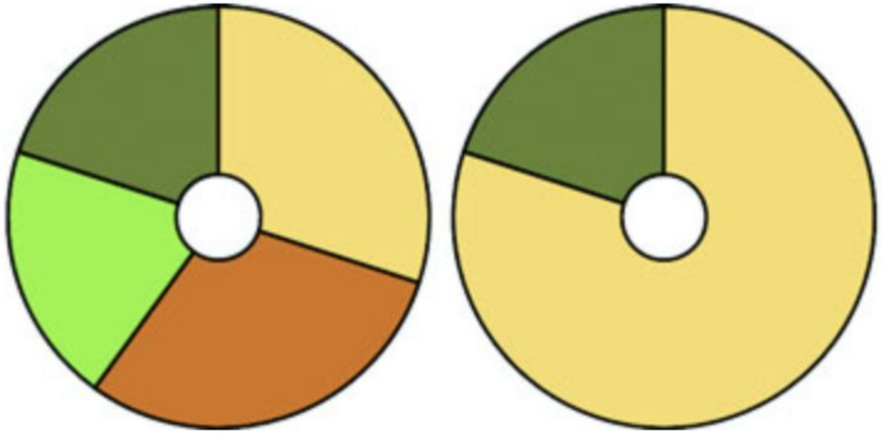


Pie charts

```
pieChart.value(d => d.numFavorites)
d3.selectAll("path").data(pieChart(nestedTweets))
  .transition().duration(1000).attr("d", newArc);
pieChart.value(d => d.numRetweets);
d3.selectAll("path").data(pieChart(nestedTweets))
  .transition().duration(1000).attr("d", newArc);
```

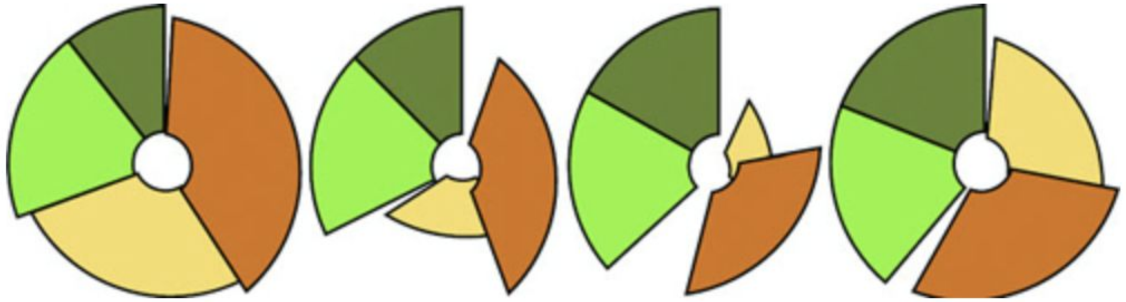
Pie charts

The pie charts representing, on the left, the total number of favorites and, on the right, the total number of retweets



Pie charts

Snapshots of the transition of the pie chart representing the number of tweets to the number of favorites. This transition highlights the need to assign key values for data binding and to use tweens for some types of graphical transition, such as that used for arcs.



Pie charts

- Data-binding uses array positions
- Pie layout also sorts the data
- When we recall data, it resorts
- Which changes array positions, creates wonky transition
- Partial solution:

```
pieChart.sort(null);
```

- But still the circles deform a bit

Pie charts

Updated binding and transitioning for pie layout

```
pieChart
    .value(d => d.numTweets)
    .sort(null)                                    1

  var tweetsPie = pieChart(nestedTweets)

  pieChart.value(d => d.numRetweets)
  var retweetsPie = pieChart(nestedTweets)

  nestedTweets.forEach((d,i) => {
    d.tweetsSlice = tweetsPie[i]
    d.retweetsSlice = retweetsPie[i]
})                                              2
```

1 Don’t sort the pie results so that they stay in the same order as the array you send
2 Take the original dataset and add to each object the results of the pie layout

Pie charts

```
...
.selectAll("path")
.data(nestedTweets, d => d.key)                    3
.enter()
.append("path")
.attr("d", d => newArc(d.tweetsSlice))
.style("fill", (d,i) => fillScale(i))
...

d3.selectAll("path")
  .transition()
  .duration(1000)
  .attrTween("d", arcTween)                   4
```

3 Notice we're appending the original dataset because it has the drawing instructions now
4 I'm only using a named function here instead of an arrow function because it's longer and so it's easier to read as a separate function

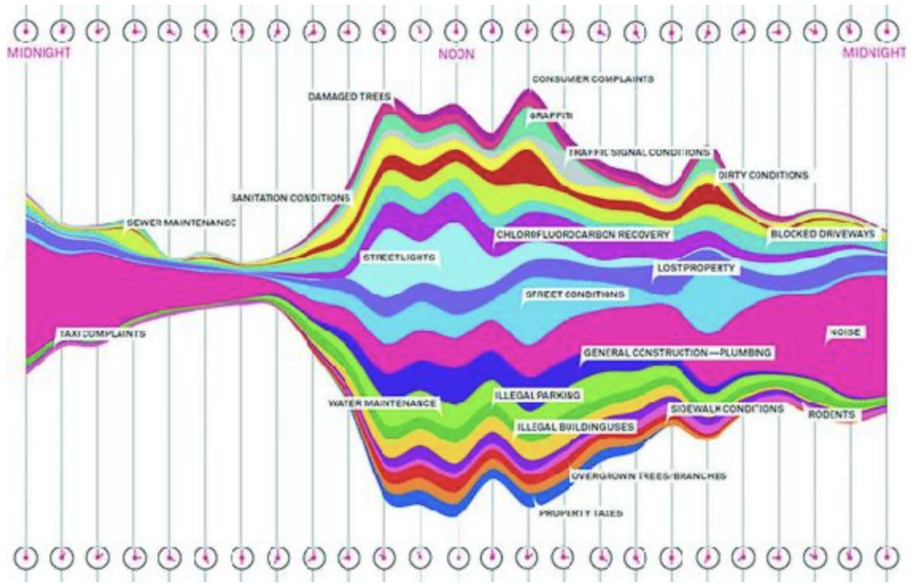
Pie charts

```
function arcTween(d) {
  return t => {                                        5
    var interpolateStartAngle = d3
    .interpolate(d.tweetsSlice.startAngle,
d.retweetsSlice.startAngle);
    var interpolateEndAngle = d3
    .interpolate(d.tweetsSlice.endAngle,
d.retweetsSlice.endAngle);
      d.startAngle = interpolateStartAngle(t);
      d.endAngle = interpolateEndAngle(t);
       return newArc(d);                                   6
  }
}
```

5 attrTween expects a function that takes the current transition value (a float between 0 and 1) and returns the interpolated value, in this case an arc drawn from the interpolated start and interpolated end angles
6 Because this is going into a d attribute, make sure to return the drawing instructions for the intermediary arc

Stack layout

The streamgraph by Pitch Interactive used in a Wired piece describing the subject of calls to 311 (a city service for reporting problems) in New York (November 1, 2010; https://www.wired.com/2010/11/ff_311_new_york/all/1)



Stack layout

Stack layout example

```
d3.csv("movies.csv", dataViz);
function dataViz(data) {

  var xScale = d3.scaleLinear().domain([0, 10]).range([0, 500]);
  var yScale = d3.scaleLinear().domain([0, 100]).range([500, 0]);
  var movies = ["titanic", "avatar", "akira", "frozen", "deliverance",
"avengers"]

  var fillScale = d3.scaleOrdinal()
    .domain(movies)
    .range(["#fcd88a", "#cf7c1c", "#93c464", "#75734F", "#5eafc6",
"#41a368"])

  stackLayout = d3.stack()
     .keys(movies)                              1
```

1 The movies dataset happens to be perfectly suited to the default stack formatting—all you need to do is pass an array of keys for each object, which happens to also be the domain of our colorScale

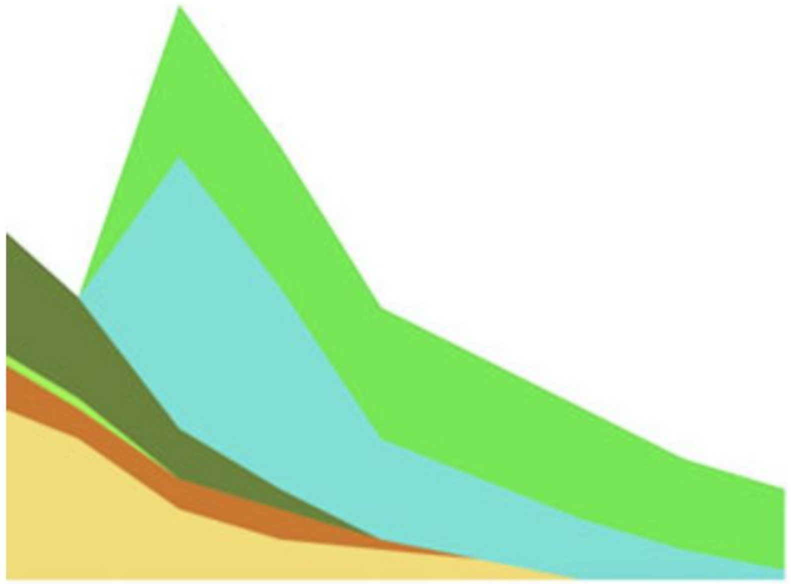
Stack layout

```
  var stackArea = d3.area()
    .x((d, i) => xScale(i))
    .y0(d => yScale(d[0]))
    .y1(d => yScale(d[1]));                    2

  d3.select("svg").selectAll("path")
    .data(stackLayout(data))
    .enter().append("path")
    .style("fill", d => fillScale(d.key))        3
    .attr("d", d => stackArea(d));
}
```

2 The stack layout is going to return an array of two item arrays, the first is the lower bound and the second is the upper bound, and the index position can be used for the x-position
3 Each array of stacked data has a key property that corresponds to the keys you sent in your layout generator

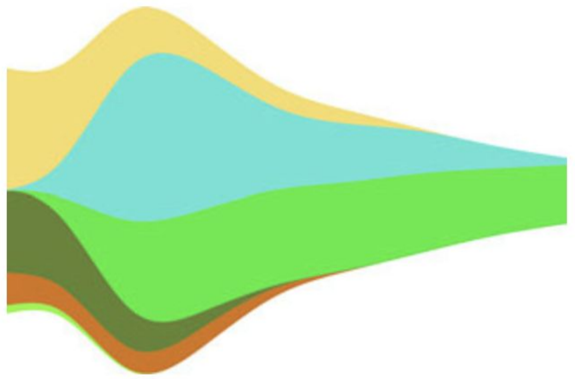
Stack layout

The stack layout default settings, when tied to an area generator, produce a stacked area chart like this one.



Stack layout

```
stackLayout.offset(d3.stackOffsetSilhouet
te).order(d3.stackOrderInsideOut)
stackArea.curve(d3.curveBasis)
yScale.domain([-50, 50])
```



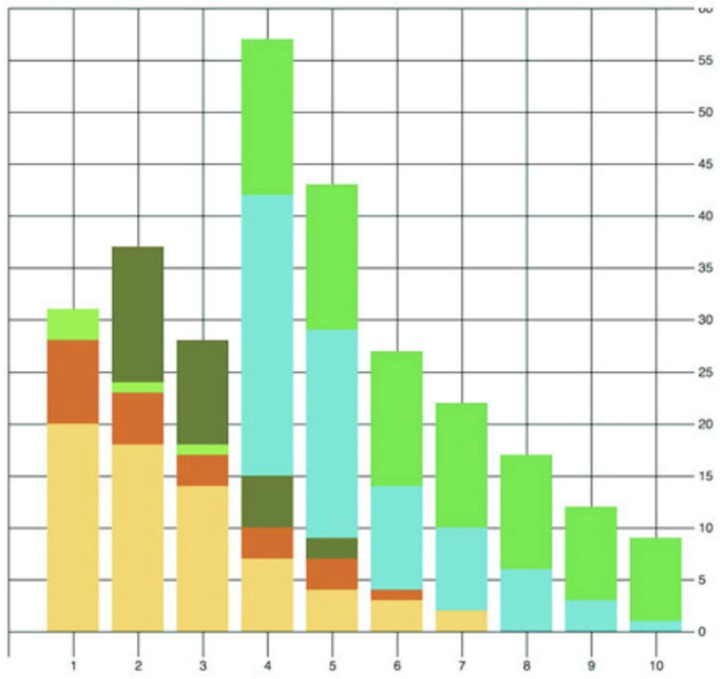
Stack layout

```
var xScale = d3.scaleLinear().domain([0, 10]).range([0, 500])
var yScale = d3.scaleLinear().domain([0, 60]).range([480, 0])
var heightScale = d3.scaleLinear().domain([0, 60]).range([0, 480])

stackLayout = d3.stack().keys(movies)

d3.select("svg").selectAll("g.bar")
  .data(stackLayout(data))
  .enter()
  .append("g")
  .attr("class", "bar")
  .each(function(d) {                                    1
     d3.select(this).selectAll("rect")
       .data(d)
       .enter()
       .append("rect")
       .attr("x", (p,q) => xScale(q) + 30)               2
       .attr("y", p => yScale(p[1]))                     2
       .attr("height", p => heightScale(p[1] - p[0]))
       .attr("width", 40)                                3
       .style("fill", fillScale(d.key));
    });
```

1 The stacked data is returned in a way so that we iterate through drawing each movie’s bars, rather than each day
2 This function is using p,q instead of d,i as a conventional approach for nested arrow functions
3 Because it’s an SVG:rect, we want it to be placed where its top position would be, and then we draw down from there

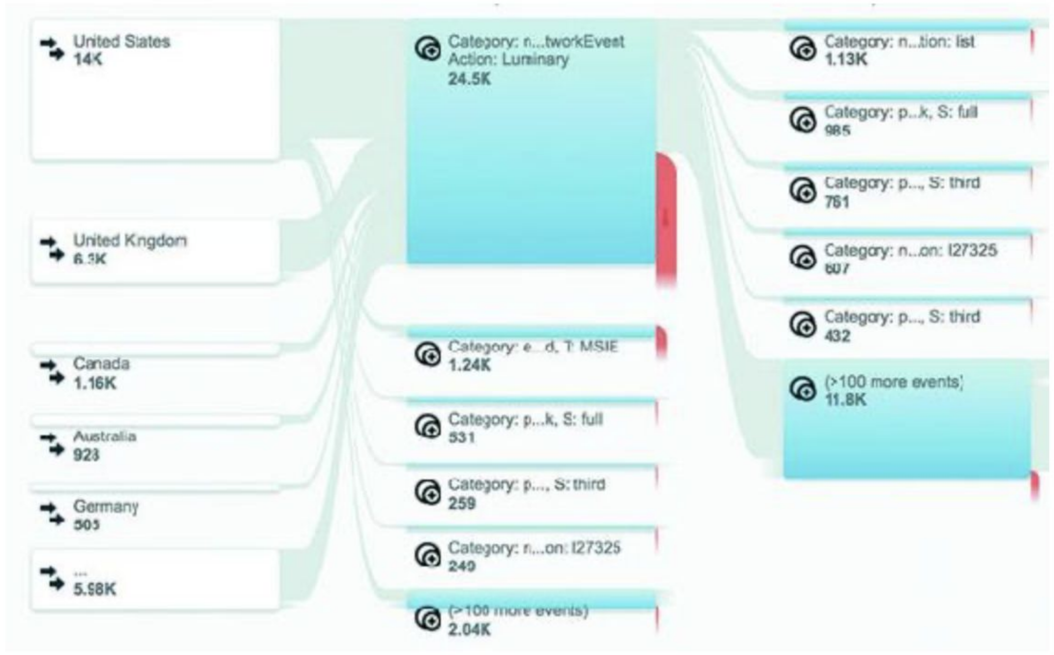
Stack layout

A stacked bar chart using the stack layout to determine the position of the rectangles that make up each day’s stacked bar



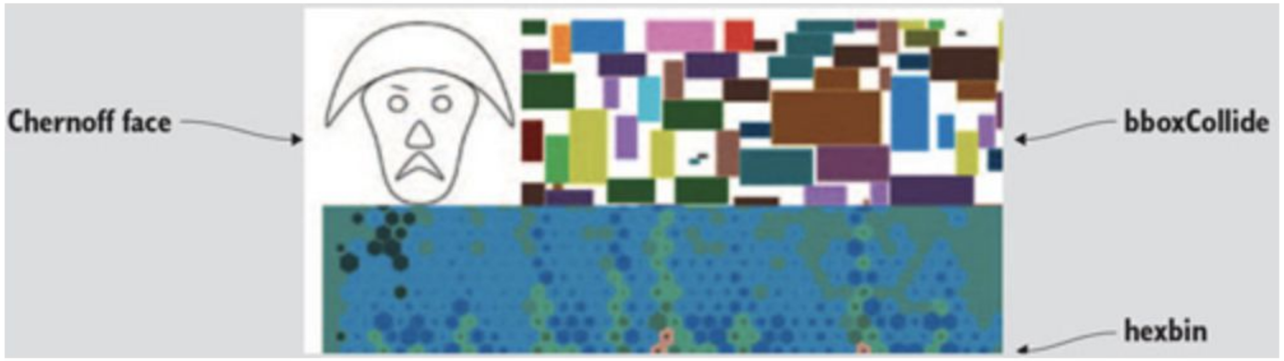
Plugins to add new layouts

Google Analytics uses Sankey diagrams to chart event and user flow for website visitors.



Plugins to add new layouts

- https://github.com/d3/d3-plugins



Plugins to add new layouts

sitestats.json

```
{
  "nodes":[                                    1
    {"name":"index"},
    {"name":"about"},
    {"name":"contact"},
    {"name":"store"},
    {"name":"cheese"},
    {"name":"yoghurt"},
    {"name":"milk"}
  ],
```

1 Each entry in this array represents a web page

Plugins to add new layouts

```
  "links":[
    {"source":0,"target":1,"value":25},       2
    {"source":0,"target":2,"value":10},
    {"source":0,"target":3,"value":40},
    {"source":1,"target":2,"value":10},
    {"source":3,"target":4,"value":25},
    {"source":3,"target":5,"value":10},
    {"source":3,"target":6,"value":5},
    {"source":4,"target":6,"value":5},
    {"source":4,"target":5,"value":15}
  ]
}
```

2 Each entry in this array represents the number of times someone navigated from the “source” page to the “target” page

Plugins to add new layouts

```
var sankey = d3.sankey()
   .nodeWidth(20)                          1
   .nodePadding(200)                          2
   .size([460, 460])
   .nodes(data.nodes)
   .links(data.links)
   .layout(200);                        3
```

1 Where to start and stop drawing the flows between nodes
2 The distance between nodes vertically—a lower value creates longer bars representing our web pages
3 The number of times to run the layout to optimize placement of flows

Plugins to add new layouts

Sankey drawing code

```
var intensityRamp = d3.scaleLinear()
   .domain([0,d3.max(data.links, d => d.value) ])
   .range(["#fcd88b", "#cf7d1c"])
d3.select("svg").append("g")
   .attr("transform", "translate(20,20)").attr("id", "sankeyG");          1
d3.select("#sankeyG").selectAll(".link")
   .data(data.links)
   .enter().append("path")
   .attr("class", "link")
   .attr("d", sankey.link())                                   2
   .style("stroke-width", d => d.dy)                                 3
   .style("stroke-opacity", .5)
   .style("fill", "none")
   .style("stroke", d => intensityRamp(d.value))                          4
   .sort((a, b) => b.dy - a.dy)
   .on("mouseover", function() {
      d3.select(this).style("stroke-opacity", .8); })                   5
   .on("mouseout", () => {
d3.selectAll("path.link").style("stroke-opacity", .5); })
```

1 Offsets the parent <g> of the entire chart
2 Sankey layout's link() function is a path generator
3 Note that layout expects us to use a thick stroke and not a filled area
4 Sets the stroke color using our intensity ramp, black to red indicating weak to strong
5 Emphasizes the link when we mouse over it by making it less transparent

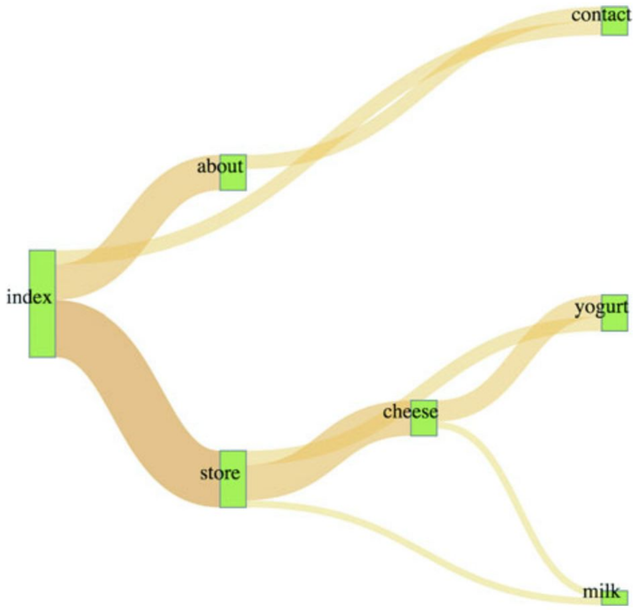
Plugins to add new layouts

```
d3.select("#sankeyG").selectAll(".node")
   .data(data.nodes)
   .enter().append("g")
   .attr("class", "node")
   .attr("transform", d => `translate(${d.x},${d.y})`)                    6
d3.selectAll(".node").append("rect")
   .attr("height", d => d.dy)
   .attr("width", 20)
   .style("fill", "#93c464")
   .style("stroke", "gray")
d3.selectAll(".node").append("text")
   .attr("x", 0)
   .attr("y", d => d.dy / 2)
   .attr("text-anchor", "middle")
   .style("fill", "black")
   .text(d => d.name)
```

6 Calculates node position as x and y coordinates from our data

Plugins to add new layouts

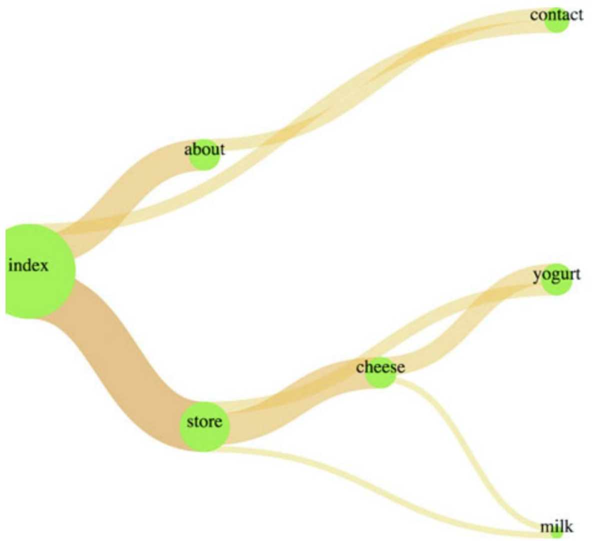
A Sankey diagram where the number of visitors is represented in the color of the path. The flow between index and contact has an increased opacity as the result of a mouseover event.



Plugins to add new layouts

A squid-like Sankey diagram

```
 sankey.nodeWidth(1);
d3.selectAll(".node").append("circle")
  .attr("height", d => d.dy)
  .attr("r", d => d.dy / 2)
  .attr("cy", d => d.dy / 2)
  .style("fill", "#93c464")
```



Plugins to add new layouts

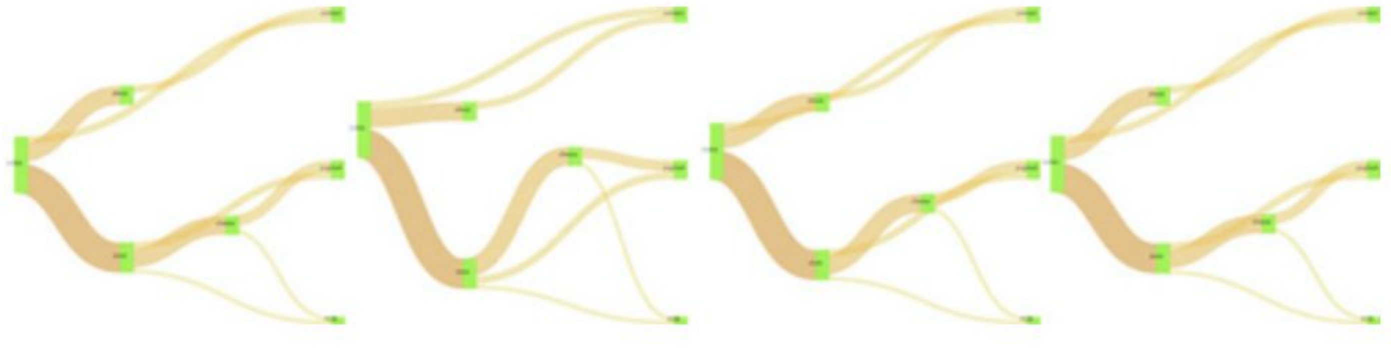
Visual layout function for the Sankey diagram

```
var numLayouts = 1;
d3.select("svg").on("click", runMoreLayouts);
sankey.layout(numLayouts);                          1
function runMoreLayouts() {
   numLayouts += 20;                             2
   sankey.layout(numLayouts);
   d3.selectAll(".link")
      .transition()
      .duration(500)
      .attr("d", sankey.link())                  3
   d3.selectAll(".node")
      .transition()
      .duration(500)
      .attr("transform", d => "translate(" + d.x + "," + d.y + ")")
}
```

1 Initializes the Sankey with only a single layout pass
2 I chose 20 passes because it shows some change without requiring us to click too much
3 Because the layout updates the dataset, we have to call the drawing functions again and they automatically update

Plugins to add new layouts

The Sankey layout algorithm attempts to optimize the positioning of nodes to reduce overlap. The chart reflects the position of nodes after (from left to right) 1 pass, 20 passes, 100 passes, and 200 passes.



Plugins to add new layouts

A word or tag cloud uses the size of a word to indicate its importance or frequency in a text, creating a visual summary of text. These word clouds were created by the popular online word cloud generator Wordle (www.wordle.net). Code: https://github.com/sly7-7/d3-cloud

Plugins to add new layouts

worddata.csv

```
text,frequency
layout,63
function,61
data,47
return,36
attr,29
chart,28
array,24
style,24
layouts,22
values,22
need,21
nodes,21
```

```
pie,21
use,21
figure,20
circle,19
we'll,19
zoom,19
append,17
elements,17
```

Plugins to add new layouts

Creating a word cloud with d3.cloud

```
var wordScale=d3.scaleLinear().domain([0,75]).range([10,120]);      1
d3.cloud()
  .size([500, 500])
  .words(data)                                              2
  .rotate(0)
  .fontSize(d => wordScale(d.frequency))                          3
  .on("end", draw)
  .start();                                               4
```

1 Use a scale rather than raw values for the font size (if you scale a word too large, the layout won’t draw it)
2 Assigns data to the cloud layout using .words()
3 Sets the size of each word using our scale
4 The cloud layout needs to be initialized—when it’s done it fires “end” and runs whatever function “end” is associated with

Plugins to add new layouts

```
 function draw(words) {                                              5
   var wordG = d3.select("svg").append("g")
     .attr("id", "wordCloudG")
     .attr("transform","translate(250,250)");
   wordG.selectAll("text")
     .data(words)
     .enter()
     .append("text")
     .style("font-size", d => d.size + "px")
     .style("fill", "#4F442B")
     .attr("text-anchor", "middle")
     .attr("transform", d =>
         "translate(" + [d.x, d.y] + ")rotate(" + d.rotate + ")")     6
     .text(d => d.text);
};
```

5 We’ve assigned draw() to “end”, which automatically passes the processed dataset as the words variable
6 Translation and rotation are calculated by the cloud layout

Plugins to add new layouts

A word cloud with words that are arranged horizontally

pie
data
chart
function
attr
need
array
layout
circle
elements
return
zoom
values
style
we'll
use
layouts
figure
nodes
append

Plugins to add new layouts

```
randomRotate=d3.scaleLinear().domain([0,1]).range([-20,20]);      1
    d3.cloud()
      .size([500, 500])
      .words(data)
      .rotate( () => randomRotate(Math.random()))                  2
      .fontSize(d => wordScale(d.frequency))
      .on("end", draw)
      .start();
```

1 This scale takes a random number between 0 and 1 and returns an angle between –20 degrees and 20 degrees
2 Sets the rotation for each word

Plugins to add new layouts

A word cloud using the same worddata.csv but with words slightly perturbed by randomizing the rotation property of each word.

zoom
data
function
chart
style
pie
values
layout
array
we'll
layouts
elements
figure
return
need
circle
attr
use
nodes
append

Plugins to add new layouts

Word cloud layout with key word highlighting

```
var keywords = ["layout", "zoom", "circle", "style", "append", "attr"]         1
d3.cloud()
  .size([500, 500])
  .words(data)
  .rotate(d => d.text.length > 5 ? 0 : 90)                                    2
  .fontSize(d => wordScale(d.frequency))
  .on("end", draw)
  .start();
```

1 Our array of keywords
2 The rotate function rotates by 90 degrees every word with five or fewer characters

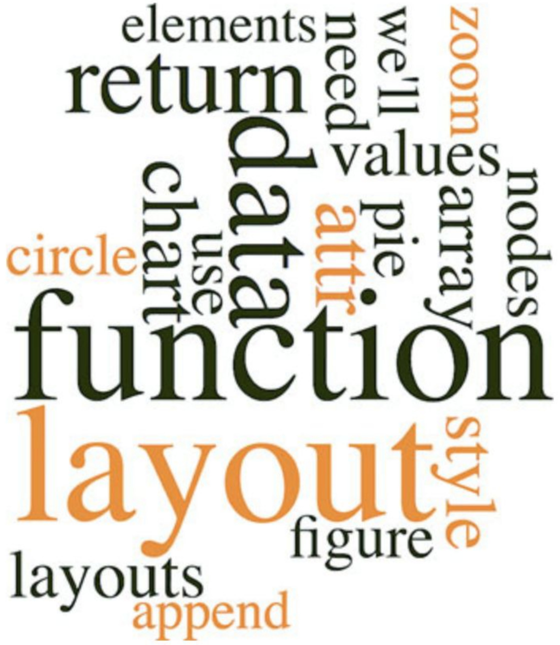
Plugins to add new layouts

```
function draw(words) {
  var wordG = d3.select("svg").append("g")
    .attr("id", "wordCloudG").attr("transform","translate(250,250)");
  wordG.selectAll("text")
     .data(words)
     .enter()
     .append("text")
     .style("font-size", d => d.size + "px")
     .style("fill", d => keywords.indexOf(d.text) > -1 ? "#FE9922" : "#4F442B") 3
     .style("opacity", .75)
     .attr("text-anchor", "middle")
     .attr("transform", d => "translate(" + [d.x, d.y] + ") rotate(" + d.rotate + ")")
.text(d => d.text);
 };
```

3 If the word appears in the keyword list, color it orange—otherwise, color it black

Plugins to add new layouts

This word cloud highlights keywords and places longer words horizontally and shorter words vertically.


elements
need
we'll
zoom
return
data
values
chart
attr
pie
array
nodes
circle
use
function
layout
style
figure
layouts
append