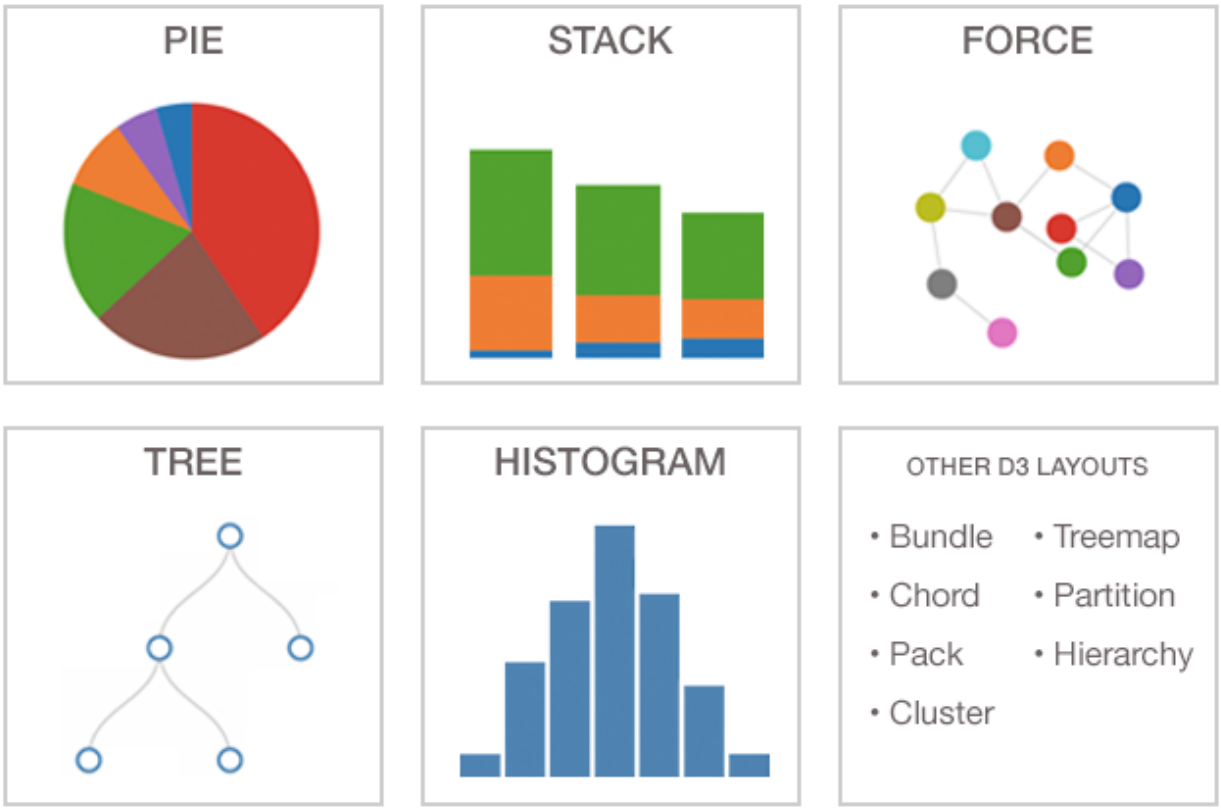
D3 Layouts

Author: Michael Oppermann. Last change date: Nov 14, 2019.

D3 layout methods are helpful for implementing a range of visualizations, such as stacked bar charts, treemaps, or node-link diagrams, that require a more advanced positioning of visual marks. For example, to lay out elements relative to each other instead of positioning them at given [x,y] coordinates.

> The D3 layout methods have no direct visual output. Rather, D3 layouts take data that you provide and re-map or otherwise transform it, thereby generating new data that is more convenient for a specific task. (Scott Murray)

D3 offers a number of different layouts, each with distinct characteristics, so make sure to consult the D3 documentation for implementation details. At the end of the article is a list of D3 layouts with links to the online documentation.



We will briefly introduce a few example layouts.

Force Layout (d3-force)

The force layout, initialized using `d3.forceSimulation()`, is typically used to create node-link diagrams. It consists of nodes and edges (links connecting the nodes) and helps you to visualize networks and the relationships between objects (e.g., social networks, relationships between politicians, protein–protein interactions, business relations, ...).

The name force-directed layout comes from the fact that these layouts use simulations of physical forces to arrange elements on the screen. The goal is to reduce the number of crossing edges, so that is easy for the user to analyze the whole network. You will learn more about networks during the lectures next week.

Force layouts can be used for other chart types too, such as beeswarm plots.

Histogram Layout (d3-array)

`d3.histogram()` creates a new histogram generator that groups many discrete samples into a smaller number of bins. Similar to other D3 layouts, you can use the accessor functions to customize it.

Example implementation:

```
const data = [1,234,25,38,100,12,34,40,150,89,199,80];

// Initialize scale
let x = d3.scaleLinear()
            .rangeRound([0, width])
            .domain([0, d3.max(data)]);

// Initialize histogram layout and set parameters
let histogram = d3.histogram()
                    .domain(x.domain())
                    .thresholds(x.ticks(10)); // number of bins

// Apply histogram function to the input data to assign data points to bins
const bins = histogram(data);
```

Stack Layout (d3-shape)

Stacked layouts, such as stacked bar charts or stacked area charts, are often used to show total values and per-category values simultaneously. d3.stack() computes layout positions that can be either used to place marks directly or they can be passed to a path generator (i.e., `d3.area()`).

Example implementation for a stacked area chart:

```
const data = [
  {
    year: 2017,
    apple: 10,
    orange: 20
  },
  {
    year: 2018,
    apple: 40,
    orange: 30
  }
  // ...
];

// Initialize stack layout
// Values for 'apples' and 'oranges' should be displayed
// on the same chart, by stacking them on top of each other.
const stack = d3.stack().keys(['apple', 'orange']);

// Apply layout function to the input data
const stackedData = stack(data);
// [
//  [[0, 10], [0, 40], key: 'apple', index: 0]
//  [[10, 30], [40, 70], key: 'orange', index: 1]
// ]

// Now you can use the 'stackedData' array
// in D3's enter-update-exit pattern to draw a path
// for each layer in the stacked area chart.
// ...
```

D3 API documentation:

- d3 stack
- d3 histogram
- d3 treemap
- d3 tree
- d3 pack
- d3 cluster
- d3 pie
- d3 force
- d3 chord

- ...