D3 Tutorial

Layouts

Edit by Jiayi Xu and Han-Wei Shen, The Ohio State University

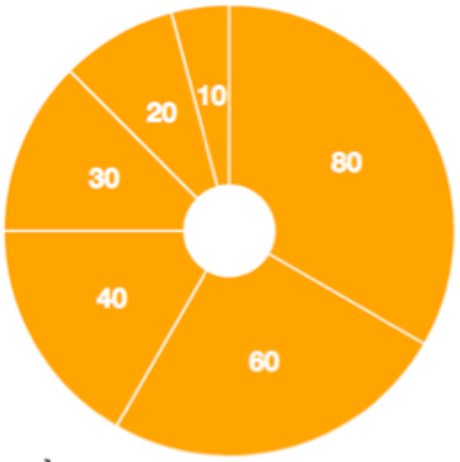
Layouts

- In essence, a **layout functio**n in D3 is just a JavaScript function that
  - Takes your data as input
  - Computes visual variables such as *position* and *size* to it so that we can visualize the data

Pie – Pie Generator: d3.pie()

- Given an array of data, the pie generator computes the necessary angles to represent the data
  - Output an array of objects containing the original data augmented by **start** and **end angles** which can be used to draw a pie chart by d3.arc()
- For example, we have an array of data

```
var data = [10, 40, 30, 20, 60, 80];
```



- Apply pie generator to the *data* to get *arcData*

```
var pieGenerator = d3.pie();
var arcData = pieGenerator(data);
```

- arcData:

```
▶ 0: {data: 10, index: 5, value: 10, startAngle: 6.021385919380437, endAngle: 6.283185307179586, …}
▶ 1: {data: 40, index: 2, value: 40, startAngle: 3.665191429188092, endAngle: 4.71238898038469, …}
▶ 2: {data: 30, index: 3, value: 30, startAngle: 4.71238898038469, endAngle: 5.497787143782138, …}
▶ 3: {data: 20, index: 4, value: 20, startAngle: 5.497787143782138, endAngle: 6.021385919380437, …}
▶ 4: {data: 60, index: 1, value: 60, startAngle: 2.0943951023931953, endAngle: 3.665191429188092, …}
▶ 5: {data: 80, index: 0, value: 80, startAngle: 0, endAngle: 2.0943951023931953, …}
```

Pie – Create a Pie Chart

- Data
  - Fruits in the warehouse

```
var fruits = [
    {name: 'Apples', quantity: 20},
    {name: 'Bananas', quantity: 40},
    {name: 'Cherries', quantity: 50},
    {name: 'Damsons', quantity: 10},
    {name: 'Elderberries', quantity: 30},
];
```

- Create a pie generator to generate *arcData*
  - Sort data by the name of fruits

```
var pieGenerator = d3.pie()
    .value(function(d) {
        return d.quantity;
    })
    .sort(function(a, b) {
        return a.name.localeCompare(b.name);
    });

var arcData = pieGenerator(fruits);
```

Pie – Create a Pie Chart

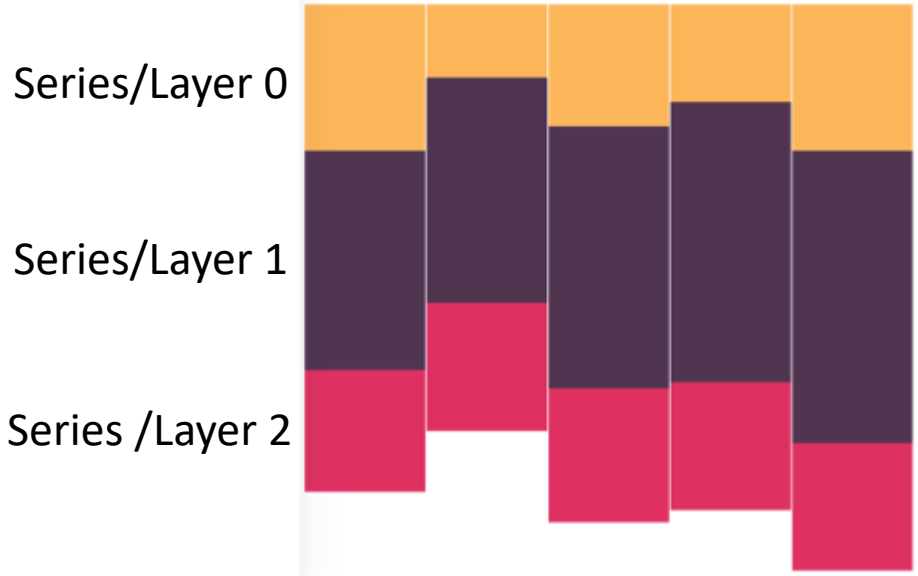
- Create an *arc* generator to draw arcs of the pie chart

```
var arcGenerator = d3.arc()
    .innerRadius(20)
    .outerRadius(100);

d3.select('g')
    .selectAll('path')
    .data(arcData)
    .enter()
    .append('path')
    .attr('d', arcGenerator);
```

Stack – Stack Bars

- Stacked bar graphs segment their bars on top of each other.
- They are used to show how a larger category is divided into smaller series/layers and what the relationship of each part has on the total amount



Stack – Create Stack Bars



- Data
  - Daily sales of fruits: apricots, blueberries, and cherries.

```
var data = [
  {day: 'Mon', apricots: 120, blueberries: 180, cherries: 100},
  {day: 'Tue', apricots: 60, blueberries: 185, cherries: 105},
  {day: 'Wed', apricots: 100, blueberries: 215, cherries: 110},
  {day: 'Thu', apricots: 80, blueberries: 230, cherries: 105},
  {day: 'Fri', apricots: 120, blueberries: 240, cherries: 105}
];
```

- Define their colors

```
// The colors of apricots, blueberries, and cherries
var colors = ['#FBB65B', '#513551', '#de3163'];
```

Stack – Create Stack Bars

- Series
  - Three fruits
    - Series 0: Apricots 
    - Series 1: Blueberries 
    - Series 2: Cherries 
- Create a stack Generator
  - Keys in generator are corresponding to keys in data

```
var stackGenerator = d3.stack()
  .keys(['apricots', 'blueberries', 'cherries']);
```

```
var data = [
  {day: 'Mon', apricots: 120, blueberries: 180, cherries: 100},
  {day: 'Tue', apricots: 60, blueberries: 185, cherries: 105},
  {day: 'Wed', apricots: 100, blueberries: 215, cherries: 110},
  {day: 'Thu', apricots: 80, blueberries: 230, cherries: 105},
  {day: 'Fri', apricots: 120, blueberries: 240, cherries: 105}
];
```

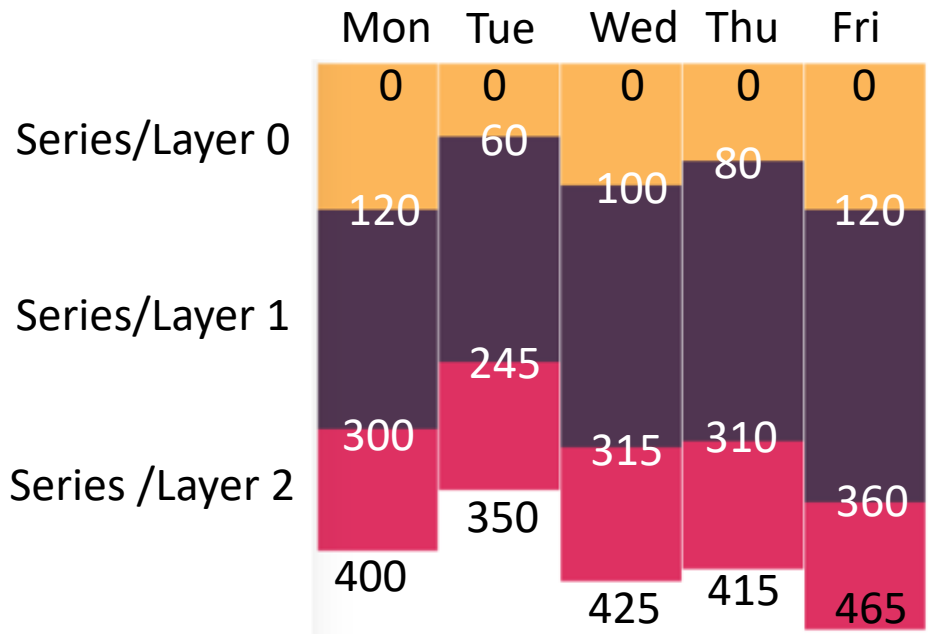
Stack – Create Stack Bars

- Series
  - Three fruits
    - Series 0: Apricots 
    - Series 1: Blueberries 
    - Series 2: Cherries 
- Stack Generator
  - The stack generator takes an array of **multi-series (or multi-layer) data** and generates an array for each series (or layer) where each array contains **lower and upper values** for each data point.
  - The lower and upper values are computed so that each series (layer) is stacked on top of the previous series (layer).

Stack – Create Stack Bars

```
var data = [
  {day: 'Mon', apricots: 120, blueberries: 180, cherries: 100},
  {day: 'Tue', apricots: 60, blueberries: 185, cherries: 105},
  {day: 'Wed', apricots: 100, blueberries: 215, cherries: 110},
  {day: 'Thu', apricots: 80, blueberries: 230, cherries: 105},
  {day: 'Fri', apricots: 120, blueberries: 240, cherries: 105}
];
```

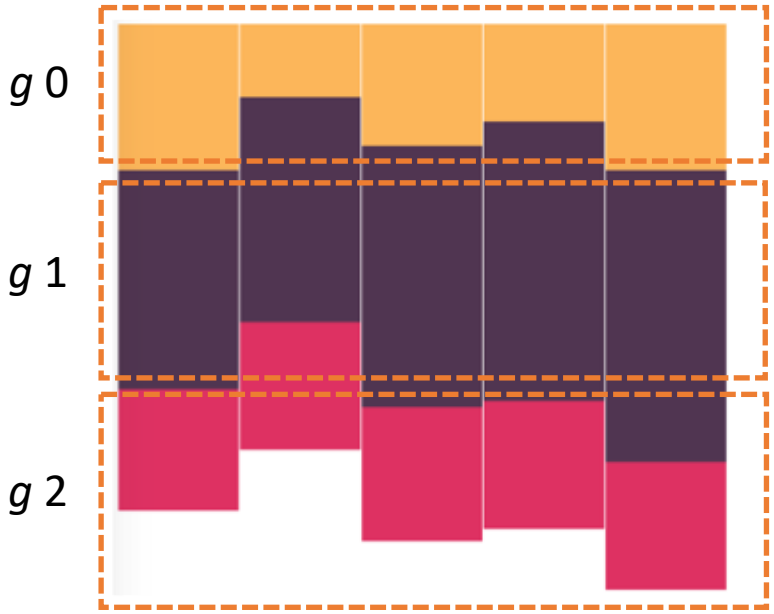
- Apply generator to data, we get:
- stackGenerator(data) = *[*
  - *[[0, 120],[0, 60],[0, 100],[0, 80],[0, 120]],// Series 0: Apricots*
  - *[[120, 300], [60, 245], [100, 315],[80, 310],[120, 360]], // Series 1: Blueberries*
  - *[[300, 400], [245, 350], [315, 425], [310, 415], [360, 465]]// Series 2: Cherries*
- *]*
- Three arrays are the computed data for three series
  - Each array (series) has 5 tuples, which are lower and upper values for the bars of 5 days



Stack – Create Stack Bars

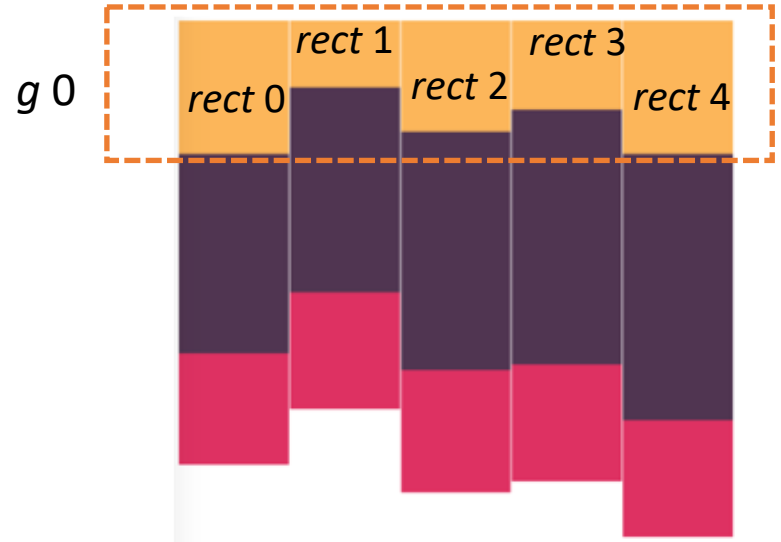
- Create a *g* tag for each series

```
var g = d3.select('svg')
    .selectAll('.series')
    .data(stackGenerator(data))
    .enter().append('g')
    .classed('series', true)
    .attr('fill', function(d, i) {
        return colors[i];
    });
```



Stack – Create Stack Bars

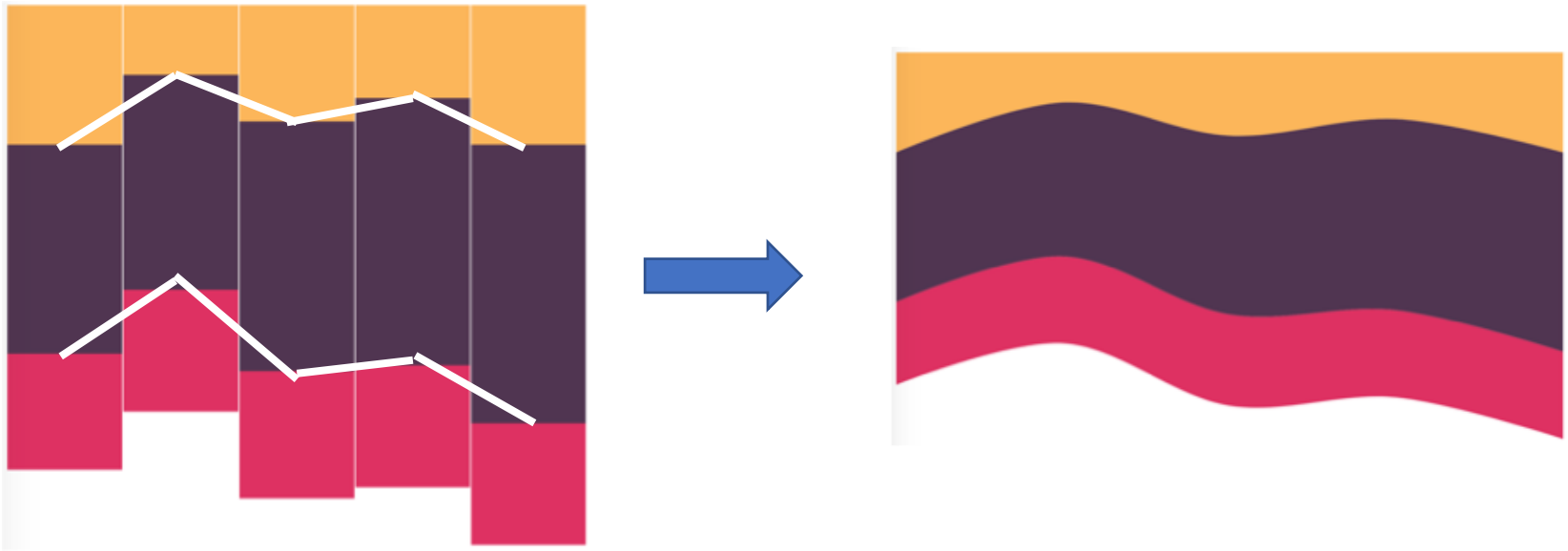
- Remember, for each series (*g* tag), the data is an array computed by stackGenerator
  - *E.g. the computed data for series 0 (apricots) is*
  - *[[0, 120],[0, 60],[0, 100],[0, 80],[0, 120]]*
  - Five tuples are corresponding to five days
- Create a *rect* element for each day




```
g.selectAll('rect')
    .data(function(d) {
        return d;
    })
    .enter().append('rect')
    .attr('x', function(d, i) {
        return i * 100;
    })
    .attr('y', function(d) {
        return d[0];
    })
    .attr('width', 99)
    .attr('height', function(d) {
        return d[1] - d[0];
    });
```

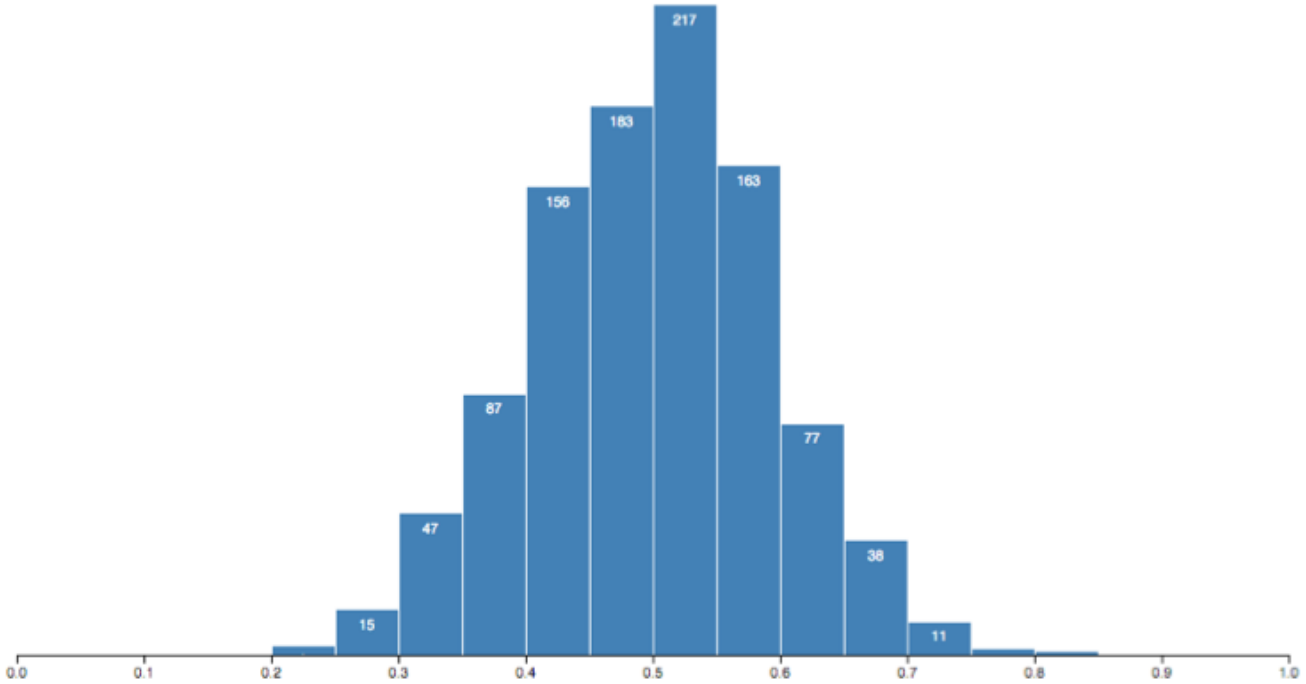
Stack – Stream Graphs

- We can generate stream graphs with the help of area generator: d3.area()



Histogram

- Histograms bin many discrete samples into a smaller number of consecutive, non-overlapping intervals.
  - They are often used to visualize the distribution of numerical data.



Histogram – Create a histogram

- Data
  - Generate 1000 samples from the normal distribution

```
var data = d3.range(1000).map(d3.randomNormal());
```

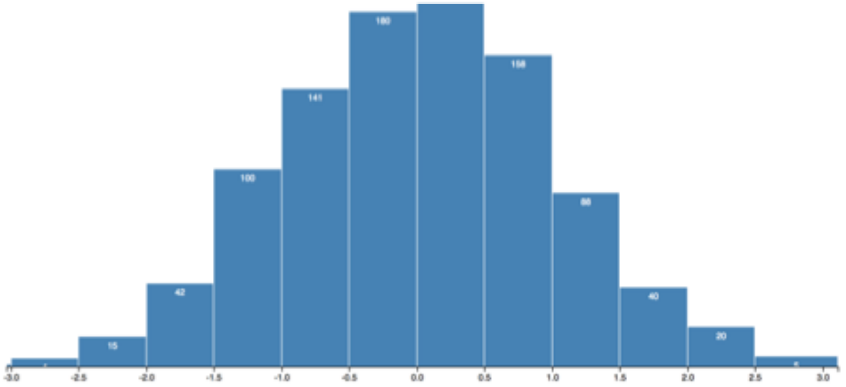
  - d3.extent() can get the extent of the data, i.e., an array [*min*, *max*] of the minimum and maximum value of this data

```
var dataExtent = d3.extent(data);
```

- Histogram generator to create data of bins

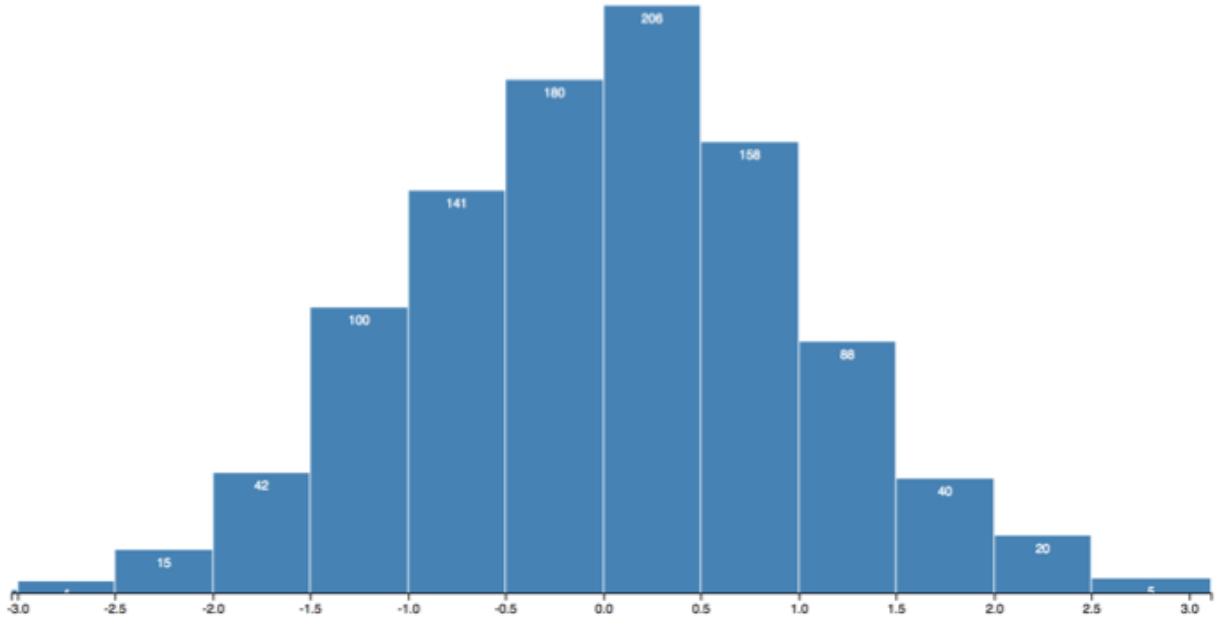
```
var binsGenerator = d3.histogram()
    .domain(dataExtent);

var binsData = binsGenerator(data);
```



Histogram – Create a histogram

- binsGenerator(*data*)
  - Computes three attributes (*length, x0,* and *x1*) for the given array of *data* samples
  - *length*
    - the *length* of the bin is the number of elements in that bin
  - *x0*
    - The lower bound of the bin (inclusive)
  - *x1*
    - the upper bound of the bin (exclusive, except for the last bin)

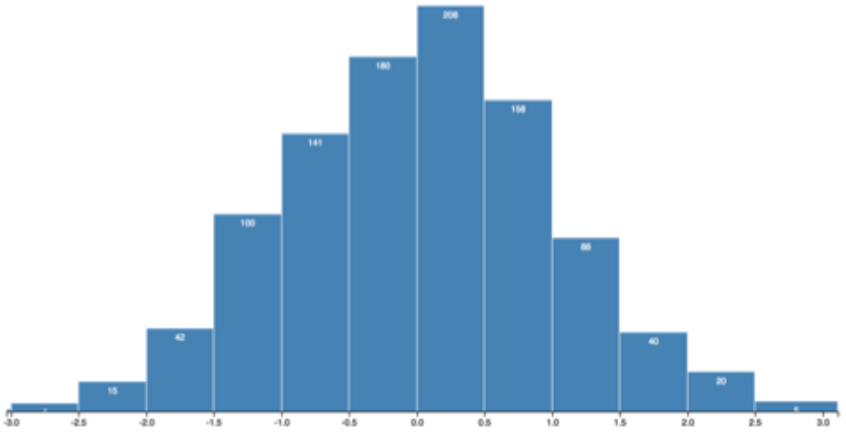


Histogram – Create a histogram

- Create scales
  - x: map values to width
  - y: map number of values in bins to height

```
var x = d3.scaleLinear()
    .domain(dataExtent)
    .rangeRound([0, width]);

var maxNumber = d3.max(binsData, function(d) {
    return d.length;
});
var y = d3.scaleLinear()
    .domain([0, maxNumber])
    .range([height, 0]);
```



Histogram – Create a histogram

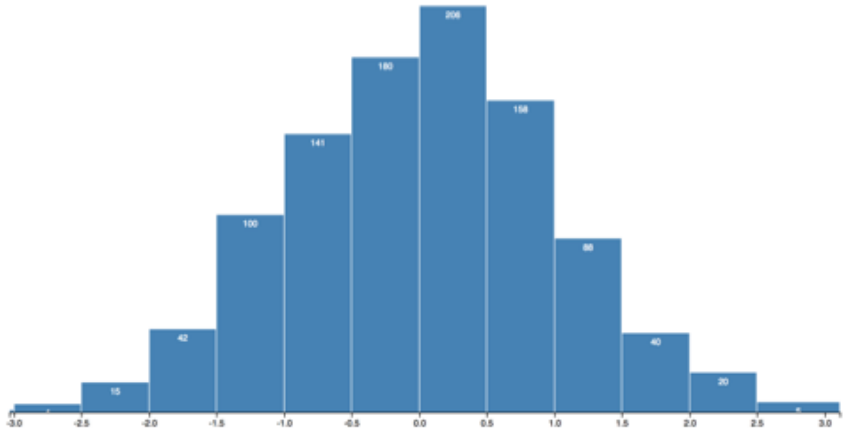
- Draw bars and an axis

```
var bar = g.selectAll(".bar")
  .data(binsData)
  .enter().append("g")
    .attr("class", "bar")
    .attr("transform", function(d) {
        return "translate("
            + x(d.x0) + "," + y(d.length)
            + ")";
    });

bar.append("rect")
    .attr("x", 0.5)
    .attr("width", function(d) {
        return x(d.x1) - x(d.x0) - 1;
    })
    .attr("height", function(d) {
        return height - y(d.length);
    });
```

padding

```
g.append("g")
    .attr("class", "axis")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x));
```



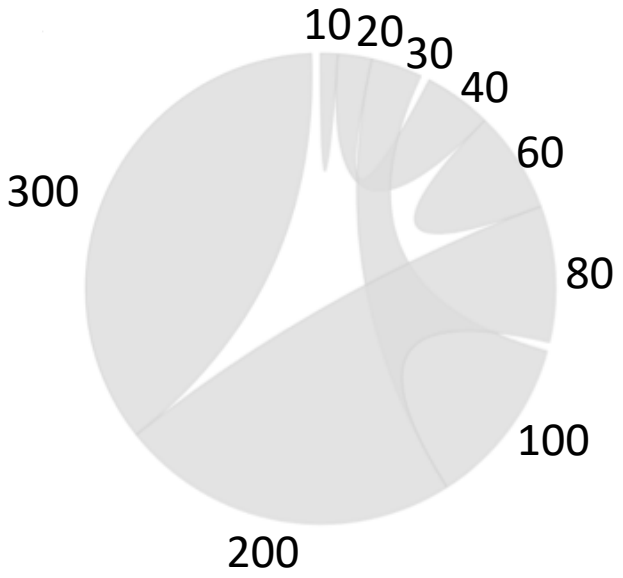
Chord

- Chord diagrams visualize links (or flows) between a group of nodes, where each flow has a numeric value.
- Example
  - Migration flow between and within regions
    - 2005 - 2010

Chord

- The data needs to be in the form of an n x n matrix (where n is the number of items)
  - First row represents flows from the 1st item to the 1st, 2nd and 3rd items etc.

```
var data = [
  [10, 20, 30],
  [40, 60, 80],
  [100, 200, 300]
];
```



Chord – Chord Generator: d3.chord()

- d3.chord()
  - Compute *startAngle* and *endAngle* of each data item
  - .padAngle(): set padding angle (gaps) between adjacent groups

```
var chordGenerator = d3.chord()
   .padAngle(0.04);
var chords = chordGenerator(data);
```

Chord – Ribbon Generator: d3.ribbon()

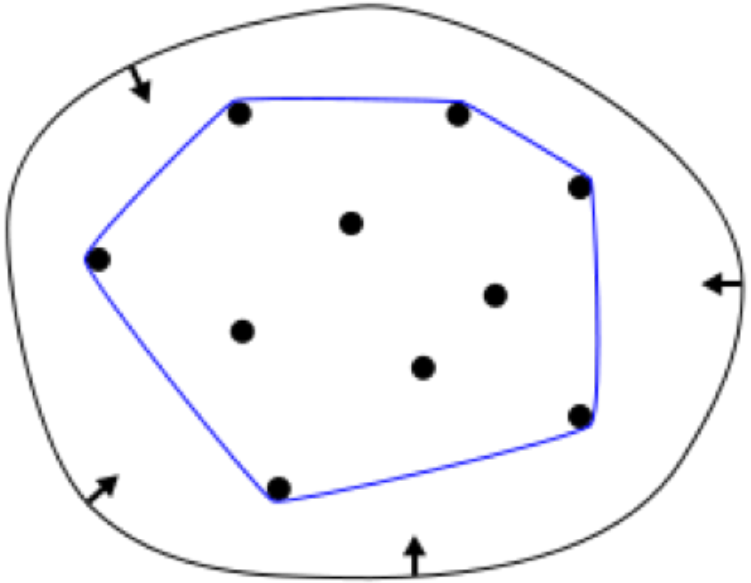
- d3.ribbon()
  - Converts the chord properties (*startAngle* and *endAngle*) into *path* data so that we can draw chords by *SVG*
  - .radius(): controls the radius of the final layout

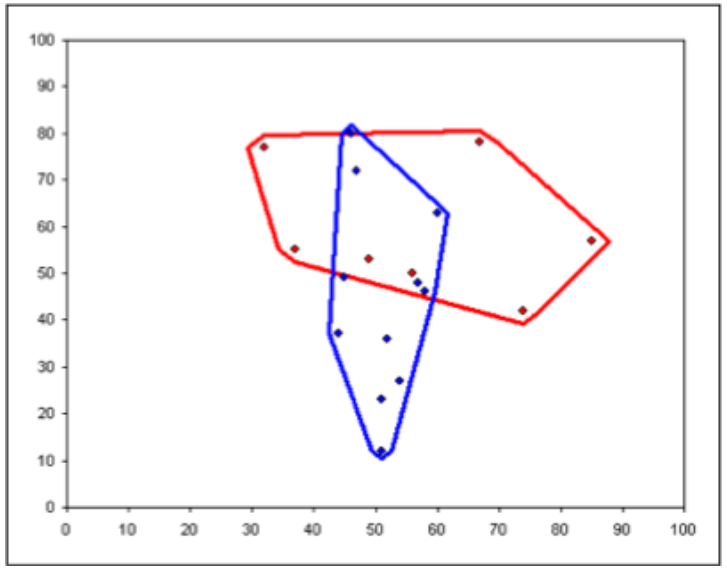
```
var ribbonGenerator = d3.ribbon()
    .radius(200);

d3.select('g')
  .selectAll('path')
  .data(chords)
  .enter()
  .append('path')
  .attr('d', ribbonGenerator);
```

Convex Hull

- In mathematics, the convex hull of a set of points in a Euclidean space is the smallest convex set that contains the points
  - Application: Visualize different sets/clusters of points on the screen



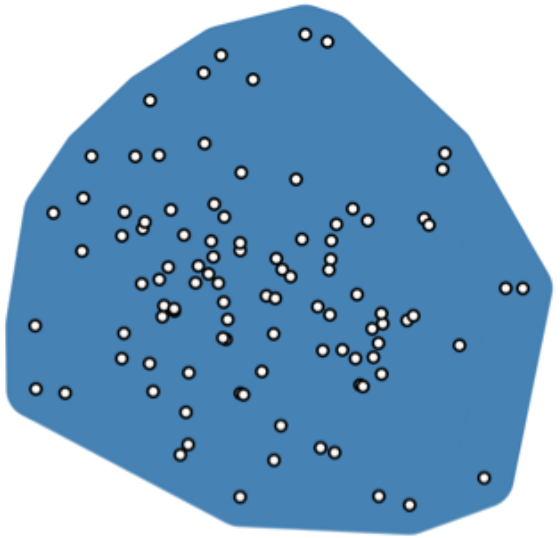


Convex Hull

```
var boundaryPoints = d3.polygonHull(points);
var hull = svg.append("path")
    .attr("class", "hull")
    .attr("d", "M" + boundaryPoints.join("L") + "Z");
```

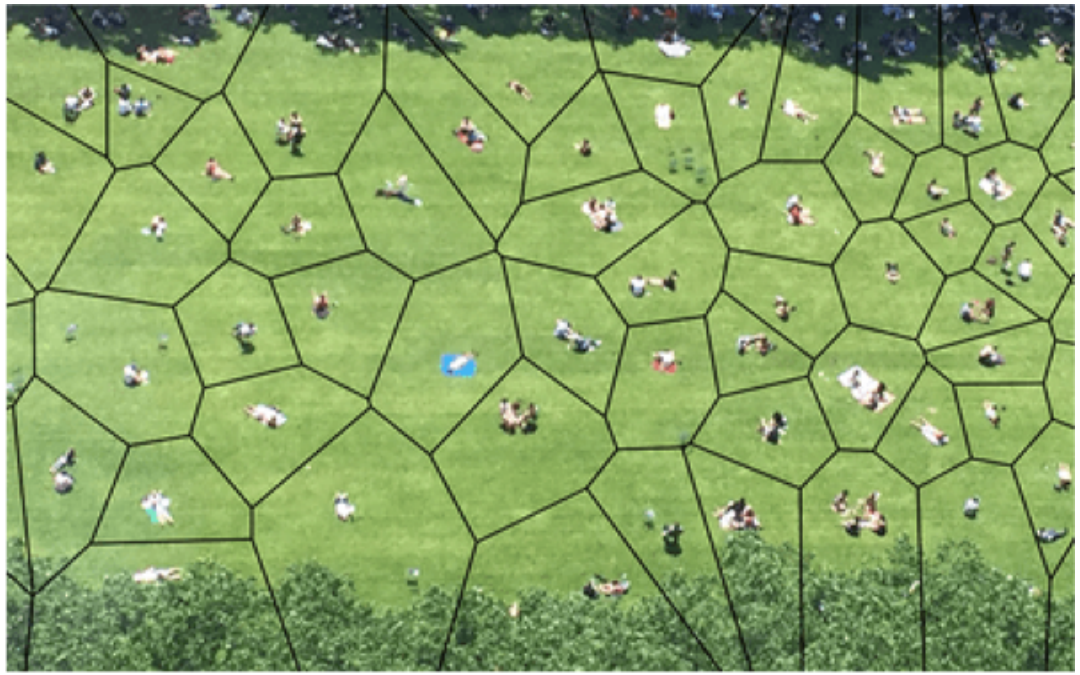
- Draw convex hull by *svg path*

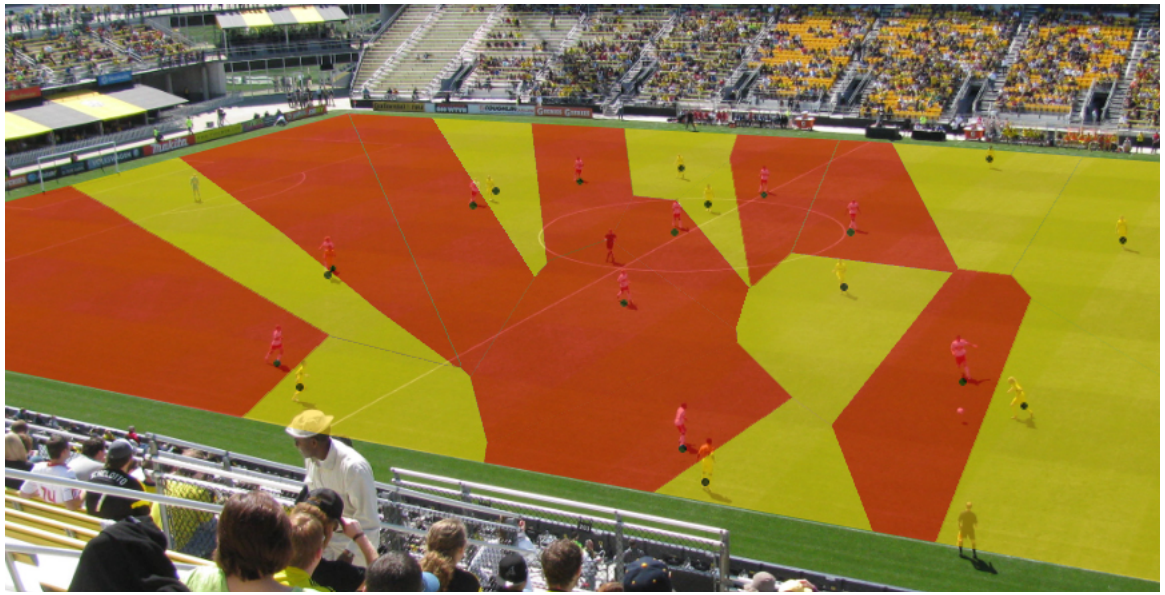
- Data
  - We have some random points stored in a variable *points*
- d3.polygonHull(*points*)
  - Generate the convex hull of the *points*
    - Returns *null* if *points* has fewer than three elements
  - The hull is represented as an array of the boundary points
    - Arranged in counterclockwise order



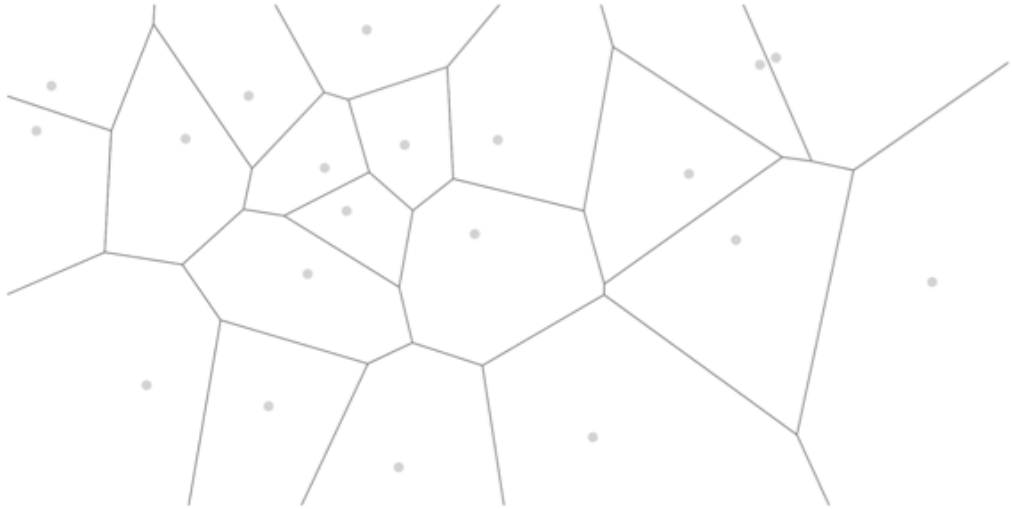
Voronoi

- In mathematics, a Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane
  - Application: Partition a plane based on points





Voronoi – d3.voronoi()



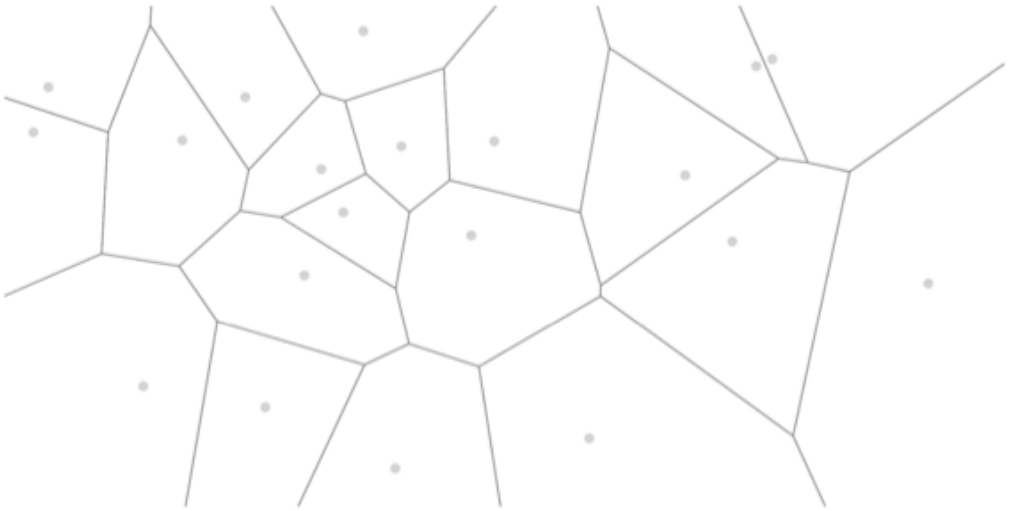
- Data
  - Generate coordinates of 20 points randomly

```
var points = d3.range(20).map(function() {
  return {
    x: Math.round(Math.random() * (width - radius * 2) + radius),
    y: Math.round(Math.random() * (height - radius * 2) + radius)
  };
});
```

Voronoi – d3.voronoi()

- d3.voronoi()
  - Computes the Voronoi diagram for the specified data points
  - .extent()
    - Take the extent of the screen

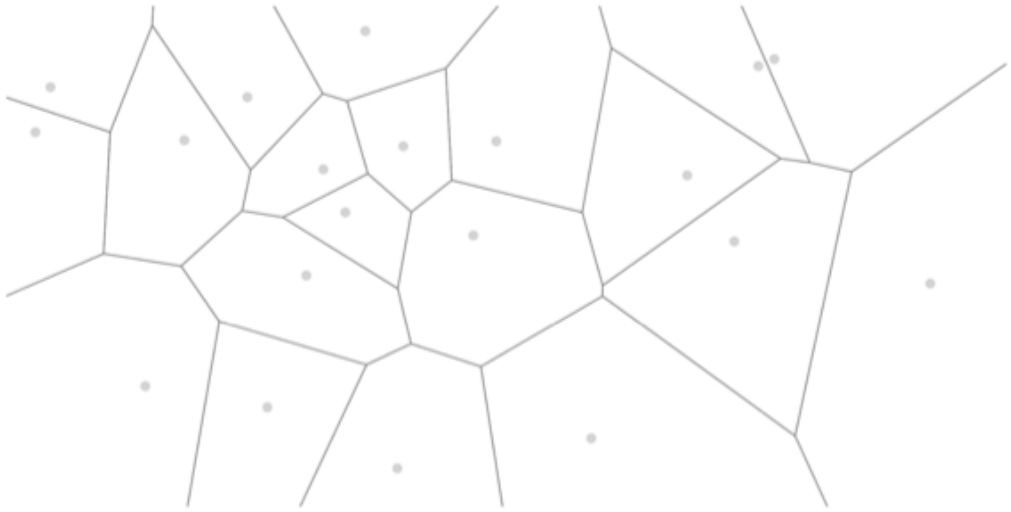
```
var voronoiGenerator = d3.voronoi()
    .x(function(d) { return d.x; })
    .y(function(d) { return d.y; })
    .extent([
        [0, 0],
        [width, height]
    ]);
```



Voronoi – d3.voronoi()

- Draw cell boundaries
  - voronoiGenerator.polygons(*points*)
    - Returns coordinates of the polygon which encloses each point

```
var cellBoundary = svg.selectAll("path")
    .data(voronoiGenerator.polygons(points))
    .enter()
    .append("path")
    .attr("d", renderCell);
```



- *renderCell* is a function to transform coordinates to *path* data of *SVG*

```
function renderCell(d) {
  return d == null ? null : "M" + d.join("L") + "Z";
}
```