Learn JS Data

Data cleaning, manipulation, and wrangling in JavaScript

Grouping Data

Grouping data is an important capability to have when doing data analysis. Often times, you will want to break apart the data by a categorical variable and look at statistics or details for each group.

D3 includes the powerful d3.nest functionality to produce these groupings with a minimal amount of code.

Nest Basics

Fundamentally, `d3.nest` is about taking a flat data structure and turning it into a nested one. The user gets to decide how the nesting should occur, and how deep to nest. This is a bit different then many group_by concepts, where only a single level of nesting is allowed.

Let's say we have the following CSV file of "expenses":

```
name,amount,date
jim,34.0,11/12/2015
carl,120.11,11/12/2015
jim,45.0,12/01/2015
stacy,12.00,01/04/2016
stacy,34.10,01/04/2016
stacy,44.80,01/05/2016
```

And that has been converted to a nice array of objects via our data reading powers into something like this:

```
var expenses = [{"name":"jim","amount":34,"date":"11/12/2015"},
  {"name":"carl","amount":120.11,"date":"11/12/2015"},
  {"name":"jim","amount":45,"date":"12/01/2015"},
  {"name":"stacy","amount":12.00,"date":"01/04/2016"},
  {"name":"stacy","amount":34.10,"date":"01/04/2016"},
  {"name":"stacy","amount":44.80,"date":"01/05/2016"}
];
```

And now we want to slice up this data in different ways.

First, let's use nest to group by `name`:

```
var expensesByName = d3.nest()
  .key(function(d) { return d.name; })
  .entries(expenses);
```

∧ This code is using d3.js

Which results in a nested data structure:

```
expensesByName = [
  {"key":"jim","values":[
    {"name":"jim","amount":34,"date":"11/12/2015"},
    {"name":"jim","amount":45,"date":"12/01/2015"}
  ]},
  {"key":"carl","values":[
    {"name":"carl","amount":120.11,"date":"11/12/2015"}
  ]},
  {"key":"stacy","values":[
    {"name":"stacy","amount":12.00,"date":"01/04/2016"},
    {"name":"stacy","amount":34.10,"date":"01/04/2016"},
    {"name":"stacy","amount":44.80,"date":"01/05/2016"}
  ]}
];
```

`expensesByName` is an array of objects. Each object has a `key` property - which is what we used as the grouping value using the `key` function. Here, we used the values associated with the `name` property as the key.

The `values` property of these entries is an array containing all the original data objects that had that key.

Summarizing Groups

The nested structure can be great for visualizing your data, but might be a little underwhelming for analytical applications. Never fear! d3.rollup is here!

With `rollup`, you provide a function that takes the array of values for each group and it produces a value based on that array. This provides for some very flexible *group by* functionality.

Here is a simple one to get back the counts for each name:

```
var expensesCount = d3.nest()
  .key(function(d) { return d.name; })
  .rollup(function(v) { return v.length; })
  .entries(expenses);
console.log(JSON.stringify(expensesCount));


=> [{"key":"jim","values":2},{"key":"carl","values":1},{"key":"stacy'
```

∧ *This code is using d3.js*

The individual records are gone (for better or worse) and in their place are the values returned by our rollup function. The naming stays the same (*key* and *values*) but the content is yours to specify. Note that the value passed into the `rollup` callback is the array of values for that key.

Here is another example where we get the average amount per person:

```
var expensesAvgAmount = d3.nest()
  .key(function(d) { return d.name; })
  .rollup(function(v) { return d3.mean(v, function(d) { return d.amou
  .entries(expenses);
console.log(JSON.stringify(expensesAvgAmount));


=> [{"key":"jim","values":39.5},{"key":"carl","values":120.11},{"key'
```

∧ This code is using d3.js

Pretty cool right? Any roll-up function you can think of, you can make happen. And you don't need to stop at just one. `rollup` can return an object, so you can easily produce multiple metrics on your groups.

```
var expenseMetrics = d3.nest()
  .key(function(d) { return d.name; })
  .rollup(function(v) { return {
    count: v.length,
    total: d3.sum(v, function(d) { return d.amount; }),
    avg: d3.mean(v, function(d) { return d.amount; })
  }; })
  .entries(expenses);
console.log(JSON.stringify(expenseMetrics));


=> [{"key":"jim","values":{"count":2,"total":79,"avg":39.5}},
 {"key":"carl","values":{"count":1,"total":120.11,"avg":120.11}},
 {"key":"stacy","values":{"count":3,"total":90.9,"avg":30.3}}]
```

∧ This code is using d3.js

Made with ♥ by The Bocoup Data Visualization Team.

Object Output

The array output can be useful for using map or forEach. But you can also have `d3.nest` return an object of the results, for direct access. Note the use of nest.object below.

```
var expensesTotal = d3.nest()
  .key(function(d) { return d.name; })
  .rollup(function(v) { return d3.sum(v, function(d) { return d.amou
  .object(expenses);
console.log(JSON.stringify(expensesTotal));


=> {"jim":79,"carl":120.11,"stacy":90.9}
```

∧ *This code is using d3.js*

And if you want to get real fancy, take a look at nest.map for getting a d3.map instance back.

Multi-Level Nesting

And you thought that single-level nesting was cool. Wait till you try multiple levels!

By adding more keys, you can sub-divide your data even further. Here is expense sums by name and then by date:

```
var expensesTotalByDay = d3.nest()
  .key(function(d) { return d.name; })
  .key(function(d) { return d.date; })
  .rollup(function(v) { return d3.sum(v, function(d) { return d.amou
  .object(expenses);
console.log(JSON.stringify(expensesTotalByDay));
```

```
=> {"jim":{"11/12/2015":34,"12/01/2015":45},
 "carl":{"11/12/2015":120.11},
 "stacy":{"01/04/2016":46.1,"01/05/2016":44.8}}
```

∧ *This code is using d3.js*

Now the `rollup` callback is called for each of our smaller subgroups.

The order of the `nest.key` calls determines the order of the grouping. If we reverse our keys, we get the totals by date and then by name:

```
var expensesTotalByDay = d3.nest()
  .key(function(d) { return d.date; })
  .key(function(d) { return d.name; })
  .rollup(function(v) { return d3.sum(v, function(d) { return d.amou
  .object(expenses);
console.log(JSON.stringify(expensesTotalByDay));
```

```
=> {"11/12/2015":{"jim":34,"carl":120.11},
 "12/01/2015":{"jim":45},
 "01/04/2016":{"stacy":46.1},
 "01/05/2016":{"stacy":44.8}}
```

∧ *This code is using d3.js*

Here the values are the same, but the mapping might be more convenient, depending on the questions you are trying to answer.

Derived Key Values

Remember, we are specifying our key value using a function. This gives us the power to group on derived or otherwise on-the-fly keys.

For example, if we wanted to find out totals for all expenses for each year, we would just do some basic string manipulation on the date string:

```
var expensesByYear = d3.nest()
  .key(function(d) { return d.date.split("/")[2]; })
  .rollup(function(v) { return d3.sum(v, function(d) { return d.amou
  .object(expenses);
console.log(JSON.stringify(expensesByYear));


=> {"2015":199.11,"2016":90.9}
```

∧ This code is using d3.js

All this flexibility provides for a powerful toolkit for exploring your data.

Next Task

Working with Strings

See Also

- Mister Nester - a `d3.nest` power tool!
- Phoebe Bright Nest Tutorial - lots more nest examples