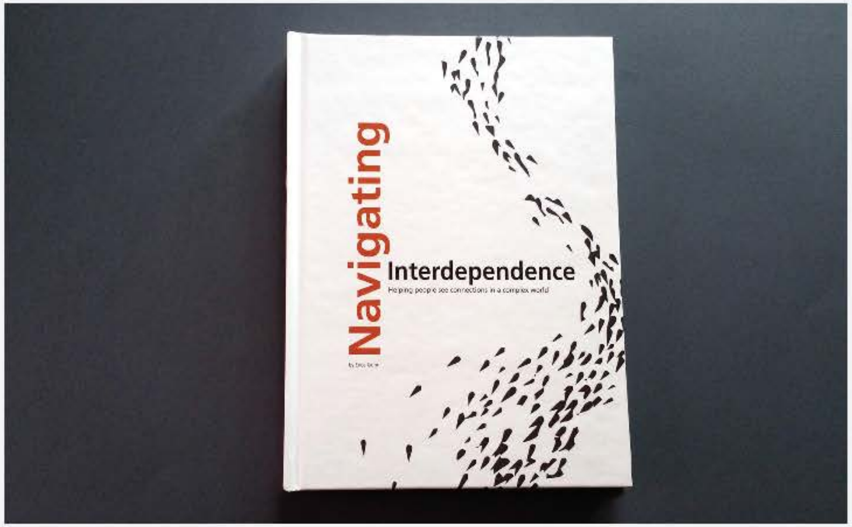
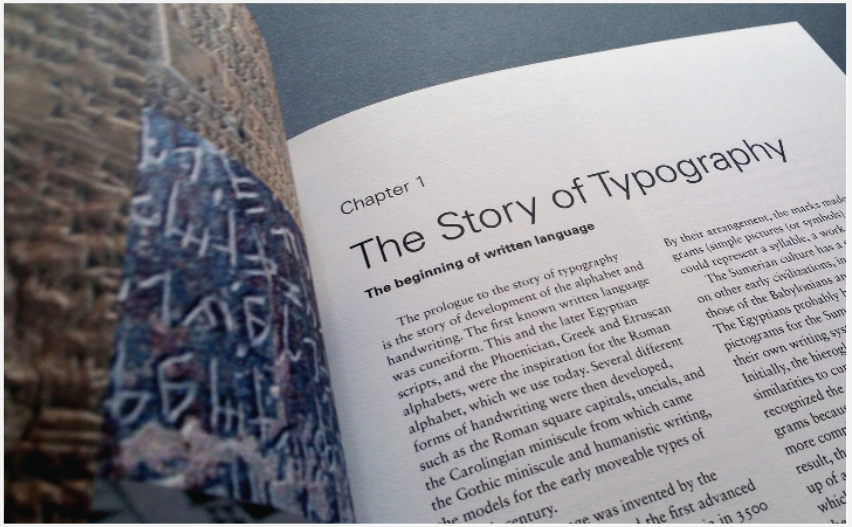

Navigating
Interdependence
Helping people see connections in a complex world


A complex system [is one] in which large networks of components with no central control and simple rules of operation give rise to complex collective behavior, sophisticated information processing, and adaptation via learning or evolution.
Melanie Mitchell
Introduction


Contents
PART 1: THEORETICAL BACKGROUND
1 Introduction
5 The Challenge of Complex Systems
9 Communicating Science
23 Design for Science Communication
29 Visualizations of Interdependence
Graphics that explain
Graphics that explore
Graphics that evoke emotion
Design for complex systems
Visualization as discourse
PART 2: PRACTICAL APPLICATION
75 Soil as Case Study
Preliminary experiments
Continuing experiments
Consolidating design
Planning the exhibition
Reflections
Conclusions and Future Work
References
Appendix


People will forget what you said. People will forget what you did. But people will never forget how you made them feel.
Maya Angelou
The deficit model



had made the connection between the written and spoken word. The seed for a fully fledged phonetic alphabet had been sown.

In 2500 B.C., the Egyptians made an ever greater contribution to the development of handwriting—the invention of the reed pen and papyrus (which came from the papyrus plant) as a writing surface. These new writing tools were to be a dynamic force, because they made the act of writing more accessible to a much wider audience. Throughout the history of handwriting and typography, the development of new image-making tools and surfaces has been responsible for corresponding changes in letterforms. The reed pen and papyrus enabled people to write faster and consequently, a new, simpler Egyptian script based on hieroglyphics developed, called hieratic.

The first alphabets

There are many theories as to the origins of the alphabet, but, whichever one is true, its invention was a landmark of great magnitude in the development of civilization. An alphabet is a writing system with one unique visual sign (letter) for each consonant and vowel sound (although there were no vowels in the earliest alphabets), which can be combined to form visual units (words) to represent a spoken language. From about 1500 B.C., the alphabet has outperformed all other systems of writing and has survived intact through many revisions.

If the origins of the alphabet are unknown, what is clear is that in 1500 B.C., a Semitic people, the Phoenicians, developed a new phonetic written language—the first alphabetic system. It consisted of a sign for each of 22 consonant sounds and showed some visual similarity to the Egyptian hieratic script.

Earliest literate society in the western Mediterranean

Significantly, though, the Phoenician writing system did not use any pictograms, which made the new language much more economical.

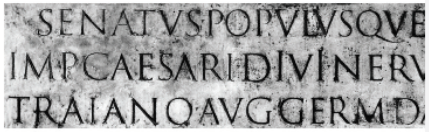
The Phoenicians used their location on the eastern Mediterranean coast to exploit sea travel as a means of exporting goods to other countries in the region. Through the culmination of these business relationships, their trading partners were gradually exposed to their alphabetic system of writing, and, by 800 B.C., its influence had permeated westward to Greece.

In Greece at this time there were many local dialects and alphabets in use. But eventually, two principal alphabets emerged—the Ionian, in the east of the country, and the Chalcidian, in the west. There were many similarities between the Phoenician and early Greek alphabets—the older and names of the letters were the same, as was the direction of the writing which was from right to left (or *sinistrorsum abversatrum*). From about 500 B.C., the direction of writing was reversed so that it read from left to right. In 403 B.C., the Ionian alphabet was officially adopted in Athens as the Classical Greek alphabet. However, it was the Chalcidian, which had been more influenced by the Phoenicians, that was to play a more significant role in the development of the Roman alphabet. It was to become the model for all the succeeding alphabets of Western Europe.

Around 625 B.C., trade developed between the Greeks and the Etruscans, a people who had settled on the west coast of Italy after migrating from their homeland in Asia Minor. It was through this trading relationship that the influence of the western Greek Chalcidian alphabet spread to Italy, and it is believed that the Etruscan alphabet derived from the Chalcidian.

The Etruscans remained dominant in Italy for about 250 years, reaching the height of their power about 500 B.C. Some one hundred years later, their conquests were lost to the rising power of Rome. The legacy that the Etruscans left to the Romans was considerable. It is to them that the Romans owed their architecture, law, roads, and other trappings of a civilized society. The Etruscan alphabet was the basis for the Roman alphabet which we use today. After modification—the Romans changed some letters, added new ones, and deleted others—they were left with an alphabet of 23 letters, which is the same as the Roman alphabet used today (not including J, U and W, which were added in the Middle Ages).

Roman capital carved into the Trajan column



The development of the roman alphabet

From about 500 B.C., the Romans began to expand their Empire through invasion and colonization and imposed their written alphabet on the conquered nations in the process. The effect was that the letters of the Roman alphabet became an established set of signs which were understood in many parts of Europe and Asia Minor—an "international" written language.

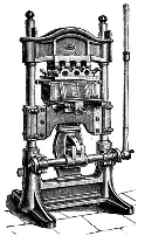
In the Roman Empire, two main kinds of letterforms were widely used—square capital letters (called monumental for formal inscriptions and, later, a cursive style (which shaped like today) for informal purposes such as letter writing. This cursive style is the origin of our lower-case letters (called minuscules). The finest example of formal Roman inscriptional square capitals, also called quadrata, is Trajan's Column in Rome, completed in A.D. 114. These authoritative and beautifully proportioned letterforms were unsurpassed from geometric sheets such as the square, circle, or triangle and were the model for the capital letters of our alphabet today.

early forms were austere, heavy, and condensed, with a strong vertical emphasis. By the 15th century, they had become more ornamented with the vertical strokes ending in points. Slowly, the ascenders and descenders became shorter, and the style became less and less stylistic. It was one of the later styles, Textura—a formal black letter used for religious and legal purposes—that was the model for Gutenberg's first movable types.

Although it is now known that the Chinese were experimenting with movable ceramic types as early as the 11th century, Johann Gutenberg is the acknowledged father of movable type. He lived in Mainz in Germany and by trade he was a goldsmith, but he had acquired technical knowledge of the art of printing (prints had been made from hewn wood blocks many years earlier). In 1440, he began a series of experiments which, ten years later, resulted in the invention of printing from movable type. He used his knowledge of existing technology and materials—the screw press, oil-based inks and paper (which was a Chinese invention)—but it was the manufacture of type that consumed his energy.

As a goldsmith he had considerable skill and knowledge of the patterning, mixing, and casting of metal, and, with great ingenuity and tenacity, he eventually developed a method of manufacturing type. It involved the engraving of every character in relief and reverse on a steel punch, which was then struck with a mallet into the softer (a blue of copper). The strike was an into the matrix (a master mold for casting each letter) by a process called justification. Then the matrix was put into an adjustable hand mold into which a hot alloy (mold of lead and antimony was poured, thereby casting each sort (the term for a single piece of type).

The process of hot metal casting gave rise to the term "hot metal typesetting" when referring to setting from metal types. This significant technological feat was the catalyst for a revolution in printing in Europe. The basic principles that Gutenberg developed for type manufacturing and letterpress printing were still used until well into the 20th century. Letterpress printing is a process whereby the impression on the paper is taken from the raised surface of the type.



Gutenberg's printing press

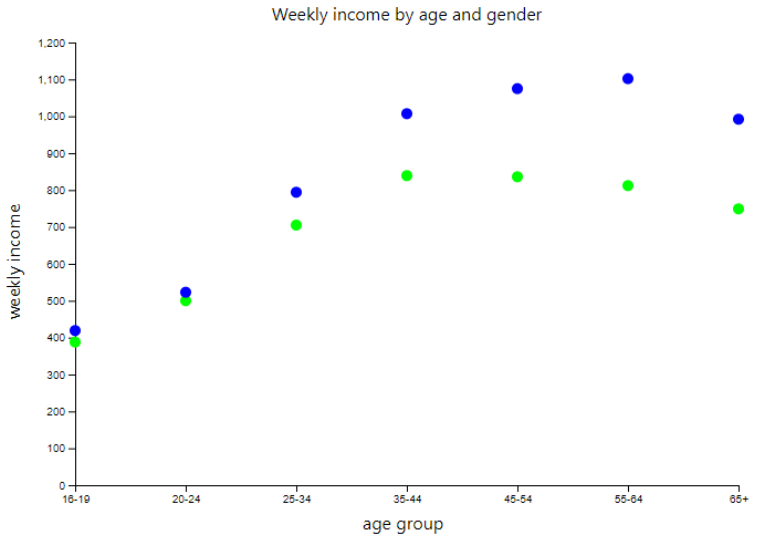
Textura Type Source page of the Gutenberg Bible

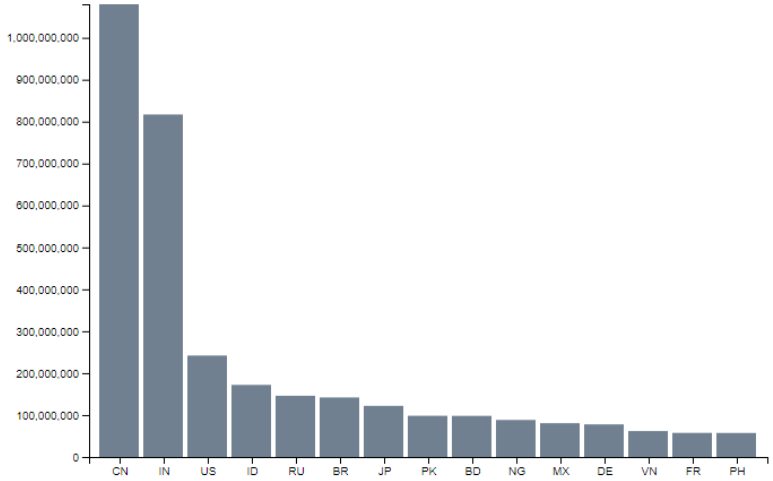
Artist's rendering of Johann Gutenberg, inventor of moveable type printing

The visible fruits of his labors finally emerged in the printing and publishing of his 42-line Bible in 1455, the earliest extant printed book from moveable types in the Western world. He had, however, printed the Mainz Indulgences during the previous year, for which he used a more cursive style of black letter type called Schwabacher or fraktur. Gutenberg's use of the Textura black letter type produced a magnificent and authoritative effect on the page, quite equaling the manuscripts that it was imitating. But it was condensed, monotonous, and heavy-looking and, therefore, difficult to read. For his Catholicon, in 1460, he used a different black letter type, Rotunda, which was more open and legible, but lacked the style and authority of his Textura. There was a human cost to pay for Gutenberg's invention—the job losses of the scribes and copyists who had been the "printers" of manuscripts until this time.

Today: Drawing with generator functions

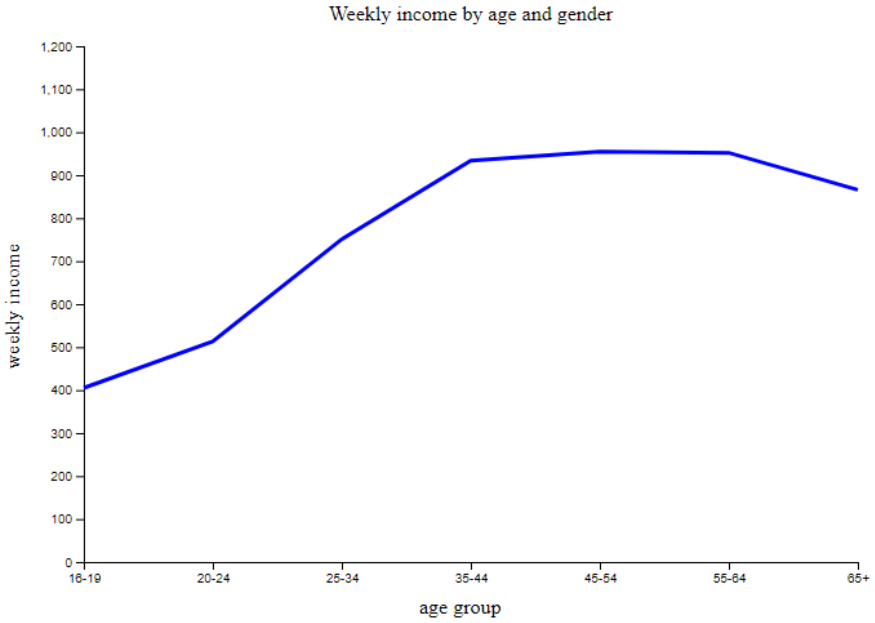
So far, we've worked with visualizations that create one object for each item in a data set.





Drawing with generator functions

What happens when you need to use a bunch of points in your data set to make a single shape?



Now, d3 needs to go through all of the points in your dataset, and use them as points for a path object. To do this, we need a generator function.

Generator functions

A generator function is a function that tells d3 what to do with a set of points. It is called from the d3 .append code, just like we did before for adding points to a scatterplot.

At the top of your file (before d3.csv), add the following:

```
var makeLine = d3 .line ( )
   .x ( function ( d )  {  return scaleX ( d.age ); } )
   .y ( function ( d )  {  return scaleY ( d.total ); } );
```

Notice:

- This function d3.line( ) is stored in a variable (similar to what we did with d3.nest).
- It contains information about how to turn data into values using scale functions.
- We've written the function, but we haven't called it yet, so right now it doesn't *do* anything.

Today: Drawing with generator functions

In the drawing code, remove .selectAll() and .enter()

```
// Add the path
   svg .append ( "path“ )
      .dat um ( dataIn )        // datum (not data!) tells d3 that all of the data
                                // belongs to a single line
      .attr ( "class“ ,  "line“ )
      .attr ( "d“ , makeLine )   // the "d" attribute is just part of how the path
                                 // element is defined, like "cx" or "cy" for a circle.
                                 // it calls the makeLine function above, and hands
                                 //it sets of points that the path should contain.
      .attr ( 'fill‘ , 'none‘ )
      .attr( 'stroke‘ , 'blue‘ )
      .attr( 'stroke-width‘ , 3 );
```

From Line to Area

To draw an area chart instead, we just adjust the generator function.

```
var makeArea = d3 .area ( )                                    //area instead of line
    .x ( function ( d )  { return scaleX ( d.age );  } )
   .y0 ( scaleY ( 0 ) )                                        //sets the baseline
    .y1( function ( d )  { return scaleY ( d.total ); } );  //y1 instead of y
```

Change the function call in the data drawing:

```
 svg.append ( "path“ )
       .datum ( dataIn )
       .attr ( "class", "area“ )
       .attr ( "d",  makeArea  )
```

And update styling (usually give it a fill, turn off stroke)

Generator functions in maps:

Generator functions and geographic projections are used together to draw map outlines using d3. To do this, we will use a special kind of file called a geoJSON

Contains all of the latitude and longitude points that define the map outlines, divided up into features, geometries, coordinates, etc.

Features usually have .properties, which is where they define things like the state name and other important information.

Inside a geoJSON

Feature collection
→ Features
→ Geometry
→ Coordinates
→ Properties


```
{"type":"FeatureCollection", "features": [
{"type":"Feature","geometry":{"type":"MultiPolygon","coordinates":[[[[-68.92401,43.885407],[-68.87478399999999,
.904714999999996],[-68.849009,43.849841],[-68.888483,43.803781],[-68.944433,43.835325999999995],[-68.92401,43.
[[[-71.083924,45.305451],[-71.03820999999999,45.311921999999996],[-71.012757,45.34476],[-70.949365,45.331536],
.912111,45.296197],[-70.892822,45.239171999999996],[-70.84443,45.234513],[-70.8340195374401,45.2717944023968],
45.286941],[-70.808613,45.311606],[-70.819471,45.341435],[-70.80624399999999,45.376557999999996],[-70.8256119
.400304999999996],[-70.781471,45.431159],[-70.755567,45.428360999999995],[-70.729972,45.399359],[-70.677995,45
[-70.634661,45.383607999999995],[-70.635498,45.427817],[-70.674903,45.452399],[-70.723396,45.510394],[-70.6882
.563981],[-70.64957799999999,45.598147],[-70.591275,45.630551],[-70.55282390337851,45.6678060578851],[-70.5527
.667836],[-70.44690299999999,45.704043999999996],[-70.383552,45.734868999999996],[-70.415684,45.786158],[-70.3
.808485999999995],[-70.329748,45.853795],[-70.259117,45.890755],[-70.252526,45.933175999999996],[-70.26541,45.
[-70.31296999999999,45.961856],[-70.303034,45.998976],[-70.317629,46.019079999999995],[-70.30673399999999,46.0
.26634899999999,46.100992999999995],[-70.239566,46.142762],[-70.290896,46.185838],[-70.255492,46.246444],[-70.
.284428],[-70.205719,46.299865],[-70.207415,46.331316],[-70.161337,46.360983999999995],[-70.118597,46.38423299
[-70.080292,46.410531],[-70.053748,46.429235999999996],[-70.02301978705619,46.573486472517295],[-69.997086,46.
.695229999999995],[-69.818552,46.875029999999995],[-69.566383,47.125032],[-69.43919799999999,47.25003299999999
.219996,47.457159],[-69.156074,47.451035],[-69.108215,47.435831],[-69.039301,47.42217],[-69.053885,47.37787799
[-69.0402,47.2451],[-68.966433,47.212711999999996],[-68.90098499999999,47.178518999999994],[-68.80353699999999
.216032999999996],[-68.675913,47.242626],[-68.60481899999999,47.249418],[-68.588725,47.281721],[-68.507432,47.
[-68.460064,47.286065],[-68.375615,47.292268],[-68.384281,47.326943],[-68.361559,47.355605],[-68.26971,47.3537
.204263,47.339729999999996],[-68.153509,47.314038],[-68.08289599999999,47.271921],[-67.998171,47.217842],[-67.
.196141999999995],[-67.889155,47.118772],[-67.789761,47.065743999999995],[-67.789799,46.794868],[-67.788406,46
```

Intro to map data

geoJSON is a special format of data used for maps. Census bureau has a good collection of US maps:
https://www.census.gov/geo/maps-data/data/tiger-cart-boundary.html

Natural Earth has a good collection of world maps:
http://www.naturalearthdata.com/

Often, you will find maps as SHP files rather than geoJSON. Mapshaper is a handy tool for converting the files :
http://mapshaper.org/

Download US States file from Census, unzip. Load both .shp and .dbf files into mapshaper, export geoJSON.

Also possibly useful:
http://geojson.io

Advanced notes:

GeoJSON files can be huge, and they are often far more detailed (higher resolution) than we either need or want. To deal with this, there is a helper library called TopoJSON that uses topology to simplify map shapes.

You download and use topoJSON just like any other library, with some small edits to how you dig into the map data.

For now, just be aware that more than one standard map file exists, and if you find tutorials using topoJSON you may need to add a library and adjust your data calls.

Making our first map: setup

Load map data into the browser using d3.json, and console.log it to look at its structure.

Before we do any more with it, first a little cleanup, because we want to be able to use the svg size to adjust the map:

 around svg with class svgBox
<svg> width and height set to 100%, add an id="svg"

CSS style the containing div to take up the whole window:

```
.svgBox  {
      width:  100vw ;
      height:  100vh ;
}
```

Making our first map: setup

Check that the width and height are set properly:

```
var  width = document .getElementById ( 'svg1‘ ) .clientWidth ;
var  height = document .getElementById ( ‘svg1‘ ) .clientHeight ;
```

Set margins to zero for now.

Drawing map outlines: projections and generator functions

Before we can draw the map, we need to identify a projection to convert the latitude and longitude values in the data into points on our screen.

https://bl.ocks.org/mbostock/29cddc0006f8b98eff12e60dd08f59a7

We want to use the Albers projection, centered on the United States.

```
var albersProjection = d3 .geoAlbersUsa ( )
   .scale ( 700 )
   .translate ( [ (width/2) ,  (height/2) ] );
```

Next, set up a path generator, and tell it to use the projection:

```
path = d3 .geoPath ( )
   .projection ( albersProjection );
```

Actually drawing the map

```
svg.selectAll ( "path“ )
        .data ( dataIn .features )
        .enter ( )
        .append ( "path“ )
        .attr ( "d“ , path )
        .attr ( "class“ ,  “state “ )
        .attr ( 'fill‘ , 'gainsboro‘ )
        .attr ( 'stroke‘ , 'white‘ )
        .attr ( 'stroke-width‘ , .2 );
```

Adding points to a map from a file:

To plot things on a map, you'll need a list of latitude and longitude points for the data.

The negative values are important!

	A	B	C	D
1	name	lat	long	
2	Harold Parker State Fc	42.6103	-71.0904	
3	Campus Life - Northea	42.3398	-71.0892	
4	Ground Zero	42.50793	-71.1048	
5	Just Between Neighb	28.0781	-82.7637	
6	Raccoon Race	41.9211	-87.8092	
7	Climbing Tree	41.8781	-87.6398	
8	At the Bottom of Miss	42.36115	-71.0571	

For now, we'll just mock this up with a single point. Next week, we'll work with data like this to put many points on a map (ideally using your data…)

Drawing points on the map

```
svg .selectAll ( 'circle' )
      .data (  [  { long: -71.0589,  lat: 42.3601}   ] )
      .enter( )
      .append ( 'circle' )
      .attr ( 'cx' ,  function ( d ) {
            return albersProjection (  [ d.long,  d.lat ]  ) [ 0 ]
      } )
      .attr ( 'cy' ,  function ( d ) {
          return albersProjection (  [ d.long,  d.lat ]  ) [ 1 ]
      } )
      .attr ( 'r' , 10 )
      .attr(  'fill' , 'purple' )
```

Chloropleth: Graphing a variable using state fill color

First, we need to load the data that we want to plot on the map (stored in a separate .csv file)

Then, we need to figure out how to connect it to the map data

Finally, we need to use the .csv data when we draw the map.

Loading multiple data files at once

Queue.js is a library that helps us to load more than one data file into the browser at once.

Add it to the HTML:

```
<script src =" ./queue.min.js" ></script>
```

From JS, the following replaces d3.json:

```
queue ( )
   .defer ( d3 .json , " . / cb_2016_us_state_20m .json "  )
   .defer ( d3.csv , " . / statePop .csv" )
   .await ( function ( err , mapData, populationData ) {

   //put your map drawing code here, update to .data( mapData )

}
```

Connecting map and .csv data

To connect the data, we want to make a "lookup table", or dictionary, that we can use to look up the data value for a particular state when we draw the map.

Unfortunately, the function that we use to do this is called array .map( ), which is confusing because it has nothing to do with geographic maps.

First, we make an empty dictionary at the top of the document:

```
var stateLookup = d3 .map ( ) ;
```

Then, we connect it to the .csv data by setting the dictionary values:

```
populationData .forEach ( function ( d ) {
      stateLookup .set ( d.name, d.population ) ;
});
```

Using an array.map( ) function to look up values

Now that we have our dictionary set up, we want to be able to use it to set the colors for different states in our map.

First, set up a color scale for the data:

```
var colorScale = d3 .scaleLinear ( ) .range ( [ 'white' , 'blue' ] );

colorScale .domain (
    [ 0, d3.max(
            populationData .map( function ( d ) {
                    return +d.population
            } ) )
     ] );
```

And apply it to the map drawing:

```
.attr ( 'fill' , function ( d ) {
        return  colorScale ( stateLookup .get ( d .properties .NAME ) );
})
```