Jesse Fried

Thematic Cartography

April 22, 2021

**Javascript Libraries for Web Mapping:**

**Cartographic Experiments and Review**

Maps are ubiquitous on the internet. From navigational tools like Google Maps to infographics produced by news outlets like the New York Times, the art and science of creating web applications with maps as interfaces has flourished and impacts our daily lives. A wide range of tools are available for bringing geospatial data to life on the internet. Some require little to no training to display, edit and share map features, such as ArcGIS StoryMaps and Google My Maps. These are limited in terms of functionality and customization. On the other hand, professionals use some very abstract tools to create custom infographics and other sophisticated applications from scratch, requiring specialized training. In the middle are numerous libraries catering to the growing populations of web developers interested in cartography and cartographers with some web development background.

To the student of Thematic Cartography interested in web mapping, it is important to assess the available tools not only in terms of their usefulness for producing a map, but also for producing a well-designed map. There is often a tradeoff between ease of use and flexibility: how much the library handles “under the hood” versus what level of customization it makes possible without getting in the way.

Javascript is “the programming language of the Web,” running in the browser on every website but the most basic. We can safely say that most if not all interactive maps found on the internet rely on it to some degree. A number of Javascript libraries make it practical for the aspiring cartographer with an intermediate level of familiarity with the language (like myself) to readily produce interactive maps from raw geospatial data. I investigated several such libraries, with a focus on cartographic design, appropriate map elements, and simple, logical code. To assess the capacity of each to produce a well-designed map, I attempted to replicate as closely as possible the proportional symbol map of Florida’s counties that won our class’s map design competition using several libraries, only adding very basic interactivity. Within reasonable time constraints for this project, I had varying degrees of success. Some substantial differences emerged.

Four JS libraries stand out from provisional research: Leaflet, OpenLayers, D3 and the ArcGIS Javascript API. The API documentation for all four is referenced frequently below, too many times for individual citations. **Leaflet** a popular and widely used tool for adding an interactive, zoomable map to a webpage. It is admired for being “lightweight” and simple, containing “all the features most developers need” but not addressing more advanced uses (Stackshare). It is relatively easy to get a basic map up and running. Because it is so popular, it is easy to look up help in online forums. My experiment with Leaflet was largely successful, and informed by some previous experience with it.

**OpenLayers** is often compared to Leaflet. Similar in purpose and overall design, it is considered more powerful but heftier and harder to learn, more suitable “for complex GIS application development” (Geoapify 2019). It is far less popular than Leaflet but highly regarded among a more specialized community. There is a lively Leaflet versus OpenLayers debate online. OpenLayers was new to me, but I was also successful with it.

**D3** (Data Driven Documents) is a data visualization library with broad applications, geographic and otherwise. It is the professional standard for custom infographics, and the subject of many how-to books. It includes tools for rendering geographic information, but not much built-in functionality related to maps. Many highly skilled developers use it to produce incredibly sophisticated geovisualizations that would be clumsy or impractical with Leaflet or OpenLayers, especially unique animations (Cook 2021). As I found out the hard way, it is hard to do much beyond a very simple map without a solid grounding in general D3 concepts. My efforts were only partly successful (see Figure 1).

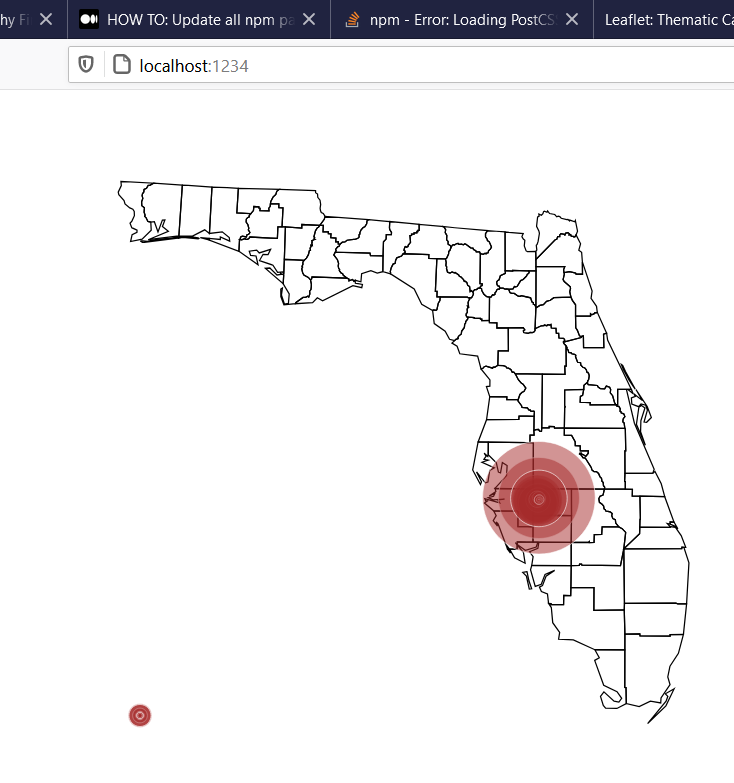


Figure 1: As far as I got with D3

The **ArcGIS Javascript API** is organized along the lines of Leaflet or OpenLayers, but it is more feature rich. It makes it easy to create a web map and apply some of the same powerful data visualization features one would find in ArcGIS Pro, but it seems less thoroughly customizable, trading design flexibility for ease of getting powerful maps up and running, and an ESRI look and feel. It is optimized to use data from ArcGIS Online, and integrate with various ESRI products. Using its full features requires an ArcGIS Developer subscription (ESRI 2021). I did not have time to produce a map using the ArcGIS Javascript API, and my knowledge of it comes from reading the documentation rather than getting my hands dirty.

**Data Format**

GeoJSON is a standard format for encoding vector map features for use on the web, specifying how to include the feature’s geometry, other attributes, and, optionally, the coordinate reference system for a group of features. It is a subset of JSON (Javascript Object Notation), a data-interchange format widely used on the internet (Butler et al 2016). All of the JS libraries used in this investigation can easily interpret GeoJSON. I used a geoprocessing tool in ArcGIS Pro to export the shapefile of Florida counties and demographic information from our Thematic Map Design lab into GeoJSON format. I hosted this file on the internet for my applications to use.

**Map Frame**

Mapping in Leaflet, OpenLayers and ArcGIS starts by initializing a map inside a specified element on a web page. A base map is set at this stage, with relative ease in all three libraries: Leaflet requires one, but it is optional in OpenLayers and ArcGIS. Basemap attribution is provided in the lower right hand corner by default, but can be moved or removed. A coordinate pair specifying the map’s center and a map scale also must be set; in Leaflet directly on new map, but somewhat separately in OpenLayers and ArcGIS in a “view” added to the map. Leaflet can only set the map’s scale using a somewhat arbitrary whole number, where 1 is the whole world and 6-8 generally fits a US state. OpenLayers lets the scale be set more precisely in terms of resolution, or “projection units per pixel.” Only in ArcGIS can a conventional ratio scale be used. All three libraries have built-in animated zooming and dragging.

In D3, there is no “map” other than the features themselves displayed on the page. Any zooming or dragging interaction would need to be custom-built. Adding a basemap could be complicated

In all cases, styles outside the map area itself, such as borders, need to be custom added.

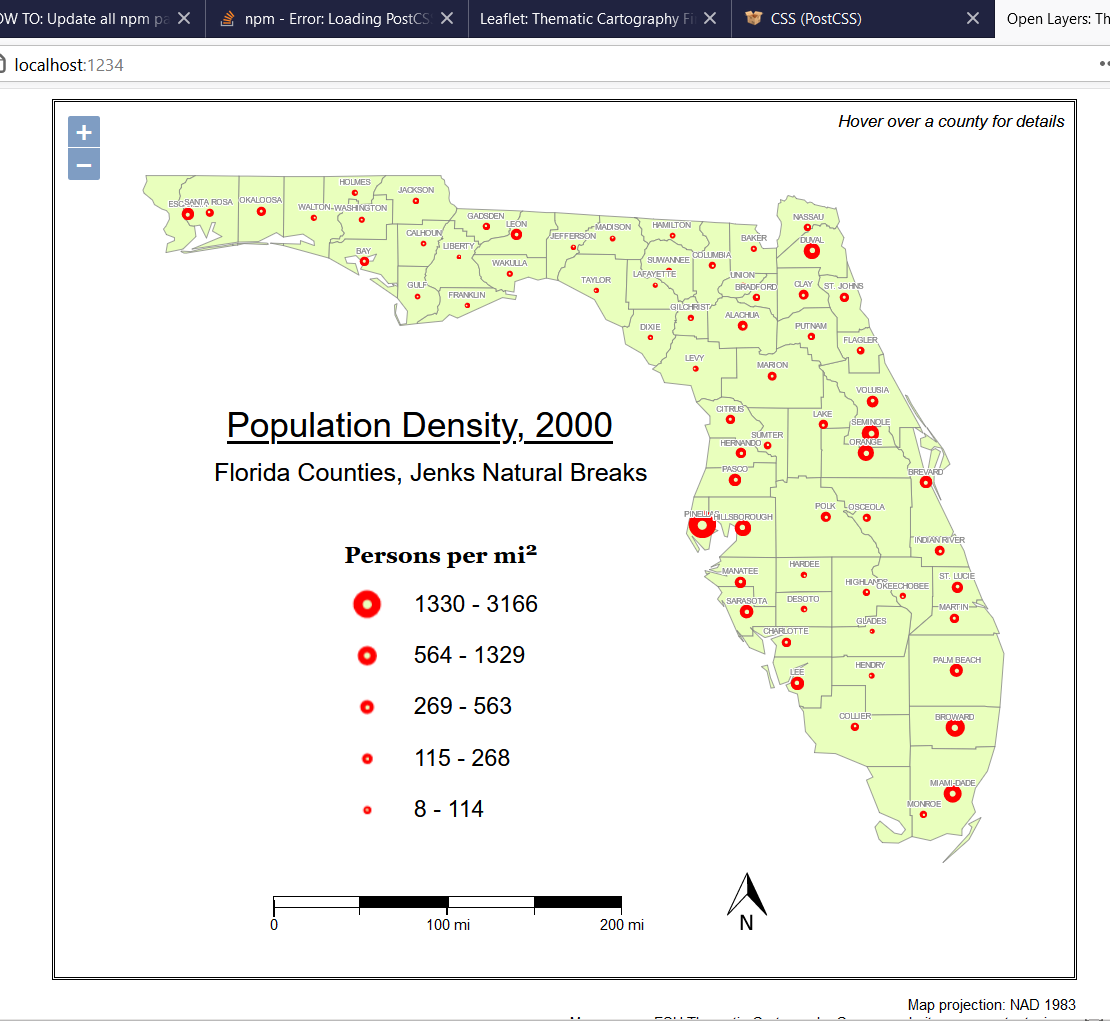


Figure 2: Screenshot of successful OpenLayers map

**Map Features**

Translating map feature geometry into a format that can be rendered on a web page is the core functionality of these libraries, but they each handle the task slightly differently. Some render the features as SVGs (scalable vector graphics) and others draw them inside an HTML canvas element. SVGs are normal parts of a web page just like text, images, links, buttons, etc. Customizing them with styles and interactions can be done the same way as any other element on the web page, making for for well-organized code and ease of use. The canvas element is more of a “black box” on a web page, inside of which everything is drawn using Javascript. It performs better for more complicated visualizations interactions, such as immersive games, but it is harder to work with, and the code can become messy. SVG is a true vector format, while the canvas is rasterized. SVG is solidly recommended for simpler use cases (Coyier 2019). For a basic, mostly static thematic map like in this project, I think the clear choice is SVG.

Leaflet renders vector map features using SVGs by default, with an option to use the canvas. OpenLayers uses the canvas, and while it can project an existing SVG map layer produced elsewhere, it cannot render them itself. ArcGIS uses canvas exclusively. D3 has functionality for both but using SVGs seems to be more common. OpenLayers and ArcGIS are built with more complicated geovisualizations in mind, so it makes sense for them to use the canvas, but for this project it has no advantage. In Leaflet, I was able to set the styles for my map’s features and symbols in the website’s stylesheet, seamlessly with the styles for the rest of the web page, using only a couple of lines of CSS code. In OpenLayers, I had to define the styles in the Javascript code using a less concise and less familiar syntax. Rendering features as SVGs makes it easier to keep the website’s style information together in one place, and separate from the code that handles the functionality, an example of the software development best practice known as the “separation of concerns” (Schults 2020).

**Spatial Reference**

Each library handles the issue of coordinate systems and projections differently. Leaflet, OpenLayers, and ArcGIS use the Web Mercator projection, based on the WGS1984 datum, by default. However, the Florida counties features in the GeoJSON file for this project are in the NAD1983 datum, EPSG:4269. Leaflet does not support very many other projections, without advanced customization, but somehow was able to transform the features into Web Mercator on the fly with no additional setup. OpenLayers supports other projections, and can even reproject raster tiles such as basemaps, so with some configuration it easily displayed the features in their original EPSG:4269 coordinate system. Transforming to any other coordinate system would be just as simple. ArcGIS also has advanced support for spatial reference and transformations.

D3 offers a choice of 30 of the most common projections, plus more available in plugins. Unlike the other libraries, it does not project features within a predefined map frame, but can render them anywhere on the page. Within a reasonable time constraint, I was not able to fully master D3’s relatively advanced, flexible system of projecting map features. I was able to use the built-in geoTransverseMercator() projection, with an appropriate rotation and scale that I found online (Veltman 2018), to approximate the Florida North State Plane NAD1983 projection. I got as far as rendering SVGs representing the outlines of Florida County outlines but could not completely control their location. I also found that the SVG shapes were drawn in such a way that it was difficult to style them. Researching and addressing these problems proved time prohibitive.

**Proportional Symbols**

The only library with built-in symbology functionality is the overwhelmingly feature-rich ArcGIS Javascript API. The others require some customization to create the symbols. In Leaflet and OpenLayers it is very light and easy to do. In Leaflet, after the features themselves are rendered, the library makes it easy to iterate through the list of features again, calculate the approximate center of each one, render a circular marker at each center point, set the radius of each circle, and add other style information. The calculation and scaling of the radius has to be done manually. After some research and trial and error, I used the below formula to calculate an appropriate range of radii in pixels for the proportional symbols:

radius: Math.sqrt(popDensity/Math.PI)\*0.3+1.5,

Pop density was previously calculated from the feature’s population and area attributes. The tan inner circle can be subsequently added in the same way, with a different scale factor. OpenLayers has equivalent features that make the process just as easy, but the structure of the code is a bit more logical. It is possible to add multiple “Styles,” for each feature at the same time, so in this case the county polygons and the population symbols are all handled in the same place in the code. D3 contains powerful tools for creating and manipulating vector shapes, such as circles, but no easy ways of positioning them on the map like the other libraries. I was able to create and style the proportionally sizes circles, but I encountered issues positioning them.

**Legend**

A drawback of Leaflet is that, to my knowledge, it contains no functionality for making a legend based on the map layers. I built a legend from scratch, using SVG circles, but this was time consuming, static, and involved manually calculating the sizes in pixels of the legend symbols. OpenLayers has an extension, part of the easy-to-use ol-ext package, that creates and positions a legend, with some built-in styling options, and a way of adding custom styles. I was able to reproduce the mapping contest winner’s legend exactly with a minimum of time and effort and define the symbols sizes in terms of population density, rather than raw pixels. ArcGIS has a built-in feature for building a legend and including all kinds of controls in it, but it appears rather prefabricated, with less design flexibility. Lacking any out-of-the-box map elements, D3 requires manually constructing a legend, but this library’s deft manipulation of SVGs would make the process easier, especially for an experienced user.

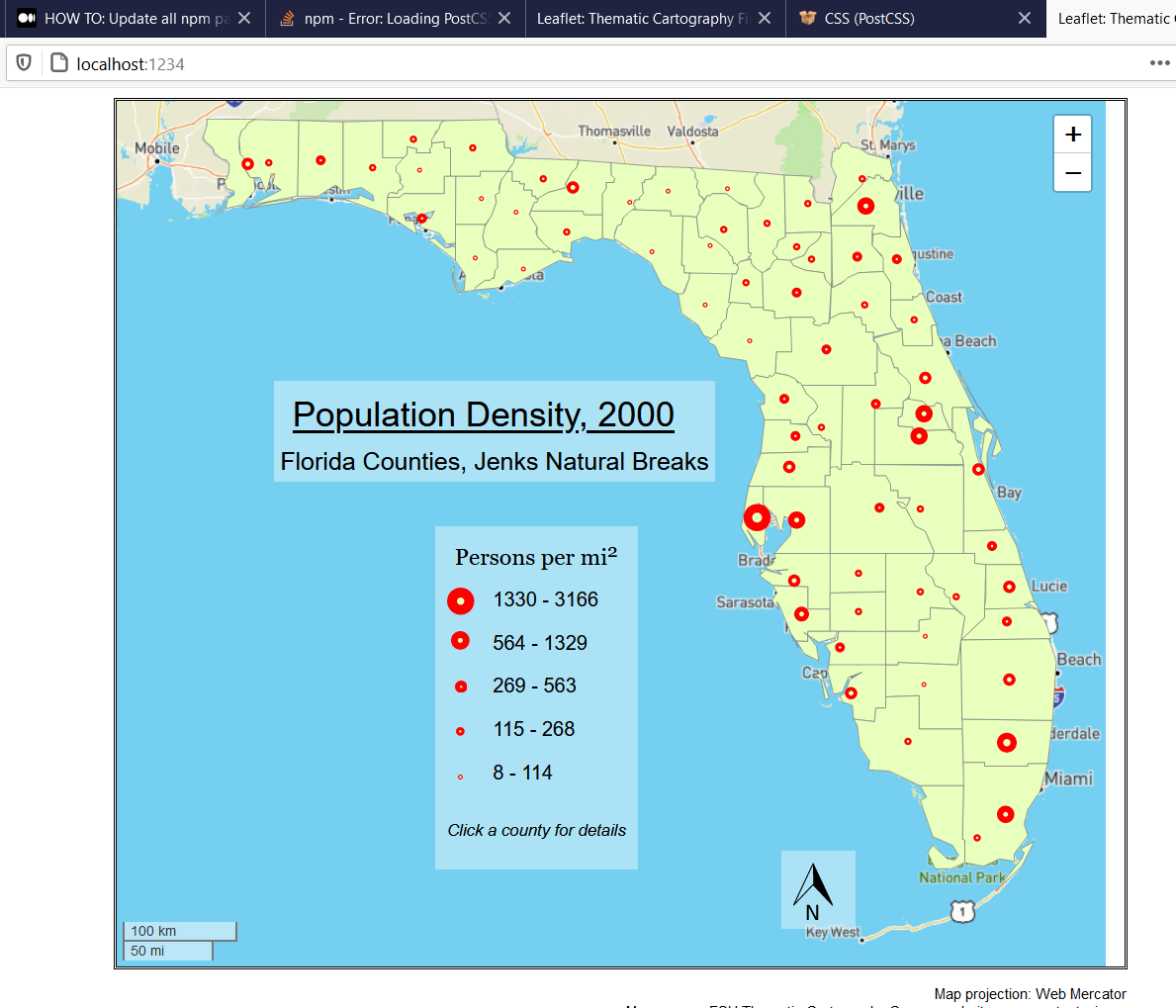


Figure 3: Screenshot of successful Leaflet map

**Scalebar**

Leaflet, OpenLayers and ArcGIS all come with dynamic scalebars that respond as the map is zoomed in or out. In Leaflet, it is easy to add a scalebar in the lower left-hand corner, showing both miles and kilometers, but its position, size and style cannot be customized in any way. The OpenLayers scalebar’s number of steps, units and width, can be set. Custom CSS styles can be added to set its position, but changing its appearance involves clumsy overrides of defaults. ArcGIS has surprisingly few options for their scalebar: metric/non-metric units and ruler/line styles. In D3, the scalebar would need to be made from scratch.

**Feature Labels**

In Leaflet, it is easy to add feature labels in a pre-designed popup or tooltip that appears on click or hover over the feature, but surprisingly harder to permanently position them on top of features on the map. That seemingly basic feature is only available in a little-used plugin that I unfortunately did not have time to experiment with (). This prioritization makes sense, given that space for text on a map is almost always limited, and it is so easy to interact with a web map using the mouse. However, it is a drawback for faithfully reproducing the contest winner map in web map form. A further drawback is that the prefabricated popups and tooltips are not well suited to custom styles or complex, dynamic content. OpenLayers, with its larger set of features, makes it easy to add and position text on top of features, or any “Style” (in its terminology) to the map. There is no elaborate label placement, but X and Y offset, font style and stroke, the equivalent of a halo, can be set. OpenLayers also has an overlay, or an “element to be displayed over the map and attached to a single map location” that can be used for popups (OpenLayers), but is more flexible than Leaflet’s. ArcGIS offers a comprehensive set of features around labels, including intelligent placement and display filters, not to mention functionality to create dynamic popups enabling rich interactions with map data. D3 would require complete customization for any sort of feature label.

**North Arrow**

Strangely, none of the libraries offer a north arrow, even though they all enable rotation of the map view, not even ArcGIS as far as I can tell. It is easy to make a custom north arrow using an image with a transparent background, but this method would become complicated with any degree of rotation.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Basemap** | **Vector Features** | **Wide support for coordinate systems, projections** | **Symbology** | **Legend** | **Scalebar** | **Feature labels** | **North Arrow** |
| **Leaflet** | Required | SVG/canvas | No | Custom: easy | No | Yes (basic) | Plugin | No |
| **Open Layers** | Optional | canvas | Yes | Custom: easy | Plugin | Yes (more flexible) | Yes | No |
| **ArcGIS Javascript API** | Optional | canvas | Yes | Built-in | Yes | Yes (basic) | Yes | No |
| **D3** | None | SVG/canvas | Yes | Custom: advanced | No | No | No | No |

**Conclusion**

The real power of web maps is outside what I attempted in this project: interactions that visualize data in ways impossible for static, printed maps. A slider that lets the user go back and forth in time is a basic example. The purpose of replicating a static map on the web, however, is so that we do not get intoxicated by the new power and complexity available to us and leave behind leave behind longstanding principles of cartographic design, and the freedom to lay out maps as we see fit rather than as software dictates.

This investigation reveals that not all the available options cover all the basics equally well. Leaflet is the fastest to get up and running for a simple map, but there are distinct tradeoffs in design options, in terms of coordinate systems and projections, zoom level, scalebars and labeling. Working around these constraints is possible, but if they arise, it may be worthwhile to use some of the other options, like OpenLayers. The only way to avoid all constraints is to use a library as flexible as D3, but my severe struggles with D3 in contrast with my success with other platforms make it clear that a greater level of expert knowledge is required. The ArcGIS Javascript API offers a shortcut to an advanced level of data visualization that many users have come to expect at this point in the history of the internet, but it comes at the cost of design freedom. Its integration with other ESRI products comes with the tradeoff that it is not open source. For the specific task of reproducing the map contest winner, climbing the mountain of D3 knowledge would be the most exact solution, but within a reasonable time and skills constraint, OpenLayers has the best results.

**Works Cited**

Bobstock, M., Riviere, P., Macy, A., van Rossum, M., Armstrong, Z., Pan, W., Persing, M., Lenahan, P., Lange, D., Silva, F., Glassman, E., (15 April 2021). *D3 API Reference: Projections*. Github. https://github.com/d3/d3/blob/master/API.md#projections

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., Schaub, T., (August 2016). *The GeoJSON Format.* Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc7946

Cook, Peter. (2021). *Geographic*. D3 in Depth. https://www.d3indepth.com/geographic/

Coyier, Chris. (2019). *When to Use SVG vs. When to Use Canvas.* CSS-Tricks. https://css-tricks.com/when-to-use-svg-vs-when-to-use-canvas/

ESRI. (19 April 2021). *ArcGIS API for JavaScript*. ArcGIS Developers. https://developers.arcgis.com/javascript/latest/

Geoapify. (12 April, 2019). *Leaflet vs OpenLayers. What to choose?* https://www.geoapify.com/leaflet-vs-openlayers

Leaflet. *Documentation - Leaflet - a JavaScript library for interactive maps*. (n.d.). https://leafletjs.com/reference-1.7.1.html.

OpenLayers. *API Documentation*. (n.d.). https://openlayers.org/en/latest/apidoc/

Schults, Carlos. (2020). *Separation of Concerns, Explained*. Ndepend. https://blog.ndepend.com/separation-of-concerns-explained/

Stackshare. *Leaflet vs OpenLayers* (n.d.) https://stackshare.io/stackups/leaflet-vs-openlayers

Veltman, N. (13 March 2018). *d3-stateplane*. Github. https://github.com/veltman/d3-stateplane

Viglino, J. (13 March 2021). *ol-ext (Extensions for OpenLayers)*. Github. https://github.com/Viglino/ol-ext