Beautiful thematic maps in Leaflet with automatic data classification

Dániel Balla [1], Mátyás Gede [1]

[1] Institute of Cartography and Geoinformatics, ELTE Eötvös Loránd University, Budapest, Hungary – (balla.daniel,saman)@inf.elte.hu

Keywords: thematic maps, data classification, symbology, Leaflet, web cartography, open source.

Abstract

Even though the web is a platform that provides lots of features, interactivity, and a high degree of customizability for creating web maps, web-based thematic maps still require expertise to visualize geospatial data in a way that highlights spatial differences and is cartographically comprehensive. The popular open-source web mapping library Leaflet lacks a straightforward approach to create thematic maps with basic principles they should adhere to (data classification, automatic symbology and legend generation). Although various tutorials and workarounds exist, those are hard-coded, only solve some principles thematic maps require and are complex to accomplish, requiring programming experience. The paper focuses on finding a way to overcome shortcomings of Leaflet in terms of thematic maps, that supports a simple, self-explanatory method of producing thematic web maps using the library. As a solution, this paper introduces an easy-to-use, open-source plugin for Leaflet, developed by the authors, which combines all processes required for creating a thematic map from a GeoJSON dataset, in one single step. For symbology, it supports graduated symbol colours and sizes, and colour and hatch pattern fills for polygons. It supports well-known classification methods for quantitative data and puts emphasis on providing numerous options for all underlying processes, to fine-tune the visualization (data normalization, rounding class boundary values, legend templating etc.). This highly customizable extension intends to help people who do not have experience with map design and are not familiar with scripting to an extent to be able to code visually pleasing thematic maps for websites.

1. Introduction

To visualize quantitative data, web-based thematic maps are a popular and great tool to aid a comprehensive data visualization, while, due to its web-based nature, offering interactivity as well. Thematic mapping emphasizes communication between the map maker to the map user via information abstraction (Linfang & Liqiu, 2014). Since choosing the most optimal way to represent data is essential in the field of thematic cartography, one of the most important tasks of an author of a thematic map is to create a product that is legible (for the type of intended target audience) and unambiguous, while still pleasant to look at. To ensure the map is not visually overloaded, besides processes like generalization, the selection of appropriate symbology with altering appropriate visual variables is of utmost importance to display both quantitative and qualitative data. Based on the summary on visual variables by (Roth, 2017), (Bertin, 1967/1863) originally identified seven visual variables (location, size, shape, orientation, colour hue, colour value and texture), which was later extended with colour saturation and arrangement by (Morrison, 1974) and crispness, resolution and transparency by (MacEachren, 1995) for an overall of 12 variables.

Strongly related to thematic mapping is data classification, which precedes any kind of visualization on the map. During classification, map features are classified into groups based on one or more of their quantitative attributes. Proven and popular statistical classification methods include Natural breaks (Jenks), Equal count (Quantile), Equal intervals, Standard deviation etc. The individual methods all both have advantages and disadvantages, when used with different spatial data types to minimize information loss (Osaragi, 2002). Choosing an optimal method and an optimal class count massively helps the map user to quickly comprehend thematic data and discover relevant spatial differences. For example, classifying data into two classes does not convey much useful information, while using 10 class bins might make the map incomprehensible by having very similar symbols which can no longer be easily distinguished. Previous research (Miller, 1956; Mersey, 1990; Cromley, 1995) suggested using five to seven classes for static maps. Most thematic maps show data with seven or less classes (Linfang & Liqiu, 2014). In order to distinguish individual classes on a graduated symbols map, an exact symbol is assigned to each, which are created with altering one or more of the visual variables to distinguish classes. Besides the more complex techniques of symbology, for point features their symbol fill colour, shape and symbol size can indicate both qualitative and quantitative data, line features have stroke colour, type and width which we can operate with, while polygon features usually have their fill colour or type (e.g., hatching) altered to indicate a given class. The use of colours and colour ramps also hugely influences the map user's perception of the product. By adhering to colour theory, using colours that create a proper visual hierarchy we emphasize the thematic overlay(s) over the background map. Based on data type, we differentiate three major types of colour ramps: sequential, diverging, and qualitative schemes (Brewer et al. 2003). Another vital part of a thematic map is a descriptive legend, which contains all the distinct classes the map features were classified in, with exact symbols for each class. Given we work with quantitative data, the individual class interval boundary values are usually rounded to a given decimal or a whole number to ease and quicken comprehension for the map user.

All the available visual variables and methods offer a wide range of possibilities for the map maker to achieve a symbology that conforms to the basic principles of thematic cartography. By conducting three studies on web mapping technologies, comparing features, user needs and experiences, (Roth et al., 2014) identified a subset of available technologies for teaching web mapping, while revealing insights into web map design. Their subset comprised of Google Maps API, OpenLayers, Leaflet, and D3. Out of the three open technologies (OpenLayers, Leaflet, D3), they concluded Leaflet as the one with the most representation and interaction requirements met by participants (60.4%). Their study also investigated the

participants' emotional experience with the technologies, in which Leaflet scored the highest (most positive experience). Based on the work of (Farkas, 2017), OpenLayers and Leaflet are two of the most feature-rich, open-source web mapping libraries. In a web environment, the interactivity and customizability offered by the likes of these further enhance the creation of a proper symbology. However, while creating rich thematic maps with both Leaflet and OpenLayers is possible, neither features a straightforward way to combine all basic principles of thematic mapping (data classification, appropriate symbology and legend creation) natively. Due to support for touch-based interaction and small library size, Leaflet is considered one of the best libraries for web mapping when designing for a mobile environment (Roth et al., 2014). Scientific literature on using Leaflet to create interactive thematic maps with built-in client-side data classification is sparse, and mainly consist of qualitative classification, as in (Horbiński and Smaczyński, 2023) and (Horbiński and Lorek, 2020), instead of quantitative.

This paper introduces a newly developed, client-side Leaflet plugin to make the creation of GeoJSON-based thematic maps much easier. The extension intends to help people who do not have experience with map design and are not familiar with programming and scripting to an extent to be able to code visually pleasing thematic maps on websites. It also aims to provide some features related to thematic mapping that are present in desktop GIS environments, like data normalization, handling of null values, class boundary value rounding and easy multiplication/division of values for changing unit of measurement.

2. Features and Shortcomings of Leaflet

Why Leaflet? Among the three open-source technologies mentioned in Introduction, Leaflet requires the least amount of programming experience to get started with. While getting started with it on a basic level is easier, creating refined and elegant thematic maps is still a challenge in all three. D3, while being a feature packed library for interactive data visualization, since its focus is not on maps, it requires workarounds to have basic features like map panning and zooming implemented. Although OpenLayers does have a rich feature set, its steeper learning curve makes its usage harder for non-expert users.

Leaflet (https://leafletjs.com/) is an open-source, client-sided JavaScript library for web mapping, which, due to its lightweight, interactive and customizable nature, is increasingly popular. Developed and maintained by Volodymyr Agafonkin, Leaflet makes use of the possibilities of HTML5 and CSS3 standards, with a designated goal of being a simple, performant, and mobile-friendly library. A lot of useful third-party plugins are available for free, which extend Leaflet's functionality. This allows developers to customize their specific use of Leaflet in a web environment. Although it is relatively easy-to-use and it renders spatial data client-side well, it is unable to do geoprocessing internally – geoprocessing algorithms are well-covered by another JS library, Turf.js (https://turfjs.org/), which can easily be integrated with Leaflet. Despite that, Leaflet offers a substantial amount of display and styling features. The visualization of spatial vector primitives like points, lines and polygons are well-handled while making use of the SVG (Scalable Vector Graphics) specification for client rendering. For symbology, a lot of styling parameters are available (colour fill, outline, line width, dashed lines, opacity, symbol size etc.) (Agafonkin, 2023). The official website of Leaflet does provide some basic examples and tutorials for symbolizing features and creating a legend, based on which a specific thematic map could be produced. A tutorial exists for an interactive choropleth map, provided by Leaflet. However, it is hard-coded, meaning that it will have to be recreated for each specific thematic map tailored to a specific data set, making them unsuitable for implementation in a dynamic data visualization. This way, they also leave data classification and symbology design completely up to the map maker/developer to figure out. The study of (Roth et al., 2014) indicates that, when faced with the task of a more complex web mapping scenario, users new to Leaflet might still find it confusing. It also highlights a participant, who noted it's difficult to figure out what they need to read first and what is most important, in order to match the requirements of the scenario. This phenomenon might be explained by the lack of variety and depth in the tutorials on Leaflet's official homepage.

Creating thematic maps does represent one of the more complicated tasks in Leaflet. To fulfil the expectations and follow the basic principles of thematic maps (data classification, appropriate symbology, and a clear legend with the individual symbols), Leaflet lacks some important features. Although developing these features can be realized one-by-one, they are quite complex and static, and require some experience with JavaScript coding to get the intended result.

2.1 Data Classification

For data classification, Leaflet does not do any classification of data natively. This is partly understandable, since Leaflet's main goal is to provide a visualization, not GIS or data manipulation functions. Any classification we want to perform on the data set has to be done beforehand with external software. In terms of external tools and helpers, ClassyBrew is noteworthy. ClassyBrew, as an open-source utility to generate class breaks and applying "colorbrewer" theory, currently only supports three methods for data classification (equal interval, quantile, Jenks natural breaks) (Tanner, 2023). It does not generate an appropriate legend for the created visualization. Since it only operates with colours (applies one of the existing colour palettes to features and class symbols, which could be useful for choropleth maps), it does not support more advanced but classic thematic visualization methods like symbol size-, line width- or hatch pattern fill-based symbology. Considering that it is an independent tool, it has no tight integration with Leaflet classes: data values must be manually parsed from GeoJSON and given to the utility as an array, from which the utility generates colours that can be used in a Leaflet "styler" function. The utility is no longer maintained, as the last commit happened on 20 June 2017 by its developer.

2.2 Symbology

When loading spatial vector data into Leaflet (e.g., GeoJSON), amongst other features, styling of the individual features can be overridden. This way, it is possible to define a "styler" function, which receives a given attribute value as an argument, processes the value using a stacked conditional statement with a pre-defined data classification, and returns the correct style attributes (e.g., fill colour) for the given class. That style is then applied to the feature as the feature's graphical representation. Even though this workflow allows for a classification based on pre-defined class boundaries, the process of data classification itself must be done manually beforehand (i.e. in GIS software). After classifying manually, the map maker has to code the styler function as well, including the exact class boundary values and colour codes to get the desired results. This workaround is complicated and is immensely prone to human error.

As for specific styling features, using a hatch fill is a common technique in traditional thematic cartography. Most server-side mapping software already support hatch fills for a long time, however, client-side mapping libraries do not natively support pattern fills (Gede, 2022). Even though it is making good use of modern web standards, Leaflet does not support hatch fills for styling features (e.g. polygon fills) natively.

2.3 Legend

Besides an appropriate symbology, a thematic map missing a legend to explain visualized data is rather pointless. Without a legend, visualized data cannot be understood by the map user. A clean, visually organized and precise legend further increases the chance of the map reader's proper comprehension. Leaflet, in addition to the default UI elements ("controls"), has an interface that supports the creation of custom map controls and elements. Making use of this feature, creation of a legend is feasible using HTML elements and CSS style definitions. The custom control must be designed from the ground-up, which requires knowledge of HTML/CSS, especially, when the goal is a clean and tidy legend. This way, once a legend is designed for a specific thematic map, it remains static in terms of data, and can only be used for that specific data set, without an easy means to modify later (reclassify, change symbology etc.).

As it has been reflected on, while being a feature-packed web map renderer, Leaflet is missing data classification and automatic legend creation natively. Pairing that with a cumbersome and static means of defining symbology results in a complicated way of creating thematic maps with Leaflet. To aid this, a new Leaflet plugin is introduced, which extends Leaflet's GeoJSON class to provide combined functionality to generate thematic maps easily, while complying with some of the basic but crucial principles of thematic cartography.

3. Leaflet Plugin for One-Step Thematic Map Creation: 'leaflet-dataclassification'

The objective of the research is to find a way to overcome the exact shortcomings presented in the previous section, that supports a simple, self-explanatory method of producing thematic web maps in Leaflet. As a solution, an extension is introduced with the aim of offering an easier, faster method to generate appealing thematic visualizations of quantitative data in Leaflet. As it extends the functionality of Leaflet's GeoJSON class, the plugin is easily integrated in a Leaflet-based web map by instantiating an L.dataClassification layer, the same way as one would without the plugin (L.geoJSON). Upon instantiation, the plugin takes care of data classification of a chosen quantitative attribute, styles the features appropriately and creates a clean, appealing legend based on its initial configuration options, to provide a seamless and comprehensive process. As the whole process happens during the instantiation of a GeoJSON layer, the plugin can easily be integrated in automatic processes for dynamic data visualization (to programmatically update data, customize visualization options etc.). Multiple instances of the plugin layers are handled well, with most standard L.geoJSON functions available (for example calling the layer's remove() method, besides removing the features from the map, also removes the linked legend instance of the layer). This also allows for more complex thematic maps using two or more layers, with different symbology on top of each other (for example a choropleth map with polygons as the bottom layer, with symbol size-based, classified point features on top, both layers with a matched legend). In this case, the use of Leaflet panes are leveraged, to set z-index of specific layers and control visual layer order.

As one of the goals is to ease the seemingly complex process of creating a thematic map using Leaflet, usage of the plugin is quite simple and straightforward. After including both the JavaScript and CSS files of the plugin and its dependencies (simple-statistics.js, chroma.js, leaflet-hatchclass), all the new options are available when using L.dataClassification. Only three options are required to be passed as an object upon the instantiation of the layer:

- `mode`: classification method,
- `classes`: desired number of classes,
- `field`: target attribute field to base quantitative data classification on.

Defaults have been defined to provide a decent initial result with only the three required options, but the features described in the following subsections are available as additional options as well, to further customize the visualization and maximize its potential.

The plugin and its full API documentation can be found on GitHub: https://github.com/balladaniel/leaflet-dataclassification.

3.1 Supported Data Classification Methods

As one of the required options, choosing a classification method to use is necessary. Currently, the plugin supports the following data classification methods: natural breaks (Jenks), quantile (equal count), equal interval, standard deviation, logarithmic scale and manual. In-code, classification functions are primarily handled by simple-statistics.js, a JavaScript library that implements statistical methods (Simple-statistics.js, 2023), extended by custom functions. Classification based on standard deviation emphasizes values below and above the mean in a normally distributed data set. It currently generates classes with a fixed interval size of a standard deviation, with plans to support 1/2, 1/3 as well. When using a manual classification, option `classes` expects an array of class boundary values, instead of an integer of class count. Although using a manual classification method partly defeats the purpose of the plugin, it is still implemented to not limit map makers and their data visualization in any way.

3.1.1 Data Manipulation Options: In desktop GIS environments, normalization in a favoured tool to adjust data attributes based on another attribute value, even before performing classification. This process allows for direct comparison of values in different scales or units. To normalize data, a normalization function has been implemented in the plugin to adjust an attributes in the loaded GeoJSON. For example, this feature comes handy when creating a population density map using a data set that does not have normalized population data, since the area-based normalization can happen upon visualization. Thus, the features can be directly compared, regardless of their respective size. The plugin also allows for some basic data manipulation, in order to create more comprehensible class boundaries by rounding or modifying their values. Generated class boundaries can be either rounded to n decimals, whole numbers, or rounded up/down to the nearest 10, 100, 1000, etc. values. The latter might come useful when working with large numbers, for example on a thematic map with raw population data, considering that it does not make much sense to classify and show population data to a precision of a single person.

3.2 Symbology Features

The plugin puts an emphasis on providing a lot of options for a highly customizable symbology for a concise graphical representation. The map maker has various tools to fine-tune the data visualization for each type of feature, increasing the chance of getting the intended message through. In the context of the plugin, all styling options generate discrete symbol classes.

All colour-related functions in the plugin are powered by chroma.js, a small, zero-dependency JS library for colour conversions, scales and colour manipulation. Chroma.js supports all the most common colour definitions (e.g., hexadecimal RGBA, individual RGBA values, colours parsed in a specific colour space (HSL, HSV, HSI, CIELAB, OKLAB, CMYK, LCH), named colours, etc.) and also has built-in Brewer colour ramps (Chroma.js, 2023). Brewer colour schemes define colours that ensure legibility of thematic maps on most output mediums (Brewer et al. 2003), while helping those, who have little or no experience with map design or data visualization (Brewer, 2003). With the plugin, the predefined colour ramps can easily be reversed with an option, which might come handy to correctly illustrate positive/negative phenomena, e.g. reversing 'RdYlGn'.
For point features, it supports styling based on symbol fill colour and symbol size (graduated symbol sizes) as a means of distinction between classes (Figure 1). Colour-based distinction can be done by using one of the predefined Brewer colour ramps (for use with sequential, diverging, and qualitative data) or by defining a custom colour ramp. For symbol size-based distinction, minimum and maximum symbol sizes can be defined, between which the plugin automatically generates symbols for the desired number of classes. Shape of point features' symbols can also be overridden with some basic, built-in SVG-based shapes like circle, square, triangle, diamond, etc. In this case, symbol shape is not manipulated as a visual variable (since it is only suitable for nominal, qualitative distinction), but rather as a static symbol to use with both colour- and size-based modes for point features.

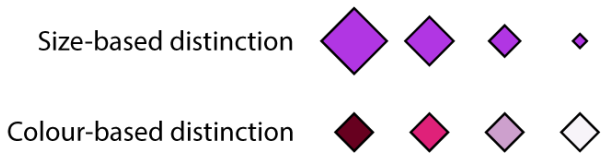

Figure 1. Means of distinction between four classes for point features, demonstrated with diamond-shaped symbols.

Line features can be symbolized based on line colour and width (graduated line widths) (Figure 2). Similarly to styling point features, line colour-based distinction uses predefined or custom colour ramps, and width-based distinction of lines generates widths between adjustable min-max values. Width-based distinction is particularly suitable for quantitative data on segmented line features, where there are data values for all individual segments, for example for traffic data, river discharge, river width (since a river has multiple sources, which usually accumulate resulting in increasing width) etc.

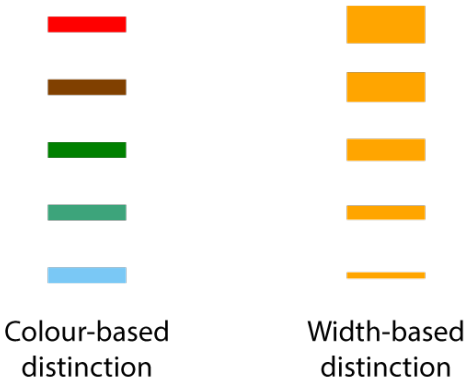

Figure 2. Means of distinction between five classes for line symbols.

Polygon features can currently be classified and distinguished by two fill types: colour or hatch pattern (Figure 3). Colour mode is similar to the other two feature types, using colour ramps. Hatch mode uses pattern fills for polygons, and offers distinction of classes by hatch line width, angle, or both (as seen in Figure 4). Hatch pattern can be customized by defining two stroke colours to alternate between, stroke widths to gradually alternate between (when the distinction base is either width or both). Initial hatch angle can also be defined, while setting a value to increment angle with, between hatch fill symbols is also possible (when the distinction base is angle or both). Integrated into the plugin, CSS hatch pattern classes are generated by plugin leaflet-hatchclass, developed by Mátyás Gede. As discussed in (Gede, 2022), compared to static maps, the existence of dynamic, zoomable web maps using hatch fills is limited, due to several visual and technical concerns. The leaflet-hatchclass plugin solves these problems by using a scale-independent hatch density in the form of SVG patterns. In case of hatch fills, the legend automatically widens the symbols to make sure hatch patterns are distinguishable. In case of a distinction based on alternating hatch pattern angle, the plugin warns if the symbols would become too similar (if the value to increment angles with between symbols was set around $\pi/2$, π, $3\pi/2$).

Figure 3. Means of distinction between four classes for polygons, with hatch fill alternating both angle and line width.

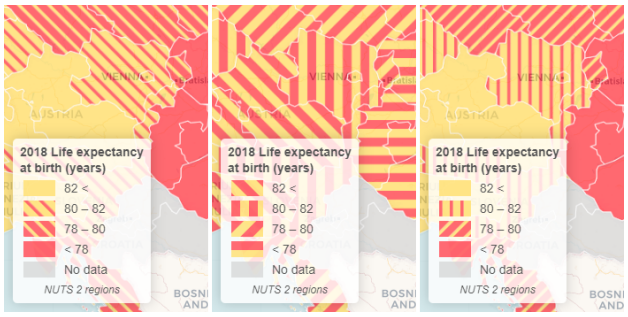

Figure 4. Hatch fill distinction modes (left to right): width, angle, both. Basemap: © OpenStreetMap, © CARTO. Data: Eurostat.

Although proportional symbols can be realized in Leaflet (Donohue, 2013), implementing this kind of symbols is not the primary goal of this plugin, since they are less relevant to data classification and delineation of exact class intervals with exact symbols.

The plugin is prepared to handle GeoJSON features with "nodata"/null attributes correctly. Nodata features get assigned a separate null value class, with appropriate distinction in the legend. By default, these features are assigned a neutral symbol (grey), depending on mode of distinction of the classes. The colour (fill/stroke) for these features can also be customized. Should the map maker wish to ignore nodata features, they can be ignored, therefore not showing up on the map, nor in the legend as a differentiated class.

In addition, all L.geoJSON/L.Path styling options can still be used for properties that are not handled by the plugin itself (e.g., when classifying line symbols with a line width distinction, it allows tweaking of every other styling property, except symbol width). Setting polygon outline to a neutral colour like white, for example, emphasizes the single- or multi-hue fill colour ramps, making the map much more legible and easier on the eyes.

3.3 Options for Legend Customization

The legend is constructed and displayed automatically as a semi-opaque floating panel, with exact symbols of the classes to truly reflect map data. A high level of customization has been provided for legends as well, with its non-essential parts being optional. One of the most important parts of a legend is a header, which usually briefly describes the visualized data for the map user, including a clear unit of measurement. An optional legend footer can be enabled for short data description (e.g. broader-than-header description, data acquisition time or method). Both the hidable header and optional footer allows for HTML-markdown and CSS-styling, should map maker wish to do so. Legend position inside the map object can be set with the usual L.Control positional options. In order to visually separate the distinct symbol classes, a row gap can be defined (which can also be further customized in the provided CSS file). Class order inside the legend is also subjective – the plugin allows for both descending and ascending sorting. An important feature is the possibility of templating legend class rows (Figure 5). A template can be defined for the highest, middle, lowest and nodata classes individually, using placeholders for high/low values and feature counts in the context of a given class interval. This templating allows for a very high level of customization, so the map maker can choose whether to display the class boundary values in a low-to-high or a high-to-low fashion, including any special characters around the values (e.g., "{low} – {high}" for middle classes, "{low}+" for the highest class, "< {high} [{count}]" for the lowest class, etc.). This approach also opens for internationalization, since this way, the map maker can format their legend to use words and prepositions of any language and order, if they wish to include any (e.g. "above 5000" in English, "5000 felett" in Hungarian).

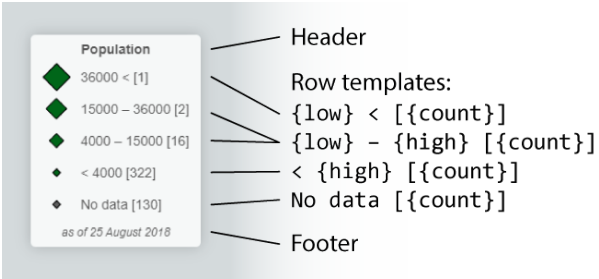

Figure 5. Legend structure with row templating.

Shown class boundary values can be easily modified by multiplying or dividing them by a number. This purely visual modification only affects the displayed legend and might come handy for quick unit conversions (e.g., raw data is in metres, but it makes more sense to display kilometres in the legend). This provides an optional tool to further fine-tune the visualization, make it more legible and appealing to the map user, and to facilitate quicker communication of spatial data.

4. Results and Discussion

In this section, various thematic maps generated with the plugin are presented as examples to show the capabilities of the extension. In addition, evaluation of processing time and overall performance will be presented.

4.1 Examples

As demonstration, various examples for all vector data types are provided on the GitHub project page. Each shows off different classification and symbology methods and plugin features, on separate data sets. Screenshots of some of the interactive examples are shown below.

Figure 6 shows an example use case of a multi-layered thematic map created using the plugin, where two layers of different vector types are present at the same time, all classified by the plugin, with their respective legends displayed. The map shows polygons distinguished by fill colour, and points distinguished by symbol size. The two legends use different templating, for demonstration purposes. The legend for capital population also shows the number of features belonging in each class.

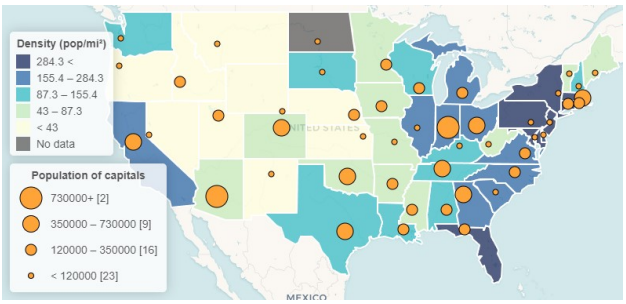

Figure 6. Multi-layered thematic map, with polygons as a choropleth base and point features, distinguished by symbol size. Basemaps: © OpenStreetMap, © CARTO.

In Figure 7, point features were classified into five classes, distinguished by symbol colour. The legend also has a footer for data description.

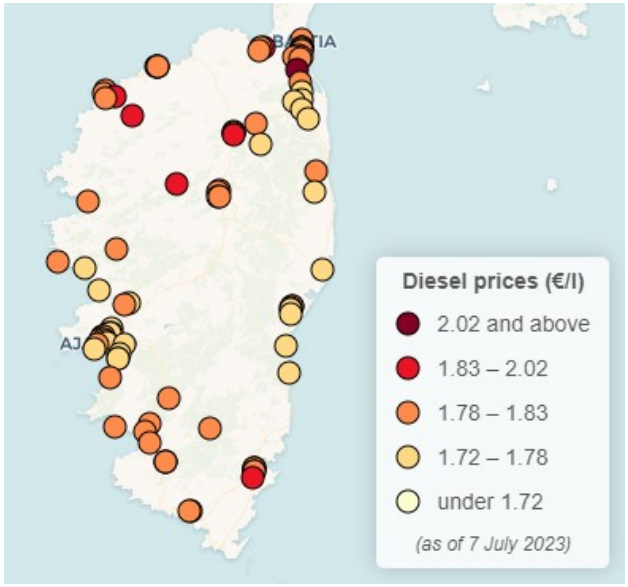

Figure 7. Point features, distinguished by symbol colour. Basemap: © OpenStreetMap, © CARTO. Data: Ministère de l'Economie, de l'Industrie et du Numérique.

Figure 8 shows the classification of line features of two separate data sets. The top map visualizes annual average daily traffic around Seattle, making use of line colour-based distinction. The bottom image shows the mean annual discharge of Danube and its tributaries, with the legend reflecting change in line symbol width between classes.

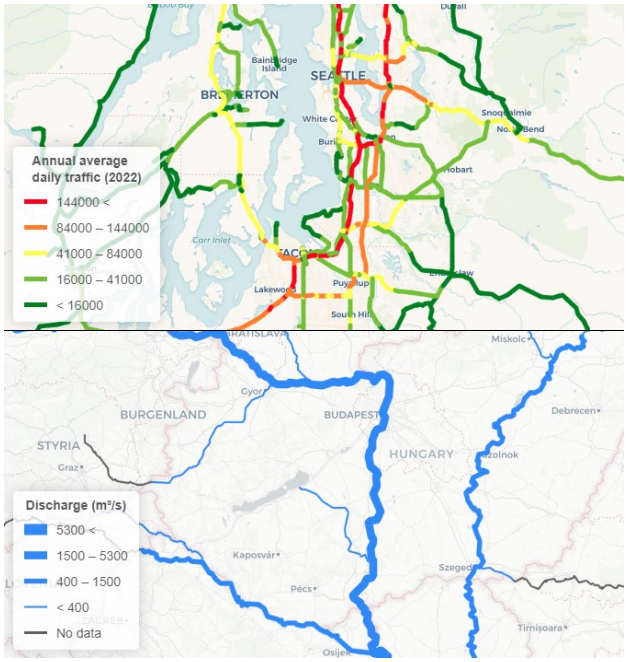

Figure 8. Line features, distinguished by line colour (top) and line stroke width (bottom). Basemaps: © OpenStreetMap, © CARTO. Data: Washington State Department of Transportation, International Commission for the Protection of the Danube River.

In Figure 9, 3220 polygons are classified into four classes and symbolized using polygon fill colours. A null value class is also present, although only a few, rather small polygons belong to this category.

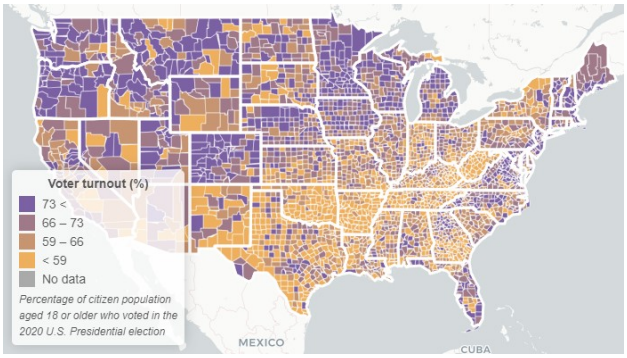

Figure 9. Polygon features, distinguished by fill colour. Basemap: © OpenStreetMap, © CARTO. Data: University of Wisconsin Population Health Institute.

Figure 10 illustrates the use of hatch fills for polygons, provided by the leaflet-hatchclass plugin. The example uses hatch angle for distinction between features.

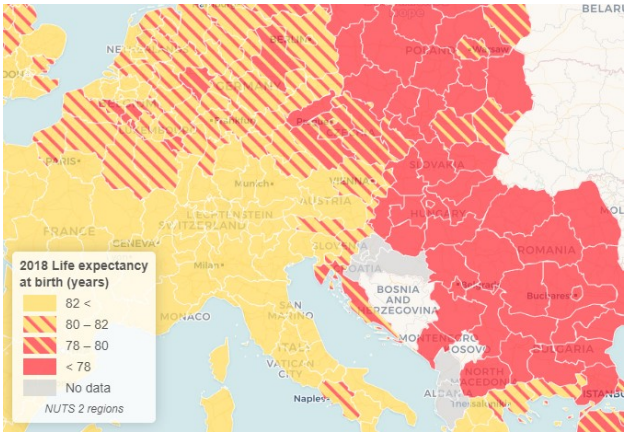

Figure 10. Polygon features, distinguished by hatch pattern fill. Basemap: © OpenStreetMap, © CARTO. Data: Eurostat.

4.2 Performance

Processing time and overall performance was measured on a desktop computer with an Intel i7-6700K CPU and 16GB RAM, using a V8 JavaScript engine (Chrome 124) on Windows 10 64-bit. Overall processing and rendering time of the layer was calculated as the sum of the time subprocesses took for the whole layer. Measured subprocesses include loading attribute values from GeoJSON, data normalization (if any), generating classes, colours, symbol size ranges, applying symbology to features and legend generation. Of the supplied GeoJSON data sets bundled with the examples, three were used to measure performance (with 109, 3220 and 4879 features, respectively).
Rendering of the larger data set comprising 3220 polygons, using a quantile classification into four classes and fill colour distinction, took 16.48 ms averaged over 50 runs (SD = 1.18). In this case, loading attribute values, generation process of classes, colours and legend were around 1-2 ms each. Due to the large amount of features, applying symbology based on the generated classes took 9 ms, still very fast.

The smaller data set consisting of 109 line features, using a natural breaks classification method into five classes and line width distinction, took 6.8 ms averaged over 50 runs (SD = 1.31). Similarly to the previous case, the subprocesses usually took around 1-2 ms (including symbology application), with class generation taking around 3-4 ms occasionally.

Upon inspecting the processing time of the largest data set with 4879 line features, a longer rendering time could be observed. Using a natural breaks (Jenks) classification into five classes and line colour distinction, took 222.34 ms averaged over 50 runs (SD = 12.58). A large part of the total time was taken up by generating classes, 204.50 ms (SD = 7.89). Since the algorithm for natural breaks is more complex, with a lot of values it clearly takes longer than other methods on the same dataset, though not to an extreme degree. This was also observed with the first, still large dataset. By using another classification method for the same exact scenario, the running time suddenly dropped (all other methods took ~1 ms for generating classes, while the classification based on standard deviation took 5 ms on average). Subprocesses took a negligible amount time (0-1 ms), while applying symbology took 10-15 ms due to the higher number of objects (the only major factor affecting this subprocess).

The very low rendering times of the plugin keep the interactive visualization smooth and enjoyable, even with larger data sets or more convoluted classification methods. The run time of background processes remain negligible (1-2 ms), with some settings, especially using the natural breaks algorithm causing a moderate delay before rendering. Generating and displaying the legend took the absolute lowest time overall, but it is understandable, since it basically is just a construction of a HTML element, which is then added to the Leaflet map as an L.control object.

4.3 Dependencies

The plugin currently depends on various third-party libraries. Naturally, it must be accompanied by Leaflet, as the plugin is an extension of the web mapping library. Furthermore, the following external dependencies must be included in the environment: simple-statistics.js, chroma.js. Given the map maker wishes to use hatch patterns for polygon fills, the inclusion of the leaflet-hatchclass plugin is also required.

5. Conclusion, Further Plans

With the web being a platform that provides lots of features and a high degree of customizability for creating web maps, web-based thematic maps still require expertise to visualize geospatial data in a way that highlights spatial differences in an exact and cartographically comprehensive approach. The popular open-source web mapping library, Leaflet, lacks a straightforward approach to create thematic maps that adhere to basic principles of thematic mapping. Even though it requires the least amount of programming experience, as previous research suggested, the feature-rich Leaflet library is quite confusing for new users to build products with for more complex web mapping scenarios. Leaflet does provide basic examples for creating thematic maps, although they describe a hard-coded, static method. By combining all the necessary processes and adhering to most basic principles required in thematic cartography, the authors developed a solution that wraps the individual processes of data classification, symbology, and creation of an appealing legend in an easy-to-use Leaflet-plugin. The resulting thematic map created is concise and visually attractive. This way, a user without any cartographic or thematic mapping knowledge immediately gets an initial product that is already presentable and pleasant to look at by default. However, numerous options are available to provide customization options for all underlying processes, including classification and presentation. This highly customizable plugin facilitates the creation of appealing and clean thematic maps, hopefully increasing the availability and accessibility of such interactive thematic maps on the web in the future. Due to its client-sided nature, the extension can also be suited for implementation in a programmatically controlled environment (e.g. in a dynamic data visualization, where the base dataset is remotely updated, and the visualization is constantly updated).

The project is still ongoing, as some aspects and methods of thematic data visualization are still missing. Future plans include support for creating multivariate maps (bivariate choropleth map), interactive class highlighting and support for raster data. The latter might be realized in a separate plugin, since this plugin heavily builds on extending L.geoJSON specifically. Potentially, support for proportional symbols (without data classification) could be implemented.

References

Agafonkin, V., 2023: Leaflet - a JavaScript library for interactive maps, API reference version 1.9.4. https://leafletjs.com/ (5 December 2023).

Bertin, J., 1967/1983: Semiology of Graphics: Diagrams, Networks, Maps. University of Wisconsin Press, Madison, WI, United States of America. ISBN: 978-0-299-09060-9.

Brewer, A.C., Hatchard, W.G., Harrower, A.M., 2003: ColorBrewer in Print: A Catalog of Color Schemes for Maps. *Cartography and Geographic Information Science*, 30(1), 5-32. doi.org/10.1559/152304003100010929.

Brewer, A.C., 2003: A Transition in Improving Maps: The ColorBrewer Example. *Cartography and Geographic Information Science*, 30(2), 159–162. doi.org/10.1559/152304003100011126.

Chroma.js, 2023. Chroma.js - JavaScript library for all kinds of color manipulations: https://gka.github.io/chroma.js/ (5 December 2023).

Cromley, R.G., 1995: Classes versus unclassed choropleth maps: A question of how many classes. *Cartographica*, 32(4). 15-27. doi.org/10.3138/J610-13NU-5537-0483.

Donohue, R.H., Sack C., Roth R.E., 2013: Time Series Proportional Symbol Maps with Leaflet and jQuery. *Cartographic Perspectives*, 76, 43-66. doi.org/10.14714/CP76.1248.

Farkas, G. (2017). Applicability of open-source web mapping libraries for building massive Web GIS clients. *Journal of Geographical Systems*, Springer, 19(4), 273–295. doi.org/10.1007/s10109-017-0248-z.

Gede, M., 2022: Hatch Fill on Webmaps – to Do or Not to Do, and How to Do. *Abstr. Int. Cartogr. Assoc.*, 5, 48, doi.org/10.5194/ica-abs-5-48-2022.

Horbiński, T., Lorek, D., 2020: The use of Leaflet and GeoJSON files for creating the interactive web map of the preindustrial state of the natural environment. *Journal of Spatial*

Science, 67(31), 1-17. doi.org/10.1080/14498596.2020.1713237.

Horbiński, T., Smaczyński, M., 2023: Interactive Thematic Map as a Means of Documenting and Visualizing Information about Cultural Heritage Objects. *ISPRS International Journal of Geo-Information*, 12(7), 257. doi.org/10.3390/ijgi12070257.

Linfang, D., Liqiu, M., 2014: A comparative study of thematic mapping and scientific visualization. *Annals of GIS*, 20(1), 23-37. doi.org/10.1080/19475683.2013.862856.

MacEachren, A.M., 1995: How Maps Work: Representation, Visualization, and Design. Guilford Press, New York, United States of America.

Mersey, J.E., 1990: Color and thematic map design: the role of color scheme and map complexity in choropleth map communication. *Cartographica 27, Monograph 41*. University of Toronto Press, Toronto, Canada.

Miller, G.A., 1956: The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2), 343-355.

Morrison, J.L., 1974: "A Theoretical Framework for Cartographic Generalization with the Emphasis on the Process of Symbolization." *International Yearbook of Cartography*, 14, 115-127.

Osaragi, T., 2002: Classification methods for spatial data representation. *CASA Working Papers (40)*. Centre for Advanced Spatial Analysis (UCL). London, United Kingdom.

Roth, R.E., Donohue, R.G., Sack, C., Wallace, T.R., Buckingham, T.M.A., 2014: A Process for Keeping Pace with Evolving Web Mapping Technologies. *Cartographic Perspectives*, 78, 25-52. doi.org/10.14714/CP78.1273.

Roth, R.E., 2017: Visual Variables. *The International Encyclopedia of Geography: People, the earth, environment and technology*, John Wiley & Sons, Ltd, 1-11. doi.org/10.1002/9781118786352.wbieg0761.

Simple-statistics.js, 2023. Statistical methods in readable JavaScript for browsers, servers, and people. http://simple-statistics.github.io/ (5 December 2023).

Tanner, J., 2023: classybrew (tannerjt, on GitHub): https://github.com/tannerjt/classybrew (5 December 2023).