SPARQL Visualization Web Components

A JavaScript-based visualization library was developed to support semantic web practitioners and decision-makers in transforming SPARQL query results into meaningful visual representations. The system integrates Vega-Lite grammar for high-level specification of charts and utilizes Web Components to ensure modularity, reusability, and performance across web platforms. The library supports a variety of visual formats including bar charts (regular, stacked, grouped, and percentage), pie charts, and geographical maps (choropleth), all configurable through a JSON-based metadata layer.

Installation

Install via npm

```
npm i kgnovis
```

Usage

1. Add the bundled script

Include the script in your HTML (use from local dist folder or CDN after publishing):

```html
<script type="module" src="dist/kgnovis.bundle.js"></script>
```

2. Use custom Web Components in HTML

```html
<bar-chart id="verticalChart1"></bar-chart>
<bar-chart id="stackedBarChart"></bar-chart>
<pie-chart id="pieChart"></pie-chart>
<map-chart id="choroplethMap"></map-chart>
```

3. The Chart Configuration

The library provides components with common properties for each default component as shown in the table below.

Property	Type	Description	Mandatory
description	String	The description of the chart. This will be the title of the visualization and it will be located below the chart.	✗
width	Number	The width of the component	✗
height	Number	The height of the component	✗

Property	Type	Description	Mandatory
data	SPARQL Json Object	The input data to be visualized. It is an array of object that each object in the array represents one data point. The input data must conform to the W3C SPARQL 1.1 Query Results JSON Format.	✓
encoding	Object	The encoding property of a single view specification represents the mapping between data and visual variables	✓
legend	Boolean	The presence of legend for visualization	✗

3.1 Bar Chart

In the bar chart encoding configuration, the fields provided include:

Field	Type	Description	Value	Mandatory
x	Object	The specification of x axis for Bar Chart		✓
y	Object	The specification of y axis for Bar Chart		✓
color	Object	The specification color for each bar. See 3.4 - Color Palettes for full usage and supported values.		✗ (✓ Required for subtypes of Bar Chart including Stacked, Normalized Stacked and Grouped Bar Chart)
direction	String	The orientation of Bar Chart	"vertical" or "horizontal"	✗
stack	String & Boolean	This attribute is required to define subtypes of Bar Chart including Stacked, Normalized Stacked and Grouped Bar Chart. - When the stack is true, then draw Stacked Bar Chart. - Else if the stack is "normalize", then draw Normalized Stacked Bar Chart. - Else the stack is false, then draw Grouped Bar Chart	true/false or "normalize"	✗

In each object, the library provide some properties:

Field	Type	Description	Value

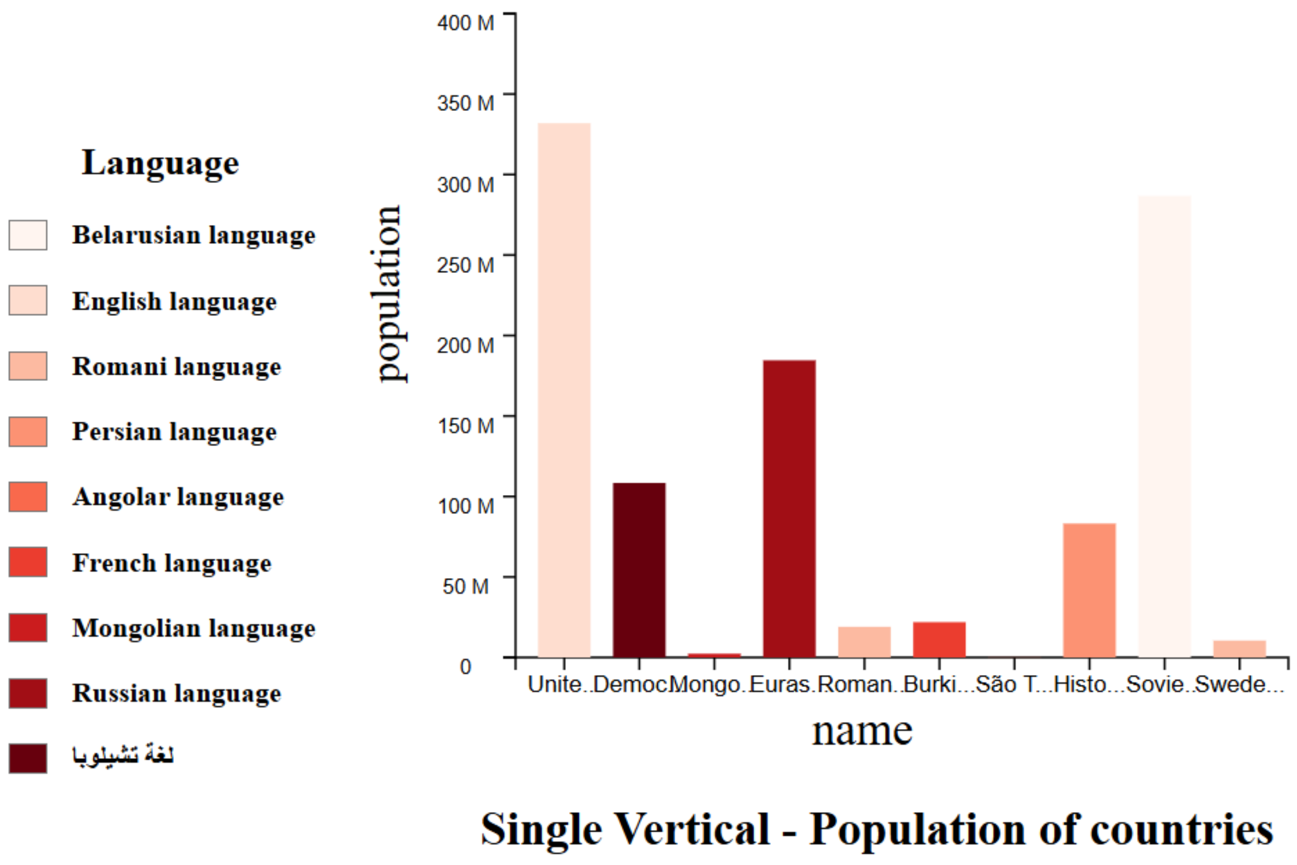
Field	Type	Description	Value

x	"field":"..."
	"axis":
	{"labelAngle":
	...}
		The details about the internal structure of object "x".
		Including:
		- Attribute "field": The "data" field that the "x" axis maps to.
		- Attribute "axis": "labelAngle" - inside axis property will allow user to adjust the rotation angle of labelAxis.
			Ex: x: {
			"field": "name"
			"axis":
			{"labelAngle":45}
			}

y	"field":"..."
	"axis":
	{"labelAngle":
	...
	"scale":{
	"type":...}
	}
		The details about the internal structure of object "y".
		Including:
		- Attribute "field": The "data" field that the "y" axis maps to
		- Attribute "axis":
		+ "labelAngle" - inside axis property will allow user to adjust the rotation angle of labelAxis.
		+ "scale": inside axis property will permit user to change the scaleType through "type" (Linear, Power, Logarithmic) // Power Scale need one more attribute "exponent" to set
			Ex: y: {
			"field":
			"population"
			"axis":
			{"labelAngle":45}
			"scale":{
			"type":"pow",
			"exponent":0.5
			}
			}

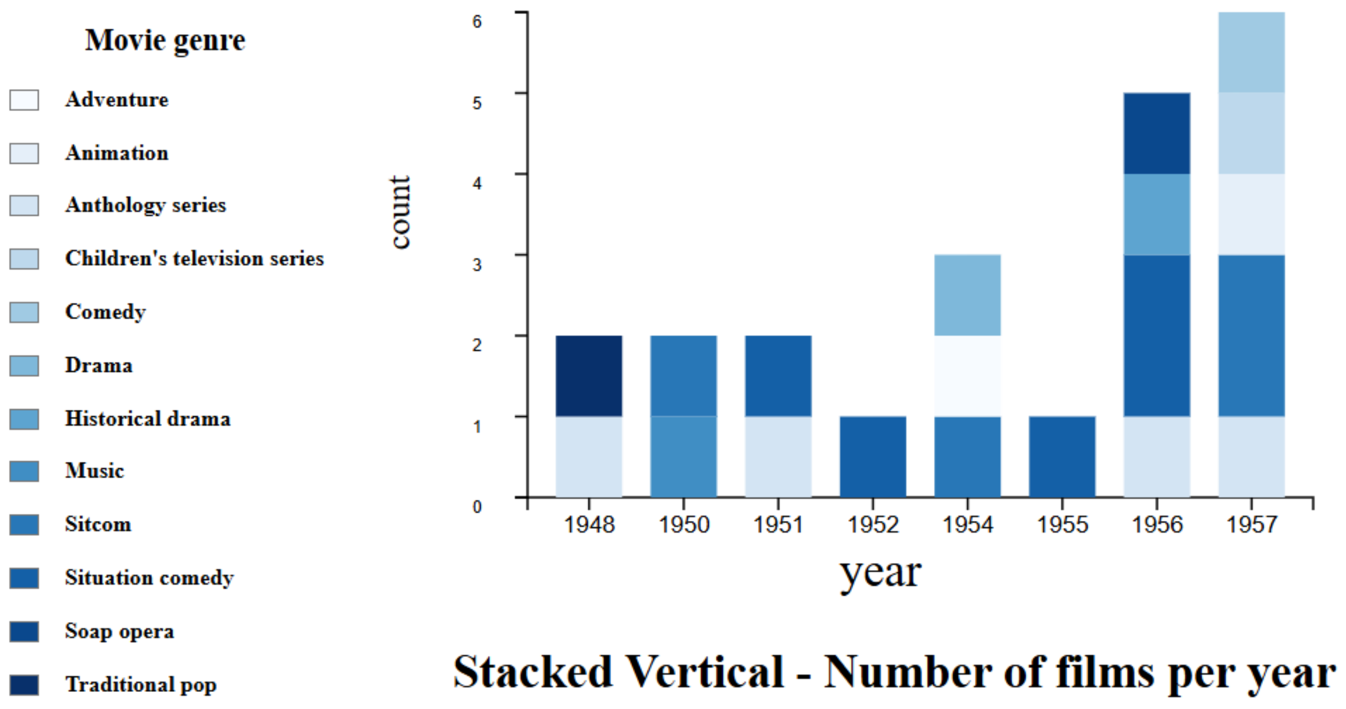
3.1.1 Regular Bar Chart

```
<bar-chart id="verticalChart1"
description="..."
width=500
height=500
data=[{
  name: "United States",
  population: 331893745,
  randomLang: "English language"}, {...}]
encoding={
  x: {"field": "name"},
  y: {"field": "population"},
  color: {
    "field": "randomLang",
    "scale": {"domain": ..., "range" ..},
    "title": "Language"
  }
  direction: "vertical"
}
legend=true
></bar-chart>
```



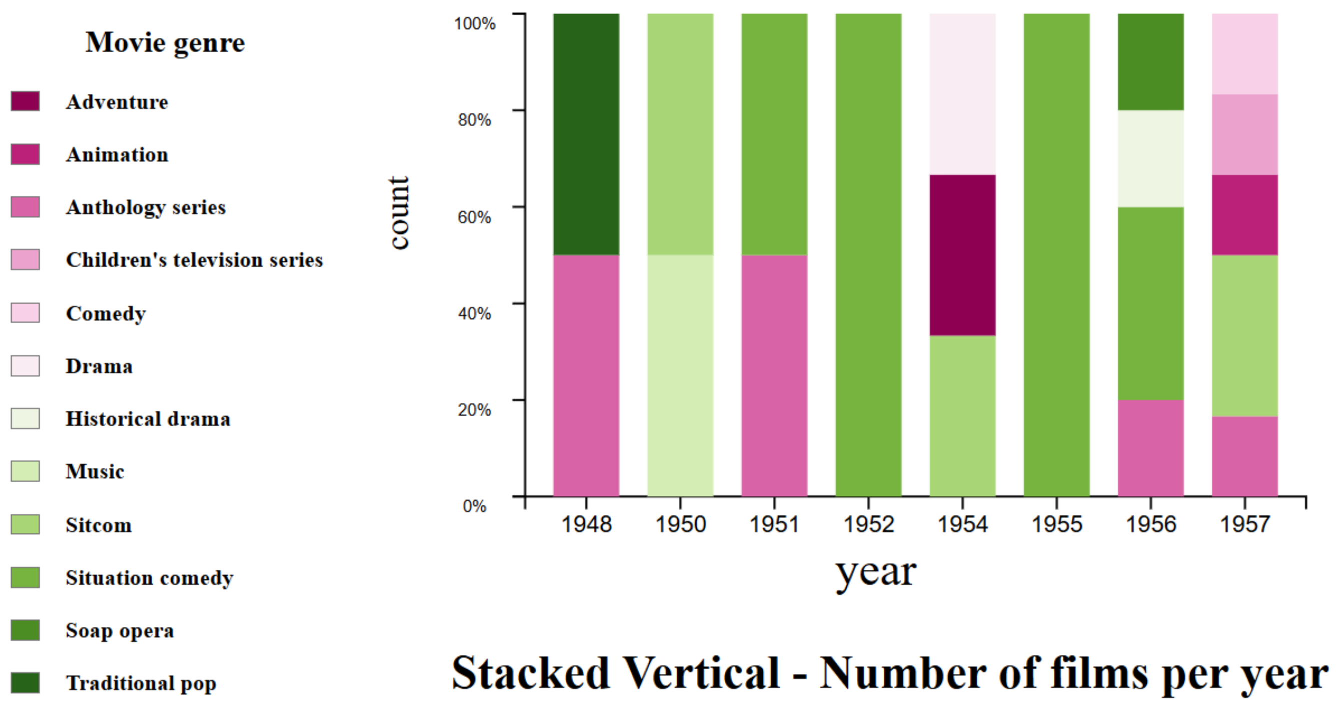
3.1.2 Stacked Bar Chart

```
<bar-chart id="stackedBarChart"
...
data=[{...}, {...}]
encoding={...,
  direction: "vertical",
  stack: true
  }
></barchart>
```



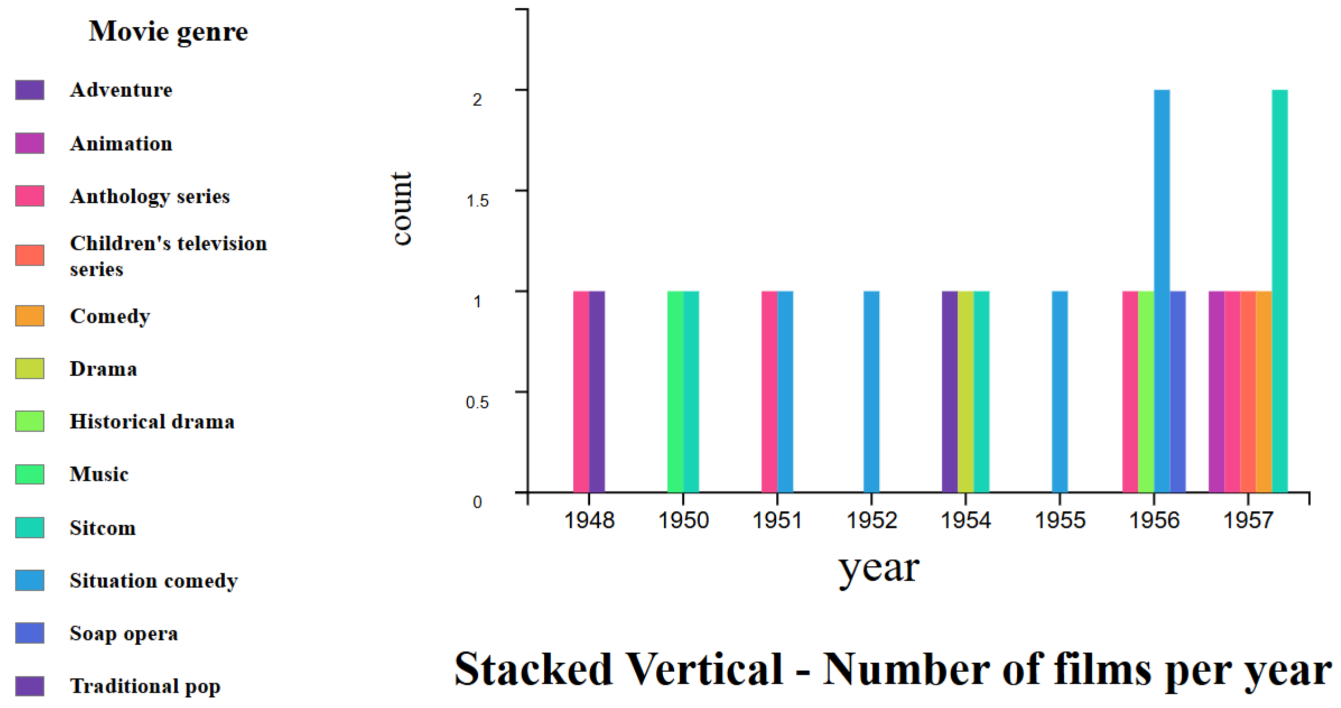
3.1.3 Normalized Stacked Bar Chart

```
<bar-chart id="normalizeBarChart"
...
data=[{...}, {...}]
encoding={...,
  direction: "vertical",
  stack: "normalize"
}
></barchart>
```



3.1.4 Grouped Bar Chart

```
<bar-chart id="groupedBarChart"
...
data=[{...}, {...}]
encoding={...,
  direction: "vertical",
  stack: false
}
></bar-chart>
```



3.2 Pie chart

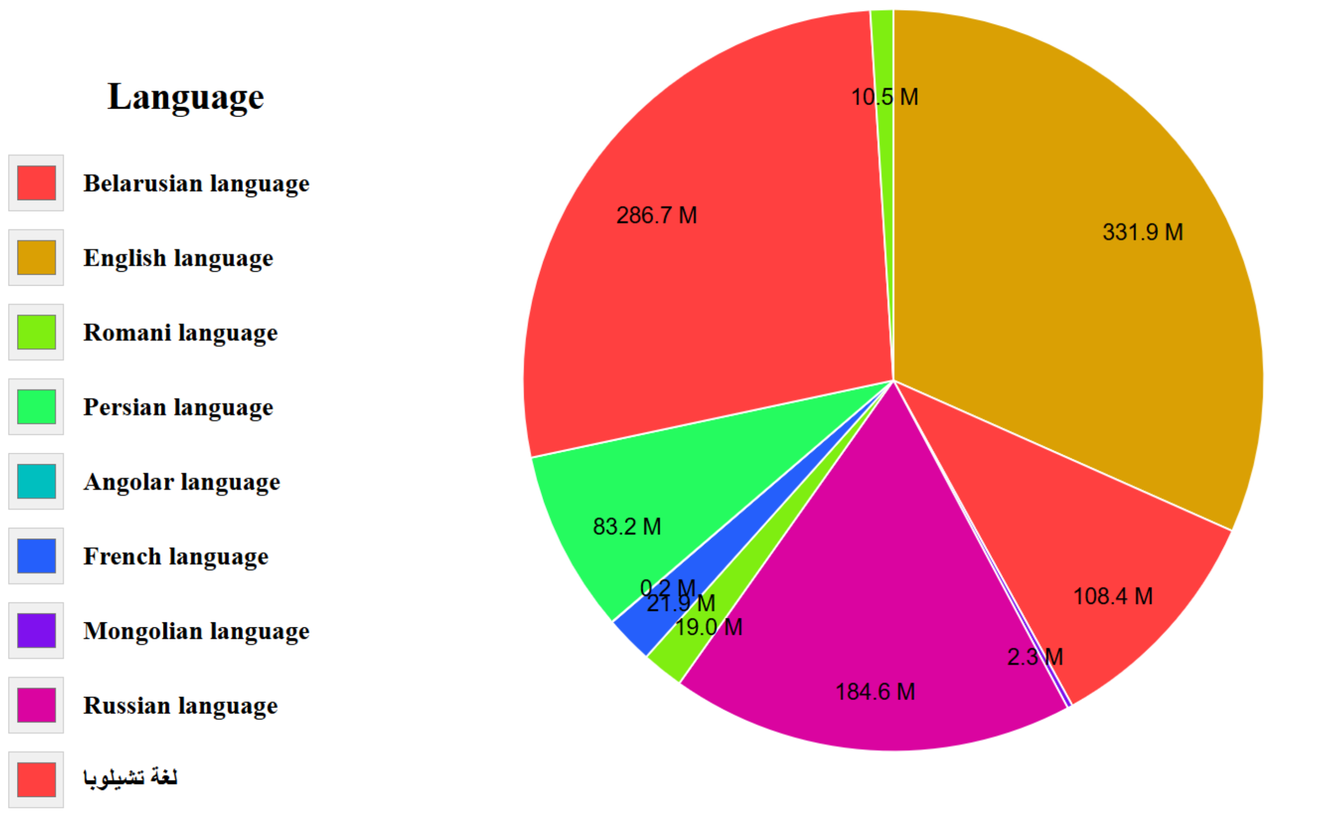
In the Pie Chart encoding configuration, the fields provided include:

Field	Type	Description	Mandatory
text	Object	The label for each section of visualization	✗
theta	Object	The quantitative value for each category	✓
color	Object	The color for each slice. See 3.4 - Color Palettes for full usage and supported values.	✓

In each object, the library provide some properties:

Field	Type	Description	Value
text	"field": "..."	The field from data that will map to the label of each section	Ex: text: {"field": "name"}
theta	"field": "..."	The field that is the quantitative value for each category	Ex: theta: {field": "population"}

```
<pie-chart id="pieChart"
description="..."
width=500
height=500
data=[{
  name: "United States",
  population: 331893745,
  randomLang: "English language"}, {...}]
encoding={
  text: {"field": "name"},
  theta: {"field": "population"},
  color: {
    "field": "randomLang",
    "scale": {"domain": ..., "range" ...},
    "title": "Language"
  },
legend=true
}
></pie-chart>
```



Pie chart - Countries with the largest population

3.3 Geographic Map (Choropleth Map)

In the encoding of Choropleth Map configuration, the fields provided include:

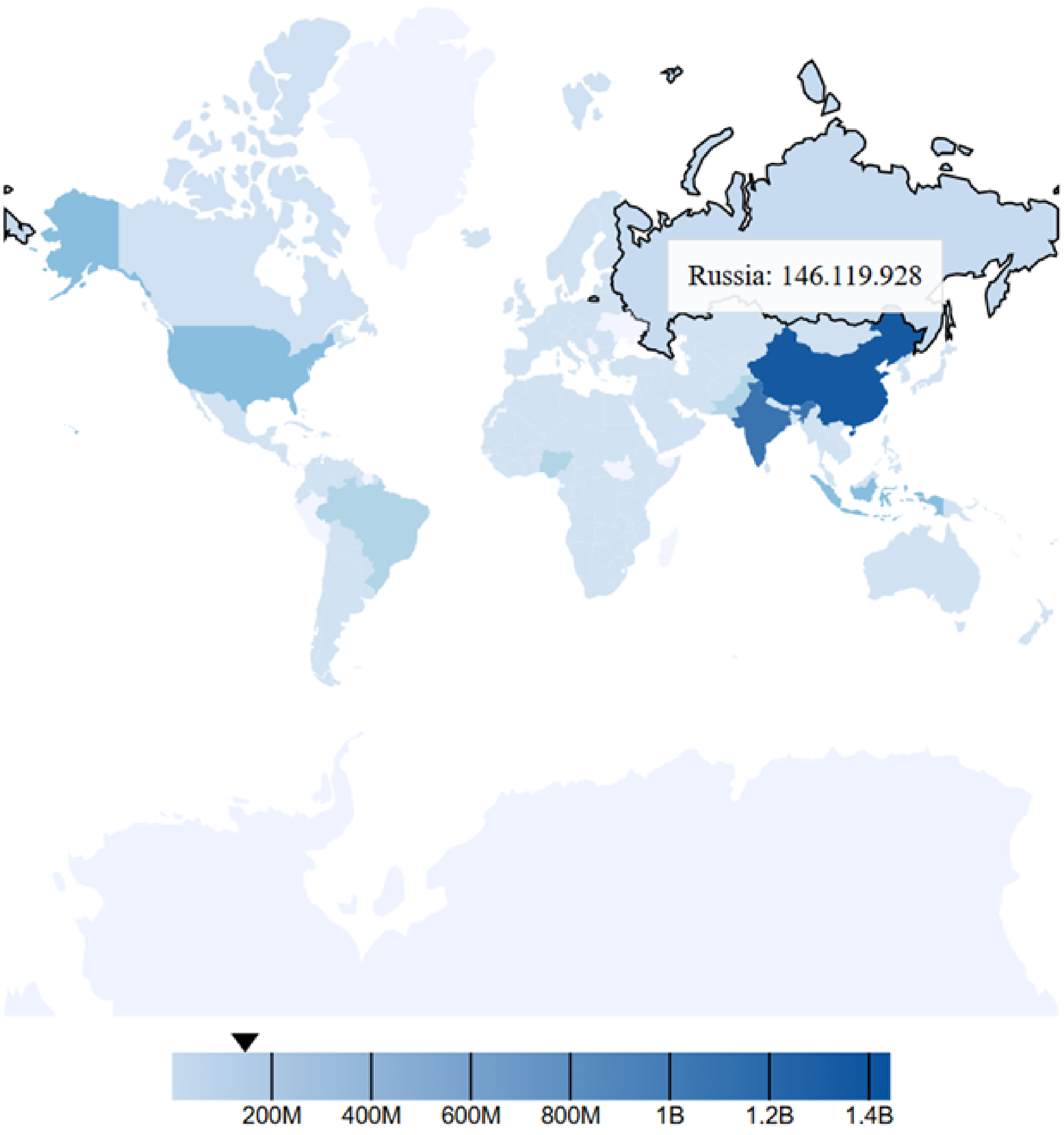
Field	Type	Description	Mandatory
id	Object	The id that used for matching the data from geoJson file and SPARQL query's result	✓
label	Object	A human-readable name that describes the geographic feature represented by the id. This is typically shown in tooltips or legends to provide a meaningful name (e.g., country or region name) instead of a technical identifier.	✗

Field	Type	Description	Mandatory
geometry	Object	Data from the geoJSON file, must be processed during querying to retrieve GeoJSON geometry data for each entity.	✓
color	Object	This field will color the regions based on data from the SPARQL query's result. See 3.4 - Color Palettes for full usage and supported values.	✓

In each object, the library provide some properties:

Field	Type	Description	Value
id	"field":"..."	The id is used for matching the data from geoJSON file and SPARQL query's result. This attribute comes from the geoJSON file.	Ex: id: {"field": "isoCode"}
label	"field":"..."	The field from data that map to name of the label, which provides additional details for describing the id	Ex: label: {field": "countryName"}
geometry	"field":"..."	The field from data that contains GeoJSON geometry data for each entity (polygon, contour line...). Used for map display.	Ex: geometry: {field": "geoJSON"}

```
<map-chart id="choroplethMap"
description="..."
width=500
height=500
data=[{
    isoCode: "USA",
    population: 340110988,
    countryLabel: "United States"
    geoJSON: {...}}, {...}],
encoding={
  id: {"field": "isoCode"},
  label: {"field": "countryLabel"},
  geometry: {"field": "geoJSON"},
  color: {
    "field": "population",
    "scale": {"range": ...},
  },
  }
></map-chart>
```



3.4 Color Palettes

The "color" property is an important property for most types of charts today. This module provides sequential, divergent, and categorical color schemes designed to work with color palettes from the D3.js library. Most of these schemes are derived from ColorBrewer by Cynthia A. Brewer.

Improvements over D3.js library To provide flexibility and ease of use when customizing chart colors, the library includes a powerful color palette parsing function. This feature allows users to define palettes using intuitive string inputs, which are automatically mapped to their corresponding color schemes in the D3.js library D3 Color Palettes

3.4.1 Operation principle

Users can apply color schemes by simply providing a string. The system automatically maps the string to the appropriate D3 color palette.

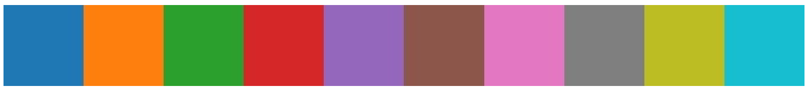
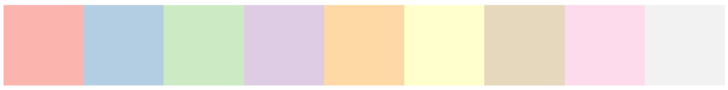
How to use

- "PaletteName" → Maps to d3.interpolatePaletteName (for continuous gradients) or d3.schemePaletteName (for discrete categories).
- "PaletteName[k]" → Maps to d3.schemePaletteName[k], where k defines the number of colors (if supported)

Below are examples for use with each type of color palette.

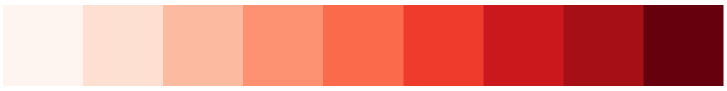
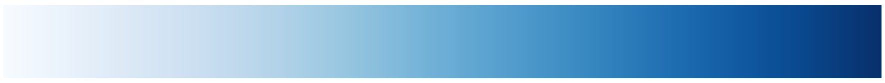
3.4.1.1 Categorical Palettes *(Discrete)*

Used to differentiate distinct groups or categories.

- "Category10" → d3.schemeCategory10
  
- "Pastel1" → d3.schemePastel1 

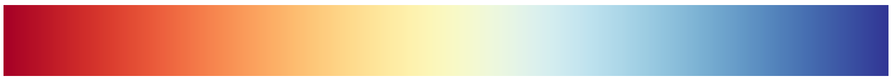
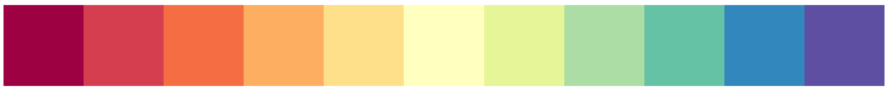
3.4.1.2 Sequential Palettes *(Ordered)*

Best for representing ranked or continuous data (e.g., values, density).

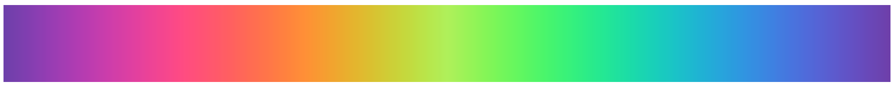
- "Reds[5] → d3.schemeReds[5] (5 shades of red), valid k ∈ [3, 11]
  
- "Blues" → d3.interpolateBlues (smooth blue gradient)
  

3.4.1.3 Diverging Palettes *(Centered Scales)*

Ideal for data with a meaningful midpoint (e.g., gain vs. loss).

- "RdYlBu" → d3.interpolateRdYlBu (red → yellow → blue)
  
- "Spectral[9]" → d3.schemeSpectral[9], valid k ∈ [3, 11]
  

3.4.1.4 Cyclical Palettes *(Repeating Patterns)*

- "Rainbow" → d3.interpolateRainbow
  

3.4.2 Color Configuration

Property	Type	Description
field	String	The field from "data" that will bind with "color"
→ domain	Array	List of distinct values that exist in the data (for categorical data). Optional for continuous values.
→ range	String	The color palette name (e.g., "Reds", "Blues", "Viridis").
title	String	(Optional) Title shown in the legend to describe what the colors represent.

Example

```
"color": {
  "field": "population",
  "scale": {
    "range": "Reds"
  },
  "title": "Population"
}
```

For categorical data, add domain:

```
"color": {
  "field": "language",
  "scale": {
    "domain": ["English", "Spanish", "French"],
    "range": "Category10"
  },
  "title": "Language"
}
```