Custom Charts for RAWGraphs

Thomas Aumüller, Martin Heider, and Abdelrahman Ramadan

706.057 Information Visualisation 3VU SS 2024
Graz University of Technology

07 Aug 2024

Abstract

RAWGraphs is a powerful, open-source tool for data visualization, which allow users to create their own charts and graphs. This report documents the steps neccessary to create custom charts for RAWGraphs. Three types of custom charts were implemented and added to RAWGraphs, namely similarity map, connected scatterplot, and polar area diagram. The charts were implemented with D3 and integrated into RAWGraphs using its recently added custom charts facility. In addition, a tutorial was written, explaining how to create custom charts for RAWGraphs.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

This project aimed to create a number of custom charts for RAWGraphs [DensityDesign 2024a; DensityDesign 2024b; Mauri et al. 2017], a web-based open-source tool for data visualization. This report first provides a general overview of RAWGraphs, then descibes how to create custom charts for RAWGraphs. Three custom charts were created as part of this project, they can be found on GitHub [Ivis-G1 2024].

RAWGraphs is an open-source web application for data visualization. It is often used by non-programmers to create static charts and graphs as vector graphics. It allows for export in SVG, JPEG, and PNG and to also save all current data and settings into a `.rawgraphs` file, for easy reloading of a project at a later date. User often save their chart as SVG and polish it in a vector graphics editor like Illustrator or Inkscape.

RAWGraphs is written in JavaScript with React [Meta 2024] and uses D3 [Bostock 2024] to create the various standard charts. There are 32 standard charts, including classics like line chart, bar chart, and pie chart, as well as more exotic charts like parallel coordinates, treemap, and chord diagram. The charts are grouped into seven categories by data type (some charts are in multiple categories): Correlations, Proportions, Networks, Distributions, Time Series, Time Chunks, and Hierarchies.

Since RAWGraphs is open source, anyone can run their own version of RAWGraphs locally or on a web server. One important aspect is the possibility to implement custom graphs and use them within the RAWGraphs framework. They can also be shared with others in the form of `.rawgraphs` files.

The RAWGraphs user interface guides a user through the process of creating a chart. As can be seen in Figure 1.1, the five steps are:

1. *Load your data*: Upload a dataset, or choose a one from the from curated set of sample datasets, which are provided for specific charts.
2. *Choose a chart*: Choose a chart to visualize the data.
3. *Mapping*: Map the dimensions of the data to chart-specific variables, by dragging dimensions to the desired fields.
4. *Customize*: A chart is generated. Its appearance can be changed with various settings and options.
5. *Export*: Export the chart in one of four formats: `.svg`, `.png`, `.jpg`, or `.rawgraphs`.

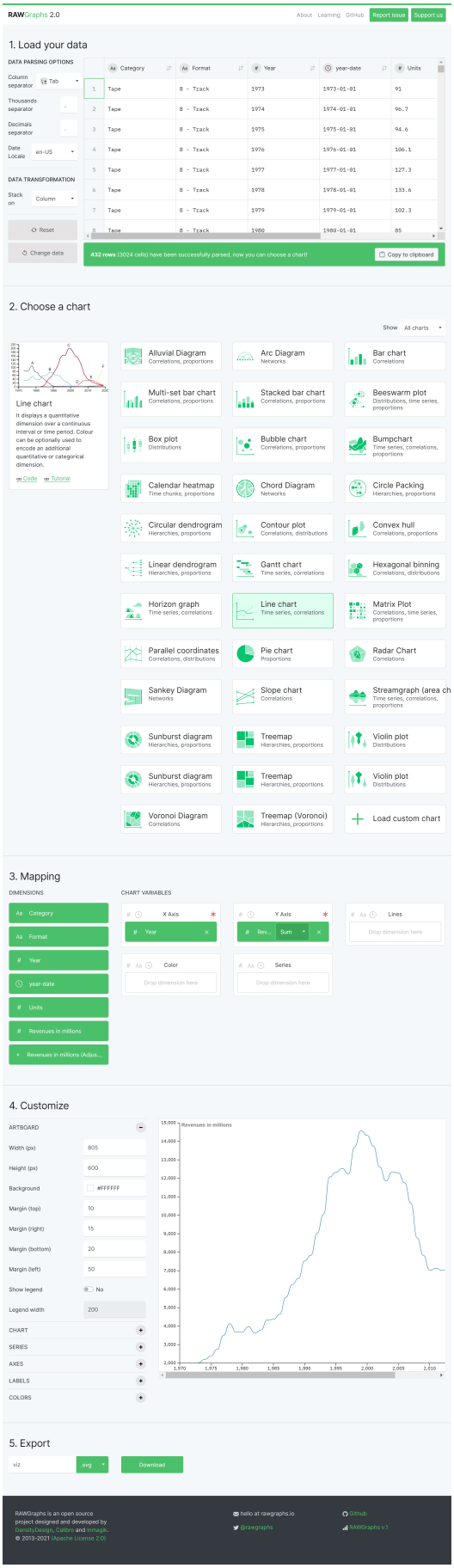

Figure 1.1: The RAWGraphs user interface, showing the five steps for creating a chart. [Image used with kind permission of Keith Andrews.]

Chapter 2

Custom Charts

RAWGraphs 2.0 now allows users to create their own custom charts which RAWGraphs can directly load. These created custom charts can then be selected and used just as regular RAWGraphs charts. Generally speaking, custom charts are code that can be loaded on the fly by the RAWGraphs app, extending its possibilities.

Until now, using charts created by someone else was possible but a little bit tricky. You had to clone the source code, modify it, build it, and host it on your server. This approach can still be done if you want to test it on your own development server but other than that RAWGraphs now officially supports loading custom charts directly on the web interface.

Even though RAWGraphs now provides some instructions and there was a video conference where everything was explained on how to create your custom chart, there is rather little information on how actually to create and test your charts. Therefore, a small tutorial on how to create and make it work was created on top of a general overview of folder structures and file examples.

Because datasets are needed for visualization and testing the custom charts an overview of all the permissions and sources for each dataset used was provided.

2.1 How to Create Custom RAWGraph Chart

Here is a small tutorial on how to create your own RAWGraphs chart.

2.1.1 Setup

Clone our repository:

```
git clone git@github.com:solidth/RAWGraphs-Custom-Charts.git
```

or the official template from RAWGraphs:

```
git clone git@github.com:rawgraphs/custom-rawcharts-template.git
```

Open the folder containing the repository:

```
cd <name-of-the-repo>
```

and install client-side dependencies:

```
npm install
```

You can now run the sandbox environment to test your charts:

```
npm run sandbox
```

After running the sandbox, you can look at the live preview in your browser under:

```
localhost:9000
```

```
// index.js
export { default as similaritymap } from './similaritymap'
export { default as connectedscatterplot } from './connectedscatterplot'
export { default as polarareadiagram } from './polarareadiagram'

// add your chart
export { default as <your-chart-name> } from './<your-chart-name>'
```

Listing 2.1: The index file `index.js` in the `src/` folder.

```
chart/
  chart_thumb.svg
  chart.js
  chart.svg
  dimensions.js
  index.js
  mapping.js
  metadata.js
  render.js
  visualOptions.js
```

Listing 2.2: Chart folder tree.

2.1.2 Create your Chart

Navigate to the `src` folder.

```
cd \src
```

Here you create a folder with the name of your chart:

```
mkdir <your-chart-name>
```

After creating the folder you need to modify the `index.js` file in the `src/` folder. The `index.js` file will typically look something like in the example in Listing 2.1. Here, you just need to add your chart.

Now you can start adding the necessary files to your `<your-chart-name>` folder, as shown in Listing 2.2. For better readability, the moniker `chart` is used on place of `<your-chart-name>`.

The added files are:

- `chart_thumb.svg`: SVG thumbnail for the chart.
- `chart.js`: Main module that ties together all the components of your custom chart.
- `chart.svg`: Sample chart displayed when your custom chart is selected.
- `dimensions.js`: Defines the dimensions and configuration of the data dimensions used in the chart.
- `index.js`: Entry point for your custom chart. It exports the main module of your chart.
- `mapping.js`: Defines the data mapping rules for your chart.

```
project-root/
  README.md
  src/
    chart/
      chart_thumb.svg
      chart.js
      ...
      visualOptions.js
    other-chart/
  ...
  index.js
```

Listing 2.3: Project folder tree.

- `metadata.js`: Contains metadata about your custom chart.
- `render.js`: Responsible for the actual rendering of the chart.
- `visualOptions.js`: Defines the visual options and customization settings for your chart.

The project folder structure should look like the one in Listing 2.3.

For a more detailed look into the files, clone the official RAWGraphs charts repository [DensityDesign 2023]:

```
git clone git@github.com:rawgraphs/rawgraphs-charts.git
```

and look at their example charts and all of files in the `src/` folder.

2.1.3 File Examples

Here, in Listings2.4 to 2.10, we give examples for each of the JavaScript files in the `chart/` folder. Remember, for better readability, we use `chart` in place of `<your-chart-name>`.

2.1.4 Test Your Chart in the Sandbox

To test your chart inside the local Sandbox `localhost:9000`, follow these steps. For consistency and readability, we continue to use the moniker `chart` instead of `<your-chart-name>`. The structure of the `example/` folder should look something like the one in Listing 2.11.

Prepare a dataset for your custom chart in a CSV fileand place it in the folder `example/datasets/`. Configure the file `chart-test.js` in the folder `example/configurations/` to be something like the example in Listing 2.12.

After creating your chart test file, run the following command in the root of your project:

```
npm run sandbox
```

After the sandbox starts correctly, you can view the live preview of your testing-file in your browser under `http://localhost:9000`.

```
import { metadata } from './metadata'
import { dimensions } from './dimensions'
import { mapData } from './mapping'
import { render } from './render'
import { visualOptions } from './visualOptions'
import styles from '../styles/base.raw.css'

export default {
  metadata,
  dimensions,
  mapData,
  render,
  visualOptions,
  styles,
}
```

Listing 2.4: The chart file `chart.js`.

```
export const dimensions = [
  {
    id: 'x',
    name: 'Left Side',
    validTypes: ['number'],
    required: true,
  },
  {
    id: 'y',
    name: 'Y axis',
    validTypes: ['number', 'string', 'date'],
    required: true,
  }
]
```

Listing 2.5: The dimensions file `dimensions.js`.

```
export { default } from './chart'
```

Listing 2.6: The index file `index.js` in the `chart/` folder.

```
export const mapData = {
  x: 'get',
  y: 'get'
}
```

Listing 2.7: The mapping file `mapping.js`.

```
import icon from './chart.svg'
import thumbnail from './chart_thumb.svg'

export const metadata = {
  name: 'Your Custom Chart',
  id: 'rawgraphs.chart',
  thumbnail,
  icon,
  categories: ['Correlations', 'Comparison', ...],
  description: 'this is a description',
  code: '...',
  tutorial: '...',
}
```

Listing 2.8: The metadata file `metadata.js`.

```
import * as d3 from 'd3'
import { legend, labelsOcclusion } from '@rawgraphs/rawgraphs-core'
import '../d3-styles.js'

export function render(
  svgNode,
  data,
  visualOptions,
  mapping,
  originalData,
  styles
) {
  // JavaScript Code for Visualisation
}
```

Listing 2.9: The render file `render.js`.

```
export const visualOptions = {
  marginTop: {
    type: 'number',
    label: 'Margin (top)',
    default: 10,
    group: 'artboard',
  },
  marginRight: {
    type: 'number',
    label: 'Margin (right)',
    default: 10,
    group: 'artboard',
  },
  marginBottom: {
    type: 'number',
    label: 'Margin (bottom)',
    default: 10,
    group: 'artboard',
  },
  marginLeft: {
    type: 'number',
    label: 'Margin (left)',
    default: 10,
    group: 'artboard',
  },
  showLegend: {
    type: 'boolean',
    label: 'Show legend',
    default: false,
    group: 'artboard',
  },
  // and many more ...
}
```

Listing 2.10: The visual options file `visualOptions.js`.

```
project-root/
 ...
 example/
  components/
  configurations/
   ...
   chart-test.js
  datasets/
   ...
   <your-dataset>.csv
  App.js
  index.css
  index.html
  index.js
```

Listing 2.11: Structure of the test folder tree to test a chart in the sandbox.

```
// import your dataset
import data from '../datasets/<your-dataset>.csv'

// import custom chart you created
import chart from 'customcharts/chart'

// this is just an example of a barchart, modify it as you see fit for your custom
    chart.
export default {
  chart,
  data,
  dataTypes: {
    Year: {
      type: 'date',
      dateFormat: 'YYYY',
    },
    Age: 'string',
    Male: 'number',
  },
  mapping: {
    x: { value: ['Male'] },
    y: { value: ['Age'] },
  },
  visualOptions: {
    width: 800,
    height: 600,
    padding: 0,
    labelLeftRotation: 45,
    labelLeftAlignment: 'start',
    background: 'white',
    title: 'My title',
  },
}
```

Listing 2.12: An example test file for custom charts `chart-test.js`.

2.1.5 Build RAWGraphs with Your Chart

When you are satisfied with your project, you can build the js bundle to be used in the RAWGraphs interface. Navigate to the folder `rawgraphs-custom-charts/`, where you cloned this template and run:

```
npm run build
```

This will generate a folder named `lib/` in which you will find three files. The file named `index.umd.js` is the bundle containing the custom chart(s) that can be loaded by RAWGraphs, as described in Section 2.2.2. This file can also be renamed to something more meaningful.

2.2 Quick Start Guide

This is a quick start guide to building the custom charts delivered with this project.

First clone the git Repository:

```
git clone git@github.com:solidth/RAWGraphs-Custom-Charts.git
```

Browse the folder containing the repository:

```
cd rawgraphs-custom-charts
```

Install client-side dependencies:

```
npm install
```

You can now run the sandbox environment to test your charts:

```
npm run sandbox
```

After running the sandbox, you can look at the live preview under:

```
localhost:9000
```

2.2.1 Build a Local RAWGraphs Server

This section describes how to build and run a local RAWGraphs server for testing the custom charts implementation as part of an integrated RAWGraphs app rather than in the sandbox.

Clone the two official RAWGraphs repositories:

```
git clone git@github.com:rawgraphs/rawgraphs-charts.git
git clone git@github.com:rawgraphs/rawgraphs-app.git
```

Install Node v14 or above if necessary:

```
nvm install 14
nvm use 14
```

Go to the folder `rawgraphs-charts/`, install the dependencies:

```
yarn install
npm install
```

and create a build:

```
npm run build
```

Then, create a link with the command:

```
yarn link
```

Now, go to the `rawgraphs-app/` folder and issue the command:

```
yarn link "@rawgraphs/rawgraphs-charts"
```

Install the dependencies:

```
yarn install
npm install
```

After that, the local development server should be ready to go and you can test it locally by starting the app:

```
npm run start
```

2.2.2 Creating a Custom Chart Bundle

When you are satisfied with your project, you can build the JavaScript bundle to be used in the RAWGraphs interface. In a terminal, navigate to the folder in which you cloned this template and run:

```
npm run build
```

This will generate a folder named `lib/` in which you will find three files. The file named `index.umd.js` is the bundle that can be loaded by RAWGraphs. This file can be renamed.

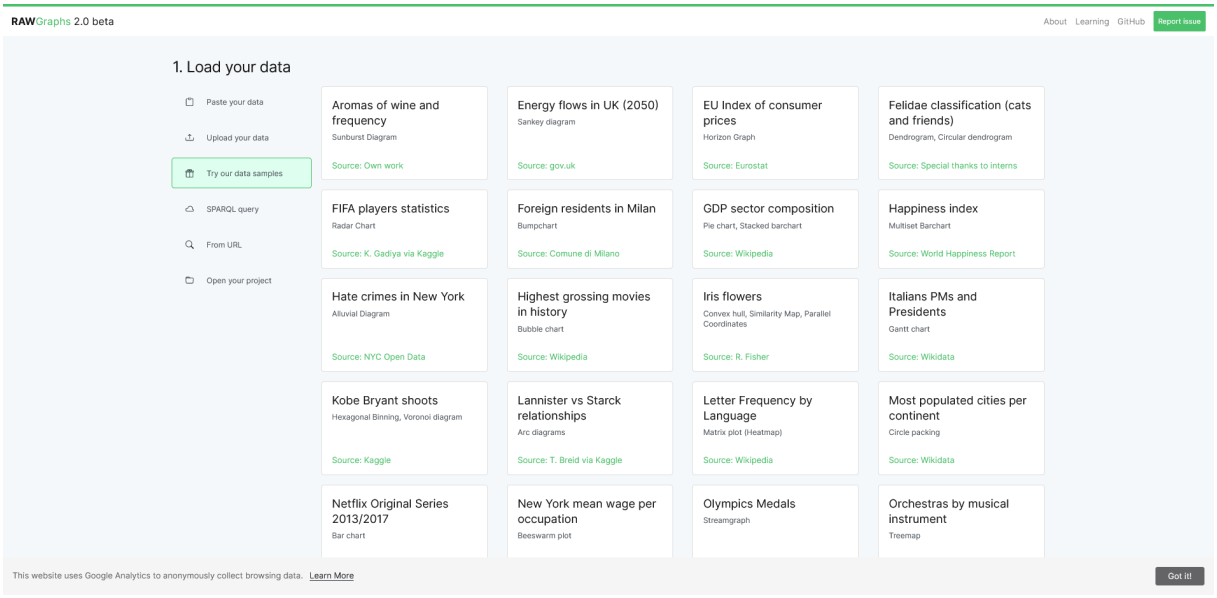

Figure 2.1: Select your dataset for the custom chart.

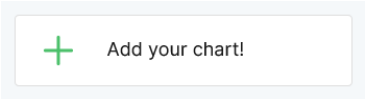

Figure 2.2: Click Add Your Chart!.

2.2.3 Loading a Custom Chart Bundle

After you created a bundle and have the `index.umd.js` file or `<name>.umd.js` file, you can start your development server and open it in a browser. Then:

1. Load your data.
2. Click Try our data samples, as shown in Figure 2.1.
3. In Choose a chart go to Add your chart!, as shown in Figure 2.2.
4. Drag or select your bundle `index.umd.js` or `<name>.umd.js`, as shown in Figure 2.3.

2.3 Permissions and Resources

This section provides information about all the permissions, licenses, and credits, as well as documentation for the sources of files, datasets, and code snippets.

2.3.1 Datasets

All sources of the used datasets and their licence terms.

Connected Scatterplot - Driving Data

- Driving.csv

`https://observablehq.com/@d3/connected-scatterplot`
`https://archive.nytimes.com/www.nytimes.com/imagepages/2010/05/02/business/02metrics.html?action=click&module=RelatedCoverage&pgtype=Article%C2%AEion=Footer`

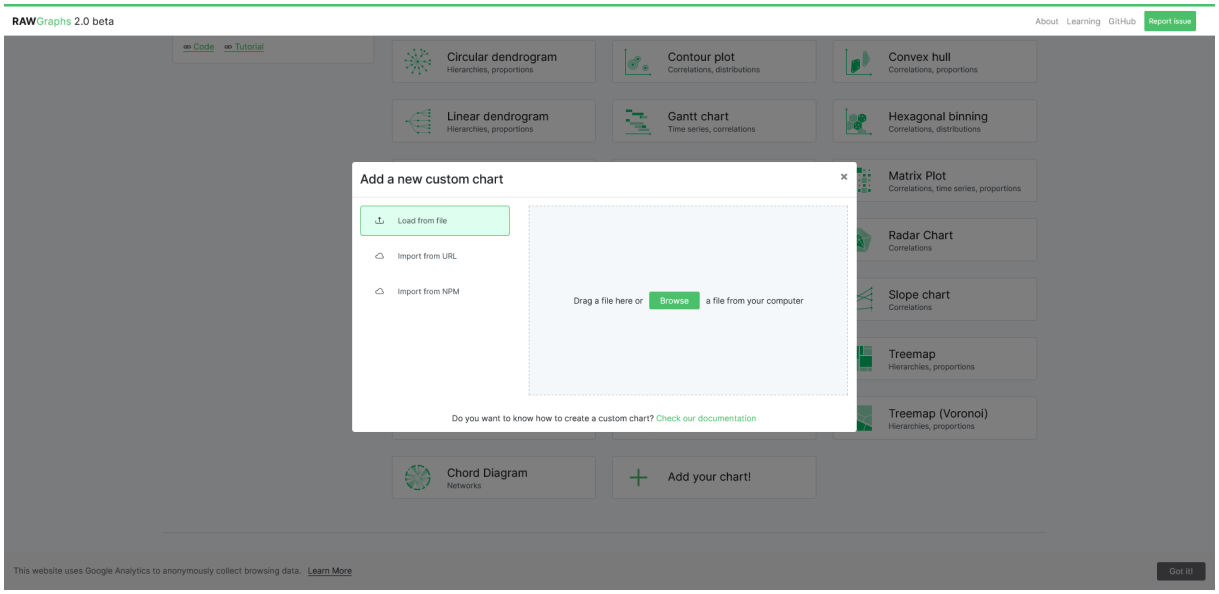

Figure 2.3: Add the bundle for your custom chart.

ISC License

Copyright 2018–2021 Observable, Inc. Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Connected Scatterplot - Germany Population Growth

- GermanyPopulationGrowth.csv

`https://blog.datawrapper.de/connected-scatterplots/`
`https://population.un.org/wpp/Download/Standard/MostUsed/`

Creative Commons License

Copyright © 2022 by United Nations, made available under a Creative Commons license CC BY 3.0 IGO: http://creativecommons.org/licenses/by/3.0/igo/ Suggested citation: United Nations, Department of Economic and Social Affairs, Population Division (2022). World Population Prospects 2022, Online Edition.

Similarity Map - Students Dropout and Academic Success

- PredictStudentsDropoutAcademicSuccess.csv

`https://archive.ics.uci.edu/dataset/697/predict+students+dropout+and+academic+success`

Creative Commons License

This dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license.

This allows for the sharing and adaptation of the datasets for any purpose, provided that the appropriate credit is given.

Polar Area Diagram - Nightingale's Rose Data

- nightingale-data.csv

https://github.com/tulsyanp/data-visualization-d3-js-nightingale-rose-chart/tree/master

No license information.

Polar Area Diagram - Global Land Temperatures by Country

- GlobalLandTemperaturesByCountry.csv

https://www.kaggle.com/datasets/vijayvvenkitesh/global-land-temperatures-by-country

CC0: Public Domain

The person who associated a work with this deed has dedicated the work to the public domain by waiving all of his or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law. You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission. See Other Information below.

2.3.2 Code

All sources of the used code as well as their licences.

TSNE

- tsne.js

Original code: https://github.com/karpathy/tsnejs

Modified code by @blindguardian50, @steve1711, @TheAlmightySpaceWarrior, @wizardry8.

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

UMAP

- UMAP.js

https://github.com/PAIR-code/umap-js

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

PCA

- PCA.js

https://github.com/bitanath/pca

2.3.3 Credits

Original code and template for Similarity Map by: @blindguardian50, @steve1711, @TheAlmightySpaceWarrior, @wizardry8.

Final implementation of Similarity Map and original code for Connected Scatterplot and Polar Area Diagram by: @solidth, @hezojez, and @Ramadan877 under the supervision of @kandrews99.

Chapter 3

Similarity Map

A similarity map is a visualization technique that takes multidimensional data and project the data points to two (or sometimes three) dimensions, so that similar records are close to one another. The projection is typically done using dimensionality reduction techniques such as PCA, t-SNE, or UMAP. For this project, the work of students of previous years was adapted and modified [Schweiger et al. 2018; Doppelreiter et al. 2022].

3.1 Tools

Students from previous years had used a JavaScript implementation of of tSNEJS by karpathy [2016a]. There is also an accompanying live demo [karpathy 2016b]. This JavaScript library was modified by our peers from previous years @blindguardian50 @steve1711, @TheAlmightySpaceWarrior, @wizardry8 for their project. It worked sufficiently well for the time being thanks to the work of our peers, so it was adopted for this project [Doppelreiter et al. 2022].

PCA is a well-known, deterministic dimensionality reduction technique. The JavaScript library chosen for PCA [bitanath 2023] offered first and foremost great performance.

The third and final dimensionality reduction technique for this project was UMAP and its implementation by cannoneyed [2024].

Thanks to all the openly available libraries it was possible to implement these techniques in the similarity map for RAWGraphs.

3.2 Data

A sample dataset called "Predict Students' Dropout and Academic Success" was chosen from the Machine Learning repository at UC Irvine [UCI 2024b; UCI 2024a].

3.3 Implementation

We started by taking the work that was done by the previous team, who already had an almost functioning implementation of the graph using only the tSNE projection method. After getting that to run locally and on the new version of RAWGraphs, the projection methods were extended by also implementing PCA and UMAP using the aforementioned libraries. The result can be seen in the porject repository [Ivis-G1 2024].

Since our peers did a great job at implementing the basic functionality of the similarity map, the majority of the workload was getting it to run, adding the additional dimensionality reduction techniques and finally figuring out how to allow the user to select options in the RAWGRaphs user interface.

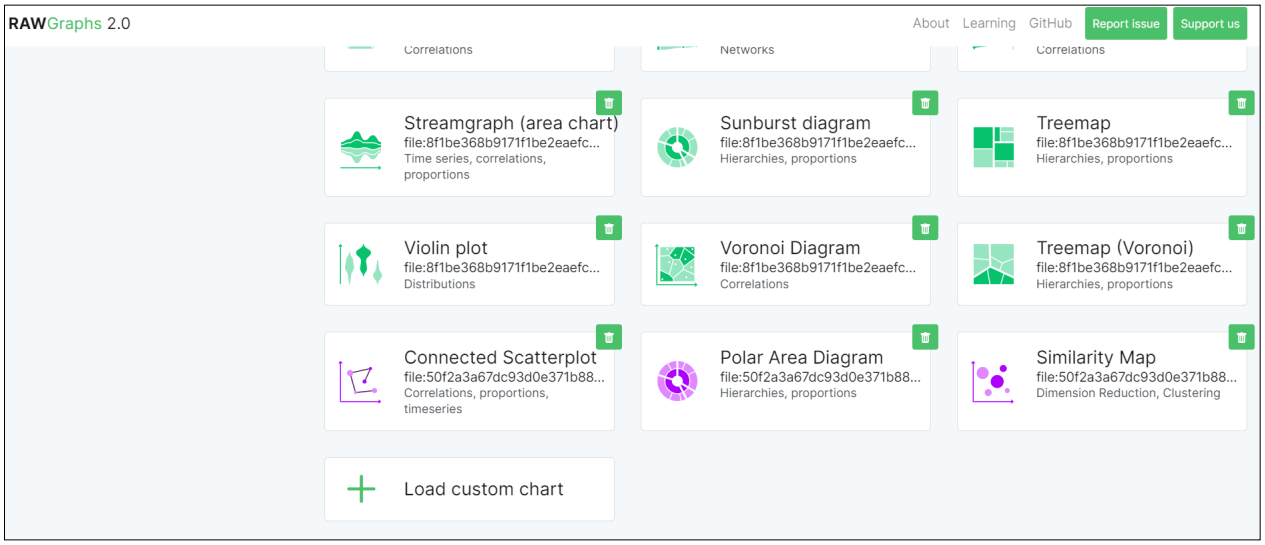

Figure 3.1: Step 2: Choosing the Similarity Map as a custom chart in RAWGraphs.

3.4 User Interface

The five steps necessary to use RAWGraphs to create a similarity map are:

1. *Load your data*: Load a multidimensional dataset, or choose the sample dataset "Predict Students' Dropout and Academic Success".
2. *Choose a chart*: Since the similarity map is a custom chart, it first has to be uploaded into RAWGraphs. Then it can be found at the end of the grid of charts, as shown in Figure 3.1.
3. *Mapping*: Dimensions from the dataset are mapped to the chart's variables, as depicted in Figure 3.2.
4. *Customize*: A multitude of options are available to configure and customize the chart, as can be seen in Figure 3.3.
5. *Export*: Export the chart in SVG, PNG, JPEG or `.rawgraphs` format. Figure 3.4 shows the final chart after exporting as SVG.

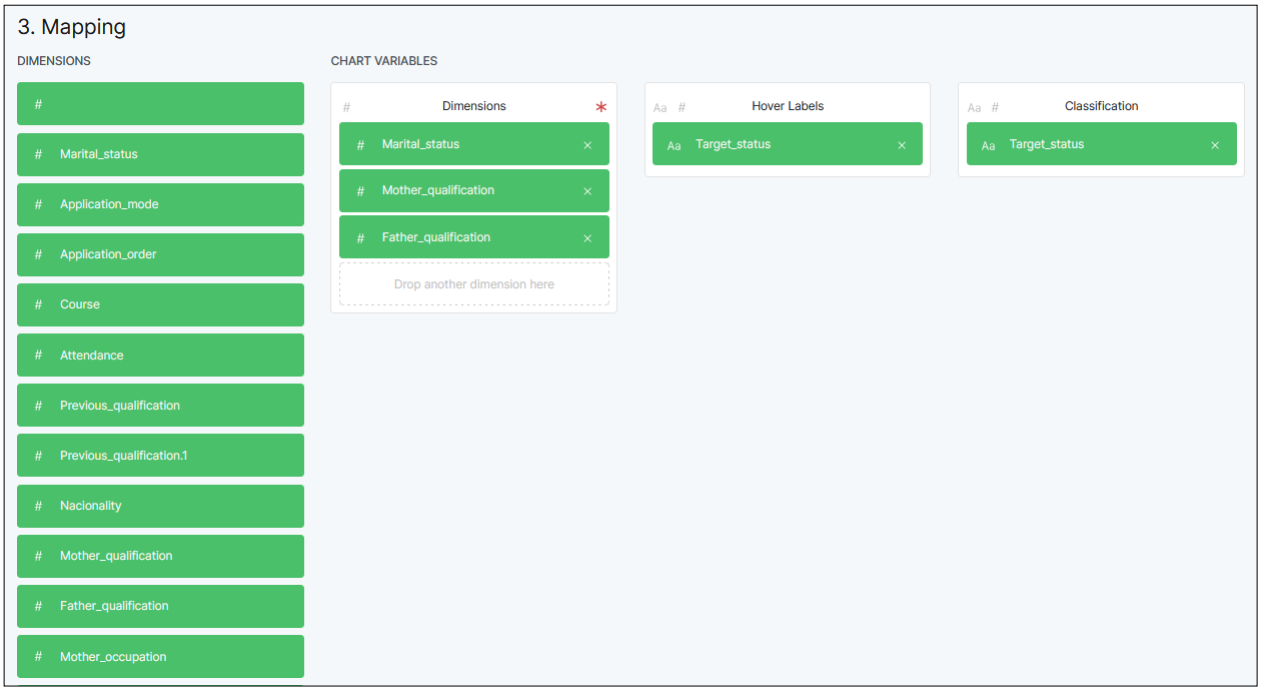

Figure 3.2: Step 3: Mapping data dimensions to chart variables.

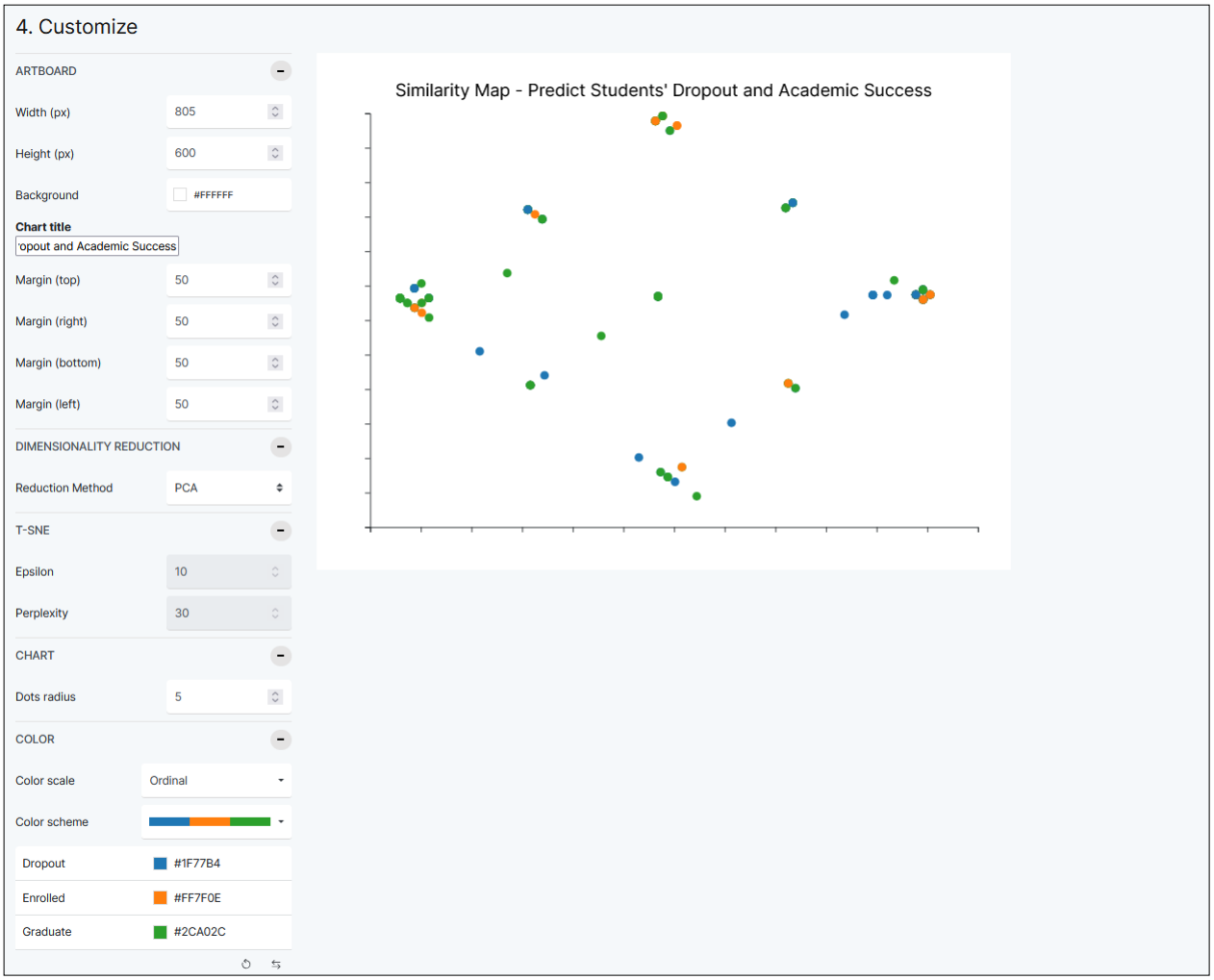

Figure 3.3: Step 4: Customising the Similarity Map with various options and settings.

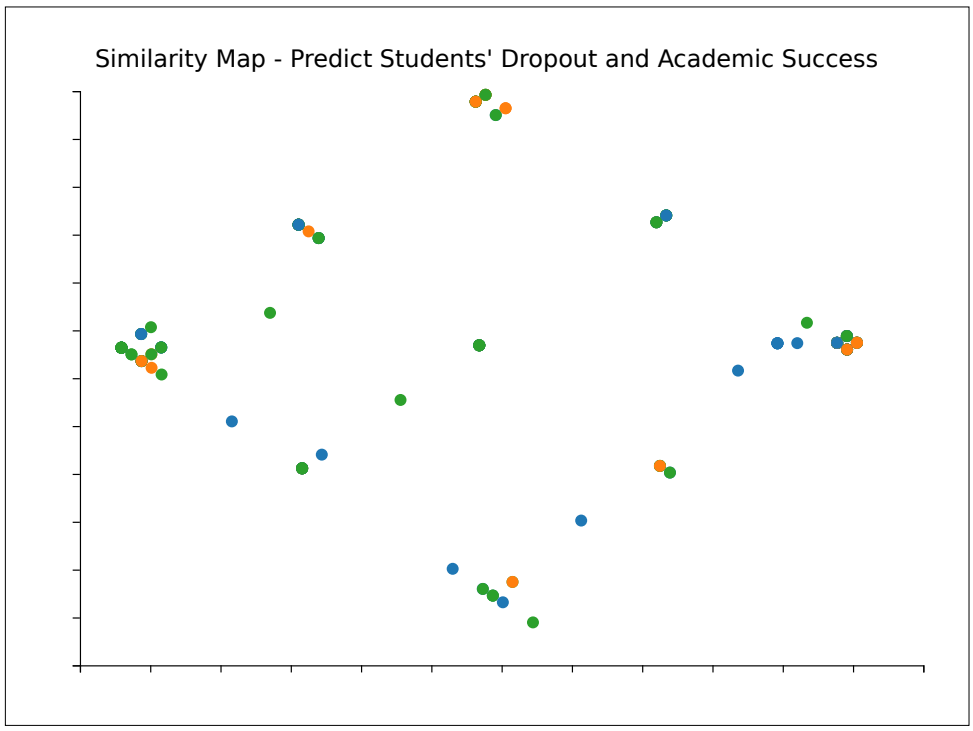

Figure 3.4: Step 5: The final Similarity Map as SVG.

Chapter 4

Connected Scatterplot

A connected scatterplot is a type of data visualization that combines elements of both scatterplots and line graphs. It is used to display the relationship between two continuous variables over time or another ordered sequence [Guillemot 2024a].

In a classic scatterplot, individual data points are plotted on a two-dimensional plane, where the x-axis represents one variable and the y-axis represents another. Each point represents a pair of values from the dataset. In a connected scatterplot, these points are connected by directional lines, usually in the order of occurrence or over time. This connection allows viewers to follow the progression of the data over time. A good example is the graphic by Fairfield [2012].

4.1 Data

The dataset used by Luc Guillemot in his connected scatterplot [Guillemot 2024b] plots migration rate against total population in Germany from 1950 to 2022. The underlying data is freely available from the United Nations under a Creative Commons License [UN 2024].

4.2 Implementation

When implementing a completely new RAWGraphs chart, it is often a good idea to begin with a similar chart type and then expand and modify the chart as needed. The default scatterplot of RAWGraphs was used as the starting template for the Connected Scatterplot. Thanks to this approach, a minimum viable prototype was created and then expanded upon step by step.

4.3 Example

The example shows the creation of a connected scatterplot for the German population growth dataset. Figure 4.1 shows the custom chart being selected, after previously being loaded into RAWGraphs. Figure 4.2 shows the customisation options and settings for the Connected Scatterplot chart. Figure 4.3 shows the final chart after exporting as SVG.

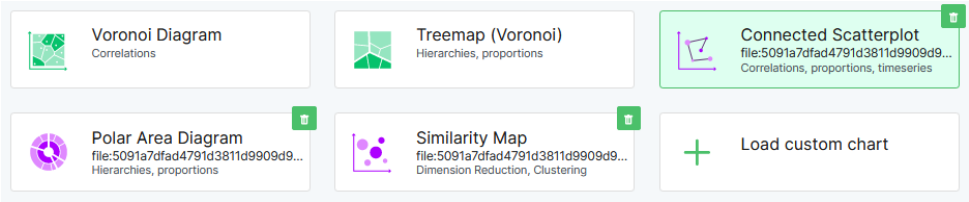

Figure 4.1: Choosing the Connected Scatterplot as a custom chart in RAWGraphs.

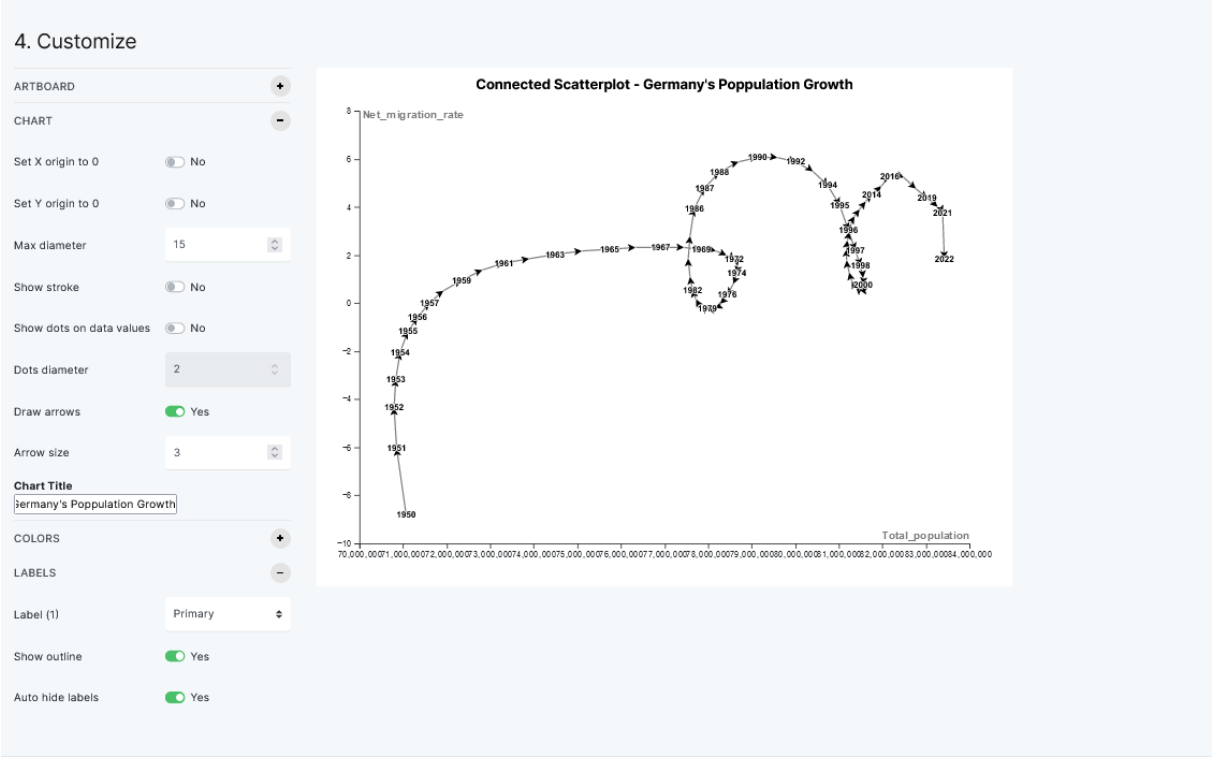

Figure 4.2: Customising the Connected Scatterplot with various options and settings.

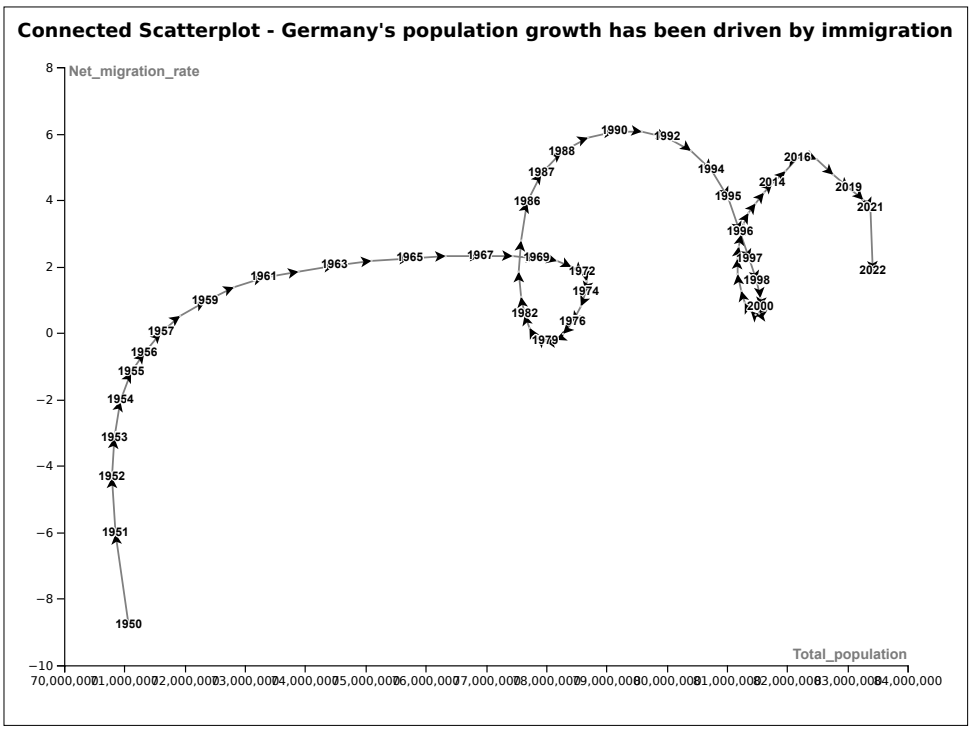

Figure 4.3: The final Connected Scatterplot as SVG.

Chapter 5

Polar Area Diagram

The final custom chart that was implemented is a prototype of a Polar Area Diagram. The Polar Area Diagram, also known as the Nightingale Rose Chart, is a circular graph used to plot cyclic phenomena. It is particularly useful for visualising multivariate data and showing changes over time.

5.1 Data

Since polar area charts are similar to pie charts, but each segment has the same angle and the area it takes up depends on the data, a matching dataset was very important.

The original dataset of Florence Nightingale was found on the internet but it did not have any licencing information associated with it so an additional dataset was aquired, the "Global Land Temperatures by Country" from Kaggle [Kaggle 2024].

5.2 Implementation

The starting point for the implementation of the Polar Area Diagram was the Sunburst Diagram of RAWGraphs, with some inspiration from two public CodePen examples [H3M4S 2018; Qiu 2023]. Since the Polar Area Diagram was the final chart to be tackled, it unfortunately remains only half finished. The segments are currently length-proportional rather than area-proportional, and no options or settings have been implemented.

5.3 Example

The example shows the creation of a prototype polar area diagram for the Global Land Temperatures dataset. Figure 5.1 shows the custom chart being selected, after previously being loaded into RAWGraphs. Figure 5.2 shows the customisation options and settings for the Polar Area Diagram. Figure 5.3 shows the final chart after exporting as SVG.

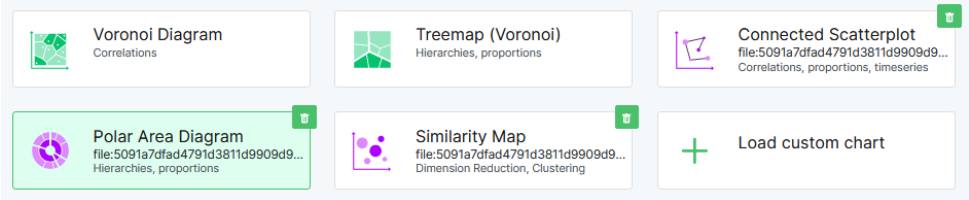

Figure 5.1: Selection of the prototype custom Polar Area Diagram.

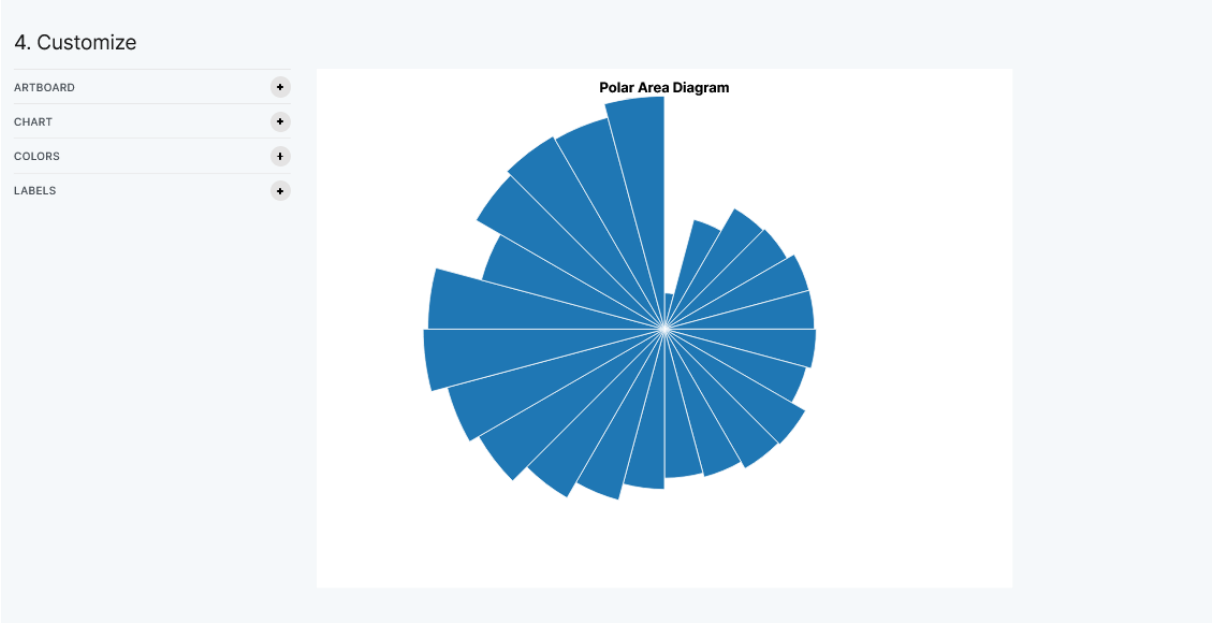

Figure 5.2: Customising the prototype Polar Area Diagram with various options and settings.

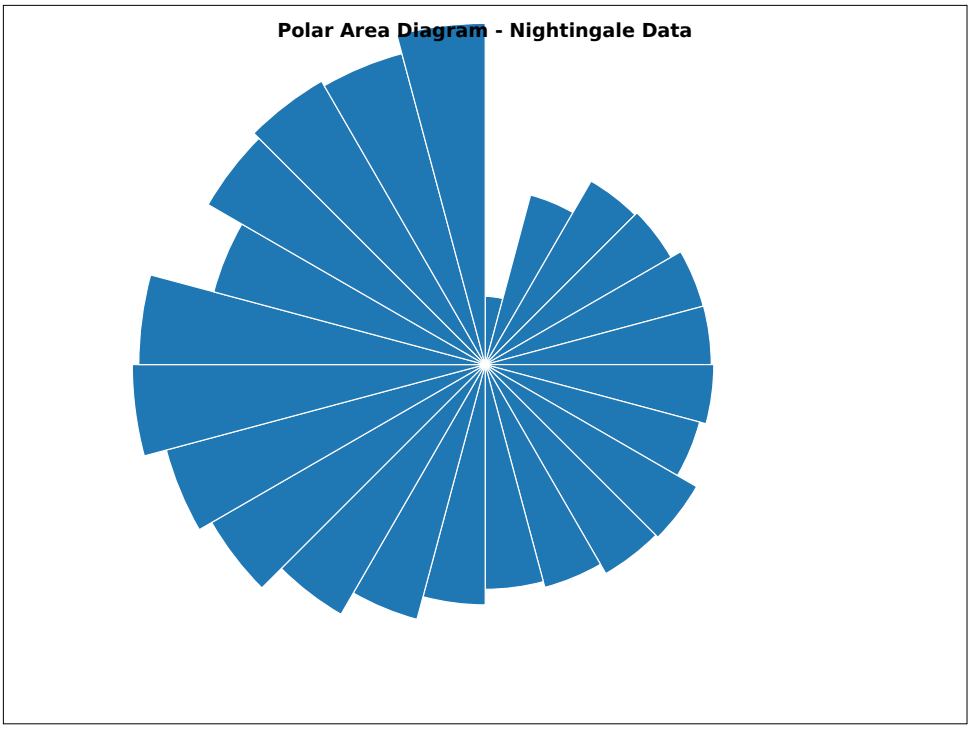

Figure 5.3: The final prototype Polar Area Diagram as SVG.

Chapter 6

Conclusion

In this project, we explored the use the popular open-source chart editor RAWGraphs to create custom charts. These tools are invaluable for non-programmers, providing a user-friendly interface that democratizes data visualization. The open-source nature of RAWGraphs not only allows users to utilize the tool but also to expand and modify the code to suit specific needs.

Setting up a local development server proved beneficial for self-hosting and further customization of RAWGraphs. This capability empowers users to add features and functionalities, enhancing the overall utility of the tool. However, working with RAWGraphs necessitates adherence to a strict custom structure, which can pose challenges.

One of the significant hurdles we encountered was the inadequacy of RAWGraphs' documentation. To address this, we created our own tutorial aimed at future developers, providing a more comprehensive guide than what is currently available. This was particularly important given that none of the project authors had prior experience with D3, the underlying library for RAWGraphs, resulting in a notable learning effort.

The project had a steep learning curve, but it was facilitated by the availability of useful and free data. This allowed us to focus more on mastering the tool rather than sourcing data. Despite the challenges, the process was enlightening and provided valuable insights into the potential and limitations of using RAWGraphs for data visualization.

Overall, our project highlighted the importance of accessible documentation and the benefits of open-source tools in educational and professional settings. We believe that our contributions, particularly the tutorial, will ease the path for future users and enhance their experience with RAWGraphs.

Bibliography

bitanath [2023]. *Principal Components Analysis in Javascript*. 12 Oct 2023. https://github.com/bitanath/pca (cited on page 15).

Bostock, Mike [2024]. *D3: Data-Driven Documents*. 03 Aug 2024. https://d3js.org/ (cited on page 1).

cannoneyed [2024]. *UMAP-JS*. 04 Jun 2024. https://github.com/PAIR-code/umap-js (cited on page 15).

DensityDesign [2023]. *RAWGraphs Charts GitHub Repository*. 13 Sep 2023. https://github.com/rawgraphs/rawgraphs-charts (cited on page 5).

DensityDesign [2024a]. *RAWGraphs*. 03 Aug 2024. https://rawgraphs.io/ (cited on page 1).

DensityDesign [2024b]. *RAWGraphs GitHub Repository*. 20 Feb 2024. https://github.com/rawgraphs (cited on page 1).

Doppelreiter, Elias, David Egger, Ludwig Reinhardt, and Stefan Schnutt [2022]. *Extending RAWGraphs*. Project Report. Graz University of Technology, 08 Jul 2022. https://courses.isds.tugraz.at/ivis/projects/ss2022/ivis-ss2022-g1-project-extending-rawgraphs.pdf (cited on page 15).

Fairfield, Hannah [2012]. *https://archive.nytimes.com/www.nytimes.com/interactive/2012/09/17/science/driving-safety-in-fits-and-starts.html*. New York Times, 17 Sep 2012 (cited on page 19).

Guillemot, Luc [2024a]. *Are Connected Scatterplots So Bad?* 31 Jan 2024. https://blog.datawrapper.de/connected-scatterplots/ (cited on page 19).

Guillemot, Luc [2024b]. *Connecting the Dots of Population Growth in Germany*. 18 Jan 2024. https://blog.datawrapper.de/germany-population-connected-scatterplot/ (cited on page 19).

H3M4S [2018]. *Nightingale Rose Chart*. 28 Oct 2018. https://codepen.io/H3M4S/pen/BqbYyw (cited on page 23).

Ivis-G1 [2024]. *RAWGraphs Custom-Charts*. 01 Jul 2024. https://github.com/solidth/RAWGraphs-Custom-Charts (cited on pages 1, 15).

Kaggle [2024]. *Global Land Temperatures by Country*. 01 Jul 2024. https://kaggle.com/datasets/vijayvvenkitesh/global-land-temperatures-by-country (cited on page 23).

karpathy [2016a]. *tSNEJS*. 22 Oct 2016. https://github.com/karpathy/tsnejs (cited on page 15).

karpathy [2016b]. *tSNEJS Demo*. 2016. https://cs.stanford.edu/people/karpathy/tsnejs/ (cited on page 15).

Mauri, Michele, Tommaso Elli, Giorgio Caviglia, Giorgio Uboldi, and Matteo Azzi [2017]. *RAWGraphs: A Visualisation Platform to Create Open Outputs*. Proc. 12th Biannual Conference of Italian SIGCHI Chapter (CHItaly 2017) (Cagliari, Italy). 28. ACM, 18 Sep 2017. doi:10.1145/3125571.3125585 (cited on page 1).

Meta [2024]. *React*. 03 Aug 2024. https://react.dev/ (cited on page 1).

Qiu, Xiaoqi [2023]. *Nightingale Chart – Apache ECharts Demo*. 07 Oct 2023. https://codepen.io/XIAOQI-QIU/pen/zYyyGjN (cited on page 23).

Schweiger, Manuel, Reinhold Seiss, Maximilian Tauß, and Stefan Tscheppe [2018]. *Similarity Map for RAW Graphs*. Project Report. Graz University of Technology, 02 Jul 2018. https://courses.isds.tugraz.at/ivis/projects/ss2018/ivis-ss2018-g2-project-raw-simmap.pdf (cited on page 15).

UCI [2024a]. *Predict Students' Dropout and Academic Success*. UC Irvine. 01 Jul 2024. https://archive.ics.uci.edu/dataset/697/ (cited on page 15).

UCI [2024b]. *UC Irvine Machine Learning Repository*. 01 Jul 2024. https://archive.ics.uci.edu/ (cited on page 15).

UN [2024]. *World Population Prospects*. United Nations. 2024. https://population.un.org/wpp/Download/Standard/Population/ (cited on page 19).