Open-Source Chart Editors

Comparing Visual and Grammar-Based Tools in Information Visualization

Elias Doppelreiter, David Egger, Ludwig Reinhardt, Stefan Schnutt

706.057 Information Visualisation 3VU SS 2022
Graz University of Technology

24 May 2022

Abstract

This document contains an overview of contemporary open source chart editors. Each chart editor is represented by a detailed description alongside three sample graphs to showcase the individual visualisation characteristics. Deciding on the right visualisation tool can greatly enhance the workflow. The intention of this survey is to assist the reader in finding the most suitable open source chart editor for different fields of application.

Contents

Contents **ii**

List of Figures **iv**

List of Tables **v**

1 Introduction **1**

1.1 Excluded Chart Editors . . . 1

1.1.1 Datawrapper . . . 1

1.1.2 Q . . . 2

1.1.3 Charted . . . 2

1.2 Chart Types and Datasets . . . 3

2 RAWGraphs **5**

2.1 Technical Details . . . 5

2.1.1 Installation . . . 5

2.1.2 Input and Output Possibilities . . . 5

2.1.3 Architecture . . . 6

2.1.4 Intelligent Features . . . 6

2.2 User Interface . . . 6

2.3 Sample Graphs . . . 7

2.4 Advantages and Disadvantages . . . 9

2.5 Conclusion . . . 9

3 Vega Editor **11**

3.1 Introduction . . . 11

3.2 Technical Details . . . 11

3.2.1 Installation and Execution . . . 11

3.2.2 Alternative Local Execution with Vega Desktop . . . 11

3.2.3 Architecture . . . 12

3.2.4 Supported Input and Output . . . 12

3.2.4.1 Input . . . 12

3.2.4.2 Output . . . 13

3.3 User Interface . . . 13

3.4 Sample Graphs . . . 14

3.5 Conclusion . . . 14

4 Voyager **17**
4.1 Introduction 17
4.2 Technical Details 17
4.2.1 Installation and Execution 17
4.2.2 Architecture 17
4.2.3 Supported Input and Output 17
4.2.3.1 Input 17
4.2.3.2 Output 17
4.3 User Interface 18
4.4 Workflow 18
4.5 Sample Graphs 19
4.6 Conclusion 21

5 Dex **23**
5.1 Technical Details 23
5.1.1 Installation 23
5.1.2 Architecture 23
5.1.3 Input Possibilities 23
5.1.4 Output Possibilities 24
5.1.5 Intelligent Features 24
5.2 User Interface 24
5.3 Workflow 24
5.4 Sample Graphs 27
5.5 Conclusion 27

6 Liquid Diagrams **29**
6.1 Installation Process 29
6.2 User Interface 29
6.3 Sample Graphs 34
6.4 Software Features 36
6.5 Conclusion 36

7 Comparison of Chart Editors **37**

8 Concluding Remarks **39**

Bibliography **41**

List of Figures

1.1 Charted: Incorrect output. 3

2.1 RAWGraphs: Example of the user-interface. 7
2.2 RAWGraphs: Bar graph sample chart. 8
2.3 RAWGraphs: Line graph sample chart. 8
2.4 RAWGraphs: Scatter plot sample chart. 9

3.1 Vega Desktop: Example of the user interface. 12
3.2 Vega Editor: Example of the user interface.. 13
3.3 Vega Editor: Left side of the taskbar. 13
3.4 Vega Editor: Right side of the taskbar.. 14
3.5 Vega Editor: Bar Charts Created with Vega and Vega-Lite 15
3.6 Vega Editor: Line Charts Created with Vega and Vega-Lite. 15
3.7 Vega Editor: Scatter Plots Created with Vega and Vega-Lite. 15

4.1 Voyager: Example of the user interface. 18
4.2 Voyager: File Upload Window 19
4.3 Voyager: Bookmarks Window 19
4.4 Voyager: Bar chart. 20
4.5 Voyager: Line Chart 20
4.6 Voyager: Scatter Plot 20

5.1 Dex: Importing Data 25
5.2 Dex: Viewing the Imported Data 26
5.3 Dex: JavaFX Bar Chart Configuration. 26
5.4 Dex: Bar Charts. 28
5.5 Dex: Line Charts 28
5.6 Dex: Scatter Plots 28

6.1 Liquid Diagrams: Main user interface. 30
6.2 Liquid Diagrams: Initial chart editor window. 30
6.3 Liquid Diagrams: Data import 31
6.4 Liquid Diagrams: Initial bar chart 32
6.5 Liquid Diagrams: General chart options 32
6.6 Liquid Diagrams: Fonts chart options 33
6.7 Liquid Diagrams: Axes chart options 33
6.8 Liquid Diagrams: Data view 34

6.9 Liquid Diagrams: Sample Bar Chart 35
6.10 Liquid Diagrams: Sample Line Chart 35

List of Tables

1.1 City of Graz Election Results 2021 4
1.2 Student Enrollment at Graz University of Technology 4
1.3 CO2 Emissions and Population by Country 4

7.1 Comparison of Chart Editors 38

Chapter 1

Introduction

Living in the digital age in which the amount of available information and data on the internet grows in an unprecedented manner, the field of information visualization is growing in regards of its importance. By using proper information visualization techniques, people are better able to understand and analyze such information. In order to create informative diagrams or charts, practitioners in the field of information visualization often rely on chart editors to create their visualizations.

Therefore, this survey paper provides an overview of various open-source chart editors to be compared against each other based of predefined characteristics. The selection of chart editors is further restricted by only considering open-source solutions. Open-source refers to software that is publicly available and free for or use, modification, or extension.

In terms of chart editors, having an open-source solution becomes especially important when users have to deal with highly sensitive or personal data they do not want to upload to some server which happens as a consequence of using web-hosted solutions. Usually, having access to the source code allows users with according technical affinity to run the chart editor locally.

First, in this chapter, other open-source chart editor candidates, which did not qualify to be included in the survey, are shortly introduced. Next, common chart types and data sets are selected to be used throughout the survey, before selected chart editors are reviewed in their respective chapters. Lastly, a summarizing comparison of the selected chart editors is provided in tabular form to give a better overview of the survey results.

1.1 Excluded Chart Editors

This section gives a short overview of excluded chart editors which were not included in the survey. Details on why these chart editors were excluded from the survey are provided in the following respective subsections.

1.1.1 Datawrapper

Datawrapper is a web-hosted chart editor which provides a very good user experience. Creating charts is a straightforward process consisting of four steps: (1) Upload Data, (2) Check & Describe, (3) Visualize, and (4) Publish & Embed.

In the first step, users are able to import data to be used for charts. Besides CSV files, Datawrapper also supports XML files for input. In addition, Datawrapper supports copy and paste from table cells from Microsoft Excel and LibreOffice Calc and supports links to Google Sheets and other external data sources.

In the next step, Datawrapper presents a live data preview of the imported data set. Here, users are able to adjust the data formats for individual columns and adjust cell entries to correct data if necessary. Transposing the data and adding or hiding columns is also supported.

Having confirmed the chart data, users are able to interactively configure charts in the third step. Datawrapper provides a selection of 20 different chart types to choose from. Using different tabs or registers in the user interface, users are able to intuitively set numerous chart settings.

Unfortunately, chart export is restricted to registered users. After successful registration, user are able to download their charts in PNG format. Other formats such as Scalable Vector Graphics (SVG) are only available for paying customers.

Although Datawrapper is an excellent visualization tool, it was excluded from this survey, because only the application's core is open-source. The attempt to compile Datawrapper locally has failed. An email request to the team at Datawrapper clarified that Datawrapper is not intended to be run locally.

1.1.2 Q

Q is a web-based application which allows the creation of a wide range of visualizations [NZZ 2022d; NZZ 2022g]. Q is developed by NZZ Editorial Tech [NZZ 2022b] and NZZ Visuals [NZZ 2022c], which both belong to the Neue Zürcher Zeitung (NZZ) [NZZ 2022a]. Q consists of several individual components [NZZ 2022d], which can be divided into three areas:

- Q Server [NZZ 2022f].
- Q Editor [NZZ 2022e].
- Tools: such as Q-Chart [NZZ 2022g] and Q Election Seats [NZZ 2021].

Unfortunately, we could not install Q locally, therefore we could only evaluate the online demo version and limited ourselves to Q-Chart [NZZ 2022g]. With Q-Chart it is possible to create bar charts, line charts, and scatterplots, among others. Visualization data can be imported in a table format, it is possible to copy data from a spreadsheet program to the Q-Chart table. Subsequently, one of the available visualization types can be selected, and depending on the selected visualization type, different configuration options are available. In the tested online demo version [NZZ 2022d], there are unfortunately no possibilities to export the created charts. However, various export options such as PNG should be available in the version for developers.

1.1.3 Charted

Locally installed, this software was not able to produce usable charts, the layout of the website that presents the visualization is broken. The official website, which is suggested for use, is not reachable.

Charted was created by the product science team at the company Medium [2022]. Medium is a company which offers a platform to readers and writers of stories. The focus of Charted is two things, simplicity and automation. To use Charted, everything the user has to do is providing a link pointing to the address of one of the supported data types and Charted will visualize this data in a chart. The chart type will either be a line chart or a stacked column chart and there are little to no options to experiment with.

The concept of Charted is based on three main ideas. The first idea is that Charted is not supposed to store any data. All Charted does is simply calling data and visualising it and doing that automatically and repeatedly every 30 minutes. Main idea number two is that the data is taken as it is, no modification is done to the data by Charted. The user is responsible to deliver data in exactly the right format to Charted. If the user wants calculations or modifications to the data the user has to do this beforehand by himself. The third main idea states that charted is not a formatting tool. There is a minimum amount of features, to ensure that the data becomes the visualisation as soon as possible.

The authors failed to produce representative sample graphs. Charts created with Charted look broken and are incoherent as seen in Figure 1.1.

Charted represents a simple unique tool, geared down for one special job. Calling it a chart editor might not be the right category. The authors imagine that Charted can be a potential visual enhancement

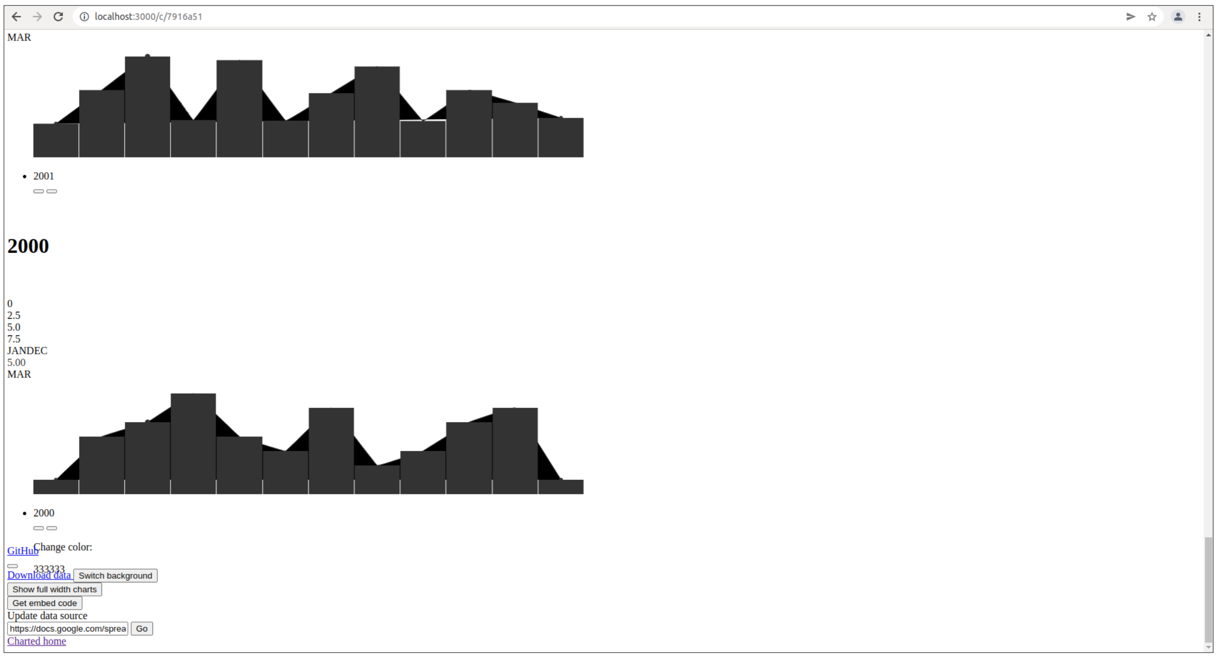


Figure 1.1: Example of incorrect charts produced by Charted. Notice the layout, captions and user-interface. [Screenshot was taken by the authors of this paper.]

tool if used in the right context. Given the limited documentation, help and instructions and Charteds limited use-cases, the authors conclude that this software was originally never intended for public use.

1.2 Chart Types and Datasets

To provide a baseline for comparision, each chart editor in this survey was used to create each of three charts:

1. A bar chart of election results from the City of Graz Election in 2021 [Graz 2021]. The dataset is shown in Table 1.1.

2. A line chart of student enrollment at Graz University of Technology from 2015 to 2021 [TUG 2021]. The dataset is shown in Table 1.2.

3. A scatter plot of CO2 emissions and population by country [EC 2021; World Bank 2020]. The dataset is shown in Table 1.3.

The chart types bar graph, line graph and scatter plot were selected by the authors for their regularity and simplicity. It is expected that most open source chart editor are able to create these fundamental types of charts.

Having a common selection of chart types using the same underlying data when inspecting selected open-source chart editors allows an easier comparison of the created charts. The charts can be found at each editors section respectively.

Party	Percent
KPÖ	28.8%
ÖVP	25.9%
GRÜNE	17.3%
FPÖ	10.6%
SPÖ	9.5%
NEOS	5.4%
BASIS	0.9%

Table 1.1: Election results from the City of Graz Election 2021 [Graz 2021].

Year	Students
2015	13167
2016	13454
2017	13737
2018	13373
2019	13566
2020	13673
2021	13712

Table 1.2: Student enrollment at Graz University of Technology from 2015 to 2011 [TUG 2021].

Country	Total CO2 in Mt	Population in M
China	11680.42	1402.00
United States	4535.30	329.50
India	2411.73	1380.00
Russia	1674.23	144.10
Japan	1 061.77	125.80
Iran	690.24	83.99
Germany	636.88	83.24
South Korea	621.47	51.78
Saudi Arabia	588.81	34.81
Indonesia	568.27	273.50

Table 1.3: CO2 emissions and population by country [EC 2021; World Bank 2020].

Chapter 2

RAWGraphs

RAWGraphs was initially created in Milan, Italy, the first Git commit dates back to July 2013 [DensityDesign 2022a]. Responsible for the development and design is the team from the DensityDesign Research Lab at the Politecnico di Milano, consisting of Girogio Caviglia, Michele Mauri, Matteo Azzi, Giorgio Uboldi and Tommaso Elli [Mauri et al. 2017; DensityDesign 2022c; DensityDesign 2022b]. The current software license of RAWGraphs legally binds users to the Apache 2 license.

The vision of RAWGraphs is to provide a connection between vector graphic programs like Adobe Illustrator or Sketch and spreadsheet applications like Excel, Google Sheets or Calc. RAWGraphs can be used to create a number of static chart types. RAWGraphs promises privacy to sensible data if RAWGraph is executed locally on the users PC as well when running RAWGraphs from the public website. The target group of users include designers and visualization geeks hence a variety of partially unconventional chart types are available. Expert users can extend the program and code their own visual representations or charts by using the D3 library and a local installation of RAWGraphs.

2.1 Technical Details

This section covers the technical characteristics of RAWGraphs.

2.1.1 Installation

There are two common ways to get RAWGraphs running. The first and simple way requires the user to open his web browser and navigate to the public website. Building charts from the public website guarantees privacy for all of the users data. One drawback of this method is that the user can't code his own visualizations or charts.

The second way to run the software is by installing RAWGraphs locally, this is for intermediate users. This is done with the help of Git, Node.js, and Yarn. The user clones the official Git repository, installs the required packages, and starts the program with Yarn. Once the service is running, the user may start his browser and open the localhost:3000 port to communicate with RAWGraphs locally. For experts it is also possible to build and upload RAWGraphs to a server.

2.1.2 Input and Output Possibilities

To input the users data, the easiest and probably most popular way, is to drag and drop the users tabular-data file into the user interface. RAWGraphs will parse the values and present a handy preview of the data to the user. Beside the drag-and-drop approach, there are options for pasting the users data, using SPARQL queries, fetching data from a remote URL or importing a proprietary RAWGraphs-project. Data formats supported are TSV, CSV, DSV or JSON. For now it is not possible to import XML. The authors tested JSON import and it isworking fine. Additionally a wide variety of data samples are provided by RAWGraphs. This sample data actually gets imported from kaggle.

Many options are available to further customize the users input data, this includes adjusting the thousands separator, the decimals separator, the date locale and transform the data to stack by different columns.

RAWGraphs output format include SVG, PNG, JPG as well as the proprietary format .rawgraphs. This enables the users to save their progress and return to it at a later time. SVG graphs are scalable, PNG or JPG are not. It is not possible to create or export animated graphs in RAWGraphs.

2.1.3 Architecture

RAWGraphs is built on top of the D3 library. The user interface is built with React. Other libraries and frameworks that make an appearance are Bootstrap, AngularJS, Bower, Canvas to Blob, CodeMirror, FileSaver.js, is.js, JQuery, NG file upload, Sheet JS and ZeroClipboard. The offical Git repositroy reports that the project consists of 77.3% JavaScript, 13.3% HTML and 9.4% CSS.

2.1.4 Intelligent Features

True intelligent features are not prevalent in RAWGraphs. There are no charts suggested to the user after data insertion. Some charts come out pretty messed up from the automatic initial creation and need manual readjustment to reach an acceptable quality. For example, labels become unreadable, dimensions are inappropriate, or colours are non-existent.

RAWGraphs is not suggesting plot dimensions it rather enforces them. Plot dimensions are scaled automatically which can lead to bad results. For example, when creating a scatter plot (bubble plot) RAWGraphs crops the graphs boundaries to match exactly the extreme mathematical values of the users input data. Sometimes the user doesn't want this because displaying a specific range is of interest or data points require more room to breath for ground line aesthetics.

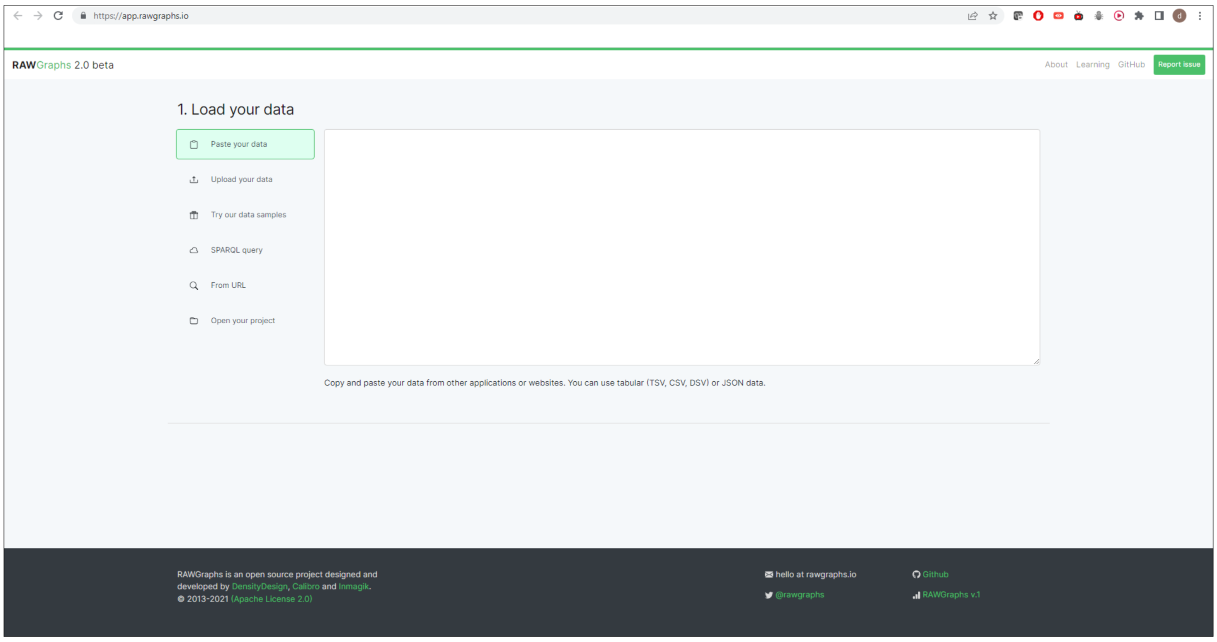
There is no adjustment of the software to the user either. RAWGraphs doesn't remember which chart the user tends to prefer or what kind of settings the user usually takes. It is not possible to log into the application.

2.2 User Interface

The user interface is exceptionally well designed, as can be seen in Figure 2.1. When the authors tested this chart editor, at no point there was the feeling of being lost. The design is intuitive, the use of colours is decent. Only one highly saturated green colour is used to give the user directions and help him navigate without thinking. The user is taken by the hand when creating a chart. The whole process is split up in five steps and only after the user finishes his momentarily step, the succeeding step is revealed. This automatically focuses the user attention to where it should be.

At Step Three, which is a crucial step where the user has many options which can lead to errors, RAWGraphs gives immediate feedback about what and where the problem is. Tooltips provide additional information to icons. When customizing the appearance in Step Four, values that are not adjustable are greyed out. The user can select his own custom colours via a colour picker popup. Unfortunately there is no pipette or the ability to save the users palettes.

It is definitely worth mentioning that the documentation of RAWGraphs is solid. Most of our questions of different depth could be answered with ease. The installation manual is short and concise. For almost every diagram there is a link from the user-interface to a Youtube video that showcases this diagrams creation process, as well another link that points to the contextual code on the Git repository. The API is well documented and even a scientific paper is shared that gives insight at an academic level.



Copy and paste your data from other applications or websites. You can use tabular (TSV, CSV, DSV) or JSON data.

RAWGraphs is an open source project designed and developed by DensityDesign, Calibro and Inmagik.
© 2013-2021 (Apache License 2.0)

Figure 2.1: The landing page of RAWGraphs. [Screenshot was taken by the authors of this paper.]

2.3 Sample Graphs

The three sample graphs were produced with RAWGraphs:

- The sample bar graph represented in Figure 2.2.
- The sample line graph represented in Figure 2.3.
- The sample scatter plot represented in Figure 2.4.

The bar graph is uses an ordinal color scheme. For the line chart only the bottom margin was slightly increased. Several modifications were applied to the scatter plot. The labels for the data points in the bottom left corner were overlapping which rendered this part of the chart unreadable. RAWGraph provides an option to blend out overlapping labels. Also the color of the circles was brightened up to make it easier to read.

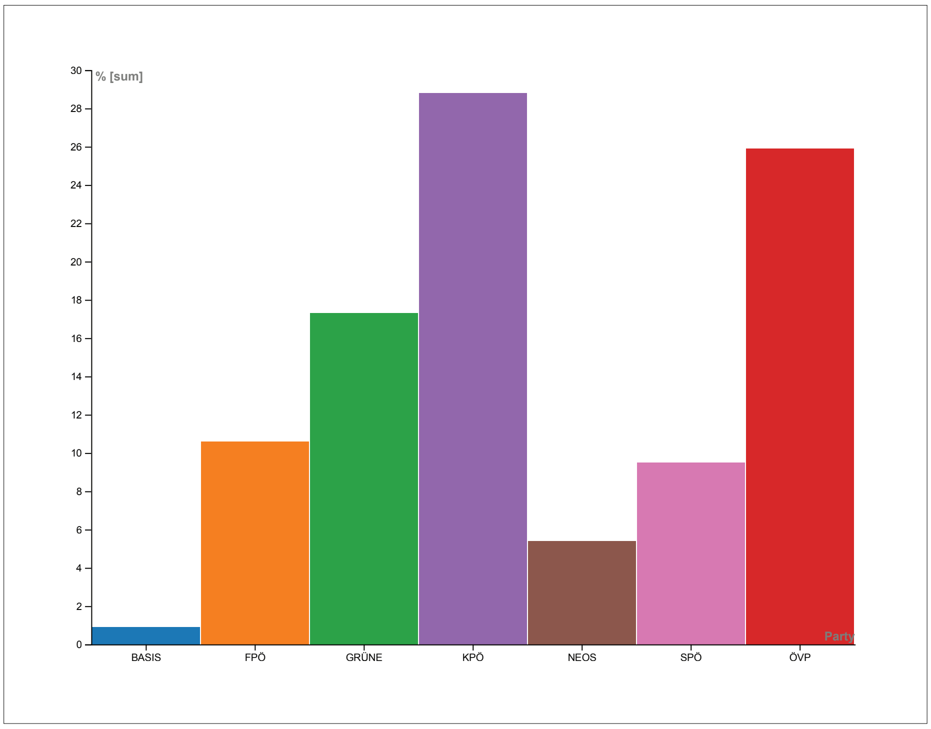


Figure 2.2: The bar graph sample chart made with RAWGraphs. [Screenshot was taken by the authors of this paper.]

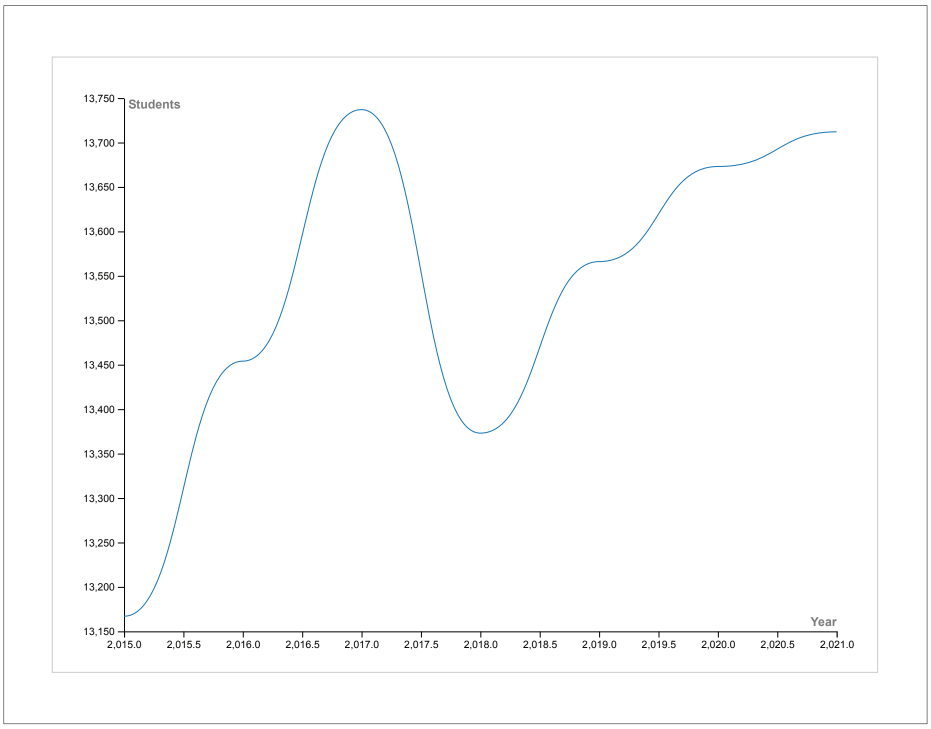


Figure 2.3: The line graph sample chart made with RAWGraphs. [Screenshot was taken by the authors of this paper.]

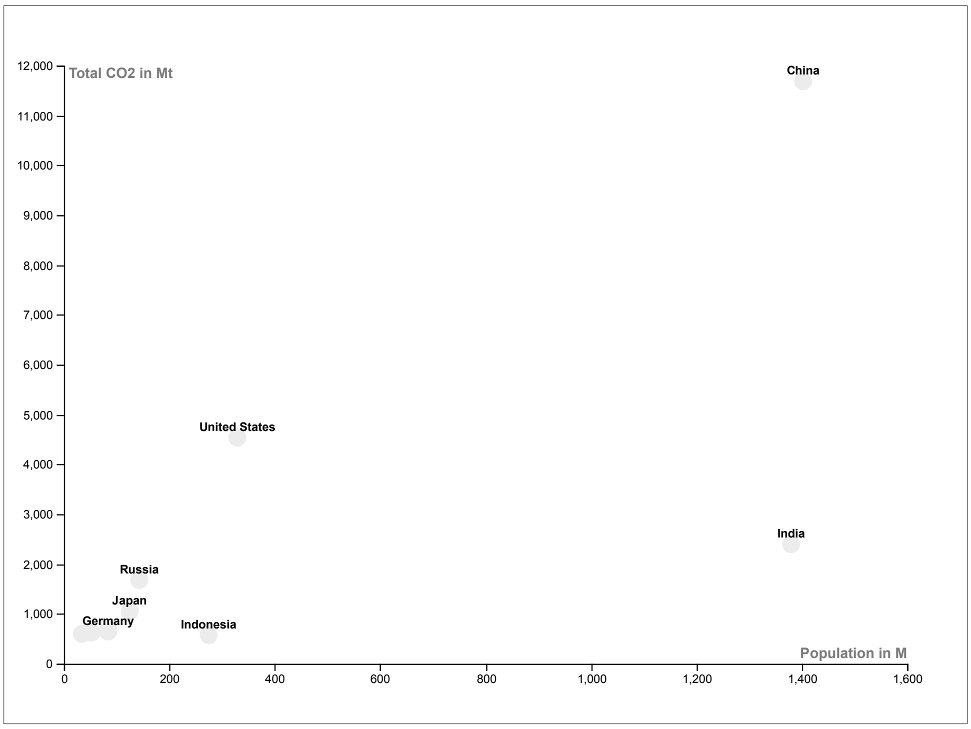


Figure 2.4: The scatter plot sample chart made with RAWGraphs. [Screenshot was taken by the authors of this paper.]

2.4 Advantages and Disadvantages

Advantages of RAWGraphs include its UX, privacy for both the official website, wide variety of graphs including non-conventional, good volume of input and output data formats and that the software is also interesting for power users. Disadvantages are that installing RAWGraphs locally requires intermediate computational skills and that the charts are not fully customisable. For example, it is not possible for the user to define a custom range for the axis of charts, although it is possible to include zero. Another problem, especially for power users, is that there is no feedback on the maximum amount of data that can be processed by RAWGraphs. If the user happens to exceed the system or browser limitations, RAWGraphs will just crash. One thing the authors would love to see is an undo function where the user can step back previous inputs in case of mistakes.

2.5 Conclusion

RAWGraphs has some years of love and experience behind its back and the authors can certainly feel and see that. At one point, RAWGraphs got crowdfunded to fuel further development. Although not perfect, when compared to other open source chart editors, RAWGraphs earns its place among the top ranks featuring exquisite user experience, an above average number of chart types, privacy and options to serve a broader spectrum of users from the casuals to the visualisation veterans.

Chapter 3

Vega Editor

3.1 Introduction

The Vega Editor is a grammar-based open source chart editor [IDL 2022c] and part of the Vega project, which is led by the University of Washington Interactive Data Lab. The goal behind the project is to provide various tools which are building on top of each other. The basis of these tools and libraries is the Vega specification grammar [IDL 2022a].

While the main objective of the specification grammar is to provide a comfortable way to build higher-level software tools on top of it, it is also possible to directly code the specification in order to retrieve a chart. This is enabled by the Vega Editor which compiles files written in the defined specification vocabulary and renders the result onto the screen. If the result should then be rendered in SVG or HTML Canvas is chosen by the user. Determined by the adjusted settings the user is able to live edit charts which makes the tool very comfortable for experienced developers. The following work is for the most part about the Vega Editor and partly about the declarative visualization grammars Vega and Vega-Lite. These two differ in the aspect that Vega-Lite is a subset of Vega, therefore everything that is written in Vega-Lite can be compiled to Vega, but not the other way around. Vega-Lite on the other hand has the advantage that visualizations can be specified in more understandable, fewer lines that take out some of the complexity of the lower Vega specification [Satyanarayan et al. 2016].

3.2 Technical Details

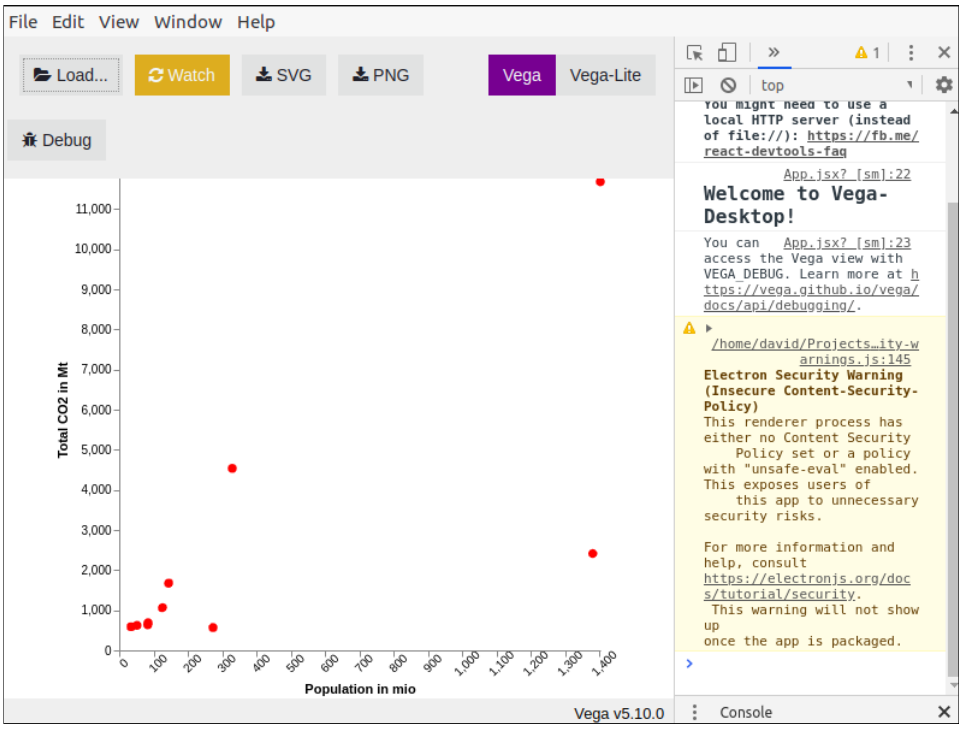
This section covers the technical characteristics of the Vega Editor.

3.2.1 Installation and Execution

Working remotely with the deployed version of the tool is quite simple and can easily be done by visiting [IDL 2022d]. However, if one wants to work with Vega specification without hard coding the data into the specification file, but instead be able to import it, one has to clone the source code from [IDL 2022c], navigate into the root directory of the project, run yarn to install the dependencies and then run yarn start to start the editor locally. The needed CSV files must be put into the public/data folder of the project to be accessible from the started editor.

3.2.2 Alternative Local Execution with Vega Desktop

The usage of Vega Desktop in combination with a text editor of choice is one alternative to locally executing the Vega Editor. Vega Desktop is a reader for Vega and Vega-Lite specifications and can watch a specification file and live-update the resulting chart. The advantage of Vega Desktop is that many developers have their own preferred text editor which contains the preferred settings and would most likely be used for everything if it were possible. The disadvantage of the tool is that it is no longer



```
You might need to use a
local HTTP server (instead
of file://): https://fb.me/
react-devtools-faq
                App.jsx? [sm]:22
Welcome to Vega-
Desktop!
You can   App.jsx? [sm]:23
access the Vega view with
VEGA_DEBUG. Learn more at h
ttps://vega.github.io/vega/
docs/api/debugging/.
 /home/david/Projects…ity-w
           arnings.js:145
Electron Security Warning
(Insecure Content-Security-
Policy)
This renderer process has
either no Content Security
    Policy set or a policy
with "unsafe-eval" enabled.
This exposes users of
    this app to unnecessary
security risks.

For more information and
help, consult
https://electronjs.org/doc
s/tutorial/security.
 This warning will not show
up
once the app is packaged.
```

Figure 3.1: The user interface of Vega Desktop. [Screenshot was taken by the authors of this paper.]

supported by the creators and has problems with compiling some parts of Vega-Lite specifications. The UI of Vega Desktop can be seen in Figure 3.1. The source code to the alternative tool is under MIT license and can be found at [IDL 2022b].

3.2.3 Architecture

According to the Github repositories, the following holds for the three tools:

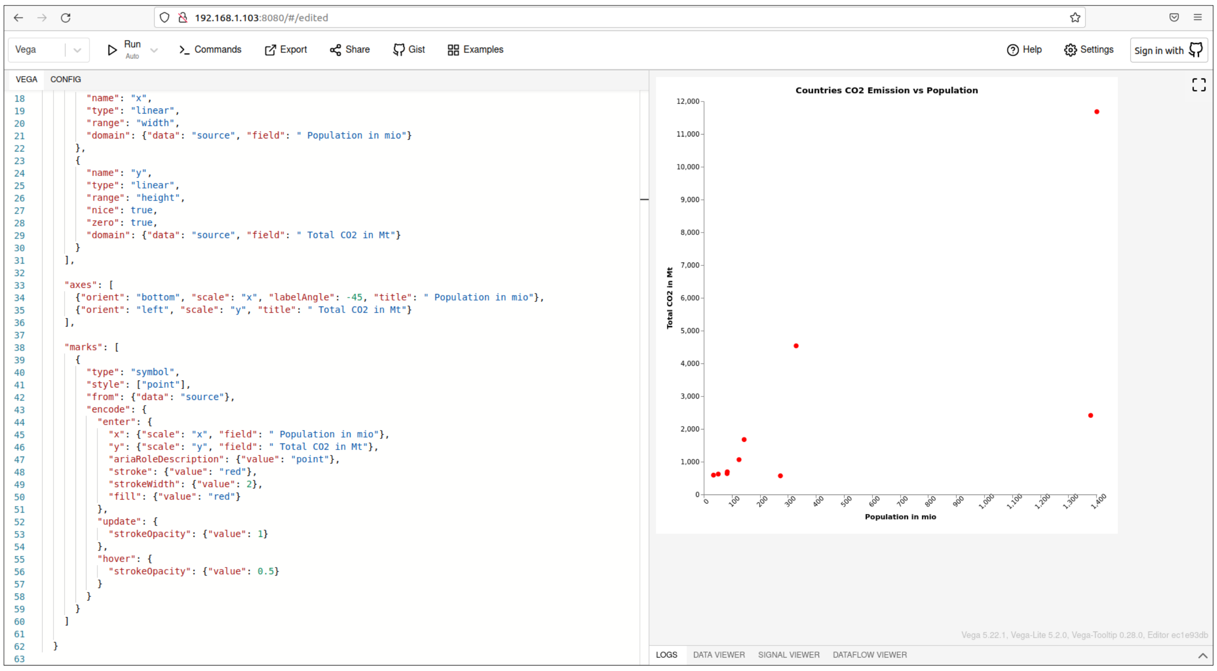
- Vega: Vega is put under BSD-3-Clause license and consists of: 73.6% JavaScript, 25.2% TypeScript, 1.1% HTML and 0.1% Shell script. Its repository can be found at [IDL 2022e].
- Vega-Lite: Vega-Lite is put under the BSD-3-Clause license and consists of: 96.3% TypeScript. Its repository can be found at [IDL 2022f].
- Vega Editor: The Vega-Editor is put under the BSD-3-Clause license and consists of: 88.8% TypeScript and 8.7% CSS. Its repository can be found at [IDL 2022c]. The hosted web application can be found at [IDL 2022d].

3.2.4 Supported Input and Output

3.2.4.1 Input

The deployed version of the editor does provide various example specifications as well as numerous CSV files which can be referenced directly in the specification. A specification file must be in JSON format and is required to have the file ending .vg.json.

For importing your own CSV data without having to hard code it into the specification file, it is necessary to clone the source code and run the tool locally. The steps to clone, install and execute the editor locally are described in Section 3.2.1.



```
        "name": "x",
        "type": "linear",
        "range": "width",
        "domain": {"data": "source", "field": " Population in mio"}
      },
      {
        "name": "y",
        "type": "linear",
        "range": "height",
        "nice": true,
        "zero": true,
        "domain": {"data": "source", "field": " Total CO2 in Mt"}
      }
    ],

    "axes": [
      {"orient": "bottom", "scale": "x", "labelAngle": -45, "title": " Population in mio"},
      {"orient": "left", "scale": "y", "title": " Total CO2 in Mt"}
    ],

    "marks": [
      {
        "type": "symbol",
        "style": ["point"],
        "from": {"data": "source"},
        "encode": {
          "enter": {
            "x": {"scale": "x", "field": " Population in mio"},
            "y": {"scale": "y", "field": " Total CO2 in Mt"},
            "ariaRoleDescription": {"value": "point"},
            "stroke": {"value": "red"},
            "strokeWidth": {"value": 2},
            "fill": {"value": "red"}
          },
          "update": {
            "strokeOpacity": {"value": 1}
          },
          "hover": {
            "strokeOpacity": {"value": 0.5}
          }
        }
      }
    ]

  }
```

Figure 3.2: The user interface of the Vega Editor. [Screenshot was taken by the authors of this paper.]

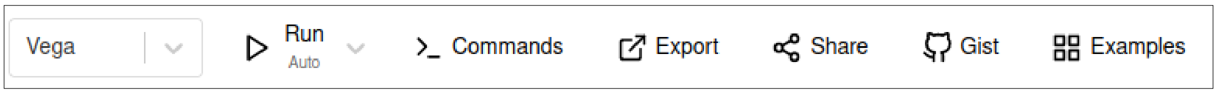


Figure 3.3: The left side of the Vega Editor taskbar. [Screenshot was taken by the authors of this paper.]

3.2.4.2 Output

The output format options for created charts are PNG, SVG, JSON, PDF, and HTML.

3.3 User Interface

The user interface of the Vega Editor is kept quite simple and straightforward. It is divided into three parts, as can be seen in Figure 3.2:

- The taskbar provides most of the additional functionality. The left side of the taskbar, shown enlarged in Figure 3.3, displays options to switch the selected specification kind, start the compiling and rendering process, settle custom keyboard shortcuts, open the export window, share the code via link or gist and load predefined examples. The right side of the taskbar, shown enlarged in Figure 3.4, offers options to sign in via Github, get help, and change one's settings.

- Coding Area: This is the part of the UI where the Code is displayed. When changing code here it gets immediately displayed in the rendering part of the Output Area. To make the life of developers easier, code completion is activated here.

- Output Area: In this part of the UI the specified chart, as well as any logging information like errors, are displayed. There are additional tabs for inspecting the loaded data.

Figure 3.4: The right side of the Vega Editor taskbar. [Screenshot was taken by the authors of this paper.]

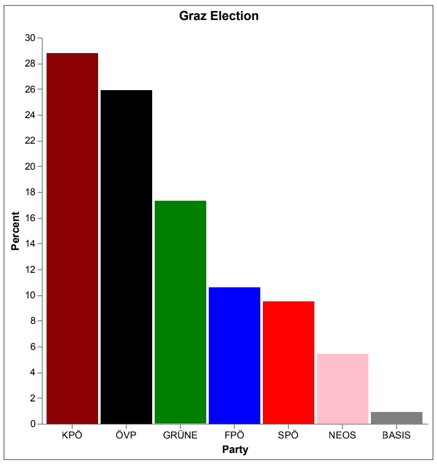
3.4 Sample Graphs

The Vega specification vocabulary leaves numerous options for creating charts. Not only is it possible to create charts, but also other interactive visualizations like simple games are possible. For example, a Pacman game can be found in the vega documentation examples under [Tiberghien 2022]. Since the editor offers the possibility to write in both Vega and Vega-Lite, charts were created using both kinds of specification:

- Figure 3.5 shows bar charts specified in Vega and Vega-Lite respectively.
- Figure 3.6 shows line charts specified in Vega and Vega-Lite respectively.
- Figure 3.7 shows scatter plots specified in Vega and Vega-Lite respectively.

3.5 Conclusion

The specification vocabularies Vega and Vega-Lite are useful tools which are mainly aimed to provide developers a possibility to create higher-level visualization tools like interactive chart editors. The Vega Editor is helpful in understanding how the vocabulary works as it offers a live edit possibility through which the effect of the different properties is learned quickly. Additionally, the editor has the advantage that existing specifications can easily be edited and extended, this is also the case for higher-level tools that offer the export of specification files.



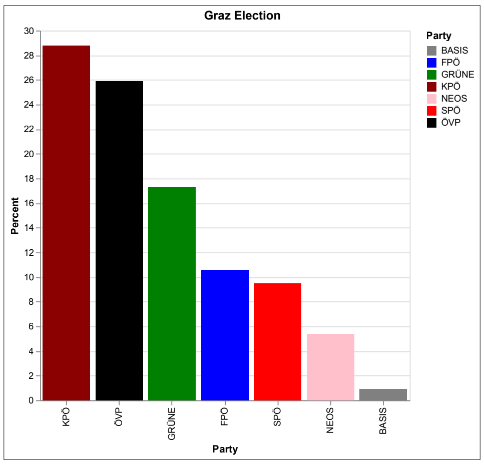
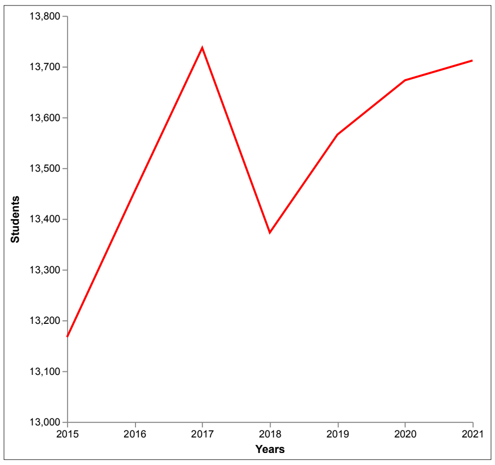


Figure 3.5: Bar charts created with Vega (left) and Vega-Lite (right). [Charts were created by the authors of this paper.]



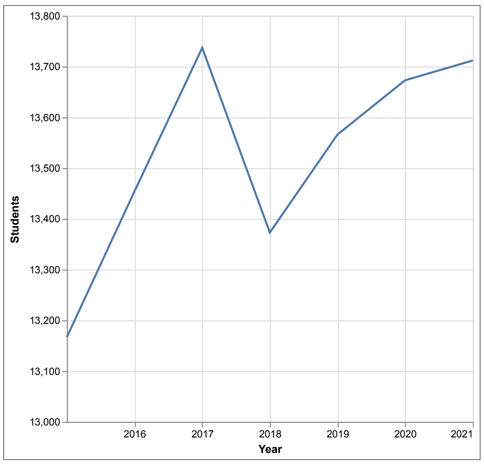
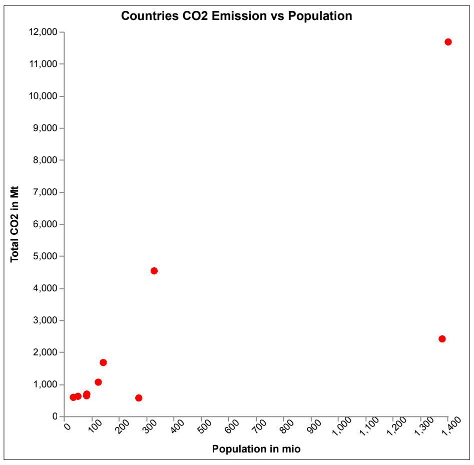


Figure 3.6: Line charts created with Vega (left) and Vega-Lite (right). [Charts were created by the authors of this paper.]



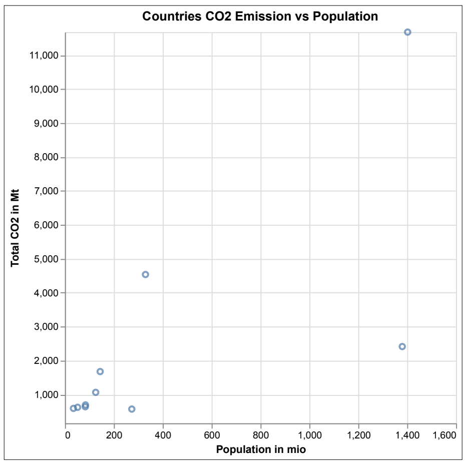


Figure 3.7: Scatter plots created with Vega (left) and Vega-Lite (right). [Charts were created by the authors of this paper.]

Chapter 4

Voyager

4.1 Introduction

Voyager is an interactive open-source chart editor which is part of the Vega project and led by the University of Washington Interactive Data Lab. It uses CompassQL, Vega-Lite, and the Vega Renderer, which are other tools of the Vega project, to offer the user a recommendation-based data exploration tool [Wongsuphasawat et al. 2015].

4.2 Technical Details

This section covers the technical characteristics of Voyager.

4.2.1 Installation and Execution

Working remotely with the deployed version of the tool is quite simple and can easily be done by visiting [IDL 2022i]. If one wants to work locally with Voyager, one has to clone the source code from [IDL 2022h], navigate into the root directory of the project, run yarn to install the dependencies, and then run yarn start to start the editor locally.

4.2.2 Architecture

According to the statistics of the Github repository, Voyager is written in Typescript (86.3 %), JavaScript (6.5 %), and SCSS (6.7 %) and uses React for rendering the user interface. To function properly, it relies on other tools from the Vega Project. After the user has specified which data should be displayed, CompassQL (a Recommender Engine) is used to get the top-ranked charts. These charts are then further processed from the Vega-Lite compiler and then again further processed from the Vega Renderer which renders the charts [Wongsuphasawat et al. 2015].

4.2.3 Supported Input and Output

4.2.3.1 Input

The tool provides two options for importing. One is to upload local CSV files or paste the data into the import window, the other one is to specify an URL to a CSV or JSON file which contains the data. In the import window is stated that the data is kept private and only visible to the user.

4.2.3.2 Output

The only export format which is offered by Voyager is the Vega-Lite specification. But the gained code can be used in combination with the Vega Editor to extend functionality and get additional access to all export formats the Vega Editor provides (described in Section 3.2.4.2).

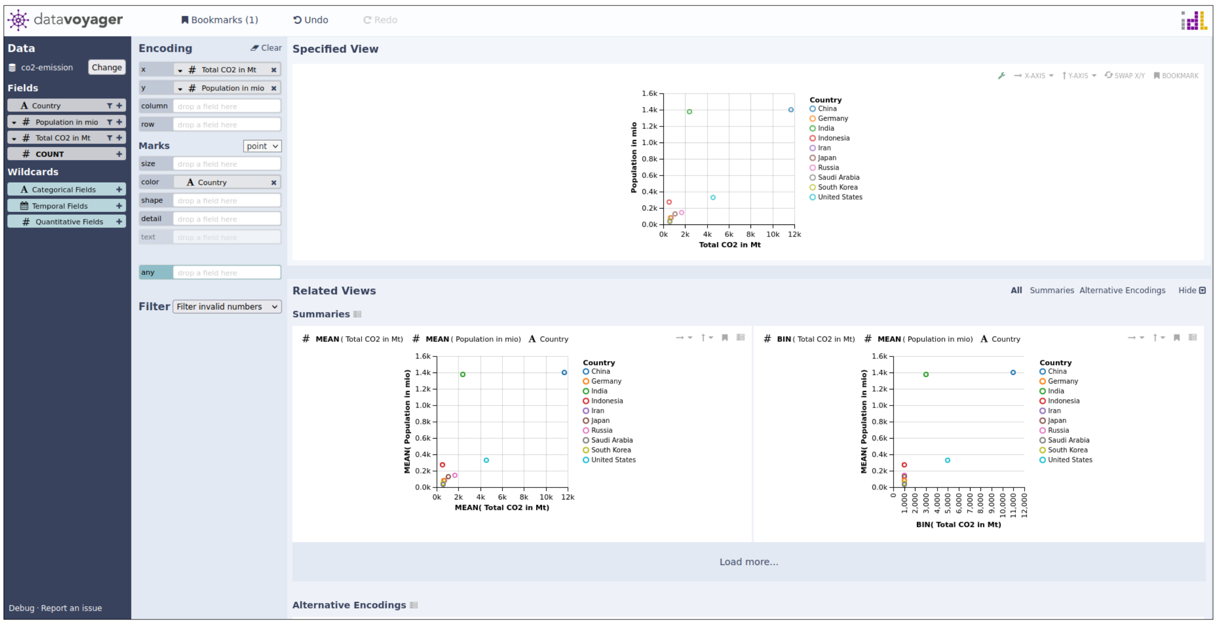


Figure 4.1: The user interface of Voyager. [Screenshot was taken by the authors of this paper.]

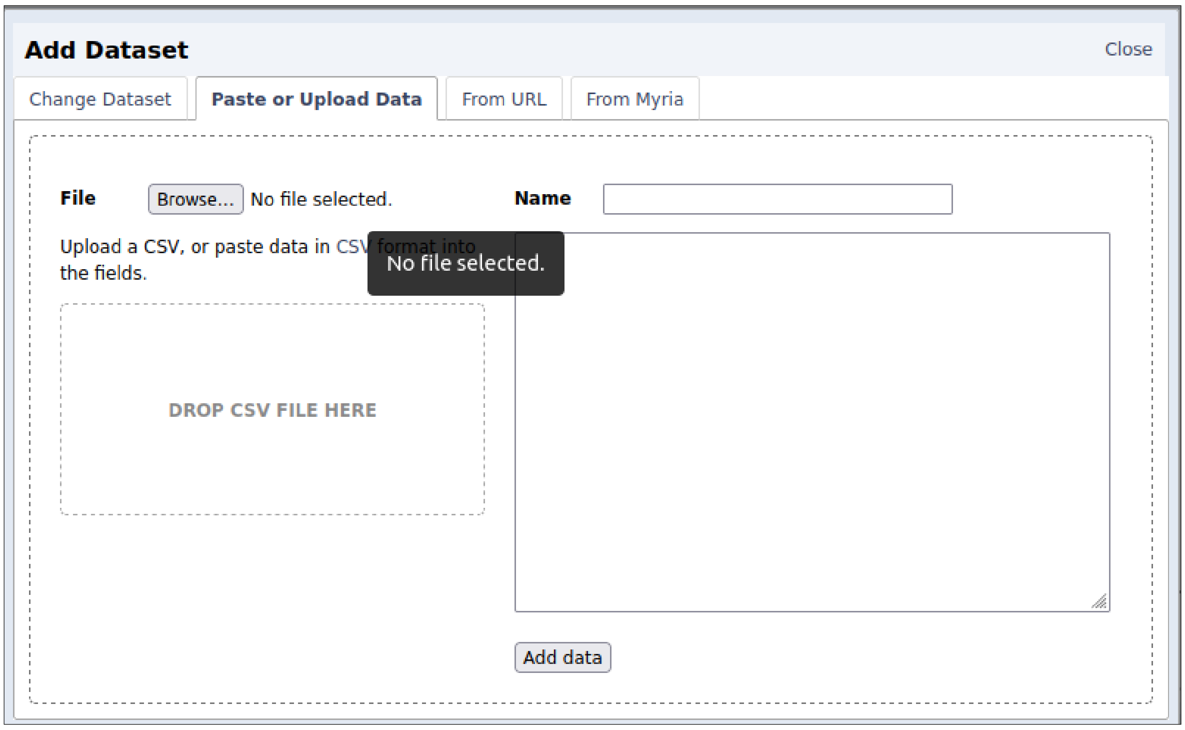
4.3 User Interface

The user interface of Voyager can be divided into several panes which can be seen in Figure 4.1. The most important ones are [IDL 2022g]:

- The top panel provides buttons for navigating through history via undo/redo and opening a window which displays the current bookmarks.
- The data panel provides all data fields that are part of the imported data and can be moved via drag and drop.
- The encoding panel in which the data fields can be dropped to specify the resulting charts regarding the user's needs.

4.4 Workflow

The typical workflow when using Voyager is at first to upload the desired data, as shown in Figure 4.2. There is a button in the data panel that opens a new window in which this action can be executed. After the data has been uploaded, immediately multiple charts are suggested. The user can then further specify what the chart should look like. This can be done by dragging and dropping the data fields to the desired axes and doing additional specifications like for instance choosing the right kind of visualization marks or adding a color scheme. If the chart fits the user's need, it can be bookmarked, as shown in Figure 4.3, and is thus saved as long as the session lasts or it is removed from the bookmarks. Even Undo and Redo Actions will not change the bookmarks list. If the user decides to finish working with Voyager, the bookmarked charts can be exported as Vega-Lite specifications. Depending on the purpose of the created charts the specification files can optionally be further progressed by the Vega Editor.



Upload a CSV, or paste data in CSV format into the fields.

Figure 4.2: Voyager: File upload. [Screenshot was taken by the authors of this paper.]

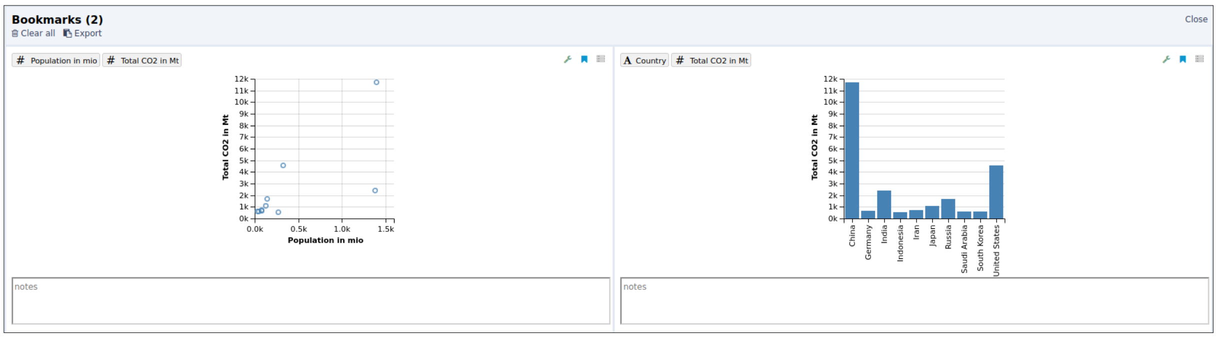


Figure 4.3: Voyager: Bookmarked charts. [Screenshot was taken by the authors of this paper.]

4.5 Sample Graphs

Vega-Lite specification code of simple charts can be easily exported from Voyager. If the details of the charts are more complex, it is often not possible to get to the exact desired result. The exported Vega-Lite specification can then be adapted to fit the user's needs via the Vega Editor. Figures 4.4, 4.5 and 4.6 show a bar chart, line chart, and scatter plot respectively from which the specification codes were always created with Voyager.

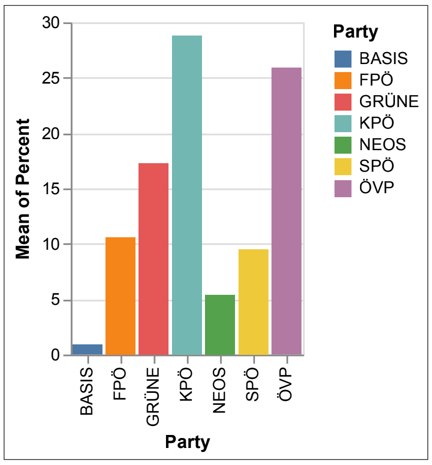


Figure 4.4: Bar chart resulting from Voyager specification code in Vega-Lite. [Screenshot was taken by the authors of this paper.]

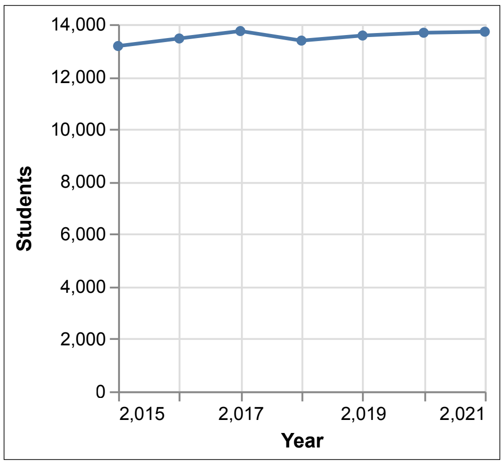


Figure 4.5: Line chart resulting from Voyager specification code in Vega-Lite. [Screenshot was taken by the authors of this paper.]

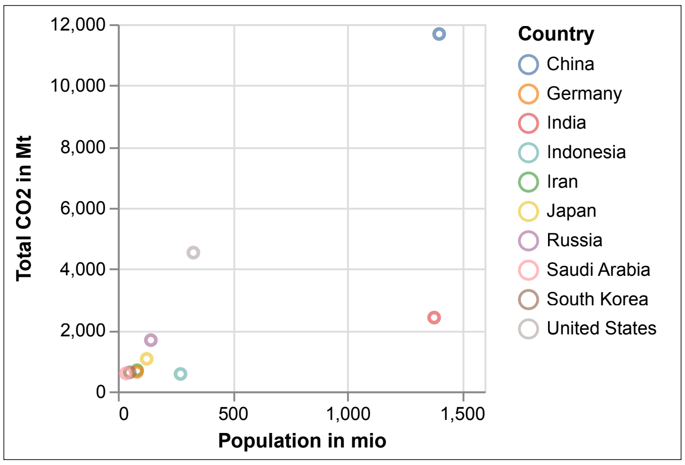


Figure 4.6: Scatter plot resulting from Voyager specification code in Vega-Lite. [Screenshot was taken by the authors of this paper.]

4.6 Conclusion

Voyager is a useful tool for quickly analyzing data and retrieving correlations between data fields. It provides an understandable, straightforward interface and charts can be created with just little effort. Browsing many related, from the system recommended charts at once does increase the speed of finding specific patterns in the data. The export of retrieved charts in Vega-Lite specification is practical on the one hand if further adaptions must be carried out, but on the other hand, there is the clear disadvantage that Voyager does not offer SVG and PNG options for exporting. To sum it up it can be said that Voyager is a good tool for the tasks it is intended for, but if the goal is to create more complex visualizations it can only serve as a starting point.

Chapter 5

Dex

This chapter about Dex is based on the material from Patrick Martin [Martin 2022b; Martin 2022a; Martin 2019; Martin 2017], the developer of Dex.

5.1 Technical Details

This section covers the technical characteristics of Dex.

5.1.1 Installation

According to the Dex's GitHub page [Martin 2019], there are two different ways to install Dex:

1. Downloading the stable version of Dex [Martin 2017]. However, the stable version of Dex is not up to date.

2. Cloning the Dex repository. This method will provide the latest version of Dex, but errors may occur.

For this survey paper, the stable version 0.9.0.1 of Dex was installed and tested.

5.1.2 Architecture

Dex is written in Java 8.0 and Groovy using the JavaFX framework. Dex incorporates a wide range of different libraries that can be chosen to create charts [Martin 2017], including: JavaFX, Google, Plotly, D3, and C3. The chart types provided by the various libraries and the scope of the chart configuration possibilities differ from each other quite significantly in some cases.

5.1.3 Input Possibilities

Dex offers several ways to import the data for visualizations. In the tested version of Dex, the following options were discovered [Martin 2017]:

- File
- jar Files
- CSV Files
- Text

5.1.4 Output Possibilities

Dex offers a variety of different export methods. In the tested version of Dex, the following methods were discovered [Martin 2017]:

- Saving the Dex project
- Groovy template
- CSV Files
- Charts via HTML-Template (Available for some charts)

Unfortunately, we miss the possibility to export the created charts as PNG or SVG.

5.1.5 Intelligent Features

Since Dex offers many different libraries for creating charts, the configuration possibilities offered by the various libraries differ in terms of their functions and features. For example, a C3 chart automatically scales the axes, but does not provide the ability to configure the chart. In comparison, a JavaFX chart does not scale the axes automatically but, can be configured in several ways. A notable general feature of Dex is the available tutorials for almost all components.

5.2 User Interface

The user interface of Dex is divided into three areas. In the Dex documentation [Martin 2022a], these three areas are referred to and described as follows:

- *Palette Plane*: Located on the left side of the user interface and contains all the components of Dex grouped into several different categories.
- *Task Plane*: Located in the centre of the Dex user interface and is used to create a sequence of components. When executing the Task Plane, all components are executed one after the other from top to bottom, thus finally resulting in a sequential process.
- *General Pane*: Located on the left side of the Dex user interface. The General Pane contains the user interface of the component currently selected in the Task Plane or the components documentation. The General Pane, therefore, allows the configuration of individual components of the Task Plane. To access the online documentation of a component, it is necessary to select a component in the Palette Plane.

5.3 Workflow

In the following, a typical workflow to create a chart as described by the demo projects that come with Dex [Martin 2017] is presented in an abbreviated version:

1. The Dex component *readCSV* can be used to input a CSV File which should be visualized. Figure 5.1 shows the *readCSV* component added to the Task Plane and the settings of this component.
2. In the second and optional step, the *readCSV* component can be used to display the data imported by the *readCSV* component, see Figure 5.2.
3. After the data to be visualized has been imported into Dex, a component can be selected for visualization in this case the *Bar Chart* component of the JavaFX library has been selected and added to the Task Plane. Figure 5.3 shows the bar chart created by Dex using the JavaFX library and the configuration options provided by Dex.

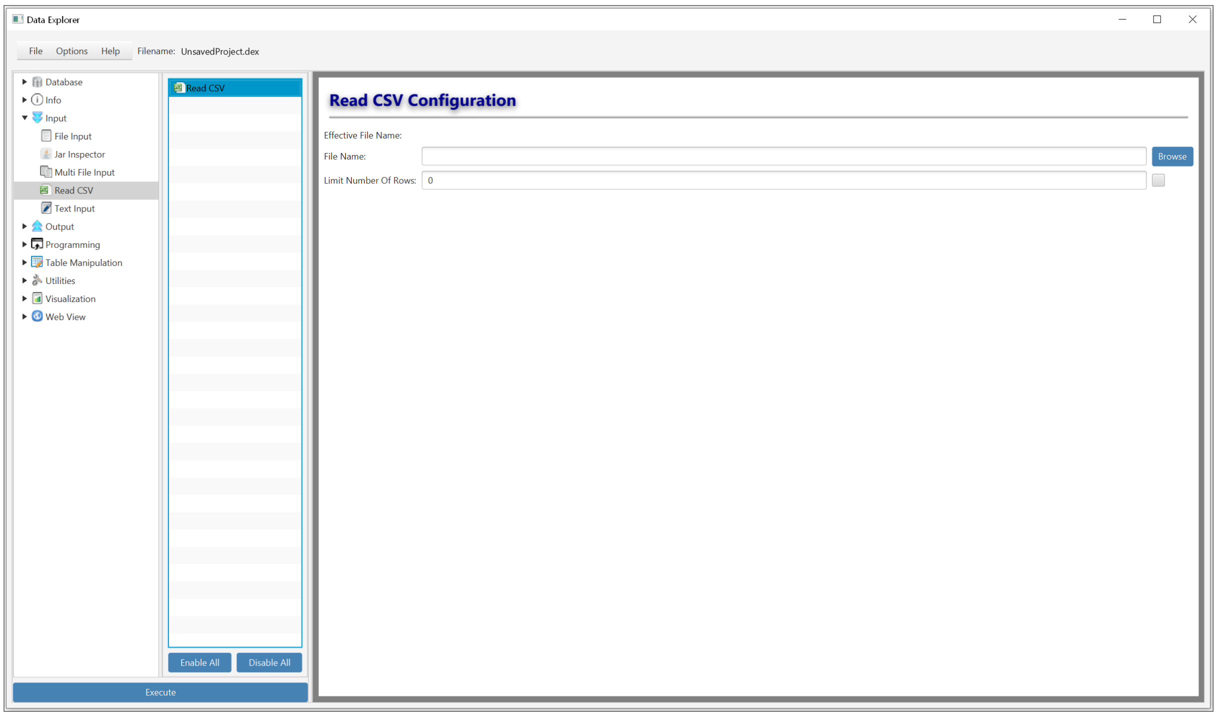
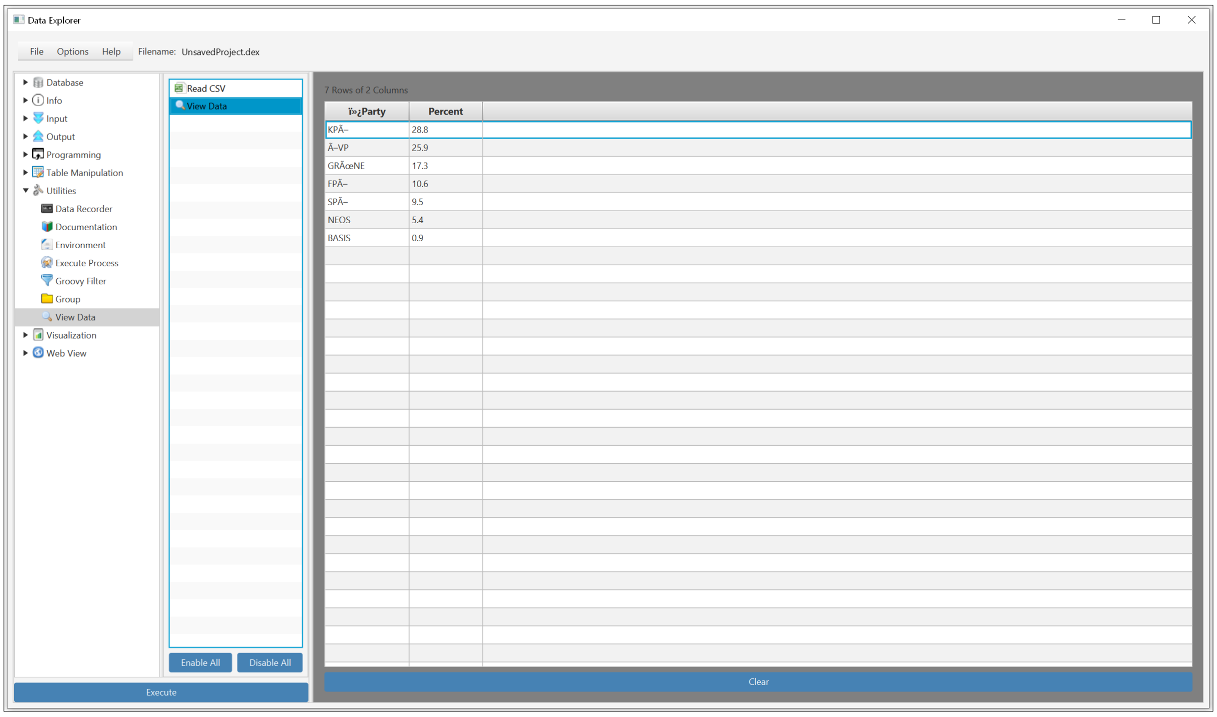


Figure 5.1: Dex: Importing data into Dex. [Screenshot taken by the authors of this paper.]

Step 2 is optional and is not required. However, it makes sense to integrate it into the workflow to verify that the data was imported successfully by Dex. In addition, the majority of the editors mentioned in this survey paper display the data after importing.

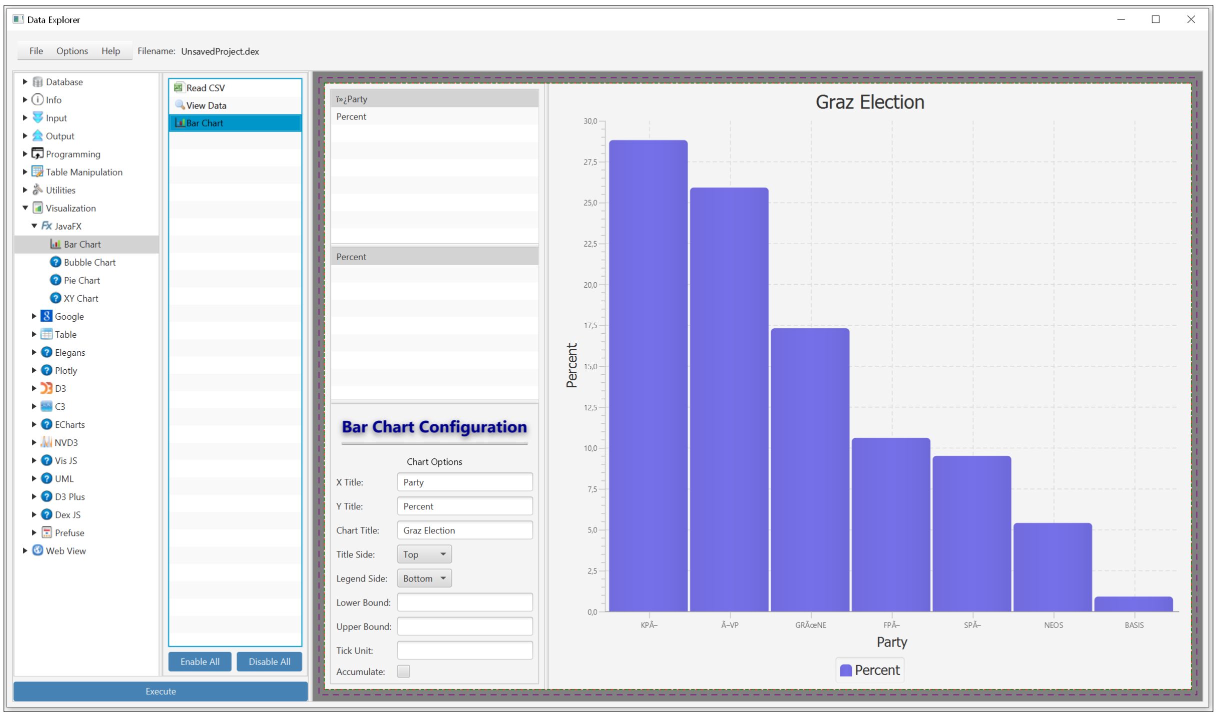


Filename: UnsavedProject.dex

7 Rows of 2 Columns

ï»¿Party	Percent
KPÃ–	28.8
Ã–VP	25.9
GRÃœNE	17.3
FPÃ–	10.6
SPÃ–	9.5
NEOS	5.4
BASIS	0.9

Figure 5.2: Dex: Viewing the imported data. [Screenshot taken by the authors of this paper.]



Filename: UnsavedProject.dex

Bar Chart Configuration

Chart Options

X Title: Party
Y Title: Percent
Chart Title: Graz Election
Title Side: Top
Legend Side: Bottom
Lower Bound:
Upper Bound:
Tick Unit:
Accumulate: ☐

Figure 5.3: Dex: JavaFX bar chart configuration. [Screenshot taken by the authors of this paper.]

5.4 Sample Graphs

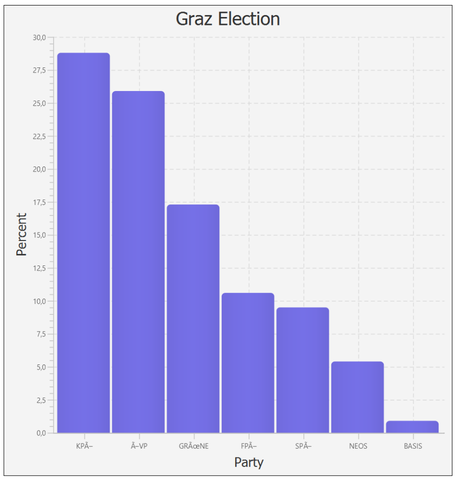
Since the charts created by Dex through different libraries are in some cases very different from each other in their appearance and the available configuration possibilities. It was decided to create both a JavaFX and a C3 chart for each of the three sample charts to demonstrate the variety of Dex:

- Figure 5.4 shows bar charts created in Dex with JavaFX (Figure 5.4a) and C3 (Figure 5.4b).
- Figure 5.5 shows line charts created in Dex with JavaFX (Figure 5.5a) and C3 (Figure 5.5b).
- Figure 5.6 shows scatter plots created in Dex with JavaFX (Figure 5.6a) and C3 (Figure 5.6b).

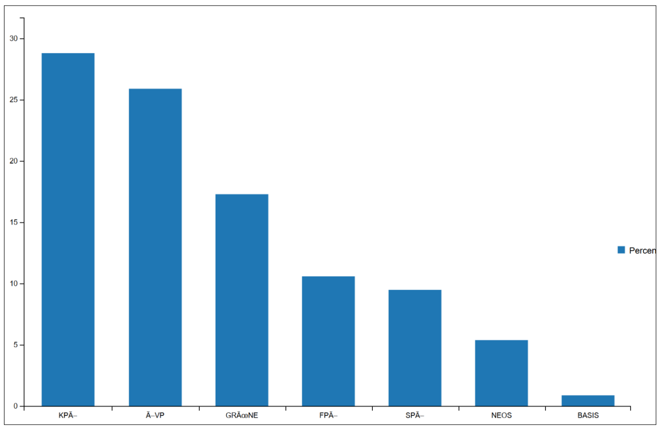
Unfortunately, Dex does not provide configuration possibilities for C3 charts. Therefore, they could not be customized in the same way as the charts created with JavaFX. In addition, the Dex user interface unfortunately minimally cuts off created chars on the right-hand side.

5.5 Conclusion

More than anything else, Dex is a great tool when it comes to visualizing and analyzing data. Due to the large number of libraries supported by Dex and the variety of different visualization options. Unfortunately, however, there is no option to export the created charts as PNG or SVG. In addition, the time needed to get used to the workflow of Dex and to familiarize with all the components of Dex is higher compared to other chart editors. However, the tutorials available for almost all components of Dex minimize the learning curve in the best possible way. Very impressive is the fact that Dex was developed by only one person in his spare time. Therefore, the set of features of Dex is outstanding.

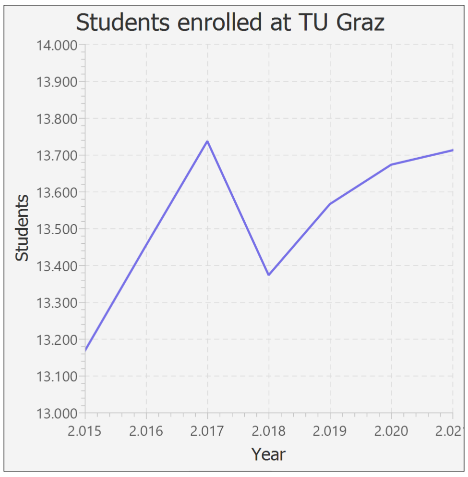


(a) Dex JavaFX Bar Chart.

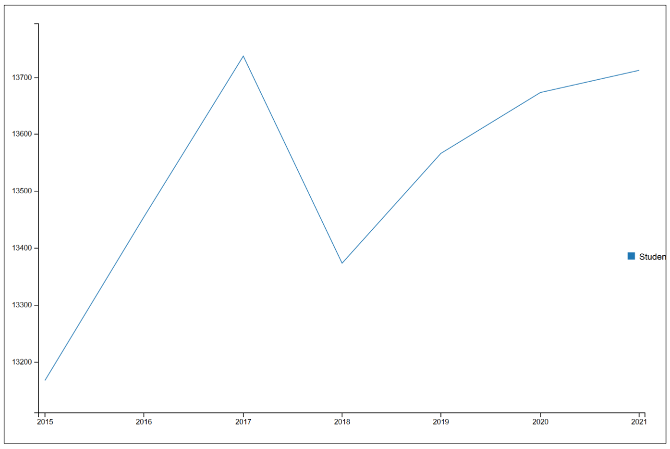


(b) Dex C3 Bar Chart.

Figure 5.4: Dex: Bar charts created with Dex. [Both charts created by the authors of this paper.]

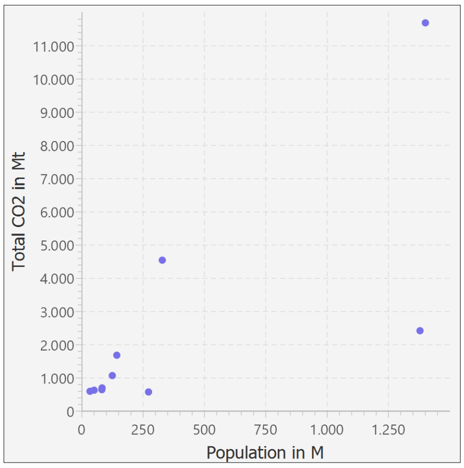


(a) Dex JavaFX Line Chart.

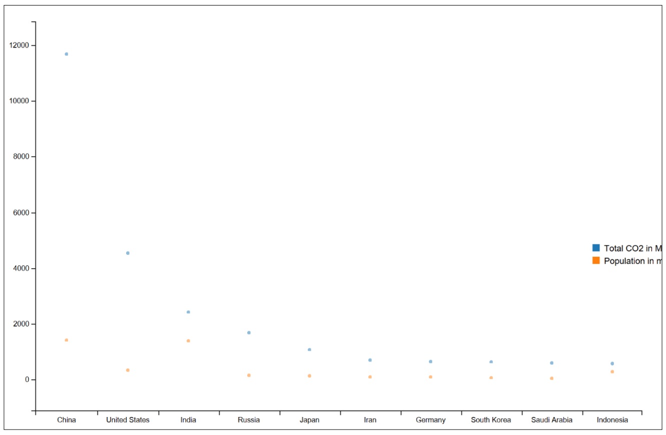


(b) Dex C3 Line Chart.

Figure 5.5: Dex: Line charts created with Dex. [Both charts created by the authors of this paper.]



(a) Dex JavaFX Scatter Plot.



(b) Dex C3 Scatter Plot.

Figure 5.6: Dex: Scatter plots created with Dex. [Both charts created by the authors of this paper.]

Chapter 6

Liquid Diagrams

Liquid Diagrams is a simple open-source chart editor which was developed by the Institute for Information Systems and Computer Media (IICM) at the Graz University of Technology using Adobe Flash and Flex [Andrews and Lessacher 2010]. This chapter takes a closer look at the installation process required to run Liquid Diagrams locally, followed by a discussion of its user interface, created charts, and related software features.

6.1 Installation Process

Liquid Diagrams is no longer in development. It was built years ago when Adobe Flash technology was considered state of the art. Since Adobe Flash was officially discontinued in 2020, compiling the source code of Liquid Diagrams using Adobe FlashBuilder, the Integrated Development Environment (IDE) for Adobe Flash, was no longer possible as the IDE was not available for download anymore. Alternative IDEs were not able to handle the proprietary file format. However, since the development of the Adobe Integrated Runtime (AIR), also known as Adobe AIR, which is the platform-independent runtime at which the application was targeted, was continued by Harman International, Liquid Diagrams can still be used. Therefore, having access to a previously compiled .air file, the binary of Liquid Diagrams can still be installed on any operating system supported by AIR.

6.2 User Interface

The user interface of Liquid Diagrams is kept rather simple. On application start, the user is presented a grid of 12 different chart types acting as a navigation hub, as illustrated in Figure 6.1. Clicking on a chart button opens a new chart editor window to create a new chart of the respective type. This way, for example, it is possible to open multiple, independent chart editor windows simultaneously. As the user interface elements are identical across chart types and only chart appearance varies, the interface is described on the bases of a bar chart in the following paragraphs.

Each new chart editor opens as a blank window as shown in Figure 6.2. From a usability point of view, this requires users to perform an additional conscious action to initiate the data import. Via the left toolbar, users can open the system file selection dialog to select a CSV data file to import.

Figure 6.3 depicts the user interface presented after selecting a CSV file for the chart. In this interface, users can preview the imported chart data, and, if necessary, adjust the data formats for each column of the data set. Supported data formats are strings, numbers, dates, and percentages. Besides the data format, users can select the proper separator symbols used to divide entries in the CSV file. Further, users can limit the data to be used in the chart by selecting first and last line indices. The interface also shows an option to transpose the data, which might be useful to swap chart axes.

Once the data is imported, Liquid Diagrams shows the chart using a default configuration (see Figure 6.4). Chart and axes titles are not automatically taken from the labeled columns of the CSV file.

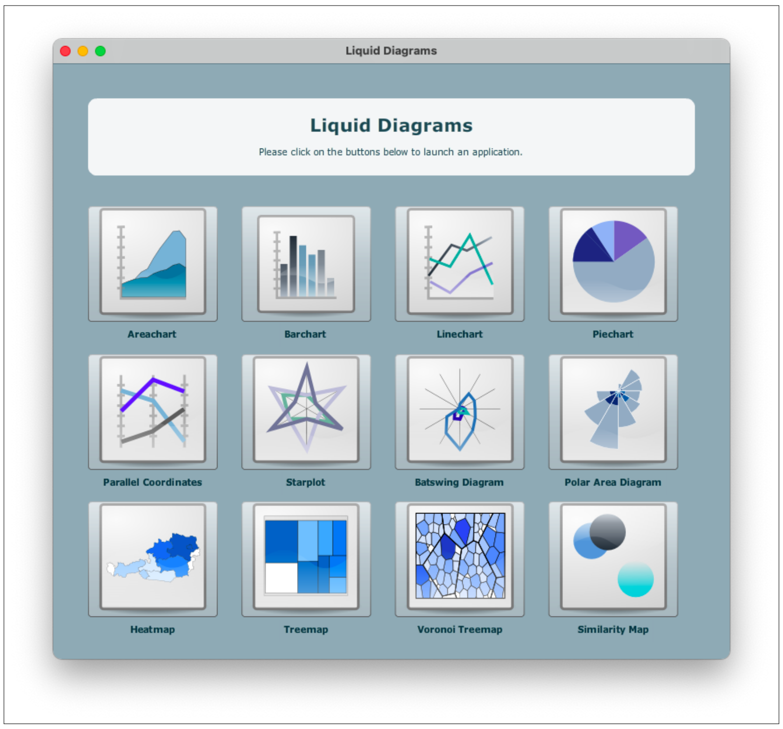


Figure 6.1: Main user interface on application start. [Screenshot was taken by the authors of this paper.]

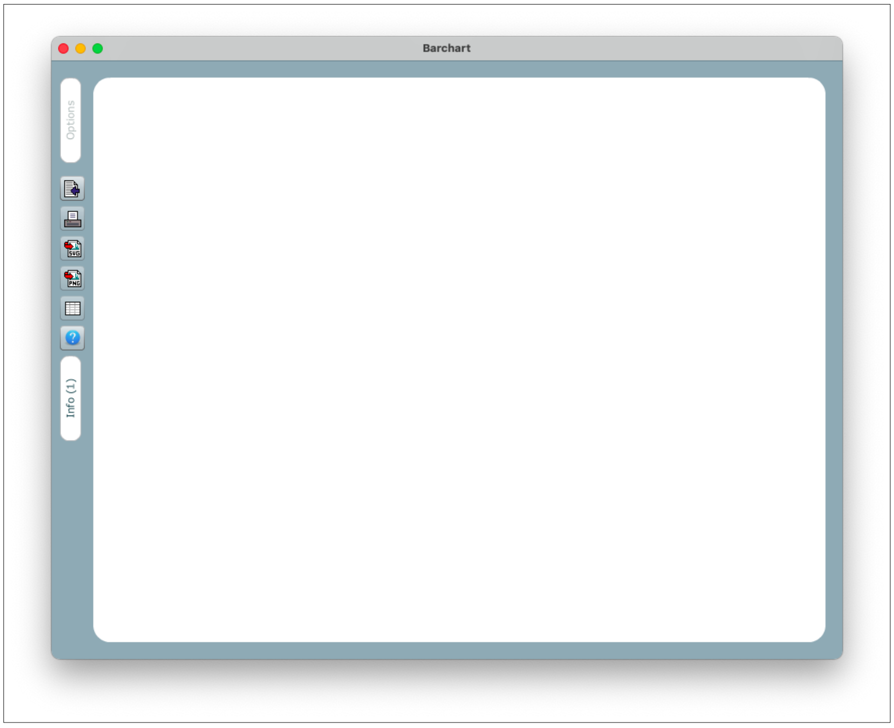
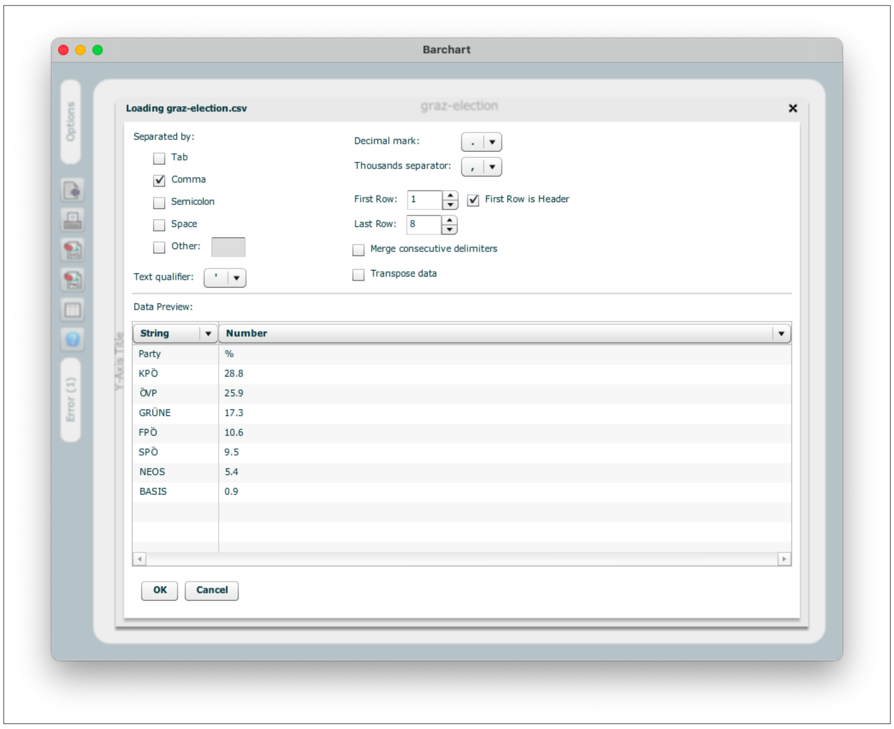


Figure 6.2: Empty chart editor window after selecting chart type in main interface. [Screenshot was taken by the authors of this paper.]



Loading graz-election.csv

Separated by:
- ☐ Tab
- ☑ Comma
- ☐ Semicolon
- ☐ Space
- ☐ Other:

Text qualifier: '

Decimal mark: .

Thousands separator: ,

First Row: 1 ☑ First Row is Header

Last Row: 8

☐ Merge consecutive delimiters

☐ Transpose data

Data Preview:

String	Number
Party	%
KPÖ	28.8
ÖVP	25.9
GRÜNE	17.3
FPÖ	10.6
SPÖ	9.5
NEOS	5.4
BASIS	0.9

Figure 6.3: User interface for data import after selecting data file from file system. [Screenshot was taken by the authors of this paper.]

However, by left-clicking onto those titles, users can edit the titles very easily. Hovering the titles with the mouse triggers a tooltip mentioning this possibility. In general, the Liquid Diagrams makes extensive use of such tooltips to provide its user hints on how to use the application.

Besides changing chart and axis titles, Liquid Diagrams provides some more chart options located at the top left of the chart editor window. Clicking on this options button opens a new settings dialog divided into three sections for general, font, and axis options.

Figure 6.5 shows general chart settings. In this section of the options dialog, users can, for example, change the color scheme of the chart or toggle the chart's title or position of the chart legend. In the fonts sections, shown in Figure 6.6, users can adjust various font properties, for example, font, face, size, or color of different titles and labels of the chart. In the last settings sections depicted in Figure 6.7, that is for axes, the application provides only limited options, which are toggling the axes' titles and the positional style for the X-axes tick labels. Although Liquid Diagrams provides settings to adjust the display units for both axes, this feature is rather unintuitive to use.

Lastly, Liquid Diagrams also supports a tabular data view by which users can have a more in-depth look at the data used for the chart at hand. This data view is illustrated in Figure 6.8. However, this data view is solely intended to view the data. Changing values in the data cells, reordering, and other interactions are not supported.

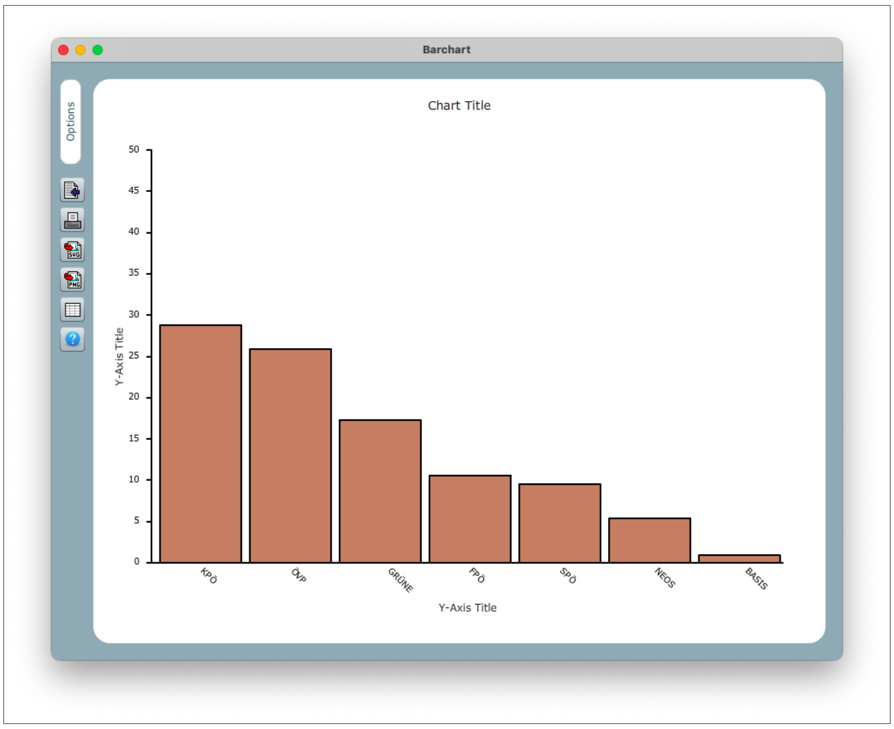
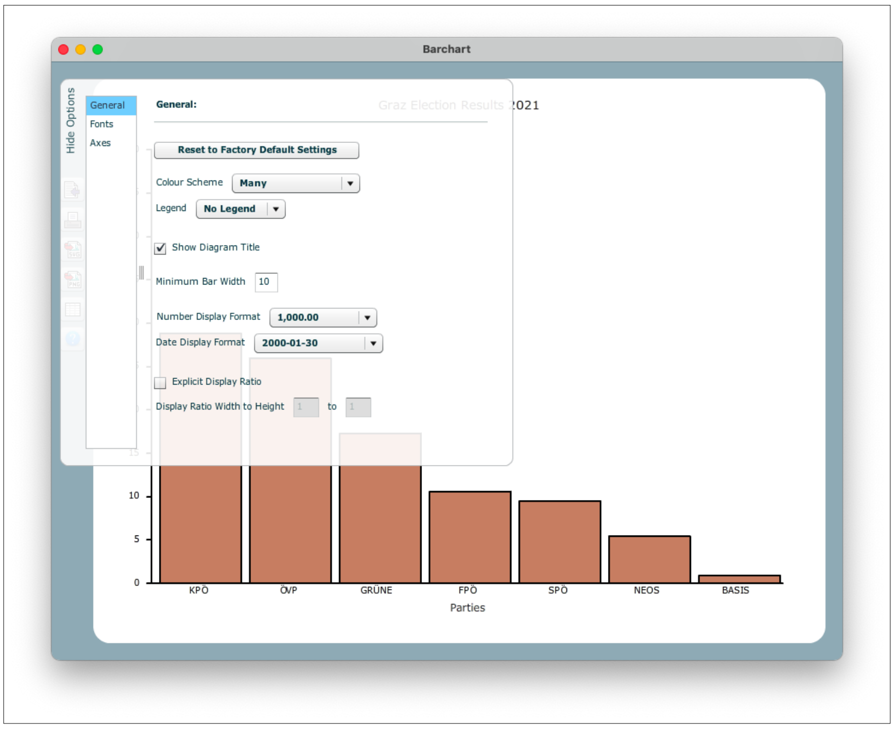


Figure 6.4: Initial bar chart configuration. [Screenshot was taken by the authors of this paper.]



General:

Reset to Factory Default Settings

Colour Scheme: Many

Legend: No Legend

[✓] Show Diagram Title

Minimum Bar Width: 10

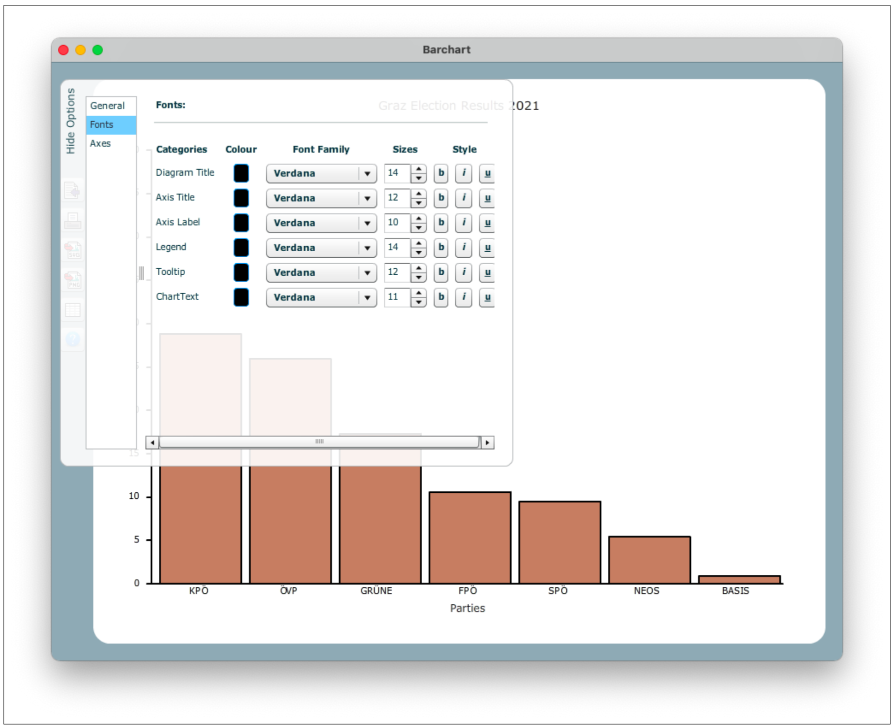
Number Display Format: 1,000.00

Date Display Format: 2000-01-30

[ ] Explicit Display Ratio

Display Ratio Width to Height: 1 to 1

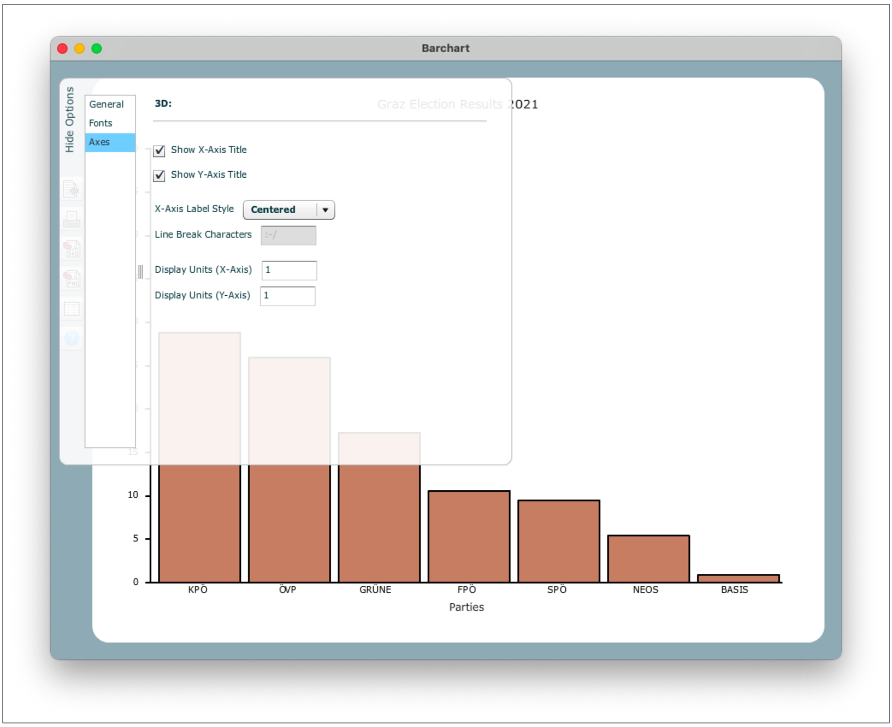
Figure 6.5: General chart options. [Screenshot was taken by the authors of this paper.]



Fonts:

Categories	Colour	Font Family	Sizes	Style
Diagram Title	■	Verdana	14	b i u
Axis Title	■	Verdana	12	b i u
Axis Label	■	Verdana	10	b i u
Legend	■	Verdana	14	b i u
Tooltip	■	Verdana	12	b i u
ChartText	■	Verdana	11	b i u

Figure 6.6: Fonts chart options. [Screenshot was taken by the authors of this paper.]

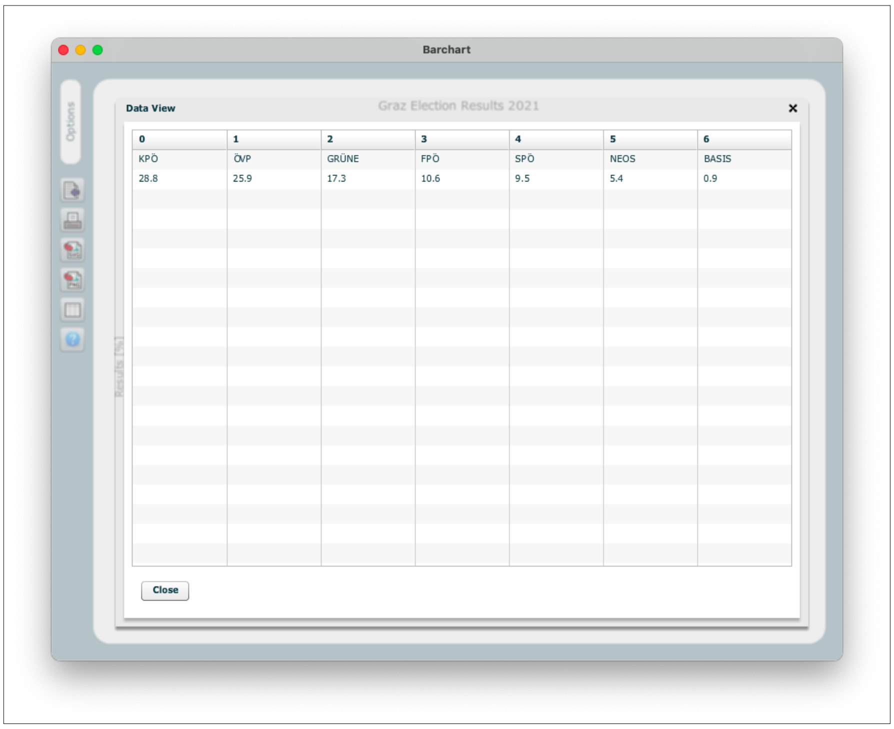


3D:

- [✓] Show X-Axis Title
- [✓] Show Y-Axis Title

X-Axis Label Style Centered
Line Break Characters :-/
Display Units (X-Axis) 1
Display Units (Y-Axis) 1

Figure 6.7: Axes chart options. [Screenshot was taken by the authors of this paper.]



0	1	2	3	4	5	6
KPÖ	ÖVP	GRÜNE	FPÖ	SPÖ	NEOS	BASIS
28.8	25.9	17.3	10.6	9.5	5.4	0.9

Figure 6.8: Data view for displaying dataset used for chart. [Screenshot was taken by the authors of this paper.]

6.3 Sample Graphs

Although Liquid Diagrams supports 12 different chart types, it does not support rather common scatter plots. In line with the other chart editors of this survey, a sample bar chart and line chart were created using the respective datasets described in Section 1.2 and are shown in Figure 6.9 and Figure 6.10 respectively. The application supports chart export as SVG and PNG. As the SVG format, by definition, is resolution independent, exported SVG charts are scalable. However, they have a fixed aspect ratio, and they support no responsiveness, for example, repositioning the legend or other chart elements depending on the chart format.

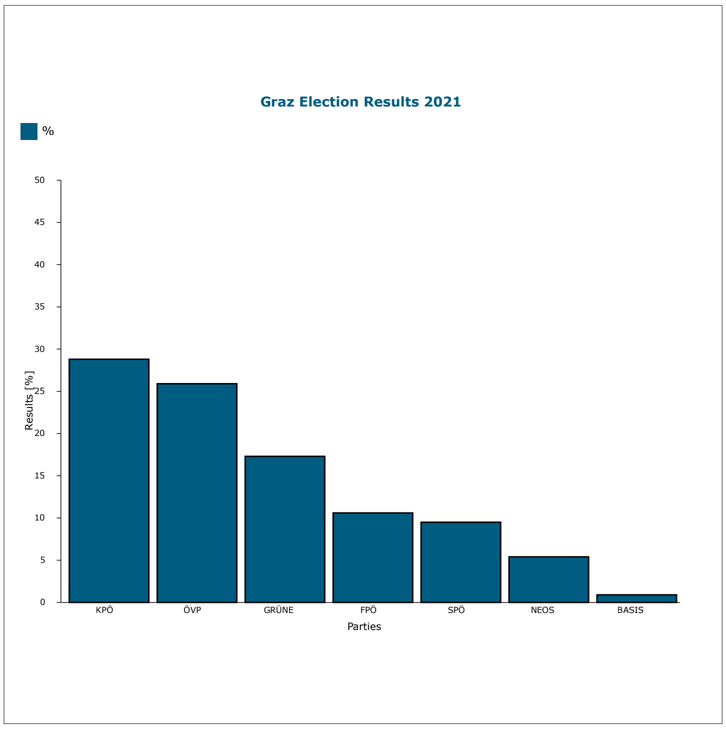


Figure 6.9: Sample bar chart created with Liquid Diagrams. [Chart created by the authors of this paper.]

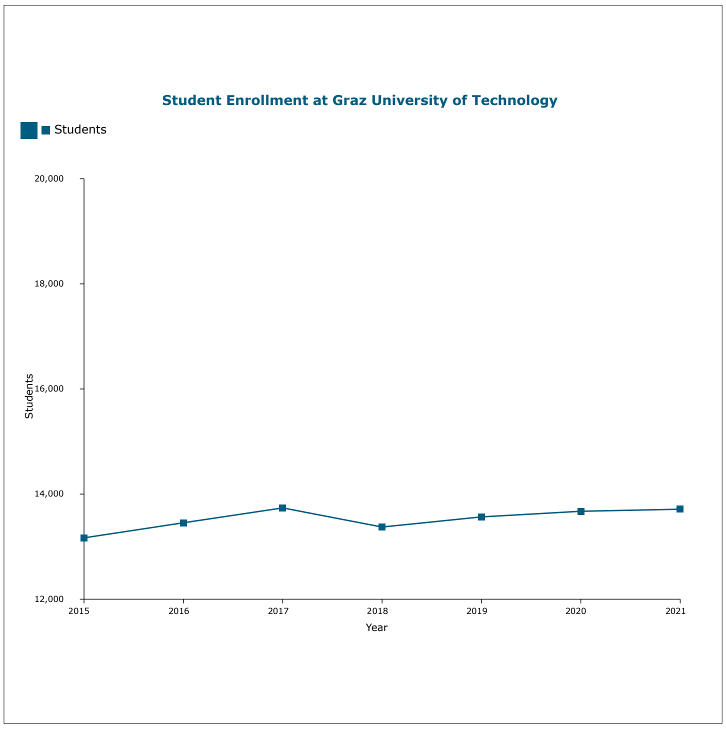


Figure 6.10: Sample line chart created with Liquid Diagrams. [Chart created by the authors of this paper.]

6.4 Software Features

Due to its limited feature set, Liquid Diagrams is a rather simple, but easy-to-use chart editor. Charts can be created through a visual and interactive interface, which, by its nature, is already more user friendly than other grammar-based chart-editors.

Import and export functionality is limited to CSV for datasets and SVG and PNG formats for chart export. Besides chart export, the application features a build-in print function allowing users to print their created charts directly from the editor. As already mentioned in Section 6.3, SVG charts do not provide any interactive or responsive functionality.

Chart configuration is also limited. Although users are able to select a color scheme for bar charts and line charts, the settings only applies to the whole bar chart. To select a custom color, for example, by specifying the RGB values or the color code in HEX format, users first need to enable the legend and then click the legend entry to open a color picker.

Axes options are even more constrained. Their configuration is limited to toggling axes titles and choosing a predefined label position for tick labels on the X-axes. Specifying a custom range for the Y-axis is not possible. Liquid Diagram attempts to figure out appropriate upper bound on its own, whereas the lower bound always starts at zero.

In return, Liquid Diagrams has rather extensive support for text styling. Each textual chart element type, for example chart and axis title can be styled separately. This way, users are able to create visually more appealing charts. However, as most charts are created with black labels, this feature does not compensate the lack of many other features found in other chart editors.

6.5 Conclusion

Considering the fact that Liquid Diagram is quite a legacy tool, it is not surprising that the application lacks technical finesse when compared with more modern chart editors. Therefore, Liquid Diagrams will not be the tool of choice when users want maximum control over their charts. Instead, Liquid Diagrams is very easy to use and heavily relies on tooltips to support the user during the chart creation process. However, it can be argued that having to use tooltips might indicate a flaw in the user interface design in the first place. Yet, Liquid Diagrams, despite its age and usage of outdated technology, is still a valid choice to create simple graphs.

Chapter 7

Comparison of Chart Editors

Table 7.1 provides a tabular overview of the five open-source chart editors selected for this survey: RAWGraphs, Vega Editor, Voyager, Dex, and Liquid Diagrams. The following three symbols are used:

1. ✓ : Fully supported
2. $\sim$: Partially supported
3. $\otimes$: Not supported

	RAWGraphs	Vega Editor	Voyager	Dex	Liquid Diagrams
Chart Editor Type					
Grammar-based	∼	✓	⊗	⊗	⊗
Visual / Interactive	✓	⊗	✓	✓	✓
Local Installation					
Binary Installable	⊗	⊗	⊗	✓	✓
Source Compilable	✓	✓	✓	✓	⊗
Chart Types					
Bar Chart	✓	✓	✓	✓	✓
Line Chart	✓	✓	✓	✓	✓
Scatter Plot	✓	✓	✓	✓	⊗
Input Formats					
CSV	✓	✓	✓	✓	✓
Proprietary File Format	✓	✓	⊗	✓	⊗
Export Formats					
PNG	✓	✓	⊗	⊗	✓
SVG	✓	✓	⊗	⊗	✓
Proprietary File Format	✓	✓	✓	✓	⊗
Chart Settings					
Chart Title	✓	✓	⊗	∼	✓
Axis Titles	✓	✓	∼	∼	✓
Axis Range	∼	✓	∼	∼	⊗
Axis Ticks	∼	✓	∼	∼	∼
Tick Labels	✓	✓	⊗	∼	∼
Color Coding	✓	✓	∼	⊗	∼
Legend	✓	✓	∼	∼	∼
Background Grid	⊗	✓	⊗	∼	⊗
Other Features					
Tabular Data Preview	✓	⊗	⊗	✓	✓
Responsive Charts	✓	✓	⊗	∼	⊗

Table 7.1: Comparison of Chart Editors

Chapter 8

Concluding Remarks

Chart editors are essential tools in the field of information visualization to visually present, and more easily understand and analyze information and data. This survey was limited to open-source chart editors, primarily due to having the possibility of running them locally. A local instance helps to keep sensitive and personal data private as a server communication is often not required or can be redirected to own servers by modifying the open source.

In general, if there is the goal to create diagrams quickly or explore data sets, visual interactive chart editors are the best choice to accomplish the task. On the other hand, if the visualizations are more complex, visual editors may come to their limits. Then it is advisable to switch to a grammar based chart editor to get to the wanted result. Grammar specifications also also the possibility to create charts via software and to build higher-level visualization tools on top of them.

Regardless of these recommendations, it is worth noting out that due to the restriction to open-source chart editors, numerous well-made, paid and free chart editors were not considered for this study. Therefore, the possibility of better suited tools for above situations needs to be accounted for when actually choosing a tool for practical application.

Bibliography

Andrews, Keith and Martin Lessacher [2010]. *Liquid Diagrams: Information Visualisation Gadgets*. Proc. 14th International Conference on Information Visualisation (IV'10) (London, UK). IEEE Computer Society Press, 26 Jul 2010, pages 104–109. doi:10.1109/IV.2010.100. https://ftp.isds.tugraz.at/pub/papers/andrews-iv2010-ld.pdf (cited on page 29).

DensityDesign [2022a]. *RAWGraphs GitHub Repository*. 15 May 2022. https://github.com/rawgraphs/rawgraphs-app (cited on page 5).

DensityDesign [2022b]. *RAWGraphs Online Documentation*. 15 May 2022. https://rawgraphs.io/rawgraphs-core/docs/api/ (cited on page 5).

DensityDesign [2022c]. *RAWGraphs Webpage*. 15 May 2022. https://rawgraphs.io/ (cited on page 5).

EC [2021]. *CO2 Output by Countries*. European Commission, 01 Jul 2021. https://publications.jrc.ec.europa.eu/repository/handle/JRC123071 (cited on pages 3–4).

Graz [2021]. *Graz Election Results*. 27 Sep 2021. https://wahlergebnis.graz.at/ (cited on pages 3–4).

IDL [2022a]. *Vega – A Visualization Grammar*. University of Washington, Interactive Data Lab, 16 May 2022. https://vega.github.io/vega/ (cited on page 11).

IDL [2022b]. *Vega Desktop GitHub Repository*. University of Washington, Interactive Data Lab, 16 May 2022. https://github.com/vega/vega-desktop (cited on page 12).

IDL [2022c]. *Vega Editor GitHub Repository*. University of Washington, Interactive Data Lab, 16 May 2022. https://github.com/vega/editor (cited on pages 11–12).

IDL [2022d]. *Vega Editor Online Tool*. University of Washington, Interactive Data Lab, 16 May 2022. https://vega.github.io/vega/about/ (cited on pages 11–12).

IDL [2022e]. *Vega GitHub Repository*. University of Washington, Interactive Data Lab, 16 May 2022. https://github.com/vega/vega (cited on page 12).

IDL [2022f]. *Vega-Lite GitHub Repository*. University of Washington, Interactive Data Lab, 16 May 2022. https://github.com/vega/vega-lite (cited on page 12).

IDL [2022g]. *Voyager Documentation*. University of Washington Interactive Data Lab, 16 May 2022. https://data-voyager.gitbook.io/voyager/ (cited on page 18).

IDL [2022h]. *Voyager GitHub Repository*. University of Washington Interactive Data Lab, 16 May 2022. https://github.com/vega/voyager (cited on page 17).

IDL [2022i]. *Voyager Online Tool*. University of Washington Interactive Data Lab, 16 May 2022. https://vega.github.io/voyager2/ (cited on page 17).

Martin, Patrick [2017]. *Dex Stable Version*. 14 May 2017. https://github.com/PatMartin/DexReleases/blob/master/dex-0.9.0.1.zip?raw=true (cited on pages 23–24).

Martin, Patrick [2019]. *Dex GitHub Repository*. 12 Feb 2019. `https://github.com/PatMartin/Dex` (cited on page 23).

Martin, Patrick [2022a]. *Dex Online Documentation*. 12 May 2022. `http://dexvis.net/docs/DexManual.html` (cited on pages 23–24).

Martin, Patrick [2022b]. *Dex Webpage*. 12 May 2022. `http://dexvis.net/` (cited on page 23).

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). ACM, 18 Sep 2017, 28:1–28:5. ISBN 1450352375. doi:10.1145/3125571.3125585 (cited on page 5).

Medium [2022]. *Charted GitHub Repository*. 15 May 2022. `https://github.com/charted-co/charted` (cited on page 2).

NZZ [2021]. *Q Election Seats*. Neue Zürcher Zeitung, 15 Nov 2021. `https://github.com/nzzdev/Q-election-seats` (cited on page 2).

NZZ [2022a]. *Neue Zürcher Zeitung*. 22 May 2022. `https://nzz.ch/` (cited on page 2).

NZZ [2022b]. *NZZ Editorial Tech Twitter*. Neue Zürcher Zeitung, 22 May 2022. `https://twitter.com/NZZEditoTech` (cited on page 2).

NZZ [2022c]. *NZZ Visuals Twitter*. Neue Zürcher Zeitung, 22 May 2022. `https://twitter.com/nzzvisuals` (cited on page 2).

NZZ [2022d]. *Q*. Neue Zürcher Zeitung, 22 May 2022. `https://q.tools` (cited on page 2).

NZZ [2022e]. *Q Editor*. Neue Zürcher Zeitung, 12 Apr 2022. `https://github.com/nzzdev/Q-editor` (cited on page 2).

NZZ [2022f]. *Q Server*. Neue Zürcher Zeitung, 05 May 2022. `https://github.com/nzzdev/Q-server` (cited on page 2).

NZZ [2022g]. *Q-Chart*. Neue Zürcher Zeitung, 13 May 2022. `https://github.com/nzzdev/Q-chart` (cited on page 2).

Satyanarayan, Arvind, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer [2016]. *Vega-Lite: A Grammar of Interactive Graphics*. IEEE Transactions on Visualization and Computer Graphics 23.12 (10 Aug 2016), pages 341–350. doi:10.1109/TVCG.2016.2599030. `https://idl.cs.washington.edu/files/2017-VegaLite-InfoVis.pdf` (cited on page 11).

Tiberghien, Mathias [2022]. *Pacman Example*. University of Washington, Interactive Data Lab, 16 May 2022. `https://vega.github.io/editor/#/examples/vega/pacman` (cited on page 14).

TUG [2021]. *Student Enrollment at Graz University of Technology*. Graz University of Technology, 2021. `https://online.tugraz.at/tug_online/studierendenstatistik.html` (cited on pages 3–4).

Wongsuphasawat, Kanit, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer [2015]. *Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations*. IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis) 22.1 (12 Aug 2015), pages 649–658. doi:10.1109/TVCG.2015.2467191. `https://idl.cs.washington.edu/papers/voyager` (cited on page 17).

World Bank [2020]. *World Population by Countries*. 2020. `https://data.worldbank.org/indicator/SP.POP.TOTL` (cited on pages 3–4).