

# An Individual-Centered Approach to Visualize People's Opinions and Demographic Information

Wanda Baltzer, Theresa Hradilak, Lara Pfennigschmidt, Luc Maurice Prestin, Moritz Spranger, Simon Stadlinger, Leo Wendt, Jens Lincke, Patrick Rein, Luke Church, Robert Hirschfeld

**Technische Berichte Nr. 136**

des Hasso-Plattner-Instituts für  
Digital Engineering an der Universität Potsdam







Technische Berichte des Hasso-Plattner-Instituts für  
Digital Engineering an der Universität Potsdam



Wanda Baltzer | Theresa Hradilak | Lara Pfennigschmidt | Luc Maurice Prestin |  
Moritz Spranger | Simon Stadlinger | Leo Wendt | Jens Lincke | Patrick Rein |  
Luke Church | Robert Hirschfeld

## **An Individual-Centered Approach to Visualize People's Opinions and Demographic Information**

### **Bibliografische Information der Deutschen Nationalbibliothek**

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.dnb.de/> abrufbar.

**Universitätsverlag Potsdam 2021**

<http://verlag.ub.uni-potsdam.de/>

Am Neuen Palais 10, 14469 Potsdam

Tel.: +49 (0)331 977 2533 / Fax: 2292

E-Mail: [verlag@uni-potsdam.de](mailto:verlag@uni-potsdam.de)

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam.

ISSN (print) 1613-5652

ISSN (online) 2191-1665

Das Manuskript ist urheberrechtlich geschützt.

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam

<https://doi.org/10.25932/publishup-49145>

<https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-491457>

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:

ISBN 978-3-86956-504-0

# Abstract

The noble way to substantiate decisions that affect many people is to ask these people for their opinions. For governments that run whole countries, this means asking all citizens for their views to consider their situations and needs.

Organizations such as Africa’s Voices Foundation, who want to facilitate communication between decision-makers and citizens of a country, have difficulty mediating between these groups. To enable understanding, statements need to be summarized and visualized. Accomplishing these goals in a way that does justice to the citizens’ voices and situations proves challenging. Standard charts do not help this cause as they fail to create empathy for the people behind their graphical abstractions. Furthermore, these charts do not create trust in the data they are representing as there is no way to see or navigate back to the underlying code and the original data. To fulfill these functions, visualizations would highly benefit from interactions to explore the displayed data, which standard charts often only limitedly provide.

To help improve the understanding of people’s voices, we developed and categorized 80 ideas for new visualizations, new interactions, and better connections between different charts, which we present in this report. From those ideas, we implemented 10 prototypes and two systems that integrate different visualizations. We show that this integration allows consistent appearance and behavior of visualizations. The visualizations all share the same main concept: representing each individual with a single dot. To realize this idea, we discuss technologies that efficiently allow the rendering of a large number of these dots. With these visualizations, direct interactions with representations of individuals are achievable by clicking on them or by dragging a selection around them. This direct interaction is only possible with a bidirectional connection from the visualization to the data it displays. We discuss different strategies for bidirectional mappings and the trade-offs involved. Having unified behavior across visualizations enhances exploration. For our prototypes, that includes grouping, filtering, highlighting, and coloring of dots. Our prototyping work was enabled by the development environment Lively4. We explain which parts of Lively4 facilitated our prototyping process. Finally, we evaluate our approach to domain problems and our developed visualization concepts.

Our work provides inspiration and a starting point for visualization development in this domain. Our visualizations can improve communication between citizens and their government and motivate empathetic decisions. Our approach, combining low-level entities to create visualizations, provides value to an explorative and empathetic workflow. We show that the design space for visualizing this kind of data has a lot of

potential and that it is possible to combine qualitative and quantitative approaches to data analysis.

# Zusammenfassung

Der noble Weg, Entscheidungen, die viele Menschen betreffen, zu begründen, besteht darin, diese Menschen nach ihrer Meinung zu fragen. Für Regierungen, die ganze Länder führen, bedeutet dies, alle Bürger nach ihrer Meinung zu fragen, um ihre Situationen und Bedürfnisse zu berücksichtigen.

Organisationen wie die Africa's Voices Foundation, die die Kommunikation zwischen Entscheidungsträgern und Bürgern eines Landes erleichtern wollen, haben Schwierigkeiten, zwischen diesen Gruppen zu vermitteln. Um Verständnis zu ermöglichen, müssen die Aussagen zusammengefasst und visualisiert werden. Diese Ziele auf eine Weise zu erreichen, die den Stimmen und Situationen der Bürgerinnen und Bürger gerecht wird, erweist sich als Herausforderung. Standardgrafiken helfen dabei nicht weiter, da es ihnen nicht gelingt, Empathie für die Menschen hinter ihren grafischen Abstraktionen zu schaffen. Darüber hinaus schaffen diese Diagramme kein Vertrauen in die Daten, die sie darstellen, da es keine Möglichkeit gibt, den verwendeten Code und die Originaldaten zu sehen oder zu ihnen zurück zu navigieren. Um diese Funktionen zu erfüllen, würden Visualisierungen sehr von Interaktionen zur Erkundung der angezeigten Daten profitieren, die Standardgrafiken oft nur begrenzt bieten.

Um das Verständnis der Stimmen der Menschen zu verbessern, haben wir 80 Ideen für neue Visualisierungen, neue Interaktionen und bessere Verbindungen zwischen verschiedenen Diagrammen entwickelt und kategorisiert, die wir in diesem Bericht vorstellen. Aus diesen Ideen haben wir 10 Prototypen und zwei Systeme implementiert, die verschiedene Visualisierungen integrieren. Wir zeigen, dass diese Integration ein einheitliches Erscheinungsbild und Verhalten der Visualisierungen ermöglicht. Die Visualisierungen haben alle das gleiche Grundkonzept: Jedes Individuum wird durch einen einzigen Punkt dargestellt. Um diese Idee zu verwirklichen, diskutieren wir Technologien, die die effiziente Darstellung einer großen Anzahl dieser Punkte ermöglichen. Mit diesen Visualisierungen sind direkte Interaktionen mit Darstellungen von Individuen möglich, indem man auf sie klickt oder eine Auswahl um sie herumzieht. Diese direkte Interaktion ist nur mit einer bidirektionalen Verbindung von der Visualisierung zu den angezeigten Daten möglich. Wir diskutieren verschiedene Strategien für bidirektionale Mappings und die damit verbundenen Kompromisse. Ein einheitliches Verhalten über Visualisierungen hinweg verbessert die Exploration. Für unsere Prototypen umfasst dies Gruppierung, Filterung, Hervorhebung und Einfärbung von Punkten. Unsere Arbeit an den Prototypen wurde durch die Entwicklungsumgebung Lively4 ermöglicht. Wir erklären, welche Teile von Lively4 unseren Prototyping-Prozess erleichtert haben. Schließlich bewerten wir unsere Herangehensweise an Domänenprobleme und die von uns entwickelten Visualisierungskonzepte.

Unsere Arbeit liefert Inspiration und einen Ausgangspunkt für die Entwicklung von Visualisierungen in diesem Bereich. Unsere Visualisierungen können die Kommunikation zwischen Bürgern und ihrer Regierung verbessern und einfühlsame Entscheidungen motivieren. Unser Ansatz, bei dem wir niedrigstufige Entitäten zur Erstellung von Visualisierungen kombinieren, bietet einen wertvollen Ansatz für einen explorativen und einfühlsamen Arbeitsablauf. Wir zeigen, dass der Designraum für die Visualisierung dieser Art von Daten ein großes Potenzial hat und dass es möglich ist, qualitative und quantitative Ansätze zur Datenanalyse zu kombinieren.



# Contents

<b>1</b>	<b>Domain and Specific Challenges of Visualizing Demographic Data and Personal Opinions</b>	<b>1</b>
1.1	Domain . . . . .	1
1.2	Project Description, Prerequisites, and Requirement Analysis . . . . .	12
1.3	Developing Customized Visualizations: State of the Art . . . . .	17
1.4	Our Approach . . . . .	23
1.5	Our Contribution . . . . .	24
<b>2</b>	<b>Concepts for Visualizations and Exploration and Categorization of the Design Space</b>	<b>27</b>
2.1	Introduction . . . . .	27
2.2	Exploration of the Design Space and Ideation . . . . .	28
2.3	Categorization and Analysis . . . . .	36
2.4	Concepts . . . . .	56
2.5	Conclusion . . . . .	63
<b>3</b>	<b>Implementation and Integration Into an Environment of Explorable Visualization Tools</b>	<b>65</b>
3.1	Introduction . . . . .	65
3.2	Individuals as Points . . . . .	67
3.3	Interaction Patterns . . . . .	69
3.4	Visualizations Tools . . . . .	77
3.5	Integration of Tools . . . . .	91
3.6	Conclusion . . . . .	101
<b>4</b>	<b>Using the Lively4 Platform with Its Active Content Capabilities</b>	<b>103</b>
4.1	Introduction . . . . .	103
4.2	Lively4 System Introduction . . . . .	104
4.3	Technical Capabilities of Lively4 . . . . .	108
4.4	Collaborating in Lively4 . . . . .	129
4.5	Conclusion . . . . .	138
<b>5</b>	<b>Mapping of Data and UI for Interactive and Explorable Visualizations</b>	<b>139</b>
5.1	Introduction . . . . .	139
5.2	Data and Provenance . . . . .	140
5.3	Bidirectional Mapping . . . . .	145
5.4	Evaluation of Existing Strategies for Bidirectional Mapping of Data and User Interface . . . . .	147

5.5	Our Approach on Bidirectional Mapping . . . . .	157
5.6	Conclusion . . . . .	166
<b>6</b>	<b>Evaluating Visualization Technologies for Individual Data Points in Lively4</b>	<b>169</b>
6.1	Introduction . . . . .	169
6.2	Visualization Environment and Approach . . . . .	170
6.3	Graphics in the Browser . . . . .	173
6.4	Rendering Visualizations with Many Points in Lively4 . . . . .	182
6.5	Benchmarking Web Technologies . . . . .	186
6.6	Future Work and Further Optimizations . . . . .	200
6.7	Conclusion . . . . .	201
<b>7</b>	<b>Evaluating Our Approach to Visualize People’s Opinions and Demographic Data</b>	<b>203</b>
7.1	Introduction . . . . .	203
7.2	Foundations . . . . .	204
7.3	Walkthroughs . . . . .	207
7.4	Value-driven Evaluation . . . . .	218
7.5	Discussion . . . . .	223
7.6	Conclusion and Outlook . . . . .	228
<b>8</b>	<b>Conclusion</b>	<b>231</b>
<b>Appendices</b>		
<b>A</b>	<b>Appendix Chapter 2</b>	<b>235</b>
A.1	Idea Collection . . . . .	235
A.2	Technical Categorization . . . . .	274
A.3	Task Categorization . . . . .	275
A.4	Interaction Level - Task Categorization . . . . .	276
A.5	Representation Mode - Task Categorization . . . . .	277
<b>B</b>	<b>Appendix Chapter 5</b>	<b>279</b>
<b>C</b>	<b>Appendix Chapter 6</b>	<b>287</b>
C.1	Benchmark protocol . . . . .	287
C.2	Benchmark Results . . . . .	289
C.3	Tables . . . . .	301
C.4	Code . . . . .	302

# 1 Introduction to the Domain and Specific Challenges of Visualizing Demographic Data and Personal Opinions

In this chapter, we will illustrate the context of this project in order to provide background for the subsequent chapters. Our project partner Africa's Voices Foundation works on developing communication capabilities between decision-makers and citizens in African countries. When communicating their results, they face challenges regarding empathy, exploration of data and generalization of findings, because they have to use standard charts. For the development of new visualizations that can be used, we will look at prerequisites and requirements from Africa's Voices and potential users. The challenge to provide customizable visualizations to different user groups has already been tackled by previous research, of which we will present three examples. Finally, to provide an overview of our approach, we briefly describe our process and our results, as well as our contribution to Africa's Voices' processes.

## 1.1 Domain

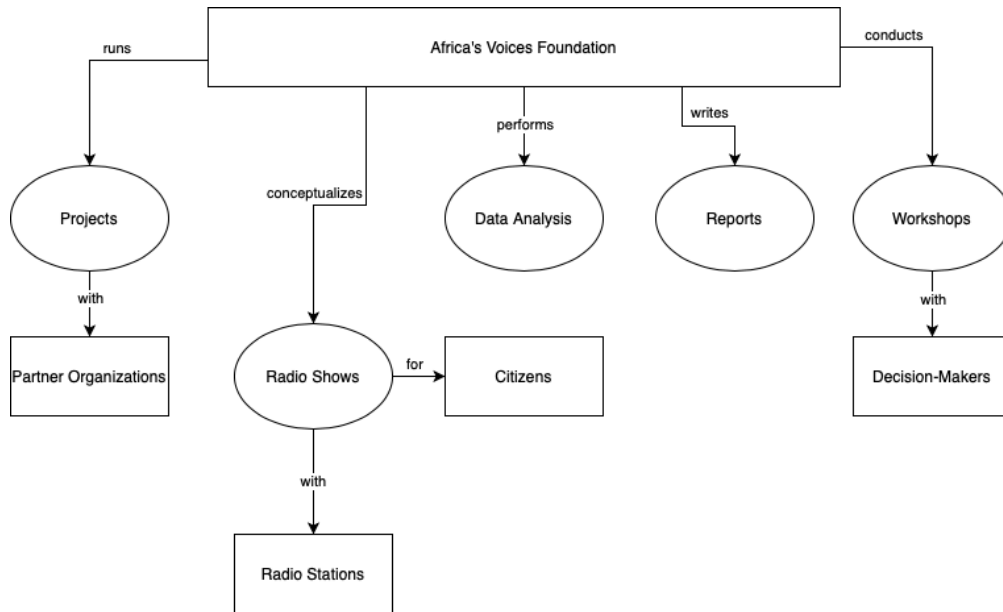
This section aims to introduce the reader to the domain of our project. It will explain who our project partner was, how they worked, and what challenges they faced.

### 1.1.1 Africa's Voices Foundation

Africa's Voices Foundation<sup>1</sup> is a non-governmental organization (NGO) that helps to develop communication capabilities in African countries such as Kenya and Somalia. On their website, they state: "By combining media and technology, our tested methods listen intelligently and amplify diverse, local **voices**". The communication between residents and decision-makers of a country is often difficult enough due to thousands or millions of different opinions about one topic. But in most cases, there is no way in which citizens can voice their opinion directly to decision-makers. Respectively, there is no way in which decision-makers can ask all citizens for their personal opinion. Africa's Voices wants to build a bridge between citizens and decision-makers so they can communicate with each other. The decision-makers

---

<sup>1</sup><https://www.africasvoices.org> (last accessed: 2020-07-29).



**Figure 1.1:** Context of Africa's Voices' work

need to know what citizens think and citizens want to be heard and considered by their government. Africa's Voices does this by helping both sides:

1. They collect opinions and voices of citizens, giving them an opportunity to speak their minds without confined answering possibilities and letting them be heard and understood.
2. They conduct research and interpret the collected voices into findings and work out policy recommendations based on real opinions of the society. They want to deliver up to date, actionable, and robust evidence to decision-makers to let them understand the dynamics, opinions, and fears of the community.
3. They give their findings back to the citizens and give them information about the views of experts and decision-makers, enabling communication in both ways. They also curate public spaces in which these discussions are held.

### 1.1.2 Process of Africa's Voices

We will now describe parts of Africa's Voices' work and their processes displayed in Figure 1.1.

#### 1.1.2.1 Radio Shows

To reach as many residents as possible, Africa's Voices relies on many local radio stations. With their help, they conceptualize radio shows containing questions to the audience, information about running projects and found insights as well as interviews with policymakers and service providers. They ask open questions that encourage citizens to answer with their own opinion in their local language. These answers can be submitted via text messages. Once a person responds, the purpose

of the radio show is clarified and their consent for saving and analyzing their data is sought. They then get follow-up questions asking for their age, gender, location, and other project or country-specific categories. This way, researchers can later look for trends in different population subgroups like age groups or regions.

The interviews, as they are made public, are important to enable a second form of communication and to let decision-makers talk to the community. This way they can directly respond to citizens' voices that are most common or most relevant.

In the radio shows, the hosts can also broadcast their own findings as well as insights from partner organizations back to the participants to inform the citizens. Additionally, they can announce promises or views stated by policymakers to increase their accountability.

#### **1.1.2.2 Projects**

Multiple radio shows thematically belong to one project, which runs over several weeks or months to investigate specific political or humanitarian subjects. Projects are carried out with varying partner organizations. The shows are dynamically tailored for the occasion, so that they are very effective for getting many opinions from people.

#### **1.1.2.3 Data Analysis and Visualization**

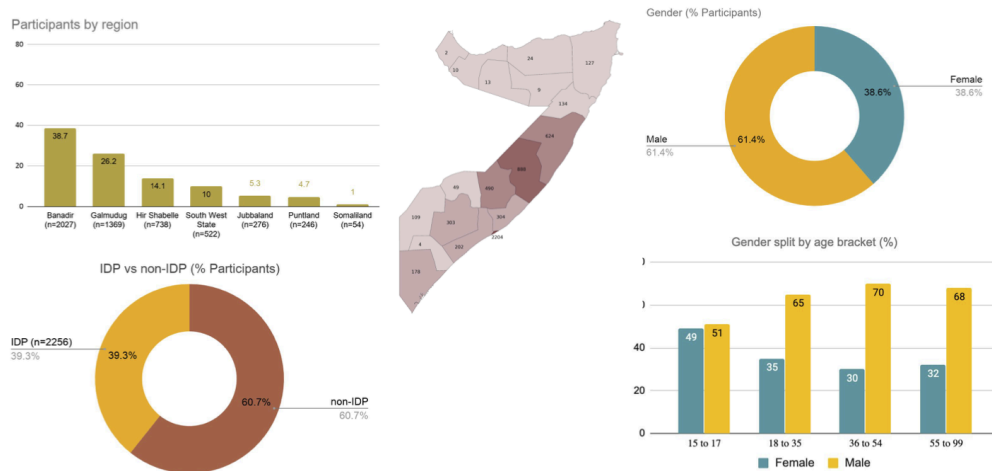
The data analysis approach of Africa's Voices combines social science, computer science, and most importantly a human understanding of both languages and context.

**Labeling** When Africa's Voices has collected texts from participants, they start labeling them. This is done by employees who know many dialects and idiomatic expressions of different regions in the country. They label the messages with common themes that occur in the answers. Depending on the project, these themes can also fall into different levels, where a tag on the first level is very general, whereas a tag on the next level makes a first level tag more specific. These sometimes are necessary, for example tagging a message with "government" does not convey what sentiment the person has towards the government. Here a next level tag can provide further explanation, like "satisfied with government" or "no trust in government".

The themes are obtained through a literature review on the project's topic. In the best case, this is done together with a domain expert. Then they do an open coding exercise, applying the knowledge from the literature review to the received quotes to understand what they are saying and to check for found themes. This is done with a handful of employees who have to label a couple of messages. They then cross-check the results to find out whether the employees have tagged messages the same. If that was successful, they run that tagging scheme on the complete data set. This tagging is performed by researchers using tools that make use of machine learning, for example for clustering data together. For Africa's Voices, it is important to include humans in every step of the process, as artificial intelligence can be biased and make categorical errors which then skew the data. Not including humans would also decrease the degree to which the researchers understand the data, which is crucial for this method.

**Documenting** This whole process is logged in the metadata for each message so that translations, changes, and categorizations can be traced back to the original message. This history of data points is called provenance information, which Africa’s Voices can use to find out where exactly a specific data point came from and what processes have been applied to it.

**Visualizing** After all statements have been labeled, Africa’s Voices starts to develop visualizations for the data, which takes up many GB. They run a couple of python scripts to produce an approximately 250 MB CSV file. This file is then put into Excel to produce visualizations with standard Excel visualizing functions. They also send that data to their resident statistician, who analyzes the data regarding all common statistical questions, like covariance analysis of attributes.

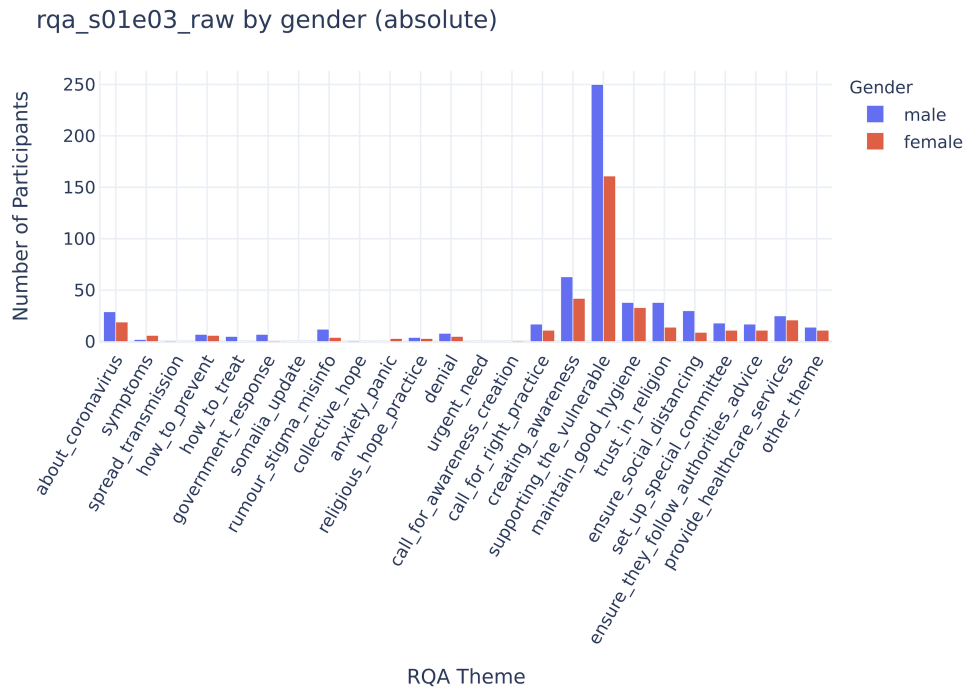


**Figure 1.2:** A variety of visualizations used in the project ‘Somali Views In The Early Days Of Covid-19: A Rapid Diagnostic’<sup>2</sup>. It includes bar and donut charts for displaying the demographic data of individuals (upper left for region, lower left for internally displaced status, upper right for gender and lower left for gender by age group) as well as a heat map for showing the local distribution of participants (center).

To view the local distribution of all participants, they use heat maps (center visualization in Figure 1.2), where every district gets a color shade according to the number of people responding and living in that district. For clarification, each district gets that number as an annotation.

To view demographics like age and gender, they use standard bar charts or donut charts for analysis (see left and right visualizations in Figure 1.2). From all quotes,

<sup>2</sup><https://www.africasvoices.org/case-studies/somali-views-in-the-early-days-of-covid-19-a-rapid-diagnostic/> (last accessed: 2020-07-29).



**Figure 1.3:** An exemplary bar chart displaying the amount of different themes mentioned by males (blue) and females (red).

they extract the most representative ones to use them for further shows, reports, or workshops with decision-makers.

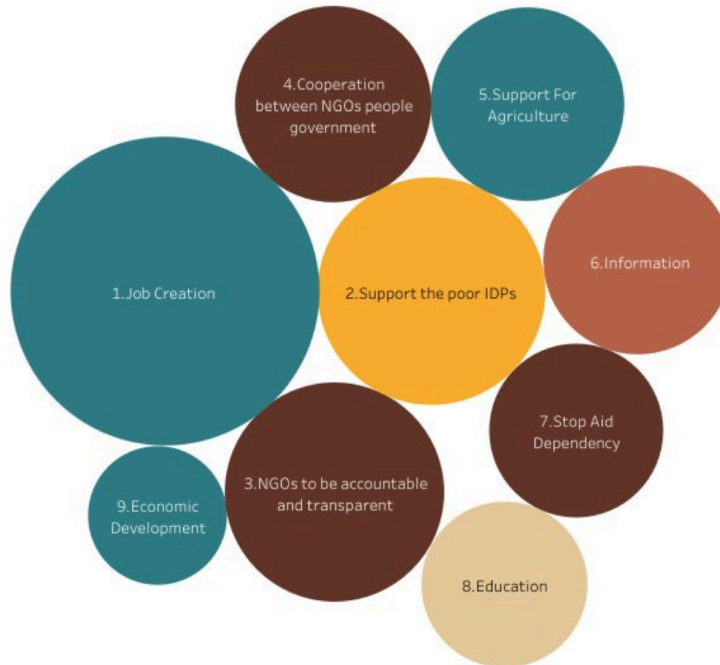
To view the distribution of themes, they use heat maps for viewing the local distribution of single themes and multi-dimensional bar charts as seen in Figure 1.3 for other demographic distributions of multiple themes.

**Analyzing** The researchers then analyze the situations of the citizens, trying to find any trends or relationships between an individual’s demographics and their statements. From these insights, Africa’s Voices develops recommendations for policymakers, including what they need to understand before acting or communicating. Additionally, they want to find out where people acquire their information from and who they listen to, so that, in case of informing the public, they know who they should work together with in order to reach the most critical groups or as many citizens as possible.

#### 1.1.2.4 Reports

Depending on the length of the project, Africa’s Voices writes one final report or a number of interim reports for their project partners. Here, they include their findings as well as supporting visualizations of the data. In their reports, they use example quotes from participants to support explaining the occurrence of a theme and to make the visualizations more personal. They need to convey that they are talking

about people's opinions and they want to create empathy in readers. Alongside exemplary quotes, they use bubble charts signaling the frequency of themes as can be seen in Figure 1.4 as well as aggregated data and percentages in bar charts and pie charts.



**Figure 1.4:** An exemplary bubble chart showing the frequency of themes<sup>3</sup>

#### 1.1.2.5 Workshops

The most important part of Africa's Voices' work is presenting the data and their findings to decision- and policymakers. They want to explain the respondents' situations as well as their views and backgrounds, so decision-makers can understand and empathize with the community to have a foundation to base decisions on: the real opinions of citizens. For that, Africa's Voices provides them with their developed visualizations as well as example quotes to support their statements and recommendations. The recommendations include context information to keep in mind, situations to consider, and tips on how to develop programs to help. For these programs, it is important to know how and what to talk to people about as well as who is most likely to respond, take part, and get the most out of it.

<sup>3</sup>[https://www.africasvoices.org/wp-content/uploads/2019/10/Africas-Voices-AAP-Consultation-\\_-Briefing-Note\\_FINAL.docx-1.pdf](https://www.africasvoices.org/wp-content/uploads/2019/10/Africas-Voices-AAP-Consultation-_-Briefing-Note_FINAL.docx-1.pdf) (last accessed: 2020-07-29).



These workshops often consist of presentations alongside slides with the aforementioned quotes, visualizations, findings, and recommendations. Afterwards, presenters encourage participants to take part in an open discussion to hear the side of the decision-makers and to develop solutions together. Africa's Voices also conducts more interactive workshops where the labeling is done together with attendees to get even more specific themes. Afterwards, they run the pipeline and explore the data together.

### 1.1.3 Problems

Summing up, Africa's Voices uses visualizations for understanding the data and views of the citizens, and for presenting it to decision-makers. Currently, they use standard and static visualizations that do not help their cause as well as visualizations could. Because one cannot interact with them, these visualizations do not support analyzing the data more deeply or making connections between demographic information and the received messages easily. Also, with static visualizations, it is harder to present findings in a logical and comprehensible way to decision-makers. This is Africa's Voices' main problem. Furthermore, they want to introduce new methods and new ethics in their work, which then lead to new understandings but also new communication challenges. In the following, we will look at the most relevant problems in detail.

#### 1.1.3.1 Slow Feedback Cycles Due to Their Current Process

Africa's Voices' current process of producing visualizations is very slow. This is why researchers only want to do it once they have all the data, because doing it multiple times takes too much time. This prevents visualizing data that changes over time. Africa's Voices cannot build real time applications that could show what messages are received when. Especially for projects that run for several months, this approach is harmful: only visualizing everything that they have at the end excludes the possibility of discovering something from interim results. For example, there could be one opinion prevalent at the beginning of a project and another one at the end. They already analyze the situation at different times and compare them for changes, but as the process is slow, results would come too late to intervene.

With a slow process when creating visualizations, there comes the problem of not having time for other important steps like the preparation of workshops, research, or writing final reports.

Added to that, the process itself might have issues, which they will find out too late, if they can only start analysis after having collected all data. Issues could include radio shows that are not working or topics that are not resonating, as well as misunderstood questions or simply targeting the wrong audience at the wrong time.

This process also cannot easily be automated, because every project collects specifically tailored and differently structured data.

To improve their pre-analysis process, employees of Africa's Voices rewrote their pipeline with the provenance tracing to make it more efficient and to make real

time tracing possible. They now use the python library Matplotlib<sup>4</sup> to create static visualizations from real time data. They also developed an automated analysis export, which answers the most important statistical questions. For visualizations for reports and presentations, they still have to go back to CSV files and Excel, as the image quality of the images from the python library is not good enough.

With this outlook on real-time data analysis and visualization, keeping the history of the data and the visualizations becomes more important to prove everything is working correctly. Additionally, real-time analysis could help to find time relevant trends more quickly.

### **1.1.3.2 Generalization of Findings Due to Quantitative Data**

The standard visualizations such as pie, bar, and bubble charts can display aggregated data. Aggregating data means counting all data points, calculating the average value, or showing absolute and relative occurrences of values within the data set. For data to be aggregated, it needs to be structured, meaning that every data point includes the same structure of information and the same categories. This way, data can be measured and thus falls into the definition of quantitative data. That also means that, among all data points, there exists a finite set of values the data points adopt, which makes them groupable by their values. So these types of charts all support the visualization of quantitative data like the demographic information of all individuals.

With aggregated data, only one value is calculated and displayed per variable for a group of data points. This could also be applied to the whole data set, reducing all individuals to a single average or distribution of values. Using these aggregates during analysis reduces complexity of the data that needs to be understood, but it is questionable whether this should be done when dealing with a group of individuals. Every human has their own mind, context, history, and needs, and too much abstraction from this complexity might be inconsiderate.

Another fact to keep in mind is, that Africa's Voices is working with a self-selected group of participants. Only messages from people who have access to radio and SMS, who can read and write, and who are willing to voice their opinion on a subject can be received. Researchers might forget that fact if working with standard charts. Especially pie charts could convey the idea of displaying 100% of the population, whereas these charts only display the received data originating from a subgroup of the population. This could lead to invalid generalizations to the whole population of a country. Africa's Voices already uses bubble charts instead of pie charts to circumvent having that image of displaying all of the population (see again Figure 1.4). Hence, dealing with these issues of abstractions and generalizations requires visualizations that remind the researchers of the true nature of their data.

---

<sup>4</sup><https://matplotlib.org>.

### 1.1.3.3 Viewing Single Individuals and Original Opinions

Standard visualizations can only show demographic information of all individuals in the data set. They do not support displaying a specific individual with their demographics, let alone their opinion. Opinions as free text are unstructured: every message references different topics, has a different grammar, some even are in different languages. This means, the messages can be categorized as qualitative data. There are so far no well-known tools which can help with displaying this amount of qualitative data clearly and effectively. For analyzing the data of 2,000 to 40,000 individuals, researchers need to categorize and even aggregate single opinions at some point, because they cannot get any findings by reading all quotes and not sorting and grouping them into common themes. That is the reason why researchers label and thereby summarize the messages. Through this labeling process, the messages are structured and can be aggregated according to their themes, so they can be visualized through standard charts. But, by doing that, the connection from the label back to the original quote it was applied to is lost, so users cannot view it. This results in loss of the original data in the visualization and – most importantly – doing that inhibits curiosity and empathy.

### 1.1.3.4 Trust in Data

**Correctness of Labeling** The aforementioned labeling process is necessary to compute the amounts of qualitative quotes Africa’s Voices receives. However, the connection between the original quote and the label or tag needs to be maintained to make data modifications traceable and reversible. For the quotes, this is important to explain and confirm the correctness of the labeling process as well as to support creating empathy by reading original statements. Therefore, it is necessary to log changes and additions to and interpretations of the data to ensure data integrity.

**Filtering** The history log is also important for other use cases. Sometimes researchers only want to look at specific parts of the data, so they filter out the rest. With standard visualizations, researchers often have to provide complete data sets that they want to visualize, which means that they have to do the filtering themselves. This may be possible for researchers who can program but much less possible for researchers needing a graphical user interface (GUI) to do that. Added to that, in the produced visualization the filters that got applied to the data are not visible. This is either because it is not supported by the software itself or the researcher had to filter the data beforehand.

In Africa’s Voices’ former workflows, filters got applied through the filter function in Excel and the resulting data was copied into a new spreadsheet. However, copying data opens up a big risk of having inconsistent and unreliable data. If changes are introduced to one sheet of data, but not the other, the data becomes inconsistent. Using one datasheet or the other results in different outcomes, making the data unreliable. As researchers at Africa’s Voices copied data for filtering, this process required a lot of checks to ensure whether the filter worked correctly. Confirming the correctness was difficult to do with standard charts that have no connection back

to the original data. For this reason, there has been a strong trend at Africa's Voices to eliminate this kind of practice and to replace it with more reliable tooling.

**Aggregates** With grouping or aggregating data, the connection from the group or aggregate back to the data points it was made of, is lost in static visualizations. Hence, if Africa's Voices wants to create trust in their visualizations and in their data, they need to make sure that the history of data aggregations are logged and displayed, so that they are understood by viewers and can be reversed by researchers.

**Direct Modifications** For data cleaning, researchers sometimes need to modify data directly by replacing or grouping values together. This is to be used very cautiously, as faulty scripts introduce new errors in the data and humans may also make mistakes by doing it by hand. To keep data integrity and to prevent data scandals, which change data on a large scale, Africa's Voices logs all modifications done by researchers and scripts in the metadata of a message.

#### 1.1.3.5 Trust in Software

To develop visualizations, researchers need to use software created by others if they do not want to program their visualization software themselves. In this case, they have to trust the software, that it does not modify data while processing and displaying it. If users were able to trace the visualization back to the code and the data it uses, that could reduce skepticism and increase trust. Africa's Voices already built a solution for their current process, which attaches SHA-sums of the code that created the data to that corresponding data.

#### 1.1.3.6 Connecting Visualizations

There is no unifying visualization that can explain everything and every angle, so researchers need a set of visualizations to get answers for specific questions. Usually, they take an ideal diagram for each use case and combine them by laying them out next to each other.

**Using the Same Data** These diagrams, however, are not obviously connected to each other. Users cannot even directly see, whether they contain the same data, if the visualizations are not annotated.

**Using Different Elements** Assuming multiple visualizations do use the same data, they still have no visible connection, if they are from different types. Different types of visualizations usually display different aspects of data, which makes it impossible to directly connect the data in one chart with the same data in another. For example, in a bar chart, which displays the amount of people in different regions, and a donut chart, which displays the amount of people in different genders, you cannot see, which data points from a specific region in the bar chart make up which parts of a specific gender group in the donut chart (see again the top left and top right chart of Figure 1.2). Going even further, with a joint interpretation of quantitative and qualitative data you cannot see, what quote comes from which age group or which

gender, or which quotes make up an age group. There are no elements in common that are linked between the visualizations because each standard chart uses its own graphical elements for its own purpose.

**Using Different Mapping of Elements** If multiple visualizations do use the same elements, such as two bar charts, it is still difficult to connect the data in these two visualizations. The problem is that the same elements – the bars – are standing for different aspects. They can represent the amount of people by age or by gender, so the functions of the bars change to visualize different attributes. A bar stands for an amount, but the sources may be different. This requires users to remap the same graphical elements to different meanings, making it more complex to view and compare.

**Using Different Color Schemes** Assuming now that multiple visualizations do use the same elements with the same meaning, they still might have different color schemes, making it difficult to compare them. This may not be such a big problem for researchers as they have domain knowledge, practice and have probably developed the visualizations from the data themselves. But for people without a statistical or analytical background, this mapping might be too much to keep in mind, so they cannot follow explanations and arguments well. Thus, it is easier to understand connections in the data, if different visualizations are explicitly connected. Added to that, it helps if they look uniform and support the same kind of interactions.

#### 1.1.3.7 Missing Interaction

With static charts, users only have this one view of the data. If there is no interaction possible, they lose the chance of understanding and finding out more. They would have to load the whole data into a fitting diagram and build it with every new question that arises. It is like a box that you can fixedly view from all six sides, but only by interacting with it, you can look inside and see its inner workings. Interacting with data could be most helpful for researchers as they then have more tools to view the data. They could let their interest and intuition guide their exploration, which would lead to a completely different exploration and analysis process. On the other hand, interactive diagrams could significantly help people, who get presented the data, to understand the relationships between multiple charts or views of the data. For example, it could help if all visualizations could morph into one another through an animation. This would enable people to comprehend how the data that made up one chart also makes up the next and how these two views are connected. Also, when they have spontaneous questions, these could be explored and answered easily by interacting with the presented visualization itself.

#### 1.1.3.8 Missing Empathy

Most charts concentrate on visualizing their data best for a specific task, for example, comparing two or more groups with a bar chart. In our case of survey data, Africa's Voices also needs to get across that they are displaying real experiences and characteristics of people and that it is important not to forget that during the

process of analysis. When dealing with this kind of sensitive data, there are people who dislike numbers and aggregated data, because they know that there are people and stories behind the data and it makes them uncomfortable seeing them being reduced to bars and numbers. But there are also people who like numbers better than complex data because it simplifies and reduces most questions to a single aspect: Which side is bigger? What are most people saying? This analysis risks being too simple, as communities can be complex social structures. Visualizations, especially in this context, need to create empathy in the people who view them, because they might base decisions on this data. There are not many visualization tools out there that help with this kind of political aspiration.

#### **1.1.3.9 Issues With the Mindset**

When using standard charts, researchers use a familiar set of tools to develop a familiar set of graphs, so the mindset of looking at these graphs will stay the same. Because they are standard, they appear everywhere: on the news, in schools, in bills; anywhere you have to visualize quantitative data. Using standard visualizations results in standard questions being asked and a standard mindset towards the data: statistical, dry, even boring. Clearly, this is not a mindset with which Africa's Voices likes the public to view sensitive data about people and their opinions. But also using completely new visualizations does not help here because people usually do not want to spend a lot of time to understand a new chart. Using new, intriguing, and easy to understand charts could help change the mindset of viewers.

The presented understanding of the domain and associated problems resulted from conversations with our contact person at Africa's Voices Foundation. With these problems in mind, we will now analyze the goal of our project and its requirements in section 1.2. Also, we will look at state-of-the-art technologies for developing visualizations in section 1.3.

## **1.2 Project Description, Prerequisites, and Requirement Analysis**

This section describes the goal of our project, our prerequisites, and the requirements that resulted from different stakeholders. These stakeholders include our project partner (Africa's Voices) and their views, the data they provided, and potential users of our developed software, as well as the Software Architecture group at Hasso-Plattner Institute who supervised the project and provided the development environment Lively4.

### **1.2.1 Project Goal**

Originally, the project has been titled "Exploring Provenance through Programming". Exploring means to interactively look into something by comparing, connecting and investigating certain aspects to be able to discover insights. That is, the main goal

of the project was to enable the investigation of the provenance of data: to find ways of visualizing that history of data modifications and trace it back to its origin. The specification included “building a platform to develop new and unforeseen visualizations” and constructing several examples that could be used by Africa’s Voices during the course of the project and for future work.

Before we could build a platform, we had to understand what the main problems were and what could be most useful to our project partner. It turned out that it was not only exploring the provenance but in general creating visualizations, that used a more qualitative approach to enable exploration of the quotes of the respondents. These new visualizations could then help researchers to find new insights in the data, ask and answer new questions, as well as change the mindset about viewing the data from doing standard procedures to exploring at will. Researchers would be able to inquire for what sparks interest and could benefit from further investigation, without being stopped before reaching the original data. By making these visualizations intuitive and easy to use, they could also help Africa’s Voices to better communicate their findings in workshops. Presenters would have more interactive tools at hand to narrate and explain trends in the data and viewers would understand better and could empathize with single individuals but also with the whole population. A visualization system that combines many new visualizations into a single application with the same underlying concepts and interactions would significantly help solve the main problems of the visualization process of Africa’s Voices. It would therefore increase trust in the organization and their data while keeping individuals in the center of processes and visualizations to create empathy.

## **1.2.2 Prerequisites and Requirements**

### **1.2.2.1 Project Partner’s Views**

Originating from their most important problems, our project partner expressed the demand to focus on keeping all individuals visible and accessible. The individuals should be the basis for each visualization from which you can then aggregate or group. To help with creating empathy, they wanted us to create something lively, something that conveys that there are real people behind the data. In their quest for data integrity we were expected to provide something that could explore the history of a data point and also to add our data modification to that history.

### **1.2.2.2 Provided Context Information**

Our contact person at Africa’s Voices provided us with information on the data collection process of the organization and what they ultimately do with their visualizations. For our project, he wanted us to be completely free in choosing directions we should explore, so he gave us no detailed description on what exactly it is that we should build. Nor did he explain their current visualization process or what their used visualizations looked like. He wanted us to create new visualizations and interactions that would benefit parts of their process, whether it being helping with research, presenting, or making the data provenance explorable.

### 1.2.2.3 Data and Data Scheme

The data Africa's Voices sent us was in a JSON or CSV format (more in chapter 5). It contained demographic information about each individual, including their age, gender, location of residence, whether they were recently displaced in case of Somali citizens, and other project relevant attributes. The data also, originally, contained the message the individual sent as well as the main themes that occurred in that message. The messages themselves were replaced by lines from popular books ("Pride and Prejudice" and "Alice in Wonderland") for data security and our emotional safety, as some quotes might have been emotionally stressful to read. The first data sets also included the provenance trace of each message, where every script that ran on the data logged itself and the changes it made. Later on, we only got the demographic information and the themes, as we only used the provenance information in our ideas, but not in our prototypes. We got data from several projects: one project in cooperation with UN OCHA<sup>5</sup> about humanitarian needs in Somalia and about developing a Common Social Accountability Platform and another project about thoughts and opinions on the Coronavirus in Kenya and Somalia in 2020.

### 1.2.2.4 Data Handling and Data Visualization

**Data Handling** The data received also demanded specific handling. We had to work with high dimensional and very sensitive, personal data. The themes data was especially important as it was the main focus of analysis while being difficult to visualize, as one quote could have multiple themes. Also, we had to deal with incomplete data, where people had omitted information, or data from people who revoked consent of their data being used.

**Data Visualization** We had to display data from thousands of individuals and also keep them clearly visible. Our visualizations had to accommodate different project related attributes and different origins while also enabling comparison between similar projects in different countries or different projects in the same country. Also, we needed to exclude attributes, which were used only internally, like IDs, from analysis.

### 1.2.2.5 Potential Users

In the problem section, we already described the two types of users that would be using our project results: researchers who are trying to find new insights and presenters, who are showing the data to policymakers. Researchers again can be divided into two categories: the ones with knowledge in programming and IT development, and those who use but cannot develop such tools. Furthermore, there could be users exploring the data themselves with guidance from others, but without a background in data analysis. These users could be policymakers getting a feel for the data themselves, which would result in even greater empathy or even the citizens who took part. These types of users all have different abilities and constraints,

---

<sup>5</sup><https://www.unocha.org> (last accessed: 2020-07-29).



interests, wants, and needs. This resulted in several requirements for our software we have developed during the project. We will now explore the potential user groups and their respective requirements in detail.

**Researchers (End Users)** Users who want to conduct quantitative and qualitative analysis by exploring the data through a graphical user interface.

- *Abilities.* End users who want to research the data will have a thorough understanding of data analysis and knowledge of social context. Also, they might have experience with commonly used software for data analysis and visualization.
- *Constraints.* They usually have no knowledge in programming.
- *Wants.* They might want to filter or select certain data as well as change the representation of data through changing the visualization type or coloring and grouping of data points. They might want to save a state of a visualization so they can come back to it later or send that state to a colleague for further exploration. For writing reports, they might want to export the visualizations in a usable file format. For further analysis and exploration, they might want to add new visualization types or interactions.
- *Needs.* They will therefore need a graphical user interface, that provides filtering, selecting, grouping, and coloring of data points. It also has to offer different visualization types and enable switching between them. It needs to enable saving states of exploration stages or settings made. For adding new visualization types, it needs to provide a toolbox to instantiate new visualizations. This user interface has to be intuitively usable and, ideally, should be similar to already well-known tools. Interaction with the data or the visualization should be easily discoverable.

**Researchers (Developer Users)** Users who want to analyze and explore the data and who can program.

- *Abilities.* Researching Developer Users will also have understanding of data analysis and knowledge of social context. As developers, they know how to program and might also know several commonly used programming languages.
- *Constraints.* They might not know every programming language there is, nor do they have the time to read and understand the entire codebase.
- *Wants.* Additionally to researching end users, they might want to adjust existing visualization types and incorporate new ones. They also might want to add interactions to single visualizations or the whole system, which can then be used by end users.
- *Needs.* The software should be written in a commonly used programming language and provide sufficient documentation to facilitate understanding of the software. It should provide a consistent API for visualizations that makes it easy to develop new visualization types and interactions and incorporate those into the existing system. Also, the code should be quickly and directly accessible, possibly even from within the system.

**Presenters** Users who use the tool to present data, trends, and findings.

- *Abilities.* Presenters will have good knowledge of social context and, ideally, understanding of data analysis. They might be familiar with typical presentation tools such as PowerPoint or Keynote. During a presentation, they know what kind of interactions the tool offers or at least, what interactions they want to use.
- *Constraints.* During a presentation, they might not have a lot of time to interact with a tool to develop their point. Also, they do not want to explain the software, new visualization types, or interactions to viewers.
- *Wants.* They want to show the data and narrate insights to others. Their explanation should be backed by visualizations that can be understood by the audience (viewers). They might want to load a prepared state of data visualization and exploration. Also, they might want to immediately explore questions arising from their audience.
- *Needs.* The tool should have a presentation mode, where distracting settings or controls can be hidden. It would also be advantageous, if the tool provided help with creating presentations in the form of slides with animations, scripts, or videos. As presenters know what the tool can do, its functionality does not need to be especially discoverable for them.

**Viewers/Passive Policymakers** Users who do not interact with the tool directly, but are presented the data with this tool.

- *Abilities.* Policymakers will be familiar with viewing graphics and charts and they will have knowledge of the social context.
- *Constraints.* They might only know standard visualizations and do not want to spend time understanding new visualizations.
- *Wants.* They want to understand the data and their visualization, as well as the presenters' explanations and their findings.
- *Needs.* For them, the visualizations have to be visually comprehensible at one glance. Therefore, the tool should show what happens during exploration so that they can follow the presenter and do not get confused.

**Active Policymakers** Users who interact with the tool themselves to confirm or disconfirm their own hypotheses.

- *Abilities.* Same abilities as passive policymakers or viewers.
- *Constraints.* Additional to the constraints of a viewer, an active policymaker might have no background in data analysis nor a particular understanding of IT systems.
- *Wants.* They might want to understand and empathize with citizens. Therefore, they might want to explore the data for their own interest by filtering, selecting, coloring, or grouping data like a researcher. They might also want to export visualizations for future reference.
- *Needs.* For policymakers who can use the tool themselves like a researcher, it needs to be self-explaining. It should be responsive, performant, stable, and error-resistant.

**Citizens** Users who participated in a project by sending their opinion.

- *Abilities.* The abilities of citizens vary widely; we can only assume that they have knowledge of their country and community.
- *Constraints.* They might have little experience with viewing diagrams and charts and interacting with these.
- *Wants.* Citizens might want to find their own message as well as similar messages or answers that came from their region.
- *Needs.* For citizens, the software has to satisfy the needs of active policymakers as well as show locally and semantically similar answers. For the integrity of the citizen's data, it should not be changed and for privacy concerns should not identify them.

#### 1.2.2.6 Lively4

Our project was supervised by the Software Architecture Research Group at Hasso-Plattner-Institute (HPI). This group has worked on the Lively Kernel,<sup>6</sup> a live environment for documenting, programming and executing written code on the web. Lively4 has a wiki-like structure, a folder browser, a text editor for Markdown and JavaScript as well as a view of the execution result of the written files. This was the platform we were to use for our project (more in chapter 4).

## 1.3 Developing Customized Visualizations: State of the Art

In this section, we will look at three different software systems/libraries for developing visualizations. We will explain their use cases, benefits, and drawbacks. The first is Tableau, a quantitative analysis tool operable through a graphical user interface. The second is named ggplot2, a package for the R environment providing separate semantic components to construct visualizations with. Finally we take a look at D3, a JavaScript library enabling data binding to DOM elements and manipulation of their properties. These three systems/libraries support different approaches to data visualization, each with their own ideal use case. We will describe how they are used, what interactions they support, and how visualizations can be customized. Concluding this section, we will evaluate how well they were suitable for our project.

### 1.3.1 Tableau

The first software we look at is Tableau.<sup>7</sup> Tableau is a visualization software with an interactive user interface to input data and drag and connect different visualizations to it. Users do not need to know how to program to create visualizations, as Tableau offers a wide variety of standard visualizations plus annotating text fields that can be arranged into dashboards.

---

<sup>6</sup><https://lively-kernel.org> (last accessed: 2020-07-29).

<sup>7</sup><https://www.tableau.com> (last accessed: 2020-07-29).

### 1.3.1.1 Usage

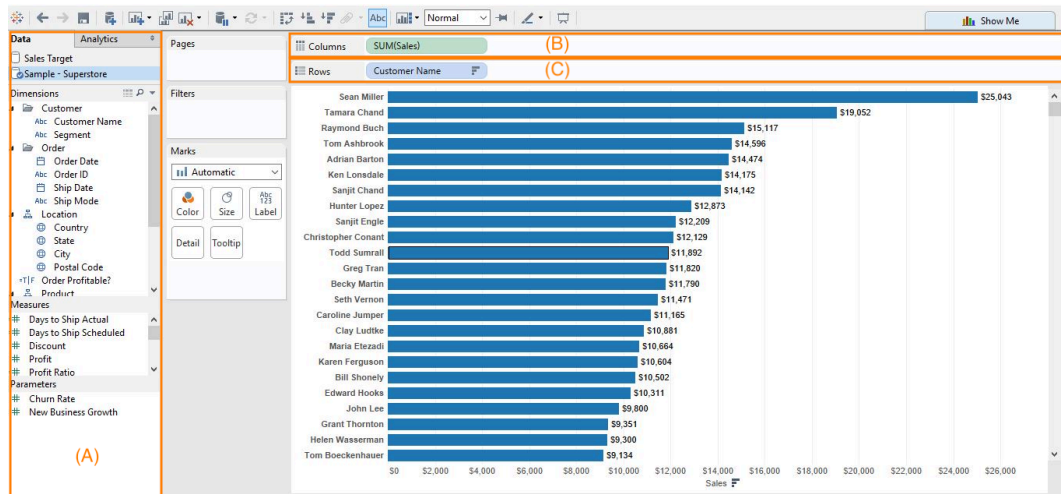


Figure 1.5: A Tableau Worksheet<sup>8</sup> with (A) the data-shelf, (B) the column-shelf and (C) the row-shelf

Users can import different data files such as Excel files, text, and JSON files. If these files contain different data tables, Tableau lets the user define relations and joins between these. Once users have selected their data, they can drag and drop categories like in Figure 1.5 from the data-shelf (A) into the rows-shelf (C) or columns-shelf (B) of a Tableau worksheet. Dragging a category to the rows-shelf then lets rows of the visualization correspond to values from this category. For example, in Figure 1.5 the category “Customer Names” was dragged to the rows-shelf (C) which results in the customer names to appear on the y-axis, whereas SUM(sales) was dragged to the columns-shelf (B) and are now displayed on the x-axis. Tableau then chooses a fitting visualization type for the kind of data to display. Users can change this visualization type, if it does not fit. They can add filter boxes for the data, where data points can be filtered out by category. Then, worksheets can be integrated into dashboards to display several views next to each other to enable comparison. Filter boxes can apply to multiple worksheets or for related data sources, giving users the ability to filter data in one chart and applying that filter to the whole dashboard. Users can also filter by clicking on representations of data points and can multiselect data by clicking and dragging a selection area.

### 1.3.1.2 Underlying Technology

Tableau, formerly Polaris [64], is mostly written in C++, but their core is the declarative language VizQL [36], that formalizes the description of tables, charts,

<sup>8</sup><https://www.analytics-tuts.com/wp-content/uploads/2015/12/1-3.jpg> (last accessed: 2020-07-29).

graphs, and maps and unifies them into one framework. Operations for retrieving, mapping, and rendering the data are generated by the VizQL query analyzer. Users only have to describe what they want to see.

### 1.3.1.3 Interaction

This underlying language makes it easy for Tableau users to switch from one visualization type to the next. Main interactions include resizing, tooltips on hovering over data representations, and filtering through filter boxes or by clicking on a visual representation in a chart. This, however, concludes interactions for Tableau worksheets and dashboards, which means, only these interactions are commonly supported.

### 1.3.1.4 Customizability

Non-developer users can plug their own visualization together, as long as it stays within Tableau’s offered range of customizations. This range includes coloring and size of data representations, changing the axis order, scale, and ticks as well as rewriting the formula for computed variables like sums and averages.

Developer users can write their own extensions to Tableau, examples can be seen in Tableau’s extension gallery.<sup>9</sup> These are JavaScript web applications that comply with Tableau’s API for dashboard objects. As they are written in JavaScript, developers can also use third-party libraries like D3.js to build their custom visualizations. Supported event listeners are `FilterChanged`, `MarkSelectionChanged`, `ParameterChanged` and `SettingsChanged`,<sup>10</sup> which means that custom interactions are still limited using these events. Extensions can be published via the extension gallery or to Tableau Server or Tableau Online. These public extensions can also be downloaded and used by non-developer users.

## 1.3.2 ggplot2

The second software we will look at is ggplot2.<sup>11</sup> This is a package enabling data visualization for scientific use in the programming language R. It breaks down charts into reusable, configurable, and combinable modules, such as scales and graphical layers. These modules can then be plugged together by writing declarative code.

**Listing 1.1:** R code resulting in a ggplot2 scatterplot

```
1 library(ggplot2)
2
3 ggplot(mpg, aes(displ, hwy, colour = class)) +
4   geom_point()
```

<sup>9</sup><https://extensiongallery.tableau.com/extensions?version=2019.4&per-page=50> (last accessed: 2020-07-29).

<sup>10</sup>[https://tableau.github.io/extensions-api/docs/trex\\_events.html](https://tableau.github.io/extensions-api/docs/trex_events.html) (last accessed: 2020-07-29).

<sup>11</sup><https://ggplot2.tidyverse.org> (last accessed: 2020-07-29).

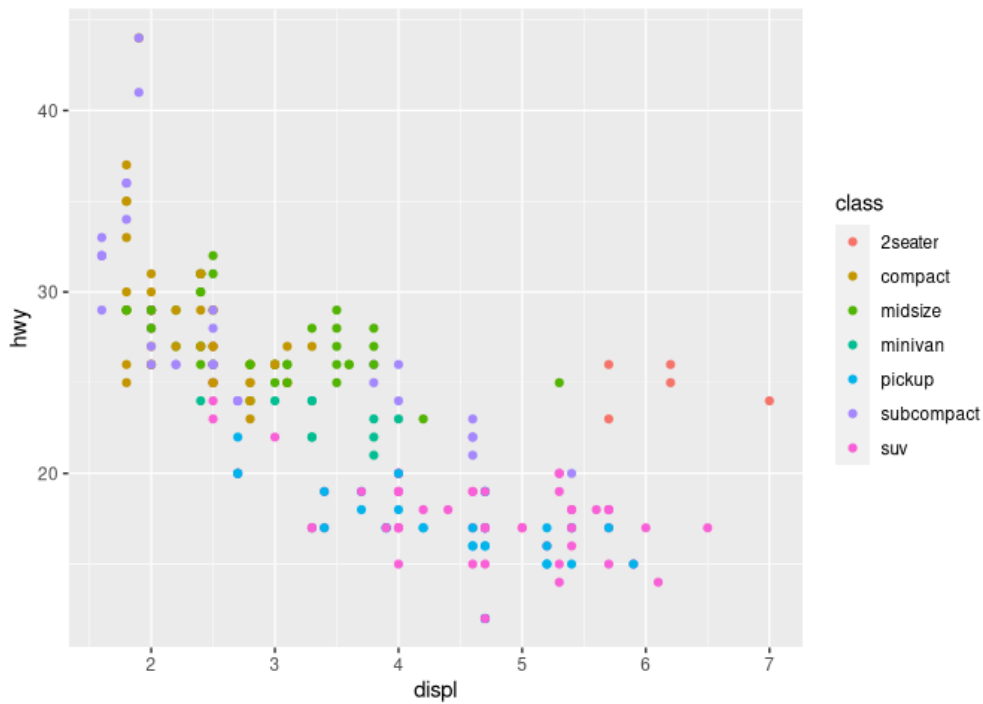


Figure 1.6: The resulting ggplot2 scatterplot

### 1.3.2.1 Usage

As it is a package for R, it can be imported in an R environment. Users then have to write code to produce a visualization as shown in Listing 1.1 and Figure 1.6. They can create a coordinate system by using the `ggplot()` function and passing a data set and aesthetics to it as can be seen in the listing Listing 1.1 taken from their website. The aesthetics take attributes for the x- and y-axis as well size, shape, or coloring variables. Then, users can add layers to visualize the data. A layer displays the visual representation of the data through mapping it to geometrical objects called “geoms”. A point geom will result in a scatterplot as seen in Figure 1.6, whereas a bar geom results in a bar chart. This way, users can change the type of their visualization very quickly by simply using a different geom. Also, they can add multiple layers to their visualization and combine different geoms. ggplot2 provides computed variables for specific geoms. Users can save their plots as PDF files or take advantage of R Markdown Notebooks, where they can create complete dashboards and send them to colleagues with the full source code.

### 1.3.2.2 Underlying Technology

The ggplot2 package implements the layered grammar of graphics [68]. It is written in R and open source on Github.<sup>12</sup> It creates an image of a plot by building, rendering, and drawing it as a PNG.

<sup>12</sup><https://github.com/tidyverse/ggplot2> (last accessed: 2020-07-29).

### 1.3.2.3 Interaction

As the resulting images of plots are PNGs, they are static, meaning there is no interaction possible. There are filter commands that can be used before the plot is created. But generally, the data exploration process of ggplot2 consists of writing code for a plot, running it, viewing the plot and then writing code for a new plot that may answer the next question.<sup>13</sup> The code can add some limited interaction, like hovering or zooming, to ggplot2 plots.

### 1.3.2.4 Customizability

Chart customization can be done within the offered range of aesthetics. Layering different geoms on top of each other enables building custom chart types. Same as Tableau, ggplot2 can include extensions,<sup>14</sup> like animations and more chart types written in R by other users that are hosted on GitHub.

## 1.3.3 D3

D3.js<sup>15</sup> is a JavaScript library for creating data-driven documents [10]. It works with HTML, SVG, and CSS to provide a data binding from data sources such as arrays, objects, JSON, or CSV files to DOM elements. Developers can select DOM elements and apply changes to them, add new elements, or remove unnecessary ones. Changes include one element's style like height, width, and color, as well as added behavior through event listeners.

### 1.3.3.1 Usage

For creating visualizations with D3, developers can import their data into a JavaScript file and map it to SVG shapes. Data attributes can be mapped to styles of these shapes as desired. To, for example, create a scatterplot, users can add d3-axes with d3-scales to an SVG to create the coordinate system and then draw the data as circles into that coordinate system using the scales for positioning. D3 also offers ready to use diagram types like histograms that only need a data input.

### 1.3.3.2 Underlying Technology

D3 creates and modifies DOM elements, that is for example, paragraphs for text and divs or SVGs for shapes. It wraps the W3C DOM API<sup>17</sup> for modifying HTML documents into declarative function calls. D3 itself is an open source project hosted on GitHub<sup>18</sup> and written in JavaScript.

<sup>13</sup><https://plotly.com> (last accessed: 2020-07-29).

<sup>14</sup><https://exts.ggplot2.tidyverse.org> (last accessed: 2020-07-29).

<sup>15</sup><https://d3js.org> (last accessed: 2020-07-29).

<sup>16</sup><https://www.visualcinnamon.com/2015/07/voronoi.html> (last accessed: 2020-07-29).

<sup>17</sup><https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html> (last accessed: 2020-07-29).

<sup>18</sup><https://github.com/d3/d3> (last accessed: 2020-07-29).

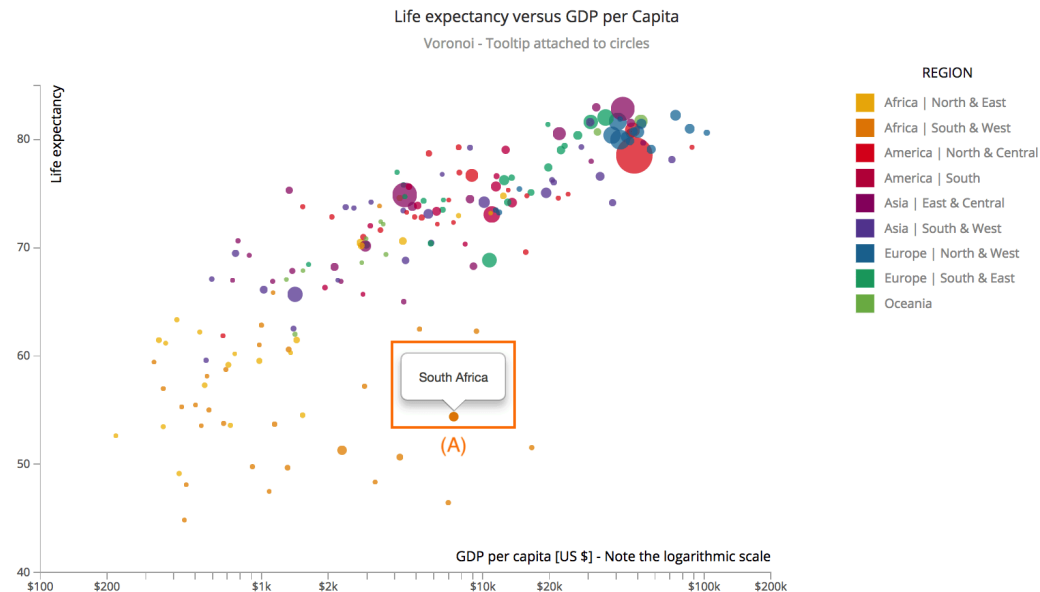


Figure 1.7: A D3 bubble chart with tooltip on hover (A)<sup>16</sup>

### 1.3.3.3 Interaction

For adding interaction, developers can add event listeners, for example, to the circles in a scatterplot so that they display detailed information on a hover event as in Figure 1.7. Or they can add behavior, so that the circles recolor or rearrange themselves on click. Here, all HTML DOM Event Listeners can be used and the event handler can be manually written.

### 1.3.3.4 Customizability

With all HTML DOM Event Listeners at hand and self-written event handling callbacks, D3 is very powerful when it comes to interactive charts. Developers can add any event listener to any HTML element and define the behavior that should occur on an event. This means, developers have full control over the interaction, which makes D3 charts highly customizable. But also the charts themselves can be created as desired, as D3 does not only offer complete implementations of charts. It provides developers with basic modules like axes and scales, animations, and interaction helpers to plug together to create their own chart.

### 1.3.4 Conclusion

As Tableau is currently used by Africa's Voices, we could have written some new visualization types as extensions to Tableau. But Tableau focuses on quantitative data, which can be aggregated. Therefore, it is a useful tool for doing quantitative analysis on our data, but not for visualizing the qualitative data we have. Disaggregating



the data in Tableau to display every data point as a dot might result in performance loss.<sup>19</sup>

ggplot2 is useful for quickly developing plots with only three lines of code for a standard scatterplot, as it only needs a data set, a mapping from aesthetics to variables, and a chosen geom to represent the data. It is, however, a package for the R environment, which might be a completely new programming language for some developers. Also, it does not deliver interactive charts on its own, making it necessary to work with add-on libraries such as Plotly.js. As plots with ggplot2 are simple to write, the library needs to abstract and provide high-level points of access. This way, building completely custom charts is hard, as developers can only build them from these abstract models.

D3.js with its direct connection to HTML and DOM via Javascript is a very good choice for creating custom visualizations with self-defined behavior. It is also a natural choice for developing visualizations in Lively4, because both are written in JavaScript and result in DOM elements on execution. However, the underlying technology uses SVG elements for drawing data representations, so developers will likely run into performance issues when visualizing large data sets. In chapter 7 we will go into more detail on the performance of these technologies.

All these three approaches to visualizing data operate on different levels. Tableau uses a graphical user interface to create visualizations. ggplot2 provides highly abstractive concepts that can be put together to create visualizations with a few lines of code. D3 gives the user full control over the outcome by providing low-level modules and manipulative access to DOM elements through JavaScript, but with a reasonable level of abstraction from the rendering and creating of images themselves. This means, all options are useful, but ideal for different uses. In our case, having as much control over visualization appearance and behavior proved valuable.

## 1.4 Our Approach

This section summarizes our approach to the project and the domain problems. A summary of our results will follow in section 1.5.

**Basis** The starting point of our project consisted of three basic things: 1) data from Africa's Voices in large JSON files, 2) the development environment Lively4, and 3) a non-specific description of the end results to be developed.

**Inspiration** With no knowledge of state-of-the-art technologies for visualizing data, we looked into visualization libraries that could be used. We also looked for inspiration on the web, in books, in reports, and in research papers.

---

<sup>19</sup>[https://help.tableau.com/current/pro/desktop/en-us/calculations\\_aggregation.htm](https://help.tableau.com/current/pro/desktop/en-us/calculations_aggregation.htm) (last accessed: 2020-07-29).

**Brainstorming** We spent time brainstorming new ideas based on found inspiration and the received data. For that, we used techniques taken from the Design Thinking approach (subsection 2.2.1) to understand our problem better and to evaluate what would really help different users. In chapter 2 we will describe and categorize our developed ideas.

**Implementation** After developing a large set of different visualizations, we started implementing those that seemed most interesting and useful. We first tried out the main concept in a script in Lively4, then added more features to that attempt to develop it into an elaborate prototype. To unify the behavior and to connect different visualizations, we had to design and implement underlying architecture as well as an API, that could be used commonly to control and interact with all visualizations. In chapter 3 we will explain our developed prototypes and their usage, as well as interactions implemented using this API. In chapter 4 we will elaborate on our usage of the Lively4 environment during the project.

To use the data provided by Africa's Voices, we stored it on an extra server, which can be queried by our visualizations. The data format kept changing over the course of our project, so we implemented a parser that would always return the data consistently in our desired format. In chapter 5 we will describe the formats of data we received and the format we developed in more detail, as well as how our visualizations handled data mapping to visual representations and vice versa.

Because we had to display a large set of points unaggregated, we researched technologies that could handle the amount of data well and that left room for interactions and animations. In chapter 6 we are going to describe commonly used web technologies and compare and evaluate them.

**Testing** To test our ideas, we presented them to the Technical Lead at Africa's Voices, who was also our contact for the project. We presented him developed ideas and implemented prototypes. As a result, he could confirm that already discovered findings could be demonstrated again using our visualizations, as well as finding new points of interest through the interactivity and explorability was possible. By letting him try out our prototypes, we could also evaluate whether they were intuitively usable and comprehensible without further explanation. In chapter 7 we are going to evaluate our end result further.

## 1.5 Our Contribution

This section summarizes our contribution to the improvement of the work of Africa's Voices with this project. We will shortly describe the concept of our newly developed visualizations and of the two implemented systems that connect them: the *Tab View* and the *Tree View* system. We will then look at how these new possibilities could affect Africa's Voices' processes and workflows.

### 1.5.1 New Visualizations

We provided Africa’s Voices with many ideas and prototypes for new visualizations and interaction patterns. The idea behind our visualizations is to keep all individuals visible by representing each individual with a single dot. We have realized that this model of representing individuals with dots comprehensibly works. Also, our design strategy to unify visualizations by using the same graphical elements is helpful, too. This way different visualizations are easier to understand and to compare. Each dot can be interacted with by clicking on it to get detailed information, which creates trust. Added to that, we provided visualization-specific controls for arranging and coloring the dots.

Visualizations can be instantiated multiple times to show the same data from different perspectives. Our visualizations can display all dimensions of a data point by inspecting it. Most importantly, they could show the original message of the individual, if it was included in the data, which then could increase empathy. However, they can take two dimensions to calculate the data point’s position and a third for it’s color. Users can decide, how they want to map dimensions to visual attributes to get the view of the data they desire (see section 3.3 for more).

### 1.5.2 Connections Between Visualizations

Having comparable visualizations with standard interactions is important. To this end, we realized two systems to connect base visualizations that focus on different usage scenarios while maintaining consistent appearance and interactions. The *Tab View* system (section 3.5.2) consists of tabs showing the same data but with different visualization types. It keeps a global legend and actions are performed on all visualizations. The *Tree View* system (section 3.5.3) enables users to create an exploration tree in which actions performed at a higher level of the tree are passed down to its children.

### 1.5.3 Changed Workflows

The prototypes implemented allow for several changes and improvements in the workflow of different types of users.

#### 1.5.3.1 Developers

Developers can change everything about our developed prototypes as Lively4 keeps code and its interpreted result close to each other. Navigation between both views – code and results – is possible with the push of a button. This allows fast and direct changes to the code and the visualizations. They can also write their own base visualizations.

For our two visualization systems, they can write and integrate new base visualizations easily by implementing the required API. They can also develop their own visualization system and use our visualizations as a basis.

### 1.5.3.2 Researchers

Researchers now have more available visualizations for exploring Africa's Voices' data. They are provided with new interactions and many exploration possibilities, which could lead them to new questions and insights.

As our systems make the connections between data in multiple views clearer, relationships in the data might be easier to spot. Our *Tree View* system could help them to structure their exploration process as it makes the exploration history and decisions visible.

### 1.5.3.3 Presenters

Presenters benefit mostly from our two visualization systems, as they can use them to narrate their findings better to an audience of people new to this kind of qualitative data visualization. The *Tab View* system could help them to show multiple views of the same data and the *Tree View* system for arranging different visualizations to explain trends and insights.

## 1.5.4 Summary

We have seen now that in the context of Africa's Voices, there exist domain-specific challenges of asking sensitive questions, displaying people's opinions and getting across the right impulses. Standard charts proved unhelpful to achieving these goals. However, we were successful in designing new visualizations that keep the individuals and their opinions visible. We helped solve some of the main problems of the visualization process of Africa's Voices through developing new visualizations with new interactions and connecting them in two different systems. For that, we explored the design space for visualizations that can show data from this domain. We realized that this space is quite rich and has a lot of potential for further development. Therefore, the following chapter 2 will describe this space and categorize our ideas for visualizations and interactions.

## 2 Concepts for Visualizations that Make Data Explorable and the Exploration and Categorization of the Design Space

Africa's Voices receives thousands of text messages of individuals that all have to be visualized and analyzed. Their main aims are to allow for an exploration of the data and not to lose the voices of the individuals. Standard charts do not allow users to look past the aggregated data and are therefore not sufficient for an interactive exploration of the data. As they can only display statistical information, the opinions and needs of the individuals are lost. We designed around 80 ideas that allow for an in-depth exploration of the data and categorized them by a technical dimension and task dimension. We further extracted several concepts that enable users to explore the data, develop empathy with the displayed individuals, and that create trust in the representation form. The main concepts are to display the individuals as points and to offer different perspectives on the data and comparisons across visualizations. These concepts provide a fundamental basis for further research and the design and creation of explorable, individual-centered visualizations.

### 2.1 Introduction

Africa's Voices receives thousands of text messages every time they conduct a radio show. The attribute values of these individuals and their messages are then visualized and analyzed using standard charts, such as bar charts or pie charts. As this kind of data is highly complex, Africa's Voices aims at creating visualizations that improve the finding of insights in this data.

These visualizations have three main requirements. They have to allow users to explore the data in more detail and need to create empathy for the visualized individuals. The third requirement is the creation of trust in the truthfulness of the visualization and the data. First, they have to allow users to explore the data in more detail. Second, they need to create empathy for the visualized individuals. Third, they have to create trust in the truthfulness of the visualization and the data.

However, standard charts do not allow for a deep exploration. As they are static, the opinions and needs of the individuals are lost. Further, users need to rebuild the visualization every time they want to adapt the data or the representation form. This process does not allow users to answer questions about the data as they emerge, which is essential for finding important insights about the data. Standard charts do not allow users to look past the aggregated data and are therefore not sufficient

for an interactive exploration of the data. Africa's Voices works with thousands of data points. Therefore, just displaying the attribute values of individuals and corresponding messages in an unstructured way is not an option.

To solve these problems, we designed novel visualizations that focus on the individual instead of the aggregated data and allow for a deep exploration of the data, while creating empathy and trust. We extracted concepts from our visualization ideas that implement these requirements. These concepts provide a fundamental basis for further research and the creation of explorable, individual-centered visualizations.

We structured our design process using the Design Thinking approach, which aims at a structured, target-oriented process to find solutions. We present an overview of the Design Thinking method, a description of our approach, and a short evaluation in section 2.2. To decide which ideas and general concepts allow best for the exploration of data and the creation of empathy and trust, we categorized the ideas regarding two dimensions. We describe these dimensions in section 2.3 in detail, give an overview of the categorization, and present the findings we extracted. The developed concepts are then discussed in section 2.4. Using these concepts and findings, we decided on the ideas that seemed most interesting regarding the design of explorable individual-centered visualizations. A conclusion is presented in section 2.5.

## **2.2 Exploration of the Design Space and Ideation**

Our goal was to develop new and unforeseen visualizations. As applying existing ideas to the problem domain is, in this case, not enough, creating something new is a difficult task. To begin searching for ideas, first, the problem has to be specified. The overall goal of our project was quite clear. We were to implement a platform and construct examples of interactive, explorable visualizations that would help Africa's Voices to find new insights in the data. The problems that led to this goal, however, were poorly defined.

To understand the problems Africa's Voices wanted to solve, we did a lot of research associated with visualizations and Africa's Voices' current process. It became clear that there could be many correct solutions to the problems described in section 1.1.3. Overwhelmed by a load of information, we had difficulties applying the broad knowledge base to the problems and selecting the most important insights to ideate on.

To achieve the best solutions possible in a limited amount of time, we needed a target-oriented approach. Such a target-oriented approach is Design Thinking.

This section clarifies our approach by giving an overview of the Design Thinking method, a structured description of the steps we used, and a short evaluation.

### **2.2.1 Overview Design Thinking**

Design Thinking aims at finding human-centered solutions and therefore focuses on the users from the beginning. An important aspect of Design Thinking is to contact users early on to consider their needs to develop a solution that has the best

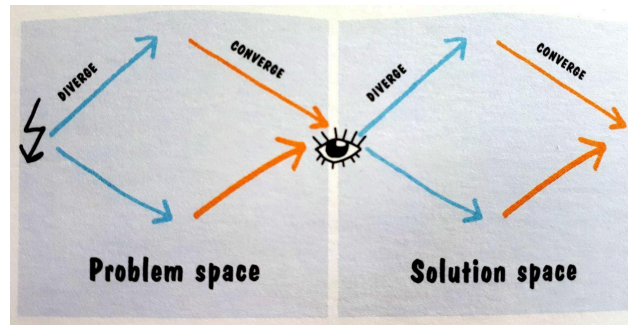


Figure 2.1: The diverging and converging process in Design Thinking [45]

possible impact. It also prevents from having to base the product's development solely on assumptions about what the user needs or wants [45]. Human-centered design is defined by an ISO standard<sup>1</sup> as an approach that focuses on the usability of the solution. It considers human factors and ergonomics when designing a system. Goals are, for example, increasing the productivity and quality of the users' work and improving the users' satisfaction. Design Thinking adopts the main principles, centering their solution finding around the user. But whereas Human-centered Design aims mainly at the good usability of the system, Design Thinking aims to develop innovative solutions for complex problems [26].

The challenge Design Thinking meets, is to cover a broad problem and solution space while at the same time developing practical solutions [45]. Design Thinking is therefore based on two mental states: *diverging* and *converging*. When *diverging*, the goal is to create ideas and generate insights, whereas *converging* stands for focusing on or limiting the ideas to individual needs or potential solutions. During the Design Thinking process, we diverge and converge at least two times, as shown in Figure 2.1. The first time happens in the problem space, the goal of which is to understand the problem more clearly, and the second time in the solution space, in which we try to find a solution for the problem [45]. This development process can be divided into several steps. For these steps, there are many different structuring possibilities. We will focus on the microcycle structure taught by the D-School in Potsdam<sup>2</sup> to explain the key concepts of the *diverging* and *converging* phase in both the problem space and the solution space.

The microcycle consists of six steps as shown in Figure 2.2: *Understanding*, *Observing*, *Defining the Point of View*, *Ideating*, *Prototyping*, and *Testing* [45]. *Understanding* and *Observing* are part of the *diverging* phase in the problem space. During these steps, we try to get a better understanding of the problem and the users. To know what to focus on when to converge again, we define the *point of view*. *Ideating* represents the *diverging* phase in the solution space. The goal is to

<sup>1</sup>ISO 13407:1999(en) Human-centered design processes for interactive systems <https://www.iso.org/obp/ui/#iso:std:iso:13407:ed-1:v1:en> (last accessed 2020-07-29).

<sup>2</sup><https://hpi.de/en/school-of-design-thinking/hpi-d-school.html> (last accessed 2020-07-29).

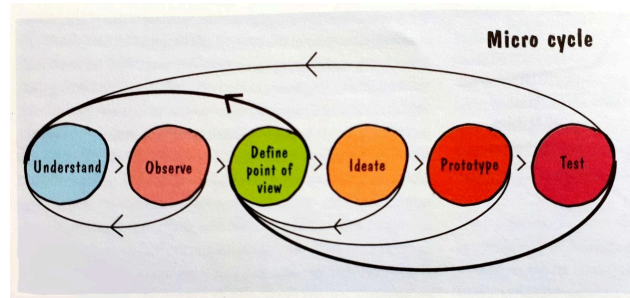


Figure 2.2: The microcycle in Design Thinking [45]

create as many ideas as possible. During *Prototyping* and *Testing*, we converge again by deciding on the best ideas and realizing them as prototypes. It is an iterative process, which makes it possible to revisit steps in one iteration when needed [45]. In most Design Thinking projects, the project consists of more than one iteration of the microcycle. Each of these iterations then has a specific goal, such as identifying critical functionalities or creating ideas from benchmarking [45]. We now give a short overview of each step in the microcycle.

#### 2.2.1.1 Understanding

The main focus of this phase is to understand the problem and its context. To understand the relevant aspects, it is often helpful to answer the six “WH questions”: who, why, what, when, where, and how. For example, “Who is the target group?” and “Why does the user think he needs a solution?” [45].

Once the problem is identified, a problem statement can be defined. As good ideas originate from good questions, Design Thinking assumes that a good problem statement is fundamental for a successful project. Its purpose is to encourage ideation in many different directions while focusing on the problem. The statement is formulated in a neutral form to prevent restriction of the solution space in an early phase. It can be reformulated based on newly gained knowledge [26]. The gained knowledge of the problem can be used to create the first persona. A persona is a fictional person that represents a group of people that share the same interests. The design team can then engage with this persona and base further brainstorming on them [63].

#### 2.2.1.2 Observing

This phase is sometimes also described as the “empathizing” phase [22]. The goal is to understand the users by understanding how and why they do what they do and understanding their needs, opinions, and beliefs. To empathize with the users, we can observe them and their behavior in their work or home environment, depending on the problem we are trying to solve. Another possibility is to engage with them by having conversations and interviews, or placing oneself in the situation of the user [22, 26, p. 27]. All findings are then documented. The gained knowledge can be used to create or revise personas [45].



### 2.2.1.3 Defining the Point of View

Based on the findings and insights from the previous phases, this phase aims to establish a common understanding and knowledge basis [45]. This step is essential to not only address a single group of people that have been observed in the previous phase but to design a solution for a wide range of users [45]. First, the information collected in the previous phases needs to be analyzed, including collecting and interpreting all information and summarizing it by collecting insights from key findings. Now the most important insights are collected, and design principles are extracted from them. The next step is to formulate key statements of a limited number of topics to focus on during the further process [45].

These statements are also described as the *point of view* and are explicit statements of the problem. The *point of view* focuses on the interpreted insights and the users' needs. It is based on the understanding of the users and the problem space. In contrast to the problem statement defined in the *Understanding* phase, the *point of view* should be more specific. Such statements tend to increase the quantity and quality of solutions in the following ideation [45, 22]

### 2.2.1.4 Ideating

After converging by defining a *point of view* in the previous phase, the goal of the ideation phase is to diverge again by creating as many ideas as possible. This conversion is mostly done by brainstorming and sketching, but Design Thinking suggests many techniques that can increase creativity. The ideation phase represents the transition from understanding the problem to finding solutions [45, 22].

### 2.2.1.5 Prototyping

A selected amount of ideas is prototyped. The prototypes should be very quick and cheap to build, especially in the early stages of the project. When the problem has become more specific, they can also be more refined. A prototype should be something with which the users can interact, such as a role-playing game or a storyboard. That way, the users can give feedback in the *Testing* phase, and the ideas and prototypes can be improved [22, 45].

### 2.2.1.6 Testing

In the *Testing* phase, the users test the prototype. The feedback is then used to iterate on the idea. If the prototype is discarded, it may be a sign that the problem was not understood correctly. The *Testing* phase, therefore, also provides an opportunity to refine the *point of view*. Testing also helps to learn more about the users and builds empathy through the interaction between the users and the design teams [22, 45].

## 2.2.2 Our Approach

We did not employ a full Design Thinking process, as we could not talk to our potential users. We used several methods of the Design Thinking approach that helped us to stimulate creativity and to understand the most important aspects of the problem we tried to solve. This section describes our approach based on the

six steps described in section 2.2.1 and explains the methods we used. We further discuss what insights we gained in each step.

#### **2.2.2.1 Understanding**

At the beginning of our project, we explored the problem space, which included understanding the domain and the goal of the stakeholders, and researching current trends and similar topics.

**The Domain and the Stakeholders** The problem domain and the stakeholders' goal are described in detail in chapter 1 and section 1.2.2. Here we only give a summary of the requirements presented in section 1.2.2 regarding the design of the visualizations.

For one, the visualizations are supposed to help researchers to explore the data. That means, the visualizations have to allow for a more in-depth exploration to make discovering insights easier. Another requirement is that the visualizations need to create empathy for the individuals. Therefore, they have to keep the individual in the center. It is also important that users trust the accuracy of the representation form and the data.

**Research** We focused our research on visualization techniques and visualization design patterns to understand what aspects are important to consider when designing visualizations. We searched for inspiration on visualization ideas on the web, literature, and reports of Africa's Voices.

Other research topics were the development tools and libraries that can create different types of visualizations. We divided this research into different aspects. One aspect was to research provenance by experimenting with frameworks such as D3 and RaphaelJS. We created data tables and visualized the data with the chosen frameworks. The goal was to let users interact with the tables and visualization to explore the data's provenance, for example, by highlighting the corresponding data. The capabilities of different visualization libraries were another important aspect. For a description of a choice of libraries, see section 1.3. Later during the project, a clear focus on the visualization of individuals as points emerged. See chapter 6 for a description regarding the rendering of large amounts of points using different web technologies.

#### **2.2.2.2 Observing**

As part of the process, we created a persona to achieve a better understanding of the users' needs and problems. Our potential users are a combination of the qualitative and quantitative researchers at Africa's Voices. For a detailed description of our users, see section 1.2.2.5. As this kind of user does not yet exist in Africa's Voices, we did not have a current user to design for and tried to design for future users. We interviewed our project partner and structured the information we gathered as a persona named Tom, shown in Figure 2.3.

Tom's work has a focus on the qualitative aspect, including labeling and translating the messages. Therefore Tom has to be able to speak multiple languages such as

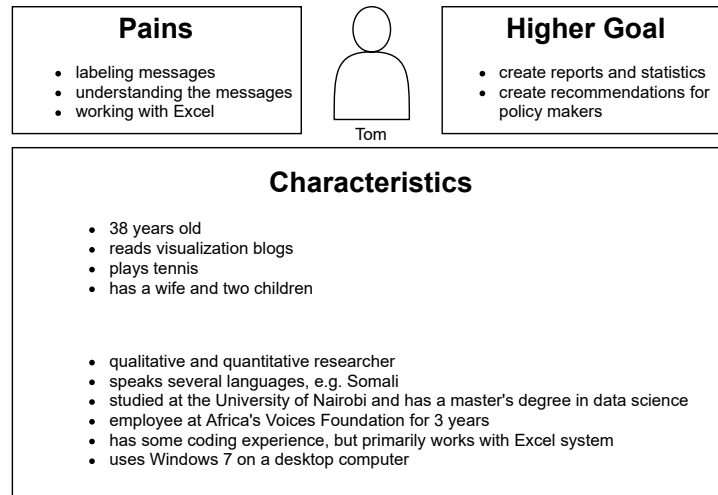


Figure 2.3: The persona named Tom

Somali. The quantitative work includes programming. As Tom is supposed to unite both aspects, the persona also needs to be able to program at least to some extent.

The created persona gave us a better understanding of what our potential users looked like, including their needs and expertise. It is essential to keep these insights in mind when ideating so that the created solutions fit their needs and capabilities.

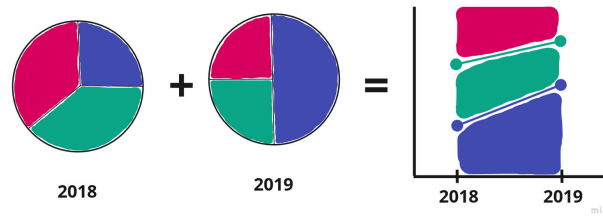
### 2.2.2.3 Defining the Point of View

We analyzed the information about Tom to formulate the *point of view*. This analysis process included specifying the persona's true needs, which helped to understand the most important aspects of the problem. Each *point of view* represents one true need. To formulate the point of view, we used the format "For [the user], it would be life-changing if [a certain goal was achievable] because [of a certain problem]". This formulation is based on examples in *The Design Thinking Playbook* [45].

The following points of view define the needs we analyzed using the insights of our persona Tom:

- For Tom, it would be life-changing if he could see insights at one glance and if gaining insights would be as intuitive as using a whiteboard because gaining insights from Excel is difficult.
- For Tom, it would be life-changing if he could understand the participants' real problems and causes because formulating recommendations without understanding the real issue does not make sense.

**"How might we"-questions** are another possibility to formulate the point of view. They have the form "How might we help [the user, customer] to achieve [a certain goal]?" [45]. They can further be used as a basis for the ideation and apply the gained knowledge to the problem. This basis enables a structured target-oriented finding



**Figure 2.4:** Fusing two pie charts into a stacked line chart (41)

of solutions and a good starting point for the next diverging phase. Therefore they are formulated based on the predetermined *point of view* [45]. Based on our point of view statements, we formulated the following “How might we”-questions:

- How might we help Tom to see insights at one glance?
- How might we help Tom make the process that led to his recommendations comprehensible for other people?
- How might we help Tom understand the messages?
- How might we help Tom to make good recommendations?
- How might we help Tom decide which recommendations are good?
- How might we point Tom to the important correlations in the data?
- How might we make gaining insights intuitive?
- How might we help Tom to back up recommendations?
- How might we help Tom compare similar diagrams easily?
- How might we make gaining insights out of Excel easier?
- How might we help Tom use the full potential of Excel?
- How might “Iron Man” find insights in the data?

#### 2.2.2.4 Ideation

Using different brainstorming techniques to stimulate creativity (see *The Design Thinking Playbook* [45] for examples), we developed ideas based on one “How might we”-question at a time. During our brainstorming sessions, we mostly sketched ideas on whiteboards and post-its. We visualized these ideas directly during the ideation process and thereby created the first small prototypes.

We concentrated on the topics our project partner seemed most interested in, which included making the recommendation process comprehensible, comparing similar diagrams, highlighting correlations in the data, and discovering insights at one glance.

As encouraged by Design Thinking, we went for quantity, creating around 80 ideas. We give an overview of the ideas in section 2.3 by presenting a categorization and giving examples. All ideas are shown in Appendix section A.1 with their enumeration.

For example, brainstorming based on the HMW-question “How might we help Tom to see insights at one glance?”, we designed several visualization ideas. Examples are *XY-Axis* (22) and *Venn* (39), which are described in detail in section 2.3.3.2 section 2.3.3.4. A first example is shown in Figure 2.4.

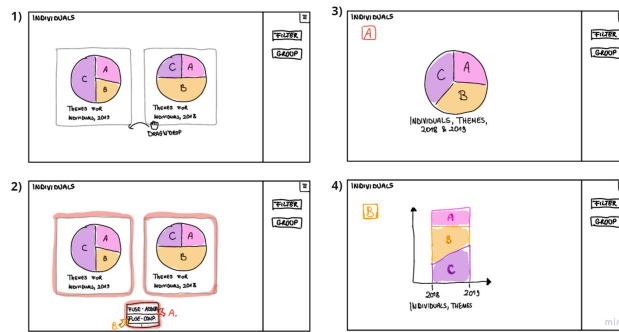


Figure 2.5: Extract of the storyboard for the idea of fusing diagrams (41)

The idea merges two pie charts into a line chart, allowing users to see the differences between the two years at one glance.

### 2.2.2.5 Prototyping

Based on what our project partner was most interested in, we selected ideas that seemed exciting or new. Usually, Design Thinking aims at selecting ideas that best fit the users' needs. As we worked with hypothetical users, and our project partner required us to implement novel solutions, selecting exciting or new visualization ideas was a valid choice.

As part of this converging step, we refined and documented the ideas using storyboards. A storyboard is a series of drawings that present the specific steps of an event. A common way to express this is to tell the story using a comic-strip format. A storyboard aims to make the key aspects of the prototyped idea comprehensible at first glance [63]. We used the concept to formulate our ideas in a more detailed way before implementing them, thereby specifying user interaction. That way, we could use the storyboard as a template for the implementation. Figure 2.5 displays an excerpt of the storyboard we created for the idea shown in Figure 2.5.

After the approval of our project partner, we spiked the ideas to test how to implement the basic features as the next level of the prototype. For a description regarding the implementation of the prototypes, see chapter 3.

### 2.2.2.6 Testing

Our project was based on work with hypothetical users. As our potential users do not exist, we tested our prototypes with a representative of Africa's Voices as a stand-in for the actual users. The feedback sessions were conducted during regular meetings. We then improved or discarded the sketches and storyboards based on the feedback we received.

## 2.2.3 Evaluation

For our project, the application of Design Thinking was limited, as we worked with hypothetical users and only accessed information about these users through a third

party. However, Design Thinking helped us apply the knowledge we gained in the *Understanding* phase of our project and increased our creativity and imagination with its creative methods for need-finding and ideation. The Design Thinking method helped us to condense our ideas and to concentrate on one important aspect at a time. As we had direct access to Design Thinking expertise, the Design Thinking approach was the reasonable choice for our project. The approach proved to be a helpful tool for the initial understanding of the problem and the finding of novel solutions.

## 2.3 Categorization and Analysis

When creating a large number of ideas, it becomes difficult to gain an overview of their distribution. Getting such an overview is essential to identify directions that have not been considered yet. It also poses an opportunity to determine ideas that seem promising. Our goal was to create visualizations that allow the exploration of data and create empathy and trust. Therefore, it was important for our project to identify ideas that seemed most promising to achieve these goals.

A helpful tool for getting an overview of created ideas is a taxonomy. A taxonomy is “a system for naming and organizing things [...] into groups that share similar qualities”.<sup>3</sup> There are many existing taxonomies for the categorization of visualizations. We chose three taxonomies that focus on different aspects of a visualization, the representation mode, the interaction level, and the tasks.

This section explains the considered dimensions and gives an overview of the categorization. It further presents the findings identified through the categorization that help to answer the stated questions.

### 2.3.1 Technical Taxonomy

We focused the categorization on two dimensions, the technical perspective and the task perspective. The technical perspective is especially interesting for developers to identify common or similar components. Understanding the interaction level may also pose an understanding of the complexity of the idea. The task perspective is interesting for designers and users. It describes the main purpose of the visualization idea and can help the designers estimate whether the visualization realizes the users' goals. We now describe the used taxonomies in more detail.

The technical categorization is inspired by the taxonomy proposed by Chengzhi, Chenghu, and Tao [57]. They create a matrix using categories of the representation mode developed by Keim and Kriegel [43] and categories of the interaction level developed by Lunzer [48]. We use the original categories for both dimensions as

---

<sup>3</sup><https://dictionary.cambridge.org/de/worterbuch/englisch/taxonomy>  
(last accessed 2020-07-29).

they are more detailed than the taxonomy proposed by Qin Chengzhi et al. The representation mode is subdivided into six categories [42, 43]:

- *Pixel-oriented Techniques*. One colored pixel represents one data value, and data values of different attributes are displayed in separate components.
- *Geometric Projection Techniques*. Visualizations that use geometric transformations and projections of multidimensional data sets. Examples are scatterplots and parallel coordinates.
- *Icon-based Techniques*. Data values are visualized as icons.
- *Hierarchy-based Techniques*. Highly dimensional spaces are subdivided and then displayed using a hierarchical partitioning. Examples are treemaps and Venn diagrams.
- *Graph-based Techniques*. Layout algorithms and abstraction techniques visualize large graphs.
- *Hybrid Techniques*. Hybrid visualizations combine multiple techniques to enhance the expressiveness of the visualizations. In our categorization, we will not mark visualizations as a hybrid of other techniques but will sort them into all corresponding categories. Visualization ideas that have a system-like character and consist of multiple visualization types are sorted into this category.

We designed some visualization ideas that do not fit into these categories. A part of these ideas is based on media techniques such as text or audio. The other ideas are either standalone interactions that do not have a specific representation mode or ideas that do not fit into any category. We describe these ideas as *media-based techniques*, *techniques with no specific representation mode*, and *other concepts*. The interaction level (Lunzer) is divided into the following five categories [48, 66].

- *Fully Manual*. The visualization provides no assistance in guiding exploration, such as when the user has to drag and drop an object manually. Visualizations that have no interaction also fall into this category.
- *Mechanized*. The visualization specifies possible directions of interaction. Examples are making a selection with a slider or coloring the data according to attribute values.
- *Instructable*. Users specify what the system is supposed to do, for example, by using formulas in a spreadsheet.
- *Steerable*. Users can steer the system to perform in a certain way. An example is directing an algorithm to perform in a certain way by specifying the parameters.
- *Automatic*. The system designs a solution without further human interaction that achieves goals that the user did not specify, for example, an algorithm that performs automatically.

Visualizations can have multiple interaction levels. Such ideas are sorted in all corresponding categories.

### 2.3.2 Task Taxonomy

We further categorize the visualization ideas using a task taxonomy. We use the taxonomy proposed by Shneiderman [60], who describes seven tasks for information visualizations. The seven tasks, as defined by Shneiderman, are:

- *Overview*. Gain an overview of the entire collection.
- *Zoom*. Zoom in on items of interest. We interpreted this as a task that focuses on a subset of data points while keeping the remaining data points visible.
- *Filter*. Filter out uninteresting data items.
- *Details-on-demand*. Select an item or group and get details when needed.
- *Relate*. View relationships among items.
- *History*. Keep a history of actions to support undo, replay, and progressive refinement.
- *Extract*. Allow extraction of sub-collections and of the query parameters. It includes saving or exporting a selection to be able to reuse it in some way.

Visualizations may have more than one main purpose. We sort such ideas into all corresponding categories.

### 2.3.3 Technical Dimension

During the ideation phase of our project, we designed about 80 ideas. As we concentrated on going for quantity, some ideas are not as useful or as thought through as others. Some ideas are quite similar and seem to extend other ideas. This similarity is based on used brainstorming techniques, where the aim is, for example, to extend the functionalities of one idea by adding or changing aspects. In the following, we regard these ideas as their own and not as an extension to another. We also applied known visualization concepts on our data, for example, line charts or bar charts. As such ideas still may pose a new perspective on the data, we still consider them in the categorization.

For the categorization, we ignore that interactions such as filtering and highlighting could be implemented for all visualizations and that some manual interactions could easily be exchanged with mechanized ones. This categorization will only focus on the main purposes the visualization idea tries to convey.

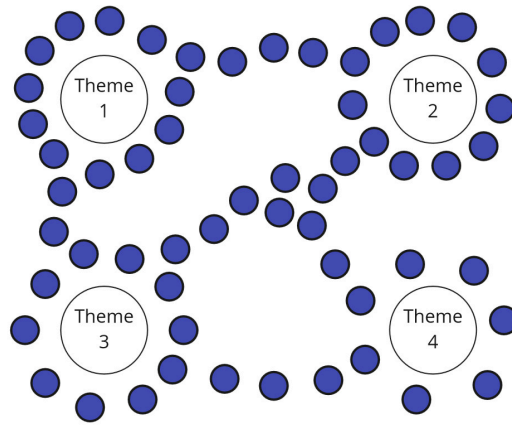
We now describe the different categories and give examples of visualization ideas that we later implemented or that stick out, for example, because they are the only ones in their corresponding categories.

For the following description, we use the structure of the categorization matrix, as displayed in Appendix section A.2, describing each existing combination of categories.

#### 2.3.3.1 Pixel-oriented

**Mechanized** There are only two ideas that have a *mechanized* interaction level and a *pixel-oriented* representation mode. One of the ideas (9) is a hybrid with a graph-structure that gives an overview of the themes in one district. The other one (21) colors the pixels, representing an individual, according to an attribute, and then





**Figure 2.6:** Theme Centers (33). Placing individuals as points around and between their themes

groups the individuals according to another attribute. That way, researchers can find insights about how the pattern changes.

### 2.3.3.2 Geometric Projection

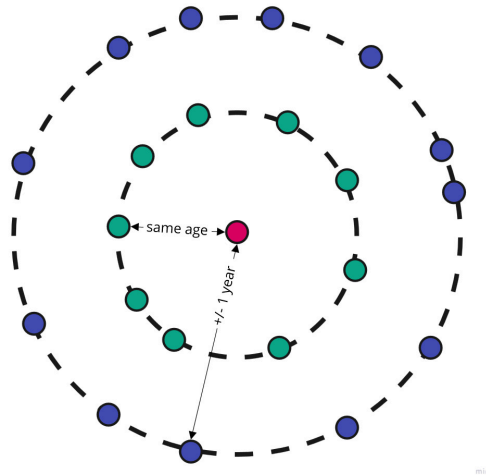
**Manual** With 25 visualization ideas, this combination of categories has the most ideas in the technical categorization. This set of ideas can be divided into two further categories, visualization ideas with and without interaction.

*Visualization Ideas Without Interaction.* There are fifteen static visualization ideas. They include a subset of five ideas that deal with the display of missing data (64, 65, 66, 67, 68) and a subset of seven ideas that use known concepts such as parallel lines, maps, word clouds, and bar charts. (1, 2, 20, 42, 50, 73, 74). Only two visualization ideas cannot be sorted into these subsets. One of them uses the idea of a magnetic field mapping one dimension to the two magnetic poles and another dimension to the vertical axis (30). The other idea displays the connection between the individuals and their corresponding themes. We now describe this idea in detail.

#### Theme Centers (33)

Individuals are placed around a theme if they only have one theme and between themes, if their messages have been coded with multiple themes. For example, the individuals displayed in the center in Figure 2.6 between themes two, three, and four sent messages that have been coded with all three themes. The goal of the visualization idea is to offer an overview of all the individuals regarding their themes and enable users to relate the accumulations of individuals.

As soon as there are more themes or more combinations of themes, this visualization becomes more complex. A variation of the idea, which we implemented in our *Movement* prototype, makes the individuals move between their corresponding themes. See section 3.4.5 for a description regarding the implementation.



**Figure 2.7:** Individual Centered (7). Placing individuals as points around one individual. Their distance is based on the difference of their attribute values regarding a certain attribute

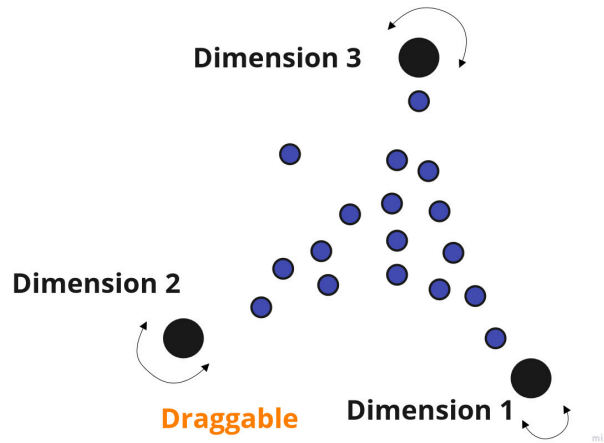
*Visualization Ideas With Interaction.* About half of these visualization ideas deal with the interaction of code and visualization. The manual interactions are editing the code, seeing the changes to the visualization immediately (54, 57, 58, 62), and hovering over code lines to highlight corresponding parts of the visualization (56). Another idea (48) is one of two visualization ideas that display Africa's Voices' process steps. The second one (49) is sorted into the categories *geometric projection* and *mechanized interaction*.

We now describe two ideas that use some kind of interaction in more detail. Both visualization ideas were inspired by the idea to use the positions of the individuals displayed as a point to convey information instead of losing one dimension by displaying them randomly.

### **Individual Centered (7)**

The visualization idea shown in Figure 2.7 is based on the idea of keeping individuals in the center of the visualization. The core concept is to keep one individual as a reference point in the center. The other individuals are arranged around this individual according to their "distance" to the referenced individual regarding one attribute, such as age or gender. By clicking on another individual, this individual becomes the new reference point. For example, if the individual in the center is 20 years old, the first ring around the individual would include all individuals that are also 20 years old. The second ring would include all that are one year older or younger, and the third ring would include all that are two years older or younger, and so on.

The idea allows users to understand the distribution of individuals across different attributes and to view similarities and differences between individuals.



**Figure 2.8:** Forces (31). Attracting individuals as points to their corresponding values

The concept of this visualization idea changed slightly during the implementation, but the core concept stayed the same. For a description regarding the implementation of our *Individual Center* prototype, see section 3.4.4.

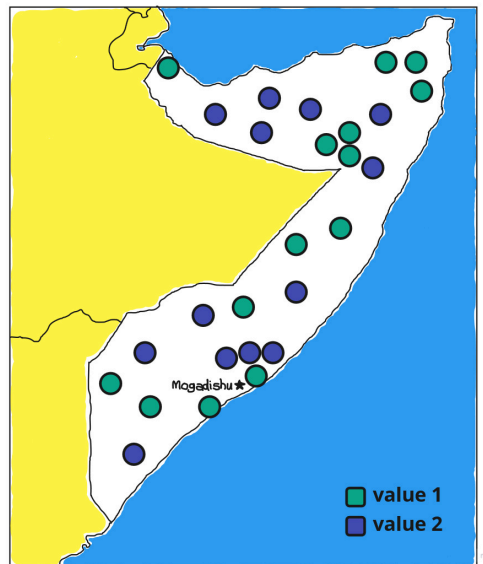
#### Forces ( 31)

Each individual has different attribute values. These values are used in this visualization idea as different dimensions that attract the individuals corresponding to their values. This idea is shown in Figure 2.8. For example, if an individual is an 18-year-old woman, who lives in Banadir, and the dimensions used in the visualization are the age of 18, female gender, and the district Bari as a home district, the individual would be displayed directly between the first two dimensions.

The idea aims to give an overview of the distribution of individuals regarding certain dimensions and enables users to discover similarities and differences between individuals.

We implemented this idea in our *Polygon Theme Forces* prototype using a different concept while keeping the idea of the forces. The prototype uses values of attributes as dimensions and aggregates the themes into bubbles with varying sizes depending on how often it got mentioned. The theme bubbles are then pulled into a direction of the dimensions according to how many individuals have the corresponding attribute values. For a description regarding the implementation, see section 3.4.7.

**Mechanized** We thought of fourteen ideas that use a *geometric projection* representation mode and *mechanized* interactions. This subset is the second largest in the categorization. It includes three more ideas that deal with the interaction of code and visualization (55, 59, 61). For these, the interaction is on a mechanized level, which means users do not have to manually edit code lines but can use dropdowns or checkboxes that support the programming aspect.



**Figure 2.9:** Map (3). Placing individuals as points on a map using the geographic positions of their districts

A big part of this category consists of combinations of different bar charts. They either focus on highlighting or excluding bars in the other charts (24, 25, 26, 27) or include a history of filter and selection steps (17).

This category also includes the idea of fusing diagrams into a stacked line chart (41).

Once again, we describe two ideas in this category in more detail. The first one is one of two map ideas (3, 4).

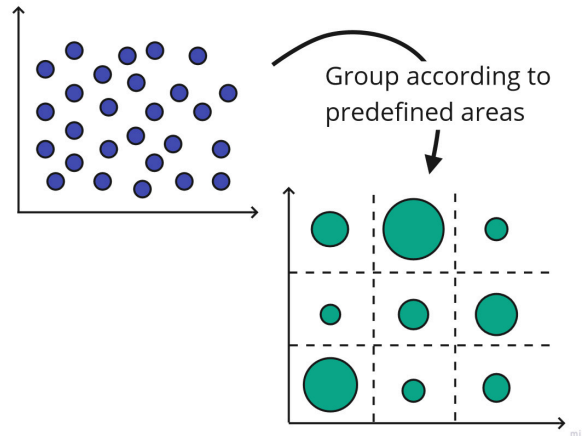
### Map (3)

The main goal of this visualization idea is to show the distribution of individuals based on their residence. This idea is illustrated in Figure 2.9. The individuals are placed on the coordinates of their residence using the geographic position on a map. All individuals can then be colored according to a selected attribute. That way, for example, rural and urban areas can be compared.

We implemented this idea in our *Map* prototype. For a description regarding the implementation, see section 3.4.1.

### XY-Axis (22)

The idea illustrated in Figure 2.10 was inspired by bar charts and scatterplots. It attempts to add interactions to bar charts by adding the possibility to determine the axes' attributes. The visualization idea aims at finding patterns and specific groups to explore. Therefore, the individuals can be grouped according to the attributes the users assigned the axes. The individuals are displayed in the areas as aggregated circles.



**Figure 2.10:** XY-Axis (22). Grouping individuals as points into predefined areas.

This concept offers the opportunity to gain an overview of individual groups according to different attribute values and to relate these groups to each other.

In the implementation of our *XY Diagram*, we maintain the individuals as points in the areas grouped by attribute. For a description regarding the implementation of this prototype, see section 3.4.2).

**Steerable** This category only consists of one visualization idea, which we now describe in detail.

#### **Individual Forces (76)**

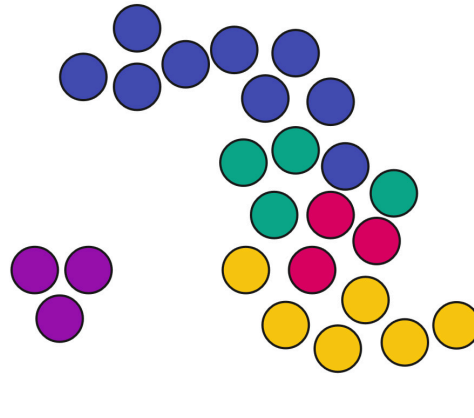
This idea is an algorithm that calculates the position of the individuals as points based on their similarity to the other individuals. That means that individuals who are more similar to each other are placed more closely. This placement is shown in Figure 2.11. The algorithm creates clusters of individuals that then show similarity patterns. Users can define the attributes the algorithm should base its calculation on and color the individuals according to a selected attribute.

We tried to implement this idea using two approaches, the *t-SNE* prototype and the *Individual Forces* prototype. For a description of the implementation of both prototypes, see section 3.4.6.

**Automatic** This category includes two visualization ideas. One idea is to automatically display a random message when clicking on a bar (35). The other idea is to display diagrams with a similar ratio to a selected one (29). As users cannot define the ratio they want to display independently from a diagram, this interaction is on an automatic level.

#### **2.3.3.3 Icon-based**

**Manual** This category combination is one of the smaller ones regarding the combination of any representation mode and a *manual* interaction level. It includes



**Figure 2.11:** Individual Forces (76). Attracting individuals as points to each other depending on their similarity

two ideas. The first idea is a map (2), in which individuals are displayed as shapes that represent, for example, their gender. All other attribute values are encoded in characteristics of the shapes, such as color, size, or position on the map. The second idea displays themes as icons (10). Per drag and drop, users can select one or more themes. Messages are then connected to these themes, creating a graph structure.

#### 2.3.3.4 Hierarchy-based

**Manual** Only one idea is sorted into this category. It attempts to convert a sunburst chart into a bar chart and thereby avoiding the 100% the sunburst chart tries to convey (16).

**Mechanized** Two ideas in this category work with the display of different diagrams for selected overlaps in Venn diagrams (38, 43). Other ideas are listing messages for a selected theme (36) and using bubbles to display the number of individuals, clustering them by theme (40).

We describe the remaining two ideas in this category in more detail.

#### Venn (39)

The idea aims at visualizing overlaps between different groups of individuals. It thereby offers an overview of all individuals, while still enabling users to compare the different individual groups. Users can determine attribute values to group by, and individuals are displayed inside their corresponding area. For example, in Figure 2.12, if an individual is a 25-year-old male, he is displayed in the overlap of the two bubbles “25 years old” and “male”.

We implemented the idea in the *Venn Diagram*, focusing on grouping by theme. See section 3.4.3 for a description regarding the implementation.

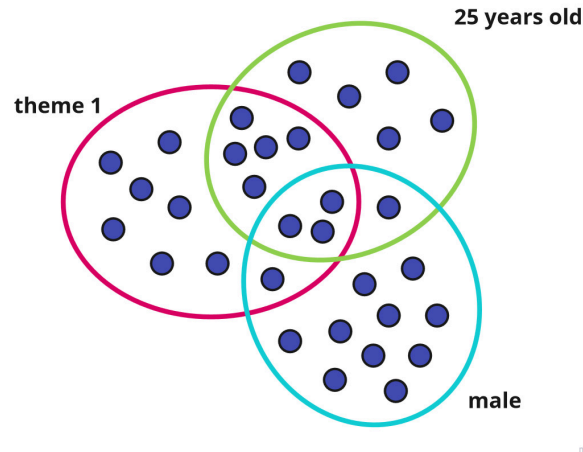


Figure 2.12: Venn (39). Individuals as points in a Venn diagram

### Group Chaining (75)

The idea was inspired by a visualization [25] that clusters dots of different sizes in circles. We adapted the concept in our *Group Chaining* idea. The core idea is that users can select different attributes, and individuals are then clustered in circles according to the attribute order. This grouping allows for an overview of all individuals and helps users to relate the individual groups with each other regarding specific attributes. The idea also includes a history of the chained attributes. For example, as shown in Figure 2.13, if users select *gender*, the individuals are clustered in circles according to their gender. If now *age* is selected, the corresponding circles are formed in each of the previously formed circles.

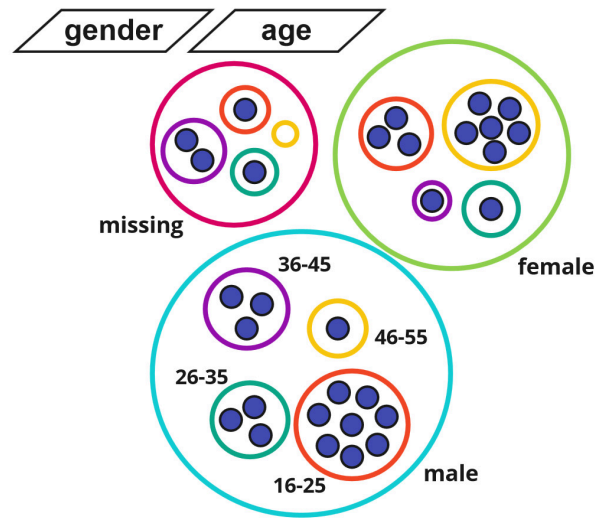
For a description regarding the implementation of our *Group Chaining* prototype, which implements this idea, see section 3.3.7.3.

#### 2.3.3.5 Graph-based

**Manual** This category contains four visualization ideas with interactions as well as four visualization ideas without interactions:

*Visualization Ideas Without Interaction.* The visualization ideas that have no interaction include an idea that uses different dimensions of a graph to convey information about messages or relationships between them (5). The color, for example, represents the theme of the message, the position represents the district of the individual, and an edge between messages displays that the individuals have the same age. Another idea is a graph structure that connects individuals to their mentioned themes (32).

*Visualization Ideas With Interaction.* The ideas with interaction include one that uses a minimap in the form of a graph structure to display an overview of all sorting and filtering possibilities in a bar chart (18). Another idea is the use of a mind map structure (19). By sorting the attributes in the center, the individuals are structured



**Figure 2.13:** Group Chaining (75). Chaining groupings of individuals as points according to selected attributes

as mind maps around the attribute values. The structuring is part of a mechanized interaction. Therefore, this idea is also sorted into the *mechanized* category.

**Mechanized** Two of the five ideas have already been mentioned in the *pixel-oriented* (9) and the *icon-based* (10) representation modes. The other ideas include a word cloud of the mentioned themes that displays a graph for a selected theme of the corresponding messages (11). As this idea uses text, it is also sorted into the *media-based* representation mode. Another visualization idea is to morph individuals according to filter arguments into different groups or shapes (6).

### 2.3.3.6 Media-based

**Manual** This category consists of two ideas that use word clouds to display an overview of the themes (1, 11).

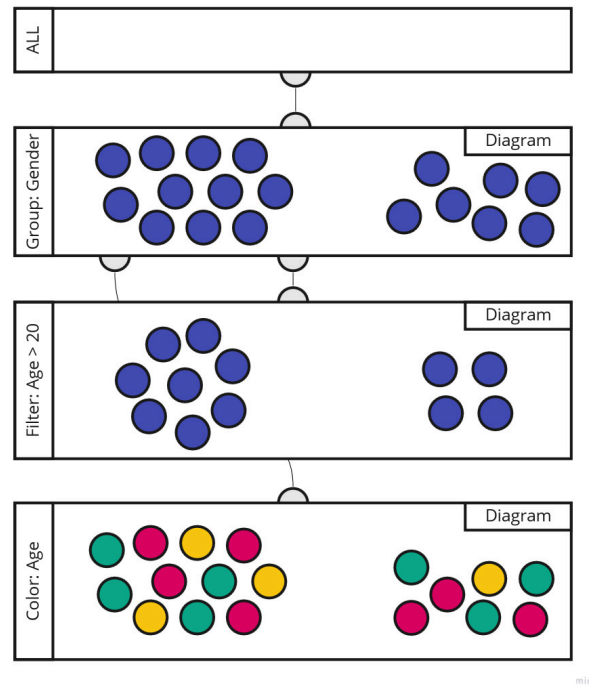
**Mechanized** We designed two visualization ideas that use sound. The first idea uses sounds that represent themes (12), such as the sound of rushing water for the theme *water*. The second idea plays recorded messages to convey the feeling of a market place (13). In both cases, the sounds can be filtered so that users can zoom in, for example, on messages they want to listen to more closely.

Another idea that uses media plays a video of the process that led to a selected recommendation, including the explored visualizations (53).

### 2.3.3.7 Hybrid

**Manual** This category includes two ideas. One idea represents a notebook (47), in which users can add different diagrams, notes, or statistics to get an overview of the





**Figure 2.14:** Panes (45). Windows containing individuals as points. Connections between windows make the exploration flow visible

data while having a place to note important insights right next to their source. We explain the other idea in more detail.

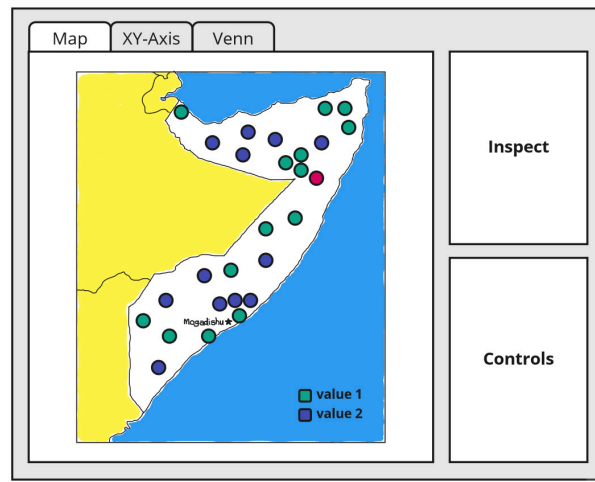
### Panes (45)

The visualization idea was inspired by the concept of swimlanes that is used, for example, by BPMN. In BPMN the swimlanes represent participants of a business model and contain objects that are executed by the corresponding participant.<sup>4</sup> In our idea, the swimlanes are windows that each represent a state of the data. Instead of objects, these windows contain individuals as points that can be grouped, filtered, and colored. Users can drag and drop individuals to select them and create new windows that are then connected to the original, as shown in Figure 2.14. They can also comprehend the history of filter and selection steps by tracing the connections between the windows. The visualization idea includes a switch between the representation of the individuals as dots and standard charts, such as bar charts, that pose a statistical view of the data.

The main goal of this idea is to relate the individuals to each other by comparing them and zooming in on interesting patterns.

We implemented the idea in two different approaches. The first, our *Panes* prototype, tries to implement the original idea. In the second prototype, the *Tree View*

<sup>4</sup><https://www.visual-paradigm.com/tutorials/bpmn2.jsp> (last accessed 2020-07-29).



**Figure 2.15:** Tab View (77). Tabs containing one visualization type each. Inspector to inspect individuals. Control panel to color, filter, and select individuals in the visualizations

prototype, each window, instead of displaying the individuals randomly, consists of one visualization type. For a description regarding the implementation of these prototypes, see section 3.4.9 for the *Panes* prototype and section 3.5.3 for the *Tree View* prototype.

**Mechanized** As the just described idea (45) includes mechanized interactions such as filtering or coloring, it is also sorted into this category. This category includes another idea for a hybrid of visualizations, the *Tab View* (77).

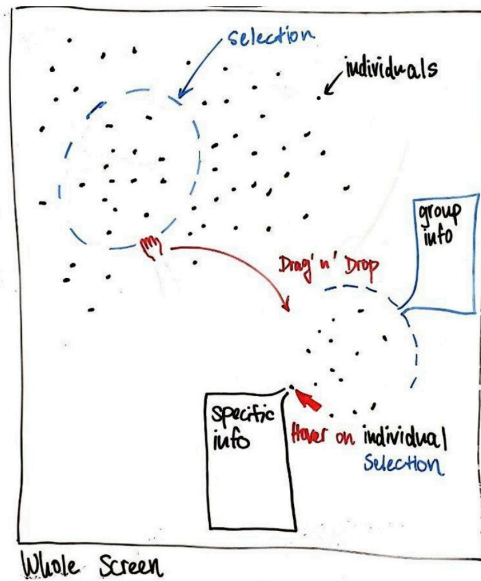
### **Tab View (77)**

The main goal of this visualization idea is to present different perspectives on the data using different visualization types. The visualization idea consists of multiple tabs. Each tab contains one visualization, such as a map or a Venn diagram, as displayed in Figure 2.15. It further includes an inspector to inspect individuals and a control panel that allows global coloring, filtering, and selecting individuals in the visualizations.

The idea allows for an overview of the data by displaying several visualizations. It further enables users to find correlations between visualizations.

We implemented this idea in the *Tab View* prototype. See section 3.5.2 for a description regarding the implementation.

**Automatic** This category includes the notebook idea (47) that was already mentioned. The *automatic* interaction level in this visualization consists of an export function that automatically creates a report for the researcher, including all diagrams and notes the user gathered in the notebook.



**Figure 2.16:** Individuals on Canvas (46). Individuals as points that can be grouped, selected, and filtered. Hovering displays information regarding the individual or a group of individuals.

### 2.3.3.8 No Specific Representation Mode

**Manual** This category includes interactions that support different kinds of actions. Two ideas support filtering by offering either a freehand selection (71) or functionality that allows dragging and dropping bars in and out of hiding (69). For a description regarding the implementation of the FreehandSelection, see section 3.3.5. Another idea provides functionality that allows users to drag an axis of their choice to the position of the x-axis, thereby adding a second dimension to the fixed y-axis (23). Another interaction idea lets users connect the attributes for a scatterplot per drag and drop (72).

The two remaining ideas support inspecting individuals. The first idea is to open an inspector for a selected individual that displays wanted information (78). The second idea uses the same concept but displays additionally to the information a sketched image of the individual that is being inspected. For a description regarding the implementation of the ideas, see section 3.3.1 and section 3.3.2.

### 2.3.3.9 Other Concepts

**Manual** This category consists of four visualization ideas that extend each other and work with the same concept. The main idea is to display individuals as points on a canvas and being able to select, inspect, and group them (46). This idea is shown in Figure 2.16. The other three ideas in this category add a filtering and coloring functionality to this concept (80, 14, 15), which assigns them to the *mechanized* category as well.

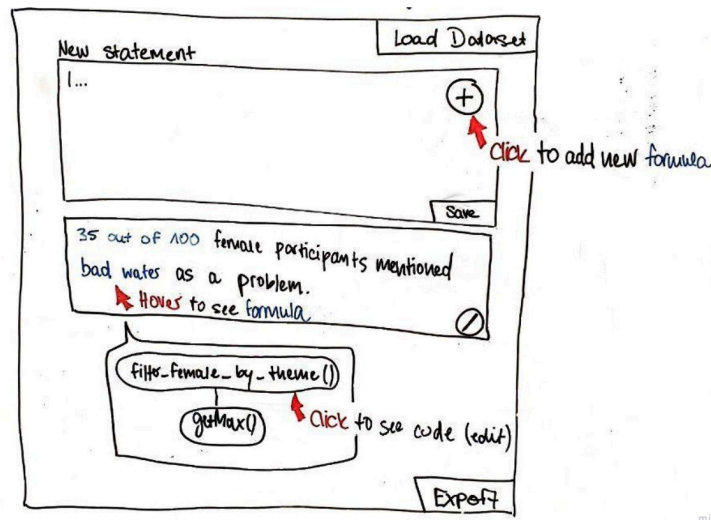


Figure 2.17: Statement Generator (63). Generating statements using code formulas and templates

**Mechanized** Two of the six visualization ideas in this category visualize selected groups or recommendations with different diagram types (52, 28). Another idea is to make selected groups of individuals vibrate to convey a feeling of liveliness (70).

**Instructable** This category consists of the only *instructable* visualization idea in the whole categorization. The idea is a statement generator that uses formulas that users can write to create statements that include statistical information (63). An example of this concept is shown in Figure 2.17.

**Steerable** This category includes one of the visualization ideas mentioned in the *manual* category (80). The *steerable* interaction level of this idea is the display of groups of individuals similar to a selected group or with no similarity at all. The remaining idea in this category deals with the interaction of code and visualization (60). It allows users to write what they want to do in natural language. An algorithm then generates a suggestion displaying the edited code and the changed visualization.

### 2.3.4 Task Dimension

In the following, we shortly describe each category of the task dimension and sort the examples we have presented in the previous section into the corresponding categories. An overview of all ideas can be found in Appendix section A.3. Regarding the categories *Overview* and *Relate*, the detailed descriptions of the ideas offer reasons for their placement into the categories. For all other categories, explanations can be found in the following descriptions.

**Overview** This category is with 45 visualization ideas, the largest in this categorization. It includes ideas from all representation modes that pose an overview of the data. Of the ideas we have already presented in more detail, the *Map* (3), *Tab View* (77), *Group chaining* (75), *XY-Axis* (22), *Forces* (31), *Individual-centered* (7), *Venn* (39), and *Theme Centers* (33) are part of this category.

**Zoom** With five visualization ideas, this category is one of the smaller ones. It includes the *Panes* (45) idea. This idea allows selecting a group of individuals to explore them further while keeping the other individuals visible. To zoom back out, users can trace the connections back to other states of the data.

**Filter** This category is with eight ideas also one of the smaller ones. It includes several interactions that enable filtering, such as the freehand selection (71) or dragging and dropping bars in and out of hiding (69). Other ideas describe filtering functionality in different contexts, such as in randomly displayed points on a canvas (14, 15). None of the more detailed described ideas are part of this category.

**Details-on-demand** This category includes all ideas that display details on a selected item. A majority of the visualizations dealing with the interplay of code and visualization are part of this category (54, 55, 56, 58, 62, 61). It includes the two inspect actions that display information regarding a selected individual (78, 79).

**Relate** *Relate* is the second largest category in this categorization. The majority of the ideas are based on *geometric projection* techniques. Another part of this category uses *hierarchy-based* and *graph-based* techniques. This category includes all visualization ideas that make it possible to comprehend correlations between visualizations, individuals, or messages. An example is highlighting a selected individual in all other displayed charts (44). Of the ideas we have already presented in more detail, the following are part of this category: *Tab View* (77), *Group chaining* (75), *XY-Axis* (22), *Forces* (31), *Individual-centered* (7), *Venn* (39), *Theme Centers* (33), *Panes* (45), and *Individual Forces* (76).

**History** This category is one of the smaller ones in this categorization. It includes visualization ideas that allow users to trace their exploration steps in the visualization (4, 17, 45, 75) or visualize the history of process steps (48, 49, 51). The former include *Panes* (45) and *Group chaining* (75).

**Extract** *Extract* is the smallest category with only four visualization ideas. It includes the freehand selection (71), as this interaction allows users to extract individuals to further explore or process them in another visualization context.

The category further includes the idea of the notebook (47) and the statement generator (36), which both support an export functionality of a report or statement.

Of the already described ideas, *Panes* (45) is part of this category. The idea allows users to extract individuals into a new window to explore or process them further.

### 2.3.5 Findings

We created the categorization to gain an overview of the distribution of the ideas. Such an overview includes determining dead ends, blind spots, and recurring ideas to comprehend which directions have not been considered yet and in which there is an accumulation of ideas. The latter can provide insights for directions that are already exhausted.

Due to the Design Thinking approach we applied, the developed ideas build upon users' needs. With the "How might we"-questions described in section 2.2.2.3, we steered the ideation to ideas that are purposeful, meaning that they directly try to solve the users' problems. Therefore, accumulations hint at ideas that are especially suited to solve the given problems.

In this section, we present the findings we were able to extract from the categorization. For this, we also consider the combination of the different taxonomy dimensions. For example, whether a representation mode displays a certain task often or whether an interaction level is needed to implement a certain task. For tables with the ideas regarding the combinations *interaction level* and *tasks*, and *representation mode* and *tasks*, see Appendix section A.4 and Appendix section A.5.

As there will always be new visualization ideas that can be added to the categorization, it has to be noted that this categorization is not complete. Therefore, the following findings only refer to the visualization ideas we designed and should only be generalized carefully.

#### 2.3.5.1 Distribution of Ideas

We structure our findings regarding the distribution of the ideas by first giving an overview of accumulations and blind spots in the technical dimension. Then we do the same for the task dimension.

#### 2.3.5.2 Technical Dimension

**Accumulations** The ideas in the technical categorization mainly have a *manual* or *mechanized* interaction level. Further, all interaction ideas that could not be assigned to a representation mode have a *manual* interaction level. *Mechanized* interactions are distributed almost evenly through all task categories, whereas ideas with a *manual* interaction level focus on *Overview* and *Relate* tasks. Therefore, all tasks have a clear focus on more direct interactions.

**Blind Spots** We focused on designing explorable visualizations. The *automatic*, *steerable*, and *instructable* categories regarding the interaction level provide more automated interactions. Users of such visualizations do not have much interaction freedom. The fact that there are only seven visualization ideas sorted into the more automated interaction levels implies that explorable visualizations need less automated and more direct interaction possibilities. Regarding the representation mode, the visualization ideas accumulate in the *geometric projection* and *graph-based* categories creating a blind spot around the other representation modes. These

techniques are commonly used representation modes, whereas the other modes are more specific, making this blind spot somewhat unsurprising.

### 2.3.5.3 Task Dimension

**Accumulations** In the task dimension, *Overview* and *Relate* are the largest categories. Many visualization ideas are also part of the *Details-on-demand* category. We designed visualizations to offer researchers an explorable way to interact with their data. The accumulation of ideas in the three task categories implies that visualizations supporting such tasks are more promising to support researchers in exploring the data in more detail.

*Overview*, *Zoom*, and *History* only have *manual* and *mechanized* interactions. Ideas in the *Filter* category focus on *mechanized* interactions. All other task categories have either an even distribution between *manual* and *mechanized* interaction levels or have a slight tendency to *manual* interactions. This distribution implies that filter functionality allows for complexity, which manual interactions cannot implement.

**Blind Spots** The underrepresentation of visualization ideas that support *Zoom*, *Filter*, *History*, and *Extract* leads to the conclusion that the other categories offer more directions for the design of visualization ideas and are more important in the context of our project. However, the visualization ideas in these categories can extend ideas in other categories by adding more functionality. This extension creates more possibilities to explore the visualization and its data.

The ideas included in *Overview* are distributed over all representation modes. In all other task categories, there are some blind spots regarding the representation mode. These blind spots imply that some representation modes are better applicable for certain tasks than others. For example, *Zoom* includes no *geometric projection*, *pixel-based*, and *icon-based* techniques, whereas *Details-on-demand* has a focus on *geometric projection* techniques.

Of the more automated interaction levels, namely *instructable*, *steerable*, and *automatic*, half of the ideas are part of *Relate*. The other ideas are distributed over *Filter*, *Details-on-demand*, and *Extract*. As we have only thought of a small number of ideas that can be sorted into the combination of *Relate* and the more automatic interaction levels, this category combination yields another blind spot. When designing visualizations that present relationships between data points, it may be interesting to look at the possibilities that more automatic interaction levels have to offer.

### 2.3.5.4 Recurring Concepts

There are several recurring concepts among the visualization ideas we designed. Such recurring concepts hint at the more promising ideas to solve the given problems.

**Individuals As Points** In about one-fourth of all ideas, we used the concept of individuals as points (2, 3, 5, 6, 7, 14, 15, 19, 22, 31, 32, 33, 34, 39, 44, 45, 46, 60, 67, 70, 75, 76). This concept prevents losing individual messages and opinions in aggregated data. The amount of ideas using this concept implies that displaying

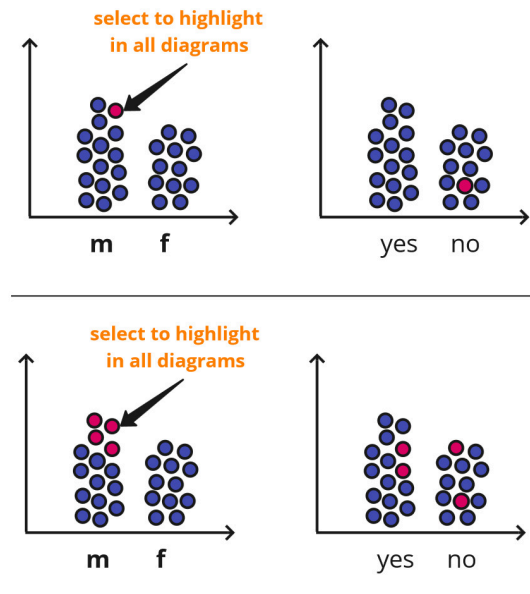


Figure 2.18: Highlighting points across diagrams (44)

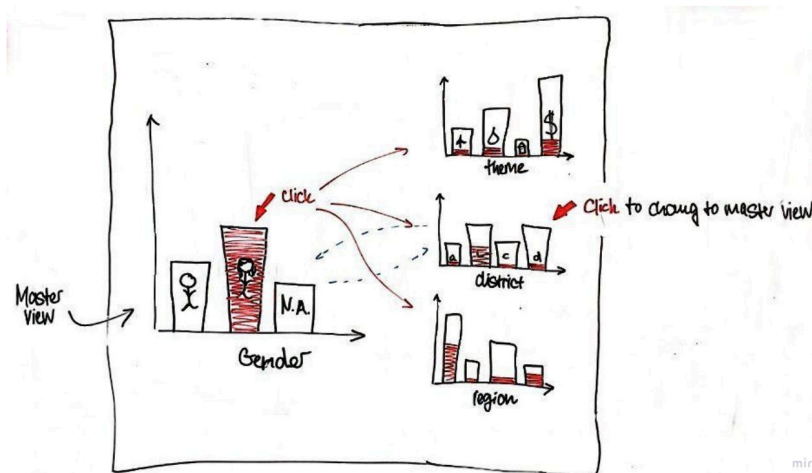


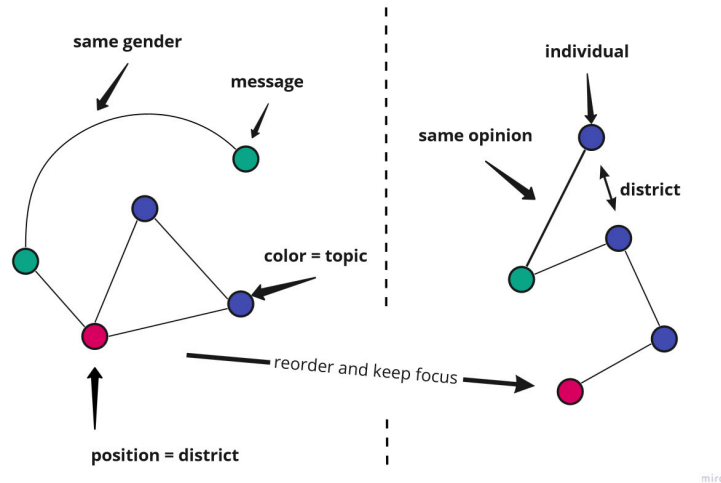
Figure 2.19: Highlighting bars across diagrams (25)

individuals as points is an easy concept to design visualizations that focus on the individual.

**Highlighting Correlations** We thought of several ideas that support users in finding correlations across diagrams by highlighting related components (24, 25, 26, 27, 34, 44). An example is illustrated in Figure 2.18.

In this visualization idea (44), users can select one or multiple points in one diagram, which are then highlighted in the other diagrams. Another idea uses a similar concept for bar charts.





**Figure 2.20:** Graph structure displaying relationships between messages and individuals (5)

In the idea (25), users can select a bar to highlight the corresponding parts in the other diagrams, creating stacked bar charts. They can further select different diagrams as the main view. This process is shown in Figure 2.19.

**Distribution On Maps** We used maps to present the distribution of individuals (2,3) or themes (1, 4) over the districts. Displaying such distributions on a map and not merely grouped by district, makes it easier for users to comprehend the distribution.

**Graph Structures to Show Connections** About one-eighth of the ideas use graph structures to visualize connections or relationships. As the edges of graphs are a common way of expressing relationships, most of these ideas are part of the *Relate* category (5, 8, 9, 10, 32). These relationships can be divided into different categories; relationships between individuals (5, 6), relationships between messages (5, 8, 9), relationships between messages and themes (10, 11), and relationships between individuals and themes (32). Figure 2.20 displays an idea that uses this concept.

The edges of this graph represent whether adjacent messages were sent by individuals that have the same age. Other attributes of the message or the individual are represented, for example, by color or the position. The graph can further be reorganized so that the points represent the individuals instead of the messages. We believe, however, that graphs are only useful up to a certain amount of individuals. As the graph becomes more complex, it is more difficult to discover insights.

**Filtering and Grouping** The concepts of filtering (4, 6, 12, 13, 14, 15, 20, 24, 45, 69, 71) and grouping data (19, 22, 33, 38, 39, 40, 43, 46, 75, 76) are part of eleven and ten visualizations, respectively. It enables users to explore the data independently and offers different perspectives on the visualizations and their data.

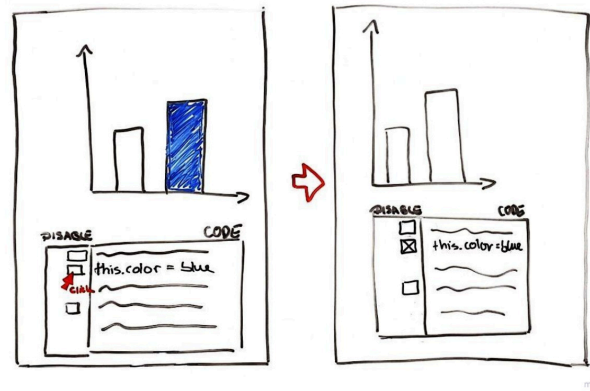


Figure 2.21: Highlighting points across diagrams (55)

**Coloring** We integrated the use of colors in fifteen visualization ideas to encode attribute values (2, 3, 5, 21, 40, 45, 73, 75, 76, 77) and in eight ideas to enable highlighting for interactions such as inspecting or selecting individuals (14, 15, 25, 26, 27, 34, 44, 56, 78, 79, 80).

**Interplay Code and Visualization** Ten visualization ideas deal with the interplay of code and visualization (54, 55, 56, 57, 58, 59, 60, 61, 62, 63). They offer a different interaction type for visualizations, allowing even programmers with little experience to adapt them easily. An example of this kind of interaction offers Figure 2.21.

This idea (55) allows users to view the code next to the corresponding visualization. Checkboxes let users disable certain lines of code. In this example, users can disable the code line that colors the bar. The changes are then adapted directly in the displayed visualization.

The interaction possibilities with the code also offer an opportunity for the work with legacy code by highlighting visualization components when hovering over the corresponding code lines.

## 2.4 Concepts

An essential aspect of our search for solutions was to consider the requirements of our project partner. These requirements included designing visualizations that make the data and its provenance explorable, develop empathy for the individuals displayed in the visualization, and create trust in the presentation form. For a more detailed description regarding the requirements, see section 1.2.2.

Based on the findings we extracted from the categorization, we discovered several concepts that underlie our visualization ideas and realize the requirements mentioned above. It has to be noted that, as the concepts are only based on the ideas in our categorization, there may be other concepts that support such goals.

In this section, we give examples and describe the concepts that implement each requirement.

### 2.4.1 Explorability

In the following, we describe the concepts in our visualization ideas that allow for an exploration of the data. The concepts can be divided into four approaches. The first approach is the display of individuals as points. The second is the possibility of users to view the data from different perspectives. Similar to this concept is the third approach that deals with the comparison of different visualizations. The fourth approach we used is the ability to trace the steps in the exploration path.

#### 2.4.1.1 Individuals As Points and Inspection

One recurring concept is the representation of individuals as points. It prevents losing the connection to individuals and their opinions in an aggregated data format. As each individual is always visible, users can, at any point, inspect them and go back to the individual messages. Thereby, the data itself and the provenance of the data can be explored, meaning all values and information the data points consist of. Whatever interaction users perform on the visualization, the representation of individuals as points in combination with the inspection enables users to always reach the original data points. The individuals' opinions can be explored and related to other information the visualization conveys about the individuals, for example, age, gender, or residence.

#### 2.4.1.2 Viewing Different Perspectives

Another concept that enables the exploration of data in visualizations is viewing the data from different perspectives. The number of perspectives on the data in one visualization is limited, and different visualizations highlight different aspects. Considering different views or states of the data therefore offers insights that one visualization may not be able to convey. Throughout our visualization ideas, four concepts support the implementation of this concept.

**Manipulation of Data** One way of altering the perspective on the data is to alter the data displayed in the visualization. A recurring concept in our visualization ideas is the filtering of data points. As all irrelevant data points are left out, filtering makes it easier for users to explore the data in certain directions.

**Manipulation of Views** Another possibility is to directly manipulate the views, meaning the arrangement of visualization components such as bars or points. This manipulation is especially interesting in the context of individuals represented as points. Concepts that we found in our categorization were grouping individuals according to attribute values and manipulating the individuals' positions in the visualization.

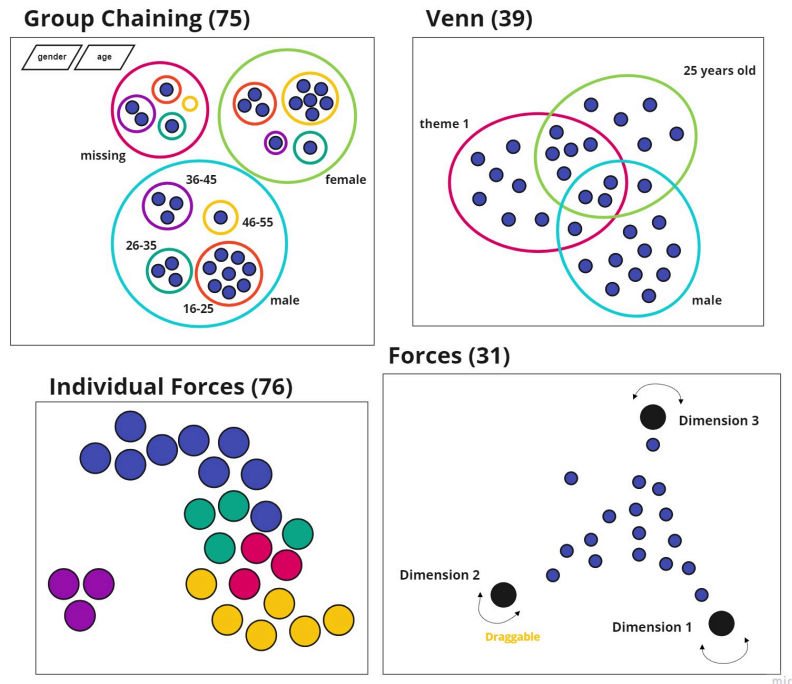


Figure 2.22: Ideas manipulating the view of data

For the former, examples are *Group Chaining* (75) and *Venn* (39). Ideas that manipulate individuals' positions are *Forces* (31) or *Individual Forces* (76). These ideas are shown in Figure 2.22.

Manipulating the view of the data can help to understand relationships across the data points. Depending on, for example, how similar they are, they will stay closer together, as in *Individual Forces*, or morph into the same group, as in *Group Chaining*.

**Manipulation of Code** The code that creates a visualization offers another perspective on the visualization. Understanding the code helps to understand the visualization and its data. Interacting with the code offers opportunities to modify the visualization that are not as limited as the interactions in the visualization UI. Therefore, interacting with the code can increase the variety of views a visualization can display. Further, the interaction with the code enables users to explore the provenance of the visualization, meaning the different steps that lead to its creation.

**Different Visualization Types in One Application** Viewing different perspectives of the data in the context of visualization means to view different visualization types. One conceptual idea is the *Tab View* (77). This idea combines different visualization types in one application. It allows users to quickly switch between different visualizations, enabling them to gain an overview of different perspectives on the data. The *Tab View* further includes global filtering, selecting, and coloring

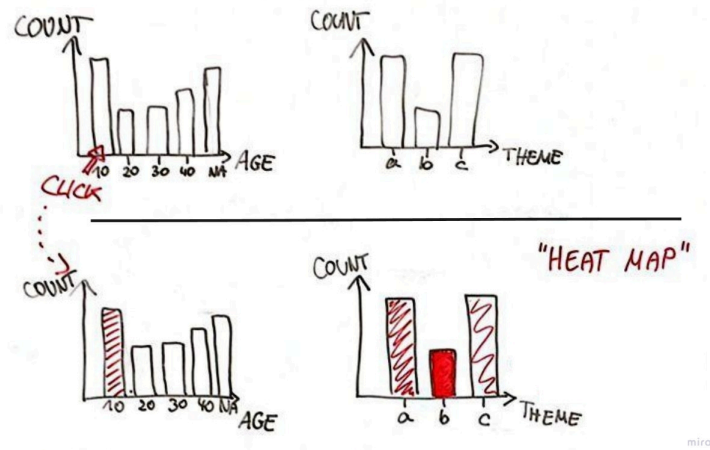


Figure 2.23: Highlighting bars across diagrams (27)

functionalities. Thus, the perspectives on the data are always related to the same state.

#### 2.4.1.3 Comparison Across Visualizations

Another recurring concept is the comparison across visualizations. This concept is closely related to viewing different perspectives of the data. But whereas the latter enables users to gain insights by displaying different visualization types, modifying the view or the data itself, the comparison of visualizations focuses on displaying different visualizations next to each other.

Comparing visualizations allows users to discover differences and similarities in the data. As users do not have to go back and forth between visualization, displaying visualizations next to each other makes the exploration of such correlations easier.

In our visualization ideas, we use different approaches when displaying multiple visualizations as part of one. One approach is the display of multiple bar charts. When selecting a bar in one of them, the corresponding parts of the bars in the other charts are highlighted, creating stacked bar charts. This highlighting enables users to easily compare proportions in the charts and find correlations that would otherwise have not been visible. An example of this concept is displayed in Figure 2.23. By selecting a bar, the bars in the other bar chart are colored depending on the number of individuals with the respective attribute value. Thereby, a heatmap is created.

Our categorization includes many ideas that work with displaying different visualizations for a selected group, message, or recommendation. Displaying different visualizations helps users to place the selected group or statement into context. It enables users to compare them to other groups or individuals, thereby allowing an exploration of similarities and differences. In the context of recommendations, it further allows the exploration of the underlying insights. The idea illustrated in Figure 2.24 displays different visualizations for a selected message.

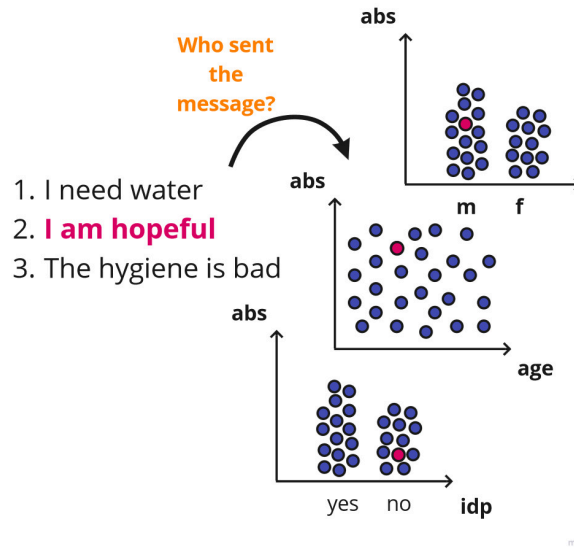


Figure 2.24: Highlighting points across diagrams (34)

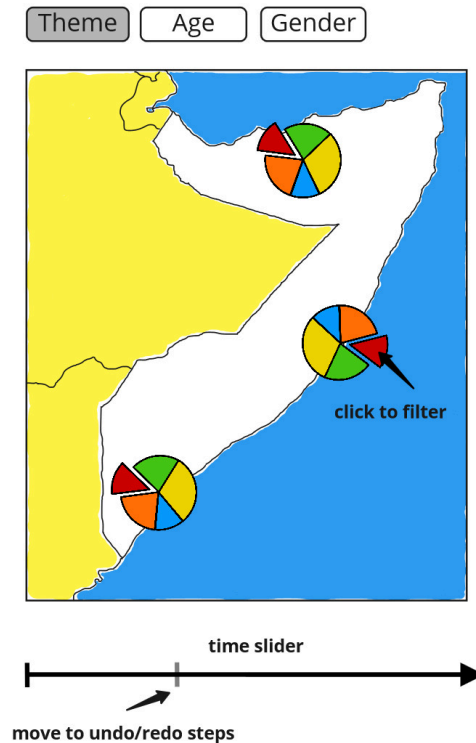
Another approach is the idea *Panes* (45) shown in Figure 2.14. All windows in the visualization idea use the same representation of individuals, the point. This unvarying representation form allows users to map individuals to different views without having to rethink frequently. Not having to rethink every time a new visualization view is displayed makes it easier for users to explore correlations. In our implemented prototype, instead of only displaying the individuals randomly, each window contains one type of visualization. The use of multiple types of visualizations in one application enables users, as already mentioned, to explore similarities and differences more easily.

#### 2.4.1.4 Traceability of Exploration Paths

Filtering can be described as a possibility to zoom in on parts of the data. To take all ignored data points back into consideration, users have to zoom out again. Therefore, an important concept is to ensure that the steps in the exploration path are kept traceable. The concept includes that users can go back to a specific state and keep exploring from there. This concept is also applicable to selecting, coloring, and grouping functionalities.

An advantage of being able to trace the exploration path up to a specific state is that users can explore the visualization in different directions without having to begin with a visualization from scratch. It enables a faster way of a broad exploration.

*Panes* (45) is one of the ideas that implement this concept. Each window represents one step in the exploration path. Users can easily trace the connections between the windows and create new windows when they desire. Another example is *Group Chaining* (75). This idea creates a chain of groupings and allows users to redo groupings as desired.



**Figure 2.25:** Map with time slider to redo or undo filter or selection steps (4)

Figure 2.25 shows a third example. This idea (4) displays pie charts for each district on a map depending on the selected attribute. By clicking on a slice of the pie chart, this attribute value is filtered out of all districts. A time slider allows users to undo or redo filter and selection steps as desired.

## 2.4.2 Empathy

The second requirement we considered while designing the visualization ideas, is to create empathy. We describe the two concepts we used in our designs that enable users to develop empathy for the individuals and their opinions and needs.

### 2.4.2.1 Individuals As Points and Inspection

It is easier to relate to individuals as points than when they are hidden behind aggregated data representations. Users are always reminded that there are real individuals behind the data points. This reminder is enforced by the possibility to inspect each individual. The messages that are displayed when inspecting an individual give the individual a voice. It further helps users to identify with the needs of the individuals, thereby creating empathy.

#### **2.4.2.2 Liveliness**

Our categorization includes many ideas that aim at conveying the liveliness of the individuals. Points already resemble individuals better than bars in a bar chart. For example, adding a certain behavior to the points helps increase empathy for the individuals behind them. It acts as a reminder that they are no simple data points, but humans with complex needs and opinions.

We used three approaches to implement this concept. One approach is the usage of movement. The ideas include moving the points between their themes (33) or letting them vibrate when selected (70).

Another approach is the usage of images. The idea is to display a sketched image of the individual being inspected to help users identify with the individual and remind them that they are real (79).

The third approach is the usage of sound to present the messages (13). The messages are read out loud to convey the feeling of a marketplace. As an individual reads each message, it helps to create empathy with the individuals behind the messages.

### **2.4.3 Trust**

Another requirement in our visualization ideas was to create trust in the truthfulness of the data's representation. The concepts we used to achieve this are the inspection of individuals or messages, the interplay of code and visualization, and keeping missing data visible.

#### **2.4.3.1 Inspection**

A concept that enables trust in the representation form is the inspection of, for example, individuals or messages. For example, when comparing individuals regarding their themes, directly getting the corresponding messages creates trust in the presentation's truthfulness. This verification can be important for all demographic data as well. The inspection of individuals and messages is one way for users to gain trust in the visualization.

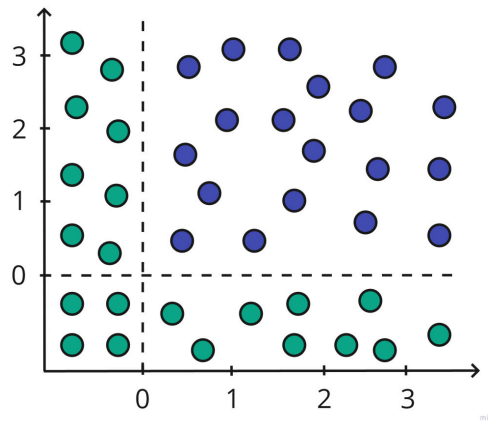
#### **2.4.3.2 Manipulation of Code**

Interacting with the visualization using the underlying code helps users to understand how the visualization is created. Understanding how the data points are modified before being displayed in the visualization creates trust in the correct visualization of the data.

#### **2.4.3.3 Keep Missing Data Visible**

In data sets, it is common that some values are missing for some data points. It is important to remind users that insights might not apply to all data points as the data is incomplete. Doing so, creates trust in the gained insights, as users can comprehend to which group of individuals insights apply. Thus, missing data should be made visible.





**Figure 2.26:** Displaying missing data in scatterplots (65)

We thought of different ideas that support this goal. One of them is displayed in Figure 2.26. The idea (65) adds an area to a scatterplot in which data points are displayed that do not have values for one or both attributes of the axes. Thereby, data points that only miss one value can still be displayed in the diagram.

For a map, a way to display missing data is to sort all individuals with a missing value for their district into a separate box at the side. That way, just as in the example shown in Figure 2.26, the individuals can still be considered for filtering, selecting, and coloring.

## 2.5 Conclusion

Our goal was to design visualizations that improve the finding of insights in highly complex data of thousands of individuals.

We designed about 80 ideas that go into various directions to allow for an in-depth exploration of the data. From those ideas, we were able to extract several concepts that help create visualizations that make the data explorable, let users develop empathy, and create trust in the truthfulness of the representation form. Our main concept is to display the individuals as points to prevent losing their voices in aggregated data. In combination with the inspection of individuals and messages, this concept allows for an interactive exploration of the data. As a visualization only has a limited number of perspectives on the data, another important concept is to enable users to relate different visualizations and perspectives.

Our work provides a foundation for further research regarding explorable visualizations and individual-centered approaches. It further poses a fundamental basis for the design of visualizations that display highly complex data about individuals. Therefore, it forms a starting point for Africa's Voices and, in general, for people who work with this kind of data, to design and create visualizations that have more to say than only statistics.

## *2 Concepts for Visualizations and Exploration and Categorization of the Design Space*

In the context of our project, the categorization and the extraction of concepts helped us decide on ideas that allow for an in-depth exploration of the data. These ideas were then implemented as prototypes in the further course of the project, offering a first opportunity to explore highly complex data with novel visualizations that go beyond a proof-of-concept.

## 3 Implementation and Integration Into an Environment of Explorable Visualization Tools

Our project aimed to create explorable visualizations centered on individuals that help Africa's Voices gain insights about their data sets. To provide Africa's Voices with a foundation for future implementation of explorable visualizations, we tested the feasibility of our ideas. Building upon the exploration of our design space described in section 2.2, we implemented ten visualization tools from which we extracted six interaction patterns. We further explored two approaches for integrating the visualization tools and interaction patterns into visualization environments. In this chapter, we provide an overview of these tools and environments and discuss essential considerations for implementing and integrating explorable visualization tools. These considerations can provide guidance for further developers and designers of explorable individual-centered visualizations.

### 3.1 Introduction

Our project's goal was to provide Africa's Voices with explorable visualizations focused on individuals. As described in section 2.2.2.4, during our idea generation phase, we designed around 80 visualization ideas. We then started implementing and validating our ideas with our project partner.

Our aim was not to create a functioning production application for Africa's Voices, but rather to examine if our ideas are feasible and to provide valuable insights and a basis for further development. In the following sections, we classify and discuss our prototypes. We additionally highlight interesting findings in implementing them which might be of use for someone attempting similar work.

We started our project by choosing the individuals as points design concept described in section 2.4.1.1, as well as interaction patterns commonly used in our ideas, like the filtering and grouping of points, to validate if they were feasible. After validating these concepts, we selected promising visualization ideas and began implementing them.

During our project, we implemented various ideas ranging from interaction patterns on visualizations to environments that integrate multiple visualizations.

The fundamental idea connecting nearly all of our prototypes was to represent individuals as points and keep them visible at all times. We were able to show

that this approach works well with several thousand individuals. Section 3.2 demonstrates and explores this further.

As described in section 2.4.1.2, one of the concepts we identified for making visualizations explorable is to allow users to manipulate the views of visualizations. Therefore we strive to allow users as much interaction as possible. We started with implementing interaction patterns that had arisen multiple times in our ideation process. Filtering, coloring, and grouping were recurring concepts in our ideas, see 2.3.5.4, their implementation is discussed in section 3.3.4, section 3.3.3, and section 3.3.7. Being able to inspect individuals to access their information at all times was an important part of the individuals as points concept described in section 2.4.1.1, therefore we also early on started working on the inspect action as is discussed in section 3.3.1. During the development of our prototypes, further interaction patterns with specific goals, for instance, creating empathy with the inspected individuals, emerged. The resulting patterns are described in section 3.3.2, section 3.3.6, and section 3.3.5. All developed interaction patterns work well with the individuals as points design model.

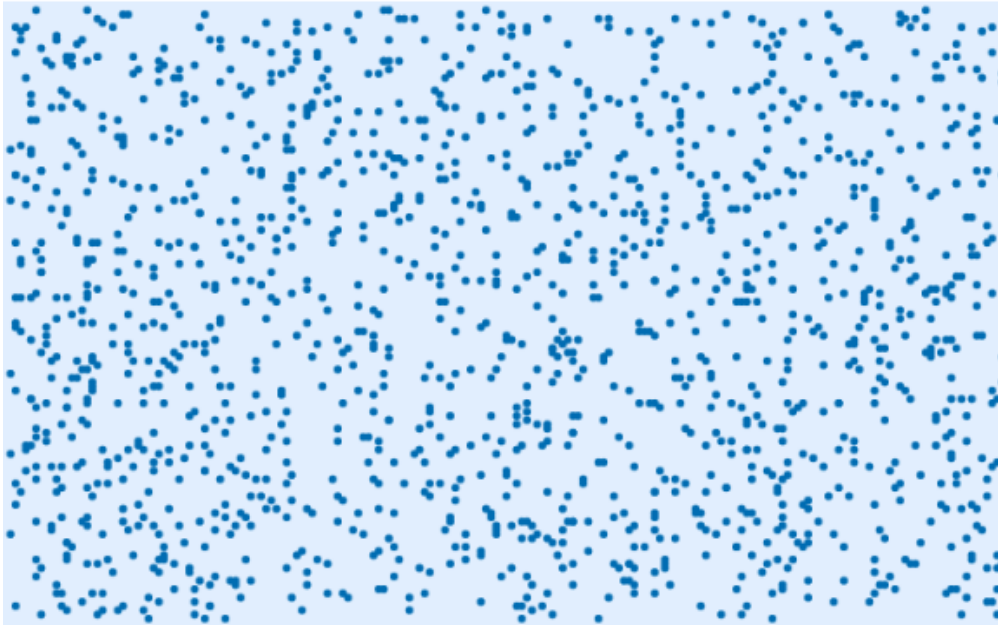
Following our first exploration of working with the individuals as points design model, we progressed to implement visualization tools. Each visualization tool provides one perspective, which means a specific view of the data. While the actual implementation often varies greatly from the original idea, most visualization tools can be traced back to an idea categorized in section 2.3. Our implemented visualizations are discussed in section 3.4.

Finally, to work with the concept of different visualization types in one application as described in section 2.4.1.2, we tested two approaches for integrating the visualization tools and interaction patterns into visualization environments. Section 3.5 discusses our two approaches, the *Tab View* prototype 3.5.2 and the *Tree View* prototype 3.5.3, compares them and evaluates the required integration levels for integrating visualization tools.

To structure our tool discussion, we discuss the following points for each presented interaction pattern, visualization tool and integration approach:

**Goal** The main goal for each prototype regarding its functionality and use for the user usually was consistent throughout the iterations and is noted first for each prototype.

**Metaphor** After prototyping for some time, we noticed that a shared metaphor had emerged, which facilitated communication and influenced new ideas. Our main goal for every prototype was to always keep a connection to every single individual. From the idea of displaying individuals as points, the metaphor of users of the visualization tools standing in front of a massive crowd of citizens and being able to interact with them emerged. Parts of that metaphor can already be found in our ideation process, where we used the experience of going from community to community and talking with its people as inspiration. Most of our resulting prototypes can be mapped to a real-world interaction with a crowd, given that they all have time and interest in following your instructions.



**Figure 3.1:** Individuals displayed as points

**Execution** During the prototyping process, new ideas emerged, and existing prototypes changed until their original idea was not necessarily recognizable anymore. We describe the most advanced state of functionality of each prototype.

**Considerations** Through implementing our ideas, we gained valuable insights, which can assist in future implementation processes. For the insights and some prototypes, starting points for future work are noted.

### **3.2 Individuals as Points**

Most of our prototypes use the individuals as points concept described in section 2.4.1.1. Provided a sufficient size of the visualization, in the developed prototypes users are able to distinguish and inspect single individuals at all times. This shows that representing individuals as points is a feasible strategy for visualization tools.

**Goal** The overall aim our prototypes is to keep a connection to each individual. We want to avoid aggregating and to allow for more empathy for the individuals represented in the visualization.

**Metaphor** Seeing a crowd of many individuals with different backgrounds, opinions and needs in front of you.

**Execution** To avoid aggregating we wanted to give each individual its own visual representation at all times. We chose to represent each individual as a point. All but three visualization tools described in section 3.4 use this design model.

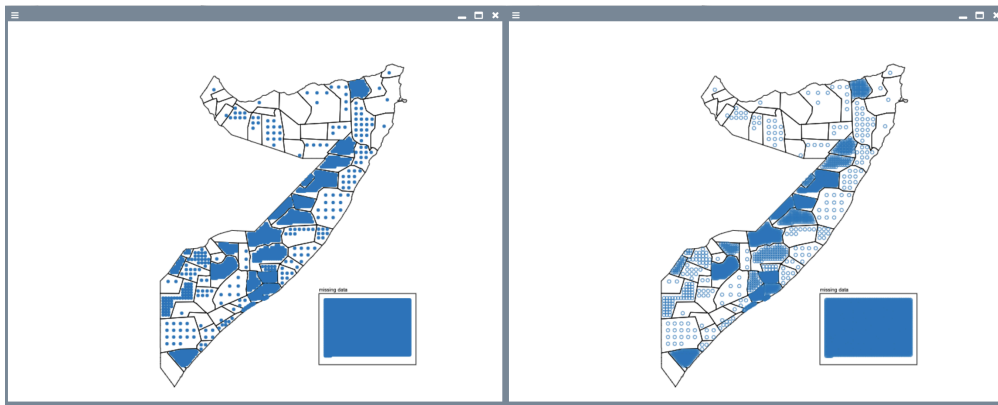
Three aspects are especially important when implementing visualization tools using the individuals as points design concept.

**Random Placement** The placement of each point on the primary canvas and also in grouped displays is random, with one notable exception in the *Map* prototype. Whether we should change the random placement to a structured placement was a recurring point of debate. The idea behind the random placement is to follow the metaphor of a crowd of individuals more closely. Without external instructions people in a crowd will mingle randomly. Recreating this behavior as much as possible in our visualizations is intended to make the idea of an interaction with a crowd more plausible.

Using random placement led to a couple of problems. Firstly, sometimes points overlap each other which defeats the idea of representing each individual equally and also makes some interactions, for example, the Inspect Action, more difficult. Secondly, when using tools like coloring, it gets much harder to see the ratios between different color groups. A special problem arises in the map where a random placement might suggest more precise geolocation information than we actually have. A structured placement implemented, for example, as stacking or alignment on a grid faces some challenges, too. First of all it loses the metaphor of seeing a mingling crowd. As stated in visualization literature the comparison of areas is difficult for humans [14], therefore structured placement might lead to the wrong impression to be able to correctly compare groups even if the differences are small. A random placement only allows for such conclusions when the differences are bigger. In the end we chose to use random placement unless this gave a wrong impression about our data as in the *Map* prototype.

**Circle Form** Due to the random placement the points representing individuals can overlap each other. This is especially true when grouping. If the grouping area is small it gets difficult to see how many individuals are in it. As soon as the whole grouping area is covered there is no visible difference between 100 or 500 individuals. To improve this, we experimented with the opacity of points. A lower opacity however made distinguishing points difficult. In the end, we chose to offer a mode where only the strokes and not the fill of the points are shown. This enables distinguishing single points as well as getting a better sense of density in a group.

**Number of Points** One of the main concerns regarding the individuals as points design concept was how clear visualizations with several thousand points can be. To gain the exploration benefits described in section, 2.4.1.1 users have to be able to distinguish each point to interact with it. As the discussion about circle form shows this is an actual issue of our visualizations. Due to the random placement of the points they can overlap. As the probability of overlapping sinks when increasing the available space, we implemented the possibility of scaling visualizations. In



**Figure 3.2:** Individuals displayed as points with fill versus only strokes

our experience given a fitting scaling, distinguishing and inspecting single points works well for several thousand individuals and acceptable well for up to 20,000 individuals. Especially since Africa's Voices works with data sets with well over 20,000 individuals, compare 1.1.3.3, the question of how to make visualizations based on individuals as points scale to include tens of thousands of individuals, for instance through implementing advanced zooming mechanisms, remains open for future work.

### 3.3 Interaction Patterns

As noted in section 2.4.1.2, one of the key ways for creating explorability in visualizations is to enable users to interact with them through manipulation of the displayed data or the created view. Therefore, we focused on allowing as much interaction as possible. During the development process, the interaction patterns discussed in this section emerged. For some of them the implementation varies from prototype to prototype, most were standardized and used to integrate our prototypes as described in section 3.5. Accordingly, the interaction patterns should not be seen as a part of each single prototype but rather patterns that can be used to enhance the interactivity of an individual-centered visualization.

#### 3.3.1 Inspect

**Goal** Provide quick and simple access to all information about an individual. Represent the qualitative and quantitative data of each individual together.

**Metaphor** Interviewing a single person on the phone.

**Execution** When clicking on a point representing an individual, all information recorded about it is shown as an explorable JavaScript object. This includes

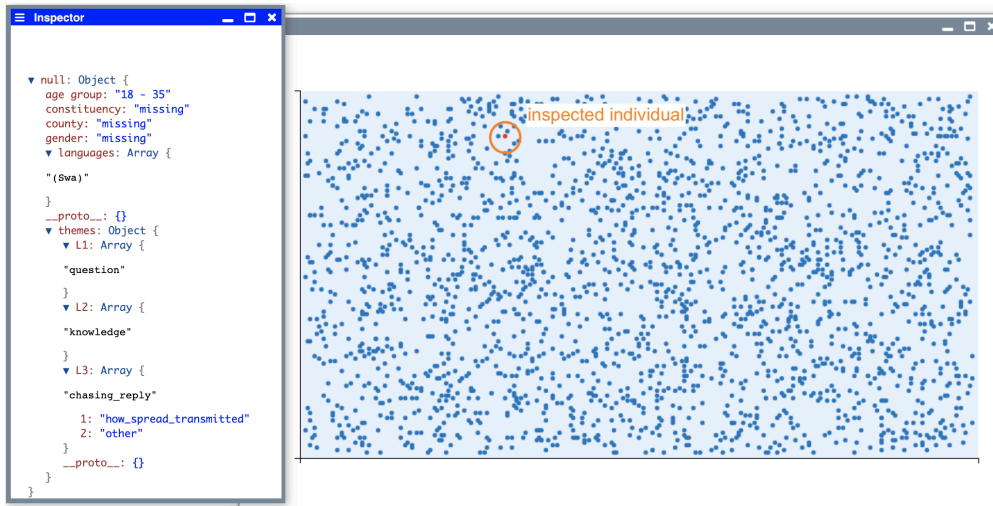


Figure 3.3: Inspecting an individual on click

demographic data, like age, gender, district, and thematic data in form of tags representing the topics of their sent messages. It was intended to also include the sent messages. Although in theory this should be easy to implement, we were not able to work with the messages due to their sensitivity. For further details on the implementation look into chapter 5.

As motivated in section 2.4.2, allowing users to inspect individuals aims at creating empathy. Therefore creating empathy should be a consideration in future work on the inspect action.

**Empathy** We assume that JavaScript objects do not facilitate empathy because they are abstract and require a high-level of imagination to see the individual behind them. We tried to improve on this with the variation *Inspect-Human*.

### 3.3.2 Inspect Variation: Inspect-Human

**Goal** The same as the inspect action. Additionally facilitate empathy with inspected individual.

**Metaphor** Conducting a face-to-face interview with a single person.

**Execution** When users click on a point representing an individual, the sketch of an individual appears on screen. The sketch shows a human of the selected individuals gender and age group. To gain further information about the individual, it can be inspected in the displayed JavaScript object as in the traditional inspect action.

While the idea has potential for a real world usable implementation two aspects have to be taken into consideration.



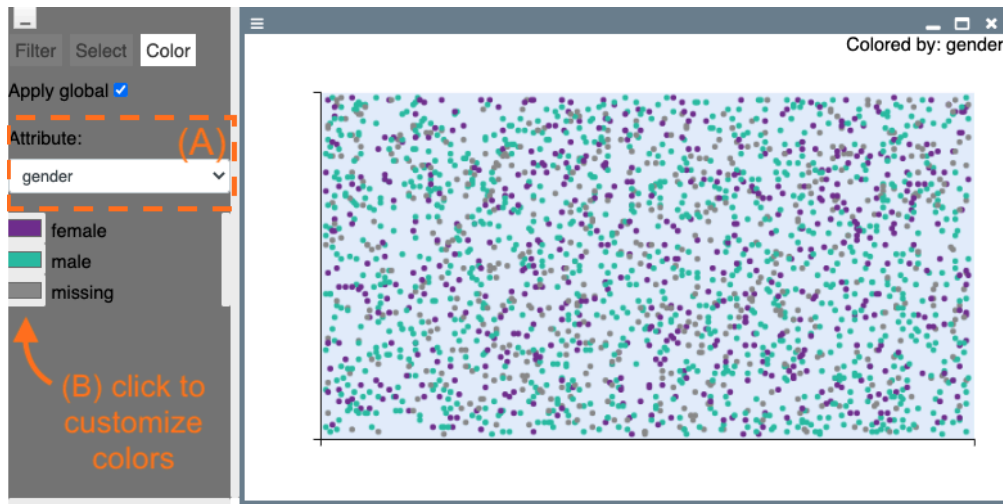


Figure 3.4: Coloring individuals according to gender

**Next Steps** It should be possible to receive all information about the individual from interacting with the displayed sketch. A possible way to do that could be allowing users to ask predefined questions and letting the sketch “answer” through speech bubbles or speech.

**Cultural Issues/Accurate Portrayal** A respectful and accurate implementation of the *Inspect Human* prototype for production is difficult. It would require hundreds of sketches, because each age and gender combination would need many different representations to not give the wrong idea of them all being the same person. Because these sketches would show people from other cultures, we would need a member of each culture to draw and correctly represent them. We realized that this effort would not fit into our scope and did not pursue the idea further.

### 3.3.3 Color

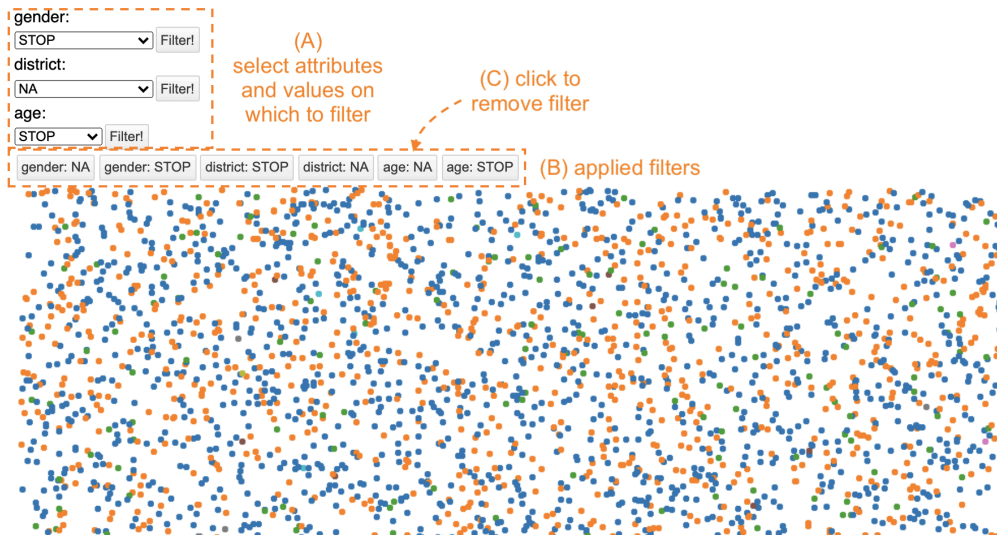
**Goal** Show the distribution of one attribute in the whole group.

**Metaphor** Asking the individuals in the crowd to put on differently colored T-shirts according to their attributes.

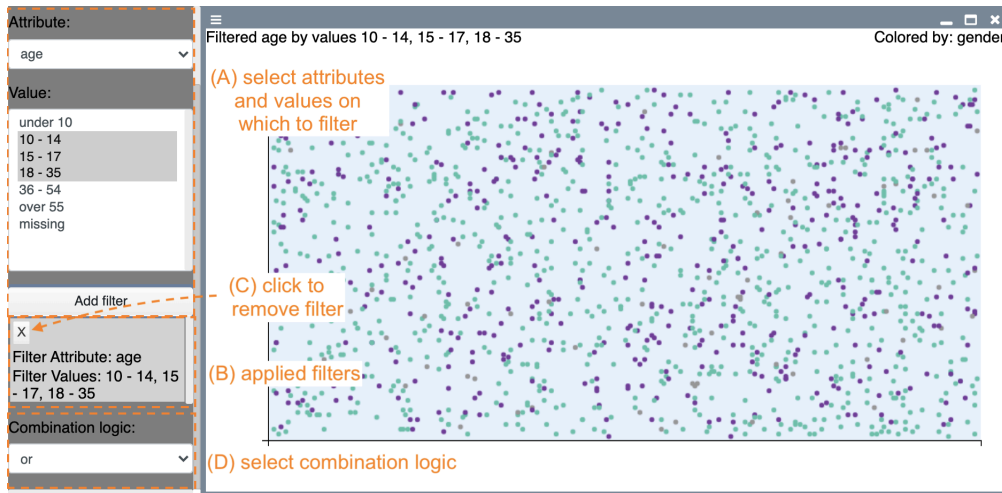
**Execution** Users can select an attribute for which they want to apply coloring (A). Then each unique value existing for that attribute is mapped to a color and each individual is colored accordingly. To ensure sufficient contrast the first colors are applied from a basis configuration file, but to allow users to use colors that are significant to them, each color can be customized (B).

As discussed in section 3.3.1 placing points randomly does not harmonize with the coloring interaction.

### 3 Implementation and Integration Into an Environment of Explorable Visualization Tools



**Figure 3.5:** Using filter-out logic to remove individuals without expected values from display in the *Filter Chain* prototype



**Figure 3.6:** Using filter-in logic in the *Filter Widget* prototype to only work with individuals up to 35 years

**Random placement** Together with the random placement of points the coloring often doesn't accomplish the goal of giving an intuition of the value distribution in the chosen attributes. Only huge disparities are visible that way.

#### 3.3.4 Filter

**Goal** Create explorability by allowing to focus on a specific group of individuals.

**Metaphor** Interviewing a specific population group.

**Execution** We implemented two types of filter interactions. Both share the main idea of removing points from the visualization by selecting values from attributes on which to filter (A). We identified the following dimensions on which filter interactions can differ:

1. *Filter-In vs. Filter-Out*: With “filter-in” logic users select the individuals which should stay in the visualization, with “filter-out” logic users select the individuals which should leave the visualization.
2. *Combination Logic*: With “no-combination” logic the filter tool doesn’t allow chaining multiple filters. With “and-combination” logic the filter tool allows for filtering for individuals on which all filters apply (intersection). With “or-combination” logic the filter tool allows for filtering for individuals for which at least one filter applies (union).
3. *Linear vs. Tree Logic*: With linear logic every filter chain has to have either only “and-combination” or only “or-combination” filters. Tree logic allows for combining subgroups of filters with “and-combination” and “or-combination” logic.

Our first filter prototype *Filter-Chain* 3.5 implements “filter-out” and “and-combination” logic with linear logic. Applied filters are each represented as rectangles in application order in the filter history on the top of the canvas (B). Filters can be removed by clicking on their visual representation (C).

Our second filter prototype *Filter-Widget* 3.6 implements “filter-in”, “and-combination” and “or-combination” logic together with linear logic. The filters are represented in application order in a scrollable window (B) and can be removed by deleting their visual representation (C). The combination logic of the whole filter chain can be selected and updated by selecting it from a drop down (D).

For implementing filter interaction for visualizations based on the individuals as points design model two aspects should be taken into consideration.

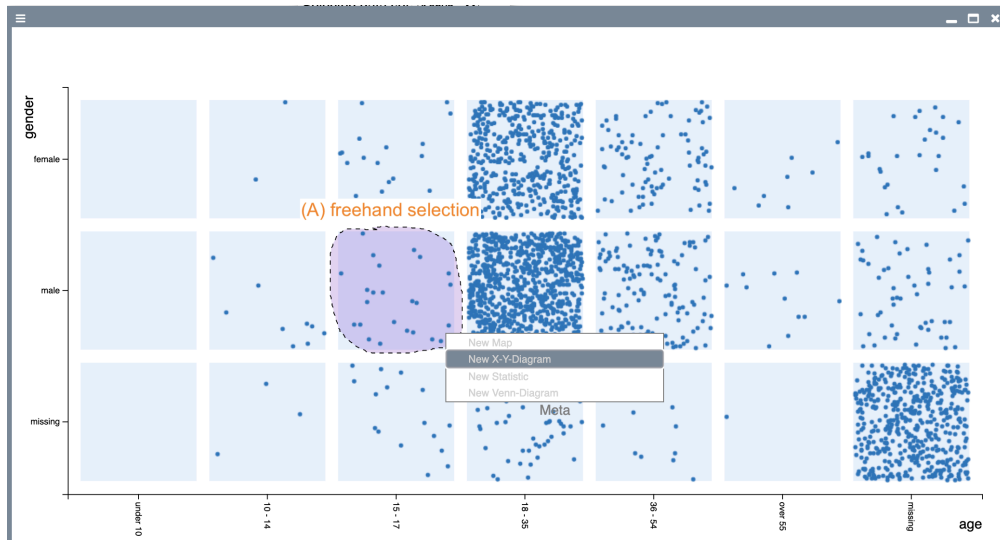
**Filter-In Vs. Filter-Out** For our use case we found that “filter-in” logic is more intuitive. It allows a “what you select is what you get” workflow. A way to support a “what you see is what you get” filtering workflow is provided by our `FreehandSelection` described in section 3.3.5.

**Tree Logic** For our domain, more sophisticated filter chains that combine “and-combination” and “or-combination” filters would be useful. For example, it might be interesting to only work with female respondents of age groups 14-17 or 18-35. Due to its complexity, especially for the user interface, and our limited time, we chose to not implement tree logic.

### 3.3.5 Filter-variation: Freehand Selection Used as Filter

**Goal** Create explorability by allowing to create new views for a specific group of individuals.

**Metaphor** Interviewing a specific population group.



**Figure 3.7:** Using a *Freehand Selection* to filter selected individuals into the next created visualization

**Execution** The *Freehand Selection* allows the user to draw a line on the canvas, whose endpoints get connected at the end of the drawing action. The area inside the line is colored with a random color. With a right click on the selected area (A) the user can open a new visualization which only contains the individuals inside the selected area.

Using the *Freehand Selection* for filtering has a relevant benefit.

**Visual Representation** The *Freehand Selection* allows for an intuitive and visually represented way of filtering individuals in, because it is directly visible which individuals are going to be filtered into the new visualization. Together with grouping tools, this creates a powerful filtering tool which maintains a visual trace of the provenance of used data.

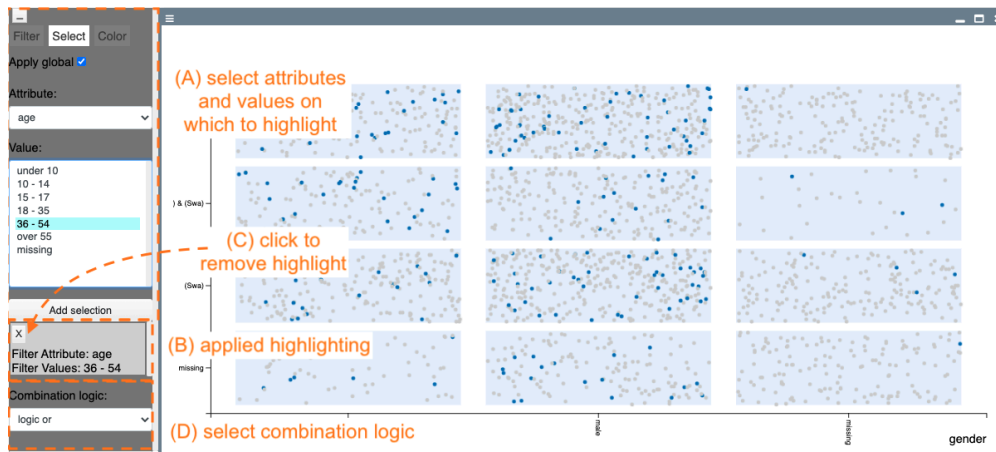
### 3.3.6 Highlight

**Goal** Focus on a specific group without losing view on the whole picture. See the distribution of a specific group in the whole group of individuals.

**Metaphor** Shining a spotlight on each selected individual in a semi dark room.

**Execution** Specifying which individuals to highlight works like filtering by selecting values for certain attributes for which to highlight. During the prototyping process we developed the following two ways of visually representing the highlighted points in chronological order:

1. Color each highlighted point in an attention grabbing color. Maintain previous coloring of not highlighted points.



**Figure 3.8:** Highlighting the distribution of middle-aged respondents in groups defined by language and gender

2. Color each not highlighted point grey. Maintain previous color of highlighted points.

We chose to use the second strategy, because it's easier to distinguish the selected points that way. Compared to the first strategy it aligns better with our goal to focus on a specific group and only keep the other individuals for context, for instance, about distribution.

### 3.3.7 Group

**Goal** Specify groups, visualize their composition, be able to focus on a specific group and to compare groups with each other.

**Metaphor** Asking the crowd to split up into groups according to their opinion on a topic or their demographic data.

**Execution** We implemented various ways of grouping as an interaction pattern as well as part of view tools. All share the same foundation which is a canvas with points. The logical way to display grouping is to rearrange the points on the canvas. The resulting tools can be categorized by how many attributes can be used for grouping.

#### 3.3.7.1 One Attribute

**Map prototype** See subsection 3.4.1

#### 3.3.7.2 Two Attributes

**XY Diagram** See subsection 3.4.2

The following aspect is relevant when applying grouping as in the *XY Diagram* prototype.

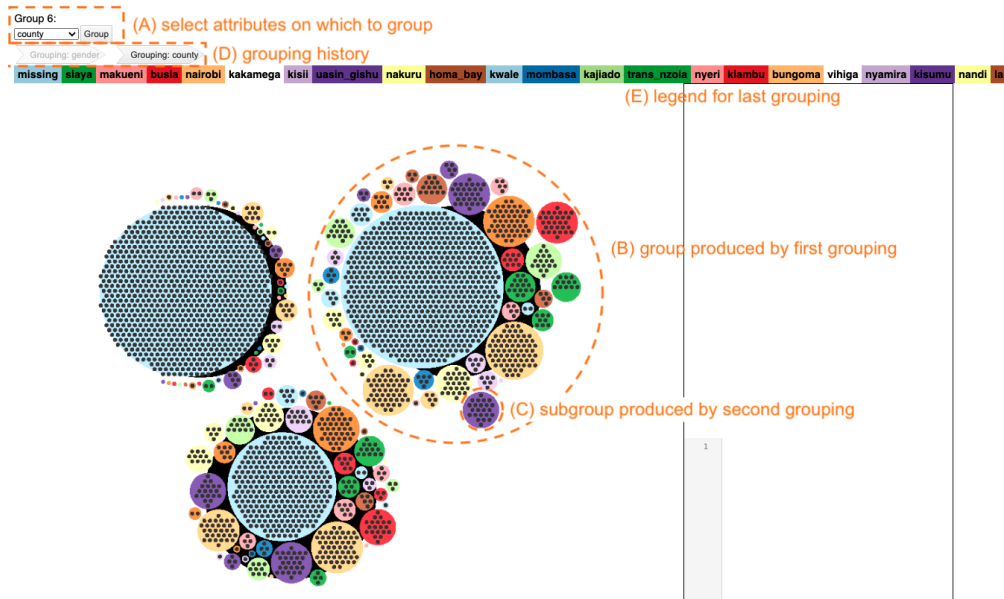


Figure 3.9: Chaining grouping for gender and county

**Discrete vs. Continuous Data** All of our data except age is discrete. Displaying the groups right next to each other therefore can give the wrong impression of a continuous scale. To avoid this we placed gaps between groups. That alone often didn't help enough for being able to distinguish the different groups, so in some tools, for example in the *XY Diagram* prototype, we added shades behind the groups.

### 3.3.7.3 More Attributes Group Chaining

**Execution** As can be seen in Figure 3.9 grouping works over selecting attributes to group for in the dropdown menu (A). The *Group Chaining* prototype automatically splits the existing data into circular groups according to the unique values for the selected attribute (B). If a previous grouping has been applied, the already existing groups are split up into subgroups (C) according to the new attribute. The applied grouping actions are displayed in the grouping history (D) and can be deleted on click.

The following two aspects are relevant when applying grouping as in the *Group Chaining* prototype.

**Distinguish Groups** Using the *Group Chaining* prototype quickly creates a lot of groups. Depending on the grouping attributes you can arrive at 20 groups with only two grouping actions. It gets difficult to distinguish groups from each other and to recognize which group is which. Using a color legend (e) for reference did not work, because with each grouping attribute requiring its own scale, the identification of each group quickly became unclear again.

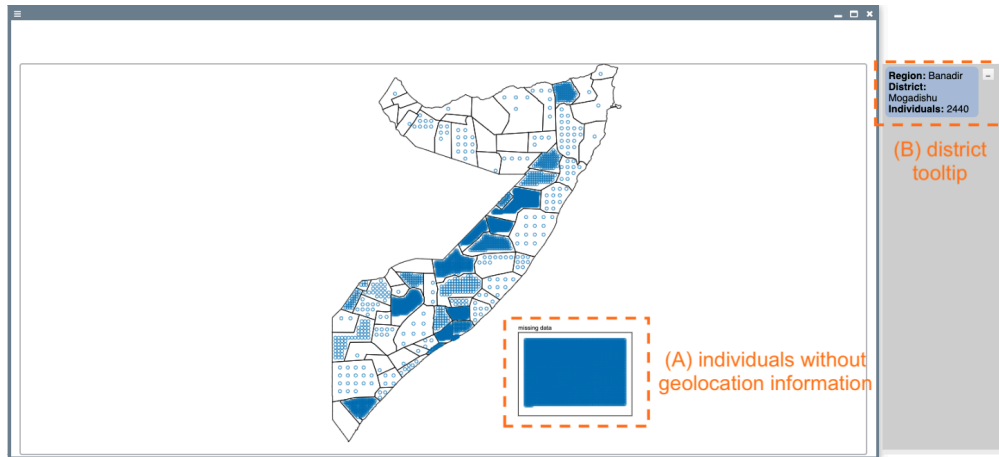


Figure 3.10: Hovering over the Mogadishu district while using the *Map* prototype

**Distinguish Individuals** On the other hand, this prototype demonstrates how easy it is to distinguish and trace individuals with only three or four known attributes. As can be seen in Figure 3.9 after grouping two times many groups only have a handful or just one individual.

**Venn Diagram** See section 3.4.3

**Individual Center** See section 3.4.4

## 3.4 Visualizations Tools

Based on our ideas which are described and categorized in section 2.3.3 we implemented ten visualization tools. Visualization tools allow for a particular view on the data, often focusing on one goal, for example, showing the theme distribution. Opposed to interaction tools, their functionality is too specific to be integrated into another prototype. They rather allow for different perspectives on the same group of people.

### 3.4.1 Map

**Goal** Show the distribution of respondents based on geographic position, allow interactions based on a place and the comparison of distributions, for example, between rural and urban areas.

**Metaphor** Asking the crowd to split up into groups according to their districts.

**Execution** Showing geolocation distribution is one of the common use cases for visualizing individuals as points. In this case, the data provided only gives higher



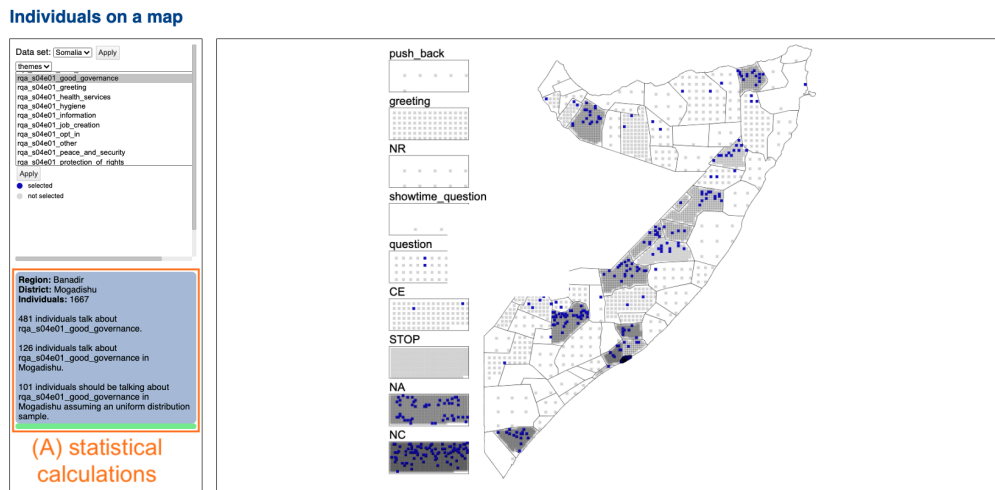


Figure 3.11: Statistical calculations about the distribution of people talking about good governance in Mogadishu

level position information like county or district. In the prototype every individual gets placed inside its district. Individuals without district information are placed inside an “information missing” district at the side (A). When hovering over a district a tool tip provides information about it (B). Zooming and panning allow a closer look.

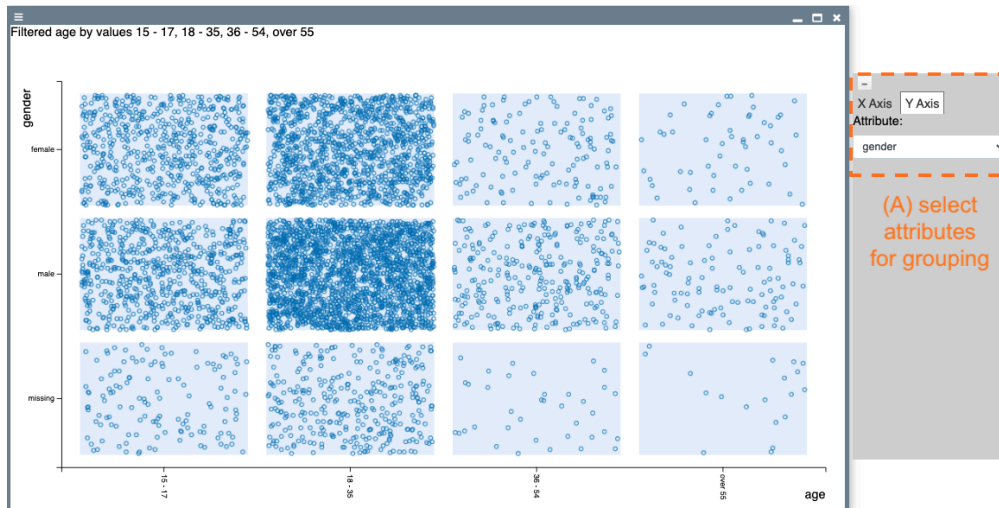
Using the individuals as points design model on a map lead to a number of considerations about rueful representation.

**Random Placement** As noted in section 3.2, the *Map* prototype is the exception to the rule regarding random placement. Here random placement is not helpful but misleading, because it suggests more precise geolocation information than we actually have. Therefore we placed the individuals on a raster inside each district.

**Density** So far every data set has shown a huge disparity in the number of respondents per district. Often especially smaller districts, like the capital, are overrepresented and single individuals are not distinguishable in the overview. To improve this we implemented zooming. This way a heat map on respondents density emerges as can be seen in 3.10.

**Statistical Calculations** As shown in Figure 3.11 in earlier stages of the prototype we included calculations (A) regarding expected numbers of individuals with a theme tag assuming an uniform distribution sample. Displays like this can help to immediately validate impressions, like “an uncommonly huge number of people talk about this topic in this district”, and might be a starting point for integrating classical statistical tools with our visualization tools.





**Figure 3.12:** Using the *XY Diagram* to display groups by gender and age

### 3.4.2 XY Diagram

**Goal** Find patterns and specific groups to explore further. Present groups in well structured formats and enable getting a sense for them.

**Metaphor** Asking the crowd to split up into groups according to one or two selected attributes.

**Execution** The *XY Diagram* uses a coordinate system in which all individuals are placed. The values along the x and y axes can be grouped by attributes (A). For that, each individual gets a new x or y position according to the corridor of its value on the axis. As an additional attribute 'amount' is added. It can only be used when the other axis has already been grouped by another attribute. It then places the individuals of a group in a corridor which has n percent of the maximum length, where n is the ratio of the amount of this group to the amount of the biggest group.

The *XY Diagram* provides two useful features for an exploration flow.

**Bar Chart** Giving the ability to group by amount leads to the creation of bar charts while keeping a connection to each individual. Even in the bar chart form, each individual can be inspected.

**Exploration Workflow** In testing our prototypes we noticed that the *XY Diagram* works especially well as a basis for using *Freehand Selections*. With the *XY Diagram* multiple groups can be specified, which, using the *Freehand Selection* workflow, can then be further compared in other visualizations.

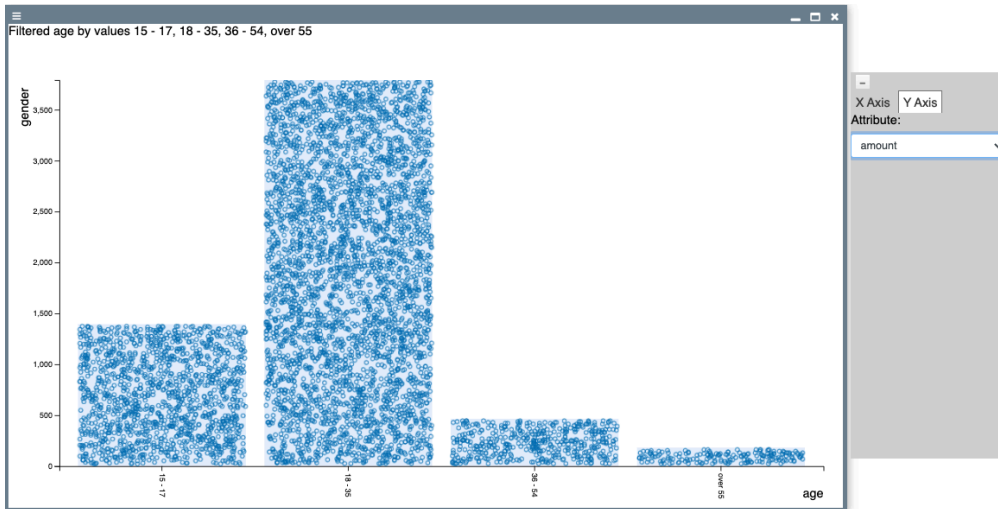


Figure 3.13: Using the *XY Diagram* to create a bar chart which maintains a connection to each individual

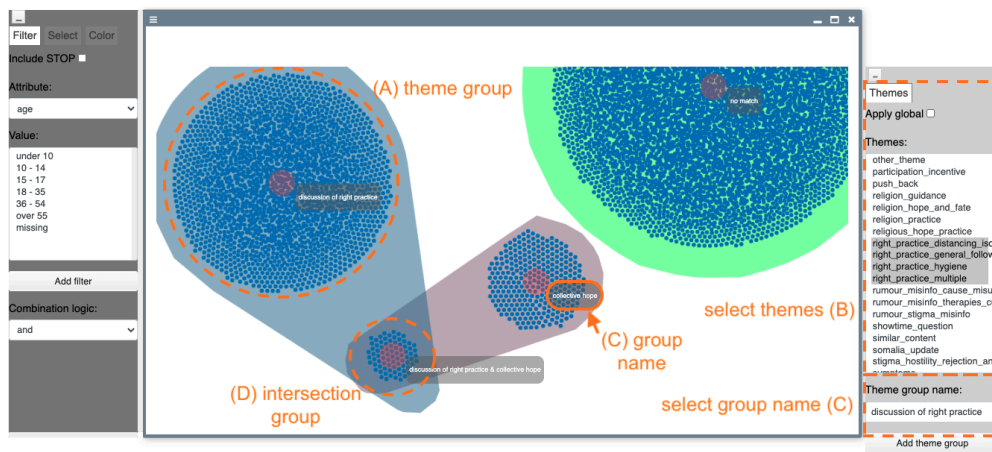


Figure 3.14: Using the *Venn Diagram* prototype to explore connections between two theme groups

### 3.4.3 Venn Diagram

**Goal** Group individuals according to thematic data. While thematic data is arguably the most valuable data, it is also the most complicated due to its high-dimensionality. An additional goal of this diagram was to place a special focus on overlays between different theme groups and handle their high-dimensional connections.

**Metaphor** Asking people to form discussion groups according to topics they want to talk about.

**Execution** The *Venn Diagram* allows users to create theme groups (A) with as many themes as wanted by selecting them from the menu (B). Groups can be assigned a name (C). Every individual whose message was tagged with at least one of the themes will be displayed in the created theme group. If a second group is created, individuals belonging to both groups will be displayed in an extra group (D), displaying the intersection of both primary groups. Adding a third group adds every possible combination group for which fitting individuals exist.

To improve the usability of the *Venn Diagram* two aspects should be considered.

**Complexity** The *Venn Diagram* starts to be confusingly complex after grouping more than two times with many different groups to display, which are all interconnected and difficult to separate. We tried to improve the arrangement with an automatic placement algorithm to make the placement more clear from the beginning on.

**Integration of Demographics** The ability to get an overview about the demographic data in the first version of the *Venn Diagram* was only given by the coloring action. To provide additional insights we integrated the *Statistics Panel* described in section 3.4.8 into the workflow. On click on the center of a theme group a *Statistics Panel* about the individuals in the theme group opens up.

### 3.4.4 Individual Center

**Goal** Be able to see how an individual fits into the crowd. That is, if there are many "alike" individuals or if this individual is an outlier. Find similar or very different individuals compared to one individual.

**Metaphor** Asking people to position themselves as close or far away as they feel to a specified person on a demographic or thematic level.

**Execution** The *Individual Center* prototype lets users select an individual and one of two compare methods, thematic or demographic. As can be seen in Figure 3.15 the selected individual gets placed in the middle of the screen (A) and every other

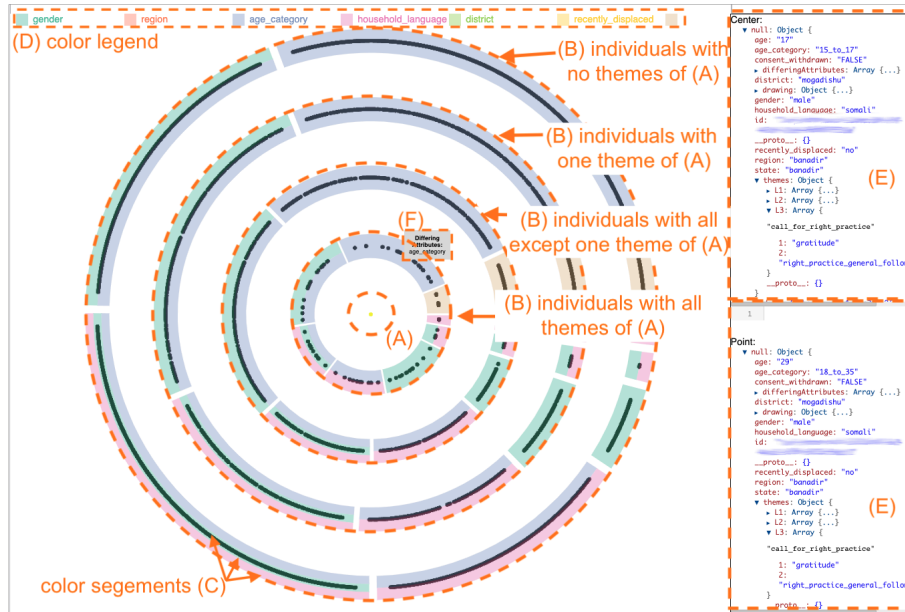


Figure 3.15: Using the *Individual Center* prototype with thematic comparison to compare and explore two individuals with equal theme tags on gender, age category and household language

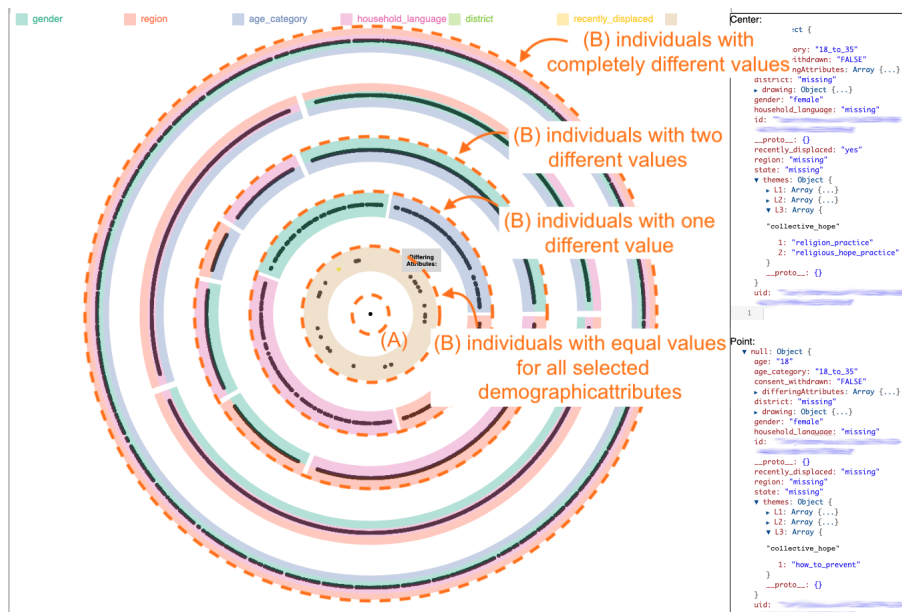


Figure 3.16: Using the *Individual Center* prototype with demographic comparison to explore two individuals with equal values for gender, household language, region and age category

individual gets a position on concentric circles around it. Their position symbolizes the closeness to the selected individual (B).

*Thematic:* Rings signify how many themes the other individuals have in common with the selected individual. We only compare by the themes of the selected individual. Therefore individuals which have all themes of the selected individual are considered equal to the selected individual, even if they have additional themes.

*Demographic:* Rings signify in how many demographic attributes the other individuals differ from the selected individual.

For both modes we use color segments on the ring to display demographic differences. Each color segment stands for one specific combination of demographic differences, for example, only age, or gender and district (C). Colors can be mapped via a color legend (D). Using the two individual inspectors on the side the selected center individual can be compared to other individuals (E).

For projecting high-dimensional data to 2D concentric rings three aspects need special consideration.

**Usage of Color for Displaying Differing Demographic Attributes** Encoding differing demographic attributes by color proved to be difficult. To display multiple differing attributes on arcs in a first attempt we tried mixing the colors assigned to each attribute from the attribute legend (A). In our experience, the resulting colors were not decodable for users. Therefore we decided to display the attribute colors next to each other as substripes (C).

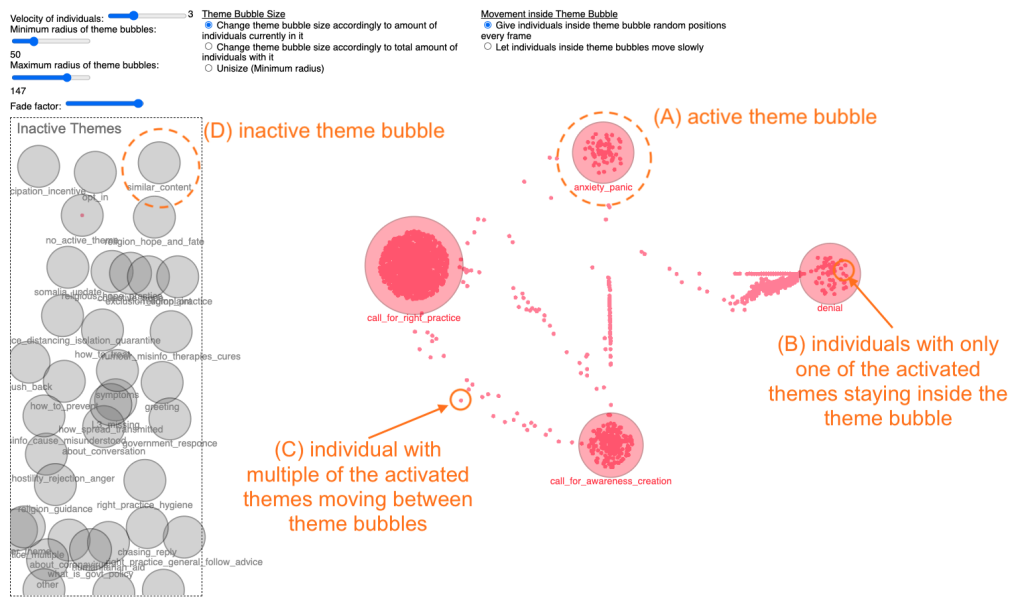
**High Complexity of Encoded Information** Substripes as a way of conveying the differing demographic attributes, in our experience, are still difficult to work with. Users have to match up to five substripes with the colors from the legend (D) displayed at the top of the visualization. To help with explorability we introduced a tooltip (F) which on hover displays the information about each arc. We maintained the coloring in stripes to showcase the heterogeneity of the groups.

**Correlating Data** Some of our attributes were inherently correlated, especially the geolocation attributes. If two individuals responded with same district as their place of residence, they will with a high probability also respond with the same region. This leads to sometimes near to empty or unnecessary many rings and could lead to wrong impressions about the data. To counteract the effect we gave users the ability to select the attributes used for comparison, effectively allowing them to create an own similarity metric.

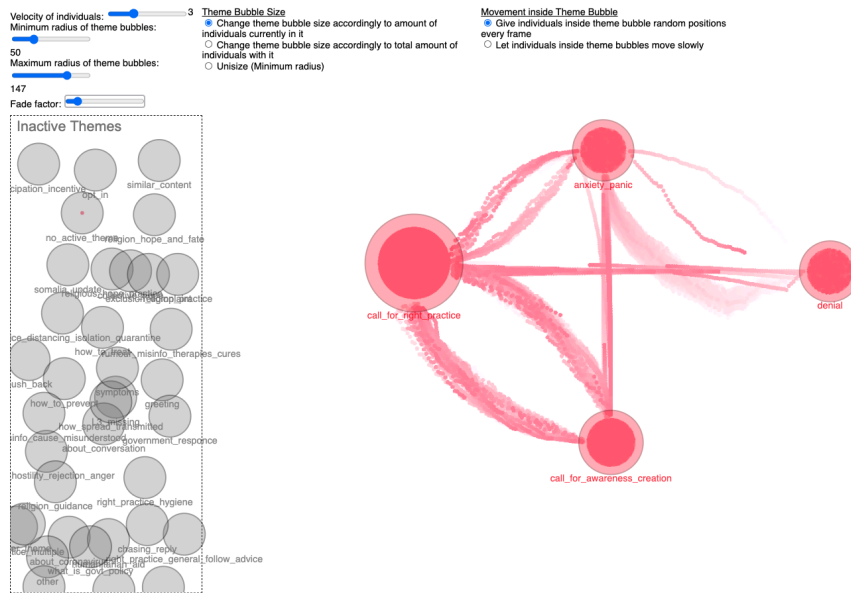
### 3.4.5 Movement Prototype

**Goal** Visualize the opinion of a crowd. Highlight connections and differences between different stances and allow the exploration of individuals with specific theme combinations.

### 3 Implementation and Integration Into an Environment of Explorable Visualization Tools



**Figure 3.17:** Using the *Movement* prototype with a very high fade factor



**Figure 3.18:** Using the *Movement* prototype with a low fade factor

**Metaphor** Letting the crowd arrange itself on its own. People who know each other or share interests will move close to each other.

**Execution** As can be seen in Figure 3.17 each theme is represented by a bubble (A). Individuals are represented as points and can move between bubbles (B), when they have multiple themes, or stay inside them, if they only have this theme (C). Themes can be arranged on the screen through dragging and can be activated and taken into account for the movement of the individuals or deactivated and taken out of the visualization (A) (D). In this case individuals will not move to them even if they have the theme belonging to the bubble. If individuals possess more than two of the activated themes they move circularly between the corresponding theme bubbles.

For future implementation work on the *Movement* idea three aspects should be taken into consideration.

**Clarity** The main issue of this prototype is the missing clarity of the visualization. This results from the following factors:

*Movement:* If all themes are activated at the same time there is too much movement on the screen to distinguish interesting patterns. We improved on this by allowing users to activate and inactivate the theme bubbles via double click or dragging the themes into the active/inactive zones.

*Paths of individuals:* Without visual help it is difficult to trace and keep in mind the path taken by individuals. Therefore we let the individuals paint on the screen while they were moving, effectively creating paths.

*Adjustability:* In later iterations we made it possible to adjust the velocity of the moving individuals and the fade factor of the paths they draw. This enables the user to pick and select certain individuals or just see their created paths. In this case our diagram basically becomes a sankey diagram in which you can move around the nodes as you wish. For this to work completely a more sophisticated movement calculation would be needed.

**Liveliness** The *Movement* prototype implements the concept liveliness described in section 2.4.2.2, which aims to help keep the image of the points representing individuals in mind.

**Wrong Impressions** There are multiple representation issues in the movement prototype:

*A. Individuals with many themes are underrepresented.* Individuals move circularly between their themes. If two themes are connected by an individual which in total has three themes, and two other themes are connected by an individual that only has these two themes, the connection between the second themes will be shown thrice as often as the connection between the first themes. It is important to keep in mind that the *Movement* prototype faithfully displays every individual and thus can lead to wrong impressions about the total number of connections between themes.

*B. Amount of individuals within one bubble* As described in 3.2 due to overlapping points it can be difficult to see how many individuals are in an area. This applies for

seeing the amount of individuals in a bubble, too. To improve this we implemented optional modes where the size of each bubble changes according to the amount of individuals in it or the total amount of individuals with this theme.

### 3.4.6 Individual Forces/t-SNE

**Goal** Use the first screen to convey meaningful information as opposed to the status quo random placement. Allow for orientation and easier exploration by arranging similar individuals close to each other. Be able to inspect clusters. Be able to create own similarity metric.

**Metaphor** In a crowd, in a simplified version, similar people attract each other. Peer and shared interest groups would stand closer to each other.

**Execution** The problem in placing individuals close to individuals they are similar with is the high-dimensionality of our data. For each individual we routinely can have more than 20 data points. Projecting this to a 2D space proved to be difficult. The *Individual Center* prototype solves this issue by using a very narrowed and predefined compare method (number of differing attributes) which also results in predefined places for each individual. To still maintain the open canvas/open field view as intended with 2.1 Individuals in Points we tried out two approaches *Individual Forces* and *t-SNE*.

#### 3.4.6.1 Individual Forces

**Execution** The *Individual Forces* prototype uses a simulation in which each individual attracts similar individuals and repels not similar individuals. The simulation stops when the individual forces have found equilibrium and leaves the individuals arranged on the screen. We used the library *d3-forces*.

Projecting our high-dimensional data to a 2D space using a forces approach faced a critical challenge.

**No conclusive picture** The way that we used individual forces too many forces were counteracting each other which stopped cluster building from happening and lead to an inconclusive arrangement as can be seen in Figure 3.19. As nothing valuable could be read from it, we decided to drop this approach.

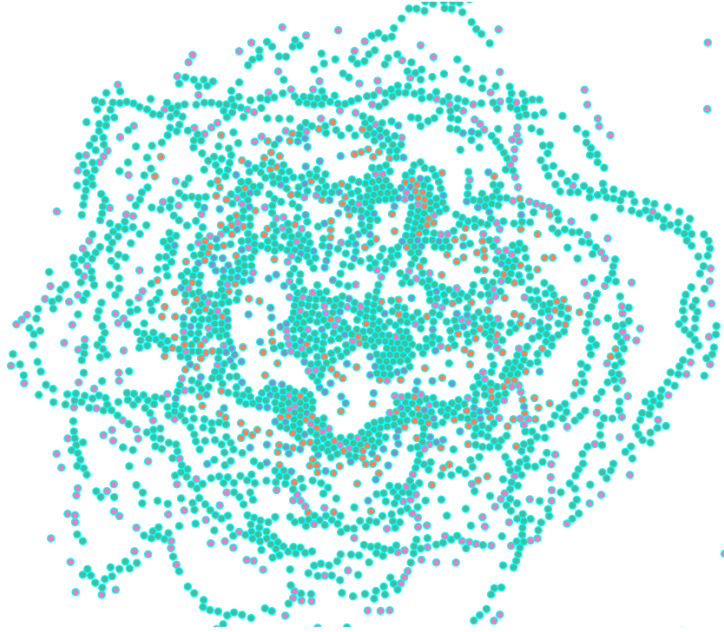
#### 3.4.6.2 t-SNE

**Execution** t-SNE (t-distributed stochastic neighbor embedding) is a machine learning algorithm used for the visualization of high dimensional data. It embeds high-dimensional data into a low-dimensional space of two or three dimensions [49].

Like the *Individual Forces* approach, using t-SNE for projecting our high-dimensional data to a 2D space faced a critical challenge.

**Performance** Running the t-SNE algorithm with thousands of individuals which each have up to 20 belonging data points is computationally expensive. Our





**Figure 3.19:** Using the *Individual Forces* prototype. Forces are calculated according to the theme categories each individual belongs in (question, answer, escalate, and missing). Points are colored according to their theme category.



**Figure 3.20:** Using the t-SNE algorithm based on demographic data. Points are colored according to county.

prototypes did not reach results in an acceptable time. Loading times ranged from 1 min to 3 mins. Critical for the performance of t-SNE is the calculation of a  $n \times n$  probability distribution matrix with  $n$  as the number of entities to compare. The matrix assigns each pair a probability with highly similar pairs having a high probability and dissimilar pairs having a low probability. Calculating the matrix client-side in the browser takes too long for an application with direct user interaction. Our next approach was to calculate the matrices server-side. The idea was to precalculate and cache matrices of often used similarity metrics. Additionally we wanted to cache a similarity matrix for each attribute, so that we could calculate custom similarity matrices by matrix addition and dividing by the amount of used attributes. This approach did not speed up the loading time as much as we wanted. The matrices become quite large and the time for requesting and transferring the matrix data nearly took as long as calculating it client-side.

Both our approaches failed to deliver the result we wanted.

#### 3.4.7 Polygon Theme Forces

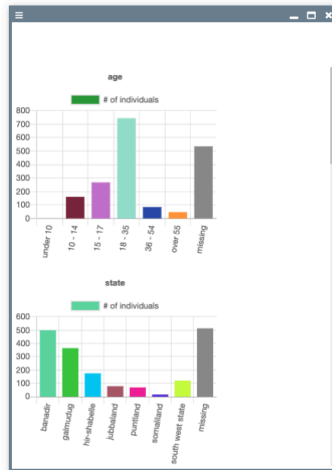
![3-PolygonThemeForces](../../figures/topic3/Screenshots/2.3.7-PolygonThemeForces-Screenshot-2020-07-27-at-19.44.28.png "Using the *Polygon Theme Forces* prototype to inspect a theme that has mostly been mentioned by male respondents or respondents who did not tell their gender.")

**Goal** See at one glance which themes get mentioned more by which demographic groups.

**Execution** Each theme is represented as a bubble (A) with varying size depending on how often it got mentioned. The unique values of the selected demographic attribute (B) are arranged as labels around the theme bubbles spanning up a field (C). On the field the theme bubbles find their position by being pulled to each demographic group with strength according to how many individuals from that group have chosen the theme. The labels representing each demographic group can be dragged to form different areas in which the theme bubbles align. Users can choose between using absolute or normalized amounts of people choosing the themes for calculating the forces (D).

For future development of the *Polygon Theme Forces* prototype three aspects should be considered.

**Normalization** In the first iteration our prototype used the total amount of individuals from each demographic group to compare and calculate forces for each theme. But not every demographic group has the same amount of individuals, which makes this way of comparing unfair and misleading. A better way of comparing is using the ratio of individuals from each demographic group that have chosen the theme.



**Figure 3.21:** Using the *Statistics Panel* prototype for displaying predefined statistics for a selected group of individuals

**Clarity** With more than 6 or 7 demographic groups, for example when using geolocation data, it gets difficult to gain insights from the visualization.

**Next Steps** Next steps would be to make the data used for the inner bubbles and the outer labels interchangeable and selectable. This would, for example, allow to have themes plotted on the ring outside and the individuals aggregated according to languages inside.

### 3.4.8 Statistics Panel

**Goal** Combining our qualitative approach with standard statistical approaches. Generate connection point to other visualization systems.

**Execution** Allow for generating classic diagrams at every point in exploration. From a specified group, for example by using the *Freehand Selection* described in section 3.3.5, our tools allow to generate predefined diagrams. Currently the age, gender, language, county and constituency distribution of the individuals in the selected group are plotted in a bar chart.

For future work integrating standard statistical approaches into our prototypes one aspect is of special importance.

**Customizability** While the workflow itself fits well into our prototypes the usability is still limited due to missing customizability. The next step for the *Statistics Panel* prototype would be to allow for user customized diagrams.

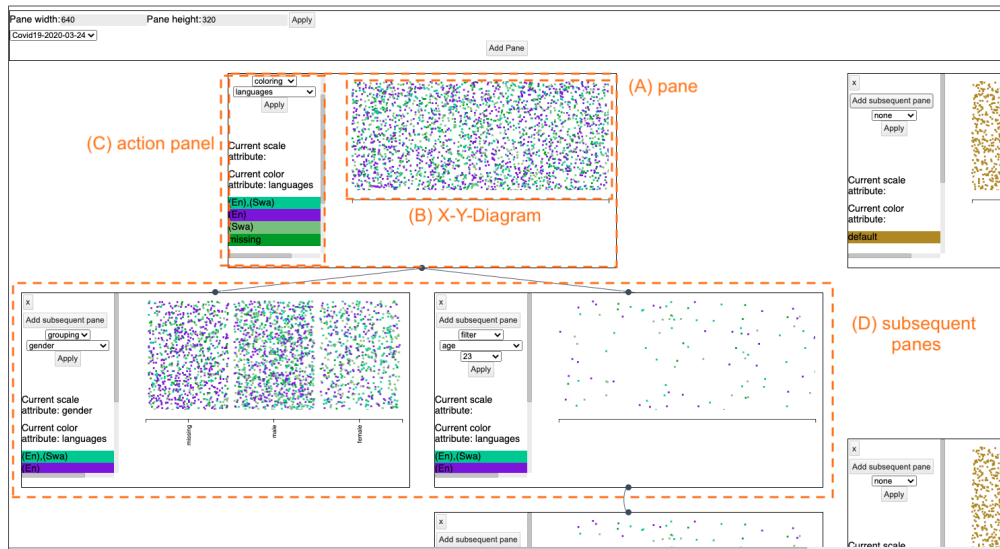


Figure 3.22: Using the *Panes* prototype to explore the data using grouping, colouring and filtering

### 3.4.9 Panes

**Goal** Make the exploration flow visible. Allow users to easily go back to earlier steps and diverge on a different path.

**Execution** As shown in Figure 3.22 in the *Panes* prototype each pane (A) consists of an *XY Diagram* (B) and an action panel (C), which allows users to filter, group or color according to available attributes from the used data set. Users can add subsequent panes (D), which get automatically placed in the row below the parent pane, effectively creating a tree form. Actions applied in a pane get applied to all its children panes. Per pane only one action can be applied. Applying a new action overwrites the previous action.

We took two important considerations from the *Panes* prototype which supported the development of the *Tree View* visualization environment prototype described in section 3.5.3.

**Further Potential** Visualizing the exploration flow and enabling users to easily create new exploration paths at any point in it, creates a new experience of exploration. While with only one view on the data the use is limited, the *Panes* prototype inspired our later developed *Tree View* visualization environment.

**Placement** The underlying placement algorithm uses nested flex CSS rows and column. The resulting placement is structured and looks organized. On the other hand, it does take the freedom of arranging the panes according to the personal exploration flow from the users. Therefore in the *Tree View* prototype placement of panes was transferred to the users by making panes draggable.

### 3.5 Integration of Tools

In this section we describe and evaluate our approach to integrating the visualization tools described in the previous section.

During our ideation process we found that enabling users to view different perspectives of the data and compare visualizations are two essential concepts for making data explorable. See section 2.4.1.2 and section 2.4.1.3.

To enable creating different views on the same data and comparing them, without forcing users to manually recreate the state from one visualization tool in another visualization tool, we decided to integrate our tools.

Tool integration is a known problem in software engineering research [4]. To analyze the level of integration between programming tools several classification schemes have been proposed [18, 17, 67].

We argue that, because of the structural similarity between programming tools and our visualization tools, these classification schemes are also applicable for analyzing our integration process.

Programming tools can be defined as “any system that assists the programmer with some aspect of programming” [58]. Classical examples include editors, linters, debuggers, and compilers, all of which fulfill a specific task on the same artifact the source code. Comparatively, our visualization tools assist the user with one aspect of exploring the data, namely, providing one specific view. Like programming tools, our visualization tools work on the same artifact, in this case, the data for the visualizations.

In conclusion, we think it fair to argue that integrating our visualization tools can be evaluated with instruments provided by programming tool integration research.

To reason about tool integration, three things are needed [3]: “a set of categories that divide an integration effort into parts that can be discussed in isolation”, “a way to measure the degree of integration”, and “a notion of what optimal integration is”.

As a literature survey on tool integration [4] analyses the classification scheme provided by Wasserman in 1990 [67] is widely accepted. As its provided integration dimensions work well to reason about our process, we chose to use this classification scheme.

Wasserman [67] defines five types of tool integration which can serve as categories for discussion about tool integration. For three of them he defines a scale of used mechanisms to measure the strength of the integration:

1. **Platform integration** classifies the interoperability of tools.
2. **Presentation integration** classifies the level of agreement of tools on a user interface standard. The possible integration levels from weak to strong are: Standard window system, Standard windows manager, Standard toolkit, Standard “look and feel”.
3. **Data integration** classifies the level of data sharing. The possible integration levels from weak to strong are: Message, Shared Files, Database, Objectbase.
4. **Control integration** classifies the level of structure used for inter-tool notification of events. The possible integration levels from weak to strong are: Explicit message, Daemon, Trigger, Message server.

5. **Process integration** classifies the integration of tools into a well-defined software engineering process.

Functioning platform integration is a necessary criterion for tool integration. In our case, it is fulfilled because all tools share Lively4 as their platform. As we had no opportunity to work with Africa's Voices to integrate our tools into their process, we will only use the remaining three categories presentation, data and control integration for analyzing our integration approaches.

For optimal integration, according to Wasserman, integrated tools should agree on an appropriate level of integration on all dimensions [67]. As Wasserman gives no definition for an appropriate level, we chose to define appropriateness as the ease of integrating tools. An approach to tool integration is appropriate if the implementation efforts needed for the integration structure and the implementation efforts needed to adapt the visualization tools together are low. Appropriateness additionally can be influenced by external requirements, for instance, regarding consistent user interfaces.

### 3.5.1 Initial Integration Level of Visualization Tools

Environments can be defined as integrated collections of tools [53]. The visualization tools in section 3.4 already possess a certain level of integration. All our prototypes were developed in Lively4 and therefore are integrated on the platform integration level. They all use the same data and data format, as described in subsection 5.2.5. Although the implementation varied from prototype to prototype, they often implemented the same interaction patterns section 3.3. Additionally, most prototypes share the individuals as points design model. While we started from already similar prototypes, the initial integration level did not suffice to reach our goal of an easier and faster exploration flow. Therefore we developed dedicated integration approaches.

Both integration approaches use our visualization tools implemented as web components. As Lively4 is built with web components itself and facilitates the development with them, this let us benefit from our platform integration. The process is described in section 4.3.2.2.

### 3.5.2 Tab View

**Goal** Give users a simple way to switch between views of data during the exploration flow.

**Execution** As shown annotated in Figure 3.23 in the center of the *Tab View* prototype is the currently selected view (A). Out of the visualizations described in section 3.4 the *Map*, *XY Diagram* and *Venn Diagram* are integrated into the *Tab View* prototype. On the left of the visualization, the panel for local controls allows for customized interaction with each visualization (B). The core feature of the *Tab View* prototype is the ability to switch between the integrated views by just one click on a tab (C). When switching views, the state of the visualization, in particular applied

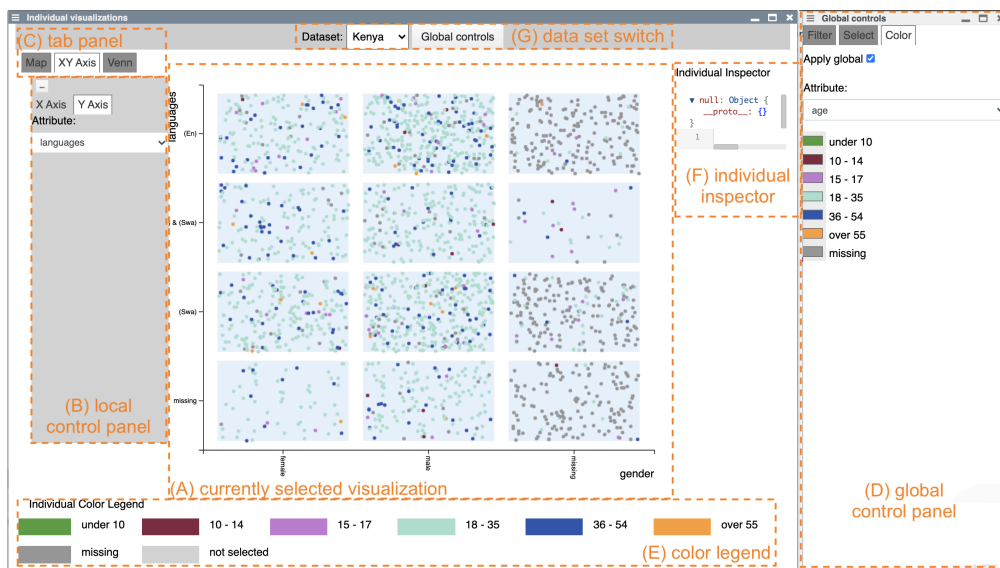


Figure 3.23: Using the XY Diagram in the Tab View prototype with coloring according to age

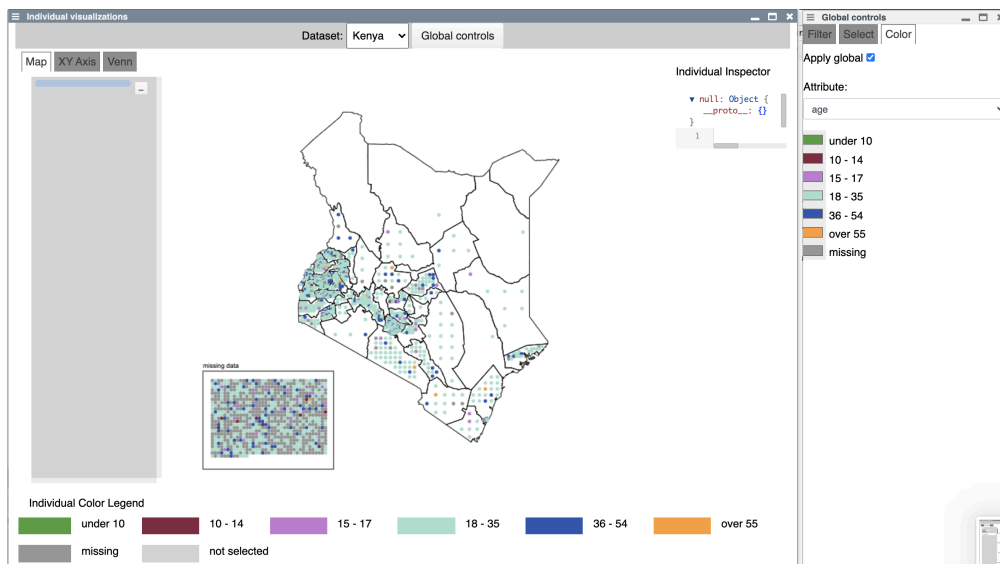


Figure 3.24: Using the Map prototype in the Tab View prototype with coloring according to age

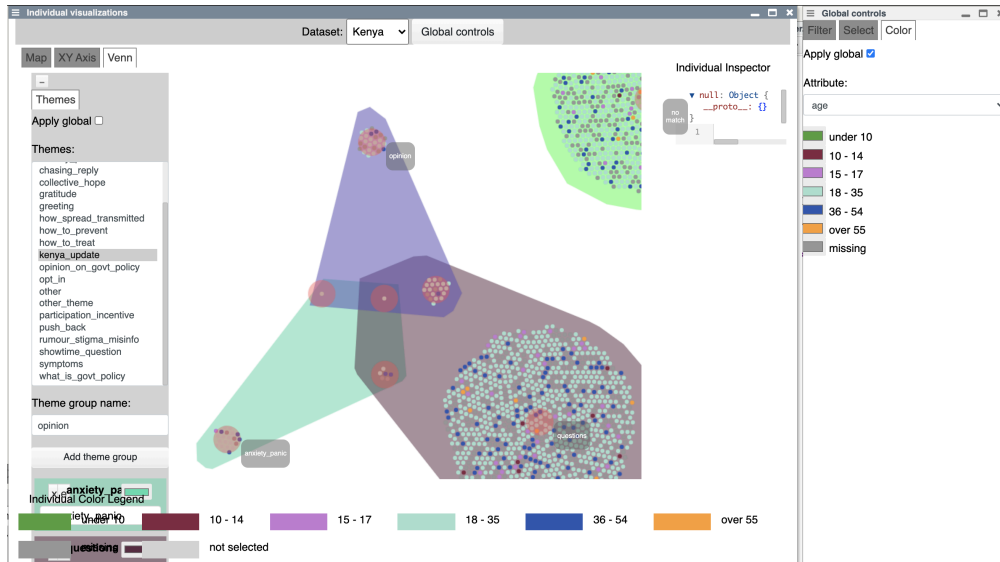


Figure 3.25: Using the Venn Diagram in the Tab View prototype with coloring according to age

filters and coloring, stays the same. This enables a seamless exploration flow. Global actions like filtering can be applied in the Global Control Panel (D), automatically updating the Color Legend (E) on the bottom side of the visualization. At all times in the exploration, users can inspect individuals. The currently inspected individual is displayed in the Individual Inspector (F) and also persistent through tab switching. This enables the user to trace individuals through different views. The *Tab View* prototype works with multiple data sets that can be switched on the top side of the visualization (G).

We have two considerations regarding the functionality of the *Tab View* prototype.

**Global and Local Actions** During the integration process, we had to decide which interactions should be standardized and integrated. The resulting global actions are: inspect, color, filter, and highlight. All of them use an information dimension for displaying their information that the visualization tools do not necessarily need to function. The group action, on the other hand, uses position as its dimension. As can be seen in subsection 3.3.7 many visualization tools have their own variation of grouping, many depending on position to convey their central information. To ensure consistent behavior, we decided to not integrate the group action.

**Persistence** To further support the exploration process, being able to save exploration states would be helpful. As reloading currently leads to discarding previous progress this remains a point for future work. As described in section 4.3.3.2 Lively4 provides mechanisms for persisting the state of web components.



### 3.5.2.1 Technical Implementation

The *Tab View* prototype uses the observer pattern to inform all integrated components about changes to the representation of its shared data.

As described in subsection 5.2.5 all visualization tools work with the same individual-centered data format. At the initialization of the *Tab View* prototype, all integrated visualization tools get initialized with the same data. Afterward, they hold and manage their data separately.

The only way for users to change the data, in this case meaning changing the appearance of the points, is through a predefined set of interactions, namely, the inspect, filter, highlight, color, and group actions. As discussed above, only inspect, filter, highlight, and color actions are applied globally. While inspect actions from the users are registered and handled at visualization tool level, filter, select, and color actions get applied through the Global Control Panel. In both cases, the notification on the change works over action objects encapsulating the applied change that get send to the Individuals Visualization Widget. The Individuals Visualization Widget then notifies all its registered listeners about the applied action. To register as a listener at the Individuals Visualization Widget components need to implement the `applyAction` interface. They can then decide themselves if and how they implement each action.

This approach enables straightforward extensibility for further interactions and gives the visualizations sovereignty about their internal procedures. Through saving the applied action objects, a history is created which allows undoing an action, for example, discarding a filter. To synchronize the data representation and used data structure, all components use the shared `ColorStore` and `DataProcessor` objects, which provide default colors for the visualized points and a bucketing structure for some data attributes. No requirements regarding component structure are made. To make building a component easier, a template used by all currently integrated visualization tools exists.

### 3.5.2.2 Integration

We use the categories and measures for the integration of tools presented in section 3.5. Since we adapted all our tools in the same way for integrating them, they can all be categorized together.

**Data Integration** The initialization with the same data object and the shared objects used for standardization implies a strong data integration with an Objectbase in the beginning. After the initialization, data integration is handled via messages informing the other components about applied actions, and can thus be categorized as weak.

**Control Integration** The Individuals Visualization Widget essentially serves as a message server indicating strong control integration.

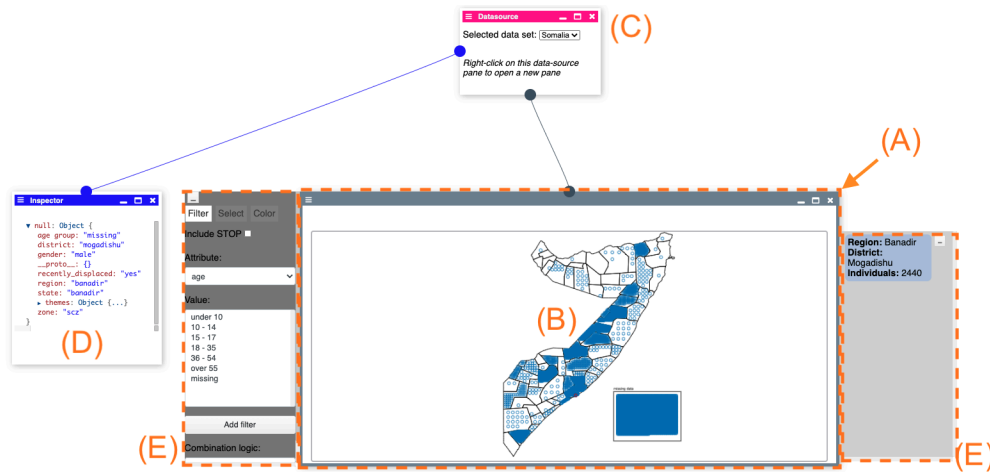


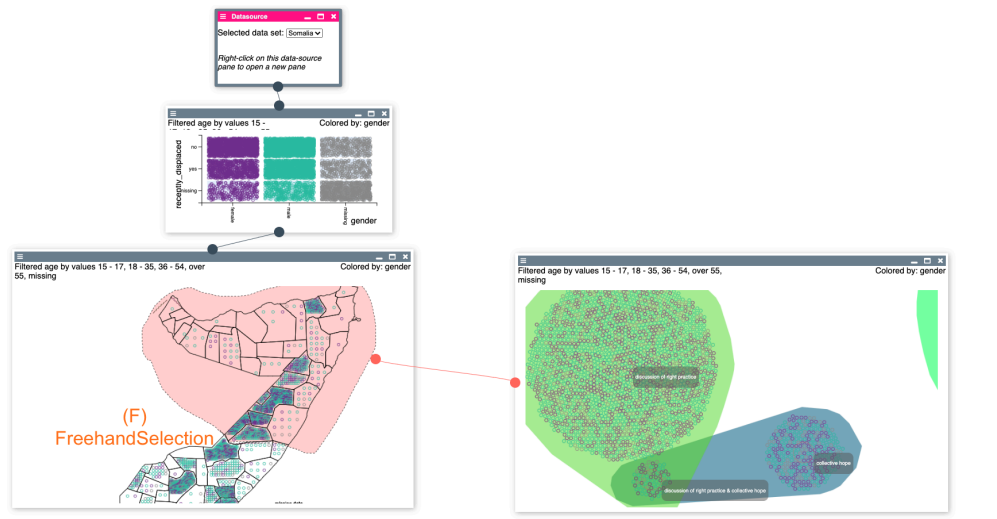
Figure 3.26: Basic components of the *Tree View* prototype



Figure 3.27: Creating a new visualization pane in the *Tree View* prototype

**Presentation Integration** Through shared components like the Global Control Widget, shared colors from the ColorStore object and the provided templates a Standard “look and feel” is reached, indicating a strong presentation integration.

As all our tools by integration design agree on the same integration levels on all dimensions, this fulfills the criterion of agreement in Wassermans notion of optimal tool integration. The question of appropriate integration levels is discussed in 3.5.4.



**Figure 3.28:** Using the *Freehand Selection* for selecting individuals to display in the newly created pane

### 3.5.3 Tree View

**Goal** Visualize the exploration flow. Allow users to easily go back to earlier steps and diverge on a different path. Make comparing different demographic groups or different data sets easy. Be able to see all views at one glance.

**Execution** As shown annotated in Figure 3.26, the centerpiece of the *Tree View* prototype is the pane (A). Each pane holds one visual component, for example, a visualization (B), a data source (C), or an individual inspector (D). When a pane holding a visualization is in focus, the user can interact with the visualization through the automatically expanded global and local controls known from the *Tab View* prototype (E). As can be seen in Figure 3.27, panes can be connected to other panes, spanning up a tree. New visualization panes can be created by a right-click on a pane (1) and choosing which visualization the new pane should display (2). The new child pane is created with the same state as the parent pane (3).

A second option for creating new panes is offered by the *Freehand Selection* (F), which allows users to visually select a group of points and create a new pane from them. When a pane receives an action, it sends it to its visualization, and all its children panes. To maintain a consistent view, inspect actions are sent to the whole tree a pane belongs to. As inspected individuals get highlighted, this additionally allows the user to see an individual in multiple contexts with little effort. Panes can be dragged and resized to allow users to construct a visual representation of their exploration flow. The data sources used as the basis of each tree are also held by a pane. Panes get their data through the connection to a data source. Using two data sources with different data sets thus allows the user to create exploration flows directly comparing two data sets.

For future work on the *Tree View* prototype three aspects should be taken into consideration.

**Reconnecting** Allowing users to detach and reconnect panes to other panes by hand, would allow an easy way of connecting two exploration paths. Different states of panes are not always combinable. We propose to combine all chainable state parts, for example, filter, and overwrite all singular state parts, for example, color, with the state of the parent pane.

**Cloning** As of now, users can clone single panes. To easily recreate exploration flows, for example, for usage with another data source, being able to clone user-defined subgroups of panes would be helpful.

**Required Space** Due to the many elements that need to fit on the screen simultaneously, a large screen size is needed to use the *Tree View* prototype. While scrolling and draggable elements already improve the experience, a zooming tool that could be applied to user-defined subgroups of elements might increase the usability even more.

#### 3.5.3.1 Technical Implementation

The *Tree View* prototype uses the composite pattern to implement an element structure that handles the integration. The connected elements are called panes and function as adaptors to integrate different components into the integration structure. The world is responsible for pane creation and the visualization of the connection between panes. For this, we use the library jsPlumb. A pane adapts precisely one component. The pane holds a state made up out of applied actions and is responsible for change propagation and state management. It gets initialized with the data and state from the parent pane, receives and propagates all applied actions to each child pane. We use the lively-window component class as the base for our panes, which comes with drag and resize support. Components that are adapted through the pane just have to offer a resizing method. Using the adaptor pattern, in the *Tree View* prototype, different components can be integrated with little effort, which has happened so far for the inspector and data source components. We continued to use action objects for communicating applied change to the representation of the data. As an additional feature, visualizations can use the *FreehandDrawer* to allow users to use the *Freehand Selection* workflow with them.

#### 3.5.3.2 Integration

We use the categories and measures for the integration of tools presented in section 3.5. Since we first defined our mechanisms for integrating and then adapted the tools accordingly, they can all be categorized together.

**Data Integration** The initialization with data from the parent pane and the following updates of state via direct messages from the parent pane indicate a weak data integration.

**Control Integration** Since the change propagation works via explicit messages from the parent to the child pane, each pane only ever can notify its children, indicating a weak control integration.

**Presentation Integration** The provided visual elements from the pane, and the integrated Global Control Panel serve as a Standard Toolkit, indicating a strong presentation integration. If the high presentation integration level of a Standard “look and feel” is reached depends on the integrated components.

As all our tools by integration design agree on the same integration levels on all dimensions, this fulfills the criterion of agreement in Wassermans notion of optimal tool integration. The question of appropriate integration levels is discussed in 3.5.4.

### 3.5.4 Discussion

In this section, we compare and discuss the two described integration approaches.

As Wasserman gives no definition for an appropriate level, we chose to evaluate appropriateness based on the ease of integrating decided by implementation effort needed for the integration structure and implementation effort needed to adapt the visualization tools. Appropriateness additionally can be influenced by external requirements, for instance, regarding consistent user interfaces.

#### 3.5.4.1 Extensibility

The Africa’s Voices Foundation operates in multiple new contexts each year and wants to shorten the feedback cycle from radio show to insights generation, compare 1.1.3.1. For this reason it’s important for them to be able to create new visualizations quickly and integrate them into their workflow. The two presented integration approaches, *Tab View* and *Tree View*, both enable integration of new visualizations with acceptable effort. For successful integration a visualization tool needs to implement the `applyAction` interface, and if applying, be able to send actions itself. For the panes prototype the visualization needs to offer a resizing method. Integrating the `FreehandDrawer` is not required, but recommended. On the other hand, integrated tools don’t need to provide a custom interaction user interface, for instance, for applying actions or viewing inspected individuals, which lessens time needed for creating new visualizations. A benefit of the *Tree View* prototype is that integrating other components than visualizations, for instance an inspector window, works well by design and doesn’t require changes to the whole prototype.

#### 3.5.4.2 Presentation Integration

Both presented approaches have a strong presentation integration. Providing a consistent user interface makes software easier to learn and use [67], which in our case could also help Africa’s Voices to speed up their insights generation, compare section 1.1.3.1. Therefore a strong presentation integration is appropriate for our visualization environments. With the commonly used templates, interaction widgets and classes the development of a Standard Toolkit has started. To advance

presentation integration this development should be continued, as well as user interface guidelines established.

#### **3.5.4.3 Control Integration**

Due to the amount of inter-tool notification needed whenever users change the data representation in one of the integrated tools, a preference for strong control integration might be assumed.

The *Tab View* and *Tree View* prototypes, however, differ strongly on their control integration levels. While the *Tab View* prototype implements strong control integration through a message server, the *Tree View* prototype uses weak control integration via direct messages. This difference is caused by their different inherent structures, which are reflected in their names. Communicating changes to every tab works well in the *Tab View* and propagating changes only to descendants works well in the *Tree View* prototype. Both methods work well regarding the effort needed for implementing the control integration structure, as well as effort needed for adapting tools, with the greatest effort for both methods being implementing the `applyAction` interface. The *Tree View* structure does have limitations, however. It makes the assumption that every component will only ever need to receive relevant information from its parent. This assumption does not hold. For instance, displaying an inspected individual in the whole pane tree, can require notifications from child panes to parent panes. Switching the control integration of the *Tree View* prototype to a message server would require either the message server or each listener to always check the existing parent-child relationships to determine if this message is relevant to the listener. As this would effectually also require maintaining a tree structure, the applied solution makes use of the already existing tree structure and uses the top pane of each tree as a reference point. Each pane has a reference on the top pane in the tree it belongs to; tree-wide actions can thus be applied by sending the action to the top pane.

While we think that both control integration levels are appropriate given the wanted control integration behavior, the direct messages integration mechanism of the *Tree View* prototype has already shown extendability issues and might prove more difficult to extend in the future.

#### **3.5.4.4 Data Integration**

Both integration approaches use a weak data integration. The used mechanism messages is the weakest on our used scale: Message, Shared Files, Database, Objectbase [67].

During the implementation of the environments, we actually discussed using a shared data object. If all visualizations tool could use a shared data object the implementation of our data integration, as well as our control integration would become easier. However, as we already noted in section 3.5.2, not all actions can be applied globally. Especially the grouping action which changes the position value of each individual does not make sense to be applied to other visualizations, because position is usually the main dimension each visualization works in. Using a shared data object would require visualizations to keep a second data object with

all individuals at least for their position value and ensure consistency between them. The action messages method we decided to use bases on the observation that in our domain the data can only be changed by predefined user interaction. Sending applied action notifications to the other components, allows each component to decide themselves if they want to display an applied action. As for the creation of a history the applied actions would need to be tracked either way, this does not create much overhead.

This leads to our conclusion that regarding data integration for visualization tools, Messages are an appropriate integration level on Wassermans scale.

### 3.6 Conclusion

Our goal was to transform our visualization ideas into working prototypes, showing that the ideas can be realized. Additionally, we wanted to provide a foundation for future implementations of the ideas.

We implemented ten visualization tools and extracted six interaction patterns from these implementations which work well with the individuals as points design model. We were able to show that displaying each individual as a point can serve as a design method for several viable visualizations. It is possible to implement visualizations with several thousand individuals represented as points in a way that, at all times, enables users to interact with a single individual. We discussed the limitations and benefits of using a random placement for the points and provided interaction concepts well suited for visualization tools using the individuals as points design model. We discussed our current exploration status on the various visualization tools we implemented and provide starting points for further exploration. We argue that integrating tools is relevant for achieving a high-level of explorability and that instruments developed in the research of programming tools can be applied to measure the integration of visualization tools. Using these instruments, we evaluated our approaches to visualization tool integration and propose ideas on which integration levels the integration of the presented visualization tools should proceed. In the context of our project, the implementation process provided insights regarding feasibility, challenges, and well-functioning approaches for the further visualization tool development of Africa's Voices. In the course of the implementation process, multiple topics regarding our chosen platform Lively4, the required Data-UI Mapping for interacting with individuals displayed as points, and the technical requirements for enabling responsive visualizations emerged, which will be discussed in the following chapters.





## 4 Using the Lively4 Platform with Its Active Content Capabilities to Conduct a Research-oriented Software Project

Working with a problem statement that is as vague as “designing interactive and explorable visualizations that do not lose the individual” is challenging. The domain-specific design space is enormous, the web browser prescribes the fundamental technologies, but the technologies to build visualizations on top of it come in a staggering variety. During the research work, we used the Lively4 system as our primary tool. Lively4 is a self-supporting collaborative development environment in the web browser which allows for conducting all of the following steps at a single place. First, we had to come up with novel interaction ideas. We documented and quickly prototyped those ideas with extended markdown capabilities in the Lively4 wiki. We then had to pure the most promising ones in advanced integrated prototypes using the web component standard. We shared the prototypes for testing purposes with our project partner via Lively4 for them to be tangible. All of this was done collaboratively in the Lively4 system in the web browser.

### 4.1 Introduction

The design of and research on novel visualizations that emphasize the individual and interaction patterns that make exploring large amounts of data possible comes with several challenges on different levels for the tools to use for this. First of all, there is a vast and unexplored space of possible ideas and solutions that are all to be discovered, tested, and evaluated according to technological feasibility and domain-specific fit. Because the design space is broad, prototyping has to be conducted rapidly to discover and focus on the most promising solutions quickly. Secondly, a variety of different technologies can do the task, but come with disadvantages and advantages, which have to be uncovered and evaluated. All of the findings have to be documented comprehensively and expressively, as our project lays the foundation for future research work. Thirdly, from a process point of view, there has to be proper tooling for productive collaboration because we were seven people working together on this research task.

To meet all those challenges, we chose the collaborative development environment Lively4. Lively4 is an environment that can be accessed through the web browser. It is built with HTML, CSS, and JavaScript and supports web standards such as web components. Active wiki documents and extended markdown capabilities to

prototype ideas as well as tooling to support the conception and implementation of advanced and modular prototypes made it effortless for us to switch from ideation to implementation without leaving the environment. A development style of direct feedback that is inherent to the Lively4 system allows for fast prototyping. In addition to that, Lively4 offers proper support for real-time collaboration and building a knowledge base as a team.

This chapter introduces the core concepts of the Lively4 development environment (section 4.2) and how we used them to conduct our project. We explain how markdown integration allows for active wiki content and prototyping with web technologies (subsection 4.3.1). After that, we introduce the web component technology, how it is integrated into Lively4, and how we used it to create advanced integrated prototypes (subsection 4.3.2). We describe how the tools of the Lively4 system allow for working in a direct and responsive way (subsection 4.4.1) to examine ideas and develop prototypes with a short feedback loop. We describe the real-time collaboration process and its support in the Lively4 system (subsection 4.4.2), as well as cover ways Lively4 uses versioning both locally and remotely to enable building and working in a robust environment (subsection 4.4.3).

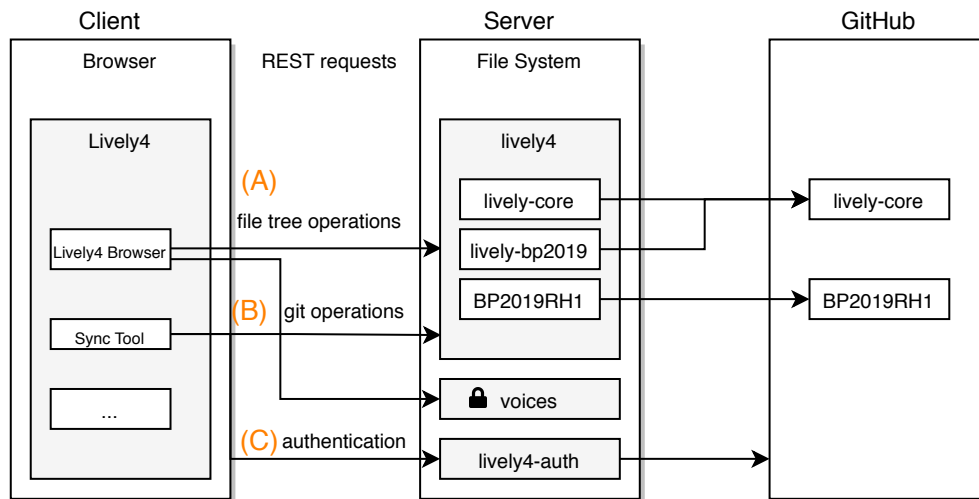
## 4.2 Lively4 System Introduction

In this section, we introduce the Lively4 environment. We explain the core concepts and ideas behind Lively4, examine the structure of client-server communication, and show the Lively4 client application and the Lively4 browser, which was the essential tool for the course of the project.

### 4.2.1 Lively4 Core Concepts

Lively4 [47] is a collaborative self-supporting development environment in the web browser [46]. We will break down this explanation in the following.

A development environment is used to create software. It offers some tooling that enables and helps a developer to edit source code. Such tools are editors, debuggers, and many more. The main tool to edit source code in the Lively4 system is introduced in section 4.2.3.1. Self-supportive development environments are environments that can evolve at runtime. This means the tools to create the software are written and run in the same environment as the software that is created. For example, developers can edit the source code of the source code editor in the source code editor. What happens when a user does that is defined by the system-specific implementation. Self supportiveness has the advantage for developers to adapt their environment, and quickly change system behavior to their needs without leaving it. One major drawback of such a system is that programmers can introduce breaking changes to the core components, which leaves the system useless for fixing the error. The collaboration aspect enables many developers to work asynchronously in and on the environment. We elaborate this in section 4.4. A wiki-like experience is established, where users can author content that everybody else can see [44]. Changes to the



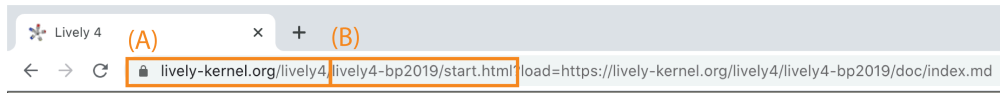
**Figure 4.1:** Overview of Lively4 client application and Lively4 server architecture

system made by users affect all other users of the system [46]. The web browser refers to the fact that the whole application is executed on the client-side in the browser with minimal needs for a server. The details on that follow in subsection 4.2.2.

Lively4 is a rewrite of the Lively-Kernel [38] that keeps the main concepts. Lively-Kernel was a research project at Sun Microsystems Labs whose initial goal was to bring live programming concepts as they are applied, for example, in Smalltalk to the web [38]. We will explain some of the essential concepts now. The first one is runtime programming [46]. With runtime programming, developers can alter the source code of an object or script, and all instances that exist in runtime instantly execute the new behavior. It enables short feedback loops, as developers can see the effect of a change right away. The second and third essential concepts are explorability and directness [50]. With them, developers are capable of inspecting HTML elements and the corresponding objects at runtime. They can examine the parent-child structure of an application visible on the screen, and even alter the state and behavior of this very instance directly [46]. Lively4 extends the live experience and ideas from Smalltalk with native support for rich media content such as images, PDF files, or videos [38].

#### 4.2.2 Server-client Structure

The Lively4 client application can be started by loading the start script of one of the Lively4 core systems, which is located at the URL that can be seen in Figure 4.2. This URL can be divided into two parts. The first one specifies the Lively4 server of the application (A). The second one specifies the Lively4 core system, from which the application boots, and the `.html` file, which contains the boot script (B). In



**Figure 4.2:** URL of Lively4 client application

this example, the Lively4 core system is `lively-bp2019`. The Lively4 server is a public instance of a Lively4 server.

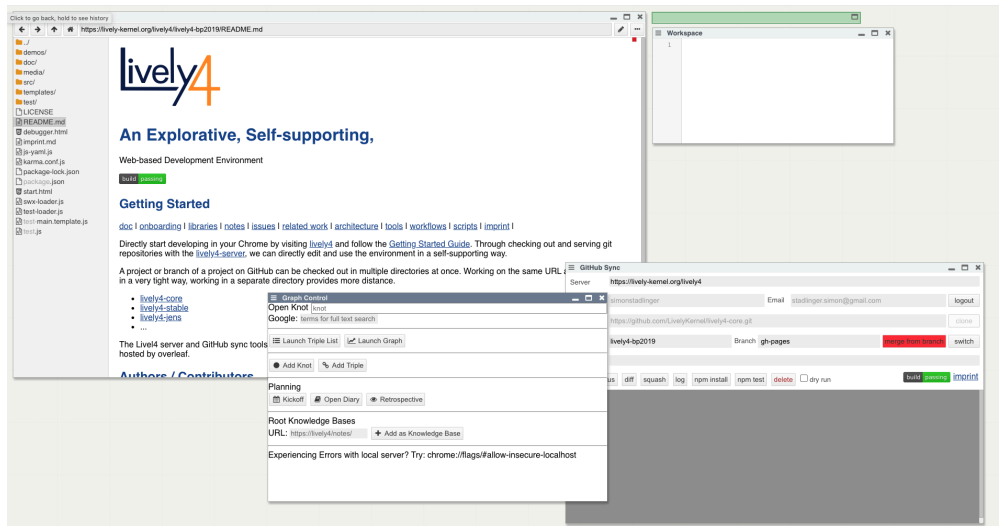
The Lively4 browser, which we explain in section 4.2.3.1, in a Lively4 client application, can access the file tree that is served by the Lively4 server instance. It communicates file tree operations via REST fetch calls to the server while using custom methods and headers to communicate the intent of the operation Figure 4.1 (A). An example such a custom call is documented in the Lively4 wiki.<sup>1</sup> Through such requests, folders, and files can be created, edited, or deleted directly from within the Lively4 client application. There are no access or operation restrictions on who can edit, create files on, or read files from the server in the first place.

The public `lively4` server instance serves several top-level folders. The `lively4-bp2019` folder contains a Lively4 core system. This folder contains all source files for all the tools and components needed in the front-end to run the Lively4 client application. It is a separate checkout of the `lively-core` core system repository on GitHub, like `lively4-core`, or `lively4-jens`. Here we can see the self-supporting aspect of Lively4. The sources of the core system that booted in the client browser in Figure 4.2 can be examined and edited within the booted Lively4 system. Another folder is `BP2019RH1`. This folder holds no core system but serves as the wiki folder of our project. We placed all the documentation and prototyping files in this folder. It is also versioned on GitHub. The folder `lively-server` contains the server's sources, which makes the file tree operations on the core system or the wiki folders possible.

A sync tool is accessible in the Lively4 client application, to operate the various repositories on the server Figure 4.1(B). To do that, users have to authenticate with an authorized GitHub account. The authentication can be done with the sync tool and a `lively-auth` server (C). How to operate the sync tool is explained in detail in section 4.4.3.2.

For reasons that we explain in section 4.4.1.3, we had a second private instance of a Lively4 server called `voices` running next to the public one.

<sup>1</sup><https://lively-kernel.org/lively4/lively4-jens/doc/architecture/filelist.md> (last accessed 2020-07-30).



**Figure 4.3:** Lively4 world loaded within the browser with a Lively4 browser, a workspace, github sync tool, and another Lively4 tool launched

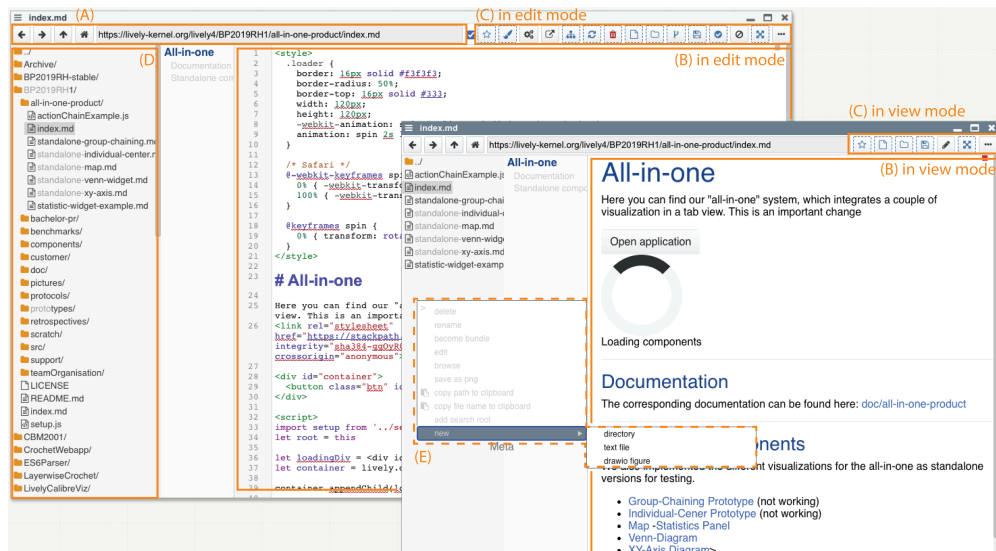
### 4.2.3 Workflows and the Lively4 Browser in the Lively4 Client Application

When a session is loaded, users are presented with a Lively4 world (Figure 4.3). A Lively4 world is a desktop-like workspace, where tools can be launched into windows. Those windows can be resized, minimized, and rearranged. This way, users can customize the development environment to their needs and likings. The state of a Lively4 world, for example, tools that are currently used, position and size of the windows that hold the tools, is stored in the local storage on the client machine. Every time users make changes to the state of the lively-world, this state is persisted in the local storage of the client machine. This way, it is ensured that when users leave a Lively4 world by closing the browser tab and return, the state of the Lively4 world from the previous session is restored, and all windows are at the desired place. With tools launched in windows, developers can explore or evolve the Lively4 core system, or collaborate with other developers on writing new applications within Lively4 and author the Lively4 wiki. We now introduce the Lively4 browser, which is an essential tool for doing that.

#### 4.2.3.1 Lively4 Browser

The Lively4 browser is implemented in the web component `lively-container`. A launched browser can be seen in Figure 4.4. Its UI mainly consists of four parts. The first is the navigation bar at the top (A). Here we can enter the URL of a specific file that resides on the server-side. The browser then fetches the file and displays it in the second part of the UI: the actual content container (B). The content container can display several types of files like images, PDF documents, draw.io figures, or simple text files, such as `.html` or `.md` files. The queried file's content can be viewed

#### 4 Using the Lively4 Platform with Its Active Content Capabilities



**Figure 4.4:** Lively4 browser with index.md loaded in \*edit\* mode (background); Lively4 browser with index.md loaded in \*view\* mode (foreground)

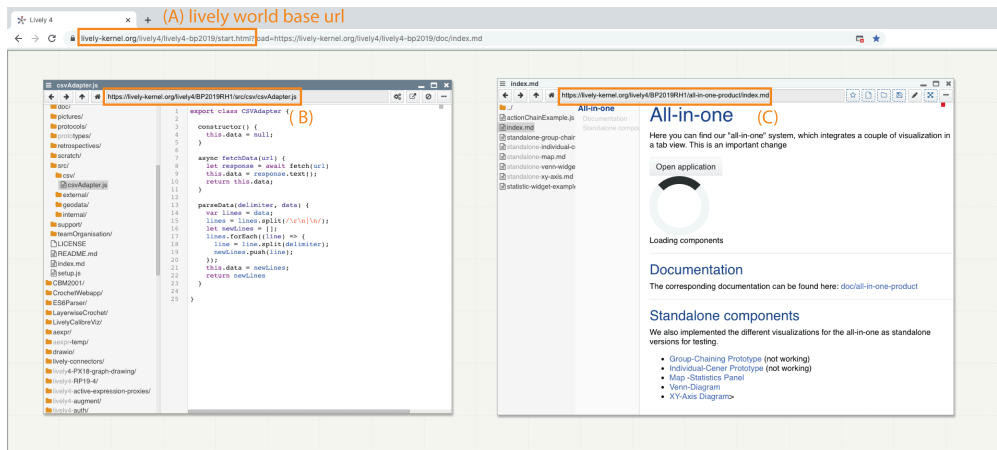
either in the edit mode (background) or in the view mode (foreground). Consider browsing a markdown file as the Lively4 browser has loaded in the Figure. If the browser is in edit mode, the raw source text of the markdown can be accessed in the editor and edited with the help of basic text editor functionality. If the browser is in view mode, the content of the markdown file gets rendered. The Lively4 browser can be switched from edit to view mode by a button in the third main UI area (C), the actions panel next to the navigation bar. Depending on the browser's mode, different actions for a file or the directory, in which the file resides, can be applied. The fourth main UI area is the file tree viewer next to the file content container (D). In this file tree viewer, we can browse and explore the Lively4 systems and wiki folders and files by hand. Basic file system operations like creating, renaming, or deleting can be accessed with a right-click and applied to files or folders (E).

The Lively4 browser is the primary tool for advancing and exploring the wiki. Using the Lively4 browser, users can navigate to any Lively4 server file without leaving the client machine browser page. In Figure 4.5, we can see the lively-bp2019 core system booted (A), while two Lively4 browser instances access different files in the Lively4 wiki (B, C).

A typical workflow of how to create and evolve the Lively4 wiki and core system with the Lively4 browser is described in section 4.4.1.2.

### 4.3 Technical Capabilities of Lively4

As our project did not have a clear problem statement, our work first consisted of exploring the design space of visualization. The outcome of this step is described



**Figure 4.5:** Lively4 client application loaded with two Lively4 browser windows launched. Browser windows both access different files from the Lively4 server.

in section 2.2. We then shifted towards experimenting with different visualization ideas and prototypes that we conceptualized during the design phase. Our third step was to implement more advanced prototypes of the visualization ideas our project partner liked the most. Those are reviewed in section 3.4. We wanted to make them reusable for multiple contexts without us needing to implement them twice or more.

We documented which libraries and technologies worked best for us during all these phases, how the implemented visualizations work, or - more towards the end of the project - how our wiki is structured. The requirements of this process were met with two technologies in Lively4. The first of which is the Lively4 wiki, in which we mainly worked with markdown files. The second is the Lively4 web component implementation, which lets us quickly built modular and reusable prototypes.

We describe both of these technologies in the following section, where we go into detail on how they work on a technological level as well as give examples of how we used them.

### 4.3.1 Markdown Files in the Lively4 Wiki

In this section, we focus on markdown, its underlying technology, and how Lively4 supports markdown.

#### 4.3.1.1 The Markdown Technology

Markdown is, first of all, a syntax specification for formatting plain text [35]. It is furthermore parsers that input markdown formatted plain text and output HTML conform content, based on the syntax rules of the markdown specification. The browser can then render this HTML output.

Markdown offers two features that helped us write more expressive and comprehensive documentation. Firstly, markdown can also include any HTML tags that are written in the plain markdown source in the output HTML tree. This

way, the expressiveness of markdown is extended to the expressiveness of HTML. For example, it allowed us to add HTML `<button>` or `<canvas>` elements into the documentation to make the wiki interactive. An example of such an interaction can be seen in section 4.3.1.5. Secondly, with extended markdown syntax, code snippets can be included in the text with three backticks before and after the code snippet. The code is then rendered as a visually separated block in a mono-type font (Figure 4.8 (A)). Some markdown parsers allow for syntax highlighting within the code block when the code snippet is tagged with the respective language. These features allow us to showcase code snippets.

##### 4.3.1.2 Lively4 Markdown Implementation

Lively4 uses the `markdown-it2` parser implementation, that converts CommonMark<sup>3</sup> Markdown to the respective HTML representation. CommonMark Markdown is a definition of markdown, that aims to standardize the partly ambiguous syntax of the original markdown. Lively4 adds the `markdown-it-attrs4` plugin to the `markdown-it` parser, which allows for specifying classes or attributes to any markdown element. This feature can be seen in the listing that generates the markdown in Figure 4.8, where the class `chartExampleLiveData` is added to the code block (line 3). Those classes or attributes are attached to the rendered HTML elements as classes or attributes.

Lively4 not only enables regular CommonMark markdown file parsing but enriches the markdown functionality with several additional capabilities that allow for a more active wiki experience. Those, as well as the markdown parsing itself, are implemented in a dedicated markdown component<sup>5</sup> in the Lively4 core system. Some of the additional functionalities that helped us and demonstrate the active content capabilities of Lively4 are now introduced by example.

##### 4.3.1.3 Documenting Our Architecture

The following markdown snippet is an extract from our documentation of the *Tab View* prototype (section 3.5.2).

```
1 ## Individuals Visualization
2 ### Basic Structure
3 #### UI
4 ![UI structure](component-view.png)
5
6 The application is build with Lively4 components. The components
   are stuck together through the components templates.
7
8 #### Backend
9 ![Basic Class Diagram](2020-04-07-class-diagram.drawio)
```

<sup>2</sup><https://github.com/markdown-it/markdown-it> (last accessed 2020-07-24).

<sup>3</sup><https://commonmark.org/> (last accessed 2020-07-24).

<sup>4</sup><https://github.com/arve0/markdown-it-attrs> (last accessed 2020-07-24).

<sup>5</sup><https://lively-kernel.org/Lively4/Lively4-bp2019/src/components/widgets/lively-markdown.js> (last accessed 2020-07-24).



## Individuals Visualization

### Basic Structure

#### UI

The application is build with lively components. The components are stuck together through the components templates.

#### Backend

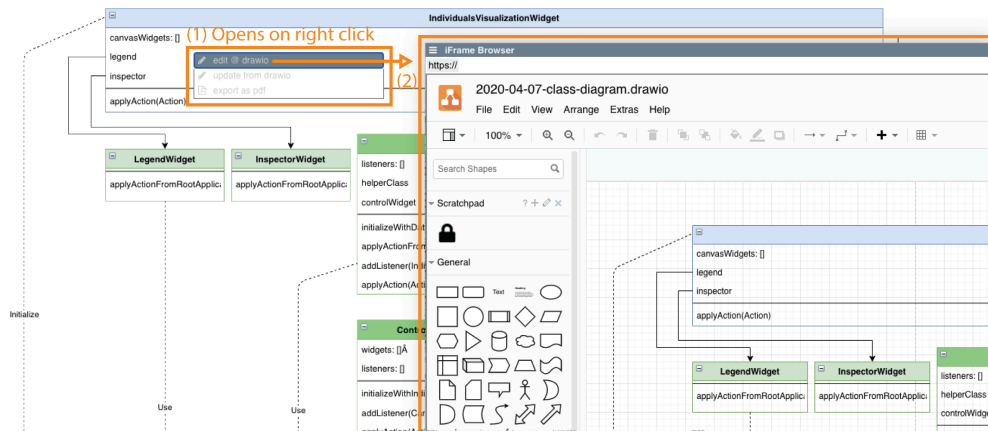


Figure 4.6: Rendered markdown with opened edit iFrame for the draw.io figure

10

11 The applications front-end view can approximately be mirrored in the back-end structure of the classes, since every component has its template, that represents its front-end as well as its implementation class, that contains business logic in the back-end.

The document uses standard markdown syntax for structuring its content. There is an overview image embedded as well as a draw.io figure. For simplicity, we exclude the overview image from the rendered markdown (Figure 4.6). The draw.io sketch of the architecture hierarchy is displayed as a regular figure. If we want to extend or alter the architecture sketch, we can open a Lively4 iFrame draw.io edit window (2) right from the documentation page (1). This way, extending and keeping the wiki up to date is direct and quick.

### 4.3.1.4 Including Code Evaluation

Consider the following markdown snippet, that resides in a small in our wiki.

```

1 ## Drop-down render text
2
3 Gender: <select id="x_axis_grouping_select"></select>
4 <button id="x_axis_grouping_button">Group</button>
5
6 <script>
7 let districtNames = ["a", "b", "c"]
8 var select = lively.query(this, "#x_axis_grouping_select");
9 for (let district of districtNames) {
10   select.options[select.options.length] = new Option(district);

```



**Figure 4.7:** Rendered markdown with executed scripts and native HTML elements embedded

```

11 }
12
13 lively.query(this, "#x_axis_grouping_button").addEventListener("
    click", () => {
14   console.log(select.options[select.selectedIndex].value)
15 })
16
17 </script>

```

The rendered result can be seen on the left side in Figure 4.7). The HTML elements from the markdown source code get rendered as well. Lively4 allows us to include the script tag and its JavaScript content in the markdown source code. The JavaScript source code inside the HTML script tags gets evaluated. We can tell because the dropdown is filled (1), and when clicking on the *grouping-button*, a message in the developer console gets displayed containing the current selection (2).

We can see another Lively4 feature in the JavaScript code. Line 13 contains a static query() function call on the Lively4 class. The Lively4 class gets imported from `./src/client/lively.js` during the boot process of the Lively4 application on the client-side. It then is available in the global document scope and, thus, inside the script of the listing. The Lively4 class implements a variety of static helper functions that developers can use anywhere in the Lively4 client application. In particular, it implements a query function. This query function takes the context of the current script as well as a CSS selector. It then queries the DOM nodes that the given selector identifies from the level of the script tag. We can use `lively.query()` for convenience because it is capable of finding elements that are hidden inside the shadow DOM of a web component as well (subsection 4.3.2. Calling the `getElementById()` on the document object would not find elements inside a shadow DOM.

#### 4.3.1.5 Making an Evaluation Interactive

The following markdown code snippet is an excerpt of our evaluation<sup>6</sup> of the visualization library `chart.js`. It shows the markdown code of a demo usage example. Its output can be seen in Figure 4.8)

<sup>6</sup><https://lively-kernel.org/Lively4/BP2019RH1/doc/research/libraries/chartJS.md> (last accessed 2020-07-26).

```

1  ## Examples
2  ### Plot live data update
3  ```JavaScript {chartExampleLiveData}
4  import Chart from "https://cdnjs.cloudflare.com/ajax/libs/Chart.js
   /2.8.0/Chart.bundle.js";
5
6  var ctx = this.parentElement.querySelector('#liveUpdateData').
   getContext('2d');
7  var chart = new Chart(ctx, {
8    type: 'line',
9    data: {
10     labels: [0,1,2,3,4,5,6,7,8,9],
11     datasets: [{
12       label: "Random Data",
13       backgroundColor: 'rgb(255, 255, 255, 00)',
14       borderColor: 'rgb(255, 99, 132)',
15       data: [0,1,2,3,4,5,6,7,8,9]
16     }]
17   },
18   options: {}
19 });
20
21
22 function randomizeDataOnChart(){
23   ...
24 }
25
26 function generateRandomData(){
27   ...
28 }
29
30 function updateChartWithNewData(chart, newData){
31   ...
32 }
33
34 let button = lively.query(this, '#randomizeButton');
35 button.addEventListener("click", randomizeDataOnChart);
36 ```
37
38 <script>
39 import boundEval from "src/client/bound-eval.js";
40 var source = lively.query(this, ".chartExampleLiveData").
   textContent
41 boundEval(source, this).then(r => r.value)
42 </script>
43 <canvas id="liveUpdateData"></canvas>
44 <button id="randomizeButton">Randomizee meee</button>

```

This snippet contains several markdown and Lively4 features that we explain step by step in the following:

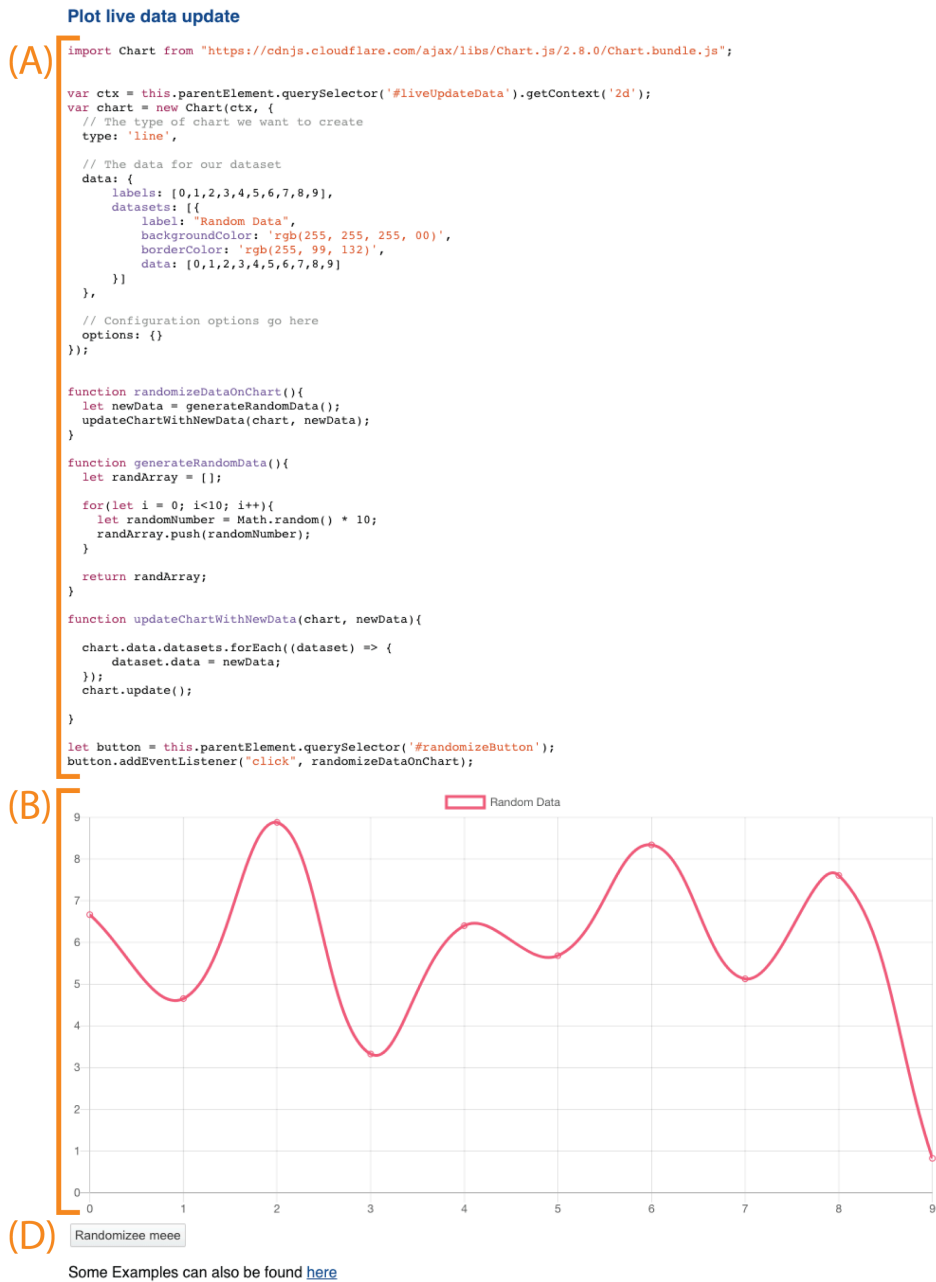


Figure 4.8: Rendered markdown with codeblock and chart on canvas

### Imports in Script Tags

```
1 import boundEval from "src/client/bound-eval.js";
```

The first line of the JavaScript inside the `script` HTML tag, uses an `import` statement, where the `bound-eval` core module of Lively4 is imported as a dependency. The `import` statement and its corresponding `export` statement were first introduced in the ES6 JavaScript language specification.<sup>7</sup> They allow for creating modular applications by importing JavaScript classes, functions, or constants into different files and contexts. Lively4 enables the scripts in markdown files to use advanced syntaxes, such as `import` statements, active expressions, or `async-await` constructions. In this example, an `async` function is exported in `bound-eval.js`. The function is now available with the `boundEval()` function call.

### Evaluation of JavaScript Code Snippets

```
1 boundEval(source, this).then(r => r.value)
```

The JavaScript code snippet from line 4 to line 36 is rendered as a separate code block with syntax highlighting as defined in the markdown syntax specification Figure 4.8) (A). It is also evaluated through the JavaScript code within the script tags (line 39 to line 42). The Lively4 core module, `bound-eval`, makes this possible. It gets the raw source code extracted from the rendered code block via CSS selector annotation and `lively.query()`, as well as the context in which the raw source code should be executed. The `boundEval()` implementation allows for an execution of JavaScript that includes module imports, something the JavaScript in-build function `eval()` can not. The execution of advanced JavaScript code is necessary because we want to import the `chart.js` library from a Content Delivery Network<sup>8</sup> at the top of the evaluated code.

### Advantages of the Lively4 Markdown Features

The listed features, combined with the markdown and Lively4 capabilities mentioned above, make documentation more expressive and interactive.

Readers of the documentation get provided with a code demo of how to import, initialize, and configure a chart with `chart.js` and how this chart is displayed in the browser. They can see the outcome of this code directly underneath it in the form of the actual chart on the canvas element (Figure 4.8)(B). They do not have to look at a screenshot whose content might not even fit the code on top of it. From the perspective of developers, this has another advantage. The code presented in the wiki also results in the chart presented in the canvas element at the end of the markdown snippet. This single source of truth prevents the possible slow divergence of two code versions and preserves the documentation's integrity.

<sup>7</sup>[https://exploringjs.com/es6/ch\\_modules.html#sec\\_basics-of-es6-modules](https://exploringjs.com/es6/ch_modules.html#sec_basics-of-es6-modules)  
(last accessed 2020-07-26).

<sup>8</sup><https://www.cloudflare.com/learning/cdn/what-is-a-cdn/>  
(last accessed 2020-07-26).

The documentation also has an interactive notion in the form of a button (C). The button gets its behavior from the code snippet that is evaluated, so readers can understand what gets triggered with the button because it is visible in the code block. Readers become users of the documentation. They can then experience the outcome of the button press live in the rendered chart. In this example, the chart data gets randomized, and the line in the chart re-drawn.

This way, we can give a direct sense of how animations look and feel not only for `chart.js`, but for all visualization libraries we tested.

#### 4.3.1.6 Client Side Wiki

Due to the client-side wiki architecture of Lively4, which is explained in detail in section 4.4.1.2, for all the examples mentioned above, users could alter the markdown source code instantly by switching the browser to edit mode. If they want to see how another configuration affects the chart's look, or how the chart performs with other data, they can use the Lively4 browsers edit mode and alter the code in the markdown sources. After finishing the editing process and saving the file, the changes can be seen rendered right away in the same Lively4 browser window.

### 4.3.2 Web Components in Lively4

Now we are going to elaborate on the technologies that make up web components. We explain how the web components standard is implemented and supported in Lively4 and how we used this implementation to create more complex prototypes for Africas Voices.

#### 4.3.2.1 The Web Component Technology

The goal of web components is to give web developers the ability to write custom and complex UI elements and define their behavior as modular, isolated, and re-usable. The web components technology consists mainly of three web standards,<sup>9</sup> which are used together. Those are custom HTML elements, shadow DOM, and templating and slotting. In the following, we explain each of them.

#### Custom HTML Elements

The web standard allows for defining custom HTML elements<sup>10</sup> with custom behavior that can then be used repeatedly throughout the website. A custom element must be registered in the window's custom element registry with the following function call:

```
1 window.customElements.define('custom-element', CustomElement);
```

<sup>9</sup>[https://developer.mozilla.org/en-US/docs/Web/Web\\_Components](https://developer.mozilla.org/en-US/docs/Web/Web_Components) (last accessed 2020-07-22).

<sup>10</sup><https://developers.google.com/web/fundamentals/web-components/custom-elements> (last accessed 2020-07-22).

This method needs two arguments. The first one being a *DOMString*, which represents the tag name of the custom element. With this name, the element can be embedded in the DOM later. The second argument is the JavaScript class, which defines the behavior of the object. The class object is written using the standard ES2015<sup>11</sup> class syntax. A basic CustomElement JavaScript class could look like the following.

```

1 class CustomElement extends HTMLElement {
2   constructor() {
3
4     super();
5     //additional custom functionality here
6     var text = this.getAttribute('elem-text');
7     this.innerHTML = text;
8   }
9 }

```

The CustomElement class object has to inherit from the predefined HTMLElement<sup>12</sup> interface object. This inheritance gives the CustomElement all necessary standard HTML element functionality like, for example, a style attribute. Within the constructor, the constructor of the inherited class has to be called. Custom behavior can also be defined here. In this example case, the element can query the attribute `elem-text`. An instance of CustomElement is the HTML `<custom-element>` element. The attribute of the element can be accessed by calling the standard JavaScript `getAttribute` function on the instance object itself. The value of the inner HTML can be defined by setting the `innerHTML` property of the JavaScript object. The usage of the custom element could look like the following:

```

1 <custom-element elem-text="this is an important message"></custom-
  element>

```

## Shadow DOM

To give the HTML `<custom-element>` element not only custom behavior but also the ability to customize the DOM structure and style of the element in an encapsulated way, the shadow DOM<sup>13</sup> technology can be used. Shadow DOM allows a hidden DOM to be attached to an element.

Figure 4.9 (Based on Mozilla Developer Network)<sup>14</sup> shows how the shadow DOM is separated from the actual document tree (left) and how it gets integrated for rendering in the browser (right). A shadow DOM tree starts with a shadow root attached to the shadow host in the actual DOM tree of the document.

<sup>11</sup><http://www.ecma-international.org/ecma-262/6.0/#sec-class-definitions> (last accessed 2020-07-23).

<sup>12</sup><https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement> (last accessed 2020-07-22).

<sup>13</sup>[https://developer.mozilla.org/en-US/docs/Web/Web\\_Components/Using\\_shadow\\_DOM](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM) (last accessed 2020-07-23).

<sup>14</sup>[https://developer.mozilla.org/en-US/docs/Web/Web\\_Components/Using\\_shadow\\_DOM](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM) (last accessed 2020-07-23).

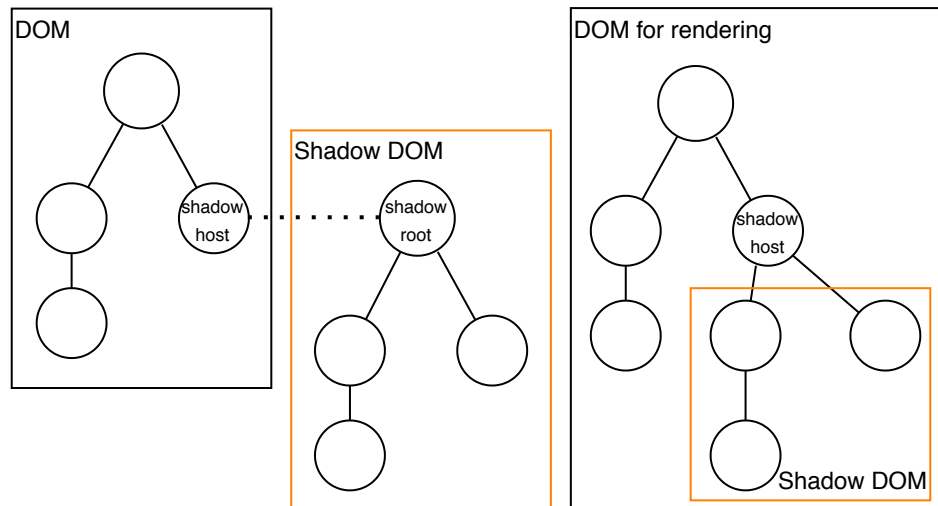


Figure 4.9: Shadow DOM in relation to the document DOM

A shadow DOM with its root can be attached to every HTML element. Especially the CustomElement can be enriched with this functionality. For this, the shadow root has to be attached to the element in the constructor of CustomElement. The HTML `<custom-element>` element acts as the shadow host for the shadow DOM. The shadow DOM tree can be built up with the standard JavaScript API for DOM manipulation (line 7 to line 9). An example can be seen in the following listing.

```

1 class CustomElement extends HTMLElement {
2   constructor() {
3     super();
4     //build up shadow DOM
5     let shadowRoot = this.attachShadow({mode: 'open'});
6     var paragraph = document.createElement('p');
7     paragraph.innerHTML = "this is some super reusable HTML"
8     shadowRoot.appendChild(paragraph);
9     // etc.
10  }
11 }

```

The mode of the shadow root (line 6) defines whether the element's shadow DOM is accessible with JavaScript that runs in the context of the surrounding document. Shadow DOM functionality allows for encapsulating a custom markup structure with custom functionality behind the reference of a custom HTML element. However, the approach of building complex DOM trees that make use of classes, ids, or styles just with JavaScript can quickly clutter the code of the class that defines the custom element. That is the problem where the third web standard comes into play.



## Templating and Slotting

When the HTML `<template>` element is used in the DOM, it and its content are not rendered but can still be referenced with JavaScript.<sup>15</sup> The following listing shows an example of a template.

```

1 <template id='custom-template'>
2   <style>
3     p {
4       color: red;
5     }
6   </style>
7   <p>
8     This is some super reusable HTML
9   </p>
10 </template>

```

Since an HTML `<template>` element can be referenced, it can be queried in the constructor of the `CustomElement`, its content extracted and attached to the shadow root. This way, the DOM structure does not have to be built with JavaScript. Another benefit of this is that the shadow DOM can have its own encapsulated styling, which is embedded in the way regular styling is in the regular DOM tree (line 3 to line 5).

```

1 class CustomElement extends HTMLElement {
2   constructor() {
3
4     super();
5     //attach shadow DOM from template
6     let template = document.getElementById('custom-template');
7     let shadowRoot = this.attachShadow({mode: 'open'});
8     shadowRoot.appendChild(template.content.cloneNode(true));
9   }
10 }

```

An important point here is that a deep clone<sup>16</sup> of the DOM tree, which resides inside the template, has to be attached to the shadow root of the `<custom-element>` since the content instance inside the HTML `<template>` element is not rendered (line 8).

Right now, only one configuration of this reusable DOM subtree can be used. If the custom styling, which this element has, should be reused at another place in the document, which needs different text to be rendered inside the paragraph, this is not possible. The text is hardcoded inside the web component's template. The solution to this is the HTML `<slot>` element. Those elements function like a placeholder, which can be filled from the outside. The following listing presents an example that extends the template from above.

<sup>15</sup>[https://developer.mozilla.org/en-US/docs/Web/Web\\_Components/Using\\_templates\\_and\\_slots](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_templates_and_slots) (last accessed 2020-07-23).

<sup>16</sup><https://developer.mozilla.org/en-US/docs/Web/API/Node/cloneNode> (last accessed 2020-07-23).

```
1 <template id='custom-template'>
2   <style>
3     p {
4       color: red;
5     }
6   </style>
7   <p>
8     <slot name="first-slot">
9       This is some reusable, replaceable text
10    </slot>
11  </p>
12 </template>
```

To fill the slot, some HTML inside the `<custom-element>` can be included in the regular document tree. The `slot` attribute has to be specified with the name of the slot in the template of the custom element to link the new HTML to the slot. The following listing shows how HTML code can replace the placeholder from the outside.

```
1 <custom-element>
2   <span slot="first-slot">This is some replacing text</span>
3 </custom-element>
```

Those three technologies allow for building web components that are encapsulated in their function and form. They can remain configurable from the outside and are reusable throughout the document. We now look at how Lively4 integrates the web component technology.

#### 4.3.2.2 Web Components in Lively4

Web components are the very basic building blocks of Lively4. Lively4 not only supports web components but also enhances the experience when creating and working with them. The features Lively4 built around web components are implemented in the `component-loader`<sup>17</sup> module of the Lively4 core system. We explain some of those features in the following paragraphs.

##### Creating a Web Component in Lively4

When we want to create a web component, Lively4 helps by abstracting all the manual steps that come with using the web component standard. These steps are: registering the component as a custom element, creating the shadow root, and appending the shadow DOM, which would have to be cloned from a template to the shadow root.

We only have to specify one `.html` file and one `.js` file. Those files have to have the same name, which is going to define the DOM string. With the DOM string, the component can be used in HTML code later on. The JavaScript file has to export the class that implements the behavior of the new web component. A Lively4 web

<sup>17</sup><https://lively-kernel.org/Lively4/Lively4-bp2019/src/client/morphic/component-loader.js> (last accessed 2020-07-23).

component differs from the standard because it has to implement an `initialize()` function that functions as the constructor of the component. The HTML `<template>` element inside the HTML file has to have the new web component's DOM string as the value of its `id` attribute.

The `component-loader` module then takes care of registering the newly created web component. The `component-loader` needs to be provided with the DOM string of the web component, to register it. It then queries an internally kept list of folders for two files that are named the same as the web component. This list of folders, which are mainly subfolders of the currently used Lively4 core system, can be extended temporarily. We used this feature to keep the source code of our web components inside our wiki folder, and only added this folder to the `component-loaders` search list when we needed the web components. The `component-loader` then extracts the class from the JavaScript file. It extracts the template from the HTML file and, with both of those objects, registers the web component using the standard JavaScript API that was mentioned in section 4.3.2.1.

### Using Web Components in Lively4

The process of registering a web component is mostly not triggered manually. Instead, it is executed when the web component is needed. To understand when web components are registered, we have to look at how they get instantiated. There are two ways web components are instantiated in Lively4. The first one is programmatically. With the static `create()` function of the global `lively` class, we can dynamically create an instance of a web component in any JavaScript snippet and append it via the standard JavaScript API to existing DOM elements. That allows users to create web components even in the code of other web components. The `create()` function uses the `component-loader` internally. The second way of instantiating a web component is by explicit using the HTML tag of the web component in an HTML markup. The tag can be inside a markdown file, as markdown supports native HTML, or in the template of another web component. The later allows for deeply nested, reusable structures.

The Lively4 UI is built with web components, which can be seen in Figure 4.10. Every Lively4 browser is an instance of the `lively-container` component (B), which itself is composed of several other web components, such as the navbar on the left (C) and the markdown container on the right (D). The container itself and every other tool that can be launched in a Lively4 world, are launched inside an instance of the `Lively4-window` component (A).

#### 4.3.2.3 Building Visualization Prototypes with Web Components

We now show how we used web components to polish ideas we had during the ideation phase and build advanced prototypes. The focus of those was to combine different visualization ideas we had tested with markdown scripting and to make them reusable.

## 4 Using the Lively4 Platform with Its Active Content Capabilities

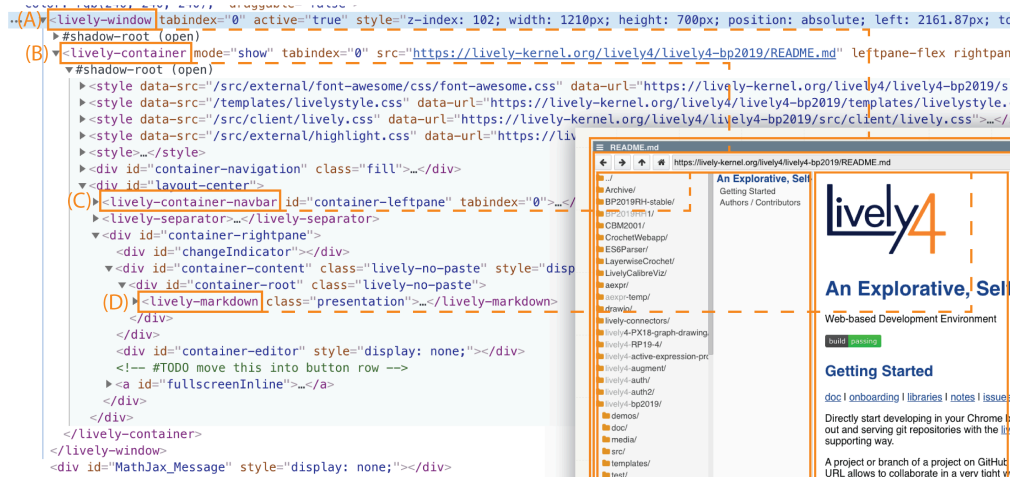


Figure 4.10: Lively4 browser web component architecture visible in the DOM

### Venn Widget Nesting

Consider the following part of a template of the component `bp2019-venn-widget`.<sup>18</sup> This component was initially created for the *Tab View* prototype. The prototype gets explained in detail in section 3.5.2. The component was designed to encapsulate the Venn diagram. More information on the Venn diagram and its concepts can be read in section 3.4.3.

```
1 <template id="bp2019-venn-widget" >
2   <style>
3     ...
4   </style>
5   <div id="venn-widget-root-container" class="canvas-widget-root-
6     container fluid-container">
7     <div class="flex-row height-100">
8       <div id="venn-widget-control-widget-container">
9         <bp2019-venn-control-widget id="venn-widget-control-widget"
10        >
11         </bp2019-venn-control-widget>
12       </div>
13       <div id="venn-widget-canvas-container" class="canvas-widget-
14         canvas-container">
15         <canvas id="venn-widget-canvas" width="1000" height="600"><
16         /canvas>
17         ...
18       </div>
19     </div>
20   </div>
21 </template>
```

<sup>18</sup><https://lively-kernel.org/Lively4/BP2019RH1/components/bp2019-venn-widget.html> (last accessed 2020-07-15).

The component essentially consists of the canvas on which the dots are drawn (line 12) Figure 4.11(B) and the `bp2019-venn-control-widget` component (line 8). This component encapsulates the look and functionality of interactive control panels to alter the visualization (A).

Part of the class's implementation that defines the `bp2019-venn-widget` component can be seen in the following listing. This class holds references to both the canvas and the `bp2019-venn-control-widget` instance (line 7, line 10). These references allow for calling functions directly on those objects. In this example, the data that is going to be visualized is propagated to the control widget (line 17).

```

1 import VennDiagram from "../src/internal/individuals-as-points/venn
  /venn-diagram.js"
2
3 export default class VennWidget extends Morph {
4   async initialize() {
5     this.listeners = []
6     this.name = "venn-widget"
7     this.controlWidget = this.get("#venn-widget-control-widget")
8     this.controlWidget.addListener(this)
9     ...
10    this.canvas = this.get('#venn-widget-canvas')
11    this.vennDiagram = new VennDiagram(this, this.canvas...)
12    ...
13  }
14
15  async setData(individuals) {
16    this.individuals = individuals
17    this.controlWidget.initializeAfterDataFetch(this.individuals)
18  }
19
20  _addThemeGroup(addedAction){
21    this.vennDiagram.addThemeGroup(
22      addedAction.uuid,
23      addedAction.name,
24      addedAction.themes,
25      addedAction.color)
26  }
27 }

```

Imports can be used to modularize applications further and take full advantage of the JavaScript module system. In this case, the `VennWidget` class only takes care of the visual representation and wrapping of all the elements of the *Venn Diagram* prototype, while all the domain-specific logic is implemented in a separate `VennDiagram` class. The `VennWidget` therefore holds an instance of a `VennDiagram` (line 11) to which it propagates the canvas on which the data needs to be drawn and actions, which come from the control-widget object (line 20 to line 26). Actions are explained in detail in section 3.3.

### Individual Visualization Slotting

The VennWidget is part of the *Tab View* prototype. The rough structure of this prototype is defined in the `bp2019-individual-visualization`<sup>19</sup> component. This component acts as the root element of the *Tab View* prototype. The following code listing presents parts of its HTML template. The instantiated prototype can be seen in Figure 4.11.

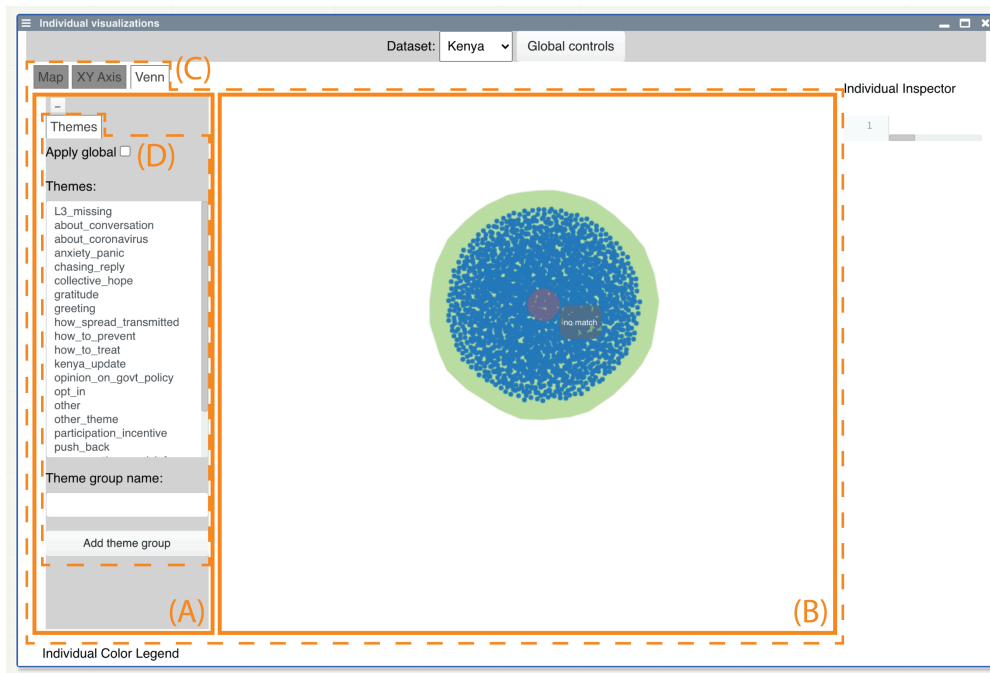
```

1 <template id="bp2019-individual-visualization" >
2   ...
3 <div id="individual-visualization-root-container" class="
4   container-fluid">
5   ...
6 <div id="canvas-inspector-row" class="row">
7   <div id="canvas-tab-view-container" class="col-10">
8     <bp2019-tab-widget id="canvas-tab-widget">
9       <div id="tab-buttons" slot="tab-buttons" class="p-1">
10        <div data-content-id="bp2019-map" class="tab">Map</div>
11        <div data-content-id="bp2019-y-axis" class="tab">XY
12         Axis</div>
13        <div data-content-id="bp2019-venn-diagram" class="tab">
14         Venn</div>
15      </div>
16      <div id="tab-contents" slot="tab-contents" class="p-1">
17        <bp2019-map-widget id="bp2019-map"></bp2019-map-widget>
18        <bp2019-y-axis-widget id="bp2019-y-axis"></bp2019-y-
19         axis-widget>
20        <bp2019-venn-widget id="bp2019-venn-diagram"></bp2019-
21         venn-widget>
22      </div>
23    </bp2019-tab-widget>
24  </div>
  ...
  </div>
  ...
  </div>
</template>

```

The template uses the `bp2019-tab-widget`, which is another web component we created (line 7). This component encapsulates the styling and behavior of a tab view and contains no domain-specific logic. In the first slot (line 8 to line 12), tabs can be registered Figure 4.11(C). In the second slot (line 13 to line 17) the content, that should be visible, when the corresponding tab is clicked, is inserted. We used the tab view here to create a tab for each visualization prototype (line 14 to line 16). Especially the `bp2019-venn-widget` component gets instantiated here. The tab view component was also used to structure different control panel views throughout

<sup>19</sup><https://lively-kernel.org/Lively4/BP2019RH1/components/bp2019-individual-visualization.js> (last accessed 2020-07-15).



**Figure 4.11:** UI of the *Tab View* prototype divided according to the web component structure

the control widgets of every visualization, that was part of the *Tab View* prototype (D).

### Reusing the Visualization Components

Our project partner wanted to see the visualizations, which we built as web components, next to each other, and wanted to be able to create many visualizations of the same type with different controls applied. For this reason, we built the *Tree View* prototype as a new way of combining visualizations. More on the concept of this prototype can be read in section 3.5.3. It was no web component on its own but was implemented as a script in a markdown file.<sup>20</sup> It yielded a completely different architecture but had to use the same visualizations as the *Tab View* prototype at its core. We were able to use the same visualizations without copy-pasting or rewriting the code by instantiating the web components we used for the all in one prototype. The components had to be programmatically created because users should be able to generate new instances of the visualizations in the exploration process. The programmatic creation of web components can be seen in the following listing. Lines 1 to 10 shows the code of the menu users is presented with. The UI of the menu can be seen in Figure 4.12. When clicked (1), the menu items ultimately trigger the creation of the visualization component (2), which gets wrapped in a prototype specific pane

<sup>20</sup><https://lively-kernel.org/Lively4/BP2019RH1/prototypes/panes/connection-spike.md> (last accessed 2020-07-16).

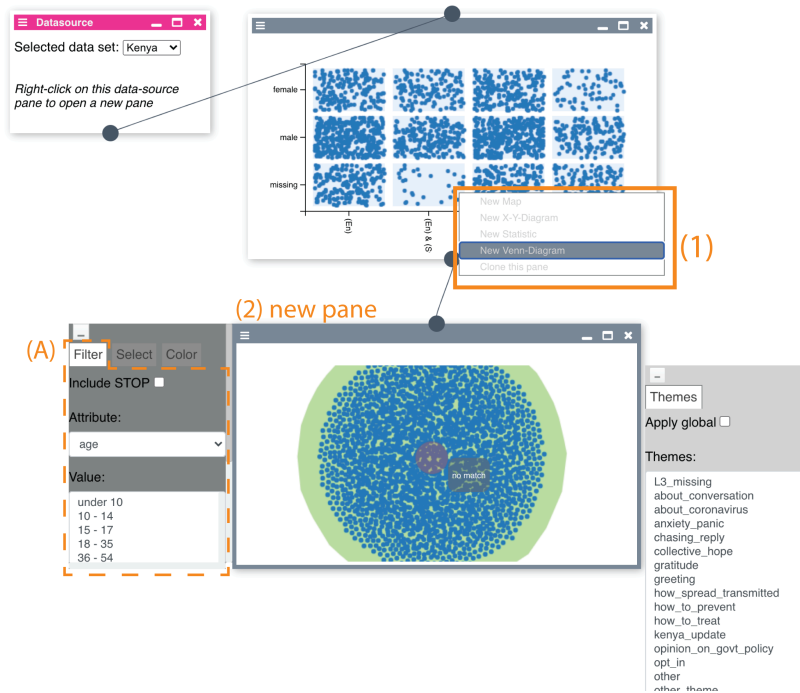
component (lines 12 to 33 ). In a global control menu, the tab view component is used again (A).

```
1 async function openChildTypeMenu(parentPane, evt, selection = false
  , dataSource = false) {
2   ...
3   const menuItems = [
4     ['New Map', () => createNewChildPane(parentPane, 'bp2019-map-
      widget', evt)],
5     ['New XY Diagram', () => createNewChildPane(parentPane, 'bp2019
      -y-axis-widget', evt)],
6     ['New Statistic', () => createNewChildPane(parentPane, 'bp2019-
      statistic-widget', evt)],
7     ['New Venn Diagram', () => createNewChildPane(parentPane, '
      bp2019-venn-widget', evt)]
8   ]
9   ...
10 }
11
12 async function createNewChildPane(parentPane, childComponentName,
  evt) {
13   ...
14   let childVisualization = await createNewVisualization(
15     childComponentName,
16     ...
17   )
18
19   let childPane = await createNewPane(
20     childVisualization,
21     childComponentName,
22     childPosition,
23     ...
24   )
25   ...
26   return childPane
27 }
28
29 async function createNewVisualization(componentName, ...) {
30   let visualization = await lively.create(componentName)
31   ...
32   return visualization
33 }
```

### 4.3.3 Markdown Vs. Web Components in Lively4

We are now going to distinguish the two technologies from a usage perspective and point out differences in their life cycle. Markdown and web components are heavily entwined in Lively4. Every markdown source that is accessed via the Lively4





**Figure 4.12:** Workflow for dynamically creating new web component instances in the UI of the *Tree View* prototype

browser is rendered in a `lively-markdown`<sup>21</sup> component Figure 4.10 (4). Such components lay the foundation on which the Lively4 wiki with its markdown files can be explored without leaving the client's browser window. However, it is also possible to instantiate web components in a markdown source since it allows for using native HTML elements and scripting. The embedding can be seen in the *Tree View* prototype, where all panes are based on a `lively-window` instance, which is placed in the DOM of the rendered markdown (Figure 4.12).

#### 4.3.3.1 Usage

Markdown and scripting in markdown files are for bringing documentation to life, or prototyping ideas and testing things out. That is what the enhancements in Lively4 regarding markdown are meant for. Lively4 offers the freedom to use modern JavaScript syntax like async functions to write expressive scripts. Much of the process of editing a wiki directly and responsively is taken care of and automated (section 4.4.1.2). The user can browse a markdown file. The file is received by the client application and gets rendered with no additional steps that transform or preprocess it in a form altering way. If users want to change something, for example, alter script behavior, they can edit the document in edit mode and sees the result of the change right after saving the file and switching the browser back to view mode. The change

<sup>21</sup><https://lively-kernel.org/Lively4/Lively4-bp2019/src/components/widgets/lively-markdown.js> (last accessed 2020-07-16).

that was introduced to the source file gets automatically persisted on the server. The markdown implementation in Lively4 has an inherent drawback. When writing more and more JavaScript code to integrate more features in the ideas that reside in the markdown file or having to build up substantial HTML structures, the code quickly gets cluttered, which yields bad maintainability.

As the problem space gets more complex than simple interaction pattern and solution strategies narrow down, so we do not have to prototype different ways but need one complex solution, web components play out their strengths. A web component DOM element is represented in a JavaScript object, which is an instance of a well-defined class implementation. This class usually resides in a single JavaScript module. In this module, imports can be used to modularize the code further and divide it into responsibilities for different functions. Within the shadow DOM template, other web components can be used to modularize form *and* function. More engineering thoughts can go into actual application software architectures, and problems can be abstracted into reusable solutions. These advantages come with several disadvantages. To make sure such encapsulated elements work in the context of the HTML and JavaScript standards, they have to be registered as explained above (section 4.3.2.1). The source code of a web component is split into two code files that are processed by the client browser during the registering to something that we can use as a component in HTML code. This step yields an additional indirection from the source code to the result makes it harder to navigate from the end representation of a rendered web component template and its behavior to the source code. The rendered instances can neither be altered as easily by users as the markdown source nor can the instant feedback behavior be experienced. The web component becomes a polished and sealed experience where only the encoded behavior is allowed.

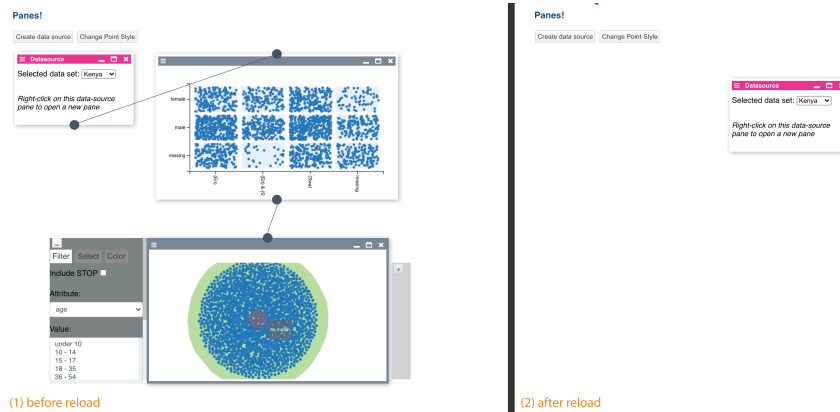
The transition from markdown to web components is a natural progression. Concepts emerge and can be poured into an application. Scripts of a markdown file can be decluttered by extracting functionality into standalone JavaScript modules.<sup>22</sup> As a next step, HTML structures can be refactored into templates and combined with functionality to form reusable web components. Our project went through this process, as already described in the introduction of this chapter (section 4.1).

#### 4.3.3.2 Life Cycle

The other level on which the two technologies can be differentiated in Lively4 is the context. Markdown files make up large parts of the Lively4 wiki. More generally speaking, the Lively4 wiki consists of files that are stored on the server, get requested by the Lively4 client application, and displayed in the front-end. When these files are altered, their content gets written back to the server file system and is re-rendered in the front-end. Lively4 web components, although defined through files, are *created* only in the Lively4 client application. They are registered in the first place for the usage in the client browser that currently needs them to be rendered. Nothing that defines an *instance* of a web component gets stored on the Lively4 server. Lively4

---

<sup>22</sup>[https://exploringjs.com/es6/ch\\_modules.html#sec\\_basics-of-es6-modules](https://exploringjs.com/es6/ch_modules.html#sec_basics-of-es6-modules)  
(last accessed 2020-07-23).



**Figure 4.13:** The *Tree View* prototype markdown view before and after reloading the Lively4 browser

provides mechanisms, where the configuration of instances of web components can be persisted in the local storage of the client's browser. That is why the state of a Lively4 world, for example, the position and size of the windows, can be restored and bootstrapped back up once the client browser reloads the page. The web components state is lost after a reload of the client browser, when not using the tools Lively4 offers to persist its configuration of it locally. This state loss also holds when reloading the Lively4 browser on a resource that contained web components, which can be seen in the following scenario.

The context differences can be showcased on the *Tree View* prototype application. The inner workings and concepts behind this prototype can be read in section 3.5.3. Figure 4.13 (1) shows the *Tree View* prototype's markdown in *view mode* after a user opened some visualizations and connected them.

The visualizations and their wrapping panes have been authored in the view mode of the Lively4 browser. That means the components that make up those visualization panes are added dynamically to the rendered DOM of the parsed markdown file temporarily. They are not persisted in the source code of the markdown file or any other configuration file. What is expressed in the source code of the markdown file is the creation of the pink data source window on file load. As a result, only that window is rendered after the file is reloaded (2). The application state that was configured previously is lost.

## 4.4 Collaborating in Lively4

To this point, we only discussed how a single user in a single Lively4 session could utilize the technologies Lively4 integrates. In this section, we look at how the wiki of Lively4 functions and how users can share changes on the Lively4 wiki with other collaborators. Since we were a team of seven people working synchronously in the

Lively4 wiki, we also cover how real-time collaboration is supported. In the end, we take a look at versioning locally and via GitHub.

#### 4.4.1 The Lively4 Wiki Workflow

We explain what principles lay behind a wiki in general, and how Lively4 integrates those to a wiki-like experience.

##### 4.4.1.1 Wiki Principles

In his pursuit to create a culture of ideation and knowledge-sharing in his company, Ward Cunningham defined what attributes a system -the world's first wiki-, that supports this process should have [20]. We name a few here and show how Lively4 implements them: *The wiki experience should be an open and organic process.* The system should not stop at presenting polished, sealed, and finished content, but allow manipulation by anyone who wishes to do that. Lively4 supports this by allowing anyone to edit a file in the Lively4 browser without any authorization. Everyone that knows the URL of the public Lively4 server and a Lively4 core system<sup>23</sup> can start to contribute. There are reasonable exceptions to this rule in Lively4, which are described in section 4.4.1.3. *It should be easy to use.* The rules and syntax to create text should work for non-programmers as well. The markdown syntax used in Lively4 was designed to support this goal. *Also, it should be incremental and linked.* That means pages in the wiki can cite and link to other pages. This way, the wiki can be explored, and connections between knowledge or ideas are drawn. Lively4 emphasizes the use of markdown links to achieve that.

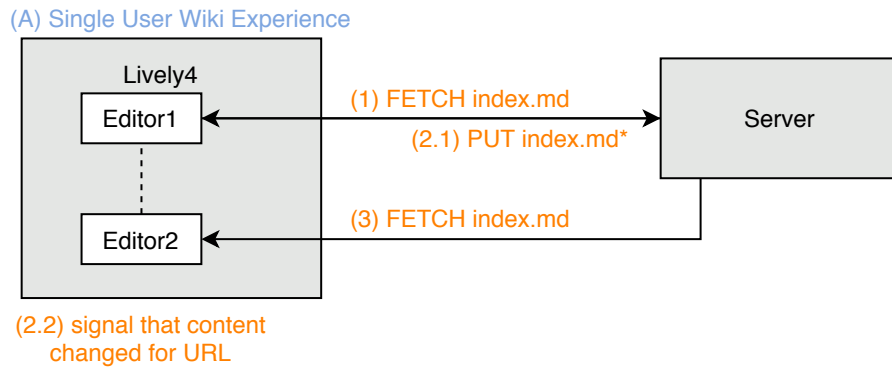
##### 4.4.1.2 Working in the Client Side Wiki

To understand how the client-side wiki experience is realized, consider the process in Figure 4.14 (A). Users, who have loaded a Lively4 session, want to edit the `index.md`<sup>24</sup> file in the tab-view documentation. With the Lively4 browser, they can navigate to the file by either clicking through the folder structure or inserting the files URL in the search bar. The Lively4 browser then fetches the file from the server and renders its contents (1). After switching to edit mode, the source code of the markdown file is now accessible in a text editor in the same Lively4 browser window as the rendered markdown before. Another Lively4 browser window can now be opened in view mode with the same `index.md` file by clicking on the Icon (Figure 4.15 (1)) next to the search bar. After making the desired changes to the file using the first Lively4 browser window, users can save the file (Figure 4.15 (2.1)). The saving causes the Lively4 browser to send a PUT request with the new file content to the server (Figure 4.14 (3.1)). Each Lively4 browser that currently displays the same

---

<sup>23</sup><https://lively-kernel.org/Lively4/Lively4-core/start.html?load=https://lively-kernel.org/Lively4/Lively4-core/doc/index.md> (last accessed 2020-07-30).

<sup>24</sup><https://lively-kernel.org/Lively4/BP2019RH1/all-in-one-product/index.md> (last accessed 2020-07-29).



**Figure 4.14:** Lively4 wiki workflow in one Lively4 session

URL then gets notified that the content for this URL has been changed since the last load (2.2). This notification causes especially the second Lively4 browser window with the same file in view mode to fetch the file again and replace the old content (Figure 4.14 (3), Figure 4.15 (2.2)). This workflow has two main benefits. The first is that users get direct feedback, how the changes affect the rendered result. They do not have to switch to another client machine browser tab or navigate to another URL. The result of the change is rendered and present right after the change was made on the client-side. The second benefit is that potential version conflicts are not possible by design when opening two Lively4 browser windows with the same file loaded. Due to the automatic re-fetch of the second one, when the first one makes changes, users can not edit an old version of the content of the file when switching to the second Lively4 browser window.

During the whole process, no credentials had to be used to authorize the users for any action.

#### 4.4.1.3 Working with Personal Data in a Wiki

To get to work with Africas Voices, we had to make sure to place the highly sensitive personal data that resulted from the work of Africas Voices (subsection 5.2.4) behind an access restriction mechanism, so that only authorized people could access it. Since Lively4 supports a very open and collaborative approach to access files, we had to take extra measures to meet the requirement of Africas Voices. We set up another Lively4 server next to the existing, open Lively4 server, which already serves the wiki and the Lively4 core system. The new server is called *voices*. The data of Africas Voices resides on this server. Because the server is a Lively4 server, we are

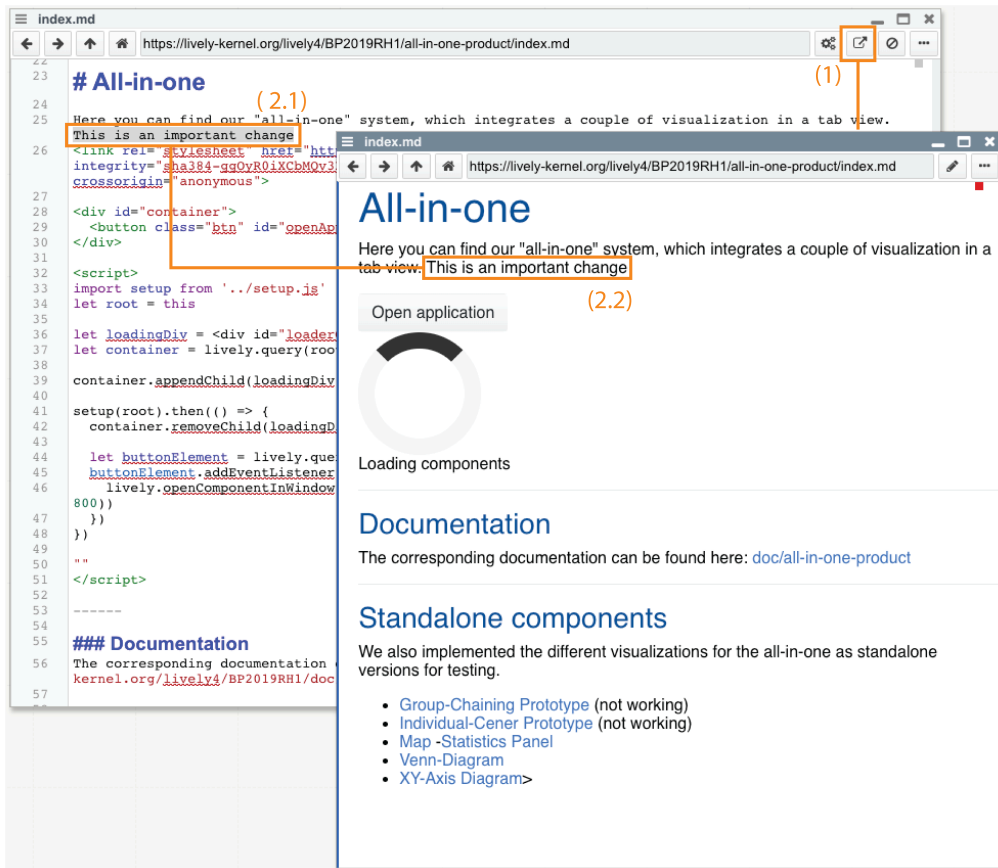


Figure 4.15: Automatic reloading of Lively4 browser windows that show the same URL



**Figure 4.16:** Forbidden query for a file on the private voices server from within a session started from the public Lively4 server

able to access it with a Lively4 browser from within the Lively4 client application Figure 4.16 (A), whose session was started from the Lively4 server (B).

The voices server has access restrictions implemented. To authenticate, we have to log in to GitHub via the GitHub sync tool, which is explained in section 4.4.3.2. Only if the GitHub account is a member of the authorized group, the access is granted. That is realized with a GitHub token, which is part of every request to the voices server.

To access the data in the open wiki space of the Lively4-server, we implemented a parser module. This parser module can be imported from the voices server. The API of the parser allows for standardized data requests. The following listing shows a typical usage of the parser module.

```

1 import { AVFParse } from "https://lively-kernel.org/voices/parsing
  -data/avf-parser.js"
2 //...
3 async _fetchKenyaData() {
4   let data = await AVFParse.loadInferredCovidData()
5   return data
6 }

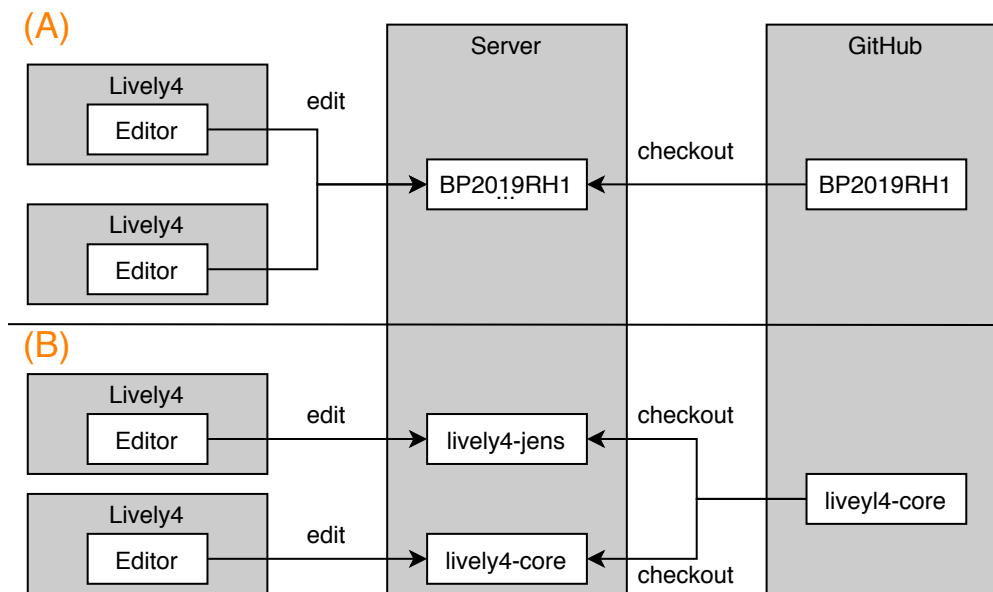
```

Within the `loadInferredCovidData()` function, the parser requests a secured URL of the voices server. The voices server checks for a valid authorization token and forwards the request to a separate node.js server, which responds with the respective chunk of JSON data from Africas Voices. In this scenario, the voices server acts like a reverse proxy<sup>25</sup> and takes care of blocking unauthorized requests. This way, it is ensured that only GitHub authenticated users can access data in the open wiki space of the Lively4-server.

#### 4.4.2 Working Collaboratively in Realtime

A wiki lives and only thrives when many individuals commit to using the tool or see the need to have a collaborative space where ideas can be written down, connections can be made, and knowledge can be shared. Lively4 offers two ways to collaborate: Working on the same checkout of a repository or working on different checkouts.

<sup>25</sup><https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/> (last accessed 2020-07-29).



**Figure 4.17:** The Lively4 one checkout and multiple checkout collaboration

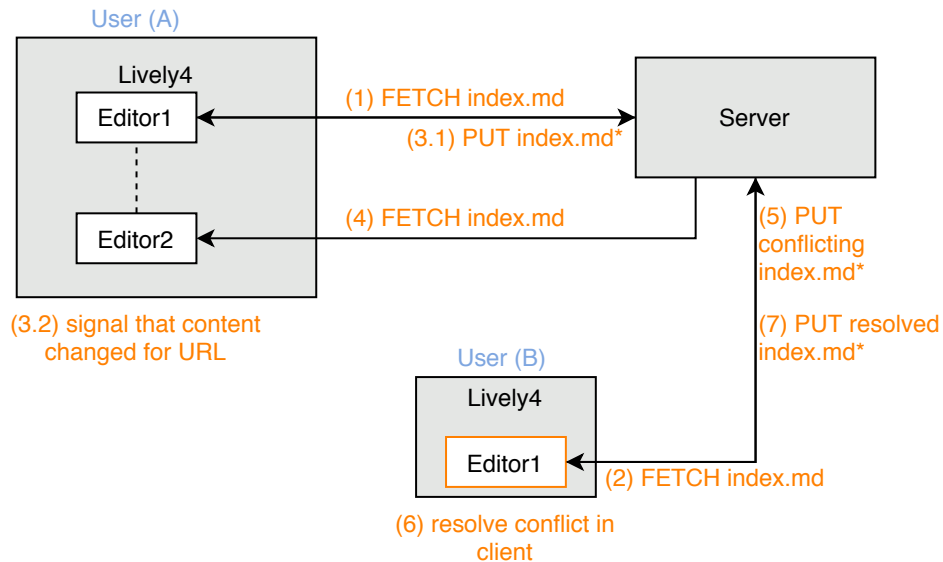
#### 4.4.2.1 Different Modes of Collaboration

Working on one checkout (Figure 4.17)(A) is described in the following. One checkout of a repository resides on the server. The users all work on the same file base. When a user edits a file, content gets saved on the server, and the change committed into the local repository on the server. As soon as another user loads the file with the Lively4 browser, the new content gets served. This approach comes with disadvantages and advantages. The advantage is that all changes can be seen immediately, and the feeling of real-time collaboration is fostered. The disadvantage results from the fact that Lively4 is a self-supporting system. The core functionality of the system itself can be edited as easy as a file in the wiki. This way, breaking changes at the core system that users introduce propagate to other users with a reload of the Lively4 client application. The disadvantage is mitigated by working with multiple checkouts (B): Multiple checkouts of one repository reside locally on the Lively4 server. Examples are the different checkouts of the Lively4 core system in the folders Lively4-core, or Lively4-jens. Users can work on crucial parts of the system on one checkout, without having the changes propagated to other users, who work on and whose Lively4 client application uses another checkout. Only after manually pushing changes to the remote origin, are they accessible to other users.

#### 4.4.2.2 Changing a File Simultaneously

In our wiki project folder BP2019RH1, we worked on one checkout. As a team of seven people working at the same office space, we had, on average, three to five Lively4 sessions running simultaneously. That increased the chance that two or more users were editing the same file at the same time. As this could result in one user overwriting changes of another, Lively4 implements a conflict resolution





**Figure 4.18:** Conflicting collaboration of two Lively4 users at realtime

mechanism. Consider the process of Figure 4.18 (A) with a second Lively4 session loaded by another user (B). The second user loads the same `index.md` file (2) after the first user. The first user saves an edited version (3). Then the second user tries to save another edited version with other changes (5). The server recognizes possibly conflicting versions of the file due to commit hashes and responds to the Lively4 client application of the second user with a changelog of both users. The client application of the second user then automatically merges the changes according to the [diff match patch<sup>26</sup> algorithm by character and issues another PUT request, to persist the merged content of the file (7).

### 4.4.3 Version Control in Lively4

The work of testing and prototyping, adapting, and re-adapting code and the corresponding documentation in a self-supporting system profits from a proper version control system and support of this system in the tooling. In the following, we talk about the Lively4 way of versioning.

#### 4.4.3.1 Versioning a Wiki

Versioning of files is heavily integrated and automated in Lively4 on the server-side and the side of the Lively4 client application. To understand versioning, we first have to look at the structure of the file system on the Lively4 server. Not the entire root directory of the Lively4 server is a single repository. Instead, the root directory is divided into several folders, for example, `Lively4-core`, `Lively4bp-2019`, or

<sup>26</sup><https://github.com/google/diff-match-patch> (last accessed 2020-07-30).

#### 4 Using the Lively4 Platform with Its Active Content Capabilities

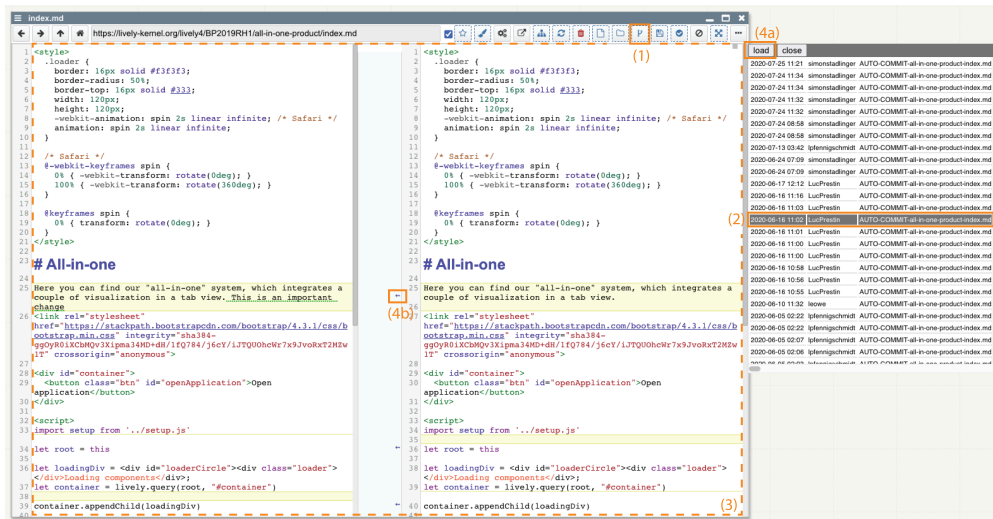


Figure 4.19: Comparing versions of ‘index.md’ in the Lively4 client application

BP2019RH1, where each folder has instantiated a separate local git repository. Some of the repositories are checkouts from a remote repository on GitHub, for example, our projects wiki folder BP2019RH1<sup>27</sup> or the Lively4-core<sup>28</sup> system folder.

Whenever a user edits a file and saves the changes via a PUT request to the server, the Lively4 server commits the changes automatically. An auto-commit is created and committed to the folder’s repository, which subtree contains the affected file. In the example of Figure 4.14, an auto-commit is generated in the BP2019RH1 repository.

The tight versioning of files has several advantages. The first one is that if users introduce breaking changes to the core system, it can be rolled back to the commit before the change happens. As the commits happen on a per change basis, the breaking change can be eliminated very fine-grained. The second advantage is that the history of changes can be looked up by everybody else. This public lookup increases accountability for the content one is publishing in the wiki. In addition to just viewing the history of a file, Lively4 offers the client-side manipulation of the current version with content from prior versions if some older content needs to be reintroduced to the file. This versioning feature can be seen in Figure 4.19. By clicking on the version icon in the “actions row” (1), the history of changes to the file, expressed through auto-commits representing single editing steps of users, can be viewed. The author of the auto-commit is listed next to the timestamp. The editor switches to a diffing view after selecting a version in the list (2) (3). Users then can either load the selected version to completely replace the current one (4a), or select specific changes from the older version to replace parts of the current version (4b).

<sup>27</sup><https://github.com/hpi-swa-lab/BP2019RH1> (last accessed 2020-07-21).

<sup>28</sup><https://github.com/LivelyKernel/Lively4-core> (last accessed 2020-07-22).

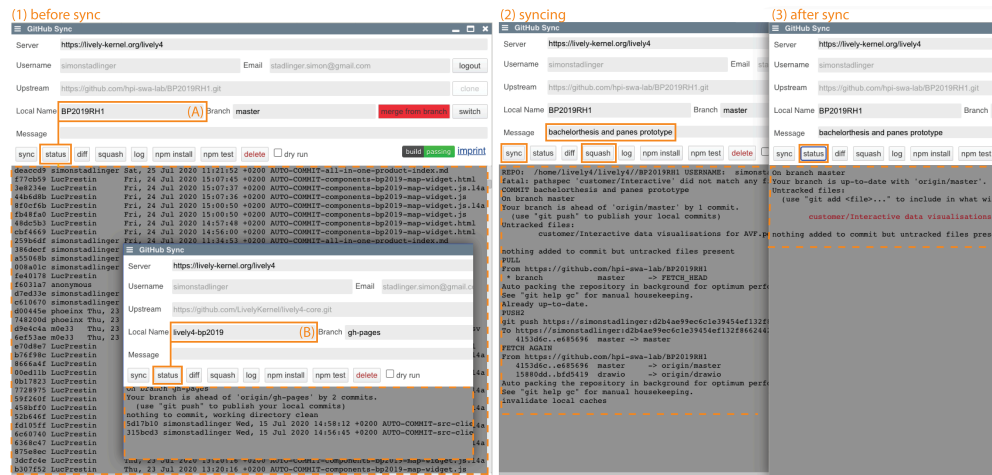


Figure 4.20: GitHub sync tool before, while and after syncing the BP2019RH1 repository with the remote origin

#### 4.4.3.2 Syncing to GitHub

Our project partner needed to be provided with a stable version of the prototypes while simultaneously extending the prototypes' functionality. As the work on the prototypes could introduce breaking changes, we set up another checkout of the BP2019RH1 on the Lively4 server. This checkout would only get synced with the origin when the prototypes were in a stable state.

For the sync process to work, the local development repository of the folder BP2019RH1 eventually needs to be pushed to the remote origin on GitHub. As the server holds the git repository, this operation needs to be conducted on the server. The GitHub Sync tool of the Lively4 client application offers a GUI for git operations on the server. It can be seen in Figure 4.20. The prerequisite for pushing to the remote origin is to be authenticated with authorized GitHub credentials. Within the sync tool, users can select the Lively4 server and the corresponding repository folder that we want to push (1). All operations that the sync tool offers are then executed on this folder's local repository on the server. The changelog in the selected repository can be inspected by clicking the status button (1A). As every folder is a separate repository, users can see different changelogs appearing in the git status, when selecting different repositories (1B). The commit history on the local repository can become cluttered with auto-commits from each file change. To keep the history of commits on the repository clean, users can squash all auto-commits into a single commit with a proper commit message from the sync tools GUI (2). With the sync button, the local repository on the server can be synced with the remote origin on GitHub.

## 4.5 Conclusion

During our project, we accomplished a variety of tasks. We collected ideas of novel visualizations at the beginning while in parallel examine and evaluate technologies to visualize data in the web browser. We then evaluated the ideas and concepts we came up with against technical feasibility and domain-specific fit. Therefore, we implemented many spikes. We had to present the prototypes and ideas to our project partner in a tangible way as interaction patterns needed to be tested live. As the last step, we had to engineer advanced prototypes that integrated many visualizations and interaction patterns into a single user experience. The Lively4 development environment made it possible to switch between those steps seamlessly. Lively4 is built around state-of-the-art web technologies. It has features that support rapid prototyping, collaboration in real-time, and advanced application engineering. Additionally, it comes with the possibility to build research-oriented, expressive documentation to structure a growing and fluctuant knowledge base as a team. It is advantageous for a design project to have all those capabilities combined in a single environment, and be able to switch between different stages of the project without changing the tooling.

# 5 Mapping of Data and UI for Interactive and Explorable Visualizations

Our goal was to develop visualizations that focus on the individual and not on aggregated data, avoiding losing track of the people's opinions, needs, and fears. Furthermore, the data and their provenance should be made explorable through the visualizations. We define explorability as the ability to interactively investigate data by comparing, connecting, and investigating their graphical representation.

In order to allow data to be explored starting from the visualization, there must be a relationship between the visualization elements and the data from which they are created that works in both directions. Such bidirectional mapping, however, is not trivial to achieve due to the loss of information that often occurs during the analysis of data due to aggregations.

In this chapter we discuss the meaning of provenance, explain the measures we took to ensure that only we can access the personal data, and evaluate the data formats in which we received the data. We explain the concept of bidirectional mapping and discuss the loss of information as a problem in its implementation. We evaluate three strategies for implementing bidirectional mapping of data and UI and explain how we used bidirectional mapping in our prototypes as well as how we dealt with information loss.

## 5.1 Introduction

Our goal was to develop visualizations that focus on the individual and not on aggregated data, avoiding losing track of the people's opinions, needs, and fears. Furthermore, the data and their provenance should be made explorable through the visualizations. We define explorability as the ability to interactively investigate data by comparing, connecting, and investigating their graphical representation.

The term "provenance" is heavily overloaded, which raises the question of what kind of provenance we need to work with. Moreover, opinions, needs, and fears in combination with the demographic data of individuals are personal data and must be stored and handled with appropriate care. In terms of visualization, the question is how provenance data should be handled, meaning to what point the origin of the data should be traceable. In order to allow data to be explored from the visualization, there must be a relationship between the visualization elements and the data from which they are created that works in both directions. Such bidirectional mapping, however, is not trivial to achieve due to the loss of information that is deliberately taken in data science.

To solve the problem of loss of information and to make data explorable through visualizations, we investigated the bidirectional mapping of data and UI. We defined what provenance in the context of our project means, secured the starting point of the mapping, the data, on a dedicated server, evaluated strategies to achieve bidirectional mapping, and implemented the most promising ones into our prototypes.

In section 5.2 we provide some background by discussing what provenance is, explaining the measures we took to ensure that only we can access the data, and evaluating the data formats in which Africa's Voices provided us with the data. In section 5.3 we explain the concept of bidirectional mapping and discuss the loss of information as a problem in its implementation. In section 5.4 we explain three strategies to achieve bidirectional mapping and evaluate which strategy is suitable for which application. In section 5.5 we explain the use of bidirectional mapping in our main prototypes and evaluate how we dealt with the problems of bidirectional mapping. A conclusion is presented in section 5.6.

## 5.2 Data and Provenance

This section is about data provenance and Africa's Voices Foundation's data used in this project. First, we define the term provenance in the context of our project. Then, we state how we were provided with data and show our measurements to keep the non-disclosure agreement we signed. After that, we present Africa's Voices' data formats to discuss how we dealt with changing data formats for our own individual-centered data format.

### 5.2.1 Data Provenance

Data provenance is a heavily overloaded term. In the context of web databases, it is used to describe the tracing of the origins of data and its movement between databases [13]. Also the term data lineage often means something similar. In the context of data warehousing, data lineage is described as tracing warehouse data items back to its original source [19]. Some work focuses on workflow provenance. In the context of curated databases, workflow provenance can be seen as tracing the users actions involved in constructing a curated database [12].

In our work, we used a more general definition of data provenance. According to Ikeda, "provenance, in its most general form describes where data came from, how it was derived, and how it was updated over time." [37]

At the beginning of our project, our task was to investigate how data and its provenance could be made explorable through visualizations. We received data that goes back to individual SMS from participants (section 5.2.4.1). But in the visualizations that we developed, we did not need this information, because they revolve around using the demographic data and themes. Therefore, we only needed the information as collected in our individual-centered data format (section 5.2.5).

As a result, in our context, provenance does not mean how the data was derived and updated. In our visualizations, the use of provenance is limited to the origin of

the data. It is still open whether the additional provenance data is useful for further visualizations.

### 5.2.2 Provision of the Data

Africa's Voices Foundation conducts radio shows where citizens can answer with an open text via SMS. These messages are then labeled with common themes that occur in the answers. The themes, raw messages, and demographic information that Africa's Voices collects via follow-up messages are then usually run through python scripts that produce CSV files that are used to generate the visualizations (See section 1.1.2.3: Visualizing).

We got three different data sets, whose compositions we state in section 5.2.4. What all data formats had in common is that they never contained the original messages that listeners of the radio shows answered to Africa's Voices. In the first data set, the messages were obfuscated by replacing them with quotes of literature. In the second data set, the messages were replaced with internal IDs. The third data set did not include any original messages, neither translated messages nor replacements. This meant the only way for us to do a qualitative analysis of the data set was by using the themes.

### 5.2.3 Non-disclosure Agreement

Everybody who works with Africa's Voices' data has to sign a non-disclosure agreement and the data must be kept private. To ensure that, we created a dedicated server for the data. The server accepts requests for the data sets and sends the data in our standardized format, which is explained in section 5.2.5. To ensure that only we can access the data, we used the authentication in lively via GitHub as explained in section 4.4.1.3. So when someone who is not part of our GitHub team tries to make a request, it gets denied.

### 5.2.4 Africa's Voices Foundation's Data Formats

Africa's Voices provided us with four data sets, coming from three different radio shows. Due to an internal restructuring of the infrastructure at Africa's Voices, these four data sets had four different formats. We describe these formats now to explain in 5.2.5 how we dealt with the changing data formats.

#### 5.2.4.1 Traced Data Objects

The first data set is a JSONL document. Different to normal JSON documents, JSONL files can be parsed line by line, since every line of the file contains a complete JSON object.<sup>1</sup> Each JSON-line in the data set is an array, corresponding to the path of an individual's data through Africa's Voices' Pipeline as stated in section 1.1.2.3. Each

---

<sup>1</sup>Ian Ward: <http://jsonlines.org/> (last accessed 2020-07-30).

element of the array, as shown in, consists of a data object, a SHA hash, metadata, and a NestedTracedData object. An example for such a data object is shown in Listing 5.1. The data object contains the last state of the respective datum. The metadata indicates which script last touched the datum and which analyst used the script and when. The NestedTracedData object is used if the same date is touched several times. The previous step is then stored in the NestedTracedData.

**Listing 5.1:** An example of a traced data object

```

1  [{
2    Data: {NC: "1"},
3    Metadata: {
4      Source: "analysis.py:205:generate",
5      Timestamp: "2019-09-25",
6      User: "john@doe.org"
7    },
8    NestedTracedData: {
9      folded_with: [{
10       Data: {NC : "MERGED"}
11     }]
12   },
13   SHA: "abcdefg"
14 },
15 // [...]
16 ]

```

#### 5.2.4.2 Conversation Objects

We call the second data set the *conversation objects*. This was an array of JSON-objects, with each object containing one individual that Africa’s Voices had a conversation with about the topic on the radio show. An example for such a conversation object is shown in Listing 5.2. The conversation itself is represented by the “messages” array. Each object in that array represents one message with the direction, whether it came from Africa’s Voices or the listeners, the date and tags, which stand for the demographic data or the themes from the previous data set.

**Listing 5.2:** An example of an individual in the first conversation object format

```

1  {
2    demographicsInfo: {},
3    messages: [{
4      datetime: "2020-03-23",
5      tags: ["tag-0", "tag-1"],
6      direction: "MessageDirection.in"
7    },
8    {
9      "datetime": "2020-03-23",
10     "tags": ["tag-2"]
11   }
12 // [...]

```



```

13 ],
14 notes: "",
15 tags: ["tag-0", "tag-1", "tag-2"],
16 unread: false,
17 deidentified_phone_number: "12345",
18 __id: "12345",
19 __reference_path: "conversations/12345",
20 __subcollections: []
21 }

```

However, we could not use that data set, because the mapping from the tags to the real demographic data and themes was missing. These information are crucial for our visualizations. Africa's Voices sent us the third data set, where the tags are replaced by the information they represent. The example object from above would then look like in Listing 5.3

**Listing 5.3:** An example of an individual in the second conversation object format

```

1 {
2   id: "12345",
3   start_date: "2020-03-23",
4   end_date: "2020-03-23",
5   tags: [
6     "other",
7     "male",
8     "opinion_on_govt_policy",
9   ]
10 }

```

### 5.2.4.3 CSV Table

The fourth data set was a CSV table. In this table, each row represents an individual and each demographic attribute and theme is represented by a column. An example for such a table is shown in Table 5.1. For the demographics, the value for that individual and attribute is stored. For the themes, it is stored whether that individual answered with a message that got tagged with that theme (1) or not (0).

uid	age	gender	district	language	idp status	theme 1	theme 2	...
0	35	male	a	missing	yes	0	0	...
1	44	female	b	swa	no	0	1	...
2	45	missing	b	en	missing	1	1	...
...	...	...	...	...	...	...	...	...

**Table 5.1:** An example CSV table

#### 5.2.4.4 Evaluation of Africa's Voices' Data Formats

In Table 5.2 four values per data set are shown: The size of the data set file, the size of the zipped data set file, the size of the data set file, when all formatting characters, like braces, whitespaces et cetera, are deleted and the number of individuals the data set contains. From this, two observations can be made.

Dataset	Size	Zip Size	Stripped Size	Number of Individuals
1	1.2 GB	162 MB	962 MB	7902
2	15 MB	1.3 MB	8.5 MB	2732
3	972 KB	162 KB	481 KB	2732
4	2.2 MB	317 KB	1.6 MB	9867

**Table 5.2:** Statistics of the Datasets

The first observation is that for each of the data formats, the formatting itself makes up a fair amount of the file size. In the case of the data sets we use, from 20% to 50%.

The second observation is the redundancy in the data. Building a ZIP archive uses the DEFLATE compression by default.<sup>2</sup> This algorithm reduces redundancy by writing a String only the first time it appears and replacing it with a pointer when it appears again [23]. Compressing the data as a ZIP file therefore resolves the redundancies in a file. The redundant data makes up between 83% and 91% of the data set. With a JSON this is not surprising, since it is an unstructured data format and so the keys are stored with the values and therefore redundant. Moreover, redundancy in the first data set is conditioned by the nature of the data set. Saving all steps of the Africa's Voices pipeline is redundancy by definition.

#### 5.2.5 Our Individual-centered Data Format

In order to develop visualizations, a stable data format is necessary because this way the visualization does not have to be modified with every new data format so that it can also handle the new format. Therefore, we have converted all data sets that Africa's Voices sent us into our own stable format. The conversion happens when booting the server. The server loads the data sets in our format into its cache so that they do not have to be recreated with every request unless explicitly requested.

A data set in our individual-centered data format is an array of JSON objects. Each of these JSON objects represents an individual with demographic information and themes. Such an individual looks like Listing 5.4.

<sup>2</sup><https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.3.5.TXT> (last accessed 2020-07-30).

**Listing 5.4:** An example object of an individual in our individual-centered data format

```

1 {
2   id: "abcdefg",
3   themes: {
4     L1: ["question"],
5     L2: ["missing"],
6     L3: ["how_to_treat", "symptoms"]
7   },
8   languages: ["(EN)"],
9   age: "35",
10  gender: "female",
11  constituency: "missing",
12  county: "nairobi",
13  consent_withdrawn: "false",
14  recently_displaced: "false"
15 }

```

Another aspect that came through forming the data into our own format is the compression of the data set. We were able to reduce the size of the first data set, which had the Traced Data Format, from 1.1 GB to 2.8 MB, because we did not use the provenance data, meaning when which script touched which datum, for our visualizations, and therefore did not have to integrate it into the Individual-Centered data format.

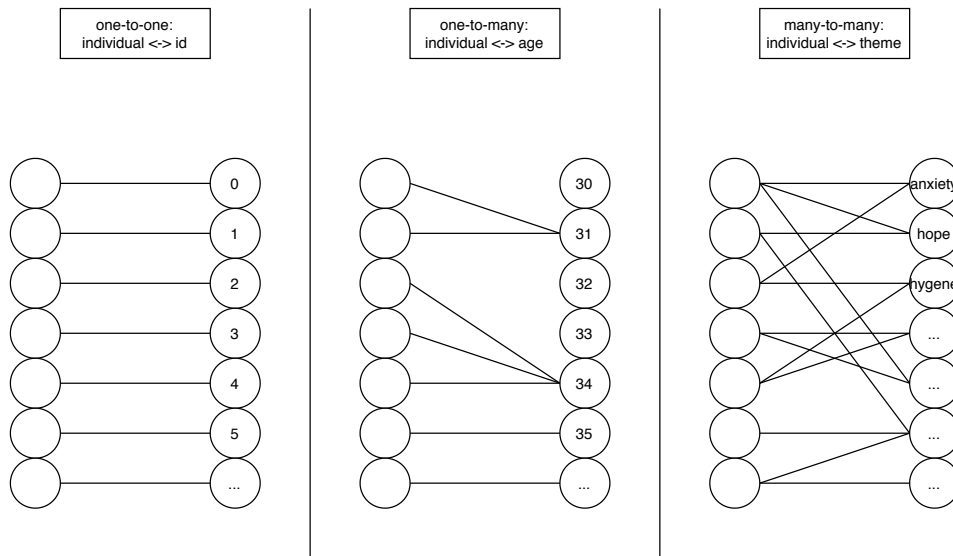
The second data set was in the Conversation Objects format. The 10.5 MB raw data was reduced to 972 KB when converted to our format because the assignment of individual messages to their tags was not important for our format.

The third data set was in CSV format. Here the data set was increased from 2.2 MB to 4.9 MB. This is mainly due to the conversion from CSV to JSON because the attribute is not only stored once in the top column as in CSV but in each individual object corresponding to a row in the CSV. This inflates the data set without adding any data.

In later sections, we describe the bidirectional mapping of data and UI elements. It is our individual-centered data format that we mean as the origin of the UI elements and thus the mapping back to data is the mapping back to the data in our individual-centered data format.

## 5.3 Bidirectional Mapping

In this section, we explain the term bidirectional mapping. First of all, we define what mapping and bidirectional mapping is. Next, we state the most common problems with bidirectional mapping. At last, we motivate the usage of bidirectional mapping in our software project.



**Figure 5.1:** An example of mapping as bipartite graphs with the restrictions of the appropriate relation type

### 5.3.1 Definition of Mapping

Mapping is a term that describes the relationship of elements of a set A to elements of a set B. Bidirectional mapping additionally has a relation of elements of set B back to elements of set A. These relations can be classified into four categories: one-to-one, one-to-many, many-to-one and many-to-many. As many-to-one and one-to-many are the same relations only reversed, we focus on the former in future considerations.

Every mapping can be described as an undirected, bipartite graph. The type of relation determines further restrictions on that graph as shown in Figure 5.1. In one-to-one relations, every vertex has a maximum of one edge. In many-to-one relations, every vertex of set A has a maximum of one edge and every vertex of set B can have multiple edges. Many-to-many relations do not have further restrictions. These are general bipartite graphs.

### 5.3.2 Common Problems with Bidirectional Mapping

The main problem with bidirectional mapping is missing data. This is either caused by data that is non-existent in the first place or it comes from applying data science. In that field, data is often aggregated to gain insights. For example, calculating an average or counting data points to create a bar chart are aggregations. The problem that comes with that is the loss of information. In aggregations, the data is actively reduced to a more condensed form. The downside of this is that it becomes hard to figure out which data went into the aggregated value. But that is what is necessary to know in order to map the output back into the input.

### 5.3.3 What We Need Bidirectional Mapping For

When visualizations are created, the underlying data gets mapped to graphical elements. Our task was not to build visualizations, but visualizations that allow to explore the data. The exploration of the data is achieved by interacting with the visualizations. In order to do so, the graphical elements of the visualizations need to be mapped back to the data they were build from. Together, these two mappings, from data to the visualizations and back, are a bidirectional mapping.

## 5.4 Evaluation of Existing Strategies for Bidirectional Mapping of Data and User Interface

In this section, first, we will give an example scenario as a use case for bidirectional mapping. We will then state three strategies that can be used to implement a bidirectional mapping of data and user interface. The three strategies are:

1. Tracing each action in a graph structure
2. Render images with unique colors
3. Match click position with object positions

One more strategy worth mentioning is a functional approach. Therefore one uses the concept of functions and inverse functions. With the name *lenses*, this has already been done, solving the update-view-problem where code gets mapped to the user interface it produces in a bidirectional way [27]. In theory, it should not be that different whether you map code or data to the user interface. But as we did not try it, we will not discuss that strategy.

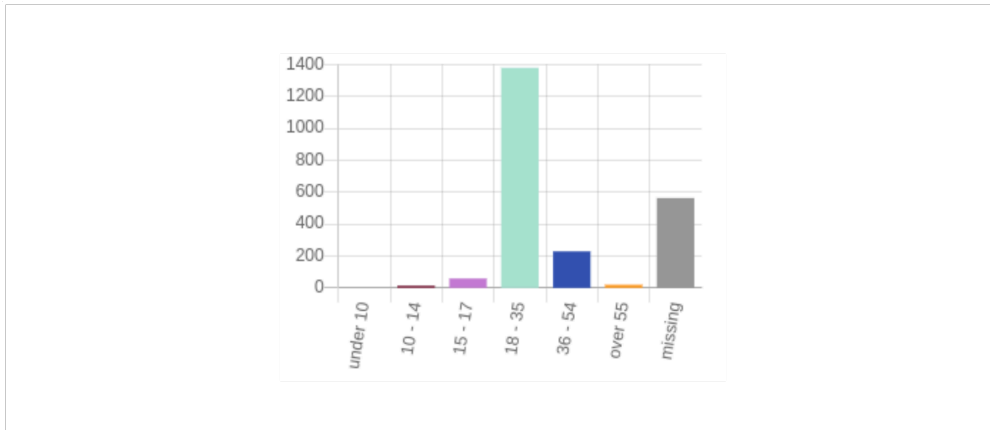
For each of the strategies will we state the idea and evaluate their presets and limitations. Then we will show, how the strategy can be used to implement the example scenario.

### 5.4.1 Scenario

To make the evaluation of mapping strategies more descriptive, we will give an example of how the strategy can be used. To do that, we will apply the strategy to the following scenario.

The basis for the scenario is the data set. This is similar to the individual-centered data set from section 5.2.5. However, fewer attributes are required for this scenario, so that an individual only has the following attributes:

```
1 {  
2   id: "0123456789",  
3   age: 35,  
4   county: "nairobi",  
5   constituency: "missing",  
6   gender: "female"  
7 }
```



**Figure 5.2:** An example bar chart of the age distribution in the age buckets [ $<10$ , 10-14, 15-17, 18-35, 36-54, over 55, missing]

First, a bar chart showing the age distribution, clustered in the age buckets [ $<10$ , 10-14, 15-17, 18-35, 36-54, over 55, missing] is generated. Depending on the data set such a bar chart could look like in Figure 5.2.

Then two individuals are selected: One from the group 18-35 and one from the group 36-54 to compare two sample individuals of different age groups. These individuals are recognizable so that they could be selected in other visualizations.

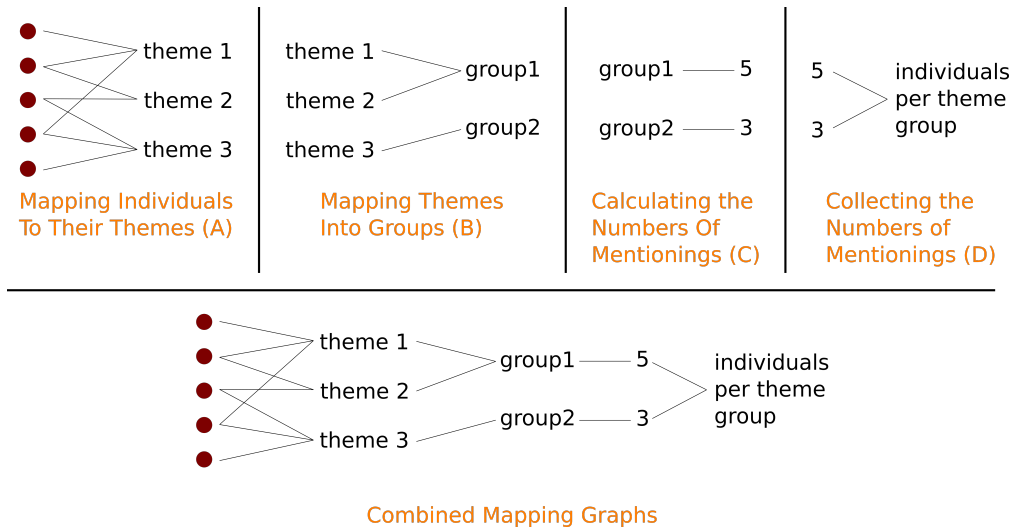
#### 5.4.2 Tracing Graph

Each operation on data resembles a mapping from source data to result data. The idea is to identify the operations mapping type from section 5.3 and view them as the corresponding graph as shown in Figure 5.1. These graphs can then be concatenated to one graph that resembles all operations that were done with the data. An example of that is shown in Figure 5.3.

Operation (A) is the mapping of individuals to the themes. Due to the nature of the themes, this is a many-to-many mapping. Operation (B) is the mapping of the themes into theme groups and therefore a many-to-one mapping. Operation (C) is the mapping of the theme groups to the number of its themes mentioned by individuals, a one-to-one mapping. Operation (D) is just the collection of all the numbers resulting from Operation (C). The concatenation then results in the graph on the right side.

##### 5.4.2.1 Evaluation

The Tracing Graph Strategy is about recording all operations that were performed on the base data. This makes the strategy very powerful. Not just two endpoints can be mapped bidirectionally, but also all intermediate steps. Therefore this strategy allows to trace every state the data has undergone.



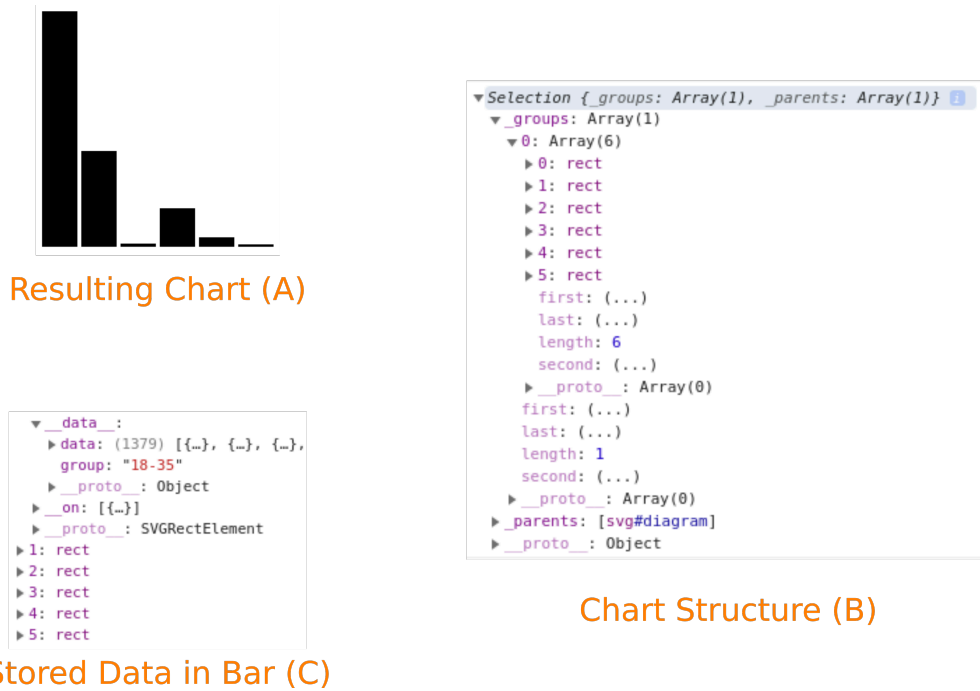
**Figure 5.3:** An example of building a tracing graph. The upper half displays the operations' single mappings. The lower half displays the concatenated mapping graph.

However, storing all intermediate steps is also the major disadvantage of this strategy, because it consumes a lot of memory. This makes it unsuitable for low-memory environments.

In the context of visualizations this strategy is also applicable, because mapping to graphical elements is only another operation on the data. For example, the number of elements could be extracted from arrays of base elements, which in turn would be used to generate an array of pixels. Each pixel has a color and a position on the screen. This information can then be used to render an image. In reverse mapping, the color or position can be used to identify the desired pixel array (see sections 5.4.3 and 5.4.4). The pixel array can then be traced back to the base elements.

Systems like HTML and JavaScript encapsulate the last steps from the number of elements to pixels and thus make it possible to handle the graphic elements in the object space of the programming environment. This makes it easier for programmers to build the mapping from and to graphical elements. HTML data attributes<sup>3</sup> allow data to be stored in graphical objects. Similarly, the data from which the element was generated can be stored there, so that the mapping fits well into the tracing graph strategy. At the same time, the encapsulation of the path from graphic objects to pixels also means that it cannot be adjusted manually. In the context of our work, however, we did not need this and therefore did not pursue this direction.

<sup>3</sup>[https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use\\_data\\_attributes](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes) (last accessed 2020-07-30).



**Figure 5.4:** Resulting bar chart (A) displayed with the diagram structure (B) and the stored data per bar (C)

#### 5.4.2.2 Application to the Scenario

The scenario as described in section 5.4.1 is to build a bar chart of the age distribution in age buckets of a data set of individuals. After that, two individuals from different bars are selected to compare them.

The code for this can be found in the appendix as Listing B.6. The first operation is to group the data by age buckets. After that, we use d3 to build a bar chart with one bar per age group. The resulting bar chart is shown in Figure 5.4 (A).

The bars d3 generates are HTML elements. These elements are then used to store the data in the corresponding bars (C). The data objects themselves store the individuals due to the grouping. The chart element stores all its bars (B). Therefore, the bar chart resembles a tree structure, that can be traced back to the individuals it is made of. This is what we use in the click handler. When a bar is clicked, the stored groups is sent to an Individual Inspector, in which an individual can be selected. When the second bar is clicked, the same happens with the corresponding group. By placing the Individual Inspectors next to each other, the two individuals from different age groups can be compared.

#### 5.4.3 Double Rendering

One technique to achieve bidirectional mapping of data and UI elements is double rendering. The double rendering process, shown in Figure 5.5, is divided into two phases: the preparation phase (steps 1-5) and the execution phase (steps 6-9).



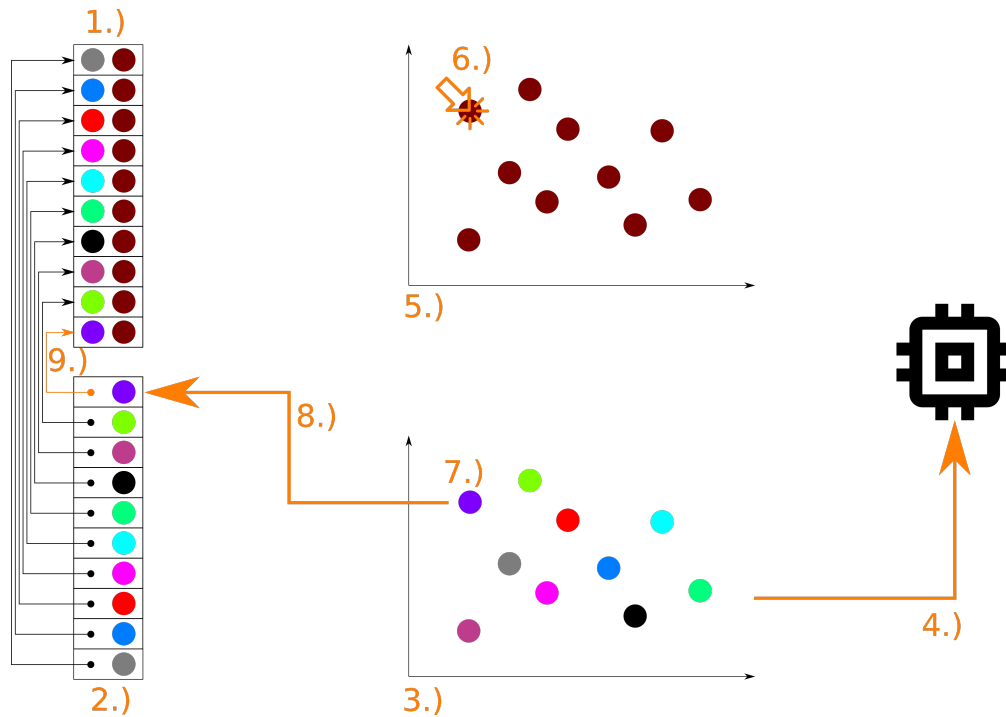


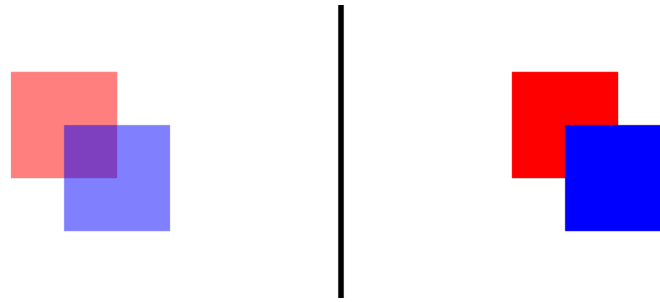
Figure 5.5: The double rendering process

In the preparation phase, the data is first prepared by assigning two colors to each data element. A unique color that is different from all other data elements and a color that is intended to be seen by users (step 1). A lookup map is then created, in which a reference to the data object behind each unique color is stored (step 2). Now the graphical representations of the data elements are rendered with the unique color (step 3). This image is then stored in the main memory (step 4). As the last step of the preparation phase, the graphical representations are rendered with the colors intended for users (step 5).

The execution phase starts as soon as users click on the canvas (step 6). Here the mouse position is queried to find the unique color in the saved image (step 7). If the color at the click-coordinate does not correspond to the background, the system searches for the corresponding entry in the lookup map (step 8). Finally, the reference in the lookup map to the actual data element is followed (step 9).

#### 5.4.3.1 Prerequisites

Double rendering is not a general-purpose strategy for bidirectional mapping. Mapping always means to map elements of two sets of elements to each other. To use the double rendering strategy, one set of these elements must consist of graphical elements. Otherwise, there is nothing to render and thus nothing to render twice. The second prerequisite is the need for rendering. Systems like HTML, except the canvas element, take rendering off programmers to allow faster development of user interfaces. This means programmers cannot render graphical elements themselves



**Figure 5.6:** Two graphical elements overlapping: Once with 50% opacity (left), once with 100% opacity (right)

and thus cannot use double rendering. So a system that allows custom rendering of graphics is a prerequisite of this strategy.

#### **5.4.3.2 Limitations**

The double rendering strategy is limited by the used color space. For example, the HTML CanvasRenderingContext2D<sup>4</sup> uses the RGBA color space with values from 0 to 255 for each color channel. Using different opacities would result in mixing colors when graphical elements overlap as shown on the left side of Figure 5.6. This is why only the channels red, green, and blue are usable. This limits the number of recognizable elements. In the context of our project, this is sufficient, but in other contexts, this could be a factor that makes this strategy unusable as it limits the total amount of renderable unique objects.

Another limitation of this strategy is the size of the canvas. If the combined area of the graphical elements to be drawn is larger than the canvas, at least two graphical elements must overlap. But since a pixel can only have one color and, as mentioned above, must be drawn with full opacity, where two graphical elements overlap, only one is drawn as shown on the right side of Figure 5.6. As a result, if a click is made in the intersection of the overlapping graphical elements, only one of the graphical elements can be recognized. If there are many graphical elements, this can lead to the fact that some graphical elements are completely covered by others and therefore cannot be traced back to the corresponding data element.

#### **5.4.3.3 Evaluation**

The slow part of this process is rendering. The lookup of whose data element's graphical representation was clicked (Figure 5.5, steps 6-9), is, when using the right data structures, independent of how many data objects are used. It happens in memory, so no I/O is needed, and therefore can be seen as a constant time operation. The rendering however can not. The more data elements are drawn, the longer this process takes when doing this on a CPU. This correlation is also reflected

<sup>4</sup><https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D> (last accessed 2020-07-30).

in our benchmarks in section 6.5.3. When doing the rendering on a GPU, this is massively parallelized, but it takes I/O to transfer the data to the GPU, which in turn is dependent on how many objects must be transferred. This means, in any case, the rendering is the expensive part of this technique, and doing it twice is even more expensive. Therefore we have to optimize, how often an image gets rendered twice.

There are two possible times when double rendering can be performed. The first is whenever the shown picture changes, for example, because the data elements have been regrouped. This means that the saved image with unique colors is always up to date. Secondly, only when the canvas is clicked, so that the image with the custom colors is only built when it is really needed. Both variants have advantages and disadvantages.

The advantage of rendering the image twice whenever it changes is that the image with the unique colors is always up to date. This means that as long as the image does not change, any number of clicks can be made without the need for costly double rendering.

The advantage of rendering the image twice with the click is that this is only done when it is really needed so that when the image changes without interaction, the expensive double rendering does not have to be done.

In terms of performance, a combination of the two approaches is best. For this purpose, a flag is used to indicate whether the stored image with the unique colors is still up to date. This way the image only has to be rendered twice if a click is executed and the saved image is not up to date. This combines the advantages of being up to date for clicks on the same images, as well as only rendering twice when the image is clicked.

#### 5.4.3.4 Application to the Scenario

The scenario as described in section 5.4.1 is to build a bar chart of the age distribution in age buckets of a data set of individuals. After that, two individuals from different bars are selected to compare them.

Since the code is too large to be shown here, only the rough sequence of the preparation phase and the click handling are shown. The rest of the code can be seen in the appendix distributed over the files Listing B.2 and Listing B.3.

The preparation phase is shown in Listing 5.5. After preparing the canvas, the data is loaded. As the true mapping is between the age buckets and bars, the data needs to be grouped. Here, the two colors from step 1 in Figure 5.5 are assigned to the groups too. Line 11 corresponds to step 2 in the process. Then step 3, rendering with the unique colors (line 14) and step 4, saving the image are performed. The last step of the preparation phase is the rendering with the colors intended for users (step 5, line 18).

**Listing 5.5:** The preparation phase of the double rendering process

```
1 (async () => {  
2   //canvas preparation  
3   let diagram = lively.query(this, "#diagram")  
4   context = diagram.getContext('2d')
```

```

5   canvasDimensions = {width: 300, height: 300}
6
7   //data preparation
8   let data = await AVFParse.loadCovidData()
9   removeUnneededData(data)
10  groups = createColoredAgeGroups(data)
11  colorMapping = createColorMapping(groups)
12
13  //drawing with identifying colors and store the image
14  drawBars(context, canvasDimensions, groups, "uniqueColor")
15  identifyingImageData = context.getImageData(0, 0,
16      canvasDimensions.width, canvasDimensions.height)
17
18  //drawing with colors meant for users
19  drawBars(context, canvasDimensions, groups, "color")
20
21  //waiting for a click
22  diagram.addEventListener("click", event => {
23      lively.openInspector(getClickedGroup(event))
24  })

```

Now we have to wait for the click event. The corresponding event listener was added to the canvas in lines 21-23. Once the canvas has been clicked, the unique color at the click position is extracted from the saved image (step 7, lines 2-6 in Listing 5.6). Then the clicked group is traced back via the saved map (steps 8, 9, lines 7, 8 in Listing 5.6).

**Listing 5.6:** The execution phase of the double rendering process

```

1  function getClickedGroup(event) {
2      let colorString = getColorStringFromImageData(
3          identifyingImageData,
4          event.layerX,
5          event.layerY
6      )
7      let groupKey = colorMapping[colorString]
8      let group = groups[groupKey]
9
10     return group
11 }

```

Now the original group is found and can be loaded into an Individual Inspector. Here, the selection of an individual from group 18-35 as required by the scenario is possible. As soon as the bar with the individuals from the age group 36-54 is clicked, an Individual Inspector from the corresponding age group opens and a second individual can be selected. The two Individual Inspectors can then be placed next to each other to compare the individuals.

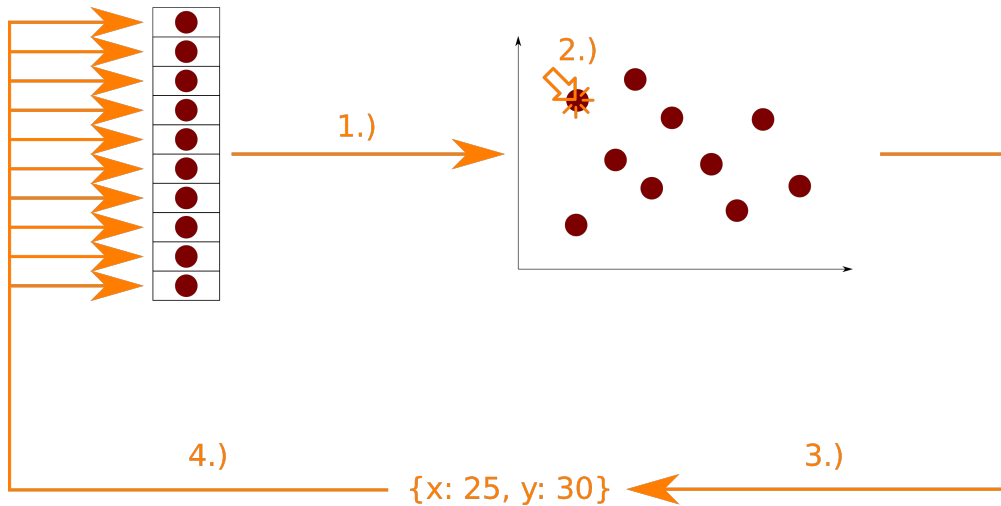


Figure 5.7: The position matching process

#### 5.4.4 Position Matching

One technique to achieve bidirectional mapping of data and UI elements is position matching. The process shown in Figure 5.7 has four steps.

At first, the image is rendered (step 1). When users click on the image (step 2), the mouse coordinate is requested (step 3). Now for each data element, it is checked whether the click occurred within the coordinates of the graphical representation (step 4). In the case of circles as the representation, for example, the Euclidean distance between the click and the center of the circle can be calculated. If it is smaller than the radius of the circle, the click took place within the graphical representation and the corresponding data element is found.

##### 5.4.4.1 Prerequisites

Position matching is not a general-purpose strategy for bidirectional mapping. One prerequisite of this strategy is, that one of the sets of elements to be mapped must be graphical elements. Otherwise, nothing can be rendered and no click coordinate can be determined which is needed for the position matching. The other prerequisite is the data structure. To use this strategy, the data objects must store the coordinates and bounds of the graphical representation.

##### 5.4.4.2 Evaluation

Since it must be checked for each data element whether the click took place within the boundaries of the graphical representation, the runtime of this strategy is linearly dependent on the number of data elements and therefore scales poorly.

An advantage of this strategy is the independence from the rendered image. Since the position and size of the graphical representation is stored in each data element, all objects can be identified even if the click occurred at a position where the representations of several graphical elements overlap.

#### 5.4.4.3 Application to the Scenario

The scenario as described in section 5.4.1 is to build a bar chart of the age distribution in age buckets of a data set of individuals. After that, two individuals from different bars are selected to compare them.

Since the code is too large to be shown here, only the rough sequence of the preparation phase and the click handling are shown. The rest of the code can be seen in the appendix distributed over the files Listing B.4 and Listing B.5

The preparation phase is shown in Listing 5.7. After preparing the canvas, the data is loaded. As the true mapping is between the age buckets and bars, the data needs to be grouped. Here, the position and bounds of the bars are stored in the groups too.

**Listing 5.7:** The preparation phase of the position matching process

```

1 (async () => {
2   //canvas preparation
3   let diagram = lively.query(this, "#diagram")
4   context = diagram.getContext('2d')
5   canvasDimensions = {width: 300, height: 300}
6
7   //data preparation
8   let data = await AVFParser.loadCovidData()
9   removeUnneededData(data)
10  groups = createPositionedAgeGroups(data, canvasDimensions)
11
12  //rendering the image
13  drawBars(context, canvasDimensions, groups)
14
15  //waiting for a click
16  diagram.addEventListener("click", event => {
17    lively.openInspector(getClickedGroup(event))
18  })
19 })()

```

When users click the canvas, for every group is checked, whether the click happened within the stored bounds (step 4, lines 6-10 in Listing 5.8).

**Listing 5.8:** The execution phase of the position matching process

```

1 function getClickedGroup(event) {
2   let groupArray = Object.keys(groups).map(key => groups[key])
3   let position = {x: event.layerX, y: event.layerY}
4
5   let results = []
6   groupArray.forEach(group => {
7     if (positionMatchesGroup(position, group)) {
8       results.push(group)
9     }
10  })
11 }

```

```

12     return results
13 }

```

Now the original group is found and can be loaded into an Individual Inspector. Here, the selection of an individual from group 18-35 as required by the scenario is possible. As soon as the bar with the individuals from the age group 36-54 is clicked, an Individual Inspector from the corresponding age group opens and a second individual can be selected. The two Individual Inspectors can then be placed next to each other to compare the individuals.

### 5.4.5 Evaluation

Each of the presented strategies has its own advantages and disadvantages. This raises the question of when which of the strategies should be used.

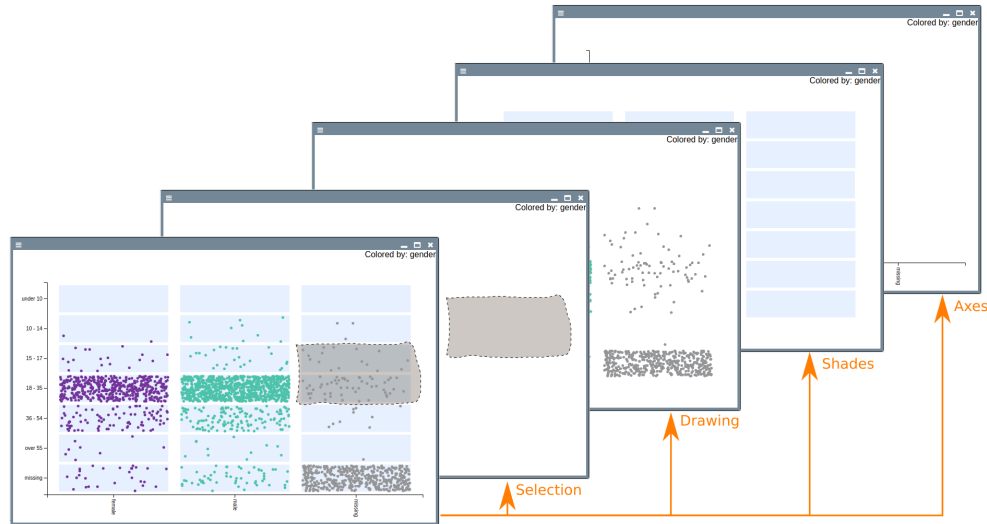
The advantage of the tracing graph is its powerfulness. In contrast to position-matching and double-rendering, not only two endpoints of an operation are mapped together. Since every intermediate step is written down, every partial operation can be traced later on. Double-rendering and position matching are special cases that only work for mapping from and to graphical elements. So if a transformation of the data should be carried out that has nothing to do with graphics, but should still be comprehensible, of the presented strategies only the tracing graph is functional.

For graphical elements, on the other hand, it is necessary to weigh up between position matching and double rendering. Position matching is generally easier to implement, since it requires only a small preparation phase, namely the calculation of the positions and bounds of the graphical elements. For this purpose, the entire data set must be processed for each interaction. The longer preparation phase of double-rendering, on the other hand, leads to a faster lookup when interacting with the visualization. It also depends on the arrangement of the graphical elements. If these overlap, but all overlapping elements are still visible, double-rendering cannot be used.

For us, a combination of both approaches worked best. The operations on the data level, aggregating, grouping et cetera were logged in JavaScript objects. In the visualizations that use HTML as graphical elements, we logged even further by attaching the data to their graphical representations and holding a reference to the graphical Elements. Whenever we could not use HTML elements, but had to use a pixel based graphic, we used either the position matching or the double rendering method to do the last steps of the mapping.

## 5.5 Our Approach on Bidirectional Mapping

Our main visualizations are the *XY Diagram*, the *Map* prototype, the *Venn Diagram*, and the *Statistics Panel*. In this section, we state how we used bidirectional mapping in these visualizations to achieve explorability of the data. For each of these visualizations, we show how we draw the elements on their canvas and how to get



**Figure 5.8:** The XY Diagram separated in its four parts

back to the data when the user interacts with them in a certain way. Furthermore, we evaluate our approaches and discuss how we dealt with the problem of aggregated data.

### 5.5.1 XY Diagram

This visualization consists of four parts as shown in Figure 5.8. There are three SVG elements that we use for the axes, the grouping shades, and the freehand selection polygon (section 3.3.5) respectively. The dots representing individuals are drawn on the fourth part, a canvas. This structure came from two design choices. The first was the separation into four parts. As a consequence of this decision, we had to make the second one, which is for which part to use which HTML element: a `div`, an SVG, or a canvas.

The first decision, the separation of the visualization into four parts, was made to have a separation of concerns. One element should only be used for one purpose. This makes it easier to track bugs, as a bug in the grouping shades cannot come from drawing the freehand selection for example.

The second decision of HTML elements means weighing speed, ease of usage, and flexibility against each other. To draw the grouping shades and scales the speed of SVG and `div` are sufficient, as these are few, large graphical objects. Therefore we do not use a canvas as that would require rendering them by hand. The final decision was to choose an SVG because drawing polygons, as we need it for the freehand selection, is easier in a vector graphic than on `div`s. To draw the individuals, we decided to use a canvas element. Our biggest data set has approximately 10,000 individuals and we wanted one graphical representation per individual. Appending one HTML element per individual resulted in a lack of performance.



Our benchmarks, whose results can be found in section 6.5.3, showed, canvases are the right choice to render many elements. Although that meant that we could not use HTML child relationships to identify the individuals when we interact with the visualization. For that, we had to come up with something new.

#### 5.5.1.1 Click Interaction

We wanted our visualizations to allow users to further inspect an individual when they click on its graphical representation. By using HTML elements as a graphic representation for individuals, that would have been easy. We could have used HTML5 data attributes<sup>5</sup> to store each individual to their dot. This way, we could use a click event listener on the dot to read the stored individual and inspect it. Unfortunately, this was not possible due to the aforementioned performance issues. The canvas element, that we had to use, does not keep track of what domain element is drawn on it, it only handles pixels. This means that the click event does not go to a graphical representation of an individual but to the canvas.

To deal with this problem we used the double rendering strategy as explained in section 5.4.3. This means rendering the canvas first with one unique color per individual, saving the image data of the canvas, and then drawing the canvas again in the way users are supposed to see it. When users now click the canvas, the click position can be used to get the color at that position. With that color, the one individual that has that color can be determined. This way we ensured a bidirectional mapping from pixels to data points.

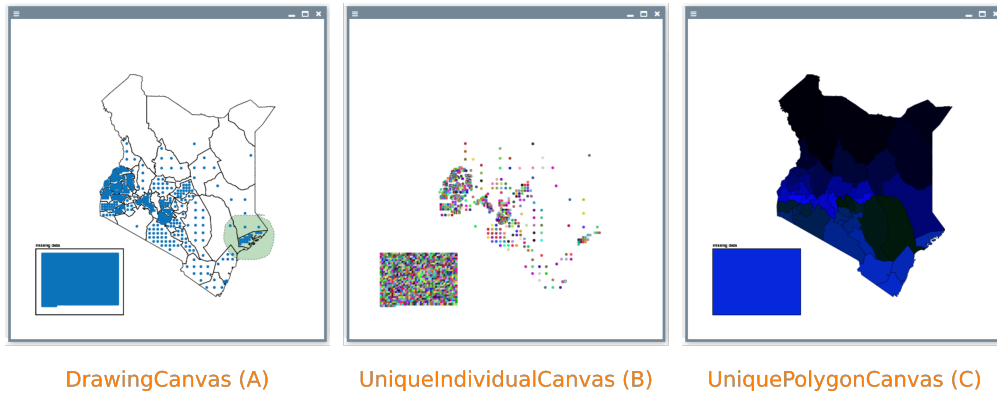
The individual is then put into an `InspectAction` which is used as a common interface for all our visualizations to handle a click on an individual. This is especially valuable in the *Tree View* prototype, as we want the inspections to be consistent across all connected visualizations. Therefore the visualization in which the click event happened notifies its pane about the inspected individual via an event, which in turn notifies the whole tree.

#### 5.5.1.2 Free-Hand Selection Interaction

For the *Tree View* prototype, we wanted our visualizations to allow a lasso select. The selected individuals could then be used for new visualizations. This would allow users to look into a specific group of people. To support this, we built the `FreehandDrawer`, a tool that draws a polygon on mouse drags onto the canvas. When a right-click is made on a drawn selection, the `FreehandDrawer` tells the visualization, where the selection polygon is. We then use the npm library `point-in-polygon`<sup>6</sup> to check for each individual whether its current position is in the polygon or not. The current position is the center of the dot that represents the individual. So if a dot is within the selection it is probable that the center of the dot is inside the selection too. We might lose a few points or get some more, but all in all, that is an acceptable deviation for the use case of getting an overview of

<sup>5</sup>[https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use\\_data\\_attributes](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes) (last accessed 2020-07-30).

<sup>6</sup><https://www.npmjs.com/package/point-in-polygon> (last accessed 2020-07-30).



**Figure 5.9:** Screenshots of the three main canvases of a map

a certain subgroup of individuals. This way, with a different technique than before, we ensured bidirectional mapping from an area of pixels to data points.

### 5.5.2 Map Prototype

This visualization consists of four parts. Three of them are shown in Figure 5.9. The part not displayed is the SVG for the selection polygon. It works the same way as described in section 5.5.1.

The goal of this visualization is to show the distribution of individuals based on geographic position. Furthermore, it allows interactions based on a geographic location and the comparison of distributions, for example between rural and urban areas. Therefore two different parts have to be identifiable: the individuals and the geographic structure of the country. To identify these, we use the double rendering strategy as described in section 5.4.3. The separation into three canvases results from that decision. The first canvas is meant to be seen by users (A). The second one (B) provides the uniquely colored individuals and the third (C) provides the uniquely colored regions.

#### 5.5.2.1 Click Interaction

We wanted our visualizations to allow users to further inspect an individual when they click on its graphical representation. This visualization uses a canvas element as its drawing canvas for the same reasons as the *XY Diagram* (section 5.5.1.1) and thus results in the problem that a canvas only handles pixels too. To deal with this problem, we use the double rendering strategy. The difference between the *Map* prototype and the *XY Diagram* is, that the *XY Diagram* generates the uniquely colored individuals when the canvas is clicked and the *Map* prototype when the displayed image changes. The *Map* prototype changes the layout of the individuals only when the individuals are filtered. Therefore the same uniquely colored individual canvas can be used for every click until the new layout is made. This reduces the number of times the unique image has to be rendered and thus saves time.

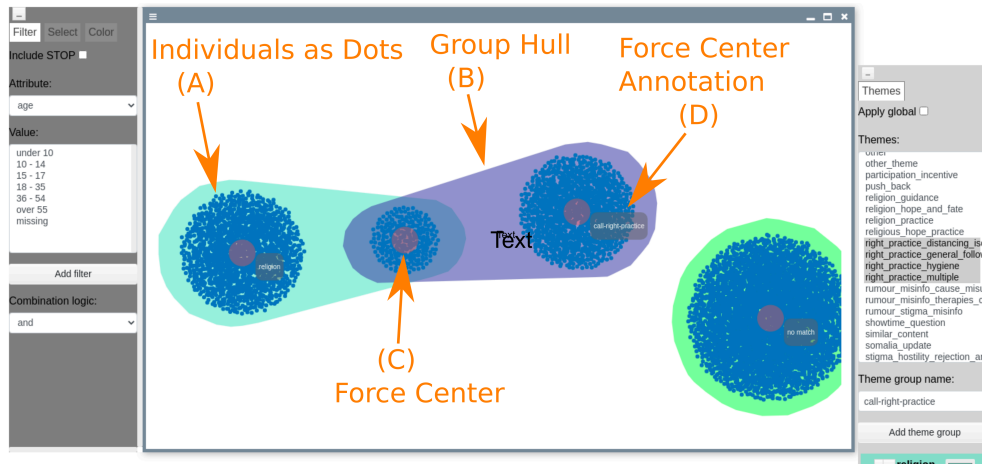


Figure 5.10: Screenshot of the Venn prototype with two theme groups

### 5.5.2.2 Hovering Interaction

We wanted the map to display information on the geographical region when users hover over it. During the initialization of the visualization, a mapping from geographical region to individuals that came from that region is generated. This way, only the district needs to be found from its graphical representation, when users hover over it. For this, we use the double rendering strategy again. For the unique colored region, a separate canvas is used. This way the hovering works even when a click is made or when users hover over individuals.

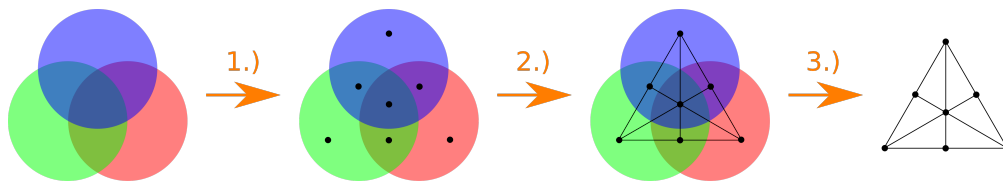
### 5.5.2.3 Free-Hand Selection Interaction

In this visualization, we allow a lasso select, too. It is implemented like in the *XY Diagram* (section 5.5.1.2).

## 5.5.3 Venn Diagram

The web component of this visualization consists of three parts. The first is the canvas on which the visualization is drawn. The second is a canvas for drawing the freehand selection that allows users to select individuals. It is stated later, why this is separate and not on one canvas, as in the *XY Diagram* or the *Map*. The third part is the freehand selection SVG which is used to draw the selection polygon on when users finish the drawing of the selection.

As stated in section 3.4.3, this visualization is designed to group individuals according to their themes. The visualization focuses on the overlapping of different themes and theme groups. To implement this concept, the visualization consists of four elements (Figure 5.10). Dots represent the individuals (A). The theme groups are represented by the so-called grouping hulls (B). These are drawn around all individuals who have at least one of the themes of the respective theme group. If individuals have more than one theme, they belong to more than one theme group and are therefore grouped separately. To have a distinction in the sense of



**Figure 5.11:** Converting a Venn diagram to a graph

the intersection in a Venn diagram, each individual is assigned to a force center (C). Each force center represents a unique combination of theme groups. As shown in Figure 5.10, two theme groups result in four force centers. One for each of the theme groups, one for the intersection of the theme groups, and one for the individuals who do not belong to any of the theme groups. To see the combination of theme groups, each force center has a force center annotation (D).

As the name *force center* suggests, this visualization uses a force simulation to assign individuals to theme groups or force centers. Originally, this was built as a simulation from the notion that *d3-forces*,<sup>7</sup> can do this, so we don't have to do the layout of the individuals ourselves.

### 5.5.3.1 Layout of the Force Centers

In the first version of this prototype, the force centers were randomly placed on the canvas and it was the responsibility of the users to arrange them to be convenient. To improve usability, we implemented a default layout so that users could get useful information right away. To achieve this, we regarded the Venn diagram as a graph (Figure 5.11).

For every force center, a vertex of the corresponding graph is created (1). The force centers that share a theme group are connected via an edge (2). We then used the resulting graph (3) to layout the force centers. The simulation results in a layout for the whole diagram and not only the force centers, because the individuals are drawn to their corresponding force center and the group hulls are drawn around the individuals. Well-known algorithms for deterministic, planar graph layout are *Eades* [24], *Davidson & Harel* [21], and *Fruchterman & Reingold* [28]. Although these algorithms produce well-designed graphs, we ultimately decided against using them. The reason for this is that we were already familiar with forces simulations. It was easier to transfer this knowledge to the layout than to familiarize ourselves with a new algorithm.

Three forces are used for the simulation. A center force<sup>8</sup> that tries to move all force centers in the middle of the canvas, a link force,<sup>9</sup> that tries to keep all edges at a specified length, and a negative many-bodies force<sup>10</sup> that tries to repel force centers

<sup>7</sup><https://github.com/d3/d3-force> (last accessed: 2020-07-30).

<sup>8</sup><https://github.com/d3/d3-force#centering> (last accessed 2020-07-30).

<sup>9</sup><https://github.com/d3/d3-force#links> (last accessed 2020-07-30).

<sup>10</sup><https://github.com/d3/d3-force#many-body> (last accessed 2020-07-30).

from another. With the interplay of these three forces, a good approximation to an evenly distributed graph is achieved.

### 5.5.3.2 Click Interaction

The *Venn Diagram* uses a canvas element as its drawing canvas for the same reasons as the *XY Diagram* (section 5.5.1.1) and thus results in the problem that a canvas can only handle pixels. Therefore click events do not go to the graphical representation of an individual, but to the canvas. To deal with this problem, we used the position matching strategy as explained in section 5.4.4.

This means getting the position of a click and then checking for every data element, in our case the individuals if the click happened within the bounds of the individual's graphical representation. As the individuals are represented by dots, we can check whether the Euclidean distance of the click position is smaller than the radius of the circle. If that is true, we found the individual, that resulted in the graphical representation and can thus map the individual's representation back to the individual.

### 5.5.3.3 Double Click Interaction

The force center annotations can be visually distracting. So we use a double click on the graphical representation of the corresponding force center to toggle whether the annotation is displayed or not. Because the canvas only handles pixels, we need a bidirectional mapping from the canvas to the force centers. Again, we use position matching for this. Only this time, the checking whether the position is in the bounds does not happen on the individuals, but on the force centers. This way we achieved a bidirectional mapping of the force centers to their graphical representations. When a force center is found, it tells its annotation to toggle its visibility.

### 5.5.3.4 Drag Interaction

For manual rearranging of the force centers and thus the Venn diagram itself, we use the d3 dragging API.<sup>11</sup> This API-call needs the draggable subject. Here again, we use the position matching strategy to find the force center at the mouse position, when dragging is started.

### 5.5.3.5 Free-Hand Selection Interaction

In the *Venn Diagram*, we allow a lasso select. It is implemented like in the *XY Diagram* (section 5.5.1.2). The only difference is that the *FreeHandDrawer* uses its own canvas to draw the selection while it is made by users instead of the default canvas for the individuals. This is due to the simulation, the visualization uses. The simulation clears the canvas with every tick, to draw the next step. If the current selection was drawn on the same canvas, it would be cleared too and not be visible to the users.

<sup>11</sup><https://github.com/d3/d3-drag> (last accessed 2020-07-30).

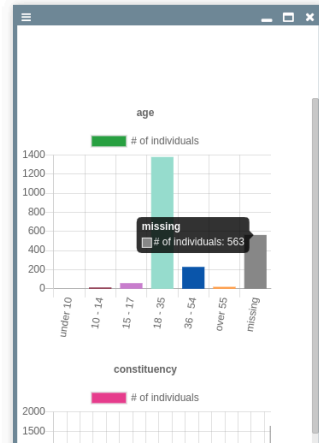


Figure 5.12: A screenshot of one bar chart of a statistics panel

### 5.5.4 Statistics Panel

The idea behind the Statistics Panel is to allow users to see the demographic composition of a data set at one glance. Therefore it consists of one bar chart per demographic attribute where the x-axis displays the values that the attribute can have and the y-axis the amount of individuals. The Statistics Panel web component is a wrapper to allow us to apply actions like filtering the same way as in the other visualizations and to have multiple bar charts in one visualization. For ease of handling, we chose not to draw the bar charts by hand but to use the library `chart.js`. We will explain why in section 5.5.4.3. When it comes to interactions, `chart.js`<sup>12</sup> only supports hovering and clicking<sup>13</sup>.

#### 5.5.4.1 Hover Interaction

When users hover over a bar chart `chart.js` per default generates a tooltip which shows the size of the bar, in our context the amount of individuals that make up the bar. Because this is useful to get a quick overview, we had to use the click interaction to allow users to further explore the demographic groups.

#### 5.5.4.2 Click interaction

The click interaction in this prototype differs from the click interaction in the other prototypes. Here it is not an individual that gets clicked, but a bar and therefore, semantically, a group of individuals. There would have been two ways to integrate the click action in our system. The first is to extend the `InspectAction` so it can inspect more than one individual. The second is to use the freehand selection mechanism. For an explanation of this interaction pattern, see section 3.3.5. Because

<sup>12</sup><https://www.npmjs.com/package/chartjs> (accessed 2020-07-30).

<sup>13</sup><https://www.chartjs.org/docs/latest/general/interactions/events.html> (accessed 2020-07-30).



**Figure 5.13:** Screenshot of a not connected child visualization of a statistics panel

an inspection limits users to explore a data set by only showing it in an Individual Inspector and highlighting individuals, we chose to use the freehand selection mechanism. This way enabled users to easily create new visualizations from a demographic group and view that group in other contexts although only in the *Tree View* prototype as explained in section 3.5.3. So when a bar is left-clicked, we notify the visualizations pane via an event that a freehand selection was right-clicked. This triggers the creation of new visualizations from the individuals sent in the event. The combination of `chart.js` and the freehand selection leads to three problems.

The first is, that usually the creation of visualizations from a freehand selection is triggered with a click of the right mouse button. `Chart.js` does not support that, so we had to use the left mouse button click. This inconsistency makes it harder for first time users to find out that this is even possible.

The second problem is, that in the *Tree View* prototype we use `jspLumb`<sup>14</sup> to draw connections between the freehand selection and the visualization made from it. To do that, the connection endpoints have to be HTML DOM elements. `Chart.js` draws the bars and hold references to them, without mounting them into the DOM. Therefore, the bar cannot be an endpoint for a connection as shown in Figure 5.13. The *XY Diagram* was created from the 18-25 bar in the statistics panel.

The third problem is that we have to send the selection individuals in the event. `Chart.js` does not support saving custom data to bars. This means we cannot store the individuals that make up a bar in the bar element in order to extract them easily when the bar is clicked. Because of that, we had to write a wrapper class around `chart.js`, which holds the grouped data. The next issue is to find out which group is needed. We found out that the bars store their label in the variable `_model.label`. Because the labels match with the values the group has for the attribute we group the individuals by, we can use that to find the right group. This means we have a mapping from bars to individuals and thus a bidirectional mapping of individuals and bars.

#### 5.5.4.3 Choice of Visualization Library

The click interaction came with three problems mentioned before, that we would not have, using `d3` instead of `chart.js` as a library to create the bar charts. `d3` supports listening on right clicks, can generate HTML DOM objects as bars, and

<sup>14</sup><https://www.npmjs.com/package/jspLumb> (last accessed 2020-07-30).

allows appending custom data to bars. So why would we choose `chart.js`? The only reason is the programming speed. `Chart.js` is less complex than `d3` and therefore has an easier to use declarative style than `d3`. This is crucial because when we added the statistics panel, we did not have much time left for the project. The focus of the project was not on standard visualizations but on new ones, which keep the individual in focus. Therefore we did not want to spend more time than necessary on bar charts to allow us to improve our prototypes of novel visualizations. So in a different context, `d3` would have worked better, but given the circumstances, `chart.js` was the better choice.

### 5.5.5 Evaluation of Our Approach on Bidirectional Mapping

The main problem that makes bidirectional mapping difficult is the loss of information (section 5.3.3). That loss can have multiple origins. The origins we dealt with are too little information in the data set and aggregation. In the previous sections, we stated how we achieved bidirectional mapping in our main visualizations. Throughout their implementation, we dealt with information loss. To solve this problem, we used several approaches.

Firstly, we traced the graphical representations back to elements in our individual-centered data format, instead of the source data that Africa's Voices sent us as stated in section 5.2.5. This way we avoided the problem of non-existent data from the start.

Secondly, we avoided the problem of aggregation by never really aggregating. An example of this is the *XY Diagram*. If the number of individuals is selected as the grouping of an axis, as shown in Figure 5.14, the individuals are arranged so that it looks like a bar chart to users. The individuals, the point to which we want to map back to, are still displayed and are therefore traceable. When we could not avoid aggregating the data, for example in the Statistics Panel, the visualization stored the data. By saving the grouping, the attribute, and the values of a group, the mapping can then be reverted.

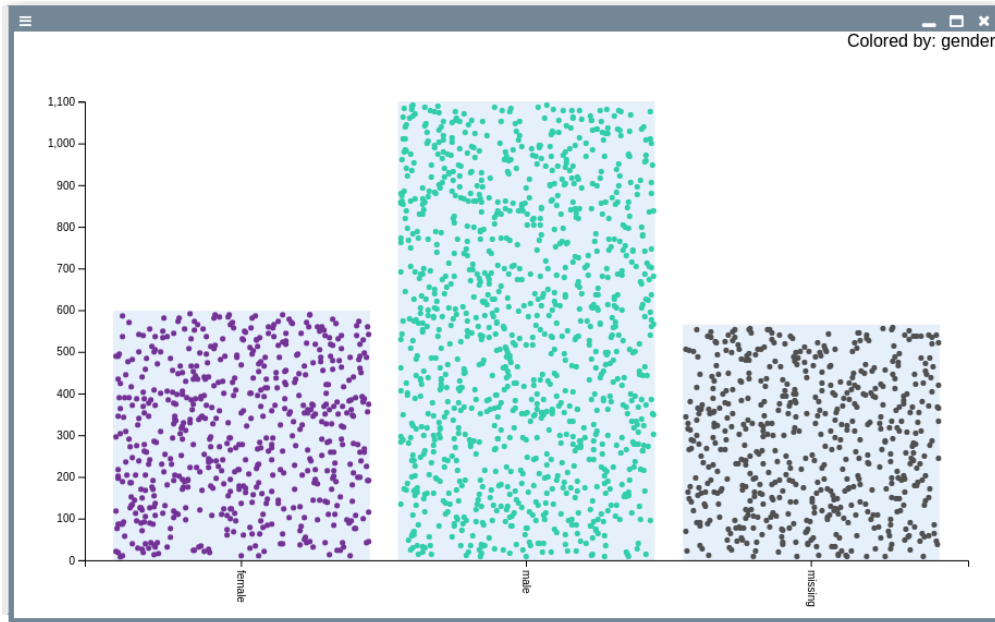
In summary, the following strategies for dealing with information loss have proven to be successful: On the one hand, choosing a reasonable data structure as a starting point for bidirectional mapping and, on the other hand, never completely aggregating the data, but always keeping at least a reference to the original data.

## 5.6 Conclusion

Our goal was to develop visualizations that allow for interactive exploration of the underlying data.

We evaluated three strategies for implementing bidirectional mapping. Based on these strategies, we were able to backtrack the data from which the visualizations were built, enabling the exploration of the data. The main idea of implementing bidirectional mapping was, in addition to the used strategies, to avoid aggregations wherever possible to work around the problem of information loss. Where





**Figure 5.14:** A screenshot of the XY Diagram, with the gender on the x-axis and amount on the y-axis

aggregation is unavoidable, we still had access to our raw data allowing us to draw inferences.

Our work shows ways to implement bidirectional mapping. However, we have evaluated only a part of the possible methods. Therefore, our work allows to gain a basic understanding of the difficulties of bidirectional mapping in order to evaluate or develop further methods to determine the origin of data in future work.

In the context of our project, the bidirectional mapping of data and UI allowed us to develop interactive visualizations that allow the exploration of the underlying data. Explorability enables users to develop empathy and creates confidence in the correct representation of data through visualization.



## 6 Evaluating Visualization Technologies to Display, Animate and Explore Individual Data Points of High Dimensional Data Sets in Lively4

The visualizations we have created are a graphical representation of individual-related data, focusing on explorability and ensuring that individuals are always visible. Rendering Interactive visualizations with up to 100,000 points on client-side in a web browser is a demanding and complicated task. This thesis's content is the benchmarking and evaluation of different web technologies to develop, render, animate, and interact with visualizations with several thousand points on the web. We determined different web technologies, found metrics to evaluate them, and used the browser and JavaScript capabilities to measure and evaluate those technologies' performance. We could show that the performance of a technology correlates with its memory consumption and how the visualization environment influences the performance of the rendering technologies.

### 6.1 Introduction

The visualizations we have created are a graphical representation of individual-related data, focusing on explorability and ensuring that individuals are always visible. To make this possible, we have assigned the data points of our data set, i.e., the individuals, their own graphical element. We have chosen a graphical primitive; the point. We implemented the prototypes for these visualizations in the web browser. While modern web browsers allow for diverse interactions, displaying complex graphics using client-side rendering is still a challenging and resource-intensive task. In our visualizations, up to 100,000 points must be rendered and animated. Furthermore, the users must be able to interact with the points, click on them, regroup them, or color them according to specific attributes. There are several technologies available for rendering our visualizations, including SVG elements, the HTML `<canvas>` element, or native HTML `<div>` and `<span>` elements. This chapter aims to work out the advantages and disadvantages of these technologies, classify them using computer graphics categories, and to benchmark different metrics. To compare the technologies in a meaningful way and to understand how they work, it is essential to understand basic rendering processes in the browser. This will be the content of the first part of this chapter. Furthermore, we will explain

the difference between Immediate mode and Retained mode, the two dominant paradigms in computer graphics, and how to best measure and quantitatively evaluate rendering processes in the first part. In the second part of this chapter, we explicitly deal with our visualizations in Lively4. We will work out the technical visualization requirements and introduce the web technologies we use to represent many points in detail. Additionally, we will categorize the technologies into the Immediate or Retained mode and discuss advantages and disadvantages. Content of the third part of the thesis is the benchmarking of these web technologies. We will first discuss benchmark techniques in general in the browser. Then we will explain the benchmarks we have carried out, present hypotheses, present the benchmark results, and discuss them. In the last part of the thesis, we will discuss the remaining problems of our visualizations, present further optimizations, and give an outlook for possible future work on our visualizations.

## **6.2 Visualization Environment and Approach**

This section contains a description of the visualization and development environment. Besides, we will briefly outline the elements of which our visualizations consist and what requirements the visualizations have.

### **6.2.1 Browser and Lively4**

We developed all our visualizations in Lively4. Lively4 is a live programming environment in a web browser based on HTML, CSS, and JavaScript[38]. So in principle, we could use all technologies for graphical presentation that are used in conventional browser applications. The resulting visualizations are rendered on the client-side. This means that parts of the data manipulation and the actual rendering of the visualizations take place in the browser. Client-side rendering is necessary because our visualizations should be interactive and not just rendered as static images, as described in section 1.2.2. The users must be able to adapt and change the visualization dynamically and see the result of their interaction immediately. In Lively4, we have written markdown documents that support HTML, CSS, and JavaScript; see section 4.3.1.2 for further information.

### **6.2.2 Visualization Approach**

The challenge and purpose of our visualizations are to present personal data graphically and to always keep the focus on the individual. To achieve this, we decided to assign a graphic element to the individual so that each individual remains tangible and visible within the visualization (section 1.1.3.3). As described in subsection 3.2, in our visualizations, this graphic element is the point. The data provided to us covers between 1,000 and 100,000 individuals. This means that we have to render and animate between 1,000 and 100,000 points. Also, as explained in

section 3.3, users must be able to interact with these points, arrange them according to specific parameters, and change their color. To better understand the graphical primitive of the dot from a technical point of view, let us briefly describe what the entities of a point are that we need to control completely:

- Radius (pixel),
- Scope (pixel),
- Surface (opacity),
- Border (pixel),
- Surface color (color),
- Border color (color),
- Position (pixel),
- Motion direction and speed (pixel per ms).

Further components of our visualizations, as further described in section 3.4, are scales, geographic maps, annotations, labels, and background colors to indicate groupings. Our visualizations go far beyond the presentation of static web content and thus use the resources of a browser enormously. In the following chapters, we will explain the technology stack that enables us to implement such visualizations in the web browser.

### 6.2.3 Visualization Requirements

In the following, we will go through the requirements for our visualizations and derive the technical implications. The list of requirements is not complete and is specifically directed to the domain of rendering technologies. Also, the requirements define the view and perspective with which we look at and compare the different rendering technologies. Besides explaining each of them, we will derive measurable metrics.

#### 6.2.3.1 Fast Prototyping

Our goal was to quickly evaluate different visualization ideas. At the beginning of the project, our project partner stated the following goal: “Try to create a visualization a day.” To develop visualizations quickly, the following must be given.

- The complexity of the code required to display them should be as low as possible. This allows to derive the code complexity for the creation of the visualizations as a metric to evaluate web technologies for rendering many points.
- Also, the available documentation should be as comprehensive as possible, and the technology should be accessible (community support, stack overflow entries).

None of us is an expert in the field of computer graphics, and the focus was mainly on the development of visualization concepts besides the development of applications for displaying those visualizations. For this reason, accessibility and complexity of the technologies play an essential role.

### 6.2.3.2 User Experience

We have implemented different visualizations, which are different views of the data, i.e., they highlight different aspects of the data. To draw conclusions from the data, users must be able to switch quickly between visualizations or display them simultaneously. To make this process as convenient as possible, the following must be considered:

- Rendering time, the time needed to display the visualization on the user's screen, should be as short as possible. This allows us to derive the "time to render" as a metric. We define the time to render as the time that passes between a render statement from the client and the actual rendering on the user's screen, so for example, the time that passes between a JavaScript call to manipulate a web page and the actual appearance of these changes on the user's screen.

In our case, the render time can also be understood as the visualization application's response time. The limit for having the user feel that the system is reacting instantaneously is about 0.1 seconds [52].

### 6.2.3.3 Explorability through Responsiveness

In addition, the data set should be explorable as stated in subsection 1.1.3.7. Explorable means that the users have to be able to interact with our visualizations instead of just consuming a simple graphic image. Part of this interaction are animations in which points are grouped, moved, and colored. Individual points can also leave color traces or repel each other in some of our visualizations. In order to make the animations as comfortable as possible for the user, they must run smoothly. Therefore the following must be considered:

- According to the RAIL model,<sup>1</sup> which was introduced by Chrome developers at Google to evaluate the performance of their applications in a user-centered way, animations with 60 frames per second are pleasant to watch. To render an animation at 60fps requires that the time needed for rendering the content is 16.67ms maximum. Conventional video films run at 24fps and are perceived as pleasant. Nevertheless, as a general rule, the higher the frame rate the more enjoyable animation is perceived by the user. The frame rate of animations depends on two circumstances. First, how long it takes to render the animation graphics, and second, the refresh rate of the display of the device on which the animation is running. An average display has a refresh rate of 60Hz, which means that a frame rate of more than 60fps is technically not possible on such a display. The frames per second and the time to render are very close together. The greater the time to render, the lower the fps and vice versa, which is the result of both metrics' definitions. Therefore, we will only record the time to render in the benchmarks later on and will not list the fps as a single metric.

---

<sup>1</sup><https://web.dev/rail/> (last accessed 2020-07-29).

#### 6.2.3.4 Bidirectional Mapping

As explained in section 5.3.3, to interact with the visualizations in a meaningful way, it must be possible to infer the corresponding data point from a graphical element of the visualization. For example, it must be possible to display an annotation with more detailed information about an individual when users click on a point. As explained in section 5.5, there are many different possibilities and approaches to implement bidirectionality in the browser using JavaScript and HTML. We have tried out different concepts in different visualizations. Often the used implementation of bidirectionality depends on the used visualization technology. To make many interaction patterns possible, the following must be given:

- Code complexity and side effects when introducing interaction patterns must be manageable. Also, from the point of view that prototypes can be created and adapted quickly. Code complexity is also suitable as a metric here.

Bidirectional mapping makes it possible, that our visualizations are not just static images, but that we can explore the underlying data set through interaction with the visualization.

#### 6.2.3.5 Novelty

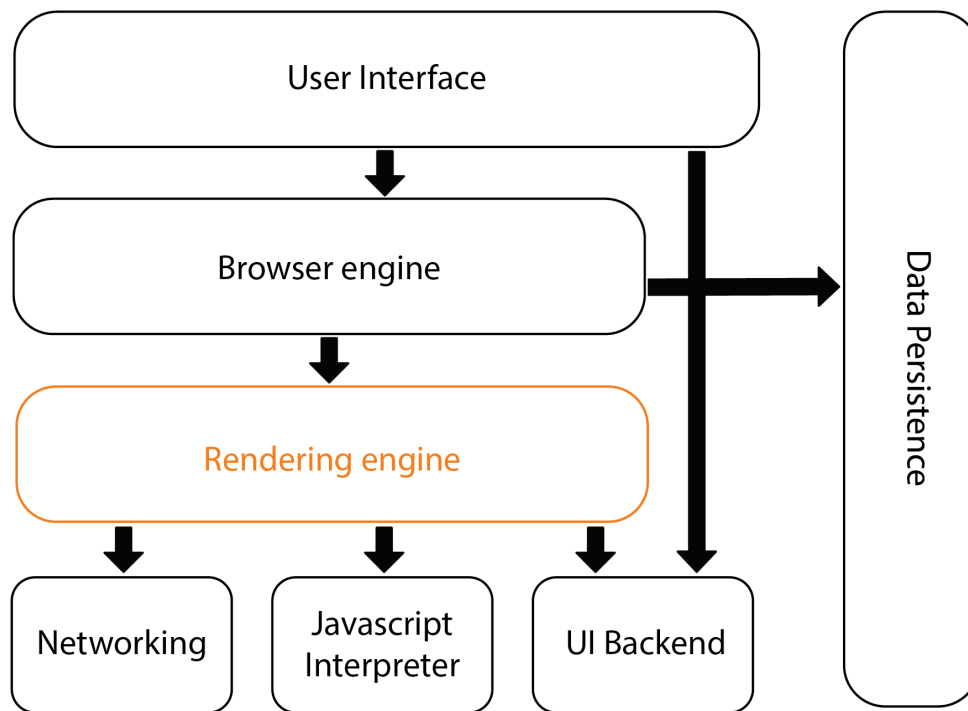
Our project partner demanded that we produce novel visualizations that put the individual in focus. So we could not use prefabricated solutions but had to go in different directions, try out different visualization ideas, and evaluate different interaction patterns. To achieve this, the following circumstances must be given:

- There must be an appropriate degree of freedom in the implementation of different ideas. The technologies used must allow us to render any graphic element as freely as possible and interact with it. At the same time, the technology or library used must offer enough abstraction so that prototypes can be implemented and tested quickly. This is not a property that can be benchmarked, but only an assessment of the technology and libraries' APIs.

The technology requirements range between complexity and accessibility, freedom, and performance. In section subsection 6.5.4 we will again discuss the requirements and classify the technologies we use.

## 6.3 Graphics in the Browser

When rendering many thousands points in the browser, performance of the underlying technology plays an enormous role (see section 6.5). As seen in subsection 6.2.3, to provide a smooth user experience, it must be ensured that the visualizations are rendered quickly and without errors on desktop devices in the browser. A good understanding of the browser structure and the rendering processes is required to optimize this process.



**Figure 6.1:** Basic browser architecture

### 6.3.1 Browser Architecture

The architecture of different browsers differs in many respects. Figure 6.1 is, therefore, only to be understood as a reference architecture. The components described there can be found in every browser and show here exemplarily, which components interact in which way [34]. The user interface is the component with which the users interact to retrieve graphical web content. The browser engine acts as a bridge between the user interface and the rendering engine. Based on the input of the users, it makes requests to the rendering engine. The rendering engine is the heart of every browser. It is responsible for parsing HTML-, CSS-, XML-content, and rendering it onto the user's screen. There are currently three popular desktop browsers<sup>2</sup> and therefore three major rendering engines, Gecko,<sup>3</sup> which is used by Firefox, Webkit<sup>4</sup> which is used by Safari, and Blink<sup>5</sup> which is used by Chrome.<sup>6</sup> More about the render engine and related processes in section subsection 6.3.2. The Networking component

<sup>2</sup><https://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-202006-202006-bar> (last accessed 2020-07-25).

<sup>3</sup><https://developer.mozilla.org/en-US/docs/Mozilla/Gecko> (last accessed 2020-07-25).

<sup>4</sup><https://webkit.org/> (last accessed 2020-07-25).

<sup>5</sup><https://www.chromium.org/blink> (last accessed 2020-07-25).

<sup>6</sup><https://arstechnica.com/information-technology/2013/04/google-going-its-own-way-forking-webkit-rendering-engine/> (last accessed 2020-07-23).



is responsible for fetching files over the Internet and the correct use of various communication protocols of the internet. The JavaScript Interpreter is responsible for parsing and executing JavaScript code. The vast majority of browsers have their own JavaScript interpreters. Chrome uses V8;<sup>7</sup> Firefox uses SpiderMonkey<sup>8</sup> (the historic first JavaScript interpreter), and Safari uses Nitro.<sup>9</sup> Because every browser uses different JavaScript interpreters, different browsers support different JavaScript features. Furthermore, the implementation differs considerably, e.g., Mozilla Firefox implements `Array.sort()` as a merge sort,<sup>10</sup> but Chrome as a timsort.<sup>11</sup> This also leads to different performance and execution times of the browsers. We will only consider the Chrome Browser in this work. The UI back-end is used to display classic widgets such as select boxes, input boxes, or checkboxes. For this reason, these widgets are often displayed differently in different browsers. Another critical feature of the browser is data persistence. The persistence layer is responsible for the persistence of data, which helps the browser store data (like cookies, local storage, session storage, IndexedDB, WebSQL, and FileSystem) locally [6, 29].

### 6.3.2 Render Process in the Browser

The implementation of web content rendering differs for browsers. Nevertheless, the general process is the same for all popular web browsers (Safari, Chrome, Firefox) and often referred to as CRP,<sup>12</sup> the critical rendering path.

Figure 6.2 describes the rendering process of Webkit. In the following part, we refer only to the abstract intermediate steps. Since there are no significant differences between the render engines at this level of abstraction, we will not differentiate between Webkit and Gecko. Optimizing each of these steps is critical to achieving optimal rendering performance<sup>13 14</sup>.

We create web content using HTML and CSS files. The rendering engine gets these documents from the network layer of the browser. When parsing the HTML, the engine converts the HTML tags into DOM nodes.<sup>15</sup> The DOM is both a data structure with which the browser describes HTML documents and an interface for the developer who can interact with the DOM using JavaScript, i.e., dynamically

<sup>7</sup><https://v8.dev/> (last accessed 2020-07-25).

<sup>8</sup><https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey> (last accessed 2020-07-25).

<sup>9</sup><https://developer.apple.com/documentation/javascriptcore> (last accessed 2020-07-25).

<sup>10</sup><https://dxr.mozilla.org/seamonkey/source/js/src/jsarray.c> (last accessed 2020-07-27).

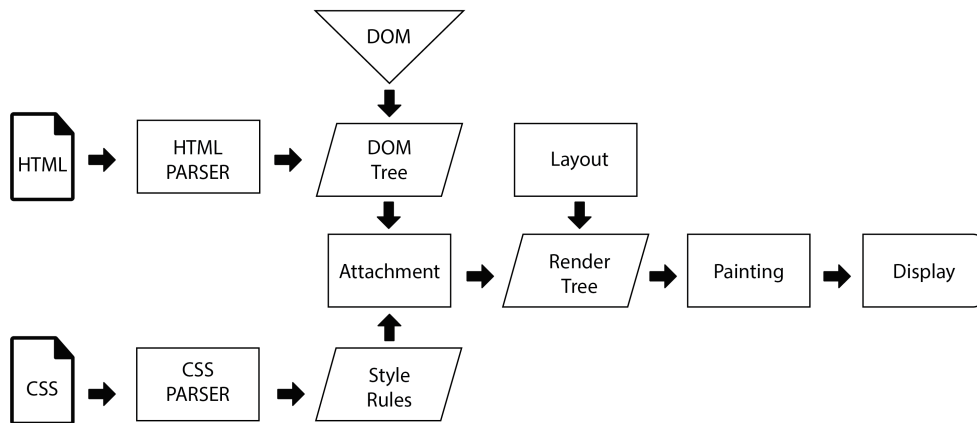
<sup>11</sup><https://v8.dev/blog/array-sort> (last accessed 2020-07-27).

<sup>12</sup>[https://developer.mozilla.org/en-US/docs/Web/Performance/Critical\\_rendering\\_path](https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path) (last accessed 2020-07-25).

<sup>13</sup><https://developers.google.com/web/fundamentals/performance/critical-rendering-path> (last accessed 2020-07-19).

<sup>14</sup>[https://developer.mozilla.org/en-US/docs/Web/Performance/Critical\\_rendering\\_path](https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path) (last accessed 2020-07-19).

<sup>15</sup><https://dom.spec.whatwg.org/> (last accessed 2020-07-19).



**Figure 6.2:** WebKit rendering process

add or remove nodes, manipulate the display of the nodes and add event listeners to nodes. The same happens with the CSS information of a website. These are also parsed and the CSSOM, the CSS Object Model,<sup>16</sup> is created.

Then the DOM and CSSOM are combined to form another tree structure, the Render-tree. This data structure contains all nodes necessary to render the website. In the Render-tree, the style information of the nodes in the CSSOM is added to the nodes in the DOM. The Render-tree contains only objects that are actually rendered on the user's screen. The objects in the Render-tree represent rectangles with geometric information such as height, width, and position. Therefore, the relationship between the DOM and the Render-tree is not 1-to-1, for example, nodes with the CSS style attribute `display: none` or the HTML `<head>` element are not included in the Render-tree. Furthermore, many nodes of the DOM are very complex and are represented within the Render-tree with multiple objects.

After Render-tree construction comes the phase of layouting. In this phase, the exact size and position of the nodes to be rendered are calculated from the Render-tree, taking the viewport into account. The Render-tree is completely traversed, and all nodes are aligned accordingly. During layouting, all relative distances (e.g., CSS style attributes like `width: 80%`, `margin: 5%`) are translated into absolute pixel values. HTML uses an intuitive flow-based layout model by default. This means that elements are arranged in the order in which they appear: from left to right, from top to bottom. A node's position in the Render-tree is specified by the upper left corner of a node. It is laid out according to a coordinate system, where  $(0,0)$  is the upper left corner of the viewport. Layouting is a recursive process, meaning that each object of the Render-tree implements a `layout()` function, which calls the same method on all child objects. The output of the layout process is a "box model," which precisely captures the exact position and size of each element within the viewport [32].

<sup>16</sup><https://www.w3.org/TR/cssom-1/> (last accessed 2020-07-17).

At the end of the rendering process comes the phase of painting. In this phase, the Box Model created from the Render Tree is rendered as pixels on the screen.

The time needed to construct, layout, and paint the render tree depends on the size of the documents, the styles applied (CSS attributes), and the device it is running on: the larger the document, the more work the browser has to do; the more complicated the styles, the more time is needed for rendering (e.g., a solid color is time-efficient to render, while a drop shadow is much more complex to calculate and render, and therefore takes more time [61]) [69, 32, 30].

Now the question arises, especially from a performance perspective, what happens when the DOM or CSSOM is modified. At this point, we distinguish between local and global layouting.

In the best case, DOM updates only require local layouting. This is the case if only single nodes and their children are affected by changes. The browser gives the nodes flags in the DOM. These flags indicate whether a node is “dirty” or “clean”, i.e., whether it needs to be relayouted. Local layouting can also be done asynchronously by the render engine, which works single-threaded. Asynchronous layouting means that the relayout operation can be pushed in a queue, and be executed in the idle time of the browser without blocking the client application.

Global relayouting, on the other hand, is done, for example, on a window resize or change of global CSS styles. Global relayouting is synchronous, i.e., the JavaScript execution of the client is blocked. To make web content as performant as possible, we have to avoid global relayouting because it is not as time-efficient as local relayouting and blocks the client application [30].

Similar to layouting, painting is divided into local and global repainting.

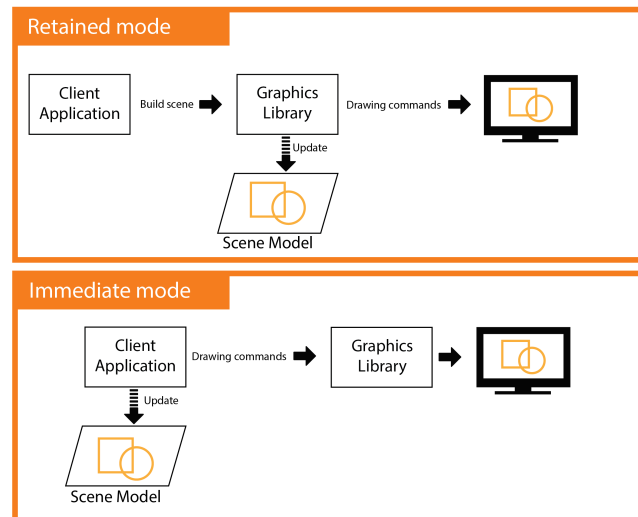
We will see in section subsection 6.5.3 how different technologies influence and use the browser’s layout and rendering behavior in different ways.

### 6.3.3 Immediate Mode Vs. Retained Mode

In the field of computer graphics, a distinction is usually made between two graphics modes, the Retained mode and the Immediate mode. Both are considered API designs or software patterns. We will assign the web technologies we use to display graphic content to the respective modes in the later course of our work and quantify their differences when benchmarking.

In Retained mode, as can be seen in Figure 6.3, the actual rendering is not directly caused by the client. Instead, an abstract internal model is built and updated with the user’s instructions, which only exists within a system (the graphics library). Such an abstract model, called scene model is a collection of objects that contain information about their graphical representation. In our case, such a scene model is the DOM. The system then takes care of how this abstract model is actually rendered. In our case, this system is the browser. It manages the DOM, the CSSOM, is responsible for creating a Render-tree and takes care of the layouting and the actual painting.

In the case of the Immediate mode, as can be seen in Figure 6.3, the client is responsible for the rendering. The instructions for describing rendering primitives are inserted frame by frame directly from the client into a command list and then



**Figure 6.3:** Immediate mode and Retained mode

executed by the rendering engine. So there is no abstract scene model, which is managed by an intermediate system and is updated declaratively. However, the client is solely responsible for managing the scene model.

Retained mode APIs can be easier to use because the API handles more work for the client, such as initialization, state management, and the scene model's cleanup. On the other hand, they are often less flexible because the API specifies its own scene model. Also, an API in Retained mode may have higher memory requirements because it must provide a universal scene model. With an Immediate mode API, targeted optimizations can be implemented, and rendering is faster and more performant than rendering in Retained mode. However, since the client has to manage the scene model on its own and write the rendering statements itself when using Immediate mode APIs, this type of rendering is associated with much greater complexity [5, 59].

We will talk about both modes in the section subsection 6.4.2 and classify the web technologies accordingly.

### 6.3.4 Measuring Rendering in the Browser

In order to evaluate web technologies, we need to be able to collect and quantitatively evaluate metrics about them. To measure the metrics, we have different possibilities at our disposal, which are explained in this chapter.

Basically, it is challenging to measure rendering processes in the browser accurately. This is because processes such as layouting and painting are entirely the responsibility of the browser's render engine. As we will see in this section, the browser does not offer an extensive API with which these processes can be controlled or information about these processes can be obtained. At this point, we differentiate between trying to measure performance metrics within the actual application, i.e.,

using JavaScript, which is executed on the client-side. Or, whether we try to look at the application from the outside, i.e., consider the application as a black box, and use the browser’s graphical performance tools to collect metrics.

#### 6.3.4.1 External

If we measure performance metrics from the outside, i.e., we view the application as a black box, we measure without adding any additional code. This means that the internal application flows and processes do not have to be known; only the execution within a web interface and metrics which can be obtained by that are crucial. The analysis of the execution from outside is mostly done via the browser’s developer tools. Every popular browser offers extensive functions precisely for this purpose. Only Chrome was relevant in the course of our project, both for the development of the visualizations and the execution within the live programming environment Lively4. For this reason, we will only discuss Chrome’s developer tools in this chapter. With these tools, we can measure a whole range of metrics and analyze entire interaction sequences with a web application. The Chrome Developer Tools offer some exciting features to measure rendering processes in the browser, among others<sup>17</sup>:

- GPU usage measurement,
- Measure FPS,
- Show Call Stack,
- Analysis of the DOM,
- Show Repainting Regions,
- Heap snapshots,
- Performance timeline recordings.

In the Figure 6.4, we can see an example of a performance timeline recording over about 14 seconds. We can see that the Chrome Developer Tools provide many metrics. In (A) we see a diagram that summarizes the key metrics measured. At a glance, we see the frame rate, the heap usage, and a color-coding for the CPU usage. In this case, yellow stands for “Scripting” and purple for “Rendering”. In (B), we see all the JavaScript-functions executed in the recorded period and their corresponding call stack. In (C), we see a detailed view of the heap usage. Furthermore, in (D) we see a summary for the measured period. It shows how much time was spent on which tasks. In Figure 6.4, we see that of the approx. Fourteen seconds recorded, approx. Nine seconds were spent on “Scripting”, and approx. Three seconds on “Rendering”.

Another method of external measurement is screen casting. Sometimes the Chrome DevTools are inaccurate because the browser is an application with many influencing factors. To measure frames per second in animations, it is common to record additional screen casts and examine the resulting video with a video motion analysis tool. This technique will not be used further in this work [11].

<sup>17</sup><https://developers.google.com/web/tools/chrome-devtools>  
(last accessed 2020-07-16).

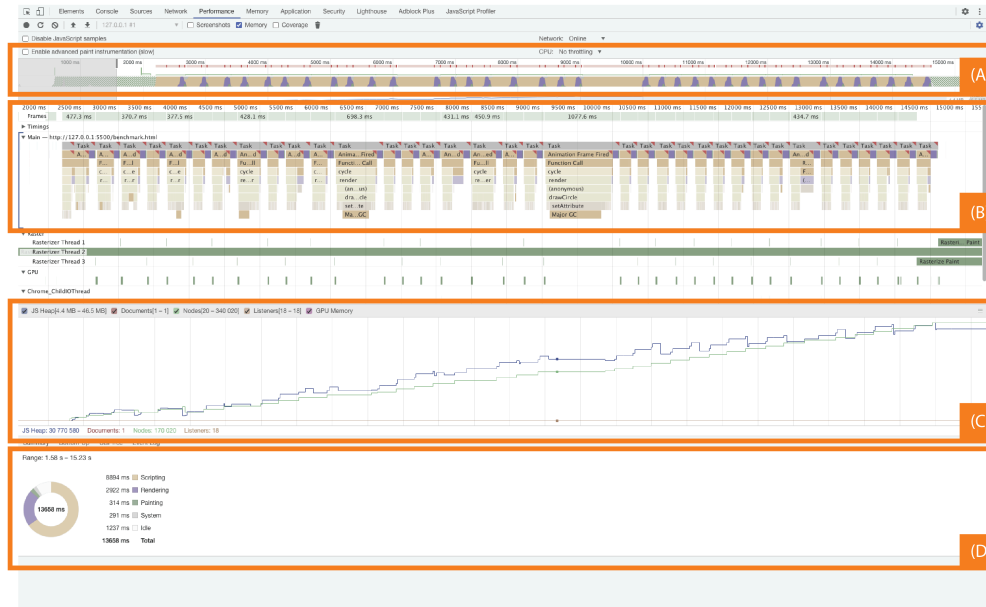


Figure 6.4: Google Chrome DevTools overview

### 6.3.4.2 Internal

When measuring internally, we add code snippets to the actual code of the application. Since source code is directly adapted here, the testers must understand the program structure and the application processes in detail. Often the time is measured, which the application needs for the execution of a particular function to optimize its performance afterward. The possibilities to make application-internal measurements using client-side executed JavaScript are minimal. For such measurements, the browser offers the `window.performance` interface as an API for the developer.<sup>18</sup> Using this interface, we can set markers within the application code using simple instructions to measure the execution time between these markers. The results of these measurements are `DOMHighResTimeStamp`,<sup>19</sup> i.e., doubles representing timestamps in milliseconds.

Another essential function of the Browser API to better understand and measure rendering processes is `window.requestAnimationFrame()`.<sup>20</sup> Calling this method tells the browser that an animation should be executed. As a parameter, we pass a callback, which is called before the browser triggers the next repaint. When

<sup>18</sup><https://developer.mozilla.org/en-US/docs/Web/API/Performance> (last accessed 2020-07-14).

<sup>19</sup><https://developer.mozilla.org/en-US/docs/Web/API/DOMHighResTimeStamp> (last accessed 2020-07-16).

<sup>20</sup><https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame> (last accessed 2020-07-16).

the browser calls the callback method, it passes a current timestamp as a parameter. Therefore it is possible to measure how much time is needed for rendering e.g., measuring the frame rate. The standard JavaScript timers—`setInterval()` and `setTimeout()`—are subject to large amounts of jitter, making them unsuitable for real-time interactivity. So `requestAnimationFrame()` is the modern way to display animations fluently on the web [39, 71]. Instead of the client controlling an animation loop itself, the browser can optimize animations, and the associated relaying and repainting with `window.requestAnimationFrame()` and adapt the animation frames to the frequency of the display (mostly 60Hz). Here is an example that shows how we can measure the render time needed:

```

1 measureRendering()
2
3 function measureRendering() {
4     displaySomething()
5     requestAnimationFrame(startRender)
6 }
7
8 function startRender() {
9     performance.mark("start")
10    requestAnimationFrame(endRender)
11 }
12
13 function endRender() {
14     performance.mark("end")
15     performance.measure("duration", "start", "end")
16 }

```

In the example, we first call `measureRendering()`. Within this method, we call `displaySomething()`. `displaySomething()` manipulates the DOM in some way so that changes have to be rendered. We now call the method `requestAnimationFrame()` and pass the callback `startRender`. `startRender()` is called before the browser decides to trigger a rendering. When `startRender()` is called, we set a start marker and request the next animation frame with the callback parameter `endRender`. When the previous animation frame is rendered and the browser renders the next frame, it calls `endRender()`. In this method, we set an end marker and measure the elapsed time between both markers with the `performance.measure()` statement by specifying a start mark and an end mark.

#### 6.3.4.3 Comparison of External and Internal Measurements

At first glance, it is evident that the Chrome Developer Tools offer considerably more features than the browser APIs provided for internal measurements. Furthermore, the metrics are displayed graphically. Also, we leave the actual application code untouched. We do not need to know the program's exact structure and make sure that the application code is not distorted by adding code that should measure the performance. However, external measurement is difficult to automate. Although Chrome offers an interface for controlling a headless version of chrome [7], there

is much more work involved in automated navigation within a web application, especially since we are on the live system Lively4. It would be easy if we only need to load a website and want to measure the performance of it. However, we need to perform many navigation steps within Lively4 to trigger the execution of a markdown file we have written. Furthermore, the written execution files of the benchmarks are stored on an access restricted Lively4-server (see section 4.4.1.3). Furthermore, the Chrome developer tools use their own file format to persist the data of a performance timeline recording. This makes it challenging to parse and analyze the stored data afterward.

When measuring metrics internally, we need to understand the application code. Also, one is very limited in the scope of the measurable metrics. Furthermore, it is possible that the code snippets that are used to measure the performance of the application can falsify the measurement result. Nevertheless, the internal measurement of metrics makes it much easier to automate the measurement process and gives freedom in terms of data export and data format. We will use both measurement techniques in the latter part of this paper but focus more on the internal measurement of metrics for the actual benchmarking.

## 6.4 Rendering Visualizations with Many Points in Lively4

In order to render several thousand points on the web, different technologies come into question. This chapter aims to introduce these, show how we have worked with them, and discuss their advantages and disadvantages.

### 6.4.1 Web Technologies to Render Points

Since we have all browser technologies available, we can use three native technologies to display points: simple HTML elements, SVG, or Canvas. To compare how we can render points with these technologies, we will briefly introduce each technology below and explain the code needed to render a point. We will take a dot (see Figure 6.5) with the following properties:

- diameter: 6px, radius: 3px
- border-width: 1px
- border color: #000000
- fill color: #eb4438

#### 6.4.1.1 Simple HTML Span- or Div-Elements

Simple HTML `<div>` or `<span>` elements can be manipulated with CSS to represent and render points. Thus, using JavaScript, one could dynamically create such span elements, adjust their geometric data using CSS, and add or delete them as desired to the DOM. The advantage would be, that we have a reference to the painted element at any time because rendered HTML elements are accessible via the DOM. The representation of such points would be easy to describe using HTML and CSS:





Figure 6.5: A rendered dot

```

1 .dot {
2   height: 6px;
3   width: 6px;
4   background-color: \\#eb4438;
5   border-radius: 50%;
6   border: 1px solid black;
7   position: absolute;
8 }

```

```

1 <div class="dot"></div>

```

#### 6.4.1.2 SVG

SVG, or Scalable Vector Graphics,<sup>21</sup> is a language in an XML file format that was explicitly designed for scalable 2D graphics. The SVG file format can be used to display Vector graphic shapes, images, or text. It is possible to use SVG in HTML files with the `<svg>` tag, which means that SVG elements are also anchored in the DOM.

```

1 <svg height="100" width="100">
2   <circle cx="2" cy="2" r="3" stroke="black" stroke-width="1"
3     fill="#eb4438" />
4 </svg>

```

As we can see in the HTML code above, SVG offers some primitives,<sup>22</sup> including the `<circle>` tag. If this primitive did not exist we would have to describe the circle with paths, which would be much more complex and time-consuming.

```

1 <svg height="100" width="100">
2 <path d="
3     M 100, 100
4     m -75, 0
5     a 75,75 0 1,0 150,0
6     a 75,75 0 1,0 -150,0
7     "/>
8 </svg>

```

<sup>21</sup><https://www.w3.org/TR/SVG2/> (last accessed 2020-07-13).

<sup>22</sup><https://www.w3.org/TR/SVG2/shapes.html> (last accessed 2020-07-13).

### 6.4.1.3 Canvas

The `<canvas>` is an HTML element and offers a free drawing surface on which various basic shapes can be drawn using JavaScript methods. The canvas itself is anchored in the DOM. The actual graphical elements drawn on the canvas are not part of the DOM, so they cannot be referenced through the DOM. It is a low-level process model that updates a pixel bitmap and does not manage its own scene model. To work with the canvas element, we have to get a drawing context for this canvas dynamically using JavaScript.<sup>23</sup>

```
1 let canvas = <canvas width="800" height="800"></canvas>;
2 context = canvas.getContext("2d")
```

There are different contexts available for the canvas element, whereby only two are relevant for us, the `CanvasRenderingContext2D` and the `WebGL` context.

The `CanvasRenderingContext2D`<sup>24</sup> interface, part of the Canvas API, provides the 2D rendering context for the drawing area of a canvas element. It is used to draw shapes, text, images, and other objects. To render the above point in this context, the following code would be required:

```
1 <div id="container">
2   <canvas id="draw-canvas" width="800" height="800">
3 </div>
```

```
1 let canvas = lively.query(this, "#draw-canvas")
2 let context = canvas.getContext('2d')
3
4 drawCircle()
5
6 function drawCircle() {
7   var centerX = getRandomFloat(0, width)
8   var centerY = getRandomFloat(0, width)
9   var radius = 3
10
11   context.fillStyle = '#eb4438'
12   context.lineWidth = 1
13   context.strokeStyle = '#000000'
14
15   context.beginPath();
16   context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false)
17   context.stroke()
18   context.fill()
19 }
```

The code of the function `drawCircle()` can be understood as follows. We define the center of the point (set by random float). We also specify the radius of the point. Now we specify how the point should be drawn. At this point, we can imagine that

<sup>23</sup><https://html.spec.whatwg.org/multipage/canvas.html> (last accessed 2020-07-15).

<sup>24</sup><https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D> (last accessed 2020-07-16).

the pencil is assigned the appropriate attributes. So we set the fill-style (inner color of the area), the line-width (border of the point), and the stroke-style (color of the border of the point). Afterward, we specify how the pen we just defined is to be moved over the bitmap. With `beginPath()` we practically place the pen on the drawing area. The `arc()` method defines a circular area. With `context.fill()` and `context.stroke()`, we draw the circle line and fill the resulting area.

The `WebGLRenderingContext`<sup>25</sup> provides an interface to the OpenGL ES 2.0<sup>26</sup> graphics rendering context for the drawing area of a canvas element. We can draw on it using WebGL. WebGL programs consist of control code written in JavaScript and shader code. A shader is a chunk of program code that implements algorithms to get the pixels onto the screen. Shaders are typically defined in a high-level C-like language and compiled into code usable by the graphics processing unit (GPU). The GPU understands just vertices and textures; it has no concept of material, light, or transform. The translation between those high-level inputs and what the GPU puts on the screen is done by the shader.[54]

By using the graphics card, WebGL is much more powerful than other web technologies. However, it also adds much more complexity. To show the code for rendering a point with WebGL would go beyond the scope of this chapter. To illustrate the complexity of the task, it should be mentioned that about 80 lines of code were needed to render the point above. These 80 lines include the JavaScript code, the shader code and the HTML code. When rendering with WebGL, we use a low-level library called ReGL.<sup>27</sup> The code used during benchmarking can be seen in section C.4

### 6.4.2 Discussion

First, we can assign the technologies to one of two modes, Immediate mode or Retained mode. SVG and simple HTML elements like `div` or `span` elements can be assigned to the Retained mode because the browser manages them in the DOM. The client is not responsible for the actual rendering, but the browser determines the display of those elements.

The Canvas element, on the other hand, can be assigned to the Immediate mode[5], since here the user specifies drawing commands for various graphics primitives and the browser calls native libraries and interfaces of the local operating system with the client's instructions. The browser is merely an intermediary that exposes an area in the browser window where the operating system is used to draw in. Thus the actual implementation is dependent on both the browser and the operating system. Because native rendering technologies are used, the browser does not have to manage a scene model, and the objects on a canvas are not managed in the DOM. Therefore much overhead is eliminated for the browser. Thus the canvas should be much more

<sup>25</sup><https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext> (last accessed 2020-07-18).

<sup>26</sup><https://www.khronos.org/opengles/> (last accessed 2020-07-18).

<sup>27</sup><http://regl.party/> (last accessed 2020-07-24).

performant, at least from a rendering perspective. We will discuss such performance metrics in the next chapter.

From the perspective of bidirectionality, all technologies that can be assigned to the Retained mode are much better suited. Because HTML elements like `div`, `span` or SVG elements are kept in the DOM by the browser, the developer has a reference to these objects at any time of execution. We can add EventListeners for e.g., click-events, so that the user can interact with the rendered elements. This is not possible with graphical elements drawn on a canvas, because the drawn objects are not part of the DOM.

The complexity of WebGL code is enormous compared to HTML elements or the `2dRenderingContext` of a canvas element. Also, the code necessary to create bidirectionality between data points and the graphical points in the visualization is much more complicated for canvas elements (see subsection 5.4.2.1). Because WebGL is executed on the graphics card, the rendering of these objects is much more efficient than with technologies of the Retained mode (see subsection 6.5.3).

## 6.5 Benchmarking Web Technologies

This section of the thesis deals with benchmarking the web technologies mentioned above. The benchmarks were performed both within the live programming environment Lively4, and using a locally running server without the Lively4 environment. This allows us to compare the technologies and to evaluate how Lively4 affects the rendering of points in the browser.

### 6.5.1 Benchmark process

To benchmark the technologies, there is an execution file that performs measurements. A measurement may include several metrics measured. In the execution file, we can define how many measurements should be made. The measurement results are then exported as a CSV file. These CSV files have a dimension of the number of measurements times number of metrics measured. The CSV files are then processed and merged by various Python scripts. The quantitative evaluation and creation of the charts takes place in Excel.

### 6.5.2 Benchmark Scenario

We benchmark the technologies only by taking internal measurements. As explained above, the metrics to be measured are limited to time and heap consumption. So we want to determine how much time is needed to render  $n$  points and how much heap is allocated by our benchmarks. In doing so, we perform measurements for different amounts of points.

Basically, we animate points during the measurements. To animate the points, we define an animation loop that repeatedly requests animation frames via the

browser API `window.requestAnimationFrame()`. We measure the time the browser needs to take up the animation frame and then returns from it by calling the passed method. Like this, we can measure how long the browser needs to render the points. Furthermore, in each iteration of this animation loop, we record the allocated heap size. All points get a random position at the beginning, i.e., an x-position and a y-position within a fixed area. We call this area the drawing area. When rendering span-elements, the drawing area is a div element; when rendering points within a 2D context, the drawing area is a canvas. In each iteration of the animation loop, the position of each point is changed. We increment both x- and y-positions by 1, and completely clear the drawing are in each iteration. Besides, we manage a counter variable that holds the number of iterations of the animation loop. So we can stop the execution of the script as soon as we have 100 measurements.

To be able to show comparisons and optimizations, we will adapt this benchmark script. We have executed this benchmark script for the different technologies and with a different amount of points to generate the following measurement series. Fewer test series were recorded for some technologies, which is due to the fact that the browser crashed due to overload while the benchmark script was being executed.

See section C.1 for the benchmark protocol. All code that was used for benchmarking can be found in section C.4

### 6.5.3 Explanation of Benchmark Results

Initial broad consideration of the measurement results (see C.2 for all results) reveals a dichotomy of the technologies, both in the measurement results for Lively4, and in the measurement results that were determined without Lively4. Thus SVG and span elements are in one group, and canvas rendering, i.e., 2D context and WebGL, are in the other group. The dichotomy becomes clear when we take a closer look at the percentage deviation of the measurement results. We add up the measured times and calculate the deviations between canvas rendering and rendering with SVG and span elements in Table 6.1.

In Table 6.1, we see that the speed difference between rendering on canvas and using SVG and span elements is enormous, especially as the number of points increases. However, as we can see in Table 6.1, when we compare rendering with span elements and rendering with SVG, that the differences are not significant.

In Table 6.1, it can also be seen that 800 points is a limit above which the performance differences of SVG and Span elements differ significantly from those of 2D context or WebGL.

The groups reflect exactly the above-described division of Immediate mode and Retained mode rendering techniques. We can clearly see the performance overhead of managing all rendered elements via the DOM. On the other hand, there is no more indirect rendering via the DOM when rendering on a canvas.

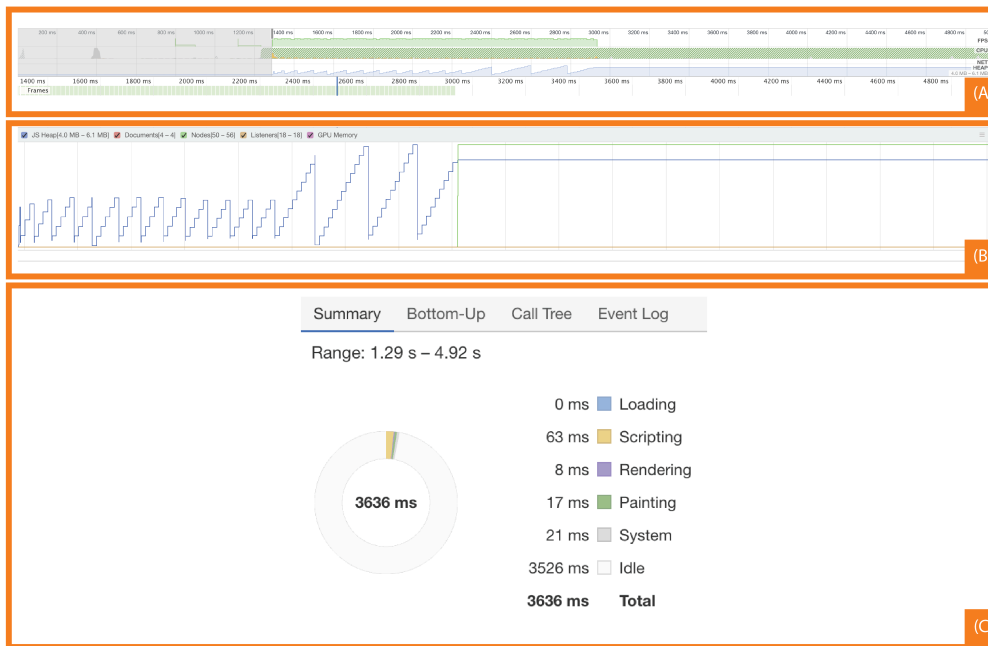
To see how the difference between Immediate mode and Retained mode rendering techniques affects the browser, we can take a closer look at the rendering process with the Chrome DevTools. To do this, we run the benchmark script for rendering span

	webgl vs. span	webgl vs. svg	canvas vs. span	canvas vs. svg
100 dots	2%	12%	-10%	-1%
200 dots	1%	1%	1%	1%
400 dots	2%	4%	0%	2%
800 dots	2%	7%	1%	6%
1600 dots	110%	124%	90%	104%
3200 dots	394%	431%	195%	217%
6400 dots	1012%	1086%	251%	274%
12800 dots	2191%	2286%	164%	175%
25600 dots	5081%	5173%	153%	158%
51200 dots	9443%	11086%	112%	149%

**Table 6.1:** Deviations of the sum of times to render. See Table C.1 absolute values

	span vs. Svg
100 dots	-10%
200 dots	0%
400 dots	-2%
800 dots	-5%
1600 dots	-7%
3200 dots	-7%
6400 dots	-7%
12800 dots	-4%
25600 dots	-2%
51200 dots	-17%

**Table 6.2:** Deviations of the time to render of SVG and span elements. See Table C.1 absolute values



**Figure 6.6:** Chrome DevTools when rendering with WebGL

elements as well as for rendering on a canvas with WebGL and record a performance timeline recording in parallel.

As we can see in Figure 6.6 (C), of the 3.6 seconds recorded, the browser spends about 8 ms in rendering. These 8 ms of rendering are needed to display the web content initially, in this case, the root div of the HTML element and the canvas (the drawing area). Also in (A), we can only see a green bar. This indicates a constant repainting of the page. The differences become even more apparent when we compare the performance timeline recording of the rendering of span elements in Figure 6.7.

Everything marked purple in Figure 6.7 (A) indicates that the browser was busy rendering content. We can see that the browser initiates a rendering every time the DOM is updated (in every iteration of the benchmark script). From the recorded 5.2 seconds in (C), the browser spends 1.6 seconds in rendering processes. When rendering on a canvas, the operating system is given a fixed area on the display, the canvas surface. With this, the operating system can render without the browser having to render all the time. This explains the performance differences between Immediate mode (the canvas element) and Retained mode (rendering using DOM elements). Also, as we can see in (C), ca. 2.9 seconds were spend on “Scripting”. This indicates, that when rendering points as span elements, we always have to iterate over every data entry to add a corresponding span element to the DOM. In comparison, when rendering with WebGL, we do not have to add nodes to the DOM; therefore we spend significantly less time in “Scripting” (See Figure 6.6 (C)).

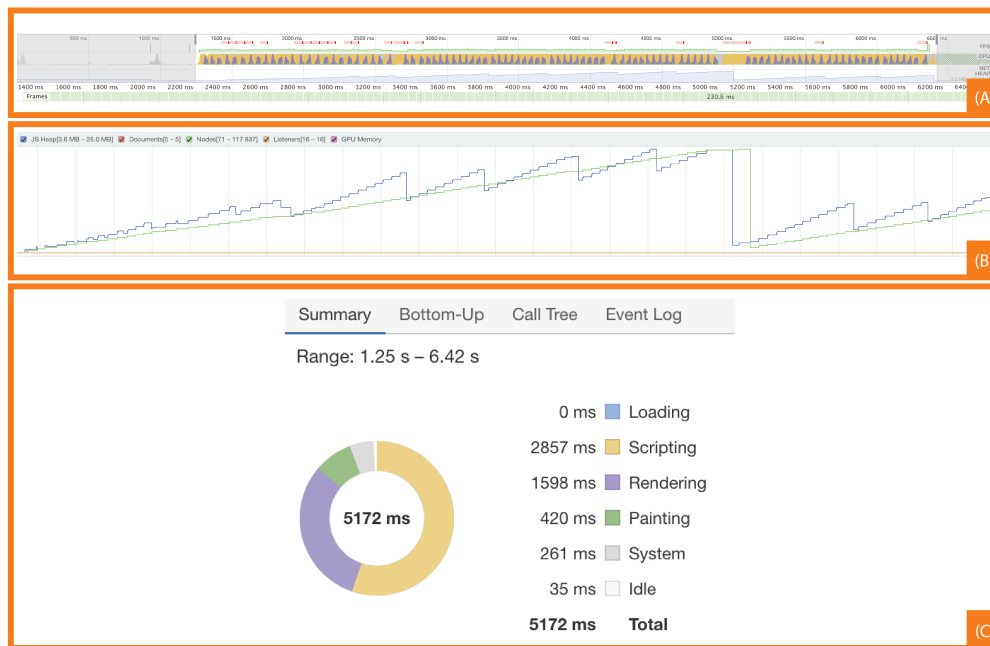


Figure 6.7: Chrome DevTools when rendering `<span>`-elements

Now we take a look at the heap consumption of the individual technologies. Again, we compare the technologies without going into the differences between rendering within Lively4 and rendering without Lively4. As expected, the same subdivision as above applies here as well:

In Table 6.3 we see, especially with a large number of points, the difference between rendering techniques of the Immediate mode and the Retained mode is visible. SVG and span elements must be mounted in the DOM, so they take up memory in the heap. Since the elements rendered on a canvas have no reference in the DOM, less memory is allocated in the heap.

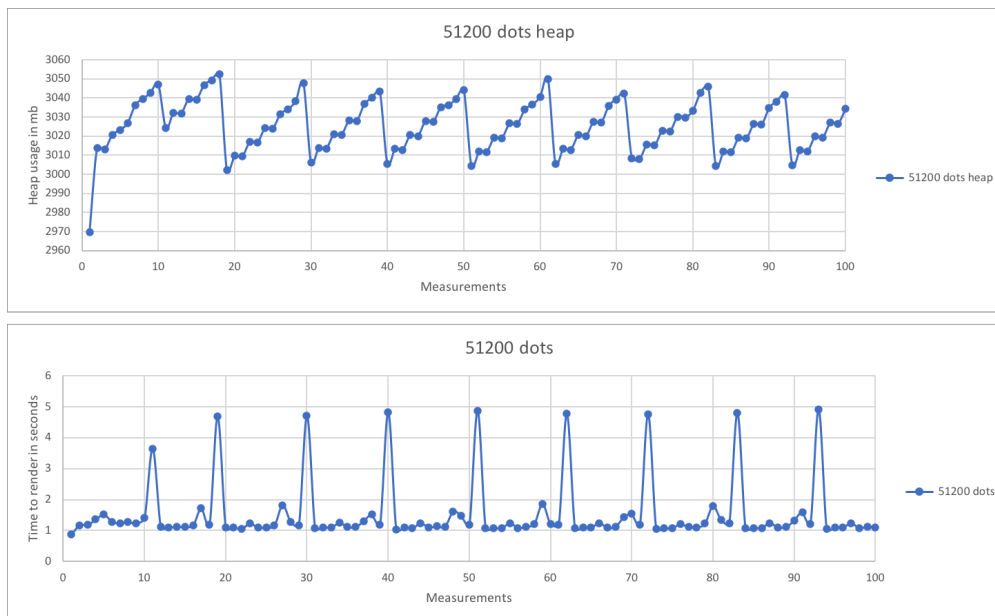
Another particularity is noticeable when we look the two metrics, both heap consumption and time to render, when rendering points with SVG (See Figure C.7 and Figure C.11). When looking at the rendering of 51,200 points with SVG elements in Lively4, it can be seen that there are always outbreaks in the time to render. We do not have a flat line, but very volatile measurements. If we look at the corresponding heap measurements, we can see that there are also such breakouts. The deflections are periodic in both cases. Every fifth measurement is a considerable deflection in the time to render (Figure 6.8).

In Figure 6.8, we can see that the rendering performance, or rather the smoothness of the animation, is strongly related to the memory used. In principle, we can observe the browser at work here. The performance outbursts can be explained with the browser's garbage collector. Chrome uses a generational garbage collector[55], which distinguishes between "young" objects and "old" objects. Deleting "young" objects is cheap and is done in idle processes of the browser. Deleting "old" objects is expensive.

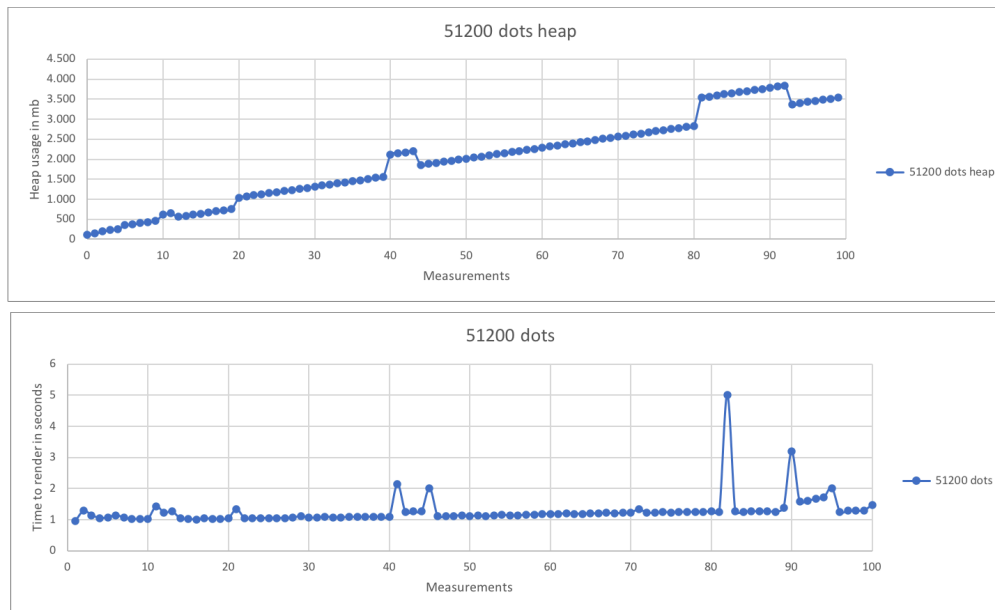


	webgl	svg	canvas	span
100 dots	8.93090123	8.55797841	8.29355524	17.4111138
200 dots	6.4900493	6.25474926	7.06782768	7.6813499
400 dots	7.02984786	8.5330229	6.8345605	6.01572303
800 dots	7.17516053	6.59598976	7.10518398	8.1429869
1600 dots	6.74618401	7.87780517	7.24223842	8.75730673
3200 dots	7.32184745	10.83753856	7.36471118	11.37400687
6400 dots	9.4806486	23.56274925	9.51280486	35.47085385
12800 dots	31.79363874	50.70004224	8.26959678	67.00521572
25600 dots	19.6146978	108.5658169	8.51574697	139.566563
51200 dots	47.12601747	176.3557883	11.88011913	306.6721561

**Table 6.3:** Heap consumption when rendering in mb



**Figure 6.8:** Heap usage and time to render when rendering point with SVG



**Figure 6.9:** Rendering with SVG without clean up of drawing area. Performed on a local server without Lively4

Here, a mark-and-sweep algorithm is executed, which, depending on the number of objects in the heap, can lead to significant performance degradation, as shown above. In our case, this is due to the structure of the benchmark script. As described above, we delete the content of the drawing area in each iteration. This results in 51,200 new SVG elements being allocated in each iteration. If we change the script so that the drawing area is not cleared, i.e., we save the references to the SVG elements and change the position directly at the reference, we get a completely different picture, as can be seen in Figure 6.9.

In Figure 6.9, we can observe that the garbage collector does not have to run the expensive mark-and-sweep algorithm for old objects once. In this case, much of the heap is allocated for iterating over the individual SVG elements, not for creating the SVG elements and mounting them in the DOM.

Now let us look at the differences between rendering in Lively4, and rendering Lively4, i.e., executing the script on a local server instance.

First, we compare the times. For this, we take the average time to render for each number of points per technology and calculate the percentage difference between execution in Lively4 and local execution without Lively4. In Table 6.4 and Table 6.5, we calculated the absolute averages, which have been rounded to 3 decimal places.

In Figure 6.10, we calculated the percentage deviation of the values between the local version and Lively4.

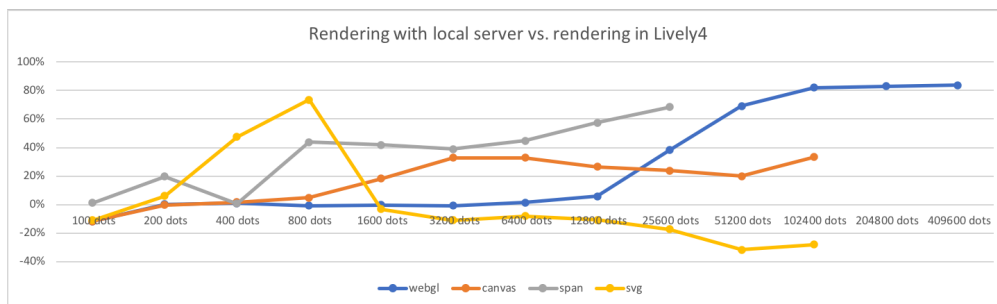
We have mixed results in Figure 6.10. With WebGL, there is hardly any difference, even with up to 12,800 points. From then on, the render times of WebGL on the local server are about 80% faster. With the 2D context of a canvas, the differences harden from 800 points on. The render times of 2D context are between 20% and 40% faster

	webgl-local	webgl-lively	canvas-local	canvas-lively
100 dots	0.016	0.016	0.019	0.017
200 dots	0.017	0.017	0.017	0.017
400 dots	0.016	0.017	0.017	0.017
800 dots	0.017	0.016	0.017	0.018
1600 dots	0.017	0.017	0.019	0.023
3200 dots	0.017	0.017	0.028	0.042
6400 dots	0.017	0.017	0.053	0.080
12800 dots	0.017	0.018	0.146	0.198
25600 dots	0.017	0.027	0.345	0.453
51200 dots	0.018	0.058	0.803	1.002
102400 dots	0.019	0.105	1.583	2.375

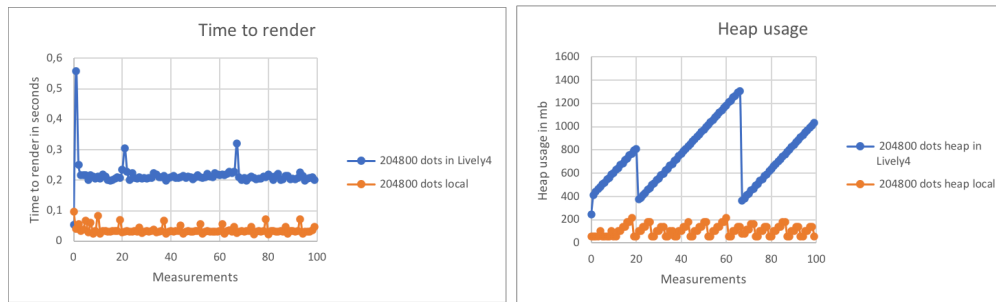
**Table 6.4:** Average time to render per technology: 100 measurements were taken.

	span-local	span-lively	svg-local	svg-lively
100 dots	0.017	0.017	0.018	0.017
200 dots	0.017	0.021	0.017	0.018
400 dots	0.017	0.017	0.017	0.033
800 dots	0.017	0.030	0.018	0.067
1600 dots	0.035	0.061	0.038	0.037
3200 dots	0.082	0.135	0.089	0.080
6400 dots	0.187	0.340	0.200	0.185
12800 dots	0.384	0.904	0.400	0.362
25600 dots	0.873	2.767	0.888	0.756
51200 dots	1.702		1.996	1.517
102400 dots			4.01	3.133

**Table 6.5:** Average time to render per technology: 100 measurements were taken.



**Figure 6.10:** Percentage differences in time to render in Lively4 and on local server



**Figure 6.11:** Comparison of rendering with WebGL in Lively4 and on local server

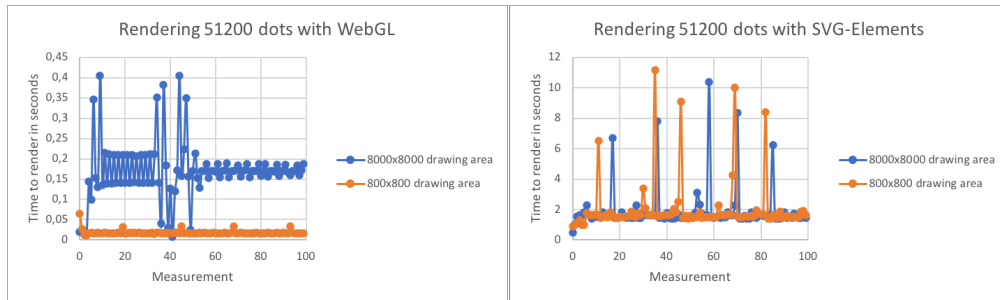
on the local server. When rendering span-elements, we can see the differences even earlier. From 400 points on, we can clearly see that the rendering of span elements is about 40% to 70% faster on the local server. Only the SVG elements are difficult to classify. Here we can observe that rendering SVG elements up to 800 points on the local server is up to 80% faster. But above 1,600 points, rendering SVG elements in Lively4 is 15% to 35% faster.

In Figure 6.11, we see the differences between rendering 204,800 points with WebGL on the local server and in Lively4. For this purpose, we examine the direct comparison of the respective measurements.

Again, we can see that the amount of allocated heap is strongly related to rendering performance. It is noticeable that in the case of Lively4, significantly more heap is allocated from the beginning. This is because Lively4 itself already consists of a large number of DOM elements. We can also see in the heap chart, that the measurement pattern is the same, but repeated at a different frequency. In the case of rendering without using Lively4, the browser seems to have a different garbage collection behavior. Here, after 6 to 10 measurements, a garbage collection process can be seen. In contrast, when rendering in Lively4, we see much more considerable deflections, and a more time-consuming garbage collection process is only noticeable after 20 to 46 measurements. The reason for this is unknown and is related to implementation details of the browser. If we compare the average of the time and heap measurements, we see that in Lively4, the average render time is 83% higher. At the same time, the average amount of allocated heaps in Lively4 is 86% higher than in the local scenario run.

Also, with span elements and the 2D context of the canvas, it can be seen that the page's higher memory consumption leads to worse performance (see section C.2). This discovery was also confirmed by the Google GMAIL development team[51]. Only with SVG elements, this assumption cannot be confirmed by our measurements. We repeated the measurements several times, both for SVG elements in Lively4 and for SVG elements on the local server, and the results can be reproduced. We suspect that this is due to the browser's caching behavior for SVG elements, but in this case, we cannot make a reliable statement.

The relationship between memory consumption and performance can be seen. We found that when rendering more memory-intensive technologies, such as span



**Figure 6.12:** Comparison of the impact of the drawing area size on performance

elements and SVG elements, the browser crashes significantly more often. Also, when rendering span and SVG elements in Lively4, the browser crashes at a much lower number of points. This is because the heap memory of a tab is limited in Chrome, and in Lively4 much memory is allocated to other system elements of the live programming environment from the outset.

Another visualization dimension is the drawing area size. The above benchmarks all took place in a fixed size of 800x800 pixels. In Figure 6.12, we changed the size to 8000x8000 pixels for two selected technologies and can compare the results for rendering 51,200 points. We use SVG and WebGL for this comparison.

In Figure 6.12 we see, while WebGL has experienced significant performance slumps, SVG performance remains constant. If we compare the average times and calculate the percentage difference per technology, the average time with WebGL is 89% higher on the 8000x8000 pixel canvas than on the previously used 800x800 canvas. Rendering of SVG elements was 6% faster on the 8000x8000 pixel drawing area than on the smaller drawing area used before.

To see how significant the performance differences of the technologies are between measurements on different days, we have conducted two benchmark series for rendering in Lively4 and running the benchmark script on a locally running server without Lively4. Both measurement series were performed with a time lag of several days. We made sure that the benchmark environment is the same.

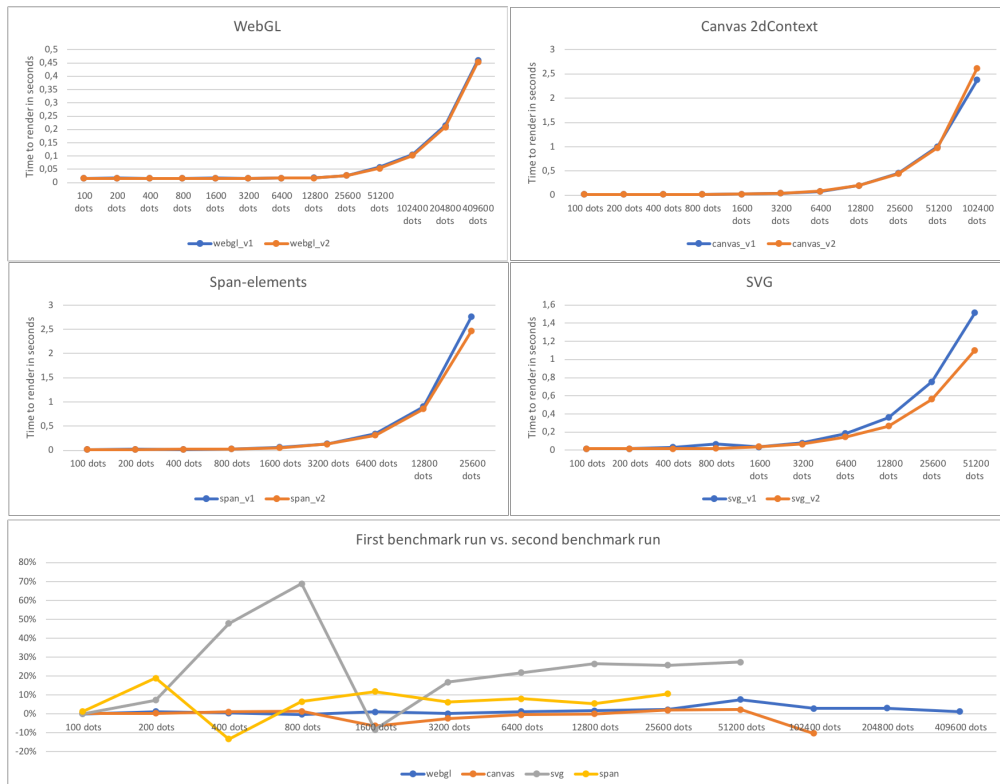
In Figure 6.13, we see the comparison of both measurements when rendering in Lively4. In the following, both measurement series are distinguished by the suffix “v1” and “v2”.

In Figure 6.13, we can see significant deviations. While the Immediate mode technologies, i.e., rendering on a canvas element, show a maximum deviation of 11%, SVG and span elements show much more significant differences. There is a performance difference of up to 70% with SVG elements when rendering 400 and 800 points. With span elements, a maximum difference of about 20% can be seen with 200 points; otherwise, the deviation is also limited.

Now let us look at the same series of measurements for rendering without Lively4 using a locally running server Figure 6.14.

In Figure 6.14, we can see that the differences are higher for all technologies except SVG than for rendering in Lively4. While Span elements show the most significant

## 6 Evaluating Visualization Technologies for Individual Data Points in Lively4



**Figure 6.13:** Comparison of the measurement series from different days using Lively4

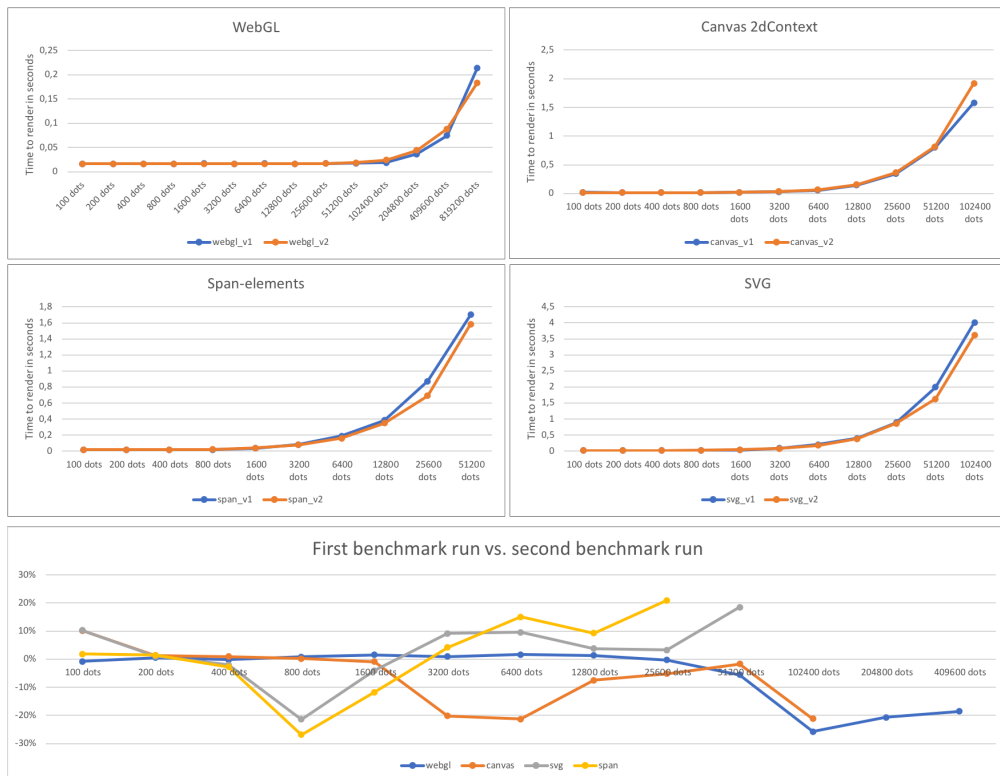


Figure 6.14: Comparison of the measurement series from different days using local server

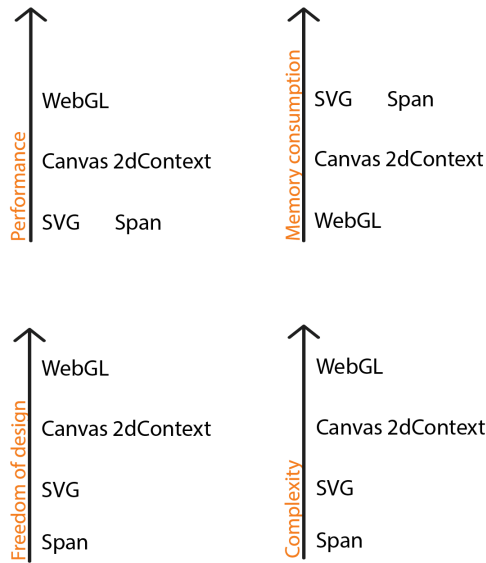


Figure 6.15: Rendering technologies and requirements

difference of about 21%, WebGL and SVG show about 28%. The performance of the individual technologies thus seems to be much more volatile on the locally running server without Lively4 than it is when rendering in Lively4.

#### 6.5.4 Discussion

In general, we could say that, as is usually the case in IT, there is an apparent trade-off between the complexity of a technology and its performance.

As we can see in Figure 6.15, we can arrange the technologies into the area of conflict of requirements discussed at the beginning.

WebGL is very performant and memory efficient, but also very complex and less accessible than the other technologies. Because we have to write our own shader code in WebGL, we can render all kinds of complex shapes, from 2D graphics to 3D graphics. So WebGL gives the programmer much freedom. Writing the code is much more complex and time-consuming than it was with SVG elements and debugging the shader code of WebGL (which is executed on the GPU) is more difficult because the error messages in the browser when executing shader code are much more limited than with JavaScript code. Also, the documentation and community content is less detailed and extensive compared to regular JavaScript content about the 2D context of HTML canvas elements or major libraries like d3.js.

In principle, much applies to rendering on a canvas using the 2D context as well, but in an attenuated form. Rendering on a 2D context is a bit more memory-intensive and not as powerful because we cannot use the GPU's parallelization technologies efficiently. On the other hand, the 2D context is not as complex as rendering with WebGL, because no shader code has to be written. Furthermore,



rendering sophisticated 3D graphics with the 2D context of a canvas is impossible, or only possible to a limited extent. Nevertheless, using the canvas element, we can access single pixels of the bitmap and manipulate them as we like.

On the other hand, SVG and span elements consume much memory, and the technologies are much less performant than rendering on canvas. With SVG, we can render all kinds of 2D shapes, but we cannot manipulate individual pixel values. With span elements, however, we can only render rectangles. The point is a particular form in this case because it is only displayed as a square with rounded edges. No other shapes, besides rectangles, are possible. Compared to the canvas element, the freedom in design is thus significantly restricted. On the other hand, these technologies are also less complex, especially the span elements.

Less sophisticated technologies, such as SVG and span elements, have a higher memory consumption, which in most cases, also leads to performance losses. The above-listed differences between rendering in Lively4 and rendering on a local server also show that the choice of technology depends on the exact circumstances. While rendering span elements can be useful in an unspoiled environment, the canvas element is more appropriate in an environment where a range of other web content is already present.

As shown in the benchmarks above, the influence of the visualization environment is enormous. Especially when visualizations on the Web are not rendered in isolation, but are part of a Web application, or an environment where other content is also rendered, the memory consumption and performance of the technologies plays an increasing role. To avoid negatively affecting the rest of the system's usability, we should use the most memory-efficient technology.

The different technologies have different strengths and weaknesses, which means that an effective combination of these technologies is usually required in a specific application. In many cases, we have used WebGL only to render points. This allowed us to keep the shader code and reuse it. The axes of some diagrams were then drawn by absolutely positioned SVG elements located above the canvas. Another technique we used was Canvas Layering. Here, a concrete use case was the grouping of points and making this grouping clear by coloring the background (colored polygons). The rendering of the points was done with WebGL, but the more complex rendering of the filled polygons was done on a canvas with a 2D context. This allowed us to take advantage of WebGL's performance when rendering many thousand points while rendering more complex shapes (with less memory consumption) with a more documented and accessible technology. Furthermore, some of our visualizations work with annotations, such as text fields that appear when the users click on a point. The content of these annotations must be dynamically created and changeable. For this, we have used absolute positioned span elements. This way, we could easily access the ID's of these span elements via the DOM and change their content. Furthermore, rendering text in a canvas is expensive and cumbersome, but in a native spa -element, it is less complicated and much more effective through browser optimizations.

To combine the strengths of all technologies, reduce weaknesses, and respond to the surrounding system, it is necessary to effectively combine the technologies and work out their use cases within the visualizations.

## 6.6 Future Work and Further Optimizations

### 6.6.1 Remaining Problems

Many of our visualizations are very resource-demanding for the browser. As we have seen in the benchmarks, it is quite challenging to render, animate, and interact with more than 100,000 points on the web. Since the different visualizations highlight different focal points of the data and thus the underlying topic, it is helpful, if not necessary, to view several of these visualizations in parallel. With the application we have built, we can open several visualizations, simulate connections between them, and data flow through these visualizations.

One problem originates directly from the programming language JavaScript used on the web. JavaScript works single-threaded.<sup>28</sup> This means that when complex calculations and functions are executed, they block the rest of the client application. This leads to the problem that several complex visualizations take away execution time from each other. The blocking behavior of long-running JavaScript functions, in turn, leads to the user not being able to interact smoothly with the application, and the animations in various visualizations stall. Especially in a live system like Lively4 this leads to limited usability of parts of the system. So it happened quite often that we had to reload Lively4 several times during programming and testing our applications to continue working with the system. A significant part of the problem were visualizations that work with simulations (long running animations). In such simulations, we have to interact with the data, and the visualization is a continuous animation of the points. If we open one of these simulations, the remaining visualizations and the whole system performance has decreased noticeably (especially with a high number of points).

When rendering points on a canvas using WebGL, the GPU is strained. This means there is a hardware limit that is not easily fixed. Different devices have different graphics cards; therefore, the graphic quality of the visualizations depends on the end device. As web developers and visualization designers, we have no influence on the solution of this problem.

While we can solve the first problem by applying further optimizations at the code level, the second problem cannot be solved by us.

---

<sup>28</sup><https://dev.to/steelvoltage/if-javascript-is-single-threaded-how-is-it-asynchronous-56gd> (last accessed 2020-07-20).

### 6.6.2 Further Optimizations

To get around the blocking behavior of the single-threaded JavaScript, we can use Web Workers.<sup>29</sup> Web Workers are a way to execute JavaScript code in the background of an application. This means a web worker can execute complex, long-lasting functions without blocking the UI, user interaction, and scripts in the main thread. A Web Worker is just a JavaScript object that executes a JavaScript file in an independent thread. Communication between the Web Worker and the actual client application takes place using messages.[8]

In our case, expensive calculations regarding the data used could be outsourced to such a worker without blocking the client application.

Another technology that became available in Chrome in 2018 is OffscreenCanvas. A disadvantage of Web Workers is that we cannot manipulate DOM elements from within them. Since the Canvas API is bound to the canvas element, it was impossible to execute the rendering instructions of a canvas element via a Web Worker. The OffscreenCanvas breaks the binding between the Canvas API and the DOM and makes it possible to execute rendering instructions in a web worker [31, 56]

```
1 const offscreenCanvas = document.querySelector('canvas').
  transferControlToOffscreen();
2 const worker = new Worker('rendering.js');
3 worker.postMessage({ offscreenCanvas }, [offscreenCanvas]);
```

Calling the method `transferControlToOffscreen()`, we get the instance of an offscreen canvas and can pass the reference of this offscreen canvas to the Web Worker we created. The Web Worker executes the JavaScript code specified in the file “rendering.js”.

In our case, we could outsource the rendering of entire visualizations in several threads running in the background. This would significantly reduce the load on the main thread, and the usability of the live programming environment Lively4 would be completely preserved. While this would create further indications and increase the complexity of the application code (by communicating via messages, propagating data updates), the user experience would be much better, as shown in an example by Chris Price.<sup>30</sup>

To make our visualizations more accessible, this would be a useful optimization approach.

## 6.7 Conclusion

In this chapter, we considered different technologies to display visualizations in the browser. This explicitly involved rendering up to 100,000 points on the web. We also looked at the rendering process of the browser to specifically address

<sup>29</sup><https://www.w3.org/TR/workers/> (last accessed 2020-07-23).

<sup>30</sup><https://chrisprice.io/offscreen-canvas/?100000> (last accessed 2020-07-28).

optimizations and explain the differences between the technologies used to render the visualizations. To discuss the advantages and disadvantages of the technologies and to evaluate their performance, we performed benchmarks both in the live programming environment Lively4 and on a local running server with a clean environment.

Four technologies were investigated, span elements, SVG, and the canvas element, both with 2D context and with WebGL context. It has been shown that WebGL is by far the most performant technology. Furthermore, WebGL offers the greatest design freedom but is also much more complex than the other technologies. The 2D context of a canvas is more accessible than WebGL, but also not as performant. Nevertheless, one can manipulate single pixels in the 2D context and has a lot of design freedom. Span elements and SVG elements have the highest memory consumption and are also much less performant than rendering on a canvas. Furthermore, the design freedom is very limited for SVG elements, but especially for span elements.

We found that the higher the number of points rendered, the more noticeable the technologies' performance differences. We were also able to see the relationship between the memory usage of a technology and its performance in the browser. Furthermore, by running the benchmarks in different environments, we were able to observe the influence of a web environment or web application on the performance of individual technologies. Our visualizations combined many of the technologies by evaluating their strengths and weaknesses, and weighing the complexity of a technology against its performance value for the corresponding use case.

The rendering of complex visualizations poses some specific problems. Multithreading approaches for JavaScript and the possibility to spread the rendering of visualizations into different threads is a promising way to further improve the performance of our visualizations and the interaction flow between user and visualization.

## 7 Visualizing Africa’s Voices: Evaluating Our Individual-centered Approach to Visualize People’s Opinions and Demographic Data

The goal of this project was to design and validate novel visualizations that among other things enable an interactive exploration process and create empathy. This chapter evaluates the approach we used to achieve this goal through some of the prototypes we built. We present walkthroughs of three prototypes. They show how to interact with the prototypes and how to gain insights from them. We also present a value-driven evaluation of the prototypes, which among other things discusses the possible questions that can be answered with them. Finally, we discuss how we solved the problems Africa’s Voices faces when using visualizations. This discussion highlights how an individual-focused approach can help to create empathy and trust in the data. Furthermore, we analyze which interactions our visualizations use and how they differ. We discuss how our visualizations enable fast feedback cycles to help in the exploration process.

### 7.1 Introduction

With this project, we pursued the goal of designing and validating novel visualizations. Those visualizations should, on the one hand, provide an interactive and explorable way to analyze multi-dimensional data and, on the other hand, make the process more empathetic towards the individuals from which the data comes. These goals are in contrast to the ones more conventional data visualizations try to solve. Those visualizations want to provide a clear view of the information and, therefore, abstract the underlying data cases away. In this chapter, we evaluate the prototypes we built with regard to these goals.

While we constantly tested our ideas and prototypes with our project partner at Africa’s Voices, we use this chapter to more thoroughly investigate the benefits of our visualization and our approach for its use case. As we aimed to support the exploration of data, this chapter illustrates how users can explore and answer questions through the visualization prototypes we built. Further, we evaluate how our prototypes differ from analysis with “standard” visualizations.

To this end, we summarize the core problems the data from Africa’s Voices presents, standard visualizations for Africa’s Voices pose, and describe the difficulties

of evaluation of visualizations as well as our evaluation method in section 7.2. To show how our prototypes work and what their functionalities are, we present 3 walkthroughs in section 7.3. In section 7.4 a value-driven evaluation of our prototypes is shown according to four categories. Coming back to the problems summarized in Section 2, we discuss our solutions to them in section 7.5. We conclude section 7.6 with an outlook on the impact our work has for further research at Africa’s Voices.

## 7.2 Foundations

In this section, we summarize the problems regarding

1. the data we used,
2. the domain where the data comes from and the intended use case for our visualizations provided, and
3. evaluation of visualizations.

### 7.2.1 Problems with Data

The data Africa’s Voices collects and which we want to visualize has two main foci: demographic and thematic information. While the demographic information includes attributes such as age, gender, geographic region, language, and whether a person is recently displaced, the thematic information includes the thematic coding of the answers from the respondents. This variety of information already shows that the data is high-dimensional, which is necessary as the data is a condensed form of what in “grounded theory” is called “thick data”. “Thick data” includes accounts of personal experience from respondents and records that provide narratives of experience [16]. This information should be grounded in the context of the respondents to understand their perspectives.

To display high-dimensional data, there are three main ways to do so: a) display a high dimensionality visualization, b) project the high-dimensional data to a lower dimensionality, or c) show multiple low-dimensional visualizations [33]. Understanding and displaying the demographic information is relatively easy, especially when compared to the challenging thematic codings. Each individual respondent has multiple themes out of all possible codings. That way, the data can be up to 50-dimensional. In information retrieval, a field which deals with similar data, these kinds of codings are often modeled in an n-dimensional vector space, where each dimension represents a theme [70]. While this makes calculating easier, this representation is not suited as a visualization. This high-dimensionality of the overall data and the thematic information, in particular, is a core problem when visualizing the kind of data Africa’s Voices works with.

Another core problem is the amount of data Africa’s Voices wants to explore. Several projects have thousands of respondents with many more messages. Future projects are planned to include even more respondents.

### 7.2.2 Problems from Partner and Domain

In chapter 1, the main problems Africa’s Voices faces when working with visualizations are explained. Those problems are either part of the domain Africa’s Voices is working in or part of the goals they want to achieve when working with visualizations. Here we summarize the problems. For a more detailed description, refer to subsection 1.1.3.

**Slow Feedback Cycles Due to Their Current Process** The data analysis process of Africa’s Voices involves manual data management and copying data from old into new spreadsheets. There are many steps between getting the data and creating a visualization. These steps result in a slow feedback cycle. A slow feedback cycle is detrimental to a data exploration process, where it is crucial to generate questions and try to validate the underlying hypotheses quickly.

**Generalization of Findings Due to Quantitative Data** “Standard” visualizations such as bar charts, pie charts, or bubble charts all display aggregated data, that means summary statistics based on all the data points. To be able to aggregate data and therefore evaluate statistical values, the data has to be quantitative. Those aggregations can suggest a generalization to the whole population. This generalization is problematic because a) the respondents are self-selected and therefore not representative for the whole population, and b) its focus on statistical summaries neglects the context in which each respondent lives.

**Viewing Single Individuals and Original Opinions** For the data Africa’s Voices collects, the original messages are crucial. They include the richest information and are necessary to convey Africa’s Voices’ insights into the presentation of their findings. However, to analyze the messages further and have a chance to visualize them, they must be encoded. When encoded themes are used in the visualization, it is challenging to keep the connection to the original message. Nonetheless, the connection from the visualization to the original message is essential.

**Trust in Data** Since Africa’s Voices wants to present their insights to decision-makers and the population, it is vital to trust the data presented in a visualization. For analysis, manipulation of the data like filtering is necessary. In their current workflow, this is tedious and involves much manual work. The manual work, in turn, cannot be easily reproduced. AVF logs all interactions with the data in a provenance tracing (refer to 5.2.4.1 for a discussion of these tracings), but the tracing ends before the data analysis step. Showing the trace information in the visualization and adding the manipulation steps from the analysis could significantly increase trust in the data and insights from it.

**Trust in Software** Going further than that, providing trust in the software by allowing for code inspection enables even greater transparency.

**Connecting Visualizations** To show multiple perspectives and aspects of the data, multiple diagrams are needed. If they are not connected, it is not easy to see for the users whether they contain the same data. In this regard, they have to trust the software and visualization creator. Furthermore, for most visualizations, it is difficult to interconnect data points across visualizations and draw conclusions from those comparisons. This difficulty is in part because “standard” visualizations use their own graphical elements. Lastly, if different visualizations do not share a similar design language, switching between them is difficult, especially for people without an analysis background. Thus, it is easier to understand connections in the data if different visualizations are explicitly connected. Added to that, it helps if they look uniform and support the same kind of interactions.

**Missing Interaction** With static diagrams, users only have one view of the data. Without interaction, the chance of direct and immediate exploration is lost. They would have to load the whole data into another diagram and build it again. Interacting with the visualization and, therefore, the data could be most helpful for researchers. They would have more direct access to the data, which leads to a completely different exploration and analysis process. Additionally, interactive diagrams could significantly help people who get presented the data understand the relationships between multiple views of the data.

**Missing Empathy** Most visualizations focus on showing the data for a specific task, for example, comparing two or more groups with a bar chart. In the case of survey data, the AVF also needs to emphasize that the data is highly personal and sensitive. Especially when presenting to decision-makers, but also during the analysis process, it is essential to create empathy for the respondents, to help understand the underlying issues. Therefore, visualizations should aim to help to foster it.

**Issues With the Mindset** When using “standard” visualizations, researchers tend to stay in familiar thought processes. Visualizations should inspire new questions and unconventional thinking.

### 7.2.3 Problems of Evaluation of Visualizations

In the field of human-computer interaction, the evaluation of tools and systems often means assessing the usability of the system or its interface. This evaluation happens through a series of benchmark tasks in which researchers evaluate whether a user could use the system to achieve a predetermined result [41]. This approach has also been proposed for evaluating visualizations [2]. However, task-focused evaluation is not sufficient for evaluating visualization systems and fails for several reasons:

- Factors of individual experience and skill in the analysis, as well as the specifics of the given data, influence performance within a given visualization system [40, 72].



- The familiarity of users with traditional interfaces: Even after extended training, it is tough to overcome the bias caused by familiarity with traditional interfaces [2].
- The task-focused approach fails to evaluate the fundamental aspects of visualizations [62].

A task-focused evaluation approach shows whether a system is learnable and comprehensible for potential users and whether users can use it to answer a specific set of questions about a specific data set. However, this approach fails to encompass all the advantages a visualization should bring to users. The goal for the usage of a visualization system is often not only to answer a specific set of questions but to explore the data set and get a “big picture” understanding of the data. Additionally, the system should help generate insights about the data beyond specific data values.

To emphasize the last argument, consider an example data set about laptops and their respective attributes, which means price, performance, storage, wattage, et cetera. After building a visualization system for this data set, we can evaluate the system by asking specific questions like:

- What is the most performant laptop?
- Which laptop has the best price-performance ratio?
- Does storage correlate with performance?

While answering those questions certainly gives information about the visualization system, those data queries do not show the benefit of the visualization. Most of those questions can be answered with a spreadsheet, maybe even quicker. The benefits of using a visualization are at a more elemental level [62, 2].

These problems are the reason why we opted to use an approach introduced by John Stasko called “Value-driven evaluation of visualizations” [62]. In essence, this approach entails a qualitative analysis of four factors:

1. the time needed to answer a wide variety of questions about data
2. the spur and discovery of insights and/or insightful questions about data
3. the ability to convey the overall essence of data
4. the ability to generate confidence, knowledge, and trust about the data, its domain, and context

## 7.3 Walkthroughs

In this section, we present walkthroughs for three prototypes. We describe how to answer emerging questions in a scenario and show how to analyze data with our prototypes. The walkthroughs are not only proof of work in that they show that our visualizations can answer and generate questions and enable an explorative workflow; they also showcase how users might use the visualizations. Additionally, they serve as a starting point for further discussions in later chapters.

After explaining the setting, we look at three prototypes. Those are *Tab View*, *Tree View*, and *Individual Center*. For a more in-depth description of the involved design decision, refer to sections 3.5.2, 3.5.3, and 3.4.4 respectively. The first two are the

integrated visualizations; the last one shows another direction of ideation. The three prototypes represent the approaches of the remaining prototypes reasonably well.

### 7.3.1 Setting

The prototypes work in the Lively4 environment described in chapter 4. The domain in this example is data about the Covid-19 pandemic. Africa's Voices Foundation ran radio shows regarding the hopes, fears, and questions of the population about Covid-19 collected the answers via text message and coded the answers with themes. For a more in-depth explanation of their general process, see subsection 1.1.2. One of the data sets from a radio show in Somalia is the focus in the following walkthroughs.

The walkthroughs have three parts, each covering another prototype we developed: (1) *Tab View*, (2) *Tree View*, and (3) *Individual Center*. Each time, the goal is to explore the data set and gain meaningful insights about the data and respondents. That means the goal is to discover interesting trends and aspects of the data about the respondents' demographic makeup, as well as what the respondents said and possible connections between those two. Consider always to be careful about generalizing the data to the total population: participants are self-selected and biased by access to radio shows and the possibility to send text messages.

### 7.3.2 Tab View

By examining all respondents' demographic information, we can gain insights into the representation of different groups in our data set. There are two different visualization types in this prototype to help inspect demographic information: *XY Diagram*, which is a mix between a bar chart and scatterplot, and a *Map*. To have the most flexibility in what groups to view, we look at the *XY Diagram* first. In this visualization, selecting appropriate demographic attributes as grouping criteria is possible to see different distributions. To see the split into male and female respondents, we group by gender. The result looks similar to a bar chart and is shown in the screenshot Figure 7.1. In contrast to a regular bar chart, it is still possible to inspect any individuals' opinions in the distribution. To get a sense of some individual respondents, we click on some points, each representing a person who answered by text message. This interaction allows users to inspect the demographic information and the answers to the questions from the radio show in an inspector window. The interesting aspect of this interaction is that it allows for a qualitative analysis inside the quantitative analysis. By inspecting the individual respondents, users connect specific knowledge about a few individuals to the general knowledge gained from the overview distribution. For example, by selecting some individuals, we find out about two interesting themes in male and female respondents: "call for right practice" and "religious practice".

In the *XY Diagram*, it is also possible to group along the y-axis to incorporate a second dimension. In this view, users can estimate correlative dependencies in the data. We group the y-axis by age and get the following view, as shown in Figure 7.2. In this view, users can see which age-gender group responded most often. The scale

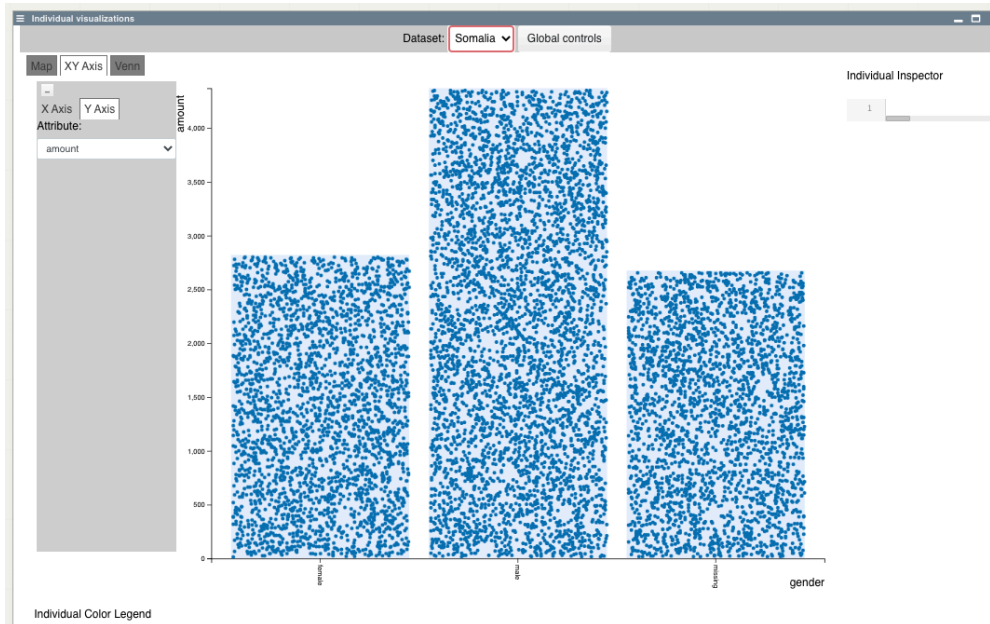


Figure 7.1: Dataset grouped by gender

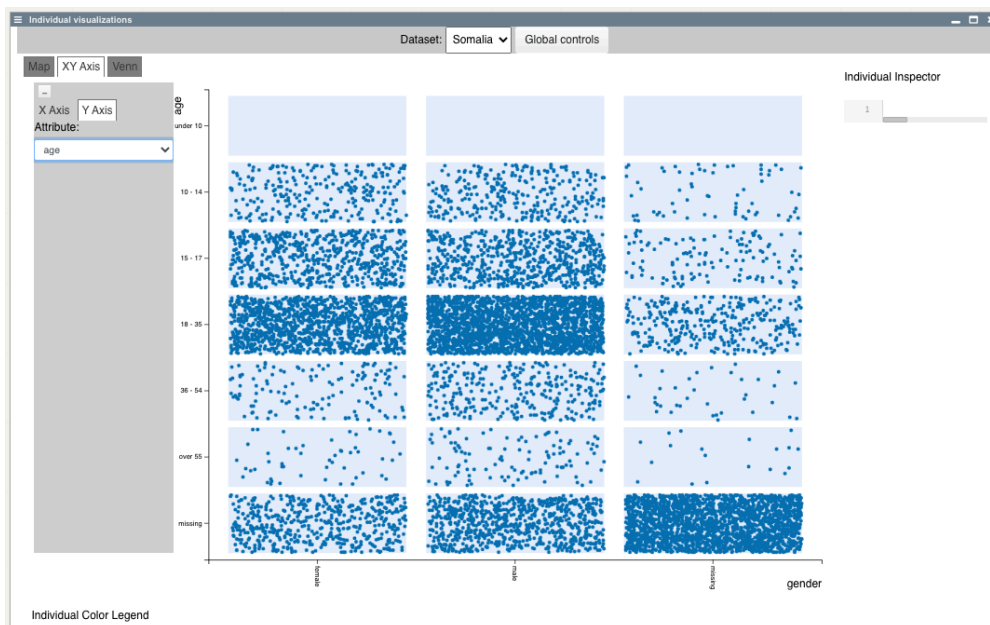
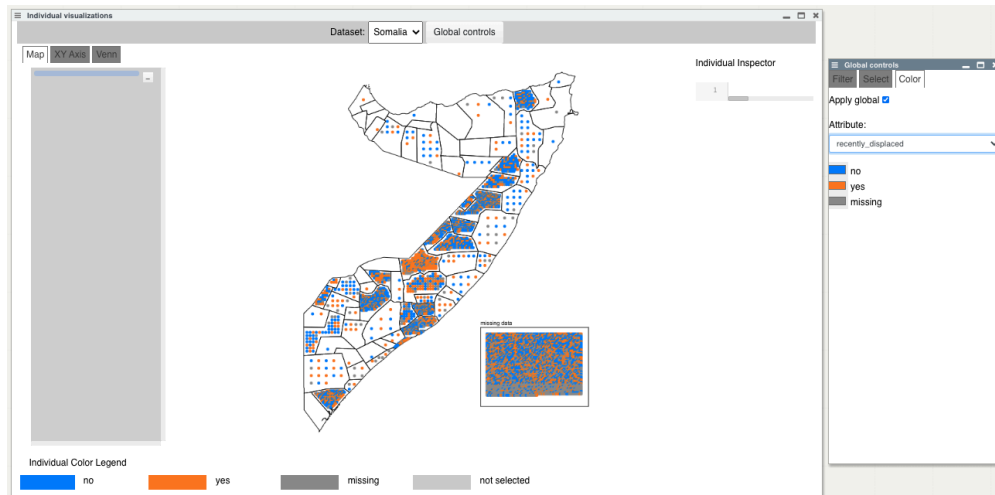


Figure 7.2: Dataset grouped by gender and age

of respondents in each age-gender bucket depends on the density of the points in the box. With this metric, it is not easy to discern small differences or accurately tell the number of individuals in each box. However, it is possible to recognize bigger trends. In the example, we see that the age group from 18-35 is the most prominent for male and female respondents, while more participants were male than female.

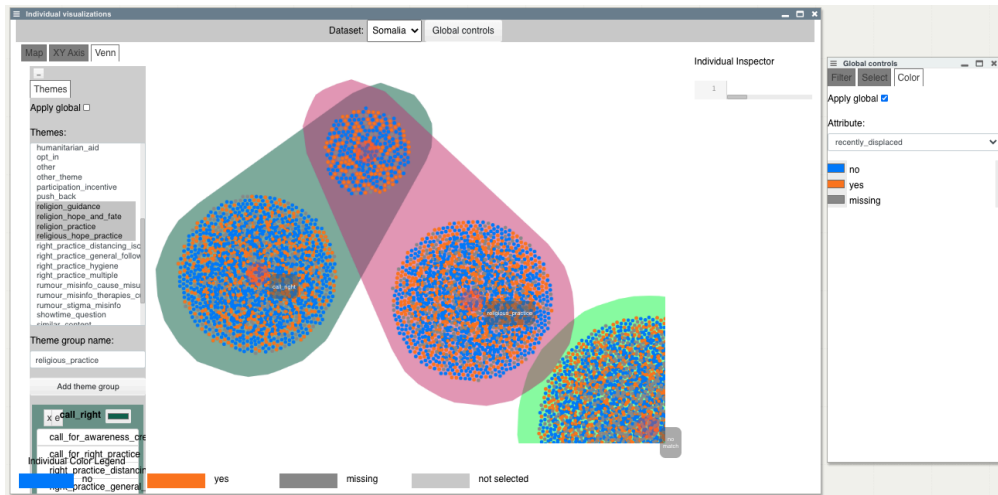


**Figure 7.3:** Map, colored by recently displaced

To see a geographical distribution of the respondents, we switch to the *Map* view and see all individuals arranged according to their home district. Once again, the scale is the district's point density, which depends on how many individuals are in each district. To add another dimension to this diagram, users can use color all dots according to one attribute. To find out whether there are regions where there are more displaced people than in others, we color all individuals on the map according to their displaced status. The result is shown in Figure 7.3. We can see that there is one district with many displaced people in the center of Somalia. This is a fascinating insight, which we can follow up on later on.

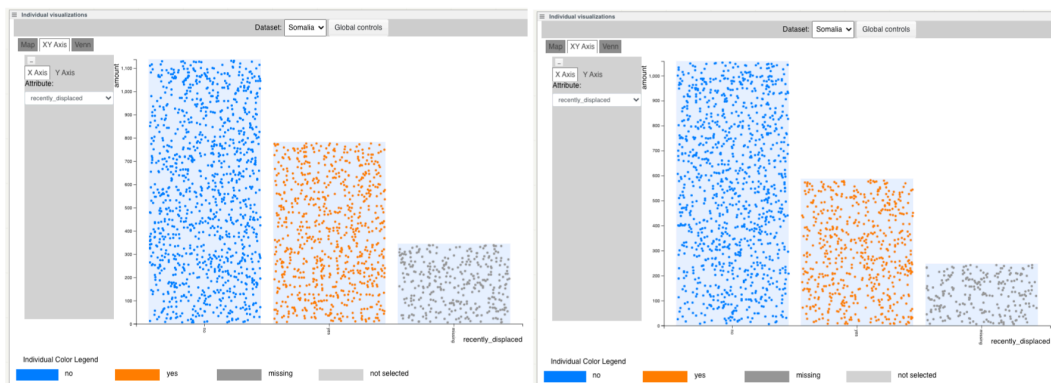
The information about what was said is represented in our data by the thematic coding. By looking at individuals earlier, we found two interesting themes: the “religious practice” and the “call for right practice”. To visualize these groups, we switch to another diagram, an adapted Venn diagram. In this diagram, it is possible to select all themes that belong either to the “religious practices” or to “call of right practice” and group them (see Figure 7.4). There are three groups of people: one group for people who mentioned “religious practice”, one for respondents who mentioned a “call for right practice”, and one for people who mentioned both. The size of the circles corresponds to the number of individuals who mentioned the corresponding theme combination. Like the density metric, this metric is better suited to detect more prominent trends than to make accurate comparisons. In the example, it is possible to see that both standalone groupings attract about the same amount of individuals. However, there are slightly more individuals who only talk about “religious practice”.

To find out how the demographic makeup of respondents mentioning specific themes looks like, users can choose a demographic distribution to look at in the *XY Diagram* and filter all individuals according to the mention of those themes. This process is limited to specific demographic attributes and themes. Therefore, users should know which themes are interesting to them. In the exploration process,



**Figure 7.4:** Venn Diagram, groupings for themes ‘call for right practice’ and ‘religious practice’

“recently displaced” and the themes “religious practice” and “call for right practice” were particularly interesting, so we focus on the possible correlation of those categories. To achieve this, we (1) switch to the *XY Diagram*, (2) filter by one of the themes, (3) set the x-axis to represent displaced status, and (4) set the y-axis to display the number of individuals. We repeat this process for both themes. The results are shown in Figure 7.5. By comparing the height of the bars, it is possible to see how the demographic distributions of people mentioning specific themes differ. Interestingly, it is possible to easily see relative sizes in the distribution, since the bars’ heights are normalized to the height of the biggest bar. In Figure 7.5 we can see that displaced people are more likely to mention “religious practice” than “call for right practice” than non-displaced people.



**Figure 7.5:** Dataset grouped and colored by recently displaced: filtered by the mention of ‘call for right practice’ (right), filtered by the mention of ‘religious practice’ (left)

This comparison leads to the conclusion that when tackling the coronavirus crisis in Somalia, it can help to work with religious authorities to ensure health messaging is effective. This insight is especially true when trying to communicate with internally displaced people.

In this walkthrough, we were able to gain different insights about the data and the respondents. The insights we gained concern the demographic as well as the thematic information. We got insights into who the respondents are and also what they are generally talking about. Additionally, we could infer some combined thematic and demographic information, which can help communicate with the population. That means we were able to achieve our goal for this walkthrough.

### 7.3.3 Tree View

The next prototype we examine incorporates the same basic visualizations as the previous one but combines them differently.

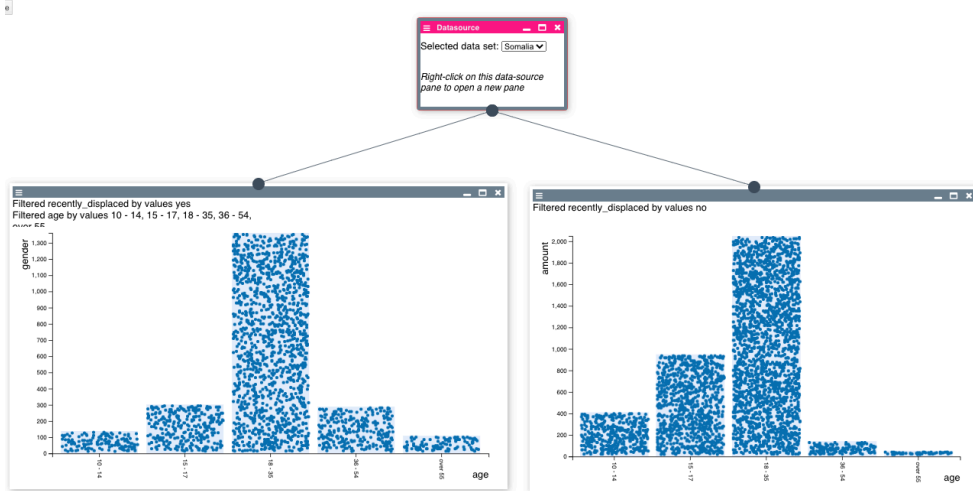
Since it served us well, we use the same data exploration structure as in the last prototype: Starting with an analysis of the demographic information followed by the thematic information. An important difference to the previous prototype is that the visualizations users look at remain visible throughout the exploration process. They form a tree-like structure, where users can see previous steps and, more importantly, make direct comparisons between similar visualizations and even across data sets.

We start the exploration by looking at the age distribution of people who were recently displaced and those who were not. This comparison is achieved by (1) creating two visualizations, (2) filtering according to displaced status, and (3) group them according to age. The result is shown in Figure 7.6. By looking at the visualizations side by side, the differences in the age structure become very apparent: The group consisting only of recently displaced people has a lower part of younger respondents and a higher proportion of older respondents. This comparison is much easier to do than in the previous walkthrough; viewing multiple visualizations at once moves the work from the users to the visualization system.

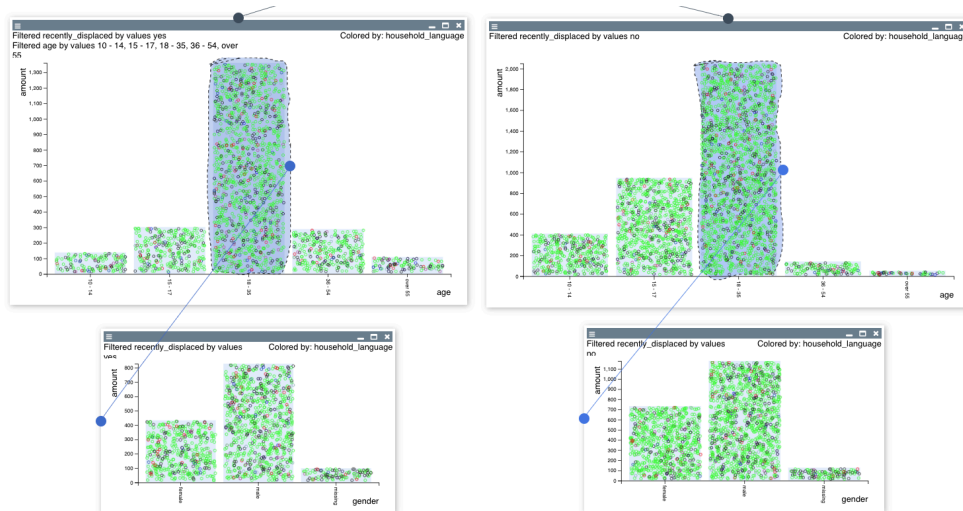
To understand the groups' internal structure more, it is necessary to add more dimensions to the visualization. In this prototype, there are the following ways to do that:

- Filtering,
- Coloring, and
- Grouping along the Y-Axis.

All of these interactions were shown in the previous walkthrough. However, in this prototype, there is another way of filtering: a lasso-select tool, which allows users very quickly to "zoom in" on a specific group. To see the gender split of the corresponding individuals aged 18 to 35, we (1) select it with a lasso-selection tool, (2) create new visualizations, and (3) group them by gender. Due to the side by side comparison, we notice the differences easily: The percentage of female respondents from all the respondents, who were recently displaced, is lower than the one from all respondents, who were not recently displaced. As in the previous prototype, it is possible to select single individuals to see all demographic and thematic information,



**Figure 7.6:** Age distributions of recently displaced (left) and not recently displaced (right) respondents

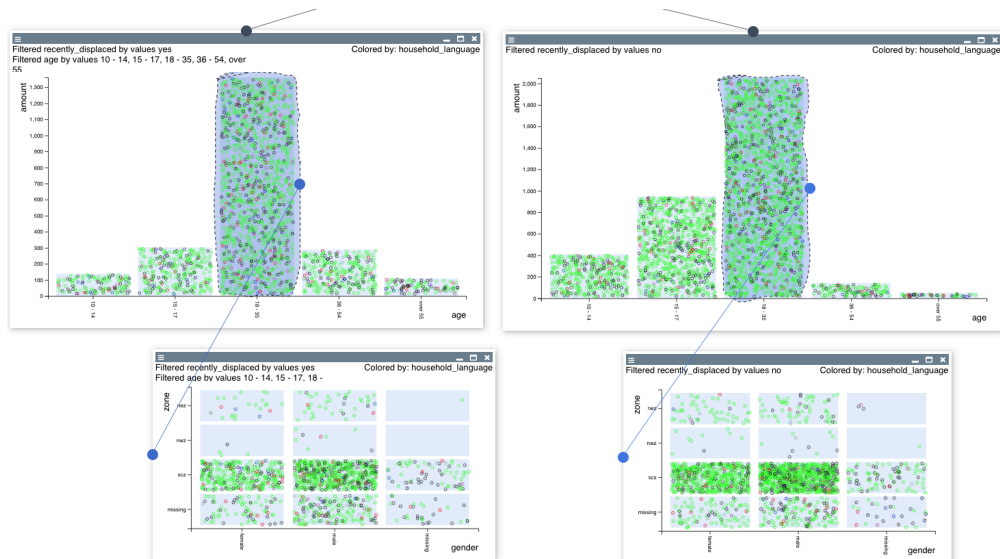


**Figure 7.7:** Different kinds of dimensionality expansions. Left visualizations include recently displaced, right visualizations, not recently displaced people. Colored by household language, neon green is Somali.

and if the information is available, also the written messages. In this prototype, the selected individual is also highlighted across all other visualizations in the tree. Users can get a sense of the context she is in and how big the corresponding groups are. This highlighting is an interesting aspect because it enables users to connect the visualizations qualitatively.

To see how the spoken household language differs in the shown groups, we color all individuals according to this attribute (see Figure 7.7). The coloring is applied to





**Figure 7.8:** Left visualizations include recently displaced, right visualizations not recently displaced people. Colored by household language, neon green is Somali. Bottom two visualizations correlate zone by gender.

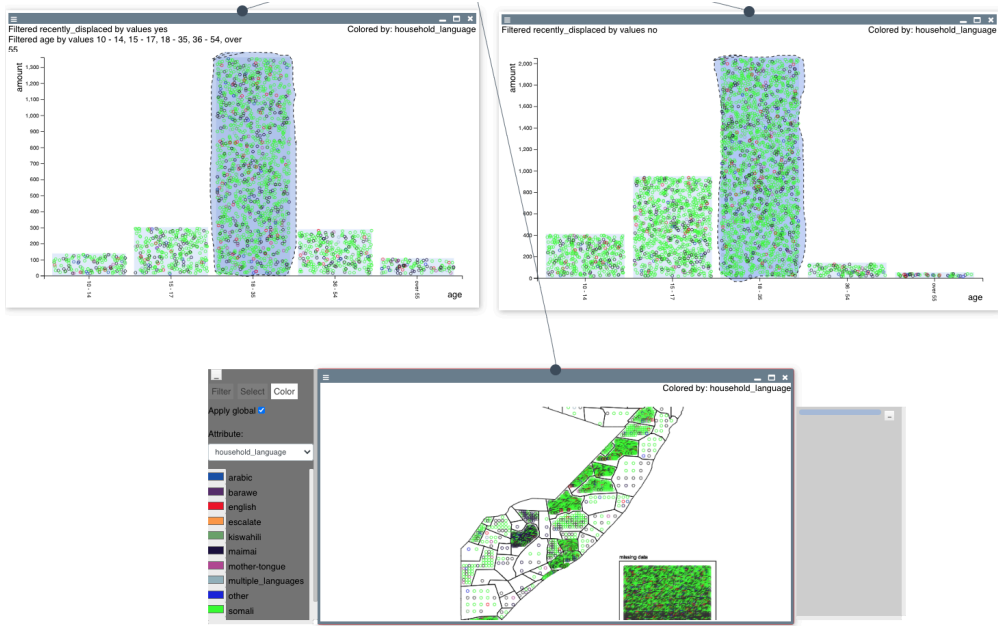
all visualizations, so users see a consistent color scheme. The majority in all groups in the visible diagrams speak Somali. Other than that, no pattern emerges.

To get a sense of the geographic spread of the filtered respondents, we create *Maps* from the selections. Since the coloring applied earlier is also present in the new visualizations, we notice that two districts have another language, “Maimai”, as the dominant household language of respondents. Also, some districts have a much higher density of respondents who are recently displaced than not.

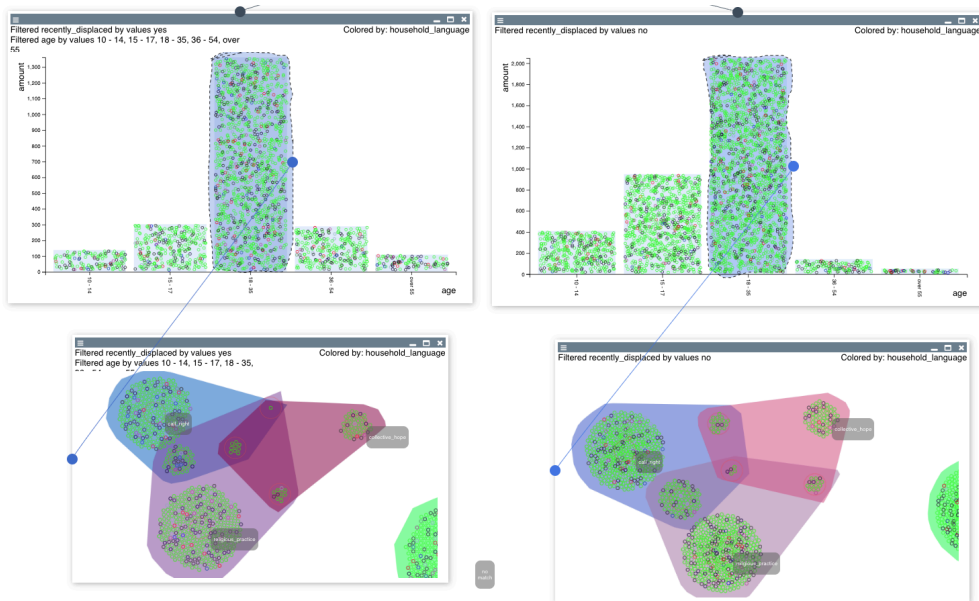
As the household language distinction is quite interesting, we want to see whether this pattern prevails in the complete data set with all respondents. Therefore we create a *Map* from the data source. This *Map* is also already colored according to household language, which saves time. The difference is still very clear, as we can see in Figure 7.9. To avoid that users have to start over building interesting views every time they create new visualizations, new visualizations inherit all actions from the visualization they were created from. Coloring is an exception from that as it is applied globally. It is globally applied because having different meanings for each color across multiple visualizations is a source of mistakes in the exploration process.

To move on to an exploration of the thematic coding, we create *Venn Diagrams* from the previous selections and group by “call for right practice” and “religious practices”. Wondering how the theme “collective hope” would interact with those groups, we add “collective hope” (see Figure 7.10). There is not a big difference when comparing the two visualizations, which is also an interesting insight. In both diagrams, statements of “collective hope” are mentioned more often in combination with “religious practice” than with “call for right practice”.

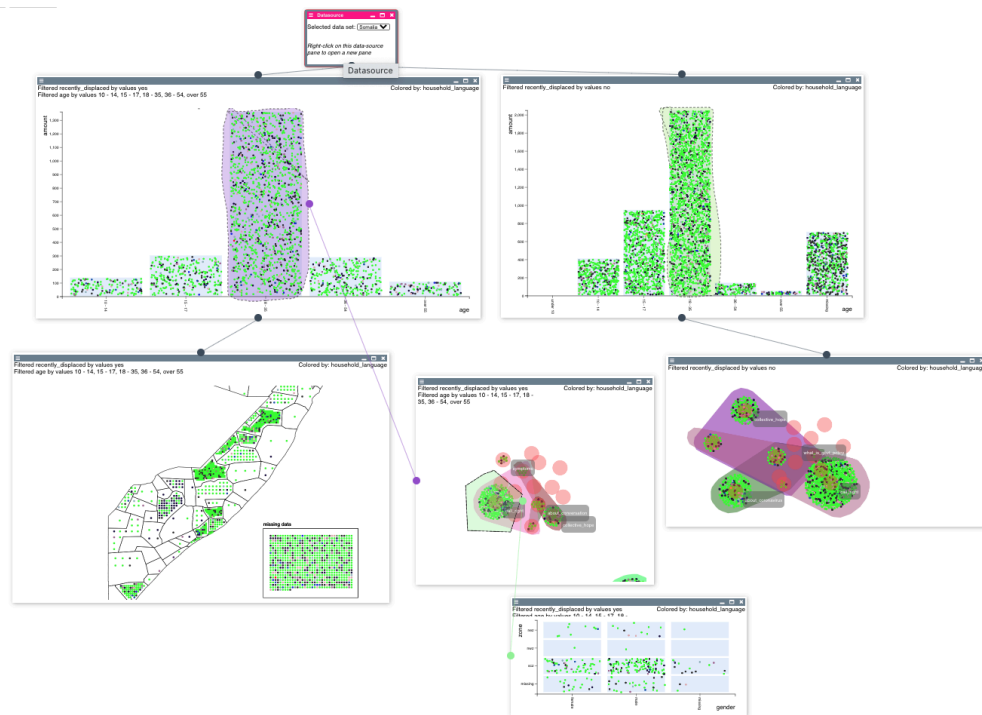




**Figure 7.9:** Age distributions of recently displaced (left) and not recently displaced (right) respondents and map colored by household language (middle), Somali is neon green, Maimai is black.



**Figure 7.10:** Venn Diagrams (bottom two) from selection in age distribution



**Figure 7.11:** Exemplary graph after exploration

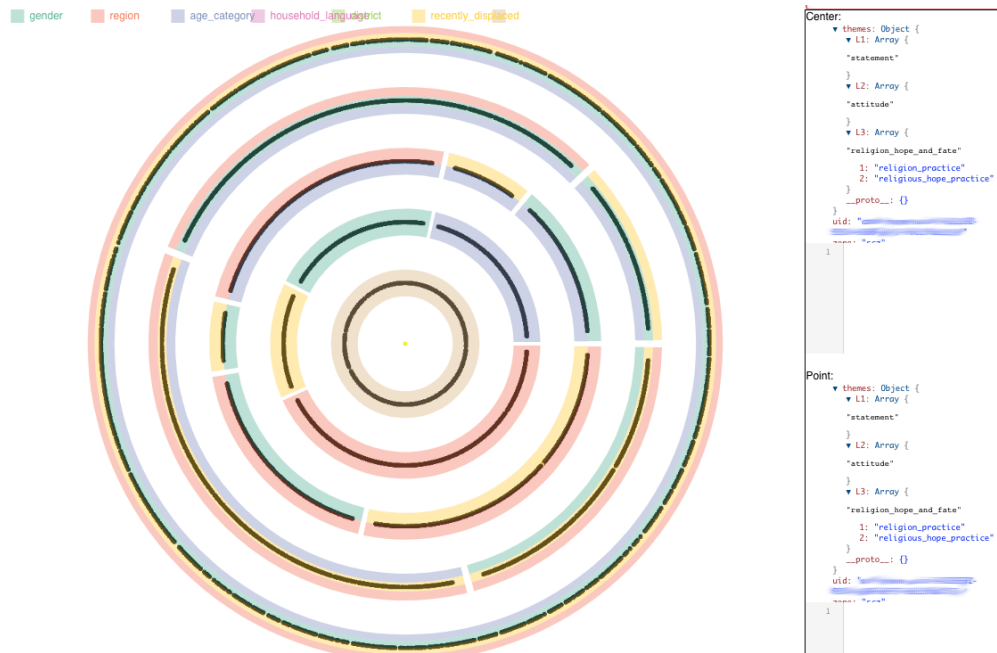
During the exploratory analysis of the data set, users constantly add and create visualizations in the tree-structure. In the end, there are many visualizations in this graph, as shown in Figure 7.11. After and during the analysis, users can always backtrack their exploration steps to either branch at a visualization or use the tree to create a narrative around their exploration. Especially the last part can be helpful if users want to communicate their findings to other people. When looking at the exploration graph, users can also see how their exploration process worked and gain meta-information about how they explored and what attributes, groups, and views they focused on in the exploration.

In this walkthrough, we were able to gain insights about groups of respondents, the different age distributions in demographic-specific groups, in what language they speak, and how this differs from district to district. We made heavy use of displaying multiple visualizations and the lasso-select.

### 7.3.4 Individual Center

This prototype is a bit different from the other two we looked at so far. It is a single visualization instead of a combination of multiple visualizations.

When opening this prototype, users see dots representing all individuals displayed at random positions. By clicking on a dot, users can inspect the corresponding individual and all its demographic and thematic information in a viewer. This interaction is central to this prototype. We click on several individuals and get to know



**Figure 7.12:** Individuals grouped in circles according to their demographic similarity to a chosen individual

them until finding someone especially interesting. We find a middle old man from a city who speaks about “religious practice”. The prototype now allows users to, in a very literal meaning, focus on the selected individual. All other individuals are then arranged with respect to and in circles around the selected individual. Individuals closer to the center are also closer to the individual concerning their attributes.

We first compare by demographic attributes and choose to exclude household language, as this information is missing from the selected individual (see Figure 7.12). We can easily see that a lot of individuals are similar to our selected individual regarding the chosen demographic information. Those individuals are displayed on the innermost ring. Also, we can see what attributes the other individuals differ. From here, we can easily explore what the individuals in the direct “peer group” as well as what people most different from our selected individual, think and what they are saying. By inspecting individuals in the closest ring, we see that while some share his views, a lot of them do not. We find another interesting individual who mentions “collective hope”. We focus on that individual. This time we use thematic codings as a comparison (see Figure 7.13). In this view, users can see who mentions the same things as the selected individual.

The comparison by theme reveals that both groups, the one mentioning “collective hope” and the one not mentioning “collective hope”, differ from the selected individual in a lot of different ways. This difference can be seen by the number of blocks in the circles and the number of colors. The colors indicate the attribute in which the individuals differ from the selected individual. We can, therefore, conclude that those are both quite diverse and heterogeneous groups. From this view, we can



**Figure 7.13:** Individuals grouped in circles according to their thematic similarity to a chosen individual

quickly see how diversely a group voices an opinion. The interesting aspect of this visualization is that it primarily performs a data dimensionality reduction. It reduces the dimensionality to one. The relevant dimension in this visualization is the distance from the center.

In this walkthrough, we were able to get a shallow look at a lot of different individuals and a more in-depth look into fewer individuals. We developed an understanding for the individuals by looking at their demographic and thematic information and getting a general feel for their peer groups. While this prototype guided a qualitative exploration, we could not gain insights as quickly as in the previous walkthroughs.

## 7.4 Value-driven Evaluation

In this section, we provide a value-driven evaluation of our prototypes. In most parts, we concentrate on general aspects shared across all our visualizations. If more in-depth examples of visualizations are necessary, we constrain them to the visualizations in the walkthroughs.

### 7.4.1 Time Needed to Answer a Wide Variety of Questions about Data

Visualizations should allow users to answer different questions about the data by only viewing or also interacting with the visualization. They should be able to

present different forms of data in parallel and thus to compare them [62]. Effective visualizations also allow for answering “low-level” questions like retrieving values, finding extrema, identifying correlations, or enablement of comparisons [62, 1, pp. 26–31, 14].

To evaluate this aspect, we split the discussion into two parts: (1) what questions are possible and (2) how fast they can be answered.

#### 7.4.1.1 Possible Questions

Possible questions that can be asked about the distribution of the data include:

1. **Demographic information–1-D distributions:** Which age/gender/recently displaced/geographic groups are represented in this data set?
2. **Demographic information–correlative:** “higher dimensional distribution”—From the age groups, how is the split according to gender, and also how many are recently displaced within these groupings?
3. **Thematic information–1-D distributions:** How many people answered with one of the themes?
4. **Thematic information–correlative:** How many people answered with a combination of specific themes?
5. **Thematic and demographic information–correlative:** Where are the people from, how old are they, what gender do they have, in what way are they distinctive and answered questions with a specific set of themes?

The last aspect, the possible interplay, and correlations of thematic and demographic information is the most interesting category since insights from there show us what specific groups are saying and might be worried about. These answers can lead to more meaningful changes.

In the following table Table 7.1 can be seen which of the visualizations from the walkthroughs can answer which questions.

**Table 7.1:** Table to test captions and labels

information category	X-Y	Map	Venn	Individual Center
demographic - distributions	X	X		X
demographic - correlative	X			X
thematic - distributions			X	X
thematic - correlative			X	X
thematic and demographic				X

The *Map* visualization can only by itself show distributions constrained to geographic information. While it is technically correct that the *Individual Center* visualization can answer questions from all categories, one always needs to find the right individual or even several individuals.

Interestingly, the three visualizations used most in our prototypes can only answer questions from at most two categories. This limitation can be expanded by using the interactions filtering and coloring.

**Filtering and Coloring** The interactions to filter and color the data allow for correlative questions from another dimension in visualizations. This dimension can be of demographic or thematic information since users can color and filter according to all attribute-value-pairs. When coloring, the points get colored according to the specified attribute, and the users see more of a general pattern than clear statistics. When filtering, the additional dimension is given by “zooming in” on a specific attribute.

**Individual-centric Questions** Our design model with the correspondence of a graphical dot to an individual enables individual-centric questions like: What does *this* individual think? How are the other individuals similar to *this* individual? What do the “peers” of *this* individual think? In an aggregated view, these kinds of questions are not easily answered. Furthermore, this design model also makes references of groups of individuals across views, as implemented in the *Tree View* visualization, very easy and allows for questions like: How does *this* group look like from different perspectives? What other attributes does *this* group share? These are questions generally associated with a qualitative analysis.

**Comparisons** Having more than one window displaying visualizations enables users to ask comparative questions about more specific groups. While it is possible to compare groups in a bar chart according to one attribute and in our *XY Diagram* it is possible to compare groups according to two attributes, by filtering and selecting subgroups and viewing those in separate windows, in theory, it is possible to compare groups along as many attributes as needed. This comparison is also possible without multiple windows, but then the users have to sequentially go through all combinations and retain all the information themselves. This retention of information becomes quickly unfeasible as the number of combinations increases exponentially with each attribute.

**Quantitative Questions** While our approach enables a lot of new questions, it also makes asking some kinds of questions more difficult. Those questions concern either the retrieval of specific values or differences or the comparison of parts of groups of respondents. An example for each of those categories are (1) “How many of the respondents are male, from Mogadishu, and were recently displaced?” and (2) “Which part of the population mentions ‘collective hope’ more often in comparison to all of the respondents?”. This is because all of our visualizations show total amounts opposed to relative amounts. In all views, users can see all individual respondents. Relative views, on the other hand, are always in relation to another scale. When all shown views are normalized and therefore relative to always the same parameter, it is easier to compare them: It is easier to always make comparisons with for example 10% of two groups even when they are a different size, instead of trying to compare

1,000 with 10,000 data points. Of the presented prototypes, only in the *XY Diagram* can users normalize the view when displaying the distribution along one dimension. All other visualizations and views are not able to do this.

#### 7.4.1.2 Time Needed

The time needed to answer a question is an essential factor when facilitating a frictionless exploration workflow. We noticed that every interaction that helps reduce the time needed to answer a new question is helpful. Most of the above questions can be answered by looking at the visualization or inspecting the displayed individuals. Especially the way back to the original messages is very short: in all visualization, it is a click on the individual in question. When comparing this to the connection to the original messages in an aggregated view, this workflow becomes even more impressive. In an aggregated view, users first have to find an unaggregated view where users can see the individuals, find the ones that have the attributes they want, and then again find the individual's original message. Many questions in our visualizations can be answered in the same view by applying filters or colorations.

#### 7.4.2 Spur and Discover Insights and/or Insightful Questions about Data

Visualizations should help users learn and gain insights from a data set, which would be more difficult to gain without the visualizations. The knowledge gained this way is more than just reading data points, but involves looking at structures and different groups of data points and comparing them. In this context, it is useful to think about insight in reference to knowledge-building and model-confirmation [15, 62]. Insight is a by-product of exploration. It cannot be gained with a task-driven analysis [73, 62].

As we have shown in the walkthroughs, our visualizations help users learn insights about the data. For example, they help to see clusters of respondents with certain attributes in a specific group or regarding other attributes, for example, respondents who were recently displaced on a map. They help to easily see the size comparisons of groups of respondents, such as in the *Venn Diagram* in the context of thematic codings. They also help generate questions about data. In the walkthroughs, we saw several examples of this; for example, when noticing anomalies or other patterns, natural questions arise to whether this prevails in another context. In the walkthroughs, we wondered whether the dominant household language in the two data sets would be different for all respondents in these districts. The visualizations help generate questions in the form of "How does this look if we change...". They help due to their relative simplicity and the ease with which to answer those questions. In the walkthroughs, this kind of question emerged and was answered in the *Venn Diagram* when we added "collective hope" as a theme to see which intersecting groups would be bigger. These kinds of questions are important for the exploration process, as they make it easy to think about the next possible steps.

Most of these insights are spurred by "emergent phenomena". That means they are generated by the way low-level entities combine and form trends [9]. In our approach,

the low-level entities are the dots representing individuals. Those phenomena are more easily perceived by humans than they are quantifiable. To illustrate, what emergent phenomena are, consider a heap of wheat. Mentally, start taking grains away. The question now is: When does the collection of grains stop forming a heap? This is hard to specify with numbers, but easy to perceive as a phenomenon.

In an exploration process with our visualizations questions arise organically and insights are discovered almost automatically.

### **7.4.3 Ability to Convey the Overall Essence of Data**

Visualizations should help users gain an overview understanding of a data set. This is a deeper understanding, which is more than just learning about each data point. It provides more of a “big picture” view on the data and helps to gain knowledge about the characteristics and specific patterns of the data set [62].

More than showing the individual respondents' answers, viewing the visualizations gives users a broad perspective of who the respondents are and what they are saying. Most of our visualizations convey a very literal sense of overview. This sense is a side effect of the design model: By showing the individuals one by one and putting them in relation to each other, an overview presents itself. Metaphorically, it provides a bird's-eye view on how individuals would look from above when they arrange themselves according to different attributes. Furthermore, most visualizations start with an overview with the ability to “zoom in” on the displayed data set with various interactions. The emergent phenomena, which are very important for our approach, can only be seen from an overview perspective. They embody characteristics and specific patterns of the data set. With our visualizations, users can also easily gain a sense of what people are saying and how many people think alike. This process embodies an in the literature established visualization “mantra” of “Overview first, zoom and filter, details on demand”[60].

### **7.4.4 Ability to Generate Confidence, Knowledge, and Trust about the Data, Its Domain, and Context**

Visualizations should help understand more than “just” the information in the data set. They should go beyond that and show the importance of the data in its domain context. This supports a sense of confidence and trust in the data and the insights gained from the data. Effective visualizations can also help see where more data is needed or what other data sets should be looked at [62].

Our individual-centric design model supports the creation of trust in the data. At all times, users can select all data points, which means all individuals and inspect them. This possibility improves trust in the data, as users can always see the individual respondents in every view in a transparent way. There is no doubt about whether people say a specific opinion since it can always be checked upon. Our visualizations help to gain knowledge about what people are saying and therefore thinking about specific topics. The visualizations generate confidence about the data and its context. They are able to emphasize that each respondent is not only an



individual but also shows their responses in the visualization. This feature is in contrast to aggregated views, which cannot accurately support the wide spectrum of responses. It is also possible to gain knowledge about the demographic makeup of the respondents. Inspecting individuals and being able to match what they are saying in the visualization to what they are saying in the original data and see possible errors quickly, helps to check the complete system on apparent errors. We used this mechanism several times in our development cycle to debug, for example, an error in our filter tool.

## 7.5 Discussion

In this section, we discuss how we handled the problems described in the foundation section (refer to 7.2.1 and 7.2.2). The discussion focuses on a higher-level evaluation of our approach. The approach we used to create visualizations is more bottom-up than top-down. We created graphical primitives that associate with low-level entities, dots represent individuals, created tools to look at trends and then at individuals (see discussion about emergent phenomena and qualitative analysis 7.4.2), and provided tools to interrogate from either or both directions by integrating the visualizations into environments.

### 7.5.1 Dealing with High-dimensional Data

Our approach of showing all individuals and building all visualizations around these primitives seems to work best with associated low-dimensional visualizations that can be interacted with to add dimension by dimension in a controlled way. This way, users can see directly what influence the additional dimension has in the view, instead of figuring out from the view which dimension caused certain patterns. As shown above in the walkthroughs, gaining insights from the prototypes built around a method of internal data-dimensionality reduction, like the *Individual Center* prototype, is more difficult. While we also had some ideas about high-dimensional visualizations, which try to display almost all of the visualization's dimensions, the tests with our partner revealed they were less useful. The success of low-dimensional visualizations could be due to their relatively simple views, contrasting high-dimensional visualizations, and their direct representation of attributes, contrasting data-dimensionality reduction visualization.

Another data-related issue is the sheer number of respondents. For aggregated views it makes no difference whether the original data set contains 100, 10,000, or 1 million data cases, because they only display statistical summaries. For our approach it makes a big difference, since the individual-centric views display all respondents. While we evaluated our used technologies for more than 100,000 points in chapter 6, due to lack of a big enough data set, we did not evaluate the prototypes in an actual exploration on such a big data set. Since most of the insight generating capabilities are due to emerging patterns, the biggest question might be how to get

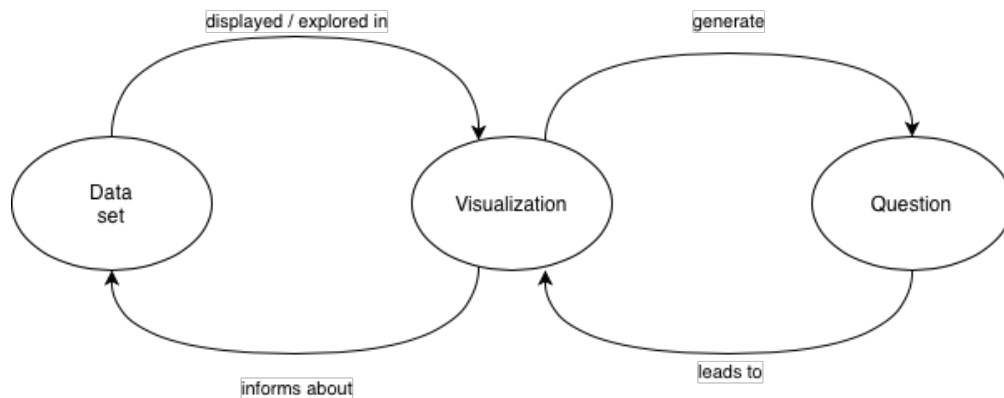


Figure 7.14: Main feedback cycles involving creation of visualizations

a big enough screen to display all points. However, this is an interesting aspect for future investigation.

### 7.5.2 Slow Feedback Cycles Due to Their Current Process

As described in the problem description, slow feedback cycles hamper the exploration process and worsen the analysis's quality. It is, therefore, essential to provide a fast feedback cycle. When looking at the feedback cycles in the analysis process, there are two important ones (see Figure 7.14): (1) from the data to the visualization and (2) from arising questions to the next visualization to answer those questions. Since the latter feedback cycle embodies the exploration, it is more influential on the exploration experience. Some tools necessitate users to go back to the code level, for example, `ggplot2`, (which was introduced in 1.3.2) or other tools require it to go back to the data level, like Excel. We chose to make this feedback cycle as short as possible and have prefigured visualization options in our integrated tools. Users can access those visualizations via interactions and also customize those views with interactions. While this decision dramatically reduces the friction in the exploration process, it limits the possible views for end users to the ones prefigured and able to create by interaction. However, the interactions provided by our visualizations enable users to change many factors in the current view. As shown above, it is also possible to answer a wide variety of questions with a somewhat limited collection of diagrams. We think that the missing flexibility does not hurt the exploration process per se, especially when it is possible to expand the range of usable visualizations programmatically.

### 7.5.3 Generalization of Findings Due to Quantitative Data

With our individual-focused approach, we did not solve the issue of aggregated views suggesting viewers to generalize what they see. Instead, as described several times, we opted to show only non-aggregated views, where all corresponding respondents are visible. By always seeing all corresponding points, users can always

see how big the current sample size is and do not get trapped into thinking it is representative of the whole population. Furthermore, the views our approach generates always show each respondent as a separate entity and therefore emphasize that the sample users look at is unique in the population.

#### 7.5.4 Viewing Single Individuals and Original Opinions

Our design model simplifies the process of viewing single individuals and original opinions a lot. By always displaying the individual respondents, it is easier to view a single one and show their original opinions (subsection 5.5.5). Thus, finding quotes from specific groups also becomes simple; for example, in the *Venn Diagram* finding individuals who expressed a set of themes is as easy as selecting the themes and clicking on an individual. Since we did not have meaningful answers in the provided data sets, displaying the opinions of individuals and displaying all opinions, was less of a focus in our prototypes, but could be an interesting point of further investigation.

#### 7.5.5 Trust in Data

Our approach improves trust in the data, as users are always able to see the individual respondents in every view and make sure they are unique. For a deeper discussion of these points, see the last paragraph and value-driven evaluation. In their data preparation pipeline, AVF goes one step further and logs all modifications in a provenance trace (for a more in-depth explanation see 1.1.2.3). While we had ideas to visualize them (they are categorized under “history” in 2.3.4), we did not pursue those in our prototypes, because our ideas in other directions seemed more promising. However, it might be interesting to see how the trace information from the data preparation could be incorporated into our design model, for example, by showing the trace information for each individual in addition to their answers when clicking on a dot. Furthermore, it could be interesting to add logs from the exploration process to the tracing information. In a sense, those are visible in the tree in our *Tree View* prototype, as seen in the walkthrough in subsection 7.3.3. Explicitly adding them to the tracing information would emphasize that the data is manipulated during the exploration process, creating awareness and trust in the exploration.

#### 7.5.6 Trust in Code

Although creating ideas in the ideation phase for creating trust in the behavior by helping to inspect the associated code, we did not build prototypes in this direction. Instead we depend in this regard on the used development environment. The development environment Lively4 provides capabilities to make it easier to examine code, change it, and see live how the system changes. The usage of web components for more complex visualization systems allows to directly select the component and inspect its code. For a more in-depth discussion about the usage of web components in Lively4 see 4.3.2. This is a way to improve trust in the used

software. However, this approach is limited to users who can understand the source code.

### 7.5.7 Connecting Visualizations

As discussed above, for high-dimensional data, it is useful to connect multiple visualizations. We built two prototypes that integrate visualizations, as described in section 3.5. Because they incorporate the same basic visualizations, comparing them comes down to the integration environment they provide. We noticed that simultaneously displaying multiple visualizations is a great benefit when exploring the data set. Exploring in different directions and comparing across visualizations becomes a lot easier. This behavior is implemented in the *Tree View* prototype. Since all our basic visualizations adhere to the same design model, this comparison across diagrams is also less exhausting as users do not have to switch contexts. Additionally, the interrelation of selected data points across diagrams becomes easier compared to some aggregated diagrams even possible, because all diagrams have as a primitive the dot as an individual.

Combining visualizations also helps in the exploration process if the same interactions and elements are used across all combined visualizations. It lets the specific interaction stay in the background and keep the focus on the data.

### 7.5.8 Missing Interaction

All prototypes we built had some form of interaction:

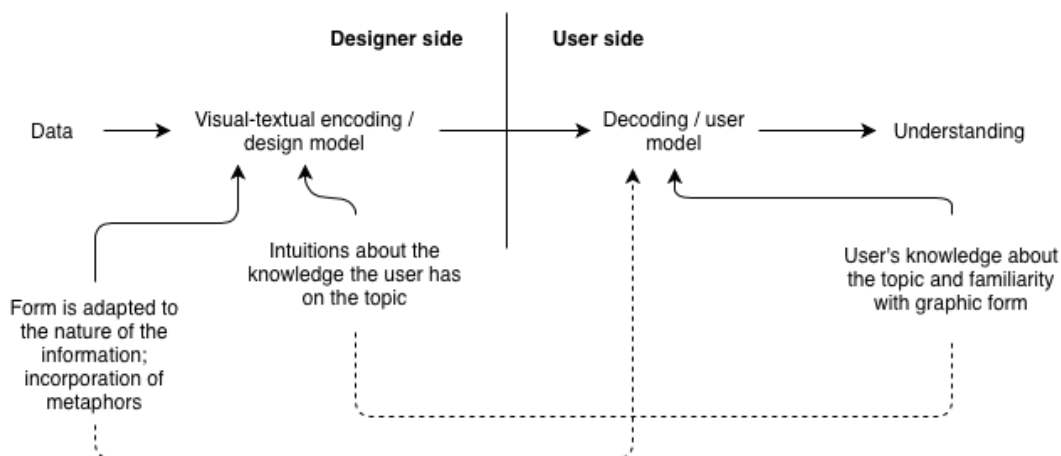
1. direct interaction with the data, for example by clicking on a dot or by selecting a group of individuals with a lasso select,
2. drop-down based interaction, for example for explicitly filtering and coloring, and
3. interactions with sliders, which set parameters for an algorithm or settings inside the visualization.

Those kinds of interactions fall correspondingly into the categories “fully manual”, “mechanized”, and “steerable” from chapter 2 (for an explanation see 2.3.1). The most useful prototypes, the ones presented in the walkthroughs, did not have any kind of interactions from the last category (3). We assume that the reason for this observation is that this interaction hints at an underlying complexity, which makes it difficult to see patterns. The prototypes *Individual Forces* and *Movement* are an example of this difficulty. In contrast to the other two kinds of interactions, the changes made by interaction (3) are not directly represented in the data set. When comparing the remaining two kinds of interactions, there seems to be a trade-off between discoverability on the one hand and more direct interaction with the data on the other hand. The more direct interactions are also directly using the graphical primitives. While a filter menu is discoverable, a lasso-select lets users directly see what data points they are selecting; it is a kind of “what you see is what you get”-interaction. The more direct interaction is faster and allows for an even shorter feedback cycle, but it is not discoverable. Considering that the intended users are

researchers, who can be trained to use the visualizations, we think that in our use case, direct interactions should be used wherever possible. Interestingly, those are the interactions that are enabled especially by our approach.

### 7.5.9 Missing Empathy

Helping to create empathy was an important goal for our visualizations. While it is difficult to determine whether a visualization achieves this goal, we suggest that our individual-centered approach helps to empathize with the people who sent responses. Every visualization has a design model, the model the creators had in mind, and a user model, the model the users have in mind by using the visualization (see Figure 7.15). Our design model, where we display each individual with a dot, makes the individuals explicit as design elements. This explicitness can help align the user model to the design model. This alignment is important, as our design model puts the focus on each individual. Our low-level graphical primitives, the dots, have a representation in the real world: the persons responding. This representation enables a more empathetic view on the data than an aggregated view can. This is because it is easier to identify oneself with a single person than a whole group. Having a concrete example of a person with age, gender, residence, and opinions helps to empathize more than having an abstract description of a group such as “all 28-year-olds”.



**Figure 7.15:** Interplay of design and user model; based on “The Functional Art” [14, 65]

Furthermore, many metaphors for our visualizations are quite vivid and also individual-centric and come to mind easily when viewing them (the metaphors are described in section 3.3 and section 3.3). The insights gained by using a visualization are influenced by the visual metaphor presented in a visualization. This result suggests that those metaphors can further create empathy in users [74]. As described in chapter 2 (see 2.4.2 for reference), we also had the idea to create empathy by

adding a sense of “liveliness” to the individuals, for example, by adding movement. The prototypes built around these ideas are interesting to look at but are often too variable to gain useful information.

#### **7.5.10 Issues with the Mindset**

The effect our visualizations have on the underlying mindset of the users is even more difficult to measure than other parts of this evaluation. We think that our visualizations can be a step in the right direction by providing new visualizations which put the focus on the individual. Some of the presented prototypes are similar to familiar, aggregated visualizations, which can help adoption.

## **7.6 Conclusion and Outlook**

Our goal was to evaluate our approach and prototypes with respect to the problems Africa's Voices is facing when visualizing their data.

We presented walkthroughs for three prototypes. In them, we showed how to use our systems to explore a data set and generate insights as well as questions. We performed a value-driven evaluation and showed that it is possible to answer a wide variety of questions in a short period of time, spur insights and questions about the data, convey the essence of data, and generate confidence and trust in the data and domain. We conclude that answering quantitative questions with our approach is possible. At the same time, in most of our visualizations, answering quantitative questions is more difficult than it is in aggregated views. Instead, our visualizations generate insights through emergent phenomena by showing trends in the data. We also discussed how we handled the main problems Africa's Voices faces when using visualizations. Our approach of visualizing every individual as a dot can help to create empathy. We evaluated that a combination of multiple visualizations and direct interaction with the data allows for a faster feedback cycle, which is essential for exploration. Furthermore, our visualizations help to create trust by allowing users to easily see the makeup of a visualization, look at the provenance information, and inspect every individual respondent closely. In this way, our visualizations combine a qualitative and quantitative analysis approach.

In this chapter, we showed that our work provides value for interactive and empathetic exploration. Therefore, our project can serve as a starting point for further research in this area.

#### **Outlook - Work at Africa's Voices**

Africa's Voices goal of helping with accountability to affected populations in the aid sector is not achieved by our project and requires ongoing commitment. The clients of the aid world are the people they are trying to help. Listening to what they say often gets forgotten. Our project tries to make their voices and the individuals behind those voices visible. If someone says something interesting, a key challenge is to keep paying attention to that individual over the exploration process. A necessary

prerequisite for that is that interactive visualization makes this possible from day one. Luke Church, technical lead at Africa's Voices, states about our work: "It is an essential requirement that all projects will start with a visualization system like that." Africa's Voices has three to four prototype visualization systems at the moment. Integrating them and thinking about how to disrupt expectations of what is possible are critical next steps. Since building the visualizations is often easier than the design of the visualizations, especially when trying to visualize tens of thousands of people, our project will guide their future work. Our design will become the basis of their design process in the future and build the core for the next generation of visualizations. To quote Luke Church: "This work has set the future for how it is that we are going to visualize citizens' conversations."





## 8 Conclusion

We set out to develop new interactive and explorable visualizations of personal opinions and demographic data with a focus on the individuals they are representing. Therefore, they need to create empathy and trust and ideally disrupt the mindset for handling visualizations that standard charts have created. With these visualizations, we wanted to solve domain-specific problems, combine quantitative and qualitative analysis, and enable the tracing of a data point's provenance.

In this report, we described our ideation process and its results for the discovered problems. We found prevalent concepts, that help with exploration. These concepts include displaying individuals as points, showing different perspectives of the same data, and being able to manipulate the view, code, and data to change visualizations. Also, we discovered that the design space has a lot of potential for future work. Based on our ideas we implemented several visualization tools and extracted interaction patterns that work well with the concept of individuals as points. We showed that the design concept of individuals as points is feasible and discussed considerations for implementing explorable visualization tools and environments. These considerations can provide guidance for further developers and designers of explorable individual-centered visualizations. Discussing used core concepts of the development environment Lively4, we explained how the platform helped us with documenting, prototyping, collaborating, and testing ideas for our project. For us, quick feedback cycles, the direct inspection of code, and the live environment proved valuable for our work. We evaluated three strategies to achieve bidirectional mapping and showed how we implemented the most promising ones into our prototypes. In our project, the bidirectional mapping of data and UI enabled us to develop interactive visualizations, that allow direct interactions with the underlying data. We considered different technologies to display visualizations with up to 100,000 points in the browser. We performed benchmarks both in Lively4 and on a local server to assess their performance. We found that, for our use case, using HTML-Canvases provides enough performance and a high enough abstraction level to enable the implementation of bidirectional mapping strategies. We showed how to generate insights with our visualizations by presenting three walkthroughs. We explored, how our approach of visualizing every individual as a dot can help create empathy. We evaluated that a combination of multiple visualizations and direct interaction with the data is essential for exploration. Our visualizations help to create trust by allowing users to easily see the makeup of a visualization, look at the provenance information, and inspect every individual respondent closely. In our work, the provenance information only traces back to the original data. However, including all information from a preprocessing pipeline in a visualization is worth further investigation.

## 8 Conclusion

Our work demonstrates that a design strategy built around primitive entities works in visualizations. With this strategy, insights are generated through emergent phenomena by showing trends in the data. We showed that it is feasible both technically and analytically to show every person as a separate entity in visualizations for around 10,000 persons. Since not any one view is sufficient to visualize opinions as well as demographic information, the visualization environment matters and needs to support an exploration workflow. We showed possible ways to integrate visualizations in the browser. Most importantly, we showed that the design space for explorative visualizations, which enable the creation of empathy and trust in the data, is quite rich and that there is a lot of potential for future visualizations.

# Appendices



# A Appendix Chapter 2

## A.1 Idea Collection

### A.1.1 Idea 1

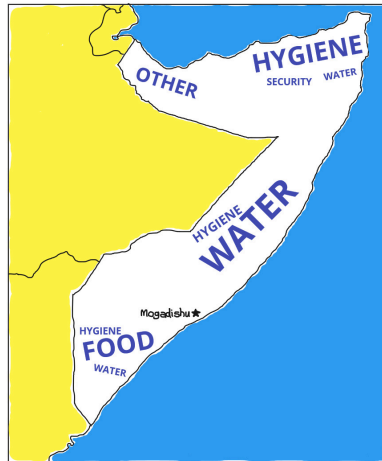


Figure A.1: Word Cloud with themes. Font size encodes the number of times messages coded with this theme were sent

### A.1.2 Idea 2

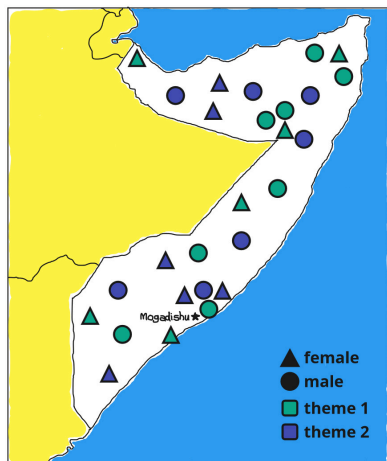


Figure A.2: Data points represented by shape. Color encodes theme, and shapes encode other attributes

### A.1.3 Idea 3

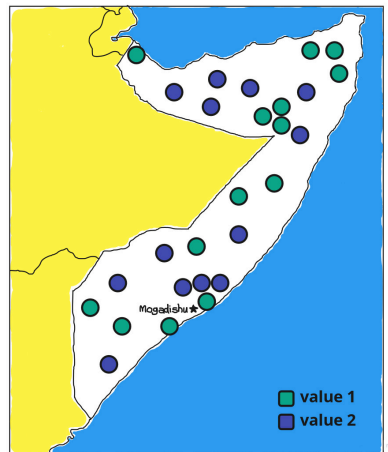


Figure A.3: Individuals as points on a map colored by the value of a selected attribute

### A.1.4 Idea 4

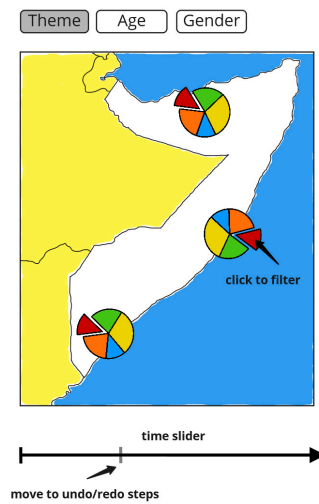
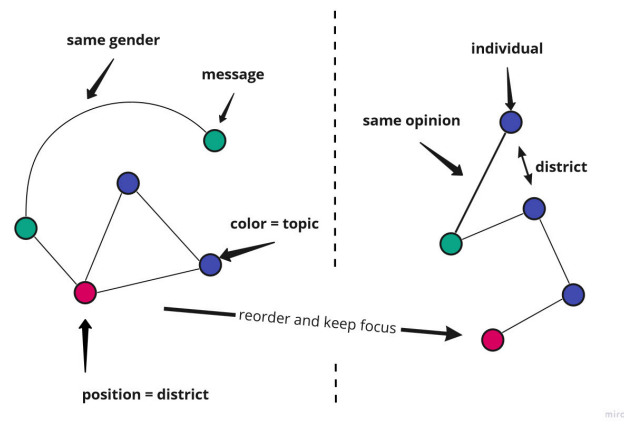


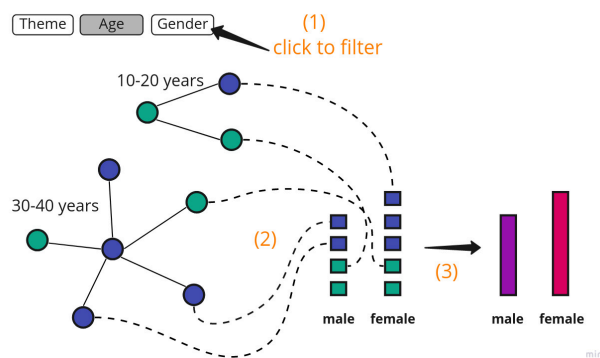
Figure A.4: Pie charts for each district displaying the values of a selected attribute for the corresponding individuals. Clicking on pie slices filters values. A time slider allows undoing or redoing filter or selection steps.

### A.1.5 Idea 5



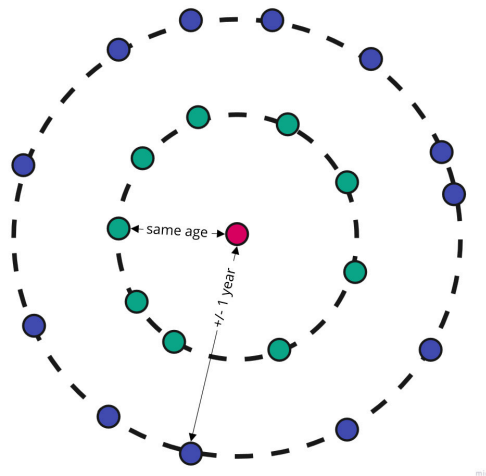
**Figure A.5:** Nodes represent individuals or messages. Each dimension encodes different information. The graph can be reordered to let nodes represent individuals or messages.

### A.1.6 Idea 6



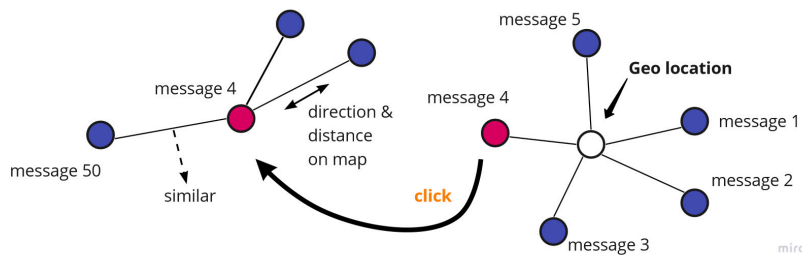
**Figure A.6:** Individuals are morphed into different representation forms, depending on selected filter argument (1). Chaining of filters by using other dimensions of data visualizations to show groupings in groups. For example, the filter argument “age” displays individuals in a graph. Clicking on “gender”, individuals are morphed into a bar chart (2),(3).

### A.1.7 Idea 7



**Figure A.7:** Individual as point as a reference point in the center. All individuals are arranged around the referenced individual according to their “distance” regarding one characteristic. Clicking on another individual makes this individual the new reference point.

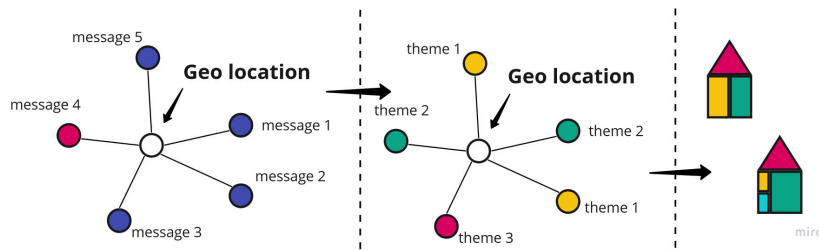
### A.1.8 Idea 8



**Figure A.8:** Message as a reference point in the center. All messages are arranged around the referenced message in a graph according to their distance between the districts of the corresponding individuals. Edges represent similarity of messages. Clicking on another message makes this message the new reference point.

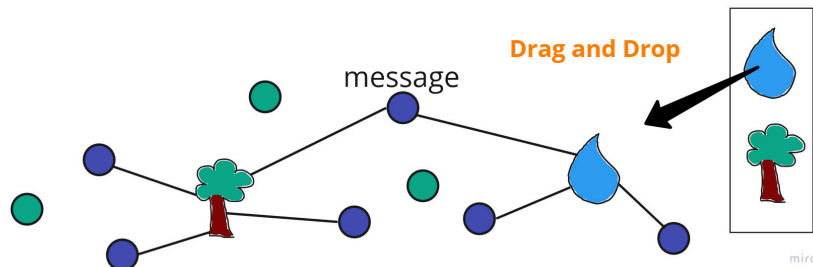


A.1.9 Idea 9



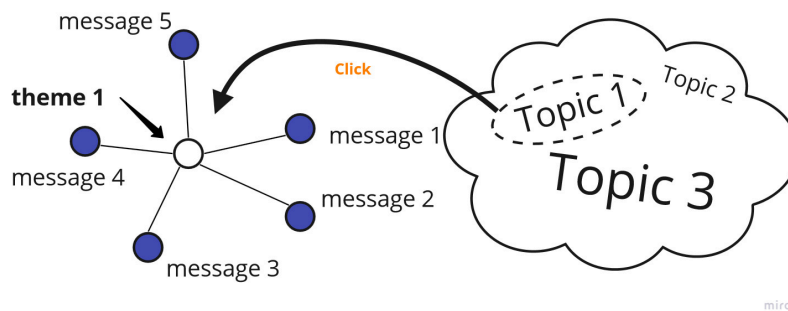
**Figure A.9:** A graph connecting messages to their corresponding district. Zooming out, messages are abstracted by their theme. Zooming out further, districts are abstracted by the combination of themes, each pixel representing one theme of a message.

A.1.10 Idea 10



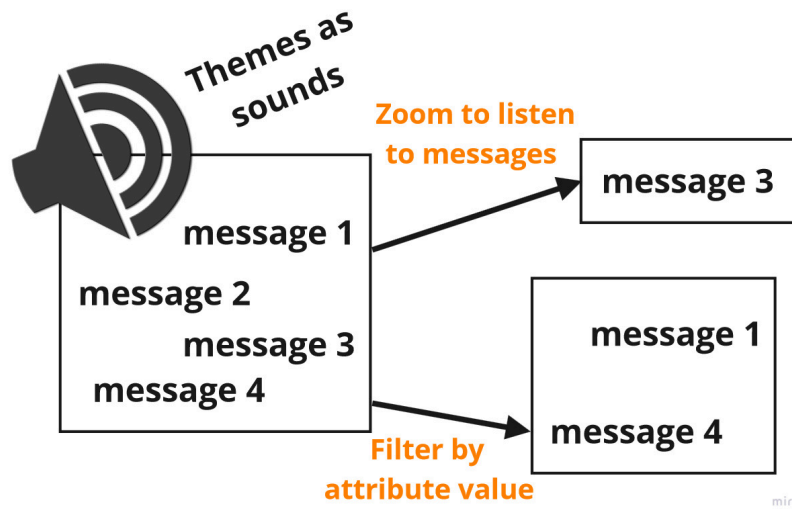
**Figure A.10:** Themes represented by icons. Dragging and dropping the icon into the canvas with messages, then messages with these themes are connected to the icon.

A.1.11 Idea 11



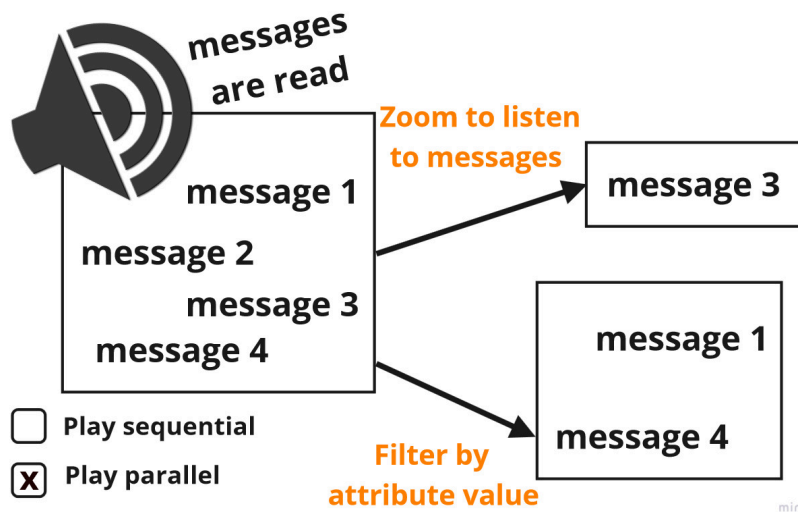
**Figure A.11:** Word cloud with themes. Font size encodes the number of times messages coded with this theme were sent. Clicking on a theme reveals all messages with the selected theme.

A.1.12 Idea 12



**Figure A.12:** Themes represented as sound. Messages are displayed in the world. The sound volume of a theme encodes the number of individuals who sent messages coded with that theme. Zooming in on a message, the message gets read. Filtering takes only the corresponding messages into the calculation of the sound volume of the theme.

A.1.13 Idea 13



**Figure A.13:** Messages are read out loud. Buttons to toggle sequential mode and parallel mode. In sequential mode, messages that are read are highlighted. Filtering and selection of messages is possible.

A.1.14 Idea 14

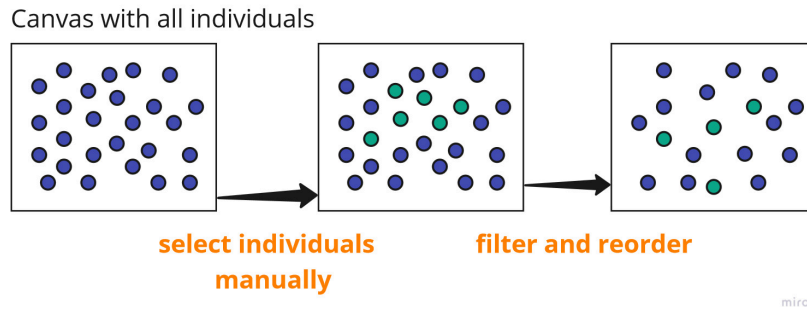


Figure A.14: Individuals as points on a canvas. Select individuals to highlight them. Filter and reorder by attributes.

A.1.15 Idea 15

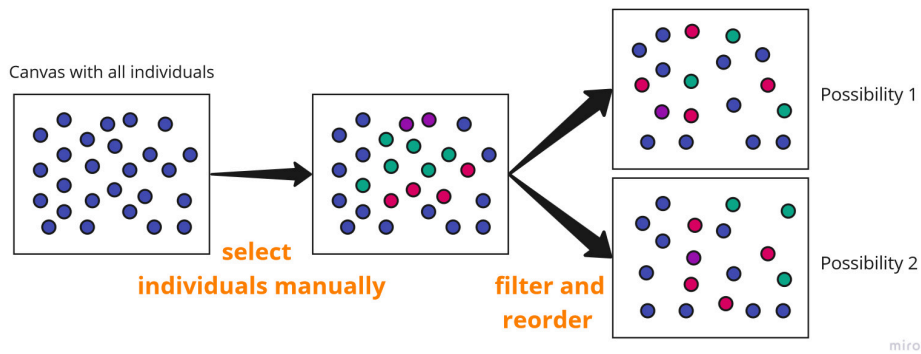
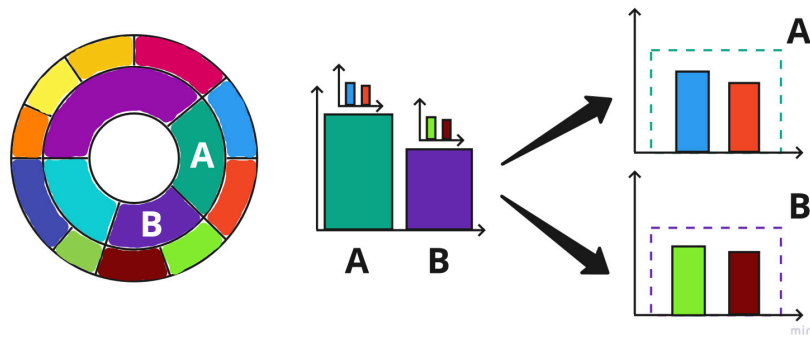


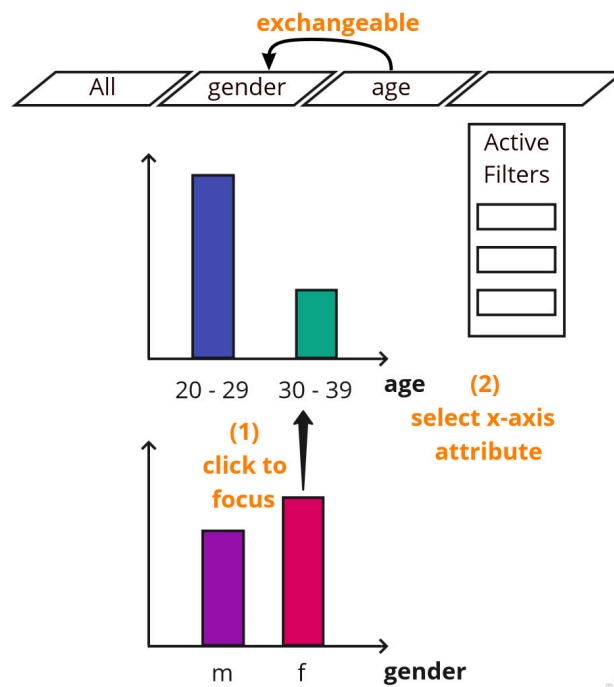
Figure A.15: Individuals as points on a canvas. Select individuals to highlight them. Filter and reorder by attributes. Display different possibilities of the reordering to ensure that no information is derived from a random position.

A.1.16 Idea 16



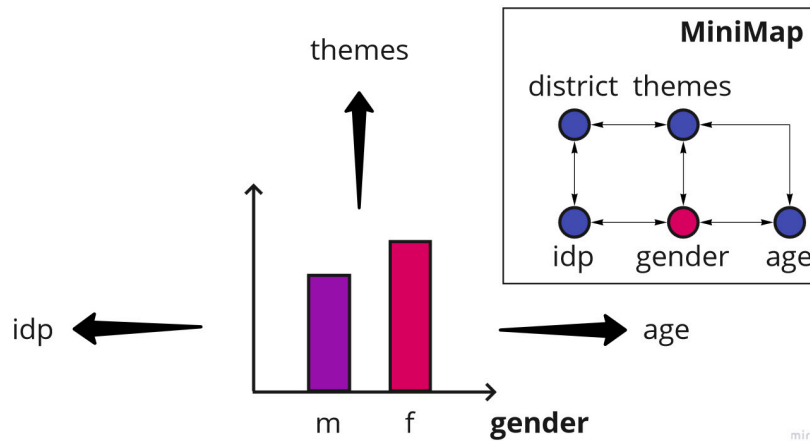
**Figure A.16:** Sunburst as bar chart without the 100% of participants a sunburst chart conveys. Show a smaller diagram above the primary point of view. Show a smaller diagram inside of the bars of the primary one.

A.1.17 Idea 17



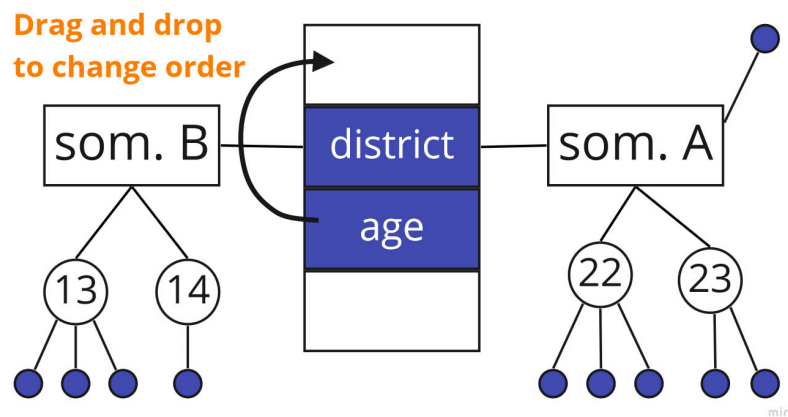
**Figure A.17:** Bar chart that allows filtering. Different filtering options are presented. Clicking on bar focuses on data (1). Selection of another parameter for comparison (2). Active filters are shown. Filter/selection history in which steps are exchangeable.

A.1.18 Idea 18



**Figure A.18:** Guided Exploration. Minimap shows which “paths” are possible. Navigation through layers by sorting or filtering the parameters. Arrows show what point of view can be shown next.

A.1.19 Idea 19



**Figure A.19:** Attributes are arranged in the center by drag and drop. Messages are structured as a mind map. Individuals who don't have an age but a district are directly connected to the corresponding district node.

A.1.20 Idea 20

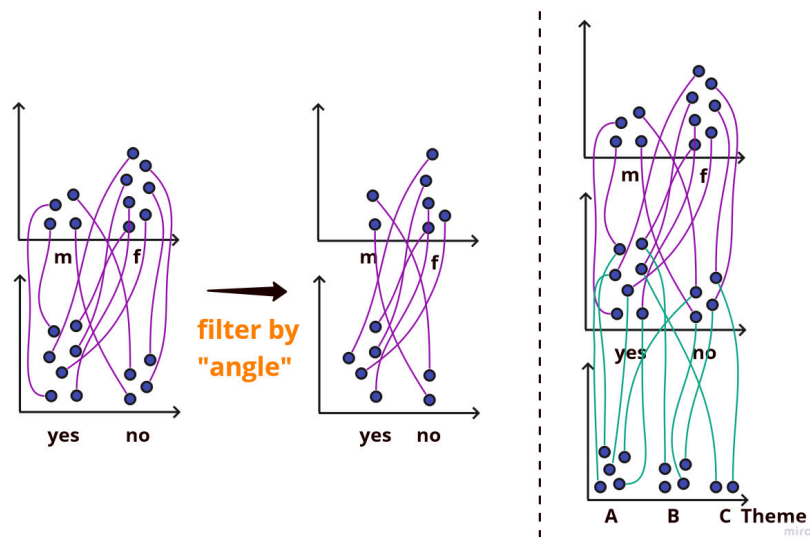


Figure A.20: Individuals as points in different diagrams regarding different attributes. Points representing one individual are connected. Possible with two or more diagrams.

A.1.21 Idea 21

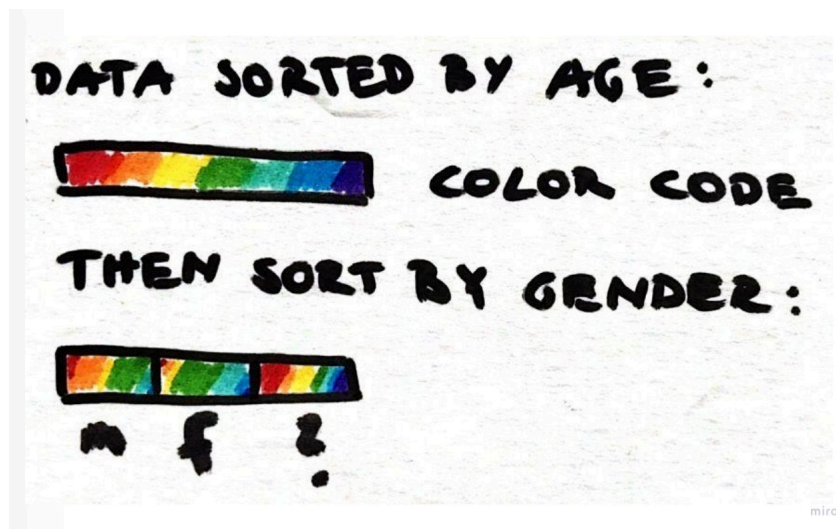


Figure A.21: Color individuals represented by pixels according to attribute value. Sort or filter them to see how the pattern changes.

A.1.22 Idea 22

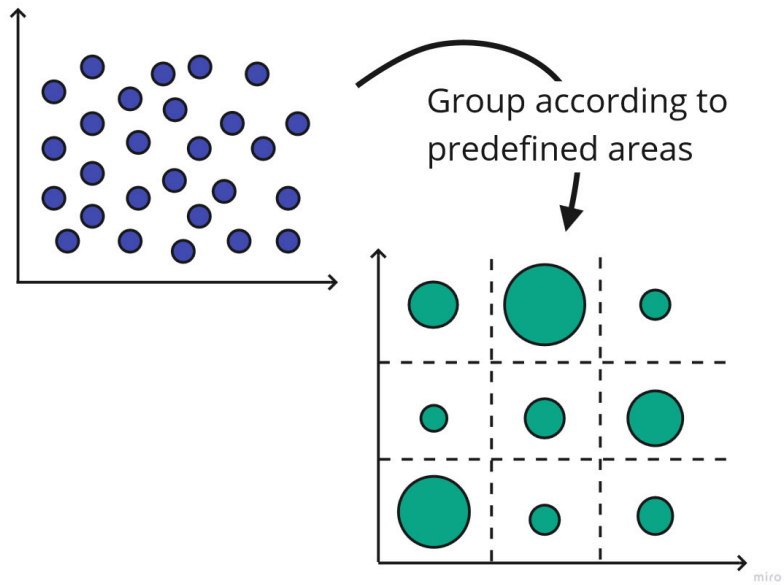


Figure A.22: Grouping individuals as points on a coordinate system in predefined areas

A.1.23 Idea 23

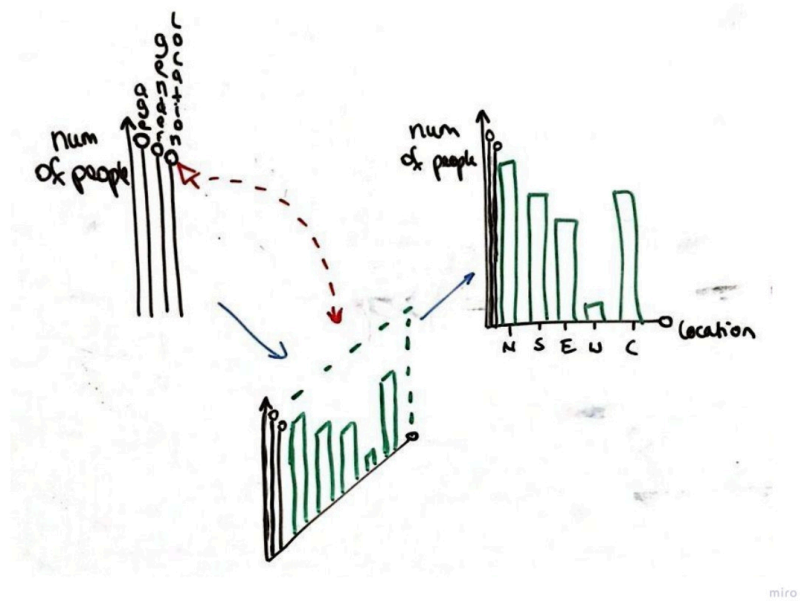


Figure A.23: Choosing x-axis per drag and drop

### A.1.24 Idea 24

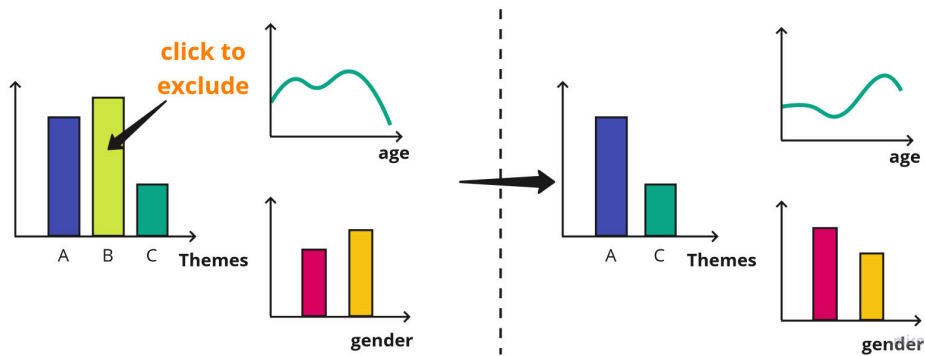


Figure A.24: Selecting a bar from a bar chart to exclude it in the other displayed diagrams

### A.1.25 Idea 25

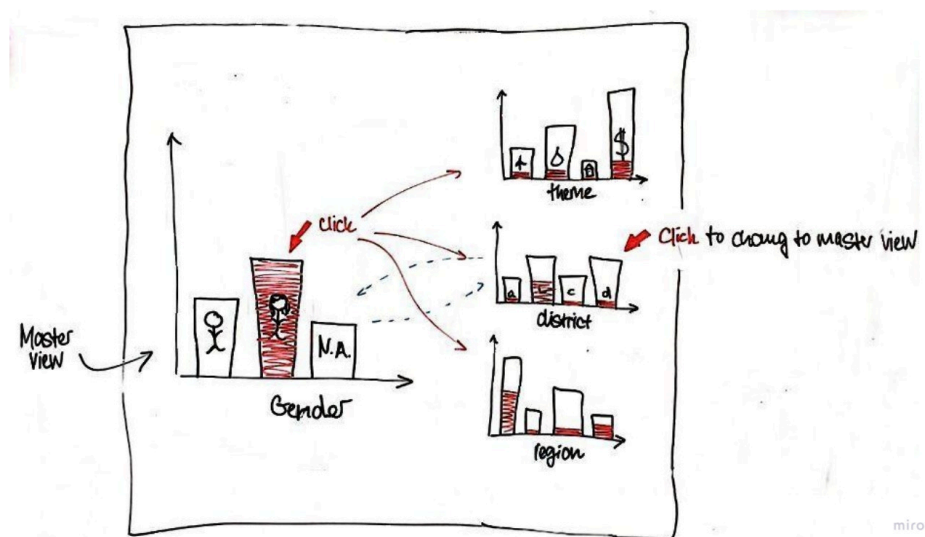


Figure A.25: One centered chart that can be interacted with. Selecting a bar to show the ratios in other diagrams, creating stacked bar charts. Selecting a different diagram centers it.



A.1.26 Idea 26

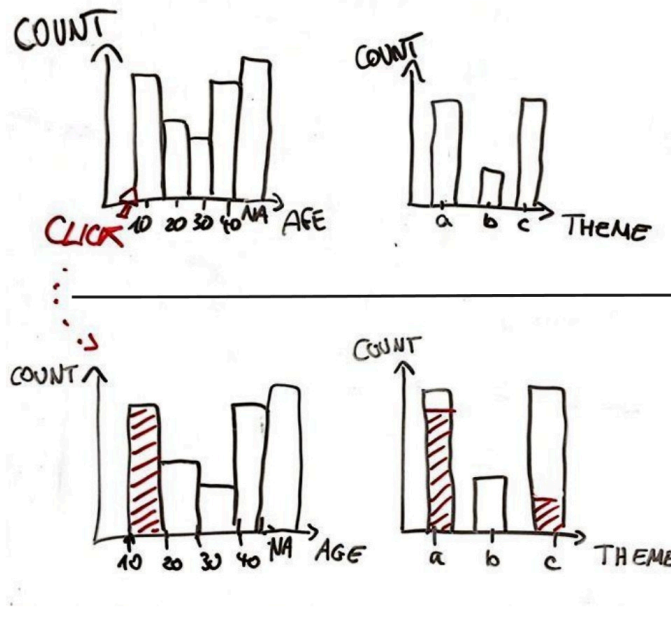


Figure A.26: Selecting a bar highlights the corresponding parts in the other charts, creating stacked bar charts

A.1.27 Idea 27

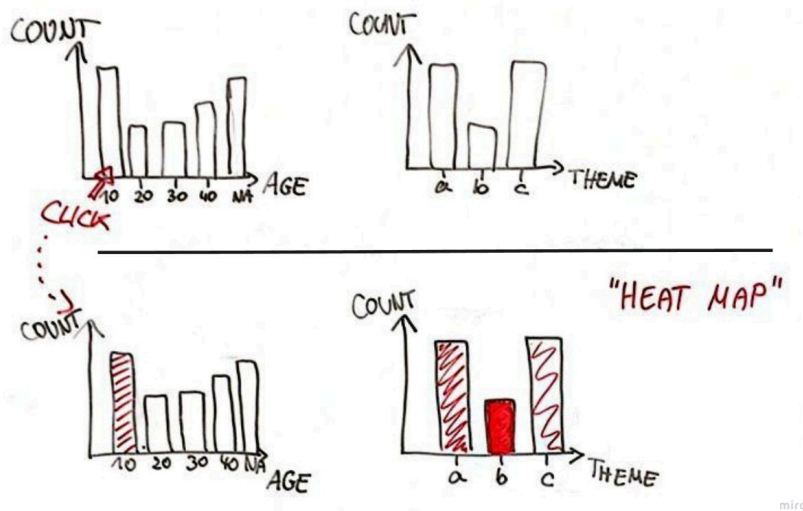


Figure A.27: Selecting a bar highlights the corresponding parts in the other charts in form of a heat map, depending on how many individuals from the selected group are in this category.

A.1.28 Idea 28

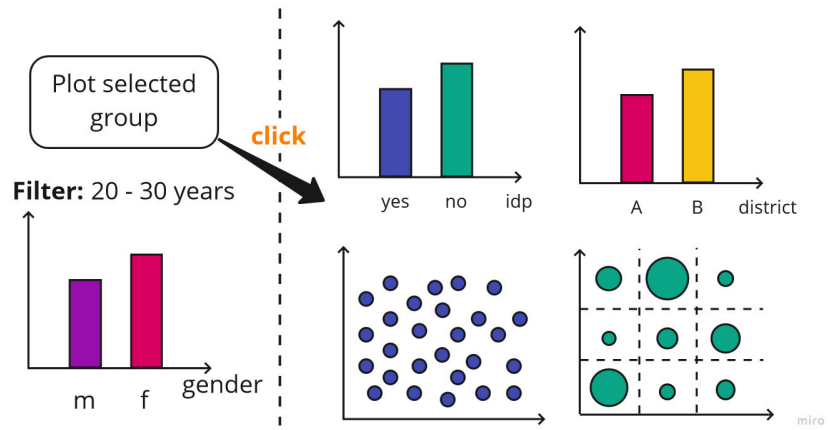


Figure A.28: Plot different diagrams for a selected group

A.1.29 Idea 29

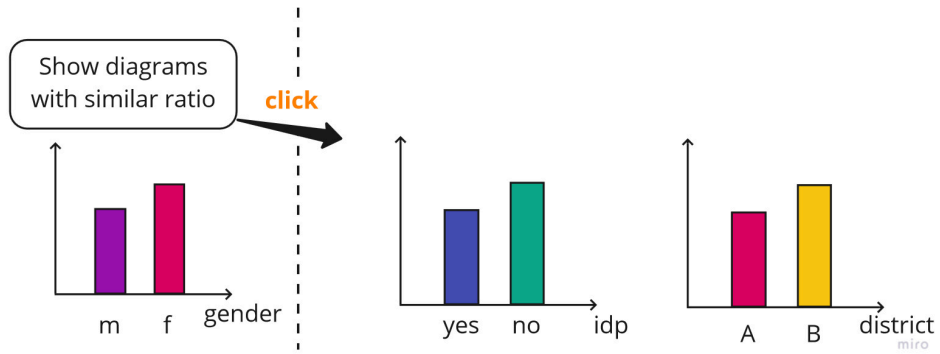


Figure A.29: Show charts with similar ratios

A.1.30 Idea 30

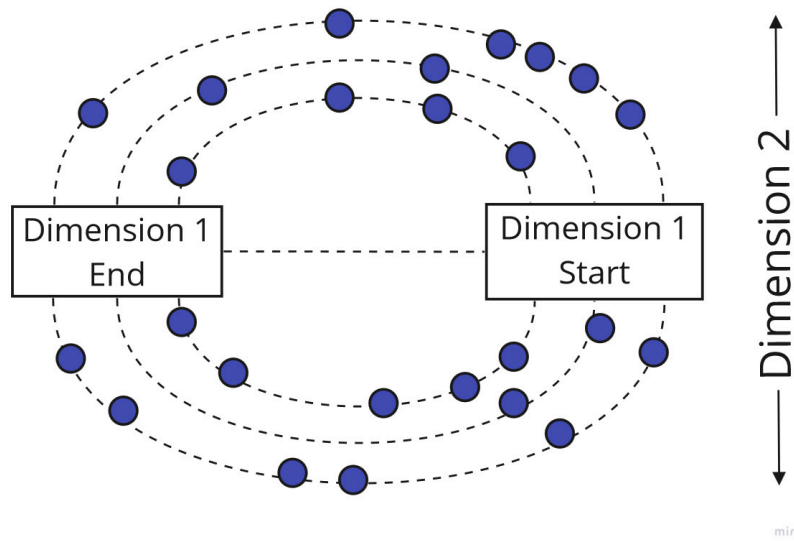


Figure A.30: Individuals as points positioned in regards to one or two dimensions as "gravity fields"

A.1.31 Idea 31

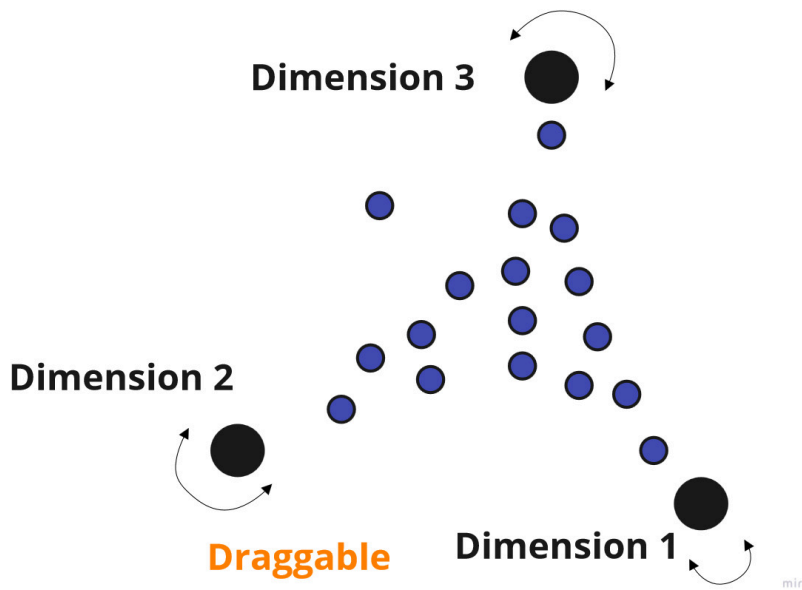


Figure A.31: Draggable three dimensions that represent attribute values. Individuals as points are placed according to their values

A.1.32 Idea 32

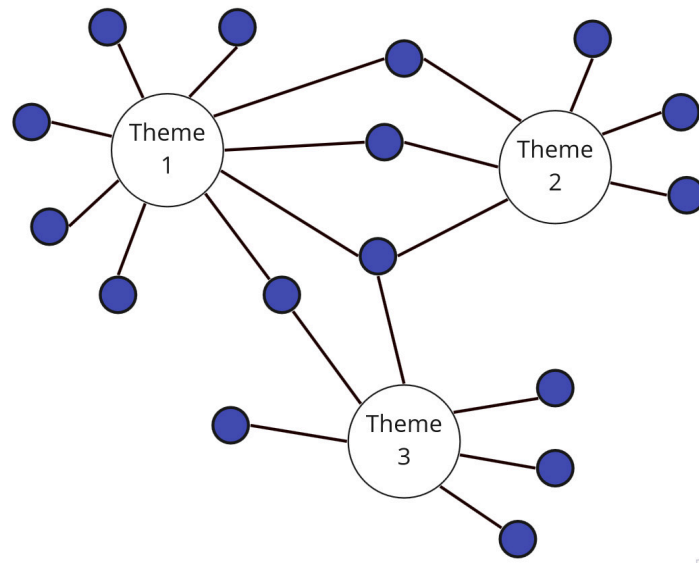


Figure A.32: Individuals as points connected to their themes

A.1.33 Idea 33

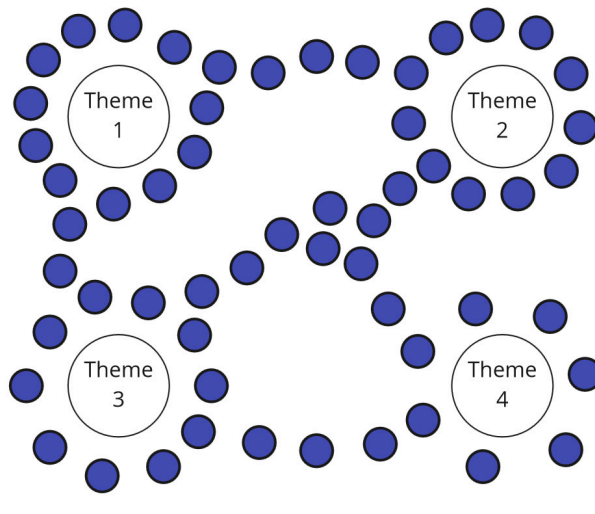


Figure A.33: Individuals as points positioned according to their themes. Variation: moving them between themes

A.1.34 Idea 34

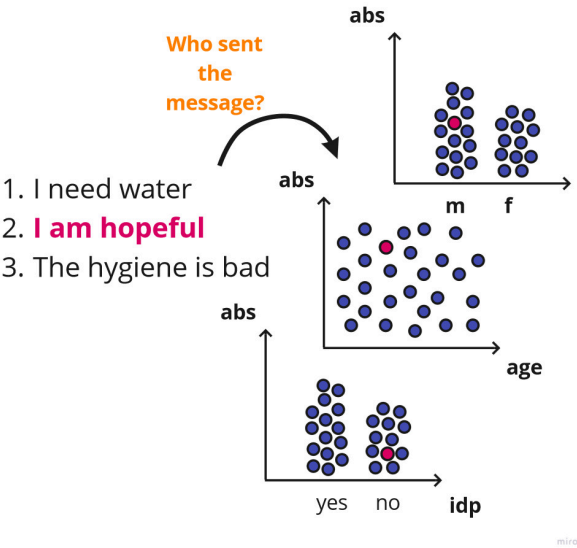


Figure A.34: List of different messages. Different diagrams are displayed for a selected message. Corresponding data points are highlighted.

A.1.35 Idea 35

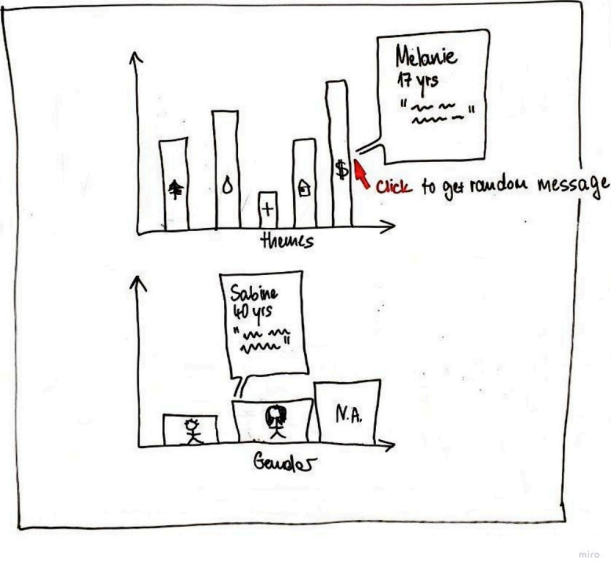


Figure A.35: Selecting a bar displays a random message

A.1.36 Idea 36

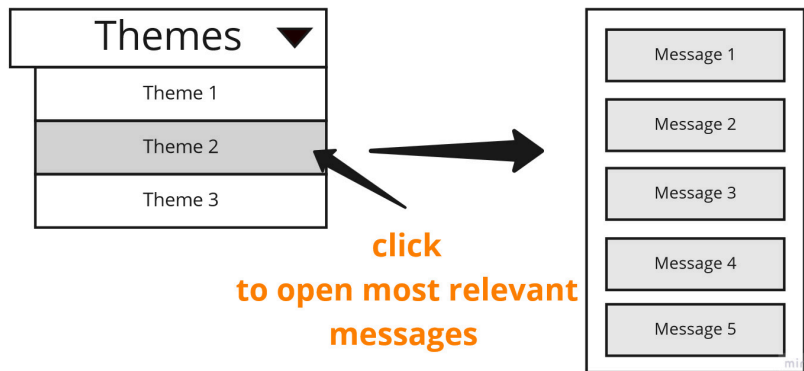


Figure A.36: Displaying the most relevant messages for a selected theme

A.1.37 Idea 37

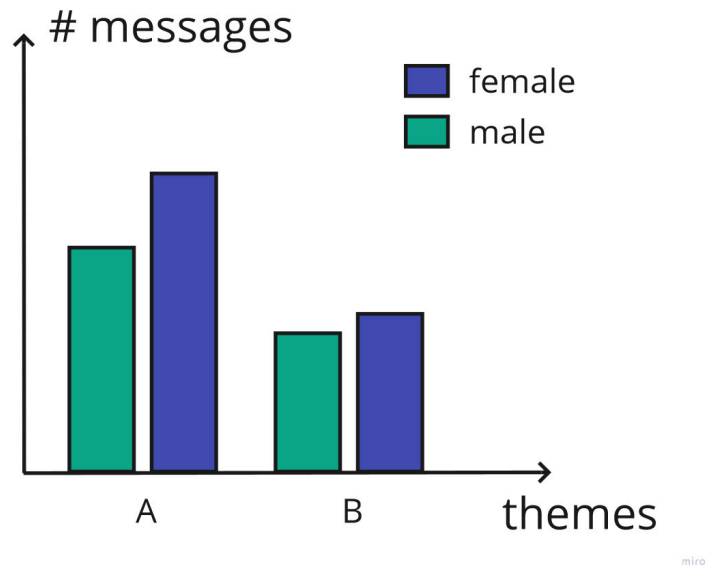
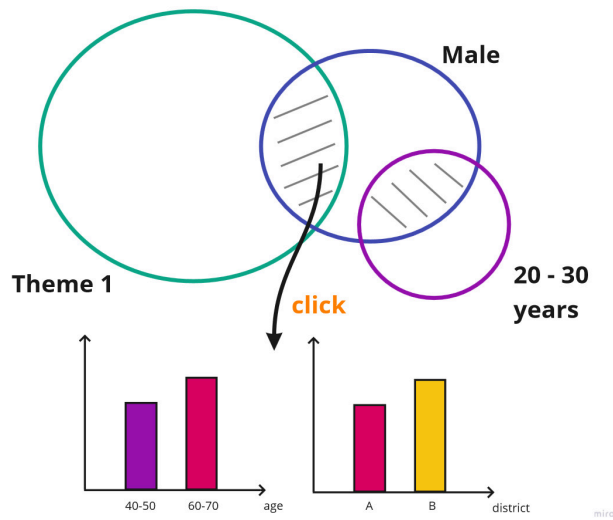


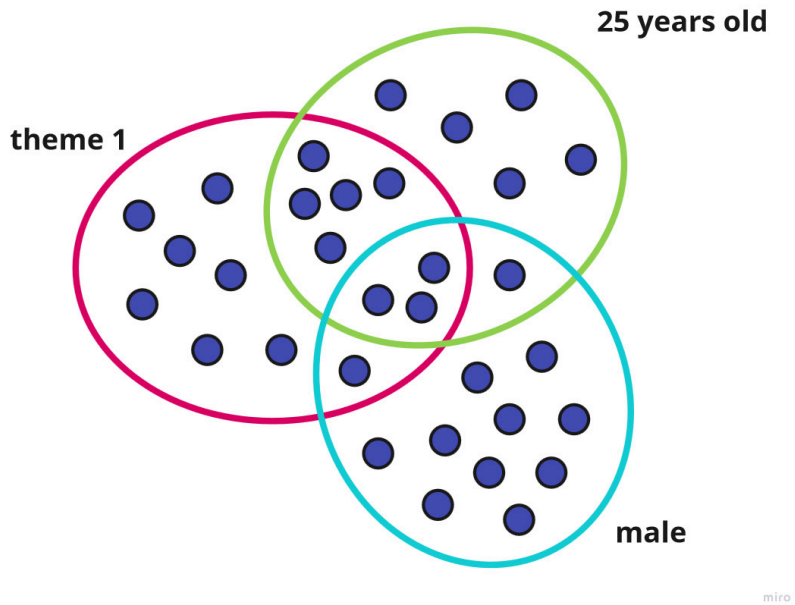
Figure A.37: Bar chart with y-axis representing number of messages

A.1.38 Idea 38



**Figure A.38:** Bubbles represent groups of individuals. Show overlaps of different attributes. Clicking on overlaps displays diagrams regarding both bubbles.

A.1.39 Idea 39



**Figure A.39:** Determine attribute values to group by. Individuals as points are displayed inside their corresponding areas.

A.1.40 Idea 40

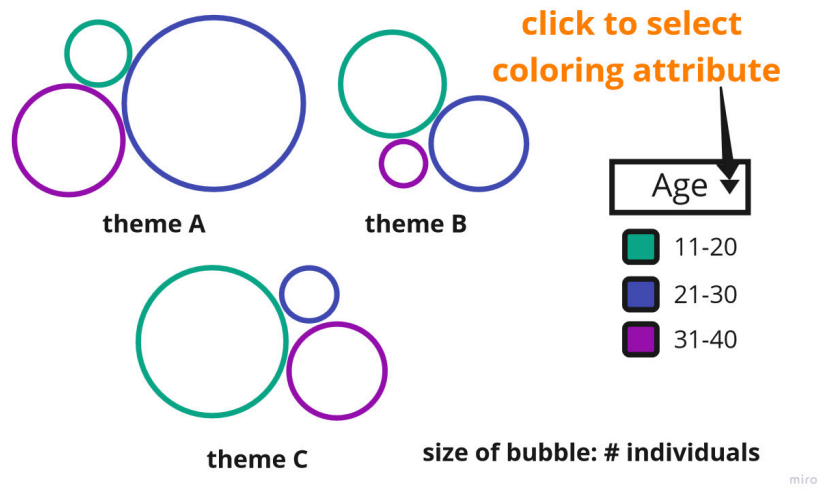


Figure A.40: Bubbles represent groups of individuals. Color encodes attribute value. Bubbles clustered by theme.

A.1.41 Idea 41

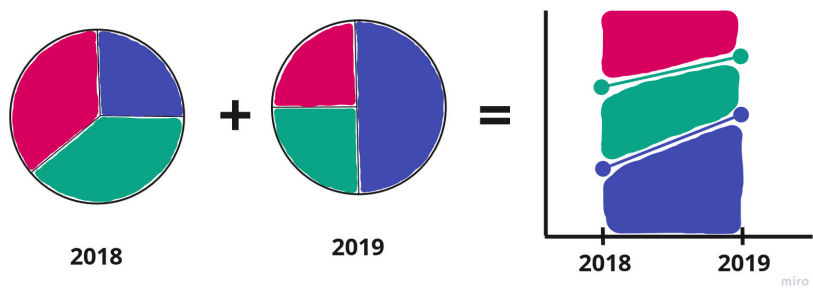


Figure A.41: Fusing two pie charts into one stacked line chart



A.1.42 Idea 42

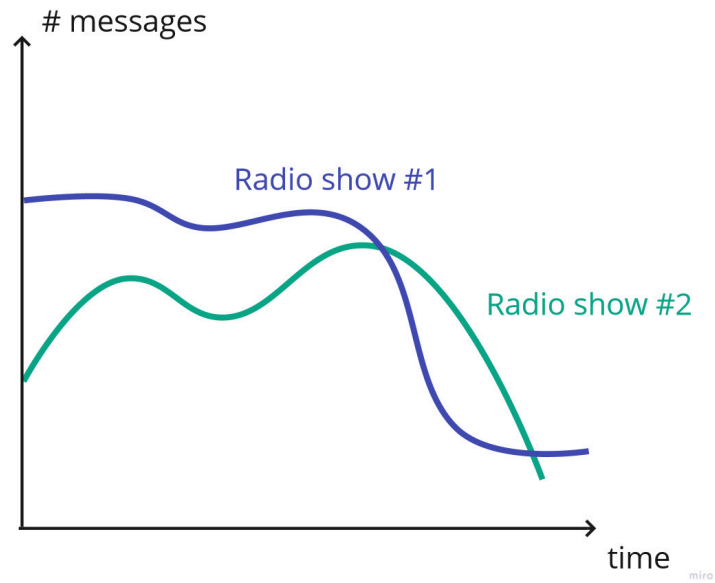


Figure A.42: Comparing different radio shows by number of messages using line charts

A.1.43 Idea 43

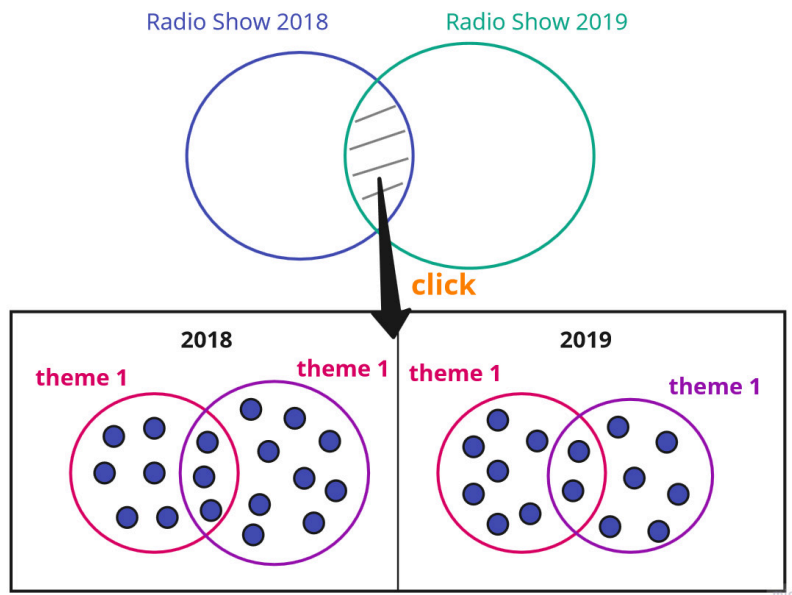
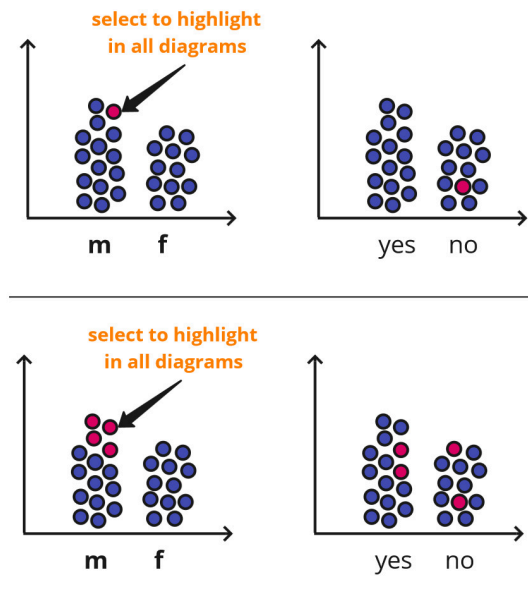


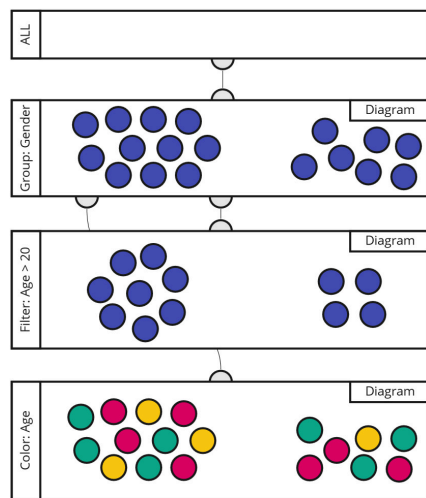
Figure A.43: Overlaps display individuals who answered in multiple radio shows. Show comparison of similar themes of messages sent by these individuals.

A.1.44 Idea 44



**Figure A.44:** Multiple scatterplots with different attributes. Selecting one or multiple individuals as points in one diagram selects the same individuals in all other diagrams.

A.1.45 Idea 45



**Figure A.45:** Individuals as points in windows connected with each other. Each window represents one state. Individuals can be grouped, filtered, and colored. Data can be displayed in a diagram as well. Individuals can be selected per drag and drop to create new windows.

A.1.46 Idea 46

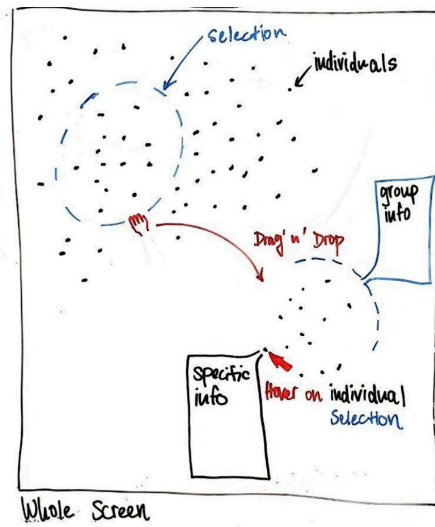


Figure A.46: Individuals as points on a canvas. Individuals can be grouped, selected, and filtered. Hovering displays information regarding the individual or a group of individuals.

A.1.47 Idea 47

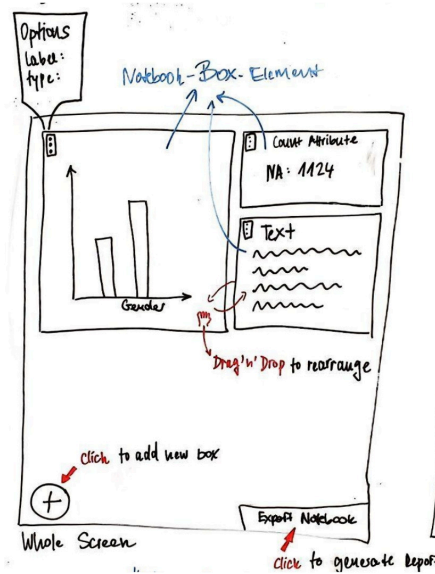


Figure A.47: Notebook with different widgets that can be arranged according to one's wishes. Different diagram types can be included. Notebooks can be exported to create a report.

A.1.48 Idea 48

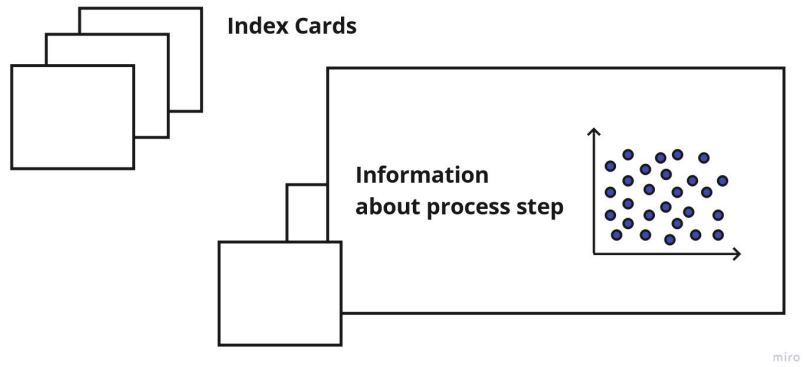


Figure A.48: Processing steps of the data are stacked like index cards. Select a card to show more information.

A.1.49 Idea 49

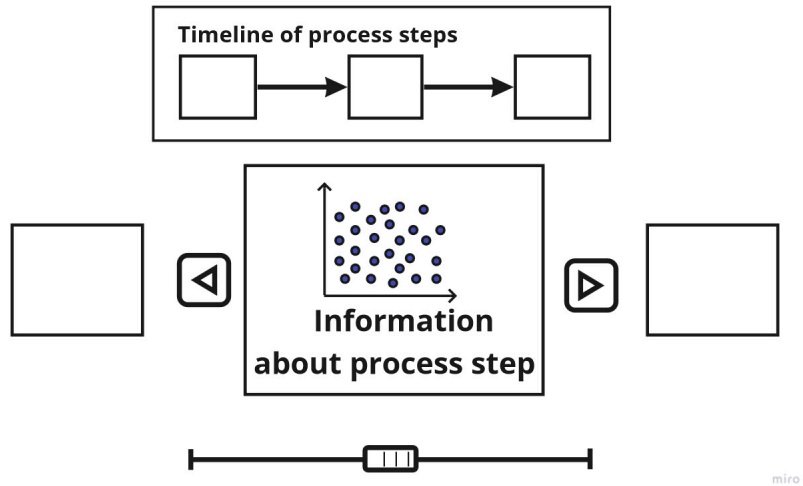


Figure A.49: Processing steps of the data are sorted in the form of a timeline. Navigation through arrows and a slider. Selected step is displayed bigger.

A.1.50 Idea 50

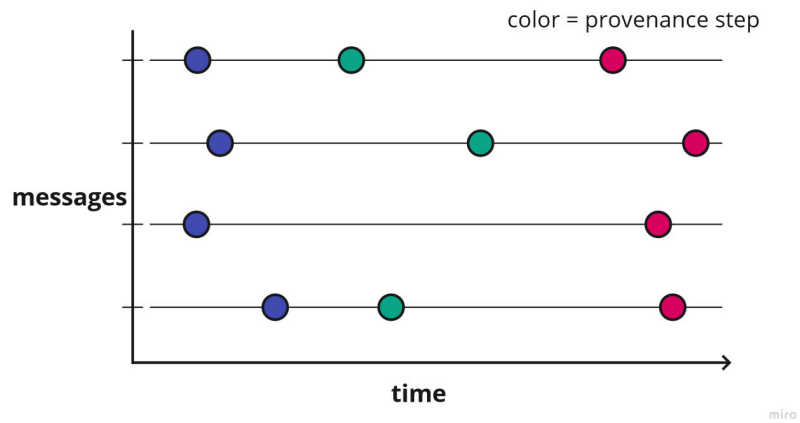


Figure A.50: Display message processing flow. Nodes represent processing steps.

A.1.51 Idea 51

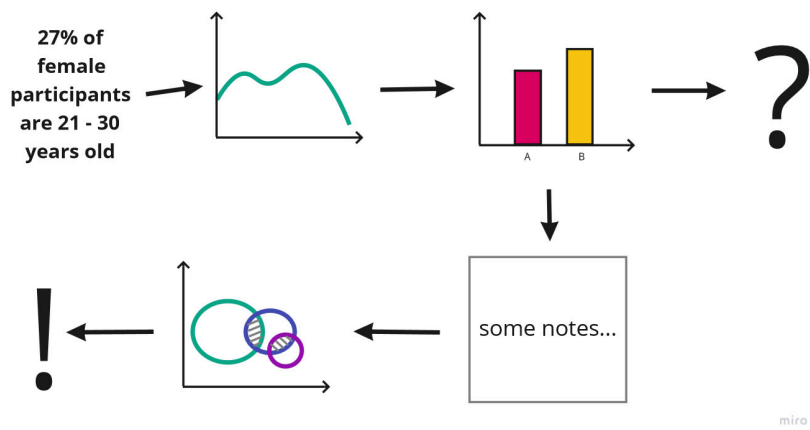


Figure A.51: Display the exploration process, including dead ends and final recommendations

A.1.52 Idea 52

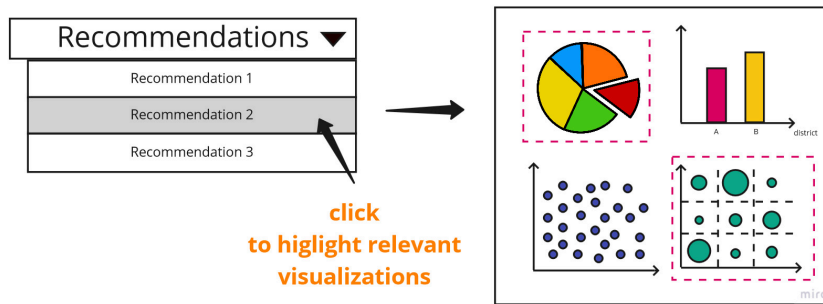


Figure A.52: Highlight relevant diagrams for a selected recommendation.

A.1.53 Idea 53

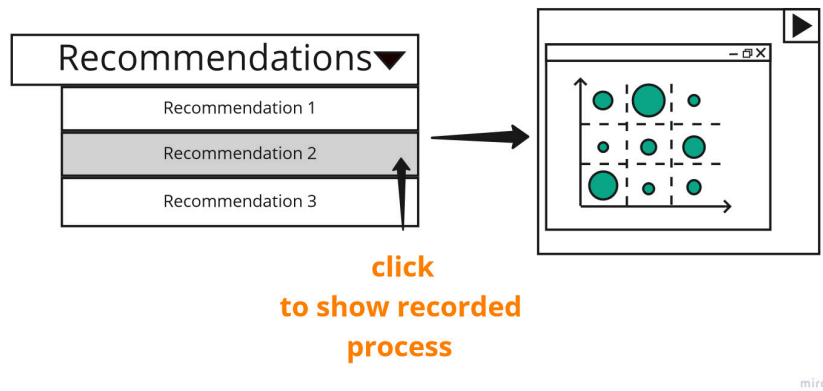


Figure A.53: Showing a video of the recorded process of finding insights for a selected recommendation.

A.1.54 Idea 54

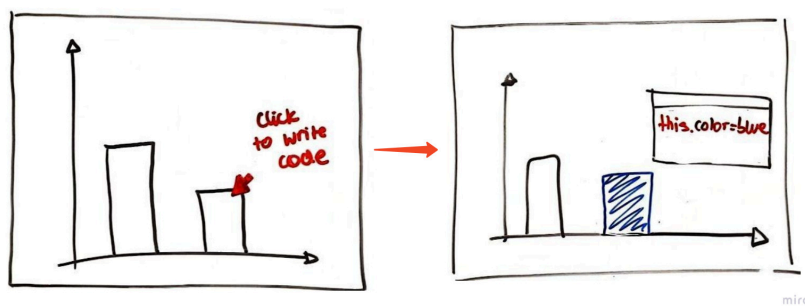


Figure A.54: Select or inspect visualization elements to adapt properties by code

A.1.55 Idea 55

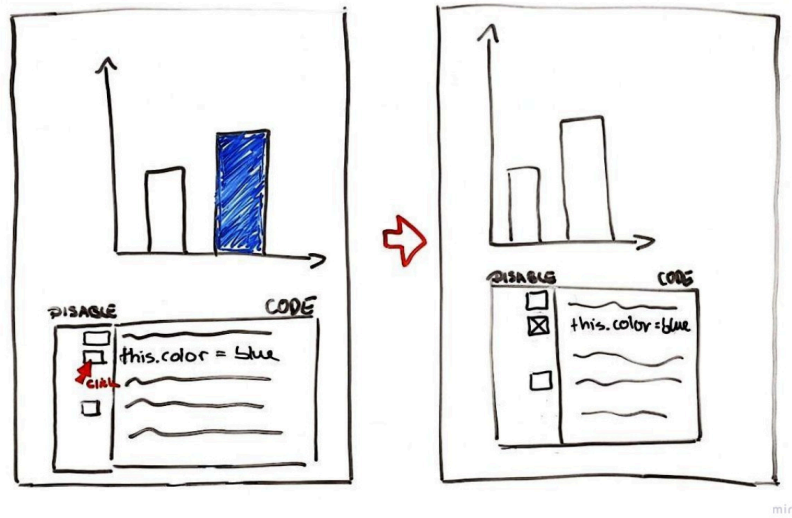


Figure A.55: Select or deselect lines of code to include or exclude the code. Adapt properties of visualization elements by code.

A.1.56 Idea 56

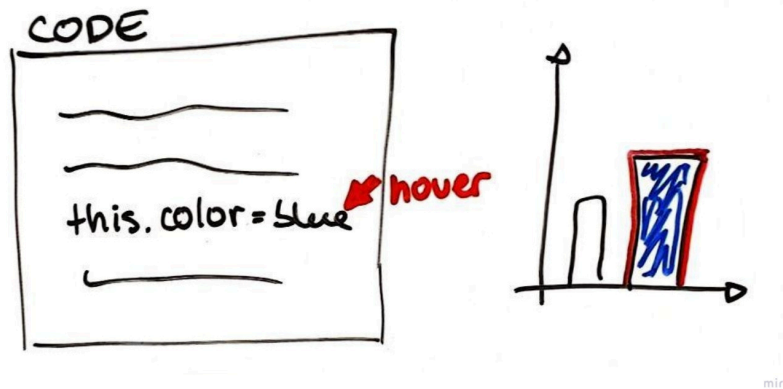


Figure A.56: Hover over a line of code to highlight the corresponding element in the diagram

A.1.57 Idea 57

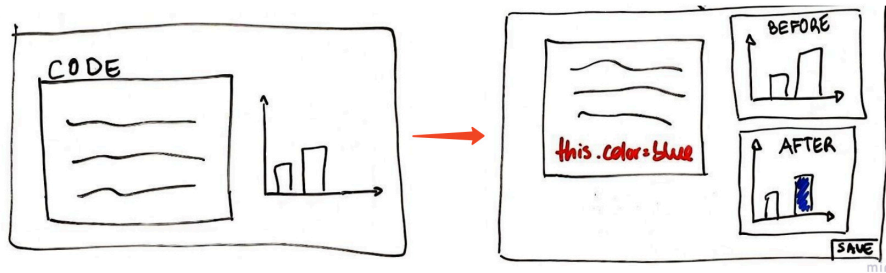


Figure A.57: Adapt code and compare the state of the visualization before and after the change directly in a window next to the code

A.1.58 Idea 58

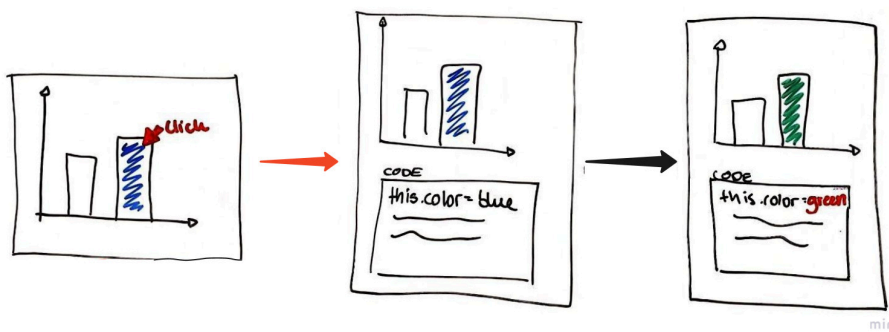


Figure A.58: Select element to edit the code. Code and visualization are then displayed next to each other. Changes are adapted directly.



A.1.59 Idea 59

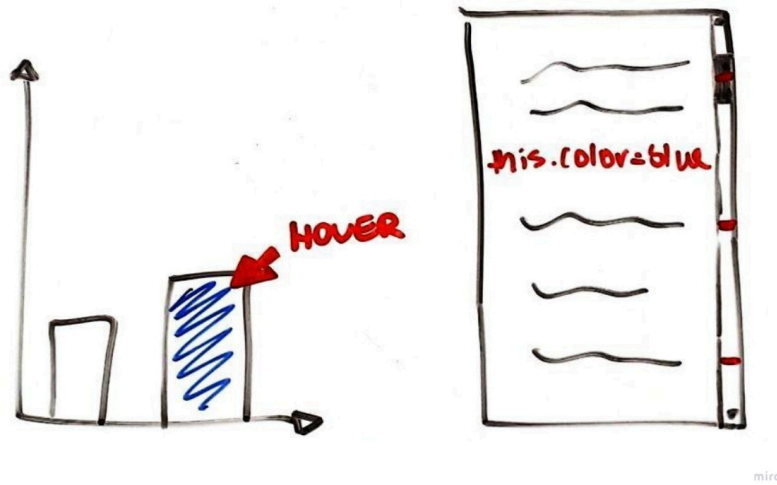


Figure A.59: Hover or select elements to highlight all parts of the code that adapt its properties

A.1.60 Idea 60



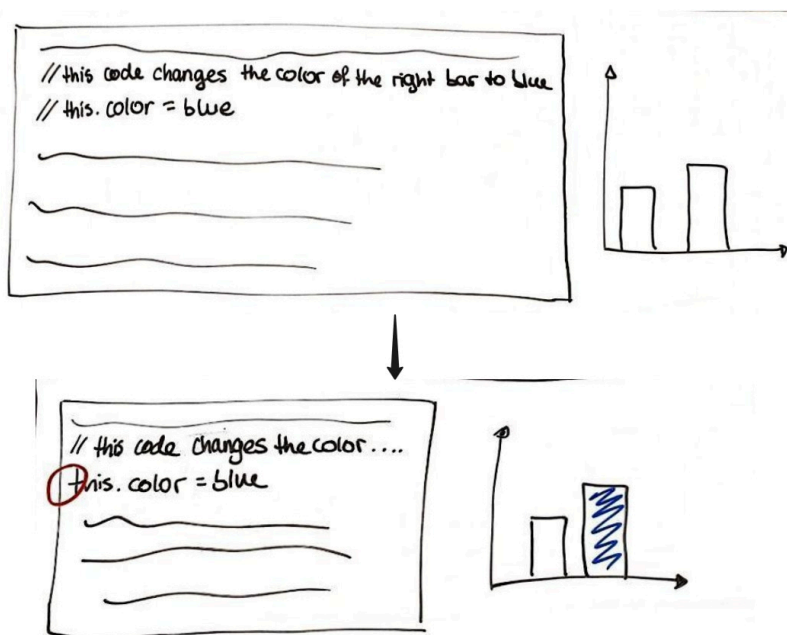
Figure A.60: Formulate the required changes as a statement. A suggestion is displayed that can be edited or accepted.

A.1.61 Idea 61



**Figure A.61:** Hovering over a line of code displays a description of what this line of code does to the element in the visualization. Dropdowns offer property values that can be selected.

A.1.62 Idea 62



**Figure A.62:** Comments describe what the line of code does. Uncommenting the following line(s) of code adapts visualization.

A.1.63 Idea 63

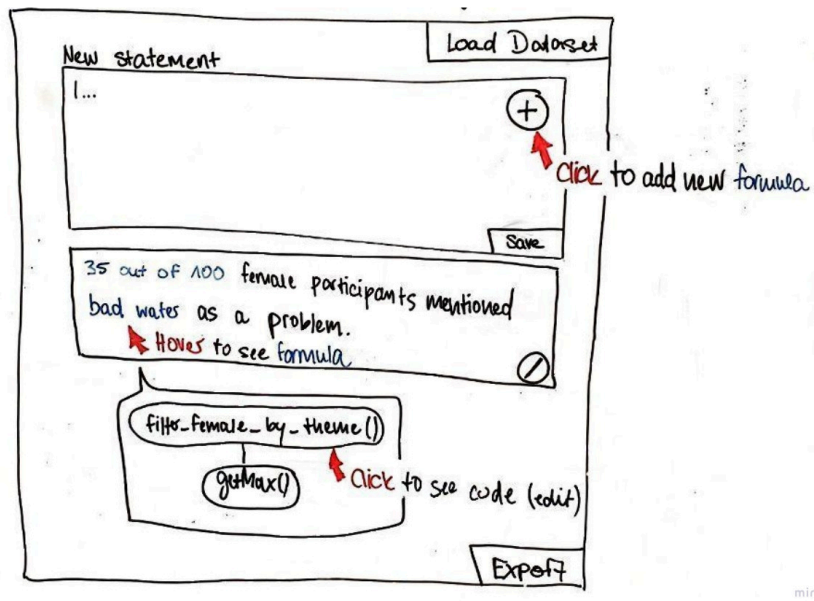


Figure A.63: Generating statements using code formulas and templates.

A.1.64 Idea 64

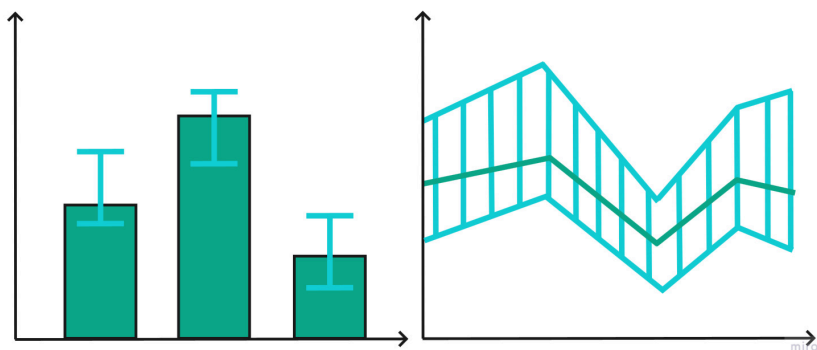


Figure A.64: Diagrams displaying divergence of missing data

A.1.65 Idea 65

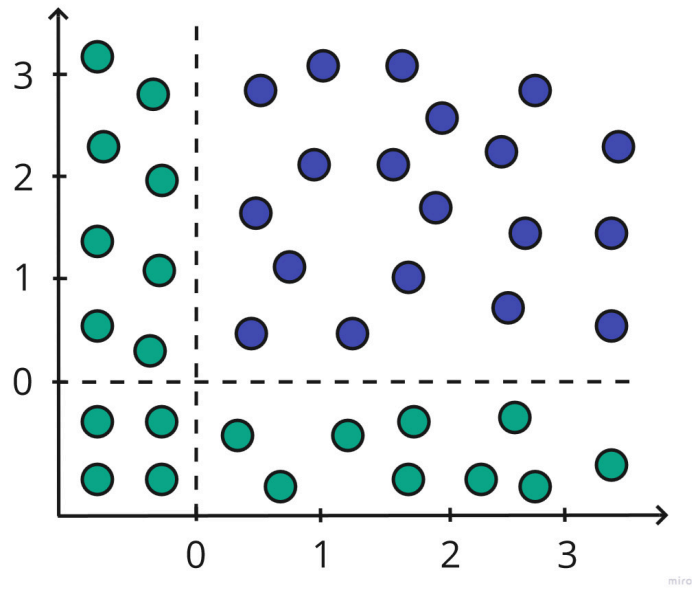


Figure A.65: Scatterplot with an area for missing values

A.1.66 Idea 66

Number of missing values per attribute

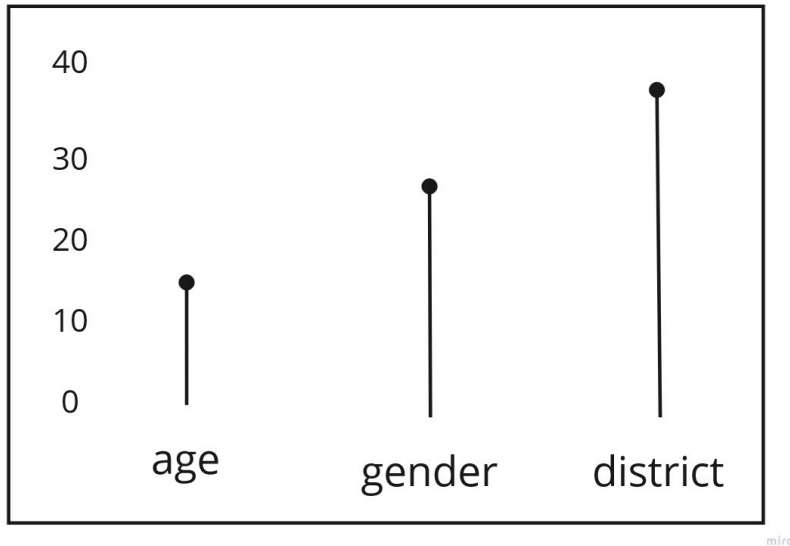


Figure A.66: Chart with number of missing values per attribute value

A.1.67 Idea 67

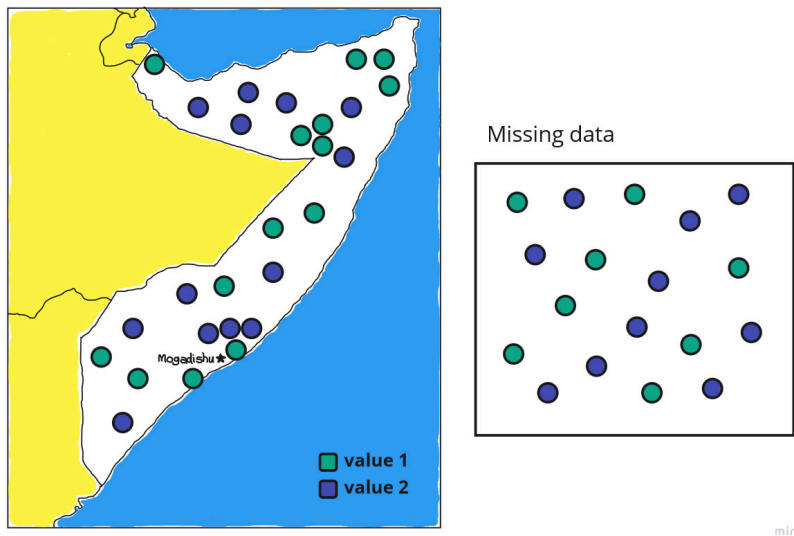


Figure A.67: Display box next to a map in which individuals are placed that have a missing value for their district

A.1.68 Idea 68

Distribution of combination of missing values

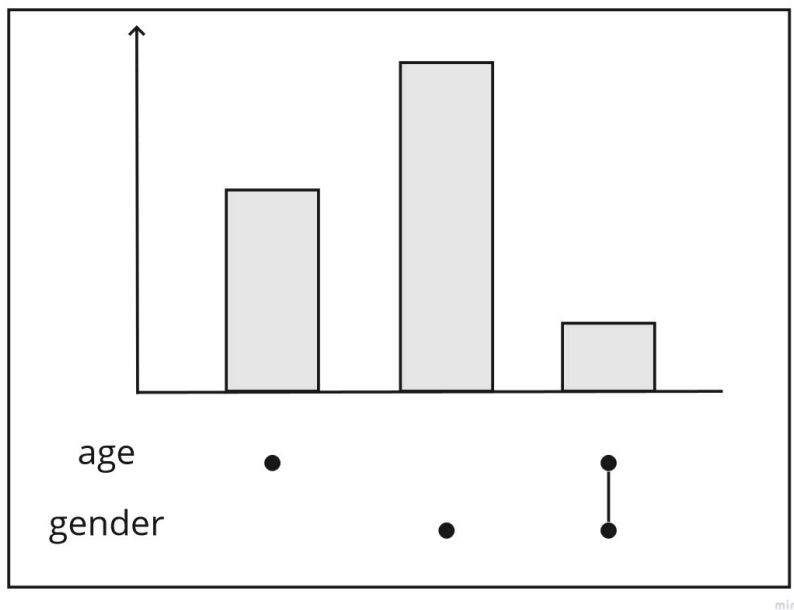


Figure A.68: Show number of missing values by attribute and combination of attributes

A.1.69 Idea 69

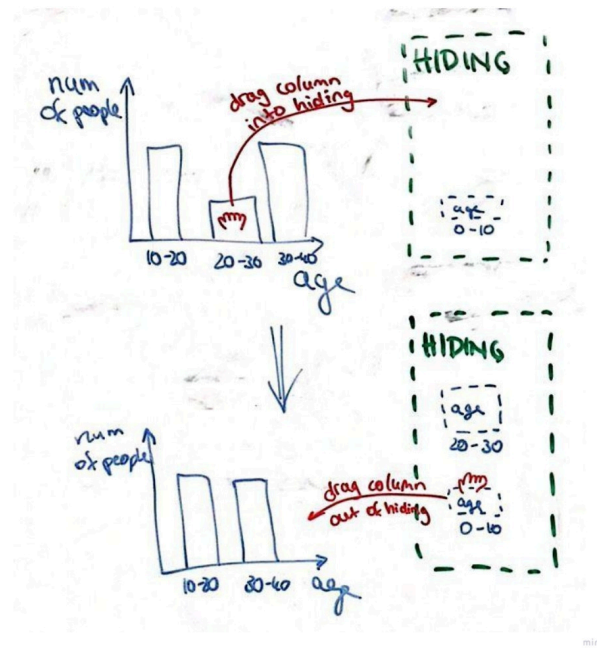


Figure A.69: Filtering by dragging and dropping bars into and out of hiding

A.1.70 Idea 70

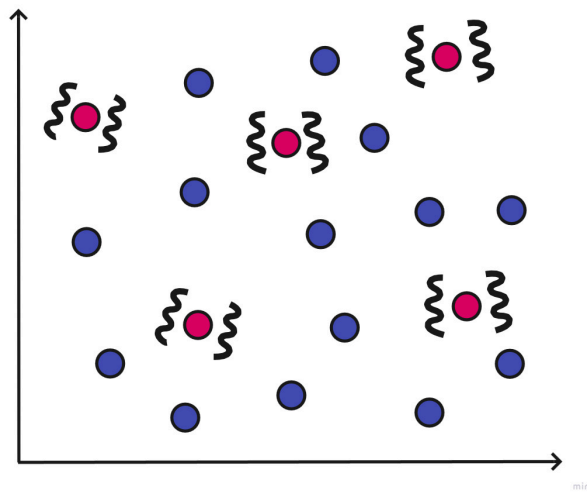


Figure A.70: Highlighting points by letting them vibrate

A.1.71 Idea 71

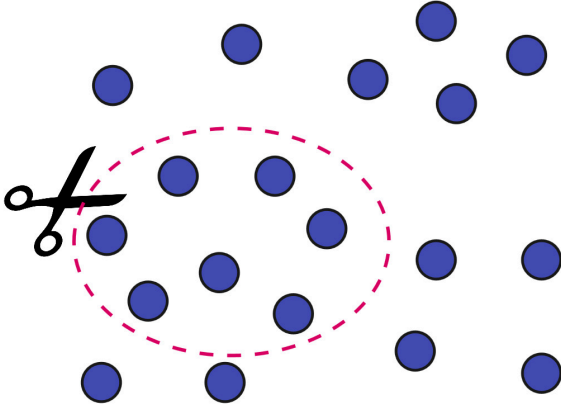


Figure A.71: Hand drawn selection lasso to select, group, or filter a subset of individuals

A.1.72 Idea 72

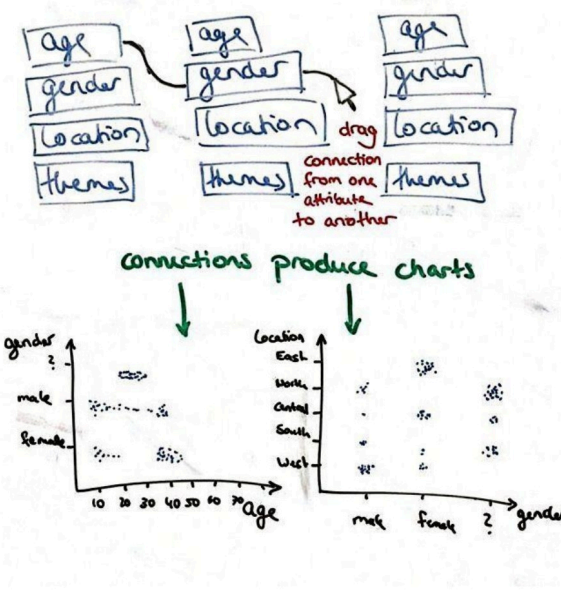
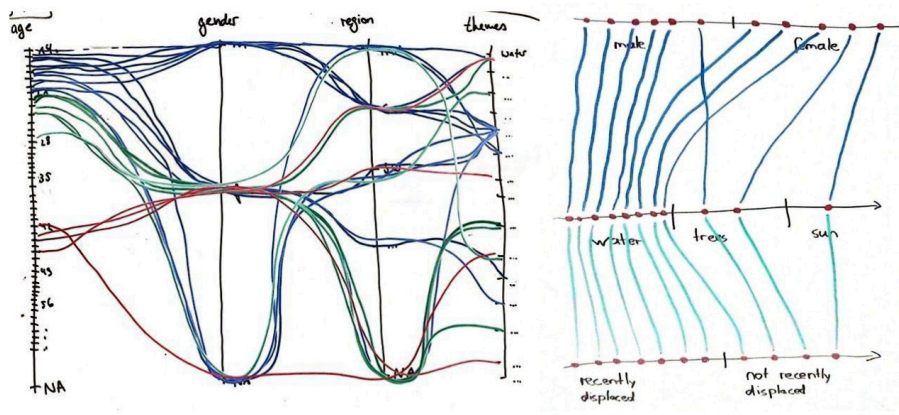


Figure A.72: Select axes attributes by connecting attribute values per drag and drop

### A.1.73 Idea 73

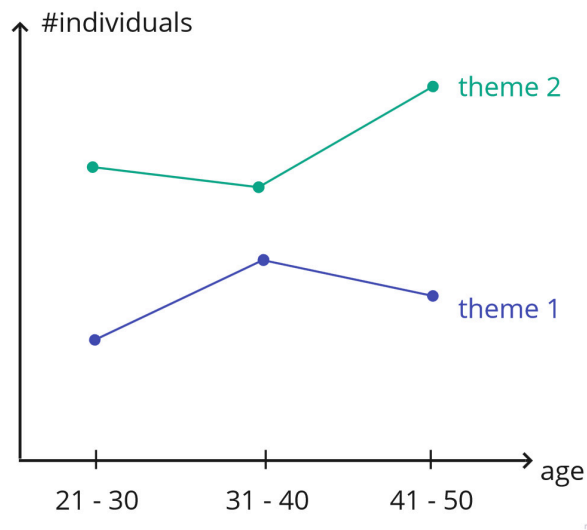


- choose axis to color code
- filter on axis to hide rest of data

miro

**Figure A.73:** Similar to a Sankey diagram. Multiple axes for different attributes. Individuals are represented by a line. Choose axis to color code. Filter on axis to hide the rest of the data.

### A.1.74 Idea 74

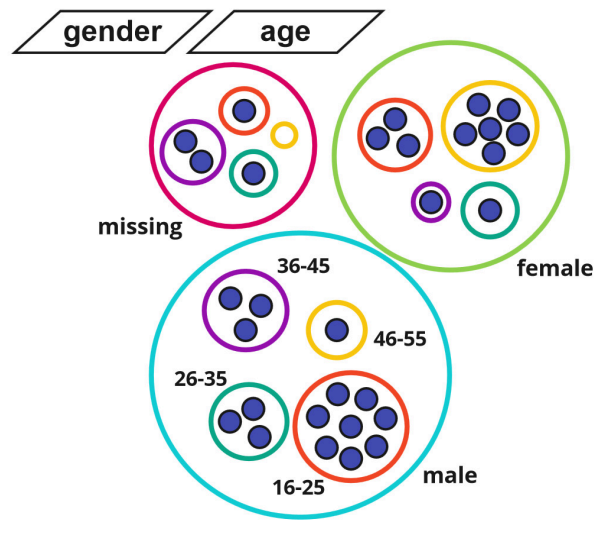


miro

**Figure A.74:** Line chart displaying the number of individuals of certain age groups who sent messages coded with certain themes

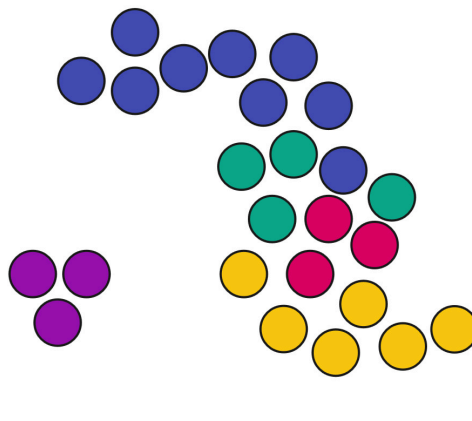


A.1.75 Idea 75



**Figure A.75:** Chaining groupings of individuals as points according to selected attributes. History of the chained attributes to undo or redo steps.

A.1.76 Idea 76



**Figure A.76:** Attracting individuals as points to each other depending on their similarity. Coloring individuals according to a selected attribute.

A.1.77 Idea 77

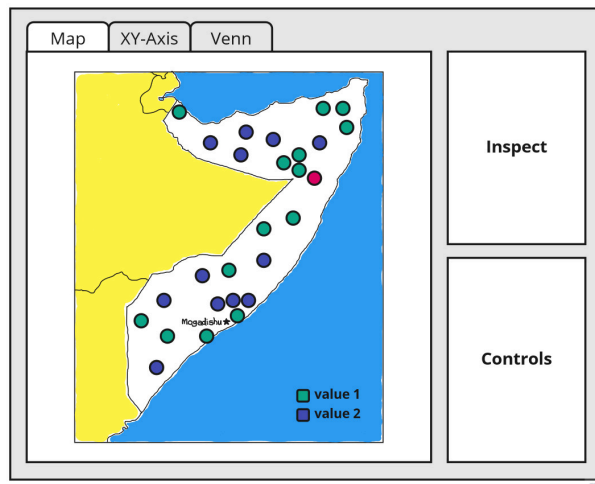


Figure A.77: Tabs containing one visualization type each. Inspector to inspect individuals. Control panel to color, filter, and select individuals in the visualizations.

A.1.78 Idea 78

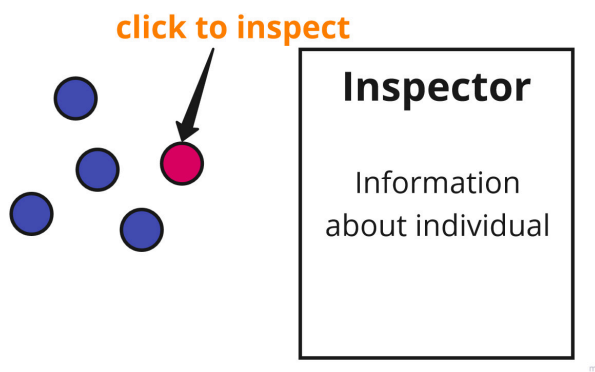


Figure A.78: Inspecting individuals as points to display information about attribute values

A.1.79 Idea 79

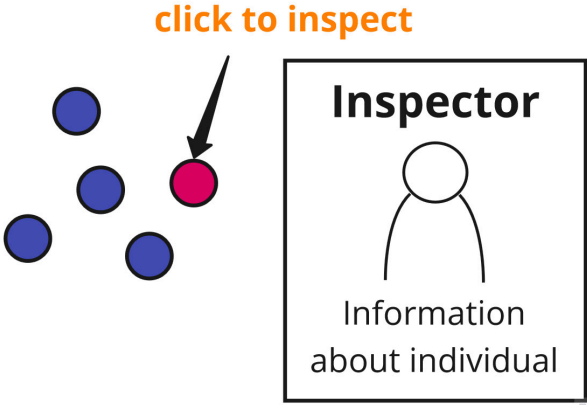


Figure A.79: Inspecting individuals as points to display information about attribute values and a (sketched) image of the individual

A.1.80 Idea 80

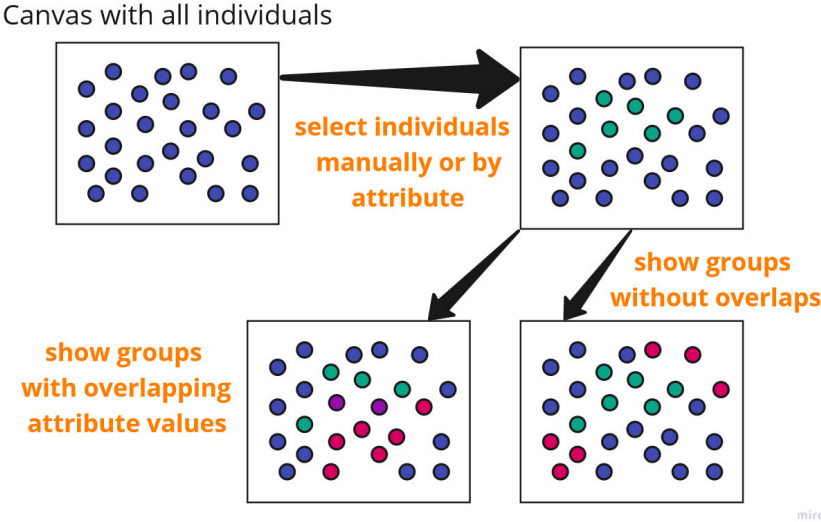
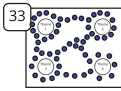
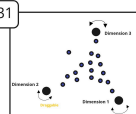
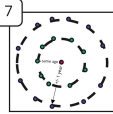
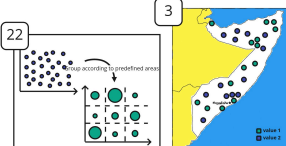

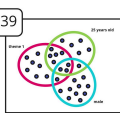
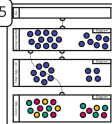
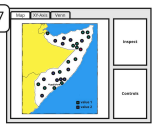

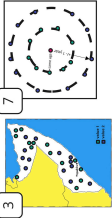
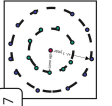
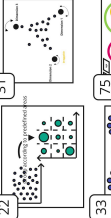



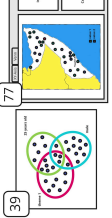

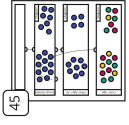

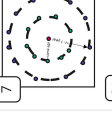
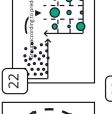

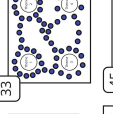



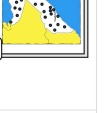





Figure A.80: Individuals as points on a canvas. Showing groups with and without overlaps

## A.2 Technical Categorization

Interaction level Representation mode	Manual	Mechanized	Instructable	Steerable	Automatic
<b>Pixel-oriented</b>		9 21			
<b>Geometric projection</b>	<p>Without interaction:</p> <p>1 2 20 30 37</p> <p>42 50 64 65 66</p> <p>67 68 73 74</p> <p>33</p>  <p>With interaction:</p> <p>18 44 48 54 56</p> <p>57 58 62</p> <p>31</p>  <p>7</p> 	<p>4 17 24 25 26</p> <p>27 34 41 49 55</p> <p>59 61</p> <p>22</p>  <p>3</p>		76	29 35
<b>Icon-based</b>	2 10	10			
<b>Hierarchy-based</b>	16	<p>36 38 40 43</p> <p>75</p>  <p>39</p> 			
<b>Graph-based</b>	<p>Without interaction:</p> <p>5 11 32 51</p> <p>With interaction:</p> <p>8 10 18 19</p>	6 9 10 11 19			
<b>Hybrid</b>	<p>47</p> <p>45</p> 	<p>45</p> <p>77</p> 			47
<b>Media-based</b>	1 11	11 12 13 53			
<b>No specific representation mode</b>	<p>23 69 71 72 79</p> <p>78</p> 				
<b>Other concepts</b>	14 15 46 80	<p>14 15 28 52 60</p> <p>70</p>	63	60 80	

## A.3 Task Categorization

Overview	Zoom	Filter	Details-on-demand	Relate	History	Extract
<p>1 2 3 4 5 6 7 8 9            10 11 12 13 14 15 16 17 18 19 20 21            22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40            41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72            73 74</p>        	<p>11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46</p> 	<p>4 6 14 15 24 63 69 71</p>	<p>34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 78 79</p> 	<p>5 7 8 9 10 20 24 25 26 27 29 30 32 37 38 40 41 42 43 44 45 57 60 73 74 80</p>        	<p>4 17 48 49 51 75</p>  	<p>47 63 71</p> 

## A.4 Interaction Level - Task Categorization

Task Representation mode	Overview	Zoom	Filter	Details-on- demand	Relate	History	Extract
<b>Pixel-based</b>	9 21				9		
<b>Geometric projection</b>	1 2 3 4 7 18 20 22 24 25 30 31 33 42 44 50 59 64 65 66 67 68 73 74		4 24	34 35 48 49 54 55 56 58 61 62	7 20 22 24 25 26 27 29 30 31 33 37 41 42 44 57 60 73 74 76	4 48 49 17	
<b>Icon-based</b>	2 10				10		
<b>Hierarchy- based</b>	16 39 40 75	16		36	38 39 40 43 75		
<b>Graph-based</b>	5 6 8 9 10 18 19 32 51	11	6		5 8 9 32	51	
<b>Hybrid</b>	47 77	45			45 77	45	45 47
<b>Media-based</b>	1 11	11 12 13		53			
<b>No specific representation mode</b>	23 72		69 71	78 79			71
<b>Other concepts</b>	28 46 70	46	14 15 63	46 52	60 80		63

## A.5 Representation Mode - Task Categorization

Task Interaction level	Overview	Zoom	Filter	Details-on- demand	Relate	History	Extract
<b>Manual</b>	1 2 5 7 8 10 11 16 18 19 20 21 23 30 31 32 33 42 44 46 47 50 51 64 65 66 67 68 72 73 74	11 16 45 46 11 12 13 45	69 71	46 48 54 56 58 62 78 79	5 7 8 10 30 20 31 32 33 37 42 44 45 57 73 74	48 49 51	45 71
<b>Mechanized</b>	3 4 6 9 10 11 22 24 25 28 39 40 59 70 75 77	11 12 13 45	4 6 14 15 24	34 36 49 52 53 55 61	9 10 22 24 25 26 27 38 39 40 41 43 45 75 77	4 17 45 75	45
<b>Instructable</b>			63				63
<b>Steerable</b>					60 76 80		
<b>Automatic</b>				35	29		47





## B Appendix Chapter 5

Listing B.1: Utility functions for the scenario implementations

```
1 export function getMax(arrayOfNumbers) {
2   let max = -1
3
4   arrayOfNumbers.forEach(number => {
5     if (number > max) max = number
6   })
7
8   return max
9 }
10
11 export function removeUnneededData(data) {
12   let unusableData = []
13   data.forEach(individual => {
14     if (individual.consent_withdrawn === "TRUE") {
15       unusableData.push(individual)
16     }
17   })
18
19   unusableData.forEach(individual => {
20     data.splice(data.indexOf(individual), 1)
21   })
22
23   data.forEach((individual, index) => {
24     individual.id = index
25     delete individual.themes
26     delete individual.languages
27     delete individual.consent_withdrawn
28     delete individual.recently_displaced
29     delete individual.end_date
30     delete individual.start_date
31   })
32 }
33
34 export function getHexStringWithTwoCharacters(number) {
35   if (number == 0) {
36     return "00"
37   }
38
39   let result = ""
40   if (number < 16) {
```

```
41   result += "0"
42 }
43
44 return result + number.toString(16)
45 }
46
47 export function createAgeGroups(data) {
48   let groups = {}
49
50   data.forEach(element => {
51     let bucket = getAgeBucketKey(element.age)
52     if (!groups[bucket]) {
53       groups[bucket] = {"data": []}
54     }
55     groups[bucket].data.push(element)
56   })
57
58   return groups
59 }
60
61 export function getAgeBucketKey(age) {
62   switch(true) {
63     case age < 10:
64       return "<10"
65     case age < 15:
66       return "10–14"
67     case age < 18:
68       return "15–17"
69     case age < 36:
70       return "18–35"
71     case age < 55:
72       return "36–54"
73     case age === "missing":
74       return "missing"
75     default:
76       return "over 55"
77   }
78 }
```

**Listing B.2:** Page for the application of the evaluation scenario with the double rendering strategy

```
1 <style>
2 #diagram {
3   width: 300px;
4   height: 300px;
5   border: 1px solid black;
6 }
7 </style>
8
9 <canvas id="diagram"></canvas>
```

```

10
11 <script>
12 import { AVFParser } from "https://lively-kernel.org/voices/parsing
    -data/avf-parser.js";
13
14 import {
15   getColorStringFromImageData,
16   drawBars,
17   createColorMapping,
18   createColoredAgeGroups
19 } from "./double-rendering.js";
20
21 import { removeUnneededData } from "./utils.js"
22
23 let canvasDimensions, context, groups, colorMapping,
    identifyingImageData
24
25 ;
26 (async () => {
27   //canvas preparation
28   let diagram = lively.query(this, "#diagram")
29   context = diagram.getContext('2d')
30   canvasDimensions = {width: 300, height: 300}
31
32   //data preparation
33   let data = await AVFParser.loadCovidData()
34   removeUnneededData(data)
35   groups = createColoredAgeGroups(data)
36   colorMapping = createColorMapping(groups)
37
38   //drawing with identifying colors
39   drawBars(context, canvasDimensions, groups, "uniqueColor")
40   identifyingImageData = context.getImageData(0, 0,
    canvasDimensions.width, canvasDimensions.height)
41
42   //drawing with colors meant for users
43   drawBars(context, canvasDimensions, groups, "color")
44
45   //waiting for a click
46   diagram.addEventListener("click", event => {
47     lively.openInspector(getClickedGroup(event))
48   })
49 })()
50
51 function getClickedGroup(event) {
52   let colorString = getColorStringFromImageData(
53     identifyingImageData,
54     event.layerX,
55     event.layerY
56   )
57   let groupKey = colorMapping[colorString]

```

```

58   let group = groups[groupKey]
59
60   return group
61 }
62 </script>

```

**Listing B.3:** Functions for the implementation of the double rendering strategy

```

1  import {
2    getHexStringWithTwoCharacters,
3    createAgeGroups,
4    getMax
5  } from "./utils.js"
6
7  export function getColorStringFromImageData(imageData, x , y) {
8    let startPosition = (y * imageData.width + x) * 4
9
10   let red = getHexStringWithTwoCharacters(imageData.data[
11     startPosition])
12   let green = getHexStringWithTwoCharacters(imageData.data[
13     startPosition+1])
14   let blue = getHexStringWithTwoCharacters(imageData.data[
15     startPosition+2])
16
17   return "#" + red + green + blue
18 }
19
20 export function drawBars(context, canvasDimensions, groups,
21   colorSelector) {
22   context.clearRect(0, 0, canvasDimensions.width, canvasDimensions.
23     height)
24
25   let groupKeys = Object.keys(groups)
26
27   let dataLengths = groupKeys.map(key => groups[key].data.length)
28   let max = getMax(dataLengths)
29
30   groupKeys.forEach((groupKey, index) => {
31     let barHeight = Math.floor(canvasDimensions.height * (groups[
32       groupKey].data.length / max))
33     let barWidth = Math.floor(canvasDimensions.width / groupKeys.
34       length)
35
36     context.fillStyle = groups[groupKey][colorSelector]
37
38     context.fillRect(
39       barWidth * index,
40       0,
41       barWidth - 5,
42       barHeight
43     )
44   })
45 }

```

```

37   })
38 }
39
40 export function createColorMapping(groups) {
41   let mapping = {}
42
43   Object.keys(groups).forEach(groupKey => {
44     mapping[groups[groupKey].uniqueColor] = groupKey
45   })
46
47   return mapping
48 }
49
50 export function createColoredAgeGroups(data) {
51   let groups = createAgeGroups(data)
52
53   Object.keys(groups).forEach((groupKey, index) => {
54     groups[groupKey]["color"] = "#ffee00"
55     groups[groupKey]["uniqueColor"] = "#ffee0" + index.toString(16)
56   })
57
58   return groups
59 }

```

**Listing B.4:** Page for the application of the evaluation scenario with the position matching strategy

```

1 <style>
2 #diagram {
3   width: 300px;
4   height: 300px;
5   border: 1px solid black;
6 }
7 </style>
8
9 <canvas id="diagram"></canvas>
10
11 <script>
12 import { AVFParser } from "https://lively-kernel.org/voices/parsing
13   -data/avf-parser.js";
14
15 import {
16   positionMatchesGroup,
17   createPositionedAgeGroups,
18   drawBars
19 } from "./position-matching.js";
20
21 import { removeUnneededData } from "./utils.js"
22
23 let groups, context, canvasDimensions

```

```

24 ;
25 (async () => {
26   //canvas preparation
27   let diagram = lively.query(this, "#diagram")
28   context = diagram.getContext('2d')
29   canvasDimensions = {width: 300, height: 300}
30
31   //data preparation
32   let data = await AVFParser.loadCovidData()
33   removeUnneededData(data)
34   groups = createPositionedAgeGroups(data, canvasDimensions)
35
36   //rendering the image
37   drawBars(context, canvasDimensions, groups)
38
39   //waiting for a click
40   diagram.addEventListener("click", event => {
41     lively.openInspector(getClickedGroup(event))
42   })
43 })()
44
45 function getClickedGroup(event) {
46   let groupArray = Object.keys(groups).map(key => groups[key])
47   let position = {x: event.layerX, y: event.layerY}
48
49   let results = []
50   groupArray.forEach(group => {
51     if (positionMatchesGroup(position, group)) results.push(group)
52   })
53
54   return results
55 }
56
57 </script>

```

**Listing B.5:** Functions for the implementation of the position matching strategy

```

1 import {
2   createAgeGroups,
3   getMax
4 } from "./utils.js"
5
6 export function drawBars(context, canvasDimensions, groups) {
7   context.clearRect(0, 0, canvasDimensions.width, canvasDimensions.
8     height)
9
10  Object.keys(groups).forEach(groupKey => {
11    let group = groups[groupKey]
12
13    context.fillStyle = "ffee00"

```

```

14   context.fillRect(group.x, group.y, group.width, group.height)
15   })
16 }
17
18 export function createPositionedAgeGroups(data, canvasDimensions) {
19   let groups = createAgeGroups(data)
20
21   let groupKeys = Object.keys(groups)
22   let dataLengths = groupKeys.map(key => groups[key].data.length)
23
24   let max = getMax(dataLengths)
25
26   groupKeys.forEach((groupKey, index) => {
27     let group = groups[groupKey]
28
29     let barHeight = Math.floor(canvasDimensions.height * (group.
30       data.length / max))
31     let barWidth = Math.floor(canvasDimensions.width / groupKeys.
32       length)
33
34     group["x"] = barWidth * index
35     group["y"] = 0
36     group["width"] = barWidth - 5
37     group["height"] = barHeight
38   })
39
40   return groups
41 }
42
43 export function positionMatchesGroup(position, group) {
44   return position.x >= group.x &&
45     position.x <= group.x + group.width &&
46     position.y >= group.y &&
47     position.y <= group.y + group.height
48 }

```

**Listing B.6:** Page for the application of the evaluation scenario with the tracing graph strategy

```

1 <svg id="diagram"></svg>
2
3 <script>
4 import { AVFParser } from "https://lively-kernel.org/voices/parsing
5   -data/avf-parser.js";
6 import { removeUnneededData, createAgeGroups, getMax } from "./
7   utils.js"
8 import d3 from "src/external/d3.v5.js";
9 ;
10 (async () => {

```

```

11 //data preparation
12 let data = await AVFParseer.loadCovidData()
13 removeUnneededData(data)
14 let groups = createAgeGroups(data)
15 groups = Object.keys(groups).map(key => {return {group: key, data
    : groups[key].data}})
16
17 let dataSizes = groups.map(group => group.data.length)
18 let max = getMax(dataSizes)
19
20 //canvas preparation
21 let diagram = lively.query(this, "#diagram")
22 let canvasDimensions = {width: 300, height: 300}
23 let margin = 5
24 let barContainerWidth = Math.floor(canvasDimensions.width /
    groups.length)
25
26 let svg = d3.select(diagram)
27   .attr("width", canvasDimensions.width)
28   .attr("height", canvasDimensions.height)
29   .selectAll('rect').data(groups)
30   .enter().append("rect")
31     .attr("width", (group) => {return barContainerWidth - margin
    })
32     .attr("height", (group) => {return Math.floor(
    canvasDimensions.height * (group.data.length / max))})
33     .attr("x", (group) => {return barContainerWidth * groups.
    indexOf(group)})
34     .attr("y", (group) => {return canvasDimensions.height - Math.
    floor(canvasDimensions.height * (group.data.length / max))
    })
35     .on("click", (group) => {
36       lively.openInspector(group)
37     })
38
39 console.log(svg)
40 })()
41
42 </script>

```



# C Appendix Chapter 6

## C.1 Benchmark protocol

### C.1.1 General

**Name of protocolant:**

- Moritz Spranger

**Date:**

- First benchmark run (v1): 19.07.2020
- Second benchmark run (v2): 26.07.2020

**Benchmark objective:**

- The goal of this benchmark series is to compare web technologies for rendering points in the browser. For this purpose, the “time to render” is measured and the heap consumption of the technologies. The benchmarks are performed both in the live programming environment Lively4 and on a locally running server without Lively4. This allows us to additionally evaluate how big the performance difference between the technologies in Lively4 is.

**Software language:** - Javascript (HTML + CSS)

### C.1.2 Hardware dependencies

- GPU
- Network

### C.1.3 Software dependencies

- Browser
- Lively4

### C.1.4 System and environment

#### C.1.4.1 Hardware

**Machine:**

- Name: MacBook Pro 16", 2019
- Model identifier: MacBookPro16,1

**CPU:**

- Name: 2,4 GHz 8-Core Intel Core i9
- Cores: 8
- Speed: 2,4 GHz

- L2 Cache (per core): 256 KB
- L3 Cache: 16 MB
- Hyper-Threading: enabled

**GPU:**

- Name: AMD Radeon Pro 5500M
- VRAM: 4 GB
- PCIe Lane Width: x8
- EFI Driver Version: 01.01.190
- gMux Version: 5.0.0
- Name: Intel UHD Graphics 630 (build-in)
- VRAM: 1536 MB
- gMux Version: 5.0.0

**RAM:**

- 32 GB 2667 MHz DDR4

**Network card:**

- Name: AirPort Extreme (0x14E4, 0x7BF)

#### C.1.4.2 Software

**OS:**

- Version: macOS 10.15.5
- Kernel Version: Darwin 19.5.0

**Google Chrome:**

- Version 83.0.4103.116 (Official Build) (64-bit)

**Active Add-ons in Chrome:**

- Bitwarden
- AdblockPlus
- SessionBuddy v3.6.4
- VueJs Dev Tools
- Facebook Pixel Helper

#### C.1.5 Recorded Benchmark Measurements

The data of all test series can be accessed via Voices repository in Lively4.<sup>1</sup>

---

<sup>1</sup>[https://lively-kernel.org/lively4/lively4-bp2019/start.html?edit=https://lively-kernel.org/voices/BP2019RH1-report/content/topic6/benchmark\\_results/cycle\\_data](https://lively-kernel.org/lively4/lively4-bp2019/start.html?edit=https://lively-kernel.org/voices/BP2019RH1-report/content/topic6/benchmark_results/cycle_data).

## C.2 Benchmark Results

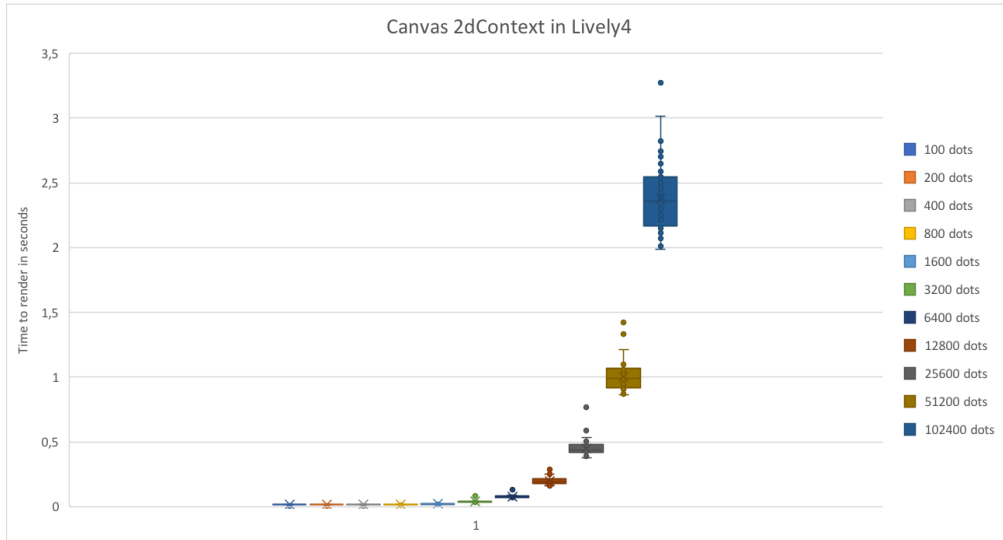


Figure C.1: Box plot for rendering points on canvas 2dContext in Lively4

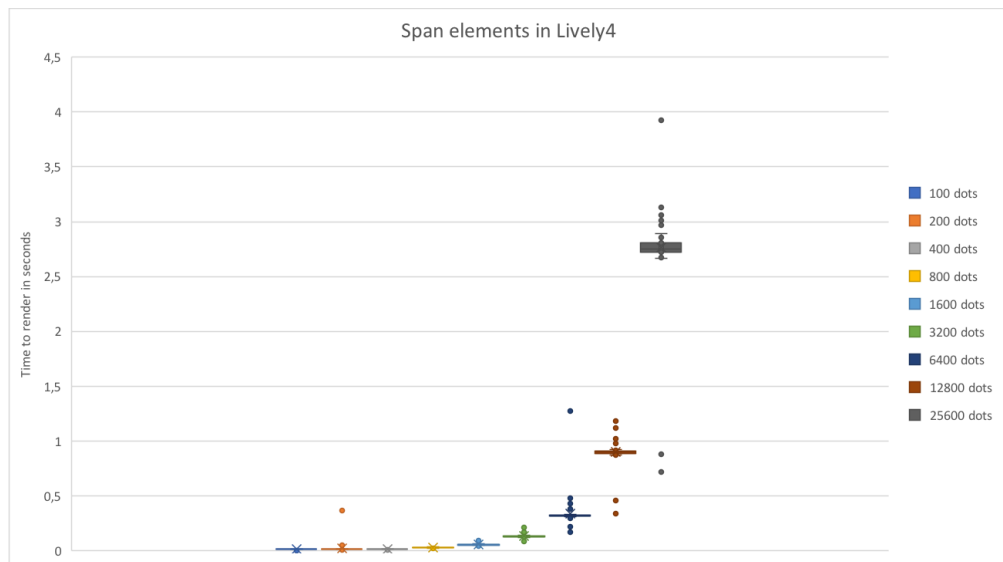


Figure C.2: Box plot for rendering points with HTML <span>-elements in Lively4

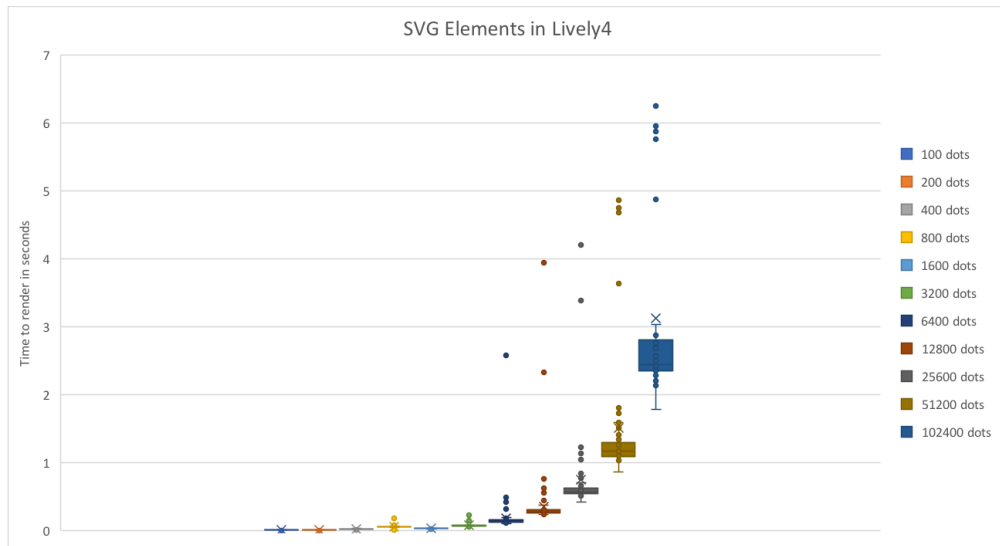


Figure C.3: Box plot for rendering points with SVG in Lively4

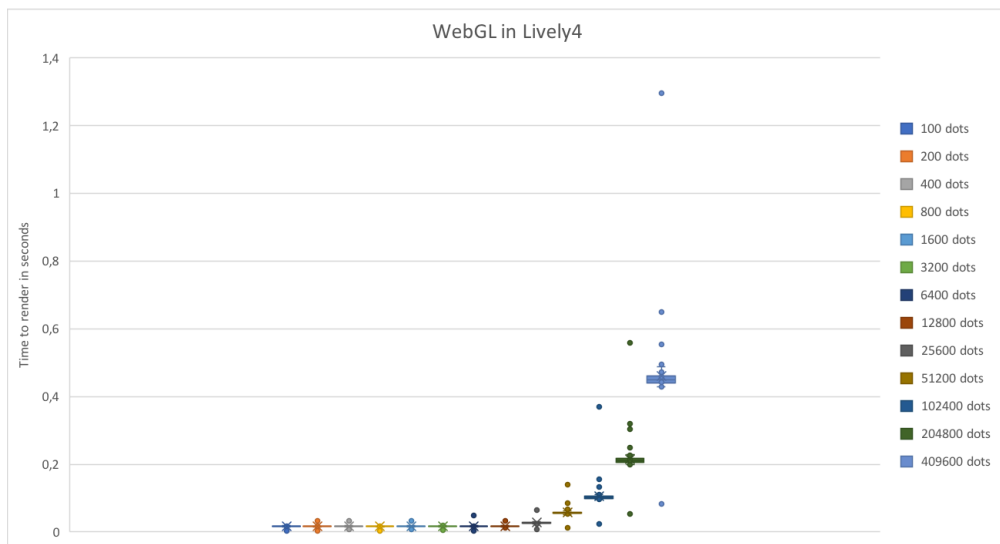


Figure C.4: Box plot for rendering points on canvas with WebGL in Lively4

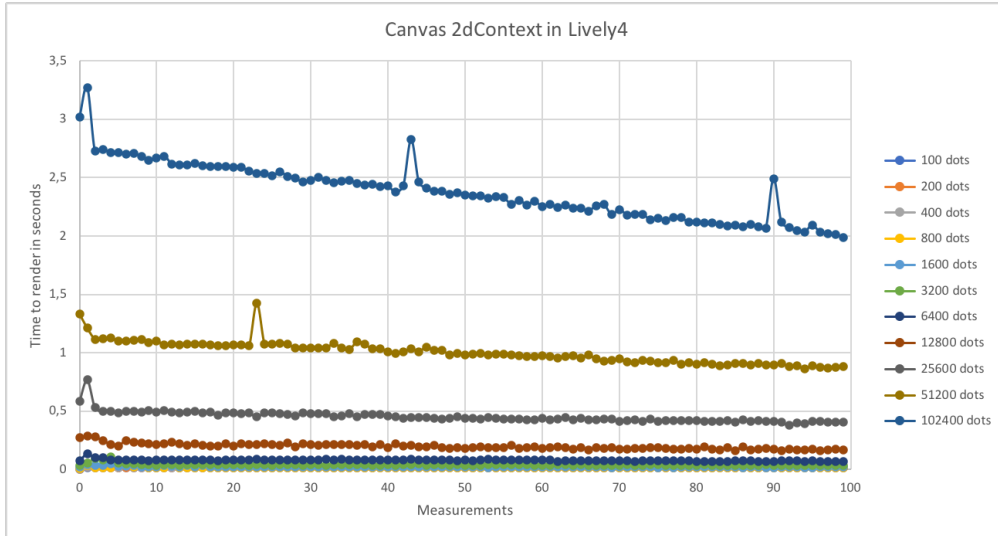


Figure C.5: Time to render when rendering points with canvas 2dContext in Lively4

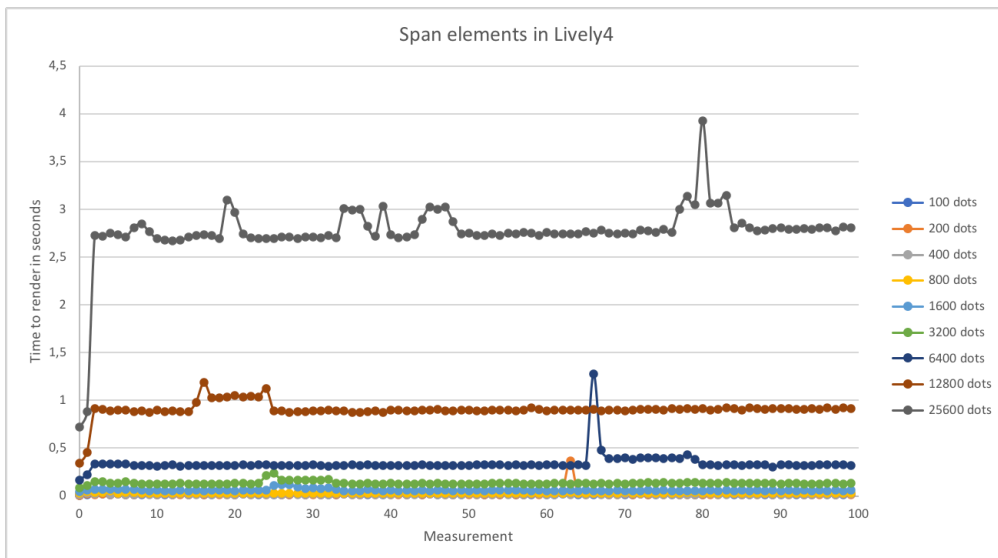


Figure C.6: Time to render when rendering points with <span>-elements in Lively4

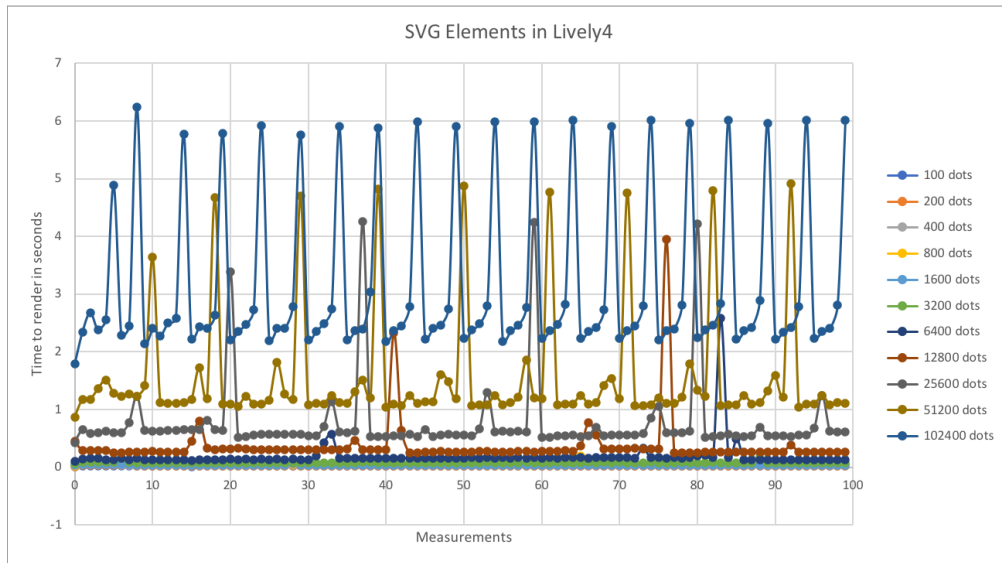


Figure C.7: Time to render when rendering points with SVG in Lively4

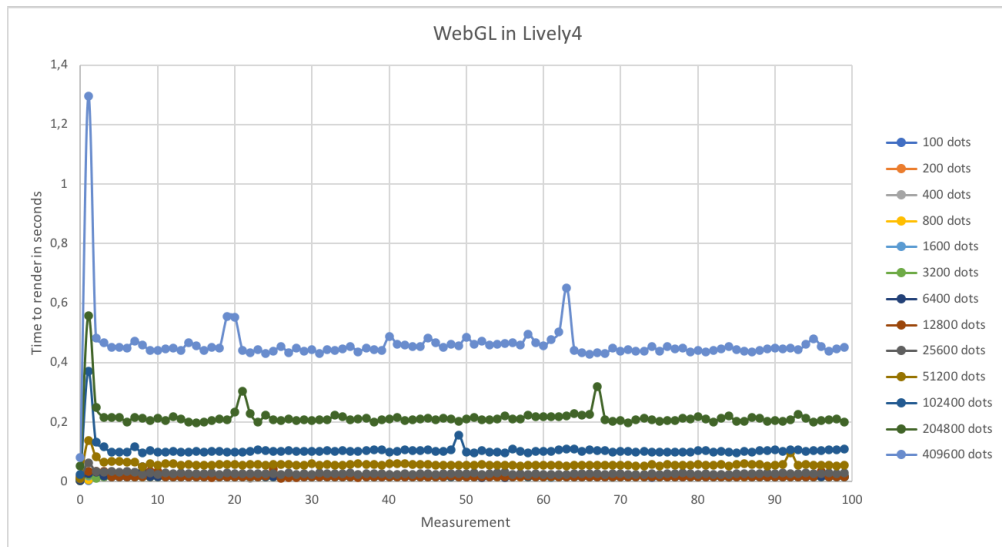


Figure C.8: Time to render when rendering points with WebGL in Lively4

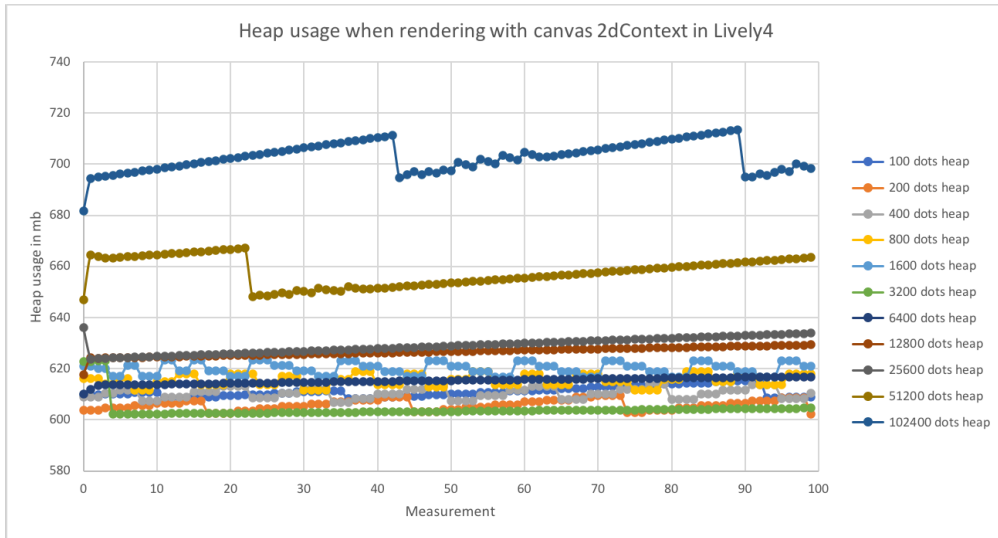


Figure C.9: Heap usage when rendering with canvas 2dContext in Lively4

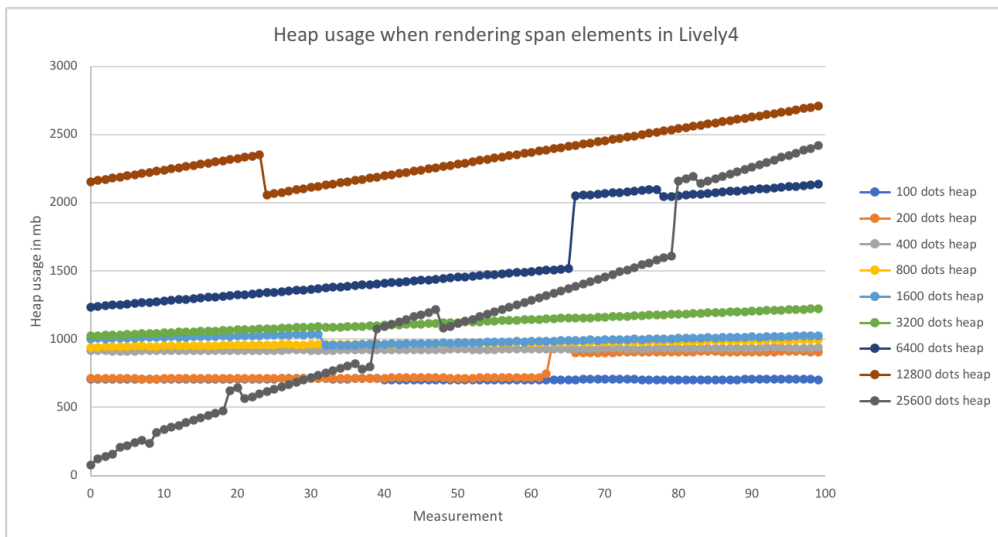


Figure C.10: Heap usage when rendering points with <span>-elements in Lively4

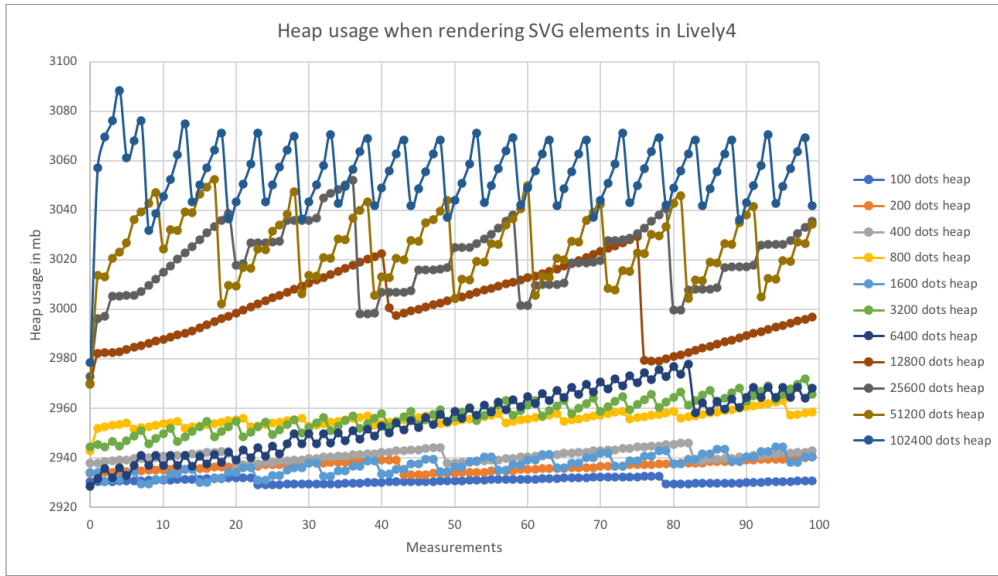


Figure C.11: Heap usage when rendering with SVG in Lively4

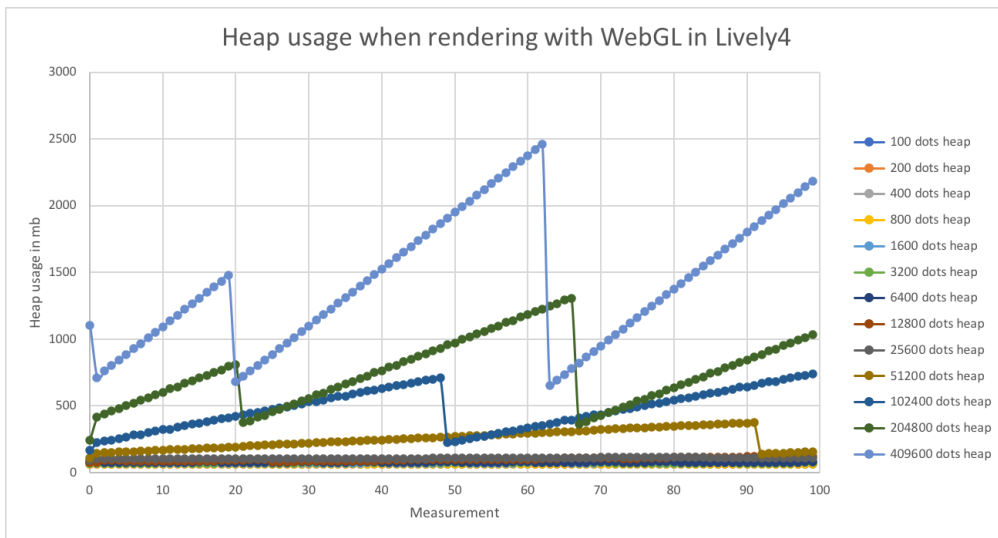


Figure C.12: Heap usage when rendering with WebGL in Lively4



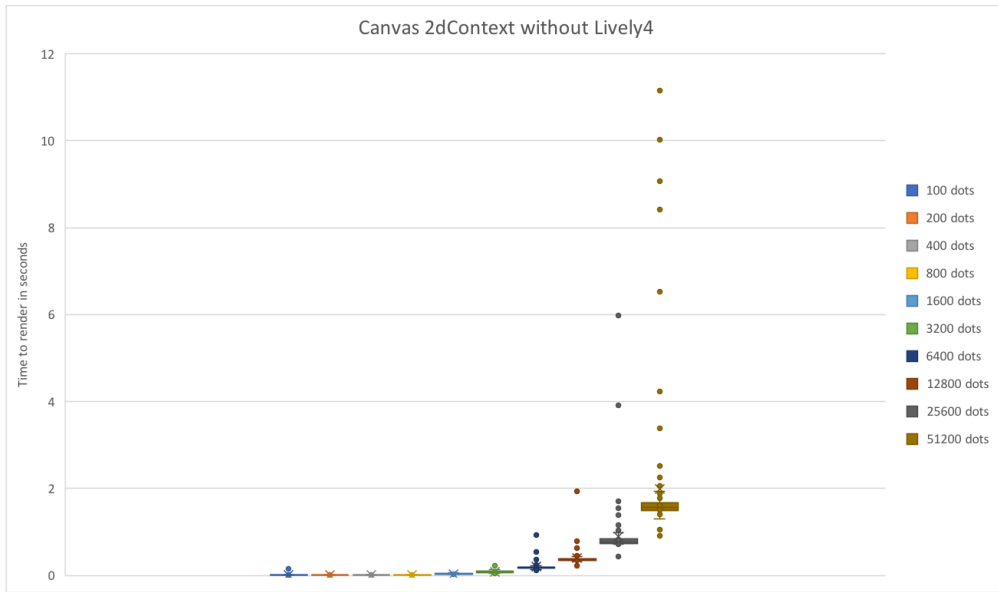


Figure C.13: Box plot for rendering points on canvas 2dContext on local server

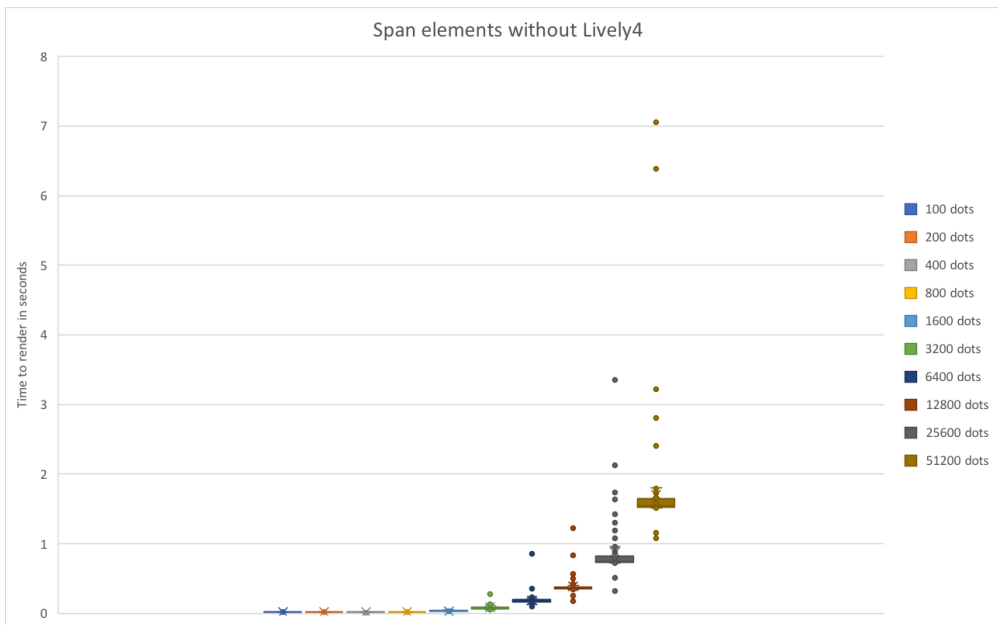


Figure C.14: Box plot for rendering points with <span>-elements on local server

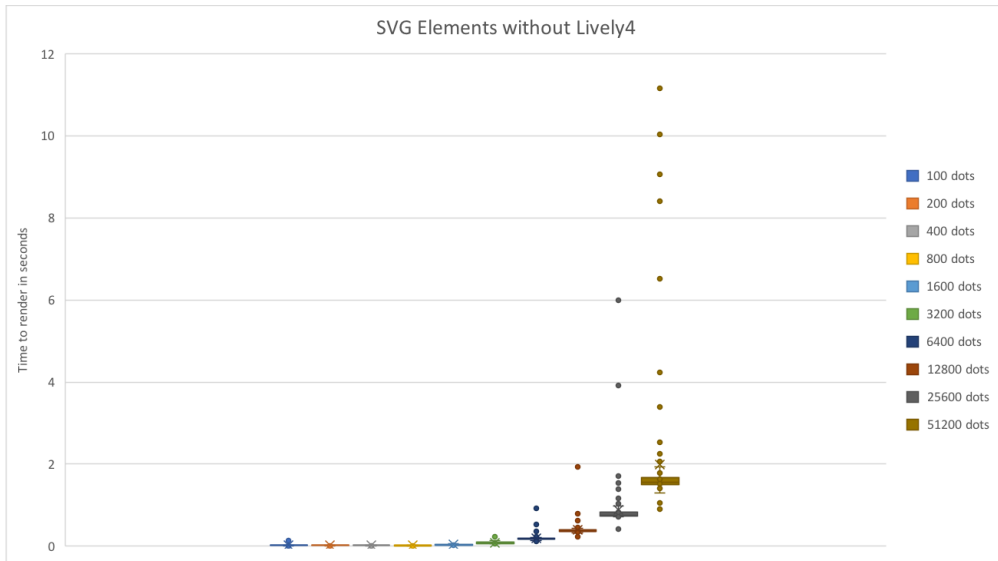


Figure C.15: Box plot for rendering points with SVG on local server

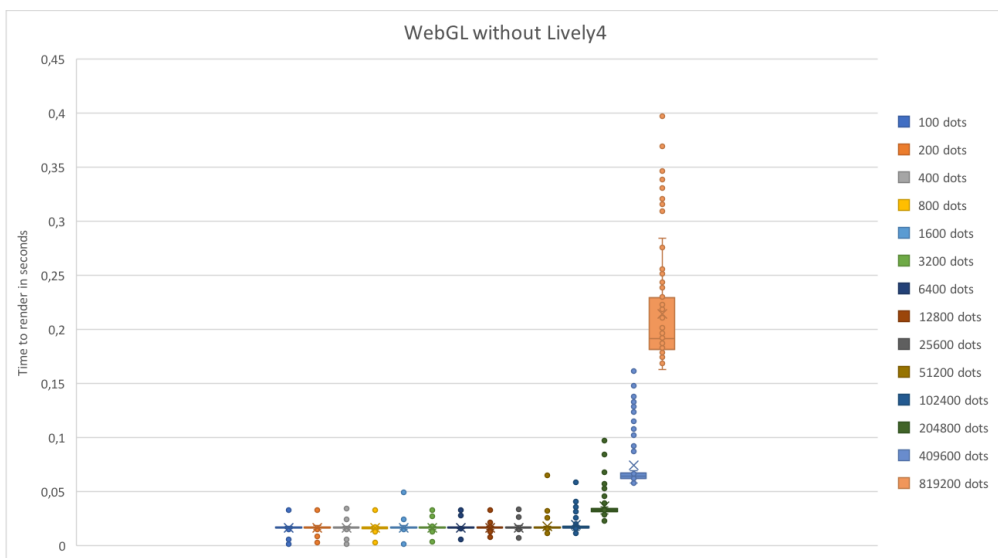


Figure C.16: Box plot for rendering points with WebGL on local server

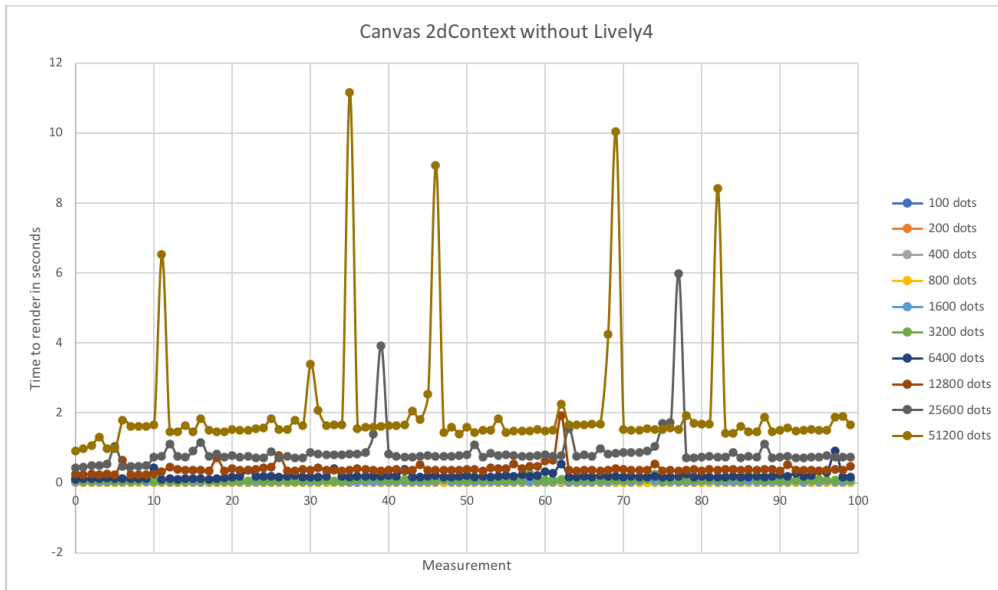


Figure C.17: Time to render when rendering points with canvas 2dContext on local server

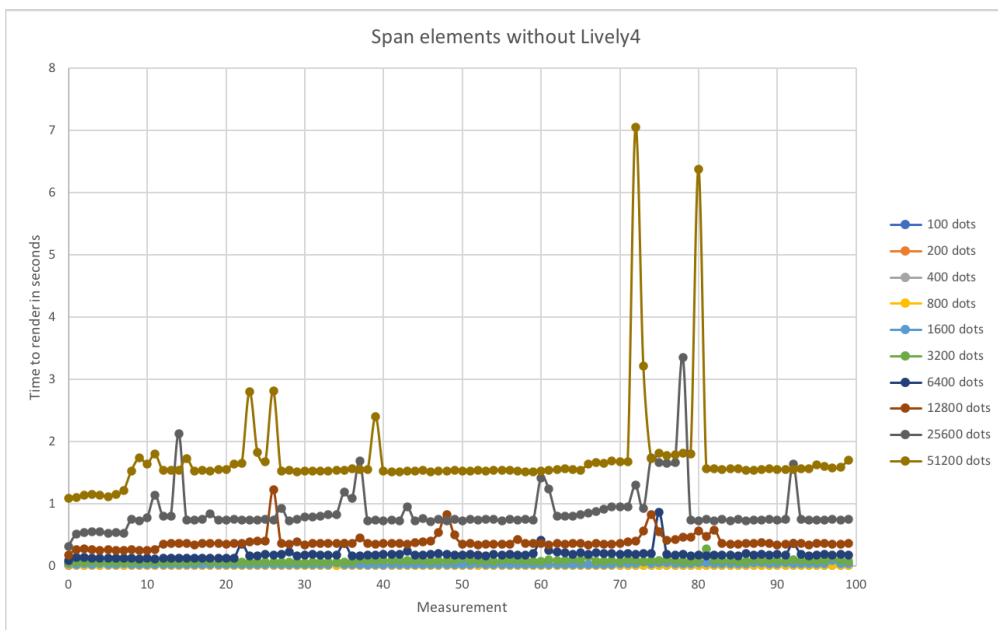


Figure C.18: Time to render when rendering points with <span>-elements on local server

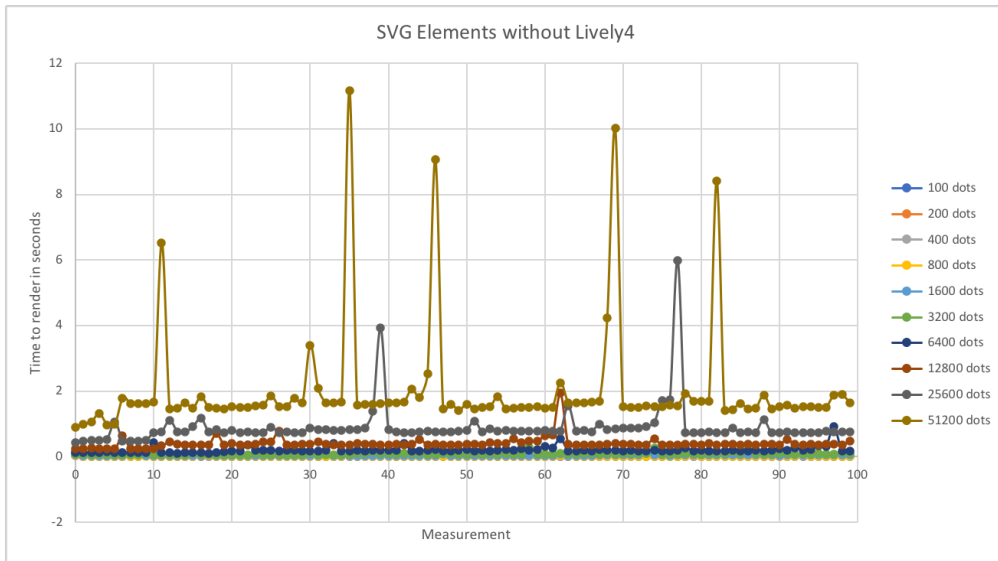


Figure C.19: Time to render when rendering points with SVG on local server

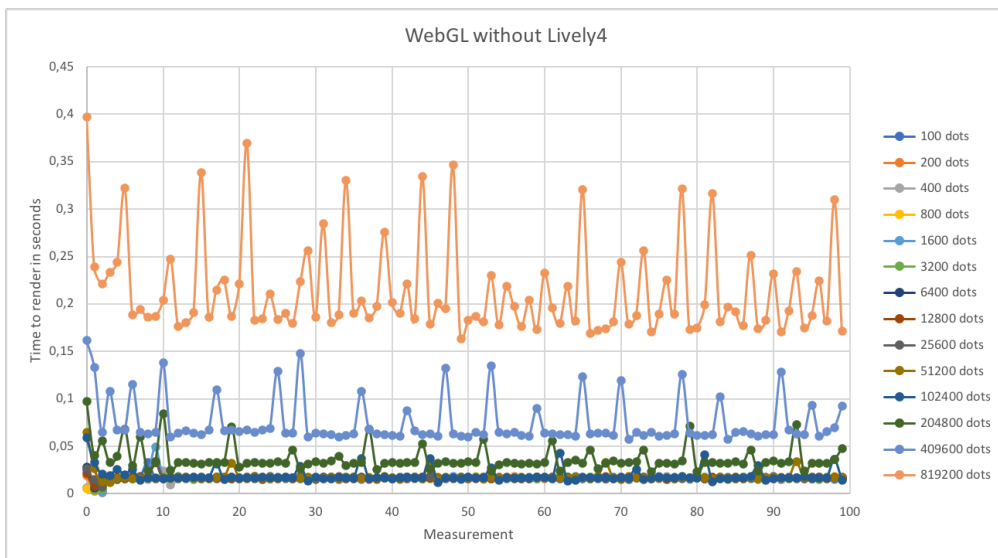


Figure C.20: Time to render when rendering points with WebGL on local server

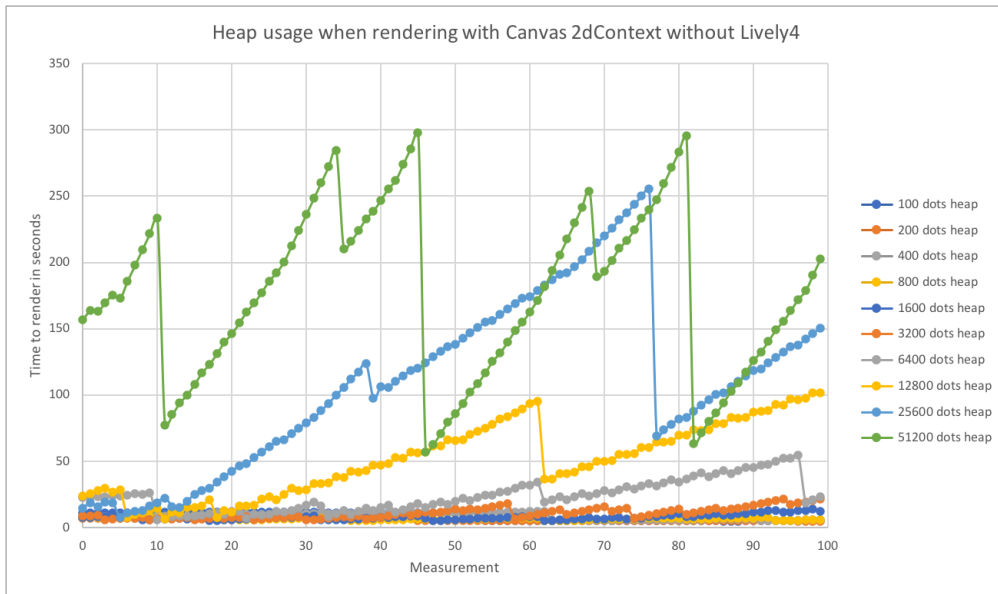


Figure C.21: Heap usage when rendering with canvas 2dContext on local server

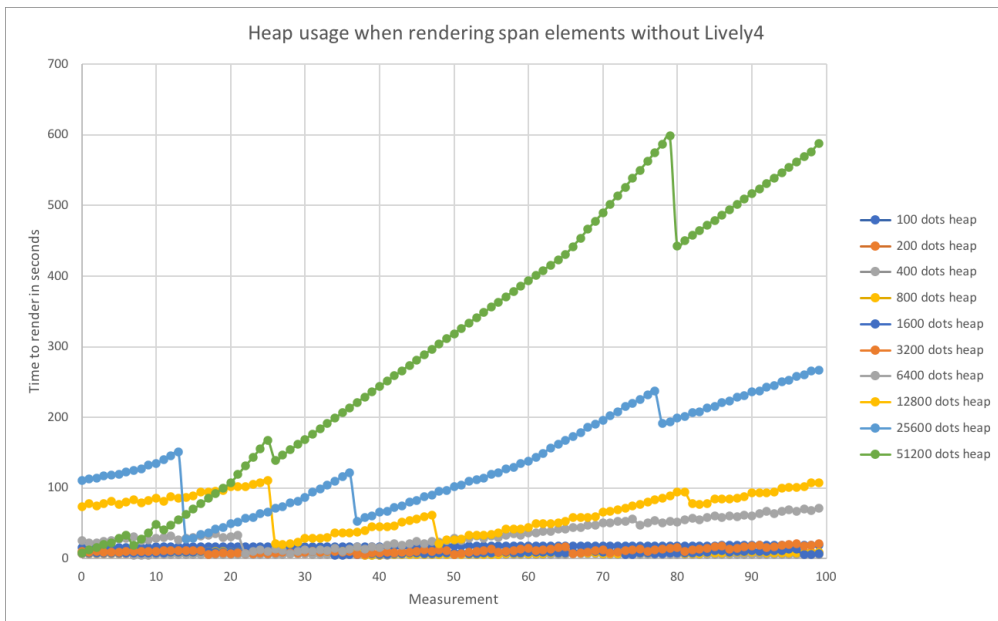


Figure C.22: Heap usage when rendering with <span>-elements on local server

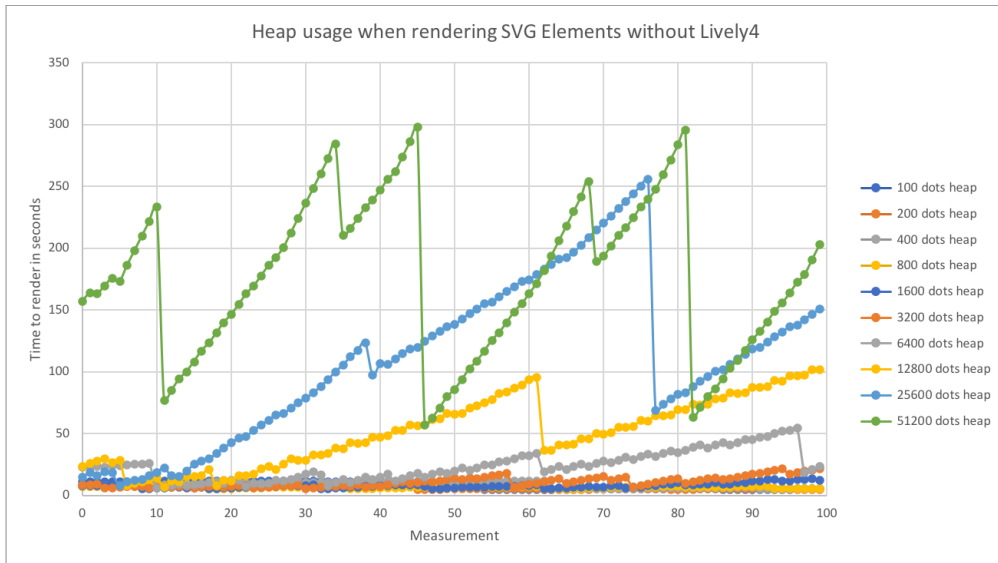


Figure C.23: Heap usage when rendering with SVG on local server

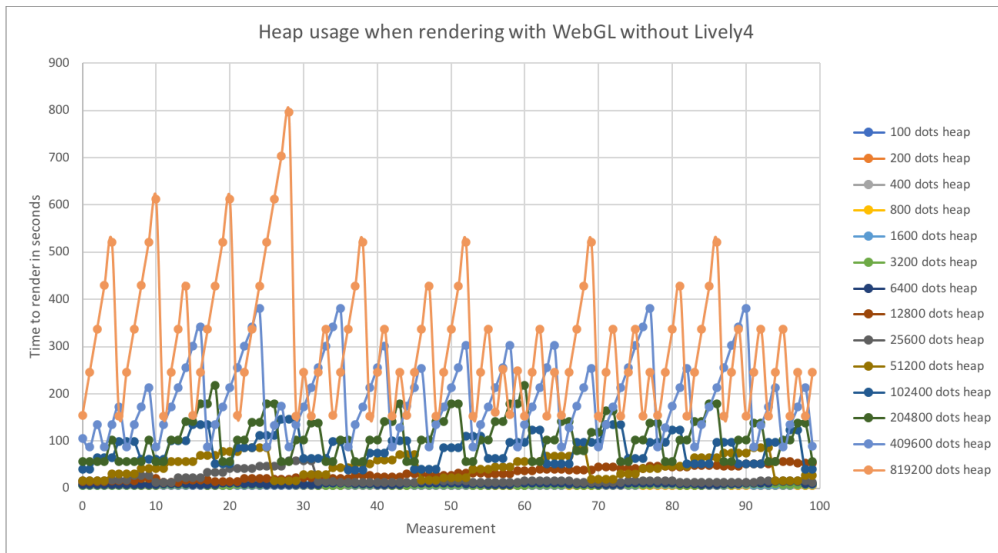


Figure C.24: Heap usage when rendering with WebGL on local server

### C.3 Tables

	webgl	canvas	span	svg
100 dots	1.644	1.850	1.669	1.836
200 dots	1.664	1.665	1.679	1.674
400 dots	1.646	1.671	1.678	1.709
800 dots	1.655	1.676	1.686	1.777
1600 dots	1.682	1.854	3.529	3.777
3200 dots	1.667	2.792	8.245	8.851
6400 dots	1.685	5.345	18.737	19.983
12800 dots	1.678	14.566	38.436	40.024
25600 dots	1.684	34.47	87.251	88.805
51200 dots	1.784	80.296	170.252	199.567

**Table C.1:** Sum of time to render for 100 measurements in seconds of the technologies.

## C.4 Code

Listing C.1: Basic benchmark code

```
1 <script language="javascript" src="https://npmcdn.com/regl/dist/
  regl.js"></script>
2 <script language="javascript" src="https://cdnjs.cloudflare.com/
  ajax/libs/d3/5.16.0/d3.js"></script>
3
4 <div id="root">
5   <button id="start">Start test</button>>
6 </div>
7
8 <style>
9 .dot {
10   height: 6px;
11   width: 6px;
12   background-color: #eb4438;
13   border-radius: 50%;
14   border: 1px solid black;
15   position: absolute;
16 }
17 </style>
18
19 <script type="module">
20
21 import { SVGRenderer } from './modules/renderer.js';
22
23 let root = document.getElementById("root")
24 let button = document.getElementById("start")
25 button.addEventListener("click", start)
26
27 // Specify number of points
28 let data = d3.range(50000)
29
30 // Kepp track of measurements
31 let numberOfCycles = 100
32 let counter = 0
33
34 // Store performance metrics
35 let performances = []
36 let usedHeapSizes = []
37
38 // Choose rendering technology
39 let renderer = new SVGRenderer(root, data)
40 renderer.setUp()
41
42 // Clean up
43 performance.clearMeasures()
44 performance.clearMarks()
```



```

45
46 function cycle() {
47   performance.mark(counter)
48
49   if(counter > 0) {
50     let time = performance.measure(counter, counter-1, counter).
      duration / 1000.0
51     let usedHeap = performance.memory.usedJSHeapSize / 1000000.0
52     performances.push(time)
53     usedHeapSizes.push(usedHeap)
54     renderer.changePosition()
55   }
56
57   if(counter < numberOfCycles) {
58     renderer.clear()
59     renderer.render()
60     counter += 1
61     requestAnimationFrame(cycle)
62   }
63
64   else {
65     performance.mark("end_total")
66     performance.measure("total", "begin_total", "end_total")
67     performances.push(performance.getEntriesByName("total")[0].
      duration / 1000.0)
68     writeToFile()
69     return
70   }
71 }
72
73 function start() {
74   performance.mark("begin_total")
75   cycle()
76 }
77
78 function writeToFile() {
79   let performanceCsv = buildCSV(performances, "dots", data)
80   let performanceFilename = renderer.name() + "_" + data.length + "_
      _performance.csv"
81   //lively.files.saveFile("https://lively-kernel.org/lively4/
      BP2019RH1/benchmarks/reports/" + performanceFilename,
      performanceCsv)
82
83   let heapCsv = buildCSV(usedHeapSizes, "dots heap", data)
84   let heapFilename = renderer.name() + "_" + data.length + "_heap.
      csv"
85   //lively.files.saveFile("https://lively-kernel.org/lively4/
      BP2019RH1/benchmarks/reports/" + heapFilename, heapCsv)
86
87   root.innerHTML = ""
88

```

```

89  var downloadLink = document.createElement("a");
90  var blob = new Blob(["\u0000", performanceCsv]);
91  var url = URL.createObjectURL(blob);
92  downloadLink.href = url;
93  downloadLink.download = performanceFilename;
94  downloadLink.innerHTML = "Download time data"
95  document.body.appendChild(downloadLink);
96
97
98  var downloadLinkHeap = document.createElement("a");
99  var blobHeap = new Blob(["\u0000", heapCsv]);
100 var urlHeap = URL.createObjectURL(blobHeap);
101 downloadLinkHeap.href = urlHeap;
102 downloadLinkHeap.download = heapFilename;
103 downloadLinkHeap.innerHTML = "Download heap data"
104 document.body.appendChild(downloadLinkHeap);
105 }
106
107 function buildCSV(datapoints, name, data) {
108   let csvContent = 'index, ' + data.length + " " + name + '\r\n'
109
110   datapoints.forEach( (datapoint, i) => {
111     csvContent += i + "," + datapoint + "\r\n";
112   })
113
114   return csvContent
115 }
116
117 </script>

```

Listing C.2: Code for different render technologies

```

1  var fragShader = `
2  precision mediump float;
3  varying vec4 fragColor;
4  void main () {
5     float r = 0.0, delta = 0.0, alpha = 1.0;
6     vec2 cxy = 2.0 * gl_PointCoord - 1.0;
7     r = dot(cxy, cxy);
8     if (r > 1.0) {
9       discard;
10    }
11    gl_FragColor = fragColor * alpha;
12  }`
13
14 var vertShaderDraw = `
15 precision mediump float;
16 attribute vec2 position;
17 attribute float pointWidth;
18 attribute vec4 color;
19

```

```

20   varying vec4 fragColor;
21   uniform float stageWidth;
22   uniform float stageHeight;
23
24   // helper function to transform from pixel space to normalized
25   // device coordinates (NDC). In NDC (0,0) is the middle,
26   // (-1, 1) is the top left and (1, -1) is the bottom right.
27   // Stolen from Peter Beshai's great blog post:
28   // http://peterbeshai.com/beautifully-animate-points-with-webgl-
    and-regl.html
29   vec2 normalizeCoords(vec2 position) {
30       // read in the positions into x and y vars
31       float x = position[0];
32       float y = position[1];
33
34       return vec2(
35           2.0 * ((x / stageWidth) - 0.5),
36           // invert y to treat [0,0] as bottom left in pixel space
37           -(2.0 * ((y / stageHeight) - 0.5)));
38   }
39
40   void main () {
41       gl_PointSize = pointWidth;
42       gl_Position = vec4(normalizeCoords(position), 0, 1);
43       fragColor = color;
44   }`
45
46
47   class Renderer {
48       constructor(rootDiv, dataPoints) {
49           this.root = rootDiv
50           this.data = dataPoints
51       }
52
53       setUpDrawingInformation() {
54           this.data = this.data.map( () => {
55               let drawingInformation = {
56                   x: getRandomFloat(0,800),
57                   y: getRandomFloat(0,800)
58               }
59               return drawingInformation
60           })
61       }
62
63       changePosition() {
64           this.data = this.data.map( datapoint => {
65               datapoint.x += 1
66               datapoint.y += 1
67               return datapoint
68           })
69   }

```

```
70 }
71
72 export class Context2dCanvasRenderer extends Renderer {
73
74   name() {
75     return "2dContext"
76   }
77
78   setUp() {
79     this.canvas = document.createElement("CANVAS")
80     this.canvas.width = "800"
81     this.canvas.height = "800"
82     this.root.appendChild(this.canvas)
83     this.context = this.canvas.getContext("2d")
84
85     this.setUpDrawingInformation()
86   }
87
88   render() {
89     this.data.forEach( datapoint => {
90       this.drawCircle(datapoint)
91     })
92
93   }
94
95   clear() {
96     this.context.clearRect(0, 0, 800, 800)
97   }
98
99   drawCircle(datapoint) {
100     var radius = 3;
101     var centerX = datapoint.x
102     var centerY = datapoint.y
103
104     this.context.beginPath();
105     this.context.arc(centerX, centerY, radius, 0, 2 * Math.PI,
106       false);
107     this.context.fillStyle = '#eb4438';
108     this.context.fill();
109     this.context.lineWidth = 1;
110     this.context.strokeStyle = '#000000';
111     this.context.stroke();
112   }
113 }
114
115 export class SpanRenderer extends Renderer {
116   name() {
117     return "span"
118   }
119 }
```

```

120  setUp() {
121      this.drawingArea = document.createElement("div")
122      this.drawingArea.style.width = "800px"
123      this.drawingArea.style.height = "800px"
124      this.root.appendChild(this.drawingArea)
125
126      this.setUpDrawingInformation()
127  }
128
129  render() {
130      this.data.forEach( datapoint => {
131          this.drawCircle(datapoint)
132      })
133  }
134
135  clear() {
136      this.drawingArea.innerHTML = ""
137  }
138
139  drawCircle(datapoint) {
140      let newElement = document.createElement("SPAN")
141      newElement.className = "dot"
142      newElement.style.top = datapoint.x + "px"
143      newElement.style.left = datapoint.y + "px"
144      this.drawingArea.appendChild(newElement);
145  }
146 }
147
148 export class SVGRenderer extends Renderer {
149     name() {
150         return "svg"
151     }
152
153     setUp() {
154         this.drawingArea = document.createElementNS("http://www.w3.org
155             /2000/svg", 'svg')
156         this.drawingArea.setAttribute("width", "800")
157         this.drawingArea.setAttribute("height", "800")
158         this.root.appendChild(this.drawingArea)
159
160         this.setUpDrawingInformation()
161     }
162
163     render() {
164         this.data.forEach( datapoint => {
165             this.drawCircle(datapoint)
166         })
167     }
168
169     clear() {

```

```

170   this.drawingArea.innerHTML = ""
171   }
172
173   drawCircle(datapoint) {
174     var newElement = document.createElementNS("http://www.w3.org
175       /2000/svg", 'circle');
176     newElement.setAttribute("cx", datapoint.x);
177     newElement.setAttribute("cy", datapoint.y);
178     newElement.setAttribute("r", "3");
179     newElement.setAttribute("stroke-width", "1")
180     newElement.setAttribute("stroke", "black")
181     newElement.setAttribute("fill", "#eb4438")
182     this.drawingArea.appendChild(newElement);
183   }
184 }
185 export class WebGLRenderer extends Renderer {
186   name() {
187     return "webgl"
188   }
189
190   setUp() {
191     this.canvas = document.createElement("CANVAS");
192     this.canvas.width = "800"
193     this.canvas.height = "800"
194     this.root.appendChild(this.canvas)
195     this.context = this.canvas.getContext("webgl")
196
197     this.setUpDrawingInformation()
198     this.setUpRegl()
199   }
200
201   setUpRegl() {
202     this.regl = createREGL(this.context)
203     this.drawPointsShader = this.regl({
204
205     frag: fragShader,
206
207     vert: vertShaderDraw,
208
209     attributes: {
210       position: function(context, props) {
211         return props.points.map(function(point) {
212           return [point.x, point.y];
213         });
214       },
215       color: function(context, props) {
216         return props.points.map(function(point) {
217           return [255/255.0, 20/255.0, 20/255.0, 1];
218         });
219       },

```

```

220     pointWidth: function(context, props) {
221         return props.points.map(function(point) {
222             return 10;
223         });
224     }
225 },
226
227 uniforms: {
228     stageWidth: this.reggl.context("drawingBufferWidth"),
229     stageHeight: this.reggl.context("drawingBufferHeight"),
230 },
231
232 count: function(context, props) {
233     return props.points.length;
234 },
235 primitive: "points"
236 });
237 }
238
239 clear() {
240     this.reggl.clear({
241         color: [1, 1, 1, 1],
242         depth: 1,
243         stencil: 0
244     })
245 }
246
247 render() {
248     this.drawPointsShader({
249         points: this.data,
250     })
251 }
252
253 }
254
255 function getRandomFloat(min, max) {
256     return Math.floor(Math.random() * (max - min) + min);
257 }

```





# Bibliography

- [1] R. A. Amar, J. Eagan, and J. T. Stasko. “Low-Level Components of Analytic Activity in Information Visualization”. In: *InfoVis 2005: Symposium on Information Visualization*. IEEE, 2005.
- [2] K. Andrews. “Evaluating information visualisations”. In: *BELIV 2006: Workshop on BEyond time and errors: novel evaluation methods for information visualization*. ACM, 2006.
- [3] F. Asplund, M. Biehl, J. El-Khoury, and M. Törngren. “Tool Integration beyond Wasserman”. In: *Lecture Notes in Business Information Processing* (2011).
- [4] F. Asplund and M. Törngren. “The Discourse on Tool Integration Beyond Technology, a Literature Survey”. In: *Journal of Systems and Software* 106 (2015).
- [5] R. Battagline. *Hands-On Game Development with WebAssembly: Learn WebAssembly C++ programming by building a retro space game*. Packt Publishing, 2019.
- [6] U. Bhardwaj. *How Web Browsers Work — Behind the scene Architecture, Technologies, and Internal Working*. 2019. URL: <https://medium.com/web-god-mode/how-web-browsers-work-behind-the-scene-architecture-technologies-and-internal-working-fec601488bfa> (visited on 2020-07-28).
- [7] E. Bidelman. *Automated testing with Headless Chrome | Web*. 2019. URL: <https://developers.google.com/web/updates/2017/06/headless-karma-mocha-chai> (visited on 2020-07-29).
- [8] E. Bidelman. *The Basics of Web Workers - HTML5 Rocks*. 2010. URL: <https://www.html5rocks.com/en/tutorials/workers/basics/> (visited on 2020-07-25).
- [9] E. Bonabeau, J.-L. Dessalles, and A. Grumbach. “Characterizing emergent phenomena (1): A critical review”. In: *Revue internationale de systemique* 9.3 (1995).
- [10] M. Bostock, V. Ogievetsky, and J. Heer. “D<sup>3</sup> data-driven documents”. In: *Transactions on Visualization and Computer Graphics* 17.12 (2011).
- [11] A. Bouchebra, T. Jankov, H. James, and Z. Antolovic. *Performance Tools*. SitePoint, 2018.
- [12] P. Buneman, A. Chapman, and J. Cheney. “Provenance management in curated databases”. In: *SIGMOD 2006: International Conference on Management of Data*. ACM, 2006.

- [13] P. Buneman, S. Khanna, and W. C. Tan. "Data Provenance: Some Basic Issues". In: *FST TCS 2000: Conference on Foundations of Software Technology and Theoretical Computer Science*. Volume 1974. Lecture Notes in Computer Science. Springer, 2000.
- [14] A. Cairo. *The functional art: an introduction to information graphics and visualization*. New Riders, 2013.
- [15] R. Chang, C. Ziemkiewicz, T. M. Green, and W. Ribarsky. "Defining Insight for Visual Analytics". In: *IEEE Computer Graphics and Applications* 29.2 (2009).
- [16] K. Charmaz. "The Search for Meanings-Grounded Theory". In: *Rethinking Methods in Psychology*. Sage Publications, 1996.
- [17] M. Chen and R. J. Norman. "A Framework for Integrated CASE". In: *IEEE Software* 9.2 (1992).
- [18] D. Clément. "A Distributed Architecture for Programming Environments". In: *SDE 4: Symposium on Software Development Environments*. Irvine, California, USA: ACM, 1990.
- [19] Y. Cui and J. Widom. "Lineage Tracing for General Data Warehouse Transformations". In: *VLDB J.* 12.1 (2003).
- [20] W. Cunningham. *Wiki Design Principles*. Wiki. 2014. URL: <https://wiki.c2.com/?WikiDesignPrinciples> (visited on 2020-07-29).
- [21] R. Davidson and D. Harel. "Drawing Graphs Nicely Using Simulated Annealing". In: *ACM Trans. Graph.* 15.4 (1996).
- [22] H. P. I. of Design at Stanford. *An Introduction to Design Thinking PROCESS GUIDE*. 2010.
- [23] P. Deutsch. "DEFLATE Compressed Data Format Specification version 1.3". In: *RFC* 1951 (1996).
- [24] P. Eades. "A Heuristic for Graph Drawing". In: *Congressus Numerantium* 42 (1984).
- [25] A. Emamdjomeh. *Animating Lots and Lots of Circles with WebGL and REGL.js*. 2019. URL: <https://web.archive.org/web/20200730064918/https://observablehq.com/@emamd/animating-lots-and-lots-of-circles-with-regl-js> (visited on 2020-07-29).
- [26] F. U. Falk and W. Brenner. *Design Thinking: The Handbook*. World Scientific, 2020.
- [27] N. J. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. "Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-update Problem". In: *Transactions on Programming Languages and Systems* 29.3 (2007).
- [28] T. M. J. Fruchterman and E. M. Reingold. "Graph Drawing by Force-directed Placement". In: *Software Practice and Experience* 21.11 (1991).

- [29] T. Garsiel. *How Browsers Work*. 2020. URL: [http://taligarsiel.com/Projects/howbrowserswork1.htm#Rendering\\_engines](http://taligarsiel.com/Projects/howbrowserswork1.htm#Rendering_engines) (visited on 2020-07-28).
- [30] T. Garsiel and P. Irish. *How Browsers Work: Behind the Scenes of Modern Web Browsers - HTML5 Rocks*. 2011. URL: <https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/> (visited on 2020-07-28).
- [31] E. Gasperowicz. *OffscreenCanvas — Speed up Your Canvas Operations with a Web Worker*. 2019. URL: <https://developers.google.com/web/updates/2018/08/offscreen-canvas> (visited on 2020-07-25).
- [32] I. Grigorik. *Render-tree Construction, Layout, and Paint | Web Fundamentals*. 2019. URL: <https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction> (visited on 2020-07-27).
- [33] G. Grinstein and M. Trutschl. “High-Dimensional Visualizations”. In: *Proceedings of the Visual Data Mining Workshop, KDD*. 2001.
- [34] A. Grosskurth and M. W. Godfrey. “A Reference Architecture for Web Browsers”. In: *ICSM’05: International Conference on Software Maintenance*. 2005.
- [35] J. Gruber. *Daring Fireball: Markdown*. Blog. 2004. URL: <https://daringfireball.net/projects/markdown/> (visited on 2020-07-29).
- [36] P. Hanrahan. “Vizql: a language for query, analysis and visualization”. In: *SIGMOD 2006: International Conference on Management of Data*. ACM, 2006.
- [37] R. Ikeda and J. Widom. *Data Lineage: A Survey*. Technical Report. Stanford University, 2009.
- [38] D. Ingalls, T. Felgentreff, R. Hirschfeld, R. Krahn, J. Lincke, M. Röder, A. Taivalaari, and T. Mikkonen. “A world of active objects for work and play: the first ten years of lively”. In: *Onward! 2016: International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. ACM, 2016.
- [39] P. Irish. *requestAnimationFrame for Smart Animating*. 2011. URL: <https://www.paulirish.com/2011/requestanimationframe-for-smart-animating/> (visited on 2020-07-28).
- [40] P. Isenberg, T. Zuk, C. Collins, and S. Carpendale. “Grounded evaluation of information visualizations”. In: *BELIV 2008: BEyond time and errors: novel evaluation methods for Information Visualization*. Edited by E. Bertini, A. Perer, C. Plaisant, and G. Santucci. ACM, 2008.
- [41] G. I. Johnson, C. W. Clegg, and S. J. Ravden. “Towards a Practical Method of User Interface Evaluation”. In: *Applied Ergonomics* 20.4 (1989).
- [42] D. A. Keim. “Visual Techniques for Exploring Databases”. In: *KDD’97: Knowledge Discovery in Databases*. 1997.
- [43] D. A. Keim and H. Kriegel. “Visualization Techniques for Mining Large Databases: A Comparison”. In: *Transactions on Knowledge and Data Engineering* 8.6 (1996).

- [44] R. Krahn, D. Ingalls, R. Hirschfeld, J. Lincke, and K. Palacz. "Lively Wiki A Development Environment for Creating and Sharing Active Web Content". In: *WikiSym '09: International Symposium on Wikis and Open Collaboration*. ACM, 2009.
- [45] M. Lewrick, P. Link, and L. Leifer. *The Design Thinking Playbook: Mindful Digital Transformation of Teams, Products, Services, Businesses and Ecosystems*. John Wiley & Sons, 2018.
- [46] J. Lincke. "Evolving Tools in a Collaborative Self-supporting Development Environment". PhD thesis. Universität Potsdam, 2014.
- [47] J. Lincke, S. Ramson, P. Rein, R. Hirschfeld, M. Taeumel, and T. Felgentreff. "Designing a Live Development Experience for Web Components". In: *PX/17.2: Programming Experience Workshop, co-located with OOPSLA*. ACM, 2017.
- [48] A. E. Lunzer. "Reconnaissance: A Widely Applicable Approach Encouraging Well-informed Choices in Computer-based Tasks". PhD thesis. University of Glasgow, UK, 1995.
- [49] L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008).
- [50] J. H. Maloney and R. B. Smith. "Directness and Liveness in the Morphic user Interface Construction Environment". In: *UIST '95: Symposium on User Interface and Software Technology*. ACM, 1995.
- [51] J. McCutchan and L. Lee. *Effectively Managing Memory at Gmail scale - HTML5 Rocks*. 2013. URL: <https://www.html5rocks.com/en/tutorials/memory/effectivemanagement/> (visited on 2020-07-29).
- [52] J. Nielsen. *Response Time Limits*. 1993. URL: <https://www.nngroup.com/articles/response-times-3-important-limits/> (visited on 2020-07-28).
- [53] H. Ossher, W. Harrison, and P. Tarr. "Software Engineering Tools and Environments: A Roadmap". In: *ICSE '00: Conference on The Future of Software Engineering*. ACM, 2000.
- [54] T. Parisi. *WebGL: Up and Running*. O'Reilly, 2012.
- [55] H. Payer and R. McIlroy. *Getting Garbage Collection for Free · V8*. 2015. URL: <https://v8.dev/blog/free-garbage-collection> (visited on 2020-07-29).
- [56] C. Price. *Rendering charts with OffscreenCanvas*. 2020. URL: <https://blog.scottlogic.com/2020/03/19/offscreen-canvas.html> (visited on 2020-07-25).
- [57] Z. C. Qin Chengzhi and P. Tao. "Taxonomy of Visualization Techniques and Systems – Concerns between Users and Developers are Different". In: *Asia GIS Conference*. Volume 35. 2003.
- [58] S. P. Reiss. "Software Tools and Environments". In: *Computing Surveys (CSUR)* 28.1 (1996).

- [59] M. Satran. *Retained Mode Versus Immediate Mode - Win32 apps*. 2018. URL: <https://docs.microsoft.com/en-us/windows/win32/learnwin32/retained-mode-versus-immediate-mode> (visited on 2020-07-27).
- [60] B. Shneiderman. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations". In: *SVL 1996: Symposium on Visual Languages*. IEEE, 1996.
- [61] B. Smus. *Improving HTML5 Canvases Performance - HTML5 Rocks*. 2013. URL: <https://www.html5rocks.com/en/tutorials/canvas/performance/> (visited on 2020-07-29).
- [62] J. T. Stasko. "Value-driven evaluation of visualizations". In: *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization, BELIV 2014, Paris, France, November 10, 2014*. Edited by H. Lam, P. Isenberg, T. Isenberg, and M. Sedlmair. ACM, 2014.
- [63] M. Stickdorn, J. Schneider, K. Andrews, and A. Lawrence. *This Is Service Design Thinking: Basics, Tools, Cases*. Volume 1. Wiley Hoboken, NJ, 2011.
- [64] C. Stolte, D. Tang, and P. Hanrahan. "Polaris: A system for query, analysis, and visualization of multidimensional relational databases". In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (2002).
- [65] M. Tory and T. Möller. "Rethinking Visualization: A High-Level Taxonomy". In: *InfoVis 2004: 10th Symposium on Information Visualization*. Edited by M. O. Ward and T. Munzner. IEEE, 2004.
- [66] L. Tweedie. "Characterizing Interactive Externalizations". In: *CHI '97: Human Factors in Computing Systems*. Edited by S. Pemberton. ACM, 1997.
- [67] A. I. Wasserman. "Tool Integration in Software Engineering Environments". In: *International Workshop on Environments on Software Engineering Environments*. Springer, 1990.
- [68] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, 2016.
- [69] B. Wilson. *How Chromium Displays Web Pages - The Chromium Projects*. 2012. URL: <https://www.chromium.org/developers/design-documents/displaying-a-web-page-in-chrome> (visited on 2020-07-28).
- [70] S. K. M. Wong, W. Ziarko, and P. C. N. Wong. "Generalized vector spaces model in information retrieval". In: *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '85*. ACM, 1985.
- [71] L. Wyse and S. Subramanian. "The Viability of the Web Browser as a Computer Music Platform". In: *Computer Music Journal* 37.4 (2013).
- [72] J. S. Yi. "Implications of Individual Differences on Evaluating Information Visualization Techniques". In: *International Journal of Human-Computer Studies* 45.6 (2012).

## *Bibliography*

- [73] J. S. Yi, Y. ah Kang, J. T. Stasko, and J. A. Jacko. "Understanding and Characterizing Insights: How Do People Gain Insights Using Information Visualization?" In: *BELIV 2008: BEyond time and errors: novel evaLuation methods for Information Visualization*. Edited by E. Bertini, A. Perer, C. Plaisant, and G. Santucci. ACM, 2008.
- [74] C. Ziemkiewicz and R. Kosara. "The Shaping of Information by Visual Metaphors". In: *Transactions on Visualization and Computer Graphics* 14.6 (2008).

# List of Figures

1.1	Context of Africa’s Voices’ work . . . . .	2
1.2	A variety of visualizations used in the project ‘Somali Views In The Early Days Of Covid-19: A Rapid Diagnostic’ . . . . .	4
1.3	An exemplary bar chart displaying the amount of different themes mentioned by males (blue) and females (red). . . . .	5
1.4	An exemplary bubble chart showing the frequency of themes. . . . .	6
1.5	A Tableau Worksheet . . . . .	18
1.6	The resulting ggplot2 scatterplot . . . . .	20
1.7	A D3 bubble chart with tooltip on hover (A) . . . . .	22
2.1	The diverging and converging process in Design Thinking [45] . . . . .	29
2.2	The microcycle in Design Thinking [45] . . . . .	30
2.3	The persona named Tom . . . . .	33
2.4	Fusing two pie charts into a stacked line chart (41) . . . . .	34
2.5	Extract of the storyboard for the idea of fusing diagrams (41) . . . . .	35
2.6	Theme Centers (33). Placing individuals as points around and between their themes . . . . .	39
2.7	Individual Centered (7). Placing individuals as points around one individual. Their distance is based on the difference of their attribute values regarding a certain attribute . . . . .	40
2.8	Forces (31). Attracting individuals as points to their corresponding values . . . . .	41
2.9	Map (3). Placing individuals as points on a map using the geographic positions of their districts . . . . .	42
2.10	XY-Axis (22). Grouping individuals as points into predefined areas. . . . .	43
2.11	Individual Forces (76). Attracting individuals as points to each other depending on their similarity . . . . .	44
2.12	Venn (39). Individuals as points in a Venn diagram . . . . .	45
2.13	Group Chaining (75). Chaining groupings of individuals as points according to selected attributes . . . . .	46
2.14	Panes (45). Windows containing individuals as points. Connections between windows make the exploration flow visible . . . . .	47
2.15	Tab View (77). Tabs containing one visualization type each. Inspector to inspect individuals. Control panel to color, filter, and select individuals in the visualizations . . . . .	48
2.16	Individuals on Canvas (46). Individuals as points that can be grouped, selected, and filtered. Hovering displays information regarding the individual or a group of individuals. . . . .	49

List of Figures

2.17	Statement Generator (63). Generating statements using code formulas and templates . . . . .	50
2.18	Highlighting points across diagrams (44) . . . . .	54
2.19	Highlighting bars across diagrams (25) . . . . .	54
2.20	Graph structure displaying relationships between messages and individuals (5) . . . . .	55
2.21	Highlighting points across diagrams (55) . . . . .	56
2.22	Ideas manipulating the view of data . . . . .	58
2.23	Highlighting bars across diagrams (27) . . . . .	59
2.24	Highlighting points across diagrams (34) . . . . .	60
2.25	Map with time slider to redo or undo filter or selection steps (4) . . .	61
2.26	Displaying missing data in scatterplots (65) . . . . .	63
3.1	Individuals displayed as points . . . . .	67
3.2	Individuals displayed as points with fill versus only strokes . . . . .	69
3.3	Inspecting an individual on click . . . . .	70
3.4	Coloring individuals according to gender . . . . .	71
3.5	Using filter-out logic to remove individuals without expected values from display in the <i>Filter Chain</i> prototype . . . . .	72
3.6	Using filter-in logic in the <i>Filter Widget</i> prototype to only work with individuals up to 35 years . . . . .	72
3.7	Using a <i>Freehand Selection</i> to filter selected individuals into the next created visualization . . . . .	74
3.8	Highlighting the distribution of middle-aged respondents in groups defined by language and gender . . . . .	75
3.9	Chaining grouping for gender and county . . . . .	76
3.10	Hovering over the Mogadishu district while using the <i>Map</i> prototype . . . . .	77
3.11	Statistical calculations about the distribution of people talking about good governance in Mogadishu . . . . .	78
3.12	Using the <i>XY Diagram</i> to display groups by gender and age . . . . .	79
3.13	Using the <i>XY Diagram</i> to create a bar chart which maintains a connection to each individual . . . . .	80
3.14	Using the <i>Venn Diagram</i> prototype to explore connections between two theme groups . . . . .	80
3.15	Using the <i>Individual Center</i> prototype with thematic comparison to compare and explore two individuals with equal theme tags on gender, age category and household language . . . . .	82
3.16	Using the <i>Individual Center</i> prototype with demographic comparison to explore two individuals with equal values for gender, household language, region and age category . . . . .	82
3.17	Using the <i>Movement</i> prototype with a very high fade factor . . . . .	84
3.18	Using the <i>Movement</i> prototype with a low fade factor . . . . .	84



3.19	Using the <i>Individual Forces</i> prototype. Forces are calculated according to the theme categories each individual belongs in (question, answer, escalate, and missing). Points are colored according to their theme category. . . . .	87
3.20	Using the t-SNE algorithm based on demographic data. Points are colored according to county. . . . .	87
3.21	Using the <i>Statistics Panel</i> prototype for displaying predefined statistics for a selected group of individuals . . . . .	89
3.22	Using the <i>Panes</i> prototype to explore the data using grouping, colouring and filtering . . . . .	90
3.23	Using the <i>XY Diagram</i> in the <i>Tab View</i> prototype with coloring according to age . . . . .	93
3.24	Using the <i>Map</i> prototype in the <i>Tab View</i> prototype with coloring according to age . . . . .	93
3.25	Using the <i>Venn Diagram</i> in the <i>Tab View</i> prototype with coloring according to age . . . . .	94
3.26	Basic components of the <i>Tree View</i> prototype . . . . .	96
3.27	Creating a new visualization pane in the <i>Tree View</i> prototype . . . . .	96
3.28	Using the <i>Freehand Selection</i> for selecting individuals to display in the newly created pane . . . . .	97
4.1	Overview of Lively4 client application and Lively4 server architecture	105
4.2	URL of Lively4 client application . . . . .	106
4.3	Lively4 world loaded within the browser with a Lively4 browser, a workspace, github sync tool, and another Lively4 tool launched . . .	107
4.4	Lively4 browser with index.md loaded in *edit* mode (background); Lively4 browser with index.md loaded in *view* mode (foreground)	108
4.5	Lively4 client application loaded with two Lively4 browser windows launched. Browser windows both access different files from the Lively4 server. . . . .	109
4.6	Rendered markdown with opened edit iFrame for the draw.io figure	111
4.7	Rendered markdown with executed scripts and native HTML elements embedded . . . . .	112
4.8	Rendered markdown with codeblock and chart on canvas . . . . .	114
4.9	Shadow DOM in relation to the document DOM . . . . .	118
4.10	Lively4 browser web component architecture visible in the DOM . .	122
4.11	UI of the <i>Tab View</i> prototype divided according to the web component structure . . . . .	125
4.12	Workflow for dynamically creating new web component instances in the UI of the <i>Tree View</i> prototype . . . . .	127
4.13	The <i>Tree View</i> prototype markdown view before and after reloading the Lively4 browser . . . . .	129
4.14	Lively4 wiki workflow in one Lively4 session . . . . .	131
4.15	Automatic reloading of Lively4 browser windows that show the same URL . . . . .	132

## List of Figures

4.16	Forbidden query for a file on the private voices server from within a session started from the public Lively4 server . . . . .	133
4.17	The Lively4 one checkout and multiple checkout collaboration . . . . .	134
4.18	Conflicting collaboration of two Lively4 users at realtime . . . . .	135
4.19	Comparing versions of 'index.md' in the Lively4 client application . . . . .	136
4.20	GitHub sync tool before, while and after syncing the BP2019RH1 repository with the remote origin . . . . .	137
5.1	An example of mapping as bipartite graphs with the restrictions of the appropriate relation type . . . . .	146
5.2	An example bar chart of the age distribution in the age buckets [ $<10$ , 10-14, 15-17, 18-35, 36-54, over 55, missing] . . . . .	148
5.3	An example of building a tracing graph. The upper half displays the operations' single mappings. The lower half displays the concatenated mapping graph. . . . .	149
5.4	Resulting bar chart (A) displayed with the diagram structure (B) and the stored data per bar (C) . . . . .	150
5.5	The double rendering process . . . . .	151
5.6	Two graphical elements overlapping: Once with 50% opacity (left), once with 100% opacity (right) . . . . .	152
5.7	The position matching process . . . . .	155
5.8	The XY Diagram separated in its four parts . . . . .	158
5.9	Screenshots of the three main canvases of a map . . . . .	160
5.10	Screenshot of the Venn prototype with two theme groups . . . . .	161
5.11	Converting a Venn diagram to a graph . . . . .	162
5.12	A screenshot of one bar chart of a statistics panel . . . . .	164
5.13	Screenshot of a not connected child visualization of a statistics panel . . . . .	165
5.14	A screenshot of the XY Diagram, with the gender on the x-axis and amount on the y-axis . . . . .	167
6.1	Basic browser architecture . . . . .	174
6.2	WebKit rendering process . . . . .	176
6.3	Immediate mode and Retained mode . . . . .	178
6.4	Google Chrome DevTools overview . . . . .	180
6.5	A rendered dot . . . . .	183
6.6	Chrome DevTools when rendering with WebGL . . . . .	189
6.7	Chrome DevTools when rendering $\langle \text{span} \rangle$ -elements . . . . .	190
6.8	Heap usage and time to render when rendering point with SVG . . . . .	191
6.9	Rendering with SVG without clean up of drawing area. Performed on a local server without Lively4 . . . . .	192
6.10	Percentage differences in time to render in Lively4 and on local server . . . . .	193
6.11	Comparison of rendering with WebGL in Lively4 and on local server . . . . .	194
6.12	Comparison of the impact of the drawing area size on performance . . . . .	195
6.13	Comparison of the measurement series from different days using Lively4 . . . . .	196

6.14	Comparison of the measurement series from different days using local server . . . . .	197
6.15	Rendering technologies and requirements . . . . .	198
7.1	Dataset grouped by gender . . . . .	209
7.2	Dataset grouped by gender and age . . . . .	209
7.3	Map, colored by recently displaced . . . . .	210
7.4	Venn Diagram, groupings for themes ‘call for right practice’ and ‘religious practice’ . . . . .	211
7.5	Dataset grouped and colored by recently displaced . . . . .	211
7.6	Age distributions of recently displaced (left) and not recently displaced (right) respondents . . . . .	213
7.7	Different kinds of dimensionality expansions. Left visualizations include recently displaced, right visualizations, not recently displaced people. Colored by household language, neon green is Somali. . . . .	213
7.8	Left visualizations include recently displaced, right visualizations not recently displaced people. Colored by household language, neon green is Somali. Bottom two visualizations correlate zone by gender. . . . .	214
7.9	Age distributions of recently displaced (left) and not recently displaced (right) respondents and map colored by household language (middle), Somali is neon green, Maimai is black. . . . .	215
7.10	Venn Diagrams (bottom two) from selection in age distribution . . . . .	215
7.11	Exemplary graph after exploration . . . . .	216
7.12	Individuals grouped in circles according to their demographic similarity to a chosen individual . . . . .	217
7.13	Individuals grouped in circles according to their thematic similarity to a chosen individual . . . . .	218
7.14	Main feedback cycles involving creation of visualizations . . . . .	224
7.15	Interplay of design and user model . . . . .	227
C.1	Box plot for rendering points on canvas 2dContext in Lively4 . . . . .	289
C.2	Box plot for rendering points with HTML <code>&lt;span&gt;</code> -elements in Lively4 . . . . .	289
C.3	Box plot for rendering points with SVG in Lively4 . . . . .	290
C.4	Box plot for rendering points on canvas with WebGL in Lively4 . . . . .	290
C.5	Time to render when rendering points with canvas 2dContext in Lively4 . . . . .	291
C.6	Time to render when rendering points with <code>&lt;span&gt;</code> -elements in Lively4 . . . . .	291
C.7	Time to render when rendering points with SVG in Lively4 . . . . .	292
C.8	Time to render when rendering points with WebGL in Lively4 . . . . .	292
C.9	Heap usage when rendering with canvas 2dContext in Lively4 . . . . .	293
C.10	Heap usage when rendering points with <code>&lt;span&gt;</code> -elements in Lively4 . . . . .	293
C.11	Heap usage when rendering with SVG in Lively4 . . . . .	294
C.12	Heap usage when rendering with WebGL in Lively4 . . . . .	294
C.13	Box plot for rendering points on canvas 2dContext on local server . . . . .	295
C.14	Box plot for rendering points with <code>&lt;span&gt;</code> -elements on local server . . . . .	295
C.15	Box plot for rendering points with SVG on local server . . . . .	296

*List of Figures*

C.16	Box plot for rendering points with WebGL on local server . . . . .	296
C.17	Time to render when rendering points with canvas 2dContext on local server . . . . .	297
C.18	Time to render when rendering points with <span>-elements on local server . . . . .	297
C.19	Time to render when rendering points with SVG on local server . . . . .	298
C.20	Time to render when rendering points with WebGL on local server . . . . .	298
C.21	Heap usage when rendering with canvas 2dContext on local server . . . . .	299
C.22	Heap usage when rendering with <span>-elements on local server . . . . .	299
C.23	Heap usage when rendering with SVG on local server . . . . .	300
C.24	Heap usage when rendering with WebGL on local server . . . . .	300

# Aktuelle Technische Berichte des Hasso-Plattner-Instituts

<b>Band</b>	<b>ISBN</b>	<b>Titel</b>	<b>Autoren / Redaktion</b>
135	978-3-86956-503-3	<b>Fast packrat parsing in a live programming environment : improving left-recursion in parsing expression grammars</b>	Friedrich Schöne, Patrick Rein, Robert Hirschfeld
134	978-3-86956-502-6	<b>Interval probabilistic timed graph transformation systems</b>	Maria Maximova, Sven Schneider, Holger Giese
133	978-3-86956-501-9	<b>Compositional analysis of probabilistic timed graph transformation systems</b>	Maria Maximova, Sven Schneider, Holger Giese
132	978-3-86956-482-1	<b>SandBlocks : Integration visueller und textueller Programmelemente in Live-Programmiersysteme</b>	Leon Bein, Tom Braun, Björn Daase, Elina Emsbach, Leon Matthes, Maximilian Stiede, Marcel Taeumel, Toni Mattis, Stefan Ramson, Patrick Rein, Robert Hirschfeld, Jens Mönig
131	978-3-86956-481-4	<b>Was macht das Hasso-Plattner-Institut für Digital Engineering zu einer Besonderheit?</b>	August-Wilhelm Scheer
130	978-3-86956-475-3	<b>HPI Future SOC Lab : Proceedings 2017</b>	Christoph Meinel, Andreas Polze, Karsten Beins, Rolf Strotmann, Ulrich Seibold, Kurt Rödszus, Jürgen Müller
129	978-3-86956-465-4	<b>Technical report : Fall Retreat 2018</b>	Christoph Meinel, Hasso Plattner, Jürgen Döllner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann, Holger Giese, Patrick Baudisch, Tobias Friedrich, Erwin Böttinger, Christoph Lippert
128	978-3-86956-464-7	<b>The font engineering platform : collaborative font creation in a self-supporting programming environment</b>	Tom Beckmann, Justus Hildebrand, Corinna Jaschek, Eva Krebs, Alexander Löser, Marcel Taeumel, Tobias Pape, Lasse Fister, Robert Hirschfeld
127	978-3-86956-463-0	<b>Metric temporal graph logic over typed attributed graphs : extended version</b>	Holger Giese, Maria Maximova, Lucas Sakizloglou, Sven Schneider
126	978-3-86956-462-3	<b>A logic-based incremental approach to graph repair</b>	Sven Schneider, Leen Lambers, Fernando Orejas





ISBN 978-3-86956-504-0  
ISSN 1613-5652