

Master's Thesis



Czech  
Technical  
University  
in Prague

**F3**

Faculty of Electrical Engineering  
Department of Computer Graphics and Interaction

## Web Application for Tabular Data Visualization

**Bc. Zuzana Štětinová**

Supervisor: Ing. Ladislav Čmolík, Ph.D.

Field of study: Open Informatics

Subfield: Computer Graphics

May 2022



## I. Personal and study details

Student's name: **Št tinová Zuzana** Personal ID number: **456883**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Computer Graphics and Interaction**  
Study program: **Open Informatics**  
Specialisation: **Computer Graphics**

## II. Master's thesis details

Master's thesis title in English:

**Web application for tabular data visualization**

Master's thesis title in Czech:

**Webová aplikace pro vizualizaci tabulkových dat**

Guidelines:

Get familiar with the properties of tabular data. Analyze visualization methods suitable for tabular data and decide which data attributes and visualization tasks are supported by which method. Further, analyze tools/frameworks suitable for the development of web-based visualizations of tabular data. Based on the analysis, design and implement a web application capable of loading tabular data from a user's disk and visualizing the data using connected (linked/coordinated) views. Further, implement at least four visualization techniques utilizing the connected views. The web application will support analysis of data with various attributes (quantitative and nominal) and various visualization tasks (e.g., identifying clusters in the data, estimating correlations between the attributes). Test the web application on at least five datasets containing at least five attributes of various types and 100 data items.

Bibliography / sources:

Tamara Munzner. Visualization Design and Analysis, CRC Press, 2015.  
Huntington, Matthew. D3.js Quick Start Guide : Create Amazing, Interactive Visualizations in the Browser with JavaScript, Packt Publishing, 2018.  
Iglesias, Marcos. Pro D3.js : Use D3.js to Create Maintainable, Modular, and Testable Charts, Apress L. P., 2019.

Name and workplace of master's thesis supervisor:

**Ing. Ladislav molík, Ph.D. Department of Computer Graphics and Interaction**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **02.02.2022** Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Ladislav molík, Ph.D.  
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

## III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature



## Acknowledgements

I would especially like to thank my supervisor Ing. Ladislav Čmolík, Ph.D., for his professional guidance and helpfulness in the consulting of this master's thesis.

I would also like to thank all my friends and colleagues who provided me with valuable advice and feedback, especially Mgr. Petr Kubát for the language correction.

## Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

.....  
Zuzana Štětínová

In Prague, May 20, 2022

## Abstract

This master's thesis discusses the design and implementation of an application for tabular data visualization in a Web environment. The application is used to visualize tabular user data with quantitative and nominal attributes and allows one to use different visualization techniques on a single dataset via connected views. The implementation focuses on extensibility and editability for future addition of visualization techniques or other modifications.

**Keywords:** visualization, tabular data, connected views, web application, glyphs, parallel coordinates, scatter plot matrix, parallel sets

**Supervisor:** Ing. Ladislav Čmolík, Ph.D.  
Karlovo náměstí 13  
Praha 2, 121 35

## Abstrakt

Tato diplomová práce diskutuje návrh a realizaci aplikace určené k vizualizaci tabulkových dat ve webovém prostředí. Aplikace slouží k vizualizaci tabulkových dat uživatele s kvantitativními i nominálními atributy a umožňuje používat na jednu datovou sadu různé vizualizační techniky prostřednictvím propojených pohledů. Implementace se zaměřuje na rozšiřitelnost a upravitelnost pro budoucí přidání dalších vizualizačních technik či jiné úpravy.

**Klíčová slova:** vizualizace, tabulková data, propojené pohledy, webová aplikace, glyfy, paralelní souřadnice, matice bodových grafů, paralelní množiny

# Contents

<b>1 Introduction</b>	<b>1</b>	5.5 Chapter Summary . . . . .	48
<b>2 Analysis of Tabular Data</b>		<b>6 Results</b>	<b>49</b>
<b>Visualization</b>	<b>5</b>	6.1 Current State of the Application	49
2.1 Attribute Types . . . . .	5	6.1.1 Quantitative Data	
2.2 Tasks . . . . .	6	Visualization . . . . .	49
2.3 Attribute Mapping . . . . .	7	6.1.2 Nominal Data Visualization .	52
2.3.1 Color Mappings . . . . .	9	6.1.3 Brushing Connected Views ..	53
2.3.2 Other Mappings . . . . .	12	6.2 Meeting the Requirements . . . . .	56
2.4 Interactivity in Visualization . . .	13	6.3 Options of Further Extension . . .	58
2.4.1 Multiple Views . . . . .	14	6.4 Chapter Summary . . . . .	58
2.5 Visualization Techniques for		<b>7 Conclusion</b>	<b>59</b>
<i>n</i> -dimensional Tabular Data . . . . .	15	<b>Bibliography</b>	<b>61</b>
2.5.1 Visualization Techniques for		<b>A User Interface Description</b>	<b>67</b>
Quantitative Attributes . . . . .	15	<b>B List of Attachments</b>	<b>71</b>
2.5.2 Visualization Techniques for			
Nominal Attributes . . . . .	19		
2.6 Chapter Summary . . . . .	21		
<b>3 Application Requirements and</b>			
<b>Tools Selection</b>	<b>23</b>		
3.1 Requirements . . . . .	23		
3.1.1 Functional Requirements . . . . .	23		
3.1.2 Non-functional Requirements	25		
3.2 Current Solutions . . . . .	25		
3.2.1 Tableau . . . . .	25		
3.2.2 XmdvTool . . . . .	26		
3.2.3 Other Tools . . . . .	28		
3.3 Visualization Library Selection .	29		
3.3.1 Recharts and Similar Libraries	29		
3.3.2 D3.js . . . . .	30		
3.4 Chapter Summary . . . . .	36		
<b>4 Design</b>	<b>37</b>		
4.1 Application Data Handling . . . . .	37		
4.2 Visualization Views . . . . .	38		
4.3 Connected Views and Brushing .	40		
4.4 Chapter Summary . . . . .	41		
<b>5 Implementation</b>	<b>43</b>		
5.1 Technologies . . . . .	43		
5.1.1 React . . . . .	43		
5.1.2 TypeScript . . . . .	44		
5.1.3 D3.js . . . . .	44		
5.1.4 Other Libraries . . . . .	45		
5.2 Source Code Structure . . . . .	46		
5.2.1 Modules . . . . .	47		
5.3 Deployment . . . . .	47		
5.4 Code Quality Tools . . . . .	48		

## Figures

1.1 Scatter plots of Anscombe's quartet [3]. . . . .	2	3.6 D3.js scatter plot example without axes. . . . .	34
2.1 Schematic of tabular data. . . . .	5	3.7 D3.js scatter plot example with axes. . . . .	35
2.2 Division of tasks. . . . .	6	3.8 D3.js scatter plot example with brushing. . . . .	36
2.3 Example of attribute mapping for cars data. . . . .	8	4.1 Class model for the application with the data context, settings, layouts and views. . . . .	39
2.4 Non-linearity of hue in standard rainbow scale (HSL) [6]. . . . .	10	4.2 Brushing model for the application with the dataset, views and most important part of the data flow. . . . .	40
2.5 Linearly perceived rainbow scale [4]. . . . .	10	6.1 Scatter plot matrix displaying the flower dataset. . . . .	51
2.6 Example of a hue-based color map. . . . .	10	6.2 Glyphs displaying the flower dataset. . . . .	51
2.7 Exemplary luminance color maps	10	6.3 Glyphs in scatter plot displaying the flower dataset. . . . .	52
2.8 Color wheel with different types of color blindness [7]. . . . .	12	6.4 Parallel coordinates displaying the flower dataset. . . . .	52
2.9 Size mappings examples. . . . .	12	6.5 Parallel sets in bundled layout, displaying the Titanic dataset. . . . .	53
2.10 Example of different shapes used for nominal attribute visualization. . . . .	13	6.6 Brushing in <i>scatter plot matrix</i> projected on parallel sets, glyphs and parallel coordinates. . . . .	54
2.11 Improving comparability with aligning, proximity and ordering. . . . .	13	6.7 Brushing in <i>parallel sets</i> projected on scatter plot matrix, parallel coordinates and glyphs in a scatter plot. . . . .	54
2.12 Example of brushing [11]. . . . .	15	6.8 Brushing in <i>parallel coordinates</i> on two axes, projected on scatter plot matrix . . . . .	55
2.13 Example of scatter plot matrix [13]. . . . .	16	6.9 Brushing in <i>table</i> with filters projected in parallel sets. . . . .	56
2.14 Options for star glyphs [15]. . . . .	17	6.10 The appearance of the resulting application. . . . .	57
2.15 Human-like glyphs. . . . .	18	6.11 The resulting application with open settings. . . . .	57
2.16 Parallel coordinates and different relationships (negative and positive) [19]. . . . .	18	A.1 Top toolbar. . . . .	67
2.17 Nominal categories with quantitative visualizations. . . . .	19	A.2 Drawer with settings of a single view. . . . .	68
2.18 Parallel sets examples. . . . .	20	A.3 View controls. . . . .	69
2.19 Visual clutter with tree layout on datasets with more attributes. . . . .	20		
2.20 Mosaic plot example [22]. . . . .	22		
3.1 Basic visualization of the scatter plot in Tableau. . . . .	26		
3.2 Visualization of the scatter plot matrix in Tableau. . . . .	27		
3.3 Visualization of glyphs in XmdvTool with visible application layout. . . . .	27		
3.4 Visualization of the scatter plot matrix in XmdvTool. . . . .	28		
3.5 Visualization of parallel coordinates in XmdvTool. . . . .	29		



## Tables

1.1 Anscombe's quartet data [3]. . . .	2
2.1 Exemplary car data created to demonstrate attribute mapping. . .	8
2.2 Color blindness in population [8].	11
5.1 Used React libraries and their versions. . . . .	44
5.2 Used D3.js libraries and their versions. . . . .	45
5.3 Used <i>@mui</i> libraries and their versions. . . . .	46
5.4 Used libraries with code quality tools and their versions. . . . .	48





# Chapter 1

## Introduction

Data visualization has helped people for longer than computers have existed, but with their help, the industry has begun to grow rapidly. It expands human possibilities of working with data, allowing us to display data on a screen using various mappings so that we can use it as an external memory. We can then complete different tasks and work with the data more effectively; search for data properties (attribute values) such as extremes and outliers or finding correlations between attribute values of the items. However, it also faces limitations of the computer, human and display. To provide as much help as we can, we also have different types of visualization for different types of data.

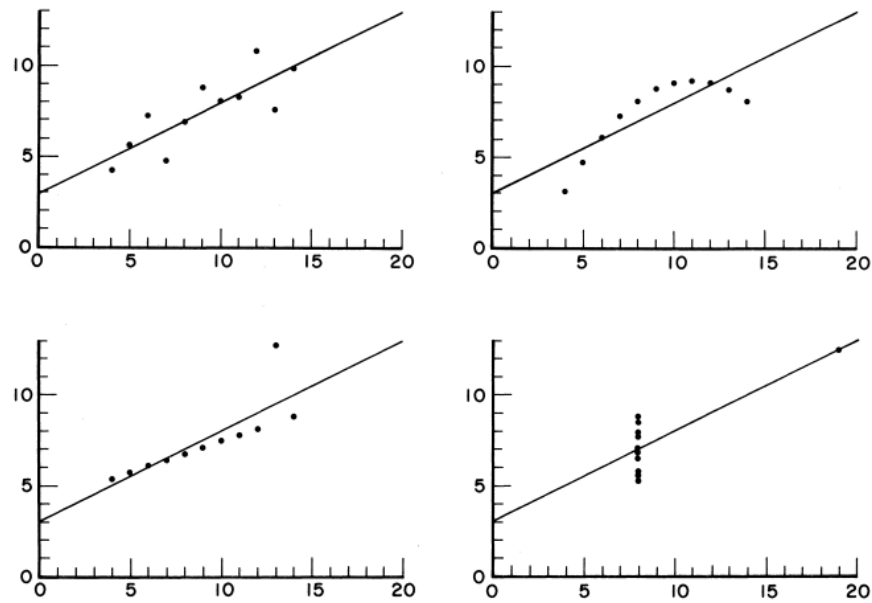
Tabular data are among the most commonly visualized data types. Good examples of such visualizations are the chart options in Microsoft Excel [1] and Google Sheets [2] applications. Tabular data visualization is used to create human-readable representations of tabular data that help the observer to work with the data and not lose any information from the original dataset. Anscombe's quartet [3] shows that the statistical summarization of given data can often be misleading and provide confirmation of incorrect conclusions. In Table 1.1, there are four different datasets with the same mean, variance and correlation, but as shown in Figure 1.1 by visualization of the scatter plots, these data have different characteristics.

Tabular datasets are also often larger and more complex, with a larger number of columns/dimensions (hereafter referred to as  $n$ -dimensional or multidimensional datasets). These datasets could not be clearly displayed on a single simple plot; it requires more sophisticated techniques, which could represent more dimensions in different ways. Depending on the types of attributes that we want to display, we can use methods such as parallel coordinates, a scatter plot matrix, glyphs, or parallel sets, which are described further in the text.

An important aspect in visualization of  $n$ -dimensional datasets is interactivity. To support multiple visualization tasks, it is important to be able to use different visualization techniques, such as those listed above, in different views. Filtering and changing the level of detail can be useful in a single view, but we also need a connection between the views. Brushing can serve that purpose, allowing the user to mark the data in one view and then see them

	1		2		3		4	
	X	Y	X	Y	X	Y	X	Y
	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
	14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
	4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89
Mean	9.0	7.5	9.0	7.5	9.0	7.5	9.0	7.5
Variance	10.0	3.75	10.0	3.75	10.0	3.75	10.0	3.75
Correlation	0.816		0.816		0.816		0.816	

**Table 1.1:** Anscombe's quartet data [3].



**Figure 1.1:** Scatter plots of Anscombe's quartet [3].

highlighted in the others.

With this knowledge in mind, this thesis aims to specify and create a web-based application for the general visualization of  $n$ -dimensional tabular data.

Chapter 2 *Analysis of Tabular Data Visualization* analyzes the data that will be visualized, the types of data attributes, and the mappings of these attributes. It also describes how to connect the data in different views and discusses the techniques used for those views. We can find a summary of

the requirements in Chapter 3 *Application Requirements and Tools Selection*, which also analyzes the current solutions and properties of various tools that can be used for the application. Then, in Chapter 4 *Design* the application design is presented. The implementation is described in Chapter 5 *Implementation*. The result of the application implementation can be found in Chapter 6 *Results*.



## Chapter 2

# Analysis of Tabular Data Visualization

In this chapter, I will analyze tabular datasets, which are one of the dataset types that we commonly visualize. As their name suggests, these are data that can be stored in a single table. We often encounter this type of dataset in databases, finance, storage management, scientific observations, or even the simplest daily tasks. Other types of datasets are network, continuous, or spatial [4].

### 2.1 Attribute Types

Tabular data are defined as items having attribute values, where each item has the same attributes but may have different values. In the table, items are represented using rows and attributes using columns; in each cell, there is a value of an attribute for a given item, as can be seen in Figure 2.1. The attributes can be *nominal* or *ordered*.

	attribute 1	attribute 2	attribute 3	attribute 4
item 1				
item 2		cell		
item 3				
item 4				
item 5				
item 6				

Figure 2.1: Schematic of tabular data.

*Nominal attributes* assign a category to an item, but no comparison relationship is defined between these categories, so they cannot be sorted. Therefore, it can be, for example, the color of a car or the breed of a dog.

*Ordered attributes* assign categories or numbers so that items can be clearly sorted according to them; we can unambiguously determine their order. Furthermore, the ordered attributes are divided into *ordinal* and *quantitative*.

*Ordinal attributes* can be sorted, but calculations cannot be performed with them, such as how much the values differ. An ordinal attribute can be, for instance, the clothing size or the day of the week.





and a location, where to look for it. The reason for the lookup may be to find out what the other attributes of the data at the given location are. We can also *locate* a known target; in other words, we can find the location of a target that we already know. *Browsing* is used if we know the location but we do not know the target. When we do not know either the location or the target, we can *explore* the data.

When we search a dataset, we may *query* various data properties. When we have a single target, we *identify* it. *Identification* gives us all the desired information about the data point. With multiple targets, we can *compare* them to each other, which indicates how much they vary in given attributes. We can also *summarize* the entire dataset.

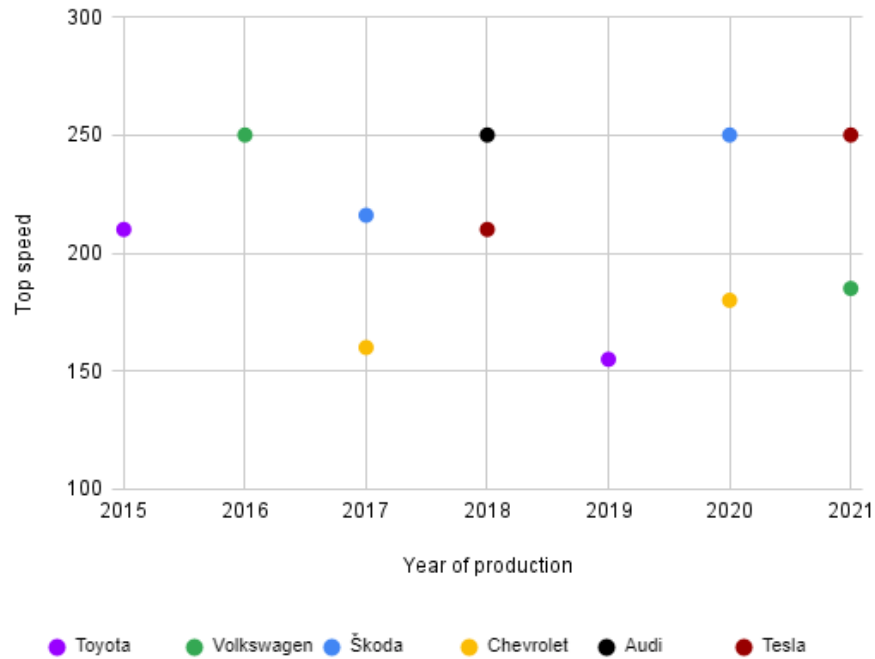
These tasks have certain goals to which they are set up. The user can look for patterns or trends in the data that can be disrupted by outliers. Outliers, data outside the pattern that most of the remaining data show, can give the user interesting information, and the user can investigate whether the outlier is a measurement error or an important value that proves the pattern incorrect. The user can also look for data features based on his current needs, such as the distribution of attribute values or extremes, like the cheapest or the fastest car. In tabular data, we can look for dependencies between attributes or correlations and similarities between items.

In different visualization methods, we can perform individual tasks with different speed and accuracy. However, if we use more visualization techniques and connect them, we can often perform these tasks even more efficiently. One of the most common use cases is to search for trends, outliers, clusters and correlations. I will deal with the connected views later in the text.

## 2.3 Attribute Mapping

Once we know what tasks the user will perform, we can continue by deciding how to display the visualization. We will begin by analyzing the attribute mapping with respect to the various visualization properties. Attribute mapping transforms the attribute value of each item into a visualization feature. As a simple example, we can map the production year of a car to the  $X$  coordinate, the maximum speed to the  $Y$  coordinate and the manufacturer of a car to the color. This example can be seen in Figure 2.3, and its source data in Table 2.1. This creates a quick overview over the whole dataset (overly simplified in this case) that can help us perform the aforementioned tasks. This example also shows us the pitfalls that will be discussed in the following text, such as the need for a legend or problems with color recognition. Only the mappings useful for tabular data will be discussed further in the text. The information in this section, especially the limits of human perception, refers to the facts obtained from Munzner [4] and Quinlan and Humphreys [5].

With tabular data, we have the advantage of being able to use space mappings. We can map the attribute to the  $X$ ,  $Y$ , or even  $Z$  coordinates on computer screens, which is not possible with spatial data (data with fixed position in space). Mapping to 2D space coordinates is perceived as the



**Figure 2.3:** Example of attribute mapping for cars data.

Manufacturer	Production year	Top speed [km/h]
Volkswagen	2021	185
Tesla	2021	250
Chevrolet	2020	180
Škoda	2020	250
Toyota	2019	155
Audi	2018	250
Tesla	2018	210
Chevrolet	2017	160
Škoda	2017	216
Volkswagen	2016	250
Toyota	2015	210

**Table 2.1:** Exemplary car data created to demonstrate attribute mapping.

most exact one; the human visual system (HVS) is adapted to it the most. Mapping to 3D space, on the other hand, can cause some problems and other interferences, with size, colors and other channels that will be mentioned.

It is not necessary to limit ourselves to direct mapping in the space. We can also use ordering, aligning, or space separation to help us navigate the data faster.

There are also non-positional channels to which we can map the attributes. There is a great difference between mapping categorical attributes, such as the manufacturer in the example above (Figure 2.3), and mapping ordered

attributes. Categories can be assigned, for instance, to the color hue or shape of a point. Ordered attributes can be assigned to color saturation, color luminance, size, angle, or others. It is important to know all the properties of different mappings because not all of them can be combined; some could potentially interfere with each other. They also differ in efficiency and have a different number of distinguishable values.

### ■ 2.3.1 Color Mappings

Color mapping, although not among the best, is used quite frequently, so we need to know the features of this mapping. We can use the color mapping on both the fill and the outlines, but we must keep in mind that the color is more recognizable on larger areas and we are not able to perceive differences in color that well if they are on an area that is too small or if they are farther apart. It is also better not to use both mappings (on outlines and fills) at the same time unless we are sure that it serves a specific purpose.

We describe the color mapping used with either a legend (for nominal data or groups) or color maps. Color mapping for nominal data uses hue and maps are always stepped. Color mapping for quantitative data uses luminance or saturation, and we can use either a stepped or continuous color map. With the stepped color map, we choose the interval so that the steps are easily recognizable. Steps can be used for discrete values or ranges, but should be accompanied by labels. With a continuous color map, we need to keep in mind that the user loses the ability to accurately assign value but gains a more holistic view and can perceive differences better if it is suitable for the attribute values.

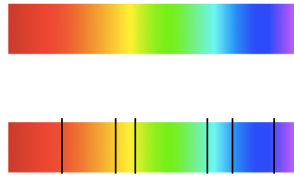
Color mapping for ordered attributes also uses different scales that can be subdivided into sequential, divergent, and cyclical. The *sequential scale* has a maximum and a minimum and grows in one direction, such as the height of the population. The *divergent scale* can be divided into two sequential parts in opposite directions that meet at one common point, as degrees Celsius meet at a freezing point. The *cyclic scale* repeats values in a cycle, as days in a week that can be sorted, but after Sunday comes Monday again. These types can also depend on their usage, so if temperatures in Celsius are measured only for a set of warm summer days, it is not necessary to consider the temperature as divergent.

### ■ Hue

The hue color mapping is not implicitly ordered, so it is often used for categorical or grouped data. We need to know that HVS can quickly distinguish only a few different hue values. The change in hue is also not linearly perceived on a common rainbow scale that changes the hue in the HSL<sup>1</sup> color model (see Figure 2.4).

---

<sup>1</sup>Hue, Saturation, Value



**Figure 2.4:** Non-linearity of hue in standard rainbow scale (HSL) [6].

To create a perceptually linear rainbow scale, we have to use different color models. An example can be seen in Figure 2.5. With that, we lose a bit of dynamic range because some of the colors were omitted, so it looks a bit pale. However, overall we can say that continuous colored scales based on the hue are rarely used.



**Figure 2.5:** Linearly perceived rainbow scale [4].

A more common case are the stepped hue scales, where we pick different recognizable colors from the original scale (with a non-linear scale in non-linear steps, as in Figure 2.4; with a linear scale in linear steps). An example of this hue-based color map can be seen in Figure 2.6. Hue mapping has about seven to twelve distinguishable steps, depending on the background.



**Figure 2.6:** Example of a hue-based color map.

## ■ Luminance and Saturation

Luminance and saturation have similar principles for their usage. For quantitative attributes, there is an urgent need for a linear scale, so the user can see how much the values differ without reading the legend, only with the help of his perception. We can see the luminance color map in Figure 2.7. There is a sequential color map on top, using one hue and different degrees of luminance, and a diverging color map, using two different hues and luminance, where white is used for the zero value.



**Figure 2.7:** Exemplary luminance color maps: top sequential, bottom diverging.

There are also perception issues with luminance. There is a low accuracy in perceiving non-continuous data due to contrast effects. So, when we use this kind of color mapping, it is always better to have a uniform background, to get more precision in perception. Luminance is also often used for other

purposes, such as shading or contrasting text and images. Saturation has similar issues and has an even lower number of discriminable steps than luminance.

We also need to know that some selected hues for luminance or saturation color maps are more appropriate and are used more often than others. The reason for this is better perception of shades in a particular color but also people who have lower recognition abilities in color shades. The biggest extreme we must deal with is color blindness.

For ordinal data that are not quantitative, we need to be careful with using this color mapping, because they can be sorted, but perceived difference in luminance or saturation does not match the difference in value, which is non-linear.

### ■ Problems with Color Mapping

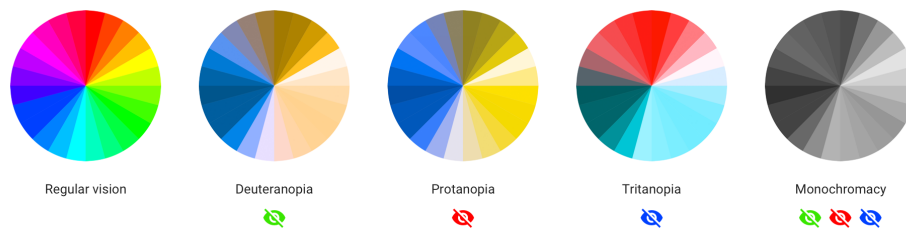
When we decide to use color mapping, we also need to consider the issues associated with it. First, we need to know that color interferes with the size of the object; it might not be easy to distinguish the color of smaller objects. It should not be combined with more levels of transparency, or transparency on a colored background either.

The use of color mapping in 3D visualization also creates problems. If we are using a 3D scene, lighting of that can interfere with the hue of a color and the shadows or self-shadows can interfere with the luminance information. But without good lighting and shadows, we are not able to recognize the 3D shape correctly.

There is also a general problem with color mapping due to a significant number of color blind people in our population. Color blindness is divided into lowered sensitivity (-anomaly) and complete loss (-anopia) of color perception. According to Flück [8], color blindness affects about 8 % of men and 0.5 % of women in the world. The most common type is green color blindness (deuteranomaly/deuteranopia), less common is red (protanomaly/protanopia) and blue (tritanomaly/tritanomia). Absolute monochromacy is very rare. Detailed data can be seen in Table 2.2 and you can see the perception of colors with different types of color blindness in Figure 2.8.

Type	Denomination	Prevalence	
		Men	Women
Monochromacy	Achromatopsia	0.00003 %	
Dichromacy	Protanopia	1.01 %	0.02 %
	Deuteranopia	1.27 %	0.01 %
	Tritanopia	0.0001 %	
Anomalous Trichromacy	Protanomaly	1.08 %	0.03 %
	Deuteranomaly	4.63 %	0.36 %
	Tritanomaly	0.0002 %	

**Table 2.2:** Color blindness in population [8].



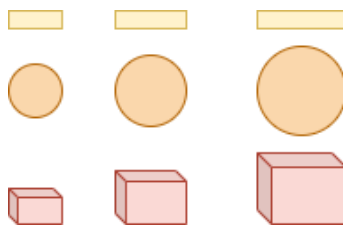
**Figure 2.8:** Color wheel with different types of color blindness [7].

For colorblind users, we have to use appropriate color maps or allow them to change to a different mapping, color or other. We should definitely not use a divergent scale with green and red since green blindness (where green looks nearly the same as red) is the most common type of color blindness. With categorical data, it also helps if we change the lightness together with the change in hue in the color map.

In the next section, we will also go through brushing, which is another method that often uses colors or interfering channels as transparency, and therefore cannot be well combined with color mapping.

### 2.3.2 Other Mappings

For ordered data, we can also use size channels. We can choose different number of dimensions for space mapping, where in 1D it is mapping on length, in 2D on area and in 3D on volume. However, there are big differences in human perception, with length being among the most accurate channels for perceiving differences, but volume among the worst. We can see all three types of size mappings in Figure 2.9.



**Figure 2.9:** Size mappings examples: length (top), area (middle) and volume (bottom).

Due to the properties of HVS, only a few specific values of different sizes have been shown to be better for accurate recognition. It is possible to use a continuous scale for size mapping, but we need to take into account that changing in both  $X$  and  $Y$  directions is not perceived linearly and is not very precise. If we resize the item only in one direction, it is considered a length mapping.

Another possible channel to use is the angle or tilt. With angle we compare two or more lines sharing a single point of origin, with tilt we use the global

frame for comparison. The angle has cyclic properties, making it a good mapping for cyclical repeating attributes. If we map continuous values to angles, we should keep in mind that our perception of angles is not uniform. We distinguish differences near axes and on diagonals ( $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ , ...) even by one degree, but we are not able to do the same elsewhere (for example  $77^\circ$ ). We perceive angle changes better than size, but not as well as length. The angle also interferes with the length channel, where we cannot see it well with small marks.

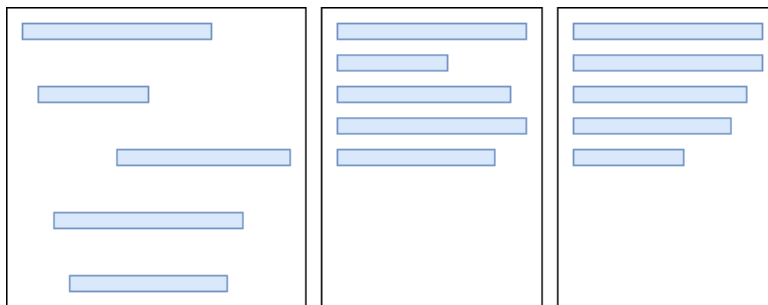
We can also map an attribute to different shapes. Shapes usually cannot be sorted and are used commonly only for categorical data. Shape mapping can be used with mapping on position and color, if chosen right, but it is not usable with size or tilt mapping. The advantage of this approach is that HVS can distinguish a huge number of different shapes, but this depends on the number of pixels available for one shape. An example of different shapes can be seen in Figure 2.10.



**Figure 2.10:** Example of different shapes used for nominal attribute visualization.

There are also other channels, such as curvature or transparency, but these are usually not very accurate, with only a few distinguishable values.

To be able to compare properties such as length or tilt, it is best if the items are close to each other and aligned on a line or with a common center point. In addition, if we sort these values, we get even better comparability. The other option is to add a frame or a grid that can help with the comparability even further. In Figure 2.11 we can see how readability improves first by aligning and moving the items closer, and then also by sorting.



**Figure 2.11:** Improving comparability with aligning, proximity and ordering.

## 2.4 Interactivity in Visualization

Now that we have covered attribute mapping, it is important to go one step further and pay attention to the view in which this mapping will be used.

We must keep in mind that we want to reduce the cognitive burden of the user and not force him to use his own memory too much, the view should





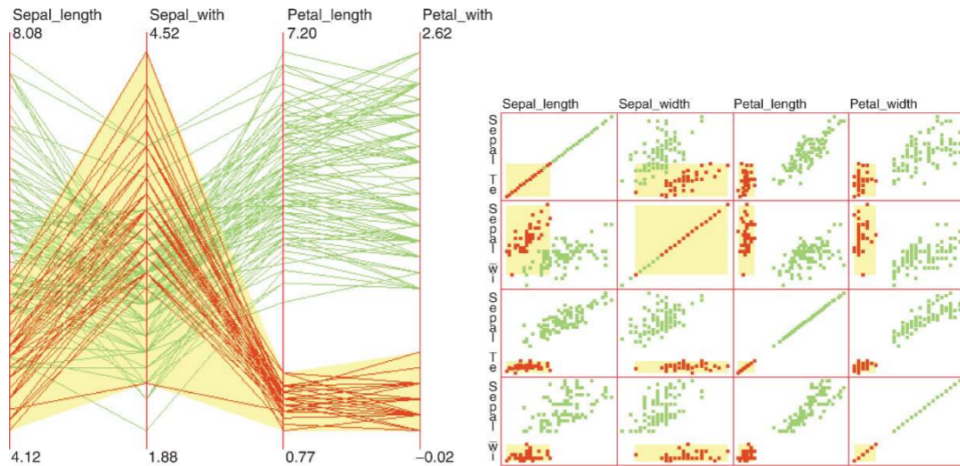


Figure 2.12: Example of brushing [11].

ones are opaque.

## 2.5 Visualization Techniques for $n$ -dimensional Tabular Data

We already presented various options for attribute mappings, but this thesis aims at visualizing tabular data with a higher attribute/dimension count. For that, we need to combine mappings in a way that is scalable and can handle higher-dimension values.

We can use various visualization techniques to visualize  $n$ -dimensional tabular data. We can choose techniques based on suitability for the tasks we want to do on the data or simply select our preferred model.

The tabular data have only items with attributes, no relations or spatial information. They use spatial mappings or ordering and space division. I will focus on selected general techniques for quantitative and nominal data. The techniques discussed are not the only available and there are many opportunities for possible expansion.

For quantitative data, I will describe parallel coordinates, scatter plots, and glyphs. For nominal data, I will mention parallel sets.

### 2.5.1 Visualization Techniques for Quantitative Attributes

With quantitative attributes, we need to remember that we should not only distinguish between values, but the HVS should also help us to tell how much the two compared values differ. As mentioned in Section 2.3 *Attribute Mapping* quantitative attributes can be mapped to position, size, luminance, saturation, angle and others.

We will discuss scatter plot (and its matrix form), glyphs, and parallel coordinates that are all among the best-known quantitative  $n$ -dimensional tabular data visualization techniques.

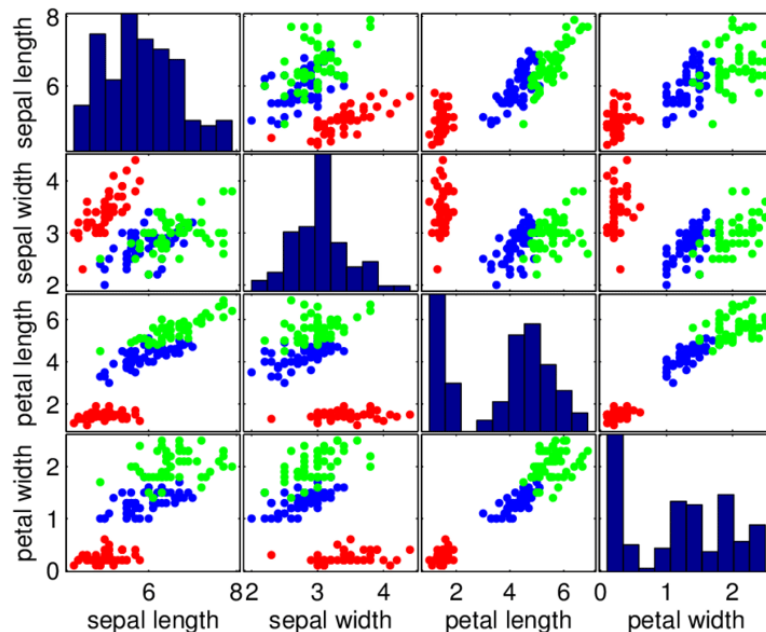
## ■ Scatter Plot

Scatter plot encodes two attribute values to coordinate  $X$  and  $Y$ . It is also possible to use the 3D scatter plot with the  $Z$  encoding, but as we already know, the third coordinate is not as accurate in HVS perception as the first two.

In most cases, we have more than two attribute values, so we must either use additional encoding or create multiple scatter plots.

The easiest encoding to combine with a scatter plot is color or shape. However, the hue or shape can only encode nominal attributes. For quantitative attributes, we have the luminance or saturation of the color. If we need to encode more quantitative attributes, we can use size instead of shape, but this can lead to shadowing of other items. We can see that this approach is not scalable since we can map five attributes at maximum.

If we want to use a scatter plot for  $n$ -dimensional data, we need to create multiple scatter plots. This can be done in the form of a *scatter plot matrix* [12]. We can imagine this matrix as a table with scatter plots as its elements. Each attribute is represented in one column and one row of the same index, and then in scatter plots displayed on the  $X$  and  $Y$  axis. The diagonal can remain empty or show a degenerate case of points on the diagonal line. We can see an example of the scatter plot matrix in Figure 2.13.



**Figure 2.13:** Example of scatter plot matrix [13]. Diagonal is used for histogram values and it maps nominal attribute to color.

In the matrix, we can still encode attributes to color (as seen in Figure 2.13), shape, or size. We can also see that we can use the diagonal fields for other visualizations (here a histogram) or labels. We also do not need to display

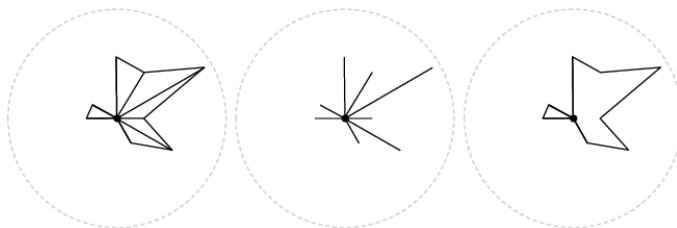
the inverted values; a triangular matrix is sufficient, and we may write, for example, correlation values in the corresponding fields.

The scatter plot matrix is good for finding trends, distributions, correlations between any two attributes, locating clusters, or looking for outliers. We use it to compare all attributes in pairs. It can also be very helpful with brushing, since we can choose data in one of the scatter plots to see the results in other scatter plots or visualization methods. This method is commonly used for up to 12 attributes.

## ■ Glyphs

Glyphs [14] are a composite structure that combines mappings for several attributes. There is one glyph for each item, which means that there are many glyphs in the entire visualization.

The star glyphs [15] map attributes to lengths in different directions from one point. This is done with length normalization, where the attribute value mapped to each star tip has the same length at the maximum value and the same length at the minimum value. In addition, the angle between the tips is the same between each pair of neighbors. You can see different types of star glyphs in Figure 2.14. We can also imagine star glyphs as parallel coordinates (see 2.5.1 – *Parallel Coordinates*) oriented in a circle. But compared to that, by using the radial coordinate system, we reduce the accuracy of the perception of value differences.

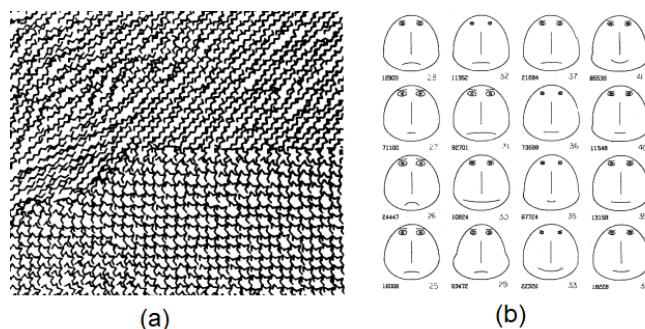


**Figure 2.14:** Options for star glyphs [15].

Another variant of glyphs can be stick figures [16] that map attributes to the length and angles of the 'limbs' of a stick figure. It is also possible to map attributes to facial features (as was used in Chernoff faces [17]). These are easily recognizable for HVS, since a human is optimized to perceive even small details in human faces or postures, even with these simplifications. Face-like glyphs can be used even for mapping ordinal attributes that are not necessarily quantitative as we are not comparing them in the same way as changes of angle or size. You can see examples of both in Figure 2.15

For some glyphs, we can even map an attribute on the glyph color, but we need to think about what the glyph looks like when it has the smallest possible shape and if we can still perceive the color well enough.

We can, on the other hand, map an attribute to the position of the glyph. We can order glyphs based on one attribute, or we can map the glyph on  $X$  and  $Y$  coordinate, making a scatter plot with glyphs, but when the values are



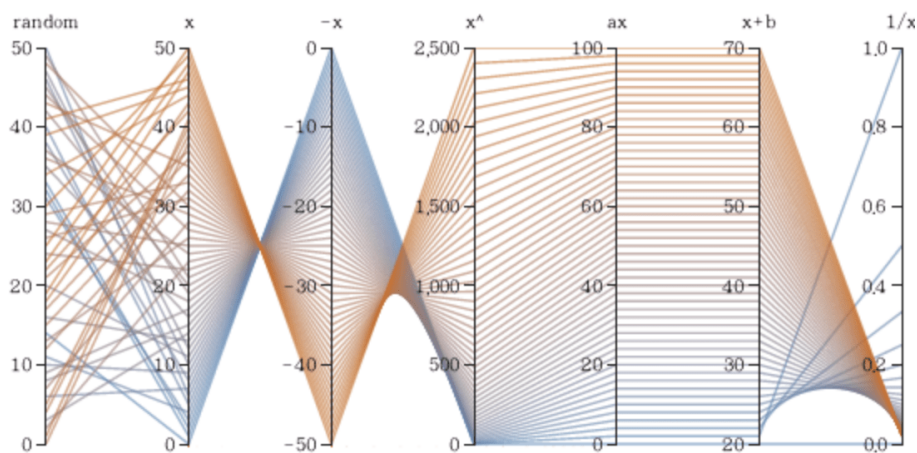
**Figure 2.15:** Human-like glyphs: (a) stick figures [16] and (b) Chernoff faces [17].

too close to each other, we could lose readability. For this type of visualization, most important attributes should always be chosen as positional ones.

Glyphs are not that helpful for brushing, but we can still brush similar-shaped glyphs to look at this selection in other views, or we can use range brushing in scatter plot with glyphs. Starting the other way around, we can brush in other views and just see the selected/colored glyphs corresponding to brushed items; then we can see if similar items have similar glyph shapes.

## Parallel Coordinates

Parallel coordinates [18] use multiple parallel axes, in the  $X$  or  $Y$  orientation, where they map the (quantitative) attribute value. We will describe the  $Y$ -oriented case. All axes are of the same length. It is a good practice to have the value direction consistent, meaning good values at the top and bad values at the bottom, or vice versa. In the method of parallel coordinates, we are mapping each item on every axis and connecting its values with polylines between adjacent axes. An example of parallel coordinates can be seen in Figure 2.16.



**Figure 2.16:** Parallel coordinates and different relationships (negative and positive) [19].

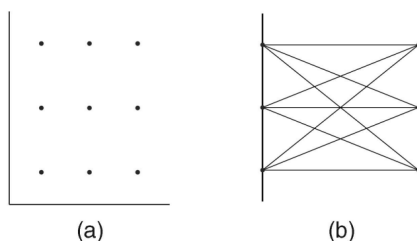
In parallel coordinates, we can observe different relationships between adjacent attributes (see Figure 2.16). To improve that ability and see relationships between non-adjacent ones, we can highlight one of the axes and use (sequential or diverging) color mapping on lines, as can be seen with the highlighted third axis in this figure.

Parallel coordinates also often use brushing, with or without other views. We should be able to select intervals on different axes to highlight the observed data. We should also be able to change the order of the attributes so that we can alter adjacent pairs.

We use parallel coordinates to find clusters in the  $n$ -dimensional space or its subspaces, correlations, outliers, or extremes. It is common to use for up to 12 different quantitative attributes.

### 2.5.2 Visualization Techniques for Nominal Attributes

If we want to visualize the dataset with nominal attributes, we usually use different types of visualization than for quantitative attributes, because we would have many items mapped to one point, line, or position, see Figure 2.17. We need to reverse our approach, and instead of mapping items to specific properties, we find out how many items have each of the attribute values.



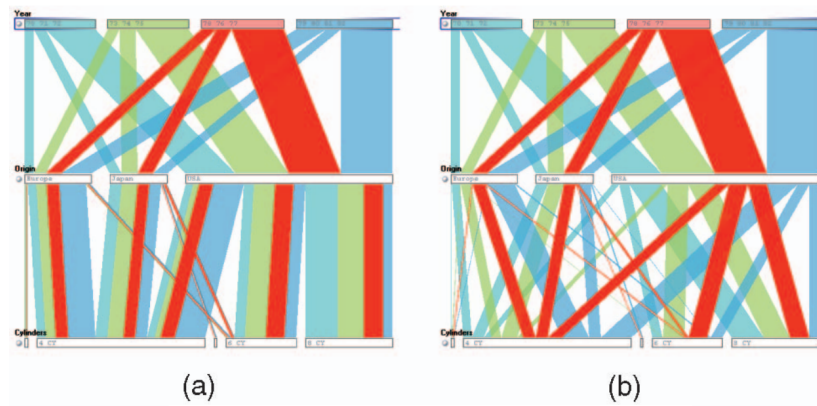
**Figure 2.17:** Nominal categories with (a) scatter plot and (b) parallel coordinates [20].

I chose parallel sets as a representative visualization for this type of data. I will also briefly mention the mosaic plot.

#### Parallel Sets

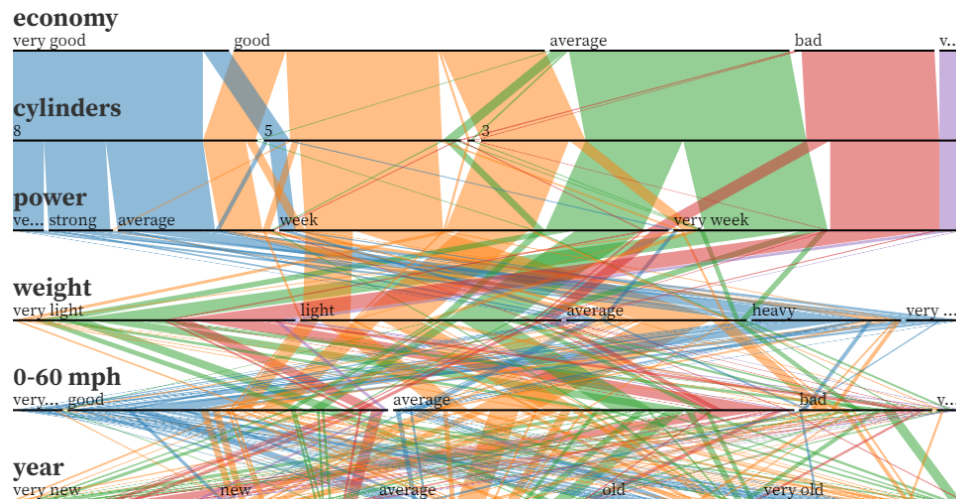
Parallel sets [20] uses similar layout to 2.5.1 – *Parallel Coordinates*, but it shows data frequencies instead of points on the axes. In place of the axes, there are categories, i.e. possible values of the nominal attribute, which scale in length to the ratio of the given value in the dataset (in plot seen as tabs). Between adjacent attributes, for each pair of categories (one of each), the amount of data having both of these values is expressed by the width of the link connecting these categories. Parallel sets can also be oriented in the  $X$  or  $Y$  direction.

Two possible parallel set layouts are described: the bundled layout and the tree layout. Both can be seen in Figure 2.18.



**Figure 2.18:** Parallel sets examples: (a) bundled layout, (b) tree layout [20].

First, I will present the *tree layout*. The tree layout always divides the links that come into each category region (from the second attribute to the third and further). The link originating in the previous attribute value is divided into links with the categories on the other side of that region (at the same place where it enters the region). The division of the links is thus easier to observe, but even with a relatively small number of attributes, a visual clutter can occur soon by reducing the width of the link nearly to the size of the lines. This happens because with each additional attribute, the possible number of links increases by a multiple of the number of its categories. We can see this behavior in Figure 2.19.



**Figure 2.19:** Visual clutter with tree layout on datasets with more attributes (created in tool made by Guerra [21]).

The other option to use is *bundled layout*. The bundled layout does not divide links, only categories. Therefore, it is calculated only within the neighboring attributes, which means that the maximum number of links between adjacent attributes is a multiplication of the number of their categories. For each category region, it is calculated which part of it goes to each category of

the following neighbor (how many data items are in both categories compared to how many data items are in the original category). The calculated ratio is then used to create neighbor links for each pair that has at least one common data item. This principle can also be observed in the tree layout between the first and second attributes.

If we want to color parallel sets in a bundled layout according to the selected category, we color the links in the ratio in which these attributes are represented in the given link, as can be seen in Figure 2.18 for the first attribute.

Parallel sets can be used to find relations between nominal attributes of the data, to get an overview of ratios of nominal attributes in the dataset, or to find outliers in nominal attribute pairs.

The bundled layout is commonly used for up to 12 different category attributes, where every category should have up to 10 values maximum; tree layout is more common to use for fewer attributes or fewer attribute values, to reduce the visual clutter.

When brushing, it should be possible to add individual categories to the selection or remove them to observe the distribution of selected attribute values across all links. We should also be able to select intersections or unions of the categories.

When brushing in another view, the part of the data corresponding to the ratio of the selected data to the not selected should be highlighted on the links to get the overall view.

### ■ Mosaic Plot

One of the other possible visualization methods for nominal values is the mosaic plot [22]. The mosaic plot exists in many different variants, each with its own construction procedure. Its main principle is the division of space into parts that fall into given categories. Most often, it combines a bar chart with a spine plot. It provides a good overview of the categories but does not allow accurate comparisons because HVS does not compare the size of the areas with different shapes well. It is also not very well scalable to the number of categories or attributes. An example of a mosaic plot can be seen in Figure 2.20.

## ■ 2.6 Chapter Summary

In this chapter, I described the main features of tabular data visualization. I discussed the types of attributes with the main division into nominal and quantitative and went through all the different tasks that the user wanted to do in the visualization. Then I analyzed the mapping of attributes to space, shape, and color, and described interactions in the visualization and the usefulness of multiple views. I closed the chapter with an overview of techniques for  $n$ -dimensional tabular data.



Figure 2.20: Mosaic plot example [22]. We can see survivors highlighted in pink.



## Chapter 3

# Application Requirements and Tools Selection

In the previous chapter, I presented the basics we need to know to visualize  $n$ -dimensional tabular data. Now, within the knowledge gained, the requirements for the application will be specified. This chapter will aim at this specification, along with comparing the requirements to the existing solutions and looking for a suitable chart-rendering library to implement the custom solution.

### 3.1 Requirements

First I focused on specifying the requirements for the application. I divided all the requirements into two groups – the functional ones that cover all the tasks that the user wants to perform using the visualization tool, and the non-functional ones that are not connected to the main functionality but are still needed for the comfort of the user.

#### 3.1.1 Functional Requirements

This section contains a summary of the data visualization methods that will be provided and includes a description of other functional requirements.

The application will provide these methods for the visualization of  $n$ -dimensional tabular data:

- For quantitative attributes
  - Scatter plot matrix
  - Parallel coordinates
  - Glyphs (sorted and in scatter plot)
- Nominal
  - Parallel sets in bundled layout

These methods should allow the user to perform the tasks mentioned above (in Section 2.2 *Tasks*) such as comparisons, finding correlations, extremes,

outliers, etc., as described in their individual sections in *2.5 Visualization Techniques for n-dimensional Tabular Data*.

Next, I will describe the functional requirements of the application as a list below. Requirements were formed on the basis of the necessary tasks and general requirements for the visualization tool.

1. *Visualization methods*: The user should be able to use any of the visualization methods for *n*-dimensional tabular data mentioned in the list above.
2. *Use more visualization methods*: The user should be able to use multiple methods/views for the same dataset in his workspace. He should be able to select which methods he wants to display and which not.
3. *Brushing and connected views*: The user should be able to use brushing in all views. Selected items should be highlighted in all visualization views in the workspace.
4. *Loading data*: The user should be able to import his own dataset using one of the commonly used data formats. For tabular data, especially in the Web environment, JSON<sup>1</sup> and CSV<sup>2</sup> are considered commonly used.
5. *Sample dataset*: The user should be able to try the application even without his own data, with a sample dataset.
6. *Layout*: The user should be able to move and resize the views to adapt the visualization to his needs. There should also be several predefined layouts with different views for quick data display.
7. *Attributes interactivity*: The user should be able to interactively change the visualization variables, such as the order of the attributes.
8. *Visualization parameters*: The user should be able to change some visualization parameters, such as choosing which attribute will be mapped to color/shape. These parameters will be chosen individually for each view.
9. *Details on demand*: The user should be able to get details about a data item from the dataset on demand (for example, by hovering over the item).
10. *SVG<sup>3</sup> export*: The user should be able to export the visualization view in its current state as an SVG file.
11. *Options for the color blind users*: The selected colors should take into account the possibility that the user is color blind. Visualizations should provide the ability to change the colors used.

---

<sup>1</sup>JavaScript Object Notation

<sup>2</sup>Comma Separated Values

<sup>3</sup>Scalable Vector Graphics

### 3.1.2 Non-functional Requirements

This section will go through the requirements that the application should meet but that do not form part of its main functionality.

1. *Computer priority:* The application should be optimized for personal computers; it is not necessary for it to work on mobile devices.
2. *Online access:* The application should allow online access and should work on the latest version of the most widely used browsers (Chrome, Safari, Edge, Firefox and Opera).
3. *Serverless:* The application should work on the client side, without the need for a connection to a server.
4. *Clarity and easy operation:* The application should have intuitive controls and a fast learning curve. All parts of the application should be clear and visible. It should contain an option to view detailed information about the controls.
5. *Open-source and extensibility:* The application should be open source and should be prepared for future extensions by other developers.
6. *Speed:* The application should respond reasonably fast to user input; for more demanding operations, it should indicate that the calculation is in progress. It can also provide options for slower computers to limit calculations at the expense of visual appearance, but not usability.
7. *Modularity:* The application should be divided into modules for maintainability according to the principle of separation of concerns (SoC). The goal is to simplify additional extensions and potential changes and fixes.
8. *Size adjustment:* The application should be able to adjust the view based on the size of the window, so that it can be used on different monitors or on just a part of the screen.

## 3.2 Current Solutions

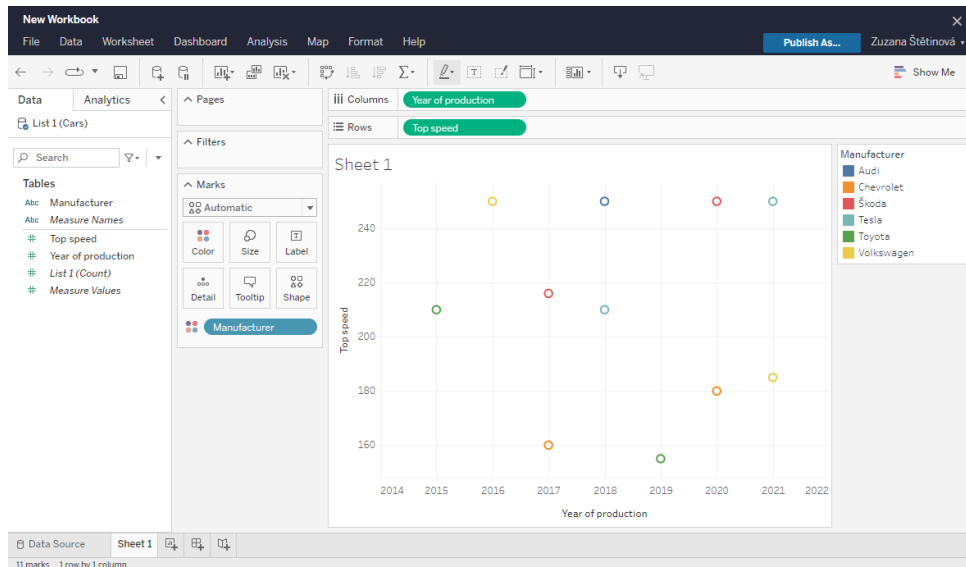
Now that all the requirements have been specified, it is necessary to find out what solutions already exist.

### 3.2.1 Tableau

Tableau [23] is a visual analytics platform. It is a very general tool, with many different versions of applications, such as desktop, server, or even an online version that potentially suits our requirements.

It has a wide range of visualizations available. A basic visualization can be seen in Figure 3.1. As you can see, Tableau provides mappings on color, size,

or shape. It also features tooltips on hover and a legend for categorical data. Custom layouts for more views are also available.



**Figure 3.1:** Basic visualization of the scatter plot in Tableau (generated in Tableau online [23]).

Tableau provides analytics that can perform various tasks, such as adding a trend line or calculating a median. It has many types of visualization, including tree maps, scatter plots, box plots, histograms, or pie charts. However, it does not include multidimensional tabular data visualization methods in its basic options. There are some tutorials [24] on how to draw parallel coordinates or the scatter plot matrix (see Figure 3.2), but it is overly complicated, missing the interaction with brushing and also the ability to make connected views. Also, Tableau is not free software – but it has a free version with fewer features and the limitation that all the projects created that way will become publicly available.

In summary, Tableau will not cover many of our requirements, but it is a pleasant looking and well-written piece of software that can serve as an inspiration for the design of the application.

A similar application is Spotfire® Analytics Accelerated [25]. However, its interface is too simplified for our case and so it is also not suitable for  $n$ -dimensional tabular data, since it provides no visualization methods for them.

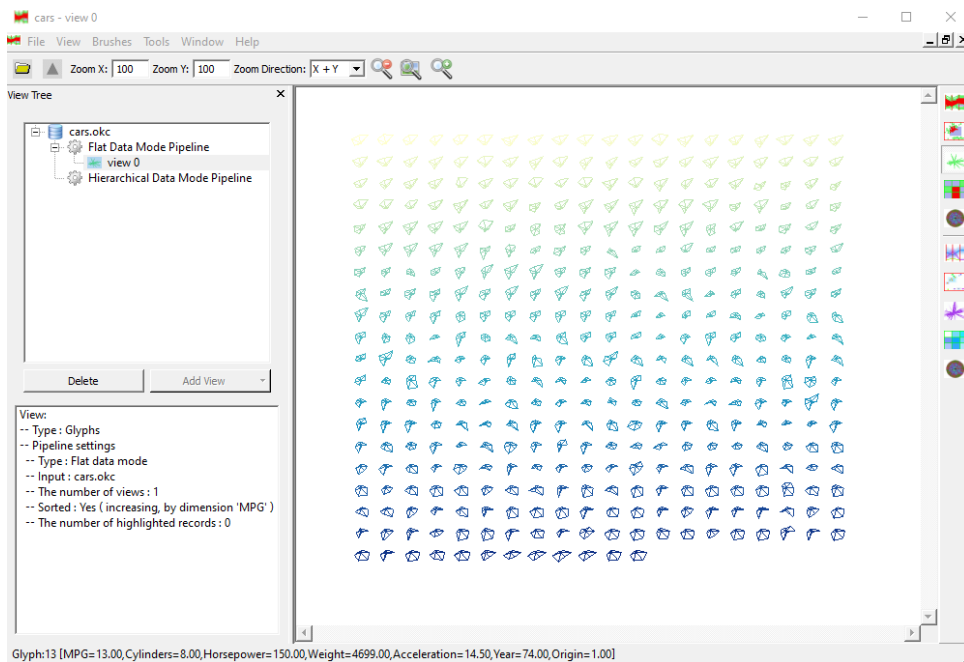
### 3.2.2 XmdvTool

XmdvTool [26] is an open-source visualization tool written in C++ (its source code is available on GitHub [27]). It should work on all major platforms (Windows, Linux and Mac OS) according to its homepage, but the installation process is quite complicated.



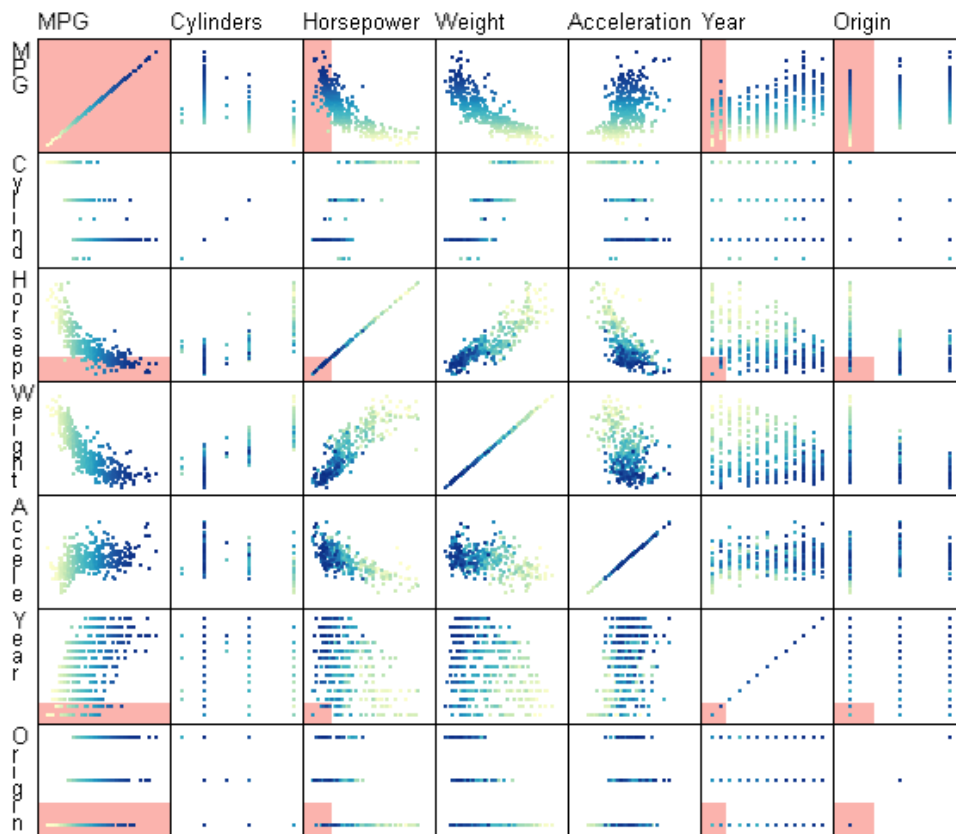
**Figure 3.2:** Visualization of the scatter plot matrix in Tableau.

XmdvTool provides all the techniques needed for visualization of quantitative data. We can see the entire layout with glyphs in Figure 3.3, scatter plot matrix in Figure 3.4 and parallel coordinates in Figure 3.5. The same data, the car dataset [28], is used for all these visualizations.



**Figure 3.3:** Visualization of glyphs in XmdvTool with visible application layout.

XmdvTool also provides additional options, such as changing the order of the attributes in parallel coordinates, although it is not easy to find where



**Figure 3.4:** Visualization of the scatter plot matrix in XmdvTool.

the option is.

Compared to Tableau, XmdvTool is not as easy to use. The application does not have a very intuitive interface and brushing is unnecessarily complicated and confusing. We can see in Figure 3.5 that some labels are truncated and cannot be read in a certain window size. You cannot even reach them by scrolling.

XmdvTool can also use other methods than those in the list, but all of them focus on quantitative data, and it does not provide any of the methods mentioned for visualizing nominal data, such as parallel sets.

### ■ 3.2.3 Other Tools

During the research, I encountered a few other libraries that are also used to visualize tabular data. The main problem was that most of them were available only as a downloadable application, written mainly in C or C++.

There are two open-source visualization libraries written in C, ggobi [29] and gnuplot [30]. They are similar to XmdvTool, with visualization methods only for quantitative data. They are also not available online.

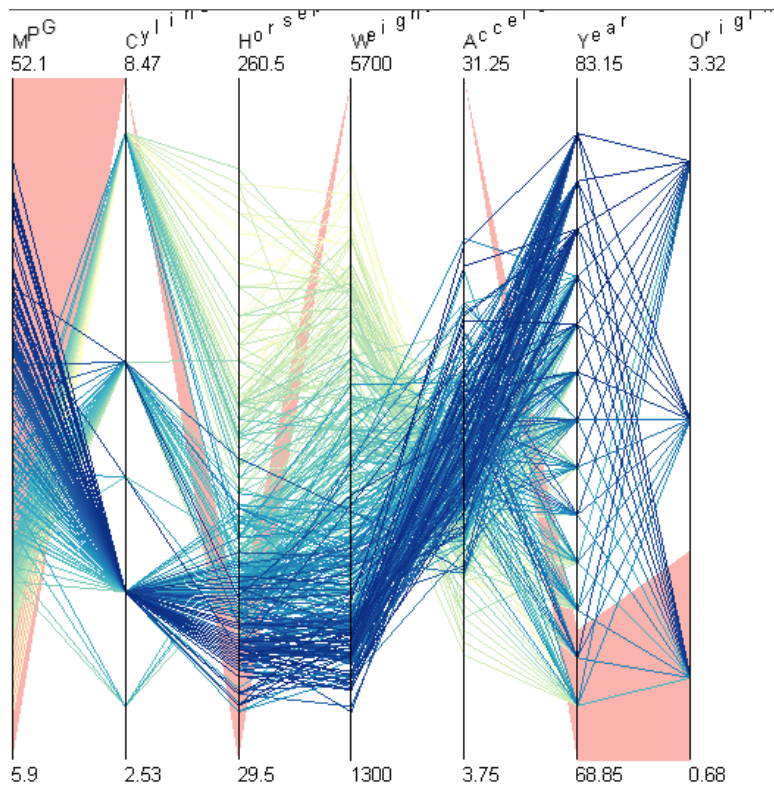


Figure 3.5: Visualization of parallel coordinates in XmdvTool.

### 3.3 Visualization Library Selection

In the research, I did not find an open source online tool that would meet all the requirements, so I decided to create a custom implementation. Therefore, as a next step, I researched visualization libraries to help me achieve this goal.

In this section, I will go through these libraries and then select the most suitable one. I will only focus on libraries that can be used for web client applications. The standard language for web client application development is currently JavaScript, an interpreted language that works in all required browsers.

When choosing a library, I focused mainly on those that use React [31], which is the most widely used framework for writing web applications in JavaScript.

#### 3.3.1 Recharts and Similar Libraries

Recharts [32] is a JavaScript library built with React and D3.js [33]. Recharts has predefined components that serve mainly for presentation purposes. It does not support all the components needed and also does not support any interactions such as brushing.

With this library, we can see that it is possible to use the combination of

React and D3.js to create visualization components. Although it cannot be used for our purposes, it can at least be used as an inspiration for handling Props<sup>4</sup> of the components. It provides basic visualizations such as scatter plots, tree maps, and common charts.

Other React libraries with predefined components are Victory [34] or React-vis [35]. Both libraries share the same problem: they cannot be easily extended and do not meet all of the requirements.

There are more libraries that support React, Vue.js [36], or both, but they face the same issues every time. They have a predefined set of visualizations and behaviors (such as mappings), and if we want to extend this behavior, we have to implement it ourselves outside of the library scope. In the worst case, we could end up with forking the library itself, which is not desirable in the long term.

With this conclusion, I will take a look at a lower-level library that most of the libraries mentioned previously themselves use for attribute mapping and visualizations rendering – D3.js.

### ■ 3.3.2 D3.js

D3.js [33] is a JavaScript library, where D3 stands for Data-Driven Documents. This library can be used to produce interactive data visualizations. It works with SVG and can be combined with common web technologies such as HTML<sup>5</sup>, CSS<sup>6</sup> and others.

D3.js is considered as a successor to Prefuse and Flare. Prefuse was a Java-based toolkit for building interactive information visualization applications [37] that was created in 2005 and its support ended a few years ago. Flare [38] was an ActionScript-based tool developed by the same team of authors that was created in 2007. Flare support was discontinued even earlier than Prefuse. It worked with Adobe Flash Player [39], which stopped being supported in all browsers from December 2020 onward. These two were followed by a JavaScript library Protovis [40], the support of which ended in 2012 and which is considered to be the direct predecessor of D3.js, since it has the same authors – Bostock, Heer, and Ogievetsky. D3.js is more effective and in line with current web standards.

D3.js consists of smaller modular libraries that can be used independently. At its core, it works with SVG elements, creates them, places them in a scene, and adds attributes or classes to them. It provides functions to work with attribute mappings, files (JSON and CSV), brushing, and others. It is a library that allows the user to create a variety of visualizations in SVG.

In our comparison, D3.js works best because it is written natively for browsers, is very flexible, and provides us with the best extensibility.

---

<sup>4</sup>React component properties

<sup>5</sup>Hypertext Markup Language

<sup>6</sup>Cascading Style Sheet



## ■ Core Principles of D3.js

Since we evaluated D3.js as the most suitable library for the application, we can focus on its core principles. I will describe these principles on a small example, the basic scatter plot with simple brushing. This example was created with the React and TypeScript [41] libraries but does not use many of their features. I used this approach to demonstrate that they can be used together with D3.js.

In React, we will first define the interface `ScatterPlotProps` for the properties that the component needs to be able to display. We create a *void function component* (VFC, component without children) `ScatterPlot` that takes the props and returns the element for the React virtual DOM<sup>7</sup>.

Inside the component, we will call a `useEffect` React hook with a function `createScatterPlot` and an empty list of dependencies (second argument), which then triggers the function any time any of the dependencies from the list is changed, so in this case it is called only the first time the component is mounted (added to the DOM). The function `createScatterPlot` (described further in the text) will contain the D3.js code for view creation. The component returns the elements to be rendered, and we can see that the returning element is captured via the `useRef` React hook. This will give us the reference to the element that will be needed for the D3.js code.

We can see the React base of the example in Listing 3.1. We may notice that we are using a reference not directly to the SVG element but rather to a group inside the SVG, where we will add all the data using direct access.

```

1  import { VFC, useCallback, useEffect, useRef } from 'react'
2  import { DataType } from './data'
3
4  // Properties (arguments) needed to produce a scatter plot
5  interface ScatterPlotProps {
6    data: DataType[] // type of the data array
7  }
8  // React Void Function Component drawing a scatter plot from
   given properties
9  export const ScatterPlot: VFC<ScatterPlotProps> = ({data})
   => {
10   // element reference
11   const component = useRef<SVGGElement>(null)
12   //...
13   // in first time rendering, call createScatterPlot
14   useEffect(() => createScatterPlot(), [])
15   // element is returning; g connects to the reference
16   return (
17     <svg width={800} height={512}>
18       <g ref={component} />
19     </svg>
20   )
21 }

```

**Listing 3.1:** Forming of React base.

<sup>7</sup>Document Object Model

After creating the base, it is time to start working with the D3.js library. First, we need to install the library and import it; see Listing 3.2. We will also include the functions necessary to work with the data, in our case functions `getValX`, `getValY` and `getValCat` that return the attribute values for  $X$ ,  $Y$  and a category respectively for a single item in our dataset.

```

1 //...
2 import * as d3 from 'd3'
3 import { DataType, getValX, getValY, getValCat } from './data'
4 //...

```

**Listing 3.2:** Include D3.js import.

Next, we need to create a `createScatterPlot` function for our component to work. We will place the function inside the component. We will start by checking the reference and adding a margin to the group to move it further away from the edge of the SVG. You can see this in Listing 3.3.

```

1 //...
2 const createScatterPlot = useCallback(() => {
3   const node = component.current
4   if (!node) return
5
6   const margin = {top: 20, bottom: 25, right: 30, left: 30}
7   const innerWidth = 800 - (margin.left + margin.right)
8   const innerHeight = 512 - (margin.top + margin.bottom)
9
10  d3.select(node).attr(
11    'transform',
12    `translate(${margin.left}, ${margin.top})`
13  )
14  //...
15 }, [])
16 }
17 //...

```

**Listing 3.3:** Adding margin settings.

We will continue with the function implementation by creating circles inside of the group. Each circle is created with a connection to one of the data items. We can see the definition of circles in Listing 3.4. In this phase, the circles have no assigned position yet, so they are all rendered at the same point.

```

1 //...
2 const circles = d3.select(node)
3   .selectAll('circle')
4   .data(data)
5   .enter()
6   .append('circle')
7 //...

```

**Listing 3.4:** Creation of circles.

The implementation of the function continues with the creation of scaling for positions  $X$  and  $Y$ . We start by creating the extents for  $X$  and  $Y$  according to the data, then we check that the result is defined. If it is, we can set a

linear scale, add a domain from the extent, and set a range for each direction. The code is described in Listing 3.5.

```

1 //...
2 const xExtent = d3.extent(data, getValX)
3 const yExtent = d3.extent(data, getValY)
4 if (xExtent[0] === undefined || yExtent[0] === undefined)
5   return
6 const xScale = d3.scaleLinear()
7   .domain(xExtent)
8   .range([0, innerWidth])
9 const yScale = d3.scaleLinear()
10  .domain(yExtent)
11  .range([innerHeight, 0])
12 //...

```

**Listing 3.5:** Initial setting for scaling.

Now, we will create the mapping of a category to a color by creating a color scale and then setting it as an attribute of the circles. We will also add mapping to  $X$  coordinate with `xScale` and to  $Y$  coordinate accordingly. We will also add radius and opacity. The code for this part can be seen in Listing 3.6. The current state of the example app can be seen in Figure 3.6.

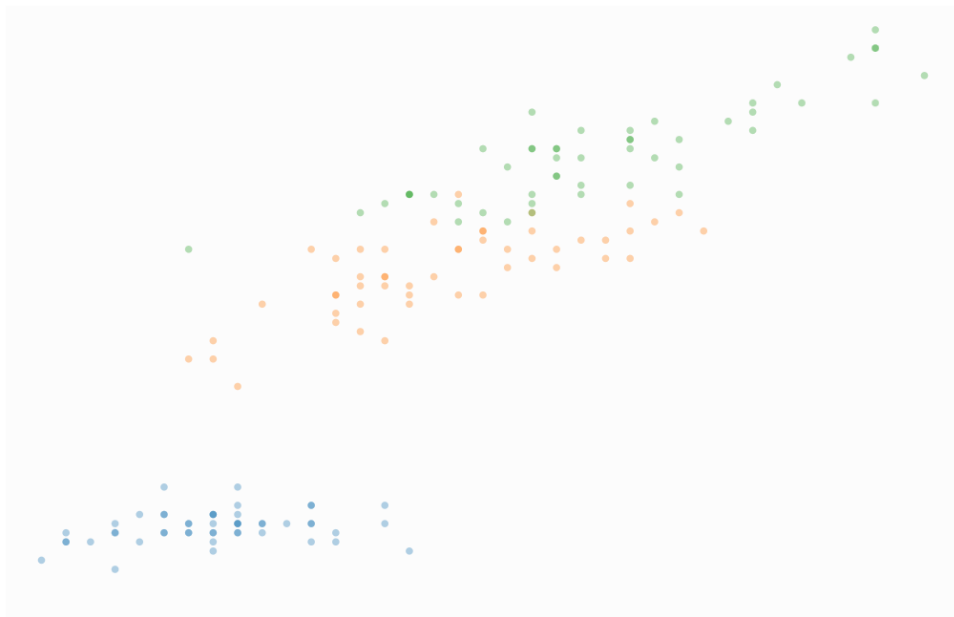
```

1 //...
2 // Create a color scale with the default D3.js color scheme
   for categories
3 const color = d3.scaleOrdinal(d3.schemeCategory10)
4 const fill = (d: TestData) => color(getValCat(d))
5
6 // Adding attributes to circles
7 circles
8   .attr('cx', d => xScale(getValX(d)))
9   .attr('cy', d => yScale(getValY(d)))
10  .attr('fill', fill)
11  .attr('r', 3)
12  .attr('opacity', 0.35)
13 //...

```

**Listing 3.6:** Code for attribute mapping.

To be able to assign values to coordinates, we need to add axes. For that we will continue working on the `createScatterPlot` function. We will add another group to our group and translate `xAxisGroup` to be at the bottom. Then we will create axes with `xScale` and `yScale` and call them for their groups. We can see the code used in Listing 3.7. We can see the result in Figure 3.7.



**Figure 3.6:** D3.js scatter plot example without axes.

```
1 //...
2 const [xAxisGroup, yAxisGroup] = [
3   d3.select(node).append('g').attr('transform', 'translate
4     (0,${innerHeight})'),
5   d3.select(node).append('g'),
6 ]
7 const [xAxis, yAxis] = [d3.axisBottom(xScale), d3.axisLeft(
8   yScale)]
9 xAxisGroup.call(xAxis)
10 yAxisGroup.call(yAxis)
11 //...
```

**Listing 3.7:** Adding axes.

The last thing we would like to do in this example is add brushing. With brushing, we will just set the opacity of the points to one, so that they are more visible, and set the stroke to a dark red color. For that, we will use a D3.js function `classed` that adds or removes a CSS class under some conditions. In our example, the condition is that the center of the point is in the selected rectangle (the function `isBrushed`). Function `startBrushing` will take the selection with the D3.js function `brushSelection` and then use `classed` to set the class `'selected'` if it meets the condition. We will then add a brush to the group node with the right extent. The example of brushing can be seen in Listings 3.8 (code inside `createScatterPlot`) and 3.9 (CSS). The CSS file also needs to be imported. The result of brushing can be seen in Figure 3.8.

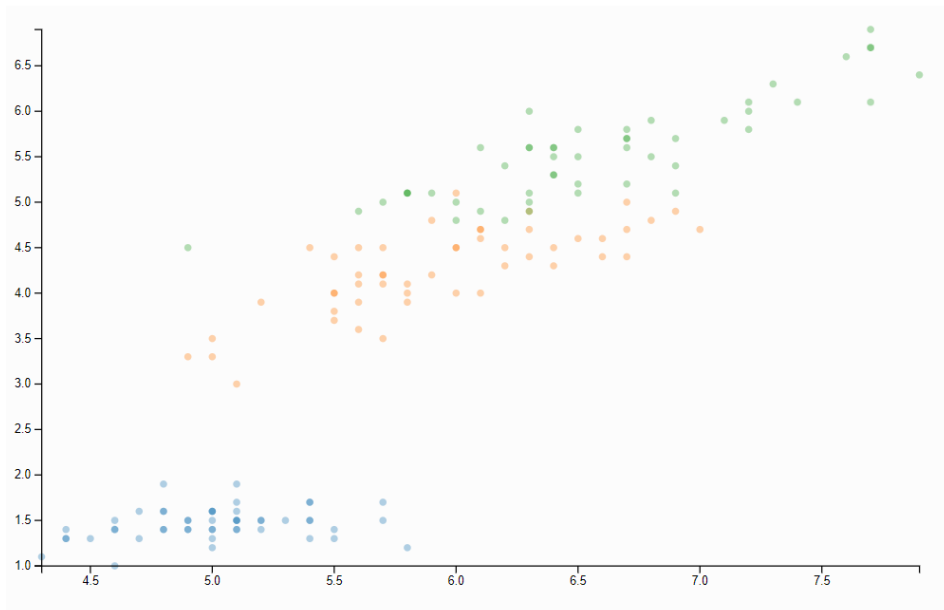


Figure 3.7: D3.js scatter plot example with axes.

```

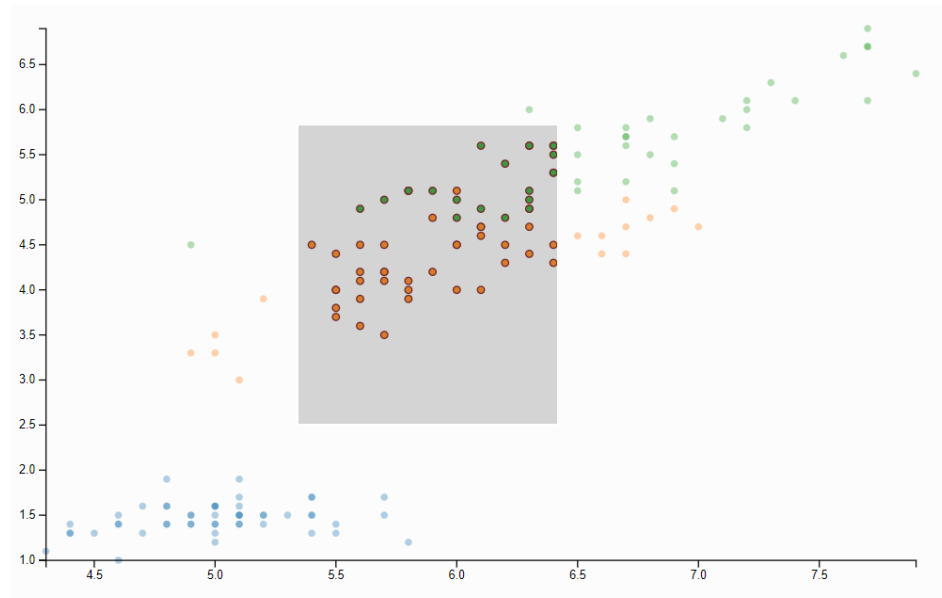
1 //...
2 const isBrushed = (brush_coords: [[number, number], [number,
3   number]], cx: number, cy: number) => {
4   const x0 = brush_coords[0][0],
5     x1 = brush_coords[1][0],
6     y0 = brush_coords[0][1],
7     y1 = brush_coords[1][1]
8   return x0 <= cx && cx <= x1 && y0 <= cy && cy <= y1
9 }
10 const startBrushing = () => {
11   const selection = d3.brushSelection(node)
12   if (selection) {
13     const extent = selection as [[number, number], [number,
14       number]]
15     circles.classed('selected', (d: TestData) => {
16       return isBrushed(extent, xScale(d.sepalLength), yScale
17         (d.petalLength))
18     })
19   }
20 }
21 d3.select(node).call(
22   d3.brush()
23     .extent([[0, -margin.top], [innerWidth, innerHeight]])
24     .on('start brush', startBrushing),
25 )
26 //...

```

Listing 3.8: Implementation of brushing.

```
1 .selected {  
2   opacity: 1;  
3   stroke: #6c0101;  
4 }
```

**Listing 3.9:** Adding CSS for brushing.



**Figure 3.8:** D3.js scatter plot example with brushing.

## 3.4 Chapter Summary

In this chapter, I discussed the functional and non-functional requirements that are needed from the application. I was searching for existing solutions and then described Tableau, XmdvTool and mentioned some other tools and their shortcomings. After that, I searched for libraries that would help me with the implementation. I selected D3.js and created an example to show its core principles and usability with React.

## Chapter 4

### Design

The application design in this chapter is based on the knowledge and requirements described previously that can be found in Section *3.1 Requirements*. It is also partially inspired by the existing solutions in Section *3.2 Current Solutions* and the features of the chosen library D3.js, described in Section *3.3.2 D3.js*.

The application is designed as a web-based client application, which means that there is no back-end or database. The application works with the dataset and can visualize it with several different views. All of these views show this dataset and are interconnected via brushing.

#### 4.1 Application Data Handling

The application shares one dataset between views. The dataset can be loaded from the JSON or CSV file. To parse a dataset, an application must have the original file in a specific form.

For JSON, the file needs to be an array of simple objects. All the objects need to have the same keys and simple JSON values: string, number, boolean or null.

For CSV, there are more formats and separators accepted. The main requirement for the format remains that we have the names of the attributes on the first line and the other values in the same order on the following lines, as is common for the CSV format.

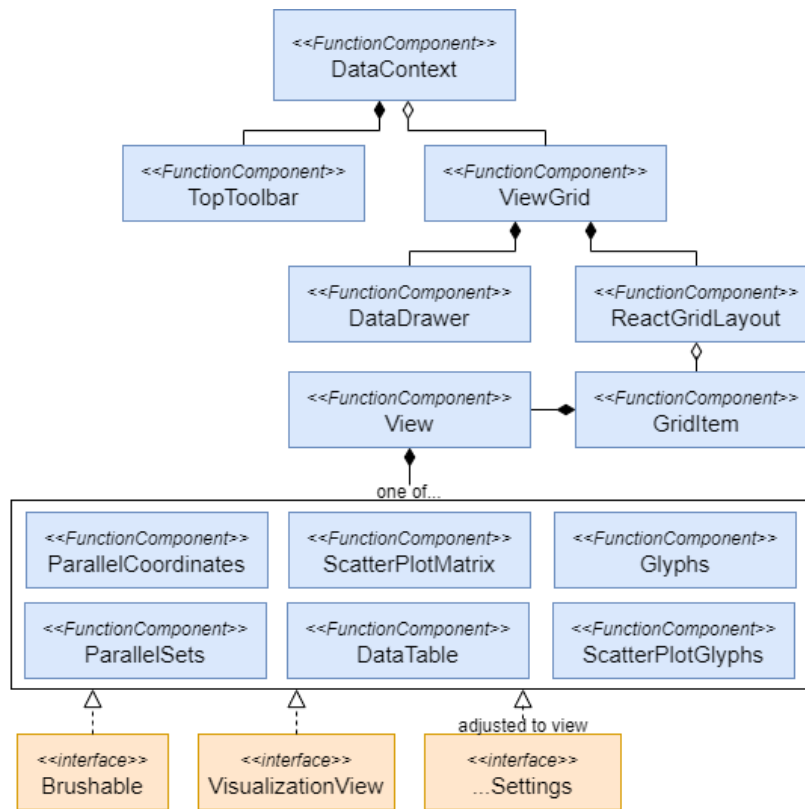
The application then converts both of these into an array of these simple objects. If the application encounters `null` values in the loaded dataset, it will ask the user how it should be processed. Application offers three options; leave them as they are, change them all to a specific value, or delete them.

The sample datasets that can be used to try the application are also in JSON format.

The individual views then extract from the dataset the keys of attributes to display. For quantitative visualizations, it is numerical (or possibly numerical) attributes; for nominal visualizations, it is attributes that have a maximum of ten different values. These attributes are used according to the visualization method in a view – parallel coordinates create one axis for each of them, scatter plot adds a dimension in a matrix, and so on. In the settings, you







**Figure 4.1:** Class model for the application with the data context, settings, layouts and views.

dataset. We assign the dataset to all the axes, which are then scaled to the extent of each quantitative attribute. Each item, according to these scalings, maps each of its given attributes to these axes and connects the path at these points. Brushing is allowed only on the axes, in the  $Y$  direction. If it is used on multiple axes simultaneously, the result is the intersection of these.

The scatter plot matrix creates cells where each cell is one scatter plot in the matrix. Each cell is assigned the data from two of the quantitative attributes that are mapped to the  $X$  and  $Y$  coordinates – scaled to the width and height of the cell with the added axes. To create a matrix, the whole cell is translated according to the index of pair of attributes to which it maps. Brushing is allowed as a rectangle in one of the cells. Activation of the brush in another cell will dismiss the previous brush.

The glyphs are all the dataset items mapped to radial coordinates. Each glyph has a single assigned item. Its attributes are mapped to the distance from the center point on their respective axes. All axes have the same length from the center, so the distances are also scaled. The minimum is shifted to some length, so if a glyph has all values minimal, it is still visible. Glyphs are mapped on a line, sorted by a quantitative attribute, with line breaks. The attribute by which they will be sorted can be passed to the view in a parameter. The glyphs can be brushed by clicking on them.

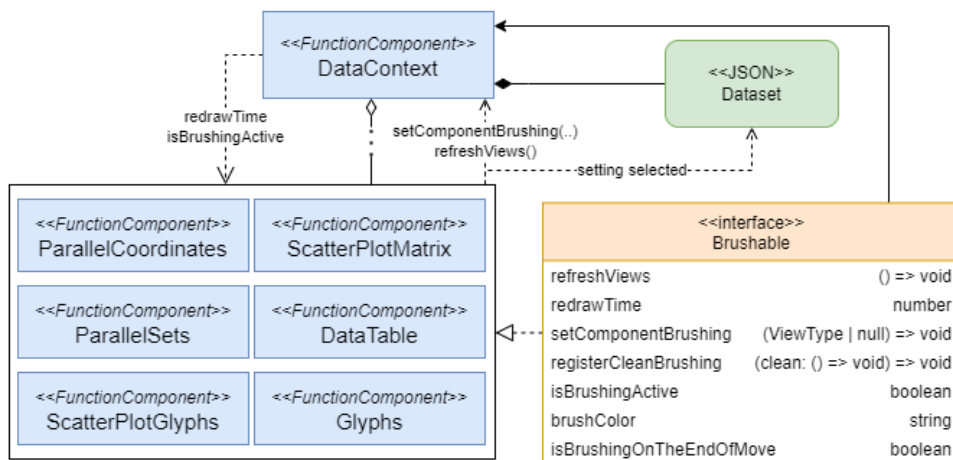
The glyphs in the scatter plot are very similar to the original ones. The only difference is that they are not sorted, but rather placed in position according to two numerical attributes, one defining the X axis and the other the Y axis. Brushing can be done by a rectangular selection that adds every glyph that has a center inside the selection.

Parallel sets, in bundled layout, calculate all the necessary ratios and create a graph to display category tabs and connectors between categories. They are using modified Sankey diagrams [42] to plot connections between adjacent attributes categories. Brushing is done by clicking on category tabs; all items that are in the given category are added to the selection; if the whole category is already in the selection, all items in the given category are removed.

There is also a data table showing all items with attributes as table columns. It can also sort the dataset according to these attributes (numerically quantitative, alphabetically nominal) and string filters can be applied to all of them.

### 4.3 Connected Views and Brushing

In order to support the connected views, we need to add additional properties to the view component. The overview of the properties can be seen in Figure 4.2



**Figure 4.2:** Brushing model for the application with the dataset, views and most important part of the data flow.

We have already mentioned that the dataset has an additional column, `selected`. At any time during brushing, `selected true` or `false` can be set for any item, depending on whether it is selected. But since this is a mutating operation and object references are not changed, React does not register the props change. To get around that, we need to call `refreshViews` after every change. That will change `redrawTime`, which is an artificial component prop whose change will be registered by React, triggering a re-render of all the views.

But we also need to switch brushing between views. We have two types of brushing, one based on an extent used by parallel coordinates, scatter plot matrix and scatter plot glyphs and the other based on individual item/group selection on click used by glyphs, parallel sets and data table. When we start brushing on one of the views, we will always set the triggering view using `setComponentBrushing`. If the component is brushing by extent, it will remove the previous selection and set as `selected` only the values in the new extent. If it is brushing by clicking, it will take the previous selection and add or remove the selected flag. If none of the items are selected and no brushing extents are active, the component brushing should be set to `null`.

Because brushing can store some settings or create visual elements, we also register a clear function for every component that needs it. For that we use `registerCleanBrushing`. Then, each time a brushing component is changed to another using `setComponentBrushing`, the registered cleaning functions are called.

`DataContext` also provides the `isBrushingActive` value, which is true if the brushing component is not null. This can help to style brushed and non-brushed items. Last but not least, two more values are provided – `brushColor`, which determines the color of selected data items, and `isBrushingOnEndOfMove`, which decides whether the view is redrawing only on the extent selection end or also when moving the brush.

Inside the component rendering, we find the items we want to color using d3.js `selectAll` and brush them by adding a class with the `classed` D3.js function. Classes are connected to a CSS-like styling.

By implementing the brushing using an additional column in the data, we achieved flexibility and simplicity in the way the views are connected using D3.js logic and React component re-rendering, receiving the same dataset as a property.

## ■ 4.4 Chapter Summary

In this chapter, I analyzed the design of the proposed application. First, I focused on how the application will work with the data. Next, I described the individual methods and their connection to the data. For these methods, I mentioned how brushing is used. Finally, I described the brushing between views and what has to be considered when designing brushing in general.



# Chapter 5

## Implementation

I implemented the application based on the design mentioned in Chapter 4 *Design*. In this chapter, I will describe the technologies used, the source code structure, application deployment, and the code quality tools used.

### 5.1 Technologies

The main technologies used in the implementation of the application were React, TypeScript and D3.js. To implement the application, I used the IDE<sup>1</sup> WebStorm [43] by JetBrains in version 2021.2.4, student license.

In this section, I will describe the libraries with the main technologies and briefly mention the libraries that I used to work with data, layout, and styling.

#### 5.1.1 React

The application uses React to run, the base of the application was built with `npx create-react-app my-app --template typescript`. More information on the running and deployment of the application is provided in Section 5.3 *Deployment*.

React is an open source library written in JavaScript (with the TypeScript extension) that allows to build a UI<sup>2</sup> in a functional way, composed of reusable components. As you can see in the React documentation, React is using virtual DOM (ReactDOM), where the UI representation is inside the browser memory and is synchronized with the real web page DOM. In this way, the React library is adapted to work with rapidly changing data in a browser. React also works as a single-page web application.

The versions of the React libraries used can be found in Table 5.1.

I use React with function components and React hooks. That is a modern alternative to class components, providing a more direct API<sup>3</sup> to the main React concepts such as component state, lifecycle, refs, and others.

---

<sup>1</sup>Integrated Development Environment

<sup>2</sup>User Interface

<sup>3</sup>Application Programming Interface

Library	Version
<i>react</i>	17.0.2
<i>react-dom</i>	17.0.2
<i>react-scripts</i>	4.0.3
<i>react-use</i>	17.3.2
Development only	
<i>@types/react</i>	17.0.2
<i>@types/react-dom</i>	17.0.2

**Table 5.1:** Used React libraries and their versions.

The *react-use* [44] library, which contains derived hook options, is used to work with component and window sizes.

### ■ 5.1.2 TypeScript

In order to at least partially overcome the disadvantage of JavaScript, that it is a language without static typing, I am using its TypeScript extension in the implementation of the application. It enables static typing and checks these types at compile time. TypeScript uses structural typing, where two types match when they have the same structure.

However, it is necessary to take into account that JavaScript with TypeScript is not a natively typed language, and therefore, various problems arise where the static types do not necessarily correspond to the run-time ones. TypeScript also allows you to use the `any` type, which says that an item can have any type or structure. This can pose a problem, especially while importing libraries without proper typing conventions.

I determined how TypeScript will be used (what is allowed, which version of JavaScript it will compile, and similar) in the configuration file `tsconfig.json`. To make the best use of TypeScript, I set strict rules in this file by setting the value of `strict` to `true`. This option turns on strict type checks, for example, for no implicit `any`, so the developer does not forget to always specify the type of each expression or argument. TypeScript was used as a development dependency in version 4.1.2.

Using TypeScript and adding types takes some time during development. However, it can save a lot of time later by helping avoid complicated debugging when using the wrong type. Good usage of TypeScript also reduces the need for documentation and increases the readability for further modifications and extensions of code compared to pure JavaScript.

### ■ 5.1.3 D3.js

For the implementation of the application, I used the *d3* library, as decided in Section 3.3.2 *D3.js*. For a valid use of TypeScript in the application, I added an extension in the form of the *@types/d3* library for development.

I also used the *d3-sankey* library for parallel sets to render multiple adjacent

Sankey diagrams. However, I encountered two problems with this library. The first was that the size of the graph adjusts to the number of tabs; this problem was solved by recalculating the height according to the number of maximal distinguishable values among all attributes. The second was that the library recalculates the tab *Y* positions to optimize the length and overlap of the connectors. However, this meant that the neighboring graphs were not contiguous. I needed to turn this optimization off, but the library did not provide a way to do it from the outside. That is why I had to fork the library to disable this behavior.

The forked library and its types can be found in the `src/lib` folder of the project. It was also needed to add one more dependency, *d3-array*, which is used by this library.

The versions of the libraries used can be seen in Table 5.2.

Library	Version
<i>d3</i>	7.1.1
<i>d3-array</i>	3.1.6
Development only	
<i>@types/d3</i>	7.1.0
Forked and modified libraries	
<i>d3-sankey</i>	0.12.3
<i>@types/d3-sankey</i>	0.11.2

**Table 5.2:** Used D3.js libraries and their versions.

The usage of the current version of the D3.js library turned out to be a bit complicated, as most of the available examples are written using an older one. The current version brought some breaking changes, so I found a migration guide [45] on the Observable website. However, the current version is more robust and intuitive.

As inspiration, I studied various examples on the Observable [46] website, from which I learned the basics about how to use the library. I also studied documentation for version 7 [47].

#### ■ 5.1.4 Other Libraries

I will also mention the *csv-string* [48] library (used in version 4.1.0), which helped me recognize the delimiter and parse the loaded string from CSV to JS objects. But it still returned all the values as strings that needed to be remapped. As part of this effort, I found out that there can be some inconsistencies in the CSV format. To solve that, I make sure that both the empty value and the string value 'null' are mapped to `null` and that the numbers can work even with a decimal comma instead of a decimal point.

For the tile layout in the application, I used *react-grid-layout* [49]. It was originally created in 2014 and even though the community is actively making updates, it uses some older code design choices, such as the need to import CSS style files in the application root. The library produces a





We can see several folders in the root of the repository. The source code is located inside the `src` folder. The `public` folder contains basic public information, such as favicons, manifest or images. The `src/load-test-data` folder contains data that were used to test data reading using file input. Other root files are various configurations, for example, for Git or TypeScript.

In `src`, there are the aforementioned folders `icons` and `lib`, which contain modified SVG icons and forked libraries, respectively. In `src/test-data` there are JSON files for sample data that can be used in the application instead of loading a file. There is also an `index` file generated by React that serves as a React root. The modules can be found in `src/app`.

### 5.2.1 Modules

The application is divided into modules that have various functions.

The `components` contain all React components and their rendering and data management logic. In `content/views` there is this logic for visualization views and their `SettingsComponents`. In `content/context` there is a `DataContext` component that contains the dataset described in *4.2 Visualization Views*. The rest of the folders within `components` contain other components, such as, for example, the top toolbar.

In the `components-style` module, custom styles are defined corresponding to the components. We may notice that for `views`, the styling is defined by a single function that sets all the styles on the SVG element and its descendants. This method was chosen so that the styles generated by `@mui` could be easily identified and thus only those belonging to each SVG were assigned to it.

Associated with it is the `styles` module, which contains the colors used for component styling.

The module `constants` contains important constant values used in the components. It contains not only enumerations for options selection or actions, but also the string and number constants used in a code.

The `text` is a similar module that contains all the text present in the application. With the change of this module, the application can be fully translated into another language.

Next, there is the `helpers` module that provides reusable helper functions for the components. There are functions to work with React, D3.js or data in general, but also functions specific for the views.

Finally, the module `types` contains the typings that the components can reuse. It provides, for instance, interfaces that the component props need to extend, settings for views, or the type of the dataset.

## 5.3 Deployment

The application is currently deployed on the GitHub pages [53].

If you want to run the app locally, download the source code [52] and install the Node [54] run-time environment and the Yarn [55] package manager. The versions of Node and Yarn used for development were `v16.13.0` and `1.22.17`,

respectively. To install the project (download libraries), use the command `yarn` inside the project root. After the installation is complete, you can use `yarn start` to run the project.

I used Yarn because it is the current standard for new React projects and performs better than the npm [56] package manager.

## 5.4 Code Quality Tools

The project is expected to be extended in the future, so I chose to use the ESLint code quality tool [57] along with Prettier [58] opinionated code formatter to maintain a consistent code style.

ESLint with Prettier statically analyzes the code and returns warnings or errors according to a customizable set of rules. For the purposes of the application, I use the TypeScript extension and rules. The styling libraries used can be seen in Table 5.4. Because it is a code quality tool, all dependencies are only for development.

Library	Version
<i>eslint</i>	7.28.0
<i>prettier</i>	2.6.2
<i>@typescript-eslint/eslint-plugin</i>	5.19.0
<i>@typescript-eslint/parser</i>	5.19.0
<i>eslint-config-react-app</i>	7.0.1
<i>eslint-config-prettier</i>	8.5.0
<i>eslint-plugin-react</i>	7.29.4
<i>eslint-plugin-react-hooks</i>	4.4.0
<i>eslint-plugin-prettier</i>	4.4.0

**Table 5.4:** Used libraries with code quality tools and their versions.

ESLint rules are set in the project through the configuration file `.eslintrc.js` and the files that will be ignored during the check are included in `.eslintignore`. When the app is executed, warnings and errors can be found in the terminal window. To show only ESLint warnings and errors, it is possible to either set it up in the IDE or call the command `yarn lint` in the root directory.

ESLint itself can solve most of the problems it encounters. This fix can be performed within the IDE or by using the `yarn lint --fix` command.

## 5.5 Chapter Summary

In this chapter, I analyzed the technologies used, mainly React, TypeScript and D3.js. I also briefly discussed the structure of the source code. I then described how the application is deployed and how to run it locally. Finally, I mentioned the tools used to keep the code clean with a single code style.

# Chapter 6

## Results

This chapter describes the current state of the application, especially the results achieved. The application was run and tested mainly on the Google Chrome [59] browser.

### 6.1 Current State of the Application

This section focuses on the results achieved and the current capabilities of the application.

#### 6.1.1 Quantitative Data Visualization

First of all, I would like to present the results of the quantitative data visualizations. All of the given examples of these visualizations were tested on three sample datasets – flower, bird, and car datasets. The number of items in the flower, bird, and car datasets is 150, 247, and 392 items, respectively. The structures of the items in each dataset can be seen in Listings 6.2, 6.1 and 6.3.

```
1 interface FlowerData {
2   sepalLength: number
3   sepalWidth: number
4   petalLength: number
5   petalWidth: number
6   species: string
7 }
```

Listing 6.1: Dataset structure for flower data.

```
1 interface BirdData {
2   species: string
3   island: string
4   culmen_length_mm: number
5   culmen_depth_mm: number
6   flipper_length_mm: number
7   body_mass_g: number
8   sex: string | null
9 }
```

Listing 6.2: Dataset structure for bird data.

```

1 interface CarData {
2   name: string
3   economy_mpg: number
4   cylinders: number
5   displacement_cc: number
6   power_hp: number
7   weight_lb: number
8   0_to_60_mph_s: number
9   year: number
10 }

```

**Listing 6.3:** Dataset structure for car data.

In all the datasets, numerical attributes are considered quantitative and the others nominal, with the exception of the `cylinders` of cars, which can be both quantitative and nominal due to a smaller number of distinguishable values. For example, a flower has quantitative attributes of `sepalLength`, `sepalWidth`, `petalLength`, and `petalWidth`, and nominal attribute of `species`.

Quantitative views have some common features. Each view gets the same list of possible display attributes (quantitative) and the same list of nominal attributes.

For each of the views, we can choose the display attributes we want to show and the order in which they appear. We can also choose which of the nominal attributes we will use to color the data items, if any.

Three different methods are implemented to visualize quantitative data in the application: *scatter plot matrix*, *glyphs*, and *parallel coordinates*. Glyphs are also implemented in two variants, sorted and in a scatterplot. In all these views, the opacity of data points in different brushing states, the margin from each side and category colors used can be set.

The *scatter plot matrix* method can be seen in Figure 6.1. All attributes are displayed in the same order as in the dataset, meaning the first attribute on the first line and the last column, and the last attribute on the last line, first column. The nominal attribute, in this case the `species` attribute, is selected to color the data points. In the scatter plot matrix view, we can also set the point size or the horizontal and vertical spacing between cells.

In Figure 6.2 we can see the mapping of the items to *Glyphs*. The attribute axes are displayed from the top tip of the glyph clockwise. The glyphs are sorted in lines according to the attribute `sepalLength`, ascending from top left to bottom right. Items with similar attribute values have similar glyph shapes. The attributes by which the glyphs are sorted and whether the sorting happens in ascending or descending way can be changed. The glyphs are also colored according to the `species` attribute value. The user can also specify the size of the glyphs and the spacing between them.

Figure 6.3 shows the glyphs in a scatter plot. Here, instead of sorting, we select numeric attributes for the X and Y axes. In this case, we are also able to set the coloring attribute and the glyph size.

*Parallel Coordinates* method is shown in Figure 6.4. The axes are in the same order as the received attributes, from left to right. The nominal

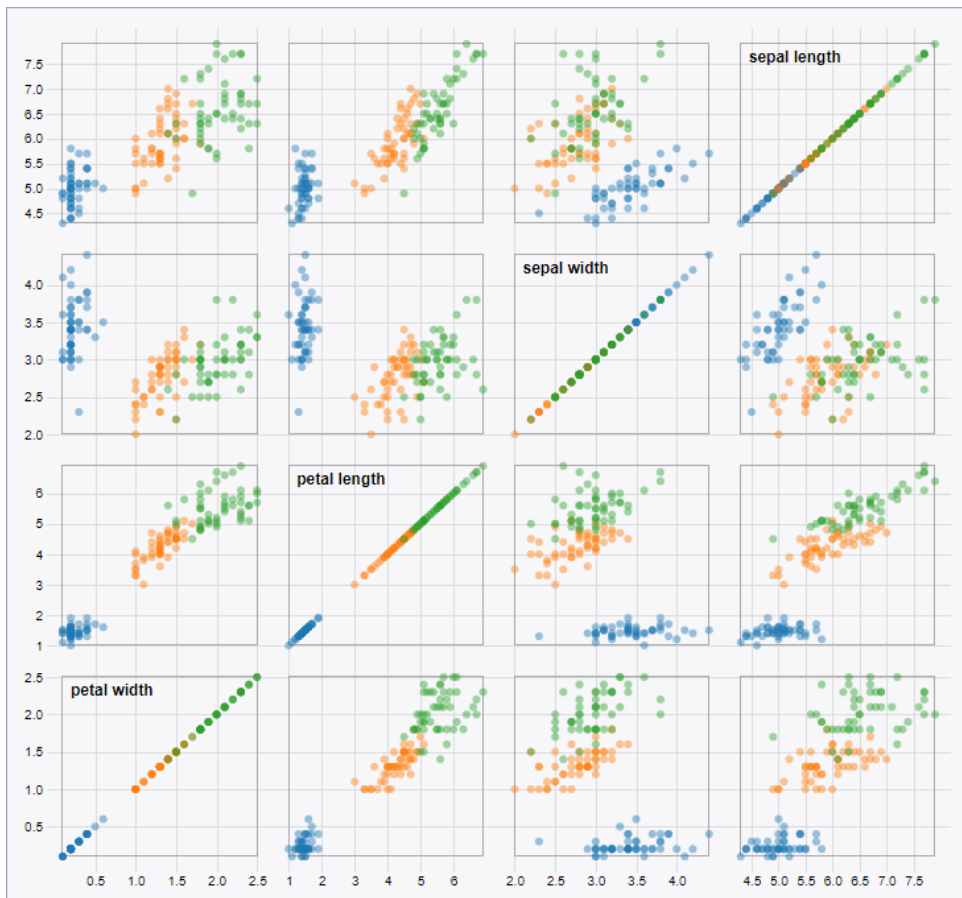


Figure 6.1: Scatter plot matrix displaying the flower dataset.

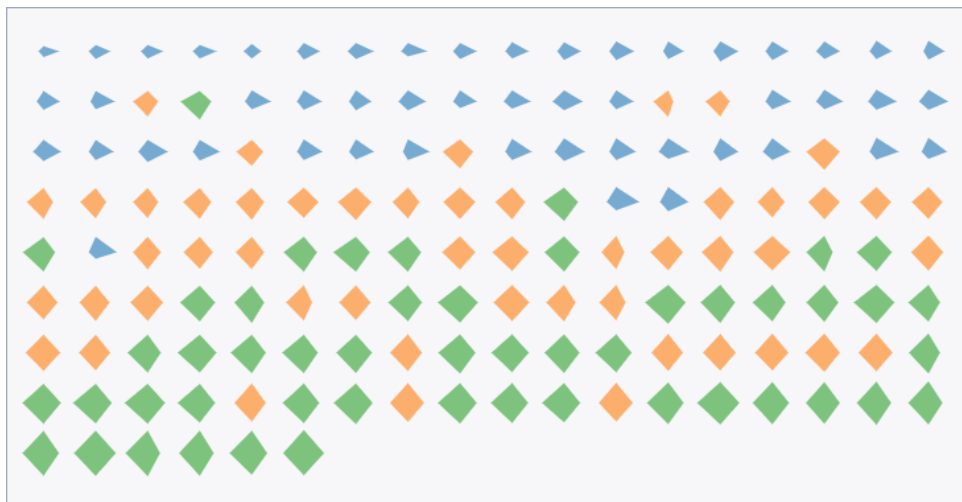
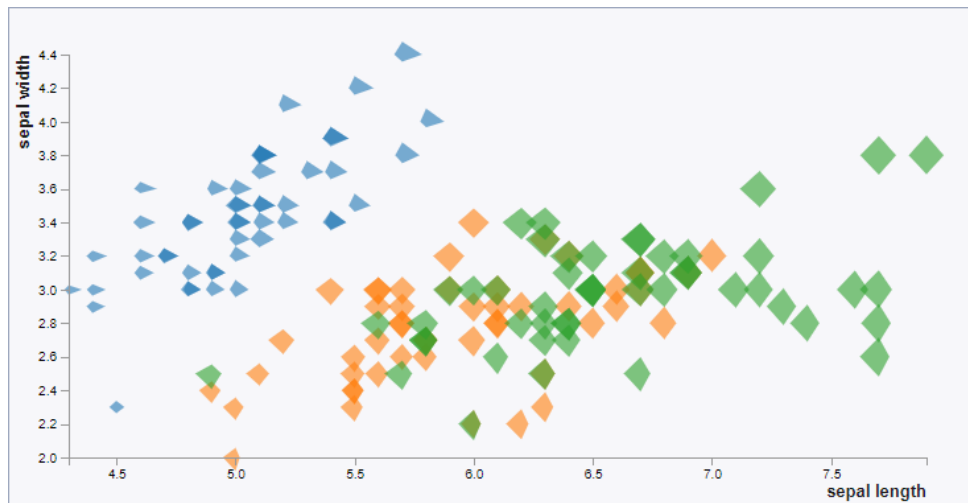
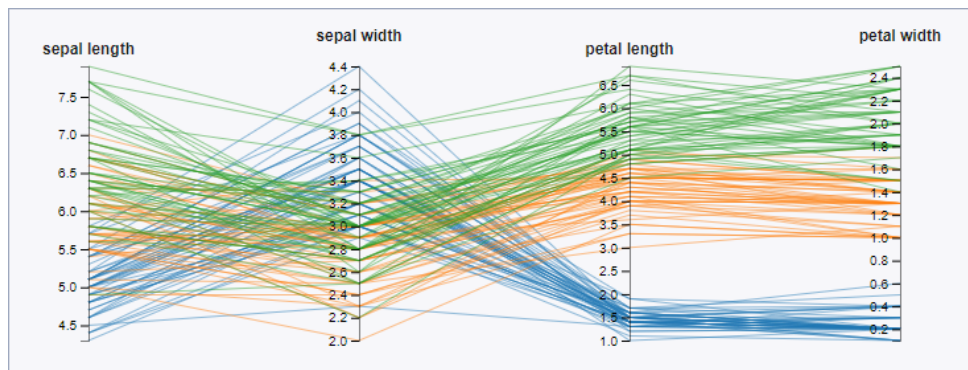


Figure 6.2: Glyphs displaying the flower dataset.

attribute selected is used to color the lines in this case. It is possible to set the width of the lines.



**Figure 6.3:** Glyphs in scatter plot displaying the flower dataset.



**Figure 6.4:** Parallel coordinates displaying the flower dataset.

## 6.1.2 Nominal Data Visualization

I would like to continue with the results of the nominal data visualization. The nominal visualization was tested on two new datasets, together with the already mentioned bird dataset (see 6.1.1 *Quantitative Data Visualization*). The new ones are the Titanic dataset and a modified car dataset for nominal attributes, with 2201 and 392 items, respectively. The structures of the dataset items are shown in Listings 6.4 and 6.5.

```

1 interface TitanicData {
2   Class: string
3   Age: string
4   Sex: string
5   Survived: string
6 }

```

**Listing 6.4:** Dataset structure for titanic data.

```

1 interface CarParallelData {
2     economy: string
3     cylinders: number
4     power: string
5     weight: string
6     0-60 mph: string
7     year: string
8 }

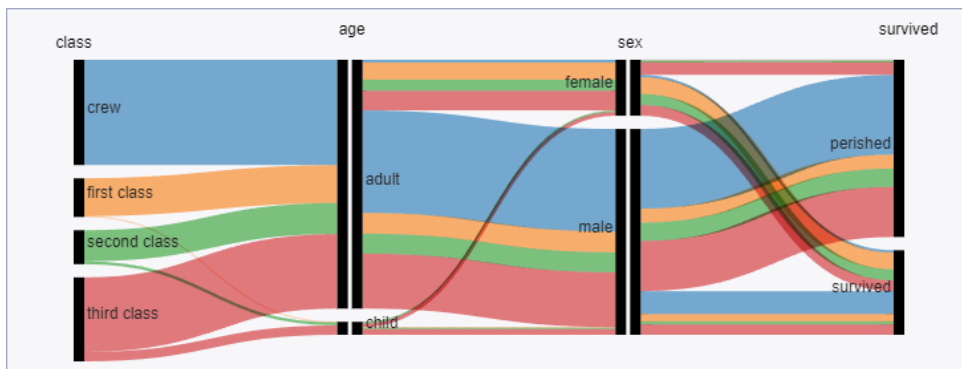
```

**Listing 6.5:** Dataset structure for car data for parallel dataset.

All attributes of these new datasets are considered nominal, since they have fewer than 10 distinguishable values.

Our only representative of the visualization of nominal attributes is *parallel sets* in a *bundled* layout variant. The view also takes the list of possible display attributes, in this case nominal. The user can again choose which of these he wants to show and in what order. It is also possible to choose which one of the same attributes to use for coloring, if any. The margins, opacities, and category colors can be set in the same manner as in the quantitative views.

Parallel sets can be seen in Figure 6.5. Here we can see that the class of the passenger was used for coloring. Attribute values are sorted by number, if they are numerical, or alphabetically otherwise. We can also choose whether the brushing will be displayed over the links or in their top part. There are more styling options such as width of the tabs, their spacing or gap between them. It is also possible to change the color of the inner font.



**Figure 6.5:** Parallel sets in bundled layout, displaying the Titanic dataset.

### 6.1.3 Brushing Connected Views

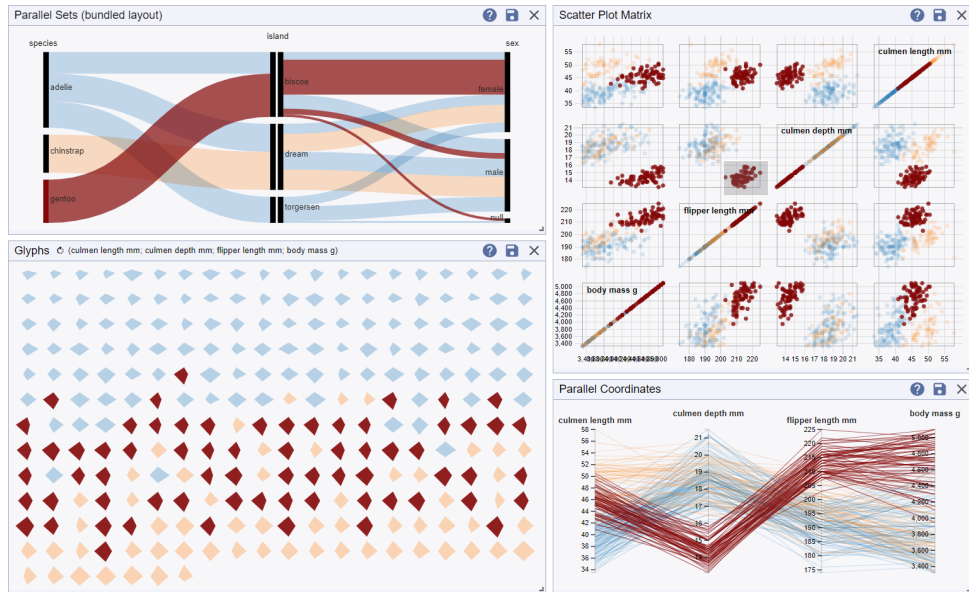
All of the aforementioned methods allow us to use brushing individually, as previously described in Section 4.3 *Connected Views and Brushing*.

Now, we will focus on the results of connecting the views so that brushing in one view is shown in the others. Since there is always at most one leading view that currently brushes, the brushing elements (extents) can be seen only in this view. These elements are not present if the brushing is clickable

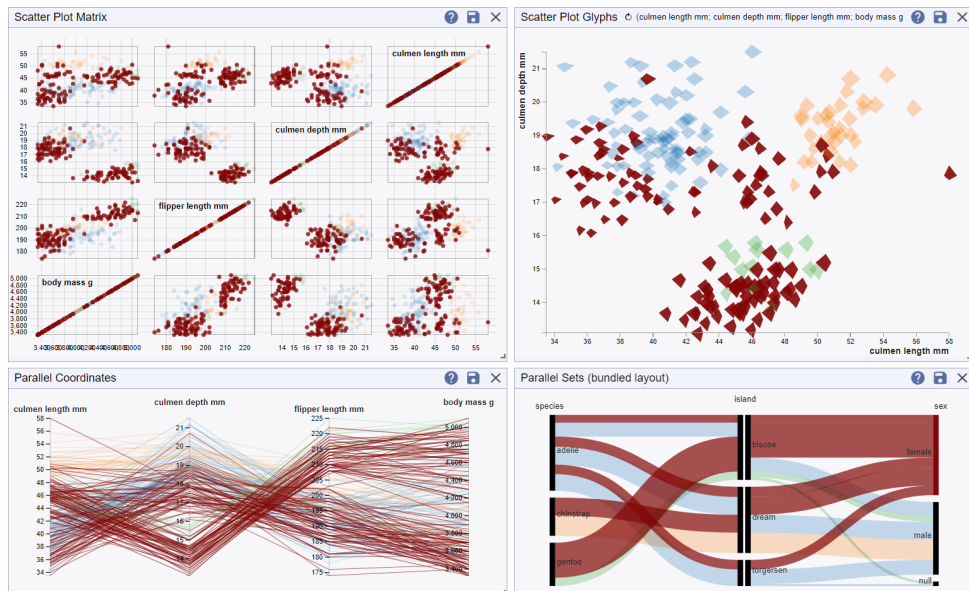
## 6. Results

(glyphs in line and parallel sets). In the other views, selected items are highlighted and the items that have not been selected are suppressed.

I added some examples of connected views while brushing in Figures 6.6, 6.7 and 6.8.



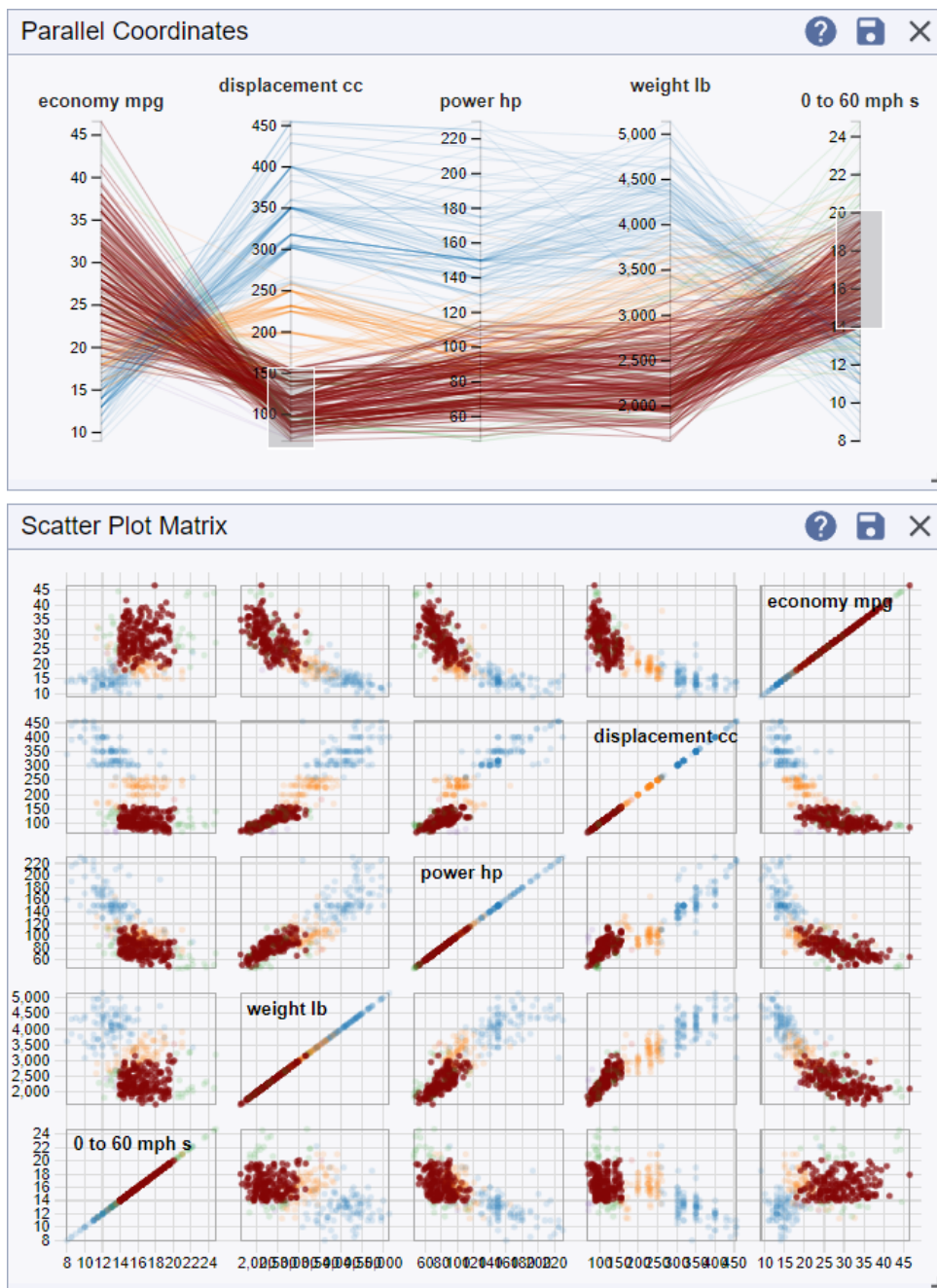
**Figure 6.6:** Brushing in *scatter plot matrix* projected on parallel sets, glyphs and parallel coordinates (using the bird dataset).



**Figure 6.7:** Brushing in *parallel sets* projected on scatter plot matrix, parallel coordinates and glyphs in a scatter plot (using the bird dataset).

Brushing is also connected with the data table. The data table behaves similarly to other views but is not rendered as an SVG. It is also possible to





**Figure 6.8:** Brushing in *parallel coordinates* on two axes, projected on scatter plot matrix (using the car dataset, without the attributes year and cylinders).

choose the attributes it shows and in what order but no longer distinguish whether they are quantitative, nominal, or other. It also allows for sorting and filtering by the values of these attributes. The brushing with the data table can be seen in Figure 6.9.



**Figure 6.9:** Brushing in *table* with filters projected in parallel sets (using the car parallel dataset, without economy and year attributes).

## 6.2 Meeting the Requirements

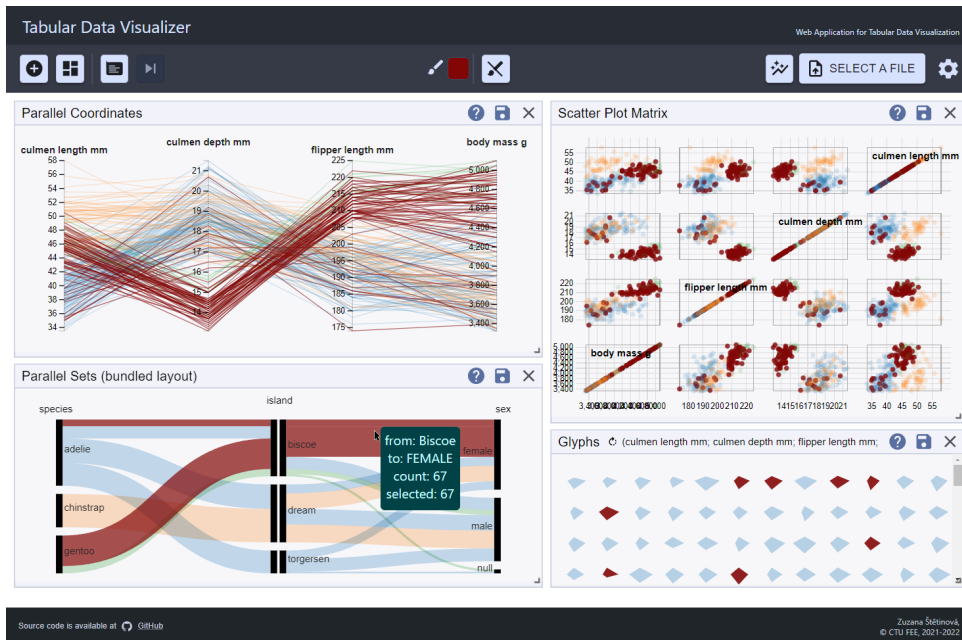
In this chapter, I would like to briefly describe whether the application created as part of this thesis meets all the requirements set in Section 3.1 *Requirements*.

As mentioned above, all visualization methods that have been identified as necessary were implemented. The application allows the user to select any of these methods and add them to the workspace. All these views allow brushing and are connected to the others. Views are located in a grid layout and can be moved or resized. The application also provides several preset layouts. The view has various basic and stylistic settings, the display attributes can be reordered or hidden. All the colors that the application uses can be changed by the user, the categories in the settings, and the brush in the top toolbar. The application also provides sample datasets and allows the user to use his own CSV or JSON files.

As for the requirements that have not yet been mentioned, the application can display details on hover in the form of a tooltip over the data element.

This option can be turned on or off in the top toolbar. Saving to the SVG file can be found in the header of each visualization view. I also consider all non-functional requirements to be met, at least to the required extent.

The resulting application is shown in Figures 6.10 and 6.11



**Figure 6.10:** The appearance of the resulting application. The tooltip is displayed after hovering over the data.



**Figure 6.11:** The resulting application with open settings.

### 6.3 Options of Further Extension

The implementation of the application reached its first full version in this thesis. However, there are still many possible improvements and enhancements that I have come across in research or implementation.

First of all, I would like to bring up what was also mentioned in the requirements of the thesis – adding other views. There are still many other methods for visualizing various  $n$ -dimensional tabular data. These methods can be integrated into the application, and thus expand the user's application capabilities. As an example, the tree layout for parallel sets should be considered.

For current visualizations, it would be beneficial for the user to be able to change the order of values for parallel sets. Currently, only alphabetical order is used, but if the data attribute is *ordered*, different ordering would be more suitable. We could also prevent overcrossing.

It is also possible to add more options to the user, such as changing the axis names.


With these additions, the application would also become more complex, which could lead to another requirement, namely, to allow the state of the application to be saved so that the user can return to it even after closing the browser.

The user could also be allowed to use multiple brushes to view multiple selections at a glance, which now requires multiple browser windows.

The application could also have other extension possibilities that were not mentioned in this thesis. For example, selected statistical methods could be added to decide on visualization parameters rather than the user.

### 6.4 Chapter Summary

In this chapter, I have described the present and future of the implemented application. I have focused on how the individual views look like and on how connected views are displayed during brushing. Regarding the future, I have described possible extensions of the application.



## Chapter 7

### Conclusion

The objective of this thesis was to conduct research on tabular data and existing applications used for their visualization and subsequently design and implement the basis for an application that serves for visualization of  $n$ -dimensional tabular data.

As part of this master's thesis, I prepared a detailed analysis of tabular data, its possible mappings to attributes and methods used for  $n$ -dimensional tabular data. I also mentioned the possibilities of using connected views and interactions in visualization methods.

Furthermore, I determined the requirements and conducted research first on existing applications that visualize  $n$ -dimensional tabular data, and because none met the requirements, I focused on selecting a library that provides the basic functionalities needed to create visualizations in a web environment.

I continued with the design of the application, where I focused mainly on working with data, how the individual visualization methods work, and the connection of views on these methods and brushing.

Based on the design, I developed an application as part of this thesis. The resulting application serves as an extensible tool to visualize tabular data in the Web browser. It works only on the client side, so no server is needed to run it. The application was tested on five datasets, which had from 150 to 2201 items.

Next, I described the implementation of the application, including the technologies used, such as React, TypeScript, and D3.js, and the structure of the source code. Here, I also mentioned the deployment of the application and the usage of tools to maintain the quality of the code.

Finally, I summarized the results and analyzed how the application will develop further.

The next steps in application development are described in Section *6.3 Options of Further Extension*. The application implemented was designed to support extensibility, so it assumes that it will be the basis for future work of other students who would be interested in related topics.

The application will also serve as a visualization tool for students of the Visualization course while discussing the topic of  $n$ -dimensional tabular data.

The benefit of this thesis is a deeper understanding of the principles of  $n$ -dimensional tabular data visualization (and visualization in general) and

## 7. Conclusion

---

going through the process of designing an application that allows visualization of such data. The work on this thesis also taught me the basics of using the interesting and popular D3.js visualization library.



## Bibliography

- [1] *Microsoft Excel Spreadsheet Software* [online]. Redmond, Washington, US: Microsoft Corporation, 2022 [cit. 2022-05-13]. Available at: <https://www.microsoft.com/en-us/microsoft-365/excel>
- [2] *Google Sheets* [online]. Mountain View, California, US: Google, 2022 [cit. 2022-05-13]. Available at: <https://www.google.com/sheets/about/>
- [3] ANSCOMBE, F. J. Graphs in Statistical Analysis. *The American Statistician*. 1973, **27**(1). ISSN 00031305. Available at: doi:10.2307/2682899
- [4] MUNZNER, Tamara. *Visualization Design and Analysis*. Boca Raton, Florida, US: CRC Press, Taylor & Francis Group, 2015. ISBN 978-1-4665-0893-4.
- [5] QUINLAN, Philip T. and Glyn W. HUMPHREYS. *Visual search for targets defined by combinations of color, shape, and size: An examination of the task constraints on feature and conjunction searches*. 1987, **41**(5), 455-472. ISSN 0031-5117. Available at: doi:10.3758/BF03203039
- [6] HATTAB, Georges, Theresa-Marie RHYNE, Dominik HEIDER and Scott MARKEL. Ten simple rules to colorize biological data visualization. *PLOS Computational Biology*. 2020, **16**(10). ISSN 1553-7358. Available at: doi:10.1371/journal.pcbi.1008259
- [7] TUCHKOV, Ivan. Color blindness: how to design an accessible user interface. *UX Collective* [online]. 2018 [cit. 2022-01-02]. Available at: <https://uxdesign.cc/color-blindness-in-user-interfaces-66c27331b858>
- [8] FLÜCK, Daniel. *Color Blind Essentials* [online]. Switzerland: Colblindor, 2010 [cit. 2022-01-07]. Available at: <https://www.color-blindness.com/wp-content/documents/Color-Blind-Essentials.pdf>
- [9] TORY, M. and T. MOLLER. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics*. 2004, **10**(1), 72-84. ISSN 1077-2626. Available at: doi:10.1109/TVCG.2004.1260759

- [10] SHNEIDERMAN, B. The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*. IEEE Comput. Soc. Press, 1996, 336-343. ISBN 0-8186-7508-X. Available at: doi:10.1109/VL.1996.545307
- [11] WARD, Matthew O. Linking and Brushing. *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009, 2009, 1623-1626. ISBN 978-0-387-35544-3. Available at: doi:10.1007/978-0-387-39940-9\_1129
- [12] Scatter Plot Matrix. *NIST – National Institute of Standards and Technology* [online]. Gaithersburg [cit. 2022-05-13]. Available at: <https://www.itl.nist.gov/div898/handbook/eda/section3/scatplma.htm>
- [13] VENNA, Jarkko. *Dimensionality Reduction for Visual Exploration of Similarity Structures*. Finland: Helsinki University of Technology, 2007. ISBN 978-951-22-8752-9.
- [14] MAGUIRE, Eamonn, Philippe ROCCA-SERRA, Susanna-Assunta SANSONE, Jim DAVIES and Min CHEN. Taxonomy-Based Glyph Design – with a Case Study on Visualizing Workflows of Biological Experiments. *IEEE Transactions on Visualization and Computer Graphics*. 2012, **18**(12), 2603-2612. ISSN 1077-2626. Available at: doi:10.1109/TVCG.2012.271
- [15] JACKLE, Dominik, Johannes FUCHS and Daniel A. KEIM. Star Glyph Insets for Overview Preservation of Multivariate Data. *IS&T International Symposium on Electronic Imaging*. University of Konstanz; Konstanz, Germany, 2016, 9. Available at: doi:10.2352/ISSN.2470-1173.2016.1.VDA-506
- [16] PICKETT, R.M. and G.G. GRINSTEIN. Iconographic Displays For Visualizing Multidimensional Data. *Proceedings of the 1988 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 1988, 514-519. ISBN 7-80003-039-3. Available at: doi:10.1109/ICSMC.1988.754351
- [17] CHERNOFF, Herman. The Use of Faces to Represent Points in K-Dimensional Space Graphically. *Journal of the American Statistical Association*. 1973, **68**(342). ISSN 01621459. Available at: doi:10.2307/2284077
- [18] INSELBERG, A. and B. DIMSDALE. Parallel coordinates: a tool for visualizing multi-dimensional geometry. *Proceedings of the First IEEE Conference on Visualization: Visualization '90*. IEEE Comput. Soc. Press, 1990, 361-378. ISBN 0-8186-2083-8. Available at: doi:10.1109/VISUAL.1990.146402
- [19] Parallel Coordinates Sample. *Wikimedia commons* [online]. Yug, 2015 [cit. 2022-01-02]. Available at:



[https://commons.wikimedia.org/wiki/File:Parallel\\_coordinates-sample.png](https://commons.wikimedia.org/wiki/File:Parallel_coordinates-sample.png)

- [20] KOSARA, R., F. BENDIX and H. HAUSER. Parallel Sets: interactive exploration and visual analysis of categorical data. *IEEE Transactions on Visualization and Computer Graphics*. 2006, **12**(4), 558-568. ISSN 1077-2626. Available at: doi:10.1109/TVCG.2006.76
- [21] GUERRA, John. Parallel Sets: Try your data. *Observable* [online]. 2019 [cit. 2022-05-14]. Available at: <https://observablehq.com/@john-guerra/parallel-sets?collection=@john-guerra/try-your-data>
- [22] THEUS, Martin. Mosaic plots. *WIREs Computational Statistics*. 2012, **4**(2), 191-198. ISSN 1939-5108. Available at: doi:10.1002/wics.1192
- [23] *Tableau* [online]. Seattle, WA: Tableau Software llc., 2022 [cit. 2022-01-08]. Available at: <https://www.tableau.com/>
- [24] Creating scatter plot matrix in Tableau. *EduPristine: Empowering Professionals* [online]. 2016 [cit. 2022-01-08]. Available at: <https://www.edupristine.com/blog/scatter-plot-matrix-in-tableau>
- [25] Spotfire® Analytics Accelerated. *TIBCO* [online]. Palo Alto, CA: TIBCO Software, 2022 [cit. 2022-01-08]. Available at: <https://www.tibco.com/products/tibco-spotfire>
- [26] WARD, Matthew. O., Elke A. RUNDENSTEINER, Kaiyu ZHAO et al. *XmdvTool Home Page* [online]. 2015 [cit. 2022-01-08]. Available at: <https://davis.wpi.edu/xmdv/>
- [27] WARD, Matthew. O., Elke A. RUNDENSTEINER, Kaiyu ZHAO et al. *XmdvTool GitHub* [online]. 2015 [cit. 2022-01-08]. Available at: <https://github.com/kaiyuzhao/XmdvTool>
- [28] Parallel coordinates: Car data. *Observable* [online]. 2019 [cit. 2022-01-08]. Available at: <https://observablehq.com/@d3/parallel-coordinates#data>
- [29] LAWRENCE, Michael, Eric HARE and Hadley WICKHAM. Ggobi. *GitHub* [online]. 2018 [cit. 2022-01-09]. Available at: <https://github.com/ggobi/ggobi>
- [30] KELLEY, Colin, Russell LANG, Dave KOTZ, John CAMPBELL et al. Gnuplot. *GitHub* [online]. [cit. 2022-01-09]. Available at: <https://github.com/gnuplot/gnuplot>
- [31] WALKE, Jordan *ReactJS* [online]. Meta Platforms, 2022 [cit. 2022-01-09]. Available at: <https://reactjs.org/>
- [32] *Recharts* [online]. Recharts Group, 2021 [cit. 2022-01-09]. Available at: <https://recharts.org/>



- [46] *Observable: Explore, analyze, and explain data. As a team.* [online]. Observable, 2021 [cit. 2022-01-14]. Available at: <https://observablehq.com/>
- [47] BOSTOCK, Michael. D3.js 7 documentation. *DevDocs* [online]. 2021 [cit. 2022-01-14]. Available at: <https://devdocs.io/d3/>
- [48] SHIPMAN, Kael, Mehul MOHAN. Hossam MAGDY et al. Javascript CSV Strings. *GitHub* [online]. MIT/X11, 2022 [cit. 2022-05-15]. Available at: <https://github.com/Inist-CNRS/node-csv-string>
- [49] React-Grid-Layout. *GitHub* [online]. MIT license, 2022 [cit. 2022-05-15]. Available at: <https://github.com/react-grid-layout/react-grid-layout>
- [50] *MUI* [online]. Material UI SAS., 2022 [cit. 2022-05-15]. Available at: <https://mui.com/>
- [51] *Material* [online]. Google, 2022 [cit. 2022-05-15]. Available at: <https://material.io/>
- [52] ŠTĚTINOVÁ, Zuzana. Tabular Data Visualization: source code. *GitHub* [online]. Praha, [cit. 2022-05-13]. Available at: <https://github.com/stetizu1/tabular-data-visualization>
- [53] ŠTĚTINOVÁ, Zuzana. Tabular Data Visualization: deployment. *GitHub pages* [online]. Praha, [cit. 2022-05-13]. Available at: <https://stetizu1.github.io/tabular-data-visualization/>
- [54] DAHL, Ryan. *Node.js* [online]. OpenJS Foundation, 2021 [cit. 2022-01-14]. Available at: <https://nodejs.org/en/>
- [55] *Yarn – Package manager* [online]. Meta Platforms, 2022 [cit. 2022-01-14]. Available at: <https://yarnpkg.com/>
- [56] *Npm* [online]. npm, 2022 [cit. 2022-05-15]. Available at: <https://www.npmjs.com/>
- [57] *ESLint* [online]. OpenJS Foundation, 2021 [cit. 2022-01-14]. Available at: <https://eslint.org/>
- [58] *Prettier* [online]. MIT license, 2022 [cit. 2022-05-15]. Available at: <https://prettier.io/>
- [59] *Chrome Developers.* [online]. Google Developers, 2022 [cit. 2022-01-14]. Available at: <https://developer.chrome.com/>



## Appendix A

### User Interface Description

The application provides three basic control elements:

- top toolbar,
- right-side settings drawer and
- the view controls.

The *top toolbar* can be seen in Figure A.1. It provides basic functionalities to work with the whole application, having an effect on all views.



**Figure A.1:** Top toolbar.

In the left section, from left to right, we can see a button for adding a view, button opening layout dialog that allows the user to choose one of the predefined layouts, tooltip toggle that is controlling if views have tooltips with data item info while hovering, and lastly the toggle for real-time brushing during the selection. This is an optimization for range brushes (scatter plots and parallel coordinates), which allows the user to calculate the brushing only when the mouse button is released.

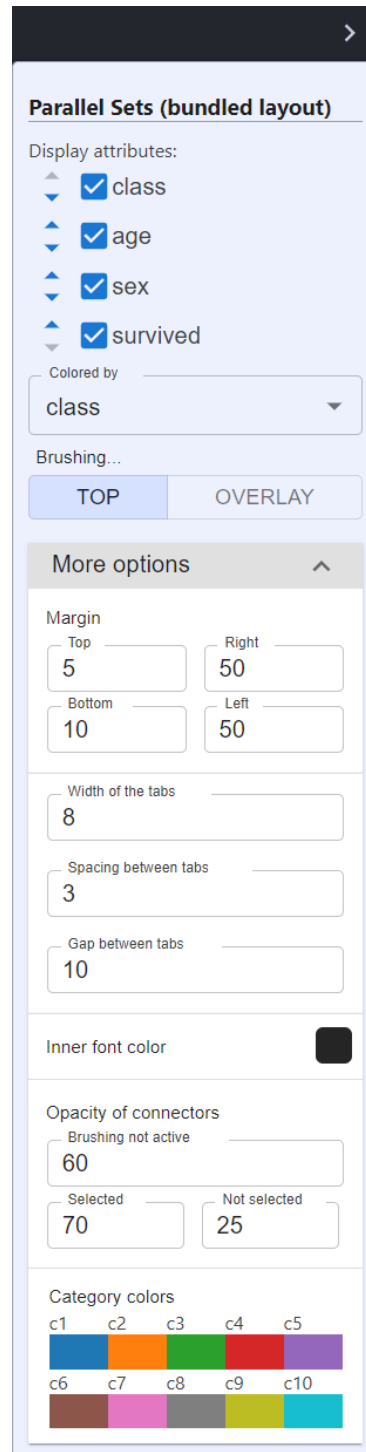
In the middle section, there are two buttons to control the brushing, the brush color picker button on the left and the cancel brushing button on the right, which removes all the brushing that is currently active.

The right section contains (left to right) the button to select the sample dataset, the button to load a custom dataset, and finally the button to open the settings drawer.

This *data drawer* contains all the settings for each view displayed. The demonstration of such a drawer with settings of a single view can be seen in Figure A.2. The settings affect only the view to which it belongs.

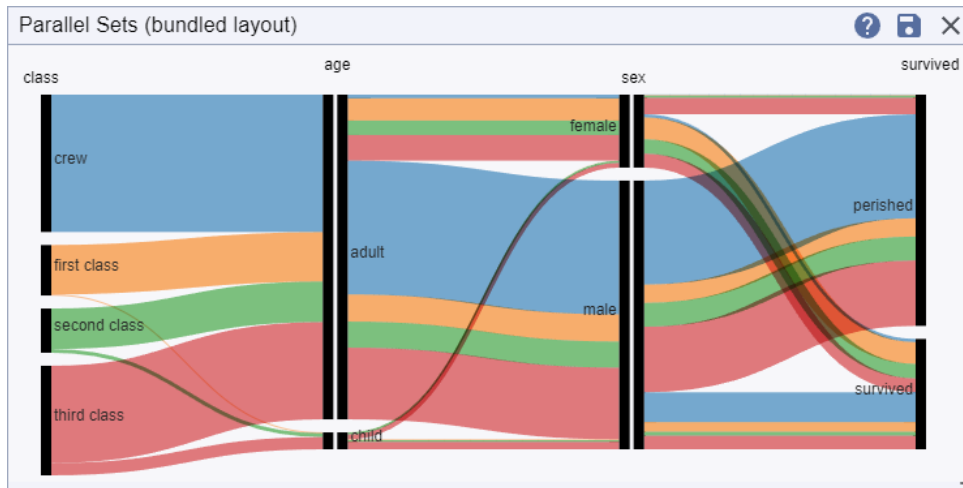
In the example, we can see the back button on the top of the drawer that hides it. In the *Parallel Sets* settings, we can see the 'display attributes' that can be moved up or down (changing the order) and checked/unchecked (hide). Below that, there is a selection box for the coloring attribute and a toggle with exclusive selection for the brushing type.

Next, there is the 'More options' section wrapped in an accordion component that can be shown or hidden. Inside it, we can set additional styling that is considered less important. It consists of number and color inputs for different styling changes of the view such as item sizes, colors, gaps, or opacity.



**Figure A.2:** Drawer with settings of a single view.

View controls are part of the grid item and they do not change the view inside. The header of the view can be used to move the grid item inside of the layout and the handle in the lower right corner allows user to resize the item. There are also three buttons in the top right corner. The first button on the left is the help button, which will display a dialog with all the information about the view. Next, we have the save button, which allows the user to save the current state of the view as an SVG file. In the case of the data table, it is replaced by the filter button, which displays filters below the table header. The last is the button to remove the view from the layout.



**Figure A.3:** View controls.

Brushing can be done by clicking on items in the view (glyphs, parallel sets) or dragging over the view (scatter plot matrix, scatter plot glyphs) or axes (parallel coordinates).







## Appendix B

### List of Attachments

- Application source code
- Original LaTeX package for the thesis