Steerable Parallel Coordinates in JavaScript

Project Report

Philipp Drescher, Jeremias Kleinschuster, Sebastian Schreiner, Burim Vrella

706.057 Information Visualisation 3VU SS 2023
Graz University of Technology

16 Jun 2023

Abstract

Explorable explainers can be used to convey complex ideas in a more interactive way. The aim of this project was to create an scrollytelling explorable explainer for parallel coordinates. To be able to do so, a prototype steerable parallel coordinates implementation was built in JavaScript, which can be steered and controlled from outside via exposed methods. The project serves as a basis for future work expanding the functionality of the library.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

An Explorable Explainer in JavaScript

Explorable explainers can aid in understanding complex topics, by guiding the user through an interactive narrative. The goal of this project was to implement an explorable in JavaScript to explain the usage of parallel coordinates. Scrollytelling was used to convey the narrative and steer the parallel coordinates plot.

To be able to interact with the plot from the outside, the library needs to expose certain functionalities like inverting, hiding, and reordering dimensions. These functions can then be used by the explorable implementation to guide the user through the narrative. In addition to the narrative being able to steer the visualisation using exposed functions, the user should also be able to interact with the plot directly.

1.1 Dataset

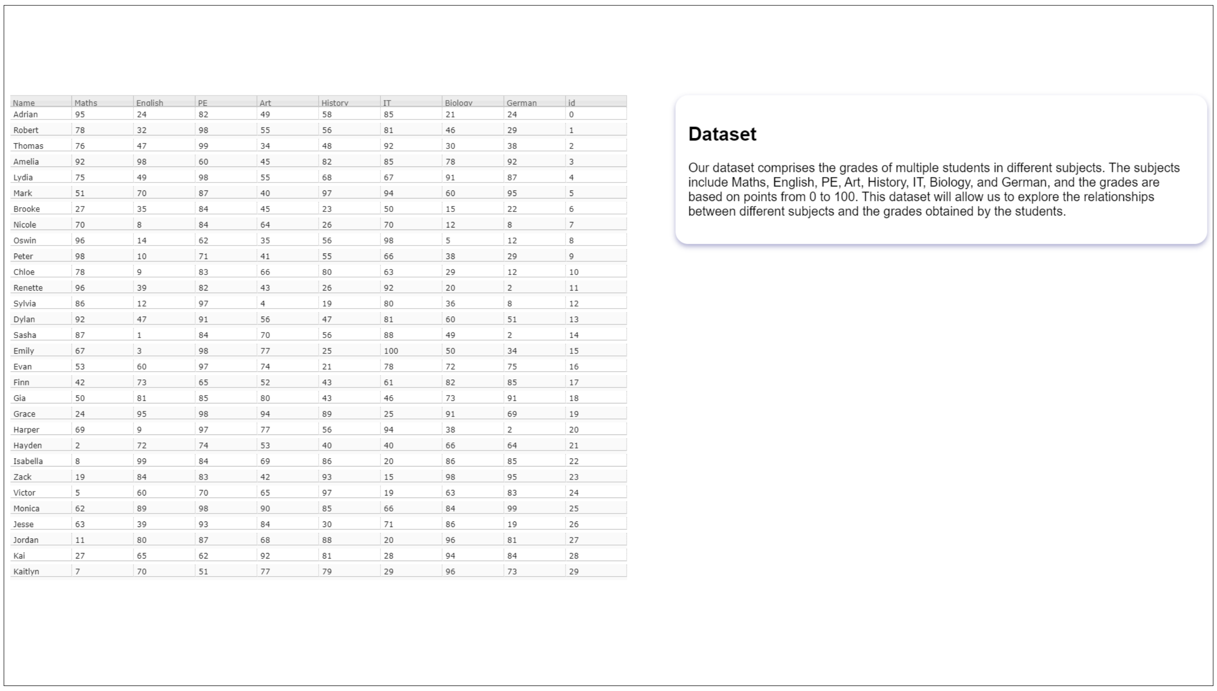
The explorable explainer should guide the user through a narrative to illustrate the usage of parallel coordinates. The main dataset used to illustrate parallel coordinates consists of fictitious marks (from 0 to 100) for 30 students in 8 subjects. It is shown in Table 1.1. Each record (row) represents a student, and each dimension (column) represents a subject. The dataset is carefully curated to contain some outliers and some records with strong correlations between different subjects to make them more apparent in the plot and therefore easier to see by the reader.

Name	Maths	English	PE	Art	History	IT	Biology	German
Adrian	95	24	82	49	58	85	21	24
Robert	78	32	98	55	56	81	46	29
Thomas	76	47	99	34	48	92	30	38
Amelia	92	98	60	45	82	85	78	92
Lydia	75	49	98	55	68	67	91	87
Mark	51	70	87	40	97	94	60	95
Brooke	27	35	84	45	23	50	15	22
Nicole	70	8	84	64	26	70	12	8
Oswin	96	14	62	35	56	98	5	12
Peter	98	10	71	41	55	66	38	29
Chloe	78	9	83	66	80	63	29	12
Renette	96	39	82	43	26	92	20	2
Sylvia	86	12	97	4	19	80	36	8
Dylan	92	47	91	56	47	81	60	51
Sasha	87	1	84	70	56	88	49	2
Emily	67	3	98	77	25	100	50	34
Evan	53	60	97	74	21	78	72	75
Finn	42	73	65	52	43	61	82	85
Gia	50	81	85	80	43	46	73	91
Grace	24	95	98	94	89	25	91	69
Harper	69	9	97	77	56	94	38	2
Hayden	2	72	74	53	40	40	66	64
Isabella	8	99	84	69	86	20	86	85
Zack	19	84	83	42	93	15	98	95
Victor	5	60	70	65	97	19	63	83
Monica	62	89	98	90	85	66	84	99
Jesse	63	39	93	84	30	71	86	19
Jordan	11	80	87	68	88	20	96	81
Kai	27	65	62	92	81	28	94	84
Kaitlyn	7	70	51	77	79	29	96	73

Table 1.1: Curated dataset of marks (from 0 to 100) for 30 students in 8 subjects, used to illustrate parallel coordinates.

1.2 Narrative

The narrative itself starts with a brief introduction to the topic and then proceeds to illustrate why a traditional table is not optimal to spot correlations in a dataset, as can be seen in Figure 1.1. The narrative proceeds with an introduction to parallel coordinates, using just 2 of the 8 dimensions, as shown in Figure 1.2. Scrolling further, the narrative then displays 4 of the 8 dimensions and explains how correlations can only be spotted between neighbouring dimensions, so sometimes dimensions have to be rearranged to place dimensions of interest next to one another. This is shown in Figure 1.3. It is often easier to see correlations if one of the neighbouring dimensions is inverted. The final section of the explorable is shown in Figure 1.4. The visualisation displays all 8 dimensions of the dataset and allows the reader to explore the dataset in its entirety and experiment with parallel coordinates.

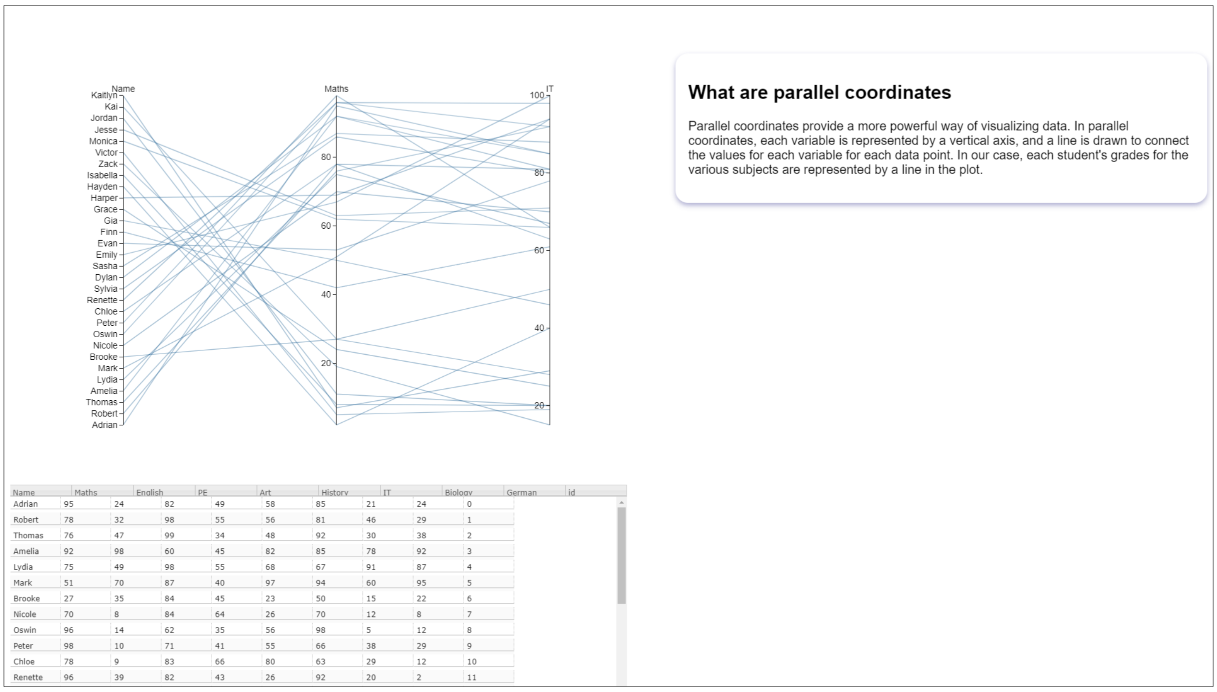


Name	Maths	English	PE	Art	History	IT	Biology	German	id
Adrian	95	24	82	49	58	85	21	24	0
Robert	78	32	98	55	56	81	46	29	1
Thomas	76	47	99	34	48	92	30	38	2
Amelia	92	98	60	45	82	85	78	92	3
Lydia	75	49	98	55	68	67	91	87	4
Mark	51	70	87	40	97	94	60	95	5
Brooke	27	35	84	45	23	50	15	22	6
Nicole	70	8	84	64	26	70	12	8	7
Oswin	96	14	62	35	56	98	5	12	8
Peter	98	10	71	41	55	66	38	29	9
Chloe	78	9	83	66	80	63	29	12	10
Renette	96	39	82	43	26	92	20	2	11
Sylvia	86	12	97	4	19	80	36	8	12
Dylan	92	47	91	56	47	81	60	51	13
Sasha	87	1	84	70	56	88	49	2	14
Emily	67	3	98	77	25	100	50	34	15
Evan	53	60	97	74	21	78	72	75	16
Finn	42	73	65	52	43	61	82	85	17
Gia	50	81	85	80	43	46	73	91	18
Grace	24	95	98	94	89	25	91	69	19
Harper	69	9	97	77	56	94	38	2	20
Hayden	2	72	74	53	40	40	66	64	21
Isabella	8	99	84	69	86	20	86	85	22
Zack	19	84	83	42	93	15	98	95	23
Victor	5	60	70	65	97	19	63	83	24
Monica	62	89	98	90	85	66	84	99	25
Jesse	63	39	93	84	30	71	86	19	26
Jordan	11	80	87	68	88	20	96	81	27
Kai	27	65	62	92	81	28	94	84	28
Kaitlyn	7	70	51	77	79	29	96	73	29

Dataset

Our dataset comprises the grades of multiple students in different subjects. The subjects include Maths, English, PE, Art, History, IT, Biology, and German, and the grades are based on points from 0 to 100. This dataset will allow us to explore the relationships between different subjects and the grades obtained by the students.

Figure 1.1: The explorable explainer first shows the dataset in tabular form to illustrate that is is difficult to see correlations.

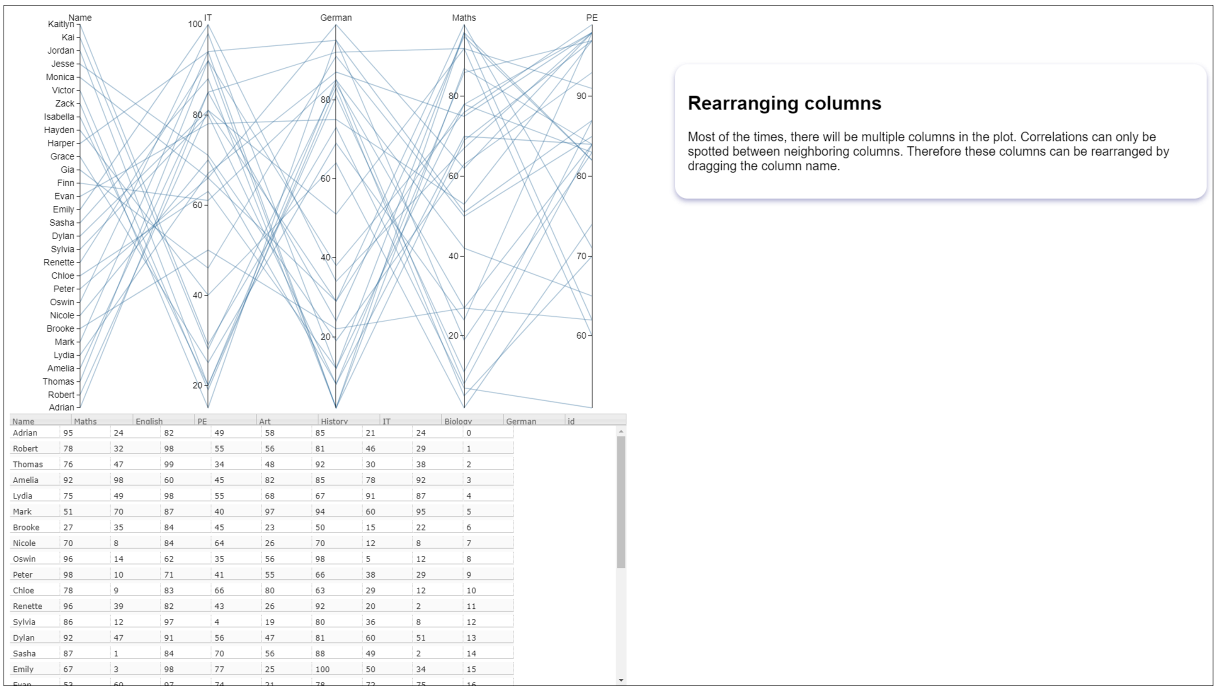


What are parallel coordinates

Parallel coordinates provide a more powerful way of visualizing data. In parallel coordinates, each variable is represented by a vertical axis, and a line is drawn to connect the values for each variable for each data point. In our case, each student's grades for the various subjects are represented by a line in the plot.

Name	Maths	English	PE	Art	History	IT	Biology	German	id
Adrian	95	24	82	49	58	85	21	24	0
Robert	78	32	98	55	56	81	46	29	1
Thomas	76	47	99	34	48	92	30	38	2
Amelia	92	98	60	45	82	85	78	92	3
Lydia	75	49	98	55	68	67	91	87	4
Mark	51	70	87	40	97	94	60	95	5
Brooke	27	35	84	45	23	50	15	22	6
Nicole	70	8	84	64	26	70	12	8	7
Oswin	96	14	62	35	56	98	5	12	8
Peter	98	10	71	41	55	66	38	29	9
Chloe	78	9	83	66	80	63	29	12	10
Renette	96	39	82	43	26	92	20	2	11

Figure 1.2: Next, the explorable introduces parallel coordinates.

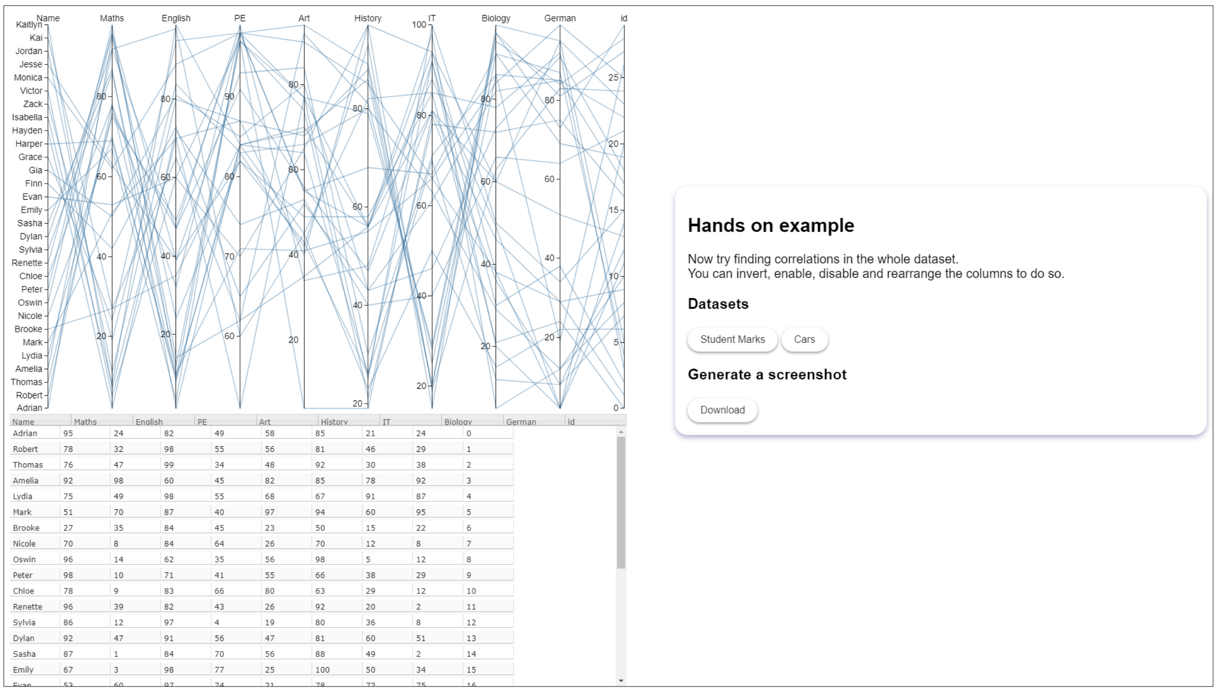


Rearranging columns

Most of the times, there will be multiple columns in the plot. Correlations can only be spotted between neighboring columns. Therefore these columns can be rearranged by dragging the column name.

Name	Maths	English	PE	Art	History	IT	Biology	German	id
Adrian	95	24	82	49	58	85	21	24	0
Robert	78	32	98	55	56	81	46	29	1
Thomas	76	47	99	34	48	92	30	38	2
Amelia	92	98	60	45	82	85	78	92	3
Lydia	75	49	98	55	68	67	91	87	4
Mark	51	70	87	40	97	94	60	95	5
Brooke	27	35	84	45	23	50	15	22	6
Nicole	70	8	84	64	26	70	12	8	7
Oswin	96	14	62	35	56	98	5	12	8
Peter	98	10	71	41	55	66	38	29	9
Chloe	78	9	83	66	80	63	29	12	10
Renette	96	39	82	43	26	92	20	2	11
Sylvia	86	12	97	4	19	80	36	8	12
Dylan	92	47	91	56	47	81	60	51	13
Sasha	87	1	84	70	56	88	49	2	14
Emily	67	3	98	77	25	100	50	34	15
Evan	53	60	97	74	21	78	72	75	16

Figure 1.3: The explainer proceeds to explain how and why to rearrange dimensions in a parallel coordinates plot.



Hands on example

Now try finding correlations in the whole dataset.
You can invert, enable, disable and rearrange the columns to do so.

Datasets

Student Marks Cars

Generate a screenshot

Download

Name	Maths	English	PE	Art	History	IT	Biology	German	id
Adrian	95	24	82	49	58	85	21	24	0
Robert	78	32	98	55	56	81	46	29	1
Thomas	76	47	99	34	48	92	30	38	2
Amelia	92	98	60	45	82	85	78	92	3
Lydia	75	49	98	55	68	67	91	87	4
Mark	51	70	87	40	97	94	60	95	5
Brooke	27	35	84	45	23	50	15	22	6
Nicole	70	8	84	64	26	70	12	8	7
Oswin	96	14	62	35	56	98	5	12	8
Peter	98	10	71	41	55	66	38	29	9
Chloe	78	9	83	66	80	63	29	12	10
Renette	96	39	82	43	26	92	20	2	11
Sylvia	86	12	97	4	19	80	36	8	12
Dylan	92	47	91	56	47	81	60	51	13
Sasha	87	1	84	70	56	88	49	2	14
Emily	67	3	98	77	25	100	50	34	15
Evan	53	60	97	74	21	78	72	75	16

Figure 1.4: Finally, the explorable allows the user to explore the full dataset with parallel coordinates.

1.3 ParCoords Library

For the explorable explainer, an already existing JavaScript library by Chang [2019] was used. Although it uses the outdated v3 of D3 for drawing, the library offers all the required functionality to be used in the context of an explorable explainer. Notably, the library provides features such as brushing, dimension inversion, and reordering. These features were accessible through both exposed functions for programmatic control and through direct user interaction.

Loading different datasets was relatively straightforward to implement. This was used to allow the user to explore parallel coordinates by using different datasets in different contexts. In addition to the interactive parallel coordinates plot, a table view was added to the explorable explainer. This functionality was already provided by the library itself, meaning only little additional extra work was needed to incorporate it into the explorable explainer. This integration provides users with an additional layer of insight into the underlying data set.

1.4 ScrollMagic

Scrollytelling in the explorable explainer was implemented using the ScrollMagic JavaScript library [Paepke 2023]. One of the key features offered by ScrollMagic is the ability to trigger specific functions based on the user's current scroll position. This enables an explorable explainer to invoke appropriate actions or transitions based on the reader's progress. By defining specific scroll triggers and associated functions, different elements within the explorable explainer can be manipulated or animated.

Additionally, the ScrollMagic library offers the option to pin HTML elements. When an element is pinned, it remains fixed at a certain position on the screen regardless of further scrolling. This functionality is particularly useful in the context of an explorable explainer, as it allows the parallel coordinates plot to stay in a specific location while the narrative continues to progress.

There is also the possibility to change the CSS class of given HTML elements using ScrollMagic triggers. This offers the possibility to emphasise certain aspects of the explorable by incorporating, for instance, animations or style changes based on triggers.

Using this library involves the creation of a *controller* to which different *scenes* can be added. A *scene* includes a trigger element and some logic whioch is executed if the trigger is invoked. To demonstrate the usage of ScrollMagic, consider the example shown in Listing 1.1. Here, the explorable explainer aimed to showcase the inversion of the *Maths* dimension within the *parcoords* object.

```
var controller = new ScrollMagic.Controller();

new ScrollMagic.Scene({ triggerHook: 0, triggerElement: '#card5' })
  .on('leave', function (event) {
    flipMaths();
  })
  .addTo(controller);

function flipMaths() {
  parcoords.animationTime(500);
  parcoords.flip(["Maths"]);
  parcoords.render().updateAxes();
}
```

Listing 1.1: Using ScrollMagic to invoke a function to invert the *Maths* dimension.

1.5 Project Structure

The structure of files and folders in the project was driven by three main factors: readability, single responsibility, and maintainability. By dividing the code base into smaller, more focused files, the overall readability of the project significantly improved. Listing 1.2 shows the structure of the project. Each file contains clear and concise funtionality, making it easier for developers to understand and navigate the code base.

The `css/` folder holds all style sheets used in the explorable. The `card.css` file is responsible for the different narrative entries. The `column.css` file specifies how a column is laid out, since the explorable is built in the form of a two-column layout. Finally, the `style.css` file holds general styling like font and colours.

The `data/` folder holds various `.csv` files used as the data source for the parallel coordinates plot. The `img/` folder contains all images used in the explorable. Currently, there is only a placeholder image present which is used in the initial view of the explorable. All files implementing some behaviour are located in the `scripts/` folder. Additionally, there is a `lib/` folder for external libraries such as d3.js and the parallel coordinates library.

The `main.js` file handles general functionality such as the initial loading of the data set. The file `parcoords_driver.js` is responsible for creating the parallel coordinates object used in the explorable. The two remaining files, namely `scrollmagic_helper.js` and `scrollmagic_trigger.js`, are mainly responsible for interaction with the parallel coordinates object. All triggers are defined in `scrollmagic_trigger.js`. These triggers use functions implemented in `scrollmagic_helper.js`, which directly accesses the parallel coordinates object to manipulate the parallel coordinates plot. The main HTML source file is `index.html`.

```
Name                          | Type
-----------------------------------------------------
- css                         | Folder
  - card.css                  | css source file
  - column.css                | css source file
  - style.css                 | css source file
- data                        | folder
  - cars.csv                  | data
  - student-marks_v2.csv      | data
- img                         | folder
  - placeholder.png           | image
- scripts                     | folder
  - lib                       | folder
  - main.js                   | javaScript source file
  - paracoords_driver.js      | javaScript source file
  - scrollmagic_helper.js     | javaScript source file
  - scrollmagic_trigger.js    | javaScript source file
- index.html                  | html source file
```

Listing 1.2: Project structure of the D3 standalone.

Chapter 2

Investigating Pluto

In addition to the ScrollMagic implementation, the possibility of utilising Pluto [fonsp 2023] for an explorable explainer was investigated. Pluto is an interactive programming tool built on top of the Julia programming language. It provides a user-friendly notebook interface which enables users to explore, experiment, and document their code in an interactive environment. Notably, Pluto.jl was developed alongside a free online course at MIT, making it exceptionally well-suited for interactive teaching and learning experiences.

2.1 Julia

As explained by Bezanson et al. [2017], Julia is a versatile and dynamic programming language known for its high-performance computing capabilities. It offers an easy-to-use syntax similar to Python, combined with the computational efficiency of lower-level languages like C, while still being a dynamically typed language. With Julia, users can perform complex numerical computations and data analysis with remarkable speed and precision. In addition, the system makes it possible to recreate the same environment on different systems. Julia was developed as an open-source project with currently over 1000 contributors on GitHub [Bezanson et al. 2023].

2.2 Setup and Execution

To get started with Julia and Pluto, the following steps need to be taken:

1. *Install Julia*: Since Pluto is built upon Julia, Julia needs to be installed first. The most recent version suitable for every system can be obtained from the Julia website (`julialang.org`). After downloading, Julia just needs to be installed.

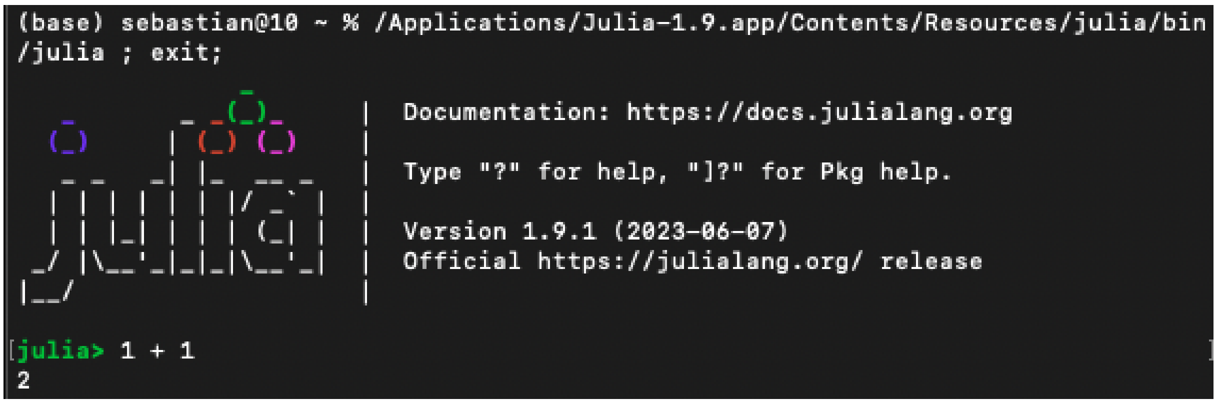
2. *Run Julia*: After installation, Julia can be run by either directly launching the Julia application or by executing the command `julia` in the terminal, depending on the operating system. The Julia terminal should open, where basic Julia code can be executed. An example of a Julia terminal can be seen in Figure 2.1.

3. *Install Pluto*: In the Julia terminal, use the Julia package manager to install Pluto:

   ```
   import Pkg; Pkg.add("Pluto")
   ```

4. *Run Pluto*: Once Pluto is installed successfully, it can be hosted by executing the following command. This will enable access to Pluto through the browser via localhost:

   ```
   import Pluto; Pluto.run()
   ```



```
(base) sebastian@10 ~ % /Applications/Julia-1.9.app/Contents/Resources/julia/bin
/julia ; exit;
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.9.1 (2023-06-07)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> 1 + 1
2
```

Figure 2.1: Example terminal output after installation of Julia.

The command also automatically opens the correct address in the standard browser. Pluto can be stopped again by pressing "Ctrl + C" in the Julia terminal.

2.3 Pluto Features

Pluto has many important and useful features for various use cases:

- *Reactivity*: Pluto introduces the concept of reactive cells, where changes in one cell can trigger updates in other, dependent cells. This allows for a more dynamic and responsive programming workflow, as modifications to code or data are immediately reflected in the notebook by updating the results. To be more efficient, Pluto selectively re-evaluates only the affected cells when changes occur.

- *HTML Interaction*: It is possible to use the `@bind` macro to ensure a binding between Julia variables and HTML objects. Combined with the reactivity of Pluto, this is a very useful tool to make variables easily changeable. The Pluto package already provides simple inputs like sliders and buttons. Further widgets can be added using HTML, CSS, and JavaScript.

- *Reproducibility*: Pluto automatically checks which packages are being used in a notebook and keeps track of them by managing a package environment. This package environment is stored in the notebook file to reproduce the exact environment when opening the file in Pluto again. The notebook can be saved as either a notebook file (Julia file), a HTML file (which can be hosted), or a static PDF file.

- *Speed*: Since it is highly efficient, Julia is perfectly suited to handling complex operations. Using this language, Pluto achieves impressive speed and maintains seamless interactivity, while enabling users to program in a script-like manner.

2.4 Remarks

Pluto is a notebook-style programming environment, it is perfectly suited to build an explorable explainer. The result would be similar to an explorable built in Jupyter with Python. With an implementation in Pluto, users could interactively explore parallel coordinates in a step-by-step approach. Additionally, Pluto implementations have the advantage of being easily shared and reproduced.

However, unlike a JavaScript explorable with ScrollMagic, it would not be feasible to create a scrollable explorable for parallel coordinates within a Pluto environment. An implementation of parallel coordinates is available in the PlutoPlotly package, but it has some limitations and is not steerable.

Chapter 3

Steerable Parallel Coordinates (SPC)

The parallel coordinates library used to build the prototype explorable described in Chapter 1 was built several years ago with the now outdated version v3 of the D3 library. To proceed with the project, it was decided to concentrate efforts on developing a new steerable parallel coordinates (SPC) library with the latest version v7 of D3. This could then be used in a future implementation of a parallel coordinates explorable explainer.

3.1 Project Structure

The files of the new SPC library implementation are divided among two main folders, `lib/` and `src/`, as shown in Listing 3.1. The `src/` folder contains the folder `scripts/` where the `parallelcoordinates.ts` TypeScript *TypeScript* [2023] source file is located. The SPC library is implemented in the file `parallelcoordinates.js`, which is generated by compiling the TypeScript source file.

The `lib` folder structure has two sub-folders `example/` and `plugins/` and contains the `parallelcoordinates.js` library. The `example/` folder contains all necessary files for a small showcase including an HTML file, a stylesheet, a dataset, and some driver code. All third-party libraries are downloaded and stored locally in the `plugins/` folder. These libraries include the current version of D3 *D3 Library* [2023] which is used to draw the parallel coordinates plot. The other two libraries, `popper.min.js` and `tippy-bundle.umd.js`, are used to display the tooltips.

The two remaining files, `tsconfig.json` and `webpack.config.js`, are responsible for specifying how the library should be built. The file `tsconfig.json`, shown in Listing 3.2, controls how the SPC TypeScript source code is transpiled into JavaScript. It is divided into three sections: `compilerOptions`, `include`, and `exclude`. In the first section, `compilerOptions`, the compiler is given various parameters to take into account during compilation. These include the output directory and the target JavaScript *JavaScript* [2023] version. This project uses version es2022 *ES2022* [2023] for compilation, which is currently the most recent available version.

The file `webpack.config.js`, shown in Listing 3.3, controls the bundling of external libraries. Libraries marked as `externals` are expected to be imported by the user. Therefore these libraries will not be bundled and shipped together with the SPC library. In the example implementation discussed in Section 3.3 these libraries are included in the header of the index.html file.

```
Name                          | Type
-----------------------------------------------------
- lib                         | Folder
  - example                   | Folder
    - data                    | Folder
      - student-marks-v2.csv| Data
    - index.html              | HTML source file
    - main.js                 | JavaScript source file
    - style.css               | CSS source file
  - plugins                   | Folder
    - d3.v7.js                | JavaScript source file
    - popper.min.js           | JavaScript source file
    - tippy-bundle.umd.js     | JavaScript source file
  - parallelcoordinates.js    | JavaScript source file
- src                         | Folder
  - scripts                   | Folder
    - parallelcoordinates.ts| TypeScript source file
tsconfig.json                 | TypeScript config file
webpack.config.js             | Webpack config file
```

Listing 3.1: Project structure of the SPC library.

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es2022",
    "sourceMap": false,
    "outDir": "lib",
    "pretty": true,
    "downlevelIteration" : true
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    "node_modules"
  ]
}
```

Listing 3.2: The `tsconfig` file used to configure TypeScript.

```
module.exports = {
  externals: {
    d3: 'd3',
    tippy: 'tippy'
  },
};
```

Listing 3.3: The Webpack config used to specify the bundling of external libraries.

```
Member functions
-----------------------------------
- loadCSV(csv)
- getData()
- setFeatures(newFeatures)
- invert(dimension)
- getInversionStatus(dimension)
- move(dimension, toRightOf, A)
- getDimensionPositions()
- getFilter(dimension)
- setFilter(dimension)
- getSelected()
- select(records)
- saveAsSVG()
- generateSVG()
```

Listing 3.4: Member functions of the SPC class, which are callable from outside.

3.2 Library Overview

The main characteristic of a *steerable* parallel coordinates library is that it has exposed methods which can be used to control and steer the plot from outside. The methods are shown in Listing 3.4. However, only five of them are currently implemented:

- **`loadCSV(csv)`**: (implemented)

  The `loadCSV` member function expects a CSV file as plain text. The text then gets parsed and stored as an array, which then serves as the data from which the parallel coordinates plot will be generated.

- **`getData()`**: (implemented)

  The main purpose of this function is, to have a simple way of retrieving the already parsed data. Therefore `getData()` serves as a getter function to retrieve the `data` member variable from the parallel coordinates object.

- **`setDimensions(newDimensions)`**: (implemented)

  With the `setDimensions` function the user can tell the parallel coordinate object which dimensions should be visible and which not. This is done via the parameter `newDimensions` which is an array that includes all the dimension names that the user wants to be able to see. In the example, this functionality is illustrated by enabling the user to select and deselect the dimensions by utilising checkboxes.

- **`invert(dimension)`**: (implemented)

  The invert function expects the name of the dimensions which should be inverted as a string. The dimension names correspond to the column names of the CSV file.

- **`generateSVG()`**: (implemented)

  This function serves as the core for drawing the plot. It generates the different dimensions including the ticks and the titles. It also generates a line for each row in the data set. Another main responsibility is to bind all event handlers used to interact with the plot. These include `onDrag` handlers for rearranging columns, `onClick` handlers for inverting them, as well as `onClick` and `onDrag` handlers to implement the brushing functionality. The `mouseover` callback is bound per data entry and is used to highlight the corresponding lines in another colour. Additionally, the data entry's name is displayed as a tooltip at the cursor's locations.

- **`getInversionStatus(dimension)`**: (not implemented)

  This function can be used as a way to retrieve the current status of a dimension which could either be inverted or non-inverted. The dimensions name should be provided as a string.

- **`move(dimension, toRightOf, A)`**: (not implemented)

  The purpose of this function is to reposition a dimension in a parallel coordinates plot. The target dimension is specified using the `A` parameter, and the `toRightOf` parameter determines whether the dimension should be positioned to the left or right of it.

- **`getDimensionPositions()`**: (not implemented)

  This function should return a list of the dimensions in the plot ordered corresponding to the current layout.

- **`getFilter(dimension)`**: (not implemented)

  The `getFilter` function returns the current filtering as an interval with normalised values from zero to one. This has to be done because the dimensions range can vary depending on the input data. If no filtering is enabled, some default value should be returned to indicate this. This default value could be for instance -1 as this is not feasible as an interval.

- **`setFilter(dimension, from, to)`**: (not implemented)

  The function sets a filter for a dimension specified by the first parameter. The `from` and `to` parameters specify an interval that also uses normalised coordinates from zero to one. This enables the usage of filtering without actually knowing the dimensions scale, which depends on the data.

- **`getSelected()`**: (not implemented)

  With this function, all currently selected paths and their information can be retrieved. This function returns a map with the id of the path and the information of the record it belongs to.

- **`select(records)`**: (not implemented)

  The `select` function expects a list of records as a parameter. The provided records should then be highlighted in another colour. Additionally, three states should be implemented to reflect this behaviour. These are namely `neutral`, `hovered`, and `selected`. `Neutral` is the default state meaning that the records are coloured in the default colour. If the user hovers over one or multiple records, the state should change to `hovered` to reflect this. Additionally, a tooltip could be shown and the hovered records could be coloured differently. The `selected` can be used to create a list of curated records.

- **`saveAsSVG()`**: (not implemented)

  To be able to download the state of the parallel coordinates plot a function is needed to convert the drawn elements to an SVG file. To be able to do so, some kind of SVG printer needs to be implemented.

3.3 SPC Demonstrator

The current status of the SPC library is that it is capable of creating fully functional parallel coordinates plots. The user can analyse a dataset by moving, inverting, and filtering dimensions. The user can hover over a polyline to see information about the corresponding record. The user can also remove dimensions from the current plot and re-include them again. However, only five of the steerable methods have currently been implemented.

In order to show the usage of the SPC library, an example demonstrator application was built. It can be found in the folder `lib/example/`. Figure 3.1 shows the initial state of the demonstrator after uploading a dataset. A dataset can be uploaded using the Upload file button, which opens a file browser to select a CSV file from the local file system. The filtering functionality of SPC is illustrated in Figure 3.2. In this example, the lower scoring students in Maths have been filtered out. The user has full control over the filter range, meaning that the interval can be moved up and down on the dimensions, cleared, and shrunk and enlarged.

The highlighting of individual records by hovering is shown in Figure 3.3. In addition to the hovered record in red, a tooltip is displayed with more information. In this case, the tooltip shows the hovered student's name, making it easier for the user to find specific entries. Dimensions can be re-ordered by dragging and dropping a dimension, as shown in Figure 3.4.

The displayed dimensions can be determined by selecting or deselecting the checkboxes at the bottom of the page. Each checkbox corresponds to one dimension in the dataset. The checkboxes are dynamically generated after the dataset is uploaded. Figure 3.5 shows the parallel coordinates plot with the English and IT dimensions deselected. As illustrated in Figure 3.6, it is possible to invert dimensions interactively by clicking on the triangle above each dimension axis. It is also possible to invert a dimension by calling the `invert(dimension)` function of the SPC instance. The buttons at the bottom of the demonstrator page trigger the function for the corresponding dimension.

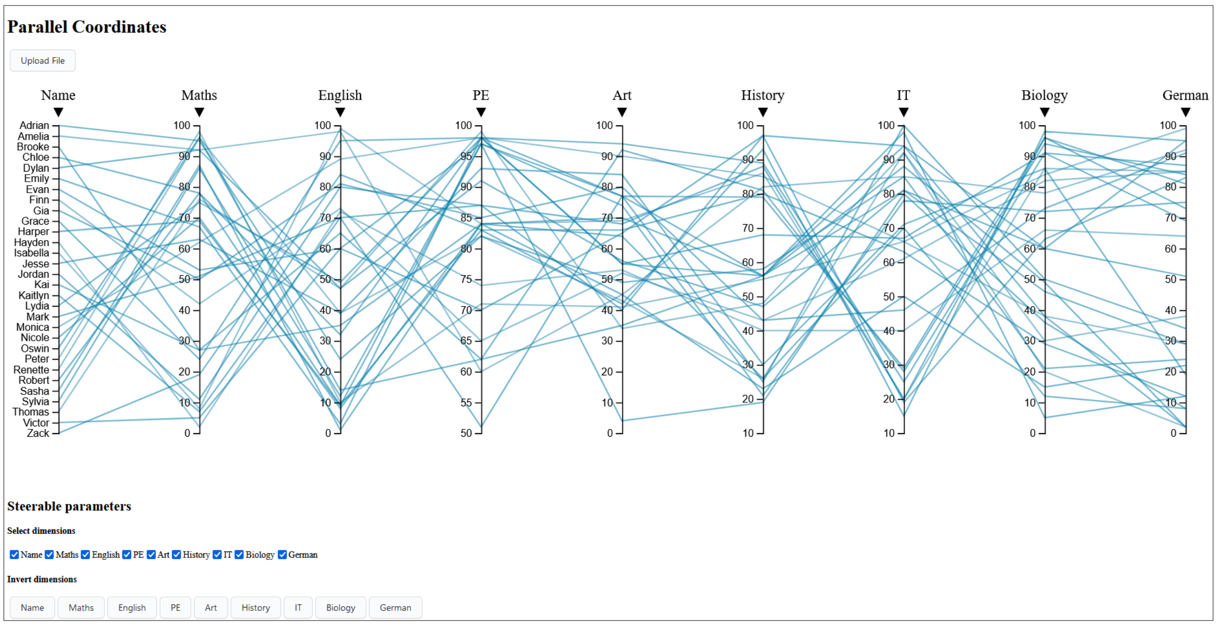


Figure 3.1: After uploading a dataset, a parallel coordinates plot is created displaying all available dimensions.

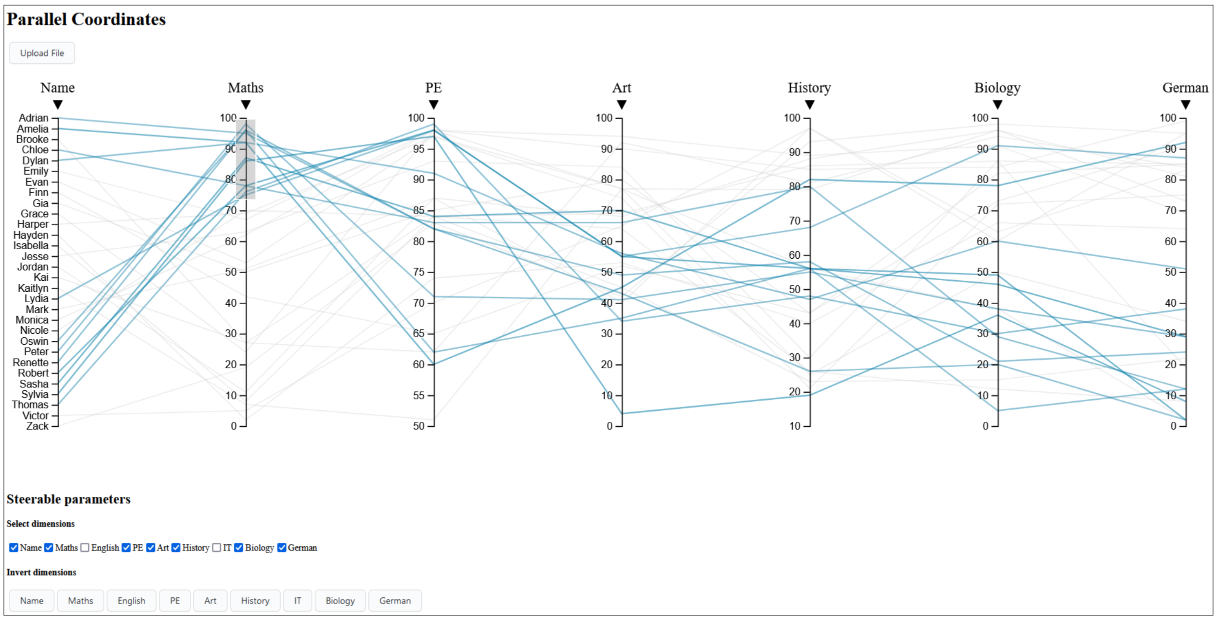


Figure 3.2: A filter can be applied to any dimension.

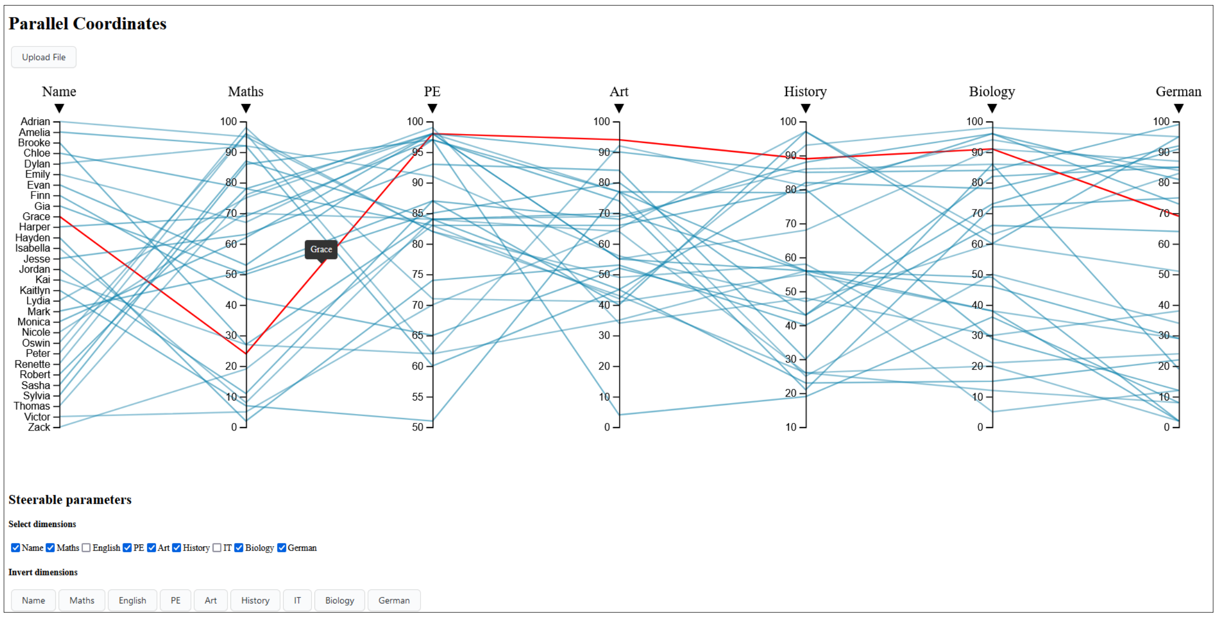


Figure 3.3: When hovering over a path the user can see the information of the current path.

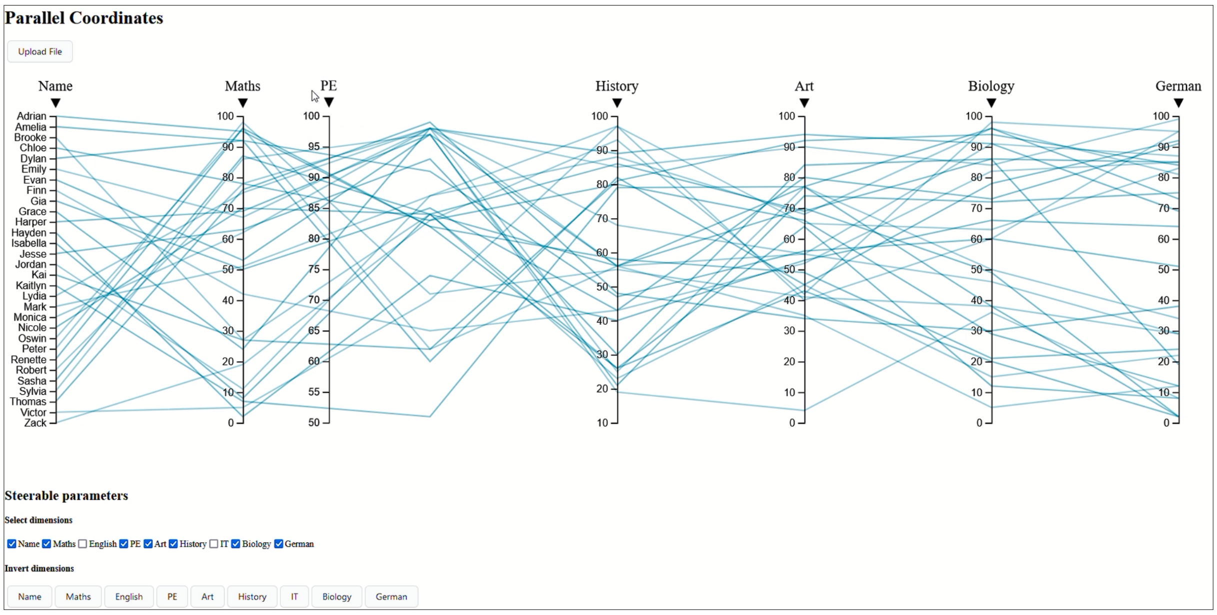
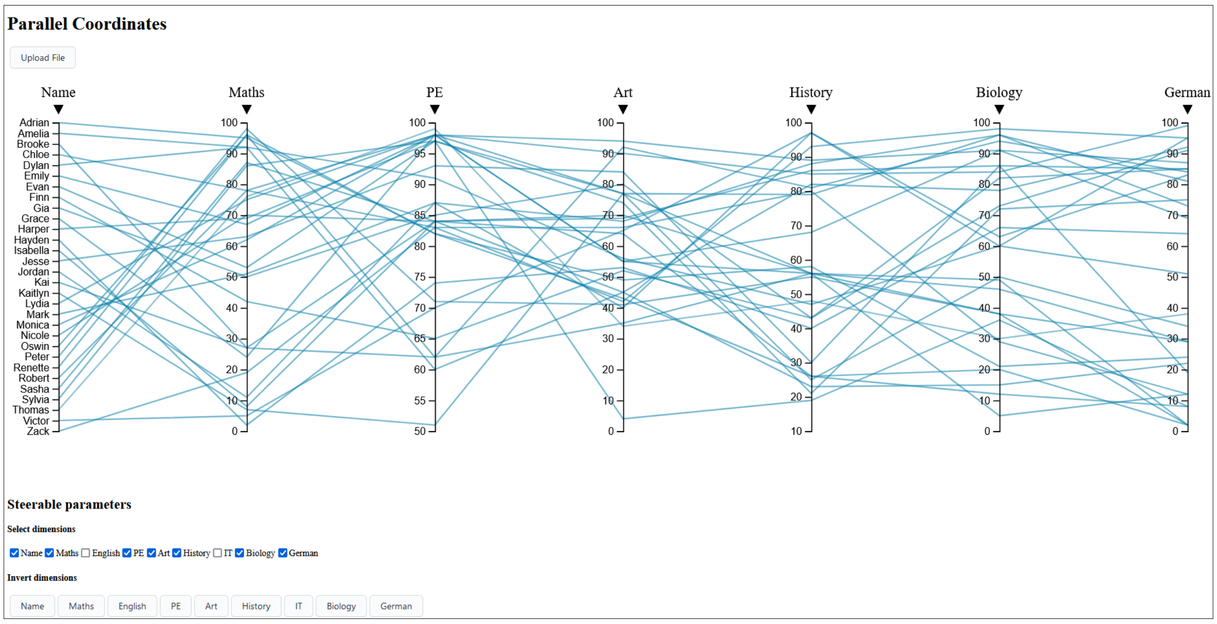


Figure 3.4: By clicking on a dimensions name and holding, one can move the dimensions to the desired place in the coordinate system.



Steerable parameters

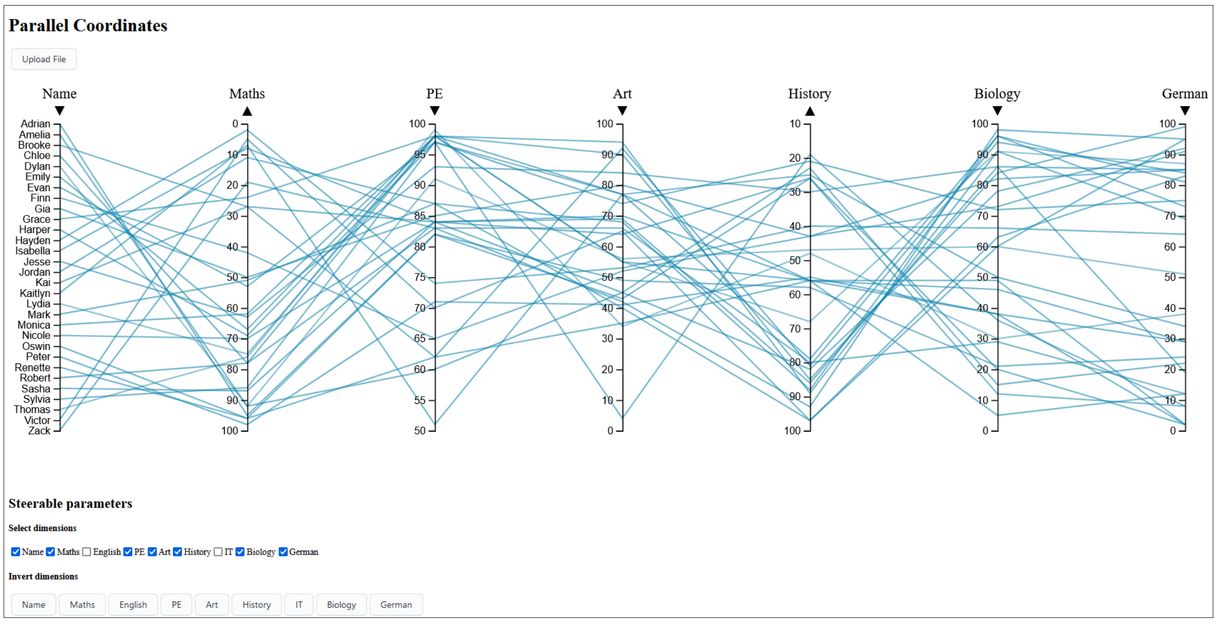
Select dimensions

☑ Name ☑ Maths ☐ English ☑ PE ☑ Art ☑ History ☐ IT ☑ Biology ☑ German

Invert dimensions

Name | Maths | English | PE | Art | History | IT | Biology | German

Figure 3.5: By selecting or deselecting the checkbox with the dimension's name, that dimension will be included in or excluded from the plot.



Steerable parameters

Select dimensions

☑ Name ☑ Maths ☐ English ☑ PE ☑ Art ☑ History ☐ IT ☑ Biology ☑ German

Invert dimensions

Name | Maths | English | PE | Art | History | IT | Biology | German

Figure 3.6: By clicking the triangle above a dimension's axis, the user can invert the dimension.

3.4 Library Packaging

There are several different ways to package JavaScript modules, each with their own advantages and disadvantages:

- *IIFE (Immediately Invoked Function Expression)*: An IIFE is a JavaScript function that is executed immediately after it is defined. It is typically used to create a separate scope for variables and functions to avoid polluting the global namespace. By wrapping code in an IIFE, it is possible to ensure that variables and functions defined within it do not conflict with those in the outer scope. An important characteristic of IIFE modules is that they are compatible with older browsers that may not support the latest JavaScript features or modules. Since IIFEs are a well-established pattern, they can be used in environments where modern module systems are not available or not well-supported.

- *CJS (CommonJS)*: CommonJS is a module system for JavaScript that was primarily designed for server-side environments, such as Node.js. It allows developers to organise their code into reusable modules using a `require()` function to import modules and `module.exports` or exports to export values from a module. CommonJS modules are synchronous, meaning that they are loaded and executed synchronously in the order specified.

- *AMD (Asynchronous Module Definition)*: AMD is a module system primarily used in client-side JavaScript applications. Unlike CommonJS, which loads modules synchronously, AMD loads modules asynchronously, which can improve performance in browser environments. AMD modules are defined using the `define(` function and can specify dependencies using an array notation. They are typically loaded on demand, allowing for better control over the loading and execution of modules.

- *UMD (Universal Module Definition)*: UMD is a module pattern that aims to provide compatibility between different module systems, including CommonJS, AMD, and globals (e.g., when using script elements on a web page). UMD modules use a combination of techniques to determine the available module system at runtime and adapt accordingly. While UMD allows modules to be used in different environments, it can be much more complex to write, because it needs to handle multiple scenarios.

- *ESM (ECMAScript Modules)*: ESM is the most recent official standard for JavaScript modules. It is built into the JavaScript language itself and provides a native module system. ESM provides the `import` and `export` keywords to define dependencies and exports between modules. This module system is widely supported in modern browsers and is commonly used in projects that leverage the latest JavaScript features. It is also commonly used in TypeScript applications, since TypeScript extends the functionality provided by ESM.

For a more thorough explanation Dixin's Blog has a wonderful overview of the different modules. *Dixin's Blog* [2023]

Chapter 4

Concluding Remarks

This project resulted in a proof-of-concept implementation for an explorable explainer for parallel coordinates using a scrollytelling approach. In addition, the base for a steerable parallel coordinates library was laid out in the form of a base class with some implemented methods, as described in Section 3.2. This class was built on the most recent version v7 of D3, and utilises the latest version of TypeScript to make it as future-proof as possible.

4.1 Shift of Focus

The primary issue that occurred during the project was the shift in focus from building an explainer to building a steerable parallel coordinates library. Initially, the objective was to implement a scrollable explorable explainer for parallel coordinates, utilising the fully functioning ScrollMagic implementation. However, this approach proved unsatisfactory due to the requirement of using the most recent technology. The first implementation relied on an older parallel coordinates library based on an old version of D3, which lead to a change in direction.

The aim of the project shifted towards developing a steerable parallel coordinates library with the most up-to-date version of D3. In addition to updating the library to use the most recent D3 version, it was crucial to structure the implementation's functionality into various small class methods to support steerability. The code was refactored with the idea of allowing users to instantiate a class object and utilise its methods to steer the plot from the outside. This was rather challenging, as the contexts, data, and events needed to work effectively within smaller structures, without interference between methods. Moreover, the new library was written in TypeScript for improved development processes and packaging capabilities.

Overall, there was a recurring need for code base refactoring and alterations, taking substantial time and resources within the project. Consequently, the resulting library remains unfinished to some extent. Nevertheless, it can serve as a starting point for future projects.

4.2 Outlook

The future plans for this project include implementing the remaining functions that are not yet implemented in the SPC library, as explained in Section 3.2. This is crucial in order to achieve a fully steerable parallel coordinates plot and make it usable in a proper explorable explainer.

Once this is done, the next step should presumably be to implement a proper SVG export by adding a secondary functionality to each of the functions that, instead of drawing the parallel coordinate system on screen, writes the svg code in a proper and legible manner to an SVG file.

To further improve usability and compatibility for users of the SPC library, various packaging methods could be explored and used to ship the library. Currently, the library is exported as commonJs, but many alternatives are available resulting in less overhead.

After this, it would be possible and desirable to implement a proper explorable explainer for parallel coordinates. While this is not anymore strictly in the scope of this project, it was nevertheless the starting point of this project and the steerable parallel coordinate library is tailor-made for such an endeavour. In this case, the proof of concept implementation described in Chapter 1 could serve as a great starting point.

Lastly, in order to use the SPC library in other projects with larger datasets, there are a number of improvements that could be made for the library, including bundling and drawing the parallel coordinates plot with WebGPU or WebGL for better performance.

Bibliography

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah [2017]. *Julia: A Fresh Approach to Numerical Computing*. SIAM Review 59.1 (2017), pages 65–98. doi:10.1137/141000671. https://doi.org/10.1137/141000671 (cited on page 9).

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah [2023]. *julia*. https://github.com/JuliaLang/julia (cited on page 9).

Chang, Kai [2019]. *parallel-coordinates*. 22 Oct 2019. https://github.com/syntagmatic/parallel-coordinates (cited on page 5).

D3 Library [2023]. https://d3js.org/ (cited on page 11).

Dixin's Blog [2023]. https://weblogs.asp.net/dixin/understanding-all-javascript-module-formats-and-tools (cited on page 19).

ES2022 [2023]. https://typescriptlang.org/docs/handbook/release-notes/typescript-4-5.html (cited on page 11).

fonsp [2023]. *Pluto*. https://github.com/fonsp/Pluto.jl (cited on page 9).

JavaScript [2023]. https://javascript.com/ (cited on page 11).

Paepke, Jan [2023]. *ScrollMagic*. https://scrollmagic.io/ (cited on page 5).

TypeScript [2023]. https://typescriptlang.org/ (cited on page 11).