Explorable Explainers

Survey Paper

Philipp Drescher, Jeremias Kleinschuster, Sebastian Schreiner, Burim Vrella

706.057 Information Visualisation 3VU SS 2023
Graz University of Technology

19 May 2023

Abstract

Explorable explainers seek to explain complex concepts and data using guided narrative linked to interactive visualizations. This survey paper provides an overview of the current state-of-the-art in explorable explainers by giving various examples. In addition, some popular software solutions were compared by implementing an explorable explainer about parallel coordinates in each of them.

© Copyright 2023 by the author(s), except as otherwise noted.

This work is placed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Explorable explainers support active reading by offering the reader the possibility to interact with the article itself [Victor 2011]. This means that an explorable informative medium offers some sort of interactive simulation which responds to user input. This enables users to play with the premise of the article and come up with their own assumptions, all while observing the changes immediately. In addition, an explorable explainer guides the reader through the article so as to provide a structure for the learning process. As Victor [2011] puts it, “An active reader asks questions, considers alternatives, questions assumptions, and even questions the trustworthiness of the author.”

Simpson’s Paradox is a classic example of an unintuitive problem, which can be more easily understood through an explorable explainer. At its core, it describes a fringe case of statistics, which is typically difficult for an unknowing observer to understand. The textbook case of Simpson’s Paradox is the story of a real life medical study [Charig et al. 1986]. In this study, it was found that two different treatments for kidney stones had a different effectiveness depending on the size of the kidney stones. The surprising part however was that, overall, treatment A was more effective than treatment B. However the study found that for large kidney stones treatment B was better. And, when looked at in this way, for small kidney stones treatment B was also better. This phenomenon is the essence of the Simpson’s paradox. An excellent explorable from Phillip Wacker details some examples and explains in great detail how Simpson’s Paradox works and why it is so unintuitive whenever it occurs [Wacker 2018].

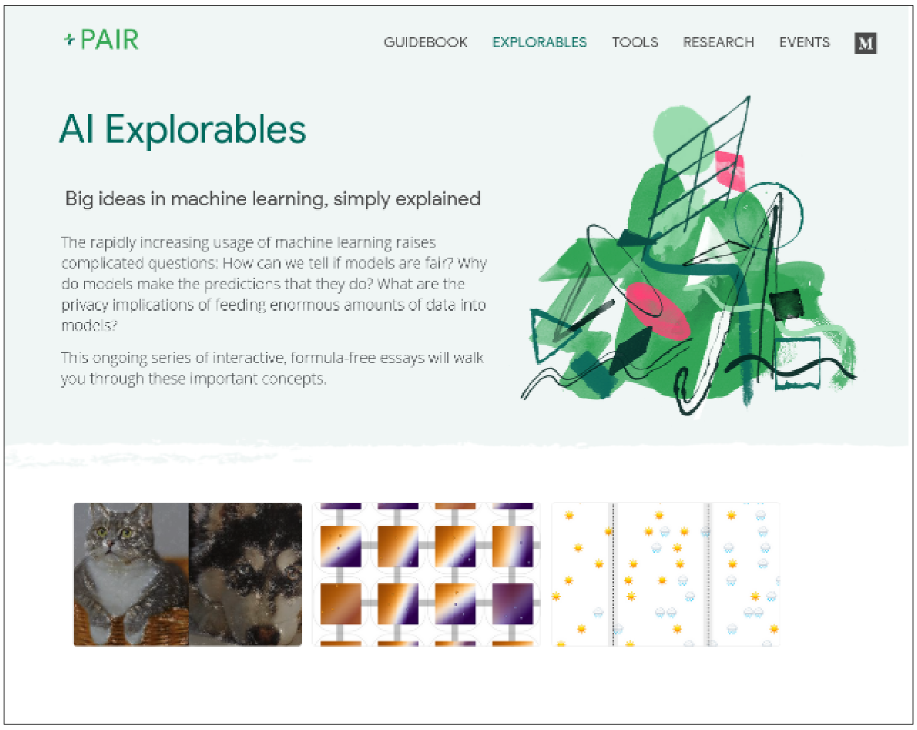
Chapter 2

Examples of Explorables

There are many different examples and websites which demonstrate how explorable explainers are created and utilized. This chapter presents some particulary good collections and examples.

2.1 Pair

Pair is a website with many interactive learning resources about artificial intelligence [Pair 2010]. It has a large collection of tools that can help people understand complicated concepts. These tools allow users to interact with and explore data, visualizations, and models to improve their understanding of AI. Pair is a great resource for people who want to learn more about AI and machine learning.



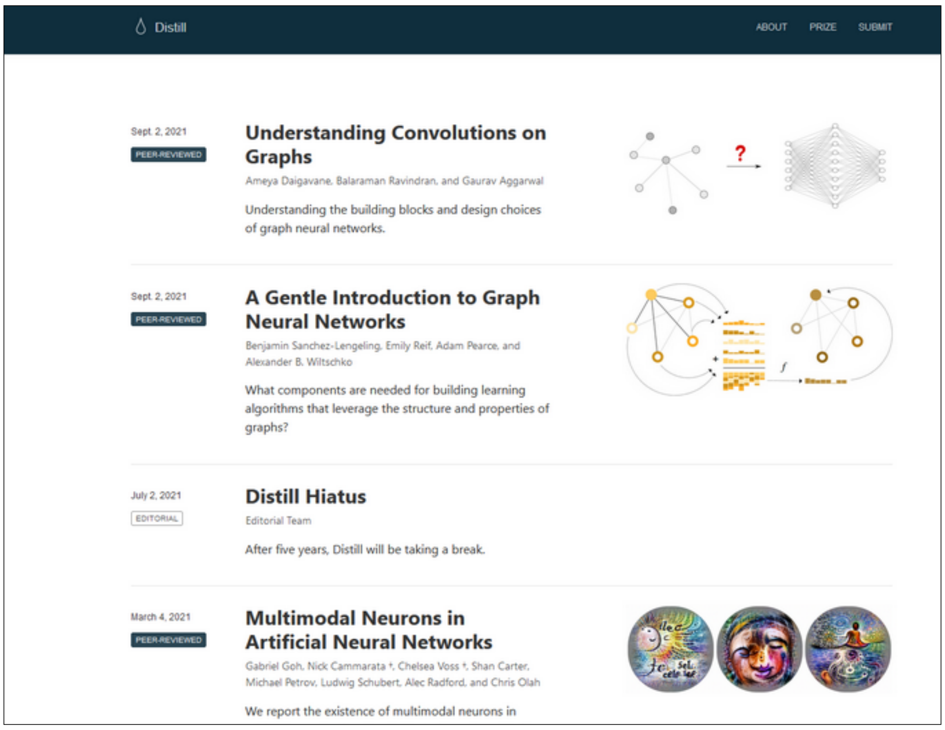
AI Explorables

Big ideas in machine learning, simply explained

The rapidly increasing usage of machine learning raises complicated questions: How can we tell if models are fair? Why do models make the predictions that they do? What are the privacy implications of feeding enormous amounts of data into models?

This ongoing series of interactive, formula-free essays will walk you through these important concepts.

Figure 2.1: Pair: A website hosting explorable explainers with a focus on artificial intelligence [Pair 2010].


Distill ABOUT PRIZE SUBMIT

Sept. 2, 2021
PEER-REVIEWED
Understanding Convolutions on Graphs
Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal
Understanding the building blocks and design choices of graph neural networks.

Sept. 2, 2021
PEER-REVIEWED
A Gentle Introduction to Graph Neural Networks
Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko
What components are needed for building learning algorithms that leverage the structure and properties of graphs?

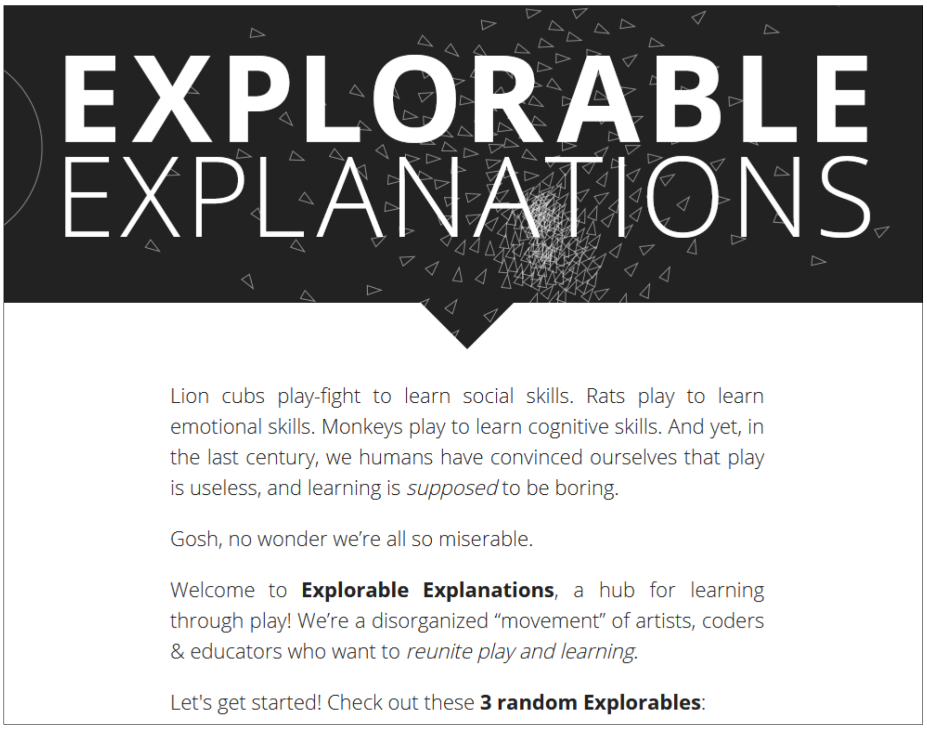
July 2, 2021
EDITORIAL
Distill Hiatus
Editorial Team
After five years, Distill will be taking a break.

March 4, 2021
PEER-REVIEWED
Multimodal Neurons in Artificial Neural Networks
Gabriel Goh, Nick Cammarata †, Chelsea Voss †, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah
We report the existence of multimodal neurons in

Figure 2.2: Distill: Hosts scientific papers as explorable explainers [Carter and Olah 2016].

2.2 Distill

Distill is an online scientific journal which hosts peer-reviewed papers in the form of explorable explainers [Carter and Olah 2016]. The website also offered a prize of $ 10,000 for outstanding work. Unfortunately, Distill was discontinued in 2021, but the content hosted on the website is still available.



Lion cubs play-fight to learn social skills. Rats play to learn emotional skills. Monkeys play to learn cognitive skills. And yet, in the last century, we humans have convinced ourselves that play is useless, and learning is *supposed* to be boring.

Gosh, no wonder we're all so miserable.

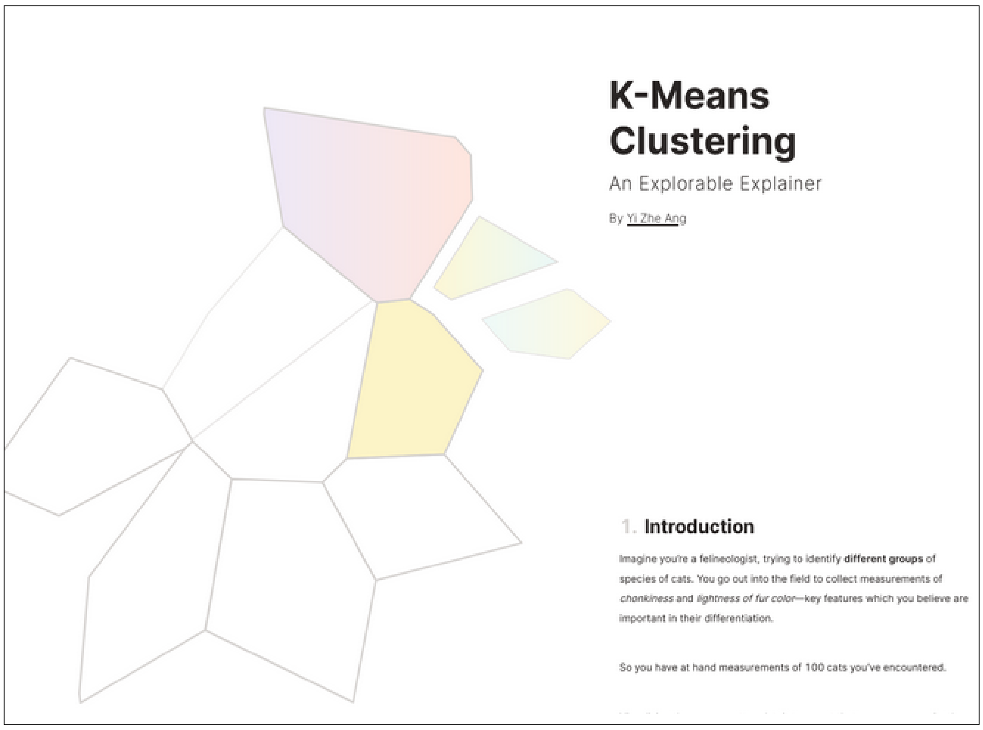
Welcome to **Explorable Explanations**, a hub for learning through play! We're a disorganized "movement" of artists, coders & educators who want to *reunite play and learning*.

Let's get started! Check out these **3 random Explorables**:

Figure 2.3: Explorable Explanations: Interactive learning platform with 180 engaging explanations [Case 2015].

2.3 Explorable Explanations

Explorable Explanations is a comprehensive web platform housing a collection of 180 interactive explorable explanations, along with articles and talks about this educational format [Case 2015]. The explorable explainers are categorized based on specific subjects, enabling users to explore various concepts through interactive experiences. The website also offers tutorials on creating explorable explainers and introduces tools that can facilitate the development of such interactive content.



K-Means Clustering

An Explorable Explainer

By Yi Zhe Ang

1. **Introduction**

Imagine you're a felineologist, trying to identify **different groups** of species of cats. You go out into the field to collect measurements of *chonkiness* and *lightness of fur color*—key features which you believe are important in their differentiation.

So you have at hand measurements of 100 cats you've encountered.

Figure 2.4: Interactive k-means explorable explainer by Yi Zhe Ang [Ang 2022].

2.4 K-Means Clustering

This explorable explainer by Yi Zhe Ang explains how the k-means clustering algorithm works and what it is used for [Ang 2022]. The explorable utilizes a scrolly-telling approach to walk the user through the narrative. The user can step through the algorithm step by step, visualizing the iterative optimization of k-means clustering. After that, the user is encouraged to try the algorithm on different datasets. This helps to visualize the limitations of the k-means approach for clustering.

This explorable is highly interactive and offers a very good explanation of the algorithm. The user is encouraged to play with the data, all while guided by the author in the form of a scrolly-telling narrative. It was awarded Best Submission at the VISxAI 2022 workshop.

Chapter 3

Tools

When it comes to creating explorable explainers, there are many tools available for different programming languages. Table 3.1 provides an overview of the range of toolkits available. The following chapters take a closer look at four of these tools: Jupyter, Observable, Shiny, and D3.

Tool	Programming Language	URL
Shiny	R	https://shiny.rstudio.com/
Jupyter	Julia, Python, R, C++, GNU Octave, Ruby, Scheme	https://jupyter.org/
Bokeh	Python	https://bokeh.org/
Vega-Altair	Python	https://altair-viz.github.io/
Plotly	Python	https://plotly.com/
Observable	Javascript, SQL	https://observablehq.com/
D3	Javascript	https://d3js.org/
Tangle	Javascript	http://worrydream.com/Tangle/
Joy.js	Javascript	https://ncase.me/joy/
Idyll	Javascript	https://idyll-lang.org
Highcharts	Javascript	https://www.highcharts.com/

Table 3.1: Overview of tools for creating explorable explainers.

Chapter 4

Jupyter Notebooks

Jupyter notebooks are a popular solution for interactable, web-based computing environments. They can contain text, code, equations, and visualizations making them relatively popular in fields such as data science and machine learning. The notebook itself is structured in code blocks, allowing the user to explore the code step by step. Naturally, Jupyter notebooks can be used to build interactive explorable explainers. Users can experiment and interact with the code snippets to gain more insight, enabling them to actively participate in the learning process.

4.1 Local Installation and Execution

Jupyter notebooks can be created and executed locally by settings up a Python development environment. After the successful creation of such an environment, Jupyter can be installed as a package by using pip with the following command:

```
pip install notebook
```

The next step is to create a .ipynb file which will serve as the notebook. This notebook can then be executed by running:

```
jupyter notebook <notebook>.ipynb
```

After running this command the notebook can be accessed using a web browser. The default port is 8888 and therefore the notebook can be reached by accessing localhost:8888.

Alternatively, JupyterLab can be used to run and edit the notebook. It can be installed by running:

```
pip install jupyterlab
```

After that, the web interface can be started by executing:

```
jupyter lab
```

4.2 Google Colab Installation and Execution

An online Jupyter notebook can be created at Google Colab. The only thing other than a web browser needed to create new Jupyter Notebooks is a Google account. Uploading and viewing notebooks on the other hand also works without an account. No local Python and Jupyter installations are required. A new notebook can simply be created by navigating to `colab.research.google.com` and selecting File → New.

4.3 Content Elements

Jupyter notebooks are a great tool for working with data and doing analysis. There are different possibilities that can be used to make an interesting, informative, and interactable explorable explainer in Jupyter.

4.3.1 Text

Text in Jupyter can be written using the markup language Markdown. It is written using plain text with a relatively simple syntax to indicate different formatting elements such as headings, lists, and images. Simple text and images can be used to provide the user with more information about a topic or equation and therefore guide the user through the explorable explainer.

4.3.2 Code

Jupyter notebooks support a wider variety of languages, including Julia, R, Haskell, Ruby, and Python. The code itself is structured in independent blocks which can be executed one after another. The blocks themselves can be manipulated and rewritten by the user, offering fine-grained control over the explorable explainer itself. After a code block is altered it can be re-run to explore the impact of the changes.

4.4 Libraries

A number of popular Python libraries can be utilized to construct visualizations.

4.4.1 Jupyter Widgets

Another way to implement user interactions are special widgets that directly alter variables and other behaviors of the notebook. These include sliders, text fields, radio buttons, checkboxes, and many more. With widgets, end users with little to no programming experience can alter certain aspects of the explorable explainer without diving too deep into the code itself.

4.4.2 Data Table

This special kind of table can be used to render data in a spreadsheet format. It also offers some interaction in the form of data filtering and keyword searching.

4.4.3 Matplotlib

Matplotlib is a data plotting library supporting a wide range of different types of visualizations including scatter plots, bar charts, line charts, and many more. The downside of Matplotlib is that it is generally not interactable. This can be overcome by altering code blocks and re-generating the plots to apply the changes to the graphics.

4.4.4 Plotly

Plotly is similar to Matplotlib in that it is also used to plot and visualize data. The main difference is that it supports different types of plots including parallel coordinates, and that it offers common user interactions out of the box. The supported types of interaction differ between the types of plots. Many common interactions like zooming, hovering over data points, and saving are already present, but extending them for more custom ones is quite difficult.

4.5 Installing Locally

One of the most common ways to host and run Jupyter notebooks is, to install Jupyter locally. Once installed users can launch Jupyter Notebooks locally in their web browser.

4.6 Hosting Online

Jupyter Notebooks can be hosted on self-owned or rented remote servers by using tools like JupyterHub or Docker.

MyBinder is a free cloud-based service to host, share and run Jupyter notebooks. It offers the possibility to directly link to a Git repository to load the notebooks from. The usage of this service is relatively intuitive and well-documented, but testing showed, that it is rather unreliable and unstable. This happens probably because of limitations such as limited computational resources and storage.

As previously mentioned, Google Colab is a free cloud-based solution to host and share Jupyter notebooks. The setup boils down to uploading or creating a new notebook. Then the link to the workspace can be shared or made publicly available for anyone to explore. Testing showed, that the service is really reliable and stable.

Chapter 5

Observable

Observable is an online platform specialised in interacting and visualising data. It allows users to create interactive code bases online in so called notebooks. These notebooks can be shared and worked on by multiple developers at once and can contain a multitude of different information such as code, raw data, images, text, charts, and more.

5.1 Installation and Execution

Observable is a purely online service that is available for free with certain caveats after signing up on the website. The signup process is as easy as submitting an email address, entering a password, and confirming the link that is sent by mail. Afterwards, you can start creating your own notebooks. However, when using the free version of the service, all notebooks and data used are publicly available to all other users.

5.2 Content Elements

Observable notebooks are perfect for data analysis and visualization. They have native support of many database connectors, APIs, file attachments, and cloud files. They also support many large libraries such as D3.js and have their own plotting library called observable plot.

5.2.1 Text

Text in Observable can be written using Markdown. Thus adding text and images to guide a user through an Explorable could easily be achieved.

5.2.2 Code

Code in Observable is written in JavaScript, but there are some things that are unique about the code in Observable that are not strictly JavaScript and can make it harder for someone expecting to write JavaScript. It is also difficult to find exactly where the differences between actual JavaScript and Observable's version of JavaScript lie. Observable also supports HTML and SQL.

5.2.3 D3.js

D3.js is a JavaScript library for manipulating documents based on data. D3 allows a user to integrate and manipulate data in a way that makes it much easier to understand. It is supported by Observable and makes visualising and manipulating data within an observable notebook possible.

5.3 Hosting Online

All Observable notebooks are hosted online on their website and can immediately be shared without needing to host or publish them separately, which would make it easier to create and distribute an explorable explainer through these means.

Chapter 6

Shiny

Shiny for R is a powerful and versatile web application framework that enables the creation of interactive dashboards, data visualization tools, and custom web applications. With Shiny, complex web applications in R can be built without needing to know HTML, CSS, or JavaScript. It provides a wide range of built-in widgets, like sliders, input boxes, buttons, and plots, that can be easily added to the applications with just a few lines of code.

Furthermore, Shiny's reactive programming feature ensures that your application updates itself automatically in response to user interactions or data changes, giving users the ability to create dynamic and responsive applications that can be updated in real-time. It also is compatible with various other R libraries, but the combination might not be as interactive and responsive. Shiny applications are not naturally designed to be explorable explainers, but the tools within might make it a candidate to build one.

6.1 Local Installation and Execution

Shiny can be used after setting up an R environment locally, most commonly using RStudio. RStudio is an IDE for the R programming language. After the successful creation of such an environment, the Shiny package can be installed using the R console with the following command:

```
install.packages("shiny")
```

After that, a simple R script needs to be created with the needed Shiny components, which are explained in Section 6.3. The following line needs to be added to the script:

```
library(shiny)
```

Finally, the app can be run by referencing the R script (for example "script.R") in the R console like this:

```
runApp('script.R')
```

Furthermore, eleven built-in examples can be used after the installation of Shiny by using this command:

```
runExample("01_hello")
```

This command would run the first example called "hello".

Every Shiny application is run on a local server ("Listening on http://127.0.0.1" should be displayed in the R console). By visiting our local host (127.0.0.1) in our browser we should see our Shiny application hosted on our local server.

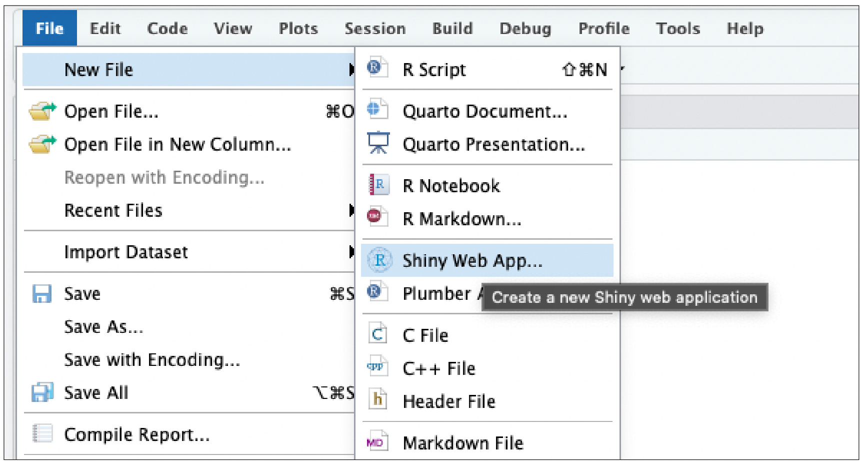


Figure 6.1: Initialisation of a Shiny application in a Posit Workspace.

6.2 Posit Cloud Installation and Execution

Another viable option is the use of the online workspace provided by Posit, which is accessible at `posit.cloud`. Since it is developed by the same group who develop RStudio, the workspace looks familiar to RStudio users. A Posit account is needed to use the workspace. After logging in, you can create a new project and create a Shiny web application, as shown in Figure 6.1.

This leaves you with a ready-to-go Shiny project as this approach takes care of installing packages and setting them up. You also have the option to click on the "Run App" button to start the application and automatically launch a browser window showing your application.

6.3 Shiny Components

Shiny consists of three main parts, which are needed for a working web application. In this section, we take a closer look at them and explore possible definitions.

6.3.1 User Interface

The first component of a Shiny app is the User Interface (UI). Within the definition, every needed component and layout can be defined. The UI is initialized with the following code:

```
ui <- fluidPage()
```

In Shiny, you can utilize the fluidPage function to design a display that automatically adapts to the size of your users' web browser window. By incorporating various elements into the fluidPage function, you can arrange and configure the user interface of your application. The following elements are most commonly added to the page:

- *Layouts* are mostly one of the first things defined in the UI. We can use single page layouts like `sidebarLayout()` or `fluidRow()`. The `sidebarLayout()` consists of one `mainPanel()` and one `sidebarPanel()`. Whereas using `fluidRow()` lets us define multiple row containers for a single page. On the other hand, multi-page layouts are also possible. To create an application with many taps we can use `tapsetPanel()` with one or more `tapPanel()`. Another way is to use a `nalistPanel()` for navigation instead of the `tapsetPanel()`.

- *Text* and *images* are really important to every UI. Text can be added by using the `p()` command. While we can also create *headers* by using `h*()` in a panel definition, where * is the level of the title (similar to HTML).

- For interactability, most Shiny apps use *control widgets*, which can be defined in any panel. There are many standard widgets like `actionButton()`, `textInput()`, `dateInput()`, `checkboxInput`, and many more. Every widget has at least a name attribute, which can be used to access the value in the program, and a label attribute, which is a label shown within the application.
- To respond to user input *reactive output* can be used, by defining an output function in any panel. They include `dataTableOutput`, `plotOutput`, `textOutput`, and many more. Every output function requires at minimum a single argument, which is a name string.

6.3.2 Server

After designing the UI, the second needed part is the server of the Shiny app. The server is responsible for defining the logic and behavior of the web application. It can be defined with the following line:

```
server <- function(input, output) {}
```

where every constraint and needed computation needs to be defined within the "{ }". The input and output provide us access to variables defined in the UI. For example, `input$var` reads the value from an input widget with the name `var`. Every dependency we want in our applications can be set up like this:

```
  output$selected_var <- renderText({
  paste("You have selected", input$var)
})
```

This example dependency reads the input at `var` and pastes the text "You have selected" + var to the output `selected_var`. To react to changing user input more efficiently, we can use the `reactive()` function to define what value gets updated every time user input is changed. Furthermore, we can use the server definition to read data from files and datasets, use other R packages, or do computations of data.

6.3.3 Running

After the UI and server are set up, we lastly need to create the Shiny app by defining our UI and server like this:

```
shinyApp(ui = ui, server = server)
```

Now, the application is ready to be run.

6.4 Installing Locally

As discussed previously, a Shiny web application can be started from a local R environment, which starts the application on a local server. Afterward, the application can be used in the browser by accessing the local host.

6.5 Hosting Online

The most straightforward way to convert your Shiny application into a web page is by using `shinyapps.io`, which is a hosting service specifically designed for Shiny applications provided by Posit. This platform gives you full control over your application, including server administration tools. The application can be directly imported from the R workspace on `posit.cloud`.

Posit Connect is a publishing platform which allows publishing Shiny applications with just one button. It is made for commercial projects and is not available for free. The cheapest solution starts at $15,745 per year.

For self-hosting, Shiny Server is a program which works alongside Shiny to create a web server specifically designed to host Shiny apps. It is a free, open-source program which can be downloaded from GitHub. Shiny Server can be run using a Linux server and can host multiple Shiny applications.

Chapter 7

D3 Standalone

D3 is a JavaScript library for creating custom visualizations. It provides a wide range of recipes and examples for many kind of visualisations, including parallel coordinates. JavaScript is a popular programming language that is widely used for building interactive websites and web applications. HTML is the markup language used to create the structure of a web page, while CSS is the styling language used to control the layout and appearance of a web page. Using D3 with JavaScript, HTML, and CSS, it is possible to constuct a fully functional and interactive explorable explainer, which facilitates user understanding of complex concepts.

7.1 Standalone File Structure

The standalone structure of a web application comprises several files that work together to create the user interface and functionality. In the case of the standalone, the files `index.html`, `ParallelCoordinates.js`, and `styles.css` are used to build the web application, as shown in Listing 7.1. The `index.html` file serves as the main entry point for the web application and contains the HTML code that defines the structure and content of the web page. The `Parallelcoordinates.js` file contains the JavaScript code that provides the functionality of the web application, including data visualization and interaction. Finally, the `styles.css` file contains the CSS code that defines the visual styling and layout of the web page.

```
Name                       | Type
-------------------------------------------------
- index.html               | HTML document
- ParallelCoordinates.js   | JavaScript code
- styles.css               | CSS styles
```

Listing 7.1: Files in the D3 standalone explainer.

The D3 library is best downloaded and included in a subfolder called, say, `libs/`, It is included in the web application by referencing its path in a script element in the `index.html` file, for example:

```
"<script src = "./libs/d3.min.js"></script>".
```

This approach is useful when working with a specific version of the library, or starting the application locally without any need for an internet connection.

7.2 Visual Studio Code Editor

Visual Studio Code is a popular code editor that has been optimized for building and debugging modern web and cloud applications [Microsoft 2015]. It has gained popularity due to its open-source nature, versatility, and extensive customization options through various extensions. Its seamless integration with different programming languages and its ability to offer an easy-to-use development environment has made it a preferred choice for many developers. Therefore, using Visual Studio Code for developing the D3 standalone explainer was a straightforward decision.

7.3 Running Locally

Running a standalone web application locally involves setting up a local server, configuring it to host the application, and opening the application in a web browser. One possibility, assuming Python is installed, is to run the one-line command:

```
python3 -m http.server 8000
```

in the folder containing the `index.html` file, and then open a web browser at `localhost:8000`.

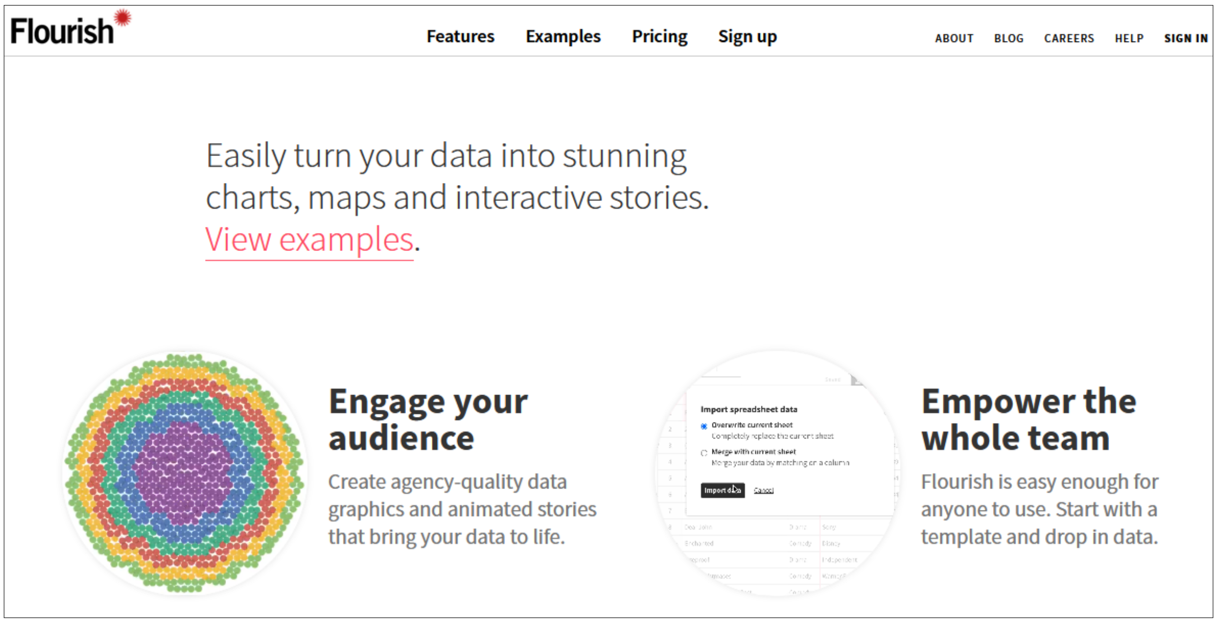
7.4 Hosting Online

A standalone web-based explainer can be hosted on either self-owned or rented remote servers. One of the easiest is through GitHub Pages [GitHub 2023].

Chapter 8

Flourish

Flourish is an online data visualization platform that allows users to create interactive charts, maps, and other forms of data visualization [Flourish 2021]. It is designed to be user-friendly, with a drag-and-drop interface that enables even non-technical users to create sophisticated visualizations. Flourish provides users with various chart types, such as line, bar, scatter, and tree maps. It also offers a wide range of customization options such as colors, fonts, and animation effects. Flourish was considered as a basis for producing explorable explainers, but was found to be unsuitable, due to the lack of support for custom code or the integration of external libraries.



Easily turn your data into stunning charts, maps and interactive stories. View examples.

Engage your audience

Create agency-quality data graphics and animated stories that bring your data to life.

Empower the whole team

Flourish is easy enough for anyone to use. Start with a template and drop in data.

Figure 8.1: Flourish an online data visualization platform [Flourish 2021].

Chapter 9

Results

To be able to better compare the different tools described in Chapter 3, four of them were chosen to be used to implement parallel coordinates for an explorable explainer, namely: Jupyter, Shiny, Observable, and D3 Standalone. Parallel coordinates are a classic technique for the visualization of multidimensional datasets. All four implementations aim to support the creation of an explorable explainer for parallel coordinates using the same dataset and narrative. It was not possible in the timeframe of the survey to build fully functional explorable explainers.

9.1 Dataset

The dataset shown in Table 9.1 was carefully curated to illustrate the use of parallel coordinates. It represents the marks (between 0 and 100) achieved by 30 students in 8 different subjects. Each row (record) represents a student, and each column (dimension) represents a subject.

9.2 Interactivity and Analysis

The following interactions are typically supported by parallel coordiniates:

- *Hovering over a record*: to temporarily highlight and identify the record(s) beneath the pointer.
- *Selection of one or more records*: to select and highlight a group of one or more records.
- *Filtering on a dimension*: to disable records from consideration.
- *Moving a dimension*: to move a dimension to another position.
- *Inverting a dimension*: to flip the dimension from ascending to descending, or vide versa.

To inspect a dataset for potential correlations, dimensions of interest have to be moved adjacent to one another. A cross pattern indicates a potential negative correlation. A straight-line pattern indicates a potential positive correlation (one of the dimensions can be inverted to check).

9.3 Narrative

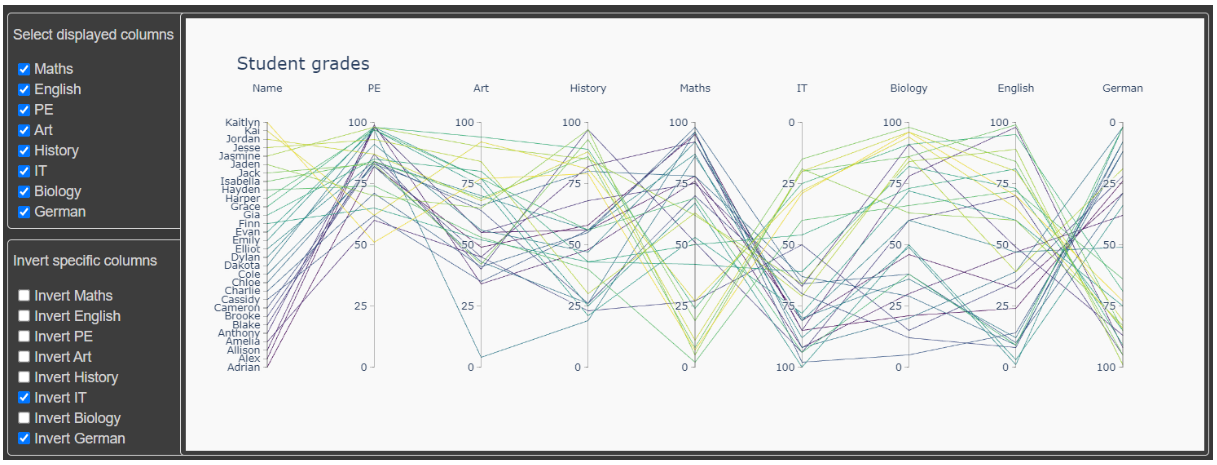
The narrative of the future explorable explainer aims to provide guidance in learning how to use parallel coordinates to explore multi-dimensional datasets for outliers, patterns, and correlations. The explorable might start with a brief introduction. After that, the user is encouraged to find correlations by looking at a traditional table. Following that, the explorable introduces the user to parallel coordinates and how to spot correlations and inverse correlations in the dataset. At the end of the explorable, the user is encouraged to find explore the dataset themselves.

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 9.1: Curated dataset of the marks (between 0 and 100) achieved by 30 students in 8 different subjects, used to illustrate parallel coordinates.

9.4 Jupyter

For the implementation of the explorable explainer as a Jupyter notebook, several Python libraries were used. The parallel coordinate plot itself was realized using Plotly. The dataset is stored as a pandas data frame which makes it simple to manipulate. Manipulation was necessary to offer the user the possibility to choose which columns of the dataset should be included in the plot. Another type of user interaction is to invert certain data columns. This is a common technique to simplify the process of finding correlations in the dataset. All user interactions were implemented using checkboxes included in the Jupyter widgets package. The left side of Figure 9.1 shows the user interaction to specify which columns to include in the parallel coordinates plot as well as the possibility to invert individual dimensions by selecting the corresponding checkboxes.


Select displayed columns

- ✓ Maths
- ✓ English
- ✓ PE
- ✓ Art
- ✓ History
- ✓ IT
- ✓ Biology
- ✓ German

Invert specific columns

- ☐ Invert Maths
- ☐ Invert English
- ☐ Invert PE
- ☐ Invert Art
- ☐ Invert History
- ✓ Invert IT
- ☐ Invert Biology
- ✓ Invert German

Figure 9.1: The Python implementation of parallel coordinates in a Juypter notebook.

9.4.1 Execution

The Jupyter Notebook is hosted on Google Colab, to make its execution as easy and frictionless as possible. By using this service, the user does not need to set up a Python environment, since the whole execution takes place in the browser. Another benefit especially for large datasets and machine learning tasks is, that the notebook will run on dedicated GPUs provided by the service, which could result in faster execution times and therefore less waiting.

9.4.2 Limitations

One limitation of the realization using a Jupyter notebook is that, although the Plotly library is very versatile and simple to use as-is, it is rather complicated and cumbersome to expand. This means, that if some necessary functionality is not already supported, a great deal of extra work is needed to implement it. This was the case with column inversion, which was bypassed by modifying the dataset directly.

9.4.3 Results

The resulting parallel coordinate plot is shown in Figure 9.1. The graphic itself is interactable, offering the possibility to rearrange and filter columns. Via the widgets, the user is able to choose which columns to include and whether a given column should be inverted. Another feature is that the plot can be downloaded directly as a png image.

9.5 Shiny

A parallel coordinates plot is not natively supported within the Shiny package for R. Therefore, several additional packages MASS, GGally, and Plotly were tested. While all three options yielded a correct representation of a parallel coordinates plot, Plotly was the only package enabling instant intractability. To enable the user to invert certain data columns, some manipulation was necessary. This was done by using UI widgets for every data column, which inverts the data before creating the plot with Plotly. The dataset gets read at the beginning of the program using the `read.csv()` function, which leads to a data format easily usable with Plotly. To react to the user checking the checkboxes, several variables were declared, with the `reactive()` function is used to react to user input changes.

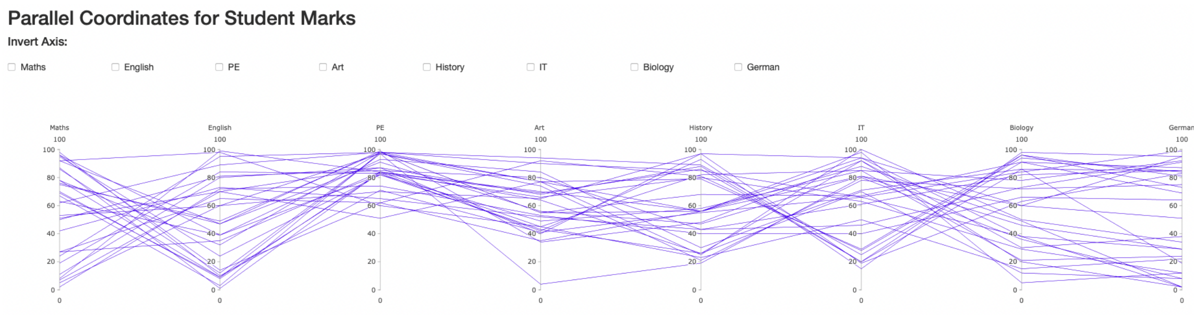


Figure 9.2: The R implementation of parallel coordinates in Shiny.

9.5.1 Execution

The Shiny web application was created on the Posit Cloud workspace, because it is an easy-to-start working environment for R and Shiny. The code can be run directly from the workspace using the "Run App" button or the R console. The Plotly package was automatically installed by the workspace. The only drawback of the Posit Cloud workspace is that you cannot publicly share your application. You can only invite specific individuals to access it.

Therefore, the application was also hosted on its companion service `shinyapps.io`. The application could be imported from the posit cloud to shinyapps and hosted instantly. With this hosting service, the application can be used by everyone knowing the link.

9.5.2 Limitations

The biggest limitation encountered while using the Plotly package for additional plot variants is that the parallel coordinates plot looks great initially, but is not very customizable. The only workaround is to “hack” certain aspects, like manipulating the dataset before plotting to invert a column.

However, this approach has some annoying drawbacks. Whenever a checkbox is changed, the plot is newly generated and resets, causing previously moved columns to return to their original position. Additionally, since an external package is used to create the plot, some of the features of Shiny are lost. For example, plot animation is not available.

Although Shiny is suitable for building an explorable explainer, there is no pre-built function or layout specifically designed to support such a project. This means everything would need to be written from scratch.

9.5.3 Results

The implementation yields a web application featuring a parallel coordinates plot using the student grades dataset. As shown in Figure 9.2, the dataset is displayed correctly and can be interacted with. The columns can be reordered and inverted using the UI widgets. Specific records can be selected in the plot.

9.6 Observable

As a starting point, the parallel coordinates implementation of Sophiegri [2020] based on D3 was forked and then reworked. Adding a selection feature to the axis was also partially realised in the forked code and could easily be expanded upon.

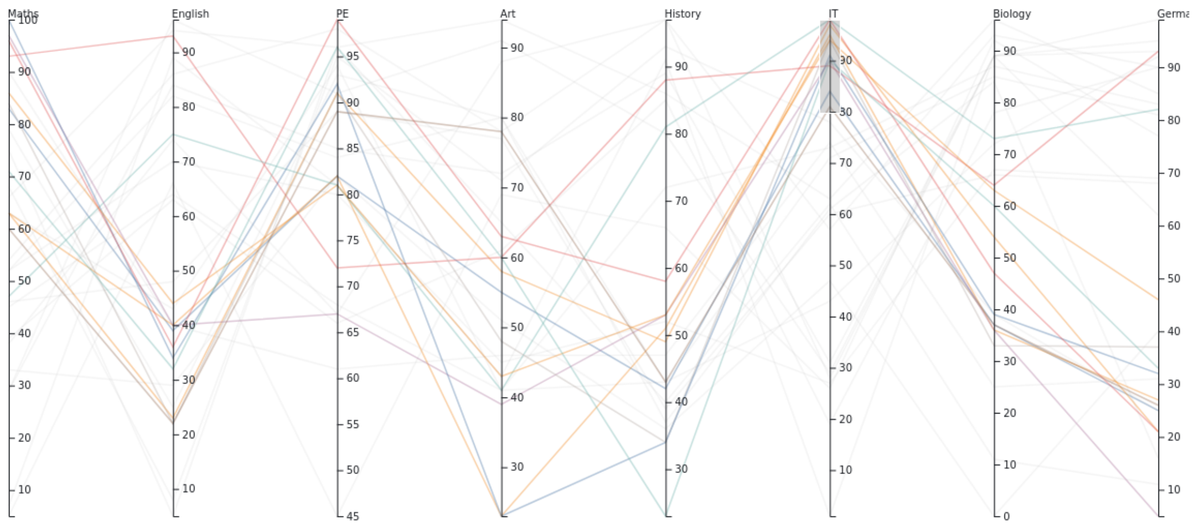


Figure 9.3: The JavaScript implementation of parallel coordinates in Observable.

9.6.1 Limitations

Some of the functionality such as reordering and inverting dimensions could not easily be implemented in Observable within the limited time and are missing from the implementation.

9.6.2 Results

The resulting parallel coordinate plot is shown in Figure 9.3. The plot does not allow for much interaction aside from selecting sections of the axis and is in its current state is not suitable for use in an explorable explainer. The fact that Observable uses its own version of JavaScript, that is just far enough from the original, makes it harder to build a fully working explorable within a short time.

9.7 D3 Standalone

In order to implement the standalone parallel coordinates implementation, the code editor Visual Studio Code was used. The parallel coordinates functionality was constructed using the D3 library.

9.7.1 Execution

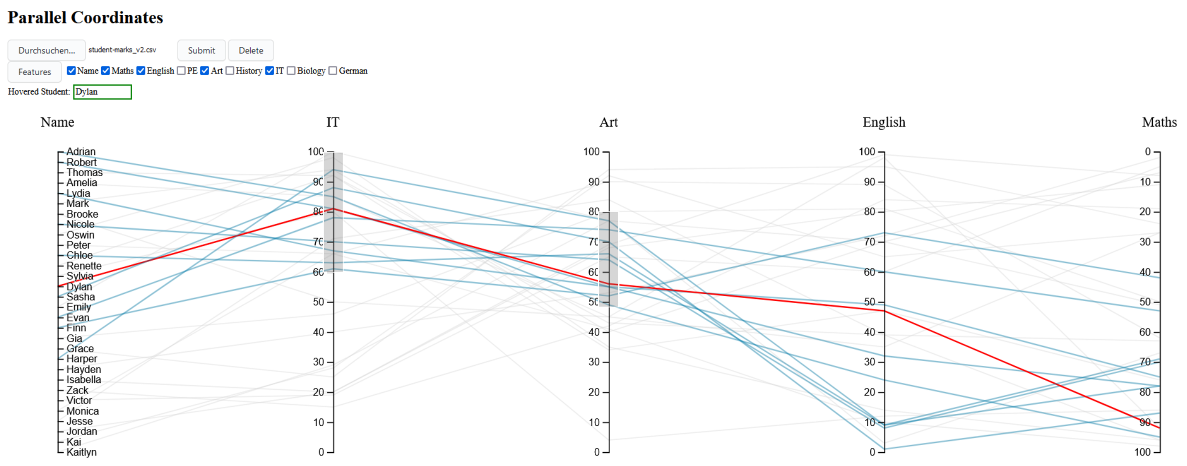
A local web server can be started in the folder containing the `index.html` file.

9.7.2 Limitations

The standalone has the potential for expansion and versatility, however, its limitations lie in the handling of large datasets. When the user attempts to load a high volume of data points onto the web application, the increased computational load may cause lagging, and ultimately lead to an unresponsive interface.

9.7.3 Results

The web application generated through this project allows the user to construct a parallel coordinate plot, as illustrated in Figure 9.4. This interactive parallel coordinate plot enables the user to modify the arrangement and filtering of the columns. Prior to creating the plot, the user can specify which dimensions should be included. Additionally, when hovering over a polyline, more information can be displayed. In this specific dataset, the name of the student is shown.


Parallel Coordinates

Durchsuchen... student-marks_v2.csv Submit Delete

Features ✓ Name ✓ Maths ✓ English ☐ PE ✓ Art ☐ History ✓ IT ☐ Biology ☐ German

Hovered Student: Dylan

Figure 9.4: The standalone JavaScript implementation of parallel coordinates with D3.

Chapter 10

Conclusion

Explorable explainers can be very helpful in teaching complex concepts through user interactions. Table 10.1 shows our verdict on the four tools explored in this survey for potentially creating an explorable explainer for parallel coordinates. A standalone implementation in JavavScript with D3 seems the most promising approach.

Tool	**Ease of use**	**Interactivity**	**Customizability**	**Graphs**
D3 Standalone	★ ★ ★	★ ★ ★ ★	★ ★ ★ ★ ★	★ ★ ★ ★ ★
Jupyter Notebooks	★ ★ ★ ★	★ ★ ★	★ ★ ★	★ ★ ★ ★ ★
Observable	★ ★	★	★ ★ ★	★ ★ ★ ★ ★
Shiny	★	★ ★ ★ ★ ★	★ ★ ★ ★ ★	★ ★ ★

Table 10.1: Comparison of different tools for creating an explorable explainer.

Bibliography

Ang, Yi Zhe [2022]. *K-Means Clustering*. 15 Nov 2022. `https://k-means-explorable.vercel.app/` (cited on page 6).

Carter, Shan and Chris Olah [2016]. *Distill*. 2016. `https://distill.pub/` (cited on page 4).

Case, Nicky [2015]. *Explorable Explanations*. 21 Mar 2015. `https://explorabl.es/` (cited on page 5).

Charig, C. R., D. R. Webb, S. R. Payne, and J. E. A. Wickham [1986]. *Comparison of Treatment of Renal Calculi by Open Surgery, Percutaneous Nephrolithotomy, and Extracorporeal Shockwave Lithotripsy*. British Medical Journal 292.6524 (29 Mar 1986), pages 879–882. doi:10.1136/bmj.292.6524.879 (cited on page 1).

Flourish [2021]. *Flourish*. 09 Nov 2021. `https://flourish.studio` (cited on page 21).

GitHub [2023]. *GitHub Pages*. 18 May 2023. `https://pages.github.com/` (cited on page 20).

Microsoft [2015]. *Documentation for Visual Studio Code*. 29 Apr 2015. `https://code.visualstudio.com/docs` (cited on page 19).

Pair [2010]. *AI Explorables*. 2010. `https://pair.withgoogle.com/explorables` (cited on page 3).

Sophiegri [2020]. *Exercise 3: Parallel coordinates*. Jun 2020. `https://observablehq.com/@sophiegri/exercise-3-parallel-coordinates` (cited on page 26).

Victor, Bret [2011]. *Explorable Explanations*. 10 Mar 2011. `http://worrydream.com/ExplorableExplanations/` (cited on page 1).

Wacker, Phillip [2018]. *An Explorable Explanation of Simpson's Paradox*. 28 Sep 2018. `https://pwacker.com/simpson.html` (cited on page 1).