

# **The Multidimensional Visual Analyser (MVA)**

Ožbej Golob



# The Multidimensional Visual Analyser (MVA)

Ožbej Golob

## Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Computer Science

submitted to

Graz University of Technology

Supervisors

Ao.Univ.-Prof. Dr. Keith Andrews  
Institute of Interactive Systems and Data Science (ISDS)  
Graz University of Technology

Doc. Dr. Aleš Smrdel  
University of Ljubljana

Graz, 17 Sep 2024



## **Abstract**

The Multidimensional Visual Analyser (MVA) is an open-source web application for visually exploring and analyzing multidimensional datasets. MVA combines the best features from the reviewed multidimensional visual analysis software and implements them as both a web and desktop application. It provides four synchronized visualizations: scatterplot matrix, scatterplot, similarity map, and parallel coordinates, together with labeled partitions (classes) and a table view. In particular, the parallel coordinates implementation is highly interactive and also displays categorical dimensions.

The application is built with Node, TypeScript, Svelte, Flowbite, D3, and Three.js. By rendering records with WebGL, MVA maintains its performance even when handling rather large datasets. MVA can be built and deployed as a web application, and can also be built as a desktop application through Tauri.

This thesis first reviews popular multidimensional visual analysis approaches and existing software tools. It then surveys modern web technologies, before describing MVA in detail. Two appendices provide a User Guide and Developer Guide.



# Contents

<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Listings</b>	<b>ix</b>
<b>Acknowledgements</b>	<b>xi</b>
<b>Credits</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Approaches to Multidimensional Visual Analysis</b>	<b>3</b>
2.1 Scatterplots . . . . .	3
2.2 Scatterplot Matrices (SPLOM). . . . .	4
2.3 Star Coordinates . . . . .	5
2.4 RadViz . . . . .	6
2.5 Dust and Magnet (DnM) . . . . .	7
2.6 Similarity Maps . . . . .	8
2.6.1 Principal Component Analysis (PCA) . . . . .	9
2.6.2 Multi-Dimensional Scaling (MDS) . . . . .	9
2.6.3 t-Distributed Stochastic Neighbor Embedding (t-SNE) . . . . .	9
2.6.4 Uniform Manifold Approximation and Projection (UMAP) . . . . .	10
2.6.5 Comparison of Similarity Mapping Techniques . . . . .	10
2.7 Parallel Coordinates . . . . .	11
2.8 Parallel Coordinates Matrix . . . . .	13
2.9 Brushing and Linking . . . . .	13
2.10 Grouping and Labeling . . . . .	14
<b>3 Tools for Multidimensional Visual Analysis</b>	<b>15</b>
3.1 XMDV . . . . .	15

3.2	Parallax . . . . .	16
3.3	GGobi . . . . .	17
3.4	InfoScope . . . . .	18
3.5	XDAT . . . . .	19
3.6	High-D . . . . .	20
3.7	TabuVis . . . . .	21
3.8	Improvise . . . . .	22
3.9	MyBrush . . . . .	23
3.10	mVis . . . . .	24
3.11	Comparison of Tools . . . . .	25
<b>4</b>	<b>Modern Web Technologies</b>	<b>27</b>
4.1	Web Applications . . . . .	27
4.1.1	Frontend . . . . .	27
4.1.2	Backend . . . . .	27
4.2	Frontend Development Frameworks . . . . .	28
4.2.1	Angular. . . . .	28
4.2.2	React . . . . .	29
4.2.3	Vue . . . . .	29
4.2.4	Svelte . . . . .	29
4.3	Web Graphics Rendering Technologies. . . . .	30
4.3.1	Canvas2D . . . . .	30
4.3.2	SVG-DOM . . . . .	30
4.3.3	WebGL. . . . .	31
4.3.4	WebGPU . . . . .	31
4.3.5	Offscreen Canvas . . . . .	31
4.4	Web Graphics Rendering Libraries . . . . .	32
4.4.1	SVG.js . . . . .	32
4.4.2	Konva.js . . . . .	32
4.4.3	Two.js . . . . .	32
4.4.4	Pixi.js . . . . .	32
4.4.5	Babylon.js. . . . .	33
4.4.6	D3.js. . . . .	33
4.4.7	Performance Comparison . . . . .	33
4.5	Desktop Development Libraries . . . . .	37
4.5.1	Electron.js. . . . .	37
4.5.2	Tauri. . . . .	37



<b>5</b>	<b>The Multidimensional Visual Analyser (MVA)</b>	<b>39</b>
5.1	Build System . . . . .	40
5.2	Dependencies . . . . .	40
5.3	Components . . . . .	41
5.4	Icons . . . . .	41
5.5	Example Datasets . . . . .	41
<b>6</b>	<b>Selected Details of the Implementation</b>	<b>43</b>
6.1	Chosen Web Graphics Rendering Technology . . . . .	43
6.2	Overlaying Canvases for Visualizations . . . . .	45
6.3	Scatterplot Matrix Rendering . . . . .	45
6.4	Using Web Workers . . . . .	46
6.5	Hovering and Brushing . . . . .	49
6.6	Filtering . . . . .	49
6.7	Selection Tools . . . . .	51
6.8	SVG Exporter . . . . .	52
<b>7</b>	<b>Outlook and Future Work</b>	<b>55</b>
7.1	Window Management . . . . .	55
7.2	Rendering Records with Pixi.js . . . . .	55
7.3	Rendering the Scatterplot Matrix . . . . .	55
7.4	t-SNE Similarity Map . . . . .	55
7.5	Parallel Coordinates Matrix Panel . . . . .	56
7.6	Handling Missing Data . . . . .	56
7.7	Automated Classification. . . . .	56
7.8	Rule-Based Definitions for Classes . . . . .	56
<b>8</b>	<b>Concluding Remarks</b>	<b>57</b>
<b>A</b>	<b>User Guide</b>	<b>59</b>
A.1	Installation . . . . .	59
A.2	Features . . . . .	59
A.3	User Interface. . . . .	59
A.3.1	Initial State . . . . .	59
A.3.2	Hovering and Brushing . . . . .	60
A.3.3	Navigation Bar . . . . .	60
A.3.4	Display Area . . . . .	62

A.4	Example Datasets . . . . .	71
A.4.1	Cars 1993 . . . . .	71
A.4.2	Cereals . . . . .	71
A.4.3	Iris . . . . .	72
A.4.4	Premier League . . . . .	72
A.4.5	Student Marks . . . . .	72
A.5	Dataset Formats . . . . .	72
A.6	Example Use Case . . . . .	74
<b>B</b>	<b>Developer Guide</b>	<b>81</b>
B.1	Quick Start. . . . .	81
B.2	Desktop Application . . . . .	81
B.3	Gulp Tasks. . . . .	82
B.4	Development Dependencies. . . . .	82
	<b>Bibliography</b>	<b>83</b>

# List of Figures

2.1	Scatterplots . . . . .	4
2.2	Scatterplot Matrix . . . . .	5
2.3	Star Coordinates Plots . . . . .	6
2.4	RadViz Plots . . . . .	7
2.5	Dust and Magnet Plots . . . . .	8
2.6	Similarity Mapping . . . . .	11
2.7	Parallel Coordinates Diagram . . . . .	12
2.8	Parallel Coordinates on Iris Dataset. . . . .	12
2.9	Parallel Coordinates Missing Data . . . . .	12
2.10	Parallel Coordinates Matrix . . . . .	13
2.11	Brushing and Linking. . . . .	14
3.1	XMDV . . . . .	16
3.2	Parallax. . . . .	17
3.3	GGobi . . . . .	18
3.4	InfoScope . . . . .	19
3.5	XDAT . . . . .	20
3.6	High-D . . . . .	21
3.7	TabuVis . . . . .	22
3.8	Improvise . . . . .	23
3.9	MyBrush . . . . .	24
3.10	mVis. . . . .	25
4.1	Client-Server Model . . . . .	28
4.2	Slay Lines Application . . . . .	33
4.3	Performance of Web Graphics Rendering Libraries on Laptop . . . . .	36
4.4	Performance of Web Graphics Rendering Libraries on Desktop. . . . .	36
5.1	MVA User Interface . . . . .	39
6.1	Lines Rendered with Pixi.js . . . . .	44
6.2	Lines Rendered with Three.js . . . . .	44
6.3	Scatterplot Matrix Panel. . . . .	46

6.4	MVA with Brushed and Hovered Records . . . . .	49
6.5	Filtering Records in Parallel Coordinates Panel . . . . .	50
6.6	Filtered Records are Inactive . . . . .	50
6.7	Example of Exported SVG Visualization . . . . .	53
A.1	MVA User Interface . . . . .	60
A.2	The Initial State of MVA . . . . .	61
A.3	Hovered and Brushed Records . . . . .	61
A.4	Navigation Bar Dropdowns. . . . .	62
A.5	Import Dataset Modal . . . . .	62
A.6	Export Dataset Modal . . . . .	63
A.7	Invalid Rows Modal . . . . .	63
A.8	About Modal . . . . .	63
A.9	Panel Buttons . . . . .	64
A.10	Scatterplot Matrix Panel. . . . .	65
A.11	Scatterplot Panel . . . . .	65
A.12	Scatterplot Panel Controls . . . . .	65
A.13	Similarity Map Panel . . . . .	66
A.14	Similarity Map Panel Controls . . . . .	66
A.15	Partitions Panel . . . . .	67
A.16	Partition Controls . . . . .	67
A.17	Partition Context Menu . . . . .	68
A.18	Table Panel . . . . .	68
A.19	Table Panel Controls . . . . .	69
A.20	Parallel Coordinates Panel . . . . .	69
A.21	Parallel Coordinates Panel Controls. . . . .	70
A.22	Parallel Coordinates Panel with Histograms. . . . .	70
A.23	Parallel Coordinates Axis Context Menu . . . . .	70
A.24	Parallel Coordinates Axis Context Menu Modals . . . . .	71
A.25	Student Marks Dataset Import. . . . .	74
A.26	MVA with Student Marks Dataset . . . . .	74
A.27	Scatterplot Matrix Panel. . . . .	75
A.28	Scatterplot Matrix and Scatterplot Panels . . . . .	76
A.29	Reordered Parallel Coordinates Panel . . . . .	76
A.30	Adding Selected Records to Engineers Partition . . . . .	77
A.31	Adding Records to Linguists Partition . . . . .	78
A.32	MVA Final Look of Student Marks Dataset . . . . .	79

# List of Tables

2.1	Iris Dataset . . . . .	3
3.1	Overview of Reviewed Tools . . . . .	25
3.2	Comparison of Reviewed Tool Features . . . . .	26
4.1	Performance of Web Graphics Rendering Libraries on Laptop . . . . .	34
4.2	Performance of Web Graphics Rendering Libraries on Desktop. . . . .	34



# List of Listings

4.1	Python Script to Measure Average FPS . . . . .	35
5.1	MVA Components . . . . .	42
6.1	CSS z-index Property for Overlaying Canvases . . . . .	45
6.2	Offscreen Canvas in Web Worker . . . . .	47
6.3	Distributing Calculating Tasks Between Calculating Workers . . . . .	48
6.4	Code for Calculating Point-in-Polygon. . . . .	51
6.5	Code for Calculating Line-Line Intersection. . . . .	52
6.6	Example Exported Shortened SVG Code. . . . .	54
A.1	Iris Dataset with Partitions (CSV Format) . . . . .	73
A.2	Iris Dataset with Partitions (Small CSV Format) . . . . .	73
A.3	Iris Dataset with Partitions (MVA Format) . . . . .	73





# Acknowledgements

I want to express my gratitude to all my colleagues, especially my supervisor Keith Andrews, for providing feedback, ideas, and all the other help with the thesis. I also want to thank my supervisor Aleš Smrdel for the help with the thesis.

I also want to express my gratitude to the University of Ljubljana and Graz University of Technology, for enabling me to take part in the double degree programme, in the scope of which this thesis was written.

Ožbej Golob  
Ljubljana, Slovenia, 17 Sep 2024



# Credits

I would like to thank the following individuals and organizations for permission to use their material:

- The thesis was written using Keith Andrews' skeleton thesis [Andrews 2021].



# Chapter 1

## Introduction

Making sense of large multidimensional datasets can be hard, especially as such datasets are growing ever larger. Data visualization provides visual tools to help humans explore and interpret large datasets [Few 2021; Kirk 2019; Fry 2007; Andrews 2024a]. Data visualization techniques take advantage of the characteristics of human visual perception, which can rapidly recognize visual patterns, trends, and outliers without conscious cognitive effort [Ware 2021].

Multidimensional datasets are typically represented in tabular (spreadsheet) form, where each column represents a dimension (variable) of the data, and each row represents a single record (data point). Multidimensional visual analysis focuses on the use of visual representations to explore and analyze multidimensional datasets. This approach provides a more intuitive and interactive way to understand and extract insights from large and complex multidimensional datasets.

One of the main challenges in multidimensional visual analysis is the effective representation of high-dimensional data in a way that is meaningful and intuitive for the user. To address this challenge, a wide range of visual encodings and interaction techniques have been developed. One of the key benefits of multidimensional visual analysis is its ability to reveal patterns and relationships in the data that may not be apparent through traditional statistical analysis methods. This is particularly useful in the exploratory phase of data analysis, where the aim is to gain a better understanding of the data, identify patterns, trends, and outliers, and determine potential areas of interest for further investigation. Multidimensional visual analysis also has a range of applications in areas such as data mining, machine learning, and business intelligence [Dzemyda et al. 2012].

This thesis presents the Multidimensional Visual Analyser (MVA). MVA is an open-source web application that allows users to explore large multidimensional datasets. MVA can display data in up to six panels, namely: Scatterplot Matrix, Scatterplot, Similarity Map, Partitions, Table, and Parallel Coordinates. The panels are synchronized, supporting brushing and linking, and highly interconnected analysis tasks. MVA can also assist a user in grouping data records into labeled partitions. By rendering records with hardware-accelerated WebGL, the application maintains performance even when handling quite large datasets. MVA source code is available on GitHub [Golob and Andrews 2024b]. MVA is deployed as a web application [Golob and Andrews 2024a] and can also be built as a desktop application for Windows.

The first part of the thesis describes work in related areas and provides an understanding of web applications in the context of information visualization. Chapter 2 gives an overview of popular multidimensional visual analysis approaches. Chapter 3 presents some existing multidimensional visual analysis software. Chapter 4 reviews web applications, frontend development frameworks, web graphics rendering technologies, and desktop development libraries.

The second part of the thesis describes the development of MVA. Chapter 5 describes the MVA application, including how to build it, its component structure, dependencies, icons, and example datasets.

Chapter 6 describes the approaches implemented and the decisions that were taken. Chapter 7 describes some potential future improvements. Chapter 8 concludes the thesis with an overview of the thesis and final remarks. Appendix A serves as a User Guide for MVA, to help users use MVA effectively and efficiently. Appendix B serves as a Developer Guide.

## Chapter 2

# Approaches to Multidimensional Visual Analysis

Multidimensional visual analysis refers to methods and techniques used to analyze and understand complex datasets using visual representations. These approaches typically involve the use of specialized software or tools that allow analysts to create and manipulate graphical representations of the data to discover patterns, trends, and relationships. This chapter reviews some popular multidimensional visual analysis approaches, following the discussions in Dzemyda et al. [2012].

Most of the figures in this chapter were generated using Python with the help of the following libraries: matplotlib, pandas, scikit-learn, and umap [Hunter 2007; McKinney 2010; Pedregosa et al. 2011; McInnes, Healy, Saul et al. 2018]. The graphics use the Iris dataset as an example, since it is widely used in the research community [Fisher 1936]. It is a dataset of Iris flower measurements with five dimensions: species (setosa, versicolor, or virginica) which can be interpreted as a class, and four numerical dimensions (sepal length, sepal width, petal length, and petal width) which are measured in centimeters. After the header row, there are 150 rows of data records representing 150 instances of flowers. The dataset consists of 50 samples from each of three species of Iris. Table 2.1 shows the first five records of the Iris dataset.

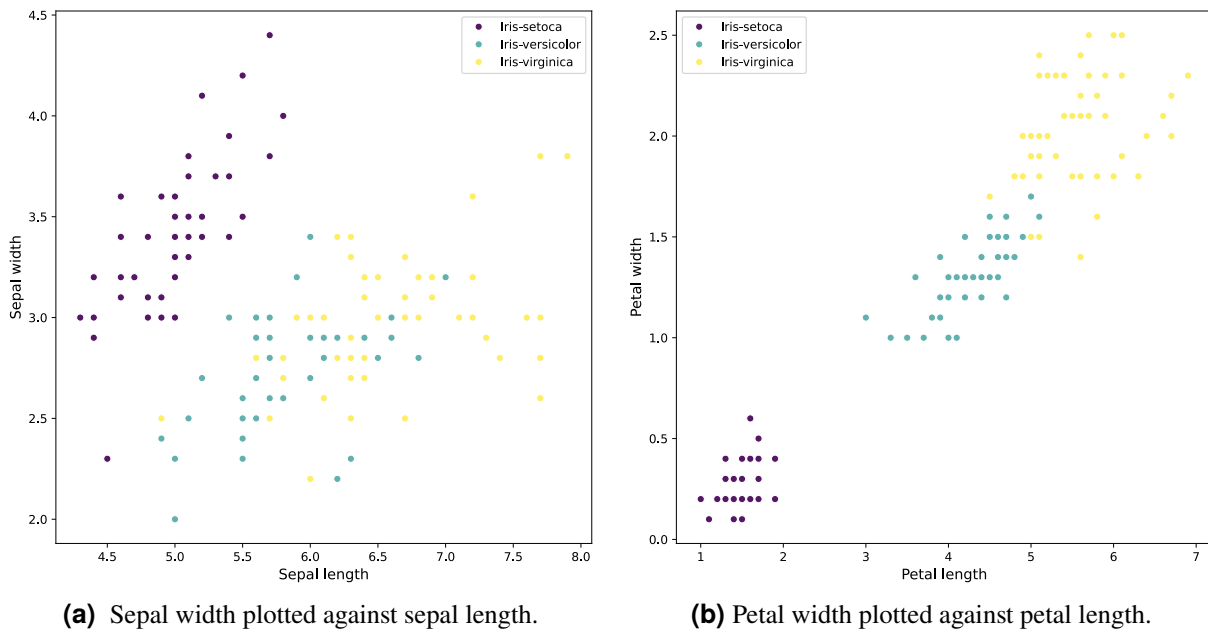
### 2.1 Scatterplots

A scatterplot is a type of chart or graph used to display the relationship between two or three numerical dimensions [Friendly and Denis 2005]. It can help identify potential trends, patterns, and outliers in the data. It uses a dot or marker to represent each record, where the position of each dot on the graph indicates the values of the record in each dimension. As such, a scatterplot is a classic 2d  $(x, y)$  or 3d  $(x, y, z)$  plot. Figure 2.1 shows two examples of classic 2d scatterplots.

In 2d, the resulting graph displays a set of dots, where each dot represents a single record. If the

Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa

**Table 2.1:** The first five records of the Iris dataset.



**Figure 2.1:** Scatterplots of the Iris dataset. In (b), there is a clear positive correlation between petal width and petal length, and the three classes are fairly well separated. [Drawn by OZbej Golob using Python.]

dots tend to form a diagonal line sloping upwards from left to right, that is an indication of a positive relationship (correlation) between the two dimensions. If the dots tend to form a diagonal line sloping downwards from left to right, that is an indication of a negative relationship (correlation) between them. If the dots appear to be scattered randomly across the graph, there is no apparent linear relationship between the two dimensions. Figure 2.1 shows two scatterplots of the Iris dataset. Figure 2.1a shows sepal width plotted against sepal length, where there is no apparent relationship between the two dimensions. Figure 2.1b shows petal width plotted against petal length, where there is an apparent positive correlation between the two dimensions.

One of the key disadvantages of scatterplots is that they can only accommodate two (or three) dimensions at a time. Thus, they are insufficient as a tool for multidimensional visual analysis on their own.

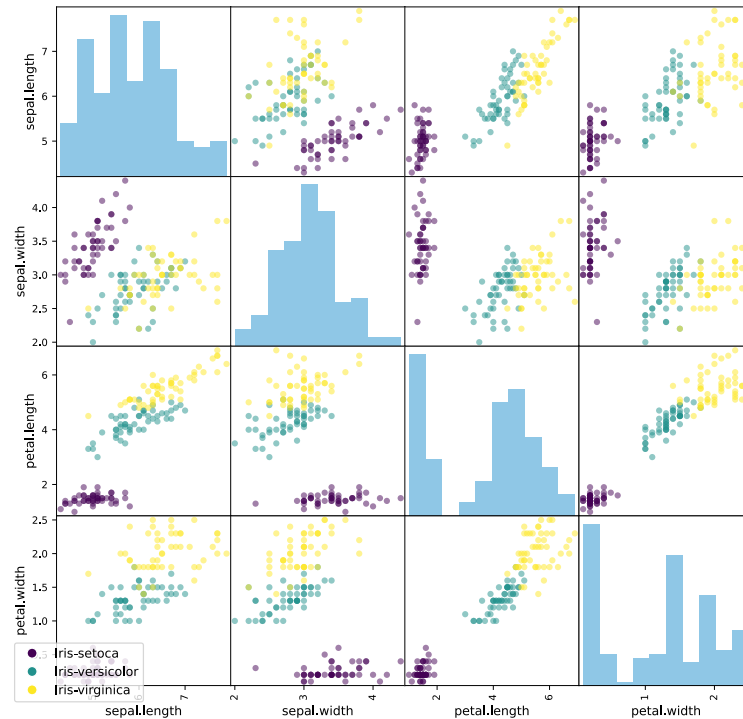
## 2.2 Scatterplot Matrices (SPLOM)

A scatterplot matrix (SPLOM) is used to explore the pairwise relationships between multiple dimensions [Carr et al. 1986] in a single view. It consists of a grid of scatterplots, with each individual plot displaying the relationship between two dimensions. The diagonal is sometimes used to display the name of the corresponding dimension and/or a histogram of the distribution of the dimension. The matrix is symmetric, in the sense that the bottom left triangle is a reflection of the top right triangle. Figure 2.2 shows an example of a scatterplot matrix for the four numerical dimensions of the Iris dataset.

One of the primary advantages of using scatterplot matrices is the ability to explore multiple relationships simultaneously. Instead of creating multiple individual scatterplots to visualize each relationship, a scatterplot matrix provides a comprehensive overview of all the pairwise relationships between dimensions in a single visualization. Typically, an individual scatterplot can be selected and enlarged for closer inspection. However, scatterplot matrices become unwieldy with a larger number (say 15 or more) of dimensions.

When interpreting scatterplot matrices, it is important to look for patterns and trends that emerge across





**Figure 2.2:** A scatterplot matrix displaying all pairwise scatterplots of the four numerical dimensions in the Iris dataset. The diagonal cells are used to display a histogram of the distribution of that dimension. The color of the individual data points indicates the species. [Drawn by Ožbej Golob using Python.]

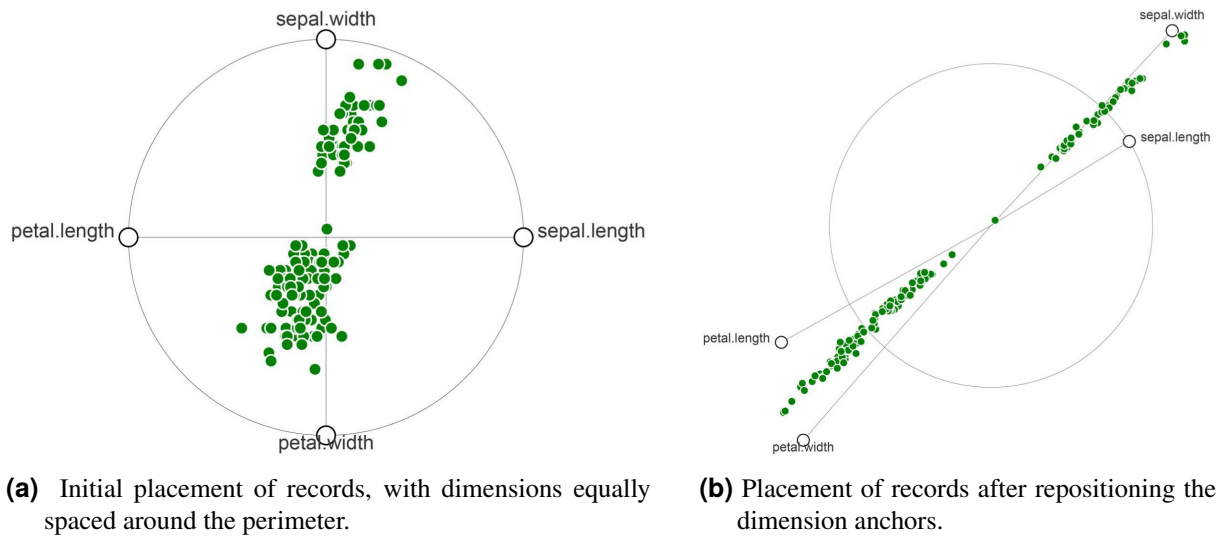
the plots. For example, if the histograms on the diagonal show that one dimension has a highly skewed distribution, it may be necessary to transform the dimension to improve the performance of a statistical model. Similarly, if multiple plots show a strong linear relationship between two dimensions, it may be worth exploring the use of linear regression models to further investigate the relationship.

## 2.3 Star Coordinates

Star Coordinates is a visualization technique that maps multidimensional data onto a two-dimensional space using a radial layout [Kandogan 2001]. The basic idea behind Star Coordinates is to represent each record as a set of coordinates on a star-shaped plot. The plot consists of a series of radial axes radiating from a central point, with each axis representing a dimension in the data. Each record is mapped to its position using a weighted average of its values in each of the dimensions. Figure 2.3 shows an example of a Star Coordinates plot displaying the Iris dataset. In the plot, the data is separated by sepal and petal features (length and width).

The end of each axis is adorned with a circular marker (anchor), which can be interactively moved to relocate the corresponding axis in the plot. The records reposition themselves to follow the anchor's motion. In Star Coordinates, an anchor can be moved outside or inside the circle to emphasize or deemphasize its influence respectively. The records too can reside either inside or outside the circle.

Star Coordinates plots can comfortably accommodate a larger number (say 10 or 20) dimensions, but like other techniques, they can become cluttered if the number of records becomes too large.

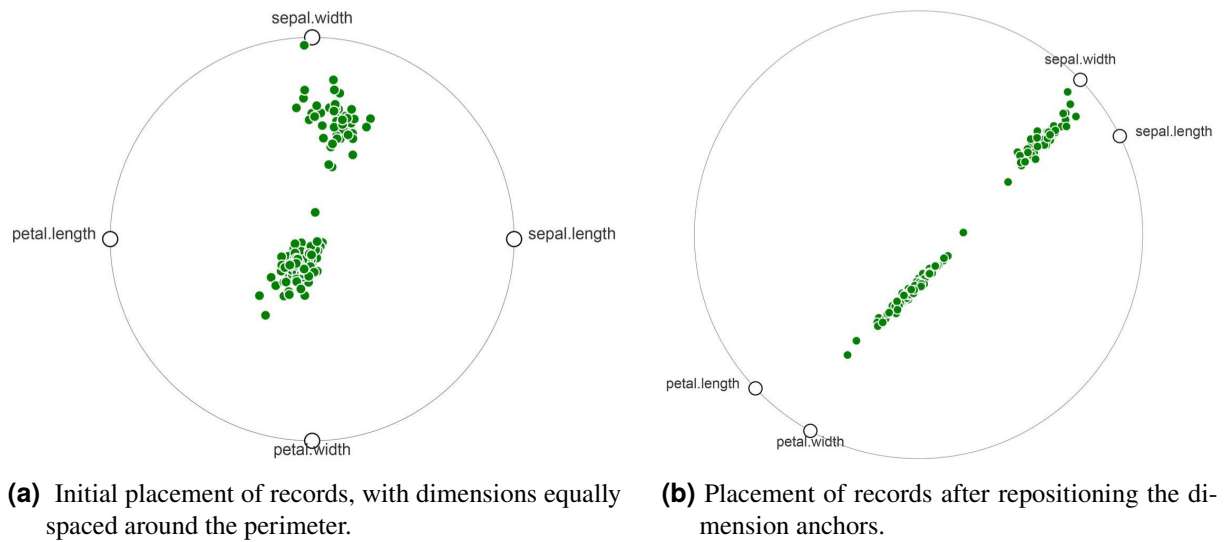


**Figure 2.3:** Star Coordinates plots of the Iris dataset. [Drawn by Ožbej Golob using RPE [Neuhold et al. 2020].]

## 2.4 RadViz

RadViz (Radial Visualization) is similar to Star Coordinates, in that it maps multidimensional data onto a two-dimensional space using a radial layout [Hoffman et al. 1997]. Dimensions are represented by anchor points placed around the perimeter of a circle. Each record is represented by a point inside the circle. The position of the record in the circle is determined by the weighted average of the dimension values associated with that point. The weight of each dimension is determined by the user and can be used to emphasize or de-emphasize certain dimensions in the visualization. Figure 2.4 shows an example of a RadViz plot displaying the Iris dataset.

In the case of RadViz, the anchors can only be moved around the perimeter of the circle, and the records always remain within the circle. A RadViz plot can comfortably accommodate a larger number (say 10 or 20) of dimensions. Like other techniques, it can become cluttered if the number of records becomes too large.

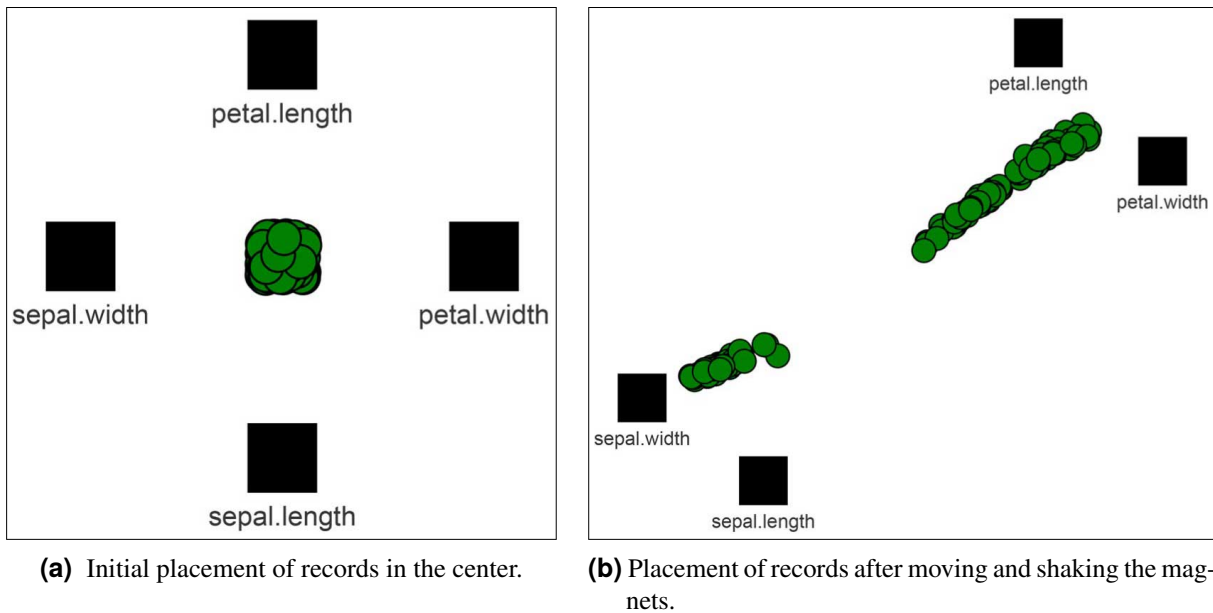


**Figure 2.4:** RadViz plots of the Iris dataset. [Drawn by Ožbej Golob using RPE [Neuhold et al. 2020].]

## 2.5 Dust and Magnet (DnM)

Dust and Magnet (DnM) is a visualization technique that maps multidimensional data onto a two-dimensional space [Yi et al. 2005]. The basic idea behind DnM is to represent each record as a particle of dust that is attracted to one or more magnets. The magnets represent different dimensions in the data, and their strength determines the influence of each dimension on the records. The stronger the magnet, the greater the influence of the corresponding dimension on the records. The user can increase the influence of a particular dimension by clicking and/or shaking the magnet of the corresponding dimension. The use of particles and magnets provides an intuitive and interactive way to explore and analyze the data. Figure 2.5 shows an example of a DnM plot displaying the Iris dataset.

DnM can comfortably accommodate a larger number (say 10 or 20) of dimensions. Like other techniques, it too can become cluttered if the number of records becomes too large. A particular advantage of DnM is its flexibility. Users can adjust the strength of the magnets to emphasize or de-emphasize different dimensions and can manipulate the position and size of the magnets to highlight specific patterns in the data. However, a potential limitation of DnM is that the technique can be sensitive to the initial configuration of the particles and magnets. This means that users may need to experiment with different settings to create an informative layout.



**Figure 2.5:** DnM plots of the Iris dataset. [Drawn by Özbej Golob using RPE [Neuhold et al. 2020].]

## 2.6 Similarity Maps

Similarity maps are projections of multidimensional datasets to two (or sometimes three) dimensions. Multidimensional datasets, by definition, have a large number of dimensions (often hundreds or thousands), which presents many computational and mathematical challenges. Projection techniques are used to reduce the number of dimensions in the data while attempting to preserve distances between items as much as possible. Items that are close in the multidimensional space should also be close in the resulting two-dimensional projection space. One of the key advantages of similarity maps is that they can accommodate any number of dimensions, which are then mapped into two (or three) dimensions. Projections can further be split into *linear* and *non-linear* projections.

A linear projection is a transformation that can be represented by a linear function. This means that the output of a linear projection is a linear combination of the input dimensions, where each dimension is multiplied by a scaling factor and then added together. In other words, a linear projection involves scaling and rotating the data without distorting it, and its output could be used to label the projected axes. Linear projections are useful for reducing the dimensionality of data while preserving its structure and relationships between dimensions. Common examples of linear projections include Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).

A non-linear projection, on the other hand, involves more complex transformations that cannot be represented by a linear function. This means that the output of a non-linear projection is a complex combination of the input dimensions that may involve multiplication, exponentiation, or other non-linear operations. Non-linear projections can distort the data to reveal patterns and relationships that may not be apparent in the original multidimensional space. Non-linear projections are particularly useful for data that exhibits complex and non-linear relationships between dimensions. Examples of non-linear projections include Multi-Dimensional Scaling (MDS), t-distributed Stochastic Neighbor Embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP).

### 2.6.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a linear projection technique [Abdi and Williams 2010]. PCA analyzes the covariance matrix of the original dataset, which describes the relationship between the different dimensions. By calculating the eigenvectors and eigenvalues of the covariance matrix, PCA identifies the directions in which the data varies the most, the principal components.

The principal components are ranked based on the amount of variance they account for. The first principal component explains the largest amount of variance, followed by the second principal component, and so on. By keeping only the principal components that explain the majority of the variance in the data, the dimensionality of the dataset can be reduced, while still retaining the most important information. Typically, the first two principal components are then mapped to the x and y axes of the similarity map. Since PCA does not involve any randomization, it is deterministic and produces the same layout every time it is run.

### 2.6.2 Multi-Dimensional Scaling (MDS)

Multi-Dimensional Scaling (MDS) is a non-linear projection technique [Morrison et al. 2003]. MDS is used to visualize the similarity or dissimilarity between different objects or observations. The technique transforms a multidimensional dataset to a lower-dimensional space, usually two or three dimensions, while preserving the relationships between the records.

MDS works by first calculating the pairwise distances or dissimilarities between all the objects in the dataset. These distances could be based on any metric, such as Euclidean distance, correlation distance, or other similarity measures. Once the distance matrix is constructed, MDS tries to find a configuration of points in a lower-dimensional space that best reproduces the distances or dissimilarities between the original objects in the higher-dimensional space. This is done by minimizing a cost function, such as stress or error, which measures the discrepancy between the original pairwise distances and the distances between the projected points.

MDS can be classified into two main types: metric and non-metric. In metric MDS, the pairwise distances between the objects are preserved exactly, while in non-metric MDS, only the rank order of the distances is preserved. Non-metric MDS is often used when the underlying distance metric is unknown or not easily quantifiable.

Depending on the implementation, MDS can be either deterministic or non-deterministic. Having the technique be deterministic and thus produce the same layout every time it is run, is preferable in terms of user experience.

### 2.6.3 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear projection technique [Van der Maaten and Hinton 2008; Wattenberg et al. 2016]. t-SNE uses a probabilistic model to preserve the local structure of the data in the multidimensional space and project the data onto a lower-dimensional space, usually two or three dimensions.

The t-SNE algorithm works by first calculating the pairwise similarities between all the records in the multidimensional space. This is typically done using a Gaussian kernel, which measures the similarity between two points as a function of their Euclidean distance. The similarities are then used to construct a probability distribution for each point that defines the likelihood of finding other points nearby. In the low-dimensional space, t-SNE tries to find a configuration of points that preserves the pairwise similarities between the multidimensional records as closely as possible. The algorithm does this by minimizing a cost function that measures the divergence between the probability distributions in the high-dimensional and low-dimensional spaces. The optimization is performed using gradient descent, which iteratively

updates the position of each point in the low-dimensional space until the cost function is minimized. The t-SNE algorithm has a number of tunable parameters, the most important of which is *perplexity*, which steers the tightness of the resulting clusters.

One of the key features of t-SNE is that it uses a Student's t-distribution to model the probability distribution in the low-dimensional space. This allows t-SNE to emphasize the differences between nearby records and de-emphasize the differences between distant records, which helps prevent crowding and distortion in the final visualization. Again, depending on the implementation and the choice of initial layout, t-SNE can be either deterministic or non-deterministic.

#### 2.6.4 Uniform Manifold Approximation and Projection (UMAP)

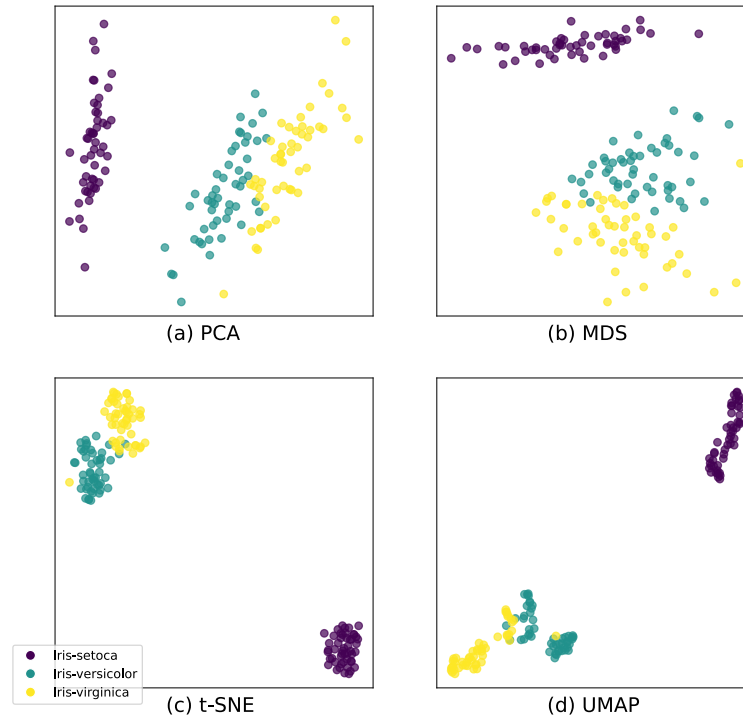
Uniform Manifold Approximation and Projection (UMAP) is a non-linear projection technique [McInnes, Healy and Melville 2018]. UMAP is based on manifold learning and topological data analysis. The technique uses three assumptions about the data to construct a manifold with a fuzzy topological structure. The embedding is set to a low-dimensional projection that best preserves the original fuzzy structure.

UMAP works by creating a low-dimensional representation of the data while preserving the global structure of the data. This is achieved by modeling the data as a multidimensional manifold, which is a mathematical object that represents the underlying structure of the data. UMAP then finds a low-dimensional embedding of the manifold that preserves the local structure of the data. The UMAP algorithm consists of several steps. First, a nearest-neighbor graph is constructed by connecting each record to its nearest neighbors. This graph is then used to create a fuzzy simplicial set, which is a mathematical object that captures the local structure of the data. The fuzzy simplicial set is then transformed into a low-dimensional representation using a stochastic gradient descent algorithm.

UMAP has several advantages over other dimensionality reduction techniques. For example, UMAP can preserve both the global and local structure of the data, which means that it can be used for both exploratory data analysis and machine learning tasks. Traditional UMAP implementations use random sampling to speed up the optimization, but the random seed can be fixed to produce a deterministic layout.

#### 2.6.5 Comparison of Similarity Mapping Techniques

Figure 2.6 shows the four similarity mapping techniques applied to the Iris dataset. All four techniques separate the Iris Setosa class from the other two classes, which cannot be clearly separated from each other. t-SNE and UMAP generate dense clusters, while PCA and MDS clusters are more spread out. The t-SNE example uses a perplexity value of 30; using a different value would result in a significantly different result.



**Figure 2.6:** Comparison of four similarity mapping techniques on the Iris dataset. [Drawn by Özbej Golob using Python.]

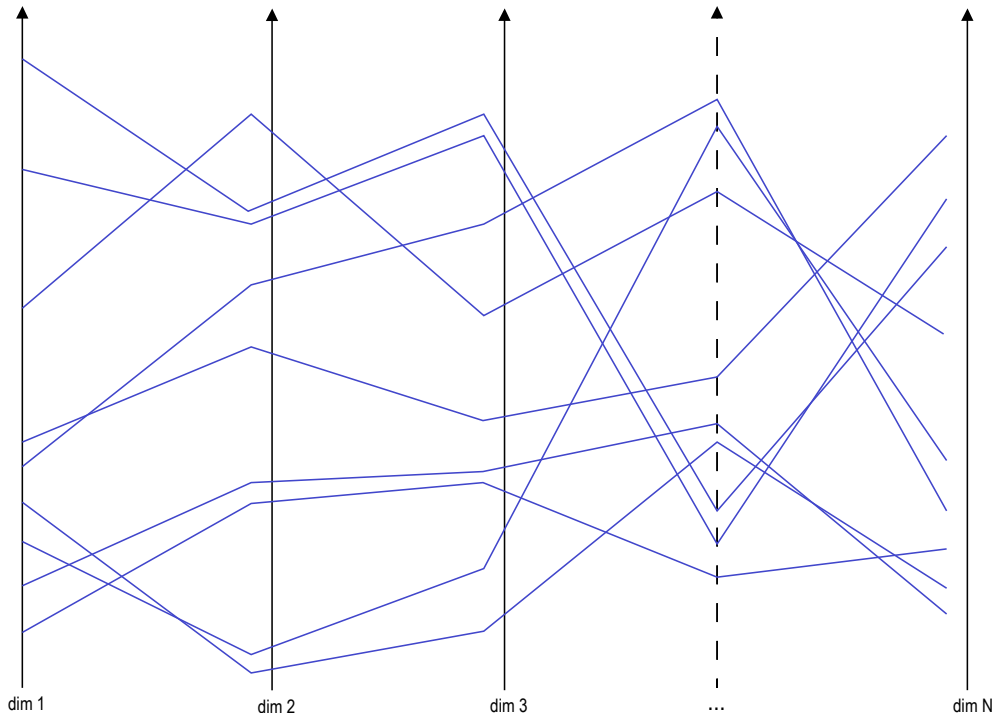
## 2.7 Parallel Coordinates

Parallel coordinates are a type of chart used to visualize multidimensional data [Inselberg and Dimsdale 1990; Inselberg 2009]. Each dimension is represented by a vertical axis arranged in parallel. Each record is represented by a horizontal polyline. The position where a polyline touches an axis indicates the value of the record for that dimension. Parallel coordinates allow multiple dimensions to be compared and analyzed simultaneously. Figure 2.7 shows an example of a generic parallel coordinates plot displaying records with multiple dimensions. Figure 2.8 shows a parallel coordinates plot for the Iris dataset.

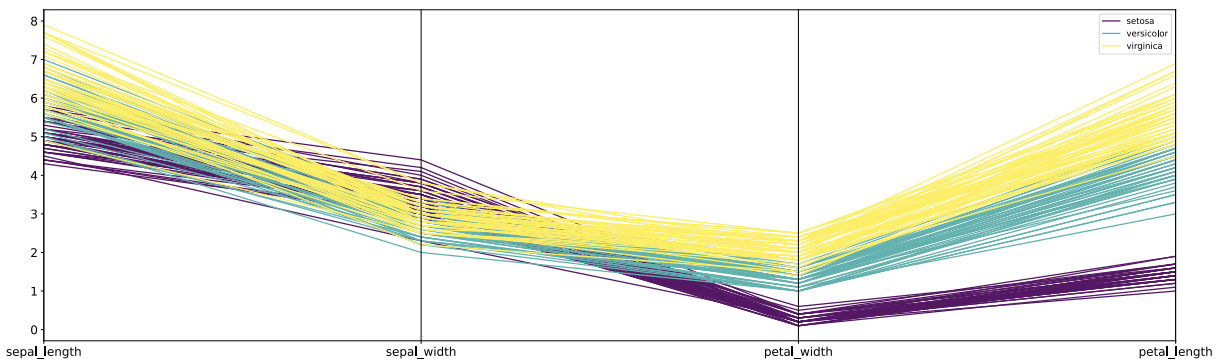
In a parallel coordinates plot, the correlation between two adjacent dimensions can be observed by the general direction and pattern of the lines connecting the two axes. When two adjacent dimensions are positively correlated, the lines connecting the points on these axes tend to move in the same direction. That is, as one variable increases, the other variable also increases. This is visually represented by lines that generally slope in the same direction between the two axes. When two adjacent dimensions are negatively correlated, the lines connecting the points on these axes tend to move in opposite directions. That is, as one variable increases, the other variable decreases. This is visually represented by lines that generally cross each other or form an X pattern between the two axes.

One of the key advantages of a parallel coordinates plot is that it can comfortably accommodate a larger number (say 10 or 20) of dimensions without scrolling, and considerably more (50 or 100 or more) if horizontal scrolling is implemented. However, like other techniques, it can become cluttered if the number of records becomes too large.

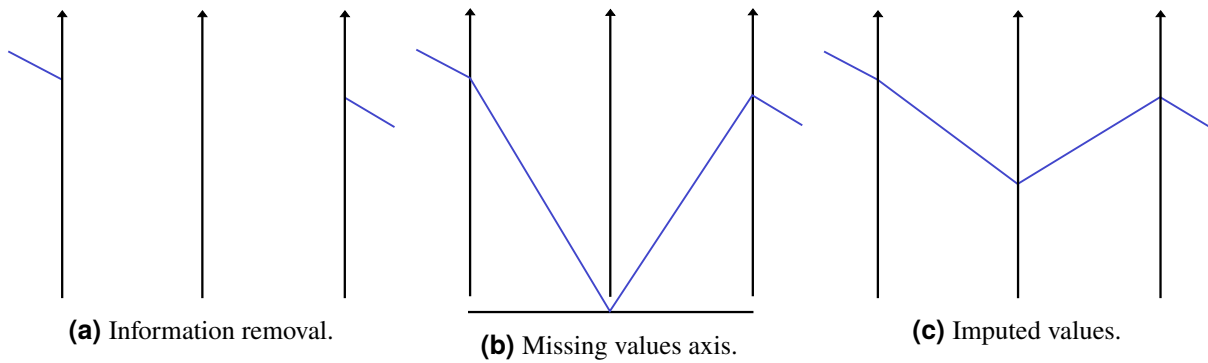
Several approaches for handling missing data have been proposed. Bäuerle et al. [2022] compared three such approaches, shown in Figure 2.9. They performed a quantitative user study and concluded that the best approaches for four representative tasks are: having a horizontal missing values axis for value estimation, information removal for outlier detection, information removal or imputed (estimated) values for trend estimation, and that no approach is significantly better than others for value retrieval.



**Figure 2.7:** A parallel coordinates plot showing dimensions as vertical axes and records as horizontal polylines. [Drawn by Ožbej Golob using Adobe Illustrator.]

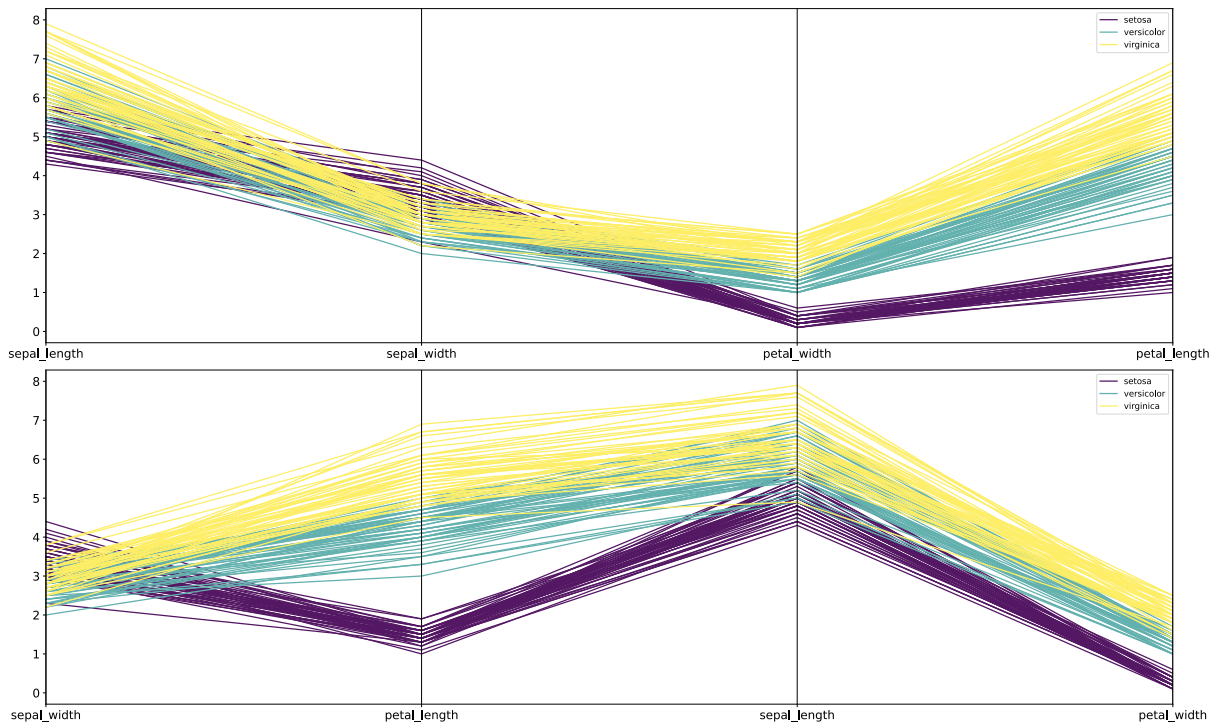


**Figure 2.8:** A parallel coordinates plot displaying the Iris dataset. [Drawn by Ožbej Golob using Python.]



**Figure 2.9:** Three approaches for handling missing parallel coordinates data suggested by Bäuerle et al. [2022]. [Drawn by Ožbej Golob using Adobe Illustrator.]





**Figure 2.10:** A parallel coordinates matrix (PCM) for the four numerical dimensions of the Iris dataset. [Drawn by Ožbej Golob using Python.]

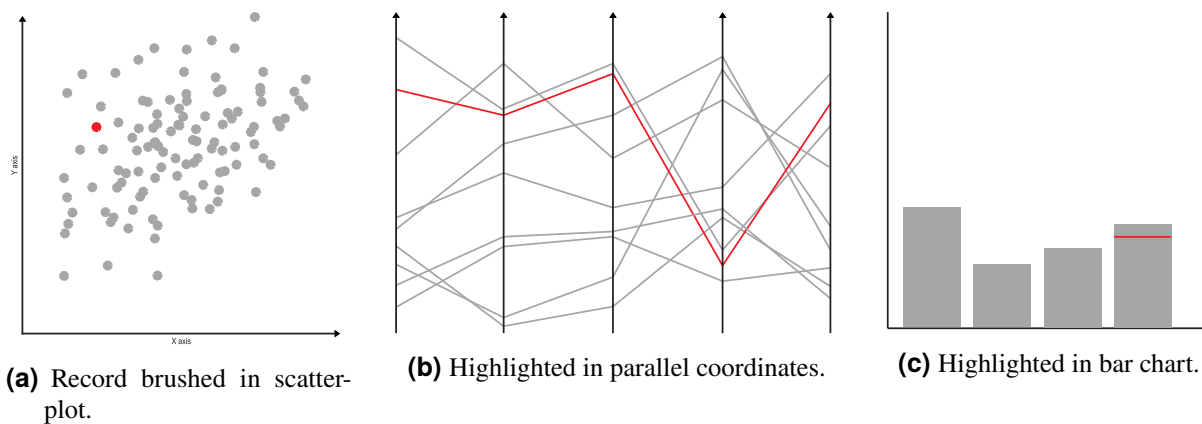
## 2.8 Parallel Coordinates Matrix

To visually analyze the relationship between two dimensions, the dimensions must be adjacent to one another. A parallel coordinates matrix (PCM) is a counterpart to a SPLOM. It displays permutations of parallel coordinates to show all pairwise adjacent dimensions [Heinrich et al. 2012]. A PCM allows users to compare each pair of these dimensions across different parallel coordinates plots arranged in a matrix, making it easier to detect how changes in one variable might relate to changes in another.

As discussed by Wegman [1990], the number of permutations needed to ensure that all possible pairwise combinations are shown is  $\frac{n}{2}$  ( $n$  even) or  $\frac{n+1}{2}$  ( $n$  odd), or in other words  $\lceil \frac{n}{2} \rceil$ , where  $n$  is the number of dimensions. For a dataset with six dimensions, numbered from one to six, the following three permutations would suffice: [1 2 6 3 5 4], [2 3 1 4 6 5], and [3 4 2 5 1 6]. For a dataset with four dimensions, two permutations would suffice: [1 2 4 3] and [2 3 1 4], as shown in Figure 2.10 for the four numerical dimensions of the Iris dataset.

## 2.9 Brushing and Linking

Brushing refers to the process of selecting and highlighting one or more records in a visualization. Linking refers to the process of synchronizing multiple visualizations, such that selections made in one visualization are highlighted in the other visualizations too. This allows the user to explore the data from different perspectives and understand how sets of records of interest are positioned in the various visualizations. Figure 2.11 shows an example of brushing and linking. The user selected a single record in the scatterplot, which is then also highlighted in the parallel coordinates plot and bar chart.



**Figure 2.11:** An example of brushing and linking. The user selected a single record in the scatterplot, which is then highlighted in red in all other visualizations. [Drawn by Ožbej Golob using Adobe Illustrator.]

## 2.10 Grouping and Labeling

Grouping refers to the process of identifying and collecting similar records, while labeling refers to the process of naming the groups. Grouping and labeling help organize and structure data records, allowing for more accurate and meaningful analysis of the data, as similar records can be grouped together and analyzed in relation to one another. It also allows a Machine Learning (ML) model to understand the context of the data and make more accurate predictions with greater interpretability.

Manual grouping and labeling refers to the process of organizing data records manually by a human, which can be time-consuming and labor-intensive. Automated clustering uses mathematical methods to identify which objects in a given data set are similar. Similar records are automatically grouped into clusters. This allows analysts to identify common patterns and trends within the data, and to gain a better understanding of the relationships between the objects in the dataset [Romesburg 1984].

## Chapter 3

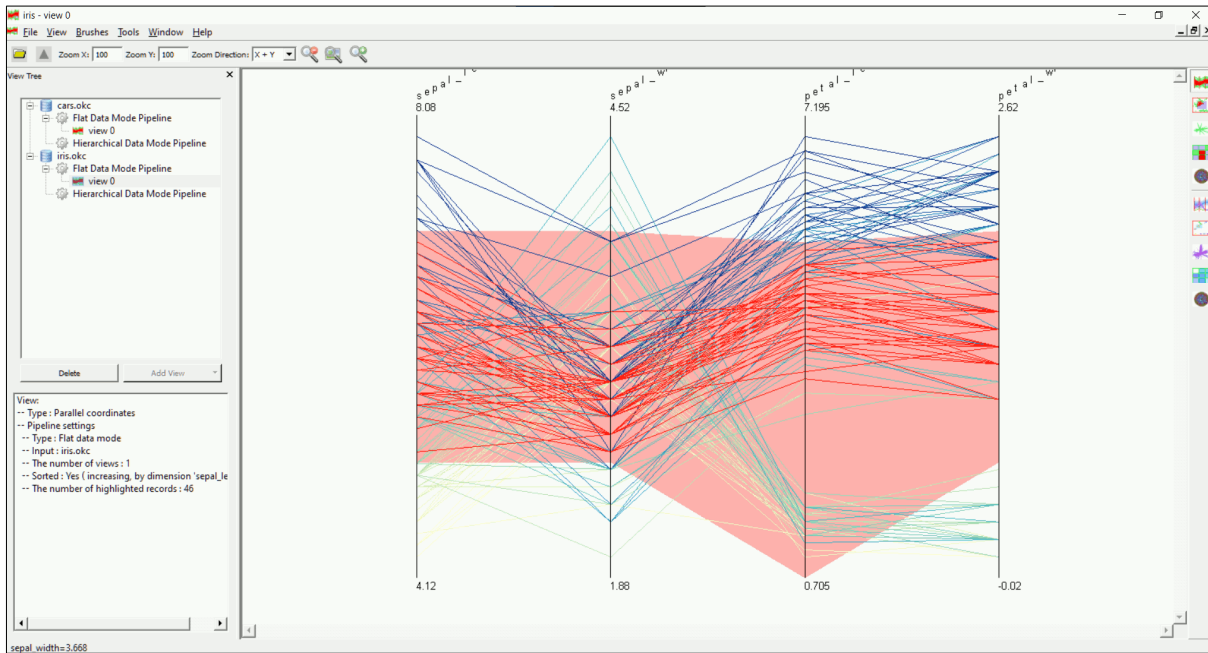
# Tools for Multidimensional Visual Analysis

Multidimensional visual analysis software tools are designed to help analysts explore and analyze complex datasets. These programs typically include a wide range of tools and features that allow users to create and manipulate graphical representations of the data, such as scatterplots, parallel coordinates, and similarity maps. Using these tools, analysts can quickly and easily explore the data and gain insights that might not be immediately apparent from looking at the raw data. Multidimensional visual analysis software is commonly used in fields such as business, finance, and scientific research to uncover hidden trends in the data and help make data-driven decisions. In this chapter, some popular multidimensional visual analysis software tools are reviewed.

### 3.1 XMDV

XMDV tool is a software package for the interactive visual exploration of multidimensional datasets [Ward 1994; Lab 2024; Zhao 2021]. XMDV is written in Qt and Eclipse CDT and is available as free, open-source software. It was initially released in 1994 and last updated on 24 Sept 2021. XMDV is available for Windows, macOS, and Linux. XMDV can load datasets in .csv format and its own custom .okc format. There is no option to export an uploaded dataset.

XMDV displays the data in the following visualizations: scatterplot matrix, parallel coordinates, star glyphs, dimensional stacking, and tree maps. XMDV also supports many interaction modes and tools, including brushing and linking. Figure 3.1 shows the XMDV tool. Visualizations can be exported in .bmp, .jpg, and .png formats.



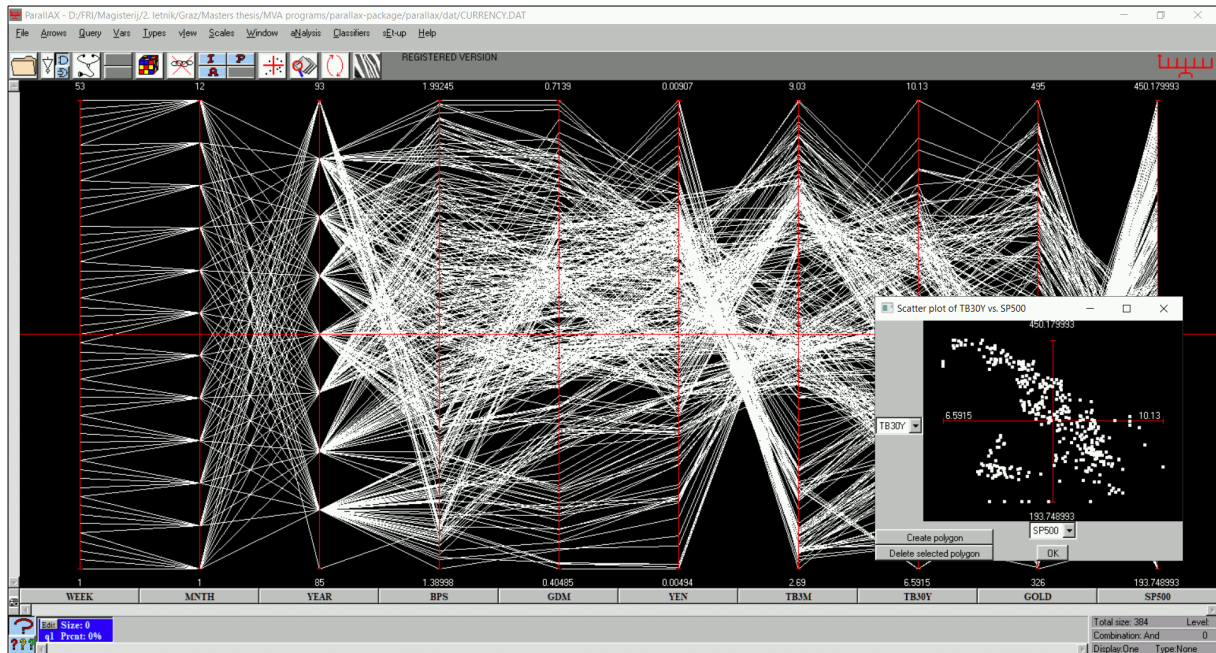
**Figure 3.1:** XMDV displaying the Iris dataset [Fisher 1936]. [Screenshot taken by Ožbej Golob using XMDV software [Ward 1994].]

## 3.2 Parallax

Parallax is a tool for effectively analyzing multidimensional datasets and discovering patterns, properties, and relations in data [T. Avidan and S. Avidan 1999; Inselberg 2009, Chapter 10]. Parallax was developed by a small Israeli software company, MDG, in collaboration with Alfred Inselberg. It was initially released in 1994 and was available as commercial, closed-source software. Parallax can load datasets in its own custom .dat format. The uploaded dataset can be exported in the same format.

The main part of Parallax is a powerful parallel coordinates visualization, which enables queries. The results of queries can be grouped and then shown separately or in combination with other queries. Parallax provides the following visualizations: scatterplot, parallel coordinates, and distribution (histogram). Parallax does not support brushing and linking. Figure 3.2 shows the Parallax tool. There is no option to export visualizations.

Alone among the tools, Parallax provides sophisticated support for defining and manipulating sets of records, in essence looking for formulaic rule-based definitions with which to specify particular classes of records.

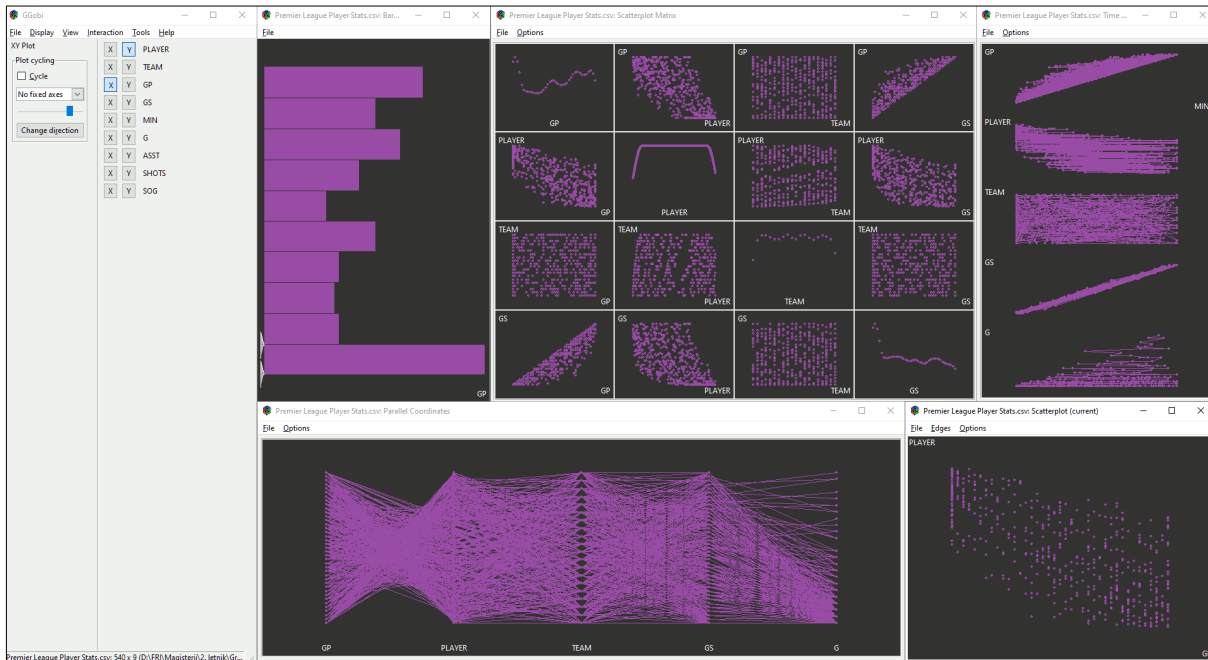


**Figure 3.2:** Parallax displaying a financial dataset. [Screenshot taken by Ožbej Golob using Parallax software [T. Avidan and S. Avidan 1999].]

### 3.3 GGobi

GGobi is a visualization program for exploring multidimensional data [Cook et al. 2007; GGobi 2024]. GGobi is written in C and is available as free, open-source software. It was initially released in 1999 and last updated on 10 Jun 2012. GGobi is available for Windows, macOS, and Linux. GGobi can load a selection of pre-prepared datasets. It is also possible to load datasets in .csv and .xml formats. The uploaded dataset can be exported to .csv and .xml formats.

GGobi provides dynamic and interactive graphics as tours, where data is displayed in an animation. The following visualizations are available: scatterplot, scatterplot matrix, parallel coordinates, time series, and distributions (histograms). GGobi supports limited brushing. The visualizations offer limited interactivity and interpretability and are not closely connected. Figure 3.3 shows the GGobi tool. There is no option to export visualizations.



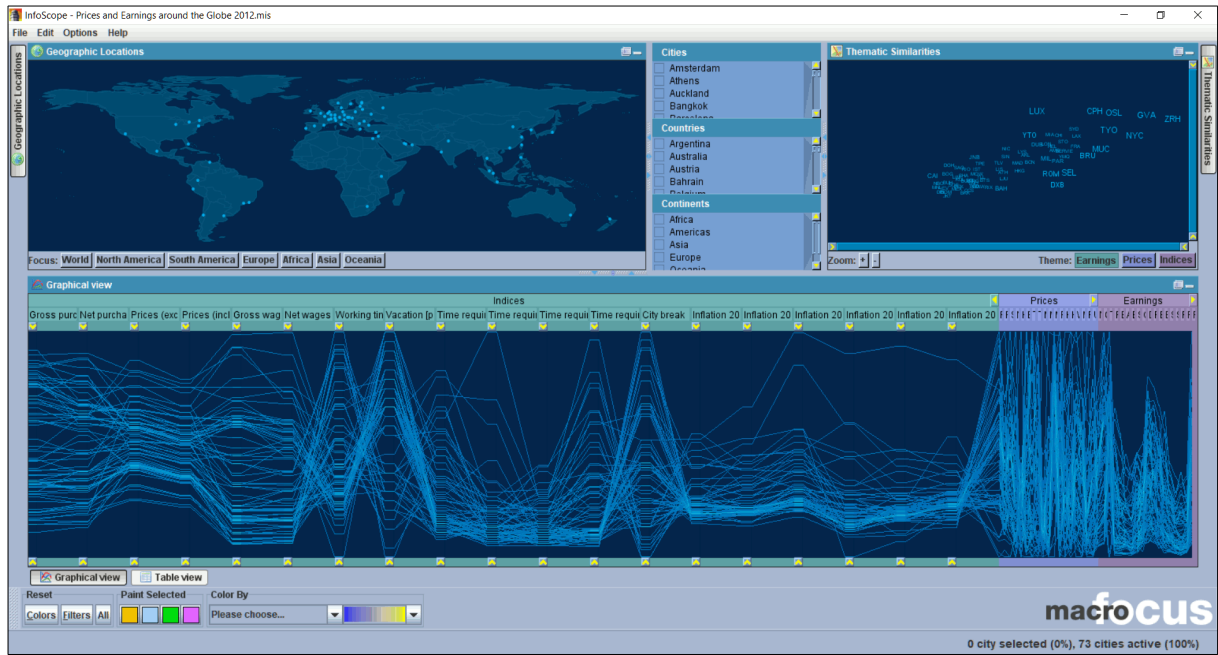
**Figure 3.3:** GGobi displaying the Premier League dataset [Samariya 2020]. [Screenshot taken by Ožbej Golob using GGobi software [Cook et al. 2007].]

### 3.4 InfoScope

InfoScope is an interactive visualization program designed for exploring and presenting large datasets [Macrofocus 2015; Girardin and Brodbeck 2001; Brodbeck and Girardin 2003]. InfoScope is available as free software, but is not open-source. It was initially released in 2001 and last updated on 19 Aug 2015. InfoScope is available for Windows, macOS, and Linux.

InfoScope can load a selection of pre-prepared datasets, mainly from the finance sector. It is also possible to load datasets, but only in the system's custom `.mis` format. A dataset can be exported in `.csv`, `.csv.gz`, `.raw`, `.tab`, `.tsv`, `.tsv.gz`, `.txt`, and `.txt.gz` formats.

InfoScope provides the following visualizations: carto plot, similarity map, parallel coordinates, and table. InfoScope supports brushing and linking, so all visualizations are highly interactive and tightly linked. Users can obtain exact record values by inspecting the records, and select a subset of records with range sliders. InfoScope supports manual grouping of records by color. Figure 3.4 shows InfoScope. It later evolved into the tool High-D. A screenshot of the whole user interface can be saved as `.pdf` or directly printed, but individual visualizations can not be exported to `.svg`.

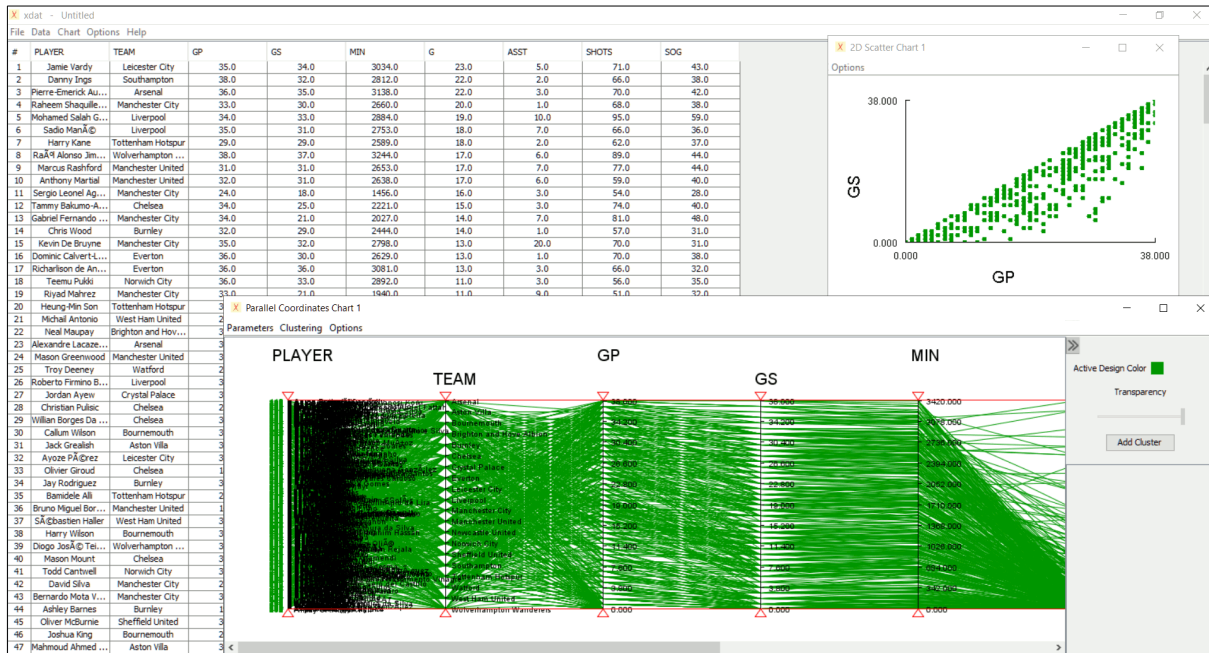


**Figure 3.4:** InfoScope displaying the Prices and Earnings around the Globe 2012 dataset [UBS 2012]. [Screenshot taken by Ožbej Golob using InfoScope software [Macrofocus 2015].]

### 3.5 XDAT

XDAT is a multidimensional data analysis tool designed to help users quickly and easily extract valuable insights from large, complex datasets with many dimensions [de Rochefort 2024; de Rochefort 2020]. XDAT is written in Java and is available as free software. It was initially released in May 2010 and last updated on 26 Aug 2020. XDAT is available for Windows, macOS, and Linux. XDAT can load datasets in .csv format. The current session can be saved to a custom .ses file and later re-imported with updated visualizations.

The following visualizations are available: parallel coordinates, scatterplot, and table. All of the visualizations are interconnected through standard brushing and linking so that any changes or selections made in one visualization are reflected in all of the other visualizations. Figure 3.5 shows the XDAT tool. Visualizations can be exported in .png format.



**Figure 3.5:** XDAT displaying the Premier League dataset [Samariya 2020]. [Screenshot taken by Ožbej Golob using XDAT software [de Rochefort 2024].]

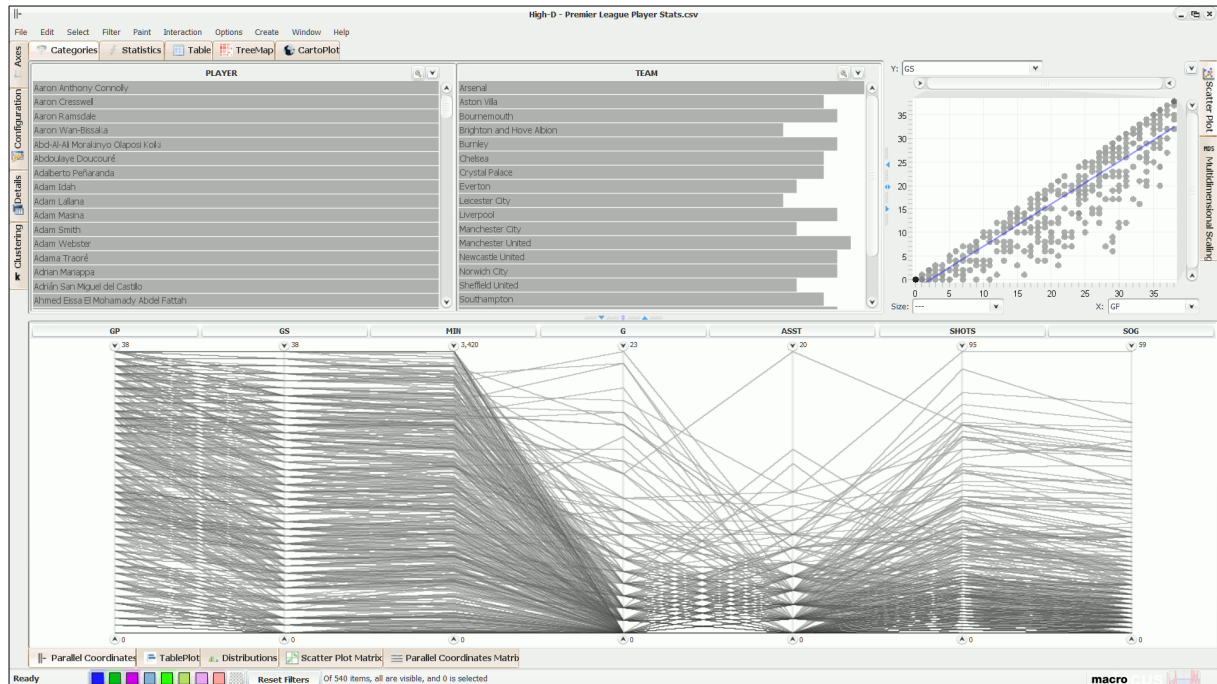
### 3.6 High-D

High-D is the successor to InfoScope [Macrofocus 2024]. It offers similar functionality with some improvements and added visualizations. High-D is a powerful visualization program for identifying trends, relationships, and anomalies in large and complex datasets. It is focused around an interactive parallel coordinates plot. High-D is available as commercial software for US\$ 199 and with a 30-day free evaluation period. It was initially released in Sept 2013 and last updated on 08 Jul 2024. High-D is available for Windows, macOS, and Linux.

High-D can load a selection of pre-prepared datasets, mainly from the finance sector. It is also possible to load datasets in .csv, .json, .tab, .tsv, .txt, .xml, a number of database formats, and custom Macrofocus formats. The uploaded dataset can be exported in the same formats.

High-D provides the following visualizations: parallel coordinates, parallel coordinates matrix, table plot, distributions, scatterplot matrix, scatterplot, similarity map (Sammon, Spring, t-SNE, and PCA), tree map, and carto plot. High-D supports manual grouping of data by color and automated clustering with a k-means algorithm. High-D also supports brushing and linking, so all visualizations are highly interactive and tightly linked. Users can obtain exact record values by inspecting the records, and can select a subset of records with range sliders. Figure 3.6 shows the High-D tool. Visualizations can be exported in .emf, .gif, .jpg, .png, .ps, .svg, .tiff, and .webp formats.



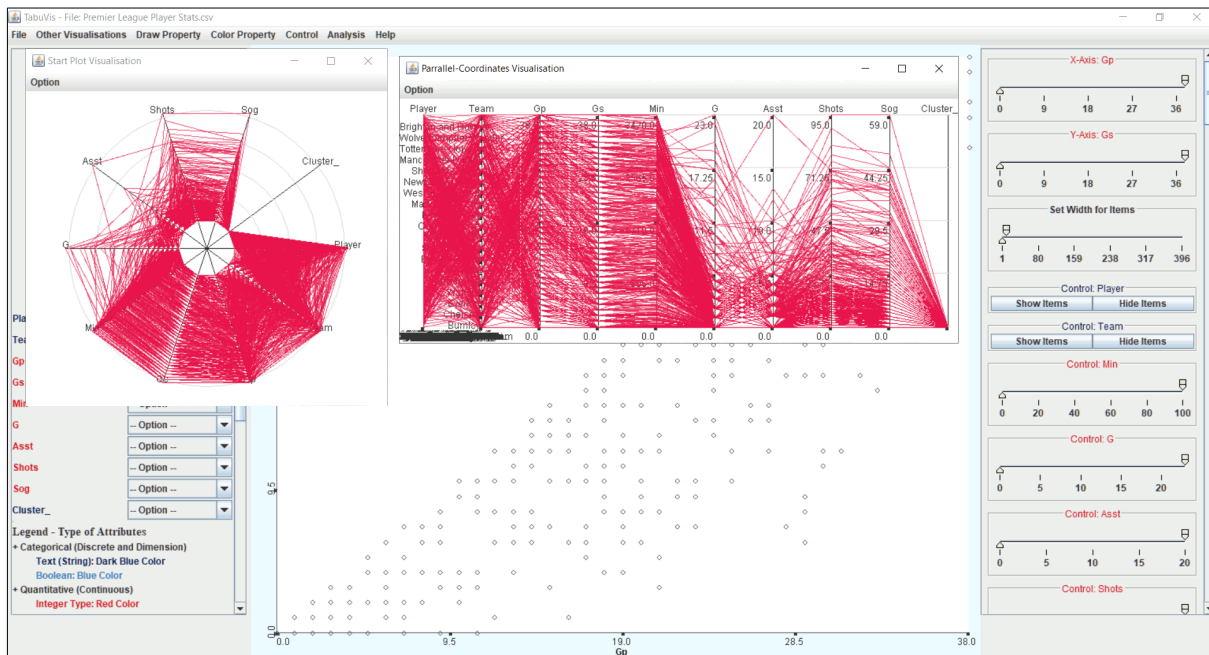


**Figure 3.6:** High-D displaying the Premier League dataset [Samariya 2020]. [Screenshot taken by Ožbej Golob using High-D software [Macrofocus 2024].]

### 3.7 TabuVis

TabuVis is a flexible and customizable visual analytics system for multidimensional data [Nguyen et al. 2023; Nguyen 2024]. Visualizations can be customized by domain experts to suit the specific needs of the data being analyzed. TabuVis is written in Java and is available as free software. It was initially released in May 2013 and last updated on 03 May 2023. TabuVis is available for Windows, macOS, and Linux. TabuVis can load a selection of pre-prepared datasets. It is also possible to load datasets in .csv format. The current session can be saved to a custom file and later re-imported.

TabuVis displays data in separate visualizations. TabuVis includes various features for analyzing data, such as the ability to process data, add automatic marks, create custom interactive visualizations, and filter the data. These features are designed to support the entire data analysis process. TabuVis displays the data in the following visualizations: scatterplot, parallel coordinates, and star plot. TabuVis does not support brushing and linking. Figure 3.7 shows the TabuVis tool. There is no option to export visualizations.



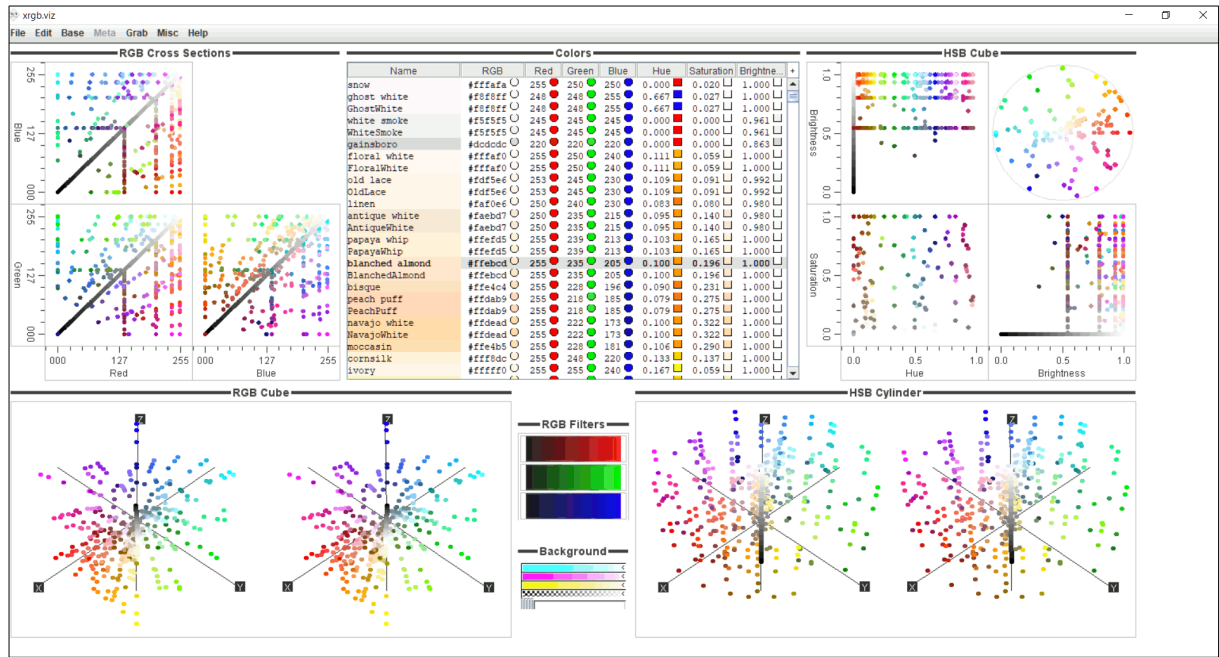
**Figure 3.7:** TabuVis displaying the Premier League dataset [Samariya 2020]. [Screenshot taken by Özbej Golob using TabuVis software [Nguyen et al. 2023].]

### 3.8 Improvise

Improvise is a program that allows users to create and interact with visualizations that are linked together in various ways [Weaver 2020]. Improvise is written in Java and is available as a free, open-source software. It was initially released in 2014 and last updated on 28 Oct 2020. Improvise is available for Windows, macOS, and Linux.

Improvise can load datasets in its own custom `.viz` format. There is no option to export the uploaded dataset.

The program uses a shared-object coordination model and a declarative visual query language to give users control over how data is displayed in multiple visualizations. This allows users to create visualizations with a variety of coordination patterns. Improvise also has a user interface that allows users to build and explore visualizations in a live environment, making it easy to modify visualizations as needed. The goal of Improvise is to provide a high level of coordination flexibility while also being easy to use. Figure 3.8 shows the Improvise tool. There is no option to export visualizations.

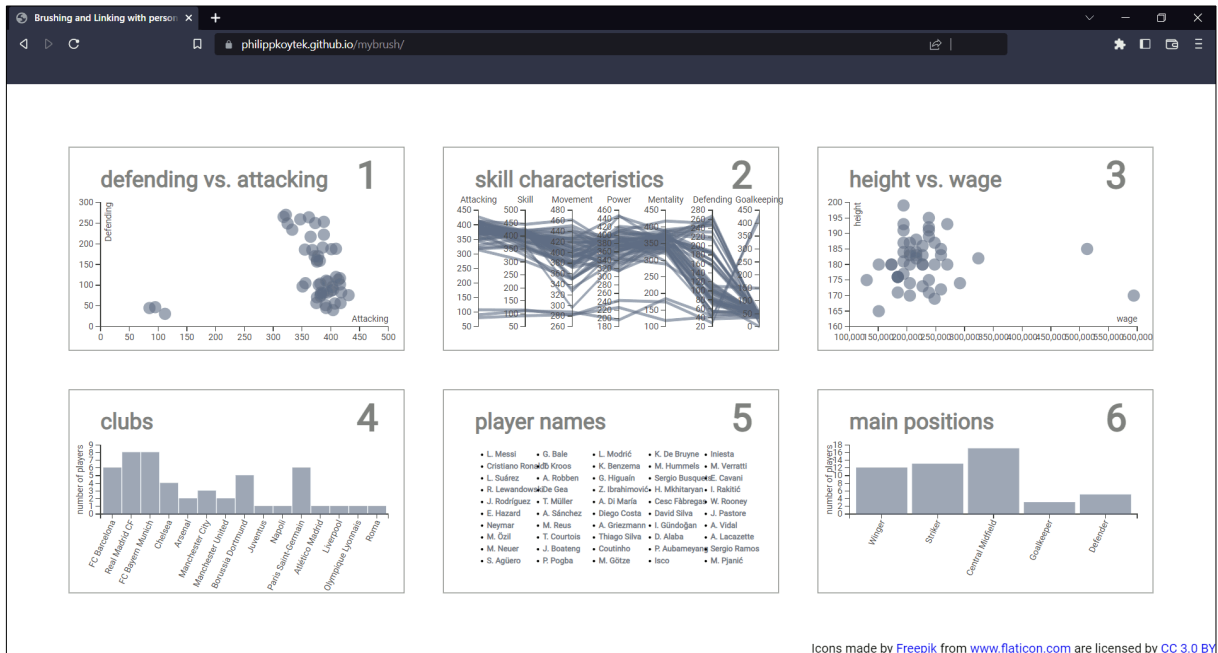


**Figure 3.8:** Improvise displaying a custom XRGB dataset. [Screenshot taken by Ožbej Golob using Improvise software [Weaver 2020].]

### 3.9 MyBrush

MyBrush is an experimental application that allows users to customize and control the brushing and linking process in their visualizations [Koytek et al. 2017; Koytek 2017]. It provides flexibility by allowing users to specify the source, link, and target of multiple brushes, and supports a variety of visualization types and multiple simultaneous brushes. Improvise is written in JavaScript and is available as a free, open-source web application. It was initially released in Jun 2016 with the last update issued on 22 Sept 2017.

MyBrush is experimental and offers limited functionality. Its purpose is to explore brushing and linking functionality. There is no option to load a dataset, but the user can explore a predetermined set of data with the following visualizations: scatterplot, parallel coordinates, and bar plot. Any changes or selections made in one of the visualizations are reflected in all of the other visualizations because they are all interconnected through standard brushing and linking. Figure 3.9 shows the MyBrush tool. There is no option to export visualizations.



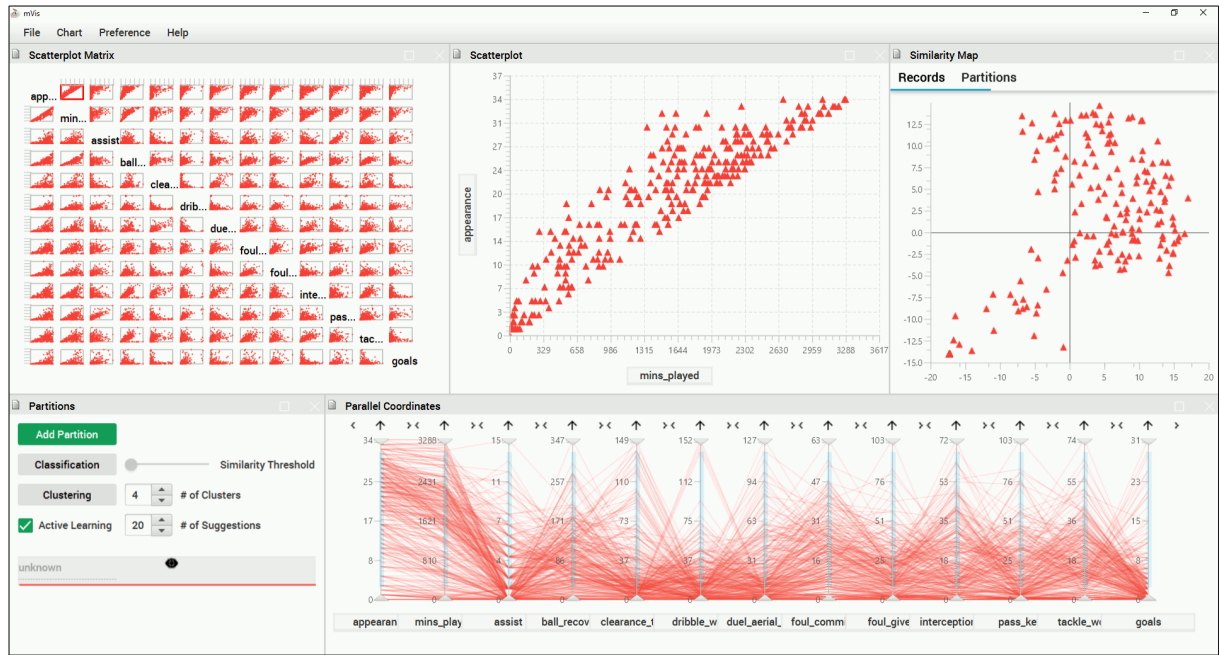
**Figure 3.9:** MyBrush displaying a custom football players dataset. [Screenshot taken by Ožbej Golob using MyBrush software [Koytek et al. 2017].]

### 3.10 mVis

mVis is a visual analytics tool for visualizing multidimensional data [Chegini et al. 2019; Chegini 2021]. mVis is written in Java and is available as free, open-source software. It was initially released in Jul 2020 and last updated on 20 Jan 2021. mVis is available for Windows, macOS, and Linux. mVis can load datasets in .csv format. There is no option to export the uploaded dataset.

mVis provides an overview of global relationships between objects by using multiple visualizations to show different aspects of the data at the same time. mVis consists of four data visualization visualizations: scatterplot matrix, scatterplot, similarity map (t-SNE, PCA, and MDS), and parallel coordinates. mVis also has a component for controlling data partitions. All of the visualizations are interconnected through standard brushing and linking so that any changes or selections made in one visualization are reflected in all of the other visualizations. Additionally, the user can close, rearrange, or expand any visualization as needed.

mVis supports the interactive visual labeling of records in a dataset, both manually with the linked visualizations and by running built-in clustering and active learning methods. An analyst can take a dataset and efficiently partition it into classes by labeling the records. Such a labeled dataset can then be used as a training dataset for machine learning. Figure 3.10 shows the mVis tool. There is no option to export visualizations.



**Figure 3.10:** mVis displaying a custom football players dataset. [Screenshot taken by Ožbej Golob using mVis software [Chegini et al. 2019].]

	XMDV	Parallax	GGobi	InfoScope	XDAT	High-D	TabuVis	Improvise	MyBrush	mVis
Initial Release:	1994	Mar 1999	1999	2001	May 2010	Sept 2013	May 2013	2014	Jun 2016	Jul 2020
Last Update:	Sept 2021	?	Jun 2012	Aug 2015	Aug 2020	Jul 2024	May 2023	Oct 2020	Sept 2017	Jan 2021
License:	Free, open-source	Commercial	Free, open-source	Free demo	Free	Commercial	Free	Free, open-source	Free, open-source	Free, open-source
Systems:	Win, MacOS, Linux	Win, MacOS, Linux	Win, MacOS, Linux	Win, MacOS, Linux	Win, MacOS, Linux	Win, MacOS, Linux	Win, MacOS, Linux	Win, MacOS, Linux	Web Browser	Win, MacOS, Linux
Language:	Qt	?	C	Java	Java	Java	Java	Java	JavaScript	Java
Installation:	Local	Local	Local	Local	Local	Local	Local	Local	Online	Local

**Table 3.1:** Overview of reviewed tools.

### 3.11 Comparison of Tools

The software tools covered in this chapter are compared in two tables: Table 3.1 shows general information for each tool, while Table 3.2 compared the features provided by each tool.

Feature	XMDV	Parallax	GGobi	InfoScope	XDAT	High-D	TabuVis	Improvise	MyBrush	mVis
Load Custom Datasets:	✓	✓	✓	✓	✓	✓	✓	✓		✓
Export Dataset:		✓	✓	✓	✓	✓	✓			
Export Visualizations:	✓			✓	✓	✓				
Brushing:	✓		✓	✓	✓	✓			✓	✓
Linking:	✓		✓	✓	✓	✓			✓	✓
Manual Grouping:	✓	✓		✓	✓	✓	✓			✓
Automated Clustering:		✓				✓	✓			✓
Table View:				✓	✓	✓		✓		
Scatterplot:	✓	✓	✓		✓	✓	✓	✓	✓	✓
Scatterplot Matrix:	✓		✓			✓		✓		✓
Parallel Coordinates:	✓	✓	✓	✓	✓	✓	✓		✓	✓
Parallel Coordinates Matrix:						✓				
Similarity Map:				✓		✓	✓	✓		✓
Time Series:			✓					✓		
Distributions:		✓	✓			✓		✓	✓	
Table Plot:						✓		✓		
Tree Map:	✓					✓		✓		
Carto Plot:				✓		✓		✓		

**Table 3.2:** Comparison of reviewed tool features.

# Chapter 4

## Modern Web Technologies

Modern web technologies have revolutionized the way users interact with the internet, transforming static web pages into dynamic and interactive experiences. With advancements in HTML5, CSS3, and JavaScript frameworks, developers can create responsive, high-performance web applications that work seamlessly across various devices and platforms. This chapter gives an overview of web applications, popular frontend development frameworks, web graphics rendering technologies, and desktop development libraries.

### 4.1 Web Applications

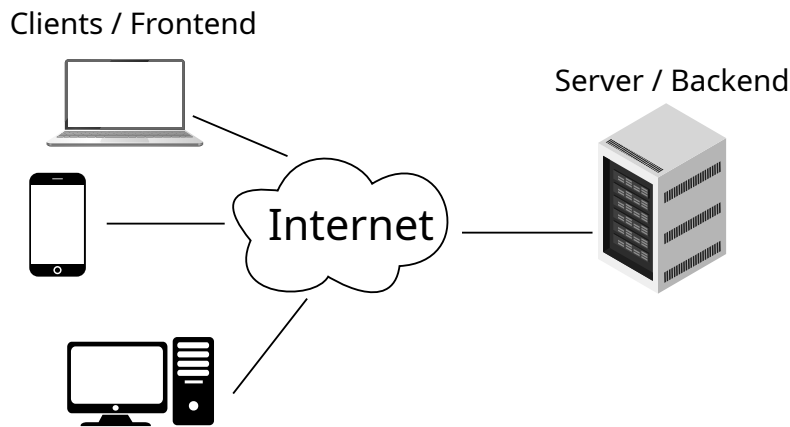
A web application is an application that can be accessed over the internet through a web browser and does not need to be downloaded or installed. Web applications offer several benefits, such as multiple user access, no need to download the application, cross-platform access, and access through web browsers. Additionally, web applications often have shorter development cycles and smaller development teams, making them more cost-effective and efficient to build and maintain. Web applications usually follow the client-server model, where the client is considered the frontend and the server is considered the backend [Berson 1996]. This is illustrated in Figure 4.1.

#### 4.1.1 Frontend

The frontend of a web application is responsible for the user interface and user experience [Ahmed 2024b]. It is the part of the application that the user interacts with directly, and its purpose is to provide a visually appealing and intuitive interface that allows users to easily navigate and perform tasks. A well-designed frontend is crucial for the success of an application, as it can greatly influence user satisfaction and adoption rates. Frontend development involves the use of the three fundamental web technologies HTML, CSS, and JavaScript to create the visual components of an application, including buttons, menus, forms, and other graphical elements. It is important to consider factors such as accessibility, usability, and responsiveness when designing a frontend, to ensure that the application can be used by a diverse range of users on a variety of devices and platforms.

#### 4.1.2 Backend

The backend of a web application is responsible for processing data and executing the logic of an application [Ahmed 2024a]. It provides the underlying functionality that supports the frontend, enabling it to interact with databases, servers, and other systems. Backend development involves the use of programming languages such as Java, Python, Ruby, and PHP to create the logic and functionality that drives an application. The backend also handles security and user authentication, ensuring that user



**Figure 4.1:** The client-server model for a web application. [Drawn by Ožbej Golob using Adobe Illustrator.]

data is protected from unauthorized access. Scalability and performance are key considerations when designing a backend, as it must be able to handle large amounts of data and support a large number of users. A well-designed backend is crucial for the success of an application, as it directly affects the reliability, performance, and security of the application.

## 4.2 Frontend Development Frameworks

A frontend development framework is a collection of pre-written code and tools that developers can use to build the user interface and functionality of a software application. These frameworks are typically built using JavaScript, and provide a set of pre-designed components and functions that developers can use to create an application's frontend quickly and efficiently. Frontend development frameworks can include a wide range of tools, such as user interface components like buttons, forms, and menus, as well as more complex features like data visualization, animations, and dynamic content loading. This section reviews several popular frontend development frameworks.

### 4.2.1 Angular

Angular is an open-source framework for building web applications using HTML, CSS, and JavaScript or TypeScript [Google 2024; Murray et al. 2018]. It was developed and is maintained by Google.

Some of the advantages of using Angular include its comprehensive set of tools and features, which can help developers build complex web applications quickly and efficiently. Angular also provides a strong focus on code organization and architecture, making it easier to maintain and update applications over time. Additionally, Angular offers a large community of developers and resources, making it easier to find support and learn new skills.

However, some of the disadvantages of using Angular include a steep learning curve, as it requires knowledge of multiple programming languages and concepts. Additionally, Angular can be more verbose than other frameworks, which can lead to larger codebases and slower performance. Finally, Angular can be more difficult to integrate with other technologies and frameworks, since it has a unique architecture and approach to web development.



### 4.2.2 React

React is an open-source framework for building web and mobile applications using JSX (a syntax extension for JavaScript), CSS, and JavaScript or TypeScript [Meta 2024; Accomazzo et al. 2017]. It was developed and is maintained by Meta (formerly Facebook).

Some of the advantages of using React include its simplicity and flexibility, which make it easy to learn and use, and its focus on reusability, which can save developers time and effort. React also offers a virtual Document Object Model (DOM), which can improve application performance by reducing the number of updates needed to the actual DOM. Additionally, React has a large and active community of developers, with many resources and tools available to help developers learn and use the framework effectively. React is highly customizable, allowing developers to use it in conjunction with other libraries and frameworks to build complex, scalable applications.

However, some of the disadvantages of using React include the lack of built-in functionality, which can require developers to use additional libraries or tools to add functionality to their applications. React can also have a steep learning curve. Additionally, React's reliance on JSX can be challenging for some developers to learn and use effectively.

### 4.2.3 Vue

Vue is an open-source framework for building web applications using HTML, CSS, and JavaScript or TypeScript [You 2024b; Filipova 2016]. It was developed by Evan You and is maintained by Vue's core team members.

Some of the advantages of using Vue include its simplicity, which makes it easy to learn and use for developers of all skill levels, and its flexibility, which allows developers to use it with other libraries and frameworks to build complex, scalable applications. Vue also offers a range of features, including reactive data binding, computed properties, and built-in directives, that can help developers build applications quickly and efficiently. Additionally, Vue has a growing and active community of developers, with many resources and tools available to help developers learn and use the framework effectively. Vue also has a relatively small learning curve compared to other popular frameworks, such as React and Angular.

However, some of the disadvantages of using Vue include its smaller community and ecosystem compared to other frameworks, which can make it more difficult to find support and resources. Vue also has fewer third-party plugins and tools available compared to other frameworks, which can limit its functionality in some cases.

### 4.2.4 Svelte

Svelte is an open-source framework for building web applications using HTML, CSS, and JavaScript [Harris 2024a; Holthausen 2022]. It was developed by Rich Harris and is maintained by Svelte's core team members. It takes a different approach to building web applications compared to other popular frameworks, by moving much of the work of generating code from the browser to the build process.

One of the primary benefits of Svelte is its small file size and fast performance. Svelte compiles the application code into highly optimized, plain JavaScript code that runs more efficiently in the browser, resulting in faster load times and improved performance. Another advantage of Svelte is its simplicity and ease of use. The framework has a small API surface area, and its syntax is intuitive and easy to learn. Svelte also has a rich ecosystem of plugins and tools that make it easy to integrate with other technologies.

However, one of the disadvantages of Svelte is its relatively new status, which means that it has a smaller community and fewer resources compared to more established frameworks like Angular, React, and Vue. This can make it more difficult to find support and resources for developers who are just starting

with Svelte. Another disadvantage of Svelte is its limited compatibility with older browsers. Since Svelte relies on newer web technologies like JavaScript modules and the CSS Grid Layout, it may not work well on older browsers that do not support these features.

SvelteKit is a framework for building web applications using Svelte [Harris 2024b]. It provides a comprehensive toolkit that includes everything you need to build a fully-featured application, such as server-side rendering (SSR), static site generation (SSG), and client-side rendering (CSR). SvelteKit handles routing, data fetching, and other essential aspects of web application development, making it easier to build scalable, high-performance applications. SvelteKit leverages the power of Svelte while adding the necessary infrastructure and features to support complex, production-ready web applications.

## 4.3 Web Graphics Rendering Technologies

Web graphics rendering technologies are used to create and display graphics, images, and multimedia content in the web browser. Modern web browsers support several ways to draw graphics, which are built into the web browser. There are four built-in web graphics rendering technologies [Andrews 2024b]:

- *Canvas2D*: Raster-based graphics are created with the HTML `<canvas>` element and the 2d rendering context [W3C 2015]. A JavaScript API is used to draw to the canvas.
- *SVG-DOM*: SVG nodes are injected dynamically into the DOM using JavaScript [Whitney 2013].
- *WebGL*: A 3d graphics API derived from OpenGL is used to construct and manipulate a scene graph [Khronos Group 2024a]. The HTML `<canvas>` element is used with the `webgl` or `webgl2` rendering context. A JavaScript API is used to draw to the canvas. WebGL makes use of hardware acceleration through a GPU, where available.
- *WebGPU*: The successor to WebGL, which is faster and non-blocking [W3C 2024d]. A JavaScript API is used to draw to an HTML `<canvas>` element with `webgpu` rendering context.

In addition, the Offscreen Canvas API [MDN 2024a] helps improve performance for the those built-in web graphics rendering technologies which use a `<canvas>` element (Canvas2D, WebGL, and WebGPU) by offloading the rendering process to a separate thread.

### 4.3.1 Canvas2D

Raster-based 2d graphics can be created from JavaScript using the Canvas API and the HTML `<canvas>` element with a 2d rendering context [W3C 2015]. This will be referred to as Canvas2D. Canvas2D is used to create dynamic and interactive graphics on the web. Canvas2D graphics are created using a raster-based approach, where pixels are directly manipulated to create the graphics. Canvas2D was introduced in 2004 by Apple [Matthew 2021] and was the only option for manipulating web graphics until SVG-DOM was introduced.

Canvas2D offers improved performance compared to SVG-DOM and can handle larger datasets and more complex animations. Canvas2D graphics are highly dynamic and interactive, which means that they can be manipulated and animated in real time using JavaScript. However, they are not naturally scalable or responsive and have to be redrawn if the dimensions change. Canvas2D is supported by all modern browsers, making it a reliable and cross-browser-compatible graphics technology.

### 4.3.2 SVG-DOM

Scalable Vector Graphics (SVG) is a standard markup language for 2d vector graphics [W3C 2001]. Based on XML, it contains elements such as lines, rectangles, circles, polygons, and text, out of which more complex graphics can be constructed. SVG graphics are scalable, which means that they can be

resized without losing quality. SVG graphics are also editable, which means that they can be easily modified or updated. Modern browsers support SVG 1.1 and some elements of SVG 2.0. This means that SVG elements can be intermingled with HTML elements in a web document, to provide embedded vector graphics. The SVG elements can be styled with CSS.

JavaScript can be used to dynamically insert SVG nodes into the browser's Document Object Model (DOM) [W3C 2011] and later manipulate or remove them [Whitney 2013]. Thus, data-driven vector graphics can be created on the fly with JavaScript. This technique will be referred to as SVG-DOM. Performance issues can arise when rendering large datasets with SVG-DOM, since each data point or shape requires its own SVG element to be inserted into the DOM.

### 4.3.3 WebGL

WebGL is a 3d web graphics rendering technology used to create high-performance graphics on the web [Khronos Group 2024a]. As of now, there are two major versions of WebGL. WebGL 1.0 was released in 2011 and enabled developers to leverage the hardware of the Graphics Processing Unit (GPU) of a device to render graphics in real-time, resulting in visually stunning and highly performant graphics [Khronos Group 2024b]. WebGL 2.0 was released in 2017 and was built on top of WebGL 1.0 [Khronos Group 2024c]. It introduced several new features and improvements, such as multiple render targets, instanced drawing, uniform buffer objects, and enhanced texturing capabilities. These enhancements allow for more advanced graphics and performance optimizations in web applications.

WebGL graphics are drawn by constructing a scene graph through the WebGL API. WebGL also enables users to manipulate and interact with the graphics in real time. Additionally, WebGL is supported by all modern browsers, making it a reliable and cross-browser-compatible graphics technology. However, WebGL has a steep learning curve that requires a high level of expertise and knowledge of both JavaScript and computer graphics. There exist several WebGL libraries that simplify the development of web applications with WebGL, some of which are described in Section 4.4.

### 4.3.4 WebGPU

WebGPU is the successor to WebGL [W3C 2024d]. It is designed to be closer to the architecture of modern GPUs and is similar in style to low-level graphics APIs like Vulkan, DirectX 12, and Metal. WebGPU aims to improve performance and efficiency by allowing developers to write code that can take advantage of the latest hardware features, while still maintaining compatibility with a wide range of devices and browsers. It provides a unified and standardized interface for accessing GPU resources, including rendering, computing, and data transfer operations.

WebGPU is still in the early stages of development and is not yet supported by all major web browsers (see [Deveria 2024]), but it has the potential to revolutionize web-based graphics and computing in the future.

### 4.3.5 Offscreen Canvas

The Offscreen Canvas API is a relatively new feature in web development that allows for the creation of a canvas rendering context that can be rendered in a separate thread from the main JavaScript thread [MDN 2024a]. In traditional web development, all canvas rendering occurs on the main thread, which can cause performance issues if the canvas is rendering complex graphics or animations. The Offscreen Canvas API alleviates this issue by offloading the rendering process to a separate thread, thus freeing up the main thread for other tasks. Using the Offscreen Canvas API, developers can create and manipulate a canvas rendering context in a separate thread while still being able to transfer the rendered output to the main thread for display on the web page. This can result in faster rendering times, smoother animations, and a more responsive user experience.

The Offscreen Canvas API is supported by all modern web browsers. Canvas2D, WebGL, and WebGPU can all make use of and benefit from Offscreen Canvas, since they all use a HTML `<canvas>` element to draw. However, SVG-DOM cannot benefit from Offscreen Canvas, since it is rendered via the DOM rather than a canvas.

## 4.4 Web Graphics Rendering Libraries

Web graphics rendering libraries have become increasingly popular in recent years, as they help developers create visually appealing and interactive graphics on web pages. These libraries provide pre-built functions and tools for creating and manipulating 2d and 3d graphics, animations, and visualizations, allowing developers to construct a wide range of applications.

### 4.4.1 SVG.js

SVG.js is a lightweight 2d drawing library that uses SVG-DOM for rendering [Fierens 2024]. It provides a simple API to create, animate, and manipulate various SVG elements.

### 4.4.2 Konva.js

Konva.js is a 2d drawing and animation library which builds upon Canvas2D [Lavrenov 2024]. In effect, it extends the utility of Canvas2D with many extra features. With Konva.js, developers can easily create and manipulate complex scenes composed of various shapes, images, text, and animations.

### 4.4.3 Two.js

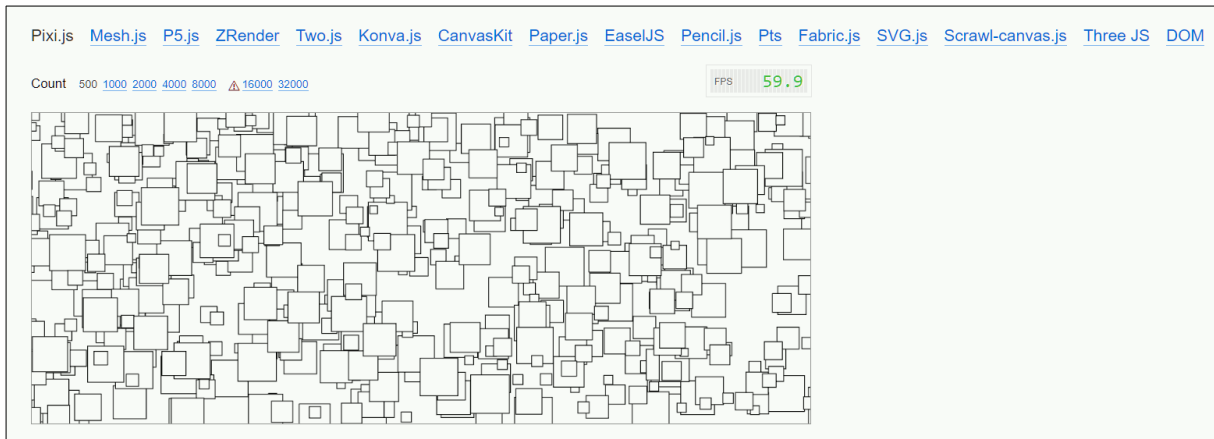
Two.js is a 2d drawing library for the web that provides an intuitive and easy-to-use API for creating complex vector graphics, animations, and interactive visualizations [Brandel 2024]. It was originally spun off from the Three.js project to focus solely on 2d graphics. Two.js supports the WebGL, Canvas2D, and SVG-DOM web rendering technologies. It has a wide range of shapes and primitives and allows developers to apply various styles and effects to them, such as gradients, shadows, and opacity. It also supports animation and interactivity, with features like tweening and easing functions, mouse and touch events, and drag-and-drop functionality. Two.js is highly performant when WebGL is used for rendering and has a modular architecture that allows developers to use only the parts of the library that they need. It has a large community of contributors and users.

### 4.4.4 Pixi.js

Pixi.js is a powerful 2d drawing library for the web, which provides a fast and efficient way to create high-performance interactive applications and games [Groves 2024]. Pixi.js supports the WebGL and Canvas2D web rendering technologies, as well as Offscreen Canvas. Pixi.js provides an easy-to-use API that allows developers to create complex graphics and animations with ease. Pixi.js supports a wide range of features, including sprite sheets, filters, masking, and particle systems. It also has built-in support for animations and interactivity, including keyboard and mouse events, touch events, and physics simulations. Pixi.js is highly optimized for performance, which makes it suitable for creating complex graphics and animations that run smoothly even on mobile devices. It has a large and active community of contributors and users who continue to improve and extend its functionality.

#### 4.4.4.1 Three.js

Three.js is a 3d drawing library for generating graphics and animations on the web [Cabello 2024a]. Three.js supports all four web rendering technologies, WebGL, WebGPU, Canvas2D, and SVG-DOM, as



**Figure 4.2:** The Slay Lines benchmark web application displaying 500 rectangles, in this case with Pixi.js. [Screenshot taken by Ožbej Golob using The Slay Lines application [Slay Lines 2024].]

well as Offscreen Canvas. Three.js provides an easy-to-use API for building complex 3d scenes, which can include a wide variety of objects, materials, lights, and effects. Three.js also includes support for advanced materials, physics simulations, and VR. With a large and active community of contributors, Three.js is constantly evolving and improving, making it one of the most popular and widely used 3d graphics libraries for the web.

#### 4.4.5 Babylon.js

Babylon.js is a 3d drawing library for generating graphics and animations on the web, similar to Three.js [Catuhe 2024]. Babylon.js supports the WebGL and WebGPU web rendering technologies, as well as Offscreen Canvas. Babylon.js provides an easy-to-use API for building complex 3d scenes, which can include a wide variety of objects, materials, lights, and effects. With a large and active community of contributors, Babylon.js is also evolving and improving, and is widely used.

#### 4.4.6 D3.js

D3.js is a lower-level 2d drawing library for manipulating and visualizing data based on SVG-DOM rendering [Bostock 2024; Bostock et al. 2011]. D3.js allows developers to create dynamic, interactive, and customizable data visualizations by composing a series of marks. It includes a powerful set of examples and recipes for visualizations, such as bar charts, scatterplots, line charts, and more. In addition, D3.js provides a wide range of utilities for working with data, including data binding, selection, filtering, transformation, reading CSV files, and creating scales. These tools can be combined and customized to create complex interactive visualizations that are tailored to specific data sets and use cases.

One of the key strengths of D3.js is its flexibility. It does not impose a specific visualization style or framework, but rather provides a set of building blocks that can be combined in creative ways. This makes it suitable for a wide range of applications, from simple charts and graphs to complex data-driven applications.

#### 4.4.7 Performance Comparison

The company Slay Lines created a benchmark web application that compares several popular web graphics rendering libraries [Slay Lines 2024]. The application consists of up to 32,000 different rectangles moving on a canvas at various speeds. Figure 4.2 shows the Slay Lines application displaying 500 rectangles

	500 R	1,000 R	2,000 R	4,000 R	8,000 R	16,000 R	32,000 R
SVG.js	52.2	26.2	13.4	8.3	4.4	1.5	0.7
Konva.js	64.5	57.0	31.6	16.0	9.2	6.9	3.5
Two.js	60.2	70.0	47.2	26.2	13.6	5.1	-
Pixi.js	60.1	60.2	60.3	60.5	56.4	31.8	17.0
Three.js	27.2	14.7	7.6	3.4	1.8	0.6	-

**Table 4.1:** Comparative performance of web graphics rendering libraries on a laptop (Setup A). Columns indicate the number of animated rectangles in the benchmark. Cells show the average FPS of a specific library. A dash (-) indicates that Chrome crashed.

	500 R	1,000 R	2,000 R	4,000 R	8,000 R	16,000 R	32,000 R
SVG.js	76.0	75.2	76.8	41.9	19.8	9.8	4.8
Konva.js	75.5	75.1	75.0	63.2	50.3	14.7	5.1
Two.js	75.0	75.1	75.8	73.8	33.9	17.7	4.2
Pixi.js	75.0	75.4	75.3	75.4	75.1	65.5	29.7
Three.js	75.2	75.2	75.1	36.4	15.4	7.4	3.6

**Table 4.2:** Comparative performance of web graphics rendering libraries on a desktop (Setup B). Columns indicate the number of animated rectangles in the benchmark. Cells show the average FPS of a specific library. A slash (/) indicates that Chrome crashed.

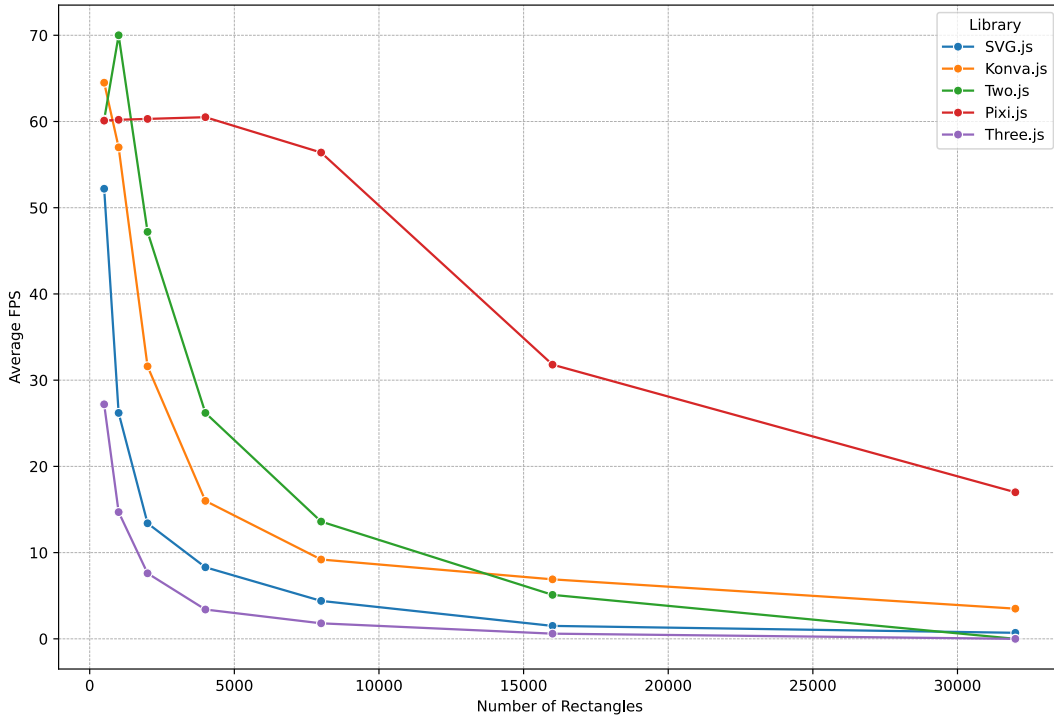
with Pixi.js. The Slay Lines web application was used to compare the following web graphics rendering libraries: SVG.js (SVG-DOM), Konva.js (Canvas2D), Two.js (WebGL), Pixi.js (WebGL), and Three.js (WebGL).

The aforementioned libraries were benchmarked by their frame rate in frames per second (FPS), as an average over ten seconds. The frame rate was measured using a custom Python script, shown in Listing 4.1. The script uses Selenium WebDriver to programmatically open a Chrome web browser and navigate to the Slay Lines application [Huggins 2024]. Then, Selenium executes a JavaScript function to measure the average FPS over ten seconds and reports it into the console logs.

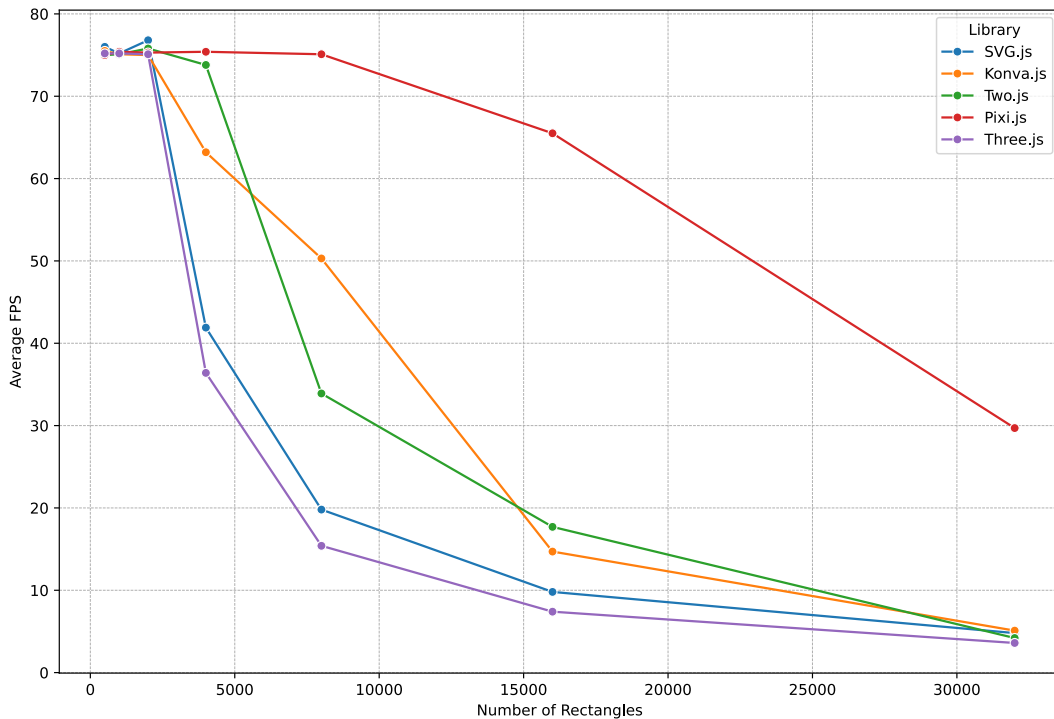
Two experimental setups were used: A) a laptop running Windows 11 Pro Version 23H2 with an Nvidia GeForce MX150 GPU, and B) a desktop running Windows 11 Pro Version 23H2 with an Nvidia GeForce RTX3060 GPU. The frame rate was measured using Chrome version 126.0.6478.126 and with Python 3.12.4. Figures 4.3 and 4.4 show the resulting performance measurements in terms of average FPS with respect to the number of animated rectangles. Tables 4.1 and 4.2 report the exact numbers. As can be seen, Pixi.js consistently offered the highest performance in both setups.

```
1 from selenium import webdriver
2 import time
3
4 # Create a new Chrome web driver
5 options = webdriver.ChromeOptions()
6 browser = webdriver.Chrome(options=options)
7
8 # Navigate to the webpage you want to measure the FPS for
9 browser.get("https://benchmarks.slaylines.io/")
10
11 # Wait for the page to load
12 time.sleep(10)
13
14 # Execute JavaScript to start measuring FPS
15 browser.execute_script("""
16     window.performance.mark('start');
17     let startTime = performance.now();
18     let lastTime = startTime;
19     let fps = 0;
20     let fpsArray = [];
21
22     function update() {
23         let currentTime = performance.now();
24         let timeElapsed = currentTime - lastTime;
25         lastTime = currentTime;
26         fps = 1000 / timeElapsed;
27         fpsArray.push(fps);
28
29         // End measurement at 10 seconds
30         if (currentTime - startTime > 10000) {
31             window.performance.mark('end');
32             window.performance.measure('measure', 'start', 'end');
33             let sum = fpsArray.reduce((a, b) => a + b, 0);
34             let average = sum / fpsArray.length;
35             window.averageFps = average; // Store the average FPS globally
36         }
37         else window.requestAnimationFrame(update);
38     }
39     window.requestAnimationFrame(update);
40 """)
41
42 # Wait for the measurement to complete
43 time.sleep(12)
44
45 # Retrieve the FPS data
46 average_fps = browser.execute_script("return window.averageFps;")
47 print("Average FPS:", average_fps)
48
49 # Close the browser
50 browser.quit()
```

**Listing 4.1:** Python code using Selenium WebDriver to measure average FPS.



**Figure 4.3:** Comparative performance of web graphics rendering libraries, measured using the Slay Lines benchmark web application on a laptop (Setup A). [Drawn by Ožbej Golob using Python.]



**Figure 4.4:** Comparative performance of web graphics rendering libraries, measured using the Slay Lines benchmark web application on a desktop (Setup B). [Drawn by Ožbej Golob using Python.]



## 4.5 Desktop Development Libraries

Desktop development libraries are software libraries and frameworks that enable developers to create web applications which also run on common desktop operating systems like Windows, macOS, and Linux. These libraries provide developers with tools and APIs to build graphical user interfaces (GUIs), handle user input and output, manage data, interact with system resources, and perform other tasks required for building desktop applications. This section describes two popular desktop development libraries for web applications: Electron and Tauri.

### 4.5.1 Electron.js

Electron.js (also known as Electron) is a longstanding desktop development library [OpenJS 2024a]. Electron works by combining two main components: Chromium and Node.js. Chromium is an open-source web browser project that powers Google Chrome and other popular browsers. It provides Electron with the ability to render web content and handle user interactions. The Chromium-based UI looks the same on Windows, Linux, and macOS. Node.js is a JavaScript runtime for running JavaScript on the desktop and accessing native APIs. In essence, Electron allows web applications to be packaged as desktop applications by bundling Chromium, Node.js, and the web application itself into a single executable package. This comes at the cost of rather large executable packages, often 100 MB or more.

In addition, Electron provides developers with a set of APIs for building desktop applications, including APIs for handling file system operations, creating menus and dialogs, and managing window events. However, using them means that the application can no longer be used as a web application.

### 4.5.2 Tauri

Tauri is a relatively new desktop development library [Tauri 2024a]. Tauri works by combining Rust, JavaScript, and the desktop platform's pre-installed local Webview (web browser) component. Rust is a programming language recognized for its strong memory safety and high performance. Rather than bundling a complete web browser, Tauri uses the local system's Webview: Edge Webview2 (Chromium) on Windows, WebKitGTK on Linux, and WebKit on macOS, making Tauri apps much lighter than Electron apps. Tauri apps also typically have better performance, launch time, and memory consumption than Electron apps.



## Chapter 5

# The Multidimensional Visual Analyser (MVA)

MVA is a web application written with the SvelteKit framework. It is rendered entirely on the client-side, meaning it has no backend. The user must first upload a dataset, and MVA then displays the data with the help of several of the multidimensional visual analysis approaches described in Chapter 2. The MVA user interface has two main parts: the Navigation Bar ① and the Display Area. The Display Area consists of six synchronized panels, which the user can hide/show, resize, or move: Scatterplot Matrix ②, Scatterplot ③, Similarity Map ④, Partitions ⑤, Table ⑥, and Parallel Coordinates ⑦. The MVA user interface is shown in Figure 5.1.

Four of the panels contain visualizations. A single scatterplot can be selected in the Scatterplot Matrix panel, which is then shown in the Scatterplot panel. The Similarity Map panel implements PCA and UMAP similarity mapping techniques. The Parallel Coordinates panel can be used to filter records, making them inactive, and the changes are reflected in all panels. All four of the visualization panels can be exported individually as SVG files (.svg).



**Figure 5.1:** The Navigation Bar ① and six synchronized panels of the Display Area: Scatterplot Matrix ②, Scatterplot ③, Similarity Map ④, Partitions ⑤, Table ⑥, and Parallel Coordinates ⑦.

The Table panel displays the dataset in tabular form, where rows are records and columns are dimensions. The user can sort the table by any dimension, ascending or descending. The Partitions panel enables the user to manually group and label records into partitions. The user can select a partition's shape and color, and can hide partitions. Partitions can be exported, and the user can re-import the dataset with existing partitions, or use the exported partitions as input to a machine learning model.

MVA also implements brushing and linking. When a record is hovered over in one panel, the changes are reflected in all panels. The user can select (brush) multiple records by clicking them or by selecting them with specialized selection tools.

## 5.1 Build System

MVA is written with SvelteKit and TypeScript. The SvelteKit build system uses npm [npm 2024] and Vite [You 2024a], which creates an optimized production build of the application. The build process creates a static site that is ready to be deployed. The task runner Gulp is used to automate common tasks during the build process [Bublitz and Schoffstall 2024].

Despite MVA primarily being a web application, it can also be built into a desktop application for Windows. This is done with Tauri, which is a Rust-based framework for building native desktop applications from a web application, described in Subsection 4.5.2.

## 5.2 Dependencies

MVA depends on the following npm packages, defined in the `package.json` file:

- `d3-array`: For array manipulation, sorting, and obtaining histogram data [Bostock 2023a].
- `d3-axis`: For drawing the scatterplot, similarity map, and parallel coordinates axes [Bostock 2023b].
- `d3-drag`: For handling dragging of parallel coordinate axes and scatterplot matrix overview map [Bostock 2023c].
- `d3-dsv`: For parsing imported datasets [Bostock 2023d].
- `d3-scale`: For calculating x and y scales for all panels [Bostock 2023e].
- `d3-selection`: For selecting SVG elements by id and class [Bostock 2023f].
- `d3-shape`: For drawing lines in parallel coordinates SVG export and scatterplot matrix lines [Bostock 2023g].
- `flowbite-svelte`: A component library that uses Tailwind CSS [Themesberg 2024a]. MVA uses several Flowbite components: buttons, checkboxes, toast, navigation, drop-down, form elements, icons, etc.
- `ml-pca`: For calculating the PCA similarity map [Zakodium 2023].
- `umap-js`: For calculating the UMAP similarity map [People+AI Research (PAIR) Initiative 2024].
- `svelte-awesome-color-picker`: For partition color selection [Dupont 2024]. It is a highly customizable color picker component library that supports HEX, HSV, and RGB color profiles.
- `svelte-splitpanes`: For resizing panels [Refalo 2024]. It is a responsive component that enables resizable panels supporting several advanced features.
- `tailwind-merge`: A utility for Tailwind CSS, required by the Flowbite component library [Castillo 2024].

- `three`: For drawing all user-visible data records [Cabello 2024a].
- `xml-formatter`: For formatting the exported SVG file [Bottin 2024]. It parses the SVG code and indents it into a user-readable form.

## 5.3 Components

MVA is split into multiple components to improve organization, maintainability, readability, and scalability. Each component is presented as a SvelteKit file with a name like `<Component>.svelte`. Listing 5.1 shows the folder structure of the most important MVA application components. Some components also have utility files (`util.ts`, `draggingUtil.ts`, or `drawingUtil.ts`) which contain helper functions, and types files (`types.ts`) which contain TypeScript types and interfaces, but these are not shown in Listing 5.1 for brevity.

## 5.4 Icons

MVA uses some icons from the `flowbite-svelte-icons` icon library [Themesberg 2024b], which is a collection of more than 430 free and open-source SVG icons. MVA also requires some custom icons. Custom icons are defined as SVG graphics in the `src/static/icons/` folder. A gulp task then collects the icons as TypeScript SVG string constants into the file `src/util/icon-definitions.ts`, from where they can easily be imported and used.

## 5.5 Example Datasets

The folder `example-datasets/` contains several example datasets, which the user can import into MVA. They are further described in Section A.4.

```

components/
├── navbar/
│   ├── Navbar.svelte
│   └── dataset/
│       ├── DatasetPreview.svelte
│       ├── ExampleDatasets.svelte
│       ├── ExportDatasetModal.svelte
│       └── ImportDatasetModal.svelte
├── panels
│   ├── Layout-[1-6].svelte
│   └── Panel.svelte
├── parcoord/
│   ├── ParcoordComponent.svelte
│   ├── ParcoordVisibleDimensions.svelte
│   ├── SelectionShapePicker.svelte
│   └── axes/
│       └── Axes.svelte
│   ├── context-menu/
│   │   └── ContextMenuAxes.svelte
│   ├── histograms/
│   │   ├── HistogramSettings.svelte
│   │   ├── HistogramSettingsModal.svelte
│   │   └── Histograms.svelte
│   └── lines/
│       └── Lines.svelte
├── partitions/
│   ├── AddPartitionModal.svelte
│   ├── ColorPickerCustom.svelte
│   ├── ColorPickerModal.svelte
│   ├── ContextMenu.svelte
│   ├── DeletePartitionModal.svelte
│   ├── PartitionElement.svelte
│   └── PartitionsComponent.svelte
├── scatterplot/
│   ├── DimensionPickers.svelte
│   ├── ScatterplotComponent.svelte
│   ├── SelectionShapePicker.svelte
│   └── axes/
│       └── Axes.svelte
│   └── points/
│       └── Points.svelte
├── simmap/
│   ├── MethodPicker.svelte
│   └── SimmapComponent.svelte
├── splom/
│   ├── OverviewSettings.svelte
│   ├── SPLOMComponent.svelte
│   └── axes/
│       ├── Axes.svelte
│       └── AxesOverview.svelte
│   └── points/
│       └── Points.svelte
├── svg-exporter/
│   └── SvgExportModal.svelte
├── table/
│   ├── ContextMenu.svelte
│   ├── TableComponent.svelte
│   └── TableVisibleDimensions.svelte
└── tooltip/
    └── Tooltip.svelte

```

**Listing 5.1:** The folder structure of the most important MVA application components.

## Chapter 6

# Selected Details of the Implementation

The previous chapter provided a high-level overview of the MVA application. This chapter describes implemented approaches in further detail, with emphasis on specific decisions taken during the development of the application.

### 6.1 Chosen Web Graphics Rendering Technology

Section 4.4.7 compared different web graphics rendering technologies. Since Pixi.js produced the highest average FPS, it was initially implemented as the selected technology. However, it was observed that Pixi.js, despite its high performance, exhibited significant issues with line smoothness. This suggested potential limitations in the library's antialiasing algorithm or its implementation in handling certain graphic elements. These findings are particularly relevant for applications like MVA, where the quality of such graphic elements is of significant importance. This drove the need to consider alternative rendering solutions.

Despite having a lower average FPS, the Three.js library was selected as a workable alternative, because of its smooth lines, ease of implementation, extensive documentation, and online examples. Figures 6.1 and 6.2 show lines rendered with Pixi.js and Three.js respectively. Although hard to see in the screenshots, lines produced with Three.js are visually much smoother.

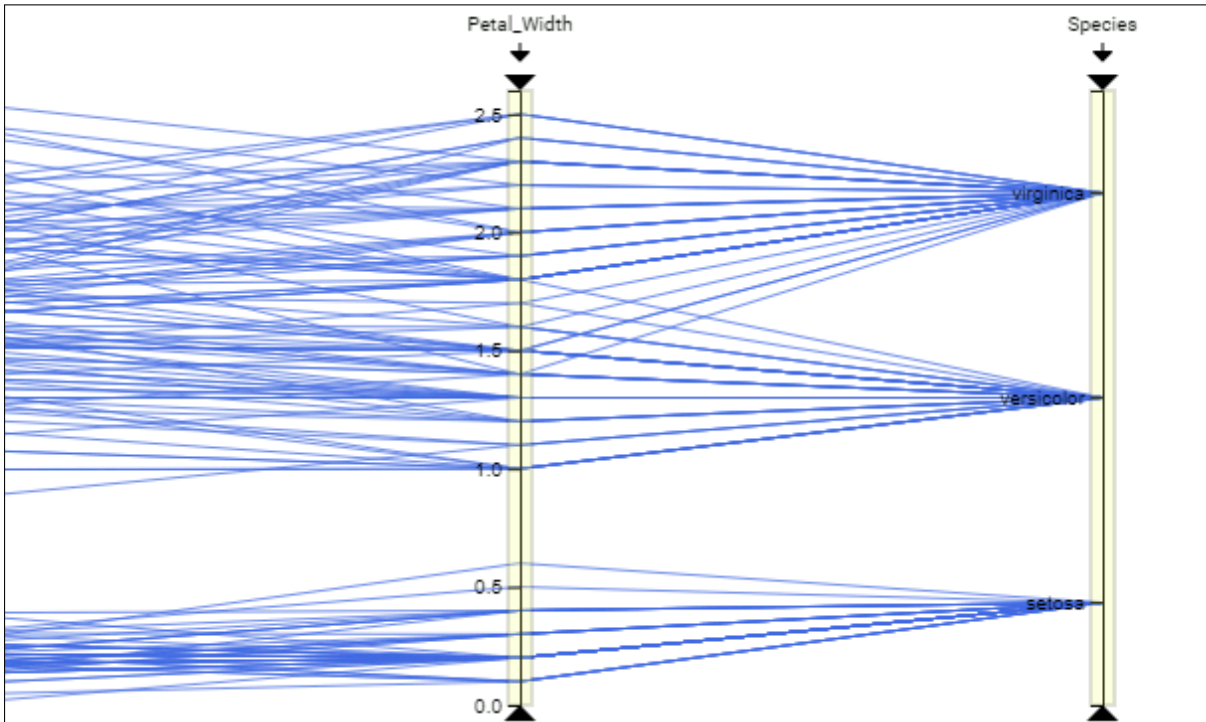


Figure 6.1: Lines rendered with Pixi.js.

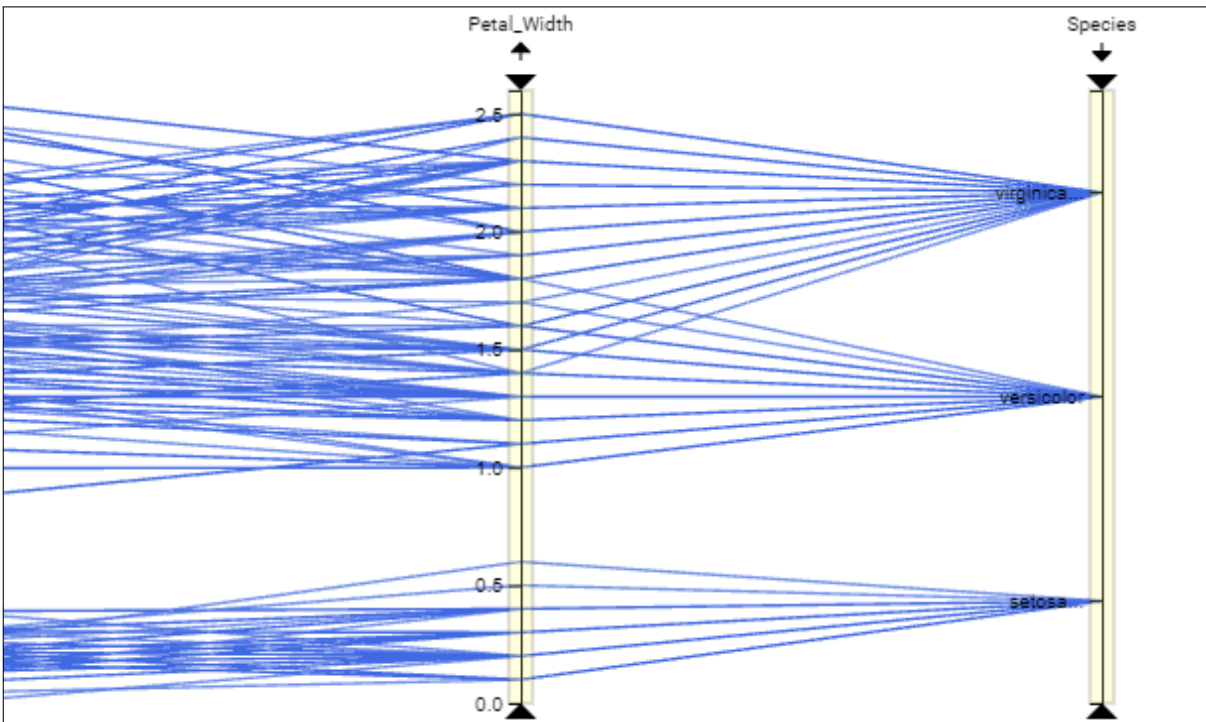


Figure 6.2: Lines rendered with Three.js.



```
1 .canvas-records {
2   background-color: rgba(255, 255, 255, 0);
3   position: absolute;
4   top: 0;
5   right: 0;
6   bottom: 0;
7   left: 0;
8   z-index: 2;
9 }
10
11 .canvas-axes {
12   background-color: rgba(255, 255, 255, 0);
13   position: absolute;
14   top: 0;
15   right: 0;
16   bottom: 0;
17   left: 0;
18   z-index: 3;
19 }
```

**Listing 6.1:** The CSS `z-index` property is used to overlay two canvases positioned one above the other. Records are drawn on the background canvas with a `z-index` of 2. Axes are drawn on the foreground canvas with a `z-index` of 3. This creates the illusion that both are drawn on the same canvas.

## 6.2 Overlaying Canvases for Visualizations

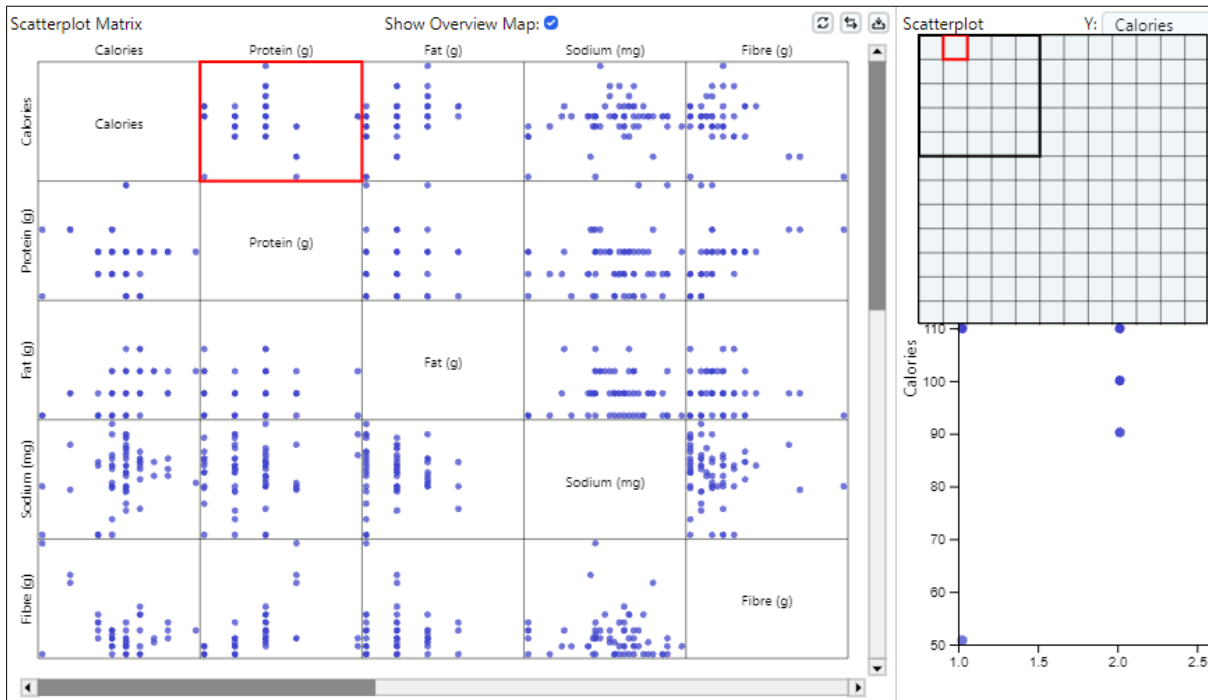
Each of the four MVA visualization panels comprises two main components: axes and records (drawn either as lines or points). There are three main options for drawing axes: SVG-DOM, Canvas2D, and WebGL. Despite WebGL being much faster, it has very limited interactivity (such as dragging, hovering, and swapping). Since axes do not represent the main rendering workload, they can be rendered as SVG elements with the help of the D3 library. This ensures better interactivity in the form of axis dragging, axis swapping, filter dragging, custom cursors, etc.

The main workload of the application is taken up rendering data records. Records are drawn using the Three.js library with WebGL. It is not possible to use the same canvas for both D3 and WebGL. Thus, the visualization panels render records with WebGL on one canvas and axes and other controls as SVG elements on a second overlaid canvas. This is achieved using the CSS `z-index` property, as shown in Listing 6.1. The user sees both canvases as one, since they are overlaid.

## 6.3 Scatterplot Matrix Rendering

Of the visualizations in MVA, the scatterplot matrix (SPLOM) presents the main rendering challenge, since for  $n$  dimensions a full SPLOM would comprise  $n^2$  individual scatterplots. Each record would need to be rendered in every scatterplot. For example, a relatively small dataset with 20 dimensions and 100 records translates to 400 individual scatterplots in a  $20 \times 20$  SPLOM, and 40,000 records being drawn each time.

To reduce the number of records having to be drawn, MVA's Scatterplot Matrix panel only shows a  $5 \times 5$  matrix of scatterplots at any one time, and the user can pan around the larger (virtual) full SPLOM. For a dataset with 20 dimensions and 100 records, the number of records having to be drawn at any one



**Figure 6.3:** The Scatterplot Matrix panel.

time is reduced from 40,000 to 2,500. Two tools are provided for the user to navigate around the full SPLOM: scrollbars to the right and bottom, and an Overview Map, as shown in Figure 6.3. Once opened, the Overview Map shows a 5×5 window into the full SPLOM, which the user can drag around with the mouse. The scrollbars can also be used to pan around. To further increase performance, web workers are used to perform calculations off the main thread, as described in Section 6.4.

## 6.4 Using Web Workers

Web workers allow JavaScript to run in the background, on a separate thread from the main execution thread of a web application, freeing up the main thread to concentrate on responding quickly to user interactions. Two types of web worker are used in MVA: *drawing workers* and *calculating workers*.

Drawing workers are used by MVA's four visualization panels (Scatterplot Matrix, Scatterplot, Similarity Map, and Parallel Coordinates) to offload the drawing of records to a separate thread in an Offscreen Canvas. The Three.js library supports OffscreenCanvas. Each of these panels creates a drawing worker and transfers a <canvas> element to it. The main thread then communicates with the worker by posting and receiving messages. Listing 6.2 shows an example of how canvas control is transferred offscreen and attached to a web worker and how the main thread and web worker communicate.

Calculating workers are used to increase the performance and rendering speed of the Scatterplot Matrix panel. The calculation of the x and y coordinates for the records of one individual scatterplot for the SPLOM is considered to be an atomic calculating task, and these tasks are distributed amongst the remaining available web workers. Listing 6.3 shows some example code to distribute tasks to available calculating workers.

```
1 <script lang="ts">
2   let canvasEl: HTMLCanvasElement;
3   let offscreenCanvasEl: OffscreenCanvas;
4
5   onMount(() => {
6     offscreenCanvasEl = canvasEl.transferControlToOffscreen();
7     worker = new DrawingWorker();
8
9     // Send initialization message to worker
10    worker.postMessage(
11      {
12        function: 'init',
13        canvas: offscreenCanvasEl,
14        width,
15        height,
16      },
17      [offscreenCanvasEl]
18    );
19
20    // Listen for received messages from worker
21    worker.onmessage = (message) => {
22      const data = message.data;
23      switch (data.function) {
24        case 'someFunction':
25          // Handle some function
26          break;
27        default:
28          break;
29      }
30    };
31  });
32 </script/>
33
34 <canvas bind:this={canvasEl} />
```

**Listing 6.2:** Svelte code for transferring a canvas to an Offscreen Canvas and attaching it to a web worker. The main thread sends an initialization message to the worker. A callback function listens for received messages.

```
1 function drawPoints() {
2   points = [];
3
4   availableWorkers = navigator.hardwareConcurrency;
5   if (availableWorkers === 0) calculatePointData();
6   else calculateDistributePointData();
7 }
8
9 function calculatePointData() {
10  // Code to calculate records X and Y scale coordinates
11
12  drawingWorker.postMessage({
13    function: 'drawPoints',
14    points
15  });
16 }
17
18 function calculateDistributePointData() {
19   completedWorkers = 0;
20   calculatingWorkers = [];
21
22   for (let i = 0; i < availableWorkers; i++) {
23     const worker = new CalculatingWorker();
24     calculatingWorkers.push(worker);
25     worker.onmessage = handleCalculatingWorkerResult;
26   }
27
28   const taskData: TaskType[] = [];
29   // Code to distribute data into tasks
30
31   // Distribute tasks among workers
32   const tasksPerWorker = Math.ceil(taskData.length / availableWorkers);
33   for (let i = 0; i < availableWorkers; i++) {
34     const tasks = taskData.slice(i * tasksPerWorker, (i + 1) * tasksPerWorker);
35     calculatingWorkers[i].postMessage({ tasks, spacing, margin });
36   }
37 }
38
39 function handleCalculatingWorkerResult(event: MessageEvent) {
40   points = points.concat(event.data.points);
41   completedWorkers++;
42   if (completedWorkers === availableWorkers) {
43     drawingWorker.postMessage({
44       function: 'drawPoints',
45       points
46     });
47
48     calculatingWorkers.forEach((worker) => {
49       worker.terminate();
50     });
51   }
52 }
```

**Listing 6.3:** Code for distributing calculating tasks between available calculating workers.

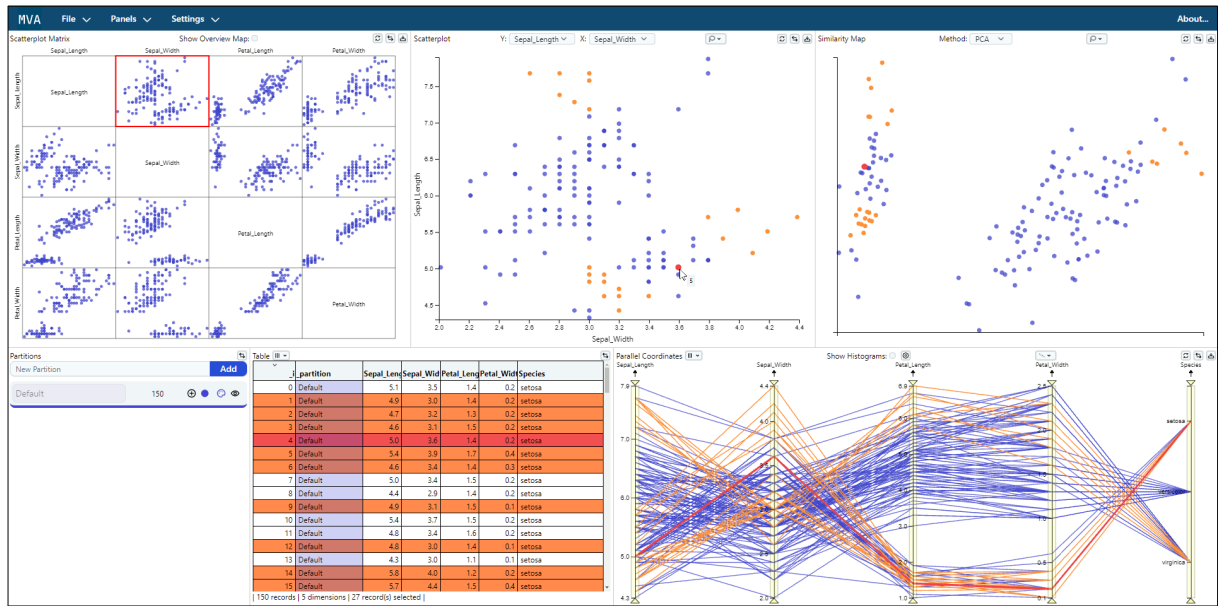




Figure 6.4: The MVA application showing brushed (orange) and hovered (red) records.

## 6.5 Hovering and Brushing

Some web graphics rendering libraries handle hovering natively using events such as mouseover and mouseout. Three.js does not support native hovering events. To detect whether a line is hovered, the Three.js Raycaster is used [Cabello 2024b]. Raycasting is a computational technique used to determine the path of rays through a 3D space to detect intersections with objects. It is often used in rendering, collision detection, and visibility determination in computer graphics. In MVA, the Raycaster is given the current mouse coordinates to determine intersections with any drawn (hovered) records. A Svelte store is used to save hovered record indices as a set of numbers. When the hovered set is updated, all panels receive a message to update the hovered items, which enables linking.

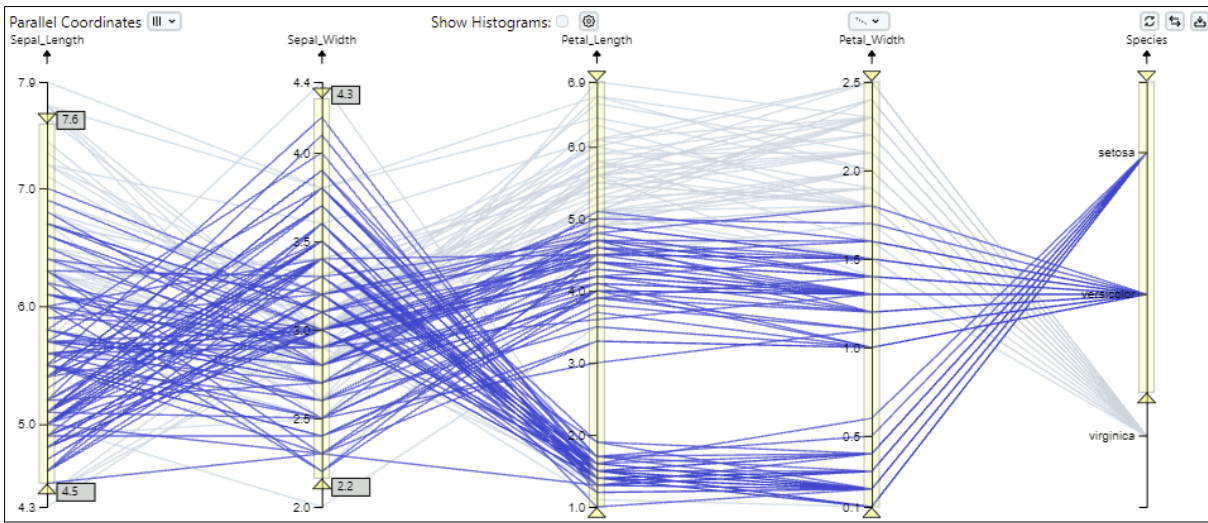
Brushing is achieved by detecting the pointerdown event. When a pointerdown event is triggered, one of three possible scenarios occurs, depending on the modifier:

- If the Shift key  is pressed during selection, all hovered records are added into the selection (brushed set).
- If the Control key  is pressed during selection, all hovered records are toggled in the brushed set (i.e. if the record is already in the brushed set, it is removed, otherwise it is added).
- Otherwise, a new brushed set is created, containing only the currently hovered records.

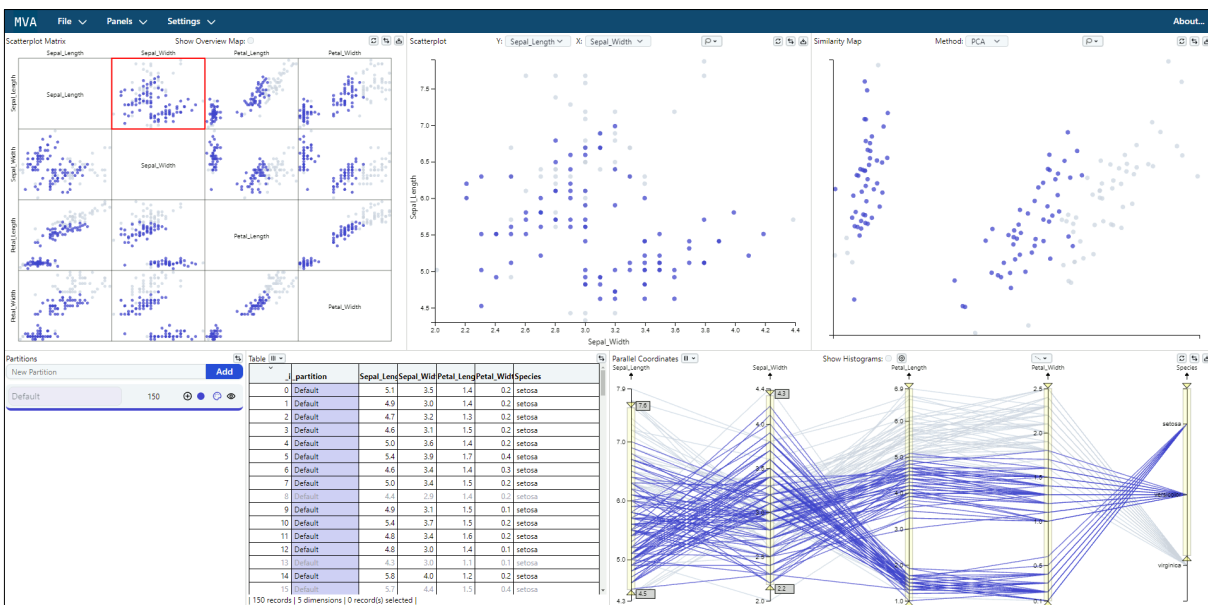
As with hovered items, a Svelte store is used to save the set of selected (brushed) record indices as a set of numbers. When the brushed set is updated, all panels receive a message to update the brushed items. Figure 6.4 shows a screenshot of MVA with brushed (orange) and hovered (red) records.

## 6.6 Filtering

Filtering is implemented in the Parallel Coordinates panel in the form of a double-edged range slider, as can be seen in Figure 6.5. Triangles at the top and bottom of the range slider can be dragged to adjust the range. In addition, the range slider itself can be moved up or down. Records which are out of the range are filtered out; they become inactive and are grayed out in every panel, as can be seen in Figure 6.6.



**Figure 6.5:** In the Parallel Coordinates panel, a double-edged range slider on each dimension (in pale yellow) can be adjusted to filter out records and make them inactive.



**Figure 6.6:** Filtered out records are inactive and are greyed out in every panel.

```
1 function isPointInPolygon(point: CoordinateType, polygon: CoordinateType[]) {
2   let inside = false;
3   const n = polygon.length;
4
5   for (let i = 0, j = n - 1; i < n; j = i++) {
6     const xi = polygon[i].x,
7         yi = polygon[i].y;
8     const xj = polygon[j].x,
9         yj = polygon[j].y;
10
11     const intersect = yi > point.y !== yj > point.y
12     && point.x < ((xj - xi) * (point.y - yi)) / (yj - yi) + xi;
13     if (intersect) inside = !inside;
14   }
15
16   return inside;
17 }
```

**Listing 6.4:** Code for calculating whether a point is contained within a polygon.

Filtering is computationally expensive, as calculations have to be performed for each adjustment of the range slider to decide which records have become inactive. Initially, filtering was implemented using a Raycaster to detect when the filter was dragged over the record. However, this approach caused performance issues and slow rendering of filtered records. For that reason, a new approach was implemented, which handles filtering calculation in the offscreen drawing worker. Each line position is checked against the current filter position. A message is then sent from the worker to the main thread that contains an array of boolean values indicating whether a record is filtered or not. This approach and the fact that calculations are performed offscreen significantly improves the smoothness and rendering performance of filtering.

## 6.7 Selection Tools

The Scatterplot and Similarity Map panels have two additional selection tools for selecting records: lasso selection and box selection. The Parallel Coordinates panel also has two additional selection tools: line selection and box selection.

Lasso selection was implemented using the point-in-polygon algorithm [Shimrat 1962; Andrews 1991; MacWright 2021]. Listing 6.4 shows the implemented code for the algorithm. Polygon points are determined by the user's mouse movement when pressing the mouse and dragging. Points are added with a debounce timer of 500 milliseconds to reduce the complexity of calculations.

The Three.js official documentation provides an example of box selection [Herzog 2024], where the user drags out a box with the mouse. However, in MVA, it was more efficient to use the same point-in-polygon test for box selection too, where the box is simply a rectangular polygon from the starting point where the mouse button was pressed down to the ending point at the current location.

In the Parallel Coordinates panel, line selection is implemented using a line-line intersection algorithm [Weisstein 2002]. Each line segment from each record (polyline) is compared to the selection line to determine whether the two lines intersect. Listing 6.5 shows the implemented code for the line intersection algorithm.

```

1 function isLineIntersecting(line: THREE.Line, lassoLine: CoordinateType[]) {
2   const linePoints = getPointsFromLine(line);
3
4   for (let i = 0; i < linePoints.length - 1; i++) {
5     for (let j = 0; j < lassoLine.length - 1; j++) {
6       if (
7         doesIntersect(
8           linePoints[i].x,
9           linePoints[i].y,
10          linePoints[i + 1].x,
11          linePoints[i + 1].y,
12          lassoLine[j].x,
13          lassoLine[j].y,
14          lassoLine[j + 1].x,
15          lassoLine[j + 1].y
16        )
17      )
18        return true;
19    }
20  }
21
22  return false;
23 }
24
25 function doesIntersect(a: number, b: number, c: number, d: number,
26   p: number, q: number, r: number, s: number) {
27   const det = (c - a) * (s - q) - (r - p) * (d - b);
28   if (det === 0) return false;
29
30   const lambda = ((s - q) * (r - a) + (p - r) * (s - b)) / det;
31   const gamma = ((b - d) * (r - a) + (c - a) * (s - b)) / det;
32   return 0 < lambda && lambda < 1 && 0 < gamma && gamma < 1;
33 }

```

**Listing 6.5:** Code for calculating whether two lines intersect.

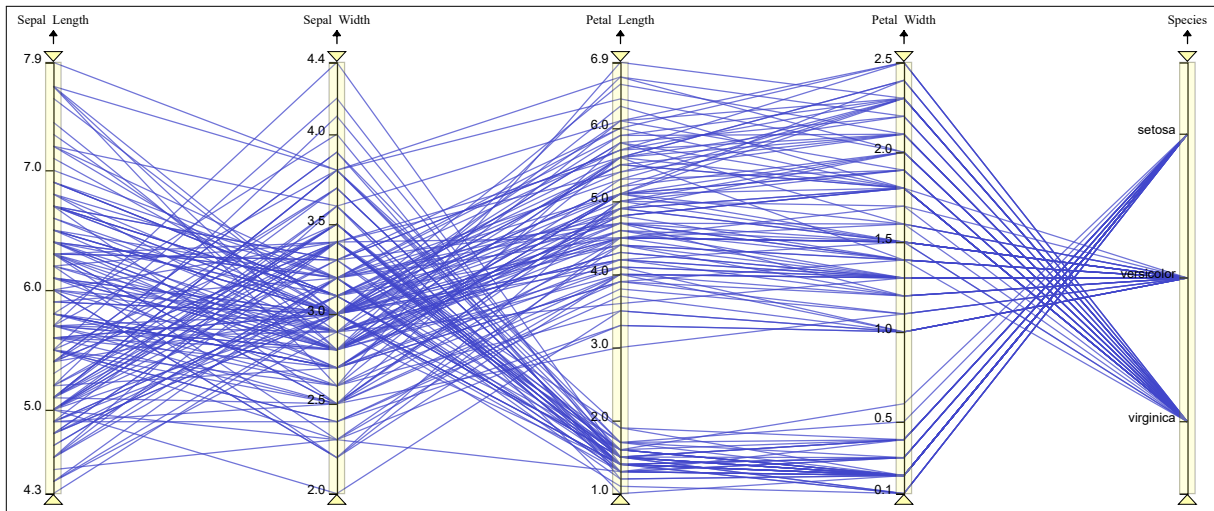
Box selection uses the same approach as line selection, where each line segment from each record is compared to all four edges of the selection box to determine whether the line and the box intersect.

## 6.8 SVG Exporter

SVG export is a basic requirement for any information visualization application. It enables the user to save the current visualization in a widely-used vector graphics format. In MVA, all visualization panel axes are drawn in an `<svg>` element using D3, which enables a straightforward way to save the axes as SVG. The XMLSerializer package [MDN 2024b] is used to serialize the `<svg>` element to a string in a human-readable format.

All data records in MVA's visualization panels are drawn using Three.js. Three.js does provide a way to save the WebGL-drawn graphics into an SVG. However, the result is not human-readable, which is not acceptable for MVA. To solve this issue, the data records are re-drawn using D3 as SVG. Records are drawn to an invisible `<svg>` element as SVG shapes, and are then serialized to a string using XMLSerializer. The invisible `<svg>` element is then deleted.





**Figure 6.7:** Example of a visualization exported as SVG.

Parallel Coordinates records are drawn as SVG `<line>` elements [W3C 2024b]. Scatterplot Matrix, Scatterplot, and Similarity Map records are drawn using SVG `<defs>` elements [W3C 2024a] to define markers and names. A `<defs>` element stores graphical objects which can later be reused with an SVG `<use>` element [W3C 2024c]. Finally, the axes and records are concatenated into a final merged SVG file for export. Figure 6.7 shows an example of a parallel coordinates visualization exported as SVG. Listing 6.6 shows some of the exported SVG code, shortened for brevity.

```

1 <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 951 386">
2   <!-- Lines -->
3   <g>
4     <path fill="none" stroke="#4146cb" stroke-width="1" stroke-opacity="0.75"
5       d="M30,301.333L250.25,166L470.5,353.22L690.75,362L911,96"/>
6     <path fill="none" stroke="#4146cb" stroke-width="1" stroke-opacity="0.75"
7       d="M30,320L250.25,236L470.5,353.22L690.75,362L911,96"/>
8   </g>
9   <!-- Axes -->
10  <g>
11    <g class="parcoord-y-axis" transform="translate(30, 40)" fill="none"
12      font-size="10" font-family="sans-serif" text-anchor="end">
13      <path class="domain" stroke="currentColor" d="M-6,336.5H0.5V0.5H-6"/>
14      <g class="tick" transform="translate(0,336.5)">
15        <line stroke="currentColor" x2="-6"/>
16        <text fill="currentColor" x="-9">4.3</text>
17      </g>
18    </g>
19    <text class="parcoord-axis-title" transform="translate(30, 10)"
20      font-size="0.625rem" style="text-anchor: middle;">Sepal_Length</text>
21    <svg class="parcoord-axis-invert-cursor-pointer" x="22" y="12" width="1rem"
22      height="1rem" stroke="#000" fill="#000">
23      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16">
24        <rect fill="white" stroke="white" width="16" height="16"/>
25        <path d="m 8.119999,13.438806 -0.01,-9.850000 m 0,0 L 5.22,
26          7.238806 M 8.109999,3.588806 10.83,7.238806 H 5.2"
27          stroke-linecap="round" stroke-linejoin="round"/>
28      </svg>
29    </svg>
30    <svg class="parcoord-axis-filter-upper" x="22" y="24" width="1rem"
31      height="1rem" stroke="#000" fill="rgba(255, 255, 100, 0.5)">
32      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 5.291666 5.291667">
33        <path stroke-width="0.259554"
34          d="M0.336363,2.522580H4.950769L2.646012,5.030853Z"/>
35      </svg>
36    </svg>
37    ...
38    <rect class="parcoord-axis-filter-rect" width="12" height="336" y="40"
39      fill="rgba(255, 255, 100, 0.2)" stroke="rgba(0, 0, 0, 0.25)"
40      transform="translate(24, 0)"/>
41  </g>
42 </svg>

```

**Listing 6.6:** Example of the exported shortened SVG code.

## Chapter 7

# Outlook and Future Work

MVA still has some limitations as well as room for potential future improvements. Some features could not be implemented due to the limitations of external libraries, and some features were out of the scope of the current thesis work.

### 7.1 Window Management

Currently, MVA does not have a comprehensive window management system for its panels. WinBox.js, a popular, lightweight, and powerful JavaScript-based window manager [Nextapps 2023], was initially used in MVA. However, WinBox.js does not support Svelte events and callbacks, upon which MVA relies heavily, especially in the Parallel Coordinates panel. The svelte-window-system was also trialed [Luneckas 2021], but also did not meet the complex requirements of MVA. In the future, alternatives for window management must be explored.

### 7.2 Rendering Records with Pixi.js

Subsection 4.4.7 compares different web graphics rendering libraries' performance. Despite Pixi.js being the top-performing library, it was not used for rendering records because of the insufficient quality of rendered lines (as noted in Section 6.1). If and when this Pixi.js rendering problem is addressed, the Pixi.js library could be used to render records. That would significantly improve the performance of MVA.

### 7.3 Rendering the Scatterplot Matrix

In the current implementation of the Scatterplot Matrix (SPLOM), the fact that the matrix is symmetric is not taken advantage of. It might be possible to speed up rendering of the SPLOM by having the web worker calculating cell  $(i, j)$  also be responsible for calculating cell  $(j, i)$ , by simply switching the x and y values.

### 7.4 t-SNE Similarity Map

Currently, the Similarity Map panel implements PCA and UMAP similarity mapping techniques. t-SNE was initially implemented using the `tsnejs` [Karpathy 2016] and the `tsne-js` [science.ai 2016] libraries. However, both of these libraries are implemented in JavaScript, and do not support TypeScript. They have also not been maintained since 2015. In the future, in order to integrate t-SNE, one of the previously mentioned libraries could be re-implemented in TypeScript.

## 7.5 Parallel Coordinates Matrix Panel

The parallel coordinates matrix is a multidimensional visual analysis approach described in Section 2.8. Providing a Parallel Coordinates Matrix panel was out of the scope of this thesis. However, it could be implemented in the future by re-using the Parallel Coordinates panel, thus adding another useful panel for the user to interpret data.

## 7.6 Handling Missing Data

Currently, records with missing data for one or more dimensions are not displayed in any panel. However, there are existing methods for handling missing data in parallel coordinates, as described in Section 2.7. In the future, all three mentioned approaches could be implemented at least in the Parallel Coordinates panel, and the user could select one based on the task at hand.

## 7.7 Automated Classification

Currently, MVA provides tools for an analyst to manually group and label records, placing them into non-overlapping classes (partitions). In the future, it would be possible to integrate some automated clustering and active learning approaches into MVA, to suggest possible classes for unlabeled records, like the features built into mVis [Chegini et al. 2019; Chegini 2021].

## 7.8 Rule-Based Definitions for Classes

Parallax [T. Avidan and S. Avidan 1999] provides sophisticated tools for defining and manipulating sets of records, in essence looking for formulaic rule-based definitions of particular classes of records to be used as potential classifiers. A similar approach could be explored in MVA.

## Chapter 8

# Concluding Remarks

This thesis presented the Multidimensional Visual Analyser (MVA), a web application for exploring and analysing multidimensional datasets. The first part of the thesis reviewed multidimensional visual analysis approaches in Chapter 2, multidimensional visual analysis tools in Chapter 3, and modern web technologies in Chapter 4. The second part of the thesis presented the MVA application. Chapter 5 described the overall architecture and implementation of MVA. Chapter 6 described some particularly interesting or tricky details of the implementation. Chapter 7 looked at some potential future improvements. The thesis has two appendices: Appendix A serves as a User Guide for end users of MVA, and Appendix B serves as a Developer Guide for developers wishing to modify or extend MVA.

This thesis produced an open-source web application which allows users to explore large, multidimensional datasets. MVA displays data in a scatterplot matrix, a scatterplot, a similarity map, parallel coordinates, and a table. It enables the user to group data records into partitions and supports brushing and linking, making the panels highly interconnected. By rendering records with WebGL, the application maintains its performance even when handling rather large datasets. MVA combines the best features from the reviewed multidimensional visual analysis software tools and implements them as both a web application and a desktop application. The parallel coordinates implemented in MVA offer more interactivity than in the other reviewed software tools. They can also display categorical dimensions, unlike the reviewed software. MVA source code is available on GitHub [Golob and Andrews 2024b]. MVA is deployed as a web application [Golob and Andrews 2024a], and can be built into a desktop application for Windows.



# Appendix A

## User Guide

This appendix serves as a User Guide for the Multidimensional Visual Analyser (MVA). Its purpose is to enable analysts to explore and analyze multidimensional datasets with MVA efficiently and effectively. All screenshots in this User Guide show MVA with the Cereals dataset, except those in Section A.6, which use the Student Marks dataset (example datasets are described in Section A.4).

### A.1 Installation

The web application version of MVA is available online at <https://tugraz-isds.github.io/mva/> [Golob and Andrews 2024a]. The source code for MVA is available at <https://github.com/tugraz-isds/mva/> [Golob and Andrews 2024b]. The desktop version of MVA for Windows can be downloaded from <https://github.com/tugraz-isds/mva/releases> and then installed and used offline.

### A.2 Features

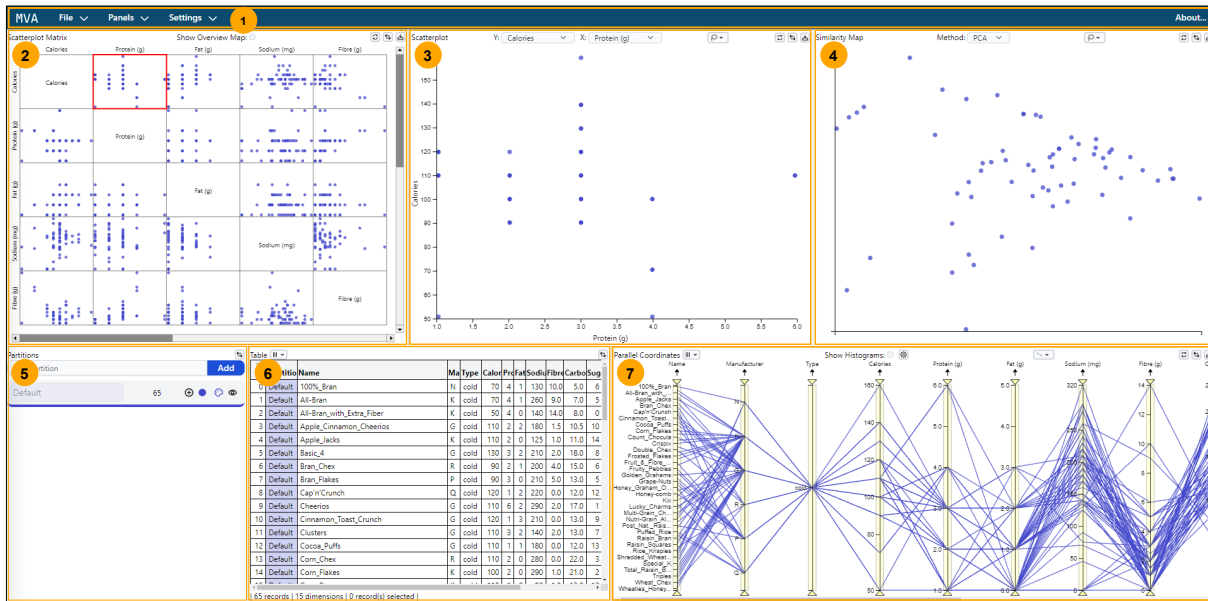
MVA is a powerful visual analysis tool that enables users to explore, analyze, and label multidimensional datasets. Once a dataset has been loaded into MVA, it is presented in up to six synchronized panels, which can be moved, resized, or hidden. The panels are highly interactive and are interconnected through brushing and linking. The user can filter data records via the Parallel Coordinates panel and group records into partitions via the Partitions panel. The datasets and created partitions can be exported and later re-imported. MVA saves the previously used dataset, partitions, visible dimensions, and panel sizes, making them available when re-opening the application. The current view of any visualization panel can be exported as SVG.

### A.3 User Interface

The MVA user interface has two main parts, as shown in Figure A.1: the Navigation Bar ① and the Display Area. The Display Area consists of up to six panels: Scatterplot Matrix ②, Scatterplot ③, Similarity Map ④, Partitions ⑤, Table ⑥, and Parallel Coordinates ⑦.

#### A.3.1 Initial State

The initial state of MVA, shown in Figure A.2, is presented once MVA is started. The application is initialized without a dataset, but gives the user the option to load the previously used dataset or to upload a new one. The user can upload a dataset from the Navigation Bar, or from the Import a Dataset section. The Example Datasets dropdown allows the user to import one of the example datasets described



**Figure A.1:** The Navigation Bar (1) and six synchronized panels of the Display Area: Scatterplot Matrix (2), Scatterplot (3), Similarity Map (4), Partitions (5), Table (6), and Parallel Coordinates (7).

in Section A.4. The Import Dataset... button opens the Import Dataset modal, shown in Figure A.5. The Import Previous button opens the previously used dataset with any saved partitions, and is shown only if a dataset was previously used.

### A.3.2 Hovering and Brushing

Hovered records are drawn in red and brushed (selected) records are drawn in orange, as shown in Figure A.3. A tooltip showing the labels of all hovered records is displayed near the mouse pointer. The dimension to be used to provide labels can be chosen from the Table panel. Records are added to the brushed set by left-clicking them, or by selecting them with one of the selection tools. Selection tools are activated by pressing the mouse down and drawing the shape. When adding to the brushed set, one of three possible scenarios occur, depending on the modifier:

- If the Shift key ( $\uparrow$ ) is pressed during selection, all hovered records are added to the brushed set.
- If the Control key ( $\text{Ctrl}$ ) is pressed during selection, all hovered records are toggled in the brushed set (i.e. if the record is already in brushed set, it is removed, otherwise it is added).
- Otherwise, a new brushed set is created, containing only the currently hovered records.

### A.3.3 Navigation Bar

The Navigation Bar contains global functions and settings for MVA. It comprises three dropdown menus: File, Panels, and Settings, and the About button.

The File dropdown is shown in Figure A.4a and consists of the following subitems: Example Datasets, Import Dataset..., and Export Dataset.... The Example Datasets menu item gives access to some example datasets that the user can easily import, described in Section A.4. The Import Dataset... menu item opens a modal to import a dataset from a local file and choose a cell and decimal separator. A dataset preview is shown, and the user can select whether each dimension is numerical or categorical. Figure A.5 shows the Import Dataset modal and the Dataset Preview table. The Export Dataset... menu item opens the Export Dataset modal, which allows the user to export the current dataset. The user can choose the dataset format



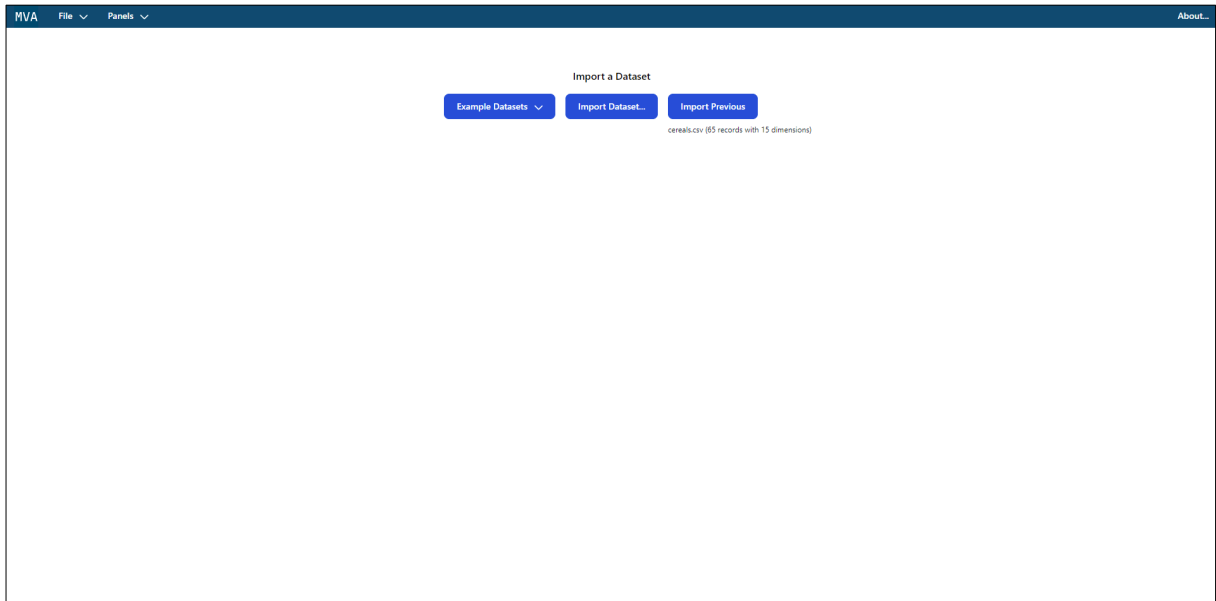


Figure A.2: The initial state of MVA.

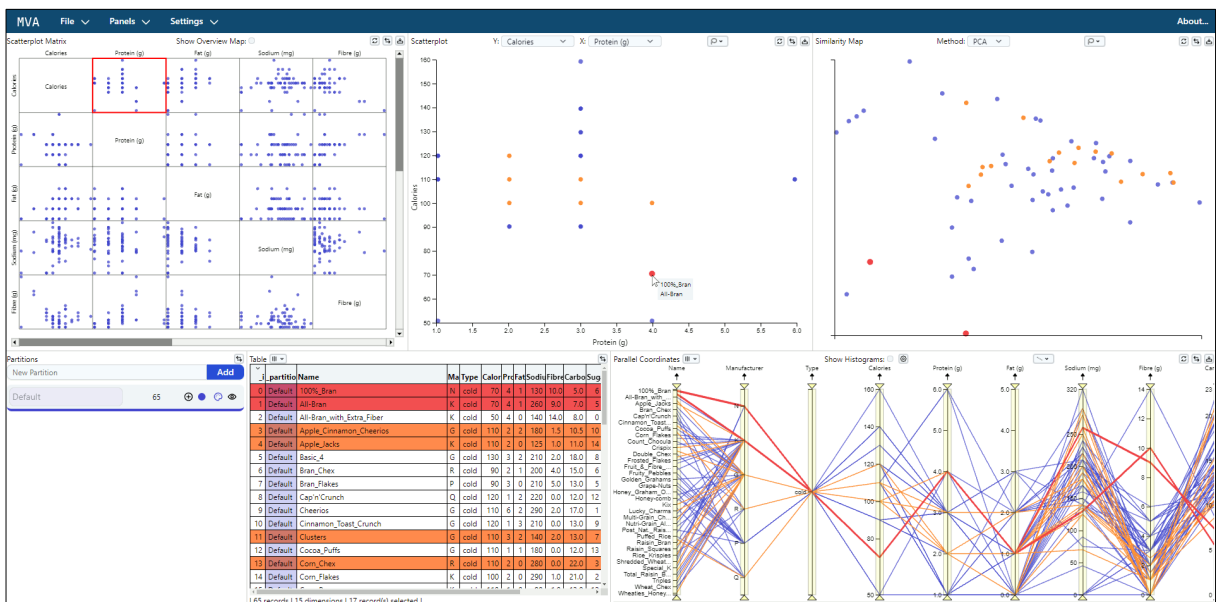
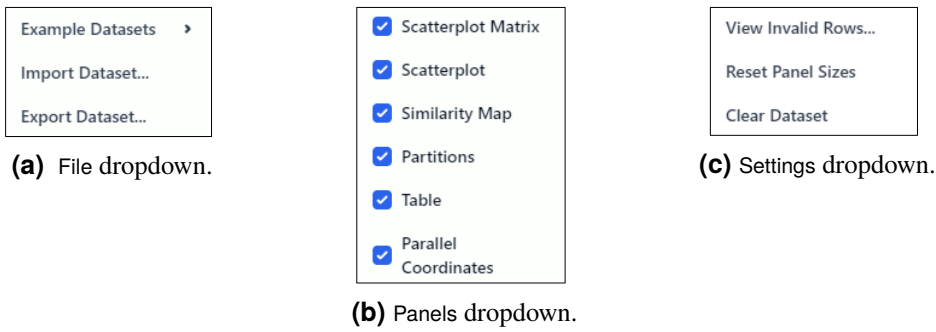
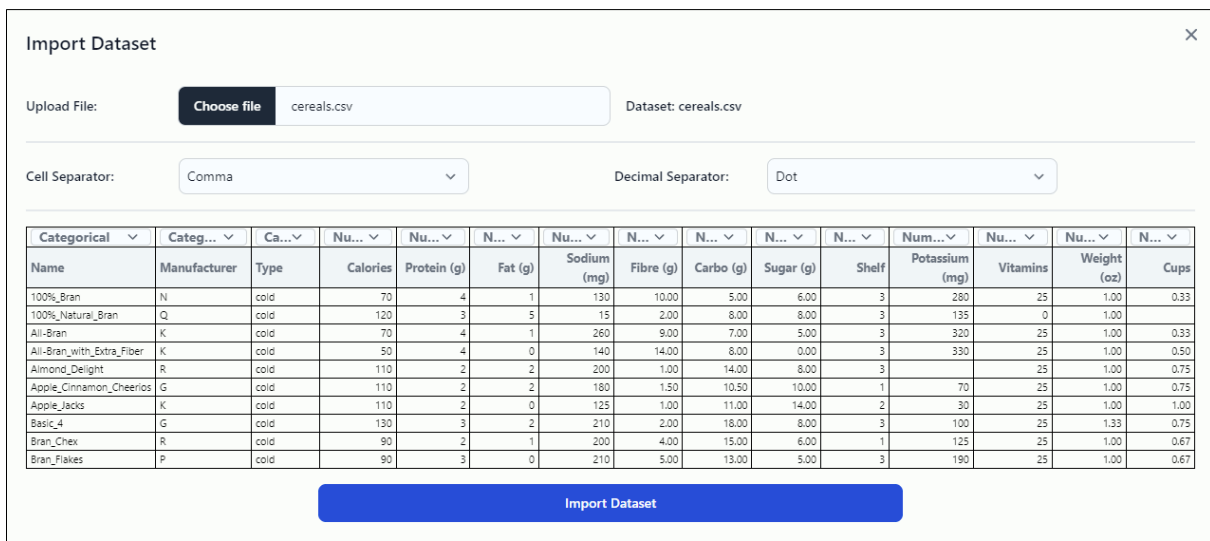


Figure A.3: MVA showing hovered records in red and brushed (selected) records in orange. The mouse pointer is hovering over two records located at the same position in the Scatterplot panel. A tooltip showing the labels of hovered records (here 100%\_Bran and All-Bran) is displayed near the mouse pointer.



**Figure A.4:** Navigation Bar dropdowns.



**Figure A.5:** Import Dataset modal.

(described in Section A.5), cell and decimal separators, and whether to export the data from inactive dimensions and partitions. Figure A.6 shows the Export Dataset modal.

The Panels dropdown allows the user to show or hide entire panels and is shown in Figure A.4b. The Settings dropdown is shown in Figure A.4c and consists of the following subitems: View Invalid Rows..., Reset Panel Sizes, and Clear Dataset. It is visible only if a dataset is loaded. The View Invalid Rows... menu item is visible only if the dataset contains one or more invalid rows, i.e. rows containing null or empty cell values. It opens the Invalid Rows modal, shown in Figure A.7, which shows all invalid rows. The Reset Panel Sizes menu item resets all panel sizes. The Clear Dataset menu item resets MVA to its initial state with no dataset loaded.

Finally, the About... button opens the About modal, shown in Figure A.8. It displays a short description of MVA, a link to the source code, and the current MVA version.

### A.3.4 Display Area

The Display Area comprises up to six synchronized panels: the four visualization panels Scatterplot Matrix, Scatterplot, Similarity Map, and Parallel Coordinates, as well as the Partitions panel and the Table panel, laid out by default as shown previously in Figure A.1. In the top right corner of each visualization panel, there are three buttons, as can be seen in the Parallel Coordinates panel in Figure A.9. The Refresh button

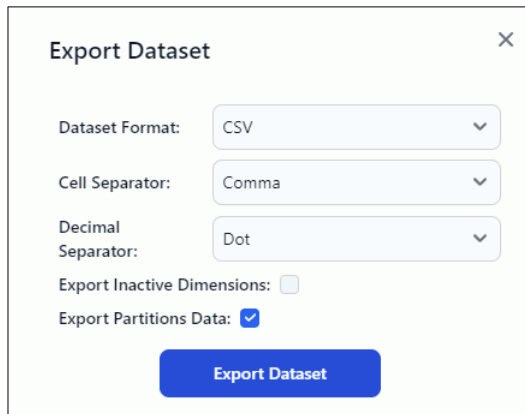


Figure A.6: Export Dataset modal.

Invalid Rows

Name	Manufacturer	Type	Calories	Protein (g)	Fat (g)	Sodium (mg)	Fibre (g)	Carbo (g)	Sugar (g)	Shelf	Potassium (mg)	Vitamins	Weight (oz)	Cups
100%_Natural_Bran	Q	cold	120	3	5	15	2	8	8	3	135	0	1	null
Almond_Delight	R	cold	110	2	2	200	1	14	8	3	null	25	1	0.75
Cream_of_Wheat_(Quick)	N	hot	100	3	0	80	1	21	0	2	null	0	1	1
Just_Right_Crunchy_Nuggets	K	cold	110	2	1	170	1	17	6	3	60	100	1	null
Maypo	A	hot	100	4	1	0	0	16	3	2	95	25	1	null
Muesli_Raisins_Dates_&Almonds	R	cold	150	4	3	95	3	16	11	3	170	25	null	null
Muesli_Raisins_Peaches_&Pecans	R	cold	150	4	3	150	3	16	11	3	170	25	null	null
Nutri-grain_Wheat	K	cold	90	3	0	170	3	18	2	3	90	25	1	null
Puffed_Wheat	Q	cold	50	2	0	0	1	10	0	3	50	0	0.5	null
Quaker_Oatmeal	Q	hot	100	5	2	0	2.7	null	null	1	110	0	1	0.67
Shredded_Wheat	N	cold	80	2	0	0	3	16	0	1	95	0	0.85	null
Strawberry_Fruit_Wheats	N	cold	90	2	0	15	3	15	5	2	90	25	1	null

Figure A.7: Invalid Rows modal.

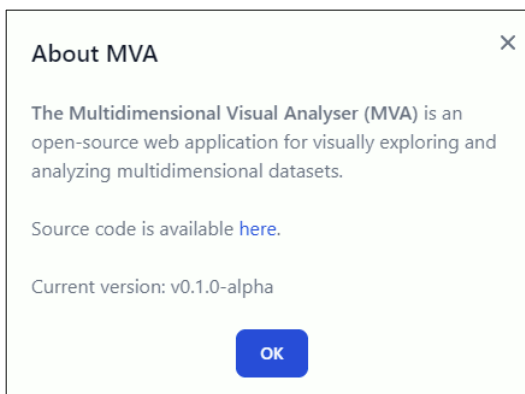
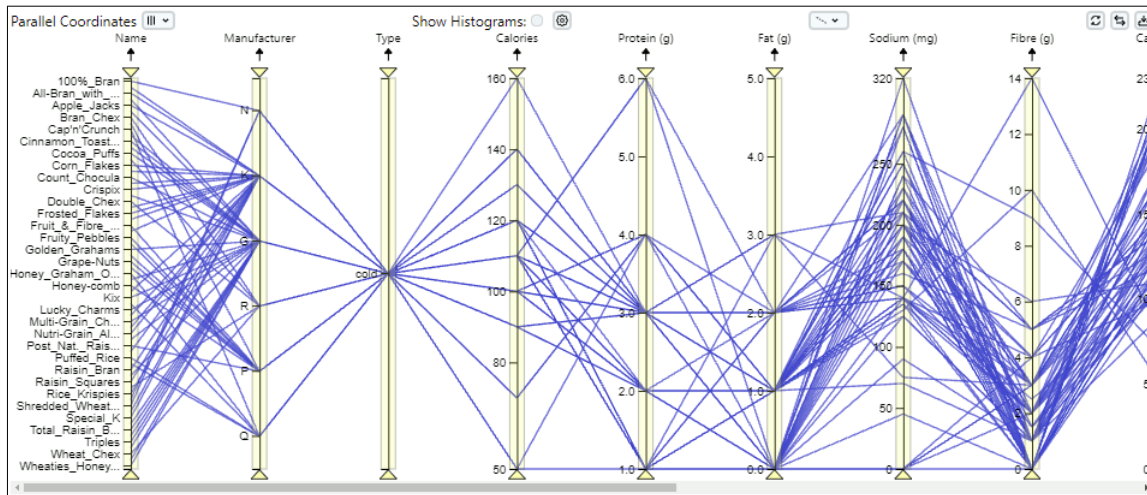






Figure A.8: About modal.



**Figure A.9:** The Refresh, Swap, and Save SVG can be found in the top right of every visualization panel, seen here in the Parallel Coordinates panel.

 is used to manually refresh a panel in case of a bug. The Swap button  is used for swapping panels. The Save SVG button  is used to save the current view of the visualization panel as an .svg file. The Partitions panel and the Table panel only have the Swap button .

### A.3.4.1 Scatterplot Matrix Panel

The Scatterplot Matrix panel displays a matrix of individual scatterplots, showing every numerical dimension plotted against every other numerical dimension in the dataset. If there are more than 5 numerical dimensions, only 5×5 scatterplots are visible at any one time and the scatterplot matrix becomes scrollable, as shown in Figure A.10. The analyst can use the scrollbars to the right and bottom of the panel to move around the full scatterplot matrix. In addition, the Overview Map can be activated by setting the corresponding Show Overview Map checkbox. It provides an overview of the 5×5 scatterplot range currently visible, and can be used to pan around the full scatterplot matrix.

The Scatterplot Matrix panel is connected to the Scatterplot panel. A scatterplot can be selected in the Scatterplot Matrix panel, which is then highlighted in red. This scatterplot is then loaded into the Scatterplot panel, so it can be examined in more detail. In the other direction, changing the scatterplot in the Scatterplot panel, by selecting new x or y dimensions, selects the corresponding scatterplot in the Scatterplot Matrix panel.

### A.3.4.2 Scatterplot Panel

The Scatterplot panel is shown in Figure A.11. A scatterplot plots any two numerical dimensions against each other. The user can select the dimension to be plotted on the x and y axes with the corresponding Dimension dropdown, shown in Figure A.12a. In order to select (brush) records in the Scatterplot panel, the user can click on records, or choose a selection tool from the Selection Tool dropdown, shown in Figure A.12b.

### A.3.4.3 Similarity Map Panel

The Similarity Map panel is shown in Figure A.13. A similarity map projects data points from a higher dimensional space to 2d space. The Similarity Map panel currently supports two similarity mapping techniques: PCA and UMAP. The user can select the desired similarity mapping technique from the Method dropdown, shown in Figure A.14a. In order to select (brush) records in the Similarity Map panel,

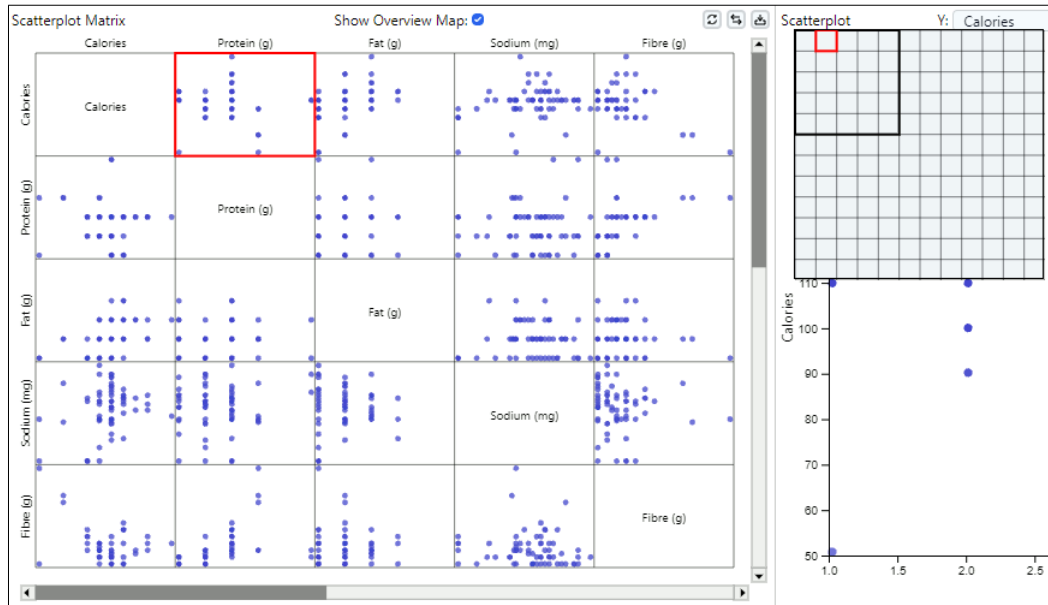


Figure A.10: The Scatterplot Matrix panel.

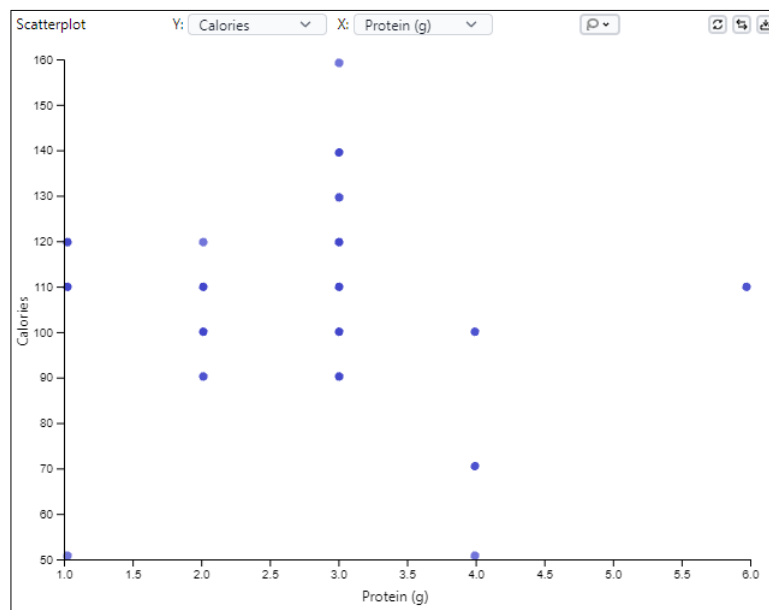
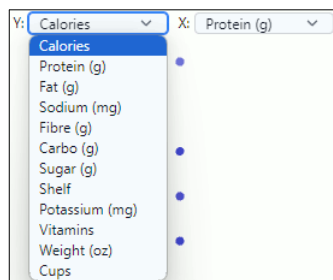
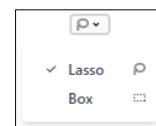


Figure A.11: Scatterplot panel.

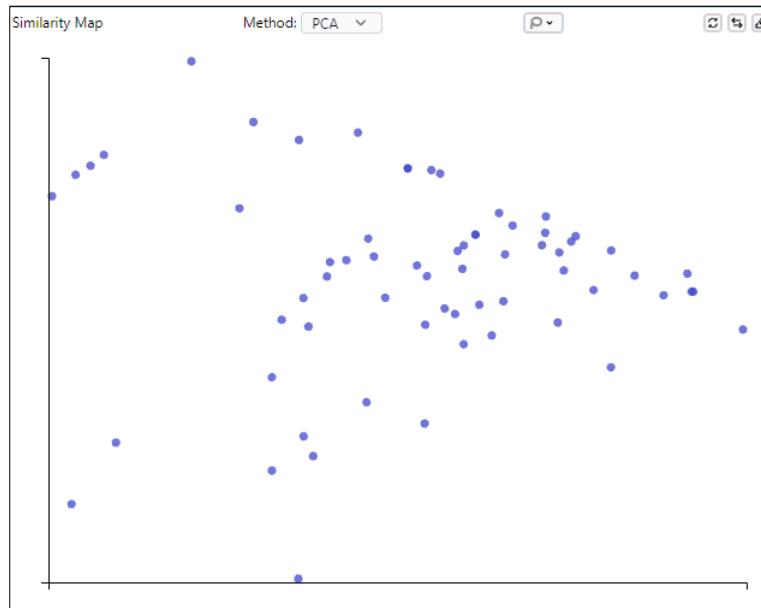


(a) Dimension dropdown.

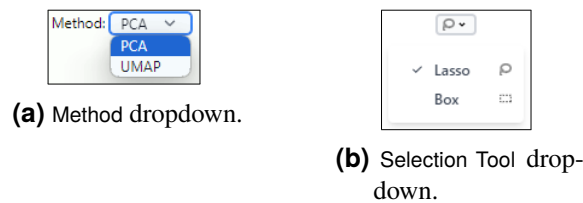


(b) Selection Tool dropdown.

Figure A.12: Scatterplot panel controls.



**Figure A.13:** Similarity Map panel.



**Figure A.14:** Similarity Map panel controls.

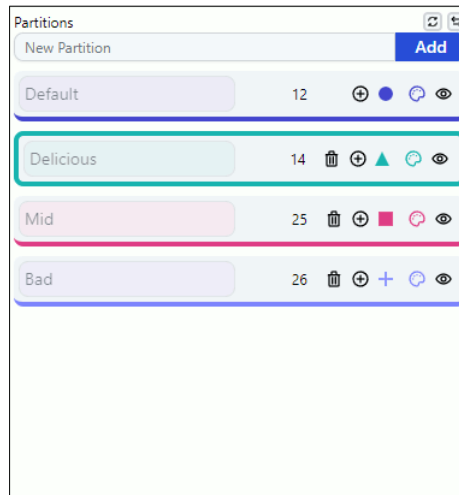
the user can click on records, or can choose one of the selection tools in the Selection Tool dropdown, shown in Figure A.14b.

#### A.3.4.4 Partitions Panel

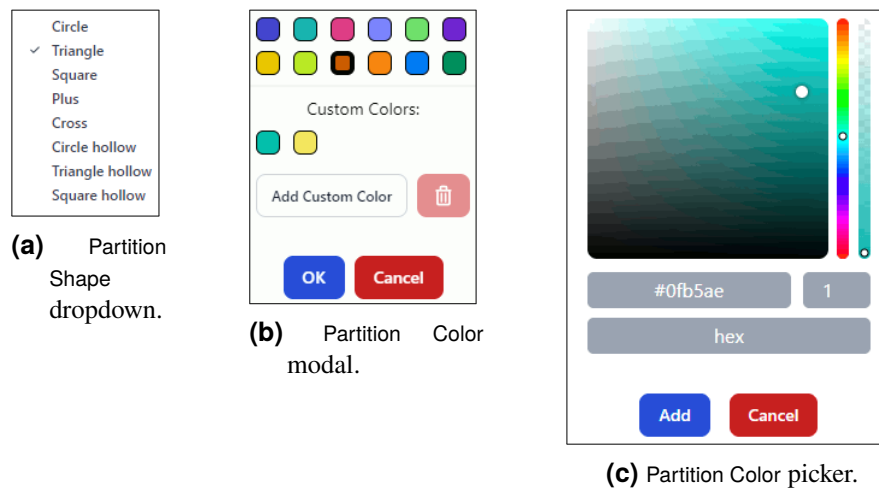
The Partitions panel displays a list of currently defined partitions. It is shown in Figure A.15. Partitions are non-overlapping groups (or classes) of data records that are labeled with the partition name. Each record may only belong to one partition. A new partition can be created from the New Partition text field, either by clicking the Add button or by pressing **Enter**. The Default partition initially contains all records and cannot be deleted. It serves as a fallback partition if other partitions are deleted.

Each partition has five properties: name, size, shape, color, and visibility. A partition can be renamed by clicking its Partition Name field, editing the name, and pressing the **Enter** key or clicking outside of the current partition element. Pressing **Esc** cancels renaming. The partition size shows the number of records in the current partition. The Delete Partition button deletes the current partition and assigns its records to the Default partition. The Add Selected Records button adds any currently brushed (selected) records to the partition.

The shape assigned to the records of a partition can be changed by clicking on the partition's shape icon to open the Partition Shape dropdown, shown in Figure A.16a. The color assigned to the records of a partition can be changed by clicking on the partition's color palette icon to open the Partition Color modal, shown in Figure A.16b. It contains 12 predefined colors, and has the option to add custom colors



**Figure A.15:** The Partitions panel shows a list of currently defined partitions. The currently selected partition is indicated by a thicker border.



**Figure A.16:** Partition controls.

with a color picker [Dupont 2024]. Pressing OK closes the Partition Color modal and pressing Cancel closes the modal and reverts the color selection. Pressing Add Custom Color opens the custom color picker component shown in Figure A.16c. The user can also delete custom colors. The records of a partition can be hidden (or shown again) in all visualization panels by clicking the partition's eye icon.

Brushed (selected) and hovered records can also be added to a partition using the Partition Context Menu, shown in Figure A.17a. The Partition Context Menu can be opened by right-clicking anywhere in the Scatterplot, Similarity Map, Parallel Coordinates, or Table panels. The Move to Default Partition menu item moves the currently selected records into the Default partition. The Add to Selected Partition menu item moves the currently selected records into the currently selected partition. The currently selected partition is indicated by a thicker border (see Figure A.15). The Add to Partition dropdown opens a list of all current partitions to move selected records to. Finally, the Add to New Partition... menu item opens the Add New Partition modal, shown in Figure A.17b, which creates a new partition and moves the selected records to it.



**Figure A.17:** The Partition Context Menu and its modal.

<i>_i</i>	<i>_partition</i>	Name	M	Type	Calo	Pr	Fa	Sodi	Fibr	Carb	Sug	SH	Pot
0	Delicious	100%_Bran	N	cold	70	4	1	130	10.0	5.0	6	3	280
1	Delicious	All-Bran	K	cold	70	4	1	260	9.0	7.0	5	3	320
2	Delicious	All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14.0	8.0	0	3	330
3	Delicious	Apple_Cinnamon_Cheerios	G	cold	110	2	2	180	1.5	10.5	10	1	70
4	Delicious	Apple_Jacks	K	cold	110	2	0	125	1.0	11.0	14	2	30
5	Delicious	Basic_4	G	cold	130	3	2	210	2.0	18.0	8	3	100
6	Delicious	Bran_Chex	R	cold	90	2	1	200	4.0	15.0	6	1	120
7	Delicious	Bran_Flakes	P	cold	90	3	0	210	5.0	13.0	5	3	190
8	Delicious	Cap'nCrunch	Q	cold	120	1	2	220	0.0	12.0	12	2	30
9	Delicious	Cheerios	G	cold	110	6	2	290	2.0	17.0	1	1	100
10	Delicious	Cinnamon_Toast_Crunch	G	cold	120	1	3	210	0.0	13.0	9	2	40
11	Delicious	Clusters	G	cold	110	3	2	140	2.0	13.0	7	3	100
12	Delicious	Cocoa_Puffs	G	cold	110	1	1	180	0.0	12.0	13	2	50
13	Delicious	Corn_Chex	R	cold	110	2	0	280	0.0	22.0	3	1	20
14	Mid	Corn_Flakes	K	cold	100	2	0	290	1.0	21.0	2	1	30

65 records | 0 selected |

**Figure A.18:** The Table panel.

**A.3.4.5 Table Panel**

The Table panel displays the dataset in tabular form, with records as rows and dimensions as columns, as shown in Figure A.18. The user can select which dimensions (columns) are shown in the table from the Table Dimensions dropdown, shown in Figure A.19a. The Table panel displays all dataset dimensions and two dimensions used internally by MVA: *\_i* and *\_partition*. The *\_i* dimension displays the index of each record (its internal numbering). The *\_partition* displays the name of each record’s partition. All dimensions can be sorted in ascending or descending order by left-clicking the column header cell. When sorting, categorical dimensions are ordered alphabetically, and numerical dimensions are ordered numerically.

The user can open the Table Context Menu by right-clicking a cell in the table header row. It is shown in Figure A.19b and contains the following items: Use as Label and Set as Active/Inactive. Use as Label sets the dimension as the label dimension, used to label records in the visualizations. Set as Active/Inactive makes the corresponding dimension inactive, removing the dimension from all visualizations. Beneath the table, a status bar indicates the number of records, number of dimensions, and number of selected records.

**A.3.4.6 Parallel Coordinates Panel**

The Parallel Coordinates panel shows the entire dataset in a parallel coordinates visualization, as is shown in Figure A.20. In parallel coordinates, each dimension is shown as a vertical parallel line (axis) and each record is represented as a polyline touching each axis once.

The user can select which dimensions are shown from the Dimensions dropdown, shown in Figure A.21a.



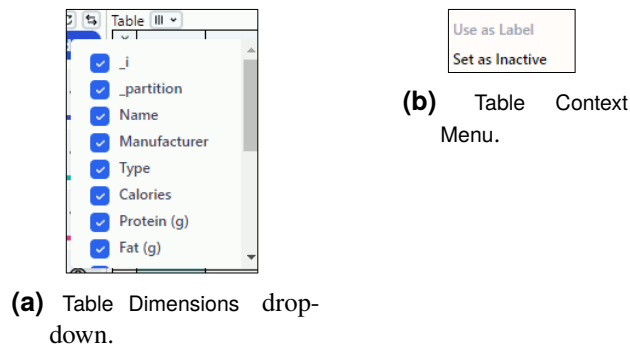


Figure A.19: Table panel controls.

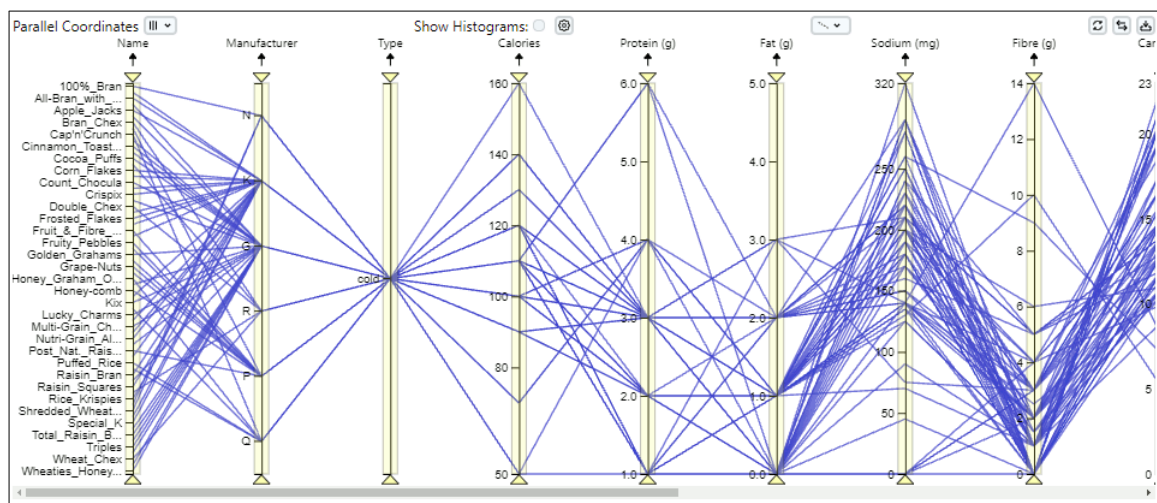


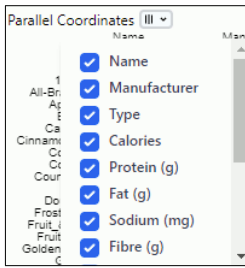
Figure A.20: The Parallel Coordinates panel.

The user can also toggle the display of histograms on every axis using the Show Histograms checkbox. Figure A.22 shows the Parallel Coordinates panel with histograms enabled. Histogram settings can be changed from the Histogram Settings modal, shown in Figure A.21b. To select (brush) records in the dataset, the user can click on records, or can choose a selection tool from the Selection Tool dropdown, shown in Figure A.21c. Line selection is the default.

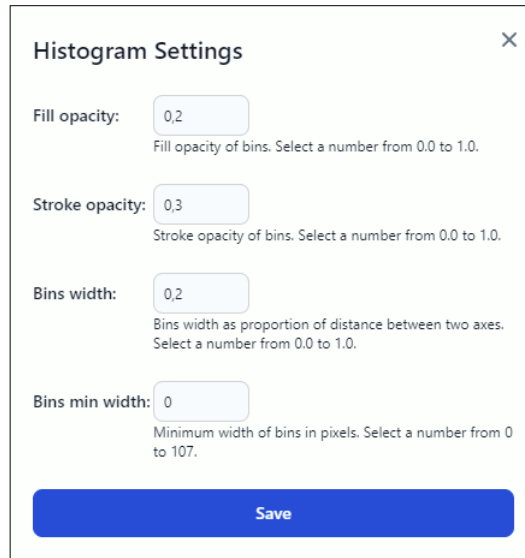
Parallel coordinates axes can be repositioned by dragging and dropping the axis title with the left mouse button. Each axis can be inverted by pressing its Invert Axis icon  $\uparrow$ . Filtering can be applied to an axis using its double-edged range slider, by dragging the Upper Limit  $\nabla$  or Lower Limit icon  $\triangle$ . The Filter Rectangle (rectangle between the Upper Limit and Lower Limit icons) can also be moved up or down. When filtering a numerical dimension, the current filter limits are displayed.

The Axis Context Menu, shown in Figure A.23, is opened by right-clicking the axis title. Hide Dimension hides the current dimension and Invert Dimension inverts it. The Show dropdown contains the following items which can be checked or unchecked: Labels (axis labels are shown/hidden), Histogram (axis histogram is shown/hidden, shown only when histograms are enabled), Filter (axis filter is shown/hidden), and Filter Values (axis filter values are shown/hidden).

The Set Range... menu item opens the Set Range modal, shown in Figure A.24a. It can be used to specify a custom range for the dimension. The Reset Range menu item resets the custom range back to the original range and is shown only when a custom range has been set. The Set Filter... menu item



(a) Parallel Coordinates Dimensions drop-down.



(b) Histograms Settings modal.



(c) Selection Tool drop-down.

Figure A.21: Parallel Coordinates panel controls.

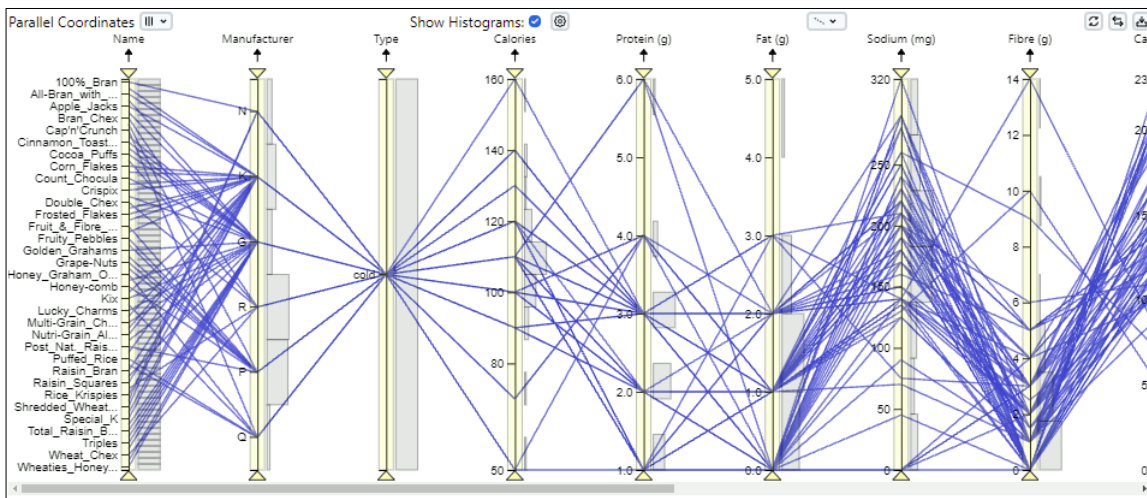


Figure A.22: Parallel Coordinates panel with histograms enabled.

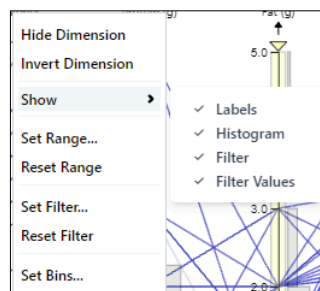


Figure A.23: Axis Context Menu.

Set Range for Dimension 'Calories' ✕

Min:  Max:

Original Range:

Min:  Max:

(a) Set Range modal.

Set Filter Values for Dimension 'Calories' ✕

Start:  End:

Dimension Range:

Min:  Max:

(b) Set Filter modal.

Set Bins ✕

Number of bins:

(c) Set Bins modal.

**Figure A.24:** Parallel Coordinates Axis Context Menu modals.

opens the Set Filter modal, shown in Figure A.24b. It enables the user to specify exact filter values for the dimension. Filter values cannot be set lower than the range minimum or higher than the range maximum. The Reset Filter menu item is shown only when filters are set and resets any filters. The Set Bins... menu item is shown only when histograms are enabled. It opens the Set Bins modal, shown in Figure A.24c, which is used to set the number of histogram bins for the current axis.

## A.4 Example Datasets

Five example datasets are included with MVA.

### A.4.1 Cars 1993

Cars 1993 is a dataset of car information from 1993 [Lock 1993]. There are 27 dimensions: make and 26 different features (manufacturer, model, type, min. price, price, max. price, MPG city, MPG highway, airbags, drive train, cylinders, engine size, horsepower, RPM, rev. per mile, man. trans. avail., fuel tank capacity, passengers, length, wheelbase, width, turn. circle, rear seat room, luggage room, weight, origin). After the header row, there are 93 data records representing 93 car makes. The 26 features in the dataset offer sufficient variety to illustrate a broad range of statistical techniques typically found in introductory courses.

### A.4.2 Cereals

Cereals is a dataset of different brands of cereals [DASL 2005]. There are 15 dimensions: cereal name and 14 different features (manufacturer, type, calories, protein, fat, sodium, fibre, carbo, sugar, shelf life, potassium, vitamins, weight, and cups). After the header row, there are 77 data records representing 77 different types of cereal.

### A.4.3 Iris

Iris is a dataset of Iris flower measurements [Fisher 1936]. There are 5 dimensions: the Iris species (Setosa, Versicolor, or Virginica) and 4 feature measurements in centimeters (sepal length, sepal width, petal length, and petal width). After the header row, there are 150 data records representing 150 flowers. The data set consists of 50 samples from each of the three species of Iris.

### A.4.4 Premier League

Premier League is a dataset of football player stats from Premier League in the 2019/2020 season [Samariya 2020]. There are 9 dimensions: the name of each player, their team, and their season stats in 7 categories (games played, games started, minutes, goals, assists, shots, and shots on goal). After the header row, there are 540 data records representing 540 players.

### A.4.5 Student Marks

Student Marks is a fictitious dataset of student marks [Drescher et al. 2023]. There are 9 dimensions: the name of each student and their marks from 0 to 100 in each of 8 subjects (Maths, English, PE, Art, History, IT, Biology, and German). After the header row, there are 30 data records representing 30 students. The dataset has been carefully curated to contain some outliers and some interesting patterns. For example, try to check the hypothesis that students who are good in one language will also be good in another language.

## A.5 Dataset Formats

MVA accepts two data formats: CSV files with the `.csv` extension and MVA files with the `.mva` extension. The data is parsed according to a user-defined separator (comma, semicolon, tab, space, or other). A dataset can be imported with or without predefined partitions. Partitions are defined with the following special columns: `_partition` (partition name), `_partition_color`, `_partition_shape`, and `_partition_order`. All of these columns must be present for partitions to be parsed correctly. Listing A.1 shows an example of the first few rows of the Iris dataset with partitions in standard CSV format.

To reduce the need for unnecessary duplication of partition data, partition data is only read from the first instance of the partition record, leading to an alternative format called Small CSV. An example of the first few rows of the Iris dataset with partitions in Small CSV format is shown in Listing A.2.

The MVA dataset format defines partition data at the start of the file, with one row per partition, each row starting with the character `#`. Listing A.3 shows an example of the first few rows of the Iris dataset with partitions in MVA format. The use of MVA format is recommended, as it is the most space-efficient, especially with larger datasets.

```

1 Sepal_Length,Sepal_Width,Petal_Length,Petal_Width,Species,_partition,
  _partition_color,_partition_shape,_partition_order
2 5.1,3.5,1.4,0.2,setosa,Setosa,#0fb5ae,triangle,1
3 4.9,3,1.4,0.2,setosa,Setosa,#0fb5ae,triangle,1
4 4.7,3.2,1.3,0.2,setosa,Setosa,#0fb5ae,triangle,1
5 4.6,3.1,1.5,0.2,setosa,Setosa,#0fb5ae,triangle,1
6 5,3.6,1.4,0.2,setosa,Setosa,#0fb5ae,triangle,1

```

**Listing A.1:** The first five records of the Iris dataset with partitions in standard CSV format.

```

1 Sepal_Length,Sepal_Width,Petal_Length,Petal_Width,Species,_partition,
  _partition_color,_partition_shape,_partition_order
2 5.1,3.5,1.4,0.2,setosa,Setosa,#0fb5ae,triangle,1
3 4.9,3,1.4,0.2,setosa,Setosa,,,
4 4.7,3.2,1.3,0.2,setosa,Setosa,,,
5 4.6,3.1,1.5,0.2,setosa,Setosa,,,
6 5,3.6,1.4,0.2,setosa,Setosa,,,

```

**Listing A.2:** The first five records of the Iris dataset with partitions in Small CSV format.

```

1 #Default,#4146cb,circle
2 #Setosa,#0fb5ae,triangle
3 Sepal_Length,Sepal_Width,Petal_Length,Petal_Width,Species,_partition
4 5.1,3.5,1.4,0.2,setosa,Setosa
5 4.9,3,1.4,0.2,setosa,Setosa
6 4.7,3.2,1.3,0.2,setosa,Setosa
7 4.6,3.1,1.5,0.2,setosa,Setosa
8 5,3.6,1.4,0.2,setosa,Setosa

```

**Listing A.3:** The first five records of the Iris dataset with partitions in MVA format.

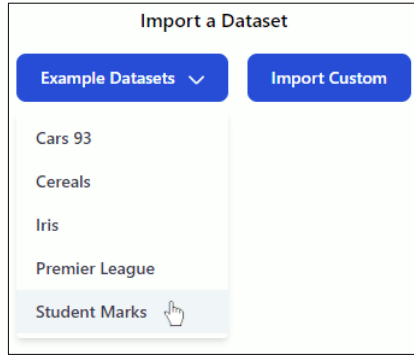


Figure A.25: Importing the Student Marks dataset into MVA.

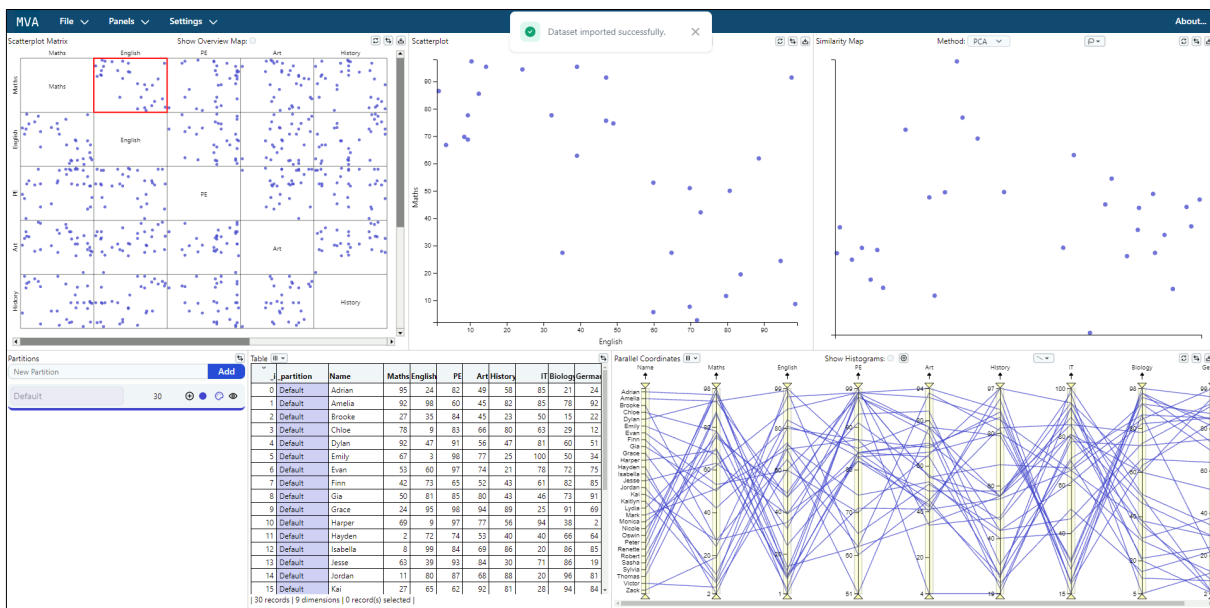


Figure A.26: MVA with the Student Marks dataset.

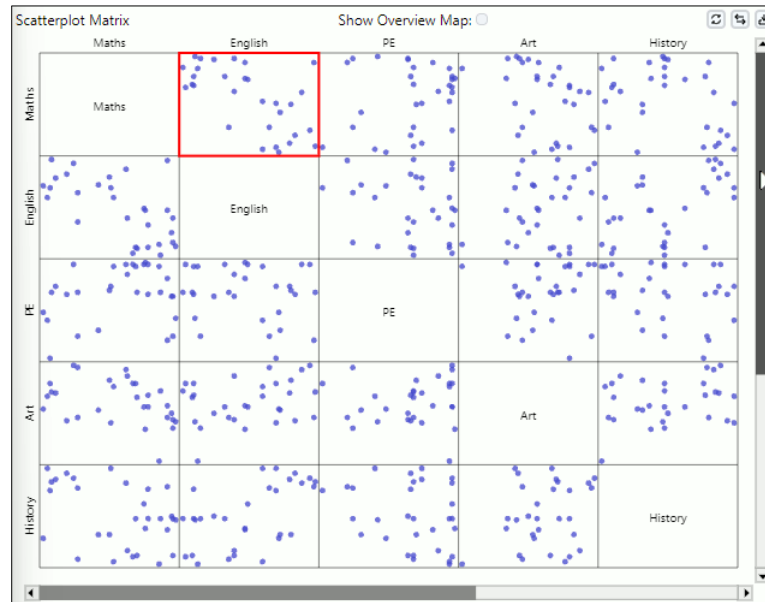
## A.6 Example Use Case

This section provides a step-by-step guide to using MVA by demonstrating its application with an example. The task involves exploring the Student Marks dataset, identifying correlations, and grouping the records into partitions.

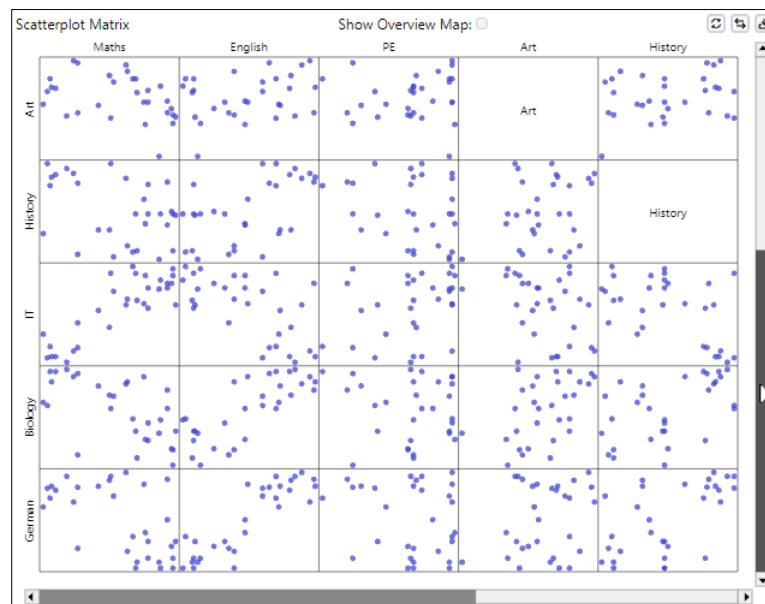
First, the user must import the Student Marks dataset by opening the Example Datasets dropdown and clicking on Student Marks, as shown in Figure A.25. Once the dataset is imported, all panels become visible and MVA is ready to use, as shown in Figure A.26.

The Scatterplot Matrix panel displays pairwise scatterplots between all dimensions, and can help identify any potential correlations, clustering, or outliers. The initial layout, shown in Figure A.27a, does not show any promising relationships. The user can scroll through the Scatterplot Matrix panel, as shown in Figure A.27b. It can be observed that there is now a potential relationship between IT and Maths, as well as between German and English, since the dots appear to form a rough diagonal line sloping upwards from left to right. If the user clicks on the IT and Maths scatterplot in the Scatterplot Matrix panel, the Scatterplot panel displays a scatterplot with Maths on the x axis and IT on the y axis, as shown in Figure A.28.

Further analysis can now be performed in the Parallel Coordinates panel by positioning the dimensions



(a) Initial Scatterplot Matrix.



(b) Scrolled Scatterplot Matrix.

Figure A.27: Scatterplot Matrix panel.

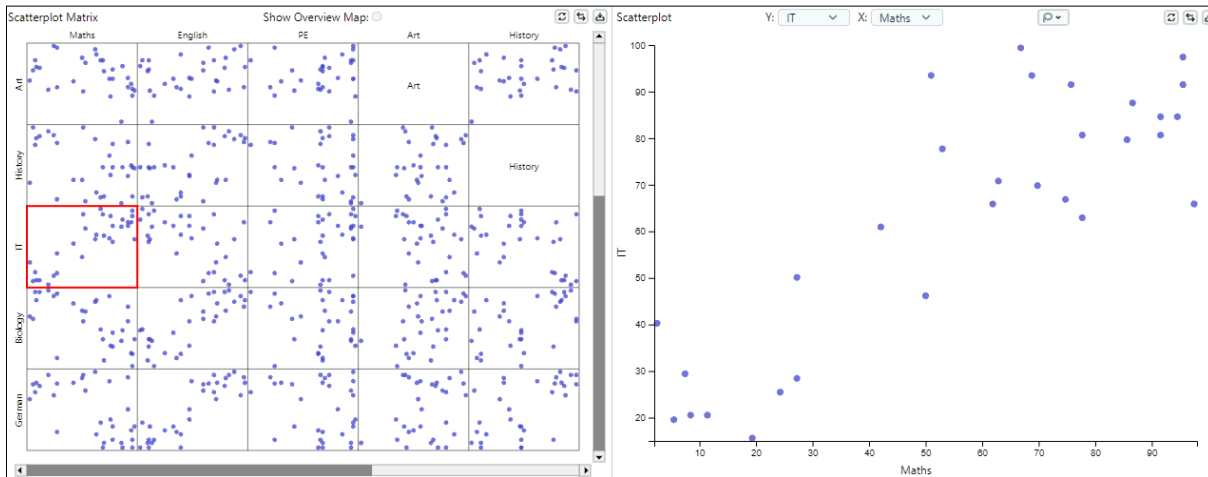


Figure A.28: Scatterplot Matrix and Scatterplot panels.

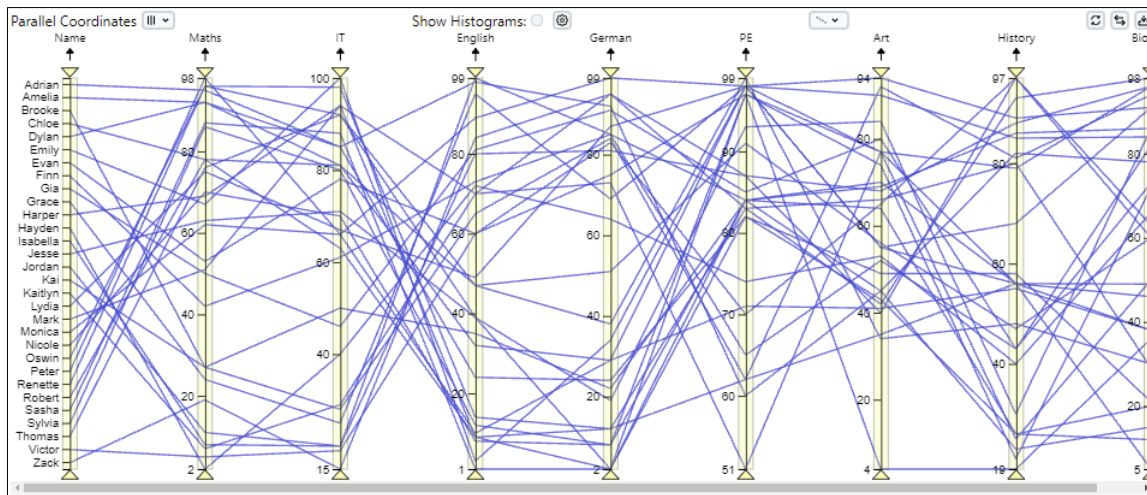


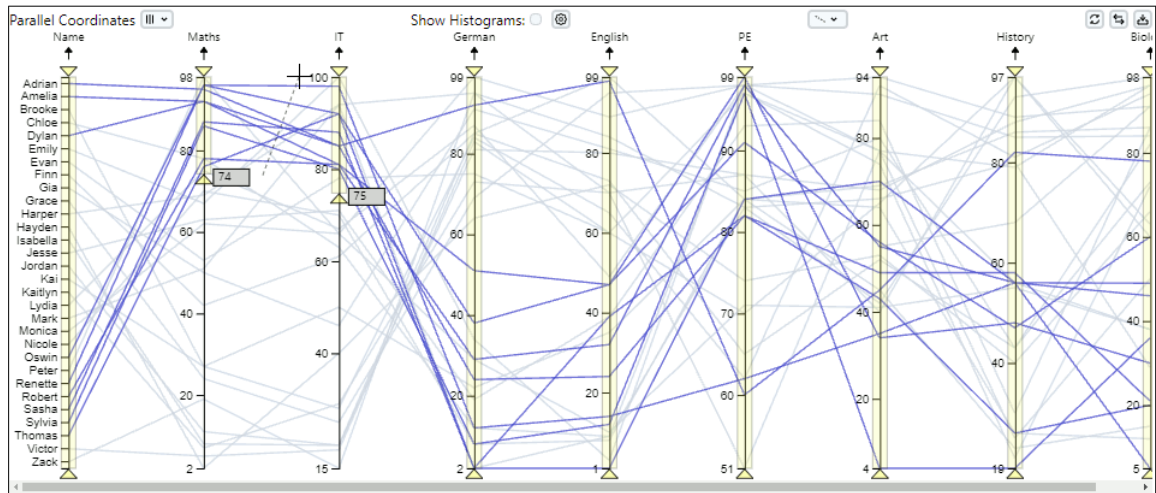
Figure A.29: Reordered dimensions in the Parallel Coordinates panel.

of interest next to each other. This can be achieved by dragging a dimension's title with the left mouse button and dropping it in the desired position. Figure A.29 shows reordered dimensions in the Parallel Coordinates panel, with Maths and IT, as well as English and German, now being adjacent to one another. It can be observed that the lines between Maths and IT, as well as the lines between English and German, are moving more or less in the same direction, indicating a possible positive correlation. It can be further observed that the lines between IT and English are moving more or less in opposite directions, forming a diagonal cross pattern, indicating a possible negative correlation.

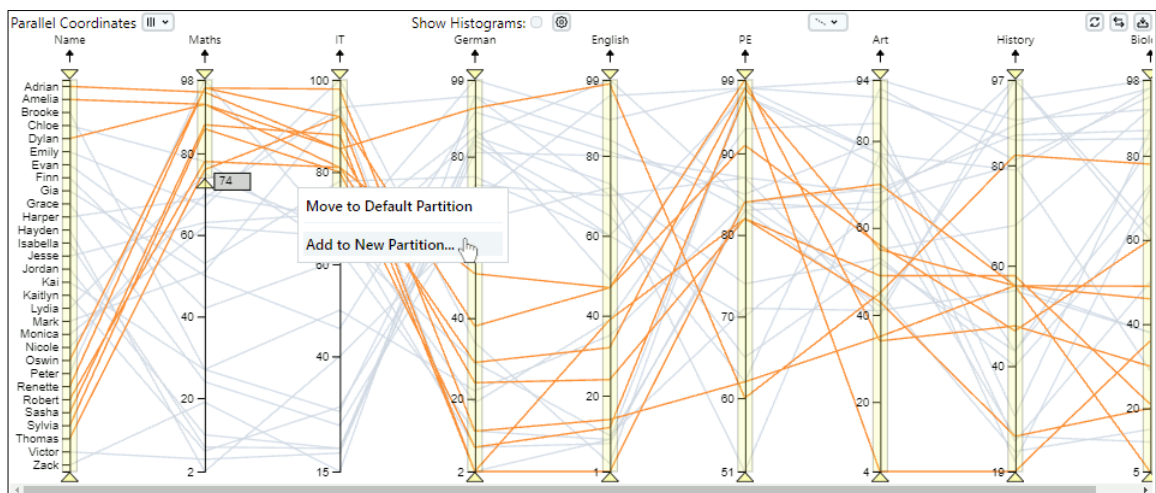
Once the relationships are apparent, the records can be arranged into partitions. In this example, two partitions will be created: Engineers (for students who excel in Maths and IT) and Linguists (for students who excel in English and German). The first step is selecting (brushing) the records of interest. Engineers will be selected by applying filtering to the Parallel Coordinates panel and brushing with the Line Selection tool, as shown in Figure A.30a. The selected records can be added to a new partition by right-clicking to open the Partition Context Menu, clicking Add to New Partition, and adding the Engineers partition through the Add New Partition modal, as shown in Figure A.30b.

Linguists will be selected from the Scatterplot panel using the Lasso Selection tool, as shown in Fig-





(a) After filtering, using the Line Selection tool.

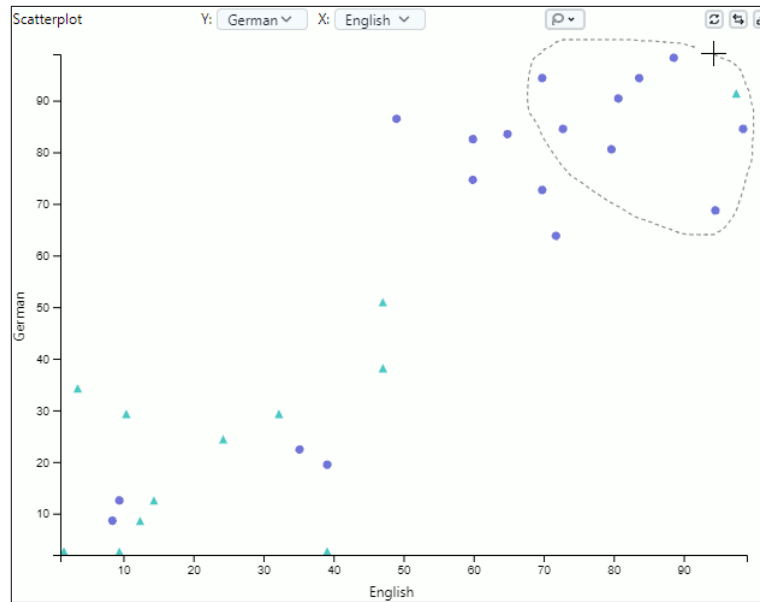


(b) Brushed (selected) records.

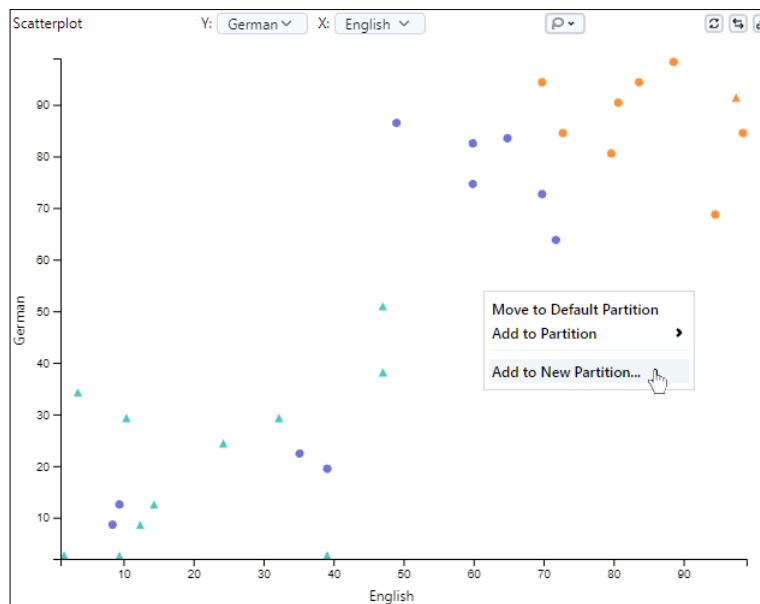
**Figure A.30:** In the Parallel Coordinates panel, selecting records and adding them to a new Engineers partition.

ure A.31a. The selected records can be added to a new partition by opening the Partition Context Menu, clicking Add to New Partition, and creating the new Linguists partition through the Add New Partition modal, as shown in Figure A.31b.

Finally, MVA displays the Student Marks records in three visually distinct partitions, as shown in Figure A.32. The partitioned dataset can now be saved in CSV or MVA format.



(a) Using the Lasso Selection tool.



(b) Brushed (selected) records.

**Figure A.31:** In the Scatterplot panel, selecting records and adding them to a new Linguists partition.



Figure A.32: The final look of the Student Marks dataset in MVA, after partitions have been added.



# Appendix B

## Developer Guide

This appendix serves as a Developer Guide for the Multidimensional Visual Analyser (MVA). MVA is an open-source web application licensed under an MIT license. The source code is available on GitHub [Golob and Andrews 2024b] at <https://tugraz-isds.github.io/mva/>.

### B.1 Quick Start

First, clone the MVA source code repository into a local folder and install Node [OpenJS 2024b] and npm [npm 2024].

To install all of the necessary dependencies, run the following command in the root folder:

```
npm install
```

To start a development server, run the following command:

```
npm run dev
```

To create a production-ready version of the application, run the following command:

```
npm run build
```

This creates a `build/` folder that can be deployed.

To preview the production build, run the following command:

```
npm run preview
```

This command previews the production build on localhost. However, it also introduces some strange UI bugs that are currently not reproducible in the development and production setup.

### B.2 Desktop Application

To create a desktop application package, the Tauri prerequisites have to be installed [Tauri 2024b]. Once the prerequisites are installed, install any fresh dependencies with:

```
npm install
```

and start a development server with:

```
npm run dev-tauri
```

To create a production-ready version of the application, run the following command:

```
npm run build-tauri
```

On Windows, this creates a folder `build-tauri/windows/` containing executable and installable packages. The desktop application is currently only supported on Windows, since the Linux and Mac versions are not built correctly.

## B.3 Gulp Tasks

The repository provides the following Gulp tasks:

- `clean`: Removes the existing `build/`, `build-tauri/`, and `src-tauri/target/` folders to enable a clean rebuild of the project. Run with `npx gulp clean`.
- `cleanAll`: Removes the existing `.svelte-kit/`, `build/`, `build-tauri/`, `node_modules/`, and `src-tauri/target/` folders to enable a clean reinitialization of the project. Run with `npx gulp cleanAll`.
- `optimizeIcons`: Reads the `src/static/icons/` folder recursively and exports `*.svg` files into a TypeScript constants file that is saved in `src/util/icon-definitions.ts`. This task should be run when new icons are added or existing ones are updated. Run with `npx gulp optimizeIcons`.  
The task is automatically run each time `npm run dev` or `npm run build` are executed.

## B.4 Development Dependencies

MVA has the following development dependencies, defined in the `package.json` file:

- `@sveltejs/adaptor-auto`, `@sveltejs/adaptor-static`, `@sveltejs/kit`, `@sveltejs/vite-plugin-svelte`, `svelte`, `svelte-check`, `vite`: SvelteKit dependencies.
- `@tauri-apps/cli`: Used for running frequent Tauri tasks.
- `@types/d3-array`, `@types/d3-axis`, `@types/d3-drag`, `@types/d3-dsv`, `@types/d3-scale`, `@types/d3-selection`, `@types/d3-shape`: TypeScript definitions for D3 submodules.
- `@types/three`: TypeScript definitions for Three.js.
- `@typescript-eslint/eslint-plugin`, `@typescript-eslint/parser`, `eslint`, `eslint-config-prettier`, `eslint-plugin-svelte`, `prettier`, `prettier-plugin-svelte`: Plugins for linting and formatting code.
- `autoprefixer`, `postcss`, `postcss-load-config`, `tailwindcss`: Tailwind CSS plugins, required by the Flowbite component library.
- `del`: Used for deleting files and folders.
- `flowbite-svelte-icons`: Free and open-source icon library.
- `gulp`: Task runner for automating development workflow.
- `tslib`, `typescript`: Plugins for TypeScript.

# Bibliography

- Abdi, Hervé and Lynne J. Williams [2010]. *Principal Component Analysis*. Wiley Interdisciplinary Reviews: Computational Statistics 2.4 (15 Jul 2010), pages 433–459. ISSN 1939-0068. doi:10.1002/wics.101. <https://personal.utdallas.edu/~herve/abdi-awPCA2010.pdf> (cited on page 9).
- Accomazzo, Anthony, Ari Lerner, Nate Murray, Clay Allsopp, David Guttman, and Tyler McGinnis [2017]. *Fullstack React*. Fullstack.io, 12 Sep 2017. ISBN 0991344626 (cited on page 29).
- Ahmed, Kamran [2024a]. *Backend Developer Roadmap*. 05 Sep 2024. <https://roadmap.sh/backend> (cited on page 27).
- Ahmed, Kamran [2024b]. *Frontend Developer Roadmap*. 05 Sep 2024. <https://roadmap.sh/frontend> (cited on page 27).
- Andrews, Keith [1991]. *Bounding Box and Hit Detection Algorithms for 2D Graphic Objects*. Proc. Conference on Intelligent Systems (CIS'91) (Veszprém, Hungary). Edited by A. György. John von Neumann Society for Computing Sciences (NJSZT). Sep 1991, pages 95–105. ISBN 9638431741. <https://ftp.isds.tugraz.at/pub/papers/andrews-cis91-bbhit.pdf> (cited on page 51).
- Andrews, Keith [2021]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. 03 Dec 2021. <https://ftp.isds.tugraz.at/pub/keith/thesis/> (cited on page xiii).
- Andrews, Keith [2024a]. *Information Visualisation: Course Notes*. 08 Mar 2024. <https://courses.isds.tugraz.at/ivis/ivis.pdf> (cited on page 1).
- Andrews, Keith [2024b]. *Introduction to Information Visualisation*. Tutorial at Interact 2023, York, UK. 30 Aug 2024. <https://keithandrews.com/talks/2023/2023-08-30-interact-infovis/> (cited on page 30).
- Avidan, Tova and Shlomo Avidan [1999]. *ParallAX – A Data Mining Tool Based on Parallel Coordinates*. Computational Statistics 14.1 (05 Sep 1999), pages 79–89. ISSN 0943-4062. doi:10.1007/PL00022707. <https://proquest.com/docview/2299676854?sourcetype=Scholarly%20Journals> (cited on pages 16–17, 56).
- Bäuerle, Alex, Christian van Onzenoodt, Simon der Kinderen, J. Johansson Westberg, Daniel Jönsson, and Timo Ropinski [2022]. *Where Did My Lines Go? Visualizing Missing Data in Parallel Coordinates*. Computer Graphics Forum 41.3 (29 Jul 2022), pages 235–246. doi:10.1111/cgf.14536. [https://viscom.publications.uni-ulm.de/api/uploads/236/pc\\_missing\\_data.pdf](https://viscom.publications.uni-ulm.de/api/uploads/236/pc_missing_data.pdf) (cited on pages 11–12).
- Berson, Alex [1996]. *Client/Server Architecture*. McGraw-Hill, 29 Mar 1996. ISBN 0070056641 (cited on page 27).
- Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer [2011]. *D3: Data-Driven Documents*. IEEE Transactions on Visualization and Computer Graphics 17.12 (23 Oct 2011), pages 2301–2309. doi:10.1109/TVCG.2011.185. <https://idl.cs.washington.edu/files/2011-D3-InfoVis.pdf> (cited on page 33).
- Bostock, Mike [2023a]. *d3-array*. 06 Oct 2023. <https://github.com/d3/d3-array> (cited on page 40).

- Bostock, Mike [2023b]. *d3-axis*. 06 Oct 2023. <https://github.com/d3/d3-axis> (cited on page 40).
- Bostock, Mike [2023c]. *d3-drag*. 06 Oct 2023. <https://github.com/d3/d3-drag> (cited on page 40).
- Bostock, Mike [2023d]. *d3-dsv*. 06 Oct 2023. <https://github.com/d3/d3-dsv> (cited on page 40).
- Bostock, Mike [2023e]. *d3-scale*. 06 Oct 2023. <https://github.com/d3/d3-scale> (cited on page 40).
- Bostock, Mike [2023f]. *d3-selection*. 06 Oct 2023. <https://github.com/d3/d3-selection> (cited on page 40).
- Bostock, Mike [2023g]. *d3-shape*. 06 Oct 2023. <https://github.com/d3/d3-shape> (cited on page 40).
- Bostock, Mike [2024]. *D3.js*. 05 Sep 2024. <https://d3js.org/> (cited on page 33).
- Bottin, Chris [2024]. *xml-formatter*. 07 Jul 2024. <https://github.com/chrisbottin/xml-formatter> (cited on page 41).
- Brandel, Jono [2024]. *Two.js*. 09 Jul 2024. <https://github.com/jonobr1/two.js> (cited on page 32).
- Brodbeck, Dominique and Luc Girardin [2003]. *Using Multiple Coordinated Views to Analyze Geo-Referenced High-Dimensional Datasets*. Proc. International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2003). 15 Jul 2003, pages 104–111. doi:10.1109/CMV.2003.1215008. <http://download.macrofocus.com/publications/cmv2003.pdf> (cited on page 18).
- Bublitz, Blaine and Eric Schoffstall [2024]. *Gulp*. 08 Apr 2024. <https://github.com/gulpjs/gulp> (cited on page 40).
- Cabello, Ricardo [2024a]. *Three.js*. 05 Sep 2024. <https://threejs.org/> (cited on pages 32, 41).
- Cabello, Ricardo [2024b]. *Three.js Raycaster*. 05 Sep 2024. <https://threejs.org/docs/#api/en/core/Raycaster> (cited on page 49).
- Carr, Daniel B., Richard J. Littlefield, W. L. Nicholson, and J. S. Littlefield [1986]. *Scatterplot Matrix Techniques for Large N*. Journal of the American Statistical Association 82.398 (01 Jul 1986), pages 424–436. doi:10.1080/01621459.1987.10478445. <https://jstor.org/stable/2289444> (cited on page 4).
- Castillo, Dany [2024]. *tailwind-merge*. 02 Sep 2024. <https://github.com/dcastil/tailwind-merge> (cited on page 40).
- Catuhe, David [2024]. *Babylon.js*. 05 Sep 2024. <https://babylonjs.com/> (cited on page 33).
- Chegini, Mohammad [2021]. *mVis Github*. 20 Jan 2021. <https://github.com/chegini91/mVis> (cited on pages 24, 56).
- Chegini, Mohammad, Jürgen Bernard, Philip Berger, Alexei Sourin, Keith Andrews, and Tobias Schreck [2019]. *Interactive Labelling of a Multivariate Dataset for Supervised Machine Learning using Linked Visualisations, Clustering, and Active Learning*. Visual Informatics 3.1 (Mar 2019), pages 9–17. doi:10.1016/j.visinf.2019.03.002. <https://ftp.isds.tugraz.at/pub/papers/chegini-pvast2019-ix-labelling.pdf> (cited on pages 24–25, 56).
- Cook, Dianne, Deborah F. Swayne, and Andreas Buja [2007]. *Interactive and Dynamic Graphics for Data Analysis with R and GGobi*. Springer, 05 Sep 2007. ISBN 0387717617 (cited on pages 17–18).
- DASL [2005]. *Cereals*. 04 Apr 2005. <http://lib.stat.cmu.edu/DASL/Datafiles/Cereals.html> (cited on page 71).
- De Rochefort, Enguerrand [2020]. *XDAT Github*. 26 Aug 2020. <https://github.com/enguerrand/xdat> (cited on page 19).



- De Rochefort, Enguerrand [2024]. *XDAT Home Page*. 05 Sep 2024. <https://xdat.org/> (cited on pages 19–20).
- Deveria, Alexis [2024]. *Can I Use: WebGPU*. 05 Sep 2024. <https://caniuse.com/webgpu> (cited on page 31).
- Drescher, Philipp, Jeremias Kleinschuster, Sebastian Schreiner, and Burim Vrella [2023]. *Steerable Parallel Coordinates in JavaScript*. Project Report. Graz University of Technology, 16 Jun 2023. <https://courses.isds.tugraz.at/ivis/projects/ss2023/ivis-ss2023-g1-project-steerable-parcoords.pdf> (cited on page 72).
- Dupont, Maxime [2024]. *svelte-awesome-color-picker*. 13 Aug 2024. <https://github.com/Ennoriel/svelte-awesome-color-picker> (cited on pages 40, 67).
- Dzemyda, Gintautas, Olga Kurasova, and Julius Žilinskas [2012]. *Multidimensional Data Visualization: Methods and Applications*. Springer, 09 Nov 2012. ISBN 144190235X (cited on pages 1, 3).
- Few, Stephen [2021]. *Now You See It: An Introduction to Visual Data Sensemaking*. 2<sup>nd</sup> Edition. Analytics Press, 15 Apr 2021. 301 pages. ISBN 1938377125 (cited on page 1).
- Fierens, Wout [2024]. *SVG.js*. 16 Sep 2024. <https://svgjs.dev/docs/3.2/> (cited on page 32).
- Filipova, Olga [2016]. *Learning Vue.js 2*. Packt Publishing, 13 Dec 2016. ISBN 1786469944 (cited on page 29).
- Fisher, Ronald A [1936]. *The Use of Multiple Measurements in Taxonomic Problems*. *Annals of Eugenics* 7.2 (1936), pages 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x. <https://1gross.utk.edu/Math589Fall2020/RAFisher1936measurementsFlowerTaxa.pdf> (cited on pages 3, 16, 72).
- Friendly, Michael and Daniel Denis [2005]. *The Early Origins and Development of the Scatterplot*. *Journal of the History of the Behavioral Sciences* 41.2 (2005), pages 103–130. doi:10.1002/jhbs.20078. <https://datavis.ca/papers/friendly-scat.pdf> (cited on page 3).
- Fry, Ben [2007]. *Visualizing Data: Exploring and Explaining Data with the Processing Environment*. O’Reilly, 18 Dec 2007. ISBN 0596514557 (cited on page 1).
- GGobi [2024]. *GGobi Home Page*. 05 Sep 2024. <http://ggobi.org/> (cited on page 17).
- Girardin, Luc and Dominique Brodbeck [2001]. *Interactive Visualization of Prices and Earnings Around the Globe*. Proc. IEEE Symposium on Information Visualization (InfoVis 2001) (San Diego, California, USA). 21 Oct 2001. <http://download.macrofocus.com/publications/infovis2001.pdf> (cited on page 18).
- Golob, Ožbej and Keith Andrews [2024a]. *MVA Live Demo*. 14 Sep 2024. <https://tugraz-isds.github.io/mva/> (cited on pages 1, 57, 59).
- Golob, Ožbej and Keith Andrews [2024b]. *MVA Repository*. 03 Sep 2024. <https://github.com/tugraz-isds/mva> (cited on pages 1, 57, 59, 81).
- Google [2024]. *Angular*. 05 Sep 2024. <https://angular.io/> (cited on page 28).
- Groves, Matt [2024]. *Pixi.js*. 05 Sep 2024. <https://pixijs.com/> (cited on page 32).
- Harris, Rich [2024a]. *Svelte*. 05 Sep 2024. <https://svelte.dev/> (cited on page 29).
- Harris, Rich [2024b]. *SvelteKit*. 05 Sep 2024. <https://kit.svelte.dev/> (cited on page 30).
- Heinrich, Julian, John Stasko, and Daniel Weiskopf [2012]. *The Parallel Coordinates Matrix*. Proc. Eurographics Conference on Visualization (EuroVis 2012). Eurographics, 2012. doi:10.2312/PE/EuroVisShort/EuroVisShort2012/037-041. [https://joules.de/files/heinrich\\_parallel\\_2012.pdf](https://joules.de/files/heinrich_parallel_2012.pdf) (cited on page 13).

- Herzog, Michael [2024]. *Three.js Box Selection*. 30 Apr 2024. [https://threejs.org/examples/?q=box%20se#misc\\_boxselection](https://threejs.org/examples/?q=box%20se#misc_boxselection) (cited on page 51).
- Hoffman, Patrick, Georges Grinstein, Kenneth Marx, Ivo Grosse, and Eugene Stanley [1997]. *DNA Visual and Analytic Data Mining*. Proc. 8<sup>th</sup> IEEE Conference on Visualization (Vis '97) (Phoenix, Arizona, USA). 24 Oct 1997, pages 437–441. doi:10.1109/VISUAL.1997.663916. <https://dl.acm.org/doi/pdf/10.5555/266989.267116> (cited on page 6).
- Holthausen, Simon [2022]. *Svelte: A Beginner's Guide*. SitePoint, 10 Feb 2022. ISBN 1925836487 (cited on page 29).
- Huggins, Jason [2024]. *Selenium*. 05 Sep 2024. <https://selenium.dev/> (cited on page 34).
- Hunter, John D. [2007]. *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering 9.3 (2007), pages 90–95. doi:10.1109/MCSE.2007.55. <https://ieeexplore.ieee.org/document/4160265> (cited on page 3).
- Inselberg, Alfred [2009]. *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*. Springer, 01 Sep 2009. 554 pages. ISBN 0387215077 (cited on pages 11, 16).
- Inselberg, Alfred and Bernard Dimsdale [1990]. *Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry*. Proc. 1<sup>st</sup> IEEE Conference on Visualization (Vis '90) (San Francisco, California, USA). 23 Oct 1990, pages 361–378. doi:10.1109/VISUAL.1990.146402. <https://ifs.tuwien.ac.at/~mlanzenberger/teaching/ps/ws07/stuff/00146402.pdf> (cited on page 11).
- Kandogan, Eser [2001]. *Visualizing Multi-Dimensional Clusters, Trends, and Outliers Using Star Coordinates*. Proc. 7<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001) (San Francisco, California, USA). 26 Aug 2001, pages 107–116. doi:10.1145/502512.502530 (cited on page 5).
- Karpathy, Andrej [2016]. *tsnejs*. 24 Oct 2016. <https://github.com/karpathy/tsnejs> (cited on page 55).
- Khronos Group [2024a]. *WebGL*. 05 Sep 2024. <https://webgl.org/> (cited on pages 30–31).
- Khronos Group [2024b]. *WebGL 1.0*. 19 Apr 2024. <https://registry.khronos.org/webgl/specs/latest/1.0/> (cited on page 31).
- Khronos Group [2024c]. *WebGL 2.0*. 25 Jul 2024. <https://registry.khronos.org/webgl/specs/latest/2.0/> (cited on page 31).
- Kirk, Andy [2019]. *Data Visualisation: A Handbook for Data Driven Design*. 2<sup>nd</sup> Edition. SAGE Publications, 08 Jul 2019. 328 pages. ISBN 1526468921 (cited on page 1).
- Koytek, Philipp [2017]. *MyBrush Github*. 22 Sep 2017. <https://github.com/philippkoytek/mybrush> (cited on page 23).
- Koytek, Philipp, Charles Perin, Jo Vermeulen, Elisabeth André, and Sheelagh Carpendale [2017]. *MyBrush: Brushing and Linking with Personal Agency*. IEEE Transactions on Visualization and Computer Graphics 24.1 (29 Aug 2017), pages 605–615. ISSN 1077-2626. doi:10.1109/TVCG.2017.2743859. [https://openaccess.city.ac.uk/id/eprint/18383/1/2018\\_VIS\\_mybrush.pdf](https://openaccess.city.ac.uk/id/eprint/18383/1/2018_VIS_mybrush.pdf) (cited on pages 23–24).
- Lab, DAISY [2024]. *XmdvTool Home Page*. 05 Sep 2024. <https://davis.wpi.edu/~xmdv/> (cited on page 15).
- Lavrenov, Anton [2024]. *Konva.js*. 05 Sep 2024. <https://konvajs.org/> (cited on page 32).
- Lock, Robin H. [1993]. *1993 New Car Data*. Journal of Statistics Education 1.1 (1993). doi:10.1080/10691898.1993.11910459. <https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/Cars93.html> (cited on page 71).

- Luneckas, Vidmantas [2021]. *svelte-window-system*. 29 Jul 2021. <https://github.com/DonutLaser/svelte-window-system> (cited on page 55).
- Macrofocus [2015]. *InfoScope*. 19 Aug 2015. <https://web.archive.org/web/20160429015130/http://www.macrofocus.com/public/products/infoscope/> (cited on pages 18–19).
- Macrofocus [2024]. *High-D*. 30 Aug 2024. <https://www.high-d.com/> (cited on pages 20–21).
- MacWright, Tom [2021]. *Understanding Point-in-Polygon*. 12 Feb 2021. <https://observablehq.com/@tmcw/understanding-point-in-polygon> (cited on page 51).
- Matthew, David [2021]. *The Canvas API, Part 1: The Background*. 27 Jan 2021. <https://davidmatthew.ie/the-canvas-api-part-1-the-background/> (cited on page 30).
- McInnes, Leland, John Healy, and James Melville [2018]. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. arXiv (09 Feb 2018). ISSN 2331-8422. doi:10.48550/arXiv.1802.03426 (cited on page 10).
- McInnes, Leland, John Healy, Nathaniel Saul, and Lukas Grossberger [2018]. *UMAP: Uniform Manifold Approximation and Projection*. *The Journal of Open Source Software* 3.29 (02 Sep 2018), page 861. doi:10.21105/joss.00861. <https://umap-learn.readthedocs.io/> (cited on page 3).
- McKinney, Wes [2010]. *Data Structures for Statistical Computing in Python*. Proc. 9<sup>th</sup> Python in Science Conference (SciPy 2010) (Austin, Texas, USA). Volume 445. 28 Jun 2010, pages 51–56. doi:10.25080/Majora-92bf1922-00a. <http://conference.scipy.org.s3.amazonaws.com/proceedings/scipy2010/pdfs/mckinney.pdf> (cited on page 3).
- MDN [2024a]. *Offscreen Canvas*. MDN Web Docs, 05 Aug 2024. <https://developer.mozilla.org/en-US/docs/Web/API/OffscreenCanvas> (cited on pages 30–31).
- MDN [2024b]. *XMLSerializer*. MDN Web Docs, 05 Sep 2024. <https://developer.mozilla.org/en-US/docs/Web/API/XMLSerializer> (cited on page 52).
- Meta [2024]. *React*. 05 Sep 2024. <https://react.dev/> (cited on page 29).
- Morrison, Alistair, Greg Ross, and Matthew Chalmers [2003]. *Fast Multidimensional Scaling Through Sampling, Springs and Interpolation*. *Information Visualization* 2.1 (01 Mar 2003), pages 68–77. doi:10.1057/palgrave.ivs.9500040. <https://dcs.gla.ac.uk/~matthew/papers/JInfoVis.pdf> (cited on page 9).
- Murray, Nate, Felipe Coury, Ari Lerner, and Carlos Taborda [2018]. *ng-book: The Complete Guide to Angular 5th Edition*. CreateSpace Independent Publishing Platform, 06 Feb 2018. ISBN 1985170280 (cited on page 28).
- Neuhold, Lukas, Ridvan Aydin, and Georg Regitnig [2020]. *The Radial Projection Explorer*. Project Report. Graz University of Technology, 29 Jun 2020. <https://courses.isds.tugraz.at/ivis/projects/ss2020/ivis-ss2020-g4-project-radial-projection-explorer.pdf> (cited on pages 6–8).
- Nextapps [2023]. *WinBox.js*. 21 Dec 2023. <https://github.com/nextapps-de/winbox> (cited on page 55).
- Nguyen, Quang Vinh [2024]. *TabuVis Home Page*. 05 Sep 2024. [https://staff.cdms.westernsydney.edu.au/~vinh/projects\\_TabuVis.php](https://staff.cdms.westernsydney.edu.au/~vinh/projects_TabuVis.php) (cited on page 21).
- Nguyen, Quang Vinh, Yu Qian, MaoLin Huang, and JiaWan Zhang [2023]. *TabuVis: A Tool for Visual Analytics Multidimensional Datasets*. *Science China Information Sciences* 56.5 (03 May 2023), pages 1–12. ISSN 1869-1919. doi:10.1007/s11432-013-4870-1. [https://researchgate.net/publication/257686743\\_TabuVis\\_A\\_tool\\_for\\_visual\\_analytics\\_multidimensional\\_datasets](https://researchgate.net/publication/257686743_TabuVis_A_tool_for_visual_analytics_multidimensional_datasets) (cited on pages 21–22).
- npm [2024]. *npm*. 05 Sep 2024. <https://npmjs.com/> (cited on pages 40, 81).

- OpenJS [2024a]. *Electron.js*. OpenJS Foundation, 05 Sep 2024. <https://electronjs.org/> (cited on page 37).
- OpenJS [2024b]. *Node.js*. OpenJS Foundation, 04 Sep 2024. <https://nodejs.org/> (cited on page 81).
- Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay [2011]. *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research 12 (2011), pages 2825–2830. <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html> (cited on page 3).
- People+AI Research (PAIR) Initiative [2024]. *umap.js*. 04 Jun 2024. <https://github.com/PAIR-code/umap-js> (cited on page 40).
- Refalo, Olivier [2024]. *svelte-splitpanes*. 28 Jun 2024. <https://github.com/orefalo/svelte-splitpanes> (cited on page 40).
- Romesburg, H. Charles [1984]. *Cluster Analysis for Researchers*. Lifetime Learning Publications, 01 Jan 1984. ISBN 0534032486 (cited on page 14).
- Samariya, Durgesh [2020]. *Premier League Data*. 27 Jul 2020. <https://kaggle.com/datasets/themlphdstudent/premier-league-player-stats-data> (cited on pages 18, 20–22, 72).
- science.ai [2016]. *tsne.js*. 22 Feb 2016. <https://github.com/scienceai/tsne-js/> (cited on page 55).
- Shimrat, Moshe [1962]. *Algorithm 112: Position of Point Relative to Polygon*. Communications of the ACM 5.8 (1962), page 434. doi:10.1145/368637.368653 (cited on page 51).
- Slay Lines [2024]. *Canvas Engines Comparison*. 20 Apr 2024. <https://github.com/slaylines/canvas-engines-comparison> (cited on page 33).
- Tauri [2024a]. *Tauri*. 05 Sep 2024. <https://tauri.app/> (cited on page 37).
- Tauri [2024b]. *Tauri Prerequisites*. 04 Sep 2024. <https://tauri.app/v1/guides/getting-started/prerequisites/> (cited on page 81).
- Themesberg [2024a]. *Flowbite Svelte*. 04 Sep 2024. <https://flowbite-svelte.com/> (cited on page 40).
- Themesberg [2024b]. *flowbite-svelte-icons*. 12 May 2024. <https://github.com/themesberg/flowbite-svelte-icons> (cited on page 41).
- UBS [2012]. *UBS Prices and Earnings 2012*. Union Bank of Switzerland, 14 Sep 2012. <https://ubs.com/global/en/media/display-page-ndp/en-20120914-20120914a.html> (cited on page 19).
- Van der Maaten, Laurens and Geoffrey Hinton [2008]. *Visualizing Data using t-SNE*. Journal of Machine Learning Research 9.11 (Nov 2008), pages 2579–2605. ISSN 1532-4435. <https://jmlr.csail.mit.edu/papers/v9/vandermaaten08a.html> (cited on page 9).
- W3C [2001]. *SVG*. World Wide Web Consortium, 04 Sep 2001. <https://w3.org/Graphics/SVG/> (cited on page 30).
- W3C [2011]. *SVG Document Object Model (DOM)*. World Wide Web Consortium, 16 Aug 2011. <https://w3.org/TR/SVG11/svgdom.html> (cited on page 31).
- W3C [2015]. *HTML Canvas 2D Context*. W3C Recommendation. World Wide Web Consortium, 19 Nov 2015. <https://w3.org/TR/2015/REC-2dcontext-20151119/> (cited on page 30).
- W3C [2024a]. *SVG Defs*. World Wide Web Consortium, 05 Sep 2024. <https://w3.org/TR/SVG2/struct.html#DefsElement> (cited on page 53).

- W3C [2024b]. *SVG Line*. World Wide Web Consortium, 05 Sep 2024. <https://w3.org/TR/2000/CR-SVG-2-0001102/shapes.html#LineElement> (cited on page 53).
- W3C [2024c]. *SVG Use*. World Wide Web Consortium, 05 Sep 2024. <https://w3.org/TR/SVG2/struct.html#UseElement> (cited on page 53).
- W3C [2024d]. *WebGPU*. World Wide Web Consortium, 05 Sep 2024. <https://w3.org/TR/webgpu/> (cited on pages 30–31).
- Ward, Matthew O. [1994]. *Xmdvtool: Integrating Multiple Methods for Visualizing Multivariate Data*. Proc. 5<sup>th</sup> IEEE Conference on Visualization (Vis '94) (Washington, DC, USA). 1994, pages 326–333. doi:10.1109/VISUAL.1994.346302. <https://davis.wpi.edu/~xmdv/docs/vis94.pdf> (cited on pages 15–16).
- Ware, Colin [2021]. *Visual Thinking for Information Design*. 2<sup>nd</sup> Edition. Morgan Kaufmann, 14 Jul 2021. 224 pages. ISBN 0128235675 (cited on page 1).
- Wattenberg, Martin, Fernanda Viégas, and Ian Johnson [2016]. *How to Use t-SNE Effectively*. Distill (18 Oct 2016). doi:10.23915/distill.00002. <http://distill.pub/2016/misread-tsne> (cited on page 9).
- Weaver, Chris [2020]. *Improvise*. 28 Oct 2020. <https://cs.ou.edu/~weaver/improvise/> (cited on pages 22–23).
- Wegman, Edward J. [1990]. *Hyperdimensional Data Analysis Using Parallel Coordinates*. Journal of the American Statistical Association 85.411 (1990), pages 664–675. doi:10.1080/01621459.1990.10474926. <https://jstor.org/stable/2290001> (cited on page 13).
- Weisstein, Eric W. [2002]. *Line-Line Intersection*. 2002. <https://mathworld.wolfram.com/Line-LineIntersection.html> (cited on page 51).
- Whitney, Justin [2013]. *Surviving the Zombie Apocalypse: Manipulating SVG with JavaScript*. 24 Jun 2013. <https://sitepoint.com/surviving-the-zombie-apocalypse-manipulating-svg-with-javascript/> (cited on pages 30–31).
- Yi, Ji Soo, Rachel Melton, John Stasko, and Julie A. Jacko [2005]. *Dust & Magnet: Multivariate Information Visualization Using a Magnet Metaphor*. Information Visualization 4.4 (11 Apr 2005), pages 239–256. doi:10.1057/palgrave.ivs.9500099. <https://faculty.cc.gatech.edu/~stasko/papers/iv05-dnm.pdf> (cited on page 7).
- You, Evan [2024a]. *Vite*. 04 Sep 2024. <https://vitejs.dev/> (cited on page 40).
- You, Evan [2024b]. *Vue*. 05 Sep 2024. <https://vuejs.org/> (cited on page 29).
- Zakodium [2023]. *ml-pca*. 23 Nov 2023. <https://github.com/mljs/pca> (cited on page 40).
- Zhao, Kaiyu [2021]. *XmdvTool Github*. 24 Sep 2021. <https://github.com/kaiyuzhao/XmdvTool> (cited on page 15).