FH University of Applied Sciences
TECHNIKUM WIEN

MASTER THESIS

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Game Engineering and Simulation Technology

A pixel-based approach for evaluating the visual scalability of the v-plot matrix

By: Nadine Buesch, BSc

Student Number: 1910585005

Supervisors: Dipl.-Ing. Dr. Gerd Hesina
Dipl.-Ing. Dr. Johanna Schmidt

Wien, May 5, 2022

Declaration

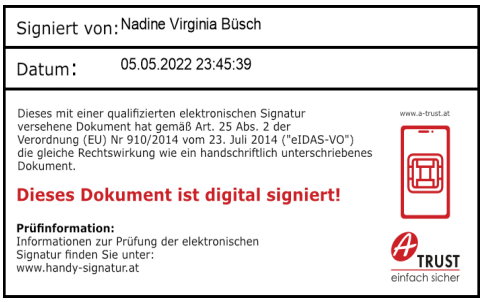
“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Wien, May 5, 2022

Signature



Kurzfassung

Die Datenvisualisierung ist eine gängige Methode, welche bei der Analyse von Daten zum Einsatz kommt. Sie findet in diversen Bereichen Anwendung, wie etwa dem Finanzsektor, in der Produktion und in der Forschung. Im Forschungsbereich der Datenvisualisierung werden neue Methoden zur Visualisierung von Daten entwickelt. Vergleichsweise wenig wird an Evaluierungsmethoden für Datenvisualisierungen geforscht. Da es zwar verschiedene Ansätze für die Evaluierung von Visualisierungen gibt, aber keinen Standard, entwickelten wir eine neue Evaluierungsmethode, welche auf der Basis von Pixeln arbeitet. Ziel dieser Methode war es zu evaluieren, wie gut eine Visualisierung skaliert, in Abhängigkeit von dem verfügbaren Platz am Bildschirm. Für diese Methode berechneten wir drei verschiedene Verhältnisse, welche sich an bestehenden Konzepten orientierten (data-ink ratio, foreground-background ratio, discriminability ratio). Diese Verhältnisse wurden über variierende Bildschirmgrößen (in Pixel) beobachtet, um eine Aussage über die Skalierbarkeit einer Visualisierung zu treffen. Um diese Evaluierungsmethode zu testen, wurden V-Plot Matrizen in die Software Visplore by VRVis integriert. V-Plots sind eine Methode um die Verteilung von Daten zu visualisieren, welche erst kürzlich entwickelt wurde. Diese Implementierung wurde dazu genutzt ein Test-Datenset, bestehend aus 375 Bildern von unterschiedlich konfigurierten V-plot Matrizen, zu exportieren. Die Berechnung der drei Verhältnisse wurde als Python Script realisiert. Der Trend der berechneten Verhältnisse, sowie diverse Hypothesentests wurden analysiert und ausgewertet. Diese zeigten, dass einzelne V-Plots bereits auf einer Fläche von 100x160px effektiv darstellbar waren. V-Plot Matrizen mit einer Dimensionalität von 6 konnten ab einer Größe von 180x320px effektiv dargestellt werden.

Schlagworte: Datenvisualisierung, Visualisierung Evaluierung, V-Plots

Abstract

Data visualization has become state of the art when analyzing data in various application domains like production, finance and science, among others. Therefore, research in the field of data visualization is concerned with the development of novel visualization techniques. Comparatively little research is dedicated to evaluating data visualizations. We looked into existing techniques to evaluate data visualizations on their scalability. Finding, that there is no state of the art method, we propose our own pixel-based evaluation technique. Its goal is to evaluate the scalability of a data visualization with respect to the available screen-space. It works by calculating three different ratios that were based on existing concepts (data-ink ratio, foreground-background ratio and the discriminability ratio). The changes of these ratios are then observed over different resolutions to make a statement about the scalability of the visualization. To test our proposed evaluation method, we integrated a v-plot matrix visualization into the software Visplore by VRVis. V-plots are a novel data visualization technique to visualize distribution data and were introduced recently. We used this implementation to export our test data set, that consisted of 375 images containing various differently configured v-plot matrices. The calculation of the proposed ratios was then realized as Python script and applied to the test set images. By observing the resulted trends of the ratios and through conducting hypothesis tests, we found that the ratios indicated that v-plot matrices up to dimensionality 6 could effectively be shown on a resolution of 180x320px and larger. Whereas single v-plots could be displayed effectively on a resolution of 100x160px and above.

Keywords: Data visualization, Visualization evaluation, V-Plots

Acknowledgements

First of all I want to thank Hannes, who always encourages me to do what I dream of, no matter how daunting it may seem. Thank you, for supporting me in my pursuits. I also want to express my gratitude towards Doris, Margarete, Nicole and Patrick for their continual positive encouragement and dedicated support.
Thank you, Marko, for elating and motivating me, when I needed it the most.
I also want to thank my supervisors, Gerd Hesina and Johanna Schmidt. Thank you, Gerd, for giving me the chance to work at VRVis and to write my master thesis there. Thank you, Johanna, for giving me invaluable advice and being such a dedicated supervisor with lots of ideas. A big thank-you also goes to my colleagues at VRVis, for sharing your knowledge with me, it was a great experience working with you.
This work was enabled by the Competence Centre VRVis. VRVis is funded by BMK, BMDW, Styria, SFG, Tyrol and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (879730) which is managed by FFG.

Contents

1 Introduction

Data analysis [68] describes all the steps that are needed to transform data to gain useful insight from it [68]. Common tasks of data scientists [40] include the validation of a specified model or hypothesis based on the gathered information as well as identifying recurring patterns in a dataset [40]. This type of data analysis is also called *confirmatory data analysis* (CDA). Another application for data analysis is the so-called *exploratory data analysis* (EDA). In this case, no pre-defined hypothesis is available that needs to be validated, but the data scientists are free to explore seemingly interesting aspects of the data and further inspect those in detail [73]. Behrens [6] describes CDA and EDA as complementary methods to analyse data. As often, in the early stages of data analysis, there are usually many questions rather than a single one, that need to be answered to understand the data [3]. Therefore, the data is inspected through EDA first to refine stated hypotheses and corroborate the mental model of the data [16]. Subsequently, CDA is applied to either confirm or reject the proposed hypothesis.
Data analysis has become state of the art in many application domains and industries like power generation [47], production [49], finance [83] and in various scientific fields [66]. With the growing capability to gather information at a faster rate than the data can be analyzed, an information overload is created [65]. This circumstance can be observed to be increasing over the recent years, and was made known by the term *Big Data*. In conjunction with the appearance of Big Data, the term *Datafication* occurred, describing the processing of various phenomena (be it aspects of the human life or measurements taken during industrial processes) into data [50]. With this, the importance to automate at least parts of the data analysis process increases. In cases where no automation can be applied, it is important to utilize techniques that turn the information into a form that is easier to understand for data analysts. For this purpose diverse software tools are being developed and maintained to assist with this tasks. An introduction to some of these tools were given by Ali et al. [2] and Caldarola and Rinaldi [14]. Where Ali et al. [2] give a more in-depth description of various software, Caldarola and Rinaldi cover a bigger variety of tools.
Data visualization [78] is a prominent way to make loads of information more comprehensible. The data visualization process describes all the steps necessary for creating graphical representations that encode an underlying dataset. These graphical representations are usually called plots [44].

1.1 Data visualization

The term data visualization is commonly understood as *"a graphical representation of data or concepts"* [78]. It was originally used to describe the process of forming a mental image. Data visualization is influenced by the development of many research fields, including statistics, psychology and computer science [5]. The three main goals of data visualizations are presentation, confirmatory analysis and exploratory analysis. For presentation, the aim is to pick a visual representation that allows for a better communication of the underlying information. As emphasized by Heer et al. [27], the creation of a data visualization for the purpose of presentation depends strongly on the end-user. This is not only, but mostly, due to the fact that data visualization as means to present information is used not only in specialized fields, but can also be seen being used increasingly often in everyday media as a part of *infographics* [70]. Confirmatory data analysis (CDA) is concerned with either proving or disproving a hypothesis. Exploratory data analysis (EDA) describes the undirected search for interesting aspects of the data.
With the "Milestone Project", Friendly [22] made an effort to document the important events in the history of data visualization from its dated first occurrences back around the 10^{th} century to modern times. The history of data visualization is divided into eight epochs. The earliest forms of data visualization found were geometric diagrams, displaying positions of stars and celestial bodies, but also maps to support navigation when exploring. Along with important advances in the field of mathematics and statistics, also visualization techniques were further developed and improved. Statistical graphics experienced a great expansion in the first half of the 19^{th} century. Data visualization techniques that are widely known nowadays were invented back then. Among those visualizations are bar charts, histograms, pie charts, line graphs, time series plots, and many more. Soon after that, attempts were made to display multivariate data. Few visualization innovations were made following that time, until the middle of the 20^{th} century. With further research in computer graphics and developments in data analysis, new possibilities for data visualizations arose as a result. Since around 1975, data visualization has become a research field that spans multiple research areas. Nowadays, computers and specialized software tools support data scientists in the visualization process, so they can better focus on the task of analysing the data without having to visualize it manually. Not only is visualization software developed in academic fields, but their use is also becoming more and more established in the fields of business intelligence [20].

Fisher and Meyer [21] propose that *data*, *tasks* and *stakeholders* are the main factors influencing which visualization to choose that will be effective for the end users. They further accent that not every task needs to be supported by a visualization. Tasks that can be put into a precise question can often be answered by computational means. More vague questions that leave room for interpretation with their answers tend to really profit from a visual representation. Visualizations are often used in combination in so-called *dashboards* which enables performing

multiple analysis tasks. Pappas and Whitman [59] point out that the assembly of a dashboard is a complex task where various factors need to be taken into consideration. This includes the capabilities and background of the end user, the task and the data at hand. Creating effective data visualizations is not an easy task but there are quite a few resources concerned with guidelines for the creation of expressive visualizations, like the work by Midway [55]. Among many others, Weissgerber et al. [79] and Cleveland [17] specifically looked into the generation of effective visualizations used in scientific papers, as these seem to contain quite an amount of visualizations that would benefit from improvements.
In this thesis the focus will be on the visualizations specifically designed for data distributions.

1.2 Visual analytics

Visual analytics describes the usage of data visualizations and automation to support humans in analyzing data, as opposed to purely visual approaches and to *Data Mining*, which describes the automated process to extract interesting and useful portions of data [13]. The visual analytics approach was formed because the purely visual as well as the purely automated ways have their limitations [42]. The limitations of Data Mining methods to analyze data lie in the fact that no expert knowledge is involved in the process, often leading to unsatisfying solutions to a problem. The possibility to analyze data by means of a visualization is strongly dependent on the dataset size. This is due to the fact that with a larger dataset size, computation times and storage space increase. The interactive visualizations of massive datasets is an especially difficult challenge, as the visualization has to stay responsive to user input while performing computations. Along with the development of data visualization techniques, the discipline of visual analytics developed. As formulated by Thomas and Cook [72]: *"Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces."*. As many other disciplines that involve the visualization of data or information, visual analytics makes use of the human visual system that provides the highest bandwidth from a computer to the human, in comparison to the other cognitive systems [78]. As opposed to information visualization, which is mainly concerned with the visual representation of abstract information, the term visual analytics combines decision-making, visualization, human factors, and data analysis [43]. As stated by Keim et al. [43], this usually involves utilizing computers and automated algorithms to tackle a data analysis task. The main purpose of visual analytics is to gain insight on the data from a visual representation. Andrienko et al. [4] describe visual analytics as a process to gain knowledge about or understand a so-called *subject*, which describes a certain thing or phenomenon and in some cases also the data itself. The analysis of this subject then includes not only understanding but can also involve creating a computer model of it, which can in turn be the subject to the analysis, as knowledge is to be gained about the model. This shows that performing a visual analysis, the subject of analysis may vary in the process. Usually some interaction with the visualization is also possible. As stated by Keim et al. [44], the main tasks of the visual analytics process are information gathering, knowledge representation as well as

interaction and decision making. Data analysis is driven by mathematics and statistics. It relies on the human capabilities to perceive the information and make relations and conclusions based on them. The process of visual analytics includes data analysis, visualization and human factors. Through these diverse factors, the visual analytics research field also profits from research in the fields of data management, knowledge representation, discovery and statistical analytics [44].

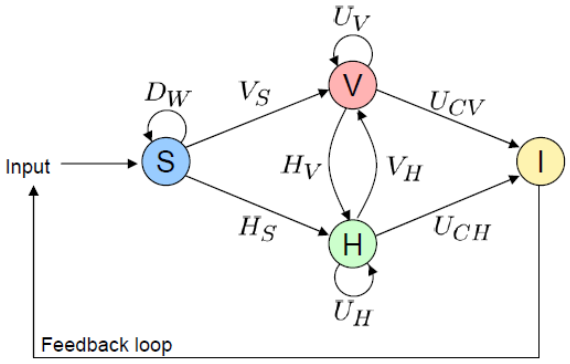


Figure 1: The visual analytics process described as transformation. The transformation function takes the data as input and yields insight *I*. *S* describes the subset of data gained through the preprocessing *Dw*. Based on that data either a hypothesis *H* is formulated and then visualized *(V)* or vice versa. User interactions can be performed on hypotheses and visualizations *U*. The image was taken from [44].

Keim et al. [44] introduce a formal description of the visual analytics process. Further they suggest to see the visual analytics process as a transformation that maps the source data set to the insight gained through the visual analysis process, as depicted in Figure 1. The traversed steps include preprocessing of the data. It can be achieved through data cleansing, selection or integration. Data preprocessing can also mean to visualize the data in some form to reveal patterns. The next step of the process is the visualization of either the data or a hypothesis, which would include the step of generating this hypothesis first. A very integral part of the process is the user interaction. The user interaction can either affect only visualizations or only hypotheses. This visual analytics process is usually iterative, meaning the user can adapt input parameters repeatedly and apply the transformation in order to gain the most insight out of it.

1.3 Visual comparison

A comparison task can be executed on data from different sources or on data from one source that contains categorical differentiation. The act of comparison can also be understood as *"the task of formulating a relationship that holds for particular subsets of the data."* [73]. This com-

parison can either be done on the data level or based on an image containing a visualization representing that data [58]. In this chapter the latter will be discussed.

Visual comparison, also referred to as comparative visualization, is another important area where visual analytics is applied. Its purpose is to compare multiple datasets or subsets of one dataset with each other. Various methods can assist the comparison, so that the viewer does not have to rely on their eyes and brain only. Although a statement can be made through mere visual observation, computations expressing the similarity of the compared data can help make a statement evident.
Tominksi et al. [73] analyzed what techniques, which are inspired by natural behaviour, can assist the comparison process. They observed how people tackle the task of comparing printed data visualizations on paper. After picking some seemingly interesting sheets to be studied in detail, those pages were compared. Comparison usually happened by laying them out side by side, overlaying sheets and holding them against a light source to seemingly merge the data or overlaying sheets and folding them back and forth to discover differences in the data. For the purpose of their work, they present a concept that describes the usual comparison process. It consists of three phases, which are commonly traversed when comparing:

1. **Selecting** which pieces of information of the data should be compared.
2. **Arranging** the selection in a way, that the comparison is made easier.
3. Actually **comparing** the selected and arranged data.

As Tominski et al. [73] stated, to form a mental model of the data, supplying a way to interact with the visualization is important. Such interaction techniques include: **Navigation techniques** like panning and zooming to navigate and focus on certain areas of the data or concentrate on the overview. **Brushing and linking** to select certain areas of the dataset that are of interest. **Layout manipulation** might ease the comparison for example through positioning visualizations, that are to be compared close to each other.

Gleicher et al. [24] proposed a taxonomy for comparison designs. It divides them into three categories: **juxtaposition**, **superposition** and **explicit encoding**.
Juxtaposition, or separation, displays the information separately and independently. Juxtaposition designs can be applied to all visual representation techniques. An example would be two plots next to each other. It can also mean that objects are displayed after one another, for example blending between two plots. These separate representation means that the user has to memorize parts of the information, while switching between the plots to make a comparison. It can be helpful to place the plots close to each other. Comparisons based on juxtaposed designs rely heavily on the users' memory. With juxtaposed designs it is also easy to automate the search for difference in the visualizations.
Superposition, or overlay, displays the information in the same space and at the same time.

For example, one plot is drawn transparent on top of the other or even occluding it. This requires the information to inhabit the same coordinate system. To transform the information into the same space, computation is usually used. It also means that this design does not scale well with lots of dense data, causing unwanted occlusions and clutter.

Explicit encoding encodes relationships of the datasets directly and visualizes them. Therefore, relations between the elements have to be established. This requires the visualization designer to have an idea of what relationship the users are looking for, and a way to compute the encoding. An example would be the calculated difference per point between two datasets, visualized in a plot. Comparisons with explicit encoding designs don't rely on the users memory but on the computation of the encoding.

These three categories can be seen as the building blocks for comparison designs, but also allow for hybrid combinations of them. Explicit encoding designs can also be thought of as replacing objects with newly computed objects, representing the relationships. Those newly computed objects may have the same type as the original data or another form. Explicit encoding designs are usually used in combination with other forms of visualizations to counteract decontextualization. For that reason they are also often used in a hybrid approach with either juxtaposition or superposition designs. Another tool to support comparison is animation, where elements are displayed one after the other. It can be seen as a juxtaposition in time. Yet another animation technique, but categorized as superposition, is the "blink" technique. In this case, the elements are placed in the same place, but it can be alternated which one is seen at any given moment.

This taxonomy can assist as a guideline when tackling the development for a new visual comparison design. Gleicher et al. [24] supply even more detailed strategies when creating new comparison designs. When designing new visual comparison techniques, it should be remembered that comparison consists of multiple tasks performed by the user.

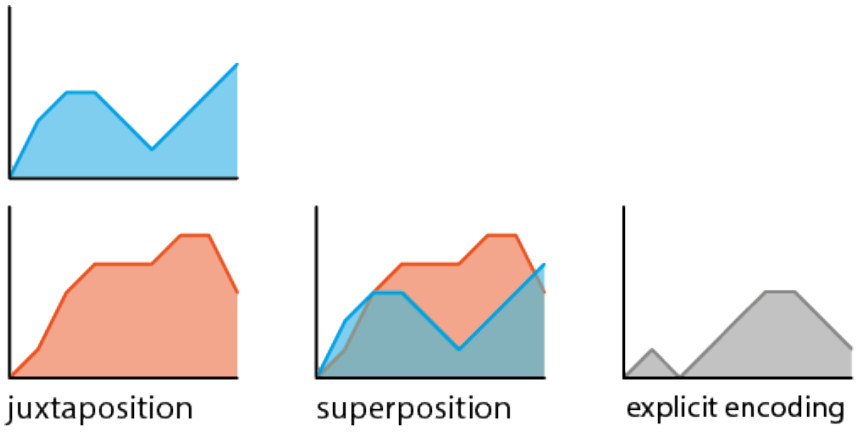


Figure 2: Juxtaposition, superposition and explicit encoding exemplified through simple filled broken line graphs. Juxtaposition shows two different datasets one below the other for comparison. In a superposition arrangement, the different graphs are overlaid and blended with each other. Explicit encoding is created through visualizing only the differences between the two graphs in the shape of a new one.

Interaction techniques like mentioned earlier are invaluable in dedicated modern software tools to support visual comparison. These tools usually include brushing and linking to highlight the coherence between elements. It can also be very helpful for the user to be able to rearrange plots for easier comparison between them. Analytical tools as well as statistical tools can also improve the readability of visual designs and help to link more complex information.

2 Visualization techniques

As mentioned earlier, depending on the type of data, different visualization techniques are appropriate for its representation. In the context of this thesis, we focus on visualization techniques for data distributions. In statistics, a data distribution refers to a function, which describes which possible values occur in a dataset and in which quantity [1]. Usually the quantity is either given as a number denoting the counted occurrences of a value, or it is given relative to the total number of values in the dataset, as percentage. A data distribution function is often created based on existing values. This function can then be used to make a prediction for the distribution of values that are not available in the dataset [36]. The prediction can either be an exact value, or be a statement that the value in question is higher or lower than a specific figure. For many distribution functions the average value (e.g., the mean or median) and a measurement for the spread of the data (e.g., standard deviation) is used to predict the distribution. The normal distribution for example, uses the mean and standard deviation for calculations.
Visualizing data distribution allows to get a quick overview and perform an in-depth analysis. Visualization techniques for data distributions usually use two dimensions. It is most common to apply the possible range of values that the distribution function covers to one axis. The other axis represents how often those values actually appear in the data. Depending on the sampling of the dataset at hand, the axis denoting the value range can either be discrete or continuous [41]. In the discrete case there are a number of countable, limited values the data can take on. Continuous ranges are used whenever an uncountable set of values can be taken on. This goes for for continuous sampling over time and whenever the sampled value may lie in a range between two values. Note that the concept for continuous sampling is more theoretical, because real continuous sampling is impossible due to the fact that the sampling rate cannot be continuous. For example if samples are taken over time, they are taken in a specified interval (eg. every second, ever millisecond). Also, the value a sample can take on is limited by the precision that it is represented with.
Visualization techniques for data distributions are often used to represent the distribution function approximating the dataset or its distribution as a whole, but it is also common to represent the underlying data through displaying aggregated statistics.

2.1 Data distribution visualization techniques

Over time, many visualization techniques for data distributions have been established. They all serve the purpose to either visualize a distribution, a derived distribution function or aggregated statistics. In this Chapter, common visualization techniques that will be of importance in the following chapters are introduced.

2.1.1 Histograms

A widely known visualization technique for data distribution is the histogram as depicted in Figure 3 on the left side. Histograms are based on bar charts [7]. Bar charts usually encode the range of values on the x-axis, and the count of their appearances in the underlying data on the y-axis. In a bar chart, to each value on the x-axis, a bar is drawn, whose height represents the frequency of the underlying dataset. In bar charts, one bar represents the frequency of a single related value. The bars in histograms on the other hand, encode the frequency of multiple values in a specified range. These value ranges are called bins. Binning is usually performed uniformly. This means that the whole range of available values is divided by a desired factor. The resulting intervals are then used as bins. The bin width is flexible but should stay consistent over all bins. A small bin width might lead to very peaky histograms and obscure the trend of the underlying data. Choosing a very wide bin width might lead to losing interesting details of the dataset. Sahann et al. [67] recently surveyed how many bins are actually needed for users to perceive the distribution of the underlying data. They showed that a higher bin count generally leads to a better estimation, where about 20 bins seemed to be the threshold where no measurable improvement could be found in the conducted user study. When working with discrete data, one bin may correspond to a single value of the data. When working with continuous data, binning must be applied. Variations of the histogram are mirrored histograms[1] and stacked histograms[2]. Histograms and their variations are especially useful for local and global analysis tasks, but are of limited use when performing aggregated analysis tasks [11].

2.1.2 Density plots

Similar-looking to histograms are density plots [80]. An example for a density plot in comparison to a histogram is depicted in Figure 3. Density plots are two-dimensional graphs, where one axis encodes the possible value range and the other axis counts the frequency of how often this value appears in the data. The points used for drawing are not sampled directly from the dataset. Usually an estimation procedure is applied instead to create a curve that meaningfully represents the data. Common estimation techniques are the *kernel density estimation* (KDE) and shape-based estimation [69]. Both techniques supply tweakable parameters to allow for

[1] https://r-graph-gallery.com/190-mirrored-histogram.html
[2] https://chartio.com/learn/charts/stacked-bar-chart-complete-guide/

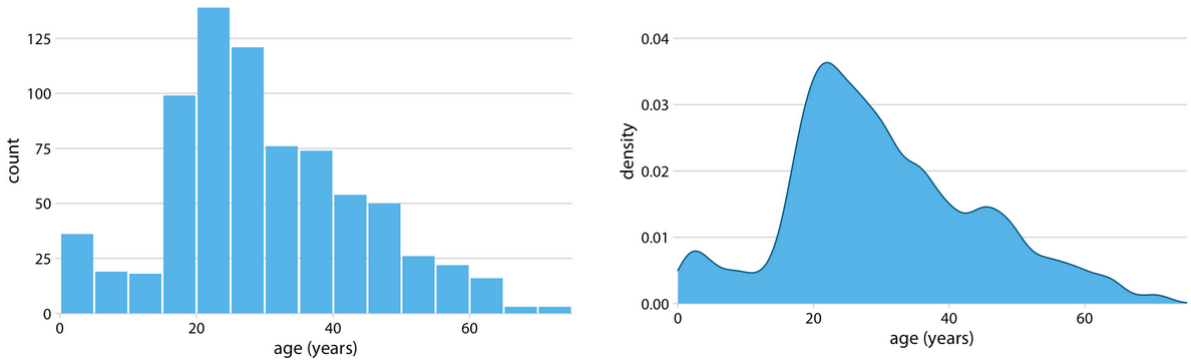


Figure 3: The age distribution of titanic passengers visualized. **Left:** Histogram visualization. **Right:** The same underlying dataset visualized as density plot. The used function for density estimation was the kernel density estimation with a Gaussian kernel and a bandwidth of 2. Images taken from [80].

adjustment of the curve. Like the bin width in histograms, those parameters have a direct effect on the visualization and can lead to overly peaky curves, or to losing important detail through over-smoothing. One pitfall of density plots that should mentioned is that they might appear to represent data, where actually no value exists. This phenomenon usually happens at the tails of the curve. Based on the points gained from the estimation procedure, a continuous line is drawn. If desired, the area below the line may be drawn filled. Due to the fact that histograms can easily be drawn by hand, they have been the preferred method to visualize data distributions for quite a while. With the possibilities of computational powers nowadays, density plots supersede the traditional histogram as visualization more and more. Density plots also work well when visualizing multiple categories overlaid in one graph as depicted in Figure 4.

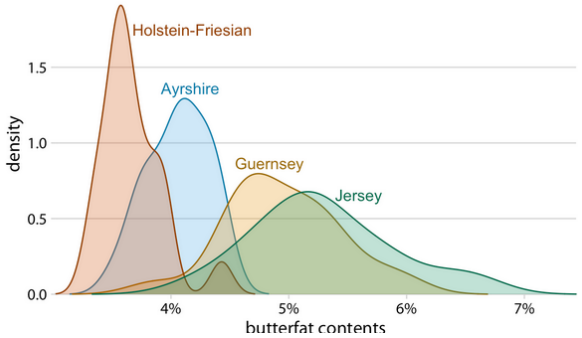


Figure 4: Overlaid density plots visualizing the butter fat contents for milk of different cow breeds, categories encoded as colors. Image taken from [80].

2.1.3 Box plots

A widely known and used visualization technique concerned with the display of aggregated statistics is the box plot [82]. Box plots have become the standard technique, whenever the requirement is to show the minimum, maximum, upper and lower quartiles and the median of a dataset [61]. Box plots along with their variations are often used to compare more than one dataset. The comparison of multiple box plots is quite straight forward, as only the five shown characteristics have to be checked against each other. Box plots are also very space-efficient representations, as their horizontal space used is not directly linked to the underlying data set. This allows for the width of the boxes to be scaled as required. Due to their simplicity, many different visual versions of the box plot exist [45]. Its simplicity also leaves room to expand the type of information visualized in a box plot, eg. by additionally displaying the data density as scatter-plot, as shown in Figure 5.

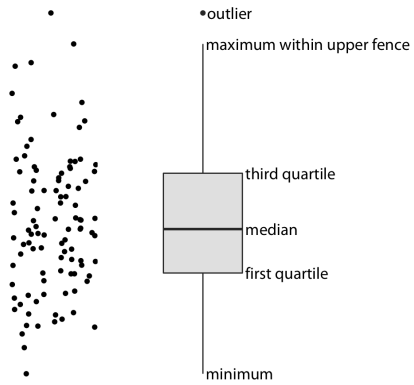


Figure 5: Left: Data visualized as points of a scatter-plot. Right: The same underlying data represented as box plot. Further the anatomy of a box plot is exemplified. Image taken from [81].

Due to its focus on only five characteristics, many other important attributes of the data stay hidden in box plots. To enhance their utility, they can be expanded by the visualization of the underlying data density in the shape of a density plot. The samples taken for the density can be varied in their intervals. One way is to only sample at the median and the quartiles, which creates a Histplot [7]. The Vaseplot on the contrary also uses sample points between the median and the quartiles [7].
Violin plots are another common extended version of the box plot [30]. In addition to showing statistic measures they also visualize the underlying data density as density plot. The rectangle between the quartiles in the box plot is colored solid black and the median is indicated through a small white circle with the goal to make understanding and comparing multiple violin plots easier. There are many more variations of the box plot. Further variations worth mentioning are bean plots [39], error bars and the gradient plot [11]. In regards to the user study by Blumenschein et al. [11], the box plot and its possible variations perform well when used for aggregated and global analysis tasks [11].

2.2 Hybrid charts

The visualization techniques discussed in Chapter 2.1 all focus on the representation of only one aspect of the data and/or were specifically designed to perform certain tasks. Therefore, attempts were made to develop a visualization for data distributions that is able to represent multiple aspects of a dataset and/or to perform multiple tasks with the aid of one single plot. Such combinations of visualizations are also referred to as so-called *hybrid charts*. Hybrid charts are usually achieved by the combination of two or more visualization techniques.
De Oliveira et al. [18] introduced a hybrid chart combining parallel coordinates and multidimensional projection. Another approach to a hybrid visualization technique was introduced by Potter et al. [62] in the form their hybrid summary plot, which is an extended box plot with the ability to additionally visualize "descriptive statistics to concisely present data with uncertainty information".
In the context of this thesis the focus lies on a quite recently introduced hybrid chart for the representation of data distributions, the v-plot.

2.2.1 V-plots

Another type of hybrid chart was developed for the comparison of data distributions and is called v-plot. Blumenschein et al. [11] introduced v-plots in 2020 along with them the v-plot designer[3], an online tool to help find a fitting v-plot configuration for a given analysis task. There are quite a few specialized visualization techniques to support a single specific analysis task, but to support more than one task simultaneously, they usually must be used in combination. V-plots were designed to simultaneously visualize information to handle multiple distribution comparison tasks. V-plots combine all three categories of the taxonomy of Gleicher et al. [24]: Juxtaposition is achieved by viewing two mirrored plots next to each other. The overlay of the various visualizations is categorized as superposition. And with the statistical measures and direct difference encodings, the requirements to qualify as explicit encoding are fulfilled. This allows for making conclusions about the data set on global, local and aggregated scale concurrently from just one plot.
The "V" in the v-plot originates from the roman number five, which is the maximum number of layers that can be used in a v-plot. Those layers display a variety of data distribution visualization techniques and labels. V-lots make use of the fact that all visualization techniques they support can be drawn mirrored. This immediately enables comparing the data distributions of two datasets or categories. The default v-plot consists of the following layers, which are exemplified in Figure 6:

- **Layer 1:** The base layer is a mirrored bar chart, which visualizes the distribution frequency.

[3] https://v-plot.dbvis.de/!/plot

- **Layer 2:** On top of it is a density distribution to represent distribution peaks properly.
- **Layer 3:** The next layer is either a difference shape or histogram. This encodes the absolute difference between the two datasets visualized by the v-plot.
- **Layer 4:** The fourth layer allows for an overlay containing a visualization for statistic measures, enabling viewing explicit encoding. An example for this layer would be showing the median and standard deviation.
- **Layer 5:** Atop those four layers, labels (e.g., a title, a grid, annotations) can be added to round up the v-plot and further enhance its comprehensibility.

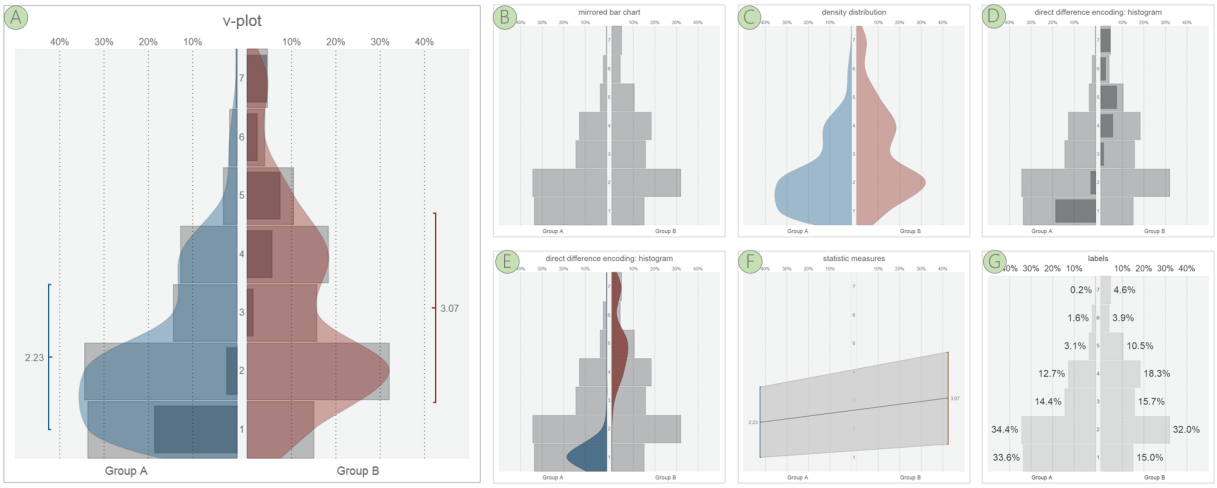


Figure 6: **(A)** V-plot with all its five layers visible. **(B)** The base layer containing a mirrored histogram. **(C)** The second layer contains the density distribution. **(D) + (E)** Contains a direct difference encoding as either histogram or shape. **(F)** The fourth layer shows statistic measures. **(G)** The top layer displays the labels of the plot. Image taken from [10].

The qualitative expert user study conducted by Blumenschein et al. [11] suggested that the v-plots are intuitive and efficient when it comes to comparing data distributions.

2.2.2 The v-plot matrix

V-plots are useful when comparing two distributions at once. An approach to compare more than two data distributions is the v-plot matrix. Displaying multiple v-plots in a matrix arrangement simultaneously allows for the comparison between even more data channels than within a single v-plot. The proposed matrix is of a square shape. Each column and each row contains a data channel or category. This results in one v-plot for every combination of data channels or categories. A challenge concerning the v-plot matrix is to keep it readable and to avoid clutter and occlusion. To maintain readability, Blumenschein et al. [11] suggest supplying the possibility to sort the v-plots in the matrix automatically, e.g., by similarity. This way, v-plots in which the two represented distributions are very similar, are placed somewhere in the matrix, where

other v-plots have a likewise similarity (e.g., upper left corner of the matrix contains v-plots with high similarity and the lower left corner has v-plots with low similarity). As further countermeasures, the layers which are visible in the v-plots are reduced. An example for this would be to only show the direct difference encodings on both sides. To make the best use of the available space and prevent repetition, the matrix is split diagonally. The v-plots that would lie on this diagonal contain only matches of the same data channel or category, meaning they would visualize two identical sides. The decision was made to not display them to prevent drawing unnecessary attention. For further sufficient use of the available screen space and to avoid repetition, the visible layers of the v-plots above and below the diagonal differ. This way, there are always two alternate representations of the same juxtaposed distributions, without having to worry about showing too much information in just one v-plot and risking clutter and occlusion. The matrix created by the v-plot designer is depicted in Figure 7.

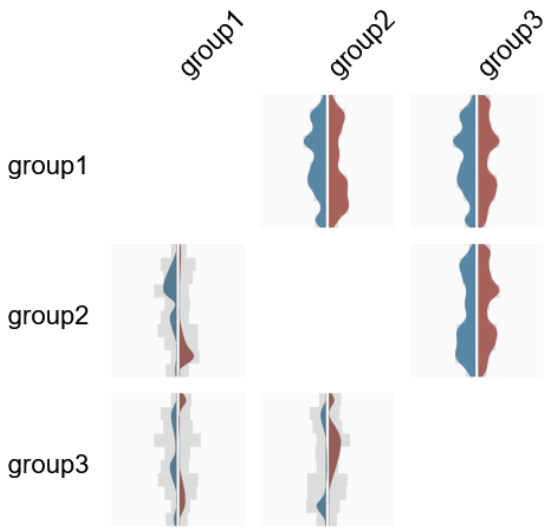


Figure 7: The v-plot matrix visualized by the online v-plot designer. It visualizes a small dataset made available by the v-plot designer to present the matrix and to show what underling data structure is expected by the v-plot matrix to construct it. In the upper right, datasets are visualized as density plots, whereas the same dataset pairings are displayed as direct difference encoding with histogram in the lower left diagonal. Image taken from [9].

2.2.3 V-plot example use cases

To give a better understanding of how to practically work with v-plots, exemplary use cases for analysis tasks are elaborated in the following two subsections. Further use cases that utilize v-plots and the v-plot matrix can also be found in the related work by Blumenschein et al. [11]. In order to present meaningful use cases, we found a dataset that represents realistic conditions without privacy restrictions and open for free use. In our case, the chosen dataset contains anonymized recordings of various measurements that were made in the process of

power generation.

About the dataset

The dataset used to create the v-plot images for the evaluation is the *Photovoltaic Solar Panel Energy Generation data* (PV) dataset [60] with some modifications. Its licensing along with the performed modifications can be found in Appendix A.1. The dataset contains data channels relevant for the production of photovoltaic energy including environmental influences like voltage, current and weather data. The dataset contains measurements in 10 minute intervals that were collected covering 480 days, resulting in over 171 million individual values.

Example 1: Solar radiation

Since the PV dataset contains data relevant for power production using photovoltaic installations, we decided to look at a use case also relevant for this application. This is why we will look at the recorded solar radiation of the PV dataset as part of this first use case demonstration. Solar radiation can be measured in different ways. Sicne we do not know the exact way it was measured, we assume that the recorded data includes direct i.e., directly hitting sunlight and indirect i.e., sunlight bounced and reflected from other different sources) solar radiation. Solar radiation is usually measured using kW/m^2 and results in values between $0kW/m^2$ and $1kW/m^2$. By looking at the available data, we assume that in this case the solar radiation is given in W/m^2, since the measured values lie between $0W/m^2$ and approximately $1000W/m^2$. Based on the recorded data we want to find out at which locations the solar radiation is noticeable higher or lower, telling us which solar plants receive more or less sunlight. We generated a v-plot matrix of dimension 5, displaying one v-plot for all combinations of locations that have solar radiation data available. To get an overview, we visualized the data frequency as histogram with 20 bins. Atop of that we made the direct difference encoding visible, also as histogram. To further quickly analyze the mean and spread of the data, we showed the statistic measures. They show the data mean and its standard deviation with an area as connection. As can be seen in Figure 8 on the left, the first impression that the visualization gave us, was that the solar radiation is quite similar among all locations as they all have their counts peak at values between $0W/m^2 - 70W/m^2$. Looking at the statistic measures, the data recorded at *Fantasytown* revealed a much smaller spread than the other recordings. To investigate that, we chose to look into one isolated v-plot that contains the *Fantasytown* data. As second location we chose the solar radiation recorded at *Bright County*, because it seemed to show the widest spread of values among others. Along the y-axis it also seemed to have recorded the most high values. This single v-plot can be seen in Figure 8 on the top right.

Looking at the comparison of *Bright County* and *Fantasytown*, our first assumption was confirmed. Almost all recorded values at *Fantasytown* lie in the range of $0W/m^2 - 70W/m^2$, 99,62% of them to be exact. *Bright County*, on the other hand, which we assumed to have the highest spread, only recorded 61,06% values in that range. Zooming in a bit

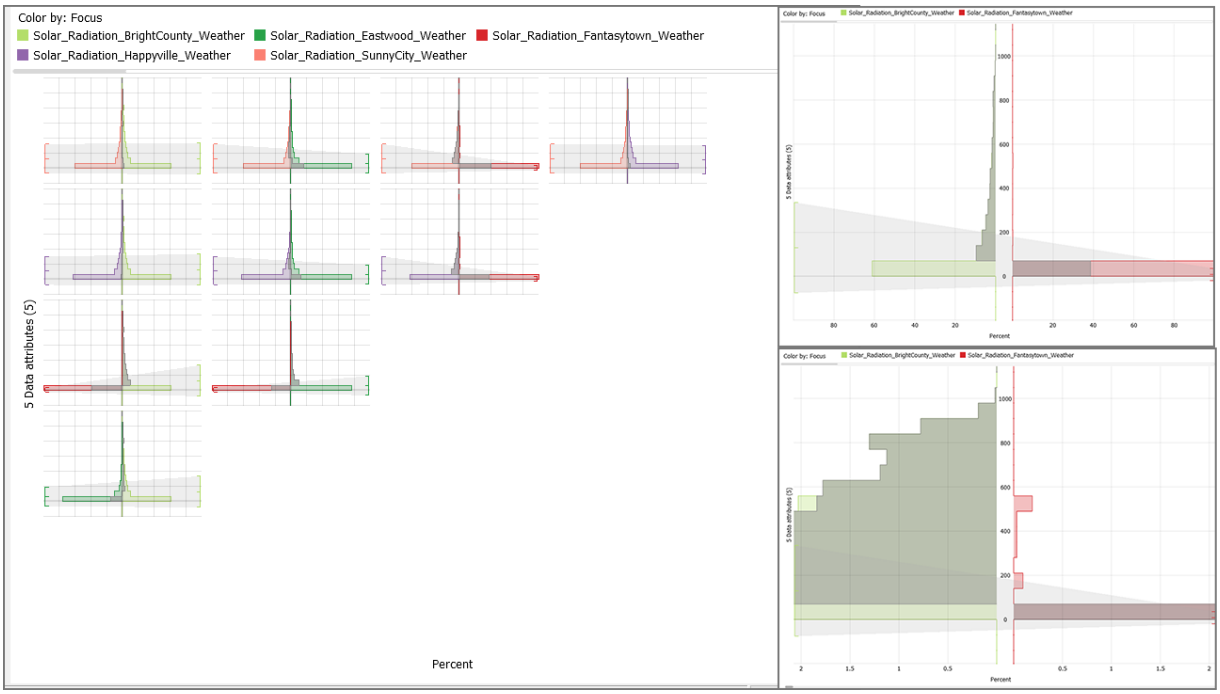


Figure 8: Data from the PV dataset containing the recorded solar radiation at five different locations. **Left:** The data visualized as v-plot matrix using a mirrored histogram, direct difference encoding in the shape of a histogram and statistic measures with an area as connection. **Top right:** Extracted a single v-plot comparing the insolation of *Fantasytown* and *Bright County* from the matrix to view in more detail. **Bottom right:** Zooming along the x-axis reveals even further detail, making especially the nuances of the distribution of the *Fantasytown* insolation data better visible.

further (see Figure 8 on the bottom right), we see that *Fantasytown* had its few highest recorded values at $490W/m^2 - 650W/m^2$, while we measured solar radiation up to values of $9800W/m^2 - 1050W/m^2$ at *Bright County*. This means that throughout the recorded time span, *Fantasytown* generally recorded lower values and never logged particularly high values compared to the other locations. Therefore, we found out that *Fantasytown* received less insolation over the recorded time span, while the other locations seem to receive similar amounts of sunlight with less noticeable differences. Having generally lower solar radiation at *Fantasytown* implies that it might be less preferable as solar plant location than the other locations we looked at.

Example 2: Temperature

For the purpose of the second v-plot use case example, we took a closer look at the recorded outdoor temperature at 5 different locations of the PV dataset. The temperature was recorded from July 2013 to November 2014, covering a time span a bit longer than a year. Thus, the dataset contains the temperature fluctuation over the year caused by the changing seasons. With the help of the v-plot visualization we wanted to find out how strong the measured temperatures differ among the locations. We made use of the density distribution layer, supporting looking at the individual distributions as a local and comparing two distribution as a global task (see Figure 9 on the left). At a first glance, the frequencies of the visualized histograms overall approximate a slightly skewed normal distribution with their counts peaking approximately in the range of 14℃ to 16℃. Their similar shapes indicate that the recorded temperatures are distributed alike among locations. Looking a bit closer, the distribution of the temperature recorded at *Fantasytown* shows a steeper climb and a higher peak than the others.

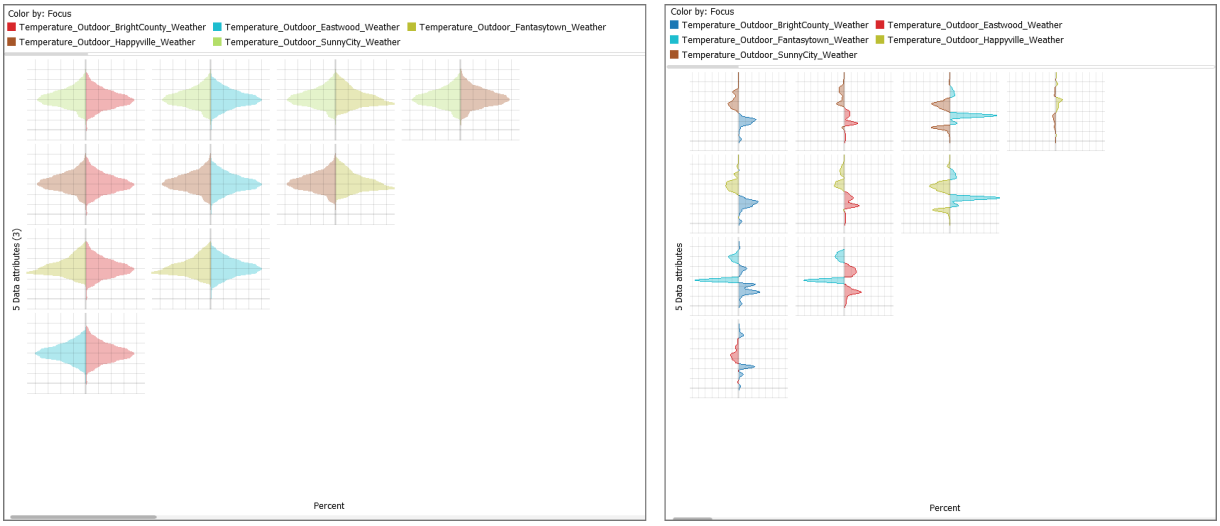


Figure 9: Data from the PV dataset containing the recorded temperature at five different locations. **Left:** The data visualized as v-plot matrix using only a shape-based density distribution. **Right:** Showing the same data as on the left side but represented by the direct difference encoding layer being visible as shape. This helps to spot differences between the data channels easier.

To investigate that further, we adapted the layers of our visualization, displaying the direct difference encoding instead and zoom in on the x-axis (see Figure 9 on the right). This emphasized differences between the displayed data channels, making them stand out more. Looking at the direct difference histogram, we detected that the recordings at *Fantasytown* also showed the biggest difference to all the other locations. Examining the data closer, we saw that *Fantasytown* most often recorded temperatures between 12℃ and 14℃ with a percentage of 24,11% of its samples. Further inspecting the direct difference, we saw that *Fantasytown* had only few recorded temperatures below 8℃ compared to the other locations. Based on these findings we could further investigate if the recorded generated power in *Fantasytown* differed from other locations, and if so, whether this could be linked to the found difference in recorded temperatures. We could also investigate that there is a connection between the recorded temperatures and the recorded insolation from the first presented use case for *Fantasytown*.

3 Evaluating data visualizations

Over time, data visualizations have become a reliable tool to support data analysis tasks. With their increased establishment, research in this field has become more popular as well. Only a relatively small amount of research focuses on the empirical evaluation of data visualizations, as pointed out by Carpendale [15].

As Carpendale [15] elaborated further, the most common approach to test the effectiveness of a visualization technique is a user study. One downside to this strategy is, that the effectiveness is often only tested once and for a few specific tasks. Undertaken user studies often do not cover the same extent of use cases and tasks that the visualization would be used for in real-life scenarios. Further, conducting an effective user study can be time-robbing and expensive. Not every data visualization creator may have access to enough suitable participants for the testing set.
A faster, more generalized, and automated way to test the effectiveness of data visualizations would be valuable. In the following related work section, evaluation techniques concerned with an alternative approach to evaluate visualizations are presented. Subsequent to the introduction of already existing evaluation techniques, we propose a new evaluation approach. This approach proposes computing different pixel-based ratios to determine the scalability performance of a visualization technique in relation to the available screen-space in pixels.

3.1 Related work

Related work regarding the evaluation and effectiveness of visualization techniques is presented in this section. This includes the approach of Eick et al. [19] to measure visual scalability and the introduction of refinement strategies, that can be applied to improve a visualization design [19]. Further, the idea of using measurements based on the saliency of an image and its application to evaluate visualization techniques, is discussed. In particular, the data visualization saliency model developed by Matzen et al. [48] is illustrated. Another approach concerned with the readability and effectiveness of visualizations is the data-ink ratio, as suggested by Tufte [74]. The data-ink ratio along with other minimalist principles by Tufte are concerned with the reduction of a visualization to only its necessary parts.

3.1.1 Measuring visual scalability

"Visual scalability is the capability of visualization tools effectively to display large data sets, in terms of either the number or the dimension of individual data elements." [19]. Questions concerning scalability and performance arise whenever new visualization techniques are developed or extensively tested. Only few approaches to measure visual scalability exist. Eick et al. [19] presented an approach to structure the problem of visual scalability. Visual scalability is a quantification that is dependent on the *factors* and *responses* of a visualization. *Factors* describe the characteristics of a visualization and *responses* can be seen as benefits gained from it, e.g., insight or discoveries. The problem with that approach is that responses cannot be measured. Instead, Eick et al. [19] suggest replacing responses with measures of visual scalability. These measures of scalability can be subdivided into database metrics and visualization characteristics with their distinction as follows:

- **Database metrics** are concerned with the size of the underlying database, which is depending on e.g., the number of rows and columns of an underlying data table, but can also account for the amount of bytes needed to store the data.
- **Visualization characteristics** are concerned with the quantity of elements visualized by a certain visualization technique. An example for that would be the number of bars visualized in a histogram.

Possible factors influencing the scalability can be quantified and are grouped in the following way:

- **Human perception** describes the fact that the capability of the human brain and eyes are a limiting factor concerning visual scalability. Given the current standard of knowledge, that around 6.5 million pixels are perceptible by the human eye, consequential the human eyes are currently not the bottleneck for scalability.
- **Monitor resolution** impacts visual scalability in terms of physical monitor size and its pixel resolution. All though monitors have increased in size and resolution, they are still

incapable of matching the perceivable amount of pixels by the human eye. This is why monitors are still limiting visual scalability.

- **Visual metaphors** are in the context of this work also called visualization techniques. This item also includes mapping of data attributes onto color and other adjustable properties of the visualization. The choice of visual metaphor affects the scalability strongly. Depending on the overall circumstances some visualization techniques scale better than others.
- **Interactivity** describes techniques used to interact with the visualization. This includes among others: panning and zooming, brushing and selection.
- **Data structures and algorithms** also have an noticeable effect on the visual scalability. Algorithms in terms of computational tasks and rendering tasks must scale well in order to support overall scalability. The importance of a very efficient algorithm increases with the complexity of the used visual metaphor and the provided user interactions. Especially for many interactions, the user expects the effect to take place instantly (e.g., panning).
- **Computational infrastructure** is concerned with the hardware that is used for computation, e.g., CPU, GPU, access times for storage devices, network speeds.

Based on these *measures of scalability* and *factors*, refinement strategies can be applied to improve visual scalability. When determining the limitations of a visualization, refinement strategies can be adopted to compensate for the limiting factors. Among others, refinements can be made for the aforementioned visualization techniques. Refinement strategies for visualization techniques include: optimizing drawing algorithms, adapting element sizes when little screen space is available, or drawing only every n^{th} label at an angle to save space and avoid overplotting. The factor interactivity also offers many possibilities to improve overall visual scalability through the following refinements: supplying zoom and panning controls, and selections like filtering and focusing.
Combining multiple views increases scalability as well, as pointed out by Eick et al. [19]. Using a *central view* that uses a visualization technique that gives a good overview in combination with *supporting views* that act as filters for the selection of data subsets.
Another related work on that topic was done by Jakobsen and Hornbaek [35], who examined how well maps scale in respect to varying screen-sizes and variable information space. They also inspected how user interactions (like zooming, focus and context, overview and detail) affect visual scalability.

3.1.2 Visual saliency models

Another approach to measure the effectiveness of data visualizations are saliency models. "Visual salience (or visual saliency) is the distinct subjective preceptual quality which makes some items in the world stand out from their neighbors and immediately grab our attention." [33]. Saliency maps can be generated to simulate where the attention of a viewer is directed

towards. More specific, they are topographical maps that represent visual saliency of a corresponding visual scene [56]. The resulting map is influenced by parallel processes in the

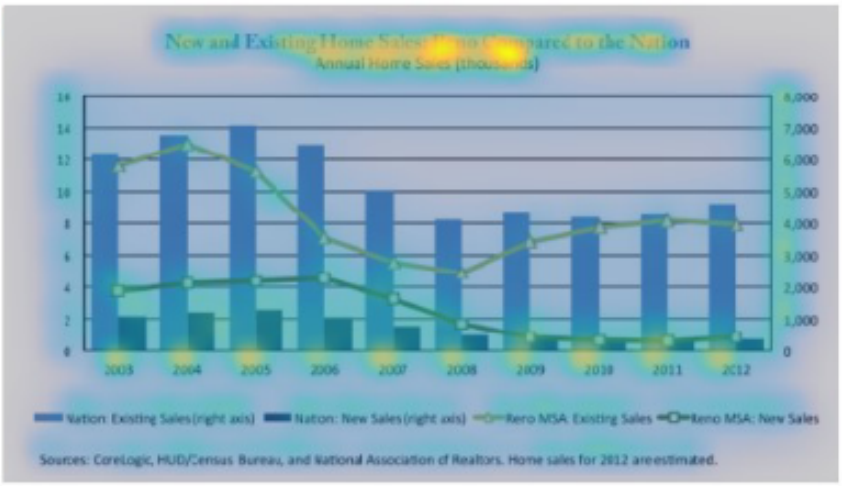


Figure 10: Showing a bar chart visualization, with its DVS map as overlay. Cropped version of the original image taken from [48].

brain, described as bottom-up and top-down visual attention. Generally speaking, properties that make a particular region stick out from its surroundings through certain sensory stimuli are bottom-up factors. Color, scale, shape, motion and contrast are examples for bottom-up factors. Experiences, tasks, expectations and goals an individual has are top-down factors that influence how saliency is perceived. This means that regions with high saliency from a purely bottom-up point of view might have overall low saliency when also taking top-down aspects into consideration. To create a saliency map for an image, a saliency model can be applied. It produces an image, encoding a value representing the level of saliency per pixel. These models generally work to estimate only bottom-up visual saliency, predicting where a viewer will look at in the image. These generated maps can further be used to determine whether the bottom-up visual attention suits an top-down motivated task. Most of the models are designed for natural scenes and photographs and do not work well with abstract data visualizations. Some of the reasons being the difference in their spatial scales and visual features like color, white space and text, as pointed out by Matzen et al. [48]. Based on these observations they developed a model concerned with evaluating the effectiveness of data visualizations through saliency, called the *Data Visualization Saliency* (DVS) model. The DVS model is based on the Itti model [34] that performed best applied to abstract data visualizations among existing models. Matzen et al. [48] further refined the model to use another color space and to better determine the saliency of text, which is a major problem of most saliency models when applied to data visualizations. To evaluate the performance of the DVS, it was tested along with other saliency models and compared for a set of visualizations where fixation data of viewers was available. The results of this test showed, that the DVS model performed significantly better than other visual saliency models. Matzen et al. [48] propose the DVS as tool for visualization designers to

simply and quickly evaluate how well their intended points of interest of the visualization overlap with the saliency map produced by the DVS, and, which parts of the visualization draw the most attention. A sample application of how to test visualizations by means of a salience metric was also shown by Jänicke et al. [37].

3.1.3 Data to Ink ratio

Tufte [74] supports the approach of minimalist designs for information visualizations. According to this strategy, elements of the visualization should either directly represent the underlying data or be left out to not distract the viewer. A summed-up principle of this approach reads as follows:

"Above all else show the data."

In respect to this principle, the data-ink ratio is formulated. The term ink is used to describe the parts of a visualization whose colors differ from the background color. The *data-ink* describes the parts of the visualization that cannot be erased without losing the actual representation of the data. An example for non data-ink are graph axes and the tick marks of those axes. From those two terms, the data-ink ratio can be formulated as follows:

$$\text{data-ink ratio} = \frac{\text{data-ink}}{\text{total ink used in the graphic}} \tag{1}$$

This simple equation returns a percentage, describing the data-ink ratio as a statistic measurement. According to Tufte [74], the share of data-ink should be maximized, but within reason [74]. A higher result implies that more of the overall used ink in the visualization is devoted to directly represent the underlying data. From this follows that the reciprocal value (1 - data-ink ratio) is descriptive of what proportion of the visualization can be erased without losing data-information. The minimalist approach further suggests to remove any redundant data-ink, as it does not give any new insight on the data. An application of maximizing the data-ink ratio is illustrated in Figure 11. It shows a bar graph, and the same graph split into its redundant and its necessary data-ink. Following the minimalist principle, ideally the variation that should be used out of the three, is the one on the right, which has the highest data-ink ratio. The work of Tufte regarding information visualization is quite well-known among visualization designers. Some studies were conducted to give an indication about the effectiveness of these principles. For example, McCormick et al. [51] suggested that in some cases redundant data-ink like tick marks along an axis can actually improve readability of a visualization. Another study was performed by Gillan et al. [23]. It covered four experiments to test Tufte's principles applied to visualizations used for data analysis tasks. The outcome of the experiments showed that non-data ink can have a positive effect on completing analysis tasks faster and with more accuracy. Depending on the location, function and user task, non-data ink can also deteriorate readability of a graph, supporting the approach to maximize the data-ink ratio.

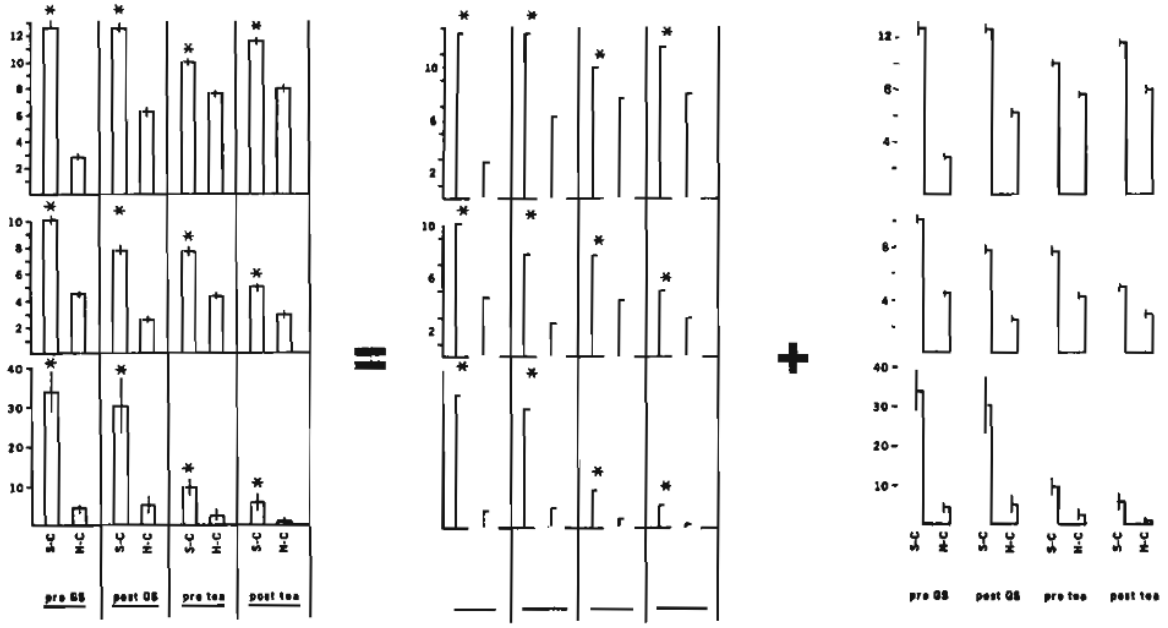


Figure 11: An applied demonstration of increasing the data-ink ratio of a bar chart. This graphic depicts how a bar chart visualization(left) consists of redundant ink(middle) and data-ink that cannot be omitted without losing information(right). Images taken from [74].

Inbar et al. [32] conducted a user study examining the preference towards minimalist visualization designs. In the respective experiment, the test group consisted of eighty-seven students. They rated a standard bar graph and three increasingly more minimalist versions of the bar graph created by Tufte [74] under various conditions. The findings of the study suggested, that users prefer less minimalist designs. This result might be influenced by the fact that most users tend to prefer already known visualization designs over novel ones [28].

3.1.4 Discriminability tests

In the context of their work, Veras et al. [75] define the term discriminability as: "*Given a collection of datasets, the average perceptual distance between the corresponding visualizations*". Discriminability tests can be conducted to evaluate the perceived differences in images. One measurement to test for image disparities is the *Structural Similarity Index* (SSIM) which is commonly used in the field of image quality analysis. A modified version of it being the *Multi-Scale Structural Similarity Index* (MS-SSIM) developed by Wang et al. [77] which also integrates image resolution and viewing conditions. MS-SSIM works on multiple images as input, generating a similarity map based on luminance, contrast and structural similarity. Veras et al. [75] used a modified version of the MS-SSIM to measure discriminability between abstract data visualizations depicting different datasets. They introduced an additional parameter to also detect

differences in hue, which the color-blind MS-SSIM does not support. They further determined that grid lines that are not aligned among visualizations would have a greater effect on the discriminability than actually perceived by the user. They suggest leaving them out completely when computing the MS-SSIM, as it leads to less distorted values. The same goes for text and labels present in the visualization. Based on the outcome of their benchmark experiments, they suggest that a discriminability test should be part of the evaluation of data visualizations that can be applied very early on in its design stage. For established visualization designs, MS-SSIM can give practical insight to their often only theoretically discussed discriminability.

3.2 V-plot matrix scalability evaluation

Blumenschein et al. [11] have tested the v-plot visualization on the basis of a user study with promising results. The matrix arranged v-plots were also part of this study. However, the authors did not examine the limits of the v-plot matrix in terms of dimensionality and configurations in reference to given screen-space. A method is needed to evaluate the scalability of the v-plot matrix that can approximate a guideline for creating and working with v-plot matrices.
In the recent subsections it was shown that there are many different approaches when it comes to evaluating data visualizations. However, notions on scalability of the methods and standards for evaluations are still open research topics. Therefore, a simple, pixel-based approach is presented to test data visualizations for their scaling performance with reference to different amounts of available screen-space. The evaluation is not concerned with the scaling of the underlying information space, which is kept constant [35]. User interactions with the data visualization are also not considered. For the purpose of this thesis, the suggested approach will be tested by means of the v-plot matrix visualization.
The aim is to answer the following formulated questions:

- *How well do v-plots scale with respect to the available screen space?*
- *How do bin count, matrix dimensionality and layer configurations affect the scaling of v-plots?*
- *Under what circumstances do the v-plots lose their discriminability?*

The proposed approach is developed with regards to the findings of the aforementioned related work in the field of data visualization evaluation. The structure of our proposed evaluation strategy consists of computing three different pixel-based ratios across exported v-plot matrix images of different sizes and correlate their results to make a statement on their scalability. Referring to previously discussed techniques, the ratios that will be observed for the evaluation are:

- **Data-ink ratio:** Following the minimalist approach and guidelines for optimizing visualization quality explained in Section 3.1.3, the data-ink ratio is computed.

- **'Foreground-background (FG:BG)' ratio:** A percentage that indicates how much of the image space is occupied by ink. This measure is inspired by a very simplistic take on saliency models as it measures the ink sticking out from the background.
- **Discriminability of v-plot sides as ratio:** A measure representing the number of different pixels between v-plot sides when comparing the shape of their data-ink. As also discussed in Section 3.1.4 grids will not be used in this calculation, as even a small offset would lead to a higher discriminability than probably noticeable by a viewer.

The idea is that observing those ratios over varying screen-spaces and with different v-plot configurations, should reveal trends and patterns when scaling v-plot matrices. Ideally, the ratios should stay relatively consistent over different sizes. This would mean that smaller plot sizes still preserve approximately as much information as they display on larger scales. However, we expect to see bigger ratio changes when it comes to small screen-space or v-plot configurations of higher complexity. The main reason for this hypothesis is the fact, that the drawn primitives cannot be scaled down to an arbitrary size without losing accuracy. This same effect is also expected to be apparent when using high bin counts on insufficient screen-space. Setting a bin count higher than the number of pixels available might yield unpredictable results as bins can no longer be mapped to pixels unambiguously.
In terms of the measures of scalability as defined by Eick et al. [19] and discussed in Section 3.1.1, this approach solely focuses on visualization characteristics and ignores database metrics. Further, its focus lies on the influence of the *factors* of *monitor resolution* and *visual metaphors*. In the context of this method, we are not concerned with the actual monitor resolution but with the screen-space in pixel available to the visualization.

4 Implementation

The implementation work done for the purpose of this thesis consisted of two parts. The first part was the implementation of the v-plot matrix visualization. This implementation was an extension to an existing visualization software. This required to get familiar with the code structure of the present software. Afterwards, a concept for the implementation was made, following its realization in the next step. The second part of the implementation consisted of a scripting solution to realize our proposed visualization scalability evaluation approach. The prerequisites, requirements and implementation of the evaluation script are outlined in the dedicated sections.

4.1 V-plot matrix implementation

To answer the questions regarding the v-plot matrix scaling performance, the first step was to implement a v-plot matrix visualization. As part of this thesis, the implementation was integrated into the software *Visplore by VRVis* [63].

4.1.1 Visplore by VRVis

Visplore by VRVis [63] is an interactive software tool for the visualization and analysis of large datasets and was specifically designed for the work with time-series data. The software supports various data visualizations as well as interaction techniques and configurable calculations to support users analyzing their data.The software supplies various dashboards, as depicted in Figure 12, that are specialized to help with certain use cases when working with different types of data. Each dashboard consists of multiple views and various functionalities to assist the user in analysing their data.

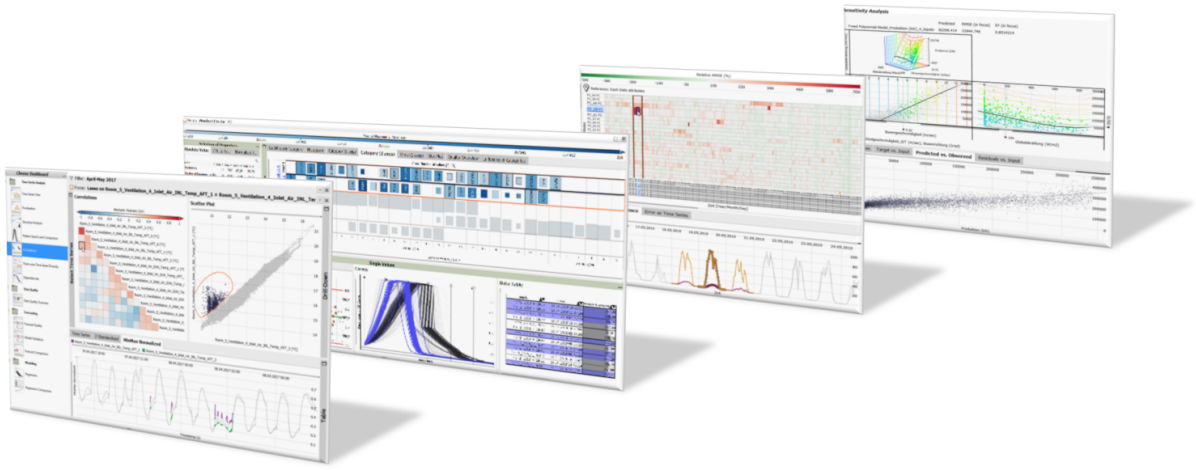


Figure 12: Multiple screenshots of the Visplore by VRVis software displaying different dashboards that support the execution of various analysis tasks. Image taken from [63].

In the context of Visplore by VRVis, the term *view* is used to describe a part of the program that contains the functionality to display a certain visualization technique that can be configured and interacted with. These *views* are displayed in their respective window which is a sub-window of Visplore by VRVis. In computing terms, a window describes a framed part of the display that is used by a program to display information. Such *views* are usually used in combination, creating a so-called *dashboard*.
The data is processed in the form of *channels*. A channel corresponds to one variable of the dataset. When the dataset is visualized as a table, one channel is contained within a column. Channels can either contain a date and/or time, numerical values or categorical values. While numerical channels can contain values from an infinite numerical value range, categorical channels contain values from a countable number of values, which do not have to be numeric.

Categorical channels often contain values from a fixed set of values, which are also referred to as *groups*. Meaning that one group of a category corresponds to a specific value, that the values of a categorical channel can take on.

Most of Visplore by VRVis is written in the C++ programming language. The choice to use C++ was made based on the fact that C++ was developed to be performant, efficient and flexible. Ideal properties for a software with the requirement to manipulate and represent large datasets in real-time. Among others, modern C++ supports object-oriented programming as well as low-level memory manipulation. Further, because C++ is based on the C programming language, it supports the usage of libraries that are written in C.
In Visplore by VRVis, some parts that are not crucial in terms of performance are written in Python [64].
The user interface is implemented using the GTK cross-platform widget toolkit (GIMP ToolKit) [25]. Rendering is performed with OpenGL (Open Graphics Library) [57], a cross-platform and cross-language application programming interface (API). OpenGL enables hardware acceleration through usage of the GPU. Visplore by VRVis also supports the usage of Mesa 3D [52], which allows graphical computations to be done on the CPU rather than the GPU.

As part of this thesis, v-plots were implemented to extend the visualization methods supported by Visplore by VRVis. More specific, the currently existing histogram view of the software was used as basis for the v-plot matrix implementation. The reason being, that the histogram view already contains some elements that are also needed for v-plots. Like the histogram visualization, and a density distribution function visualization. Therefore, the internal workings of the histogram view are explained further in the following subsection.

Histogram view

As part of the software, data distributions can be visualized as histograms in a dedicated view. This view is implemented as the so called *histogram view*. The histogram view already supplies the functionality of binning the underlying data and visualizing it either as a bar graph or a filled broken line chart. *Layers* can be used to draw multiple different visualizations of the data on top of each other. In case of the histogram view, the histogram visualization itself is the bottom layer. Functionality to show additional layers is provided, e.g., in the form of a distribution function layer. This distribution function can either be a normal distribution or a logarithmic normal distribution visualized through a stippled polyline. The comparison of multiple histograms representing different datasets is possible in juxtaposition, through showing multiple histograms one below the other. It is also possible to show a superposed arrangement, enabling the comparison of different groups of a category in a dataset. This is achieved through separation by a categorical channel through color encoding. The differently colored histograms are then drawn as overlays and with transparency. The supported modes of the view are shown in Figure 13.
Existing interactions to assist in the comparison of the data include zooming and panning the

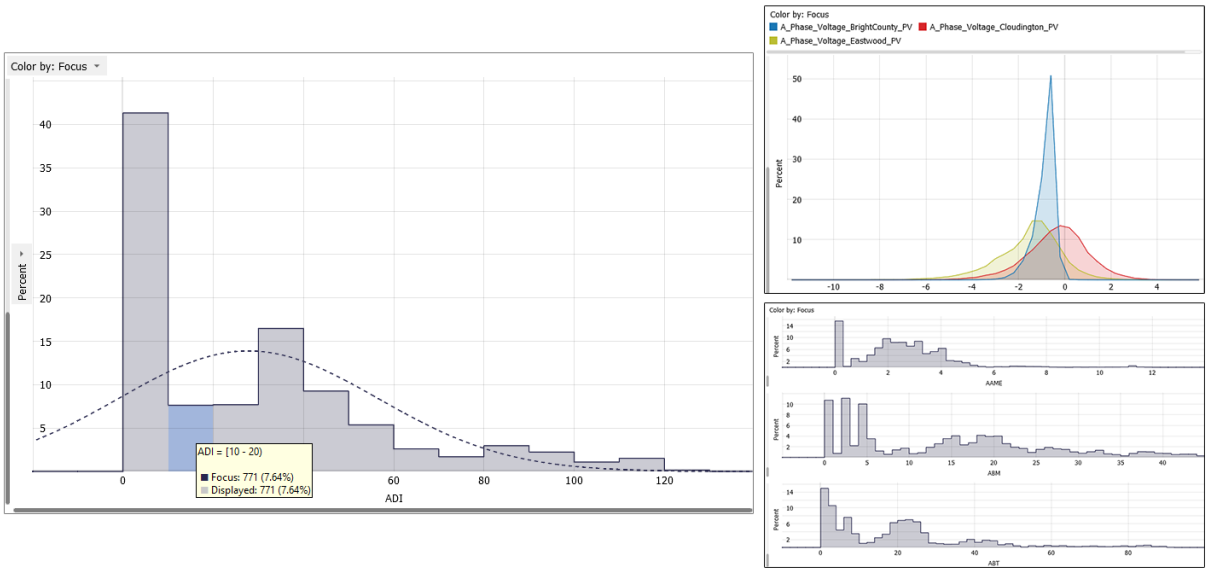


Figure 13: Graphics are taken from screenshots from Visplore by VRVis. **Left:** Single histogram with overlaid normal function visualized as stippled line. Information about the underlying data of the hovered blue bin is listed in a popup that appears on hovering. **Top right:** Multiple histograms in a superposition arrangement drawn with transparency as filled broken line graph. **Bottom right:** Multiple histograms drawn one below the other in a juxtaposed arrangement internally also called a trellised layout.

view. As all other visualizations in Visplore by VRVis, the histogram view supports cross-filtering between views of a dashboard. This enables the histogram view to be updated whenever the current selection of data is changed in another view and vice versa. The binning of the data can also be adapted by changing how many bins the data should be split into. This directly influences how detailed the underlying data is represented, and should be used considerately, as discussed previously in Chapter 2.1.1. The minimum number of bins supported by the histogram view is 1 and the maximum number of bins is 256. The bin width cannot be adapted by the user. It is calculated in relation to the total screen-space available, with respect to the specified number of bins. Another adjustment that can be made is whether the frequency of appearing values should be represented as total count or as percentage.
The source code of the histogram view consists of multiple C++ classes. The most important classes that help explain its logical structure are listed below, briefly summarizing the functionality that was reused for the v-plot implementation:

- **Main view class:** It holds the instances of all other classes of the view needed for computations, drawing and more. It also stores parameters shared among them. These parameters include, but are not limited to, the *mapping* for the axes and the currently used visualization mode for the histogram (bars or polyline) and the used density distribution function. The *mapping* supplies the computations needed to map values from a source value range to a target range and vice versa. In this case, the *mapping* is used to project the frequency values on the y-axis of the histogram (source range) to the given

screen-space (target range). The *mapping* is further used for projecting the values of the x-axis, that represent the binning, to the given screen-space.

- **Data storage:** A storage class for the components which are drawn later on. Those components include the vertex data for drawing the histogram and the vertex data for the density distribution function.
- **Data updater:** Provides functionality to access the *data storages* and update their content.
- **Renderer:** Calls the draw function of the *data storages'* components in the right order after evaluating their visibility. Utilizes the OpenGL API for its drawing routines.
- **Backend:** Responsible for coordinating *the data updater* and the *data storages*. This includes keeping the *data storages* up-to-date whenever the underlying data changes, for example, through user selections. It further issues the *renderer* to execute whenever a redraw is needed. It also stores and updates the binning of the histogram. The *backend* holds an instance of the *grid* class, which is responsible for drawing the axes along with their legends, ticks, and grid lines.
- **Frontend:** Contains the controls of the *view* and displays them. This includes buttons, option dialogues and the coloring legend. It uses the GTK library for drawing them. It notifies the *backend* whenever the user performed an action that requires re-computation or redraw.

To display a histogram within the *histogram view*, the following steps are traversed. When a data channel is selected, the *backend* notifies the *data updater*, which is responsible for updating the stored data within the *data storages*. The *data storages* are updated to contain the vertex data for drawing the histograms and the distribution function of the currently selected channel. The axis mappings, which are stored in the *main view class*, are re-calculated, so that their source range fits the value range, that the selected channel spans. The target range is set to the resolution of the view window. The binning and the grid, which are stored in the *backend* are updated. When the necessary components have been updated successfully, the *backend* issues the *renderer* to draw the vertex data of the *data storages*. The user can then interact with the histogram through controls that are supplied by the *frontend*. For example, zooming or panning the view leads to the re-calculation of the binning and to updating the zoom factor of the mappings. Following these steps, the *renderer* clears the view and draws the vertex data again.

Coloring legend

Coloring legends play another important role when visualizing data. The purpose of a color legend is to map the colors that are used in a visualization to a certain data channel or category. Many *views* of Visplore by VRVis use a coloring legend, including the *histogram view*. In

Visplore by VRVis, the color legend is usually displayed at the top or bottom of a view. Besides the color mapping, the coloring legend also displays whether the underlying dataset is currently shown as a whole, or if a categorization is enabled. A categorical channel can be used for categorization. If a categorization is enabled, the legend displays the name as label and a colored square for each category group. The coloring legend allows the user to choose whether to show the whole dataset, or split the data into subsets by the requested categorization. In case categorization is chosen, for each group of the category, multiple histograms are then drawn as overlays in one single plot. Each histogram is displayed with the according color for the category group it represents. Which color is assigned to which category group is calculated internally when categorization is performed. This color stays persistent over all views. The user might change category colors according to personal preference or to better fit a certain use case. The coloring legend also supports choosing which category groups are visible. The data of all other classes can then either be not shown at all, or can be combined into one category class named *other* which is then treated like its own category group. Figure 14 shows the coloring legend as it is currently implemented in Visplore by VRVis along with its options dialogue. The coloring legend can also be seen in Figure 13 in the top bar on the right of the screenshots.

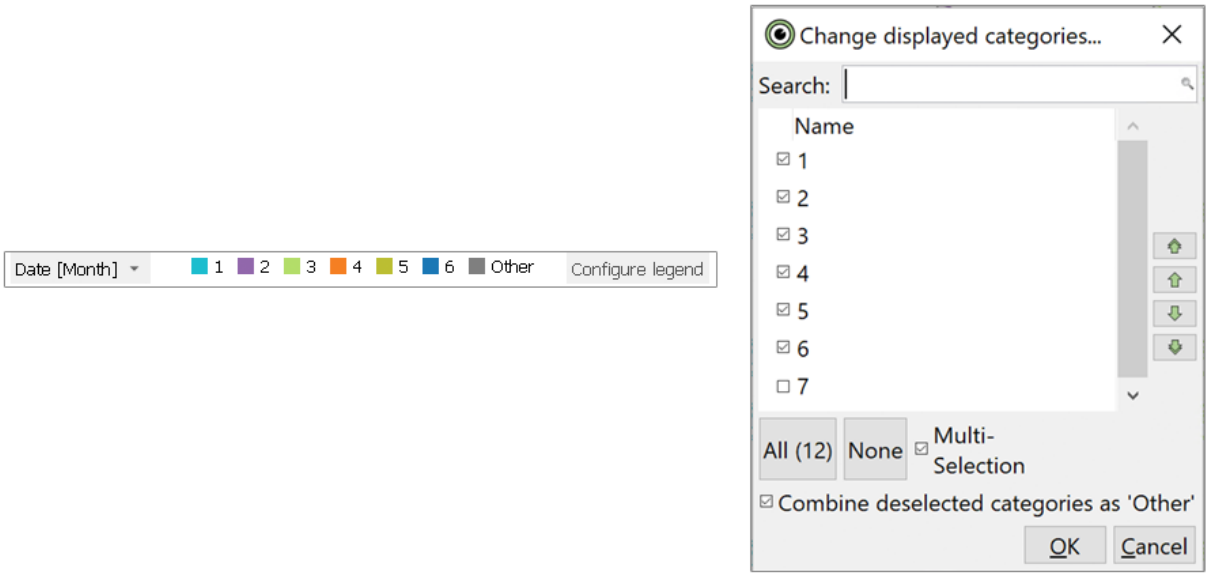


Figure 14: **Left:** The coloring legend as displayed in the histogram view, as bar above the histogram plot. With the button labelled *Date [Month]* on the left, the categorization can be chosen. The middle of the coloring legend shows the colors per category group in small squares along with the category group's name, in this case numbers representing months. On the far right of the legend a button labelled *Configure legend* is available to open the options dialogue. **Right:** The options dialogue of the coloring legend. It allows for selecting which category groups should be shown and whether deselected classes should be combined to be represented as an accumulated class labelled *other*.

4.1.2 Implementation steps

One major goal of this thesis was the integration of a v-plot implementation into the existing histogram view. Therefore, the histogram view was expanded to support drawing the layers, of which a v-plot consists, as overlays on top of the displayed histogram. The binning and drawing routines of the histogram remained as-is and were re-used. The binning is set at the time of creation of the view and can be adapted by the user at run-time. Other than that, the steps executed to expand the histogram view to support drawing v-plot matrices are listed subsequently:

1. **View mode:** In order to also keep the existing histogram view working as is, a new view mode was introduced internally. It tells the software whether to visualize the data as histograms or v-plots.
2. **Axis mapping and labelling:** To fit the v-plot representation, the histograms of the view were expanded to support being drawn mirrored and with their x- and y-axis switched. In addition, the already existing labels and the grid, which are the final layer of the v-plots, were adapted. This step involved centering the x-axis between plots and making the y-axis symmetric.
3. **Layer expansion:** The histogram view already offered overlaying a distribution function. This layer was expanded to support other density functions: the KDE function and a shape-based density function.
4. **New layers:** The layers for direct difference encoding and statistic measures were created.
5. **Rendering:** The rendering functions were expanded by functions to support drawing the additional layers, the existing histograms, and the density distribution in a reasonable order.
6. **Layout:** The layout of the histogram view was changed to enable viewing multiple v-plots in a matrix visualization.
7. **Controls:** The v-plot layers were made configurable through an options dialogue. Layers can be configured individually. Further, various v-plot configurations are provided as presets, to assist the user in finding a fitting configuration.

Desired behaviour

With our implementation, we want to achieve that the view behaves as described here. When the view is newly created and shown, only the first channel of the loaded dataset is selected and drawn as a histogram. When the user performs a selection of two or more channels of the dataset or selects a categorical channel, the view mode is switched to display v-plots. In case

three or more channels are selected, or the chosen categorical channel contains more than two category groups, a v-plot matrix is created. To distinguish channels or category groups in the data, each is assigned a color automatically. This color may also be changed by the user at run-time. The newly implemented v-plot visualization also uses this color to draw its visible layers. This color encoding allows for not needing to draw the grid and labels for the matrix view on small screen-space. Through the consistently visible color legend of the view, the v-plots stay referencable to their underlying data. An options dialogue offers available presets for the v-plot layers and the options to adjust each layer separately.

4.1.3 Implemented changes

Based on the implementation steps described in Section 4.1.2, the following changes were implemented to achieve a working v-plot integration into Visplore by VRVIS in compliance with the desired behavior described in Subsection 4.1.2.

View mode

The histogram view had to remain usable as-is, therefore, *view modes* were introduced. The default view mode was the *VIEW_MODE_HISTOGRAM*. When set, the view had the functionality to draw histograms like before the v-plot extension. On selection of multiple data records, histograms were either drawn trellised or as colored overlays within a single plot, as described in Section 4.1.1. The other view mode which introduced new functionality upon activation was the *VIEW_MODE_VPLOT*. On selection of multiple data records or a category, a single v-plot or a v-plot matrix was drawn. The switch was implemented as evaluation of a simple if-statement. The view mode parameter was set externally, and was evaluated on view creation and, therefore, could not be changed by the user at run-time.

Axis mapping and labelling

The main view class of the histogram view stored and maintained two data mapping instances, one for the x-axis and one for the y-axis. In this case, x and y corresponded to the coordinate system of the screen, where x referred to the horizontal and y to the vertical axis. The data mapping contained multiple parameters, these included a zoom factor, the mapping's source range, and its target range. Among other functionalities, the data mapping was capable of mapping a given number of values of a passed array from its source range to its target range. This mapping function was used for mapping the histogram vertex data to its screen-space position relative to the viewport that was set in OpenGL.
The labelling of the axes was performed by the grid. Based on the mappings of the x- and y-axis, it calculated how many axis ticks could be drawn on the available screen-space. The labelling was adapted whenever the mapping changed.
In the view mode *VIEW_MODE_HISTOGRAM*, the range of values that the distribution could

assume was mapped onto the x-axis. The number of values per bin was mapped onto the y-axis. For the *VIEW_MODE_VPLOT* this axis mapping was switched. The count of values was mapped to the x-axis and the possible value range was mapped to the y-axis.
The grid was expanded to support drawing the axis labels of the horizontal axis symmetric around zero. The vertical axis was extended to be drawn in the center of the available space.

Layer expansion

Software logic to draw the histogram layer already existed. The histogram could be drawn either as bar chart or as filled broken line graph. In addition, support to draw an overlaying distribution function as a stippled line with a fixed color was already implemented. A normal distribution as well as a logarithmic normal distribution were already supported and reused as an option in the v-plot visualization.
The distribution function overlay was expanded to also support the computation and rendering of a shape-base density distribution as well as a kernel-density-estimation (KDE) with adjustable bandwidth and a variable number of ticks to use. Both the shape-based density function and the KDE were implemented to draw a catmull-rom spline. The generated curve was displayed as a filled area with transparency and without an outline. The existing normal and logarithmic normal density distribution functions were adapted to also support being visualized in that manner when v-plots are drawn.

New layers

Newly introduced layers were the statistical measures layer with the possibly shown connections realized as a separate class and the direct difference encoding. The statistic measures were computed based on the binning of the underlying data for each v-plot side. The measures were always represented as colored lines according to the existing color legend. The statistic measures connections layer and the direct difference encoding layer were different to the above mentioned layers. Due to their nature, they used both sides of a v-plot as basis for their computations.
The statistic measures connections used the calculated aggregates from the statistic measures layer. These were then mapped to the screen-space and connections were drawn either as grey lines and/or a transparent grey area.
The direct difference encoding was computed based on the binned values from both v-plot sides. The total difference per bin was calculated. Based on the outcome, drawing primitives were computed. This layer could then be drawn as histogram and/or shape with an outline and transparent filling. The direct difference encoding was either drawn in its encoded color or in a dark grey. When both, difference shape and difference histogram were drawn, the shape was displayed using its color encoding and the histogram uses dark grey.

Rendering

For rendering the v-plots, a new renderer class was created. This decision was made after a first attempt to restructure the existing renderer. The functionalities needed for the rendering of the v-plots were very different to the existing rendering functionalities of the histogram view. Some of the major differences that led to this decision being made are listed below. First of all, the drawing of a matrix was not handled in the histogram renderer yet. This included not only drawing multiple histograms, but also to adjust the used line width accordingly, as it had to get smaller with an increasing count of plots in the matrix and a decreasing amount of available screen-space. The existing renderer did support the rendering of a layer containing a data distribution. This was reused and the optional rendering of the other additional layers of the v-plot were added. The new renderer class was capable of handling the different axis mappings that were needed to draw the plot sides. This included one mapping to draw the left v-plot side and one mapping for the right side. As it will be elaborated in the following Subsection 4.1.3, the new renderer was also capable of automatically choosing colors for the v-plot layers and made sure that each v-plot side stayed relatable. Since the usage of multiple colors was not yet supported by the histogram renderer, this functionality had to be implemented in the new v-plot renderer.

Layout

The existing layout supported displaying multiple histograms only in a way that one could be displayed below the other, or all in one plot as overlay. To allow for the display of multiple v-plots in a matrix arrangement, a new class was introduced, responsible for computing the layout of the v-plot view. This class stored instances of the newly created renderer and the grid used for the v-plots. Apart from maintaining the grid, the class was also responsible for updating the newly created layers.
The layout was created with respect to the available screen-space and the number of selected data channels or category groups.

Controls

The frontend was extended to create and display an options dialogue containing most settings relevant for the v-plots, as can be seen in Figure 15. Apart from the settings per v-plot layer, the number of histogram bins could be adjusted here. An exception were the settings for the labels layer which were related to the grid display, so that their controls remained in the histogram view window. The options dialogue offered some preset configurations to cover common use cases quickly. To be very flexible when using v-plots, advanced users could set the options for each v-plot layer separately. This included hiding or showing a layer. When a layer was shown, different visualization styles could be chosen. Configuration presets could also be used as starting points. Based on them, layers could be further adjusted. The visualization options

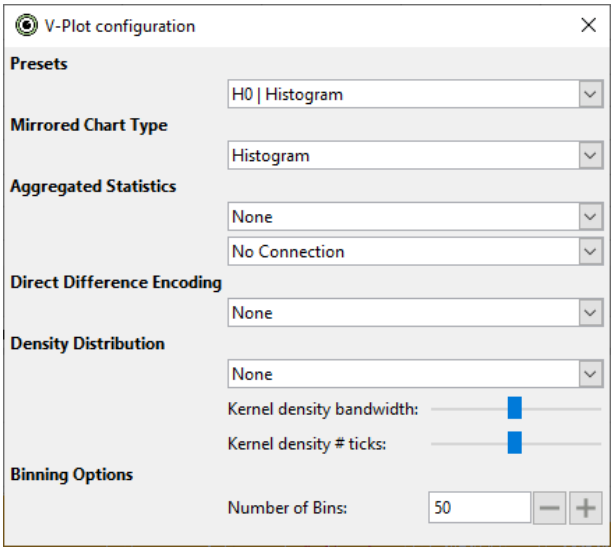


Figure 15: The implemented options dialogue, containing the configurable settings of the v-plot implementation.

per layer consisted of:

1. **v-plot presets:** Lets the user choose one of eight presets that contain different combinations of v-plot layer configurations to apply to the v-plots. Separately changing layers afterwards is still possible. Per default, the v-plots are set to draw only the first layer containing the mirrored histogram and the labels if sufficient screen-space is available.

2. **Mirrored chart type:** Can either be drawn as bar graph or not be visible at all.

3. **Density distribution:** Supports showing the normal distribution, logarithmic normal distribution, a shape-based distribution or a kernel density estimation. It can also be hidden.

4. **Direct difference encoding:** Can be shown as either bar graph or density shape. Also supports showing both at once, or none at all.

5. **Statistic measures:** Currently supports showing the following combinations of indicators at once: mean and standard deviation, mean and standard error, mean and quartiles or median and quartiles. Can be either shown or hidden. If statistic measures are shown, it is possible to visualize a connection between the two sides of the plot, either through lines and/or a transparent filled area.

6. **Labels:** Are an exception, because their configuration cannot be set in the options dialogue, but must be adjusted in the view window directly. Labels containing the name of the currently selected data columns are automatically generated and cannot be adapted. What can be changed is, whether the data is shown as percent or count, which directly influences the labels of the grid ticks. The displayed grid is affected by the mapping and

adapts automatically when zooming or panning to find a meaningful representation. Further, grid ticks and grid lines are hidden automatically when the available screen-space per v-plot undercuts a certain threshold.

Coloring

For good readability, the coloring of the v-plots in this implementation was oriented at the default settings of the v-plot designer by Blumenschein et al. [11]. It can be seen in Figure 16 on the right. This design proved to perform well in terms of usability in their qualitative expert user study. The default coloring in the v-plot designer is composed as follows: the histograms of both sides are drawn in a light grey and for the direct difference encodings a darker grey is used. The overlaid density plots are drawn in a color, that is specifically assigned to one side. In this case, the left side is assigned blue and the right side data is visualized in red. The handles indicating the statistic measures are also drawn in the according color of the side they belong to, either blue or red in this default case.

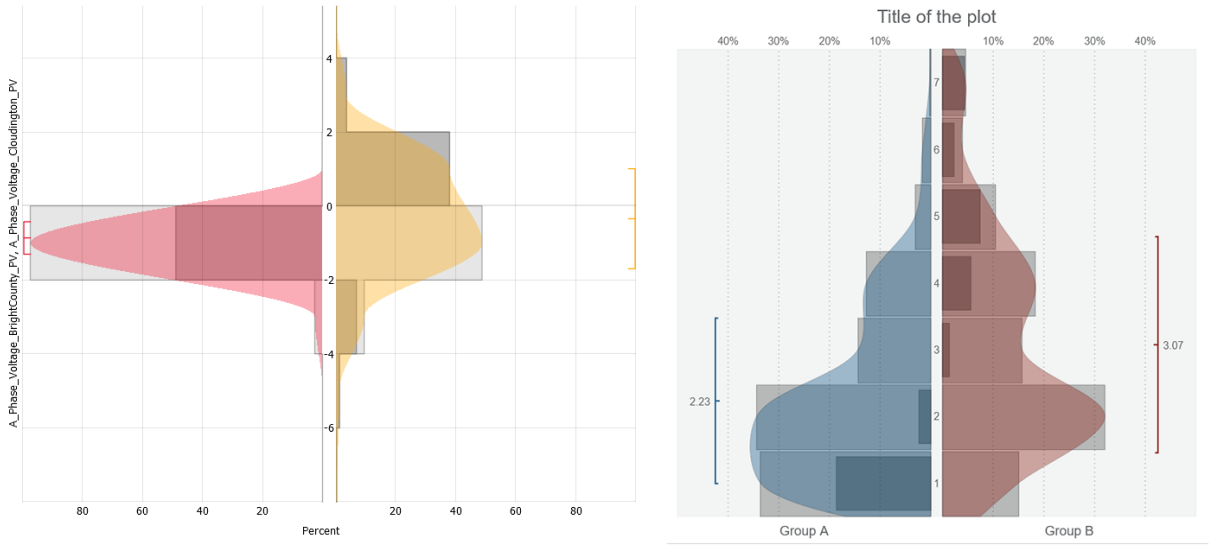


Figure 16: The coloring used for the components of our v-plot implementation was oriented at the color scheme the v-plot designer uses. **Left:** Comparing two columns of the PV dataset [60] with our implemented v-plot visualization [60]. **Right:** The default v-plot configuration when opening the online v-plot designer, which comes with a test dataset loaded [8].

The version implemented in the course of this thesis used almost the same presets, with the difference being, that if no distribution shape was shown, the used colors per side referred to the color of the underlying category group or data channel assigned by the coloring legend. An example of it is shown in Figure 16 (left). The v-plot designer also offered the possibility to alter the color of every element used in the v-plot separately. The implemented version in Visplore by VRVis supported only altering the coloring through the color legend globally.

Another aspect that needed to be considered was how many layers, and which layers, were shown, to make efficient use of the coloring. The default colors that were assigned to each

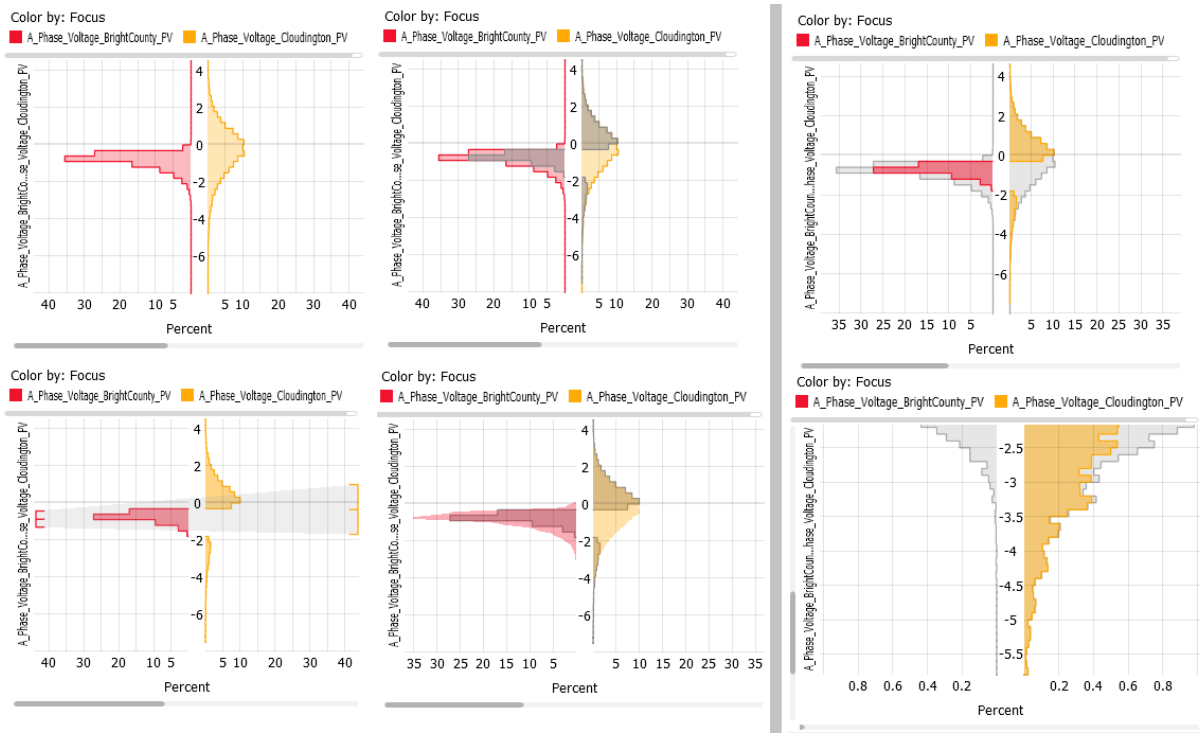


Figure 17: **Left:** Various color configurations that the implemented v-plot view in Visplore by VRVis used. Top left: Histogram layer visible. Top right: Histogram layer and direct difference encoding as histogram visible. Bottom left: Direct difference encoding as histogram and statistic measures with a connection as area visible. Bottom right: Direct difference encoding as histogram and distribution as shape visible. As can be seen, each configuration uses the category's color for either of its elements. The first draft of color assignments used the v-plot designer's [8] default configuration as reference [8]. **Right:** This color configuration was discarded. The reason for that being, that it could not be ensured that there is always a colored element visible on each v-plot side. As seen on the bottom, only one side displays color anymore, which is not a problem for single v-plots but gets problematic when showing a matrix arrangement.

v-plot layer (see Figure 6) were: *mirrored histogram layer:* category color, *direct difference encoding:* dark grey, *distribution shape:* category color, *statistic measures:* category color and light grey for connection. It had to be taken into account how much screen space was available for each visible v-plot when multiple plots were visualized as a v-plot matrix. Due to the fact that the individual labelling per v-plot was left out when the available screen space per plot undercut a certain threshold, it had to be assured that v-plots could be unambiguously associated with their underlying data (category or data channel) at any time. This was achieved by making sure that there was always a colored element visible in each plot. For example: for a v-plot matrix only having the direct difference encoding layer containing a histogram set to visible, the default coloring for the histogram layer changed from a dark grey to using the corresponding color from the coloring legend. This way, each v-plot side still clearly referenced the underlying data with the assistance of the coloring legend. Some coloring examples and the elaboration of a problematic coloring case are contained in Figure 17. It shows a case in which no color was visible on one side, therefore, it was not clearly referenceable when used in a data matrix anymore.

A screenshot of our final v-plot implementation in Visplore by VRVis can be seen in Figure 18. Another screenshot, displaying a v-plot matrix can be found in Appendix A.2.

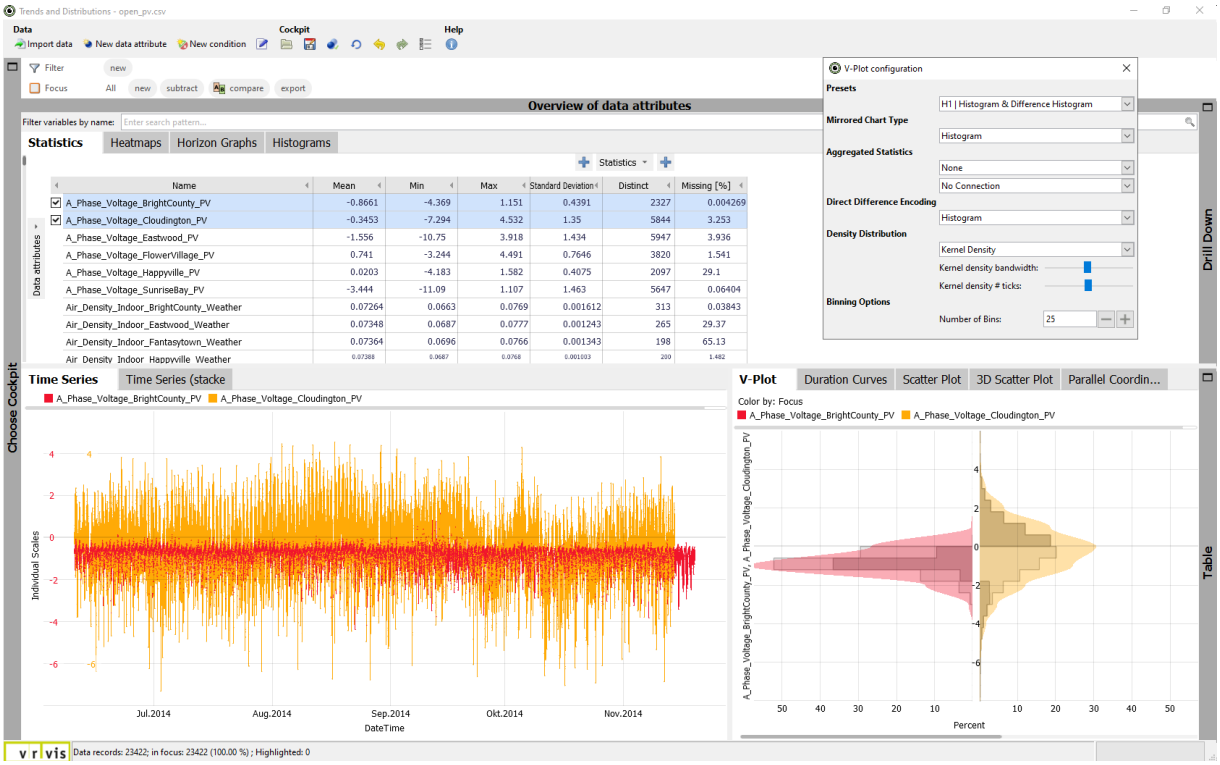


Figure 18: A screenshot from the v-plot visualization that was implemented in Visplore by VRVis. Here, it can be seen being used in a dashboard, that was arranged for the visualization and analysis of the trends and distributions of data.

4.2 Evaluation script implementation

With a working implementation of the v-plot matrix, the next step was to create the script for our approach on evaluating the scalability of this data visualization technique. For compatibility reasons and the possibility for rapid prototyping, we decided for a scripting solution which was written in Python 3 [64], using Jupyter Notebook [38] as computing platform. The free scikit-image [76] library for image processing in Python was used. Among many provided algorithms it offered useful functionality. This included: loading and saving images, as well as color and encoding conversions. The script also made use of the open-source NumPy project [26] which offered methods for numerical computations in Python. NumPy also offered easy-to-use functionality to modify images, which were internally represented as multi-dimensional arrays. This allowed for the quick transformation of images by means of mirroring or cropping them. Vector computations were also supported by NumPy, which came in very handy when using colors in a calculation. Last but not least, the script made use of the Matplotlib [31] to visualize intermediate outputs in the shape of labelled and arranged images.

4.2.1 Requirements

To be appropriate for our proposed evaluation approach and evaluation setting, the script had to meet certain requirements. These requirements specify the input that the script has to operates on and the challenges for the actual computation. The output created by the script is also specified in this Section.

Input

Image files served as input for this evaluation script. The file format of choice for this implementation was the Portable Network Graphic (PNG) file format. One reason for making this decision was the fact that Visplore by VRVIS already supported the export of a view as a PNG file. The image could either contain a single v-plot (matrix dimension 1) or a v-plot matrix.

Computation

The computation of the three proposed evaluation ratios described in Section 3.2 had to be applicable for the given input images. The script had to be able to detect the pixels as either data-ink, non data-ink or background correctly, with a minimal error rate. These assignments had to be made in order to calculate the three ratios: data-ink ratio, foreground-background ratio and the discriminability of v-plot sides as ratio. Not mandatory, but desirable, were some sort of intermediate results throughout the computation. They served for debugging purposes and allowed for supervision of the computations. Also, for the purpose of this thesis, they helped in visualizing the internal workings of the evaluation script.

4.2.2 Concepts for the computation logic

After loading the images of the v-plot matrices, it had to be detected whether a pixel contained data-ink, non data-ink or background. In the context of this thesis, we will refer to these three possibilities (data-ink, non data-ink or background) as *groups*. The differentiation into these three groups was needed to calculate the evaluation ratios, as explained in Section 3.2. The script determined a pixel's affinity to one of these groups solely by its color. To accomplish that, the colors of background, text and grid had to be known. With this information, pixels with any different color could be marked as data-ink. The colors used for text, grid lines, and background stayed consistent across the exported v-plots and could, therefore, be set to the default values. Due to the fact that the implemented v-plot visualization used transparency and smoothed lines, pixels could consist of ambiguous mixed colors, which were not classifiable through referencing the default values. This issue occurred between smooth drawn text and the background, the grid that was drawn transparently on top of the data, and when a statistic measures connection area was drawn on top of the vertical axis labels, which could, furthermore, intersect with smoothed text. These mentioned cases are visualized in Figure 19.

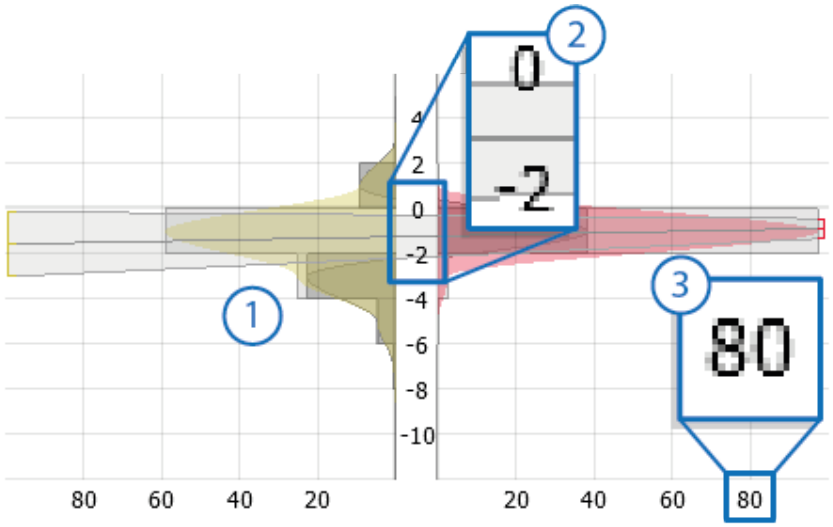


Figure 19: Emphasizing challenging areas when categorizing pixels by color. They emerged through the usage of transparency and smoothing. **1)** Grid lines are drawn with transparency above the data. **2)** Statistic measures connection area and lines are partially drawn on top of the axis labels. **3)** Axis labels and text in general are automatically drawn using some sort of smoothing.

To handle these circumstances, we introduced priorities to the three groups. A priority defined which affinity a pixel should be assigned to when a color was detected that did not clearly refer to one group and, therefore, indicated that this pixel contained the colors of multiple groups. When a pixel with ambiguous ink was detected, it was checked, to what groups the mixed color of the pixel referred to. These groups were then checked for their priority and the pixel was assigned the affinity to the group with the highest priority. An example: data-ink and non-data ink were detected in a pixel. The group representing data-ink had the priority 1, while the non

data-ink group had the priority 2. Therefore, the pixel was assigned to the data-ink group. The priority behaved inversely to its numeric value, which ranged from 1 to 3. Meaning that the group with a priority of 1 had the highest precedence and the group of priority 3, the lowest. In the context of this script, we used the following naming conventions for the groups and assigned them the following priorities:

- ***Data-ink group* (DI):** Contained the data-ink. Had the highest priority with *priority 1*. This meant that whenever data-ink was detected predominantly in a pixel, its other shares were ignored and the pixel was marked DI.
- ***Non data-ink group* (NDI):** Contained non data-ink, which either represented labels or the grid. Had the second highest priority with *priority 2*.
- ***Background group* (BG):** Contained all pixels that did not have any ink, meaning they were drawn using the background color. Had the lowest priority with *priority 3*, meaning that it contained everything that could neither be assigned to DI nor NDI.

To compute the introduced ratios, the pixels of the visualization image had to be unambiguously assigned to either of the groups. With the introduced prioritized groups, we established a guideline of how to assign the pixels of an input image to these groups.

Each pixel needed to be checked for its affinity to every available group and was then assigned to the found group with the highest priority. This seemed unnecessary expensive, as we already knew that there were certain areas in the image that could only contain certain types of ink, or only background. To avoid unnecessary computations, the image was split into *regions*. A region was defined through the x and y coordinate of its starting pixel in screen-space and its width and height in pixels. Most importantly, each region stored an array, containing a reference of all the groups that a pixel in this region might belong to. The regions themselves could not be detected automatically but had to be declared manually for each v-plot input image. This definition of regions enabled a simple opportunity to speed up the computation. For example: A region that was marked to contain only non data-ink (NDI) did not have to be checked for data-ink. Further, the definition of regions was necessary to handle certain areas of the input image. The reason was, that every pixel that contained a color other than one of the known colors for the grid, labels or background, was assigned to the data-ink group (DI), as explained at the beginning of this section. This led to falsely assigning pixels that belonged to e.g., the smoothed edges of text (as depicted in Figure 19, 3), to the data-ink group (DI) instead of the non data-ink group (NDI). This could be prevented, by explicitly marking a region to only contain NDI and BG, as we knew that no data-ink was present in this region. Therefore, a pixel of this region that contained a mixed color, would not automatically be assigned to the data-ink group anymore. Instead it was tested for its affinity to either the non data-ink group or the background.

4.2.3 Implementation

The actual implementation of the script consisted of two major parts. The objective of part one was to generate an evaluation result for a given image that contained only a single v-plot. Subsequently, the implementation for square v-plot matrices of arbitrary size was realized reusing the script for single plots. The workings of the script could be logically arranged into the three stages: image processing, pixel affinity detection to one of the three groups, and the actual ratio computations. Before elaborating these stages further, the data structure used to realize the script's components is presented. The complete script can be found in Appendix A.3.

Data structure

The concepts for the computation logic introduced in Section 4.2.2 as well as the computation stages were implemented as part of the evaluation script as either data structure or function.
Some of the *groups* defined in Section 4.2.2 were represented through global strings. The main purpose of these groups encoded as strings was to decide which approach should be taken when detecting the pixels' affinity of a region to one of the three groups. The groups that were represented in the script were the data-ink group (DI) and the non data-ink group (NDI). For further enhancing computation times, a sub-group was derived from the non data-ink group, that described pixels that could only contain labels but not the grid. This group was titled *Labels ink* (LI). This group was handled with the same priority 2 as the NDI group. This was possible because we could foreclose that a region would contain labels and grid pixels. The background was not represented as string explicitly, as it could be contained by every region.
The *Region* class stored the information of an image region as discussed in Section 4.2.2. It was defined through its origin point's x and y coordinate in screen-space relative to the image it belonged to. It further stored the region's width and height in pixels. A Region class instance further stored an array that contained the possible groups (DI, NDI, LI) other than the BG, that a pixel in this area could belong to. The BG did not need to be stored in this array explicitly, as every region could contain background pixels.
The *EncodedImageRegion* class was specifically designed to store the regions of our v-plot implementation. It explicitly stored the six regions that a v-plot could be divided into for the purpose of this evaluation script. These six regions entailed: the left side of the v-plot, the right side of the v-plot, the bottom legend and the left legend. Further optional regions, which were not shown on all v-plot configurations, were the padding on the right of the v-plot and the middle legend. The image regions are illustrated in Figure 20 on the left. This class also held the functionality to determine the width and height of a v-plot.

The definitions for the groups as strings, the *Regions* and the *EncodedImageRegions* were conveniently declared in the global scope of the file. This was handy when accessing it from either the computation class for the evaluation of a single plot or a matrix.
The computation logic of the script was encapsulated in the *VisualizationEvaluation* class. For

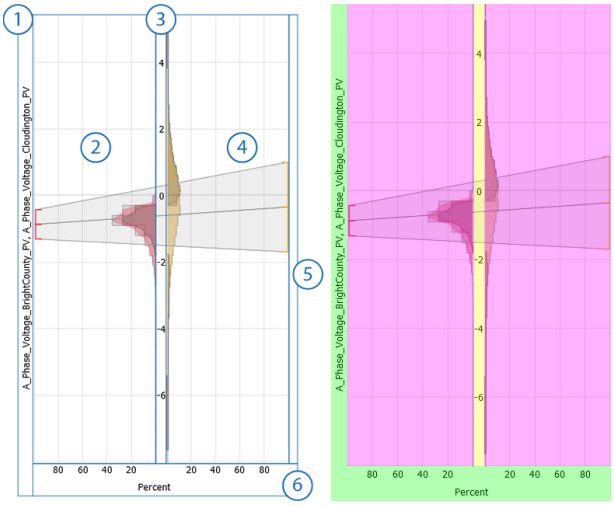


Figure 20: Defined regions of a v-plot image. V-plot image is taken from the input data set. **Left:** The image can be split into the following six regions: **1)** left legend, **2)** left plot side, **3)** middle legend, **4)** right plot side, **5)** right padding, **6)** bottom legend. **Right:** Image of a single v-plot with the encoded image regions as colored overlay generated by the script. The coloring represents which groups were assigned to the regions. Green represents a region marked as LI. Pink being a mixture of DI(red) and NDI(blue) and the yellow region marks a region assigned DI(red) and LI(green).

its instantiation, an image, as well as its according encoded image regions had to be passed. The specification of colors for the grid, background and the data colors were mandatory. If non were given, the default colors for each of these components was used. Note that the explicitly given data colors were only concerned with the color of the data covering the middle legend through a statistic measures connection. All other colors used for drawing data did not have to be specified manually. They were deduced by the script through an elimination process, which is explained in more detail later in this chapter. The *VisualizationEvaluation* kept track of the counted pixels per group and stored a visual representation per group in the form of an image. It further stored parameters and contained functions that were needed for the computation of the evaluation ratios. These functions containing the computation logic are the topic of the following subsections. Last but not least, the *VisualizationEvaluation* supplied various functions to display and save intermediate results.

The *MatrixEvaluation* class reused the computation logic introduced with the *VisualizationEvaluation* and supplied further functionality to reuse it for v-plot matrices. This included extracting single v-plots from the matrix, based on the manually specified dimensionality of the matrix. Further, the foreground-background ratio calculations were changed slightly, as it was not calculated per v-plot of the matrix, but for the matrix as a whole. The other ratios were computed per v-plot, calculating their arithmetic mean to gain one single end-result per matrix.

Preparations

Before entering the actual computation stage, an image had to be loaded as input. This was accomplished through functionality of the scikit-image library. Its image read function loaded the specified image as a three-dimensional NumPy array, where the first dimension represented the y-coordinate, the second dimension was the image's x-coordinate, and the pixel color was encoded in the third dimension as an RGB vector. The image regions had to be specified manually. This had to be done for each image size and matrix dimensionality separately. Once specified, the determined regions could then be reused with v-plot images of the same resolution but with different layer complexities and bin counts. It also had to be determined whether the colors representing the grid were among the *VisualizationEvaluations's* default grid colors. If any other colors appeared due to the usage of transparency when drawing the grid, these colors had to be explicitly passed as an array of colors. The data and background colors both stayed consistent over all of our input images. Therefore, their default colors were used throughout.

Image processing

Based on the given input, which consisted of an image depicting the visualization to be evaluated, the encoded image regions, and the specified colors, the *VisualizationEvaluation* class could be instantiated. The given colors, input image and regions were stored within the instance. The pixel counts per group were initialized with zeroes. The passed image was then processed for further computation. First, the width and height of the image in pixels were stored as parameters. Further, three empty images of the size of the input image were created. These images corresponded to the three groups: DI, NDI and BG. Before any calculations were done, the images contained only black pixels. To make sure that the manually passed image regions corresponded correctly to the v-plot, a map was created that marked the given regions of the image with the color belonging to its specified group. Encoded image regions and their denoted colors were: DI was red, LI was green, NDI was blue and BG was black. To create this map, the initially black image, in the size of the input image, was iterated over the specified regions and each pixel's color was added the color(s) of their corresponding region. Due to the fact that a region could belong to multiple groups, mixtures of the above mentioned colors could appear. Those colors were created through additive mixing, which allowed for tracing back to their related groups easily, as the most complex case was a region where all groups were possible, which resulted in white pixels. The resulting image was then blended with the original image and optionally output as file and as image preview within the programming environment (see Figure 20 on the right). In case the regions were declared wrongly, for example because regions overlapped or some part of the image was not covered by any region, the script stopped executing. In case this stage was passed successfully, the script continued on to the next stage.

Pixel affinity detection

The goal of this stage was to assign all pixels of the input image to one of the three groups as specified in Section 4.2.2. This was accomplished by comparing pixel colors with the known colors of different components (text, grid, data or background). As mentioned in Section 4.2.2, a pixel did not always match the specified colors exactly. Therefore, functionality was introduced to determine which group a pixel was more likely to belong to, based on its ambiguous color.
One region, where mixed colors appeared, were the adjacent pixels to text. These pixels' colors were a mixture of the text color and the background color. A function was introduced, that determined whether the pixel belonged to the text based on its color's euclidean distance to the text color. A tweak-able parameter had to be given to the function, declaring the maximum distance a pixel might have to the text color to still be counted as text pixel. The distance value was given as percentage of the maximum distance that two colors in the RGB color space could have. This distance parameter was tweaked so that the pixels that were detected as label, closely resembled how the text would be perceived by the viewer. With the distance parameter set to a value of 0.8, satisfying results were achieved. As the size of the letters stayed consistent over different v-plot sizes this configuration worked for all v-plots of the input images. The allocation of text pixels to either NDI or BG and the different results when tweaking the distance parameter is visualized in Figure 21.

Another function that was implemented and worked likewise, checked for the similarity to the data color. As mentioned before, data color was usually recognized through a process of elimination, meaning that every pixel that did neither belong to NDI nor to BG, had to belong to the DI group. The middle legend region was the exception, as can be seen in Figure 19. This was the only region of a v-plot where DI and LI collided. Still, the priorities of the groups defined, that a pixel which represents data among others must be assigned to DI. In this case, pixels of the statistic measures connection clearly contained data but were drawn in the text color, as they also showed the axis ticks. Therefore, only checking for color similarity to the text was not sufficient in this case, as it would have wrongly marked the pixel as NDI. Instead, this case was handled as follows: First it was checked whether the DI marked region was also marked LI. Through a process of elimination, this could only mean that this was the middle legend region. If so, it was checked whether the current pixel was text. If the current pixel was text, its surrounding pixels had to be checked. The pixels of a 5x3 region, with the center being the current pixel, were checked for pixels matching a data color. If any match was found, the current pixel was assumed to be covered up by data and therefore, assigned to the DI. The region size covered by the check was deduced through experimentation, where 2 pixels in both directions on the x-axis and also 1 pixel in each direction on the y-axis (resulting in a 5x3 rectangle) yielded the best results, as can be seen in Figure 22. The number of adjacent pixels that needed to be found containing data color was also varied, with the outcome that finding just one bordering data pixel is enough to mark the current pixel DI.

Having explained the functionality to determine a pixel's affinity to one of the groups even for ambiguous colors, the iteration logic to detect pixel affinities can be explained next. The detec-

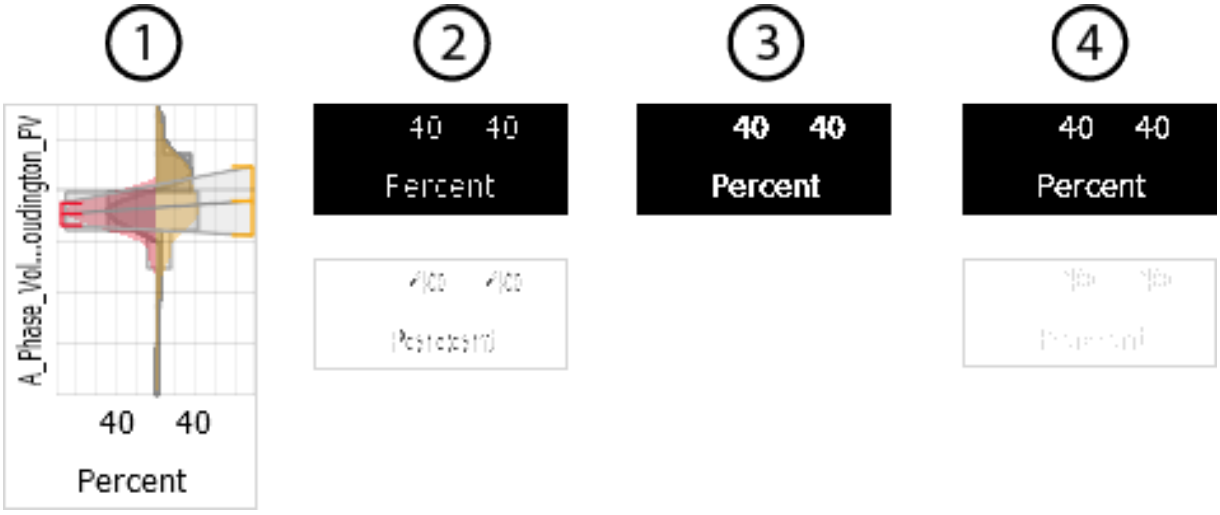


Figure 21: In the upper row on the right are several images containing only black and white pixels. White pixels encode pixels recognized as text. Below them are extracts of the original image, showing the remaining pixels that are counted to BG. **1)** The original input image, that contains text in the form of labels. To demonstrate how text is detected when changing the distance parameter, the bottom legend of the v-plot is looked at. **2)** The text as seen by the script when only checking for pixels that match the given text color exactly (distance parameter = 0.0). This leads to broken strokes in the letters. **3)** Shows what happens when everything that is not background color is counted as text (distance parameter = 1.0). This leads to very fat letters that do not match the viewers perception. **4)** A distance parameter of 0.8 yielded satisfying results, sorting darker pixels to the NDI and the very light pixels surrounding the text to the BG.

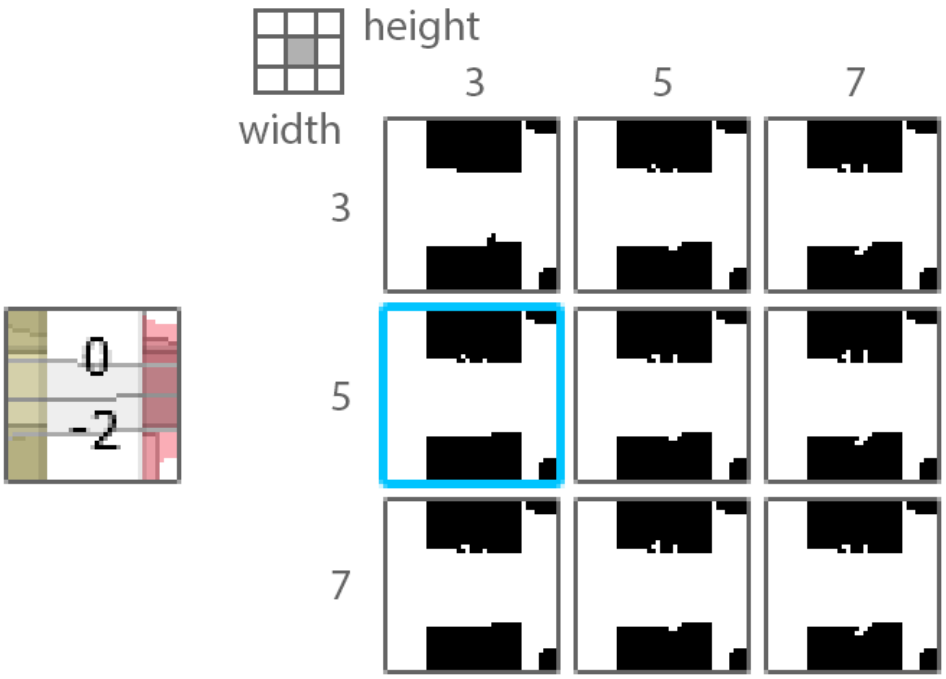


Figure 22: The classification of pixels in a region where labels (LI) and data (DI) occupied the same pixel. **Left:** Cut-out of a v-plot showing a region where LI and DI occur. **Right:** Showing the results of varying how many adjacent pixels are checked for containing data colors in a matrix. The number of checked pixels in width increase on the y-axis of the matrix and respectively the number of checked pixels in height is displayed on the x-axis. The width and height are defined in respect to the currently checked pixel, which lies in the center of the resulting rectangle. The chosen shape for the rectangle that is used in the evaluation is 5x3, which lead to the least errors among all tested cases.

tion of pixel affinities was done per region, where the order in that the regions were traversed was arbitrary. The encoded image regions were iterated in an outer loop. Within the inner loop the pixels of the region were traversed going from left to right and from top to bottom. For each pixel, a varying number of different checks were made to identify its affinity. The following checks were performed per pixel, in the following order:

1. The pixel was checked against the background color. If they matched, no further checks had to be evaluated and the pixel could be marked as BG. This handling could be applied because the known background color only appeared where background was actually present.

2. If the pixel could not be assigned to BG, it was checked whether the current region was marked DI and therefore, could contain data-ink. If so, it was proceeded as follows:
   - It was checked whether the region could contain labels (LI). If so, the region under observation could only be the middle legend, which could contain parts of DI, LI and BG. To find out to which of the three groups the pixel belonged to, it was checked whether its color was most similar to the text color and not obscured by data, or if its color was closer to the data color. If the pixel's color was to neither text nor data similar enough, the pixel was assigned to BG. These functions used to determine color similarity were previously explained in more detail.
   - When the region was not marked LI additionally, it was checked whether it could contain NDI. If so, the pixel's color was compared to the grid colors. If the colors matched, the pixel was assigned to NDI.
   - Otherwise, the pixel belonged to DI.

3. If the region was not marked DI, it was checked whether it was marked NDI and if so, it was checked if one of the grid colors matched the current pixel's color. If they matched, the pixel was marked NDI.

4. If the pixel could not be assigned to any of the layers, it was checked whether the region could contain LI. If so, the pixel's color was compared to the text color. In case they were similar enough, the pixel was denoted NDI.

The above-mentioned order was mainly reasoned by the priority of the groups. According to these priorities, whenever a pixel was identified as data, even when it overlapped with grid or text, it was assigned to DI. Therefore, the affinity to DI was checked first, and NDI afterwards. Entailing the priorities, the affinity to the BG should have been examined last but was executed in the beginning. This exception was made, as it was a very easy check that could save some computation time.
After this iteration process, all pixels were assigned to either of the three groups (DI, NDI or BG). In the development of the script, some pixels belonging to the data were wrongly detected as grid pixels. This happened because some components of the data, for example the direct

difference encoding and the statistic measures connections, were drawn using some nuance of grey. As we know, the grid was also drawn using layering of transparent grey tones. To handle this circumstance, functionality was introduced to reduce the number of wrongly detected grid pixels. This entailed iterating over the regions marked as DI and NDI again, as falsely detected grid pixels could otherwise only belong to DI. The surroundings of every pixel that was assigned to the NDI was then examined once more. If a pixel belonged to the grid, either the top, right, bottom or left adjacent pixel had to also belong to the grid. If none of them were assigned to the NDI, it was assumed that the pixel was detected falsely. It was then re-assigned to the DI group. This cleaning method only worked for *'floating'* pixels, that had no adjacent grid pixels. Unfortunately, after cleaning up these wrongly detected pixels as explained, there still remained pixels that were wrongly marked NDI. This happened at the borders of data components, where adjacent pixels also contained grid colors. This error rate was overall quite low throughout the evaluated images and was therefore accepted in the scope of this evaluation script, as it hardly impacted the calculated ratios. The workings of the grid cleaning is demonstrated by means of an example in Figure 23.

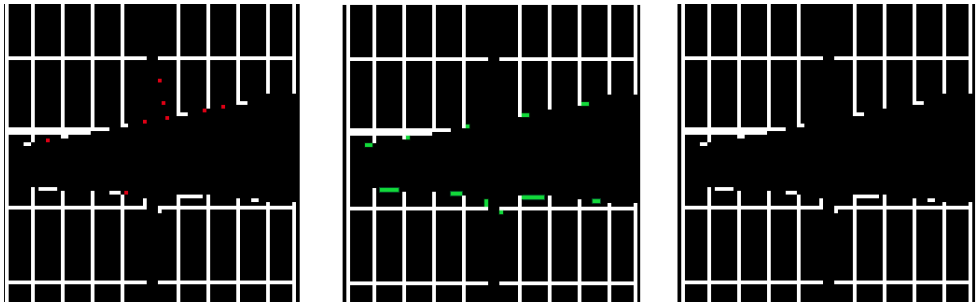


Figure 23: The effect of the grid cleaning is visualized in this image based on a cropped area from the detected NDI image of a v-plot. **Left:** Wrongly detected grid pixels that were removed by the script are marked red. **Middle:** Green marks pixels that were wrongly detected but could not be removed by the script. **Right:** The detected NDI pixels when removing the wrongly detected *'floating'* grid pixels (red).

With the completed execution of this phase, all pixels were unambiguously assigned to one of the groups, therefore, divided into data-ink, non-data ink and background. Further, along with their assignment, the pixels per group were counted. This was the basis for the last stage, the ratio computations. To check for the correctness of the pixel assignments, the script offered functionality to show and save an image per group. These images encoded the assigned pixels per group, which were visualized through white color in an otherwise black image. The intermediate output of this stage is shown in Figure 24. A mandatory check for whether all pixels of the input image were assigned to a group concluded this stage.

Ratio computations

Finally, after executing the image processing and the pixel affinity detection stage, all values needed for the ratio computations were available and the script could continue with the ded-

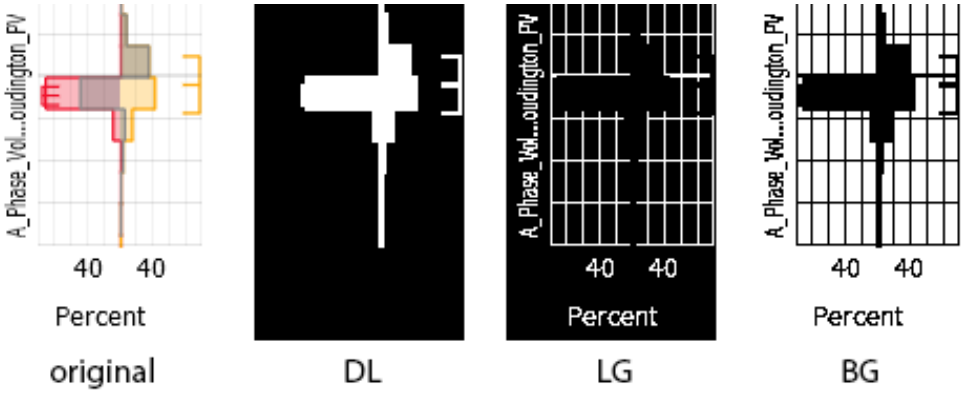


Figure 24: Group affinity as determined and visualized by the script as optional intermediate output. White pixels represent pixels assigned to the group. Black pixels depict pixels belonging to other groups. **Left:** the original v-plot input image. **Middle left:** data-ink group (DI, priority 1). **Middle right:** non data-ink group containing labels and grid (NDI, priority 2). **Right:** background group (BG, priority 3).

icated ratio computations stage. The ratio calculations worked on the basis of the counted pixels per group. The ratios themselves were explained in section 3.2. The computations for the foreground-background ratio and the data-ink ratio were pretty straight-forward and are explained subsequently:

Listing 4.1: Data-ink ratio calculation

```
#data-ink ratio
inkCount = DIcount + NDIcount
if inkCount > 0:
   dataPercent = DIcount / inkCount * 100.0
   nonDataPercent = NDIcount / inkCount * 100.0
else:
   dataPercent = -1.0 #no ink present in the plot
   nonDataPercent = -1.0
```

Listing 4.1 shows a code snippet from the script for calculating the data-ink ratio. It was slightly adapted for better readability. The data-ink ratio was calculated as discussed in Section 3.2. The script also handled the exceptional case where no ink was present in the plot, which happened when very little screen-space was available. The data-ink ratio was then set to a value of -1.

Listing 4.2: Foreground-background ratio calculation

```
#foreground-background ratio
foregroundPercent = (DIcount + NDIcount) / totalPixelCount * 100.0
backgroundPercent = BGcount / totalPixelCount * 100.0
```

The code in listing 4.2 shows the calculation for the foreground-background ratio. Like in the

data-ink ratio computation, its reciprocal value was also calculated, just in case we wanted to visualize or examine it.

Listing 4.3: plot-side discriminability as ratio

```
#plot-side discriminability ratio
diffImage = np.zeros((leftPlot.shape[0], rightPlot.shape[1], 3),
    dtype=np.uint8)
diffPixels = 0
equivPixels = 0
allPixels = leftPlot.shape[0] * leftPlot.shape[1]
#check for differences in plot side data-ink and write them to a new image
for w in range(leftPlot.shape[1]):
    for h in range(leftPlot.shape[0]):
        pixelL = leftPlot[h][w]
        pixelR = rightPlot[h][w]
        if np.array_equiv(pixelL, pixelR):
            equivPixels += 1
        else:
            diffPixels += 1
            diffImage[h][w] = [255, 255, 255]

diffPixelRatio = diffPixels / allPixels * 100
```

The discriminability ratio between v-plot sides could only be calculated for v-plot sides of the same size. If v-plot side dimensions matched, the ratio was calculated as described in listing 4.3. For debugging purposes, an empty image in the size of a v-plot side was generated. It encoded the found pixel differences through white pixels. As we know, the v-plot sides used the same axes, just with the x-axis mirrored. In order to make a meaningful comparison between plot sides, one side had to be mirrored so that their mapped spaces matched. The plot sides were then iterated pixel by pixel. The discriminability between plot sides was determined based on the previously denoted DI pixels. The grid pixels were ignored, as a slight offset between grids would have distorted the ratio. More differences would have been recognized through non-aligned grids, than a viewer would have actually perceived, as also discussed in Section 3.2. If the current pixels on both sides were either both DI or not DI, no difference was found and it proceeded with the next pixel. If two pixels were compared and one of them was marked DI while the other wasn't, a difference was found. The respective pixel was then marked in the discriminability image, as can be seen in Figure 25. Further, the difference counter was increased by one. When the whole plot sides had been iterated, the discriminability ratio could be computed as all required variables were known.

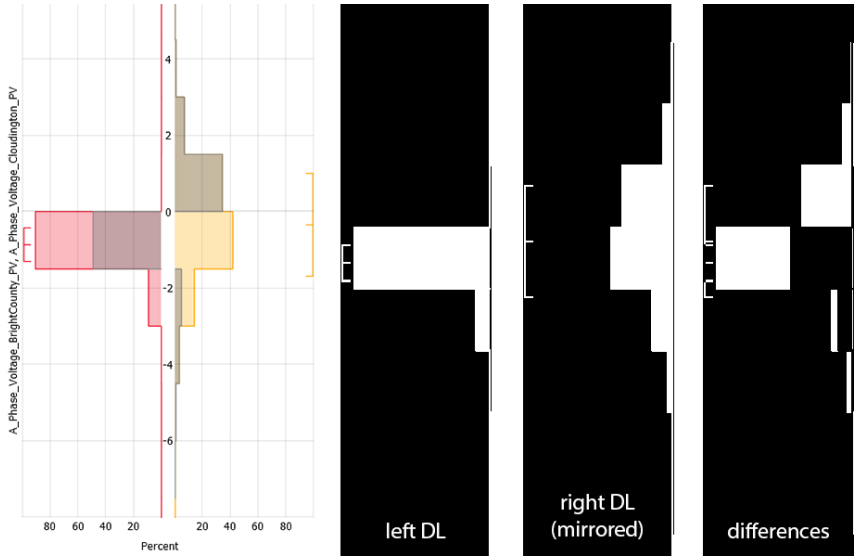


Figure 25: The original v-plot image (**left**) and the plot sides with their differences as detected by the script (**right**).

Matrix support

With the evaluation script working for a single v-plot, the first implementation step was finalized and the matrix support was realized next. The data structures introduced for the single plot evaluation were mostly reused. A new class, the *MatrixEvaluation* had to be created to make the iteration of an input image containing a v-plot matrix possible. The new class further offered functionality to calculate the ratios on a per-plot basis, except for the foreground-background ratio. This decision was made so that the labels of the left legend and the bottom legend were also taken into consideration for the evaluation. The *MatrixEvaluation* was created similarly to the *VisualizationEvaluation*. The *MatrixEvaluation* needed an additional dimensionality parameter passed on creation. This parameter told the instance the shape of the matrix in the input image. Another difference to the *VisualizationEvaluation* was, that the specified regions were used to describe the region of a single v-plot of the matrix. Only the left legend was defined for the whole image, as it only existed once in a v-plot matrix, and not per plot. Further, another parameter describing the padding on the right of the whole matrix was introduced. In this case, the existing right padding of the *encodedImageRegion* could not be overridden, as the v-plots of a matrix had a padding on the right themselves. The part of the bottom legend containing only the x-axis label was defined once for the whole matrix, while the bottom legend containing the axis ticks, was defined per v-plot. On instantiation of the *MatrixEvaluation*, a *VisualizationEvaluation* was created for each v-plot in the matrix. Therefore, the *encodedImageRegion*s from the *MatrixEvaluation* could be used, as they were already defined to match the single v-plots of the matrix. Further, the input images that were needed for the creation of the *VisualizationEvaluation*s were cropped from the image containing the matrix, to fit each v-plot. Finalizing the creation of the *MatrixEvaluation*, a check was performed to assure that every pixel of the

input image was either marked to belong to a v-plot, the right padding, the left legend, or the bottom legend. The functionality to detect a pixel's affinity was then performed through the *VisualizationEvaluation* per v-plot. Additional functionality that the *MatrixEvaluation* supplied, was to iterate multiple v-plots of a matrix, as well as the left and bottom legend and to skip empty spaces, which occurred due to the v-plots only existing in the upper left of the matrix diagonal. The pixels of these empty spaces were marked as BG. After the pixel affinity detection was completed, the three groups with their assigned pixels could be viewed as intermediate output (see Figure 26). The ratios for the v-plot matrix were computed reusing the functions from the

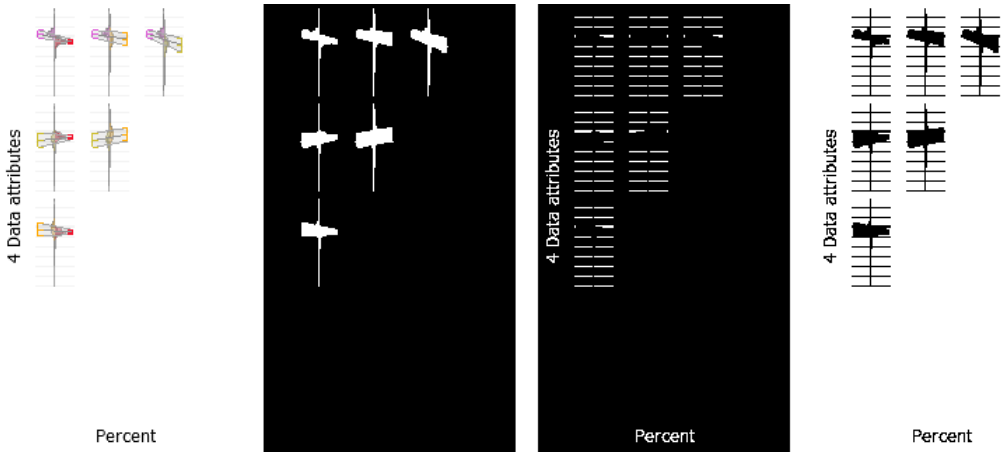


Figure 26: From left to right: The original v-plot image, map containing the DI marked as white pixels, map with the NDI and map with the BG as detected by the script.

VisualizationEvaluation class. The data-ink ratio was computed per v-plot and the arithmetic mean was used as result. The regions containing bottom legend and the left legend of the matrix were ignored in this case. The discriminability ratio between plot sides was also calculated per v-plot, using their arithmetic mean as end result. The foreground-background ratio was computed for the whole v-plot matrix image, including the bottom and left legend as well as the empty space of the lower right half of the matrix.

Executing the evaluation script for a v-plot image was accomplished as demonstrated in the following example in listing 4.4.

Listing 4.4: running the evaluation script

```
#run evaluation for a single v-plot
vplot = io.imread('../images/single/c0 - shapes/single_100x160_10bin.png')
regions = encodedImageRegion(Region(20, 44, 40, 116, [DI, NDI]), #left side
                             Region(60, 44, 40, 116, [DI, NDI]), #right side
                             Region(0, 0, 100, 44, [LI]), #bottom legend
                             Region(0, 44, 20, 116, [LI]), #left legend
                             Region(0, 0, 0, 0, [])) #right paadding
gridColors = [[243, 243, 243], [218, 218, 218], [229, 229, 229]]
```

```
evaluation = VisualizationEvaluation(vplot, regions, gridColors=gridColors)
evaluation.showAreaEncoding() #optional
evaluation.countPixelsPerGroup()
evaluation.checkRegionCompleteness() #optional

#to print the results
evaluation.getCounts()   #counts per priority layer
evaluation.printDataToInkRatio() #data-ink ratio
evaluation.printFGBGRatio() #forground-background ratio
evaluation.printNumDifferences() #discriminability of plot sides
```

Executing the script for a v-plot matrix worked similar, with only few discussed differences as shown in listing 4.5.

Listing 4.5: runnning a matrix evaluation

```
vplot = io.imread('../images/multi/6/30bin/c2 -
    all_layers/multi6_180x320_30bin.png')
regions = encodedImageRegion(Region(0, 2, 15, 66, [DI, NDI]), #left
                      Region(15, 2, 15, 66, [DI, NDI]), #right
                      Region(0, 0, 39, 2, [LI]), #bottom legend
                      Region(0, 0, 25, 272, [LI]), #left legend -> not used
                      Region(30, 2, 9, 66, [])) #right padding
#legends of whole image
leftLegend = encodedImageRegion(leftLegend=Region(0, 0, 25, 272, [LI]))
bottomRegion = encodedImageRegion(bottomLegend=Region(0, 0, 180, 48, [LI]))
gridColors = [[244, 244, 244]]

evaluation = MatrixEvaluation(vplot, regions, leftLegend, bottomRegion,
    dimPlots=4, gridColors=gridColors)
evaluation.defaultRightPadding = 8 #of whole image
evaluation.showAreaEncoding() #optional
evaluation.countPixelsPerGroup()
evaluation.checkRegionCompleteness() #optional

#to print the results
evaluation.getCounts()
evaluation.printDataToInkRatio()
evaluation.printFGBGRatio()
evaluation.printNumDifferences()
```

For the evaluation, various files were created, that contained the evaluation execution calls for a subset of the v-plot input images. For further automation, another short Python script was written, that called these scripts and executed them one after another (see listing 4.6).

Listing 4.6: run multiple evaluations one after another

```
import os
directory = "./"
for file in os.listdir(directory):
   if file.endswith(".ipynb"):
      #filter which evaluations to run (single or matrix of dimensionality)
      if "single" in file:
         name = directory + file
         print(name) #optional: to structure output
         %run $name
```

5 Evaluation

In our implementation in Visplore by VRVis, the v-plot view was usually shown as part of a dashboard, that contained many other views. This led to the circumstance that there was only a limited amount of screen-space available for displaying the v-plots. Therefore, we needed to determine the minimal amount of screen-space, where v-plot matrices would still be readable. The main focus for this evaluation lay on the scalability of the v-plot matrix in relation to the given screen-space in pixels. Further factors we wanted to test for, that might influence the scalability, were matrix dimensionality, the number of bins, and the complexity of the used v-plot layer configuration. Factors that were not taken into consideration, which might affect the scalability, were user interactions and a variable information space.

5.0.1 Evaluation setting

To cover a variety of different configurations of the v-plot visualization, the evaluation was performed on multiple v-plots with the following settings:

- **Screen-space:** The plots were evaluated on various screen-spaces. The resolutions were based on the screen sizes for responsive designs, as defined by Bose [12]. We further added some smaller resolutions. The used sizes in *'pixel (px)'* were (width x height): 100x160px, 180x320px, 360x640px, 414x896px and 1366x768px. Examples for resulting v-plots of that size can be seen in Figure 27.
- **Matrix dimensionality:** Because the screen-space per v-plot varied strongly with the number of v-plots that were displayed in the view, the evaluation was performed on a single v-plot as well as on v-plot matrices of various dimensionality. Those dimensions were (with their effectively shown v-plots): 3x3 (3 plots), 4x4 (6 plots), 5x5 (10 plots) and

6x6 (15 plots). The current matrix representation only showed v-plots in above the matrix diagonal, to prevent displaying a v-plot repeatedly.

- **V-Plot layer configurations/complexity:** Because different v-plot layer configurations used up varying amounts of screen-space, the different complexities were taken into account for the scalability evaluation.
  Three different complexities of v-plot layer configurations were used for the evaluation:
  - **"Complexity 0" (c0):** Only displayed the distribution layer as shape-based distribution. All other layers were not shown.
  - **"Complexity 1" (c1):** Displayed the histogram with the direct difference encoding as histogram and statistic measures (standard deviation and mean) without a connection.
  - **"Complexity 2" (c2):** Displayed all layers of the plot. This included a histogram and the direct difference encoding being represented as both: histogram and shape. Further the distribution was shown as shape-based distribution and the statistic measures (standard deviation and mean) were shown, using lines and area as connection.
- **Number of bins:** The number of used bins affected the detail of the representation of the data in the v-plots strongly. Therefore, the evaluation was performed with the following number of bins: 5, 10, 30, 50, 75.

Considering all these factors, the evaluation was performed on 5 resolutions x 5 dimensions x 3 complexities x 5 bin counts = 375 different images of v-plots in total.

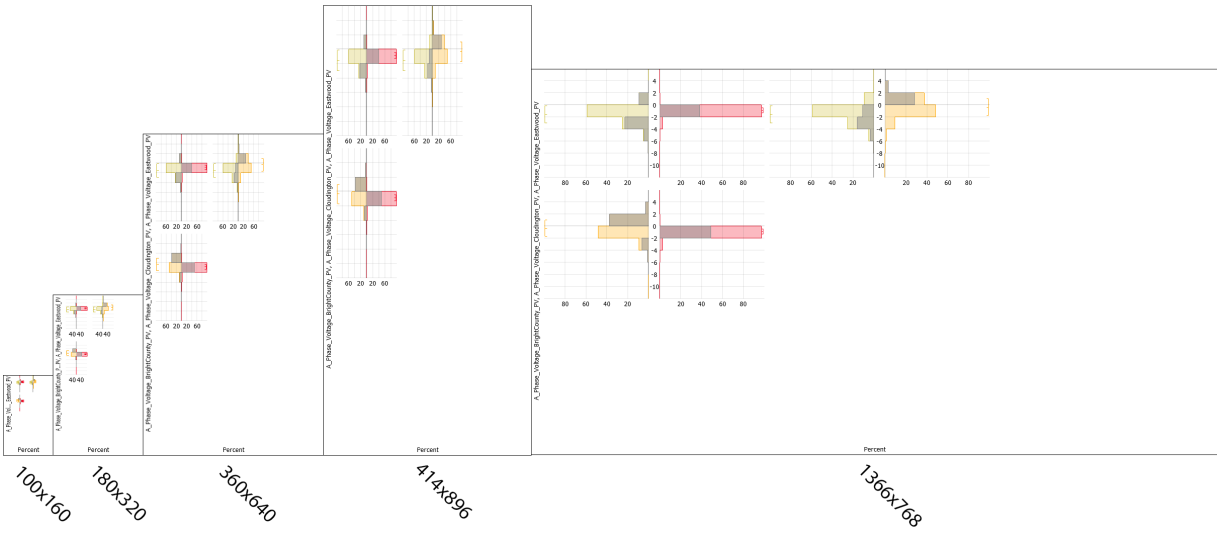


Figure 27: The resolutions used to render v-plots. Visualized side by side and labelled, measurements are in pixel. Images are an excerpt of the exported images used for the evaluation. They show a matrix of dimensionality 3, layer complexity c1 and use 10 bins.

Input data

The images to be used for the evaluation were exported as Portable Network Graphics (PNG) images from Visplore by VRVis. The software already provides an export function for PNG images, including several configuration options for the view. This allowed for creating a view of a specified size, as well as hiding the coloring legend. This was useful as the images had to be exported manually. The exported images showed single v-plots as well as square v-plot matrices of up to dimensionality 6. As broken down in the previous subsection, the input data for the evaluation was made up of 375 images in total. The data that was used to generate the v-plots was taken from the PV dataset, that was introduced in Section 2.2.3 and that was also used to demonstrate the v-plot use cases in Section 2.2.3. To generate the input images for the evaluation, various data channels containing information about the *a-Phase voltage* recorded at different locations were used. In the field of power generation, the term *a-phase voltage* is used to describe the voltage on the a-phase, which is one of the phases of a three-phase system. Such a three phase system is often used used for transmitting the generated power, in this case, of the solar plant. Therefore, the *a-Phase voltage* channels of the dataset inform us about the distribution of the current load on the related a-phase. Recordings at 6 different imaginary locations made it suitable for a meaningful comparison in the shape of a v-plot matrix of up to dimension 6x6, effectively displaying 15 v-plots at once. Using samples from the recorded *"A Phase Voltage"* at different locations made sure that the v-plots showed a meaningful comparison, using channels with a similar value space but with different enough distributions.

5.0.2 Results

The results of the v-plot matrix evaluation using the Python script are presented in the following subsections. All calculated ratios were stored in a table and can be found in the Appendix A.4. The results of the three different ratios are first presented and interpreted for themselves and then examined for correlations and their assumed effectiveness afterwards. For a better illustration of the results, the interactive data visualization software Tableau [71] was used to visualize them. Several visualizations can be found in the Appendix A.6.
In addition to interpreting the visualized test results, a two-tailed Student's t-test as statistical hypothesis test was performed on basis of the three computed ratios of various v-plots of different resolutions. Multiple tests were performed where the test set contained subsets of the computed ratios, classified by the respective resolutions. This resulted in a total of 5 test sets per ratio, each consisting of 75 samples. Multiple pairings of these test sets are examined in the following sections. Comparing the means of these test sets revealed whether predictions about the test set's ratio of one resolution could be made, based on the calculated ratios of the test set of another resolution. This was not possible when the natures of the test sets' ratios differed too strongly, which indicated bad scalability between the test sets. Different classifications used for the creation of the test sets helped indicate which v-plot configurations (dimensionality, com-

plexity, bin count) had a bigger effect on the overall scalability than others. The null hypothesis H_0, that was used throughout was generally formulated as follows:

"Varying the v-plot resolution leads to no significant change in the specified ratio."

This null hypothesis was slightly adapted for each tested ratio.
Hence the basis for the alternate hypothesis H_a was:

"Varying the v-plot resolution leads to a significant change in the specified ratio."

For all conducted t-tests it was defined, that a measured p-value below the significance level of $\alpha = 0.05$ rejected the null hypothesis H_0, therefore, proving the alternate hypothesis H_a that there was a significant change in the specified ratio between the tested resolutions. The Student's t-test was calculated using the respective *T.TEST()* [54] function that was supplied by the Microsoft Excel software [53]. It calculated and returned the p-value for the given test sets. To make sure that the performed t-tests had credibility, the variance of each test set was calculated first. If the variances of two test sets differed too strongly (as a rule of thumb we used $var_1 > 2var_2$ or $var_2 > 2var_1$ as stated by Hemmerich [29]), the parameters passed to the *T.TEST()* function were changed accordingly. This led to using a test method for heterogeneous variances instead, making sure that the test results stayed valid [29]. It is assumed that in this case a Welch's t-test was performed by the software, but this was not explicitly stated in the documentation [54]. This assumption was also supported through [46].

Data-ink ratio

For single v-plots, the data-ink ratio showed a relatively linear trend that increased over growing plot sizes. This could also be observed in v-plot matrices, but with the exception that matrices starting with dimensionality 3, partially had a 100:0 data-ink ratio on a 100x160 resolution. This was due to the fact that there were no grid lines visible and the grid labels were not taken into consideration when calculating the data-ink ratio for matrices. Further, matrices that had a dimensionality of 5 or higher, showed no non-data ink at on smaller resolutions than 360x640px. A higher complexity generally resulted in a higher data-ink ratio. This made sense, because with an increasing complexity, more v-plot layers were drawn. Changing the used bin counts did not affect the overall trend greatly. An increasing bin count led the data-ink ratio to generally decrease. This might have been due to the fact, that smaller bins used up less space and thus the distribution could be represented in more detail. This meant that distributions showed more peaks and lows, which were not visible with a lesser bin count.
The display of grid lines greatly affected the data-ink ratio. Whenever the size of a v-plot changed, it was re-calculated whether the horizontal and/or vertical grid lines should be visible and how many ticks should be used per axis. This led to sudden visible changes in the data-ink ratio. Mostly, when grid lines became visible, that were not shown in the preceding resolution. These sudden changes were especially visible in v-plots matrices of dimensionality 5 and 6 between the resolutions 360x640px and 1366x768px, as can be seen in Figure 28.

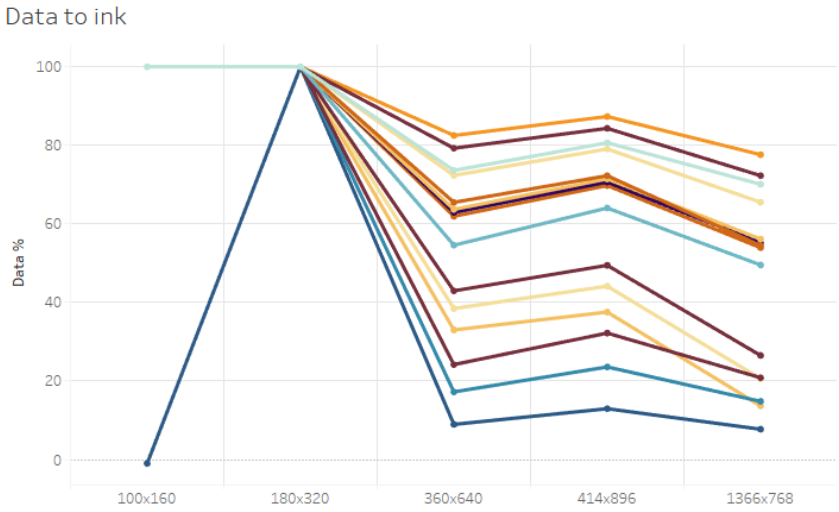


Figure 28: It shows the data-ink ratio over resolutions of the evaluated v-plot matrices with dimensionality 6. Each colored line represents one v-plot configuration (5 bin counts and 3 complexities resulting in 15 lines) of matrices with dimensionality 6. It shows a 100:0 data-ink ratio for all v-plots with resolutions of 100x160px and 180x320px. An extreme case occurred when using 75 bins and a complexity of c0. It can be seen, that when these configurations where used on a resolution of 100x160px, the result got negative (-1). Therefore, the data-ink ratio could not be calculated for this v-plot matrix as there was no ink available. This plot was generated using the Tableau software [71].

The resulted data-ink ratios spanned a range from 4.51% of data-ink to 100%, which strongly depended on the combination of the used v-plot configurations. Overall, for each dimensionality, the configuration resulting in the lowest data-ink ratio was using c0 and 75 bins, while the highest data-ink ratio occurred through the usage of c2 and 5 bins. Resulting in a difference of 74,15% on average between these two configurations that were used over different dimensions and resolutions. This showed that the number of bin counts and complexity strongly affected the data-ink ratio.

One recorded extreme case was a matrix of dimensionality 6 that used a resolution of 100x160px and complexity c0 (showing only a shape-based distribution). This resulted in no ink being drawn at all, as there were insufficient pixels to represent the value range with enough detail (also visible in Figure 28).

To perform the t-test for the data-ink ratio, the appropriate null hypothesis H_{0D} and the alternate hypothesis H_{aD} were formulated:

H_{0D}: *"Varying the v-plot resolution leads to no significant change in the data-ink ratio."*

H_{aD}: *"Varying the v-plot resolution leads to a significant change in the data-ink ratio"*

Looking at the resulting p-values of the t-tests, which can be seen in Figure 29, significant changes appeared involving the test sets that contained smaller resolutions like 100x160px and 180x320px. The null hypothesis H_{0D} was rejected between the resolutions (100x160px, 180x320px), (180x320px, 360x640px) and (100x160px, 360x640px), which indicated that the

scalability for these small resolutions is bad. Further significant changes between (100x160px, 1366x768px) and (180x320px, 1366x768px) supported this assumption. Comparisons of increasing resolutions starting with 360x640px supported H_{0D} and therefore, suggested good scalability.

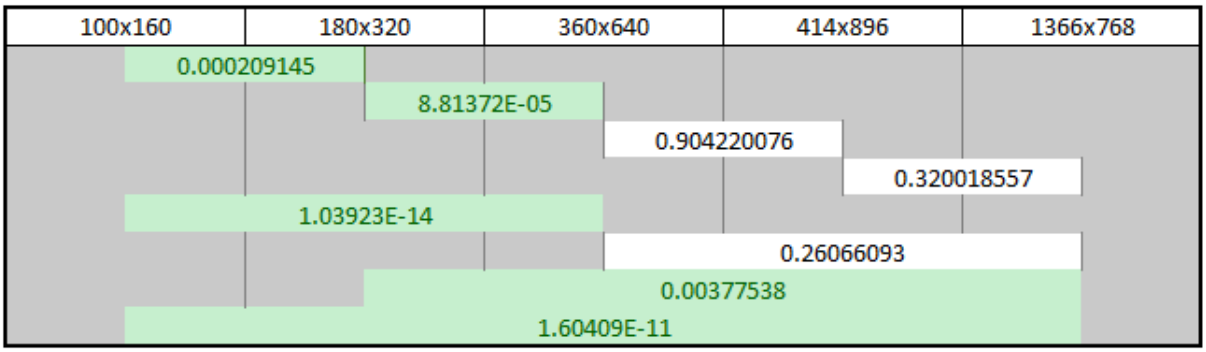


Figure 29: **T-test results for the data-ink ratio:** The p-values of the executed t-tests between different pairings of the test sets. The span of the cells containing the p-values indicate which test set pairing it belongs to. Meaning the left border of a cell marks the first test set used in the pairing and the right border of the cell resides in the column of the second test set. Green highlighted cells mark p-values smaller than the significance level $\alpha = 0.05$ and therefore, rejecting the null hypothesis H_{0D} and supporting the alternate hypothesis H_{aD}. The table was generated using Microsoft Excel [53].

Foreground-background ratio

The foreground-background ratio generally returned low percentages of pixels that contained foreground. More precisely, all images contained more background than foreground. Across all images, the measured foreground occupied spaces in the range of 0.9% up to 31.89% of the respective image. As opposed to the data-ink ratio, there was no case where the FG:BG ratio could not be calculated. Further, no extreme case like, e.g., a 100:0 ratio occurred. The overall trends of the FG:BG ratio stayed linear, although the slopes varied quite strongly when the dimensionality of the matrix was changed. Single v-plots showed a slightly decreasing linear trend over an increasing resolution. Matrices of dimensionality 3 and 4 showed a similar trend, but with a slightly increasing foreground-background ratio for smaller resolutions. Matrices of dimensionality 5 and 6 showed a similar trend but were very heavily influenced by the change of the grid display between resolutions, which led to more fluctuating values (an effect that was also discussed having an impact on the data-ink ratio). Varying the complexity mainly affected the value span of the resulting ratios but less the overall trend. A higher complexity generally led to a higher FG:BG ratio and vice versa. Increasing the bin count led to a decrease of the FG:BG ratio. Therefore, the dimensionality seemed to have had the biggest influence on the foreground-background ratio. The biggest differences were observed between single v-plots

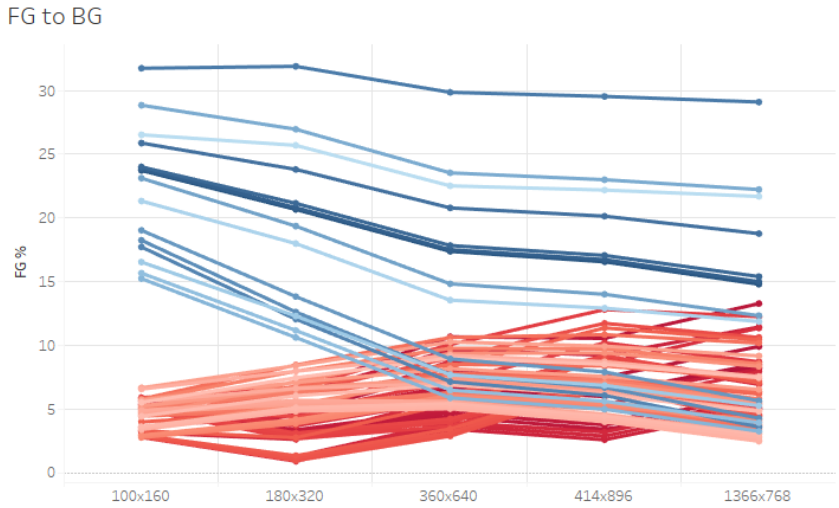


Figure 30: This plot was generated using the Tableau software [71]. It shows the trend of the FG:BG ratio over different resolutions. Blue colors encode all images of v-plot configurations containing a single v-plot. Nuances of red represent all v-plot matrices using a higher dimensionality (3, 4, 5 and 6).

100x160	180x320	360x640	414x896	1366x768
0.665652912				
	0.496869689			
		0.871273809		
			0.346244816	
0.885372113				
		0.440681293		
	0.979880332			
0.647400858				

$p < 0.05$

Figure 31: **T-test results for the foreground-background ratio:** The p-values of the executed t-tests between different pairings of the test sets. The span of the cells containing the p-values indicate which test set pairing it belongs to. Meaning the left border of a cell marks the first test set used in the pairing and the right border of the cell resides in the column of the second test set. Green highlighted cells would mark p-values smaller than $\alpha = 0.05$ and therefore, reject the null hypothesis H_{0F}. As can be seen, all t-tests performed for the FG:BG ratio supported the null hypothesis. The table was generated using Microsoft Excel [53].

(dimensionality 1) and all v-plot matrices of a higher dimensionality, as drastic changes of the slope and the value range appeared, as seen in Figure 30. Other than that, the FG:BG ratio indicated, that the v-plots scaled well across resolutions without any major information loss.
Similar to how it was done for the data-ink ratio, the hypotheses for the t-tests were adapted for the FG:BG ratio, where H_{0F} was the null hypothesis and H_{aF} was the alternate hypothesis:

H_{0F}*: "Varying the v-plot resolution leads to no significant change in the foreground-background ratio."*

H_{aF}*: "Varying the v-plot resolution leads to a significant change in the foreground-background ratio"*

A quick look at the table in Figure 31 reveals that all performed t-tests for the FG:BG ratio resulted in p-values above $\alpha = 0.05$. Therefore, all examined test set pairings supported the null hypothesis H_{0F}. Seeing that no significant changes were detected in the t-tests, suggested a good scalability over all tested v-plot resolutions.

Discriminability of v-plot sides as ratio

The discriminability ratio between v-plot sides was especially interesting as it represented how much of a difference between the v-plot sides would have been perceived by a viewer. Overall, different pixels between plot sides ranged from 0.92% to 27.02% of the total pixels of a v-plot side. Very low values indicated v-plots with low discriminability.
Looking at the discriminability ratio visualized, it can be seen that the difference between plot sides showed a very steady linear and slightly decreasing trend over increasing resolutions for images of single v-plots. V-Plot matrices showed similar results, but had slightly different trends for small resolutions. Those deviations increased with the used complexity. Complexity c0 resulted in still mostly linear trends. Complexities c1 and c2 showed a higher discriminability ratio between plot sides, especially on small resolutions. Bin counts including 30 and above led to higher discriminability ratios as well. Whereas a bin count of 5 and 10 showed much less of a deviation for small resolutions from the linear trend. An extreme case at dimensionality 6 and a resolution of 100x160px is also visible looking at the discriminability ratio. This case occurred because without any data-ink, the discriminability ratio could not be calculated.

Several t-tests were performed based on the measured differences between v-plot sides, to find out whether they lose their discriminability on certain resolutions. The respective hypotheses were formulated as follows:

H_{0S}*: "Varying the v-plot resolution leads to no significant change in the discriminability of v-plot sides as ratio."*

H_{aS}*: "Varying the v-plot resolution leads to a significant change in the discriminability of v-plot sides as ratio"*

Further, it was examined whether this loss could be linked to specific v-plot properties like bin count, dimensionality or complexity. First, the t-tests were performed on test sets classified only by the resolution. Their resulting p-values can be seen in Figure 32. As the highest res-

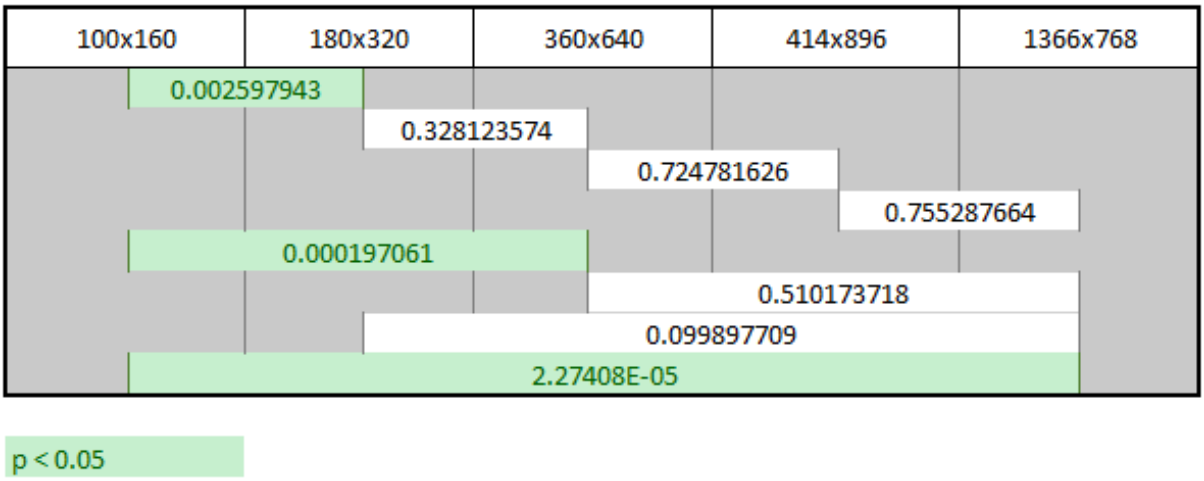


Figure 32: **T-test results for the discriminability ratio:** The p-values of the executed t-tests between different pairings of our test sets. The span of the cells containing the p-values indicate which test set pairing it belongs to. Meaning the left border of a cell marks the first test set used in the pairing and the right border of the cell resides in the column of the second test set. Green highlighted cells mark p-values smaller than $\alpha = 0.05$ and therefore, rejecting our null hypothesis H_{0S} and proving the alternate hypothesis H_{aS}. The table was generated using Microsoft Excel [53].

olution (1366x768px) of our test set contained the most screen-space per v-plot it had to be showing the most detail. A higher level of detail influenced the visible differences between plot sides. Therefore, we were especially interested in seeing whether H_{0S} was confirmed when comparing large and small resolutions. As we can see in the table in Figure 32, the p-values that resulted from the pairings *(180x320px, 1366x768px)*, *(360x640px, 1366x768px)* and *(414x896px, 1366x768px)* were above $\alpha = 0.05$, and therefore, supported the null hypothesis H_{0S}. Testing for *(100x160px, 1366x768px)* and *(100x160px, 360x640px)* both yielded p-values below $\alpha = 0.05$, disproving H_{0S} for these two cases. For these pairings, the alternate hypothesis H_{aS} was confirmed. From that we conclude that the significant changes measured between large and very small resolutions resulted in a major loss of detail in the v-plots, making only screen-space above 100x160 pixels recommendable for displaying v-plots. The p-value that resulted from *(100x160px, 180x320px)* also supported H_a, showing that in fact the difference between these smaller resolutions was significant.

Further t-tests regarding the discriminability ratio were performed on different subsets of the data. The null hypothesis and the alternate hypothesis stayed unaltered. What changed was the test set, which was adapted for each formulated question. The tables containing all p-values that were tested for, and the related test sets along with the information about their classification can be found in appendix A.5.

First, we generated new test sets through classification by dimensionality and examined the resulting p-values of various pairings. Classifying by dimensionality led to smaller test sets, consisting of 15 values per test set. The results of these test sets are summarized here while the complete test results can be found in Appendix A.5.1.

- Dimensionality 1 (single plot), 4 and 6: P-values of various pairings supported H_{0S}, that there was no significant difference between resolutions. This indicated that their discriminability scaled well throughout resolutions.
- Dimensionality 3 and 5: Showed significant differences involving parings with the smallest resolution: (100x160px, 180x320px), (100x160px, 360x640px) and (100x160px, 1366x768px). These test pairings showed a discriminability loss for smaller resolutions, similar to the test data without classification.

This indicated that significant changes alternated with the used dimensionality. Further, it suggested that there were certain matrix dimensions that scaled better than others. The complete test results for the classification by complexity can be found in appendix A.5.2. When examining the p-values for the test sets that were classified by the used complexity, no significant changes for c0 occurred, supporting H_{0S}. In both classifications, c1 and c2 very small p-values with pairings including the smallest resolution resulted. H_{aS} was supported for test sets using c1 and c2 for the resolution pairings (100x160px, 180x320px), (100x160px, 360x640px) and (100x160px, 1366x768px) therefore, indicating there were significant differences. Complexity c0 scaled well over all tested resolutions, while c1 and c2 showed similar scaling behaviour to the test sets without further classification. This indicated that the complexity played an important role in the discriminability between plot sides.
Last but not least, we examined the effect of the used bin count on the v-plot discriminability throughout different resolutions. Their complete test results can be found in appendix A.5.3. Our hypotheses H_{0S} and H_{aS} remained as they were but the test sets were created accordingly. Using 5 bins or 50 bins yielded overall p-values above $\alpha = 0.05$, supporting H_{0S}, that there were no significant changes. Pairings using 75 bins showed only significant differences when looking at (100x160px, 1366x768px). Using 10 bins yielded significant changes between test sets including the smallest resolution: (100x160px, 180x320px), (100x160px, 360x640px) and (100x160px, 1366x768px). Test sets using 30 bins showed less significant changes, only between (100x160px, 1366x768px). Overall, this showed that the bin count had no specific influence on the discriminability between v-plot sides. Many significant changes could be observed in pairings including the smallest tested resolution (100x160px).

5.0.3 Interpreting the ratios

To start off this chapter, the questions formulated in Section 3.2 can be answered as follows:

- ***How well do v-plots scale with respect to the available screen-space?***
  Observing the calculated ratios revealed a linear trend over the tested resolutions. Ex-

ceptions could be seen for small resolutions like 100x160px and in the case of matrices also for 180x320px. The conducted t-tests back up this assumptions as they revealed significant changes when the test sets involved these small resolutions.

- ***How do bin count, matrix dimensionality and layer configurations affect the scaling of v-plots?***
  Bin count: The data-ink ratio generally decreased with an increasing bin count. The decrease in the data-ink ratio was caused by the change in the visualization, displaying the data with higher granularity and therefore, uncovering more peaks and lows. The decreasing FG:BG ratio with an increasing bin count most likely also came from the fact that generally less ink was used when the data was visualized with more peaks and lows. The discriminability ratio generally showed higher values with a decreasing bin count and a steeper decrease over growing resolutions for higher bin counts. Summarizing, the resulting ratios indicated that if the bin count was chosen from a reasonable range, like it was for the v-plots of the test set, it was not a main factor that caused the linear trend (that indicated good scalability over resolutions) to deviate.
  Dimensionality: Matrices of high dimensionality showed no non-data ink on small resolutions. In an extreme case, that used dimensionality 6 and a resolution of 100x160px, showed no ink at all. Therefore, the discriminability ratio could not be calculated. The FG:BG ratio was strongly affected by whether a single v-plot or a matrix was shown. The dimensionality affected the discriminability ratio strongly, although there was no certain correlation describing this effect, as it seemed to be alternating with the number of used dimensions. Depending on the used dimensionality, many significant changes in the discriminability ratio or non at all could be seen between resolutions. Overall, it seemed that the higher the dimensionality, the larger the deviation from the linear trend. This could be seen having a stronger effect when small resolutions were used.
  Complexity: Higher complexities resulted in higher data-ink ratios, higher FG:BG ratios and a higher discriminability ratio. Otherwise, the complexity seemed to not influence the overall trend of the ratios greatly.

- ***Under what circumstances do the v-plots lose their discriminability?***
  As elaborated in more detail in subsection 5.0.2, the performed t-tests indicated that v-plots lost their discriminability on resolutions <= 180x320 px.

The fact that all ratios measured for single v-plots (matrix dimensionality 1) scaled more or less linearly over the tested resolutions indicated that there was no ideal size for single v-plots in the range of the tested resolutions. For v-plot matrices, it seemed that resolutions up to 180x320px were insufficient for a lossless display, but matrices scaled well on screen-spaces larger than that. This was indicated by the results of the t-tests, that were performed for the data-ink ratio and further supported by looking at the 100:0 data-ink ratio on small resolutions (except for single v-plots). While the data-ink ratio suggested that a resolution of at least 360x640px was needed, the discriminability ratio implied that a resolution of 180x320px was suitable for a loss-

less display. Further supporting the assumption that a resolution of 100x160px was insufficient, was the extreme case where no data-ink was present in the image. This appeared when a high matrix dimensionality was used in combination with a low layer complexity on the smallest tested resolution. The occurrence of this special case strongly suggested bad scalability for these conditions. Observing the FG:BG ratio implied otherwise, as it showed a generally linear trend over all tested configurations.

In terms of examining how well the proposed ratios performed in testing for visual scalability, the following findings were noted. While the data-ink ratio and the discriminability ratio indicated that v-plots did not scale well for small resolutions, this was not visible in the foreground-background ratio at all. It was neither apparent when looking at the visualized results nor when looking at the conducted t-tests. Further, the extreme case where no ink was displayed in the image, was picked up by the data-ink ratio and the discriminability ratio but not by the FG:BG ratio. This cast doubt on the expressiveness of the foreground-background ratio regarding the v-plot scalability and the way it was measured for matrices. One reason for that outcome was the handling of labels when calculating the FG:BG ratio. Since the labels were rightly counted to the foreground pixels, it was never checked whether there was otherwise no ink present in the image. This made it impossible to detect a complete loss of data-ink in the v-plots through calculating the FG:BG ratio.
Summing up the interpretation of the ratios, the question of how much screen-space must be dedicated to the v-plot view that was implemented in Visplore by VRVis is answered. The FG:BG ratio implied that the v-plots scaled well over all tested resolutions. But referring to its partially contrary results as opposed to the results of the other ratios, the FG:BG ratio did not serve as basis for this answer. Looking at the other ratios, it can be said, that generally a screen-space above 180x320px and larger was sufficient for a lossless display. Single v-plots were effectively shown on the smallest tested resolution of 100x160px. Other configurations than the dimensionality of the v-plots generally did not need be considered to ensure good scalability.
The proposed data-ink ratio and the discriminability ratio appear to be good indicators for visual scalability as they proved to be sensitive to greater changes in the visualization. The effectiveness of the proposed foreground-background ratio is questioned because it ignored the complete loss of actual data information in v-plot matrices for small resolutions.

6 Conclusion

An introduction to the basics in the field of data visualization, visual analytics and data comparison was given. Common visualization techniques were explained and the concept and usage

of hybrid data visualization techniques were elaborated. Of special interest for the purpose of this thesis were the v-plots, which were explained. Further, their usage was demonstrated by means of two use cases.
The v-plot visualization was then integrated into the software Visplore by VRVis. Therefore, the workings of the existing histogram view were analyzed and explained. Then the histogram view was extended to support computing and displaying single v-plots and v-plots in a matrix arrangement. The layers of the v-plot could be adapted in a variety of ways through user interaction. The fact that the v-plot view in Visplore by VRVis was usually visible as part of a dashboard along with other views, gave rise to the question of how much screen-space was needed by the view to display its information effectively. To find out how well the implemented v-plots scaled, based on the available screen-space in pixels, it was looked at existing methods to evaluate data visualizations. For the evaluation that was part of this thesis, three pixel-based ratios, that were based on existing concepts were proposed. The introduced ratios consisted of the data-ink ratio, the foreground-background ratio and the discriminability between v-plot sides as ratio. These ratios were tested and used for the evaluation of the scalability of the v-plot implementation. For the calculation of the ratios, a script was implemented in Python. The script operated on input images that were produced by the v-plot implementation in Visplore by VRVis. With some additional manually specified parameters, the script calculated the three evaluation ratios for a total of 375 input images that were exported from the implementation in Visplore by VRVis. These images contained various differently configured v-plots and v-plot matrices. The resulted ratios were visualized and interpreted. Further, several two-tailed statistical hypothesis tests were performed on the results to support or reject the findings. The assumption was made, that the data-ink ratio and the discriminability ratio seem descriptive for the scalability of the v-plots, while the foreground-background ratio lacked sensitivity to some drastic changes in the visualization. Therefore, the effectiveness of the foreground-background ratio as a tool to evaluate visual scalability was questioned. Otherwise, looking at the data-ink ratio and the discriminability ratio, the visual analysis and the statistical hypothesis tests suggested that a resolution of 180x320px is sufficient for a view containing v-plot matrices and as little screen-space as 100x160px is enough for views containing only a single v-plot.

6.1 Contributions

The v-plot implementation in Visplore by VRVis extended the software by another unique view mode. It supported preset configurations and also allowed for configuring it explicitly. It was integrated using the software's coloring legend and made sure to distribute colors thoroughly, so that v-plot sides stay traceable to their related data channel through every v-plot layer configuration. Further, the implemented view supported cross-filtering between the v-plots and other views of the dashboard, allowing for more advanced interactions.
We observed the trend of three calculated ratios that were based on existing concepts to make

an assumption about the scalability of a visualization technique. To test the effectiveness of the ratios as a tool for evaluating visual scalability, and to test for the scalability of the v-plot implementation, a script was written that computed the proposed ratio on the basis of an image and manually defined input parameters. The script was written in a way that it could be adapted to be applicable to images containing other visualization techniques easily. It supported the ratio calculations for images containing a single plot and also for images containing a matrix of plots. Applying this approach revealed that the data-ink ratio and the discriminability ratio showed mostly linear trends over various v-plot sizes, suggesting them to be valid indicators for the scalability of a visualization technique. While the foreground-background ratio partially failed to detect some information loss.

6.2 Outlook

Quite a few topics were tackled in the context of this thesis, that may be worth looking into further. One of them are improvements for the v-plots. Additional layers containing further data visualization techniques for the v-plots could be researched. Another possibility is to expand the visualization options for the existing layers. Concerning this, Blumenschein et al. [11] already specifically proposed researching the integration of dot plots into the v-plots and to experiment bringing a measure for uncertainty to the v-plot visualization.
Concerning our v-plot implementation, the developed matrix implementation could be extended to use up the space of the diagonal and in the lower right half. As an example, the diagonal space could be used for labelling the columns and rows. This would also solve the problem of needing a certain coloring for each layer configuration, as the data of a v-plot side would no longer only be relatable through its color. Alternatively, the space in the lower right half of the view could be used to show v-plots with a different configuration, following the example of the v-plot designer [8] by Blumenschein et al. [11]. In our implementation, the v-plots of a matrix appeared in the same order in which the assigned data channels were loaded. As further improvement, providing an automatic sorting of the v-plots of a matrix by a variety of factors could be integrated. An example would be to sort the v-plots by the absolute differences between the two v-plot sides or by their image similarity.
The evaluation script to calculate the ratios was written to support the v-plots, but with the extension of its applicability to other visualization techniques in mind. Therefore, it could easily be adapted and applied to evaluate other visualization techniques.
Future research could investigate the proposed evaluation ratios further, to find out how representative they are for the actual human perception. This is especially interesting for the discriminability ratio as it is meant to resemble the difference between plot sides as perceived by the viewer. For the data-ink ratio and the FG:BG ratio, the trend across resolutions is to be compared to the perceived changes by the viewer. The data-ink ratio and the discriminability ratio showed similar results, whereas the foreground-background ratio differed. Therefore, a user study could clarify which ratio resembles the human perception of a visualization the clos-

est. Based on that, each ratio could either be supported in being effective or be rejected as a tool to evaluate visual scalability. Referring to this findings of this thesis, it would especially clarify whether the assumption that the data-ink and discriminability ratios seem to uphold more expressiveness about the scalability than the foreground-background ratio, is supported.

Bibliography

[1] Basant Lal Agarwal. *Basic statistics*. New Age International, 2006.

[2] Syed Mohd Ali et al. “Big data visualization: Tools and challenges”. In: *Proceedings of the 2nd International Conference on Contemporary Computing and Informatics (IC3I’16)*. IEEE. 2016, pp. 656–660.

[3] Sara Alspaugh et al. “Futzing and moseying: Interviews with professional data analysts on exploration practices”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (2018), pp. 22–31.

[4] Natalia Andrienko et al. *Visual analytics for data scientists*. Springer, 2020.

[5] Manuela Aparicio and Carlos J Costa. “Data visualization”. In: *Communication design quarterly review* 3.1 (2015), pp. 7–11.

[6] John T Behrens. “Principles and procedures of exploratory data analysis”. In: *Psychological Methods* 2.2 (1997), p. 131.

[7] Yoav Benjamini. “Opening the box of a boxplot”. In: *The American Statistician* 42.4 (1988), pp. 257–262.

[8] Michael Blumenschein et al. *The v-plot designer*. https://v-plot.dbvis.de/#!/plot. Accessed: 2021-08-15.

[9] Michael Blumenschein et al. *The v-plot designer for matrices*. https://v-plot.dbvis.de/#!/vplotmatrix. Accessed: 2021-08-15.

[10] Michael Blumenschein et al. *V-plots: Designing Hybrid Charts for the Comparative Analysis of Data Distributions*. https://v-plot.dbvis.de/#!/. Accessed: 2021-08-15.

[11] Michael Blumenschein et al. “v-plots: Designing Hybrid Charts for the Comparative Analysis of Data Distributions”. In: *Computer Graphics Forum* 39 (June 2020), pp. 565–577. DOI: 10.1111/cgf.14002.

[12] Shreya Bose. *What is the ideal screen size for responsive design*. https://www.browserstack.com/guide/ideal-screen-sizes-for-responsive-design. Accessed: 2022-05-02.

[13] Max Bramer. *Principles of Data Mining*. 2nd. Springer Publishing Company, 2013. ISBN: 1447148835.

[14] Enrico G Caldarola and Antonio M Rinaldi. “Big data visualization tools: a survey”. In: *Research Gate* (2017).

[15] Sheelagh Carpendale. “Evaluating information visualizations”. In: *Information Visualization*. Springer, 2008, pp. 19–45.

[16] Chris Chatfield. “Exploratory data analysis”. In: *European Journal of Operational Research* 23.1 (1986), pp. 5–13.

[17] William S Cleveland. “Graphs in scientific publications”. In: *The American Statistician* 38.4 (1984), pp. 261–269.

[18] Renan Augusto Pupin De Oliveira, Lenon Fachiano Silva, and Danilo Medeiros Eler. “Hybrid visualization: a new approach to display instances relationship and attributes behaviour in a single view”. In: *Proceedings of the 19th International Conference on Information Visualisation*. IEEE. 2015, pp. 277–282.

[19] Stephen G Eick and Alan F Karr. “Visual scalability”. In: *Journal of Computational and Graphical Statistics* 11.1 (2002), pp. 22–43.

[20] Stephen Few and Perceptual Edge. “Data visualization: past, present, and future”. In: *IBM Cognos Innovation Center* (2007).

[21] Danyel Fisher and Miriah Meyer. *Making Data Visual: A practical guide to using visualization for Insight*. O’Reilly, 2018.

[22] Michael Friendly. “A brief history of data visualization”. In: *Handbook of data visualization*. Springer, 2008, pp. 15–56.

[23] Douglas J Gillan and Edward H Richman. “Minimalism and the syntax of graphs”. In: *Human Factors* 36.4 (1994), pp. 619–644.

[24] Michael Gleicher et al. “Visual comparison for information visualization”. In: *Information Visualization* 10.4 (2011), pp. 289–309. DOI: 10.1177/1473871611416549.

[25] *GTK*. https://www.gtk.org/. Accessed: 2022-02-07.

[26] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[27] Jeffrey Heer et al. “Creation and collaboration: Engaging new audiences for information visualization”. In: *Information Visualization*. Springer, 2008, pp. 92–133.

[28] Paul Hekkert, Dirk Snelders, and Piet CW Van Wieringen. “‘Most advanced, yet acceptable’: Typicality and novelty as joint predictors of aesthetic preference in industrial design”. In: *British Journal of Psychology* 94.1 (2003), pp. 111–124.

[29] W.A. Hemmerich. *T-test*. Accessed: 2022-03-08. URL: https://matheguru.com/stochastik/t-test.html.

[30] Jerry L Hintze and Ray D Nelson. “Violin plots: a box plot-density trace synergism”. In: *The American Statistician* 52.2 (1998), pp. 181–184.

[31] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[32] Ohad Inbar, Noam Tractinsky, and Joachim Meyer. “Minimalism in information visualization: attitudes towards maximizing the data-ink ratio”. In: *Proceedings of the 14th European Conference on Cognitive ergonomics: Invent! Explore!* 2007, pp. 185–188.

[33] Laurent Itti. “Visual salience”. In: *Scholarpedia* 2.9 (2007), p. 3327.

[34] Laurent Itti, Christof Koch, and Ernst Niebur. “A model of saliency-based visual attention for rapid scene analysis”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.11 (1998), pp. 1254–1259.

[35] Mikkel R Jakobsen and Kasper Hornbæk. “Interactive visualizations on large and small displays: The interrelation of display size, information space, and scale”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2336–2345.

[36] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[37] Heike Jänicke and Min Chen. “A salience-based quality metric for visualization”. In: *Computer Graphics Forum*. Vol. 29. 3. Wiley Online Library. 2010, pp. 1183–1192.

[38] *Jupyter Notebook*. https://jupyter.org/. Accessed: 2022-02-07.

[39] Peter Kampstra. “Beanplot: A boxplot alternative for visual comparison of distributions”. In: *Journal of Statistical Software* 28 (2008), pp. 1–9.

[40] Sean Kandel et al. “Enterprise data analysis and visualization: An interview study”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2917–2926.

[41] Maurits Kaptein and Edwin van den Heuvel. *Statistics for Data Scientists: An Introduction to Probability, Statistics, and Data Analysis*. Springer Nature, 2022.

[42] Daniel Keim et al. *Mastering The Information Age: Solving Problems with Visual Analytics*. Eurographics Association, Goslar, Germany, 2010.

[43] Daniel Keim et al. “Visual analytics: Definition, process, and challenges”. In: *Information Visualization*. Springer, 2008, pp. 154–175.

[44] Daniel Keim et al. “Visual Analytics: Scope and Challenges”. In: *Lecture notes in computer science, No. 4404, pp. 76-90* 1 (Apr. 2008). DOI: 10.1007/978-3-540-71080-6_6.

[45] Martin Krzywinski and Naomi Altman. “Visualizing samples with box plots: use box plots to illustrate the spread and differences of samples”. In: *Nature Methods* 11.2 (2014), pp. 119–121.

[46] Tanuj Kumar. *T-test in Excel*. Accessed: 2022-05-02. URL: https://www.wallstreetmojo.com/t-test-in-excel/.

[47] Shengchi Liu et al. “Wind power prediction based on meteorological data visualization”. In: *Proceedings of the 8th EAI International Conference on Green Energy and Networking (GreeNets ’21), June 6-7, Dalian, China* (2021). DOI: 10.4108/eai.6-6-2021.2307765.

[48] Laura E Matzen et al. “Data visualization saliency model: A tool for evaluating abstract data visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2017), pp. 563–573.

[49] Felix Mayer et al. “Automatic generation of integrated process data visualizations using human knowledge”. In: *Proceedings of the International Conference on Human Interface and the Management of Information*. Springer. 2015, pp. 488–498.

[50] Viktor Mayer-Schönberger and Kenneth Cukier. *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

[51] Ernest James McCormick and Mark S Sanders. *Human factors in engineering and design*. McGraw-Hili Companies, 1993.

[52] *Mesa 3D*. https://www.mesa3d.org/. Accessed: 2022-02-07.

[53] Microsoft Corporation. *Microsoft Excel*. Version 2019 (16.0). Accessed: 2022-05-02. URL: https://office.microsoft.com/excel.

[54] *Microsoft Excel T-Test*. Accessed: 2022-03-08. URL: https://support.microsoft.com/en-us/office/t-test-function-d4e08ec3-c545-485f-962e-276f7cbed055.

[55] Stephen R Midway. “Principles of effective data visualization”. In: *Patterns* 1.9 (2020), p. 100141.

[56] E. Niebur. “Saliency map”. In: *Scholarpedia* 2.8 (2007). revision #147400, p. 2675. DOI: 10.4249/scholarpedia.2675.

[57] *OpenGL*. https://www.opengl.org/. Accessed: 2022-02-07.

[58] Hans-Georg Pagendarm and Frits H. Post. “Comparative Visualization - Approaches and Examples”. In: *Visualization in Scientific Computing*. Springer Verlag, Wien, 1995.

[59] Lisa Pappas and Lisa Whitman. “Riding the technology wave: Effective dashboard data visualization”. In: *Proceedings of the Symposium on Human Interface*. Springer. 2011, pp. 249–258.

[60] *Photovoltaic (PV) Solar Panel Energy Generation data*. https://data.london.gov.uk/dataset/photovoltaic--pv--solar-panel-energy-generation-data. Accessed: 2022-03-07.

[61] Kristin Potter et al. “Methods for Presenting Statistical Information: The Box Plot”. In: *Visualization of Large and Unstructured Data Sets, GI-Edition Lecture Notes in Informatics (LNI) S-4* (2006), pp. 97–106.

[62] Kristin Potter et al. “Visualizing summary statistics and uncertainty”. In: *Computer Graphics Forum*. Vol. 29. 3. Wiley Online Library. 2010, pp. 823–832.

[63] *Produkte Lizenzen*. Accessed: 2022-05-02. URL: https://www.vrvis.at/produkte-loesungen/produkte-lizenzen/visplore.

[64] *Python 3*. https://www.python.org/. Accessed: 2022-02-07.

[65] Tilmann Rabl and Hans-Arno Jacobsen. “Big data generation”. In: *Specifying Big Data Benchmarks*. Springer, 2012, pp. 20–27.

[66] Prabhu Ramachandran and Gaël Varoquaux. “Mayavi: 3D visualization of scientific data”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 40–51.

[67] Raphael Sahann, Torsten Müller, and Johanna Schmidt. “Histogram binning revisited with a focus on human perception”. In: *Proceedings of IEEE VIS (Short Papers)*. IEEE. 2021, pp. 66–70.

[68] Rachel Schutt and Cathy O’Neil. *Doing data science: Straight talk from the frontline*. O’Reilly Sebastopol, CA, 2013.

[69] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[70] Evan F Sinar. “Data visualization”. In: *Big Data at Work*. Routledge, 2015, pp. 129–171.

[71] *Tableau software*. https://www.tableau.com/. Accessed: 2022-03-02.

[72] James J Thomas and Kristin A Cook. “A visual analytics agenda”. In: *IEEE Computer Graphics and Applications* 26.1 (2006), pp. 10–13.

[73] Christian Tominski, Camilla Forsell, and Jimmy Johansson. “Interaction Support for Visual Comparison Inspired by Natural Behavior”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2719–2728. DOI: 10.1109/TVCG.2012.237.

[74] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press, 2018, pp. 91–105.

[75] Rafael Veras and Christopher Collins. “Discriminability tests for visualization effectiveness and scalability”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2019), pp. 749–758.

[76] Stéfan van der Walt et al. “scikit-image: image processing in Python”. In: *PeerJ* 2 (June 2014), e453. DOI: 10.7717/peerj.453.

[77] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. “Multiscale structural similarity for image quality assessment”. In: *Proceedings of the Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. Vol. 2. IEEE. 2003, pp. 1398–1402.

[78] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2019.

[79] Tracey L. Weissgerber et al. “Beyond bar and line graphs: time for a new data presentation paradigm”. In: *PLoS Biology* 13.4 (2015), e1002128.

[80] Claus O. Wilke. *Visualizing distributions: Histograms and density plots*. https://clauswilke.com/dataviz/histograms-density-plots.html#histograms-density-plots. Accessed: 2021-08-14.

[81] Claus O. Wilke. *Visualizing many distributions at once*. https://clauswilke.com/dataviz/boxplots-violins.html. Accessed: 2021-08-14.

[82] David F Williamson, Robert A Parker, and Juliette S Kendrick. “The box plot: a simple visual method to interpret data”. In: *Annals of Internal Medicine* 110.11 (1989), pp. 916–921.

[83] Yu Zhang et al. “Research on Application of Data Visualization in Finance”. In: *DEStech Transactions on Engineering and Technology Research,(acaai)* (2020).

List of Figures

List of Tables

List of Abbreviations

CDA confirmatory data analysis

EDA exploratory data analysis

KDE kernel density estimation

PV Photovoltaic Solar Panel Energy Generation data

DI Data-ink group

NDI Non data-ink group

LI Labels ink group

BG Background group

FG:BG Foreground-background

px pixel

A Appendix

A.1 Photovoltaic (PV) Solar Panel Energy Generation data

License Statement for Photovoltaic and Weather dataset:
"Contains public sector information licensed under the Open Government Licence v3.0."
Source of Dataset (in its original form): https://data.london.gov.uk/dataset/photovoltaic–pv–solar-panel-energy-generation-data
License: UK Open Government Licence OGL 3: http://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/

———

Dataset was modified to fit the needs and purposes of demonstrating USPs of the Visplore software. E.g. - geographic places were anonymized, - time rasters were changed and synchronized, - a few data quality issues were introduced for showcase purposes - technical column names were modified for simpler explanation + as not to confuse at first glance with original, unmodified dataset.

A.2 The v-plot implementation in Visplore by VRVis

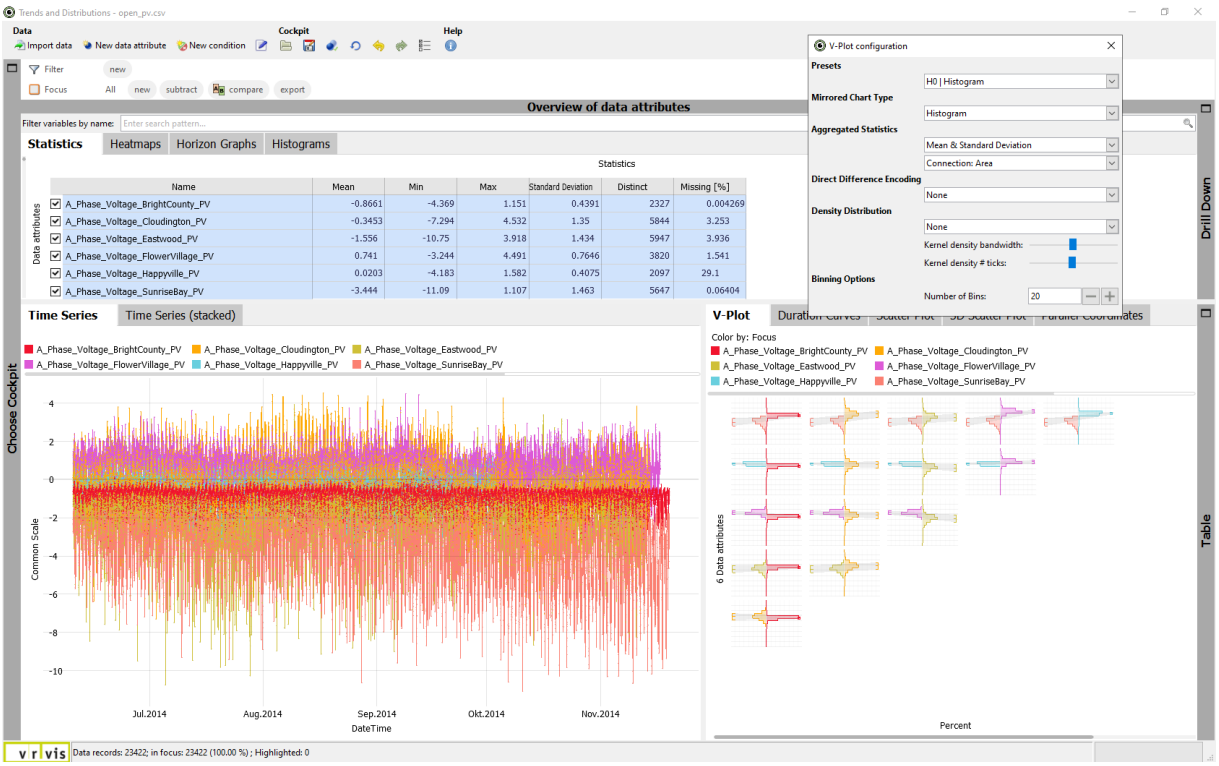


Figure 33: A screenshot from the v-plot visualization that was implemented in Visplore by VRVis. Here, it can be seen being used in a dashboard, that was arranged for the visualization and analysis of the trends and distributions of data. As can be seen here, he selection of more than two data channels, leads to the v-plots being shown in a matrix arrangement. On the top right, the *V-Plot configuration* dialogue can be seen. It contains the layer configuration options and offers the functionality to set the bin count. The options that were set in this dialogue, resulted in the displayed v-plots in the view below.

A.3 Evaluation script

Listing A.1: Visualization evaluation script

```
#Buesch Nadine 2021/2022
#Script for computing pixel-based ratios used in the evaluation of v-plots

from skimage import io
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
#PRIORITY OF EVALUATION LAYERS:
DI = "dataLayer"    #contains data and/or background - prio1
```

```
NDI = "gridLayer"    #contains grid and/or bg            - prio2
LI = "labelsLayer" #contains labels and/or bg         - prio2 (special case of NDI)
#BG only logical concept and therefore, not needed in the computation- prio3


#defines an area in image space and the layers it can contain
class Region:
    def __init__(self, x=0, y=0, width=0, height=0, layers =[]):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.layers = layers

#contains a specific number of Regions, that make up a single v-plot
class encodedImageRegion:
    def __init__(self, plotLeft=Region(0, 0, 0, 0, []), plotRight=Region(0, 0, 0, 0, []),
                 bottomLegend=Region(0, 0, 0, 0, []), leftLegend=Region(0, 0, 0, 0, []),
                 rightPadding=Region(0, 0, 0, 0, []), middleLegend=Region(0, 0, 0, 0, [])):
        self.regions = [plotLeft, plotRight, bottomLegend, leftLegend, rightPadding, middleLegend]
        self.plotLeft = plotLeft
        self.plotRight = plotRight
        self.bottomLegend = bottomLegend
        self.leftLegend = leftLegend
        self.rightPadding = rightPadding
        self.middleLegend = middleLegend

    def GetMatrixPlotWidth(self, withRightPadding=True):
        if withRightPadding:
            width = self.plotLeft.width + self.plotRight.width + self.rightPadding.width +
                self.middleLegend.width
        else:
            width = self.plotLeft.width + self.plotRight.width + self.middleLegend.width
        return width

    def GetMatrixPlotHeight(self):
        height = self.plotLeft.height + self.bottomLegend.height
        return height

from skimage.util import img_as_float, img_as_ubyte

#the maximum distance two RGB colors can have
#is used to check similarity of colors
maxColorDist = np.sqrt(np.sum((np.array([0, 0, 0]) - np.array([255, 255, 255]))** 2, axis=0))
```

```
#Evaluation class
class VisualizationEvaluation:
    def __init__(self, original, regions, labelColor=[0, 0, 0],
                 gridColors=[[229, 229, 229], [206, 206, 206]],
                 backgroundColor=[255, 255, 255],
                 dataColors=[[238, 238, 238], [146, 146, 146]]):

        self.dataColors = dataColors #colors used for LI and DI regions
        self.labelColor = labelColor #text color
        self.gridColors = gridColors #used for the grid
        self.bgColor = backgroundColor

        #the input image and its dimensions
        self.originalImage = original
        self.width = original.shape[1]
        self.height = original.shape[0]

        #empty/black images with size of the original
        #to be filled with white pixels of the specified layer
        self.DIimage = np.zeros((self.height, self.width, 3), dtype=np.uint8)
        self.NDIimage = np.zeros((self.height, self.width, 3), dtype=np.uint8)
        self.BGimage = np.zeros((self.height, self.width, 3), dtype=np.uint8)

        self.regions = regions
        self.encodedImage = self.createAreaEncodedImage(regions)

        #storing pixel count per layer
        self.DIcount = 0
        self.NDIcount = 0
        self.BGcount = 0
        self.count = self.width * self.height

    #Debug function to output layer pixel counts
    def getCounts(self):
        if self.DIcount > 0 or self.NDIcount > 0:
            print("DI: ", self.DIcount)
            print("NDI: ", self.NDIcount)
            print("BG: ", self.BGcount)
            print("All pixels: ", self.count)

    #Calculates and outputs the FG:BG ratio based on the computed pixel counts
    def printFGBGRatio(self):
        foregroundPercent = (self.DIcount + self.NDIcount) / self.count * 100.0
        backgroundPercent = self.BGcount / self.count * 100.0
```

```
        print("FG : BG = ", "{0:.2f}".format(foregroundPercent), ":",
            "{0:.2f}".format(backgroundPercent))
        return foregroundPercent

    #Calculated and outputs the data:ink ratio based on the computed pixel counts
    def printDataToInkRatio(self):
        inkCount = self.DIcount + self.NDIcount
        if inkCount > 0:
            dataPercent = self.DIcount / inkCount * 100.0
            nonDataPercent = self.NDIcount / inkCount * 100.0
        else:
            dataPercent = -1.0 #no ink present in the plot
            nonDataPercent = -1.0
        print("data : ink = ", "{0:.2f}".format(dataPercent), ":",
            "{0:.2f}".format(nonDataPercent))

    #Calculates and outputs the number of different data pixels between plot sides
    #Also saves the data per plot side and the differences as image
    def printNumDifferences(self, printResult=True):
        diffPixelArray = []
        leftWidth = regions.plotLeft.width
        rightWidth = regions.plotRight.width

        leftPlot = self.DIimage[0:regions.plotLeft.height,
                                regions.plotLeft.x:regions.plotLeft.x + leftWidth]

        rightPlot = self.DIimage[0:regions.plotRight.height,
                                regions.plotRight.x:regions.plotRight.x + rightWidth][:, ::-1]

        io.imshow(leftPlot)
        io.show()
        io.imshow(rightPlot)
        io.show()

        io.imsave(fname="leftPlot.png", arr=img_as_ubyte(leftPlot))
        io.imsave(fname="rightPlot.png", arr=img_as_ubyte(rightPlot))

        #plot side differences can only be calculated on plot sides of same size
        if leftPlot.shape[1] == rightPlot.shape[1]:
            diffImage = np.zeros((leftPlot.shape[0], rightPlot.shape[1], 3), dtype=np.uint8)
            diffPixels = 0
            equivPixels = 0
            allPixels = leftPlot.shape[0] * leftPlot.shape[1]

            #checks for differences in plot side data and writes them to a new image
```

```
            for w in range(leftPlot.shape[1]):
                for h in range(leftPlot.shape[0]):
                    pixelL = leftPlot[h][w]
                    pixelR = rightPlot[h][w]
                    if np.array_equiv(pixelL, pixelR):
                        equivPixels += 1
                    else:
                        diffPixels += 1
                        diffImage[h][w] = [255, 255, 255]
            io.imshow(diffImage)
            io.show()
            io.imsave(fname="diffPlots.png", arr=img_as_ubyte(diffImage))
            if printResult:
                print("Number of different pixels between plot sides: ", diffPixels)
                print("Percent different pixels of amount pixels per plot side: ",
                      "{0:.2f}".format(diffPixels / allPixels * 100), "%")
            diffPixelArray.append(diffPixels)
            diffPixelArray.append(allPixels)
        else:
            print("image widths don't match.")

        return diffPixelArray

#create image containing color encoded regions for debugging:
# DATA -Red
# LI   -Green
# NDI  -Blue
# black pixels indicate background
def createAreaEncodedImage(self, encodedRegions):
    image = np.zeros((self.height, self.width, 3))
    for region in encodedRegions.regions:
        for pixelw in range(region.width):
            x = region.x + pixelw
            for pixelh in range(region.height):
                y = region.y + pixelh
                pixel = image[y][x]
                if DI in region.layers:
                    image[y][x] = [1, pixel[1], pixel[2]]
                if LI in region.layers:
                    image[y][x] = [pixel[0], 1, pixel[2]]
                if NDI in region.layers:
                    image[y][x] = [pixel[0], pixel[1], 1]

    image_flipped = image[::-1, :]
    image_flipped = 0.7 * img_as_float(self.originalImage) + 0.3 * img_as_float(image_flipped)
```

```
        io.imsave(fname="encodedRegions.png", arr=img_as_ubyte(image_flipped),
            check_contrast=False)

        return image[::-1, :]

    #debugging: visualizes regions as overlay on top of the original
    def showAreaEncoding(self):
        placeholder = True
        io.imshow(0.7 * img_as_float(self.originalImage) + 0.3 * img_as_float(self.encodedImage))
        io.show()

    #to take into account the smoothed rendering of text,
    #check whether color is closer to text color or BG color
    #the default factor 0.8 yielded overall best results for text recognition
    def isText(self, pixelColor, factor=0.8):
        txt_squaredDist = np.sum((self.labelColor - pixelColor)** 2, axis=0)
        txt_dist = np.sqrt(txt_squaredDist)

        if (txt_dist < maxColorDist * factor):
            return True
        else:
            return False

    #checks if the given color is the same as one of the defined data colors
    def isData(self, pixelColor, factor=0.0):
        minDist = maxColorDist
        for color in self.dataColors:
            data_squaredDist = np.sum((color - pixelColor)** 2, axis=0)
            data_dist = np.sqrt(data_squaredDist)
            if data_dist < minDist:
                minDist = data_dist

        if (minDist <= maxColorDist * factor):
            return True
        else:
            return False

    #checks if the given pixel color is close enough to the BG color to be considered BG
    def isBG(self, pixelColor, factor=0.2):
        bg_squaredDist = np.sum((self.bgColor - pixelColor)** 2, axis=0)
        bg_dist = np.sqrt(bg_squaredDist)

        if (bg_dist <= maxColorDist * factor):
            return True
        else:
```

```
        return False

#counts and marks the pixels per layer according to the layer priority
def countPixelsPerLayer(self, showImages=True):
    original_flipped = self.originalImage[::-1, :]
    white = [255, 255, 255]

    for region in self.regions.regions:
        for pixelw in range(region.width):
            x = region.x + pixelw
            for pixelh in range(region.height):
                isDI = False
                isNDI = False
                isBG = False

                y = region.y + pixelh
                pixel = original_flipped[y][x]

                if np.array_equiv(pixel, self.bgColor) or not region.layers:
                    isBG = True

                elif DI in region.layers:
                    numdatapixels = 0
                    if LI in region.layers:
                        #check for labels in data region
                        if self.isText(pixel, 0.8):
                            isNDI = True
                            #if label close to data, its probably covered up and therefore,
                                pixels count to data
                            for ky in range(-1, 1):
                                kypos = y + ky
                                if kypos > 0 and kypos < self.height:
                                    for kx in range(-2, 2):
                                        kxpos = x + kx
                                        if kxpos > 0 and kxpos < self.width:
                                            kpixel = original_flipped[kypos][kxpos]
                                            if self.isData(kpixel, 0.0):
                                                isNDI = False
                                                break
                        #smoothing of labels should be counted to bg not to data
                        elif self.isData(pixel, 0.2):
                            isDI = True
                            for ky in range(-1, 1):
                                kypos = y + ky
                                if kypos > 0 and kypos < self.height:
```

```
                                for kx in range(-2, 2):
                                    kxpos = x + kx
                                    if kxpos > 0 and kxpos < self.width:
                                        kpixel = original_flipped[kypos][kxpos]
                                        if self.isBG(kpixel, 0.0):
                                            isBG = True
                                            isDI = False
                            else:
                                isBG = True

                        if NDI in region.layers:
                            for color in self.gridColors:
                                if np.array_equiv(color, pixel):
                                    isNDI = True

                        if not (isNDI or isBG):
                            isDI = True

                    elif NDI in region.layers:
                        index = 0
                        for color in self.gridColors:
                            if np.array_equiv(color, pixel):
                                isNDI = True
                                break
                            index += 1

                    elif LI in region.layers:
                        if self.isText(pixel, 0.8):
                            isNDI = True
                        else:
                            isBG = True

                    if isDI:
                        self.DIimage[y][x] = white
                        self.DIcount += 1
                    elif isNDI:
                        self.NDIimage[y][x] = white
                        self.NDIcount += 1
                    elif isBG:
                        self.BGimage[y][x] = white
                        self.BGcount += 1

        #clean up falsely assumed grid pixels
        self.cleanGridPixels()
```

```
        #debug print pixels of assigned layers
        self.DIimage = self.DIimage[::-1, :]
        io.imsave(fname="countedDI.png", arr=img_as_ubyte(self.DIimage), check_contrast=False)
        self.NDIimage = self.NDIimage[::-1, :]
        io.imsave(fname="countedNDI.png", arr=img_as_ubyte(self.NDIimage),
            check_contrast=False)
        self.BGimage = self.BGimage[::-1, :]
        io.imsave(fname="countedBG.png", arr=img_as_ubyte(self.BGimage), check_contrast=False)

        #make sure all pixels were assigned to a corresponding layer
        diffCount = (self.width * self.height) - (self.DIcount + self.NDIcount + self.BGcount)
        if diffCount != 0:
            print("Counts differ by ", diffCount)
        else:
            print("Image size of original and calculated match.")

        #debug show the images encoding which pixels belong to which layer
        if showImages:
            fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(50, 10))
            fig.suptitle('Image regions:', size=30)
            axes[0].imshow(self.DIimage)
            axes[0].set_title('Data layer', size=25)
            axes[0].axis('off')

            axes[1].imshow(self.NDIimage)
            axes[1].set_title('Grids&Labels layer', size=25)
            axes[1].axis('off')

            axes[2].imshow(self.BGimage)
            axes[2].set_title('Background layer', size=25)
            axes[2].axis('off')
            plt.show()

    #to account for some falsely detected grid pixels, iterate the generated NDI image and detect
        loose pixels marked as grid. If they have no direct grid connection on either side, they are
        marked DI
    def cleanGridPixels(self):
        white = [255, 255, 255]
        correctedPixelsImage = np.zeros((self.height, self.width, 3), dtype=np.uint8)
        flippedRegionImage = self.encodedImage[::-1, :]

        for x in range(self.width):
            for y in range(self.height):
                pixel = self.NDIimage[y][x]
                numGridPixels = 0
```

```
                #only if region can contain DI
                if np.array_equiv(pixel, white) and flippedRegionImage[y][x][0] == 1.0:
                    #top
                    if y < self.height - 1:
                        if np.array_equiv(self.NDIimage[y + 1][x], white):
                            numGridPixels += 1
                    #bottom
                    if y > 0:
                        if np.array_equiv(self.NDIimage[y - 1][x], white):
                            numGridPixels += 1
                    #right
                    if x < self.width - 1:
                        if np.array_equiv(self.NDIimage[y][x + 1], white):
                            numGridPixels += 1
                    #left
                    if x > 0:
                        if np.array_equiv(self.NDIimage[y][x - 1], white):
                            numGridPixels += 1

                    if numGridPixels == 0:
                        self.NDIcount -= 1
                        self.DIcount += 1
                        correctedPixelsImage[y][x] = white

        self.NDIimage = self.NDIimage - correctedPixelsImage
        self.DIimage = self.DIimage + correctedPixelsImage

    #debug checks whether every pixel was assigned to a layer
    def checkRegionCompleteness(self):
        completenessImage = img_as_ubyte(self.DIimage + self.NDIimage + self.BGimage)

        numCountedPixels = 0
        for x in range(self.width):
            for y in range(self.height):
                if np.array_equiv(completenessImage[y][x], [255, 255, 255]):
                    numCountedPixels += 1

        completeness = numCountedPixels / (self.width * self.height) * 100
        title = "% of pixels assigned: " + str(completeness)

        if completeness != 100.0:
            fig = plt.figure()
            fig.suptitle(title)
            plt.imshow(completenessImage)
            plt.show()
```

```
            io.imsave(fname="completenessImage.png", arr=img_as_ubyte(completenessImage),
                check_contrast=False)
        #else:
            print( title )

#Matrix support
#Creates VisualizationEvaluations based on the given dimension, with the size of the given regions
    FG:BG is calculated for the whole image, while data:ink and different Pixels between plot sides
    are calculated per v-plot and then computed using the arithmetic mean
class MatrixEvaluation:
    def __init__(self, original, regions, leftRegion, bottomRegion, dimPlots=3, labelColor=[0, 0, 0],
                 gridColors=[[229, 229, 229], [206, 206, 206]],
                backgroundColor=[255, 255, 255],
                dataColors=[[238, 238, 238], [170, 170, 170]]):

        self.width = original.shape[1]
        self.height = original.shape[0]
        self.count = self.width * self.height

        self.dimPlots = dimPlots
        self.plotWidth = regions.GetMatrixPlotWidth()
        self.plotHeight = regions.GetMatrixPlotHeight()


        #variable offset on the right of the view
        self.defaultRightPadding = 4

        #no leftLegend for individual plot regions
        self.plotRegions = encodedImageRegion(regions.plotLeft, regions.plotRight,
            regions.bottomLegend,
                                              Region(0, 0, 0, 0, []), regions.rightPadding,
                                                  regions.middleLegend)

        # left legend is defined once for the matrix, not per v-plot
        self.leftLegend = VisualizationEvaluation(original[0:leftRegion.leftLegend.height,
                                                   0:leftRegion.leftLegend.width],
                                                   leftRegion, labelColor, gridColors,
                                                   backgroundColor, dataColors)

        #bottom legend is defined once for the matrix, not per v-plot
        plotYBegin = self.height - bottomRegion.bottomLegend.height
        self.bottomLegend = VisualizationEvaluation(original[plotYBegin:
                                                             plotYBegin +
                                                                 bottomRegion.bottomLegend.height,
                                                    0:bottomRegion.regions[2].width],
```

```
                                        bottomRegion, labelColor, gridColors,
                                        backgroundColor, dataColors)

#empty/black images with size of the original
#to be filled with white pixels of the specified layer
self.DIimage = np.zeros((self.height, self.width, 3), dtype=np.uint8)
self.NDIimage = np.zeros((self.height, self.width, 3), dtype=np.uint8)
self.BGimage = np.zeros((self.height, self.width, 3), dtype=np.uint8)
regionImg = np.zeros((self.height, self.width, 3), dtype='float')


#the individual VisualizationEvaluations are created therefore, the respective v-plot
    positions are computed with reference to the matrix dimension and the defined Regions
self.vizEvaluations = []
xOffset = regions.regions[3].width #leftlegend
vizIndex = 0
for w in range(dimPlots):
    if w < dimPlots - 1:
        offsetW = xOffset + w * self.plotWidth
        endW = offsetW + self.plotWidth
        for h in range(dimPlots):
            if h < dimPlots - 1:
            #adapt region encoding, to wherever we are in the image
                offsetH = h * self.plotHeight
                endH = offsetH + self. plotHeight
                self.vizEvaluations.append(VisualizationEvaluation(original[offsetH:endH,
                    offsetW:endW],
                                                                   self.plotRegions,
                                                                   labelColor, gridColors,
                                                                   backgroundColor,
                                                                       dataColors))
                io.imsave(fname="./defregions/defregion" + str(w) + str(h) + ".png",
                          arr=img_as_ubyte(self.vizEvaluations[vizIndex].encodedImage),
                              check_contrast=False)

                #for debugging purpose do a region check: mark pixels that were assigned to
                    a plot white if a pixel gets wrongly assigned again, set its color to red
                regionOffsetH = offsetH
                regionOffsetW = offsetW
                while regionOffsetH < endH:
                    regionOffsetW = offsetW
                    while regionOffsetW < endW:
                        if np.array_equiv(regionImg[regionOffsetH][regionOffsetW], [1, 1,
                            1]):
                            regionImg[regionOffsetH][regionOffsetW] = [1, 0, 0]
```

```
                elif not np.array_equiv(regionImg[regionOffsetH][regionOffsetW], [1,
                    0, 0]):
                    regionImg[regionOffsetH][regionOffsetW] = [1, 1, 1]
                regionOffsetW += 1
            regionOffsetH += 1

    #append bottom legend to region check
    yEnd = self.height
    xEnd = bottomRegion.bottomLegend.x + bottomRegion.bottomLegend.width
    y = self.height - bottomRegion.bottomLegend.height - 1
    while y < yEnd:
        x = bottomRegion.bottomLegend.x
        while x < xEnd:
            if np.array_equiv(regionImg[y][x], [1, 1, 1]):
                regionImg[y][x] = [1, 0, 0]
            elif not np.array_equiv(regionImg[y][x], [1, 0, 0]):
                regionImg[y][x] = [1, 1, 1]
            x += 1
        y += 1
    #append left legend to region check
    yEnd = leftRegion.leftLegend.height
    xEnd = leftRegion.leftLegend.width
    y = 0
    while y < yEnd:
        x = 0
        while x < xEnd:
            if np.array_equiv(regionImg[y][x], [1, 1, 1]):
                regionImg[y][x] = [1, 0, 0]
            elif not np.array_equiv(regionImg[y][x], [1, 0, 0]):
                regionImg[y][x] = [1, 1, 1]
            x += 1
        y += 1

    print(regionImg.shape[0], regionImg.shape[1])
    io.imshow(regionImg)
    io.show()

#calls the showAreaEncoding function of the VisualizationEvaluation class per v-plot
def showAreaEncoding(self):
    for vizIndex in range(len(self.vizEvaluations)):
            self.vizEvaluations[vizIndex].showAreaEncoding()

#calls the countPixelsPerLayer function of the VisualizationEvaluation class per v-plot. Marks
    'empty' plots that occur through drawing only one diagonal of the matrix, as BG
def countPixelsPerLayer(self):
```

```
#count for  left  and bottom legend, which are not per v-plot
self.leftLegend.countPixelsPerLayer(False)
self.bottomLegend.countPixelsPerLayer(False)

vizIndex = 0
for w in range(self.dimPlots):
    for h in range(self.dimPlots):
        if w == (self.dimPlots - 1):
            plotWidth = self.plotRegions.GetMatrixPlotWidth(False) +
                self.defaultRightPadding
            currentPlotimageDI = np.zeros((self.plotHeight, plotWidth, 3), dtype=np.uint8)
            currentPlotimageNDI = np.zeros((self.plotHeight, plotWidth, 3), dtype=np.uint8)
            currentPlotimageBG = np.full((self.plotHeight, plotWidth, 3), 255,
                dtype=np.uint8)
        elif h == (self.dimPlots - 1) :
            currentPlotimageDI = np.zeros((self.plotHeight, self.plotWidth, 3),
                dtype=np.uint8)
            currentPlotimageNDI = np.zeros((self.plotHeight, self.plotWidth, 3),
                dtype=np.uint8)
            currentPlotimageBG = np.full((self.plotHeight, self.plotWidth, 3), 255,
                dtype=np.uint8)
        else:
            self.vizEvaluations[vizIndex].countPixelsPerLayer(False)

            currentPlotimageDI = self.vizEvaluations[vizIndex].DIimage[:, ::-1]
            currentPlotimageNDI = self.vizEvaluations[vizIndex].NDIimage[:, ::-1]
            currentPlotimageBG = self.vizEvaluations[vizIndex].BGimage[:, ::-1]
            vizIndex += 1

        if h == 0: # first  assignment of row
            image_columnDI = currentPlotimageDI
            image_columnNDI = currentPlotimageNDI
            image_columnBG = currentPlotimageBG
        else:
            image_columnDI = np.concatenate((image_columnDI, currentPlotimageDI))
            image_columnNDI = np.concatenate((image_columnNDI, currentPlotimageNDI))
            image_columnBG = np.concatenate((image_columnBG, currentPlotimageBG))

    if w == 0: # first  assignment of column
        self.DIimage = image_columnDI
        self.NDIimage = image_columnNDI
        self.BGimage = image_columnBG
    else:
        self.DIimage = np.concatenate((image_columnDI, self.DIimage), axis=1)
        self.NDIimage = np.concatenate((image_columnNDI, self.NDIimage), axis=1)
```

```
            self.BGimage = np.concatenate((image_columnBG, self.BGimage), axis=1)

        #append bottom and left legend
        self.DIimage = np.concatenate((self.leftLegend.DIimage, self.DIimage[:, ::-1]) , axis=1)
        self.NDIimage = np.concatenate((self.leftLegend.NDIimage, self.NDIimage[:, ::-1]), axis=1)
        self.BGimage = np.concatenate((self.leftLegend.BGimage, self.BGimage[:, ::-1]), axis=1)

        self.DIimage = np.concatenate((self.DIimage, self.bottomLegend.DIimage), axis=0)
        self.NDIimage = np.concatenate((self.NDIimage, self.bottomLegend.NDIimage), axis=0)
        self.BGimage = np.concatenate((self.BGimage, self.bottomLegend.BGimage), axis=0)


        #debug save and show pixels of assigned layers
        io.imsave(fname="countedDI.png", arr=img_as_ubyte(self.DIimage), check_contrast=False)
        io.imsave(fname="countedNDI.png", arr=img_as_ubyte(self.NDIimage),
            check_contrast=False)
        io.imsave(fname="countedBG.png", arr=img_as_ubyte(self.BGimage), check_contrast=False)


        fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(50, 10))
        fig.suptitle('Image regions:', size=30)
        axes[0].imshow(self.DIimage)
        axes[0].set_title('Data layer', size=25)
        axes[0].axis('off')

        axes[1].imshow(self.NDIimage)
        axes[1].set_title('Grids&Labels layer', size=25)
        axes[1].axis('off')

        axes[2].imshow(self.BGimage)
        axes[2].set_title('Background layer', size=25)
        axes[2].axis('off')
        plt.show()

    #debug check that every pixel was assigned to a region
    def checkRegionCompleteness(self):
        completenessImage = self.DIimage + self.NDIimage + self.BGimage

        numCountedPixels = 0
        for x in range(self.width):
            for y in range(self.height):
                if np.array_equiv(completenessImage[y][x], [255, 255, 255]):
                    numCountedPixels += 1

        completeness = numCountedPixels / (self.width * self.height) * 100
```

```
        title = "% of pixels assigned: " + str(completeness)

        if completeness != 100.0:
            fig = plt.figure()
            fig.suptitle( title )
            plt.imshow(completenessImage)
            plt.show()
            io.imsave(fname="completenessImage.png", arr=img_as_ubyte(completenessImage),
                check_contrast=False)
        else:
            print( title )

    #debug show the counts of all VisualizationEvaluations of the matrix
    def getCounts(self):
        for viz in self.vizEvaluations:
            viz.getCounts()
        self.leftLegend.getCounts()
        self.bottomLegend.getCounts()

    #show data to ink ratio as arithmetic mean of individual data:ink ratios
    def printDataToInkRatio(self):
        print("----------DATA TO INK: ")
        data = 0
        other = 0
        ink = 0
        for viz in self.vizEvaluations:
            data += viz.DIcount
            other += viz.NDIcount

        ink = data + other
        if ink > 0:
            dataPercent = data / ink * 100.0
            nonDataPercent = other / ink * 100.0
        else:
            dataPercent = -1.0
            nonDataPercent = -1.0
        print("data : ink = ", "{0:.2f}".format(dataPercent), ":",
            "{0:.2f}".format(nonDataPercent))

    #show FG:BG ratio of the whole matrix image
    def printFGBGRatio(self):
        print("----------FG TO BG: ")
        fg = 0
        bg = 0
```

```
        fg += self.leftLegend.DIcount
        fg += self.leftLegend.NDIcount
        bg += self.leftLegend.BGcount

        fg += self.bottomLegend.DIcount
        fg += self.bottomLegend.NDIcount
        bg += self.bottomLegend.BGcount

        for viz in self.vizEvaluations:
            fg += viz.DIcount
            fg += viz.NDIcount
            bg += viz.BGcount

        foregroundPercent = fg / self.count * 100.0
        backgroundPercent = (self.count - fg) / self.count * 100.0
        print("FG : BG = ", "{0:.2f}".format(foregroundPercent), ":",
            "{0:.2f}".format(backgroundPercent))

    #show the pixel differences of v-plot sides in % as arithmetic mean of individual v-plots
    def printNumDifferences(self):
        #print("---------PLOT DIFFERENCES: ")
        diffCount = 0
        absCount = 0
        numPlots = 0
        for viz in self.vizEvaluations:
            if viz.DIcount > 0:
                differences = viz.printNumDifferences(False)
                diffCount += differences[0]
                absCount += differences[1]
                numPlots += 1
        if diffCount > 0:
            print(diffCount, absCount)
            print("Number of different pixels between plot sides (sum of all plot diffs): ",
                diffCount)
            print("Percent different pixels of amount pixels per plot side (sum of all plot diff
                %s): ",
              "{0:.2f}".format(diffCount / absCount * 100), "%")
        else:
            print("Number of different pixels between plot sides (sum of all plot diffs): ", -1.0)
            print("Percent different pixels of amount pixels per plot side (sum of all plot diff
                %s): ",
            # -1.0)
            print(self.results)
```

A.4 Evaluation results table

Table 1: The calculated ratios of the test set as computed by the evaluation script.

numBins	dimension	complexity	resolution (wxh)	data %	ink %	FG %	BG %	diffPixels %
10	1	c0	100x160	41.03	58.97	21.32	78.67	13.62
10	1	c0	180x320	52.98	47.02	17.98	82.02	10.79
10	1	c0	360x640	69.57	30.43	13.54	86.46	9.37
10	1	c0	414x896	74.53	25.47	12.91	87.09	9.17
10	1	c0	1366x768	85.79	14.21	11.86	88.14	8.94
30	1	c0	100x160	17.05	82.95	16.53	83.47	5.71
30	1	c0	180x320	26.08	73.92	12.29	87.71	4.69
30	1	c0	360x640	41.98	58.02	7.68	92.32	4.26
30	1	c0	414x896	47.93	52.07	6.85	93.15	4.15
30	1	c0	1366x768	64.96	35.04	5.25	94.75	3.91
50	1	c0	100x160	11.78	88.22	15.66	84.34	4.12
50	1	c0	180x320	18.07	81.93	11.16	88.84	3.22
50	1	c0	360x640	30.68	69.32	6.48	93.52	2.87
50	1	c0	414x896	35.91	64.09	5.60	94.40	2.77
50	1	c0	1366x768	52.52	47.48	3.92	96.08	2.53
5	1	c0	100x160	56.24	43.76	26.52	73.47	17.91
5	1	c0	180x320	69.53	30.47	25.69	74.31	16.50
5	1	c0	360x640	83.06	16.94	22.51	77.49	16.07
5	1	c0	414x896	86.33	13.67	22.18	77.82	15.86
5	1	c0	1366x768	92.99	7.01	21.68	78.32	15.54
75	1	c0	100x160	9.03	90.97	15.23	84.77	3.28
75	1	c0	180x320	13.65	86.35	10.61	89.39	2.47
75	1	c0	360x640	23.27	76.73	5.87	94.13	2.10
75	1	c0	414x896	27.66	72.34	4.98	95.02	2.01
75	1	c0	1366x768	42.47	57.53	3.25	96.75	1.77
10	1	c1	100x160	46.27	53.73	23.11	76.89	16.01
10	1	c1	180x320	57.33	42.67	19.35	80.65	13.40
10	1	c1	360x640	72.19	27.81	14.82	85.18	11.59
10	1	c1	414x896	76.87	23.13	14.00	86.00	11.30
10	1	c1	1366x768	86.81	13.19	12.32	87.68	11.18
30	1	c1	100x160	30.73	69.27	19.02	80.98	9.50
30	1	c1	180x320	35.63	64.37	13.82	86.18	6.25

30	1	c1	360x640	50.74	49.26	8.93	91.07	4.57
30	1	c1	414x896	55.57	44.43	7.90	92.10	4.43
30	1	c1	1366x768	67.89	32.11	5.64	94.36	4.20
50	1	c1	100x160	26.42	73.58	18.24	81.76	7.56
50	1	c1	180x320	28.65	71.35	12.61	87.39	4.58
50	1	c1	360x640	42.48	57.52	7.70	92.30	3.13
50	1	c1	414x896	47.12	52.88	6.69	93.31	2.97
50	1	c1	1366x768	56.88	43.12	4.27	95.73	2.73
5	1	c1	100x160	61.99	38.01	28.84	71.16	22.93
5	1	c1	180x320	72.65	27.35	26.95	73.05	20.88
5	1	c1	360x640	84.42	15.58	23.53	76.47	19.85
5	1	c1	414x896	87.59	12.41	22.99	77.01	19.57
5	1	c1	1366x768	93.61	6.39	22.22	77.78	19.84
75	1	c1	100x160	23.93	76.07	17.71	82.29	6.72
75	1	c1	180x320	25.15	74.85	12.05	87.95	3.80
75	1	c1	360x640	37.80	62.20	7.14	92.86	2.31
75	1	c1	414x896	41.43	58.57	6.06	93.94	2.15
75	1	c1	1366x768	48.35	51.65	3.59	96.41	1.87
10	1	c2	100x160	53.08	46.92	25.87	74.13	14.37
10	1	c2	180x320	66.77	33.23	23.80	76.20	10.94
10	1	c2	360x640	81.27	18.73	20.78	79.22	9.96
10	1	c2	414x896	84.55	15.45	20.13	79.87	9.84
10	1	c2	1366x768	91.78	8.22	18.76	81.24	9.73
30	1	c2	100x160	48.07	51.93	24.00	76.00	12.16
30	1	c2	180x320	61.40	38.60	21.14	78.86	9.52
30	1	c2	360x640	76.93	23.07	17.83	82.17	8.76
30	1	c2	414x896	80.78	19.22	17.05	82.95	8.60
30	1	c2	1366x768	89.48	10.52	15.39	84.61	8.27
50	1	c2	100x160	48.36	51.64	23.84	76.16	11.01
50	1	c2	180x320	60.86	39.14	20.80	79.20	8.93
50	1	c2	360x640	77.15	22.85	17.50	82.50	8.07
50	1	c2	414x896	80.89	19.11	16.72	83.28	7.91
50	1	c2	1366x768	89.20	10.80	14.99	85.01	7.72
5	1	c2	100x160	66.17	33.83	31.74	68.26	20.80
5	1	c2	180x320	78.07	21.93	31.89	68.11	18.12
5	1	c2	360x640	88.47	11.53	29.85	70.15	17.29

5	1	c2	414x896	90.88	9.12	29.53	70.47	17.02
5	1	c2	1366x768	95.46	4.54	29.09	70.91	17.11
75	1	c2	100x160	48.26	51.74	23.72	76.28	10.82
75	1	c2	180x320	60.58	39.42	20.66	79.34	8.74
75	1	c2	360x640	76.86	23.14	17.37	82.63	7.81
75	1	c2	414x896	80.67	19.33	16.55	83.45	7.66
75	1	c2	1366x768	89.05	10.95	14.79	85.21	7.44
10	3	c0	100x160	100.00	0.00	4.43	95.57	6.86
10	3	c0	180x320	40.26	59.74	6.04	93.96	6.77
10	3	c0	360x640	43.86	56.14	6.88	93.12	7.05
10	3	c0	414x896	50.30	49.70	6.09	93.91	7.15
10	3	c0	1366x768	61.77	38.23	4.81	95.19	7.32
30	3	c0	100x160	100.00	0.00	3.62	96.38	2.70
30	3	c0	180x320	15.39	84.61	5.10	94.90	2.58
30	3	c0	360x640	17.34	82.66	5.43	94.57	2.74
30	3	c0	414x896	21.24	78.76	4.45	95.55	2.81
30	3	c0	1366x768	30.44	69.56	3.00	97.00	2.94
50	3	c0	100x160	100.00	0.00	3.51	96.49	1.47
50	3	c0	180x320	10.86	89.14	4.97	95.03	1.94
50	3	c0	360x640	11.99	88.01	5.23	94.77	2.02
50	3	c0	414x896	14.95	85.05	4.22	95.78	2.04
50	3	c0	1366x768	22.33	77.67	2.74	97.26	2.17
5	3	c0	100x160	100.00	0.00	5.51	94.49	9.80
5	3	c0	180x320	60.56	39.44	7.45	92.55	10.43
5	3	c0	360x640	64.90	35.10	9.00	91.00	10.70
5	3	c0	414x896	70.78	29.22	8.50	91.50	10.85
5	3	c0	1366x768	78.86	21.14	7.46	92.54	11.10
75	3	c0	100x160	100.00	0.00	3.39	96.61	1.10
75	3	c0	180x320	5.19	94.81	4.82	95.18	0.94
75	3	c0	360x640	6.14	93.86	5.02	94.98	1.10
75	3	c0	414x896	7.87	92.13	3.99	96.01	1.04
75	3	c0	1366x768	12.40	87.60	2.47	97.53	1.17
10	3	c1	100x160	100.00	0.00	5.58	94.42	15.32
10	3	c1	180x320	51.90	48.10	6.61	93.39	11.01
10	3	c1	360x640	49.24	50.76	7.02	92.98	9.79
10	3	c1	414x896	55.18	44.82	6.29	93.71	9.53

10	3	c1	1366x768	64.65	35.35	4.89	95.11	9.20
30	3	c1	100x160	100.00	0.00	4.89	95.11	15.32
30	3	c1	180x320	31.14	68.86	5.67	94.33	7.02
30	3	c1	360x640	25.18	74.82	5.59	94.41	5.03
30	3	c1	414x896	28.96	71.04	4.59	95.41	4.09
30	3	c1	1366x768	34.85	65.15	3.07	96.93	3.39
50	3	c1	100x160	100.00	0.00	4.81	95.19	15.56
50	3	c1	180x320	27.18	72.82	5.53	94.47	6.26
50	3	c1	360x640	20.16	79.84	5.38	94.62	4.26
50	3	c1	414x896	23.06	76.94	4.34	95.66	3.32
50	3	c1	1366x768	27.49	72.51	2.82	97.18	2.57
5	3	c1	100x160	100.00	0.00	6.56	93.44	16.30
5	3	c1	180x320	67.22	32.78	7.96	92.04	14.06
5	3	c1	360x640	68.14	31.86	9.21	90.79	13.07
5	3	c1	414x896	73.11	26.89	8.71	91.29	13.34
5	3	c1	1366x768	80.19	19.81	7.55	92.45	13.48
75	3	c1	100x160	100.00	0.00	4.70	95.30	15.56
75	3	c1	180x320	22.95	77.05	5.36	94.64	5.41
75	3	c1	360x640	14.75	85.25	5.15	94.85	3.28
75	3	c1	414x896	16.79	83.21	4.10	95.90	2.25
75	3	c1	1366x768	17.69	82.31	2.53	97.47	1.47
10	3	c2	100x160	100.00	0.00	5.71	94.29	14.46
10	3	c2	180x320	58.76	41.24	7.08	92.92	10.47
10	3	c2	360x640	59.86	40.14	8.16	91.84	9.88
10	3	c2	414x896	65.85	34.15	7.55	92.45	9.39
10	3	c2	1366x768	76.03	23.97	6.53	93.47	9.09
30	3	c2	100x160	100.00	0.00	5.24	94.76	14.22
30	3	c2	180x320	50.28	49.72	6.65	93.35	8.51
30	3	c2	360x640	51.39	48.61	7.40	92.60	8.35
30	3	c2	414x896	57.41	42.59	6.64	93.36	7.61
30	3	c2	1366x768	68.79	31.21	5.54	94.46	7.37
50	3	c2	100x160	100.00	0.00	5.22	94.78	14.22
50	3	c2	180x320	49.57	50.43	6.61	93.39	8.29
50	3	c2	360x640	50.45	49.55	7.34	92.66	8.01
50	3	c2	414x896	56.48	43.52	6.56	93.44	7.25
50	3	c2	1366x768	68.05	31.95	5.45	94.55	6.96

5	3	c2	100x160	100.00	0.00	6.66	93.34	15.44
5	3	c2	180x320	71.02	28.98	8.40	91.60	12.68
5	3	c2	360x640	73.50	26.50	10.26	89.74	12.09
5	3	c2	414x896	78.34	21.66	9.93	90.07	12.18
5	3	c2	1366x768	85.22	14.78	9.16	90.84	12.23
75	3	c2	100x160	100.00	0.00	5.19	94.81	14.83
75	3	c2	180x320	49.01	50.99	6.57	93.43	8.20
75	3	c2	360x640	49.77	50.23	7.29	92.71	7.72
75	3	c2	414x896	55.77	44.23	6.50	93.50	6.87
75	3	c2	1366x768	67.41	32.59	5.38	94.62	6.58
10	6	c0	100x160	100.00	0.00	3.49	96.51	7.31
10	6	c0	180x320	45.61	54.39	5.36	94.64	9.24
10	6	c0	360x640	36.30	63.70	7.38	92.62	9.65
10	6	c0	414x896	42.52	57.48	7.04	92.96	9.79
10	6	c0	1366x768	53.47	46.53	5.64	94.36	9.76
30	6	c0	100x160	100.00	0.00	2.94	97.06	2.69
30	6	c0	180x320	17.81	82.19	4.18	95.82	3.25
30	6	c0	360x640	13.20	86.80	6.06	93.94	3.51
30	6	c0	414x896	16.48	83.52	5.42	94.58	3.54
30	6	c0	1366x768	23.78	76.22	3.81	96.19	3.55
50	6	c0	100x160	100.00	0.00	2.88	97.12	2.56
50	6	c0	180x320	12.16	87.84	4.00	96.00	2.34
50	6	c0	360x640	9.08	90.92	5.87	94.13	2.47
50	6	c0	414x896	11.41	88.59	5.19	94.81	2.48
50	6	c0	1366x768	25.91	74.09	4.17	95.83	4.48
5	6	c0	100x160	100.00	0.00	4.38	95.62	14.74
5	6	c0	180x320	66.01	33.99	7.16	92.84	16.99
5	6	c0	360x640	57.42	42.58	9.37	90.63	17.37
5	6	c0	414x896	63.74	36.26	9.40	90.60	17.51
5	6	c0	1366x768	72.45	27.55	8.34	91.66	17.50
75	6	c0	100x160	100.00	0.00	2.76	97.24	0.92
75	6	c0	180x320	6.14	93.86	3.84	96.16	1.20
75	6	c0	360x640	4.51	95.49	5.69	94.31	1.26
75	6	c0	414x896	5.77	94.23	4.97	95.03	1.26
75	6	c0	1366x768	9.13	90.87	3.29	96.71	1.28
10	6	c1	100x160	100.00	0.00	4.98	95.03	16.15

10	6	c1	180x320	60.55	39.45	6.29	93.71	13.96
10	6	c1	360x640	43.29	56.71	7.59	92.41	12.87
10	6	c1	414x896	49.64	50.36	7.28	92.72	12.85
10	6	c1	1366x768	56.63	43.37	5.79	94.21	12.32
30	6	c1	100x160	100.00	0.00	4.49	95.51	13.33
30	6	c1	180x320	41.53	58.47	5.26	94.74	9.26
30	6	c1	360x640	22.32	77.68	6.29	93.71	5.91
30	6	c1	414x896	25.40	74.60	5.61	94.39	5.47
30	6	c1	1366x768	28.40	71.60	3.99	96.01	4.36
50	6	c1	100x160	100.00	0.00	4.46	95.54	13.33
50	6	c1	180x320	37.98	62.02	5.10	94.90	8.25
50	6	c1	360x640	18.26	81.74	6.06	93.94	4.73
50	6	c1	414x896	20.79	79.21	5.37	94.63	4.30
50	6	c1	1366x768	21.96	78.04	3.71	96.29	3.20
5	6	c1	100x160	100.00	0.00	5.70	94.30	22.31
5	6	c1	180x320	74.17	25.83	7.96	92.04	20.86
5	6	c1	360x640	62.22	37.78	9.76	90.24	22.23
5	6	c1	414x896	67.58	32.42	9.71	90.29	22.05
5	6	c1	1366x768	74.12	25.88	8.55	91.45	21.85
75	6	c1	100x160	100.00	0.00	4.44	95.56	13.85
75	6	c1	180x320	34.06	65.94	4.89	95.11	7.12
75	6	c1	360x640	13.93	86.07	5.85	94.15	3.53
75	6	c1	414x896	15.39	84.61	5.11	94.89	3.06
75	6	c1	1366x768	14.64	85.36	3.44	96.56	1.88
10	6	c2	100x160	100.00	0.00	5.01	94.99	15.77
10	6	c2	180x320	65.57	34.43	6.77	93.23	13.77
10	6	c2	360x640	51.74	48.26	8.50	91.50	13.02
10	6	c2	414x896	58.96	41.04	8.38	91.62	13.20
10	6	c2	1366x768	68.68	31.32	7.37	92.63	13.16
30	6	c2	100x160	100.00	0.00	4.54	95.46	12.95
30	6	c2	180x320	57.27	42.73	6.19	93.81	10.94
30	6	c2	360x640	41.74	58.26	7.81	92.19	10.01
30	6	c2	414x896	49.69	50.31	7.46	92.54	9.98
30	6	c2	1366x768	60.46	39.54	6.37	93.63	9.70
50	6	c2	100x160	100.00	0.00	4.54	95.46	13.08
50	6	c2	180x320	56.50	43.50	6.14	93.86	10.56

50	6	c2	360x640	40.49	59.51	7.71	92.29	9.27
50	6	c2	414x896	48.83	51.17	7.37	92.63	9.41
50	6	c2	1366x768	59.62	40.38	6.26	93.74	9.15
5	6	c2	100x160	100.00	0.00	5.78	94.23	21.28
5	6	c2	180x320	77.07	22.93	8.45	91.55	20.07
5	6	c2	360x640	67.51	32.49	10.61	89.39	21.33
5	6	c2	414x896	73.17	26.83	10.83	89.17	21.36
5	6	c2	1366x768	80.05	19.95	10.17	89.83	21.59
75	6	c2	100x160	100.00	0.00	4.52	95.48	13.46
75	6	c2	180x320	55.99	44.01	6.08	93.92	10.37
75	6	c2	360x640	39.37	60.63	7.66	92.34	8.82
75	6	c2	414x896	47.75	52.25	7.29	92.71	8.93
75	6	c2	1366x768	58.84	41.16	6.17	93.83	8.57
10	10	c0	100x160	100.00	0.00	3.28	96.72	6.84
10	10	c0	180x320	100.00	0.00	2.75	97.25	7.83
10	10	c0	360x640	59.96	40.04	5.39	94.61	8.57
10	10	c0	414x896	38.33	61.67	8.85	91.15	8.37
10	10	c0	1366x768	54.26	45.74	7.05	92.95	8.50
30	10	c0	100x160	100.00	0.00	2.92	97.08	2.63
30	10	c0	180x320	100.00	0.00	1.33	98.67	3.11
30	10	c0	360x640	28.70	71.30	3.44	96.56	3.60
30	10	c0	414x896	14.11	85.89	7.12	92.88	3.51
30	10	c0	1366x768	24.17	75.83	4.75	95.25	3.59
50	10	c0	100x160	100.00	0.00	2.79	97.21	1.75
50	10	c0	180x320	100.00	0.00	1.12	98.88	2.24
50	10	c0	360x640	20.39	79.61	3.14	96.86	2.62
50	10	c0	414x896	9.66	90.34	6.88	93.12	2.55
50	10	c0	1366x768	17.34	82.66	4.43	95.57	2.62
5	10	c0	100x160	100.00	0.00	3.98	96.02	15.26
5	10	c0	180x320	100.00	0.00	4.80	95.20	14.56
5	10	c0	360x640	77.45	22.55	8.29	91.71	15.67
5	10	c0	414x896	59.85	40.15	11.36	88.64	15.25
5	10	c0	1366x768	73.59	26.41	10.43	89.57	15.47
75	10	c0	100x160	100.00	0.00	2.79	97.21	1.75
75	10	c0	180x320	100.00	0.00	0.94	99.06	1.22
75	10	c0	360x640	10.80	89.20	2.87	97.13	1.28

75	10	c0	414x896	4.86	95.14	6.65	93.35	1.29
75	10	c0	1366x768	9.27	90.73	4.10	95.90	1.35
10	10	c1	100x160	100.00	0.00	5.31	94.69	20.18
10	10	c1	180x320	100.00	0.00	4.39	95.61	13.50
10	10	c1	360x640	68.69	31.31	6.14	93.86	11.78
10	10	c1	414x896	46.53	53.47	9.18	90.82	11.50
10	10	c1	1366x768	58.63	41.37	7.18	92.82	10.70
30	10	c1	100x160	100.00	0.00	5.05	94.95	23.51
30	10	c1	180x320	100.00	0.00	3.11	96.89	11.31
30	10	c1	360x640	44.68	55.32	4.23	95.77	6.32
30	10	c1	414x896	24.44	75.56	7.38	92.62	5.74
30	10	c1	1366x768	29.40	70.60	4.89	95.11	4.41
50	10	c1	100x160	100.00	0.00	4.98	95.03	25.61
50	10	c1	180x320	100.00	0.00	2.86	97.14	10.80
50	10	c1	360x640	39.91	60.09	3.97	96.03	5.25
50	10	c1	414x896	20.16	79.84	7.11	92.89	4.75
50	10	c1	1366x768	22.85	77.15	4.55	95.45	3.35
5	10	c1	100x160	100.00	0.00	5.79	94.21	21.23
5	10	c1	180x320	100.00	0.00	6.20	93.80	20.45
5	10	c1	360x640	81.86	18.14	9.02	90.98	19.95
5	10	c1	414x896	64.32	35.68	11.72	88.28	19.64
5	10	c1	1366x768	76.07	23.93	10.61	89.39	19.34
75	10	c1	100x160	100.00	0.00	4.92	95.08	27.02
75	10	c1	180x320	100.00	0.00	2.65	97.35	9.79
75	10	c1	360x640	33.30	66.70	3.65	96.35	4.12
75	10	c1	414x896	15.79	84.21	6.85	93.15	3.58
75	10	c1	1366x768	15.04	84.96	4.20	95.80	1.98
10	10	c2	100x160	100.00	0.00	5.38	94.62	19.82
10	10	c2	180x320	100.00	0.00	4.73	95.27	13.62
10	10	c2	360x640	75.19	24.81	7.14	92.86	12.27
10	10	c2	414x896	54.64	45.36	10.19	89.81	12.00
10	10	c2	1366x768	68.19	31.81	8.70	91.30	11.55
30	10	c2	100x160	100.00	0.00	5.05	94.95	23.51
30	10	c2	180x320	100.00	0.00	3.82	96.18	11.89
30	10	c2	360x640	65.49	34.51	5.90	94.10	9.57
30	10	c2	414x896	43.70	56.30	9.03	90.97	9.29

30	10	c2	1366x768	57.75	42.25	7.20	92.80	8.66
50	10	c2	100x160	100.00	0.00	4.99	95.01	25.09
50	10	c2	180x320	100.00	0.00	3.73	96.27	11.58
50	10	c2	360x640	64.73	35.27	5.80	94.20	8.99
50	10	c2	414x896	42.59	57.41	8.92	91.08	8.67
50	10	c2	1366x768	56.72	43.28	7.06	92.94	8.08
5	10	c2	100x160	100.00	0.00	5.89	94.11	18.60
5	10	c2	180x320	100.00	0.00	6.63	93.37	19.62
5	10	c2	360x640	84.97	15.03	10.06	89.94	19.33
5	10	c2	414x896	69.84	30.16	12.80	87.20	19.11
5	10	c2	1366x768	80.38	19.62	12.25	87.75	18.78
75	10	c2	100x160	100.00	0.00	4.93	95.07	26.84
75	10	c2	180x320	100.00	0.00	3.68	96.32	11.08
75	10	c2	360x640	63.84	36.16	5.69	94.31	8.64
75	10	c2	414x896	41.61	58.39	8.83	91.17	8.31
75	10	c2	1366x768	55.70	44.30	6.93	93.07	7.46
10	15	c0	100x160	100.00	0.00	3.15	96.85	4.94
10	15	c0	180x320	100.00	0.00	2.61	97.39	8.53
10	15	c0	360x640	54.57	45.43	5.89	94.11	10.82
10	15	c0	414x896	64.04	35.96	5.34	94.66	10.36
10	15	c0	1366x768	49.54	50.46	7.93	92.07	10.71
30	15	c0	100x160	100.00	0.00	2.81	97.19	1.85
30	15	c0	180x320	100.00	0.00	1.28	98.72	3.21
30	15	c0	360x640	24.18	75.82	3.96	96.04	4.19
30	15	c0	414x896	32.20	67.80	3.19	96.81	4.03
30	15	c0	1366x768	20.84	79.16	5.62	94.38	4.17
50	15	c0	100x160	100.00	0.00	2.78	97.22	1.85
50	15	c0	180x320	100.00	0.00	1.09	98.91	2.26
50	15	c0	360x640	17.22	82.78	3.70	96.30	2.97
50	15	c0	414x896	23.58	76.42	2.88	97.12	2.85
50	15	c0	1366x768	14.81	85.19	5.31	94.69	2.95
5	15	c0	100x160	100.00	0.00	3.57	96.43	8.15
5	15	c0	180x320	100.00	0.00	4.47	95.53	13.10
5	15	c0	360x640	73.62	26.38	8.76	91.24	16.75
5	15	c0	414x896	80.65	19.35	8.55	91.45	16.18
5	15	c0	1366x768	70.08	29.92	11.36	88.64	16.57

75	15	c0	100x160	-1.00	-1.00	2.77	97.22	-1.00
75	15	c0	180x320	100.00	0.00	0.90	99.10	1.10
75	15	c0	360x640	8.95	91.05	3.42	96.58	1.48
75	15	c0	414x896	12.95	87.05	2.58	97.42	1.45
75	15	c0	1366x768	7.72	92.28	4.98	95.02	1.50
10	15	c1	100x160	100.00	0.00	5.28	94.72	16.67
10	15	c1	180x320	100.00	0.00	4.46	95.54	15.18
10	15	c1	360x640	65.47	34.53	6.83	93.17	15.04
10	15	c1	414x896	72.27	27.73	6.14	93.86	13.89
10	15	c1	1366x768	54.48	45.52	8.11	91.89	13.05
30	15	c1	100x160	100.00	0.00	5.08	94.92	19.63
30	15	c1	180x320	100.00	0.00	3.36	96.64	12.97
30	15	c1	360x640	42.95	57.05	4.95	95.05	7.86
30	15	c1	414x896	49.45	50.55	4.06	95.94	6.70
30	15	c1	1366x768	26.50	73.50	5.80	94.20	5.18
50	15	c1	100x160	100.00	0.00	5.04	94.96	20.37
50	15	c1	180x320	100.00	0.00	3.18	96.82	12.44
50	15	c1	360x640	38.46	61.54	4.67	95.33	6.59
50	15	c1	414x896	44.14	55.86	3.75	96.25	5.37
50	15	c1	1366x768	20.59	79.41	5.46	94.54	3.88
5	15	c1	100x160	100.00	0.00	5.57	94.43	15.19
5	15	c1	180x320	100.00	0.00	6.14	93.86	18.24
5	15	c1	360x640	79.27	20.73	9.64	90.36	21.19
5	15	c1	414x896	84.32	15.68	9.22	90.78	19.84
5	15	c1	1366x768	72.28	27.72	11.43	88.57	19.72
75	15	c1	100x160	100.00	0.00	5.01	94.99	21.11
75	15	c1	180x320	100.00	0.00	3.07	96.93	12.33
75	15	c1	360x640	33.00	67.00	4.38	95.62	5.25
75	15	c1	414x896	37.55	62.45	3.42	96.58	4.11
75	15	c1	1366x768	13.66	86.34	5.10	94.90	2.32
10	15	c2	100x160	100.00	0.00	5.33	94.67	15.80
10	15	c2	180x320	100.00	0.00	4.73	95.27	14.95
10	15	c2	360x640	72.34	27.66	7.92	92.08	16.74
10	15	c2	414x896	79.07	20.93	7.50	92.50	15.93
10	15	c2	1366x768	65.51	34.49	9.90	90.10	15.80
30	15	c2	100x160	100.00	0.00	5.08	94.92	19.63

30	15	c2	180x320	100.00	0.00	3.98	96.02	13.83
30	15	c2	360x640	63.68	36.32	6.81	93.19	13.17
30	15	c2	414x896	71.45	28.55	6.24	93.76	12.37
30	15	c2	1366x768	56.12	43.88	8.54	91.46	12.01
50	15	c2	100x160	100.00	0.00	5.08	94.92	19.75
50	15	c2	180x320	100.00	0.00	3.90	96.10	13.60
50	15	c2	360x640	62.92	37.08	6.70	93.30	12.51
50	15	c2	414x896	70.67	29.33	6.12	93.88	11.65
50	15	c2	1366x768	55.11	44.89	8.39	91.61	11.30
5	15	c2	100x160	100.00	0.00	5.65	94.35	13.58
5	15	c2	180x320	100.00	0.00	6.42	93.58	17.51
5	15	c2	360x640	82.51	17.49	10.66	89.34	21.08
5	15	c2	414x896	87.33	12.67	10.54	89.46	20.27
5	15	c2	1366x768	77.62	22.38	13.27	86.73	20.41
75	15	c2	100x160	100.00	0.00	5.01	94.99	20.99
75	15	c2	180x320	100.00	0.00	3.86	96.14	13.83
75	15	c2	360x640	61.94	38.06	6.58	93.42	11.98
75	15	c2	414x896	69.76	30.24	5.98	94.02	11.14
75	15	c2	1366x768	53.94	46.06	8.24	91.76	10.48

A.5 T-test results of the discriminability ratio

Contains the p-values that resulted from the various performed t-tests for the discriminability between plot sides as ratio. These results are documented in the form of tables per categorization. The related hypotheses of the t-tests were discussed in subsection 5.0.2. The span of the cells containing the p-values indicate which test set pairing it belongs to. Meaning the left border of a cell marks the first test set used in the pairing and the right border of the cell resides in the column of the second test set. Green highlighted cells mark p-values smaller than the significance level $\alpha = 0.05$. The tables were all generated using Microsoft Excel [53].

A.5.1 Dimensionality

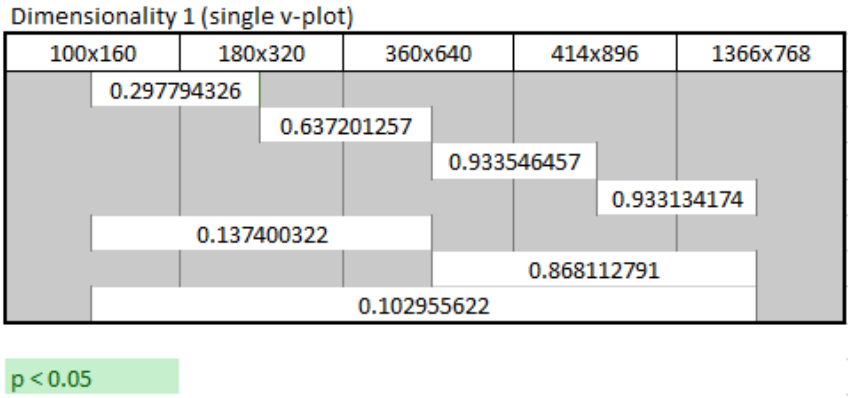


Figure 34: The results of various t-tests conducted on the measured discriminability ratio of v-plots of dimensionality 1 (single v-plots). No significant changes can be seen among the tested pairings.

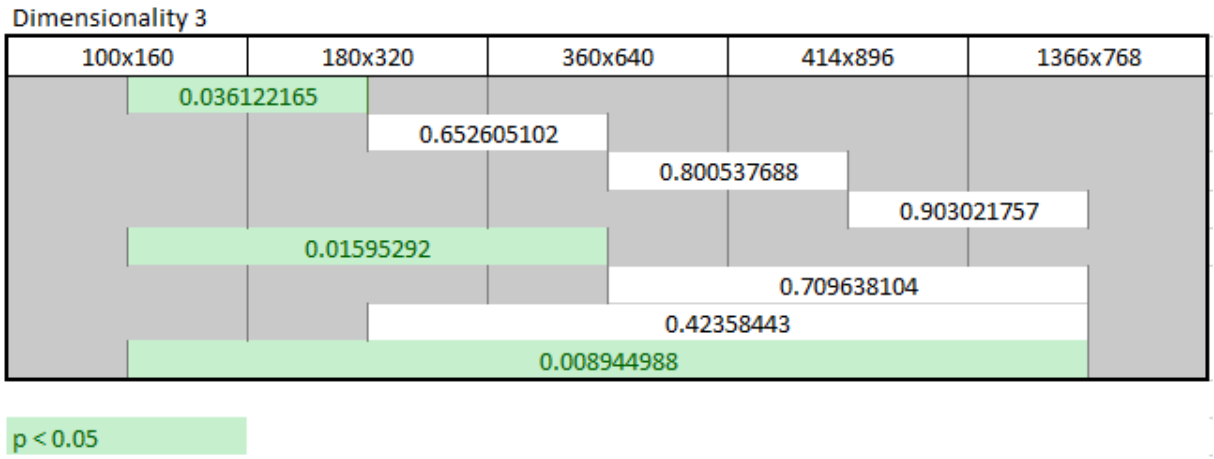


Figure 35: The results of various t-tests conducted on the measured discriminability ratio of v-plots of dimensionality 3. Several significant changes can be seen among tested pairs, resembling the the test results of the t-tests conducted on the test sets that use no further filtering (see Figure 32).

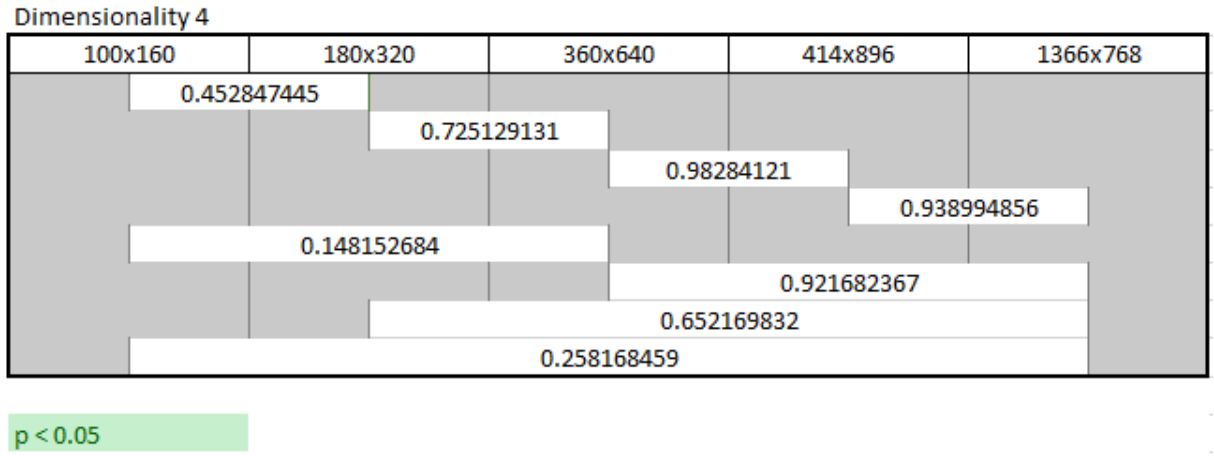


Figure 36: The results of various t-tests conducted on the measured discriminability ratio of v-plots of dimensionality 4. No significant changes can be seen among the tested pairings.

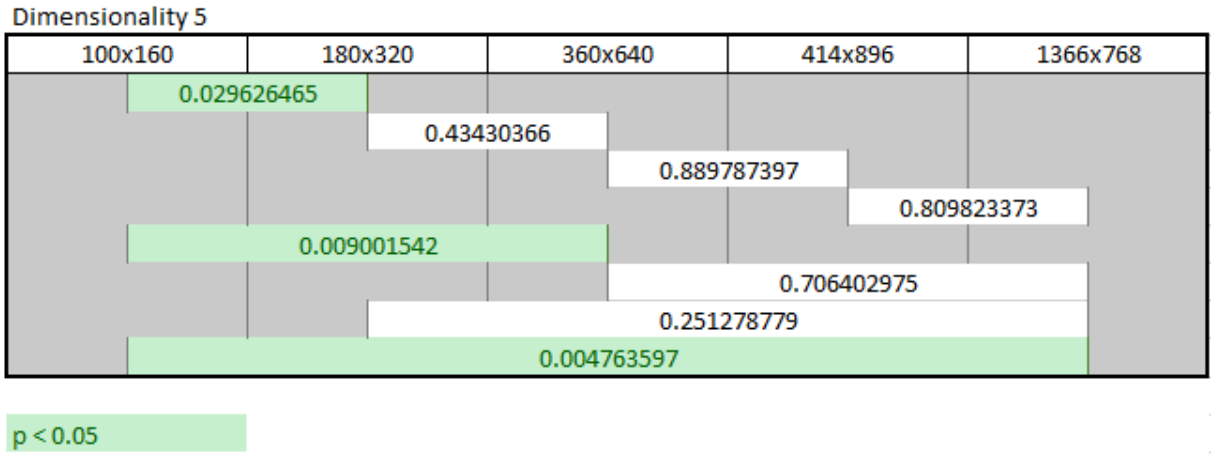


Figure 37: The results of various t-tests conducted on the measured discriminability ratio of v-plots of dimensionality 5. Several significant changes can be seen among tested pairs, resembling the the test results of the t-tests conducted on the test groups that use no further filtering.

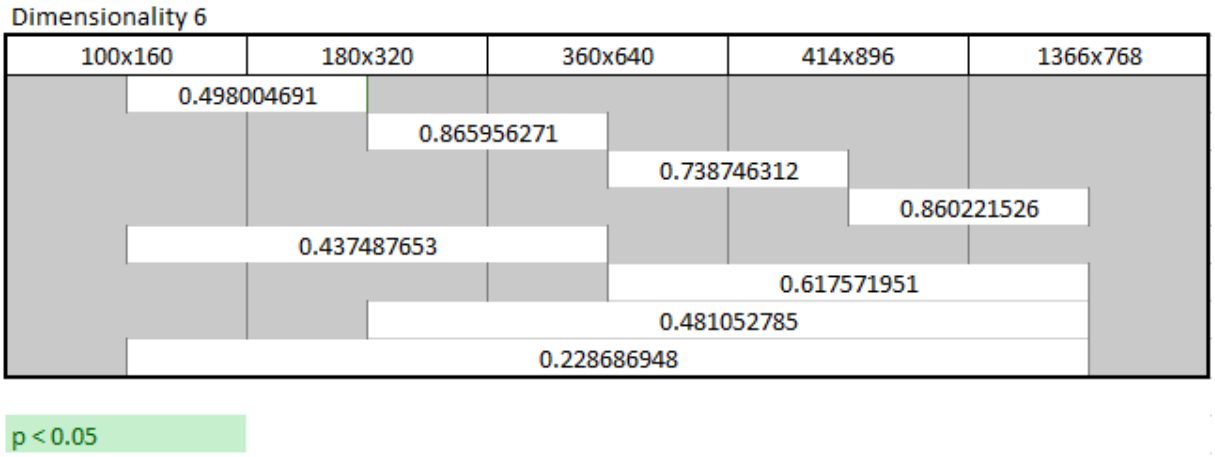


Figure 38: The results of various t-tests conducted on the measured discriminability ratio of v-plots of dimensionality 6. No significant changes can be seen among the tested pairings.

A.5.2 Complexity

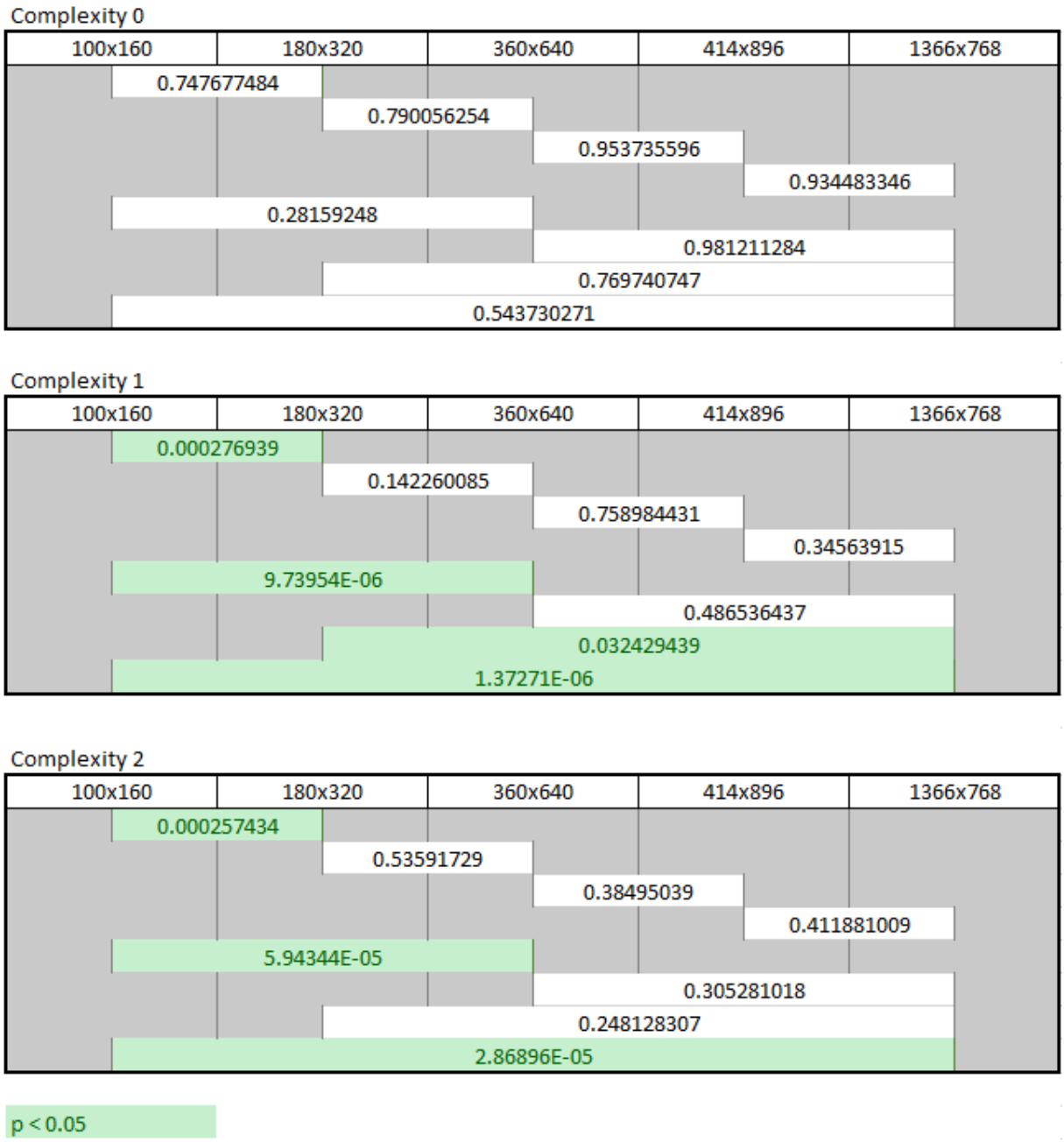


Figure 39: The results of various t-tests conducted on the measured discriminability ratio of v-plots using various complexities. **From top to bottom:** C0: No significant changes can be seen among the tested pairings. C1 and C2: Significant changes can be seen between some of the pairings, similar to the changes appearing in pairings of the whole test data set (see Figure 32).

A.5.3 Bin count

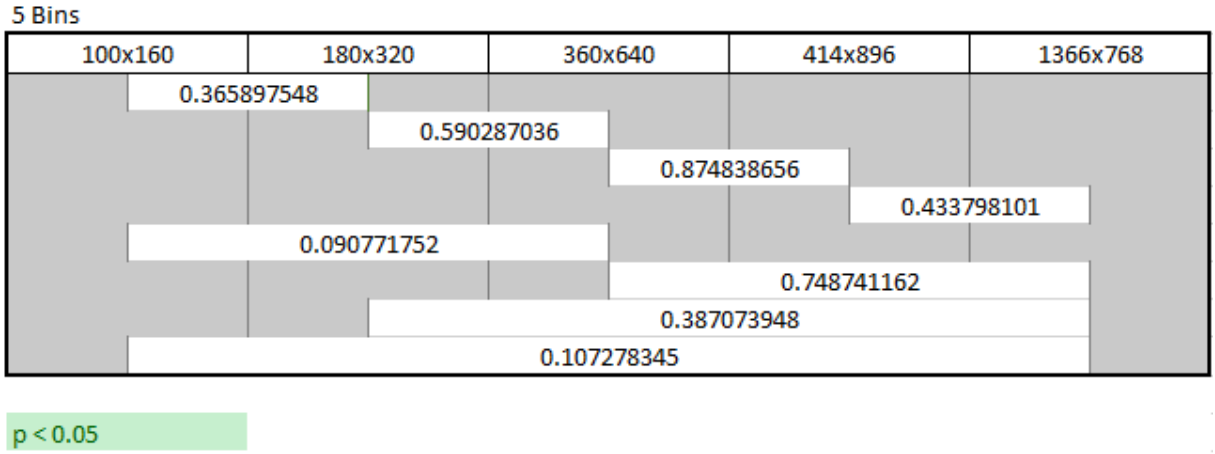


Figure 40: The results of various t-tests conducted on the measured discriminability ratio of v-plots using bin count 5.

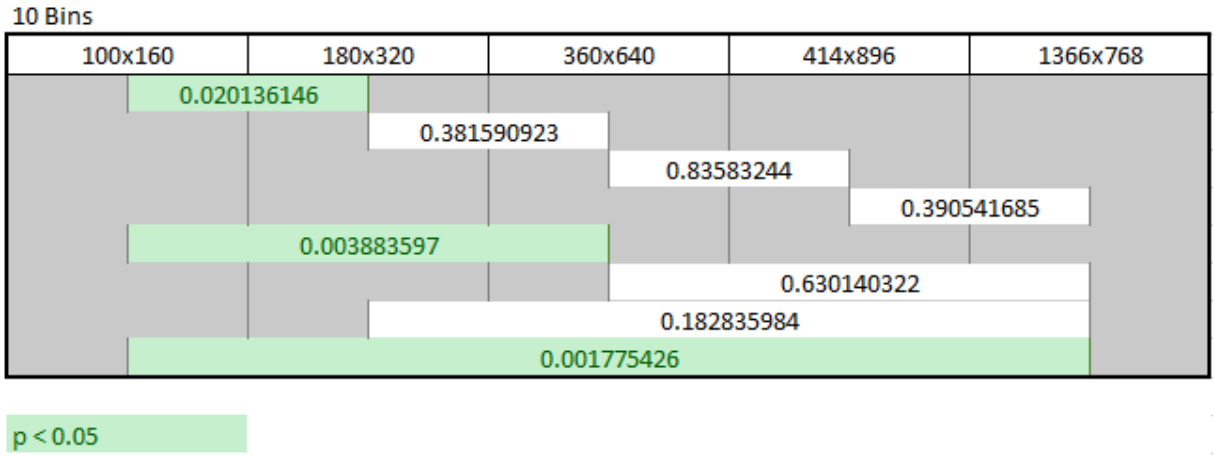


Figure 41: The results of various t-tests conducted on the measured discriminability ratio of v-plots using bin count 10. Various significant changes can be seen in pairings involving the smallest tested resolution.

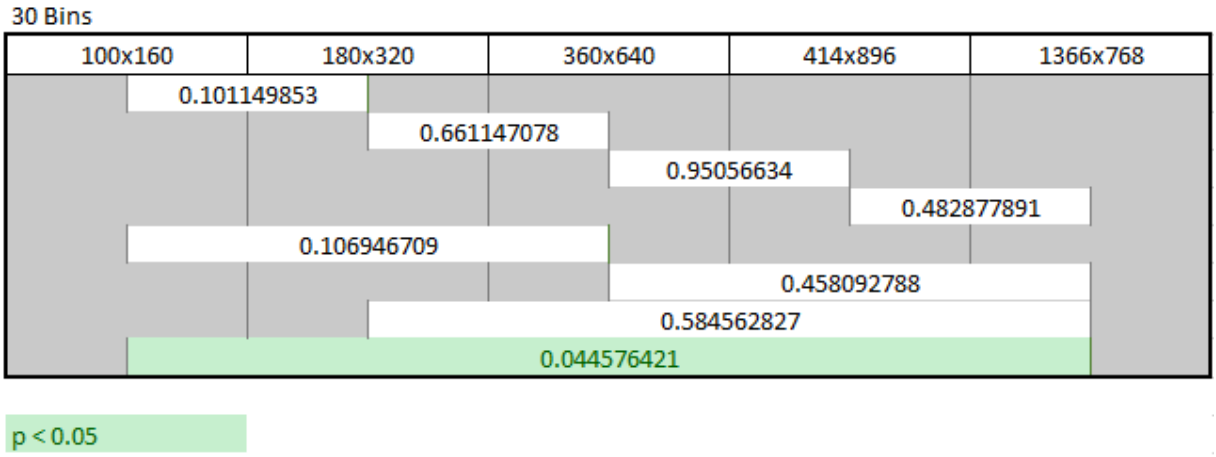


Figure 42: The results of various t-tests conducted on the measured discriminability ratio of v-plots using bin count 30. A significant change can only be seen between the smallest and the largest tested resolution.

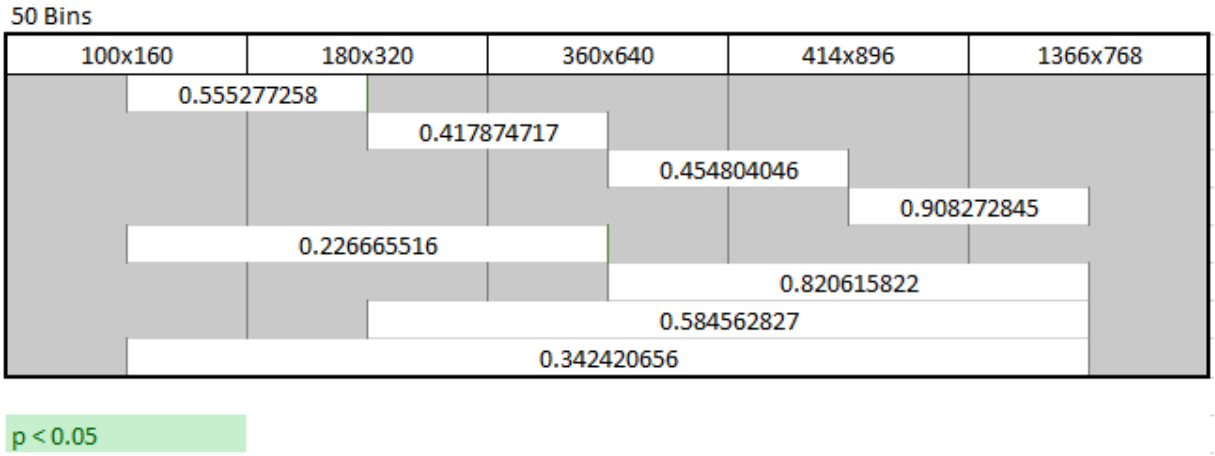


Figure 43: The results of various t-tests conducted on the measured discriminability ratio of v-plots using bin count 50. No significant changes can be seen among the tested pairings.

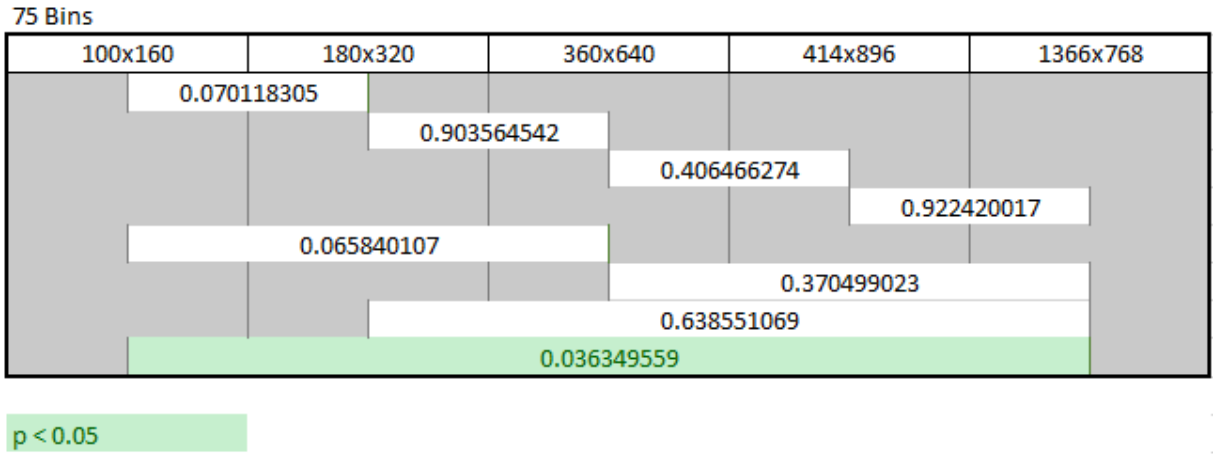


Figure 44: The results of various t-tests conducted on the measured discriminability ratio of v-plots using bin count 75. A significant change can only be seen between the smallest and the largest tested resolution.

A.6 Ratio results visualized

Several visual representations of the calculated ratios are contained in this section, sorted by their corresponding ratios. These visualizations were generated using Tableau [71].

A.6.1 Data-ink ratio

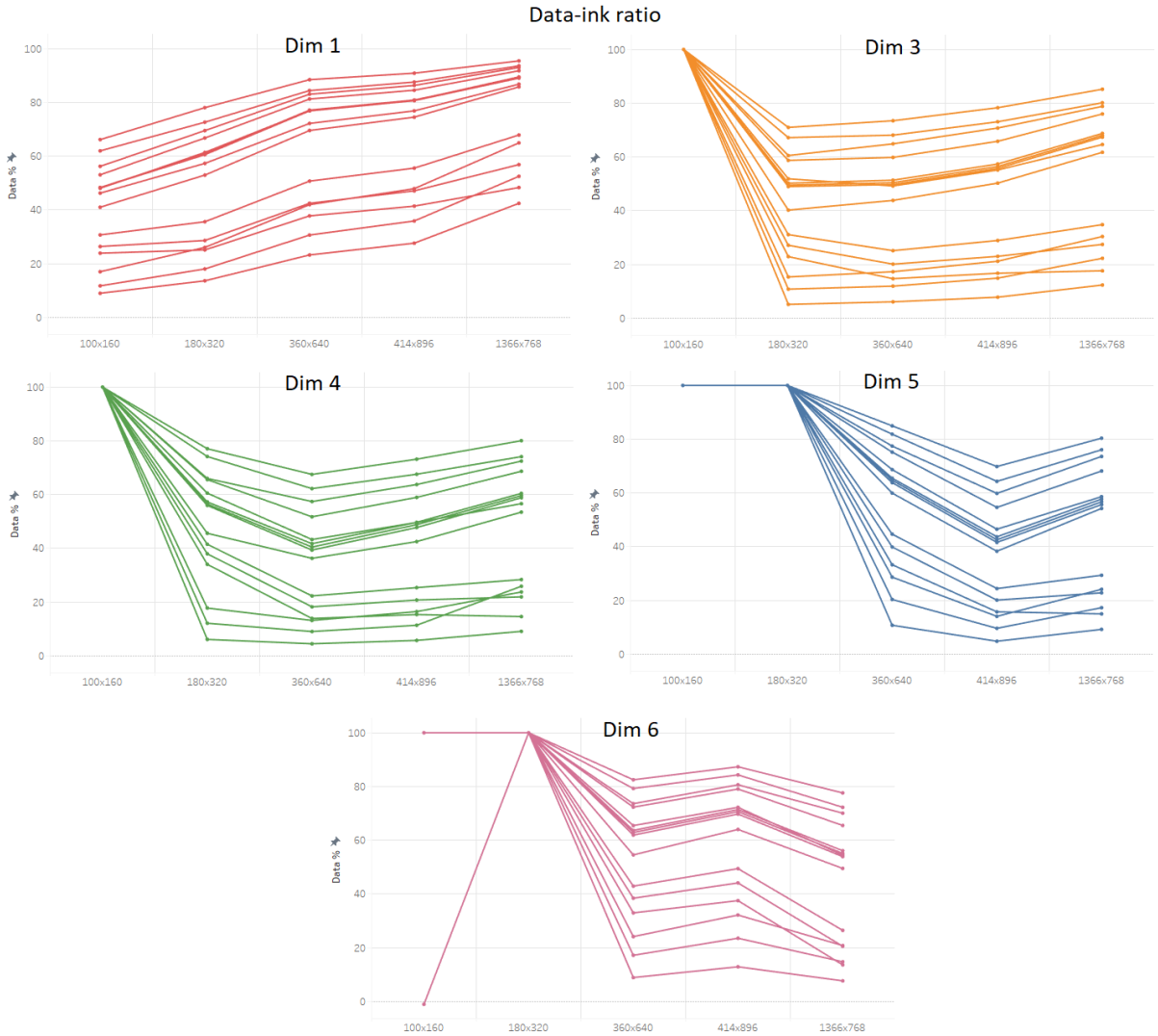


Figure 45: Shows the calculated data-ink ratio over time, categorized by dimensionality. Each plot contains 15 lines, being the result of different configurations (5 different bin sizes, 3 different complexities). Generally, a slight increase over growing resolutions can be seen in all plots. Deviations happen in v-plots with higher dimensionality, caused either by the scaling of the grid or the loss of ink. The mapping is fixed across plots.

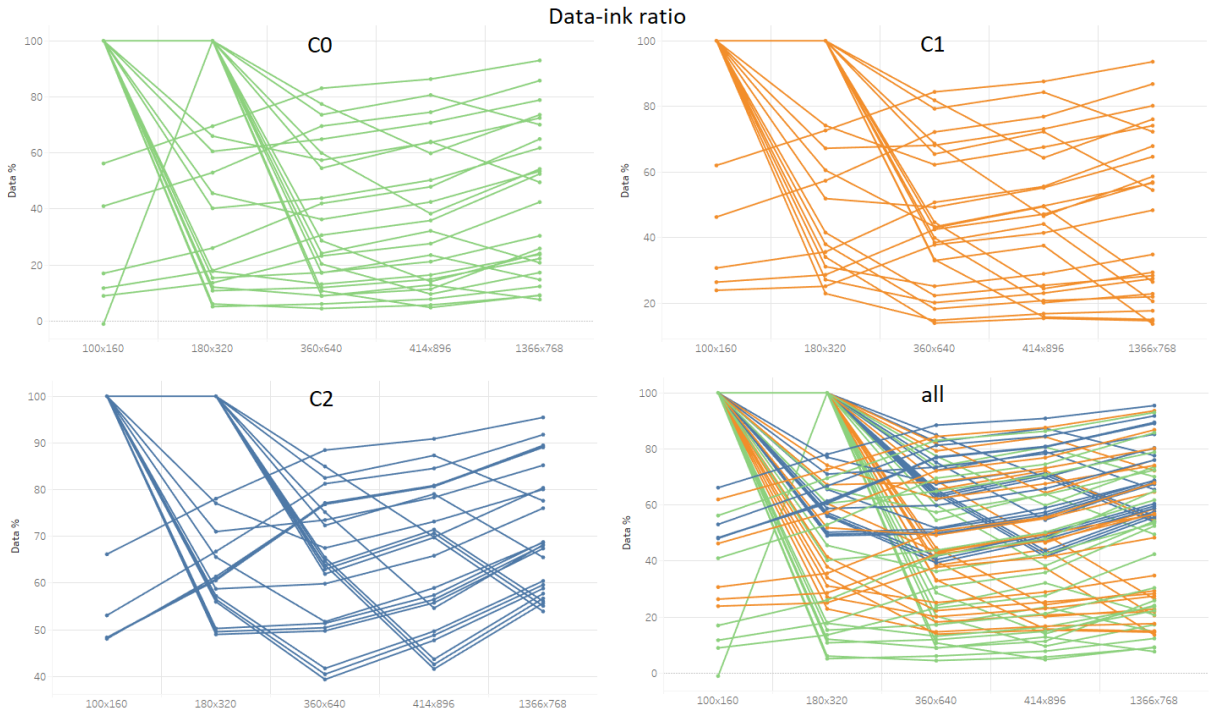


Figure 46: Shows the calculated data-ink ratio over time, categorized by complexity. The mapping is not fixed across plots. **Top left:** c0 (green). **Top right:** c1 (orange). **Bottom left:** c2 (blue). **Bottom right:** All configurations, colored by complexity. A higher complexity generally leads to a higher data-ink ratio.

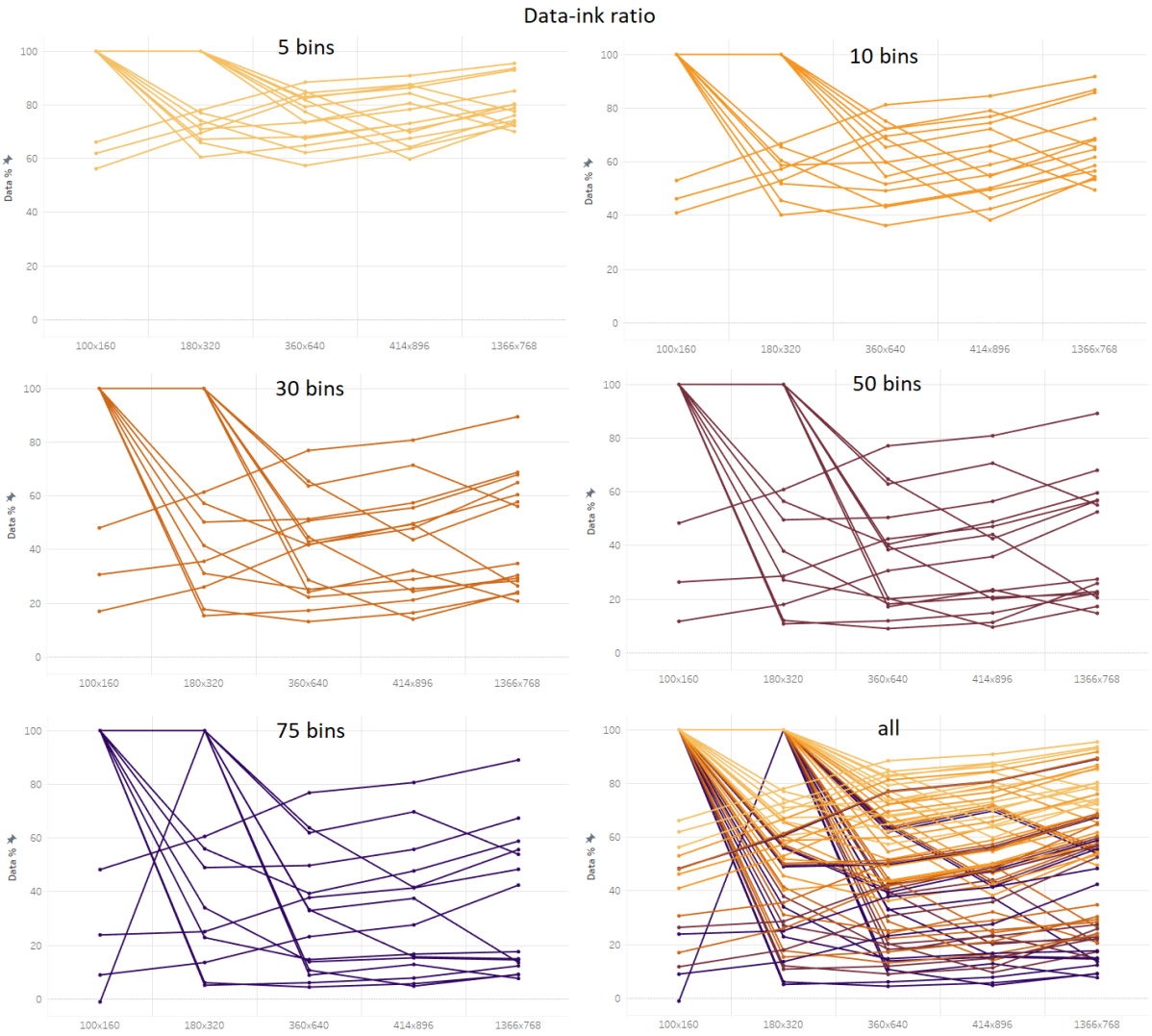


Figure 47: Shows the calculated data-ink ratio over time, categorized by bin count. Each plot contains 15 lines, being the result of different configurations (5 different dimensions, 3 different complexities). It shows that with a smaller bin count the data-ink ratio generally increases. The mapping is fixed across plots.

A.6.2 Foreground-background ratio

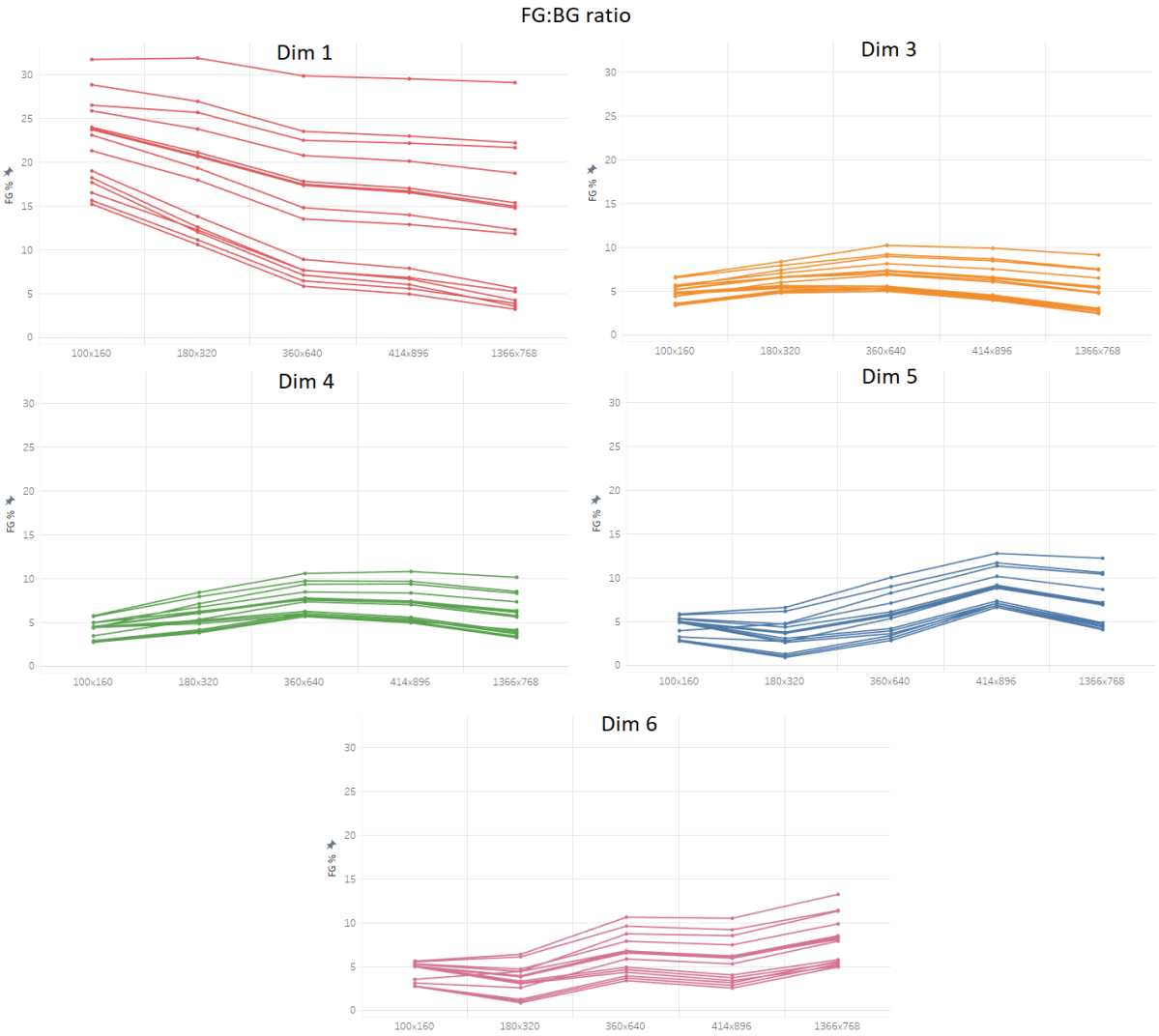


Figure 48: Shows the calculated FG:BG ratio over time, categorized by dimensionality. Each plot contains 15 lines, being the result of different configurations (5 different bin sizes, 3 different complexities). Single v-plots (dimensionality 1) can be seen to generally have a wider spread, reaching higher values. The mapping is fixed across plots.

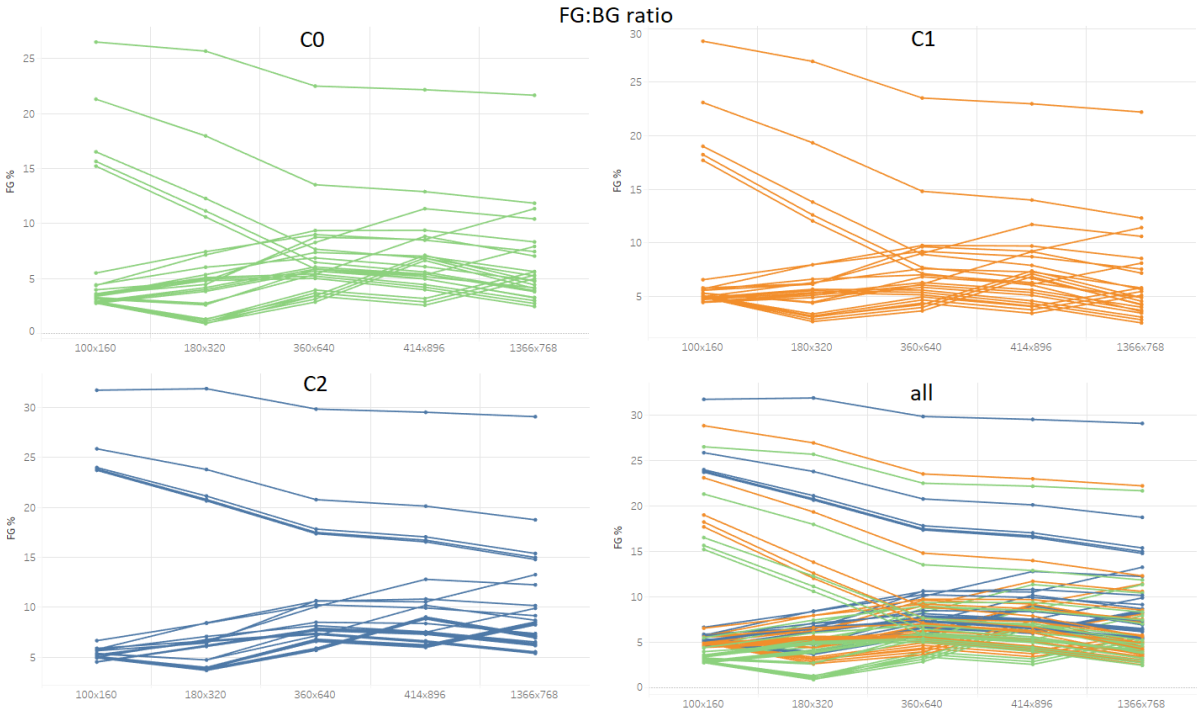


Figure 49: Shows the calculated FG:BG ratio over time, categorized by complexity. **Top left:** c0 (green). **Top right:** c1 (orange). **Bottom left:** c2 (blue). **Bottom right:** All configurations, colored by complexity. A higher complexity generally leads to a higher FG:BG ratio. The mapping is not fixed across plots.

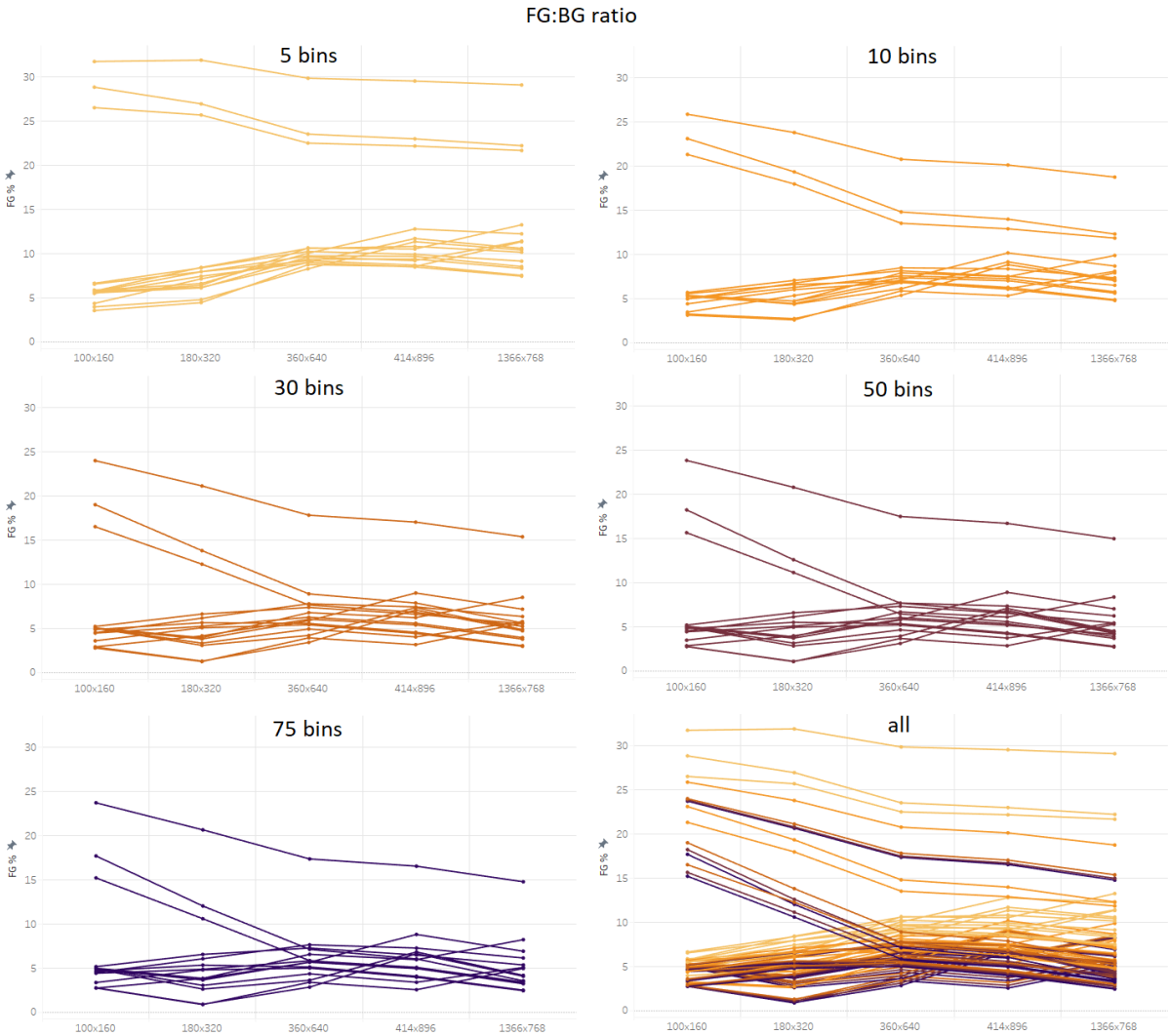


Figure 50: Shows the calculated FG:BG ratio over time, categorized by bin count. Each plot contains 15 lines, being the result of different configurations (5 different dimensions, 3 different complexities). It shows that with a smaller bin count the FG:BG ratio generally increases. The mapping is fixed across plots.

A.6.3 Discriminability ratio

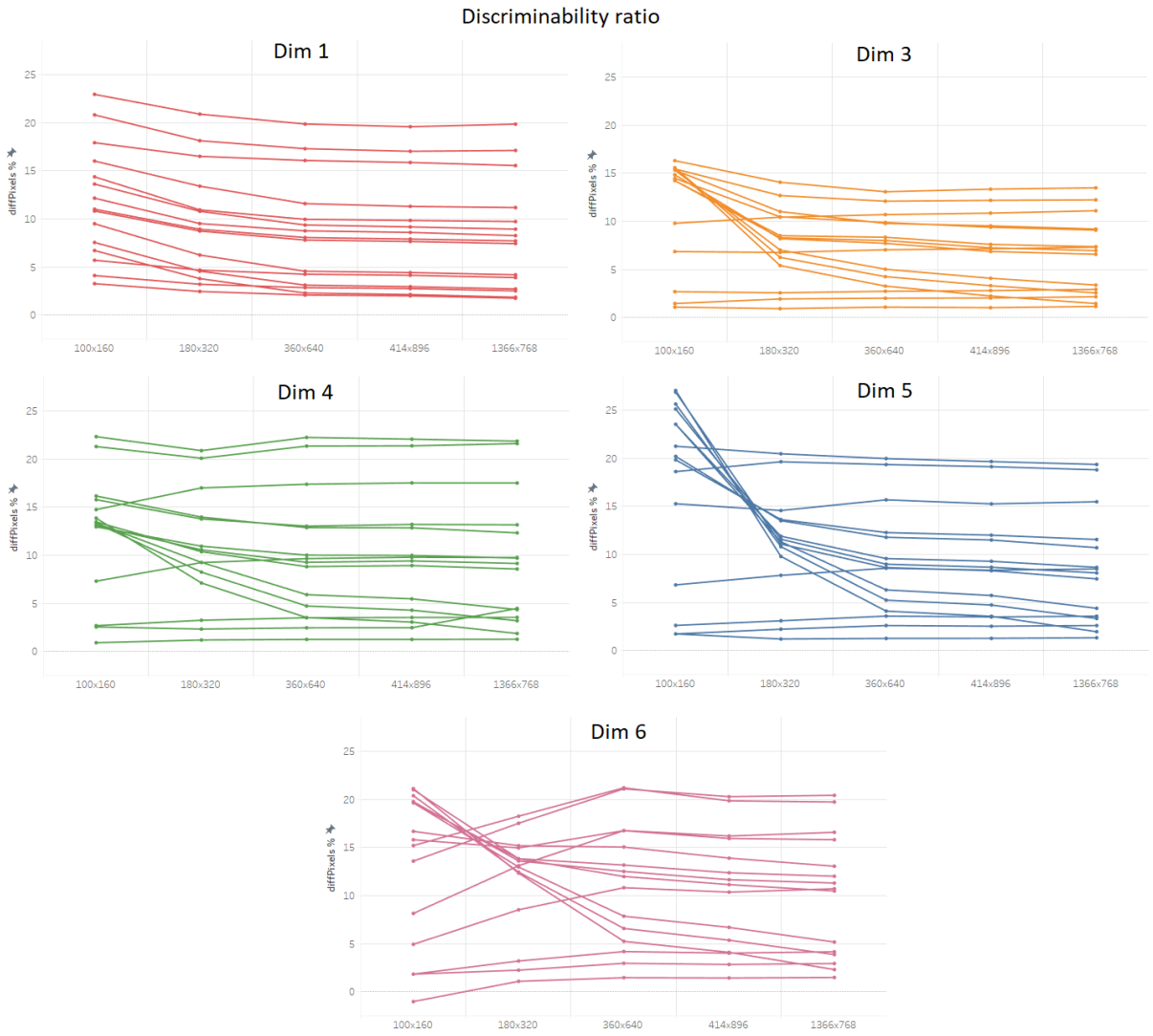


Figure 51: Shows the calculated discriminability ratio over time, categorized by dimensionality. Each plot contains 15 lines, being the result of different configurations (5 different bin sizes, 3 different complexities). The mapping is fixed across plots.

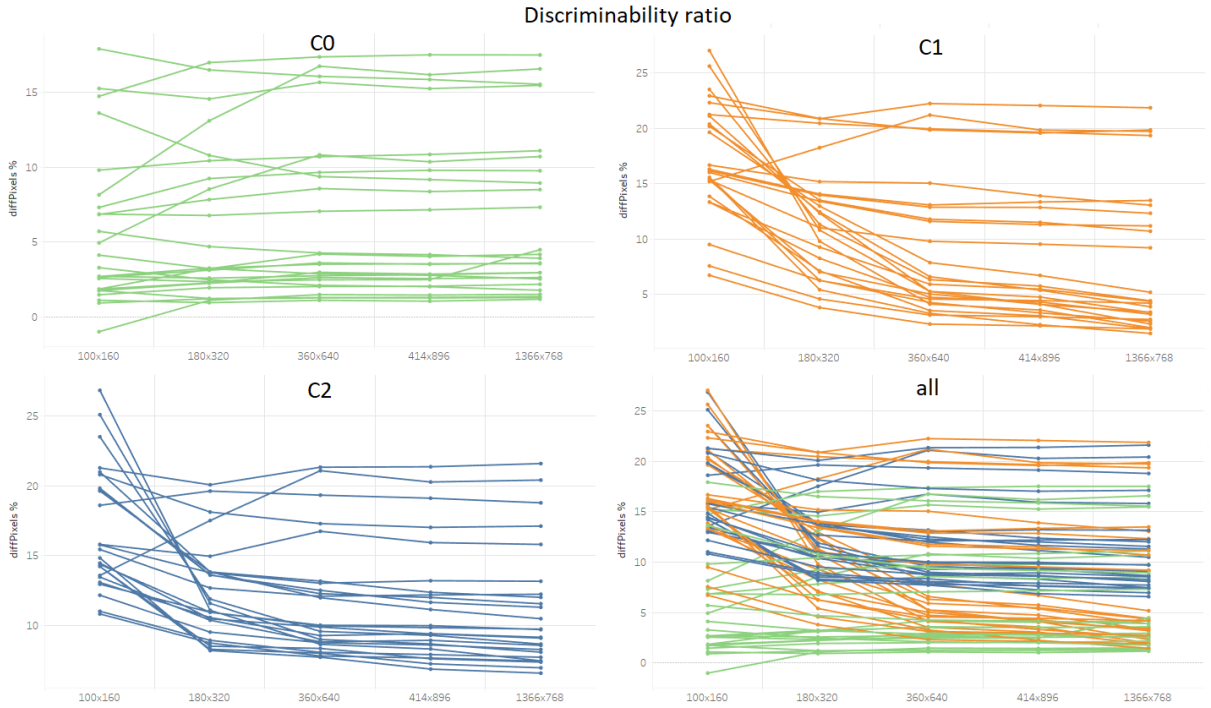


Figure 52: Shows the calculated discriminability ratio over time, categorized by complexity. **Top left:** c0 (green). **Top right:** c1 (orange). **Bottom left:** c2 (blue). **Bottom right:** All configurations, colored by complexity. The mapping is not fixed across plots.

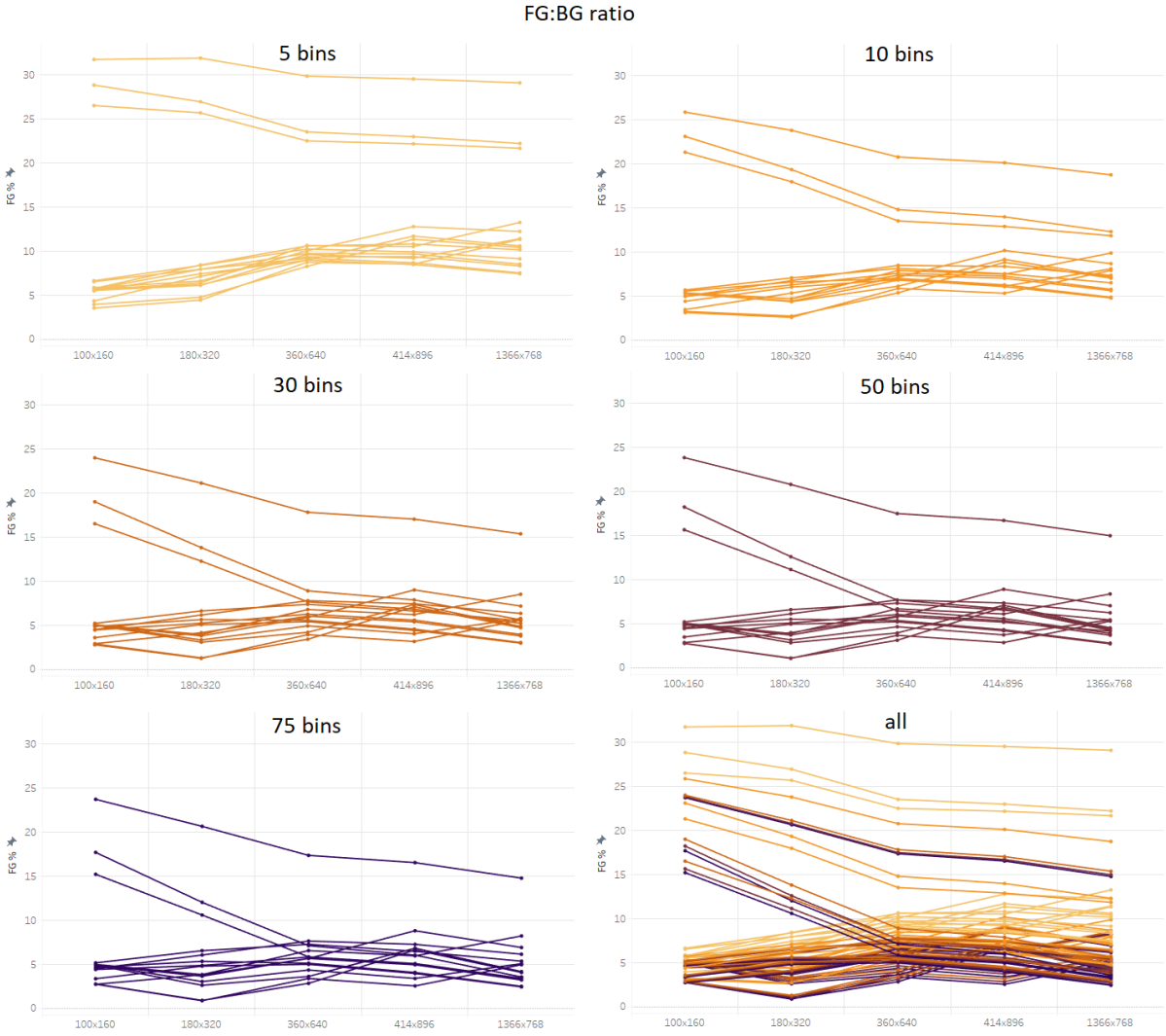


Figure 53: Shows the calculated discriminability ratio over time, categorized by bin count. Each plot contains 15 lines, being the result of different configurations (5 different dimensions, 3 different complexities). The mapping is fixed across plots.