FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Hana Roubalová

Detecting Misleading Features in Data Visualization

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	Mgr. Marta Vomlelová, Ph.D
Study programme:	Computer Science
Study branch:	Computer Science with Specialisation in Artificial Intelligence

Prague 2023

I want to thank my supervisor, Marta Vomlelová, for her guidance, expertise, insights, and support while writing this thesis.

Thanks also go to my family for their understanding and encouragement; their belief in my abilities has been a source of motivation.

Last but not least, I want to thank my incredible partner, whose love, support, and understanding got me through this journey. Thank you.

Title: Detecting Misleading Features in Data Visualization

Author: Hana Roubalová

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Marta Vomlelová, Ph.D, Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis explores the identification and detection of misleading elements in data visualizations. The theoretical portion focuses on understanding various types of misleading features commonly encountered in scientific figures and recognizing them. The implementation introduces an application designed to detect colorblind-unfriendly graphs with the analysis of various algorithms. The thesis raises awareness about misleading visualizations and demonstrates how software can simplify the detection of misleading features for the everyday user. This thesis highlights the importance of addressing misleading features in data visualizations and introduces an application to assist in their detection. The study advances our understanding of this field and offers insights into reducing the negative effects of misleading data visualizations.

Keywords: data visualization, misinformation, colorblindness, color perception, color vision deficiency

Název práce: Rozpoznání zavádějících prvků ve vizualizacích dat

Autor: Hana Roubalová

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Marta Vomlelová, Ph.D, Katedra teoretické informatiky a matematické logiky

Abstrakt: Tato práce se zabývá detekcí zavádějících prvků ve vizualizacích dat. Teoretická část práce seznamuje s různými typy běžně se vyskytujících zavádějích prvků a ukazuje je na konkrétních příkladech. Implementační část práce pak s pomocí analýzy různých algoritmů představuje aplikaci určenou k rozpoznání nevhodných grafů pro jedince s barvoslepostí. Práce ukazuje, jak může software zjednodušit detekci zavádějících prvků pro běžného uživatele. Zároveň zvyšuje povědomí o tomto problému a představuje možnosti prevence proti negativním důsledkům zavádějících vizualizací.

Klíčová slova: vizualizace dat, dezinformace, barvoslepost, vnímání barev, vada barevného vidění

Contents

Introduction

With data analysis uncovering massive amounts of useful information, data visualization techniques allow us to communicate facts and guide decision-making efficiently. Moreover, when visualization is done correctly and with good intentions, it is an excellent resource for forming opinions and drawing conclusions. However, unfortunately, it can very easily be misinformative. This thesis will examine misleading data visualization elements and their detection.

Even though visualization is a powerful way to convey information and has a significant effect on fortifying a message or prompting a change Lo et al. [2022], the main focus, when discussing misinformation and disinformation, is on the textual form. Furthermore, we need to learn how to interpret graphs correctly; therefore, for most people, misleading visualizations can be hard to spot.

The goal of this work is to give an overview of known misleading features that are common in graphs. Then, explain how we can spot them and implement a simple application that will automatically detect the presence of some of these elements in a graph image.

Since misinformation can happen in every stage of creating the visualization Lo et al. [2022], we will focus mainly on misinformation and confusion created by the visualization design of the graph itself. Therefore, we will assume the underlying data was correctly selected and analyzed since we can only verify this by examining the original data.

We will also assume that the observer interprets the data based on the visualization itself. Therefore, we will not consider any additional textual information except in the chart.

The structure of the thesis is as follows. In Chapter 1, we describe frequent misleading features in visualizations. In Chapter 2, we introduce essential concepts and describe the techniques and algorithms used for application implementation. Chapter 3 discusses the implementation process and architecture design of the application, while Chapter 4 describes the experimental tests and their results.

1. Misleading Features in Data Visualization

This section will categorize and classify some known misleading features and explain how and why they occur. Then, show some examples of these features in graph images and try and improve comprehension of these example visualizations.

There are many ways to classify misleading mistakes people often make when creating data visualization and the effects they can have on people. In this work, we will group these elements by the creation stage in which they arise.

Creating data visualization starts with choosing and collecting the correct data and preprocessing and analyzing it. Unfortunately, errors occurring in this stage are hidden in the data, and detecting them from the visual representation without knowing the data's context is impossible. That is why we will cover this section just briefly.

The subsequent step is the design of the visualization, which is the main focus of this work. Following the creation of the visualization, we also have to consider the perception and interpretation of the viewer. However, that depends on each individual and their abilities. Therefore, it cannot be detected from the visual depiction and will be covered concisely.

1.1 Data Analysis Pitfalls

Before creating a visual representation of data, we want to extract some valuable information to illustrate. We call this process data analysis consisting of data collection, cleaning, preprocessing, and analyzing. All this is done to uncover helpful relations and patterns, answer questions, and forecast trends. However, the methods which we can choose for data analysis are numerous. Nevertheless, not all are suited for every problem. Therefore, knowing how to collect and store the data and what methods to use is crucial. Unfortunately, these decisions are difficult to make, and it is easy to fall into pitfalls resulting in faulty conclusions and, inevitably, faulty visualizations. The first problem comes with data collection.

1.1.1 Collecting Data

Collecting data can be demanding and time intensive. Nevertheless, we need a lot of data for data analysis to be meaningful. Analyzing too few data points can be misleading Lo et al. [2022] as it can create patterns that would not show with better data collection. It can also omit important information pivotal to the bigger picture and conceal important tendencies and patterns.

Another pitfall while collecting data is choosing suitable events to examine. It is crucial to collect data congruous with the depicted topic because comparing unrelated events can lead to showing relations that are not there. Correlation does not necessarily mean causation. Moreover, even if there is a relation, it is still difficult to tell which event is the cause of the other Klass [2009], Klass [2008].

Visualizations then may show incorrect relations nonexistent in reality, which is misleading.

In Figure 1.1, we can see such misleading visualization of events. When we look at this graph, we can see a high correlation between the birth rate and the number of stork pairs in European countries Matthews [2000]. Linear regression shows that the more stork pairs breed, the more babies are born. The straightforward conclusion is that, of course, storks deliver babies. That is naturally nonsense, and in this example, it is obvious. We know that storks have nothing to do with how babies are born. However, that is a common mistake people make. They assume that correlation implies causation. However, just because there is a correlation between the number of storks and the birth rate of babies with a p-value of 0.008, it does not mean that storks deliver babies with 99.2% probability Matthews [2000]. The most plausible explanation of the observed correlation is the existence of some common factor.

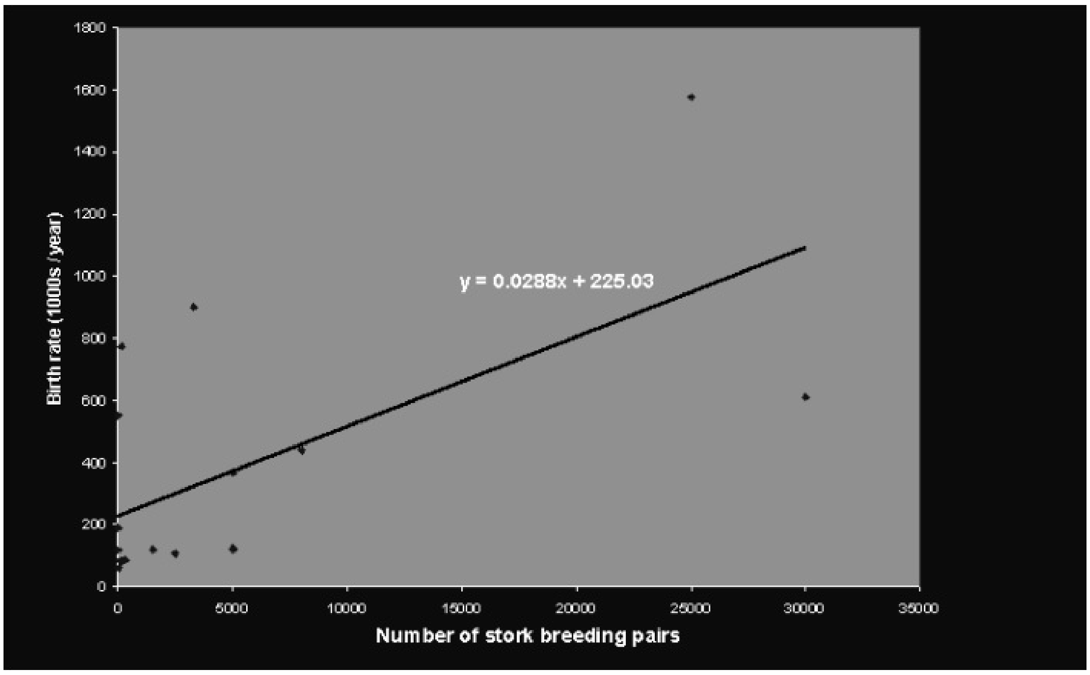

Figure 1.1: Example of correlation not causation, Matthews [2000], (in original quality)

Thus, When looking at a depiction such as this, we need background knowledge of the context of both events. However, this context is not clear from the visualization alone. Therefore, even when the visualization depicts the data correctly, the choice of depicted events might be misleading.

The other well-known problem with data collection is cherry-picking. Cherry-picking is the intentional or subconscious selection of data points that are strategically chosen to display the intended agenda Lo et al. [2022], Klass [2009], Klass [2008]. Unfortunately, this means that the data shown is incomplete and does not realistically show the whole picture. Cherry-picking is often done by zooming in on a specific time or picking particular comparison criteria.

In Figure 1.2, we can see an example of cherry-picking specific comparison criteria. This graph was shown to an audience of Bill O'Reilly's talk show Klass

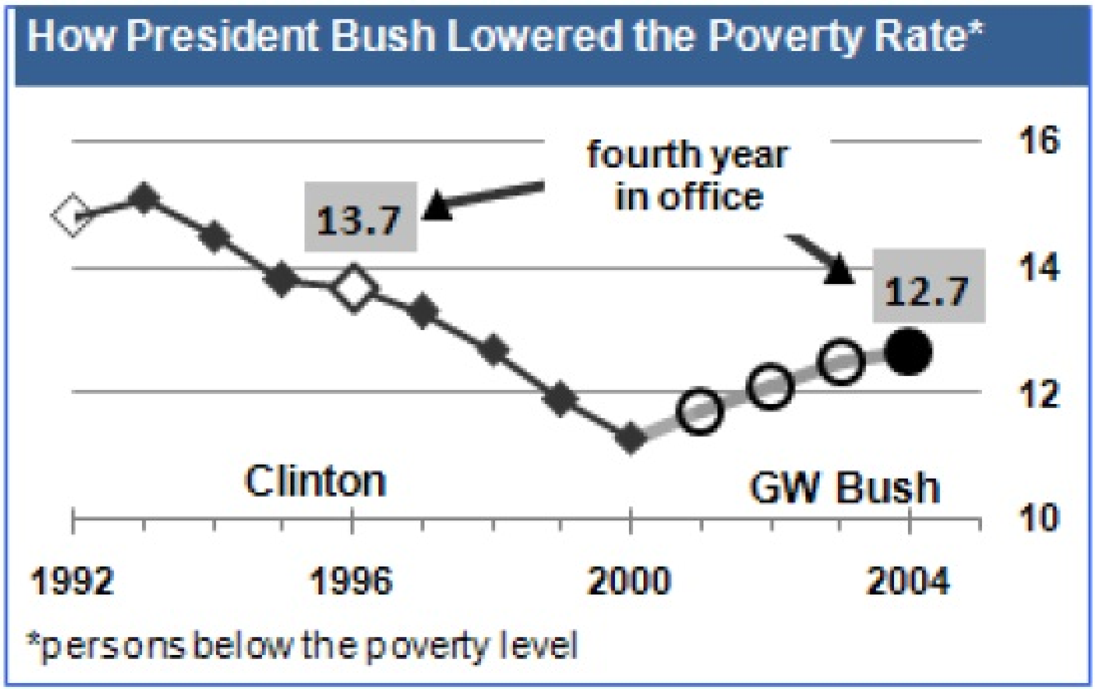

Figure 1.2: Example of cherry picking comparison criteria, Klass [2009], Klass [2008]

[2009], Klass [2008]. He compared the absolute number of people below the poverty level during the fourth year of the presidency of both Clinton and Bush. He pointed out that since the number during Bush's term was lower than during Clinton's, Bush's administration lowered the poverty rate. That is a persuasive argument that could fool many people. However, once we stop zooming in on the absolute numbers and look at the graph as a whole, we can see that during Clinton's presidency, the poverty rate decreased until the year 2000, when Bush started his presidency, when the poverty rate started to grow quite rapidly.

1.1.2 Preprocessing and Analyzing Data

Regardless, mistakes happen when manipulating data as well. Analysis can uncover and highlight trends and patterns in data so they can be better comprehended. Nevertheless, getting information out of data is easy if we are looking for it. We must be mindful of personal biases that could affect analysis results.

For example, some comparisons in data are not valid without normalization Lo et al. [2022]. Showing the absolute values is misleading when comparing events with different magnitudes. In Figure 1.3, we see the difference when plotting the absolute values compared to values per capita, which is more telling about the actual murder rate. Cities with large populations will have higher absolute numbers even though, given the number of people, the murder rate is low.

However, when using normalization, we also need to be careful. If we have multiple values in a single plot, we need to normalize them in the same way Szafir [2018]. Normalizing these values to a different range will distort data and show patterns, shapes, and relations that are not there.

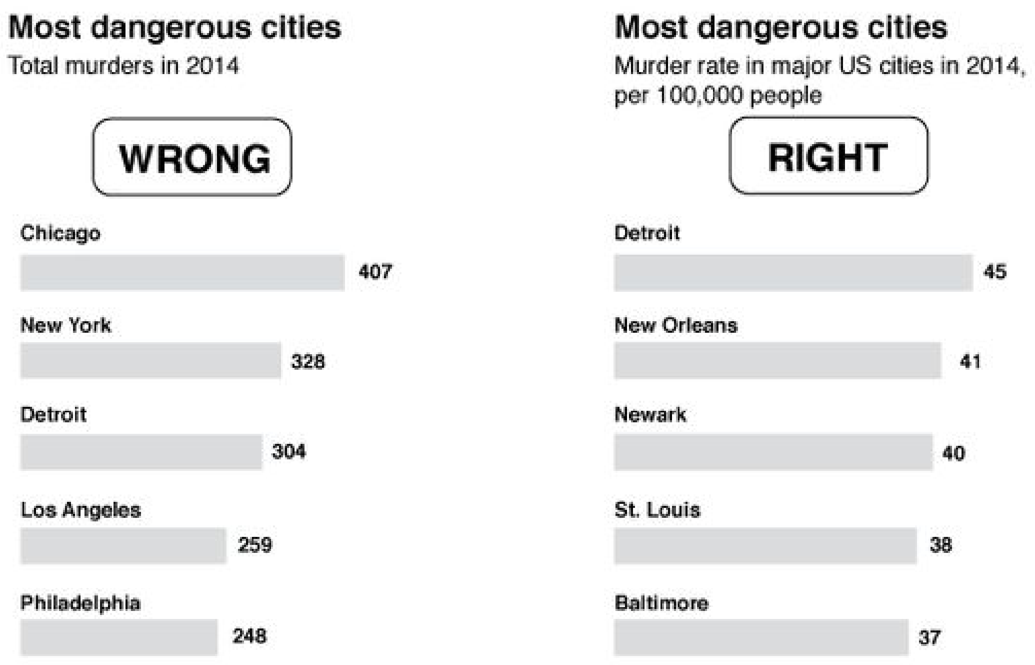

Figure 1.3: Example of missing normalization, Lo et al. [2022]

For example, in Figure 1.4, we see a graph meant to persuade us that the number of abortions grew rapidly while other life-saving procedures dropped in numbers. However, when we examine the plot more closely and look at the presented numbers, we realize that different metrics were used for each value to achieve this look of false crossing around the year 2008. When in reality, at no time on the timeline does the count of abortions exceed the number of life-saving procedures made.

1.2 Visualization Design Pitfalls

Now that we have analyzed the data and know the points we want to make, it is time to design the visualization. To create a visualization, we must map the data to different visual channels, for example, position, size, or color Szafir [2018]. We can create engaging and informative visualizations using a few visual techniques. However, choosing these techniques is where most visualizations become misleading, either by mistake or malice.

1.2.1 Choice of the Plot

Let us start with choosing the type of plot. While most well-known graphs are generally ok to use, there are many creative ways to display data. Unfortunately, although beautiful, unusual, and innovative does not necessarily mean better Lo et al. [2022]. On the contrary, using graphs unfamiliar to the general public can cause confusion. Correctly interpreting graphs is a skill one must learn; therefore, encountering a new type of graph is misleading. For example, where to look for

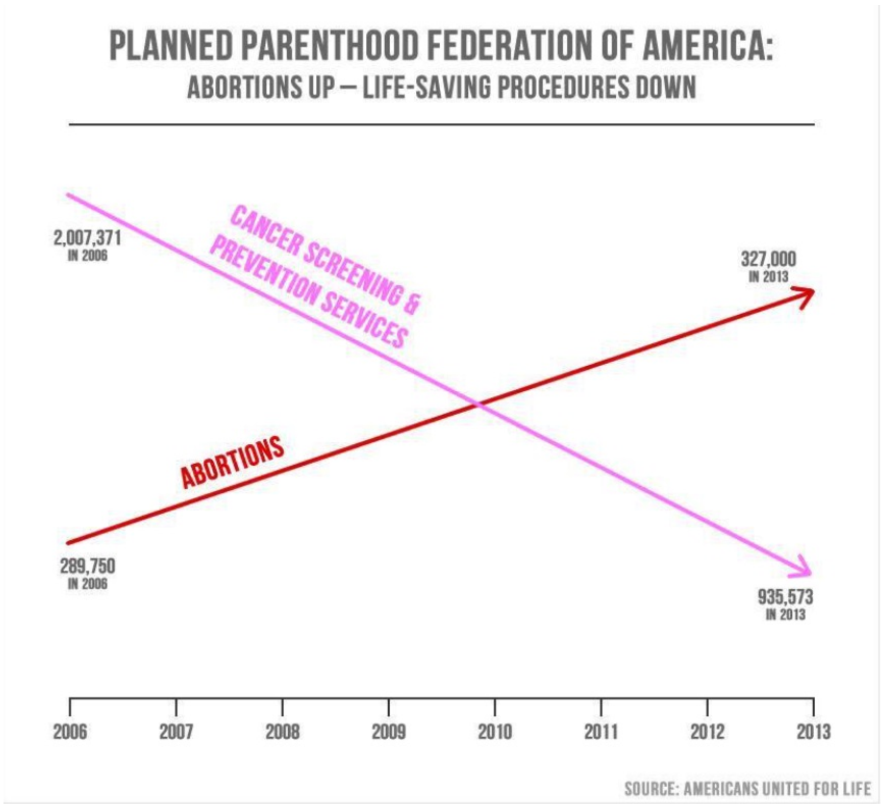

Figure 1.4: Example of two-way normalization, Szafir [2018]

information? What message is the graph trying to convey? What do colors mean in this context, and how are connections displayed? All these are essential questions that the plot must straightforwardly convey. Unfortunately, with new innovative visualizations, this does not happen.

For example, in Figure 1.5, we see an unusual graph. Even though the visualization is heavily labeled, it is unclear what information it is supposed to convey, what the different colors mean, or the connections between data bars.

3D visualizations are widespread and well-known but are also frowned upon by experts. Even though the interpretation of 3D graphs may seem straightforward, that is unfortunately not the case Szafir [2018]. There are several significant problems with 3D plots stemming mainly from the projection of 3D to the 2D plane. When we do this, we lose our brain's ability to resolve positions based on the angles between the perceived object and our eyes. It also distorts our perception of the size since objects further away appear smaller, whereas a small object can also mean a small value in a visualization.

Another problem is occlusion Szafir [2018]. When depicting data in 3D, it is frequent that some objects partially obscure others. This makes it difficult for people to perceive differences between depicted data correctly, and occluded data is effectively lost. It is, therefore, better to exchange the third dimension with some other visual representation, such as color or size.

In Figure 1.6, we can see such improvement. The original graph is hard to interpret due to the perspective and some data bars hiding others. The improved images eliminate these problems by representing the third dimension by size and color.

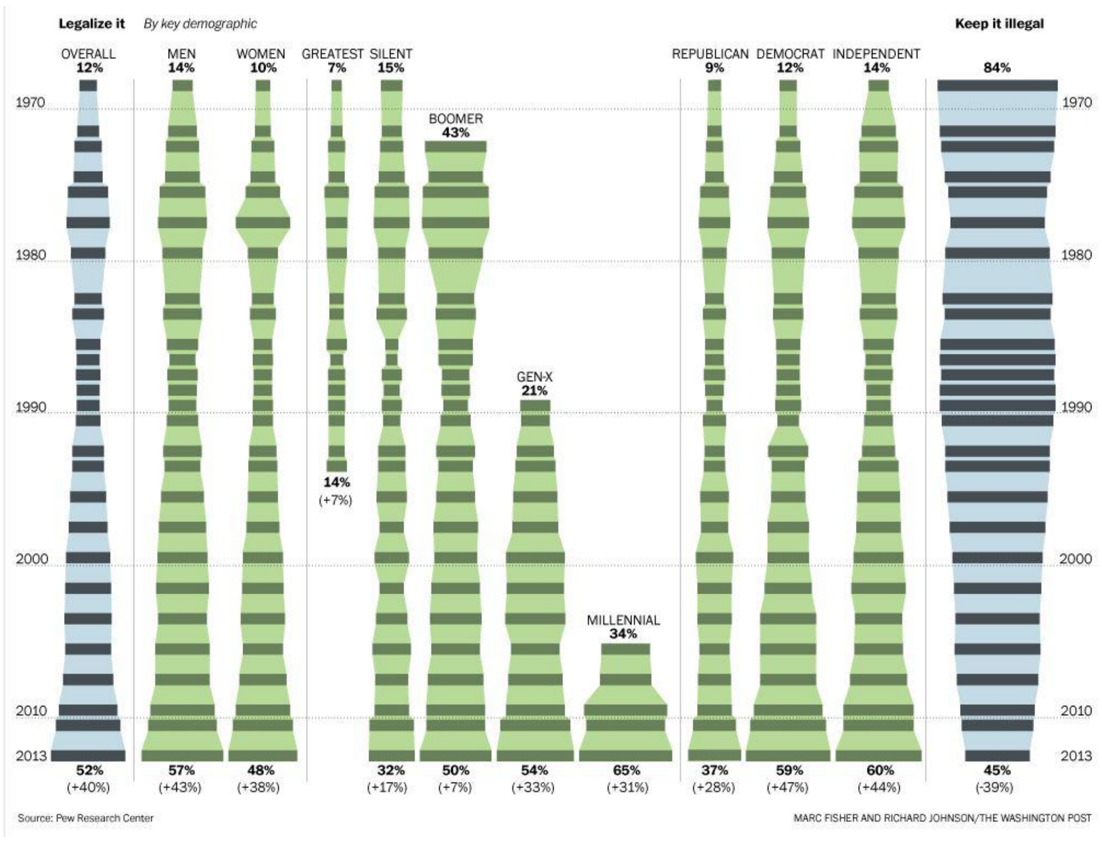

Figure 1.5: Example of confusing choice of plot, Lo et al. [2022]

1.2.2 Choice of an Axis

The following pitfalls lie in choosing an axis, with the most well-known being the truncated axis Lo et al. [2022]. It typically happens when we want to use plot space the most effectively and therefore map the axis to the minimum and maximum value of presented data Szafir [2018]. However, as shown in Figure 1.7, this is misleading as it visually shows differences in data that are not there. Furthermore, since people interpret plots by observing the visual differences between the axis and the data itself Driessen et al. [2022], an axis that does not start at zero will distort these differences. Even though the research shows that labeling the axis counterfeits this phenomenon, most people will not look at the labels and will form conclusions at first glance at the plot. Moreover, people must learn how to interpret graphs correctly to avoid this illusion.

The same goes for other pitfalls concerning axes, such as inverting the axis, which reverses the values, irregular intervals between axis marks, or plotting data of different magnitudes on the same axis Lo et al. [2022]. All these examples distort the visual differences and correlations between presented data and, therefore, can lead to misinformation.

As mentioned before, labeling the axis and values can compensate for these issues Driessen et al. [2022], which is why missing labels, titles, or legends are self-explanatory misleading fallacies Lo et al. [2022].

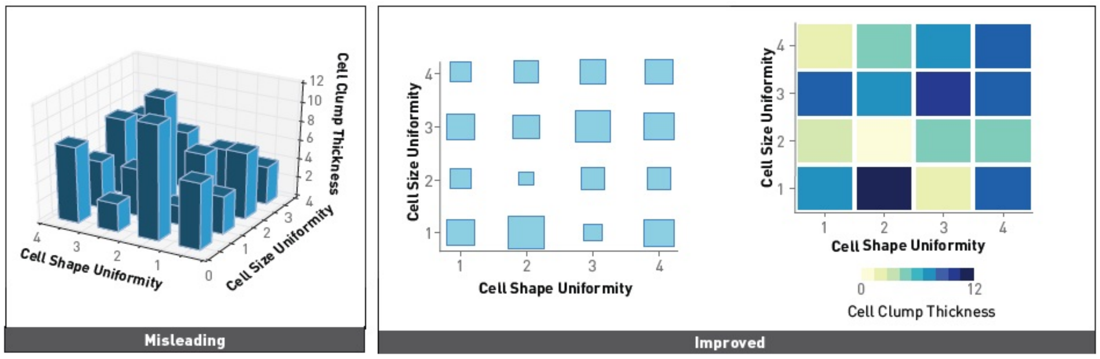

Figure 1.6: Example of improving 3D graph by size or color, Szafir [2018]

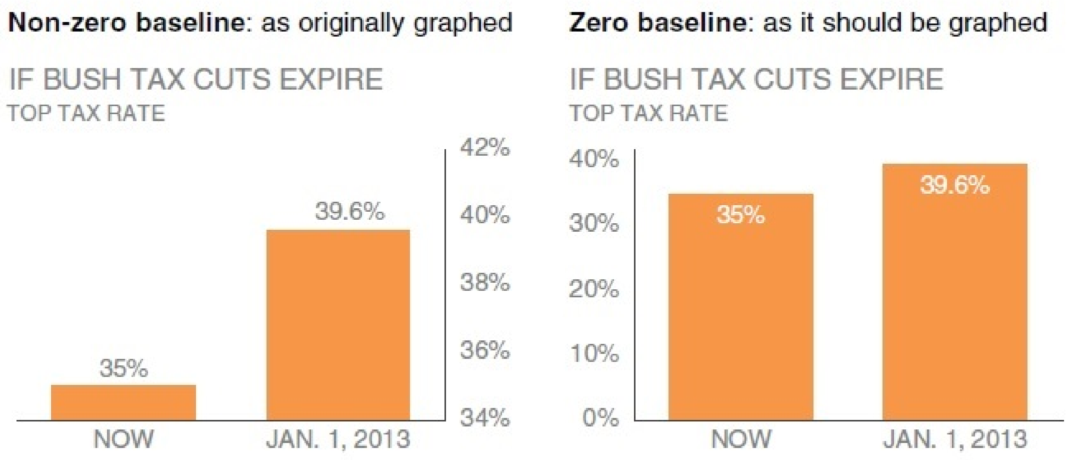

Figure 1.7: Example of truncated axis, Knaflic [2015]

1.2.3 Choice of Color

Lastly, let us mention color as it is a powerful way to add a third dimension without using 3D graphs Szafir [2018], or it can help highlight important patterns in data. Many visualizations depend on color to represent data. Even though color is helpful when visualizing categorical data, it is often misused.

The most common example is using color transitions for continuous data, creating so-called rainbows. When looking at the grayscale representation of a rainbow in Figure 1.8, the problems with this approach become more evident. The color rainbow does not represent a smooth increasing scale Szafir [2018]. Therefore, the human eye will be subconsciously more drawn to specific colors and group colors with similar hues no matter their position on the rainbow.

Another issue is the perceived distance between adjacent colors Szafir [2018]. We can see in the greyscale version sharp transitions several times throughout the rainbow. Our eyes will therefore perceive even a small change in values as significant since there is a big difference in intensity. For all these reasons, the rainbow transitions can lead to perceiving patterns that are not there and omitting others.

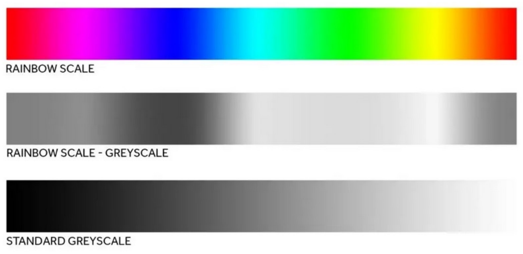

Figure 1.8: Problems with mapping values to rainbows, DePasquale [2019]

Such fallacies can be easily avoided by consistently transitioning from light color to dark, as demonstrated in Figure 1.9.

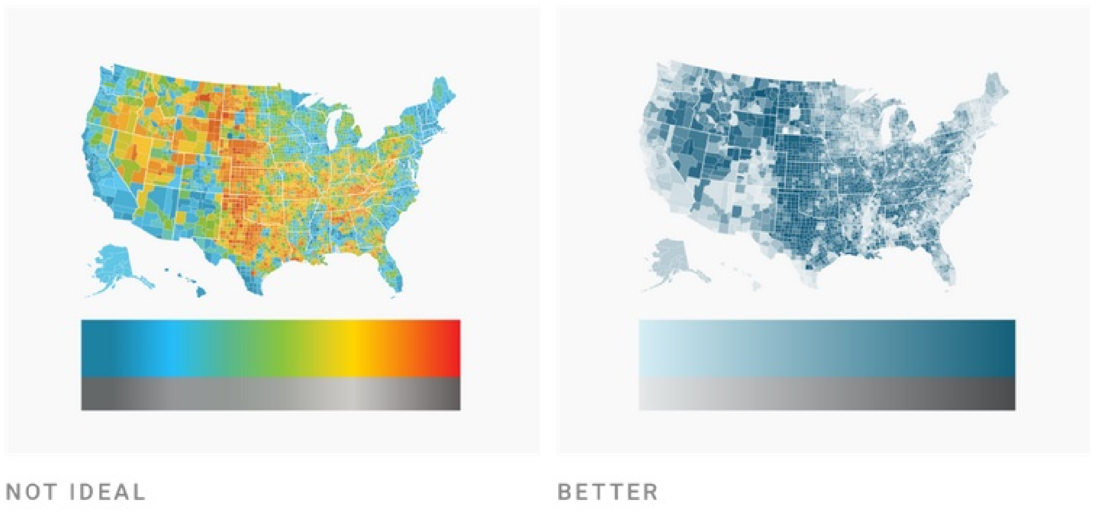

Figure 1.9: Example of avoiding rainbow pitfalls, Muth [2021]

Finally, using too many colors can cause a problem for colorblind people since they see colors differently from the average person Lo et al. [2022], Szafir [2018]. Colorblindness is a frequent phenomenon, especially among men, and therefore designing color graphs needs to reflect this and use colorblind-friendly colors.

In Figure 1.10, we see an example of an unsuitable color scheme and how a person with Deuteranopia, a.k.a inability to see green color and Protanopia, a.k.a the inability to see the color red, would perceive it. Again, using colorblind-friendly color combinations or lightness gradients can address this issue.

1.3 Interpretation and Perception Pitfalls

Analyzing and designing a visualization is just part of the visualization process. It is not always possible to design a visualization that will communicate intended information without misleading it. We often need additional information that cannot be directly in the data visualization itself because it would become illegible. We need someone to tell us the essential context we need to know to understand the problem presented correctly. We might need to explain what is

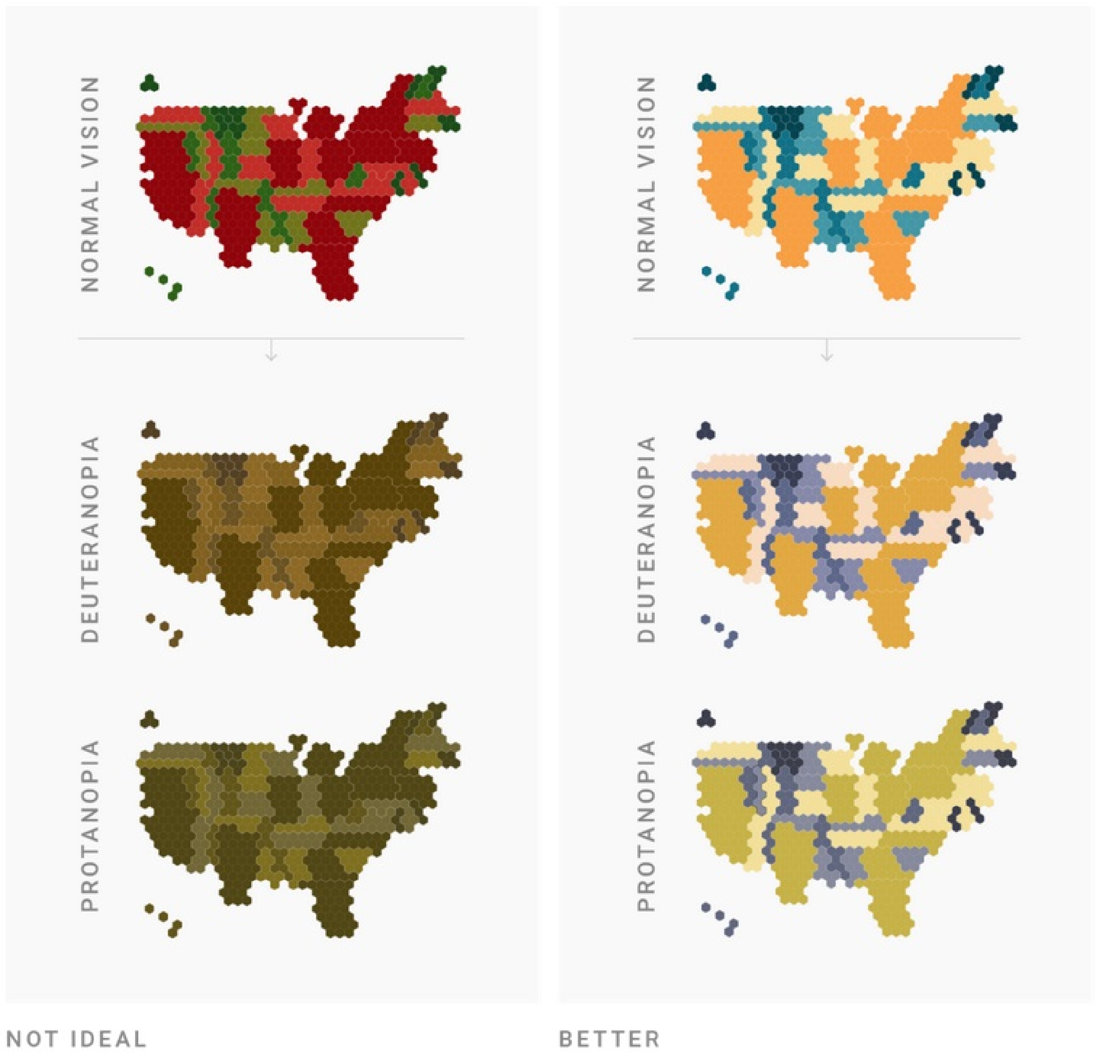

Figure 1.10: Example of colorblindness affecting perception, Muth [2021]

displayed in the graph and why it is crucial. In business, this is usually done by presentation. Someone with deep knowledge of the underlying data and the visualization design will narrate the asked questions, the process of collecting data, the analysis used, and what the visualization shows; what are the conclusions, why are they important, and what action needs to be taken.

Unfortunately, this is not always possible. We encounter many graphs in our life with no one to give us this much-needed presentation. We are left to analyze and draw conclusions from the visualization by ourselves, and this is where interpretation and perception pitfalls occur. It is not possible to completely eliminate misleading features from graph design. Moreover, people can make inaccurate conclusions from an objectively correct graph.

2. Colorblindness Detection

In addition to showing the misleading features in a graph and their recognition, this work aims to create an application that, given a graph, will automatically detect the presence of some particular misleading feature. The misleading feature this work decided to focus on is colorblindness. This section will describe all the necessary concepts, libraries, and software used to implement the application.

2.1 Types of Colorblindness

We define Colorblindness or Color Vision Deficiency as the inability or decreased ability to perceive or distinguish colors Daniel [2020]. The human eye has three types of color photoreceptors located at the retina; individuals have difficulty perceiving colors if there is a problem with any of these receptors.

Protanopia means the total absence of red photoreceptors in the affected individuals. This results in difficulty in distinguishing between red and green colors. The absence of green photoreceptors causes deuteranopia, which manifests similarly to protanopia and again causes difficulty distinguishing red and green colors. That is why both protanopia and deuteranopia are sometimes called red-green blindness. Tritanopia signifies the complete absence of blue receptors and results in the inability to distinguish between blue and yellow, therefore is often called blue-yellow blindness (Fig. 2.1).

Most people only experience partial forms of these deficiencies and can more or less adjust to everyday life.

We should also mention monochromacy, which is the complete loss of color vision, but we will only concern ourselves with protanopia, deuteranopia, and tritanopia.

2.1.1 Colorblindness Diagnosis

It is essential to diagnose color vision deficiency as it can make some activities very difficult or even impossible Daniel [2020]. However, diagnosing colorblindness is not easy, especially when the colorblindness is mild. There are some tests for testing color deficiencies, most commonly known being the Ishihara Plate test Miquilini et al. [2019] for detecting mainly red-green colorblindness. The test consists of pseudo-isochromatic plates. Each plate is designed to have some shape or a line difficult to distinguish for people with red-green colorblindness (Fig. 2.2).

2.2 Color Representation

There are many different ways we can represent color information digitally. Each color representation model has different qualities and is suitable for different purposes Gonzalez and Woods [2018]. This subsection will introduce the RGB and the L*a*b* models we use in our application and introduce perceptual uniformity.

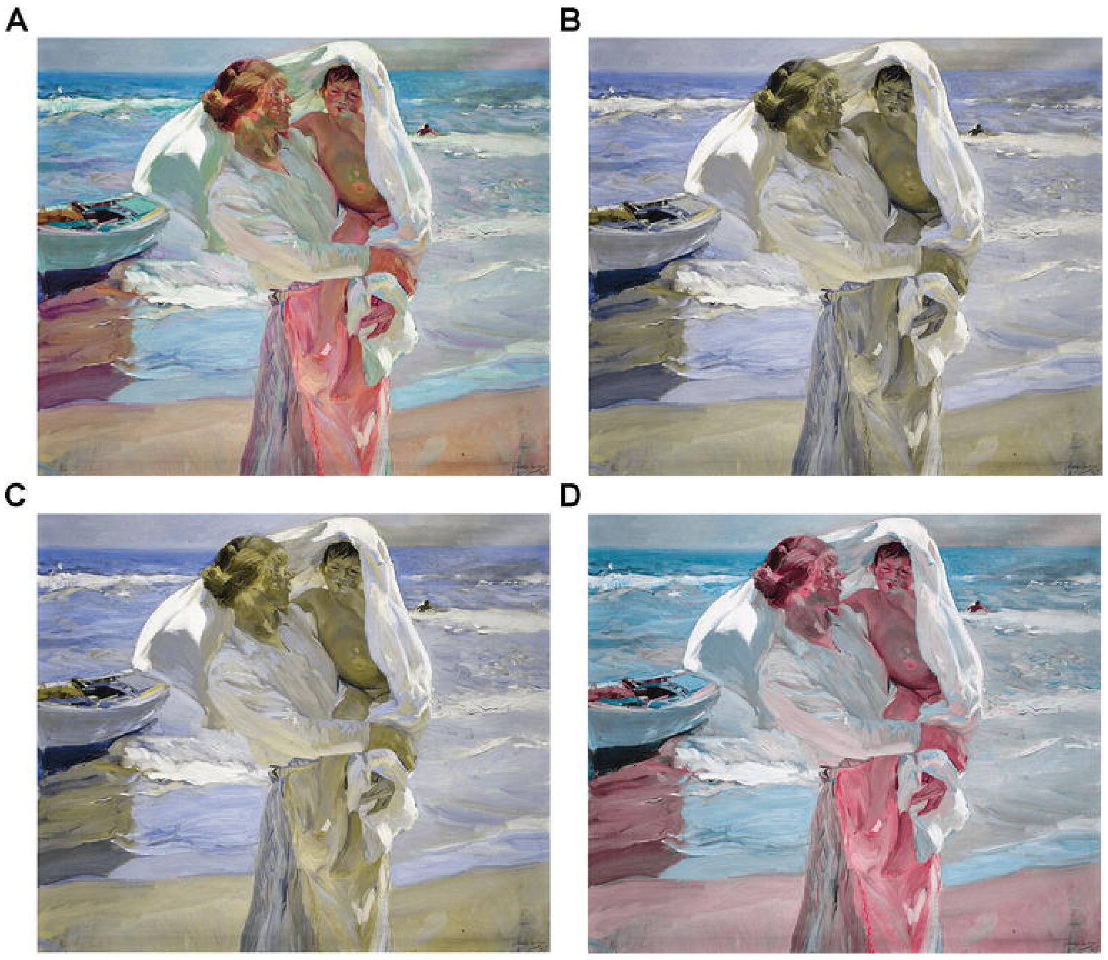

Figure 2.1: Color vision deficiency simulation. (A) Original Joaquin Sorolla's painting Saliendo del baño (”Coming out of the bath”); (B) protanopia simulation; (C) deuteranopia simulation; and (D) tritanopia simulation., Moreira et al. [2017]

2.2.1 RGB Color Space

The RGB color space is designed to relate to human color vision. As mentioned above, three types of photoreceptors are present in the retina Poynton [2003]. These photoreceptors are called cones, and each type responds to different wavelength bandwidths that loosely correspond to the red, green, and blue primary colors. In the RGB color space, each of these three components is represented by an 8-bit value that spans from 0 to 255 and is proportional to the intensity of that particular color channel.

This model is generally sufficient to represent colors. However, the human eye has a far lower ability to detect color detail than lightness Poynton [2003]. The RGB color space is, therefore, not perceptually uniform. That means the perceptual distance between two colors does not match the Euclidean distance between them. That may not be ideal for our application; therefore, we introduce the L*a*b* color space.

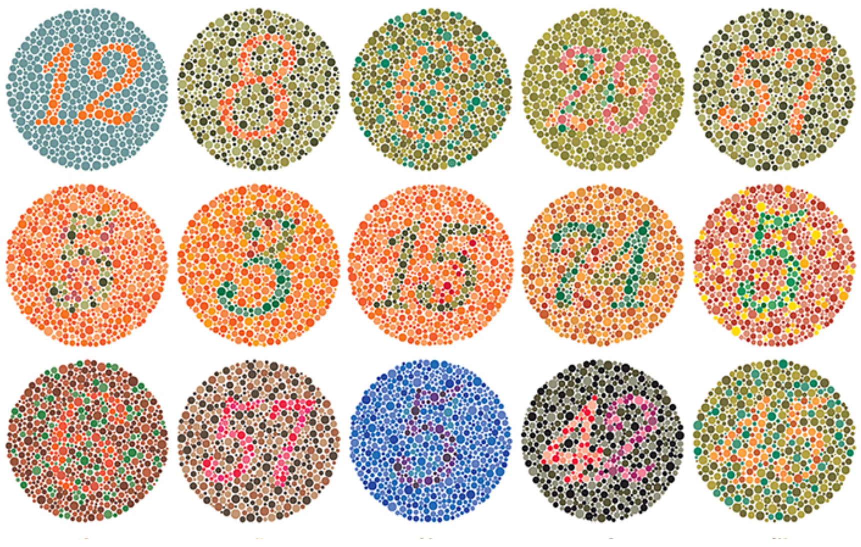

Figure 2.2: Ishihara test plates, Association

2.2.2 L*a*b* Color Space

The L*a*b* color space decouples intensity and color in its representation Gonzalez and Woods [2018]. Color is represented by the a* component encoding red minus green color and by the b* component, which represents green minus blue. L then represents intensity as lightness.

The L*a*b* color space has the advantage of being almost perceptually uniform and colorimetric Gonzalez and Woods [2018], which means that if we perceive colors as matching, they are encoded the same. It is also device independent, which means it maintains a high degree of color consistency between different devices. However, even though it can encompass the entire visible spectrum, it is not directly displayable, so we need to convert it to other color spaces.

2.3 Color Distance

We must introduce some color distance metrics to compare, match, and group colors. In this section, we will introduce two commonly used metrics for color comparison; Euclidean and CIEDE2000 distance.

2.3.1 Euclidean Distance

Euclidean distance is a metric measuring the distance between two points in Euclidean space. In RGB color space, the Euclidean distance calculates the square root of the sum of the squared difference in individual RGB values of the two colors Gonzalez and Woods [2018](Eq. 2.1).

$$\Delta E = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2} \tag{2.1}$$

Unfortunately, Euclidean distance assumes, among other things, uniform variance. As mentioned above, the RGB color space is not perceptually uniform;

therefore, the Euclidean difference does not exactly align with human perception of color similarity Paschos [2001].

The L*a*b* color space considerably improves this issue as it is very close to human perception Paschos [2001].

$$\Delta E^*_{ab} = \sqrt{(L_2 - L_1)^2 + (a_2 - a_1)^2 + (b_2 - b_1)^2} \tag{2.2}$$

2.3.2 The CIE 2000 Color-Difference Formula

Even though L*a*b* color space significantly improves the RGB color space's perceptual nonuniformity, it is still not truly uniform Poynton [2003]. With a genuinely uniform color space, the color difference could be computed with Euclidean distance Fairchild [2005]. Since even the L*a*b* has some nonuniformity, more sophisticated color difference equations were formulated.

The CIEDE2000 color-difference formula is based on the CIELAB color space and addresses the limitations of the Euclidean distance in CIELAB Luo et al. [2001]. Apart from weighting functions for lightness, chroma, and hue, it also considers the interactive term between chroma and hue differences and the scaling factor for CIELAB a* scale. Doing that, CIEDE2000 improves performance for blue as well as grey colors.

2.4 K-Means

K-Means is a widely used clustering algorithm. Clustering is an unsupervised learning method to group similar-looking data Ichikawa and Morishita [2014]. The K-Means algorithm first selects k random points as the initial centers of expected groups. Then it repeatedly assigns each point to its closest center and selects new group centers as the mean of points assigned to that group. This step is repeated until convergence or our patience runs out (Fig. 2.3).

It is crucial to mention that the final grouping is very sensitive to the initial random selection of k centers Ichikawa and Morishita [2014]; thus, more efficient methods have been implemented for the centroid selection.

While introducing the K-Means algorithm, we mentioned assigning points to its closest group center. We must define a distance metric for this, and Euclidean or Manhattan distance are available candidates Ichikawa and Morishita [2014].

Using Euclidean distance while grouping colors with the K-Means algorithm, we may encounter the same problem with perceptual uniformity as mentioned in subsection 2.3.1.

2.4.1 Silhouette Score

The silhouette score is a metric for evaluating the quality of clusters a clustering algorithm creates Mishra et al. [2022]. It measures how well each data point fits within its assigned cluster compared to others. Its values range from -1 to 1, and a higher silhouette score indicates that the data points are well-clustered, have high similarity within their clusters, and are neatly separated from others.

In our application, the silhouette score determines the optimal number of clusters k for the K-Means algorithm. The Silhouette score is calculated for each

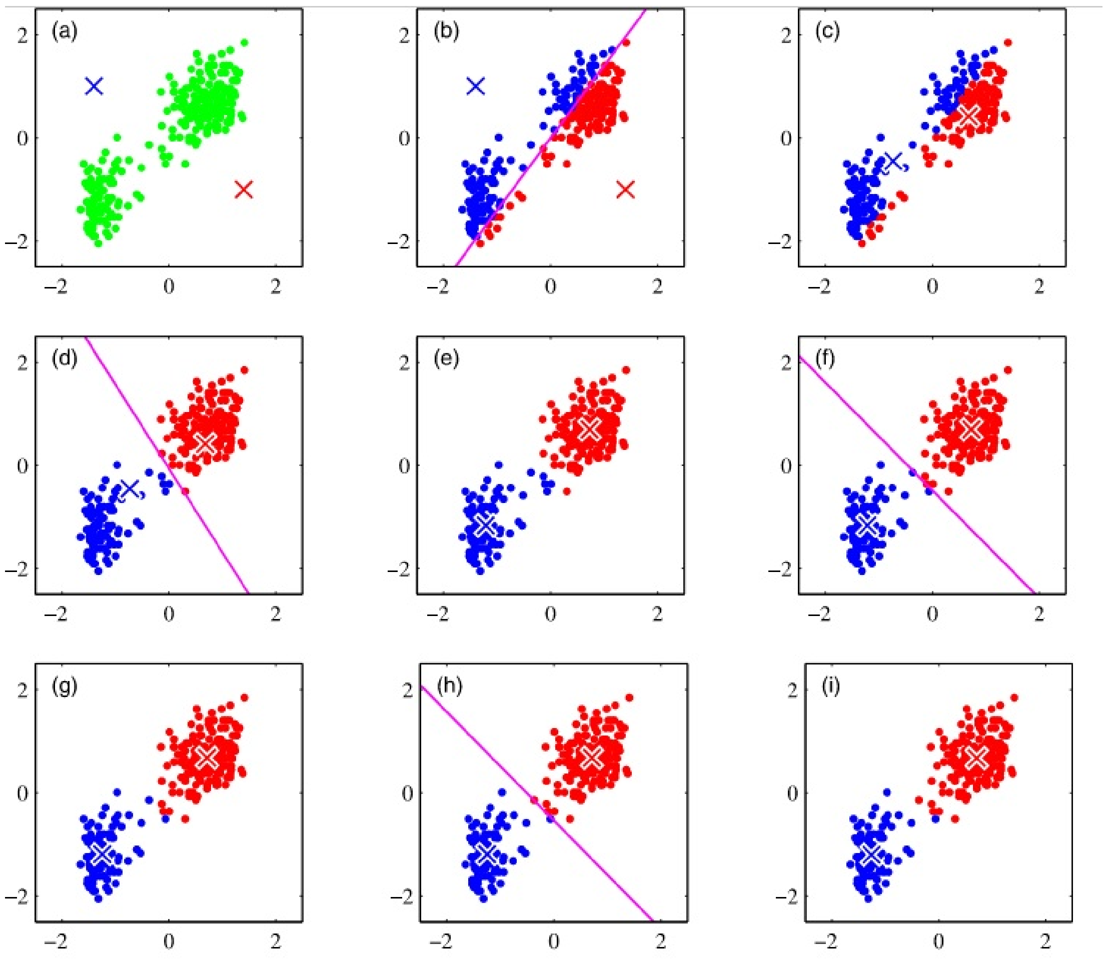

Figure 2.3: K-means algorithm iteration example, Straka [2022]

iteration of algorithms run with different values of k. The highest silhouette score indicates the optimal number of clusters, so the colors are grouped based on their similarity and distinguishability.

2.5 Colorblind Library

The Colorblind is a computer vision library that transforms images into a version that is more accessible to individuals with color blindness, taking into account the specific type of color vision deficiency Rahfeldt [2021]. It supports the three previously mentioned types of color vision deficiencies: protanopia, deuteranopia, and tritanopia.

Apart from implementing various algorithms for converting images into more colorblind-friendly versions, the library can also simulate different types of color-blindness, a feature we utilize in our application.

3. Colorblindness Detection Application Design

Now that we know all the concepts used in our application, let us describe the development process. This project aims to develop an application with a graphical interface that determines if a given graph image is colorblind-friendly or not. The application will analyze the colors of a given image and assess their distinguishability for people with different types of color blindness.

Throughout this section, the implementation process, including the methodology analysis, system design, and performance evaluation, will be discussed.

3.1 Dataset

The evaluation dataset consists of 80 randomly generated graph images. Each image was randomly assigned a chart type, either a bar, line, or scatter chart. Furthermore, the number of colors used in each graph was randomly chosen from a range of 1 to 3. Specific colors were selected from a set of potentially problematic colors for individuals with different types of color blindness. Subsequently, the dataset underwent a manual scanning process to identify and remove any graphs whose color distinction was deemed unreadable, even for individuals with intact color vision. Additionally, each image was classified by hand with information about which color vision deficiency it triggers.

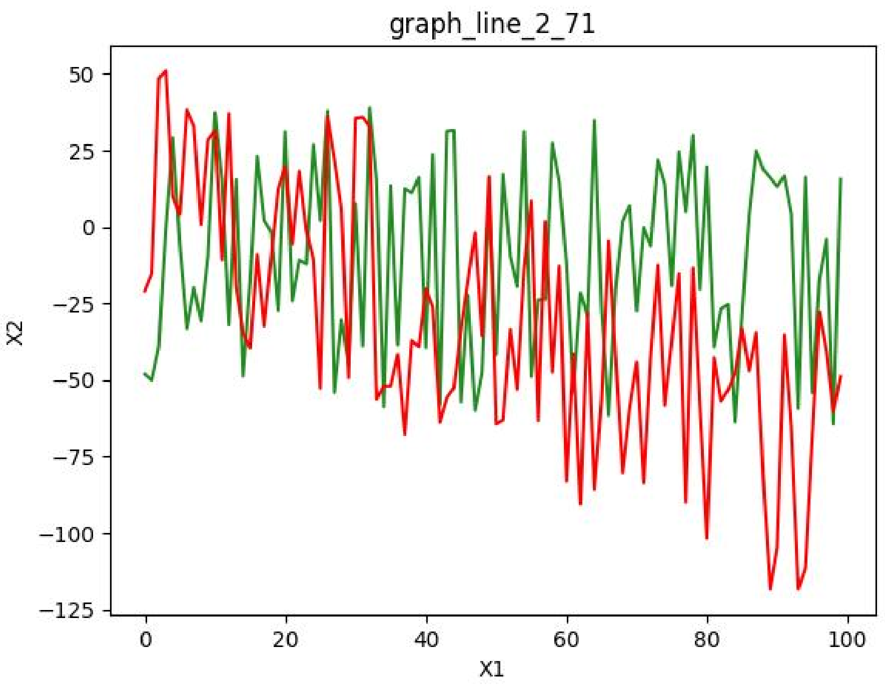

Figure 3.1: Example of an image from the threshold evaluation dataset.

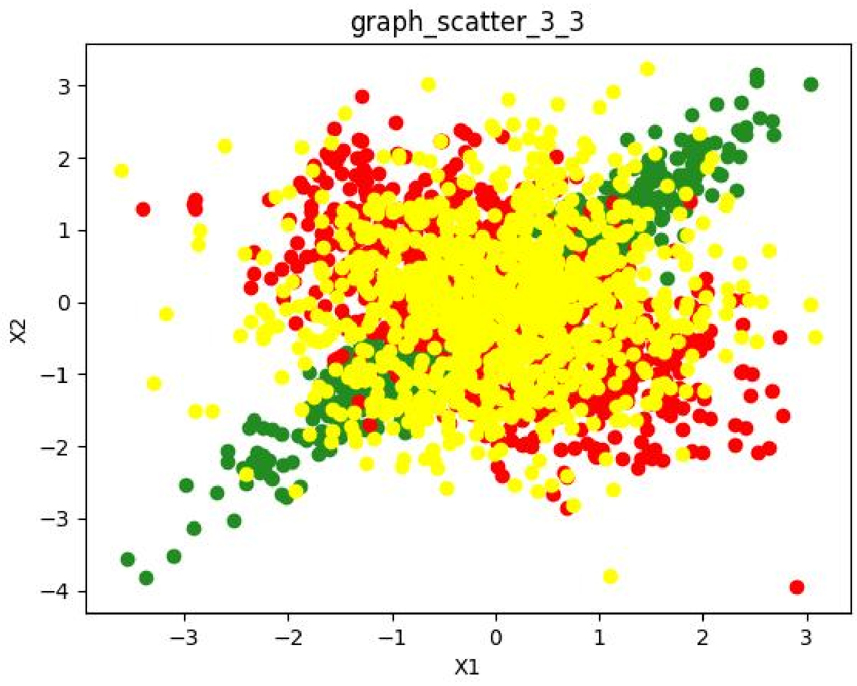

Figure 3.2: Example of an image from the threshold evaluation dataset.

3.2 Methodology

As we can see in figures 3.1 and 3.2, the image quality is not the greatest. That is expected since our application assumes the input image is in either jpg or png format. The jpg format is lossy; therefore, saving an image in this format results in loss of information, and the resulting image is not precisely the same as the original one Hamzah et al. [2021]. This means that even though only four colors were used to create the dataset image 3.1, when we load the image for processing, we receive 12 736 unique color pixels. Unfortunately, a vast amount of colors will decrease the computation process drastically, and as we need an application that will be user-friendly, we need to speed up our computation.

Therefore, we need a way to compare colors and decide which ones are close enough to be considered the same color group; this is where color distance comes in.

Thus, the color grouping process will involve experimenting with different techniques to determine the most effective approach, which will then be chosen as the default strategy for the final user application.

One approach for grouping colors is to use the K-Means algorithm. However, that requires specifying the number of groups beforehand. To determine the optimal value of k, we use the Silouhette score, which is computationally expensive.

An alternative would be grouping the colors based on similarity given by their Euclidean distance. This approach may improve the algorithm's time complexity but compromise the grouping accuracy as RGB color space is not perceptually uniform. Therefore, the Euclidean distance will be additionally tested with the L*a*b* color space.

Another potential improvement may arise from using the deltaE distance metric instead of Euclidean; however, this might add time complexity.

Overall, we strive to balance speed and accuracy to develop an application that maintains high colorblindness issue detection accuracy while remaining user-friendly.

The algorithm for grouping colors based on similarity utilizing the Euclidean or delatE distance is further described in section 3.3.2, and section 4 describes the experiment evaluation process.

3.3 The Core of the Program

As the foundation for our implementation, we used the algorithm described in the article "I wrote some code that automatically checks visualizations for non-colorblind safe colors. Here's how it works" by Gregor Aisch Aisch [2022]. This algorithm extracts a representative color sample and then computes color differences with the deltaE distance metric. Then does the same for colorblind simulated representative colors and finally triggers a color warning if the computed difference ratios are above some threshold. From this, we extract our algorithm structure as shown in Algorithm 1.

Algorithm 1 Application's algorithm structure

```
1: $img \leftarrow$ LoadImage($image\_path$)
2: $colors \leftarrow$ LoadPixelColors($image$)
3: $color\_groups \leftarrow$ GroupColors($colors$)
4: $detected\_issues\_warnings \leftarrow$ DetectColorblidnIssues($color\_groups$)
```

After loading the image from the given path and extracting the image colors 3.3.1, we first group all the image colors and select the representative color from each group 3.3.2. Then we analyze these grouped colors to detect colorblindness issues for different color vision deficiencies returning the triggered warnings 3.3.3. Each of these steps is described more elaborately in the corresponding subsections.

3.3.1 Loading the Image and Colors

We load the image using the *open* method from the PIL.Image library. Next, we load the RGB pixel values using the *convert* and *getdata* methods from the same library. We can use these color values directly for grouping using the K-means algorithm. However, the grouping algorithm based on similarity using the Euclidean or deltaE distance needs further preprocessing.

3.3.2 Grouping Colors

For the grouping of colors based on similarity, we first filter unique colors and sort them in descending order based on the number of occurrences.

Then the grouping algorithm starts with the color with the highest occurrence frequency and calculates the distances between this color and all other ungrouped colors using either the Euclidean or deltaE distance metric. Then, it compares the distances with some threshold value, and if the distance between the currently first color and some ungrouped color is below this threshold, the ungrouped color is grouped with the first color. The optimal threshold value that needs to be set will be determined using experiment evaluation described in section 4.

Then the algorithm takes the next ungrouped color with the highest frequency and repeats the grouping process. This step is repeated until there are no ungrouped colors left. The output is then single representative color for each final color group. The representative color is computed as the mean of all the colors within that group. The pseudocode for this grouping algorithm can be seen in Algorithm 2.

Algorithm 2 Color Grouping

```
1: function GROUP_COLORS(colors, threshold)
2:     if ungrouped colors left then
3:         $distances \leftarrow$ [compute_distance(colors[0] - color) for color in colors]
4:         $group \leftarrow$ colors[$distances \leq$ threshold]
5:         $rest \leftarrow$ colors[$distances >$ threshold]
6:         return [$group$] + group_colors($rest$, threshold)
7:     end if
8:     return []
9: end function
```

One notable aspect of the color grouping method is the importance of sorting the colors by occurrence frequency. Since the grouping process is order sensitive, sorting the colors allows the algorithm to prioritize the most frequently occurring colors, which are most likely the actual colors in the image. Therefore, we group any perceptually similar colors to the actual color in the image, making it a valid choice for a group.

3.3.3 Detecting Colorblindness Issues

Upon receiving the representative colors, the detection of colorblindness issues can begin. We start by calculating the distance matrix using the deltaE distance. Since the number of representative colors is small, we do not need to concern ourselves with the computational intensity. Then we utilize the Colorblind Rahfeldt [2021] library to simulate the representative colors to the different types of colorblindness and calculate the same distance matrix for these color values. Finally, we compute the difference ratio, and if the simulated distance between some colors has decreased as opposed to the original by more than half, then we consider the image unfriendly for the corresponding colorblindness.

3.4 System Design and Architecture

The application is designed with a separation of concerns approach, separating the UI design from the business logic, which should ensure the modularity and maintainability of the application.

For developing the user interface, the application utilizes the Kivy library, which provides a framework for building cross-platform graphical user interfaces.

The UI communicates with the underlying business logic through a Facade design pattern, which acts as an interface between the two components.

The UI is simplistic and user-friendly, with five buttons to control the application 3.3, 3.4. Application input assumptions and usage is described in user documentation, located in Attachment A.1.

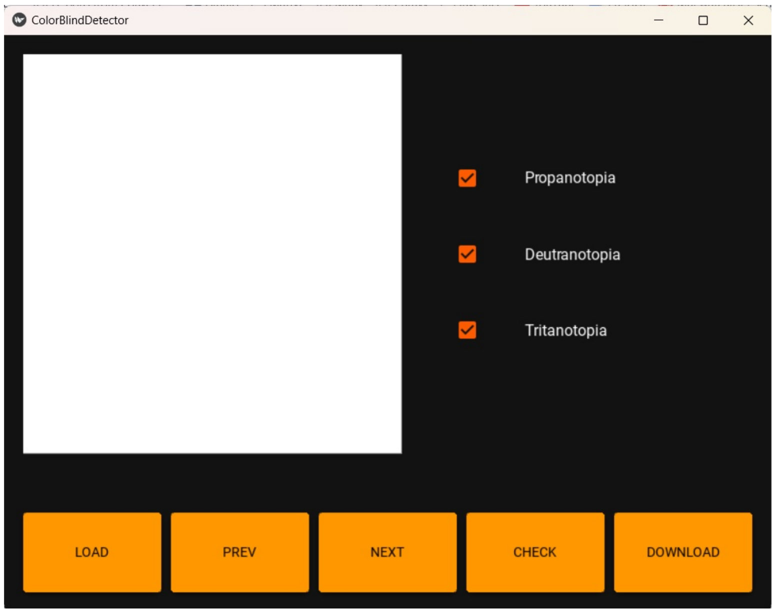

Figure 3.3: User interface after application launch

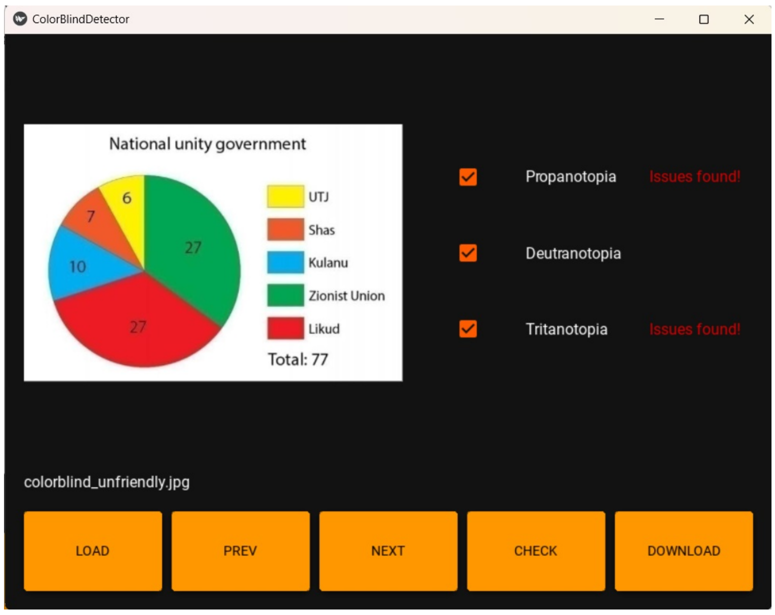

Figure 3.4: User interface image analysis demonstration

The business logic is further divided into three parts; the grouping of colors algorithms, the detection of colorblindness issues logic, and the experiments evaluator. The color grouping algorithms are implemented as a Strategy design pattern with the different grouping approaches acting as concrete strategies and the experiment evaluator as both the context and client. Moreover, the Facade also acts as a context; however, this time without the client logic since the user is not allowed to change grouping algorithms, but rather the best performing one will be set in the Facade constructor as the default one. However, during the implementation phase, the Facade acting as a context came in handy.

Additionally, the experiments evaluator is implemented as a template design pattern, where the evaluation method defines the structure of the evaluation algorithm. At the same time, two concrete classes redefine some of the steps based on whether they perform threshold experiments or the k-means performance experiment.

The overall design and architecture, including the detailed description of all components, is described in the code documentation, which is situated in Attachment A.2.

4. Experimental Evaluation

This section describes the methodology and evaluation results of the performed experiments. It involves finding the optimal threshold values for the Euclidean and delatE color grouping algorithm and comparing and evaluating the overall performance in terms of accuracy and running time of all proposed approaches. Finally, this section states the reasons for the algorithm choice utilized in the final application.

4.1 Experiment Methodology

Since the Euclidean distance paired with the RGB color space, Euclidean distance paired with the L*a*b* color space, and the delatE distance all need a threshold value for separating colors, the first experiment aims to find an optimal threshold value for these three algorithms. Since the threshold affects the grouping of colors, it would make sense to evaluate the threshold experiments on grouping accuracy; the accuracy calculated from the number of actually used colors in the image and the number of detected color groups.

However, that might not be ideal because we are not strictly interested in the number of groups alone but rather in what groups we find and, therefore, what representative colors we get. The best grouping accuracy then might not correspond to the overall accuracy, which is described in Section 4.1.1.

We might prefer lower threshold values over higher ones, and that is because of the ratio method we use to trigger warnings, which was described in Section 3.3.3. We detect colorblindness issues only if the simulated distance between some colors has decreased by more than half compared to the original distance of these two colors.

With a low threshold, we might divide a single color into two similar color groups. However, this will not result in unwanted triggers since the original distance between these similar groups will be low, just as the simulated distance will.

However, if we choose a higher threshold and find fewer color groups than we were supposed to, it means we connected distinct colors into a single one. Consequently, we will not calculate the distances between these two colors, and if these two colors are unfriendly, we will not trigger a warning as we were supposed to.

Therefore, we will evaluate the threshold experiments on the overall performance accuracy and compare each best-performing algorithm with the K-Means performance results.

The K-Means performance evaluation will be run independently since the K-Means algorithm does not need the threshold or sorted colors by occurrence.

The tested threshold values and step size range was picked based on the analysis with a wide threshold range on a single image from the dataset.

4.1.1 Performance Accuracy

Accuracy is a simple measure of correctness. It tells us the ratio of correct predictions to the total number of predictions. However, since each image can trigger none or up to all three color deficiencies, we need to adjust the accuracy score accordingly.

In our evaluation, we consider each type of colorblindness issue as an individual prediction. A value of 1 represents that the issue was triggered, while a value of 0 indicates that it was not. Thus, if the correct classification for an image is no triggers, it is counted as three 0 predictions. This approach ensures that even if our algorithm correctly predicts one deficiency but misses or incorrectly predicts others, we still count the correctly predicted trigger as a correct prediction.

4.2 Euclidean Distance Threshold Results

We list the threshold experiment results for Euclidean distance with RGB color space in Table 4.1. Performance accuracy is rounded to three decimal places, and the row with the best accuracy score for the threshold value **180** is highlighted.

In the table, we can also observe a tendency for the running time to decrease with increasing threshold value. This results from the grouping algorithm because, with a larger threshold value, more colors are immediately included in groups, which results in less iteration and, therefore, faster grouping.

Threshold	Performance Accuracy	Running Time
150	0.775	1 min 16 s
160	0.796	1 min 13 s
170	0.829	1 min 11 s
180	**0.842**	**1 min 9 s**
190	0.825	1 min 8 s
200	0.800	1 min 9 s
210	0.788	1 min 8 s
220	0.779	1 min 7 s
230	0.800	1 min 6 s
240	0.779	1 min 5 s
250	0.758	1 min 4 s

Table 4.1: Euclidean Distance Results

4.3 Euclidean Distance with L*a*b* Threshold Results

The results for Euclidean distance with the L*a*b* color space can be seen in Table 4.2. Again, performance accuracy is rounded to three decimal places, and the row with the best accuracy score for the threshold **95** is highlighted.

Threshold	Performance Accuracy	Running Time
55	0.463	2 min 38 s
60	0.488	2 min 42 s
65	0.517	2 min 44 s
70	0.533	2 min 44 s
75	0.529	2 min 46 s
80	0.554	2 min 43 s
85	0.583	2 min 34 s
90	0.625	2 min 40 s
95	**0.708**	**2 min 39 s**
100	0.663	2 min 35 s

Table 4.2: Euclidean Distance with L*a*b* Results

4.4 DeltaE Distance Threshold Results

The original threshold range for the deltaE threshold experiments was from 30 to 40. However, based on the initial results, it seemed that lowering the threshold may improve the accuracy further. Therefore, we run additional experiments with threshold values from 20 to 30. And it paid off, as we achieved the best result with threshold value **27**, which is highlighted in Table 4.3.

Threshold	Performance Accuracy	Running Time
20	0.604	35 min 59 s
21	0.650	31 min 7 s
22	0.688	34 min 42 s
23	0.704	34 min 48 s
24	0.696	33 min 17 s
25	0.746	42 min 22 s
26	0.758	1h 8 min 29 s
27	**0.779**	**48 min 55 s**
28	0.758	23 min 7 s
29	0.767	21 min 35 s
30	0.754	20 min 1 s
31	0.696	19 min 2 s
32	0.733	17 min 16 s
33	0.713	16 min 46 s
34	0.708	17 min 24 s
35	0.713	16 min 23 s
36	0.725	14 min 53 s
37	0.721	14 min 49 s
38	0.708	14 min 38 s
39	0.696	14 min 12 s
40	0.708	14 min 3 s

Table 4.3: DeltaE Distance Results

4.5 Performance Evaluation

After determining the threshold values, we proceeded with the performance evaluation of the K-Means algorithm and compared its results to the other algorithms. Since the K-Means algorithm is nondeterministic and its results depend on the random initialization of the cluster centers, we ran this algorithm ten times and compared the overall average performance accuracy and the total average running time to each algorithm's optimal threshold results.

4.6 Performance Evaluation Results

The performance results are sorted by performance accuracy in Table 4.4. First, let us look at the running times, which turned out as expected, with the Euclidean algorithm performing the best and the delatE algorithm being the worst. That is, Euclidean paired with L*a*b* is over 20 times faster than the deltaE algorithm, while Euclidean paired with RGB is even over 40 times faster. While K-Means is slower than both Euclidean algorithms, it still performed very well compared to the deltaE algorithm in terms of running time.

Now let us focus on the accuracy. Surprisingly, both the Euclidean algorithm with the RGB color space and the K-Means algorithm outperformed the deltaE algorithm. That is unexpected since the RGB color space is not perceptually uniform. However, since the dataset images are of graphs and a limited set of colors was used, it may be that the colors are so far apart that perceptual uniformity does not matter. Therefore, additional research and experimentation on a more diverse dataset is recommended to either validate or refute this result.

Algorithm	Performance Accuracy	Running Time
Euclidean, RGB	0.842	1 min 9 s
K-means	0.802	5 min 38 s
DeltaE	0.779	48 min 55 s
Euclidean, Lab	0.708	2 min 39 s

Table 4.4: Performance Evaluation Results

This experimentation aimed to find the best method for our final application while considering both the accuracy and the speed. Unfortunately, the deltaE algorithm is out of running. Even though it performs solidly in terms of accuracy, its running time is 48 min 55 s, which is over half a minute on average, and we believe that to be too long for a user's convenience.

The same is true of the K-Means algorithm, whose accuracy is sufficient; however, 4 s on average still seems too long for a user-friendly application, considering we can do better.

Therefore, the final application will employ the Euclidean distance working with the RGB color space as it outperformed all the other algorithms both in terms of accuracy as well as running time.

Conclusion

This thesis focused on detecting misleading features in data visualizations. It provided an overview of known misleading features in data visualization with an emphasis on recognizing these elements and improving validity and inclusion in data visualizations. Furthermore, with the analysis of various algorithms, this work aimed to develop an application that detects the presence of colorblind-unfriendly elements in graphs.

The results of performed experiments show the efficiency and shortcomings of different algorithms. The deltaE distance and the K-Means algorithms demonstrated high accuracy in detecting colorblindness triggers but at the cost of longer processing time. On the other hand, The Euclidean distance paired with the L*a*b* color space, while performing sufficiently in terms of running time, did not do well in terms of accuracy. In contrast, the Euclidean distance with the RGB color space outperformed all other algorithms in both cases.

This research's topic is important, as it contributes to the awareness and understanding of misleading features that can affect data interpretation and, consequently, the public's trust in scientific research. By developing an application that can detect colorblindness-unfriendly graphs, this thesis shows how software can simplify the detection of misleading features for the everyday user and increase the accessibility of data visualizations.

However, it is essential to acknowledge the limitations and challenges encountered during this research. The use of self-generated data and the design of experiments may have introduced biases and limitations in the findings. Further analysis with additional diverse datasets is therefore recommended to validate the results and uncover potential discrepancies.

Future research directions could investigate the reasons behind the unexpectedly good performance of the Euclidean distance algorithm paired with the RGB color space. Additionally, further optimizations could be examined to enhance the overall performance of the detection algorithms.

In summary, this thesis contributes by raising awareness of the issues surrounding misleading features in data visualizations and demonstrating the potential of software implementations to address these challenges. This research hopes to inspire further exploration and advancements in data visualization to create more accurate and inclusive visual representations of data.

Bibliography

Gregor Aisch. I wrote some code that automatically checks visualizations for non-colorblind safe colors. here's how it works, Nov 2022. URL `https://www.vis4.net/blog/2018/02/automate-colorblind-checking/`.

American Optometric Association. Color vision deficiency. URL `https://www.aoa.org/healthy-eyes/eye-and-vision-conditions/color-vision-deficiency`. visited 2023-06-27.

Ana Daniel. Color blindness (color vision deficiency– daltonism). *Medical Surgical Ophthalmology Research*, 3, 11 2020. doi: 10.31031/MSOR.2020.03.000551.

Nick DePasquale. 3 common misuses of color in data visualization, Jan 2019. URL `https://medium.com/@enneyeseakay/3-common-misuses-of-color-in-data-visualization-98aa18d7c940`. visited 2023-01-03.

Jannetje Driessen, Daniël Vos, Ionica Smeets, and Casper Albers. Misleading graphs in context: Less misleading than expected. *PloS one*, 17:e0265823, 06 2022. doi: 10.1371/journal.pone.0265823.

M.D. Fairchild. *Color Appearance Models*. The Wiley-IS&T Series in Imaging Science and Technology. Wiley, 2005. ISBN 9780470012697. URL `https://books.google.sk/books?id=8_TxzK2B-5MC`.

R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Pearson, 2018. ISBN 9780133356724. URL `https://books.google.sk/books?id=0F05vgAACAAJ`.

Rostam Hamzah, Muttaqin Roslan, ahmad fauzan Kadmin, Shamsul Fakhar Abd Gani, and Khairul Azha A Aziz. Jpg, png and bmp image compression using discrete cosine transform. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 19:1010, 06 2021. doi: 10.12928/telkomnika.v19i3.14758.

Kazuki Ichikawa and Shinichi Morishita. A simple but powerful heuristic method for accelerating k -means clustering of large-scale data in life science. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11 (4):681–692, 2014. doi: 10.1109/TCBB.2014.2306200.

Gary Klass. Just plain data analysis: Common statistical fallacies in analyses of social indicator data. 09 2009. URL `http://www.statlit.org/pdf/2008KlassASA.pdf`.

G.M. Klass. *Just Plain Data Analysis: Finding, Presenting, and Interpreting Social Science Data*. G - Reference, Information and Interdisciplinary Subjects Series. Rowman & Littlefield Publishers, 2008. ISBN 9780742560536. URL `https://books.google.cz/books?id=e1BE2UbBhKgC`.

C.N. Knaflic. *Storytelling with Data: A Data Visualization Guide for Business Professionals*. Wiley, 2015. ISBN 9781119002253. URL `https://books.google.cz/books?id=retRCgAAQBAJ`.

Leo Yu-Ho Lo, Ayush Gupta, Kento Shigyo, Aoyu Wu, Enrico Bertini, and Huamin Qu. Misinformed by visualization: What do we learn from misinformative visualizations? *Comput. Graph. Forum*, 41(3):515–525, 2022. doi: 10.1111/cgf.14559. URL `https://doi.org/10.1111/cgf.14559`.

Ming Luo, Guihua Cui, and B. Rigg. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research Application*, 26:340 – 350, 10 2001. doi: 10.1002/col.1049.

Robert Matthews. Storks deliver babies (p= 0.008). *Teaching Statistics*, 22:36 – 38, 06 2000. doi: 10.1111/1467-9639.00013.

Leticia Miquilini, Mauro Ratis, Monica Lima, Natali Bento-Torres, Eliza Lacerda, Maria Côrtes, Anderson Rodrigues, Luiz Carlos Silveira, and Givago Souza. A proposed correction in the weighted method to score the ishihara test. *BMC Research Notes*, 12, 05 2019. doi: 10.1186/s13104-019-4320-2.

Ram Krishn Mishra, Harshit Raj, Siddhaling Urolagin, J. Angel Arul Jothi, and Nishad Nawaz. Cluster-based knowledge graph and entity-relation representation on tourism economical sentiments. *Applied Sciences*, 12(16), 2022. ISSN 2076-3417. doi: 10.3390/app12168105. URL `https://www.mdpi.com/2076-3417/12/16/8105`.

Humberto Moreira, Leticia Álvaro, Anna Melnikova, and Julio Lillo. Colorimetry and dichromatic vision. In Carlos M. Travieso-Gonzalez, editor, *Colorimetry and Image Processing*, chapter 1. IntechOpen, Rijeka, 2017. doi: 10.5772/intechopen.71563. URL `https://doi.org/10.5772/intechopen.71563`.

Lisa Charlotte Muth. What to consider when choosing colors for data visualization, Jan 2021. URL `https://blog.datawrapper.de/colors/`. visited 2023-01-03.

G. Paschos. Perceptually uniform color spaces for color texture analysis: an empirical evaluation. *IEEE Transactions on Image Processing*, 10(6):932–937, 2001. doi: 10.1109/83.923289.

Charles Poynton. *Digital Video and HDTV*. Morgan Kaufmann, 01 2003. doi: 10.1016/C2010-0-68987-5.

Wolfgang Rahfeldt. Files · master · wolfgang rahfeldt / colorblind · gitlab, Jan 2021. URL `https://gitlab.com/FloatFlow/colorblind/-/tree/master`.

Milan Straka. PCA, K-Means, Dec 2022. URL `https://ufal.mff.cuni.cz/~straka/courses/npfl129/2223/slides/?11#26`. Published as presentation, slide number 26.

Danielle Albers Szafir. The good, the bad, and the biased: five ways visualizations can mislead (and how to fix them). *Interactions*, 25(4):26–33, 2018. doi: 10.1145/3231772. URL `https://doi.org/10.1145/3231772`.

List of Figures

List of Tables

A. Attachments

A.1 User Documentation

A.1.1 Objective

The Colorblind Detector application aims to help users determine if a graph image is colorblind-friendly. It analyzes the colors in the image and checks if they are distinguishable for people with different types of color blindness.

A.1.2 System Requirements

- Windows Operating System
- Python version 3.7

A.1.3 Installation

Follow these steps to set up the application:

1. Download the zip Colorblindness Detector project from https://github.com/roubalovaHana/colorblindnessDetector
2. Extract the project to a desired location on the computer.
3. Open the project in IDE and create a 3.7 version Python interpreter.
4. Install the required Python packages by running the following command in the terminal:

   ```
   pip install -r requirements.txt
   ```

5. Once the installation is complete, launch the application.

A.1.4 Application interface

The Colorblind Detector application provides a user-friendly graphical interface. After launching the application, the main screen will appear with five buttons for controlling the application and three checkboxes for selecting the color vision deficiency.

A.1.5 Usage

Loading image

Press the **Load** button and find the desired image in the file manager. Selecting a folder of images is also possible by pressing the check button while in the file manager. In this case, the first image in the selected folder will display.

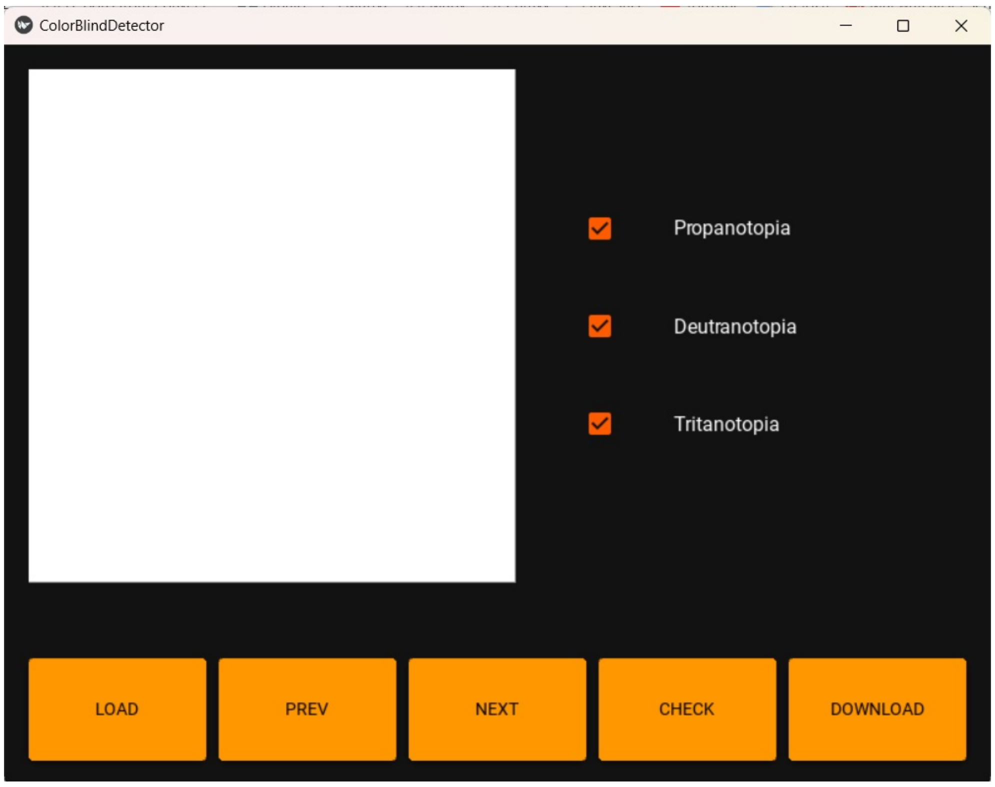

Figure A.1: User interface after application launch

Navigating folder

To navigate the images, use the **Prev** and **Next** buttons. The **Prev** button will display the previous image in the folder, while the **Next** button will display the following image.

Choosing the type of color vision deficiency for analysis

The checked boxes indicate the selected types of color vision deficiencies for analysis. Choose the desired combination by selecting appropriate checkboxes.

Starting analysis

Pressing the **Check** button will start the analysis, which may take a while. After the analysis, if the image is unfriendly, a warning will display next to the corresponding color vision deficiency. If no warnings appear, the picture is colorblind-friendly.

Downloading the analysis report

After the analysis, download the pdf report about the current image by pressing the **Download** button. The report will show the original image and a simulated image for each triggered deficiency. Therefore, if no issues were triggered, no report will download.

A.1.6 Limitations

The application assumes that the input image is in a standard PNG or JPEG format. It is not excessively large or complex. And that the image is of a single graph.

The application has no memory. When loading the same image, the analysis must be performed again.

A.2 Code Documentation

A.2.1 Input data

The input for this application is an image of a single graph in a JPEG or PNG format, selected by the user using the file manager. Input can also be a user-selected folder with images.

To test this application, one can utilize the TestFiles directory of prepared images or the GroupingDataset directory containing generated graph images.

A.2.2 Code Overview

The application consists of three main modules. The first module handles the graphical user interface. Second takes care of the underlying application logic, which is separated from the user interface module by a Facade. The third module handles the experiments performed to compare the performance of different color-grouping algorithms.

main.py

The *main.py* serves as the entry point for the application. It initiates the execution of the Kivy application and launches the user interface.

User interface

design.py

The *design.py* defines the handling methods for events triggered in the user interface, including all methods for pressing the buttons and navigating in the file manager.

design.kv

Defines user interface design and layout of the Kivy application using a declarative syntax specific to the Kivy library.

Application logic

facade.py

The *facade.py* implements the Facade design pattern. It serves as a separation of the application logic from the user interface logic and as a simplified entry point to the application logic. It defines two main methods, *find_issues*, and *generate_report*, that intermediate the communication with the application logic subsystem.

It also acts as the context for the Strategy design pattern that implements the color grouping algorithms. This functionality was used during the implementation phase. The best-chosen strategy is set in the constructor as the default in the final application.

color_grouping_algs.py

Defines three distinct implementations for grouping colors of an image using the Strategy design pattern. Both the abstract Strategy class and the concrete implementations are located here.

The context and client code are placed in the *evaluation.py* file.

The concrete strategies of grouping colors are implemented using the K-means algorithm from the *Sklearn* library and the Euclidean and the deltaE distance calculations, which are implemented in the *delatE_distance_calculator.py* file.

detection_algs.py

The *detection_algs.py* implements the analysis part of the application logic. For given grouped colors, it computes the distance between the original and the simulated colorblindness colors. Then triggers a warning if some of the original distances have shrunk considerably in the simulated versions. The results are conveyed using the *report_result_object.py* objects. For each type of colorblindness, one report object is created that carries the information on whether the corresponding issue was triggered.

deltaE_distance_calculator.py

Calculates the CIE2000 delatE distance for every pair of given colors using the *scipy* and *colormath* libraries. It returns the distances as an upper triangular matrix flattened to a list.

report_result_object.py

Represents warning objects. Stores information about the type of colorblindness and whether it was triggered or not. It also stores the simulated image for the corresponding colorblindness.

pdf_generator.py

The *pdf_genereator.py* generates the pdf report based on the report result object. It first renders the *report_template.j*2 to html with the given information and then converts the html to pdf using the *xhtml2pdf* library.

report_template.j2

The *report_template.j*2 contains the jinja2 template for the pdf report generation.

Experiments

dataset_generator.py

The method *generate_graph_images* generates the random test images using the *matplotlib* library and *genData* method adapted from
https://github.com/ddecatur/VizExtract/blob/main/create_graph.py.
Additionally, the method *generate_dataset* creates a csv dataset containing the generated images and required information about them. Furthermore, the *simulate_colorblindness* method simulates all types of colorblindness for them.

evaluation.py

Runs the experiments and calculates the evaluation metrics. It is implemented as a context to the Strategy pattern that implements color grouping algorithms from the *color_grouping_algs.py* file.

A.2.3 System Requirements

- Windows Operating System
- Python version 3.7

The application relies on the following external libraries:

- **Kivy**: Graphical interface development framework.
- **Colorblind**: Library for simulating color blindness and evaluating color distinguishability.
- **NumPy**: Computational and array manipulation library.
- **Scikit-learn (Sklearn)**: Machine learning library. Used for K-means algorithm and metrics.
- **Matplotlib**: Plotting library used for generating datasets.
- **Pandas**: Data manipulation library used for CSV file operations.
- **Pillow**: Image manipulation library.
- **SciPy** and **colormath**: Libraries used for calculating deltaE distance.
- **Jinja2** and **xhtml2pdf**: Libraries used for generating PDF reports.