Responsive Visualization Patterns and Tools

David Egger

706.424 Seminar/Project Interactive and Visual Information Systems SS 2023
Graz University of Technology

26 Apr 2024

Abstract

The field of data visualization provides techniques for representing complex data in graphical form to ease the process of identifying patterns and relations between data points. The web has become an increasingly important medium for sharing charts and visualizations. Visualizations on the web must be able to be viewed on a wide range of end user devices, with a variety of display sizes and resolutions. The automatic adjustment to as many possible usage scenarios is at the core of what is called *responsive visualization*.

This survey describes prior work in the field of responsive visualization. It collects and summarizes existing responsive patterns, which are commonly used to transform visualizations to fit all possible target devices. It then defines its own set of responsive visualization patterns, grouped into three kinds: *visual*, *interactive*, and *data* patterns.

Finally, the survey describes a number of tools which can be used to create responsive visualizations. These tools are grouped into responsive visualization libraries, responsive visualization systems, and responsive visualization transformation tools.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

The field of data visualization seeks to present data and information visually, so as to facilitate its rapid assimilation and understanding [Andrews 2024]. Early figures in the field, like William Playfair and Florence Nightingale, produced visual representations of data on paper [Infogram 2016].

Today, charts and visualizations are largely viewed online on the web. Smartphones and tablets are also increasingly displacing desktops and laptops as the viewing platform of choice. In the first quarter of 2023, already 58.33% of all web page views worldwide were requested from mobile devices [Statista 2023].

Responsive web design refers to a set of techniques which can be used to create web pages which adapt to the characteristics of the user's device [Marcotte 2014]. These days, all web design is responsive web design. Data visualizations which are capable of adapting to the different requirements of desktops, tablets, smartphones, and other devices, are called *responsive visualizations* [Andrews 2018b].

This survey describes prior work in the field of responsive visualization. It collects and summarizes existing responsive patterns and then synthesizes its own set of responsive visualization patterns, grouped into: *visual*, *interactive*, and *data* patterns. Finally, the survey describes a number of tools which can be used to create responsive visualizations. These responsive visualization tools are grouped into libraries, systems, and transformation tools.

Chapter 2

Responsive Visualization

In essence, the field of responsive visualization describes the process of designing visualizations capable of adapting to different device characteristics.

2.1 Mobile Visualization

Horak et al. [2021, page 37-40] explain in detail the factors which must be considered when designing for mobile devices such as phones, tablets, and smartwatches:

- *Usage factors*: Usage factors describe the impact of a user's posture when using a device as well as the position of the device itself. This is very different from desktop devices, which are operated typically by a person sitting in front of the device, resulting in fixed posture and position of the device.
- *Environmental factors*: Environmental factors represent surrounding influences, which may have impact on the user's perception and/or interaction capabilities. A noisy bus tour may prevent a user from catching audio messages or force the user to use one hand to hold on. Lighting conditions can influence how a visualization is perceived.
- *Data factors*: Data factors embody the difficulty to render visualizations for large datasets on devices with limited available screen sizes and/or computational power.
- *Human factors*: Human factors stand for the individual motivation, background knowledge, attention span, goals, and subjective preferences a user may have. Depending on those factors, completely different aspects of a visualization may be of interest to the user.
- *Device factors*: Device factors include screen size and interaction modalities. Issues like the fat-finger-problem [Horak et al. 2021, page 38] on touch devices have to be resolved for a visualization to become truly responsive.

2.2 Display Properties

The size of a screen can be measured in two ways, either in number of pixels or in physical size. When referring to the physical size of a screen, the correct term is *physical size* or *screen size*, which is typically stated in centimeters or inches. When speaking of the number of pixels, the correct term is *display resolution*, or simply *resolution* [Christensson 2019]. Typically, resolution is stated as *width*×*height*, where width is the number of horizontal pixels and height is the number of vertical pixels, for example 1920×1080.

Pixel density is a measure of detail, which sets the number of pixels in relation to the physical space. The corresponding unit is *pixels per inch* (ppi). The *aspect ratio* defines the relation between width and

```
@media only screen and (max-width: 40rem) {
  // styling for narrow screens
}

@media only screen and (min-width: 40rem) and (max-width: 60rem) {
  // styling for medium screens
}

@media only screen and (min-width: 60rem) {
  // styling for wide screens
}
```

Listing 2.1: Media queries for three viewport sizes, with breakpoints at 40 and 60 rem.

height of an object, for example 16:9.

2.3 Responsive Web Design

In 2010, Marcotte [2010] published his article entitled "Responsive Web Design", which was the catalyst for this new design approach to web sites and applications. The intention was to design only once for all kinds of devices, and to serve the same code from the same URL to all devices. A year later, Marcotte [2011] expanded on the article with a book of the same title.

Marcotte described three core technical concepts for responsive web design:

- *Flexible Grids*: The width and height of container elements should be defined in relative CSS units such as `%` and `em`, rather than absolute units such as `px`. Font sizes should also be specified in relative units such as `em` or `rem`, depending on whether the font sizes should cascade or not.
- *Flexible Images*: Images and other media elements should adapt their size to the available space, for example by specifying `max-width: 100%;`.
- *Media Queries*: Media queries were introduced in CSS3 and allow selected styles to be applied if certain conditions are met. In most cases, such conditions query the current viewport width, although other factors like aspect ratio, screen orientation, or screen resolution can be queried. Listing 2.1 illustrates three typical media queries.

Media queries with viewport widths are often used to define layout breakpoints, at which the screen layout changes. For example, as shown in Figure 2.1, setting layout breakpoints at 40 rem and 60 rem defines three layout widths: narrow, medium, and wide. This could be used to implement a one-column layout for narrow screens (<40 rem), a two-column layout for medium width screens (between 40 and 60 rem), and a three-column layout for wide screens (>60 rem).

For a web page or application, the *viewport* is the area of the page or application visible to the user [W3Schools 2023]. The viewport depends on the display properties of the device and the screen real estate used by the web browser, as well as on the size of the browser window.

Paddings and margins sometimes have to be reduced on narrower screens, and font sizes may have to be adjusted to the current viewport too. Some elements may have to be reordered or removed, depending on the initial structure of the user interface. Wider screens may also pose problems, like elements becoming too large or lines of text becoming too long. This is especially problematic for paragraphs of text, since lines longer than around 75 characters are harder to read [Rendle 2019].

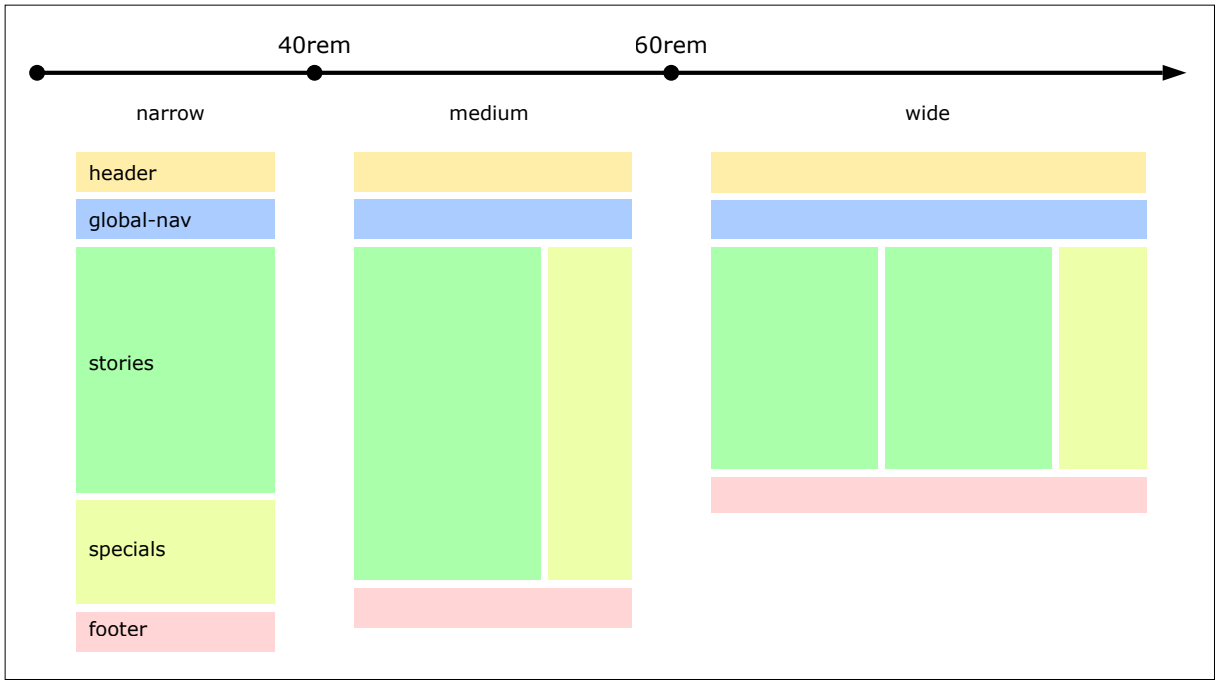


Figure 2.1: A responsive breakpoint diagram. Setting layout breakpoints at, say, 40 rem and 60 rem provides for three different layout widths: narrow, medium, and wide. The layout scales smoothly between breakpoints and changes at a breakpoint. [Used with kind permission of Keith Andrews.]

2.3.1 Responsive Design Strategies

In the early days of the web, developers designed applications only for desktops and laptops. Therefore, it is not surprising that when web browsers became available for mobile devices, the first strategy was to adapt a design for a desktop device to also fit a mobile device. This design strategy is known as *desktop-first* design.

As the number of mobile devices out-shipped the number of desktop and laptop devices in 2010, a rethinking of the design strategy started. Web designers began to design their applications for narrower screens first, later ensuring that the design responded well to make use of the extra space available on wider devices. This strategy became known as *mobile-first* design [Wroblewski 2011].

The increasing usage of the mobile web led to a strategy which went even further, with the objective to solely design for mobile devices, so-called *mobile-only* design. This strategy can make sense for projects which are clearly dominated by mobile devices, like location-based services [Dreher 2023].

Perhaps the best strategy is to define a number of logical layout widths (say narrow, medium, and wide) using layout breakpoints, like those in Figure 2.1, and to design for all of them in parallel. A good name for it would be *everything-in-parallel* design.

2.3.2 Modern Responsive Design

Over ten years have passed since the term responsive web design was first used [Marcotte 2010]. Naturally, the web continued to evolve and new tools for creating responsive web applications were created. In his online article, Shadeed [2023] discusses the current state of responsive web design in 2023.

The core message is to reduce the number of media queries and instead use more recent CSS layout techniques like Flexbox [MDN 2023a] and Grid [MDN 2023b], and viewport units [MDN 2023c]:

- The Flexbox property `flex-wrap` can be used to make elements fill up the available space, and realign themselves automatically into additional rows if space becomes too narrow.

- CSS Grid layout allows elements to be placed in a 2d grid, with sizing according to a variety of criteria, such as auto-resizing for columns:

  ```
  .grid {
    grid-template-columns: repeat(auto-fit, minmax(10rem, 1fr));
  }
  ```

  Here, the columns will always have a minimum width of 10rem. If the grid container grows enough to fit another column without breaking the 10rem constraint, the number of columns will automatically increase. A more detailed explanation can be found at [Soueidan 2017].

  Another possibility is to explicitly assign `grid-template-areas` for specific named elements:

  ```
  grid-template-areas: "a a a"
                       "b c c"
                       "b c c";
  ```

- With the introduction of CSS comparison functions, another tool for increasing the fluidity of layouts became available. For example, the clamp function allows font sizing to fluidly change between a minimum and maximum size:

  ```
  h1 {
    font-size: clamp(2rem, 2rem + 0.5vw, 3rem);
  }
  ```

- Size container queries are similar to media queries, but refer to the size of the parent element (container), rather than the viewport. This is often what is wanted, so size container queries can be expected to replace media queries for responsive layout. Container query units (cqw, cqh) work like viewport units (vw, vh), but are relative to the corresponding parent container.

- Style container queries allow the current styling of a container element to be queried, so as to conditionally apply styling to its contents.

At the time of writing, all of the above CSS features are supported by all modern web browsers, according to Deveria [2023], with except to style container queries which are still being standardized.

2.3.3 Avoiding Horizontal Scrolling

In his blog post, Juviler [2021] explains why it is generally a bad idea to encounter horizontal scrolling in a web application:

- Since the beginning of the web, the convention was and is to scroll vertically through a web application, not horizontally. Changing this rule would result in a higher cognitive workload for the user.

- Many users might simply not notice that horizontal scrolling was possible (discoverability).

- Vertical scrolling is easy to do with a mouse wheel, horizontal scrolling, on the other hand, is not. Similarly, users of mobile phones have a higher range of motion to scroll vertically than horizontally.

For these reasons, it is generally advisable to avoid horizontal scrolling in web development and to deal with issues of fitting content into narrower widths in other ways.

2.4 Responsive Visualization Challenges

In responsive visualization, a common strategy is to first design a visualization for wider viewports and then apply responsive transformations to adapt the visualization to also fit narrower ones [Kim et al. 2021]. Such transformations often include reducing or removing non-essential information in order for

the visualization to take up less space. However, the author has to be careful not to change the original wider version in a way that communicates a different message.

As Kim et al. [2021] explain, the key challenge when creating a responsive visualization is the density-message tradeoff. This means shrinking the size of a chart alone will not be sufficient to create a good visualization for a smaller viewport. They distinguish three types of challenge in this context:

- *Graphical Density Challenges*: Visualizations often include many smaller elements like axis ticks, points, lines, labels, etc. When a visualization shrinks, these elements can only shrink to a certain extent. Otherwise, users would not be able to read labels or differentiate between similar-looking elements like data points with different radii. However, if elements do not shrink with the visualization, this will inevitably lead to overlapping elements. A typical approach for avoiding such scenarios is to thin out elements in a way that preserves the original message, but does not show too many elements.

- *Layout Challenges*: Visualizations are often built from a set of components. If enough width is available, it makes sense to place the legend to the right of the chart. If less width is available, placing the legend above or below the chart is a more reasonable approach.

- *Interaction Complexity Challenges*: For the majority of interaction modalities on desktop devices, equivalent interactions exist for mobile devices. However, some interactions, like hovering or navigating by tabbing (pressing the Tab key), are only available when using a mouse or keyboard, respectively [Korduba et al. 2022, page 4]. Additional factors like the precision difference between a finger and a mouse pointer must also be taken into account when designing responsive visualizations.

2.5 Approaches to Responsive Visualization

Two levels of responsiveness can be achieved when creating visualizations: multiple separate pre rendered visualizations, or a single responsive visualization.

In the first approach, certain ranges of viewport width are predefined and visualizations are created for each of them beforehand. This is typically done by first creating a base visualization and then applying responsive transformations at fixed-width breakpoints. The breakpoints should be chosen carefully to provide appealing visualizations for as many devices as possible. This method comes with the advantage of having stable, fixed visualizations for specified screen sizes, easing the integration of visualizations. Certain visualizations can be prepared as raster graphics, if necessary due to performance constraints. Recently developed tools like Hoffwell's visualization system [Hoffswell et al. 2020] support this approach. Modifications to one visualization can be propagated down to the others, and a simultaneous preview of all visualizations helps avoid inconsistencies. While using multiple pre rendered visualizations has advantages, it breaks the principle of shipping a single codebase for all kinds of devices.

The second approach is to create one visualization for specific viewport dimensions, which is capable of adapting itself automatically and fluidly if the dimensions change. This method comes with the advantage of visualizations being always able to adjust perfectly to the available space. Furthermore, the transition between different visualization sizes is smoother than with multiple pre rendered visualizations. For these reasons, visualizations of this type are considered to be truly responsive. The disadvantage of creating a single responsive visualization is the complex process behind it. Not all transformations are applied simultaneously and are sometimes capable of influencing each other. Furthermore, breakpoints will still be necessary to switch between different layouts at specific viewport or container widths. A good example of a tool for creating a single responsive visualization is RespVis [Andrews et al. 2023]. The designer is provided an API for creating visualizations directly in the DOM of an HTML document. Chart components can be laid out using standard CSS layout techniques like Flexbox and Grid.

Chapter 3

Responsive Visualization Patterns

Kim et al. [2021] list 76 design strategies for transforming wider visualizations into narrower ones. Each strategy includes a target element and an executed action. Targets are grouped into five types: data, encoding, interaction, narrative, and references/layout, while actions are split into five kinds: recomposition, rescaling, transposition, reposition, and compensation.

This chapter presents a curated set of 16 tried-and-tested responsive visualization patterns, divided into three groups: *visual patterns* (10), *interaction patterns* (3) and *data patterns* (3). These patterns are generic, best practice examples, which can theoretically be applied by any arbitrary visualization system. However, since the web is the primary medium for consuming visualizations, all the patterns assume web-based visualizations. Practical examples of the patterns can be seen at the showcase web sites of Andrews [2018a] and Egger and Oberrauner [2023b].

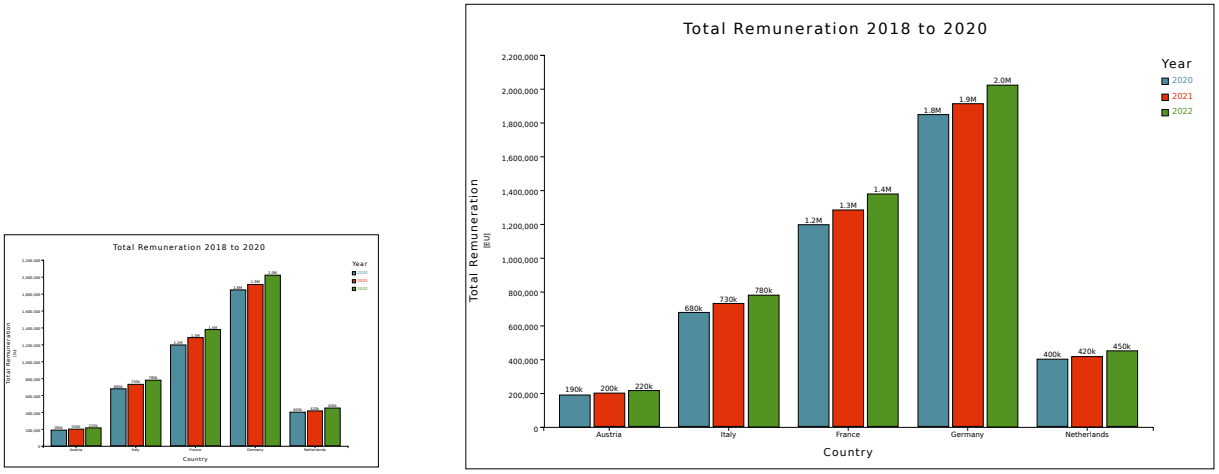
3.1 Visual Patterns

Visual patterns are applied directly to visualizations in order to change the state and appearance of a visualization's components to maximize the user experience depending on the available space:

- V1: Scaling Entire Chart Down
- V2: Repositioning Element Labels
- V3: Using Tooltips Instead of Element Labels
- V4: Rotating Axis Tick Labels
- V5: Shortening Labels and Titles
- V6: Scaling Labels Between Minimum and Maximum Size
- V7: Scaling Down Visual Elements
- V8: Hiding Elements and Labels
- V9: Rotating Chart 90°
- V10: Using a Different Chart

3.1.1 V1: Scaling Entire Chart Down

The most straightforward visual transformation is to scale the whole chart down. However, if downscaling is too excessive, this approach becomes problematic as certain components become unreadable. While there is no fixed rule on a minimum font size, there is some consensus around the root font size being 16px



(a) Entire chart simply scaled down.

(b) Wide version of chart.

Figure 3.1: V1: A grouped bar chart illustrating that a chart can become unreadable when the entire chart is simply scaled down. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

or 12pt [PennState 2023]. The Web Content Accessibility Guidelines (WCAG) documentation also states that text should be scalable up to 200% of its original size [WCAG 2023]. Text labels in particular should not be scaled down below this tolerance. Similar issues arise when downscaling markers for data points; they can become too small to distinguish. Another problem occurs when downscaling a visualization in just one direction. This can lead to an unacceptable degree of distortion. As Andrews [2018b] states, it is not enough to just scale down a visualization to make it responsive. If it were, it would be the only pattern needed for achieving responsiveness. Figure 3.1 illustrates the problems which occur when only scaling down the entire visualization.

3.1.2 V2: Repositioning Element Labels

Repositioning of labels can help avoid intersections and clutter. In web-based visualizations, this can be achieved using media queries or container queries. Both of these are discussed in Section 2.3. Figure 3.2 illustrates how repositioning element labels can help avoid clutter in a visualization.

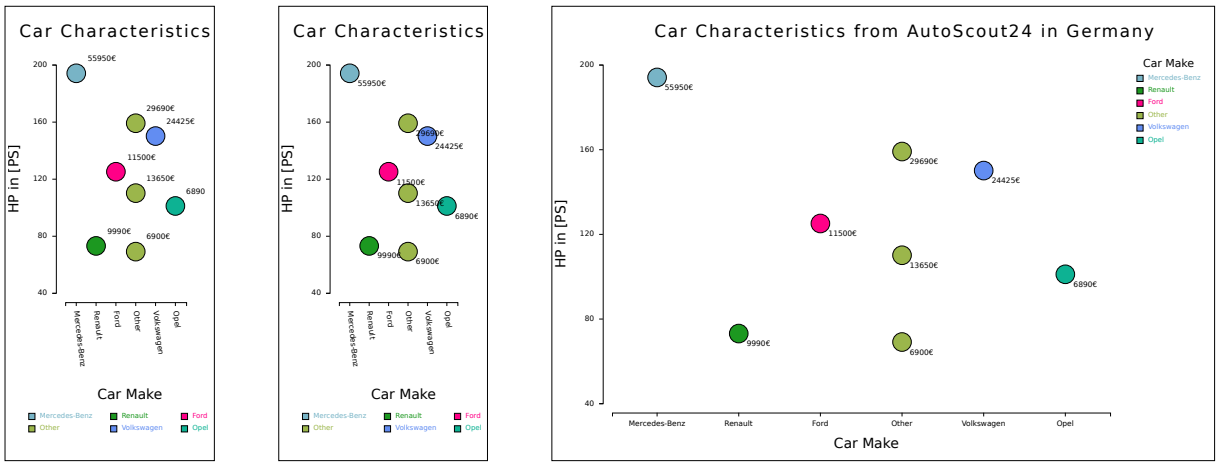
3.1.3 V3: Using Tooltips Instead of Element Labels

If a visualization contains many data points, but has only limited space, a helpful pattern is to hide the labels of all elements and display an element's label upon hover or selection. In web-based visualizations, the CSS hover selector can be used to display a tooltip while hovering over an element with a pointer device. For touch devices, a single touch can be used to toggle the display of a tooltip. The disadvantage of this pattern is that it is not immediately obvious to users (discoverability). A practical example of the usage of tooltips can be seen in Figure 3.3.

3.1.4 V4: Rotating Axis Tick Labels

An especially useful method for avoiding intersections and clutter of axis labels is to rotate the x-axis tick labels. As the available horizontal space decreases, this approach helps preserve more of the original axis label information (rather than thinning out or shortening the axis labels). The rotation of y-axis tick labels can also be considered, but is less useful.

For web-based visualizations, a possible way to achieve rotating tick labels is to use a combination of JavaScript and CSS. A JavaScript algorithm and event listeners are necessary to detect the currently

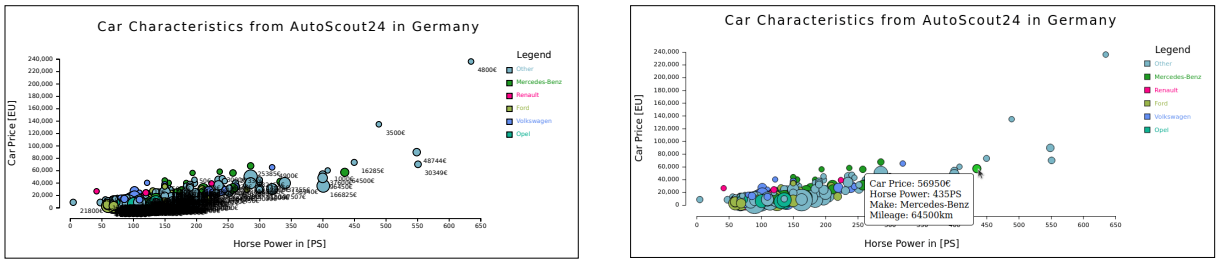


(a) Narrow: Adapted label positions.

(b) Narrow: Unchanged label positions.

(c) Wide: Plenty of room for element labels.

Figure 3.2: V2: A scatterplot containing element labels. Clutter can lead to labels overlapping elements at narrow widths. Repositioning the element labels resolves the issue. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]



(a) Overwhelming number of element labels.

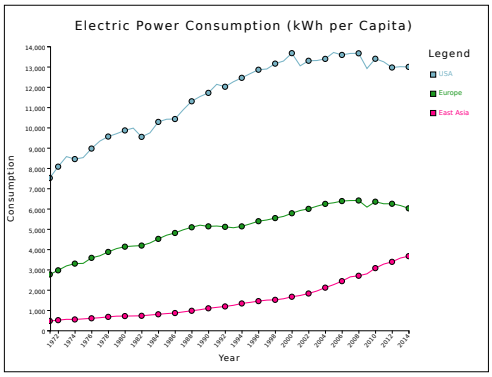
(b) Tooltips instead of labels.

Figure 3.3: V3: If a scatterplot contains many data points in a narrow space, it makes sense to use tooltips instead of element labels. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

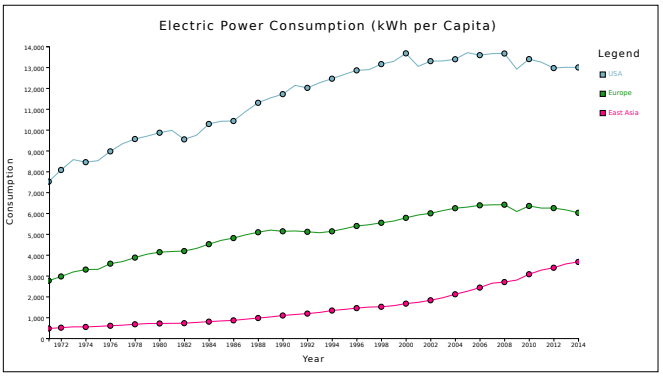
appropriate angles of labels, while the styling of the labels themselves can be achieved via the CSS properties `rotate` or `transform: rotate()`. The advantage of this pattern is the preservation of the original axis information. On the downside, the axis labels are harder to read, since the natural reading direction is not retained. A practical example of a chart making use of rotating x-axis tick labels can be seen in Figure 3.4.

3.1.5 V5: Shortening Labels and Titles

A common technique for avoiding clutter and overlaps is to have different formats for labels for different space requirements. Numbers, for example, can be shown in full length if enough space is available (e.g. 2,200,000), but shortened to well-known abbreviated forms (e.g. 2.2M) when space is limited. Similar strategies can be applied to other types of text such as dates, organization names, and geographic locations. However, when using shortened labels in a visualization, any resulting information loss, such as that caused by the rounding of numbers, should also be considered. Shortened texts should use well-known and understandable formats so as not to confuse users. Figure 3.5 demonstrates how the technique can be applied in practice.

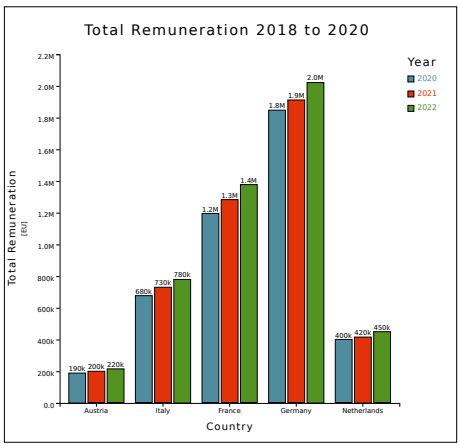


(a) Narrow: Rotated x-axis labels.

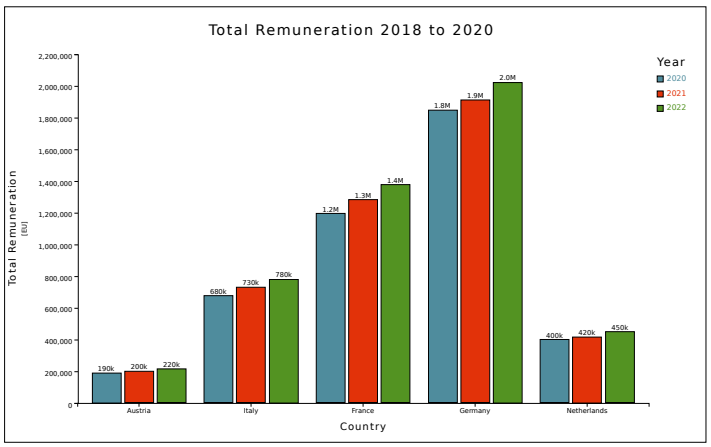


(b) Wide: Room for horizontal x-axis labels.

Figure 3.4: V4: A multi-line chart which avoids intersection of x-axis tick labels at narrow widths by rotating the labels. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]



(a) Narrow: Shortened y-axis labels.



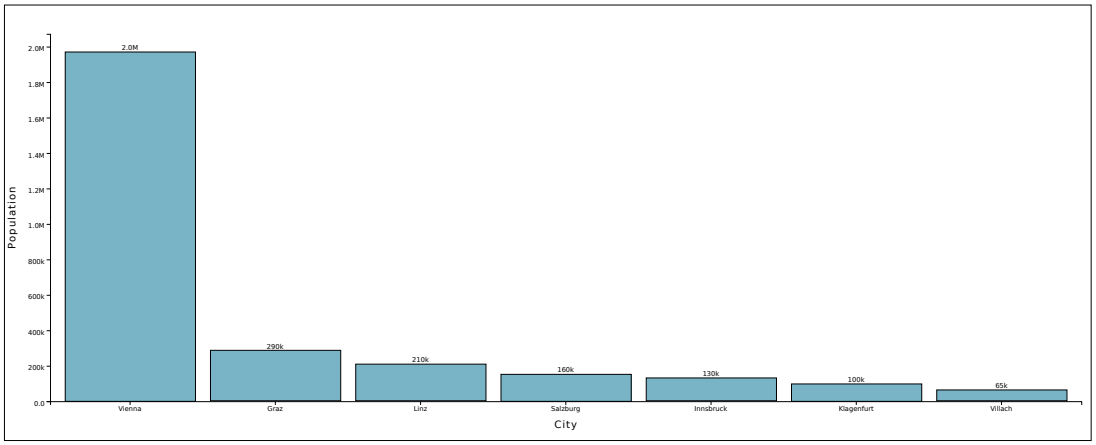
(b) Wide: Original y-axis labels.

Figure 3.5: V5: A grouped bar chart with label shortening applied to the y-axis tick labels. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

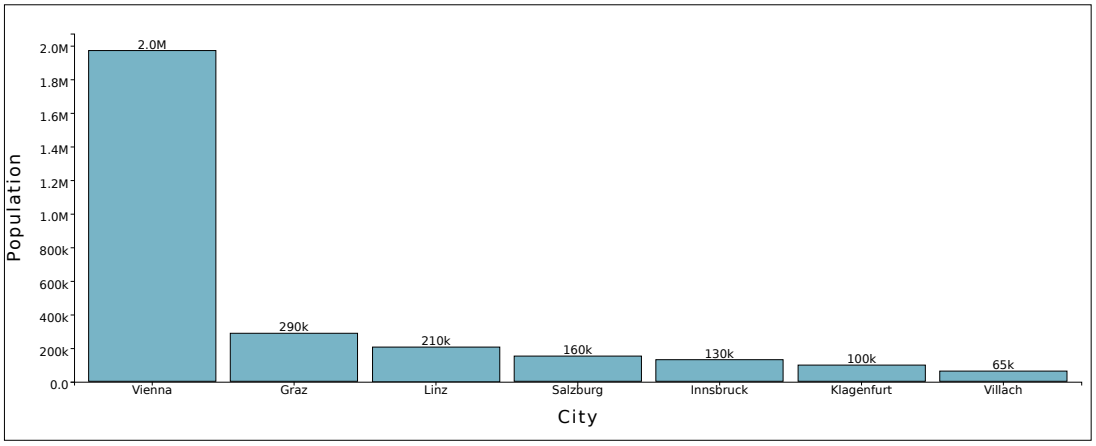
3.1.6 V6: Scaling Labels Between Minimum and Maximum Size

Having different font sizes for different space requirements can be a useful method not only for increasing the readability of labels, but also improving the overall aesthetics of a visualization. Having larger font sizes where space allows looks much better than choosing the easier, safe option of always having the same smaller font size, as can be seen in Figure 3.6.

For web-based visualizations, there are two approaches to implement this transformation. The first relies on media or container queries and defines fixed font sizes for different breakpoints. The other uses CSS comparison functions to fluidly transition between smaller and larger font sizes. The second approach can also be combined with container query units to relate the font sizes to the size of the visualization container. This method relies heavily on modern CSS techniques, which are discussed in Subsection 2.3.2.



(a) Fixed-size (small) unscaled labels.



(b) Scaled labels.

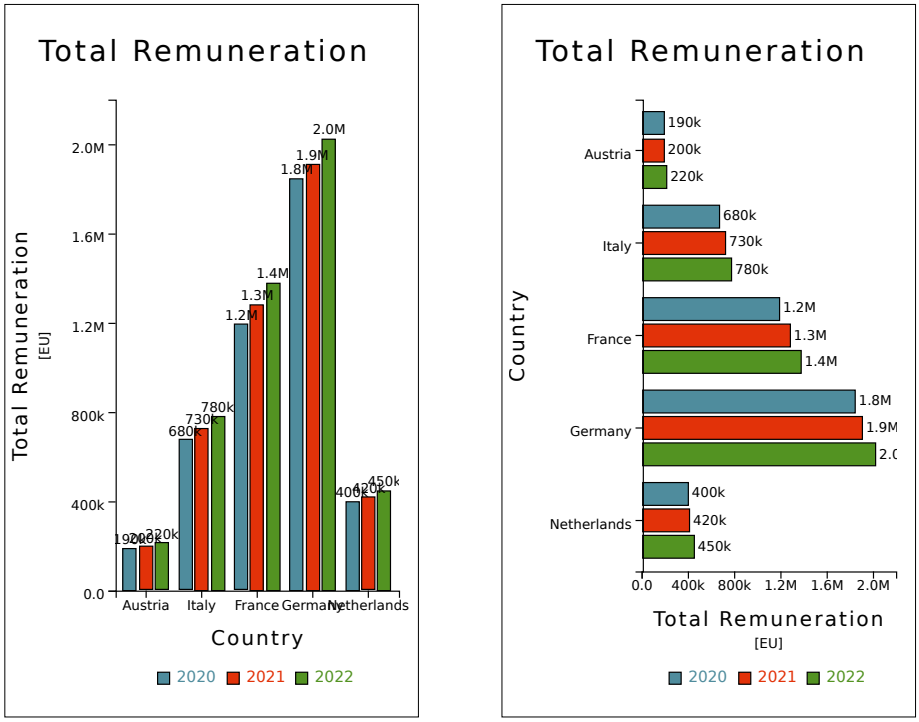
Figure 3.6: V6: A bar chart with scaled labels. When a visualization has the available space, it makes sense to enlarge the font size of the labels and titles. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

3.1.7 V7: Scaling Down Visual Elements

Scaling is a transformation applicable at multiple levels and in multiple variants. One form of scaling is scaling down selected visual elements. This approach varies for different kinds of element. A bar element, for example, can be scaled in both horizontal and vertical directions without problems. The same holds for lines in a line chart, since these simply have to update their thickness and target points. A marker for a data point is more complicated, since it must retain its aspect ratio during scaling to avoid distortion. The same holds for any other marker elements.

Freely scaling down a selection of elements is applicable only to visualizations with a variable aspect ratio. Other types of visualization like pie charts, chord diagrams, and maps can apply this kind of transformation to their elements only when retaining their original aspect ratio. A good example of a practical use case of this pattern can be seen in Figure 3.5. The width of the bars of the grouped bar chart becomes smoothly narrower as less space is available.

The advantages of this technique are that much space can be saved without information loss and smooth transitions via event listeners appear very natural. On the downside, the pattern is only applicable to visualizations with a manageable number of elements, since otherwise elements are already quite small even at larger widths. Another disadvantage is that line elements in line charts appear steeper at narrower widths and flatter at larger widths, affecting the perception of the original message.



(a) Horizontal space is cluttered.

(b) Transposing the chart so it extends vertically rather than horizontally.

Figure 3.7: V9: Rotating a grouped bar chart by 90° to make better use of vertical space. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

3.1.8 V8: Hiding Elements and Labels

One possibility to adapt a visualization to narrower widths is to remove some elements or labels completely. When applying this technique, care must be taken to not alter the original message of the visualization. The advantage of this pattern is that an arbitrary amount of space can be saved by removing enough elements. However, this comes at the cost of information loss with respect to all the removed elements. When removing whole categories or dimensions, it is advisable to offer interactive possibilities, so the user can choose which dimensions or categories are of interest. This is described in more detail in Subsection 3.2.2. A practical example of hiding labels can be seen in Figure 3.3, which demonstrates how visible labels can be replaced with tooltips.

3.1.9 V9: Rotating Chart 90°

Transposing or rotating a chart by 90° can be a convenient way to align the dimension which requires more space vertically rather than horizontally. Even if vertical scrolling is required, it is much more acceptable than horizontal scrolling. An example of a rotated grouped bar chart can be seen in Figure 3.7. The advantage of this pattern is that no information is lost by the transformation process. The main disadvantage is the major change of the visualization which may affect other ongoing transformations.

3.1.10 V10: Using a Different Chart

At some point, the best option may be to completely swap a visualization for a different one. This technique is the last resort, when other techniques either lead to unacceptable information loss, alteration of the original message, distortion of the visualization, or unreadable elements or labels. The advantage of this technique is that it provides a solution where all others do not. On the downside, maintaining two different visualizations is more effort.

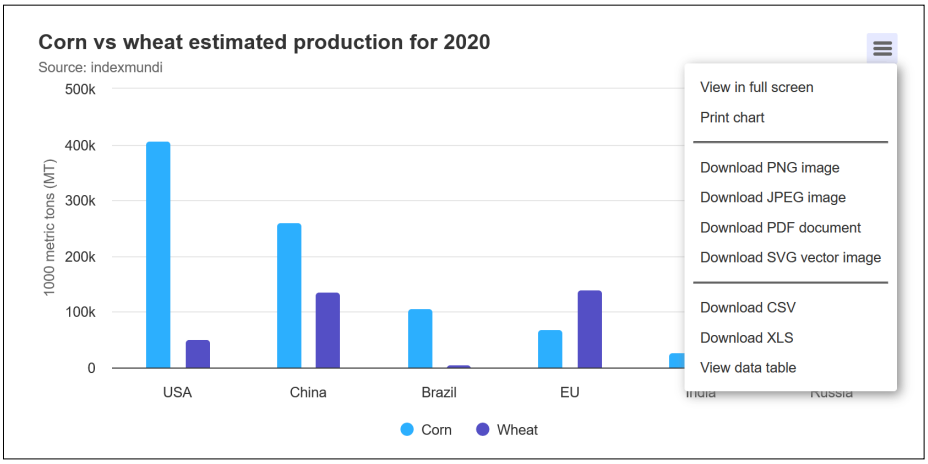


Figure 3.8: I1: The menu provided by Highcharts. [Image created with Highcharts [Highsoft 2023] by Keith Andrews and used with kind permission.]

3.2 Interaction Patterns

Interaction patterns support responsiveness by providing interactive functionality such as zooming and filtering:

- I1: Supporting Toolbar and Menus
- I2: Filtering Dimensions and Records
- I3: Supporting Zooming

3.2.1 I1: Providing a Toolbar or Menu

Interactivity bound to visual elements, such as hovering or a right-click context menu, suffers from poor discoverability. The user has to know such actions are possible or discover them by trial and error. A toolbar or menu, on the other hand, is visible to users, and its features can be explored. Typical actions provided by toolbars or menus include being able to download a chart as SVG, download the data as CSV, view the chart in full screen, view the data as a table, and show and hide specific records and dimensions in the data.

The advantages of this pattern are the theoretically unlimited interaction options that can be added to a visualization and the high likelihood of the toolbar being discovered by the user. Disadvantages of the pattern include the space needed for the additional control elements and the effort for the user to find them if they are hidden by default. A practical example of a menu from Highcharts is shown in Figure 3.8. Another example showing the toolbar provided by Plotly.js can be seen in Figure 3.9.

3.2.2 I2: Filtering Dimensions and Records

If space requirements are very tight, there is often no other solution than removing information from a visualization. However, when doing so it is a good idea to empower the user to choose which dimensions or records should be shown or hidden. The user may not be able to see all the data at once, but still has access to all information if necessary.

Possible interaction elements for the filtering of data can be the legend of a chart, the elements themselves, or separate control elements such as dropdown menu. Figure 3.10 shows the filtering of records from a grouped bar chart.

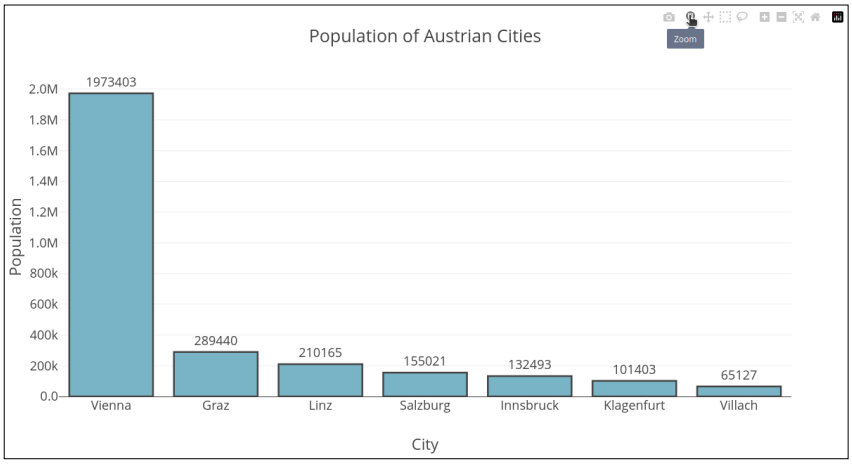
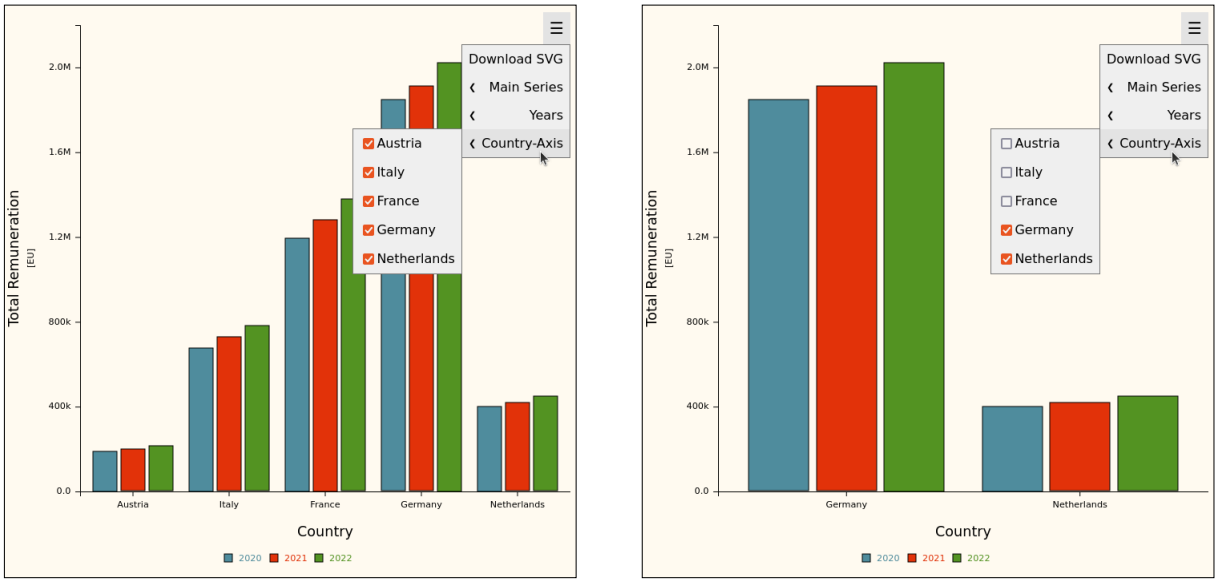


Figure 3.9: I1: The toolbar provided by Plotly.js. [Image created with Plotly.js [Plotly 2023] by the author of this survey.]



(a) All five records visible.

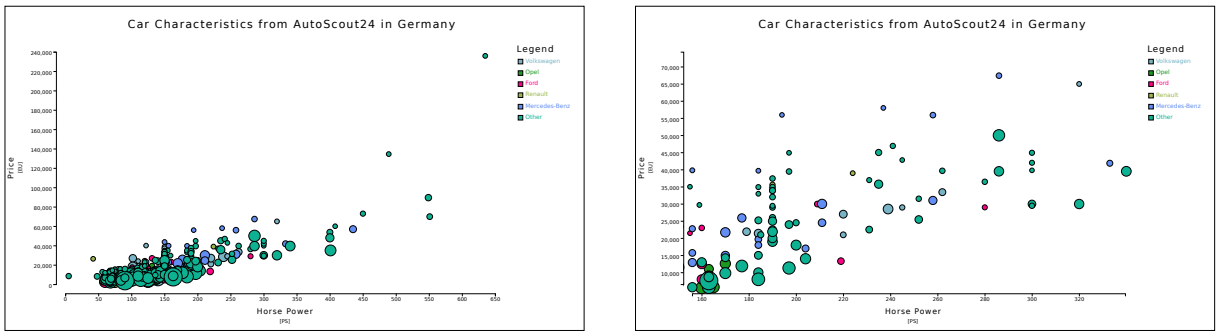
(b) Only two of five records visible.

Figure 3.10: I2: A grouped bar chart where records (Countries) or dimensions (Years) can be filtered with a control menu. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

3.2.3 I3: Supporting Zooming

Zooming is a crucial tool for overcoming the problems of limited resolutions and narrow screens. The standard approach, *geometric zoom*, allows a user to control the magnification of a visualization, and thereby trade the space needed for irrelevant information for more space for areas of interest [infovis-wiki 2006]. The scatterplot example presented by Egger and Oberrauner [2023b] demonstrates perfectly how this technique can be combined with grabbing and panning to make data points accessible. Figure 3.11 shows how zooming can be used to solve problems with dense data and intersecting elements.

In *fisheye zoom*, the focus area is magnified and the surrounding context area reduced, like using a magnifying glass to view the chart or visualization. Instead of removing the context completely, it is distorted to take up less space. For cartesian visualizations, with perpendicular axes such as x and y, *cartesian zoom* can be used. This technique divides the chart into a grid of cells and distorts the size of



(a) Unzoomed. **(b)** Geometric zooming.

Figure 3.11: I3: A scatterplot with many overlapping data points. To enable the user to inspect all points, geometric zooming is supported. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

the cells such that more interesting cells are enlarged to show more detail, while surrounding cells are made smaller. Fisheye and cartesian zoom are described in detail by Sarkar and Brown [1992], interactive examples can be explored in the responsive scatterplot example by Andrews [2018a].

Another approach is called *semantic zoom* [Bederson and Hollan 1995]. This form of zooming does not simply change the sizes of items, but considers which items to display and how to display them. To apply the technique, a form of data structuring is needed to create different layers of detail. Depending on the current zoom factor, elements can be removed, split into sub-elements, or change size or shape [infovis-wiki 2014].

3.3 Data Patterns

The applicability of visual patterns is highly dependent on the size of the dataset and the chosen visualization type. In many cases, it is necessary to group and transform the original data to obtain new datasets and statistics, which can be visualized more easily. Such data patterns include:

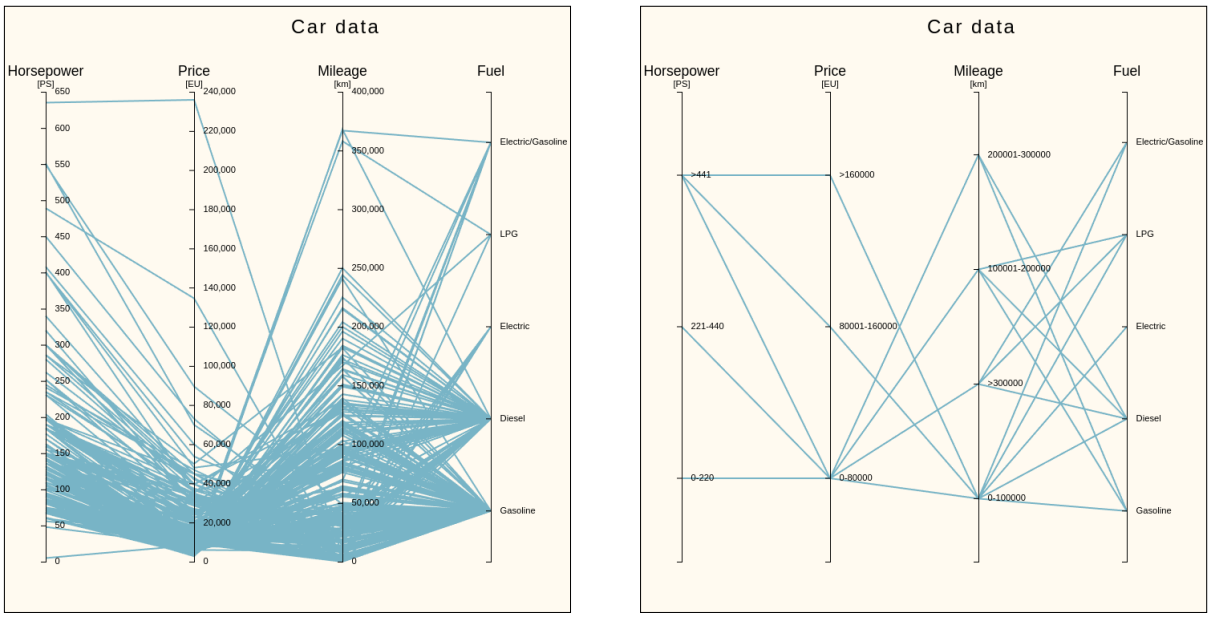
- D1: Data Generalization
- D2: Data Aggregation
- D3: Data Clustering
- D4: Data Sampling

3.3.1 D1: Data Generalization

Data generalization combines many data points into manageable groups. For example, individual ages can be binned into age ranges [Satori 2021]. The approach allows information to be presented in a more compact way, consuming less space and not overwhelming the user with too many data points.

An example can be seen in Figure 3.12, which shows a parallel coordinates chart about used cars in Germany. In Figure 3.12a, each car is displayed as its own polyline. After binning the records along the first three dimensions, classes of cars have been created, allowing Figure 3.12b to show a different view on the data.

Rabinowitz [2014] shows another way of how data generalization can be used to create responsive visualizations. He created an interactive online prototype of a scatterplot, where the user can select how much space is available to the visualization and how many data points should be included. The visualization transforms into a heatmap if the density of the data points exceeds a certain threshold. The



(a) Each car is displayed as its own polyline.

(b) Binning produces classes of car.

Figure 3.12: D1: A parallel coordinates chart with binning applied to the first three dimensions. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

resulting heatmap is a generalized version of the scatterplot, in that it bins data points into a grid with the cell coloring indicating the data point density, as shown in Figure 3.13.

3.3.2 D2: Data Aggregation

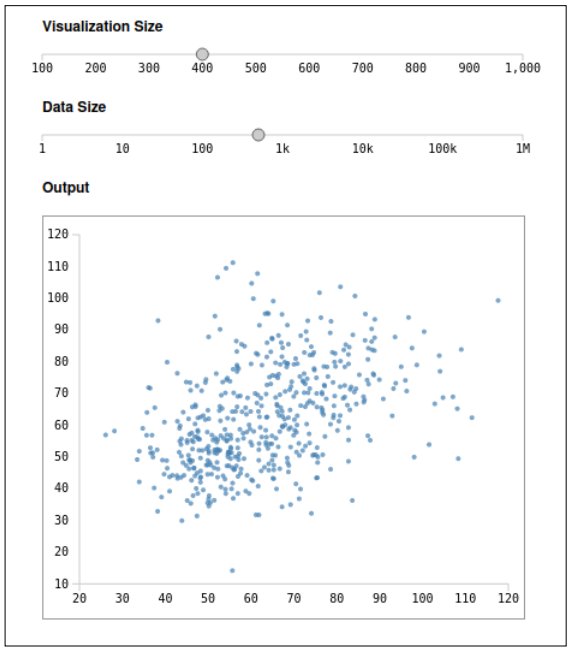
Data aggregation is a method used to summarize the information from a collection of data points into one or multiple useful statistics, such as average or sum [Zanini 2023]. When creating responsive visualizations, it allows the presentation of large amounts of underlying data as single visualization elements.

Figure 3.14 shows the global surface temperature anomalies of the years 1850 and 2023, i.e. the difference in global surface temperature in those years compared to the average temperature in the 100 years from 1901 to 2000 [NCEI 2024]. The bar chart in Figure 3.14a shows the data for 1850 and 2023 in monthly intervals. The bar chart in Figure 3.14b aggregates the monthly data into two bars, one for each of the two years. This version of the chart is much more compact, but the original message of a significant rise in average global surface temperature between the years 1850 and 2023 is preserved.

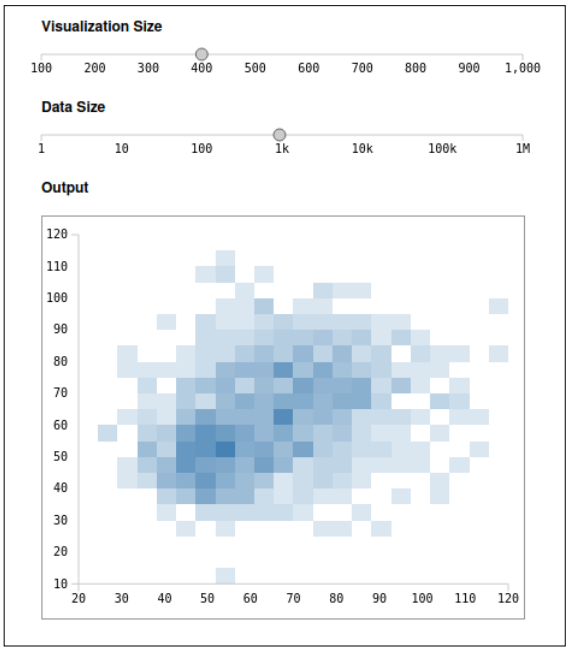
3.3.3 D3: Data Clustering

Clustering or cluster analysis groups objects (records) based on their similarity. The most common clustering algorithms include: hierarchical clustering, k-means clustering, model-based clustering, density-based clustering, and fuzzy clustering [Matilda 2023].

For responsive visualizations, clustering algorithms enable new options for dealing with large datasets as it allows the abstraction of similar data points within a dataset. One example is the use of agglomerative hierarchical clustering to replace overlapping data points with a cluster element, with interactions such as displaying cluster information and toggling presentation as a cluster versus individual data points.

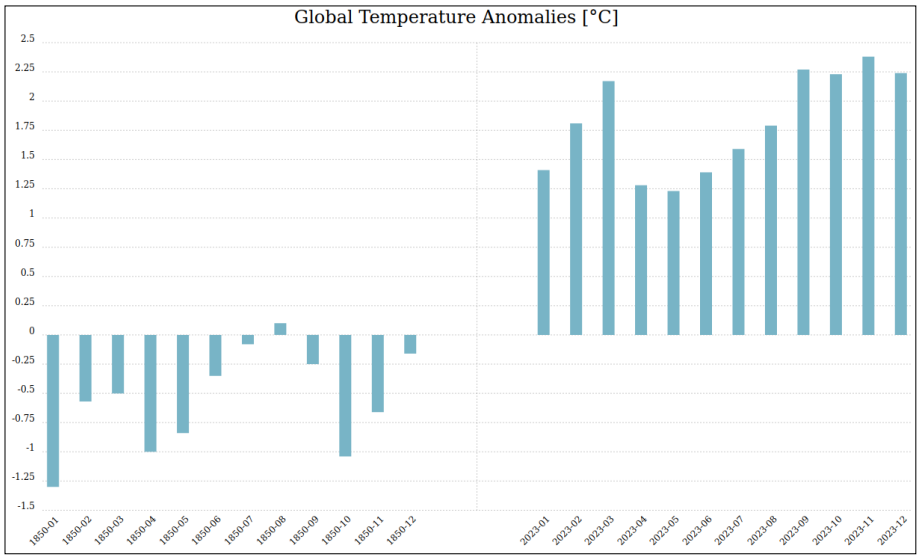


(a) Each data point is displayed individually.

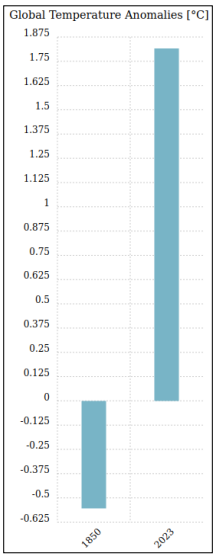


(b) Binning produces cells shaded according to data point density.

Figure 3.13: D1: A scatterplot transforms into a heatmap if a certain threshold of data point density is exceeded. [Images created with Rabinowitz' prototype [Rabinowitz 2014] by the author of this survey.]



(a) Monthly data.



(b) Yearly data.

Figure 3.14: D2: Two versions of a bar chart with different time aggregation intervals. [Images created with Chartist [Kunz 2017] by the author of this survey.]

3.3.4 D4: Data Sampling

Data sampling is a technique typically used in statistical analysis to identify patterns and trends in a population by extracting, processing, and analyzing a representative sample of an overall population [Egnyte 2022]. It can also be used to improve the responsiveness and performance of visualizations by avoiding overplotting.

Probabilistic sampling has the objective of creating samples which represent the overall population as accurately as possible. Different types of probabilistic sampling include: random sampling, stratified sampling, cluster sampling, and systematic sampling. Non-probabilistic sampling techniques are less rigorously representative and include: convenience sampling, quota sampling, snowball sampling, and purposive sampling.

A good example of how data sampling can be used to create responsive visualizations can be seen in Figure 3.11. The scatterplots show a random sample of 500 cars from the original 46,405 cars in the dataset [Ander 2021].

Chapter 4

Responsive Visualization Tools

A number of tools are available to make it easier for developers to create responsive visualizations. These can be grouped into three types:

- *Responsive Visualization Libraries*: JavaScript code libraries which can be used by developers to run responsive visualizations directly in the browser. They help developers create highly customizable, interactive visualizations adaptable to all possible space requirements.
- *Responsive Visualization Systems*: These systems provide a user interface for creating responsive visualizations without having to write any code. They can be used by non-programmers, but provide less customizability than working directly with code. The created visualizations can then be embedded into the desired applications.
- *Visualization Transformation Tools*: This kind of tool assists in the design of responsive visualizations by taking a base design and automatically applying transformations to it to obtain visualizations which fit other space requirements.

4.1 Responsive Visualization Libraries

Responsive visualization libraries help developers code responsive visualizations which run directly in the browser. Five responsive visualization libraries were explored: Chart.js, Plotly.js, Chartist, Highcharts, and RespVis. The following criteria were used to compare them:

- *Render Method*: One of SVG-DOM, Canvas2D, or WebGL.

  SVG-DOM refers to the use of JavaScript to dynamically create and modify SVG nodes in the browser's internal data structure (DOM). SVG charts are vector graphics and are freely scalable. They can be styled by CSS and can be relatively easily saved as SVG files.

  Canvas2D refers to the use of the HTML `<canvas>` element and its `2d` rendering context to draw graphics to a 2d pixel-based canvas. Swensen [2021] discusses the differences between SVG rendering and canvas rendering.

  WebGL refers to the use of the HTML `<canvas>` element and its 3d `webgl` or `webgl2` rendering context to create scene-based 3d graphics with the WebGL 3d graphics API [Khronos 2024]. WebGL is a low-level API, but its graphics are hardware accelerated where a device has a GPU, making WebGL graphics highly performant. WebGPU is a new low-level 3d graphics API [W3C 2024], designed to supersede WebGL.
- *Available Charts*: The suite of chart types provided by the library.
- *Licensing*: The license(s) under which the library is available.
- *Bundle Size*: The unpacked size and the minified gzipped size of the library bundle available in the

City	Population
Vienna	1,973,403
Graz	289,440
Linz	210,165
Salzburg	155,021
Innsbruck	132,493
Klagenfurt	101,403
Villach	65,127

Table 4.1: The population of seven Austrian cities from 2023 [Hernández 2023]. The dataset was used to create a simple bar chart with each of five visualization libraries.

npm package registry, according to Bundlephobia [2023].

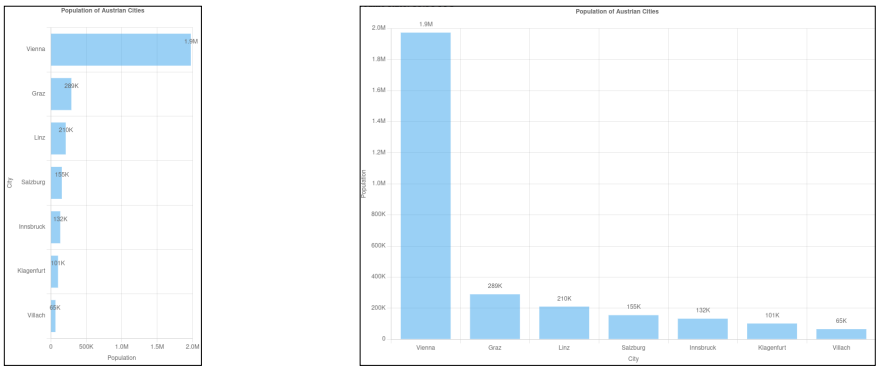
- *Popularity*: The number of stars on GitHub.
- *Performance*: Performance is measured by the lowest frame rate (fps) occurring while resizing a bar chart for 10 seconds, once with the unchanged dataset (7 bars) and once with the dataset included ten times (70 bars). The tests were performed using Firefox 124.0.1 on Ubuntu 22.4.4 LTS, on a Acer laptop with 16 GB RAM.
- *Provided Functionality*: The features and functionality provided by the library.

To analyze and compare the libraries, the same bar chart was created with each of the five libraries using the dataset shown in Table 4.1. The data represents the population of six Austrian cities as provided by Hernández [2023] in 2023. The five charts are hosted on a dedicated web site [Egger 2023].

4.1.1 Chart.js

Chart.js is an open-source library for creating charts via the HTML canvas element (Canvas2D) [Chart.js 2023]. Figure 4.1 shows the example bar chart created with Chart.js. The following points sum up the most important information about Chart.js v4.3.3:

- *Render Method*: Canvas2D.
- *Available Charts*: Eight different chart types: area charts, bar charts, bubble charts, doughnut and pie charts, line charts, polar area charts, radar charts, and scatterplots.
- *Licensing*: Open-source. Anyone can make contributions via pull requests.
- *Bundle Size*: Unpacked 4.92 MB, minified gzipped 66.6 kB.
- *Popularity*: 62 k GitHub stars.
- *Performance*: The performance test resulted in a minimum of 57 fps with 7 bars and 53 fps with 70 bars.
- *Provided Functionality*: The reason for Chart.js' popularity is its well thought-out default configuration, making it possible to create responsive charts with smooth animations with just a few lines of code. The team behind the library provides solid documentation for all chart types, with extensive examples too. Chart.js offers good basic interactiveness like smooth animations and tooltips. However, elements cannot be directly customized, since are only drawn to a HTML5 canvas element.



(a) Narrow. **(b)** Wide.

Figure 4.1: Chart.js: The example bar chart created with Chart.js on narrow and wide screens. [Images created with Chart.js [Chart.js 2023] by the author of this survey.]

4.1.2 Plotly.js

Plotly.js renders its content inside an SVG element [Plotly 2023]. It is built on top of D3 and provides a broad selection of available visualization types. Figure 4.2 shows the example bar chart created with Plotly.js. The following points sum up the most important information about Plotly.js v2.26.2:

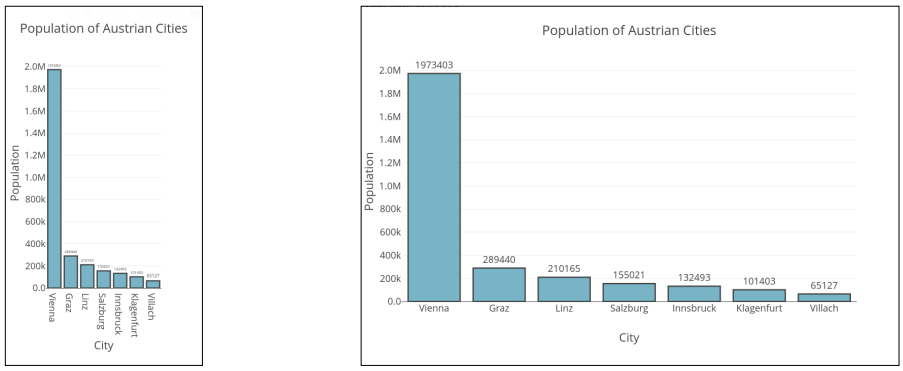
- *Render Method*: SVG-DOM.
- *Available Charts*: Over 40 different chart types, including 3D charts and geographical maps.
- *Licensing*: Open-source. Anyone can make contributions via pull requests.
- *Bundle Size*: Unpacked 60.4 MB, minified gzipped 1.1 MB.
- *Popularity*: 15.9 k GitHub stars.
- *Performance*: The performance test resulted in a minimum of 54 fps with 7 bars and 20 fps with 70 bars.
- *Provided Functionality*: The configuration of charts is not as straightforward as with Chart.js. The default settings do not lead to such smooth and pleasing results, making it harder for developers to find the perfect configuration. Activating the responsive mode enables the automatic resizing of a visualization, but the adjustments to the visualization do not look smooth by default. The chart creator can adjust hover, click, zoom, and pan events. Plotly.js provides a toolbar for its charts, containing controls for the end user to zoom, pan, and download the chart.

4.1.3 Chartist

Chartist is a lightweight tool primarily used for generating basic chart types [Kunz 2017]. Before Chartist was created, there already existed many different charting libraries. The motivation behind creating Chartist was to solve all the little problems of the existing libraries and to create an even better one [Kunz 2017]. Figure 4.3 shows the example bar chart created with Chartist.

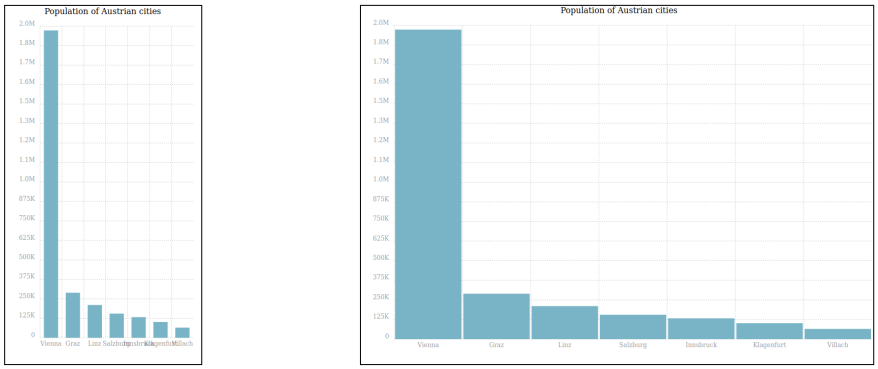
The following points sum up the most important information about Chartist v1.3.0:

- *Render Method*: SVG-DOM.
- *Available Charts*: Bar charts, line charts, and pie charts, as well as subtypes and variations of these basic chart types.
- *Licensing*: Open-source. Anyone can make contributions via pull requests.



(a) Narrow. **(b)** Wide.

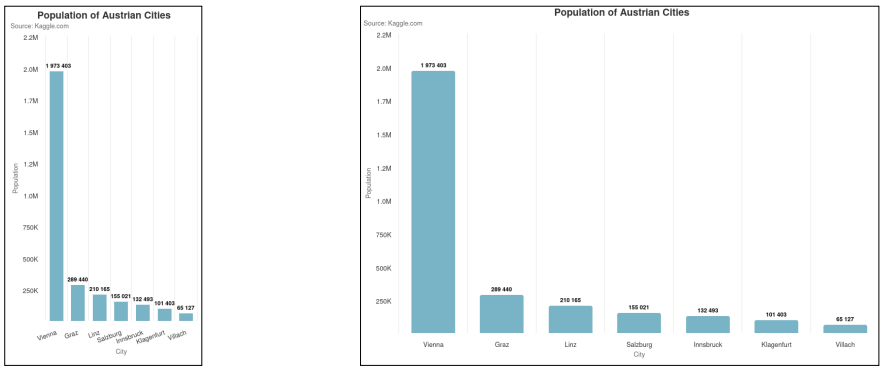
Figure 4.2: Plotly.js: The example bar chart created with Plotly.js on narrow and wide screens. [Images created with Plotly.js [Plotly 2023] by the author of this survey.]



(a) Narrow. **(b)** Wide.

Figure 4.3: Chartist: The example bar chart created with Chartist on narrow and wide screens. [Images created with Chartist [Kunz 2017] by the author of this survey.]

- *Bundle size*: Unpacked 1.35 MB, minified gzipped 11.7 kB.
- *Popularity*: 13.3 k GitHub stars.
- *Performance*: The performance test resulted in a minimum of 57 fps with 7 bars and 49 fps with 70 bars.
- *Provided Functionality*: When creating a chart with Chartist, one will find two popular versions of the library. A GitHub Pages homepage [Kunz 2017] of version v0.11 is returned as the first result when browsing for *Chartist*. However, this version is deprecated, the documentation at [Kunz 2023] shows information about the current version v1.3.0. When trying to experiment with different chart configurations, one can see that the docs are only partly complete. Using TypeScript and exploring directly how types are built up is perhaps a faster way to search for specific chart options. Since Chartist primarily focuses on the visualization and rendering of charts, it is up to the developer which event handling and interaction possibilities are implemented. The Chartist team recommends using the default styles and overriding them as necessary.



(a) Narrow. **(b)** Wide.

Figure 4.4: Highcharts: The example bar chart created with Highcharts on narrow and wide screens. [Images created with Highcharts [Highsoft 2023] by the author of this survey.]

4.1.4 Highcharts

Highcharts is a commercial charting library with source code available, which offers multiple different licenses [Highsoft 2023]. For non-commercial and educational purposes, the library can be used for free. Figure 4.4 shows the example bar chart created with Highcharts.

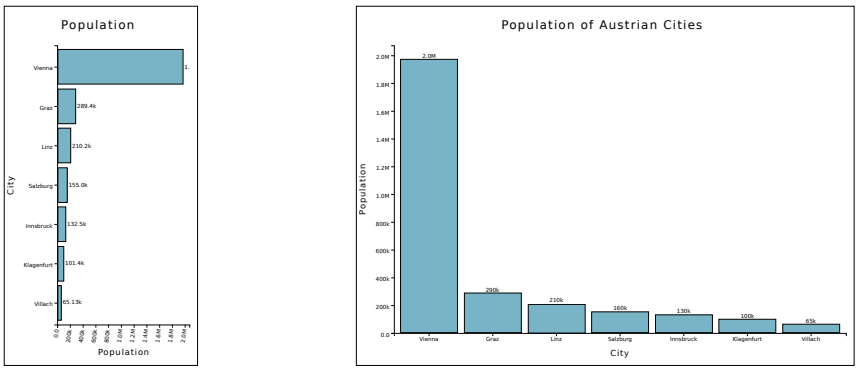
The following points sum up the most important information about Highcharts v11.1.0:

- *Render Method*: SVG-DOM.
- *Available Charts*: Over 50 different chart types, and additional modules for creating dashboards, maps, stocks and Gantt charts.
- *Licensing*: Commercial product with a variety of licenses for commercial use. Free for non-commercial and educational use. The source code is available. Developers can contribute to the project via pull requests.
- *Bundle Size*: Unpacked 116 MB, minified gzipped 96.5 kB.
- *Popularity*: 11.5 k github stars.
- *Performance*: The performance test resulted in a minimum of 56 fps with 7 bars and 52 fps with 70 bars.
- *Provided Functionality*: The setup needed for creating a first chart with Highcharts is somewhat cumbersome, because the demonstration examples do not include the components which must be imported from the library. Also, the declaration file is very large (over 23 MB). This can lead to limitations when making use of TypeScript's intellisense system. Highcharts provides a broad selection of possibilities when it comes to user interactions. By default, elements appear via animations and are highlighted on hovering. Tooltips are included by default too. However, the transition between different widths and heights of a visualization is not smooth.

4.1.5 RespVis

RespVis is an open-source visualization library and built on top of D3 [Egger and Oberrauner 2023a; Andrews et al. 2023]. It uses a novel layouting approach to make it possible to position SVG elements via CSS. Figure 4.5 shows the example bar chart created with RespVis. he following points sum up the most important information about RespVis v1.0.0:

- *Render Method*: SVG-DOM.



(a) Narrow. **(b)** Wide.

Figure 4.5: RespVis: The example bar chart created with RespVis on narrow and wide screens. [Images created with RespVis [Egger and Oberrauner 2023b] by the author of this survey.]

- *Available Charts*: Bar charts, line charts, and scatterplots, as well as variations and subtypes thereof.
- *Licensing*: Open-source. Anyone can make contributions via pull requests.
- *Bundle Size*: Unpacked 377 kB, minified gzipped 57 kB. RespVis is currently not available in the npm package registry, therefore the size was directly measured by cloning and building the source code of the library.
- *Popularity*: RespVis is currently not widely used.
- *Performance*: The performance test resulted in a minimum of 47 fps with 7 bars and 7 fps with 70 bars.
- *Provided Functionality*: RespVis currently does not provide extensive documentation on how to create visualizations, but has showcase examples included in the source code of the repository [Egger and Oberrauner 2023a]. The library lets users create charts by calling the corresponding render methods in JavaScript. The rendered charts can then be styled via CSS and responsive transformations can be specified either via CSS or by assigning event listeners and changing the data passed to the visualization.

Some further improvements which could be made to RespVis include:

- *Automatic Responsive Transformations*: Applying responsive transformations automatically, without requiring the user to specify them, is one of the reasons responsive libraries are popular. Many users simply want a working solution for all screen sizes and do not care about when exactly which transformation is applied. For example, automatic tick reduction when the chart shrinks is very practical.
- *Smooth Animations*: Some chart libraries support appealing animations on the first rendering of a chart. This would fit well in RespVis.
- *Grid Lines*: Toggling the visibility of grid lines would be a nice feature for users of RespVis.
- *Developer Experience*: A well-documented, consistent API with extensive examples is crucial for a library to be adopted by developers.
- *Applying Bundle Splitting*: Chart.js has a separate guide to how the tree shaking process can be optimized for the library. A similar approach would make sense for RespVis.
- *Automatic Text Wrapping*: Highcharts is the only library which creates titles that wrap automatically

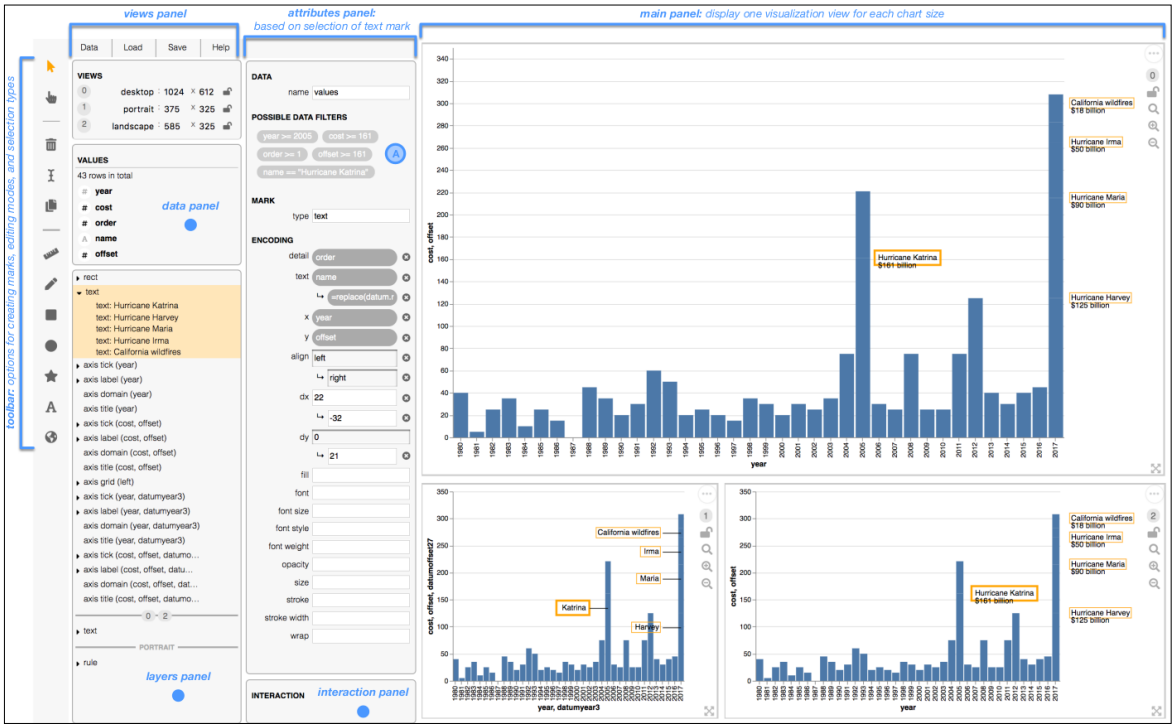


Figure 4.6: Hoffswell et al: The tool supports the design of multiple size variants of a chart in parallel. [Image extracted from Hoffswell et al. [2020]. Copyright © 2020 Association of Computing Machinery and used under § 42f.(1) of Austrian copyright law.]

if too little space is available. This is a useful feature and should be included in RespVis too, as it reduces the need to specify different title text for different breakpoints.

4.2 Responsive Visualization Systems

This section discusses recent tools which support the creation of multiple versions of a visualization to fit different space requirements.

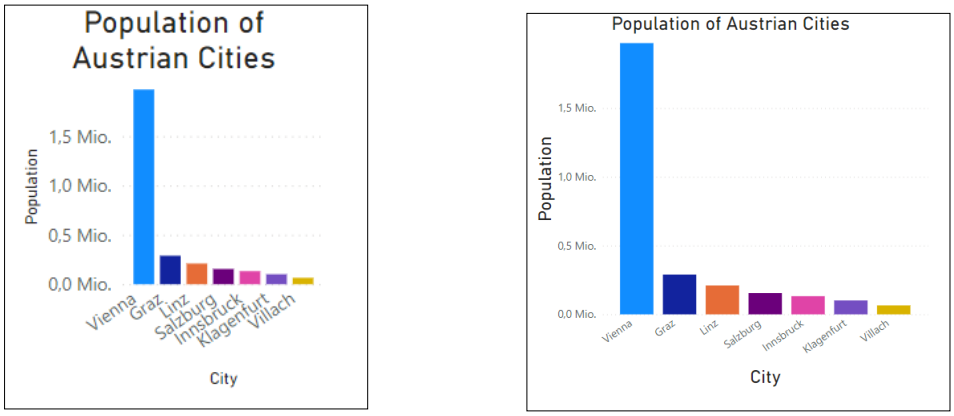
4.2.1 Hoffswell et al Visualization System

Hoffswell et al. [2020] implemented a tool for designing multiple views of a visualization in parallel, for instance, a landscape desktop version, and both portrait and landscape mobile versions. Constructed visualizations are specified in Vega-Lite [Heer 2024], a high-level grammar for interactive visualizations. Modifications made to one view are propagated to the other views by default. All views are previewed at the same time to help avoid inconsistencies and keep all views up to date. Views can also be locked and therefore excluded from propagation. If only two views are unlocked one can adjust the differences between the views.

The visualization tool is not publicly available, but there is an example gallery showcasing visualizations created by the tool [Hoffswell et al. 2023]. Figure 4.6 shows the user interface of the tool during the design process with multiple views of the visualization in parallel.

4.2.2 Power BI

Power BI is a set of services provided by Microsoft [Microsoft 2024] for collecting, analyzing, transforming data from one or multiple sources and finally creating visualizations and reports which can be collaboratively shared.



(a) Narrow. **(b)** Wide.

Figure 4.7: Power BI: When scaling a visual up or down in a report, it responsively adjusts to the new space requirements. However, it remains static once published. [Images created with Power BI [Microsoft 2024] by the author of this survey.]

Gal [2017] provides a short description of the responsive features available for Power BI. In short, when working on a report via Power BI Desktop, a visualization designer can change the size of an existing visual and the visual will automatically adapt to its new width and height. This feature comes in handy when creating a visual for different devices with the visual automatically adapting to the new requirements. However, the visual then remains static when published at that particular size. Figure 4.7 shows the same bar chart adapted by Power BI Desktop to narrower and wider sizes.

4.3 Visualization Transformation Tools

This section discusses tools which automatically apply responsive transformations to existing visualizations to adapt them to different space requirements.

4.3.1 MobileVisFixer

MobileVisFixer is a tool to automatically transform SVG-based graphics into mobile-friendly ones [Wu et al. 2021]. MobileVisFixer uses a reinforcement-learning-based approach, producing a model capable of finding mobile-friendly solutions for visualization problems in a claimed 89% of cases.

Initially, 374 visualization examples from 103 different domains were collected by a web crawler. The authors then manually went through all the examples and identified five main issues with the visualizations in terms of mobile-friendliness:

- Content placed outside the viewport.
- Font size unreadable.
- Cluttered and overlapping text.
- Unnecessary white space.
- Visualization layout becomes distorted.

In the end, the model was trained to optimize the first four of the above-listed issues. Successful transformations can be seen in Figure 4.8. Unfortunately, the source code of MobileVisFixer is currently not public, which prohibits practical evaluation.

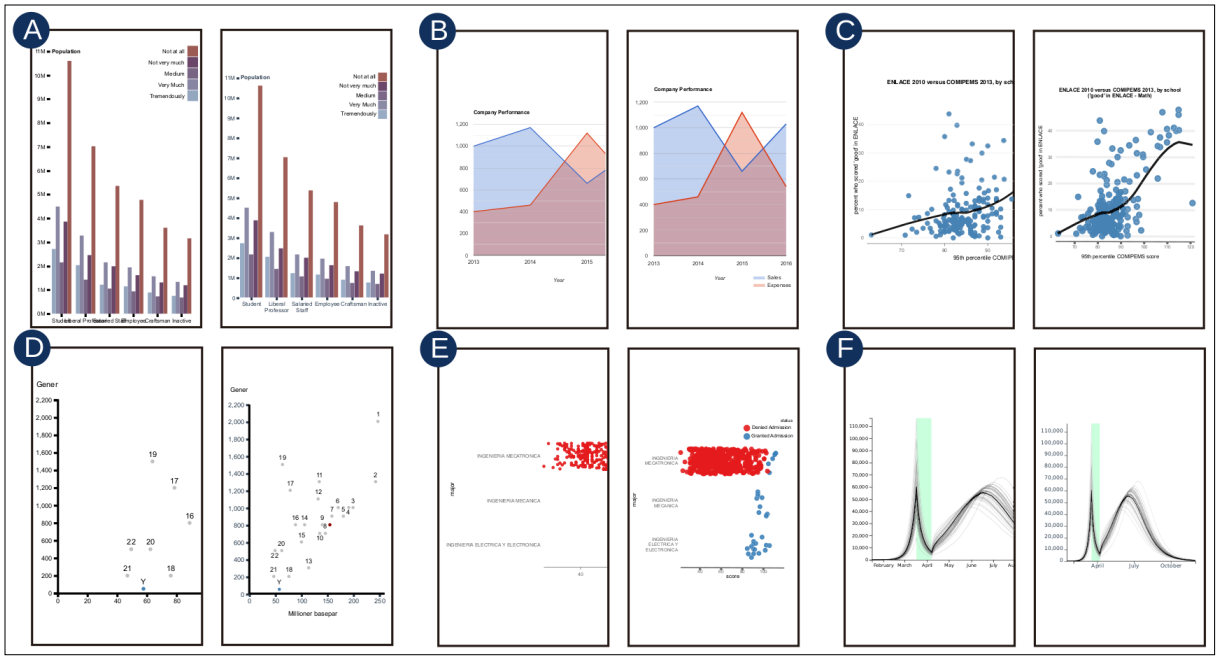


Figure 4.8: MobileVisFixer: Uses an AI model to transform a visualization to a more mobile-friendly version. [Image extracted from Wu et al. [2021]. Copyright © 2021 IEEE and used under IEEE Thesis/Dissertation Reuse Rule.]

4.3.2 Setlur and Chung Line Chart Resizer

Setlur and Chung [2021] implemented a line chart resizing algorithm to automatically adapt line charts to fit different size requirements. The creation of the tool was inspired by cartographic generalization principles, which help to remove less semantically meaningful elements and emphasize more relevant elements in the field of cartography.

The algorithm first analyzes all elements of an existing line chart to assess their semantic meaning. Then a number of spatial metrics are calculated: density, distance, area ratio, and collision between elements. The algorithm first repositions labels via jittering and eliminates semantically less important elements until the spatial constraints are satisfied. Subsequently, the line is simplified and ticks are merged to avoid congestion. Unfortunately, the source code of the line chart resize algorithm is currently not public, which prohibits practical evaluation.

According to Setlur and Chung [2021], the algorithm was implemented in HTML and JavaScript using D3. Figure 4.9 shows an example of a line chart resized to fit narrower screen sizes.

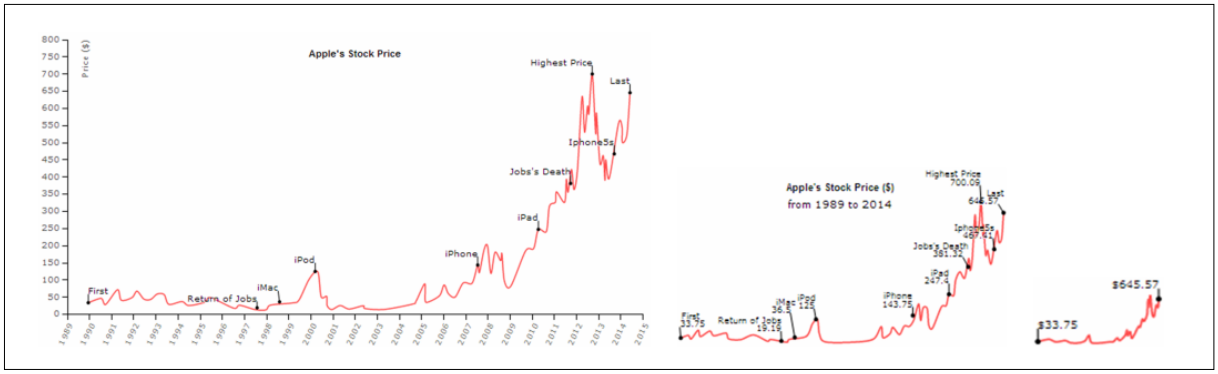


Figure 4.9: Setlur and Chung: Automatic resizing of line charts on narrower spaces. [Image extracted from Setlur and Chung [2021]. Used under the terms of a Creative Commons Attribution 4.0 International (CC BY 4.0) license.]

Chapter 5

Concluding Remarks

This survey summarizes existing work in the field of responsive visualization and its related field responsive web design. It describes a set of patterns applicable to create responsive visualizations. Existing libraries, systems, and tools which ease the creation of responsive visualizations are discussed and analyzed.

The presented patterns vary deeply in type and degree of implementation complexity and include techniques for conducting small changes as well as complete transformations of a visualization. The decision as to which pattern to use always depends on the visualization type and the resulting advantages and disadvantages of the pattern in the given scenario. In most cases, the combination of a number of patterns leads to the desired result.

Bibliography

Ander [2021]. *Germany Cars Dataset*. 2021. https://kaggle.com/datasets/ander289386/cars-germany?resource=download (cited on page 20).

Andrews, Keith [2018a]. *Responsive Data Visualisation*. 2018. https://projects.isds.tugraz.at/respvis/ (cited on pages 9, 17).

Andrews, Keith [2018b]. *Responsive Visualisation*. CHI 2018 Workshop on Data Visualization on Mobile Devices (MobileVis 2018) (Montréal, Québec, Canada). 21 Apr 2018. https://mobilevis.github.io/assets/mobilevis2018_paper_4.pdf (cited on pages 1, 10).

Andrews, Keith [2024]. *Information Visualisation Course Notes*. 08 Mar 2024. https://courses.isds.tugraz.at/ivis/ivis.pdf (cited on page 1).

Andrews, Keith, David Egger, and Peter Oberrauner [2023]. *RespVis: A D3 Extension for Responsive SVG Charts*. Proc. 27th International Conference Information Visualisation (IV 2023) (Tampere, Finland). 25 Jul 2023, pages 19–22. doi:10.1109/IV60283.2023.00014. https://ftp.isds.tugraz.at/pub/papers/andrews-iv2023-respvis.pdf (cited on pages 7, 25).

Bederson, Benjamin B. and James D. Hollan [1995]. *Pad++: A Zoomable Graphical Interface System*. Conference Companion, ACM Conference on Human Factors in Computing Systems (CHI 1995) (Denver, Colorado, USA). 07 May 1995, pages 23–24. doi:10.1145/223355.223394. https://cs.umd.edu/~bederson/images/pubs_pdfs/p23-bederson.pdf (cited on page 17).

Bundlephobia [2023]. *Bundlephobia*. 06 Dec 2023. https://bundlephobia.com/ (cited on page 22).

Chart.js [2023]. *Chart.js*. 19 Aug 2023. https://chartjs.org/ (cited on pages 22–23).

Christensson, Per [2019]. *Resolution Defiinition*. 26 Aug 2019. https://techterms.com/definition/resolution (cited on page 3).

Deveria, Alexis [2023]. *Can I Use*. 19 Aug 2023. https://caniuse.com/ (cited on page 6).

Dreher, Stefan [2023]. *Digitalisierung mit Apps: Mobile First wird zu Mobile Only.* 11 Oct 2023. https://almato.com/blog/mobile-first-zu-mobile-only/ (cited on page 5).

Egger, David [2023]. *Responsive Visualization Tools*. 19 Aug 2023. https://responsive-visualization-tools.netlify.app/ (cited on page 22).

Egger, David and Peter Oberrauner [2023a]. *Respvis Repository*. 24 Apr 2023. https://github.com/tugraz-isds/respvis (cited on pages 25–26).

Egger, David and Peter Oberrauner [2023b]. *Respvis-V2 Release Demo*. 24 Apr 2023. https://respvis.netlify.app/ (cited on pages 9–14, 16–18, 26).

Egnyte [2022]. *Data Sampling*. 19 Apr 2022. https://egnyte.com/guides/life-sciences/data-sampling (cited on page 20).

Gal, Roy [2017]. *Responsive Visualizations Coming to Power BI*. 06 Jul 2017. https://powerbi.microsoft.com/fr-be/blog/responsive-visualizations-coming-to-power-bi/ (cited on page 28).

Heer, Jeffrey [2024]. *Vega-Lite – A Grammar of Interactive Graphics*. 12 Apr 2024. https://vega.github.io/vega-lite (cited on page 27).

Hernández, Juanma [2023]. *World Cities Database*. 31 Mar 2023. https://kaggle.com/datasets/juanmah/world-cities (cited on page 22).

Highsoft [2023]. *Highcharts*. 30 Aug 2023. https://highcharts.com/ (cited on pages 15, 25).

Hoffswell, Jane, Wilmot Li, and Zhicheng Liu [2020]. *Techniques for Flexible Responsive Visualization Design*. Proc. ACM Conference on Human Factors in Computing Systems (CHI 2020) (Online). ACM, 25 Apr 2020, Paper 648, pages 1–13. doi:10.1145/3313831.3376777. https://jhoffswell.github.io/website/resources/papers/2020-ResponsiveVisualization-CHI.pdf (cited on pages 7, 27).

Hoffswell, Jane, Wilmot Li, and Zhicheng Liu [2023]. *Responsive Visualization System Example Gallery*. 30 Aug 2023. https://jhoffswell.github.io/website/resources/supplemental/responsive-supplemental/ (cited on page 27).

Horak, Tom, Wolfgang Aigner, Matthew Brehmer, Alark Joshi, and Christian Tominski [2021]. *Responsive Visualization Design for Mobile Devices*. In: *Mobile Data Visualization*. Edited by Bongshin Lee, Raimund Dachselt, Petra Isenberg, and Eun Kyoung Choe. CRC Press, 23 Dec 2021. Chapter 2, pages 33–65. ISBN 0367534711. doi:10.1201/9781003090823-2. https://imld.de/cnt/uploads/Horak2021_MobileDataVisBook_Chap02_Responsive.pdf (cited on page 3).

Infogram [2016]. *Key Figures in the History of Data Visualization*. 15 Jun 2016. https://medium.com/@Infogram/key-figures-in-the-history-of-data-visualization-30486681844c (cited on page 1).

infovis-wiki [2006]. *Zoom*. 05 Oct 2006. https://infovis-wiki.net/wiki/Zoom (cited on page 16).

infovis-wiki [2014]. *Semantic Zoom*. 10 Jul 2014. https://infovis-wiki.net/wiki/Semantic_Zoom (cited on page 17).

Juviler, Jamie [2021]. *Horizontal Scrolling in Web Design: How to Do It Well*. 14 Jun 2021. https://blog.hubspot.com/website/horizontal-scrolling (cited on page 6).

Khronos [2024]. *WebGL Overview*. 01 Mar 2024. https://khronos.org/webgl/ (cited on page 21).

Kim, Hyeok, Dominik Moritz, and Jessica Hullman [2021]. *Design Patterns and Trade-Offs in Responsive Visualization for Communication*. Computer Graphics Forum 40.3 (29 Jun 2021), pages 459–470. ISSN 1467-8659. doi:10.1111/cgf.14321. https://arxiv.org/abs/2104.07724 (cited on pages 6–7, 9).

Korduba, Yaryna, Stefan Schintler, and Andreas Steinkellner [2022]. *Responsive Data Visualization*. Information Visualisation SS 2022. Survey Paper. Graz University of Technology, Austria, 31 May 2022. 33 pages. https://courses.isds.tugraz.at/ivis/surveys/ss2022/ivis-ss2022-g2-survey-resp-vis.pdf (cited on page 7).

Kunz, Gion [2017]. *Chartist*. 08 Dec 2017. https://gionkunz.github.io/chartist-js/ (cited on pages 19, 23–24).

Kunz, Gion [2023]. *ChartistV1*. 30 Aug 2023. https://chartist.dev/ (cited on page 24).

Marcotte, Ethan [2010]. *Responsive Web Design*. 25 May 2010. https://alistapart.com/article/responsive-web-design/ (cited on pages 4–5).

Marcotte, Ethan [2011]. *Responsive Web Design*. A Book Apart, 07 Jun 2011. 143 pages. ISBN 098444257X. http://abookapart.com/products/responsive-web-design (cited on page 4).

Marcotte, Ethan [2014]. *Responsive Web Design*. 2nd Edition. A Book Apart, 02 Dec 2014. 153 pages. ISBN 1937557189. http://abookapart.com/products/responsive-web-design (cited on page 1).

Matilda, Sarah [2023]. *A Comprehensive Guide to Cluster Analysis: Applications, Best Practices and Resources*. 21 Nov 2023. https://displayr.com/understanding-cluster-analysis-a-comprehensive-guide/ (cited on page 18).

MDN [2023a]. *CSS Flexible Box Layout*. MDN Web Docs, 24 May 2023. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout (cited on page 5).

MDN [2023b]. *CSS Grid Layout*. MDN Web Docs, 15 Jun 2023. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout (cited on page 5).

MDN [2023c]. *CSS Values and Units*. MDN Web Docs, 06 Sep 2023. https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units (cited on page 5).

Microsoft [2024]. *Power BI*. 07 Mar 2024. https://powerbi.microsoft.com/ (cited on pages 27–28).

NCEI [2024]. *The Global Anomalies and Index Data*. National Centers for Environmental Information, 21 Feb 2024. https://ncei.noaa.gov/access/monitoring/global-temperature-anomalies/anomalies (cited on page 18).

PennState [2023]. *Font Size on the Web*. Pennsylvania State University, 13 Nov 2023. https://accessibility.psu.edu/fontsizehtml/ (cited on page 10).

Plotly [2023]. *Plotly.js*. 30 Aug 2023. https://plotly.com/javascript/ (cited on pages 16, 23–24).

Rabinowitz, Nick [2014]. *Responsive Data Visualization*. 25 Sep 2014. https://nrabinowitz.github.io/rdv/?scatterplot (cited on pages 17, 19).

Rendle, Robin [2019]. *Six Tips for Better Web Typography*. 27 Feb 2019. https://css-tricks.com/six-tips-for-better-web-typography/ (cited on page 4).

Sarkar, Manojit and Marc H. Brown [1992]. *Graphical Fisheye Views of Graphs*. Proc. ACM Conference on Human Factors in Computing Systems (CHI 1992) (Monterey, California, USA). 03 May 1992, pages 83–91. doi:10.1145/142750.142763. https://www.cs.montana.edu/courses/spring2005/430/pg/ft_gateway.cfm.pdf (cited on page 17).

Satori [2021]. *Data Generalization: The Specifics of Generalizing Data*. 03 Nov 2021. https://satoricyber.com/data-masking/data-generalization/#when-is-data-generalization-important (cited on page 17).

Setlur, Vidya and Haeyong Chung [2021]. *Semantic Resizing of Charts Through Generalization: A Case Study with Line Charts*. Proc. IEEE Visualization Conference (Vis 2021) (Online). 24 Oct 2021, pages 46–50. doi:10.1109/VIS49827.2021.9623306. https://vidyasetlur.com/recent-papers (cited on pages 29–30).

Shadeed, Ahmad [2023]. *The Guide To Responsive Design In 2023 and Beyond*. 01 Feb 2023. https://ishadeed.com/article/responsive-design/ (cited on page 5).

Soueidan, Sara [2017]. *Auto-Sizing Columns in CSS Grid: 'auto-fill' vs 'auto-fit'*. 29 Dec 2017. https://css-tricks.com/auto-sizing-columns-css-grid-auto-fill-vs-auto-fit/ (cited on page 6).

Statista [2023]. *Percentage of Mobile Device Website Traffic Worldwide*. 19 Aug 2023. https://statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/ (cited on page 1).

Swensen, Matthew [2021]. *Using SVG vs. Canvas: A Short Guide*. 10 May 2021. https://blog.logrocket.com/svg-vs-canvas/ (cited on page 21).

W3C [2024]. *WebGPU*. 05 Feb 2024. https://w3.org/TR/webgpu/ (cited on page 21).

W3Schools [2023]. *Responsive Web Design - The Viewport*. 19 Aug 2023. https://w3schools.com/css/css_rwd_viewport.asp (cited on page 4).

WCAG [2023]. *How to Meet WCAG (Quick Reference)*. Web Content Accessibility Guidelines, 13 Nov 2023. https://w3.org/WAI/WCAG22/quickref/ (cited on page 10).

Wroblewski, Luke [2011]. *Mobile First*. A Book Apart, Oct 2011. 130 pages. ISBN 9781937557027. https://abookapart.com/products/mobile-first (cited on page 5).

Wu, Aoyu, Wai Tong, Tim Dwyer, Bongshin Lee, Petra Isenberg, and Huamin Qu [2021]. *MobileVisFixer: Tailoring Web Visualizations for Mobile Phones Leveraging an Explainable Reinforcement Learning Framework*. IEEE Transactions on Visualization and Computer Graphics 27.2 (Feb 2021), pages 464–474. doi:10.1109/TVCG.2020.3030423. https://hal.inria.fr/hal-03001709/ (cited on pages 28–29).

Zanini, Antonello [2023]. *Data Aggregation – Definition, Use Cases, and Challenges*. 20 Nov 2023. https://brightdata.com/blog/web-data/data-aggregation (cited on page 18).