d3.js and its potential in data visualization

Creating a diagram showcase using ukrainian refugee data

Luis Rothenhäusler

20202459

Bachelorarbeit

Fachbereich Informatik
und Medien
Technische Hochschule Brandenburg

Betreuer: Prof. Julia Schnitzer

2. Betreuer: Prof. Alexander Peterhänsel

Brandenburg, den 24.05.2022

Bearbeitungszeit: 07.07.2022 - 01.09.2022

Brandenburg, den 15.08.2022

Ich, Luis Rothenhäusler, Student im Studiengang Informatik der Technischen Hochschule Brandenburg, versichere an Eides statt, dass die vorliegende Abschlussarbeit selbstständig verfasst und nicht mit anderen als den angegebenen Hilfsmitteln erstellt wurde.

Sie wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Luis Rothenhäusler

Abstract - German

Englische Arbeiten brauchen eine Zusammenfassung auf Deutsch. Mal abgesehen davon, dass wenn die Zusammenfassung interessant ist man ohne English eh nicht weiter kommt...

Contents

1. Introduction

Describe the background of the thesis, why it is important, what do we want to achieve. Why is this thesis (Motivation)? What do we want to do? What is the status quo? What benefits will result from this thesis? Something about the importance of infographics and comprehendible data.

The postmodern world produces huge amounts of data every second. Analyzing this data can lead to better-informed decision-making in every sector. Yet the wast amounts of gathered data is often hard to comprehend with the human mind. Data visualization is about finding ways to represent this data in visually appealing and easily comprehendible ways. Doing this quickly and always up to date can be crucial. Therefor, while it is possible to create data visualizations manually, it is common to use computer tools to help in their creation. There are many tools available to help with the creation of infographics. Some of the data visualization tools have a graphical-user-interface, like Excel, others are code based. As the requirements for a data visualization project can vary, it is often not easy to decide which tool best suits ones needs. Therefore this thesis will be a deep dive into the broad possibilities of one of these tools, the 'd3.js'(D3) library for JavaScript. Whilst there is a lot of information and examples on how to use D3, the available information makes use of a variety of code styles and different versions of D3. This makes it hard to properly evaluate the possibilities of D3 as a data visualization tool. Yet knowing when to use which tool can be greatly beneficial for all parties involved.

To evaluate D3 and its possibilities there are three main questions that will be answered in this thesis. What is the potential of D3 in data visualization? What are the advantages and disadvantages of using D3? When is it reasonable to use D3? To be able to evaluate these questions a showcase of a several different diagrams is created.

2. Basics

Everything necessary to understand the implementation as well as anything which is done beforehand, will come up here

In the following, all concepts, technologies and required backgrounds for understanding this thesis are explained. Firstly data and data types are described. Secondly diagrams and how they are structured are described. Lastly D3 as a tool to create diagrams is described.

2.1 Data

Well talk about data a bit. Where does it come from? How is it structured? What kind of attributes? What even are attributes?

Since ancient times, humans have recorded data. Recording the ins and outs of available resources was one of the driving factors behind the conceptualization of writing. (TODO: Check sources of the beer brewing video series) With the introduction of computers the amounts of gathered data have grown drastically. Nowadays vast amounts of data are gathered across all aspects of life.

The vast amounts of data gathered in databases are often hard to comprehend and evaluate with the human mind. They are also unwieldy to present them in the often limited space of articles, dashboards or other informative purposes. Therefore data visualization (Figure 2.1) is used to turn these datasets, collections of data-points, into diagrams.

Data is commonly preprocessed before turning it into diagrams. Depending on the dataset and the desired result, this can mean different things. One might want to remove excessive information from the dataset, which is not necessary for the representation. On the other hand, additional data can be added by evaluating the existing data-points. These could for example be the median of values or grouping of certain value ranges. It is important to note, that this preprocessing can happen with specific intentions in mind. While it is only supposed to make the representations easier and more concrete, it can

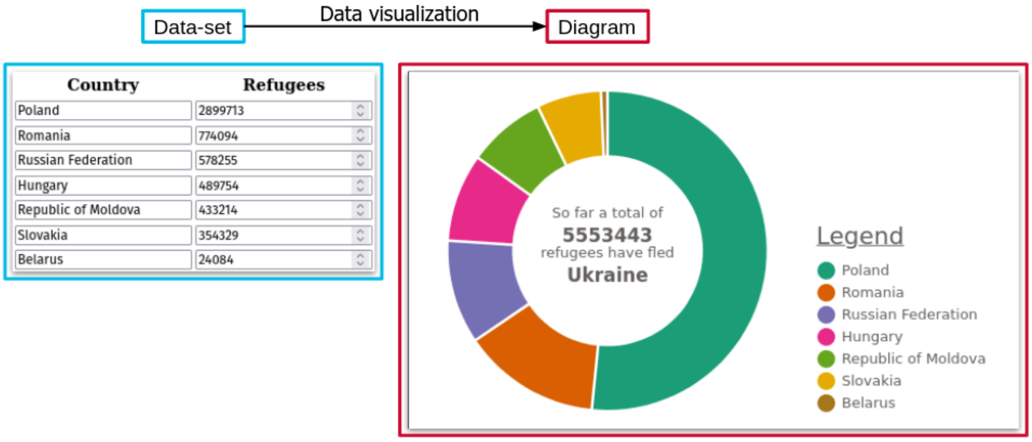

Figure 2.1: Where data visualization comes into play.

be abused to make data align with the desired results or to create a certain emphasis. This thesis is not too concerned with this, as the possibilities of D3 have nothing to do with the correctness of the chosen data.

Even though data comes from a huge variety of sources and can express a plethora of things, there are only four different types of data. They are split into two categories. Categorical and numerical data. Each category has two subtypes. In the following each of the types of data will be explained.

2.1.1 Categorical

What is categorical data? Nominal and ordinal data

Categorical or qualitative data is information collected in groups. It is often of descriptive nature. Whilst the values can be represented in numbers, they do not allow for arithmetic operations. There are two types of categorical data. Nominal and ordinal data.

Nominal data is mostly descriptive in nature. They are independent and have no inherited order. Examples are 'Country of origin', 'Color of paint', 'Brand of car'.

Ordinal data is also descriptive, yet the data does have a internal order. For example different dates each describe a day, but one day also comes after another. Grades also have an internal order, as one grade is better then another. Whilst ordinal data has an ordering, the order is not necessarily equidistant.

2.1.2 Numeric

What is numeric data? Continuous and Discrete

Numeric or quantitative data is all data expressed in numbers, where numbers do not represent categories. It allows for arithmetical operations and can be split into discrete and continuous data.

Discrete data can only take certain defined values. This usually means whole numbers to represent things that can not be split up further. Like the 'Number of Refugees' or 'Tickets sold'. Discrete data is countable.

Continuous data can be measured. It can have any real number as value. Therefore fractions are possible as well. For example when measuring the temperature, or the length or weight of an object.

2.2 Diagrams

What diagrams exists? Which are the most common? What possibilities do they offer for encoding data? Which considerations for readability? Why do some diagrams not make as much sense? Which considerations where made for fulfilling the showcase requirements?

We constantly come across the results of data visualization in everyday life. They can be commonly found across all kinds of reports, information campaigns or as part of user-interfaces in machinery or control systems. Yet the selection of which diagram should be used to visualize which data-set is not trivial. Mostly there are several possible diagram choices for the given data. Furthermore there are a plethora of diagrams already in use and anyone can create totally new diagrams to suit their needs. Yet there are diagrams which are used more commonly. These include bar and pie-charts, scatter plots and heat-maps (TODO: Find source (lel)).

Whilst there are countless types of diagrams, all diagrams use a combination of marks and channels to encode data. Marks are used for entries in the diagram. Channels describe the way specific marks encode data. The three possible marks are points, lines and areas. Each mark should use at least one channel to encode data. Otherwise it does not convey any information. The most commonly used channels are position, size, color and texture. The position in 2D can be split into the x and y positions. The color is split into hue and luminescence. For example looking at fig. 2.2 we can see lines being used as marks for each entry. It might seem like we are using areas, but the thickness of the line only serves visual understanding. The lines also use three

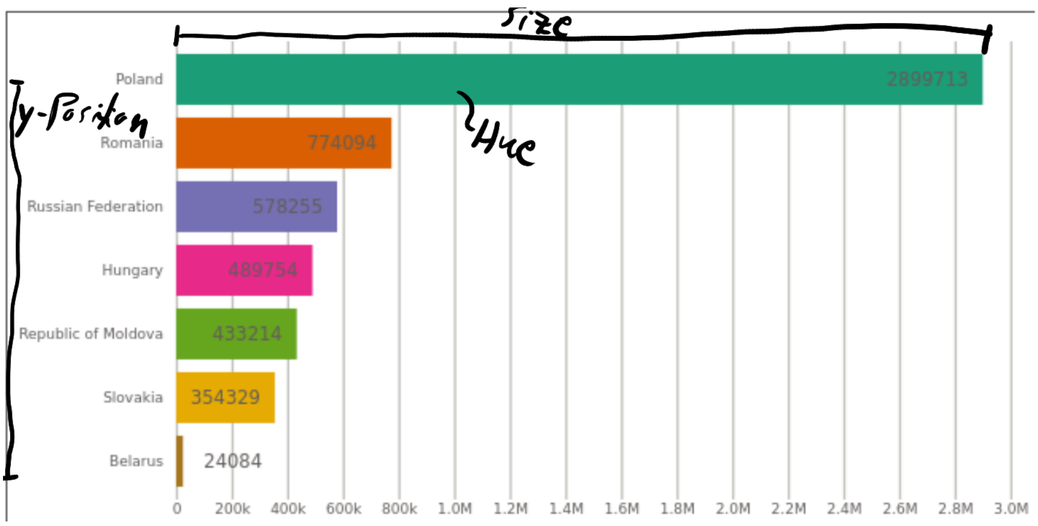

Figure 2.2: This is a bar-chart with used channels shown.(TODO: which needs a frame..? Also draw in marks and channels)

channels to encode data. The y-position is used to represent the categorical data of which country. The hue of the bar encodes the same data. This is a big redundant, as the country is already encoded. Yet the hue makes it easy to follow along when data is changing and bars are shifting positions. The size, in this case length, of the bar encodes the discrete data of how many refugees have crossed into the country. In fig. 2.2 we see areas used as marks. Just like in the previous example the hue encodes the country and the size encodes the refugee count.

All marks can be used with all channels. But not all data types should be represented by all channels. For example nominal data should not be encoded using the size channel. The different sizes would lead to a perceived order, which does not exist in nominal data. As the channels all differ in their appearance they are also not equally good in adequately representing the data types. Therefore it is important to consider which channels are chosen to represent the given data types. Therefore the selection of marks and channels should be considered carefully. If chosen poorly it can lead to undermine the purpose of the diagram, easily presenting data to a viewer. According to a study by Jock Mackinlay from 1986, the position channels can always be considered the strongest channels, no matter which marks are combined with it[1].

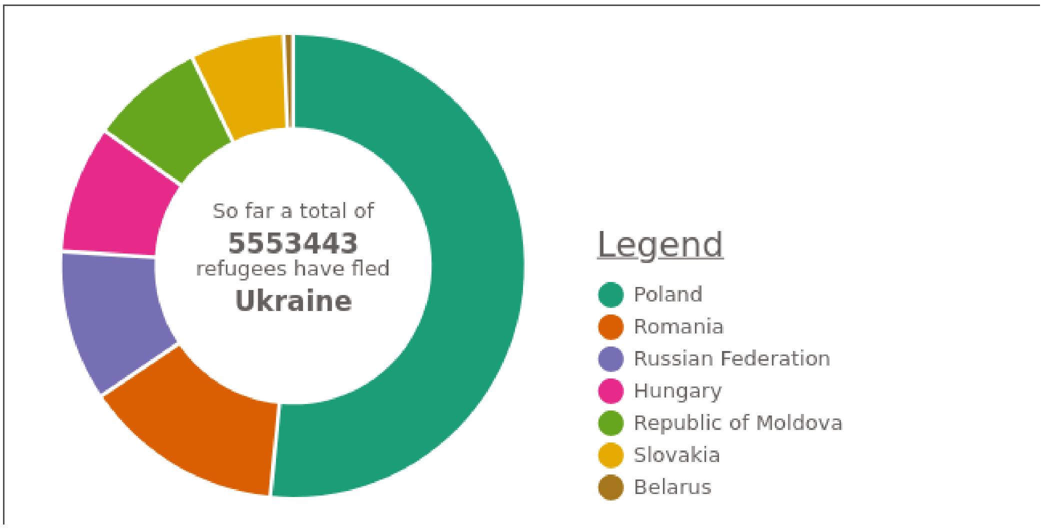

Figure 2.3: This is a donut-chart (TODO: which needs a frame..? Also draw in marks and channels)

2.3 D3.js

This is all about d3. What is it? Where does it come from? What is it used for? Who uses it? Why should it be used? How does it work? Enter, update and exit pattern. Something about the modular structure of D3 as well. Might be worth mentioning "observables" as well.

"D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS."[2]. The name D3 is short for data-driven documents. The D3 library was originally created by Mike Bostock and is published under the BSD-3-Clause open-source license. It is about 500kb in size. It does not require a specific framework and can therefore be easily integrated into all kinds of web based projects. Whilst D3 is not limited to using svg, the visualization created using D3 mostly rely on svg elements for their implementation.

D3 is not a high-level API for creating out of the box visualizations. Instead it is a library which aims at making the tedious parts of DOM manipulation easier. It also provides some helper functions, scales, to decrease the amount of mathematical equations needed to convert from the data extends to the necessary coordinates in the desired visualization.

TODO: merge the section above and below to one longer introduction

General functioning of D3.

"D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document."[2].

There are three main concepts that make up the core of D3. Selections, data joins and the general update pattern. All three of these concepts are working closely together. Whilst selections can be used without data joins and the general update pattern, these two aspects both rely on selections. Data joins can also be used without explicitly using the general update pattern. Usually all three of these concepts are used consecutively. First a selection is created. This selection is provided with a data join. Finally the behaviors for the general update pattern are defined for this data join. In the following all three of the core concepts of D3, as well as scales and D3's modularity are explained.

2.3.1 Selections

What are they? Why are they useful?

All operations in D3 run on an arbitrary collection of nodes. These collections of nodes are called selections. There are two functions in D3 to create a new selection: `d3.select("selector")` and `d3.selectAll("selector")`. Both functions require a selector for identifying the appropriate elements. The selectors are defined in the W3C Selectors API[3] and function like CSS selectors. Whilst select only selects a single element, the first element matching the selector, selectAll selects all elements which match the selector. It is important to note, that select also propagates the existing information of this node, whilst selectAll does not. Selections can also be extended or shrunken by adding or removing nodes, or by combining multiple selections. It is possible to directly access DOM elements through the selections. The respective DOM elements are linked in the nodes which make up the selection. But usually this is not required, as there are predefined functions for modifying the nodes properties. This includes the modification of attributes and styles, as well as event handling.

2.3.2 Data Joins

What are they and why are they important?

Data joins are a key feature of D3. They link up a specific data-point to a specific DOM element. To create a data-join, one has to call the `.data(dataset)` function on a selection. It takes a dataset, an array of data-points, as parameter. This will bind the data-points to the nodes in the selection. This is achieved by using an identifier function. The default identifier function returns the index of the data-point in the dataset. When we want to create diagrams which can respond to data changes over time, this is not a reliable identification. When data-points are removed or added

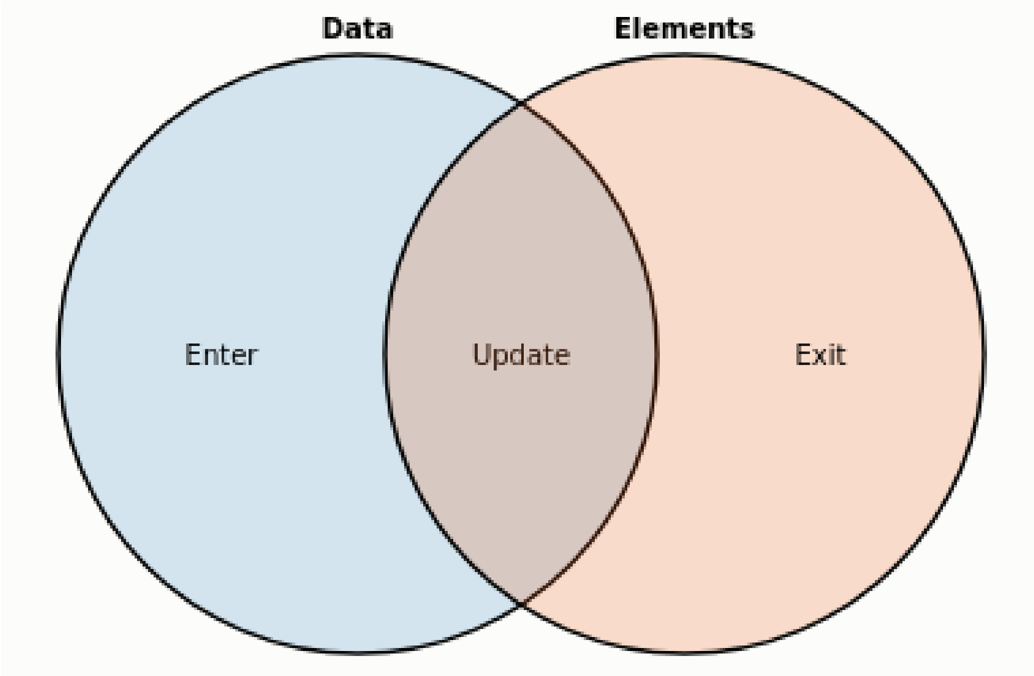

Figure 2.4: A visual representation of the make up of the general update pattern[4]

in arbitrary locations, the index will not match the elements it previously did. Therefore we can specific a custom identifier function. This is passed as the second parameter of the data function, will be called for each data-point and has to return some value which will be used as an id.

When creating a data join, it can be that the number of data-points does not match up with the number of elements to represent them. Therefore the data-joins provide an empty placeholder node to all data-points which are not matched up with an element. What happens to the placeholders is defined in the general update pattern.

2.3.3 General Update Pattern

What is it? What can it do? Describe data joins and dom element links.

The general update pattern is another core concept of D3. Every time a data join is created or updated, it comes into play. The general update pattern differentiates between three different cases. For each of these cases a sub-selection is created. For each of these three selections the behavior can be defined. The first selection is the enter selection. All data-points which have been matched up with a placeholder node while creating the data-join are in here. In the behavior for the enter selection, usually a corresponding element is created as the first step.

All the elements which are already linked to a data-point using the identifier function, make up the update selection. The last selection, the exit selection, is made up of all the elements for which the corresponding data-point has been removed. The behavior of the exit selection is by default defined to remove the respective elements.

When the goal is to create only static diagrams, which are only initially created from data, it is enough to define the behavior for the enter selection, as all data-points will be matched up with a placeholder when creating the data-join. Here the identifier function is also not important, as the created element will not need to change over time and therefore do not need to be able to be appropriately selected again as data changes. If diagrams should be able to react to data changes and update their appearance, like in this thesis, it is also important to define the update behavior as well as a proper identifier function, so elements are always matched with the same data-points. The exit behavior can be defined if a more visually pleasing removal of elements is desired, like fading out before deleting.

2.3.4 Scales

What are scales? What do they do? Why and when are they used? How do they look like?

Scales are a way to convert between two data-spaces. Some scales can even convert between two data-types. Scales can be found in many places. For example converting percentages of correct answers in a test, continuous data, to the appropriate grade, ordinal data. Or the scale factor of maps and model-kits.

As most diagrams created with D3 are created as SVG, the scales provided by D3 are mostly used to convert from the data-space to the coordinate space in which elements should be drawn. All scales require a domain and a range. The domain describe the input values, the range where they should map to. Some types of scales also allow to be used in reverse.

2.3.5 Modules

The way D3 is split up into modules, the core package and what kind of extensions are there.

D3 provides the most used, general functionalities in the core library. Yet there are many more modules which can be added, which add functionalities for more specific use-cases.

3. Implementation

How are the chosen diagrams implemented? Which D3 modules have been used? How was the implementation done?

In the following sections the process of creating the showcase and the diagram are described. There are several parts to this. At first the data-sets, which should be represented, are chosen. In most real world usages, this is already given. Afterwards the possible diagrams are considered and chosen. Their implementation and usages of D3 are described. Finally the showcase bringing all the diagrams together is described.

3.1 Datasets

What are our datasets about? Where do they come from?

In this thesis, two data-sets are used. They are both from UNHCR[5]. The first data-set contains information about the total number of refugees per country[6]. The second dataset is about the total cumulative total amount of refugees per day[7]. Both data-sets are in the JSON format. As different data-types allow for different representations and require varying parts of D3, the data-sets have been specifically chosen to cover all data-types.

It would be possible to use both these data-sets as is. Yet the vast amounts of filler data, which does not contain any valuable information, makes data accesses unnecessarily complicated and hard to follow along and understand the diagrams implementations. Therefore both data-sets are pre-processed. In both cases the data-sets are read and the valuable information extracted and saved in the CSV format. Both of the newly created CSV files have two columns and one header row. The resulting CSV of the refugees per country dataset, contains the two columns of country and refugees. The other resulting CSV contains a column for the date and one for the cumulative refugees.

(The data-set about the refugees per country can also easily be converted into using percentages. After adding up the total amount of refugees from each data-point, one can convert the absolute number of refugees into percent.)

The two chosen data-sets already cover most of the data-types. Both dataset contain two attributes per data-point. The country is a categorical attribute. The number of refugees is discrete. When converting this dataset into using percentages, the percentage of refugees becomes a continuous attribute. In the refugees per date data-set, the amount of refugees is still a discrete attribute. The date itself is an ordinal attribute though. As one day clearly comes before and after other days.

Choosing data-sets which cover all types of data-types was an important consideration. Different data-types can have different ways of representation, as well as different ways of implementation on the programming side of things.

3.2 Diagrams

describe all the diagrams and why they are special and what makes them tick. Why have they been chosen?

The following section is all about the selection and implementation of each diagram. Whilst all diagrams are presented in one showcase, each diagrams is implemented to work standalone. This makes the comparison between diagrams, as well as evaluating the effort needed to create them easier. It also allows for easier adaptation if a one is to use one of the diagrams as a template. Therefore all diagrams follow the same pattern. As each diagram is independent, they all consist of three parts. A HTML, a CSS an a JavaScript file. The HTML loads the D3 library in the header. The body of the HTML consists of a svg tag where the diagram will be drawn, and a script tag which loads the JavaScript file. The CSS defines the general styling of the diagram which is not dependent on the input data. The main part of the implementation is done in the Javascript section.

The JavaScript file also follows a general pattern. At first there is a initialization section which is run once as the website is loaded. It is followed by a render function which is responsible for drawing and updating the diagram.

3.2.1 Initialization

Generally all things which are data independent are done during the initialization. It starts with setting some core variables. A reference to the svg tag which will be used as the container is made. It is followed by a margin definition, where the margin of our diagram content in relation to the container size is defined. The resulting values for `ourHeight` and `ourWidth`, which we will use as space to draw the diagram, are saved.

Following there are a few group elements which are added to the svg tag. These group elements provide a general hierarchy for different aspects of the diagram. For example the bar chart has separate groups for the axes and the content, while the circle diagram has groups for the background legend and the content. Having a proper structure in place makes working with selections easier as well as helping with human readability of the svg's content. This general hierarchy is only created to a level which is independent of the provided data.

Finally if there data independent scales, they are defined here. A common example here is a color scale for discrete values. It is not important to already know the specific input values, to be able to create a list of colors which is used by the scale. When queried, it will then return a new color from the list, for each new query value. It is important to note that when the color list runs out of new colors, it restarts at the beginning of the list.

If there are any static elements, they are also defined in the initialization. For example the tooltip used by the tree-map or the center text fields in the donut-chart. They are already created here, so they can be filled with appropriate data later.

3.2.2 Render

Following the initialization section is the render function. The render function is called once in the beginning and every time the provided data changes. The render function covers all data dependent tasks. If there are helper functions or constants required by the render function they are defined first. Following this all the data dependent scales are defined.

Scales

The data dependent scales in this thesis are mostly used to acquire the coordinate position and sizing of elements in the diagrams. The bar-chart for example defines two scales. A linear scale to convert from a domain of the refugees [0-MaxNumberOfRefugeesInACountry] to a domain of the available space [0-ourWidth]. Converting from any given country to a y position is done using a scale band. As both these scales depend on the provided data, they are redefined with every render call.

Data Joins

After the scales are defined, the data joins are created. While some diagrams, like the bar-chart, only use a single data-join, other diagrams, like the circle-diagram, make use of several data joins. Usually this is in accordance to how many independent parts the diagram consists of. The circle diagram uses one data join for the size legend in the background, one to update the circle showing the current data and one to update the text showing the current number of total refugees.

A data join is created when binding data to a selection. This is achieved by first calling the `.data(DATA)` function of a selection. The data function creates pairs of elements and data entries. By default, these are matched through their index in the selection and data arrays. This can lead to unexpected behavior when entries are removed or inserted at the not last position. Therefore the default identifier function can be overwritten by passing a custom identifier function as the second optional parameter to the data function. A custom identifier function should return a value and is called for each element in the data array. For the refugees per country data-set in this thesis, the identifier is usually `d => {return d.country}`.

When we initially create the data join, or when data-points are added, we do not have sufficient elements in the selection to pair them with data entries. D3 will therefore create empty placeholders for these elements. To make these placeholders become a part of the DOM, we add the `.join()` after the data() call. There are two ways to use the join function. We can either pass a string which will result in adding a matching tag to the DOM. The attributes and style for each new element can then be defined by method chaining. This approach is reasonable for diagrams that do not need to react to daa changes. In this thesis we want all diagrams to implement the full extend of the general update pattern, to be able to react to changing data and use the full possibilities of D3.

General Update Pattern

How is this implemented? Where does it come into play?

When the join function is called, instead of passing a single string as parameter, three functions can be passed as parameters. These three functions correspond to the three cases of the general update pattern and describe their respective behavior. Each of the three function has one input parameter, corresponding to the respective sub-selection. In the enter function usually a element is added to the DOM. In the exit selection we remove elements again. The update function is optional, but always used in this thesis, as this is the place to update existing elements to accommodate for data changes and therefore possibly removed or newly added elements as well. All three functions run on all the elements of the appropriate sub-selection.

The enter function should add the applicable placeholder element to the svg as actual content. Therefore the first part of the enter function is usually an `.append(string)` call. The string describes the tag which will be added to the DOM. Following this the applicable styles, attributes and sub-elements are added. This can be achieved with the `.attr("attributeName", "value")`. Whilst styles can be added with the `.style("property", "value")` function, the same can be achieved by predefining styles in the css and adding applicable classes to the element. It is important to add enough attributes, that the provided selector which was used to create the selection for the data join whose behavior we are defining, can also match the newly created element when called again for an update. When positioning a new or existing element the scales are used to find the applicable coordinate space.

The update function is necessary when we want to react to data changes. It is usually similar to the enter function, in that is adjusts the positioning and sizing of the elements according to the possibly changes scales. The exit function is defined by default to simply remove the applicable elements.

All three functions can make use of animations and transitions to improve their feel.

Animations

What are they? How do they work? Why are there two kinds? What makes them tick?

Animations can improve the feel, appeal and readability of diagrams. Especially when reacting to data changes, it is easier to understand and see the changes when for example bars in a bar-chart shift to their new positions, instead of a seemingly entirely new diagram popping up out of thin air. The animations allow the viewer to keep track of the existing entries and visually follow any changes. It is also possible to see the changes of existing values by following, for example, the growth or shrinking of the length of a bar in a bar-chart. Animations can also be used when initially drawing the diagram, to guide viewer attention.

Animating elements in D3 is achieved by using transitions. Transitions are called from a selection and run on all the elements of the selection. A transition requires a duration and can also be provided with a delay and an easing function. The duration and delay are both in milliseconds. Animating numerical, color or string values is very easy with transition. It is only required to call the attribute or style with the target value and the transition will take care of the rest. This makes it very fast and easy to animate for example positioning or sizing.

```
    enter.call(enter => enter.transition(t)
    .attrTween('d', (d, index, nodes) => {
            const i = d3.interpolate(0, d.startAngle);
            const j = d3.interpolate(0, d.endAngle);

            nodes[index].previousStartAngle =
               d.startAngle;
            nodes[index].previousEndAngle = d.endAngle;

            return time => {
                d.startAngle = i(time);
                d.endAngle = j(time);
                return arc(d);
            }}))
```

Instead of using the default behaviors for numbers, string and colors or when trying to animate other values like svg paths, a tween function can be defined using `attrTween` or `styleTween`. Both tweens need to return a function which will be invoked for each frame of the animation, with a time value between 0 and 1, depending on the frame. The returned function must itself return a value, which is applied to the desired style or attribute every frame. In this thesis tweens are only specifically defined to animate svg path tags.

3.2.3 Bar Chart

How does it work? Which d3 features does it use? how do they work?

3.2.4 Pie chart

How does it work? Which d3 features does it use? how do they work?

3.2.5 Tree map

How does it work? Which d3 features does it use? how do they work?

3.2.6 Sankey

How does it work? Which d3 features does it use? how do they work?

3.2.7 Area graph

How does it work? Which d3 features does it use? how do they work?

3.2.8 Circle graph

How does it work? Which d3 features does it use? how do they work?

3.3 Showcase

How is the showcase structured? How can you get there? Why does it exist? Who might benefit? How can you reuse a part the interesting parts?

To bring all the diagrams together, a showcase has been created. It is split into two main parts. Firstly all the diagrams for refugees per country are covered. Secondly the diagrams for refugees over time are shown. As all diagrams in one section represent the same data-set, it allows for an easy visual comparison, as well as easier comparison of the code. When each diagram would show different data, it would be harder to distinguish between implementation differences which are due to the different representation and differences which are caused by accommodating different data-sets. Additionally each section has a table where the input data can be seen and modified.

3.3.1 Integration of each diagram

How is each diagram integrated? How can you access them? Where can you grab them standalone?

Each diagram is implemented to work on its own and without the showcase. Each diagram is also designed to use all the space available in its container. When loading one of the diagrams HTMLs directly, it will therefore fill the whole browser window. The showcase loads each of the diagrams into a separate IFrame tag with a consistent aspect ratio.

3.3.2 Data Updates

How can you simulate data changes? Why is this useful?

Each section of the showcase has a table which allows for data manipulation. Rows of data can be modified, added or removed here. The data changes here are not persistent and therefore do not get written in the original data csv files. When data is changed, the diagrams are provided with the updated data and adapt accordingly. For this thesis it is important to be able to modify the data, as one of the core features of D3 tested in this thesis is reacting to changes in data. This manual style of changing data is probably not so common in real world applications. Yet it is easy to replace these manual data changes to regular API calls or other automatically updating data-sources. As the source of the data changes does not matter for the functionality of D3, the manual approach chosen here is sufficient in demonstrating the possibilities of D3.

4. Discussion

Discuss pros and cons and all things that came up. Pro and cons.

4.1 Evaluation Criteria

How will we evaluate? What?

The two main questions this thesis answers are both subjective to a certain extend. The potential of D3 in data visualization will be determined and evaluated by looking at the many use-case examples which can be found online, as well as the resulting show-case of this thesis. Besides having a look at how D3 can be used, we will also look at where it is possible to use D3. As there are many scenarios in which the JavaScript environment is not required, desired or possible. This leads us into the second question of the advantages and disadvantages of D3. To evaluate this question, we will evaluate the initial learning curve, the effort to create static and dynamic diagrams, the amount and functionalities of available modules, the completeness of documentation as well as pros and cons which come with the web environment.

5. Conclusion

How well did it work? Was it worth the effort? What could be improved?

Bibliography

[1] J. Mackinlay, "Automating the design of graphical presentations of relational information," *Acm Transactions On Graphics (Tog)*, vol. 5, no. 2, pp. 110–141, 1986.

[2] M. Bostock, "Data-driven documents," accessed:31.03.2022. [Online]. Available: https://d3js.org/

[3] A. v. Kesteren and L. Hunt. [Online]. Available: https://www.w3.org/TR/selectors-api/

[4] M. Bostock, Feb 2012. [Online]. Available: https://bost.ocks.org/mike/join/

[5] UNHCR, "Operational data portal," accessed:12.05.2022. [Online]. Available: https://data2.unhcr.org/en/situations/ukraine

[6] ——, "Refugees per country," accessed:12.05.2022. [Online]. Available: https://data.unhcr.org/population/get/sublocation?widget_id=312121&sv_id=54&populatio
01-01

[7] ——, "Refugees per day," accessed:12.05.2022. [Online]. Available: https://data.unhcr.org/population/get/timeseries?widget_id=312123&sv_id=54&population_
01-01

6. Appendix

I guess this should contain all the source code. Maybe there are ways to import it automatically too?