Description, implementation and validation of a user interface for complex datasets in the social sciences

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK

BACHELOR OF STATISTICS

BACHELOR'S THESIS

19. September 2018

AUTHOR
Samuel Lippl

SUPERVISOR
Dr. Fabian Scheipl

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

Samuel Lippl, 19.09.2018

Contents

1 | Introduction

One of the main advantages of the statistical programming language R (R Core Team, 2018) lies in its conciseness. With just one line of code, it is possible to build a linear model, visualize a variable's distribution or conduct complex modifications on several datasets. This is possible because R is a domain-specific language and is therefore able to make strong assumptions – many people will need to build a linear model or read in a csv file and it is therefore sensible to create custom functions for these purposes.

An important property of this conciseness is that the code is easy to read. Two features that are especially important for this both rely on the specific domain of statistical analysis for which R was created:

- Specialized functions: as an example, `read.csv` essentially calls `read.table` with a few modified parameters. Nonetheless, the function immediately makes it clear what this line of code is supposed to achieve.
- Default values: The user does not need to specify every single parameter of a function. For instance, it is helpful that `read.table` contains the parameter `na.strings` that allows the user to specify values that encode `NA`s. However, in most cases, `NA`s are encoded by the string `NA` or a missing value[1]. By setting default values, the user only needs to think about this parameter when the file structure is out of the ordinary.

These advantages are certainly not unique to R. They are designed to minimize the expected time a user needs to spend with coding his decisions while maintaining easy reproducibility of his work. If there are more complicated tasks to undertake, a consistent interface allows the user to do that, as well.

A good example for this concept is the package `stringr` (Wickham, 2018). Functions like `str_trim` (trim whitespace) or `str_to_title` (capitalize) make special use cases easily accessible. On the other hand, `str_replace` allows more complicated operations with regular expressions using the same consistent interface.

Things, however, start to fall apart when one attempts to modify default values. These modifications are sometimes possible by setting the global options in R; however, relying on the global options makes reproducibility more difficult. On the other hand, one could write new functions to solve this problem. This is, however, more laborious than such an endeavour should be.

The need to modify default values arises, for instance, in the analysis of datasets with many variables as they occur in the social sciences. As an example, I will consider the Varieties of Democracy (V-Dem) dataset which produces indicators of democracy (Coppedge et al., 2018b, Pemstein et al. (2018)). It contains many variables on different aspects of democracy with values per country and year. If one wishes to visualize the development of this variable over time, a simple line plot often makes sense. Consider, for instance, the variable which characterizes the freedom of religion on a scale between 0 and 4 for Germany in figure 1.1.

Although there are large changes within a single year, freedom of religion is a continuous value and linear interpolation of this development within a year makes sense.

[1] The latter is only implemented in `read_delim` from the package `readr` but the advantage of default values remains valid nevertheless.

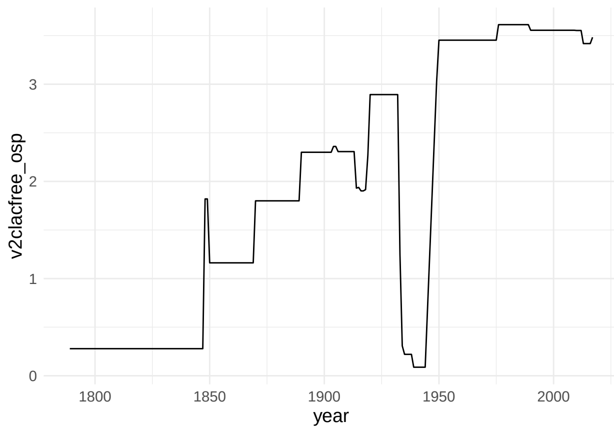

Figure 1.1: Example: Freedom of religion in Germany over time

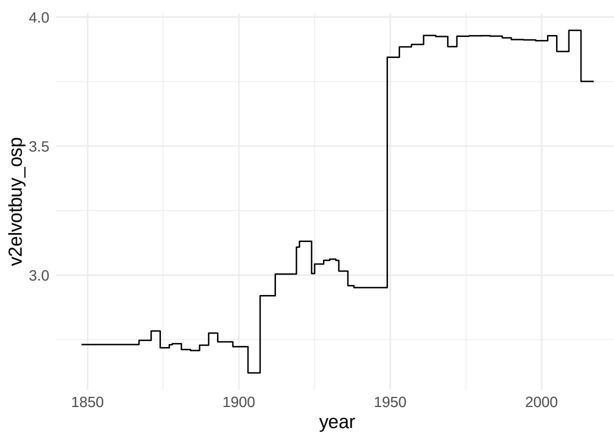

Figure 1.2: Example: Election vote buying in Germany over time

Considering the variable `v2elvotbuy_osp`, however, a line plot makes less sense. This variable captures whether there was evidence of vote buying during a national election and is therefore only present in years where there has been a national election. A step plot as depicted in figure 1.2 seems more sensible in this case as the current value would always refer to the last election.

Furthermore, the scale titles should be modified to show an interpretable variable name and the scale should in most cases be standardized to depict the entire range between 0 and 4.

In summary, there are many considerations one needs to regard in such a visualization. Therefore, the statistician needs to think about these questions whenever they want to implement such a visualization, which is time expensive and impossible to apply to interactive user interfaces. This problem is not limited to visualization; another example would be descriptive tables of a linear model or a report summarizing all covariates that have been used.

In summary, R provides powerful opportunities to outsource everyday thought processes in data analysis. However, adapting these mechanisms for application-specific thought processes is expensive and difficult. A broad framework for such an adaptation would enable researchers to think about certain decisions (like the visualization of a specific variable) once and then be done with it. Both the researcher himself and his colleagues who might not need to think about this at all would benefit from this.

In this Bachelor's Thesis, I describe such a framework, implement it as the package `tectr` in R and apply it to the V-Dem dataset. The next chapter discusses some details regarding the package construction and the dataset before we get a first look in the third chapter. The fourth chapter will present the framework and the implementation in a more specific way. Chapter five presents the application of `tectr` to the V-DEM dataset and the final chapter summarizes the thesis and discusses the next steps regarding `tectr`.

Note

When I consider a statistician as an example, I will refer to them by a singular "they" as I deem this the most inclusive. Regarding the possible objection that this is ungrammatical, I will refer to a blog post (Soanes, 2012) which discusses this question.

2 | Methodology

This chapter introduces the V-Dem dataset and discusses the methodological background of `tectr`'s construction.

2.1 V-Dem

2.1.1 Introduction to the database

The Varieties of Democracy Institute is concerned with measuring different aspects of democracy. It distinguishes between seven high-level principles: electoral, liberal, participatory, deliberative, egalitarian, majoritarian and consensual. These are measured by a variable in the interval $[0, 1]$. Each is the result of several mid-level indices in the interval $[0, 1]$ and low-level indices in the domain $[0, 4]$. The low-level indices are coded with the help of several country experts who answer a detailed questionnaire. Most questions can be answered by an ordinal scale of five alternatives. Consider, as an example, the variable "Disclosure of campaign donations":

> **Are there disclosure requirements for donations to national election campaigns?**
>
> 0: No. There are no disclosure requirements.
>
> 1: Not really. There are some, possibly partial, disclosure requirements in place but they are not observed or enforced most of the time.
>
> 2: Ambiguous. There are disclosure requirements in place, but it is unclear to what extent they are observed or enforced.
>
> 3: Mostly. The disclosure requirements may not be fully comprehensive (some donations not covered), but most existing arrangements are observed and enforced.
>
> 4: Yes. There are comprehensive requirements and they are observed and enforced almost all the time.

The answers are then analyzed for inter-coder reliability and a standardized average of the responses together with a confidence interval which contains 68 % of the probability mass is created. Lower-level indices are created from these answers which are aggregated to the mid-level and then high-level indices. An overview over the structure can be found in appendix D of the codebook (Coppedge et al., 2018c). The database contains data on 201 countries between 1789 and 2017. (Coppedge et al., 2018b, Pemstein et al. (2018))

2.1.2 `vdem.tectr`

The package `vdem.tectr`, which I have created, contains version 8 of the country-year dataset. It can be found in the supplementary material (see Appendix A)[1].

[1] Unfortunately, due to some issues with Git, the package cannot be found in its current version on Github.

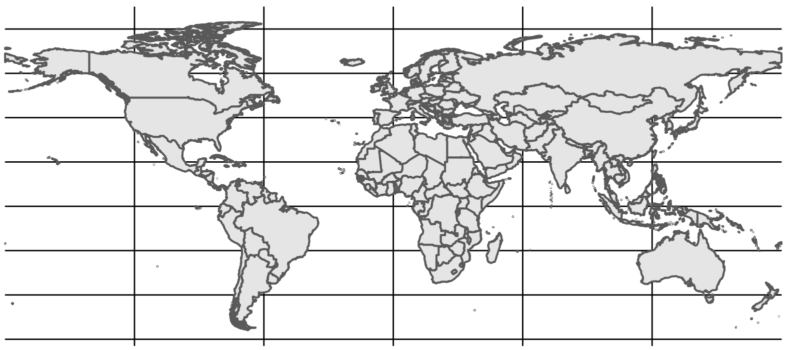

Figure 2.1: Country borders in 2017 in the V-Dem database

The package contains three datasets:

- `df_vdem`: This dataset contains all variables from the V-Dem dataset where interval variables are numerics and categorical variables are saved as factors or ordered factors where appropriate.
- `vdem_spatial`: This simple features object (Pebesma, 2018) contains the polygon shapes of the different countries for every year between 1945 and 2017. The borders in 2017 are depicted in figure 2.1. I have used the CShapes dataset (Weidmann et al., 2010, Weidmann and Gleditsch (2010)), sovereignty- and state-level maps data from Natural Earth and the details from the document on country coding units from V-Dem (Coppedge et al., 2018a). Note that the coded country borders by V-Dem do not constitute an endorsement of controversial entities such as Zanzibar. The function `vdem_geocode` makes it possible to join the shapes polygons to a data frame as long as the columns `country_name` and `year` exist.
- `vdem` which contains the variables from `df_vdem` and further metainformation (see below)

Details on these datasets and the reproducible code can be found in the folder "data-raw" in the package.

2.2 Package construction

The package has been constructed with the packages `devtools` (Wickham et al., 2018b), `roxygen2` (Wickham et al., 2018a) and `testthat` (Wickham, 2011). The Bachelor's Thesis has been written with `bookdown` (Xie, 2016, Xie (2018)).

3 | First look

This chapter is devoted to a first applied example of the package before the fourth chapter introduces the concepts of `tectr` in a more comprehensive way.

As an example, we will consider the V-Dem database, which has been introduced in the preceding chapter. We will focus on the function `fx_ggplot` as an example. It is based on the package `ggplot2` which implements the grammar of graphics in R in order to produce visualizations. (Wickham, 2016)

3.1 `fx_ggplot`: Basics

In the following, I will present different example applications of `fx_ggplot`.

At first, we will consider the electoral democracy index and its distribution. The result is depicted in figure 3.1.

```
df_vdem %>%
  select(v2x_polyarchy) %>%
  fx_ggplot(aes(x = v2x_polyarchy)) +
  my_theme # my_theme makes the text larger
```

This plot is evidently a histogram. However there are several differences to the ordinary call in figure 3.2.

```
ggplot(df_vdem, aes(x = v2x_polyarchy)) +
  geom_histogram() +
  my_theme
```

Namely, the axis title lists the number of missing values and the x-axis has been log transformed. This is because `tectr` attempts to guess reasonable parameter values which yield more informative plots than the default values of `ggplot`. It is able to do so because these default values can flexibly be changed.

Behind the scenes, the function calls `fx_default` which sets several default values for every column and saves them in a metaframe which is stored as an attribute of the data frame. The result can be seen in figure 3.1.

```
fx_ggplot_columns
```

```
## [1] "fxGeom_class"  "fxGeom_limits" "fxGeom_trans"  "fxInfo_name"
```

```
data <-
  df_vdem %>%
  select(v2x_polyarchy) %>%
  fx_default(columns = fx_ggplot_columns)
```

In `fxGeom_class`, for instance, the class of the variable is listed, which has an influence on the defined graphics and affects many other default values.

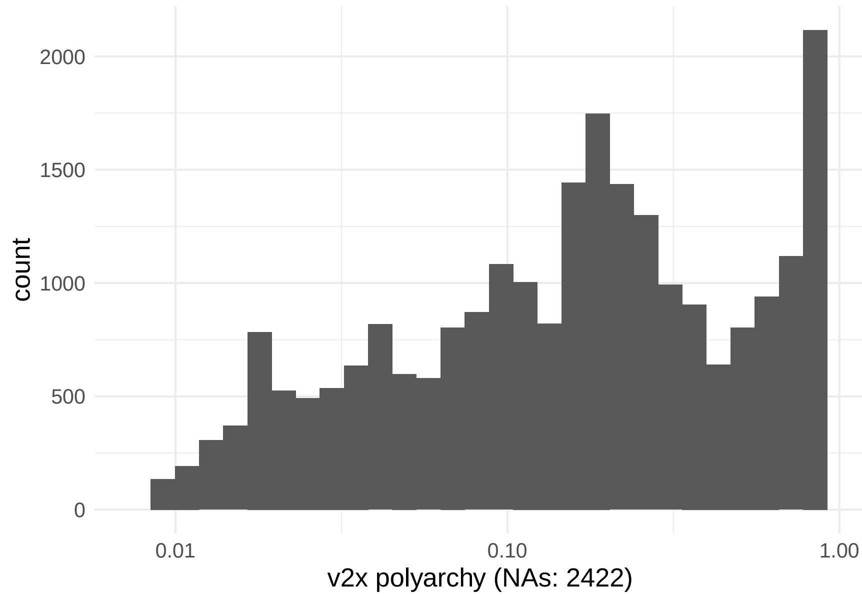

Figure 3.1: Visualization of v2x_polyarchy with `fx_ggplot`

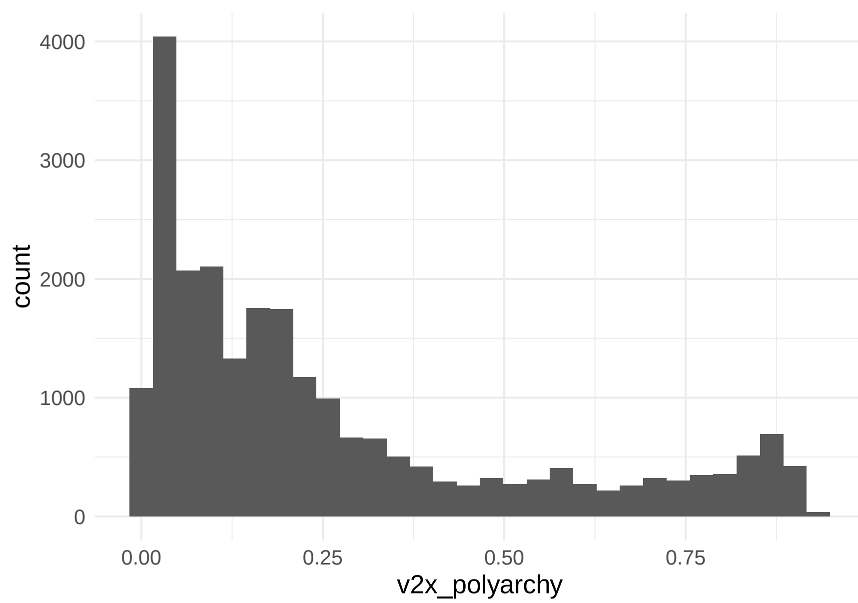

Figure 3.2: Ordinary histogram of v2x_polyarchy

Table 3.1: The default metaframe of v2x_polyarchy.

name	fxGeom_class	fxGeom_limits	fxGeom_trans	fxInfo_name
v2x_polyarchy	Continuous	c(0.007, 0.94)	log10	v2x polyarchy

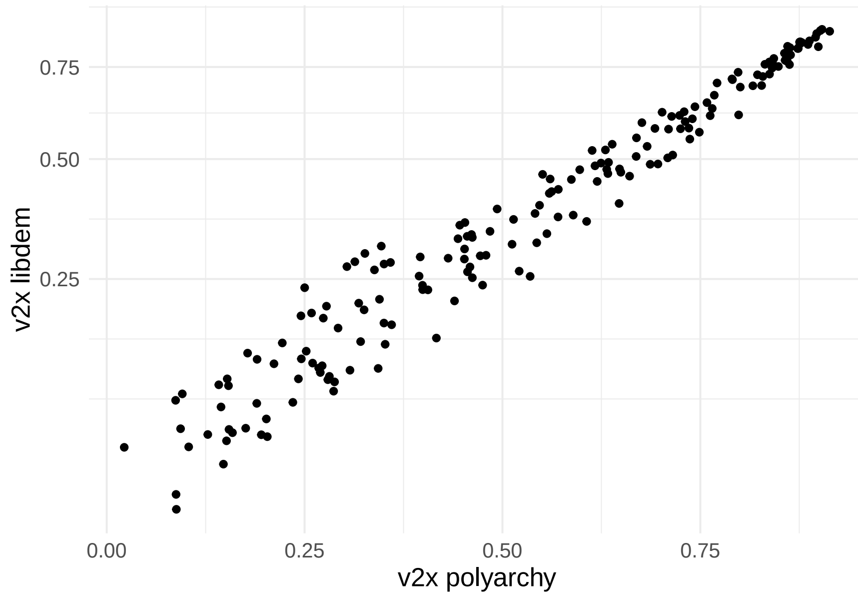

Figure 3.3: Scatter plot with `fx_ggplot`

`fx_ggplot` itself attempts to infer the appropriate visualization from the specification of aesthetics and the class of the data. Thus, we receive a different plot if we specify the y variable, as well. Let us first consider the relationship between the electoral and the liberal democracy index in the 2017 which can be seen in figure 3.3.

```
df_vdem %>%
  filter(year == 2017) %>%
  select(v2x_polyarchy, v2x_libdem) %>%
  fx_ggplot(aes(x = v2x_polyarchy, y = v2x_libdem)) +
  my_theme
```

In this case, the scatter plot is a sensible choice. If we wish to visualize the entire database (which consists of 26537 observations), an ordinary scatter plot would lead to severe overplotting. `fx_ggplot` recognizes this and chooses a more appropriate graphic in figure 3.4.

```
df_vdem %>%
  select(v2x_polyarchy, v2x_libdem) %>%
  fx_ggplot(aes(x = v2x_polyarchy, y = v2x_libdem)) +
  my_theme
```

This choice is flexible, as well; should we, for instance, wish to use the colour aesthetics to represent the year of the observation, the plot adapts, as can be seen in figure 3.5.

```
df_vdem %>%
  select(v2x_polyarchy, v2x_libdem, year) %>%
  fx_ggplot(aes(x = v2x_polyarchy, y = v2x_libdem, colour = year)) +
  my_theme
```

The increased transparency of the points improves oversight over the plot. Note that `fx_ggplot` is not intended to protect the user from specifications within `aes` that do not make sense but rather intends to build the best plot for the specified aesthetics. Adding the size of the points as an additional aesthetic would make the plot confusing. This is, however, not the scope of `fx_ggplot`. I will discuss this point further in chapter 4.

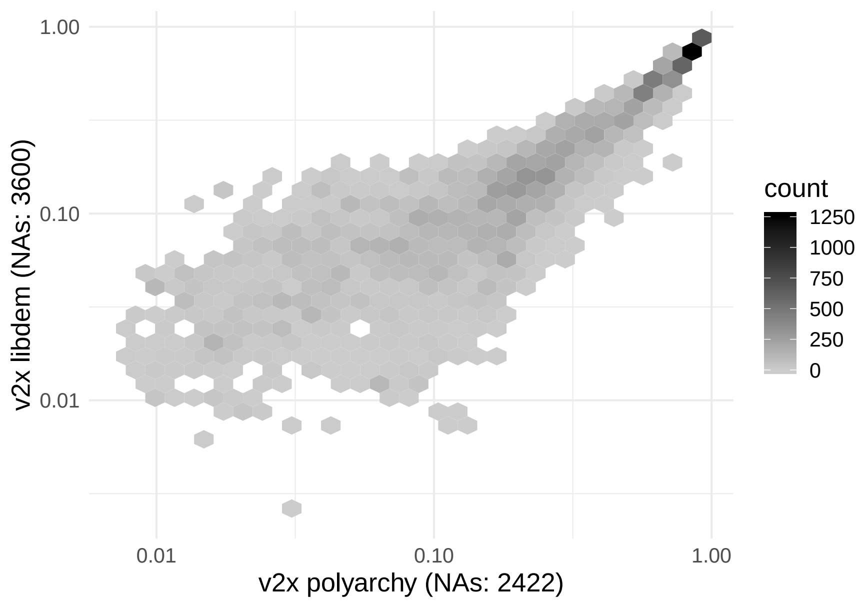

Figure 3.4: `fx_ggplot` prevents overplotting

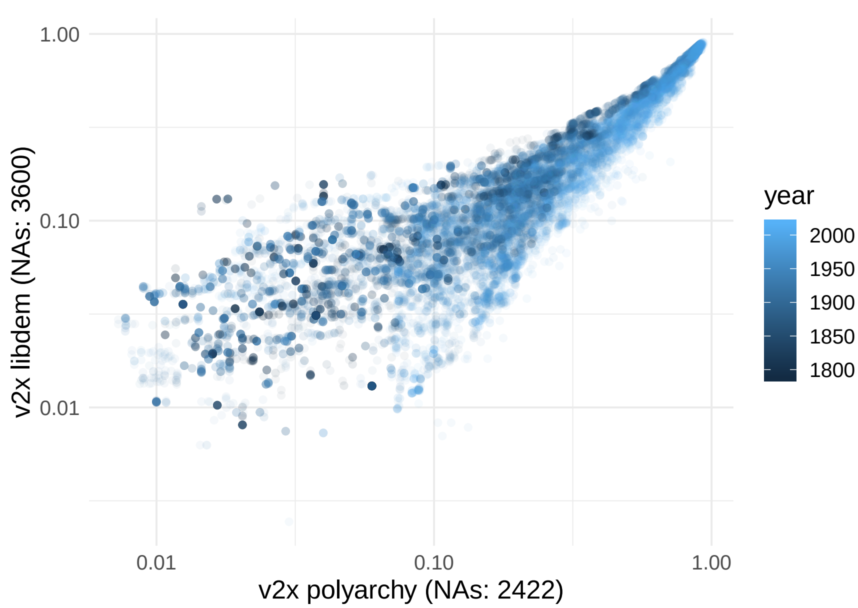

Figure 3.5: `fx_ggplot` with three variables

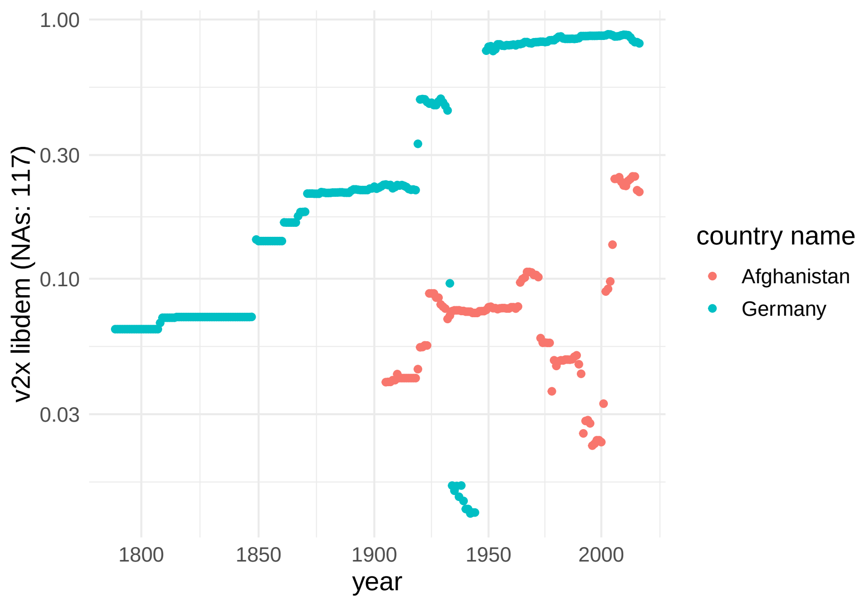

Figure 3.6: Time series with `fx_ggplot`

3.2 Modifying the default values

The default values which the plot uses may be changed by modifying the metaframe in which they are stored. Consider, as an example, the development of the liberal democracy index in Germany and Afghanistan over time (figure 3.6).

```
df_vdem %>%
  filter(country_name %in% c("Germany", "Afghanistan")) %>%
  select(year, v2x_libdem, country_name) %>%
  fx_ggplot(aes(x = year, y = v2x_libdem, colour = country_name)) +
  my_theme
```

This is evidently not a very useful plot; although we are able to extrapolate the developments in both countries, a line plot would be preferred. The problem is that `fx_ggplot` does not know that year represents a time. The fxGeom class corresponding the time variable is "Time". We can modify the metaframe accordingly.

```
# To modify default values, we need to define a function that changes
# the fxGeom_class of the questionable names and returns that
# (modified or unmodified) argument.

year_is_time <- function(fxGeom_class, name) {
  if(name == "year") return("Time")
  fxGeom_class
}

df_vdem %>%
  filter(country_name %in% c("Germany", "Afghanistan")) %>%
  select(year, v2x_libdem, country_name) %>%

  # If called without an argument, fx_default simply instantiates the
  # metaframe with the names.
```

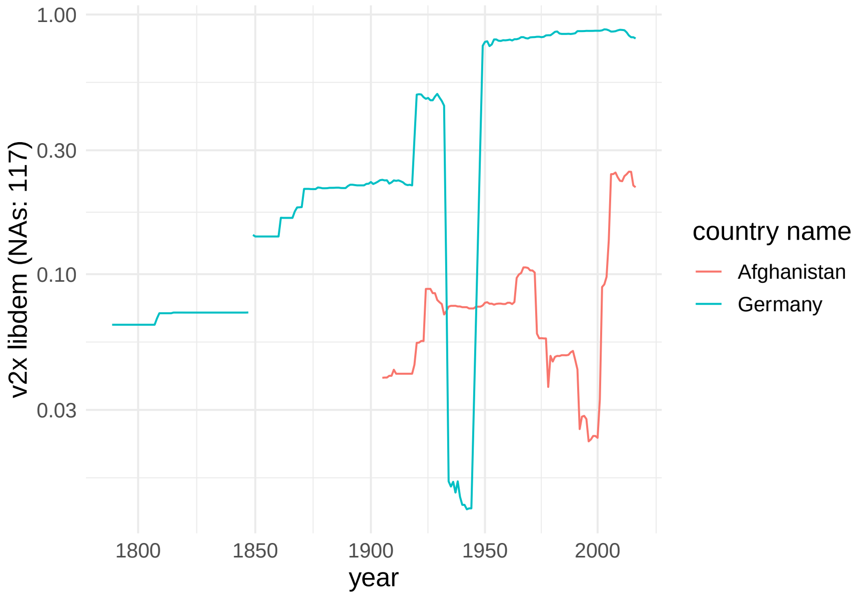

Figure 3.7: Time series with modified metaframe

```
  fx_default() %>%

  # The function responsible for the default value is fx_default_fxGeom_class.
  # It accepts as an argument a function that changes the default classes.

  mutate_mf(
    fxGeom_class = fx_default_fxGeom_class(., custom_fun = year_is_time),
    fxGeom_assoc_vars = aes(group = country_name)

    # The second variable specifies which observations should be viewed as
    # instances of a coherent unit, i. e. be connected by lines.

    ) %>%
  fx_ggplot(aes(x = year, y = v2x_libdem, colour = country_name)) +
  my_theme
```

In the resulting figure 3.7, the expected line plot is displayed. The time class is able to prevent overplotting, as well. Suppose that we now consider all observations:

```
df_vdem %>%
  select(year, v2x_libdem, country_name) %>%
  fx_default() %>%
  mutate_mf(
    fxGeom_class = fx_default_fxGeom_class(., custom_fun = year_is_time),
    fxGeom_assoc_vars = purrr::map(
      name,
      function(name) {
        if(name == "year") return(aes(group = country_name))
        aes()
      }
    )
```

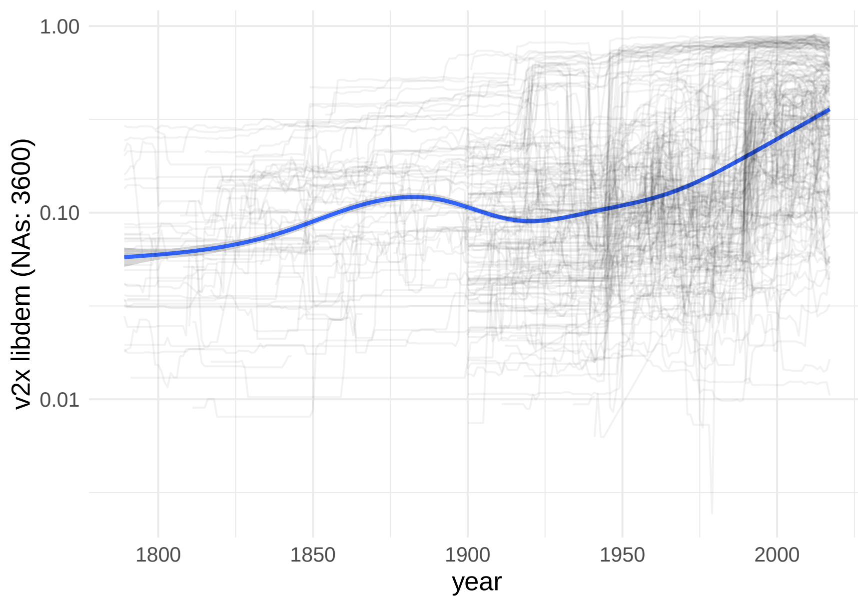

Figure 3.8: `fx_ggplot` prevents overfitting in time series

```
) %>%
fx_ggplot(aes(x = year, y = v2x_libdem)) +
my_theme
```

3.3 Summary

The ability to modify the default values is a crucial feature of `tectr` and I will discuss the broader background of these considerations in the next chapter.

Summarizing the depicted workflow:

- at first, we were interested in different visualizations and we approximated this by specifying aesthetics,
- the first visualizations broadly matched our idea and we saw no need to adapt the plots,
- as we encountered a plot we were unhappy with, we changed two default parameters. We were content with the result once again and the modification was both quick and easy to store.

This is the essence of `tectr`'s goal and the considerations in the subsequent chapter will be a more extensive embedding of the following two priorities:

1. Implement strong defaults which the user needs to change as scarcely as possible,
2. When the user needs to change something, make it easy and permanent.

These are, of course, very general principles. When applied to `tectr`, however, it becomes evident in what way they would not be of use in other packages.

Take, for instance, the scale transformations. While they will be discussed in more detail below, the default assumes certain transformations for skewed data. This would be completely unsuitable in the case of the underlying package `ggplot2`. Imputed transformations introduce uncertainty as the users are unsure what exactly their command will produce unless they specify a lot of parameters. In most cases, easy defaults are therefore more sensible than strong defaults. Only the second priority makes them useful; if they could not reliably and permanently be changed, they would constitute a constant burden. With these two priorities, however, they are more likely to fulfill their intended purpose of decreasing the cognitive cost of the user.

4 | Description of `tectr`

4.1 Effective Explicitness: A perspective of knowledge

Let us revisit the introduction: specialized functions and default values render R a powerful programming language for statistical analysis. These concepts are so useful because they constitute crystallized knowledge: when the user calls the function `read.csv`, they effectively use the programmer's knowledge of how to read in csv-files and therefore lighten their own cognitive load. Of course, this also applies if they have written the function themselves; in this case, by externalizing the knowledge they need not think about the specifics anymore. There are therefore two main advantages of externalizing knowledge. Other users benefit from the user's knowledge and the user benefits from its automation.

Normally, these default values cannot be particularly fancy as the user needs to understand under all circumstances what the function yields – even if this result is not ideal. This is the restriction that will be loosened for `tectr`. The intention behind this is to automate more knowledge.

How does this work? Consider a hypothetical statistician who looks at a dataset. They might begin by plotting the distribution and a few summary statistics of the variable. Simultaneously, they learn the name and the definition of a variable. They might look at relationships between several variables. Thereby, the statistician creates internal knowledge concerning a proper form of the statistical summary or a certain visualization. This internal knowledge accounts for different aspects; for instance, they might determine a better axis title than `v2x_polyarchy` or find it helpful to denote the mean of the variable on the axis. They might also determine that a line plot is more suitable than a scatter plot or find that a certain scale transformation is helpful.

This knowledge is seldomly notated which leads to a less efficient workflow. After all, this means that while exploring the data, the statistician will need to think about every modification of the plot which deviates from the default value. As a by-product, any non-crucial change will be left out, even though the statistician might actually benefit from e. g. a more descriptive axis title. More severely, when they want to communicate their results, they need to reproduce the internally held knowledge: if they create a certain visualization, they need to modify the axis title, in a table describing the variables they need to replace the non-descriptive variable names. If they produce an interactive visualization of the statistical analysis, the problem becomes even bigger; in this case, they need to externalize their knowledge after the fact.

In order to resolve these issues, `tectr` attempts to track these decisions while they are being made. The program does that by asking the user to be explicit about their knowledge – which is usually only represented internally.

If `tectr` used default values of a similar kind as other functions, this would lead to a severe burden on the user as it is time expensive to externalize knowledge. `tectr` therefore uses more flexible default values that attempt to approximate the statistician's thought process. The user therefore mainly has to intervene when something does not adhere to canonical assumptions. As they are more likely to think about these situations explicitly anyway, the burden might even decrease considering that under usual circumstances, the users will be forced to make these explicit decisions several times.

I will call this concept effective explicitness and it is central to `tectr`. To emphasize that, this term guides the semantics of `tectr`: those functions and objects which support effective explicitness will be preceded by `fx` (e**f**fective e**x**plicitness, pronounced: effex).

The notion of explicitness is clear in this context; if the user gains new knowledge that is not represented within the metaframe, they should be explicit about it. This applies to visualizations as well as to general observations which he may notate in a comments field. This process should be effective in two ways: firstly, the metaframe needs to imitate the statistical thought process closely enough that many of its approximations are valid; secondly, if adaptations need to be made, they should be scalable. Consider, as an example, the variable year in the previous chapter. The program's assumption that this was an ordinary continuous variable was wrong. In order to correct this, it was sufficient to make explicit that the variable referred to time and tracked the development for different countries. On the other hand, the variable could have been so unique that these little changes would not have sufficed. In these cases, it should be possible to make broader changes. The remaining chapter will present a pilot implementation of this more general philosophy. Before going on to discuss specific functions, I will present the underlying concepts which power these functions. In order to properly discuss the functions, I will focus on the following aspects:

- What do the functions return?
- How does the function allow the user to be explicit?
- What would be alternative implementations?
- How mature is the function?

The last point is especially important: as a pilot implementation, `tectr` provides functionality that seems to go into the right direction. However, only a combination of more exhaustive features and applications will determine whether the approach was the right one. It is therefore important to identify those functions where an entirely different track of thought might be necessary in the future. In order to make the description lightweight and not focus too much on the particularities of implementation, I will often refer to the package documentation for further information.

4.2 Underlying structure

4.2.1 The metaframe

The metaframe is the attribute of a database which captures the knowledge of the user. It adheres to the principle of tidy data (Wickham, 2014):

> Each variable forms a column.
>
> Each observation forms a row.
>
> Each type of observational unit forms a table.

Clearly, additional information about observations can be stored in an additional column. The metaframe stores additional information about variables by treating every variable as an observation and every type of information as a column. It consists of different protected columns which fulfill specific functions. The `name` column refers to the corresponding variable name. The remaining information is preceded by a keyword to make clear what the variable refers to. More specifically, those variables that refer to semantic information are preceded by `fxInfo` (e. g. the "proper" name of the variable `fxInfo_name`) and those variables that provide parameters to the visualization are preceded by `fxGeom`.

For details on metaframes, how to set or change them, see the documentation. In my opinion, the best workflow is instantiating the metaframe with `fx_default` and then adding columns via `mutate_mf`. This allows you to only use the data frame itself as a function argument and provides a concise terminology:

Table 4.1: A simple metaframe

name	fxInfo_name
ht	Height

```
data <-
  tibble(ht = c(158, 189, 178)) %>%
  fx_default() %>%
  mutate_mf(fxInfo_name = "Height")
metaframe(data)
```

4.2.2 Extending fx functions

4.2.2.1 Dispatch

Whereas in most cases, modifying the parameters to the functions should be sufficient, there are also many applications where a more extensive modification is necessary – or at least more effective. `tectr` uses method dispatch to coordinate these extensions. In certain occasions, S4 classes are necessary. As long as S3 class suffice, they are utilized.

It is important to emphasize that the fx functions themselves are not generics. Internally, every fx function calls an `fxi` (short for fx internal) function which modifies its arguments to make dispatch possible[1]. I will elaborate on the particular mechanism in the following sections. By defining dummy classes, the calls to so called `fxe` (short for fx extendible) functions employ dispatch and can be extended. The metaframe information of the current variable is called as a spliced input to the `fxe` function. For instance, in the example above, an extendible function `fxe_fun` would have received as additional input the argument `fxInfo_name` and `name`.

4.2.2.2 S3 dummy classes

S3 dummy classes are mainly powered by the function `fxd` which creates a subclass for a specific task. For instance, the task "default" which specifies the default values for the metaframe columns has as its subclasses the corresponding columns, e. g.:

```
fxd("default", "fxInfo_name")
```

```
## list()
## attr(,"class")
## [1] "fxd_default_fxInfo_name" "fxd_default"
## [3] "fxd"
```

This provides a lightweight and effective mechanism for dispatch via dummy classes.

4.2.2.3 S4 dummy classes

S4 dummy classes are powered by different classes for every task they employ. For instance the `fxGeom` dummy classes all inherit from `fxGeom` and consist of the aforementioned possible classes of input to visualization. Consider, for instance,

[1] In some cases, these modifications are so simple that they happen in the fx function and the fxi function is omitted.

```r
fxGeom("Continuous")
```

```
## An object of class "fxGeomContinuous"
## [1] "Continuous"
```

which yields the corresponding S4 class `fxGeomContinuous`.

This mechanism provides a better overview over all possible classes. However, it is necessary to explicitly define every task and every subclass. Normally, the `fxd` mechanism should be preferred. S4 dummy classes are mainly employed when multiple dispatch becomes necessary (see `fx_ggplot`).

4.2.3 Other concepts

I have defined a few other objects that can be useful.

4.2.3.1 Filepaths

The S3 class `filepath` consists of a character vector which contains paths to external data and a `reader` attribute which can read in the data. It allows you to retrieve stored data within a metaframe in an uncomplicated way.

4.2.3.2 fx factors

fx factors are a very simple class which extends the concept of factors from `character`s to arbitrary classes. Certain columns of the metaframe might, for instance, contain different functions many times and it would be a waste to store them separately. `fx_factor` creates an object which has its unique values as attribute levels and refers to them within every element via the index of the levels. The original object can be recreated via `fx_evaluate`.

4.3 `fx_default`

4.3.1 General purpose

`fx_default` computes default values for every column of the metaframe so that the different fx functions can fill up the metaframe with the remaining necessary columns. The function therefore uses specific heuristics for the different columns to infer sensible values. As in many cases, the default values are modified, for every column there is a function `fx_default_<colname>` (e. g. `fx_default_fxInfo_name`) which allows the user to access the default values separately from the internal mechanism of `fx_default`.

A call to `fx_default` specifies the columns to be imputed. By default, if the data frame has no metaframe, it creates the metaframe with one observation for every variable name.

4.3.2 Extensions

Extensions may be provided via S3 method dispatch over the function `fxe_default` and the class `fxd_default_<colname>`. The function may depend on the data and on its metaframe. If possible, it is recommended that the functions only depend on the data frame and the name column of the metaframe. Consider, for instance, the default function for `fxGeom_class`:

```r
tectr:::fxe_default.fxd_default_fxGeom_class
```

```
## function(data, mf, col, ...)
##   fx_default_fxGeom_class(data, mf)
## <bytecode: 0x000000001d610da8>
## <environment: namespace:tectr>
```

Internally, it calls the more accessible function `fx_default_fxGeom_class` which does the work (for details, see the documentation).

4.3.3 Alternative implementations

Like with respect to general functions in R, there are essentially two possible implementations of default values in `tectr`: either the default is specified in the head or the function recognizes that the default is needed if the argument has the value `NULL`. Which mechanism is preferable, depends strongly on the context. It is recommended that parameters that are specific to one function and have a very simple default value (for instance, a constant numerical value) are marked via `NULL` whereas the user may want to access broadly applicable parameters that employ a more elaborate inference mechanism directly. These should therefore be implemented via an `fx_default` function.

4.3.4 Lifecycle

The purpose of `fx_default` is clearly outlined and its application is very simple. Modifications of the internal dispatch are possible but the extendible functions and `fx_default` itself are unlikely to be reworked.

4.4 `fx_info`

4.4.1 General purpose

This function provides information on the different variables to the user. Examples for this information would be

- the name of the variable (which is often different from its internal name),
- the description or definition of the variable,
- summary statistics.

A call to `fx_info` consists of the data itself, a topic and potentially additional parameters. The topic may either be a certain semantic topic like the name or the description or it may invoke a specific method. The most useful example of a topic is “stats” which renders a summary table with descriptive statistics that can be specified in the argument statistics:

```
fx_info(mtcars, "stats", statistics = c("mean", "quantile"))
```

```
## # A tibble: 11 x 7
##    name     mean `quantile: 0%` `quantile: 25%` `quantile: 50%`
##    <chr>   <dbl>          <dbl>           <dbl>           <dbl>
##  1 mpg    20.1           10.4            15.4            19.2
##  2 cyl     6.19           4               4               6
##  3 disp  231.            71.1           121.            196.
##  4 hp    147.            52              96.5           123
##  5 drat    3.60           2.76            3.08            3.70
##  6 wt      3.22           1.51            2.58            3.32
##  7 qsec   17.8           14.5            16.9            17.7
##  8 vs      0.438          0               0               0
```

```
##  9 am      0.406           0               0               0
## 10 gear    3.69            3               3               4
## 11 carb    2.81            1               2               2
## # ... with 2 more variables: `quantile: 75%` <dbl>, `quantile: 100%` <dbl>
```

The function returns a data frame with a certain information in every column. If no method for the particular topic exists, the function searches the metaframe for a column of the name "fxInfo_" and returns that. Consider, for instance, a name and a short comment:

```
mtcars %>%
  select(mpg) %>%
  fx_default() %>%
  mutate_mf(
    fxInfo_name = "Miles/(US) gallon",
    fxInfo_comment = "A gallon is 3.79 litres."
  ) %>%
  fx_info(c("name", "comment", "stats"), statistics = c("mean", "quantile"))
```

```
## # A tibble: 1 x 9
##   name  Name  Comment  mean `quantile: 0%` `quantile: 25%` `quantile: 50%`
##   <chr> <chr> <chr>   <dbl>          <dbl>           <dbl>           <dbl>
## 1 mpg   Mile~ A gall~  20.1           10.4            15.4            19.2
## # ... with 2 more variables: `quantile: 75%` <dbl>, `quantile: 100%` <dbl>
```

We will later use this functionality to create meaningful descriptive tables.

You may provide either characters or functions to statistics. Characters are evaluated via `do.call`, functions are called directly. If you specify a function, you should allow for arbitrary parameters via `...` as a function will receive the columns of the metaframe as input. A function's first argument should be the corresponding column of the data frame. Consider the following example, in which the number of digits of the computed mean depends on the information in the metaframe:

```
stat_mean <- function(x, ..., fxInfo_digits)
  x %>% mean() %>% round(digits = fxInfo_digits)
mtcars %>%
  select(mpg) %>%
  fx_default() %>%
  mutate_mf(
    fxInfo_name = "Miles/(US) gallon",
    fxInfo_digits = 3
  ) %>%
  fx_info(c("name", "stats"),
          statistics = list(mean = stat_mean))
```

```
## # A tibble: 1 x 3
##   name  Name               mean
##   <chr> <chr>             <dbl>
## 1 mpg   Miles/(US) gallon  20.1
```

4.4.2 Extensions

Extensions can be provided via dispatch over `fxe_info`. The dummy class for a certain topic will be provided by `fxd("info", <topic>)`, e. g.

```
fxd("info", "stats")
```

```
## list()
```

```
## attr(,"class")
## [1] "fxd_info_stats" "fxd_info"       "fxd"
```

4.4.3 Lifecycle

In my opinion, the combination of simplicity and power makes `fx_info` a very useful function that will most likely retain its structure. This means that new functions which enhance the power of `fx_info` and especially its stats will be the next step in its development.

4.5 `fx_output`

4.5.1 General purpose

This function makes `fx_info` immediately applicable in a wide variety of applications. It transforms the data frame returned by `fx_info` into a certain form. In this case, a sparse wrapper around a function from another package often suffices. `fx_info` may return different formats. Currently, the supported formats are rst, html, latex and markdown. The two most useful forms that have been implemented so far are table and collapse.

As an example, we will consider the mtcars summary statistics:

```
stat_median <- function(x, ...)
  x %>% median() %>% round(digits = 3)
stat_mean <- function(x, ...)
  x %>% mean() %>% round(digits = 3)
info <-
  mtcars %>%
  fx_info("stats", list(mean = stat_mean, median = stat_median))
info
```

```
## # A tibble: 11 x 3
##    name     mean median
##    <chr>   <dbl>  <dbl>
##  1 mpg    20.1    19.2
##  2 cyl     6.19    6
##  3 disp  231.    196.
##  4 hp    147.    123
##  5 drat    3.60    3.70
##  6 wt      3.22    3.32
##  7 qsec   17.8    17.7
##  8 vs      0.438   0
##  9 am      0.406   0
## 10 gear    3.69    4
## 11 carb    2.81    2
```

```
fx_output(info, form = "table", out_format = "markdown")
```

name	mean	median
mpg	20.09	19.20
cyl	6.19	6.00
disp	230.72	196.30
hp	146.69	123.00

name	mean	median
drat	3.60	3.69
wt	3.22	3.33
qsec	17.85	17.71
vs	0.44	0.00
am	0.41	0.00
gear	3.69	4.00
carb	2.81	2.00

The form "collapse", on the other hand yields one row per observation. This row consists of different cells in which the variable information and its name are listed in a specified format and which are then glued together:

```r
fx_output(info, form = "collapse")
```

```
## mean:  20.09, median:  19.2
## mean:   6.19, median:   6.0
## mean: 230.72, median: 196.3
## mean: 146.69, median: 123.0
## mean:   3.60, median:   3.7
## mean:   3.22, median:   3.3
## mean:  17.85, median:  17.7
## mean:   0.44, median:   0.0
## mean:   0.41, median:   0.0
## mean:   3.69, median:   4.0
## mean:   2.81, median:   2.0
```

These specifications can also be modified:

```r
fx_output(info, form = "collapse", cell_scheme = "The {name} is {value}.", cell_sep = " --- ")
```

```
## The mean is  20.09. --- The median is  19.2.
## The mean is   6.19. --- The median is   6.0.
## The mean is 230.72. --- The median is 196.3.
## The mean is 146.69. --- The median is 123.0.
## The mean is   3.60. --- The median is   3.7.
## The mean is   3.22. --- The median is   3.3.
## The mean is  17.85. --- The median is  17.7.
## The mean is   0.44. --- The median is   0.0.
## The mean is   0.41. --- The median is   0.0.
## The mean is   3.69. --- The median is   4.0.
## The mean is   2.81. --- The median is   2.0.
```

Further information can be found in the documentation.

4.5.2 Extensions

`fx_output` may be extended via the `fxd` dummy class with the topic "output".

4.5.3 Lifecycle

`fx_output` is stable and the next step will be to refine and extend the output forms. In particular, the next version will provide an output form "report" which creates a dynamical report of all variables.

4.6 `fx_write`

4.6.1 General purpose

Besides providing summary tables, the main purpose of `fx_info` is to make information about the variable accessible. The foundation of this information can often be imported from a codebook of some form (see the application). Adding and editing this information – e. g. correcting typos – is, however, an awkward endeavour in R. A text editor of some sort would be more suitable. The `fx_write` family of functions attempts to adress this by allowing you to export the semantic data to human-readable documents which can then be modified. As it returns the data frame with a `filepath` column which refers to the export file, it can easily be read in – this functionality is provided by `fx_read`.

Currently, only one `fx_write` function is implemented: `fx_write_json`. This function creates several files in JavaScript Object Notation which is well readable by humans. The function is powered by the package `jsonlite` (Ooms, 2014) which uses a class based mapping to convert between JSON data and R objects. For details, see the documentation and the fifth chapter.

4.6.2 Extensions

While `fx_write` does not employ any kind of dispatch but is a family of functions, it can be extended (in a looser definition of the word) by providing a new function with a consistent interface `fx_write_<format>`.

4.6.3 Alternative implementations

`fx_write` is currently an experimental feature which has its disadvantages. Most notably, it is difficult to change the codebook import after somebody has changed the output of this process (i. e. the exported files). While version control would make this simpler, the fact remains that all changes to the document would have to be manually reentered. A primitive editor within R which tracks changes would render the process more reproducible and less complicated. However, this would be a considerable effort with limited applicability and for the time being, it seems to me that a method as represented by `fx_write_json` (or possibly another format) represents the best alternative.

4.6.4 Lifecycle

This function is at the moment very unstable and should be used with caution.

4.7 `fx_ggplot`

At last, I present the fx function which has the most extensive implementation so far: `fx_ggplot`, which is concerned with the visualization of variables. This functions makes a more extensive introduction necessary. The first section will discuss the general concept behind `fx_ggplot`. We will then discuss the implementation in `tectr` and how it may be extended. The fourth section discusses possible alternatives and the fifth section elaborates how development of the function will progress.

4.7.1 Concept of `fx_ggplot`

This function intends to provide a flexible visualization. As an input, it accepts aesthetics and then attempts to produce the best visualization that is compatible with the specified aesthetics. Therefore, it does not answer the question “What should I know about these five variables?”. More specifically, it does not prevent you from overplotting, nonsensical aesthetics etc. In my opinion, it is best applied by a person who has an idea of the expected results. In this case, it is especially adept at flexible, interactive visualizations (see below). For instance, if a statistician is interested in several time series of different variables, these should be differently visualized depending on the available information.

While I will discuss this concept below, I will assume, for now, that this method is valid and consider the following question: How do we infer the best visualization from a given set of aesthetics? As the function is created within the syntax of `ggplot2`, our task is to infer the set of elements which constructs the plot. These elements belong to two categories: dependent and independent.

Independent elements will be added for a certain specified aesthetics regardless of the other aesthetics, e. g. the title of the x axis. This title only depends on the x aesthetic. The y aesthetic has no influence on this layer.

On the other hand, the y aesthetic has a big influence on the question what geometry we should add, as the suitable geometry depends on all aesthetics. Such elements are therefore called **dependent**.

The determination of independent elements is relatively clear and simple; it only depends on one aesthetic and it is therefore easy to define a function which depends on certain parameters of the variable and yields a layer. On the other hand, the determination of dependent layers is messier, as we will see below.

For that purpose, I will first present the original concept I had implemented for dependent layers: a strictly rule-based structure which specifies, for any combination of relevant aesthetics which layers this would yield. For instance, this structure included:

- continuous x-variable and continuous y-variable: `geom_point()`
- discrete x-variable and continuous y-variable: `geom_boxplot()`

The basic plots are easily produced with such a system but whereas it allows the user to be very explicit, it is very difficult to implement a meaningful influence via parameters. Instead, any new kind of visualization made necessary an entirely new class with a wide variety of methods. Furthermore, despite the seeming clarity of these simple rules, the resulting massive dispatch was likely to end in a lot of confusion and an unmanageable code base.

I have therefore developed an alternative approach which implements a voting system for the dependent layers. (In contrast, I will refer to my initial idea as a rule-based system.) As a brief overview, the voting system consists of three phases:

- In the **nomination phase**, every variable nominates possible sets of layers which would make sense with this variable. These nominations depend on the aesthetic, the `fxGeom_class` and possibly the metaframe parameters.
- In the **veto phase**, every variable receives the entire list of nominations and removes those, it is incompatible with.
- In the **voting phase**, every variable distributes a number of votes on the various remaining nominations. These votes depend on properties of the nominations. The nomination with the most votes is chosen as the winner and yields the dependent layers.

This approach may appear unconventional, at first, and I will adress some counterarguments below. But before that, I will present the implementation and process of extension for both the independent and the dependent layers.

I will introduce the implementation with a simple example from the diamonds dataset: we will consider as x aesthetic the weight in carat and as y aesthetic the price. In the end, we will have created figure 4.1.

```
fx_ggplot(diamonds, aes(x = carat, y = price)) +
  my_theme
```

The first step is to infer the default values of the metaframe. All required columns for `fx_ggplot` are stored in the vector `fx_ggplot_columns`:

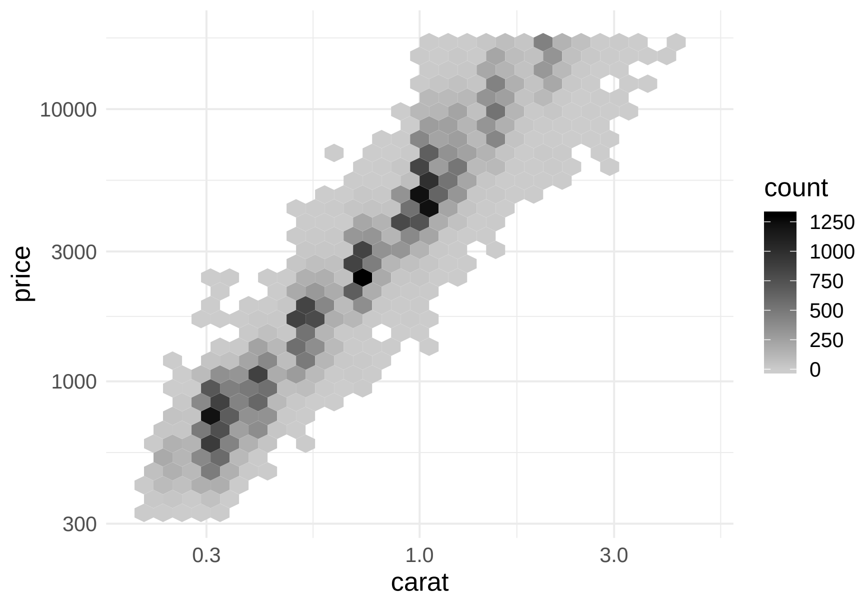

Figure 4.1: Example for `fx_ggplot`

Table 4.3: The default metaframe for `fx_ggplot`.

name	fxGeom_class	fxGeom_limits	fxGeom_trans	fxInfo_name
carat	Continuous	c(0.2, 5.01)	log10	carat
price	Continuous	c(326, 18823)	log10	price

```
fx_diamonds <-
  diamonds %>%
  select(carat, price) %>%
  fx_default(columns = fx_ggplot_columns)
metaframe(fx_diamonds)
```

4.7.2 Implementation of the independent layers

The independent layers are determined by the internal function `fxi_layer_single` which calls `fxe_layer_single` for every aesthetic. This function dispatches over `fx_geom`, the argument that is created by the `fxGeom_class` column. Therefore, the function is once called for the class `fxGeomContinuous` and `xAesName` and once for the class `fxGeomContinuous` and `yAesName`. The data frame itself is passed to the extendible function as well. The default method will simply concatenate the results of two underlying functions: `fxe_layer_scale` and `fxe_layer_other`, of which the former is usually more important. `fxe_layer_scale` depends on almost all arguments (each with the prefix "fxGeom_") which can be provided to a scale in `ggplot2`. If none are specified the default values will be used. Normally, at least the transformation and the limits will be inferred by `fx_default`. In this case, the limits correspond to the range of the data and therefore to the default of the scale, as well. However, both the x and the y scale are logarithmized in this case.

Whereas the remaining scale arguments are relatively evident and may be looked up in the documentation, automatically determining the transformation is a bit more complex. It is inferred by the function `fx_default_fxGeom_trans` which depends on two additional parameters: `fxGeom_trans_simple`, a boolean, and `fxGeom_trans_p.threshold`, a numerical value between 0 and 1. `fxGeom_trans_simple` determines how complicated the transformations are allowed to be. If it is true (the default), the function only chooses out of three transformations: identity, square root and log. It determines the variable's skewness as implemented in the `moments` package (Komsta and Novomestky, 2015) and chooses the transformation with the least absolute skewness. To ensure well-definedness, this only applies to positive values. In the case of nonpositive values, the identity is chosen by default. On an exploratory basis, this rule yielded good results. In the case of the weight and price of diamonds, a log-transformed scale is a sensible choice, as well.

The aforementioned three transformations are special cases of the more general boxcox-transformations which have first been proposed by G. E. P. Box and D. R. Cox (1964). These depend on the parameter $\lambda >= 0$ and take the following form:

$$y^{(\lambda)} = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{if } \lambda > 0 \\ \ln y & \text{if } \lambda = 0 \end{cases}$$

Thus, the log transformation is given by $\lambda = 0$, the square root transformation by $\lambda = 0.5$ and the identity by $\lambda = 1$.[2] These transformation can now be further generalized to allow that y is previously shifted by an offset in order to adapt the transformation to negative values, as well. This boxcox-transformation with offset is employed by the non-simple inference mechanism (i. e. `fxGeom_trans_simple = FALSE`).

The function first uses the Agostino-test as implemented in `moments` as a heuristic whether the data is skewed. If the p-value of the test is below the threshold given by the parameter `fxGeom_trans_p.threshold`, it fits λ and the offset with the help of the function `boxcoxfit` from the package `geoR` (Ribeiro Jr and Diggle, 2018). The resulting transformation is used for the corresponding scale.

While the latter method is more sensitive to the patterns in the data, it has several disadvantages: firstly, the Agostino-test is flawed as a heuristic as, for large datasets, it essentially always denies the null hypothesis, even if the data is only slightly skewed and the identity transformation is still suitable. These leads to many general boxcox transformations which do not adhere to common first analyses whereas a log- or square root-scale is employed more frequently. Finally, in some instances, the boxcoxfit function does not converge resulting in an error.

These are the reasons why the simple transformation inference is set as the default.

Due to the large skewness, both scales therefore return a scale with a log transformation.

It might also be interesting to see that `fx_info` is applied at this stage - this is how the number of missing values was displayed in the plots in chapter three. The relevant topic is the "title" which can be modified by parameters such a `fxInfo_title_na.show` or, more generally, `fxInfo_title_stats`. Consider the following example with modified statistics and a unit on the y axis.

```
fx_diamonds <-
  diamonds %>%
  select(carat, price) %>%
  fx_default(columns = fx_ggplot_columns) %>%
  mutate_mf(fxInfo_title_na.show = name == "price",
            fxInfo_title_n.show = TRUE,
            fxInfo_title_stats = "mean",
            fxInfo_unit = purrr::map(name, ~ if(. == "price") "$" else NULL),
            fxInfo_title_unit.show = TRUE)
metaframe(fx_diamonds) %>%
  select(name, fxInfo_name:fxInfo_unit)
```

This modified metaframe (see table 4.4) results in the plot in figure 4.2.

[2] More precisely, they are affine transformations of these functions and scale transformations are invariant in affine functions.

Table 4.4: Metaframe of diamonds with modified parameters (Extract)

name	fxInfo_name	fxInfo_title_na.show	fxInfo_title_n.show	fxInfo_title_stats	fxInfo_unit
carat	carat	FALSE	TRUE	mean	NULL
price	price	TRUE	TRUE	mean	$

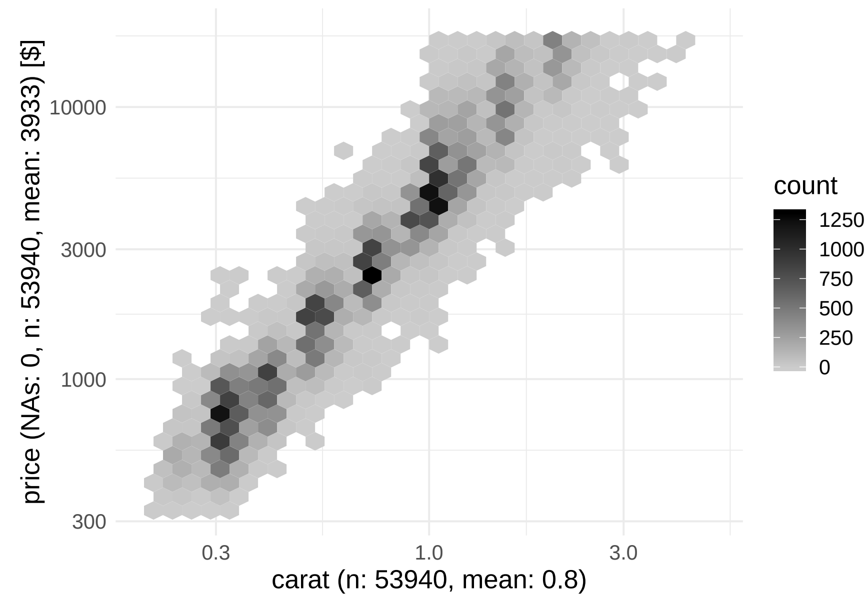

Figure 4.2: Visualization with metaframe parameters as in table 4.4

```
fx_ggplot(fx_diamonds, aes(x = carat, y = price)) +
  my_theme
```

I refer to the documentation for more details.

Another independent layer function is given by `fxi_labeller` which provides the labeller for a facetting variable. Facetting variables are specified in the argument `facet_vars` and need to be captured by the function `vars()`. An example can be seen in figure 4.3.

```
diamonds %>%
  filter(color %in% LETTERS[c(4,10)]) %>%
  fx_ggplot(aes(x = carat, y = price), facet_vars = vars(color)) +
  my_theme
```

4.7.3 Implementation of the voting system

4.7.3.1 Nomination

The dependent layers are determined by the internal function `fxi_layer_complete`. The nominations are handled by `fxe_layer_complete_nominate`, which is dispatched over `fx_geom` and `aes_name`, as well. The function accepts the parameter `fxGeom_nominations` which may provide additional nominations. As these are absent, only the default nominations are provided.

These are handled by the S3 class `nominations` which returns, depending on the access function, all layers,

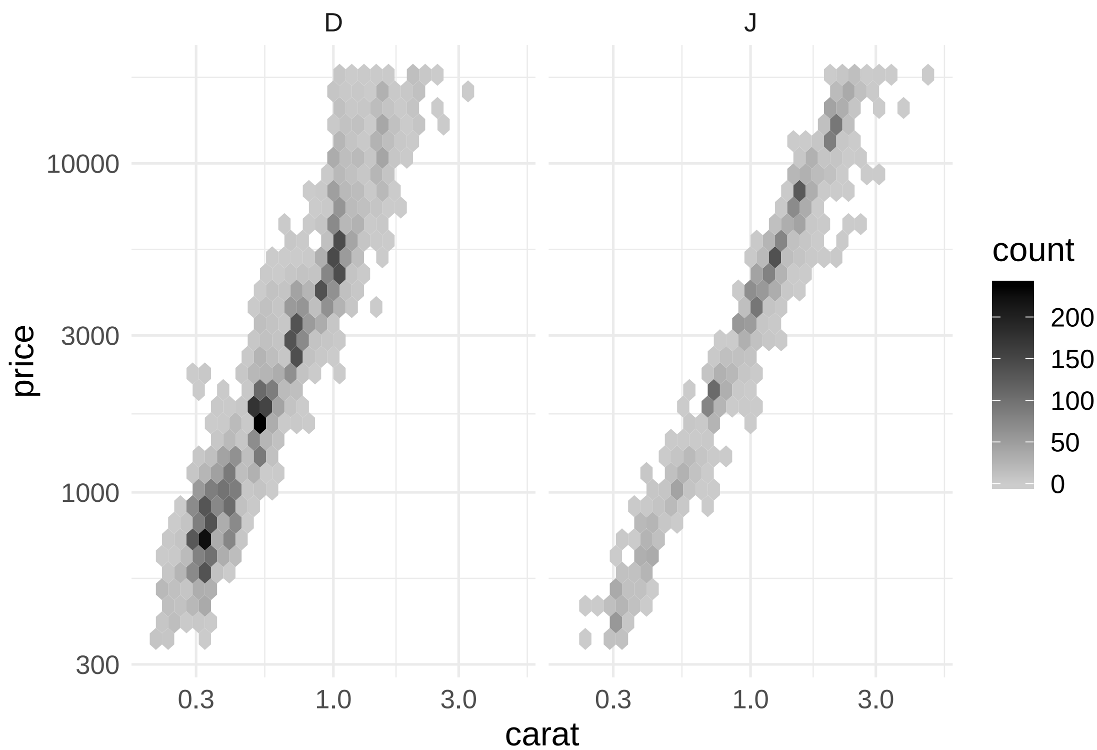

Figure 4.3: Facetting in `fx_ggplot`

scales, facets, coordinate system or other `ggproto` objects within a certain nomination (see the documentation). This will become important for votes and vetos.

The nominations by the x aesthetic can be caught by

```
x_noms <- fxe_layer_complete_nominate(fxGeom("Continuous"), AesName("x"), diamonds)
```

As such an object is messy to print, I will simply list the four nominations:

- `geom_point()` with an automatically determined transparency value `alpha`
- `geom_histogram()`
- `geom_density()`
- `geom_hex()`, `scale_fill_gradient()` with limits which are anchored at 0 and a custom palette.

Evidently, these layers are suitable for very different plots. If you consider which layers to nominate, you can simply choose all those that make sense for any combination that includes this aesthetic and `fxGeom_class`. Votes and vetos will take care of the rest.

Besides supplying additional nominations via `fxGeom_nominations`, you can extend the function by importing the S4 generic and defining a new method.

4.7.3.2 Vetoes

Going on, the function `fxe_layer_complete_veto` is responsible for the vetoes. Its structure is

```
fxe_layer_complete_veto(nomination, fx_geom, aes_name, data, ..., fxGeom_veto = NULL)
```

and it dispatches over `fx_geom` and `aes_name`. `fxGeom_veto` can specify a function which vetoes additional nominations. The internal function calls the appopriate method for every nomination. The method simply returns a boolean where `FALSE` causes no action and `TRUE` indicates that the nomination ought to be removed. Consider, for instance, the call below:

```
fxe_layer_complete_veto(
  nomination = nomination(geom_density()),
  fx_geom = fxGeom("Continuous"),
  aes_name = AesName("y"),
  data = diamonds)
```

```
## [1] TRUE
```

This aesthetic vetoes a density plot because it is incompatible with an existing y aesthetic. Consequently, only the scatter plot and the hexagonal heatmap remain as valid options (all other nominations from the y aesthetic have been vetoed, as well).

4.7.3.3 Votes

Finally, the following function casts the votes:

`fxe_layer_complete_vote(nomination, fx_geom, aes_name, data, ..., fxGeom_vote)`

`fxGeom_vote` can specify a function which may cast additional votes. `fxe_layer_complete_vote` dispatches over the same arguments and returns, for a given nomination, an integer which represents the votes for a certain nomination. In this case, we observe:

```
fxe_layer_complete_vote(
  nomination = nomination(geom_point()),
  fx_geom = fxGeom("Continuous"),
  aes_name = AesName("x"),
  data = diamonds)
```

```
## [1] 0
```

```
fxe_layer_complete_vote(
  nomination = nomination(geom_hex()),
  fx_geom = fxGeom("Continuous"),
  aes_name = AesName("x"),
  data = diamonds)
```

```
## [1] 3
```

The same applies to the y aesthetic. Why does the heatmap receive more votes? The reason lies in the large amount of data – the threshold at which the heatmap receives the majority of votes can be determined by the parameter `fxGeom_hex.threshold` and its default is set to 1000.

The hexagonal heatmap therefore receives 4 votes whereas its only rival, the scatter plot, receives 0 votes. From these steps, the above pictured plot emerges.

4.7.4 Discussion and alternatives

This subsection discusses the implementation of `fx_ggplot`, lists advantages and adresses possible disadvantages.

4.7.4.1 Easy extendibility

The partition of the `fx_ggplot` functionality into these functions allows the user to build a complex visualization by many very simple steps; he only needs to think about one scale, one possible (resp. impossible) visualization and so on at a time. If a visualization does not correspond to the user's need, it is simple enough to fix it; they need not define an entire new class but only modify certain parameters and if the need for a new class arises, its definition is both shorter and easier. This corresponds to the guideline of `tectr` that it should be easy to integrate into the common explorative process of a statistician.

4.7.4.2 High flexibility

This advantage is as much hypothesis as observation. The aforementioned easily acquired extension will also easily work with different classes that have not actually been intended to work together – they may even have been specified by different persons. In a rule-based process, it is not possible to achieve this kind of flexibility, as far as I can see it.

4.7.4.3 What about intransparency?

A possible argument against the voting system might be the following:

> It is very intransparent what arguments will result in which plot. All these different functions occlude what could better be solved by an independent and a dependent element function where the independent element function is simply a rule-based process.
>
> The amount of time such detailedness would require is easily exceeded by the follow-up process regarding an unsuitable visualization in the voting system. The user then needs to consider three different functions and identify a suitable solution which they must validate by some kind of testing. Finally, even when they have found a solution, changes in the functions of another aesthetic might result in an unsuitable plot once again.
>
> The voting system arises from laziness in the wrong domain and requires a higher effort than a rule-based system for a coequal result.

As applies to most arguments with regards to programs in such early stages of development, the most reliable answer would be to wait and see which system works best. However, I would argue in favor of a few heuristics that are on my side. Firstly, I would argue that the amount of time for such an adaptation is limited, as the user can basically follow a four-step program: let us assume that the dependent elements x are shown but the user would like to see the elements y.

1. See if y is nominated. If it is not, nominate y and see if the plot changes.
2. See if y is vetoed. If it is, reconsider either the veto or your own preferences.
3. See if x should be vetoed. If it should, implement the veto and see if the plot changes.
4. See where x and y gather their vote. Change the voting patterns at the appropriate place.

At any step, it might become useful to define a new class. However, this class would be quickly implemented because the user could start with few definitions whereas the rigidity of the rule-based system appears to me more vulnerable to modifications.

This also applies to the second part of the argument. Of course, changes might affect certain plots. However, such a negative change is, in my estimate, more likely in a rule-based system, as the voting system cuts the function some slack and allows the user to demonstrate the importance of a certain nomination in a flexible manner.

4.7.4.4 What about numbers?

This is, of course, a good starting point for a new criticism. It is actually a special case of the intransparency argument:

> Even if these numbers, if applied properly, support a useful visualization system, it is too hard to apply them properly. Humans are not good at handling arbitrary numbers and in a complex system, the effect of these numbers is completely intransparent.

I concede that the introduction of the votes is slightly artificial. I would argue, however, that the proper remedy is not admitting defeat or employing a rule-based system but rather to make these numbers less arbitrary by agreeing on certain standards. After a few application, analysis of the resulting problems might yield a good guideline. Complementary to such an approach, an automated system might be able to compensate for human inability. For instance, an automate system might infer appropriate voting patterns by letting the statistician rank different plots which correspond to certain specified aesthetics.

4.7.4.5 What about letting it go?

> I agree that the voting system appears more suitable for automatic visualization than a rule-based process. Regardless, these automatic visualizations are a fruitless endeavour. The independent elements yield a good adpatation of the plot but the dependent elements adress a problem in a very complicated manner that does not arise as much.
>
> In most cases there are only a few different kinds of plots that are employed and it is easier to define them directly and from case to case manually. Such a definition would be improved by integrating independent elements.

Again, the most reliable response to this is to implement both methods, apply them and compare the results. However, I would like to raise a different point at this place: the definition of the dependent elements could very well occur with the aim of building a few kinds of visualizations. This might even be a good starting point for a heuristic. After building such a model, whenever the need of a new visualization arises it should become increasingly simple to adapt the system such that the new combinations fit together. Therefore, the situation that has been described in the counterargument might also serve as a suitable situation for the implementation of a voting system.

Nevertheless, the answer to this counter-argument needs not be binary. We might very well employ dependent elements with voting system in certain contexts whereas a simpler solution might be suitable in other applications.

4.7.5 Lifecycle and further development

As I have repeatedly remarked in the previous paragraphs, I believe further validation of the current approach to be crucial. Thus, `fx_ggplot` in its current implementation is potentially useful but far from stable. In particular, the internal nomination process and investigation of the plot elements are a bit awkward. A proper implementation within the `ggproto` system of `ggplot2` might alleviate these problems. However, determining this need and a suitable implementation requires more practical experience with the voting system.

I will therefore apply the voting system to more databases to determine its strengths and weaknesses. In particular, I will consider possible voting pattern standards and a more suitable implementations of the internals. Another important question in this regard has already been raised in the last chapter: perhaps, aesthetics are not optimally suited to specify the desired plot. For instance, the question whether a certain variable is represented by fill or colour is largely dependent on the layer. Perhaps another mapping would provide better results.

5 | Application

This chapter is concerned with the application of `tectr` to the V-Dem database. At first, it will present the metaframe and discuss the `fx_write` function. The second section will present `fx_info` and `fx_output` whereas the third section will consider `fx_ggplot`.

5.1 The metaframe

The folder "data-raw" contains the creation of the metaframe in detail. I have imported the codebook (Coppedge et al., 2018c) with the help of the `pdftools` package (Ooms, 2018). Subsequently, I have read out different categories of information about the variables. The resulting data frame has been exported by `fx_write_json` which yielded the data frame `mf_revisable`:

```
mf_revisable
```

```
## # A tibble: 909 x 8
##    part_num chapter_num ind_num ind_type name  part_title chapter_title
##       <dbl>       <dbl>   <int> <chr>    <chr> <chr>      <chr>
##  1        2           1       1 D        v2x_~ V-Dem Dem~ V-Dem High-L~
##  2        2           1       2 D        v2x_~ V-Dem Dem~ V-Dem High-L~
##  3        2           1       3 D        v2x_~ V-Dem Dem~ V-Dem High-L~
##  4        2           1       4 D        v2x_~ V-Dem Dem~ V-Dem High-L~
##  5        2           1       5 D        v2x_~ V-Dem Dem~ V-Dem High-L~
##  6        2           2       1 D        v2x_~ V-Dem Dem~ V-Dem Mid-Le~
##  7        2           2       2 D        v2x_~ V-Dem Dem~ V-Dem Mid-Le~
##  8        2           2       3 D        v2x_~ V-Dem Dem~ V-Dem Mid-Le~
##  9        2           2       4 D        v2x_~ V-Dem Dem~ V-Dem Mid-Le~
## 10        2           2       5 D        v2x_~ V-Dem Dem~ V-Dem Mid-Le~
## # ... with 899 more rows, and 1 more variable: main <S3: filepath>
```

In particular, the different information categories have been collapsed into a filepath called "main".

The files themselves have the following structure:

```
[
  {
    "fxInfo_name": "Electoral democracy index",
    "fxInfo_notes": "",
    "fxInfo_ordering": "",
    "fxInfo_project_manager": ["Jan Teorell"],
    "fxInfo_question": "To what extent is the ideal of electoral democracy in its fullest
                        sense achieved?",
    "fxInfo_responses": [],
    "fxInfo_scale": "Interval, from low to high (0-1)",
```

```
    "fxInfo_sources": "v2x_freexp_altinf v2x_frassoc_thick v2x_suffr v2xel_frefair v2x_elecoff",
    "name": "v2x_polyarchy"
  }
]
```

I have only included a few exemplary fields in order to demonstrate that such a file is well-readable by humans.

At this stage, these variables may be modified by hand in order to add additional fields or correct typos.

Afterwards, the metaframe is completed by adding the parameters of `fxGeom`. This results in the `vdem`-object:

```
metaframe(vdem)
```

```
## # A tibble: 2,671 x 32
##    name  fxInfo_name part_num chapter_num ind_num ind_type part_title
##    <chr> <chr>          <dbl>       <dbl>   <int> <chr>    <chr>
##  1 coun~ Country Na~       NA          NA      NA <NA>     <NA>
##  2 year  Year              NA          NA      NA <NA>     <NA>
##  3 v2x_~ Electoral ~        2           1       1 D        V-Dem Dem~
##  4 v2x_~ Electoral ~        2           1       1 D        V-Dem Dem~
##  5 v2x_~ Electoral ~        2           1       1 D        V-Dem Dem~
##  6 v2x_~ Liberal de~        2           1       2 D        V-Dem Dem~
##  7 v2x_~ Liberal de~        2           1       2 D        V-Dem Dem~
##  8 v2x_~ Liberal de~        2           1       2 D        V-Dem Dem~
##  9 v2x_~ Participat~        2           1       3 D        V-Dem Dem~
## 10 v2x_~ Participat~        2           1       3 D        V-Dem Dem~
## # ... with 2,661 more rows, and 25 more variables: chapter_title <chr>,
## #   fxInfo_aggregation <chr>, fxInfo_answer_type <chr>,
## #   fxInfo_citation <chr>, fxInfo_clarification <chr>,
## #   fxInfo_cpp_tag <chr>, fxInfo_cross_coder_aggregation <chr>,
## #   fxInfo_data_release <chr>, fxInfo_historical_clarification <chr>,
## #   fxInfo_notes <chr>, fxInfo_ordering <chr>,
## #   fxInfo_project_manager <list>, fxInfo_question <chr>,
## #   fxInfo_responses <list>, fxInfo_scale <chr>, fxInfo_sources <chr>,
## #   fxInfo_cautionary <chr>, fxGeom_class <chr>, fxGeom_assoc_vars <list>,
## #   fxGeom_limits <list>, fxGeom_trans <list>, fxGeom_veto <list>,
## #   fxGeom_vote <list>, fxGeom_drop <lgl>, fxGeom_pal.colour_fill <list>
```

5.2 fx_info

We can use `vdem` to produce uncomplicated and well-structured descriptive tables:

```
vdem %>%
  select(v2x_polyarchy, v2x_libdem, v2x_partipdem, v2x_delibdem, v2x_egaldem) %>%
  fx_info(c("name", "question", "stats"),
          statistics = list(
            mean = function(x, ...) mean(x, na.rm = TRUE),
            quantile = function(x, ...)
              quantile(x, probs = seq(0.25, 0.75, 0.25),
                       na.rm = TRUE)
            )) %>%
  select(-name) %>%
  fx_output("table", out_format = "markdown")
```

Name	Question	mean	quantile: 25%	quantile: 50%	quantile: 75%
Electoral democracy index	To what extent is the ideal of electoral democracy in its fullest sense achieved?	0.26	0.06	0.17	0.37
Liberal democracy index	To what extent is the ideal of liberal democracy achieved?	0.22	0.06	0.12	0.29
Participatory democracy index	To what extent is the ideal of participatory democracy achieved?	0.19	0.04	0.10	0.29
Deliberative democracy index	To what extent is the ideal of deliberative democracy achieved?	0.23	0.04	0.12	0.37
Egalitarian democracy index	To what extent is the ideal of egalitarian democracy achieved?	0.24	0.07	0.15	0.33

A more vertical structure is given below by the "collapse"-output-form:

```
vdem %>%
  select(v2x_polyarchy) %>%
  fx_info(c("name", "question", "clarification", "stats"),
          statistics = list(
            mean = function(x, ...) mean(x, na.rm = TRUE),
            quantile = function(x, ...) quantile(x, na.rm = TRUE)
            )
          ) %>%
  fx_output("collapse",
            cell_scheme = "{name}: {value}",
            cell_sep = "\n\n",
            out_format = "markdown")
```

Name: Electoral democracy index

Question: To what extent is the ideal of electoral democracy in its fullest sense achieved?

Clarification: The electoral principle of democracy seeks to embody the core value of making rulers responsive to citizens, achieved through electoral competition for the electorate's approval under circumstances when suffrage is extensive; political and civil society organizations can operate freely; elections are clean and not marred by fraud or systematic irregularities; and elections affect the composition of the chief executive of the country. In between elections, there is freedom of expression and an independent media capable of presenting alternative views on matters of political relevance. In the V-Dem conceptual scheme, electoral democracy is understood as an essential element of any other conception of representative democracy — liberal, participatory, deliberative, egalitarian, or some other.

mean: 0.26

quantile: 0%: 0.0073

quantile: 25%: 0.062

quantile: 50%: 0.17

quantile: 75%: 0.37

quantile: 100%: 0.94

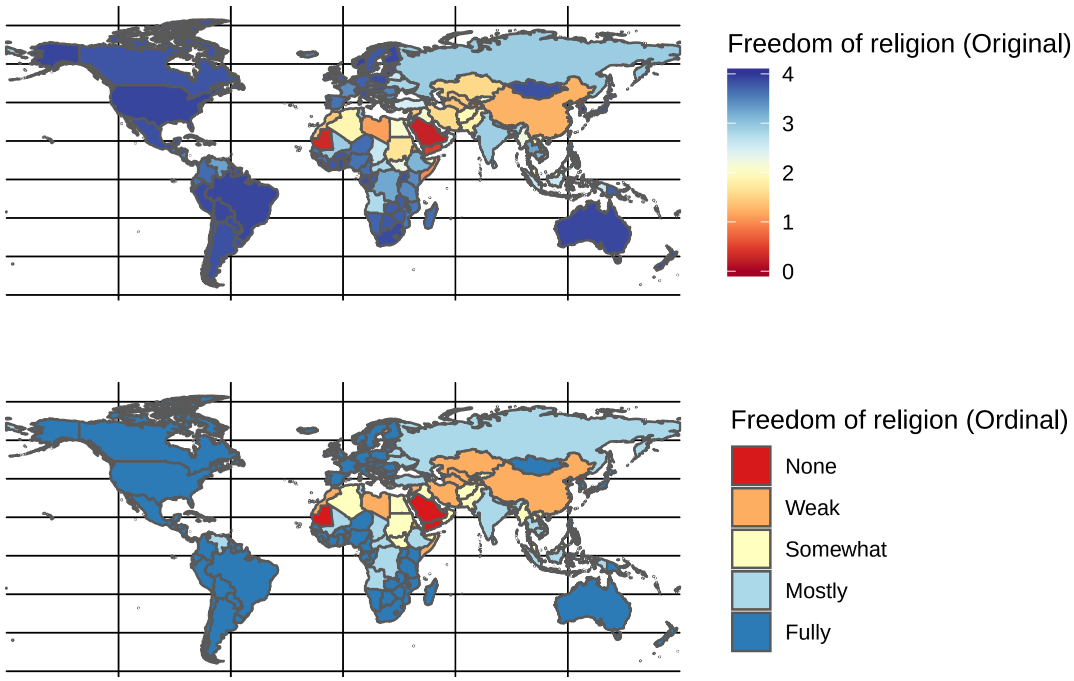

Figure 5.1: The ordinal scale provides a discretized version of the original scale.

5.3 fx_ggplot

As the general features of `fx_ggplot` have already been presented in the previous chapter, I will focus on a few changes due to the adaptations of the metaframe. Firstly, I have provided a consistent set of palettes, as can be seen in figure 5.1.

```
p1 <- vdem %>%
  filter(year == 2015) %>%
  select(v2clrelig_osp, country_name, year) %>%
  vdem_geocode() %>%
  fx_ggplot(aes(geom = geometry, fill = v2clrelig_osp)) +
  my_theme +
  theme_void()

p2 <- vdem %>%
  filter(year == 2015) %>%
  select(v2clrelig_ord, country_name, year) %>%
  vdem_geocode() %>%
  fx_ggplot(aes(geom = geometry, fill = v2clrelig_ord)) +
  my_theme +
  theme_void()

p1/p2
```

Furthermore, we plot a time series with confidence intervals for different countries in figure 5.2.

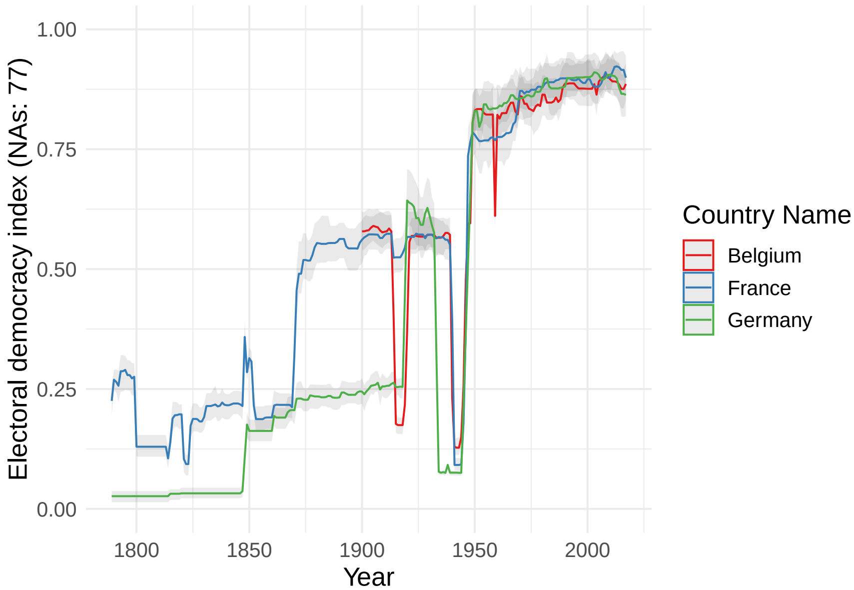

Figure 5.2: A time series of the electoral democracy index for Germany, France and Belgium

```
vdem %>%
  filter(country_name %in% c("Germany", "France", "Belgium")) %>%
  select(country_name, year, contains("v2x_polyarchy")) %>%
  fx_ggplot(aes(x = year, y = v2x_polyarchy, colour = country_name)) +
  my_theme
```

If we include all countries, `fx_ggplot` prevents overplotting, as in figure 5.3.

```
vdem %>%
  select(country_name, year, contains("v2x_polyarchy")) %>%
  fx_ggplot(aes(x = year, y = v2x_polyarchy)) +
  my_theme
```

Of course, we can also plot two democracy indices against each other, see figures 5.4 and 5.5.

```
vdem %>%
  select(v2x_polyarchy, v2x_libdem) %>%
  fx_ggplot(aes(x = v2x_polyarchy, y = v2x_libdem)) +
  my_theme
```

```
vdem %>%
  filter(country_name %in% c("Germany", "France")) %>%
  select(v2x_polyarchy, v2x_libdem, country_name, year) %>%
  fx_ggplot(aes(x = v2x_polyarchy, y = v2x_libdem,
                colour = year),
            facet_vars = vars(country_name)) +
  my_theme
```

The time series work for the low-level indices, as well, as can be seen in figure 5.6.

```
vdem %>%
  filter(country_name %in% c("Germany", "France", "Belgium")) %>%
  select(contains("v2meharjrn"), country_name, year) %>%
```

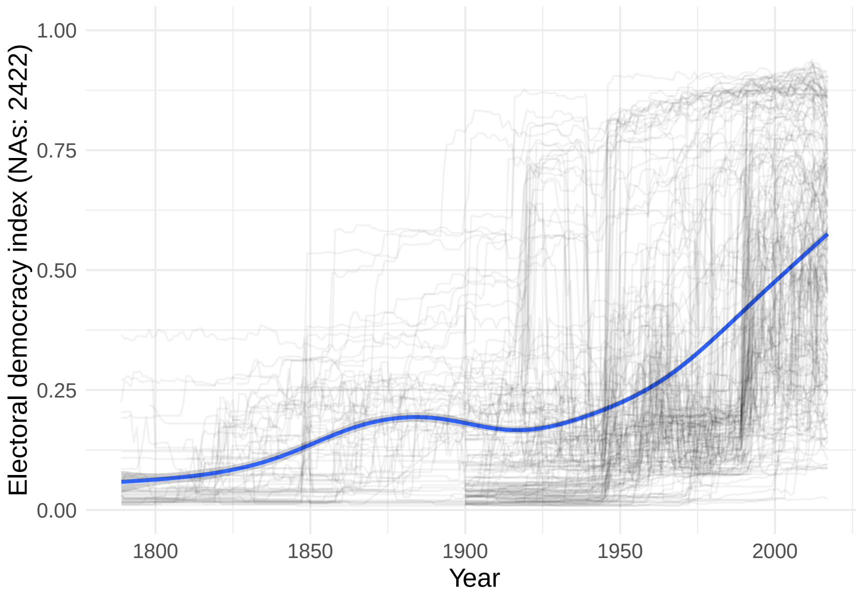

Figure 5.3: A time series of the electoral democracy index for all countries

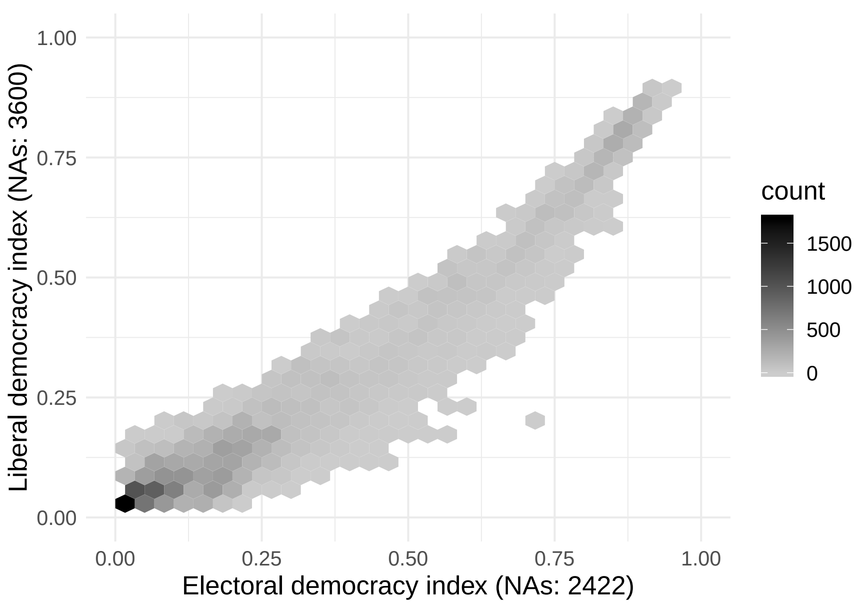

Figure 5.4: The electoral democracy index and the liberal democracy index.

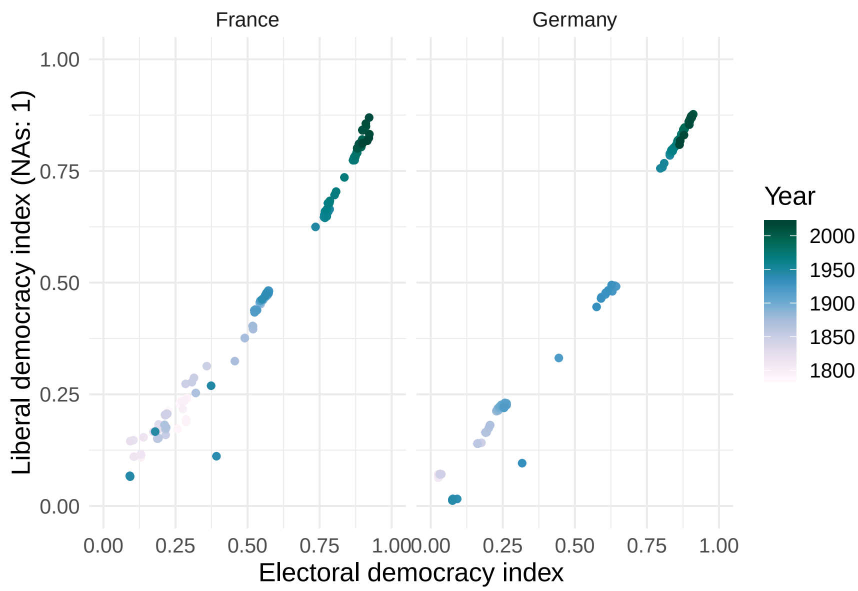

Figure 5.5: Electoral and liberal democracy index in Germany, France and Belgium over time

```
  fx_ggplot(aes(x = year, y = v2meharjrn_osp, colour = country_name)) +
  my_theme
```

Figure 5.7 shows the discretized version.

```
vdem %>%
  filter(country_name %in% c("Germany", "France", "Belgium")) %>%
  select(contains("v2meharjrn"), country_name, year) %>%
  fx_ggplot(aes(x = year, y = v2meharjrn_ord, colour = country_name)) +
  my_theme
```

The time series still struggles with the missing values which occur in the election variables. Figure 5.8 demonstrates that the step plot is correctly chosen. However, it is complicated to omit the missing values without breaking anything. This leads to an uninformative plot at the present version. An implementation of `fx_ggplot` which is more native to `ggplot2` would most likely lead to better results.

```
vdem %>%
  filter(country_name %in% c("Germany")) %>%
  select(contains("v2elvotbuy"), country_name, year) %>%
  fx_ggplot(aes(x = year, y = v2elvotbuy_osp)) +
  my_theme
```

Finally, we plot a discretized variable against a high-level index in figure 5.9.

```
vdem %>%
  select(contains("v2meharjrn"), v2x_polyarchy, country_name, year) %>%
  fx_ggplot(aes(x = v2meharjrn_ord, y = v2x_polyarchy)) +
  my_theme +
  theme(axis.text.x = element_text(angle = 30))
```

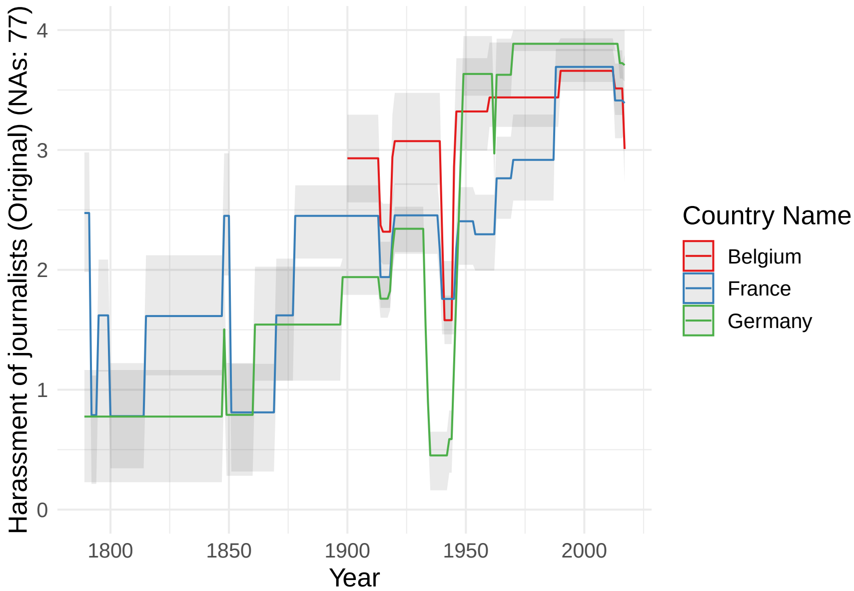

Figure 5.6: The electoral democracy index and the liberal democracy index.

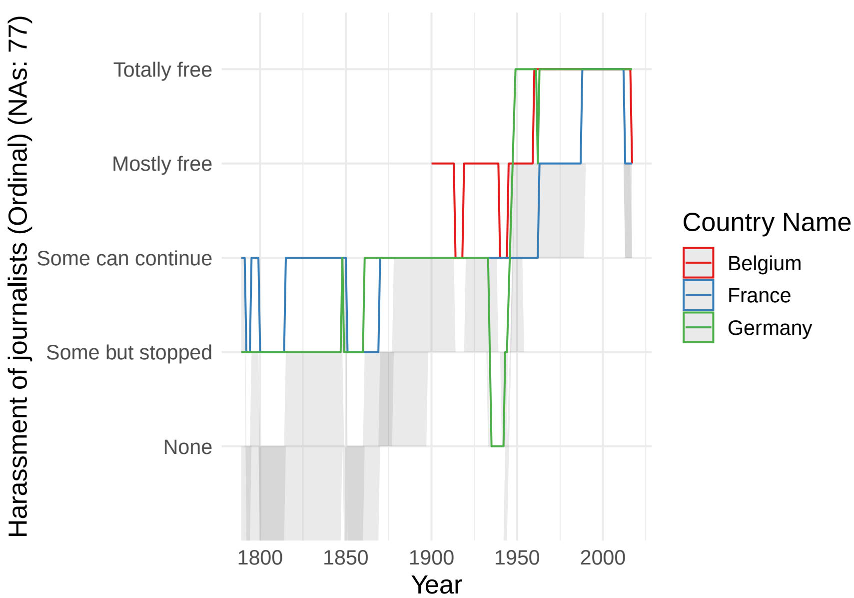

Figure 5.7: Harassment of journalists in Germany, France and Belgium over time (discretized)

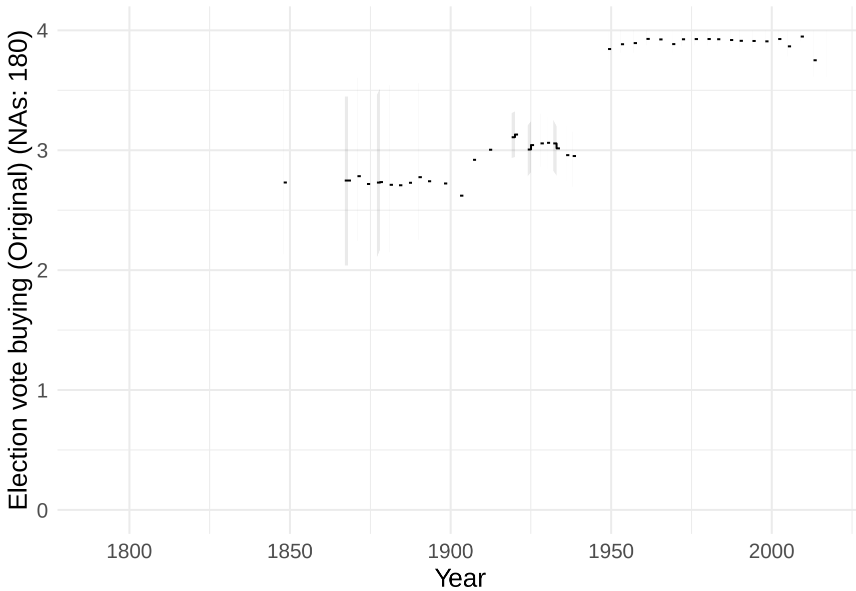

Figure 5.8: Election vote buying in Germany over time

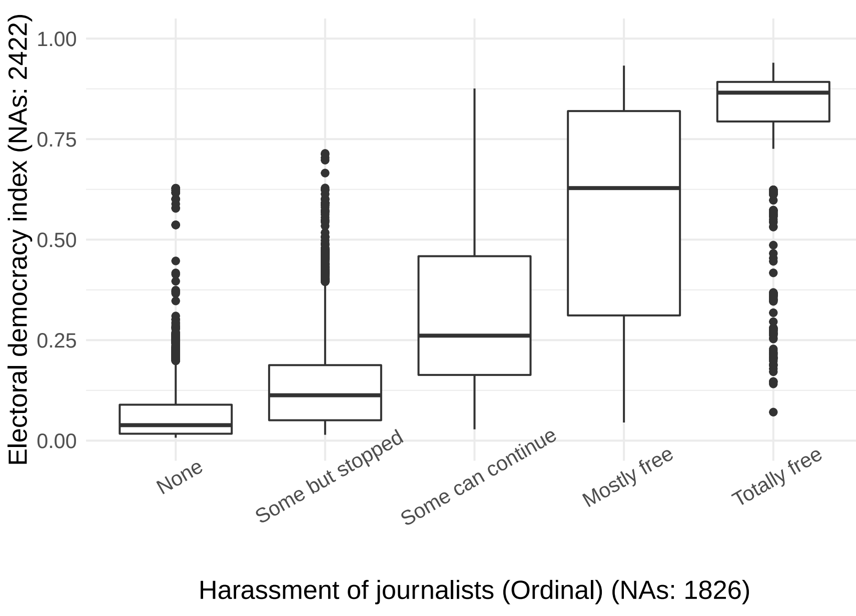

Figure 5.9: The electoral democracy index and the liberal democracy index.

6 | Summary

In this Bachelor's Thesis, I have presented the package `tectr` which intends to assist in data analysis and visualization by allowing the user to set default values. I have discussed reasons why this is useful and described a framework which I then implemented. At last, I presented an example of application to the Varieties of Democracy dataset.

As my next steps regarding the development of `tectr`, I will implement a report function for `fx_output` which assists in creating reports of all relevant variables. Furthermore, I will revise `fx_ggplot` such that it is more native to the internal structure of `fx_ggplot`.

A | Packages

This Bachelor's Thesis includes two packages: `tectr` and `vdem.tectr`. Both are included in the supplementary material. It contains:

- binary and source builds of the packages
- the original R projects
- the source code for this Bachelor's thesis

`tectr` can additionally be found on Github:

```r
# install.packages("devtools")
devtools::install_github("sflippl/tectr")
```

Bibliography

Box, G. E. P. and Cox, D. R. (1964). An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2):211–252.

Coppedge, M., Gerring, J., Knutsen, C. H., Lindberg, S. I., Skaaning, S.-E., Teorell, J., Ciobanu, V., and Olin, M. (2018a). V-Dem Country Coding Units v8. Technical report, Variaties of Democracy (V-Dem) Project.

Coppedge, M., Gerring, J., Knutsen, Carl Henrik Lindberg, S. I., Skaaning, S.-E., Teorell, J., Altman, D., Bernhard, M., Fish, S. M., Cornell, A., Dahlum, S., Gjerløw, H., Glynn, A., Hicken, A., Krusell, J., Lührmann, A., Marquardt, K. L., McMann, K., Mechkova, V., Medzihorsky, J., Olin, M., Paxton, P., Pemstein, D., Pernes, J., von Römer, J., Seim, B., Sigman, R., Staton, J., Stepanova, N., Sundström, A., Tzelgov, E., Wang, Y.-t., Wig, T., Wilson, S., and Ziblatt, D. (2018b). V-Dem Country-Year Dataset v8.

Coppedge, M., Gerring, J., Knutsen, Carl Henrik Lindberg, S. I., Skaaning, S.-E., Teorell, J., Altman, D., Bernhard, M., Fish, S. M., Cornell, A., Dahlum, S., Gjerløw, H., Glynn, A., Hicken, A., Krusell, J., Lührmann, A., Marquardt, K. L., McMann, K., Mechkova, V., Medzihorsky, J., Olin, M., Paxton, P., Pemstein, D., Pernes, J., von Römer, J., Seim, B., Sigman, R., Staton, J., Stepanova, N., Sundström, A., Tzelgov, E., Wang, Y.-t., Wig, T., Wilson, S., and Ziblatt, D. (2018c). V-Dem Codebook v8.

Komsta, L. and Novomestky, F. (2015). moments: Moments, cumulants, skewness, kurtosis and related tests.

Ooms, J. (2014). The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects. arXiv:1403.2805 [stat.CO].

Ooms, J. (2018). pdftools: Text Extraction, Rendering and Converting of PDF Documents.

Pebesma, E. (2018). sf: Simple Features for R.

Pemstein, D., Marquardt, K. L., Tzelgov, E., Wang, Y.-t., Krusell, J., and Miri, F. (2018). The V-Dem Measurement Model: Latent Variable Analysis for Cross-National and Cross-Temporal Expert-Coded Data.

R Core Team (2018). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Ribeiro Jr, P. J. and Diggle, P. J. (2018). geoR: Analysis of Geostatistical Data.

Soanes, C. (2012). 'He', 'he or she', 'he/she', 's/he', or 'they'.

Weidmann, N. B. and Gleditsch, K. S. (2010). Mapping and Measuring Country Shapes. The R Journal, 2(1):18–24.

Weidmann, N. B., Kuse, D., and Gleditsch, K. S. (2010). The geography of the international system: The CShapes dataset. International Interactions, 36(1):86–106.

Wickham, H. (2011). testthat: Get Started with Testing. The R Journal, 3:5–10.

Wickham, H. (2014). Tidy Data. Journal of Statistical Software, 59(10):1–23.

Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York.

Wickham, H. (2018). stringr: Simple, Consistent Wrappers for Common String Operations.

Wickham, H., Danenberg, P., and Eugster, M. (2018a). roxygen2: In-Line Documentation for R.

Wickham, H., Hester, J., and Chang, W. (2018b). devtools: Tools to Make Developing R Packages Easier.

Xie, Y. (2016). bookdown: Authoring Books and Technical Documents with R Markdown. Chapman and Hall/CRC, Boca Raton, Florida.

Xie, Y. (2018). bookdown: Authoring Books and Technical Documents with R Markdown.