Analysis and Graphical Representation of Health and Labour Force Participation among the Elderly in Europe

Humboldt-Universität zu Berlin
School of Business and Economics
Ladislaus von Bortkiewicz Chair of Statistics

Statistical Programming Languages
Winter 2017/18

Seminar Paper by

Claudia Günther, Phi Nguyen, Julian Winkel
576419, 526624, 562959

Berlin, 2018-03-15

List of Tables

List of Figures

Abbreviations

xx xxxxxxxxxx

Contents

1 Introduction

- relevance of exploring relationship between health and labour force participation due to changing demographic in Europe
- cite relevant study about ageing
- few sentences about relevant papers exploring relationship -> use paper from DIW
- very short literature overview on relationship between health and labour force participation
- share data set as rich data set for this purpose: 2 sentences about it
- introduction of journal article
- our approach: replicate results an enrich analysis
- especially: introduce graphical visualization tools for descriptive statistics -> ease interpretation of variables
- our aim: write code in a way that allows the user to work with easySHARE data set, even when working on different question

2 Data Cleaning and Manipulation

The original SHARE data set used by Kalwij and Vermeulen (2005), released in spring 2005, contains data on the life circumstances of eligible members from approximately 18,000 households. In order to be eligible for participation in SHARE, at least one household member must have been born in or before 1954. The SHARE survey contains a variety of different health, social, and socioeconomic indicators such as labor force participation or household composition. SHARE contains data from eleven countries: Austria, Belgium, Denmark, France, Germany, Greece, Italy, the Netherlands, Spain, Sweden and Switzerland. This cross-national, multidisciplinary, and longitudinal nature of the SHARE data set provide an abundance of unique and useful opportunities for analysis.

2.1 Theory and Design

The easySHARE data set, released in spring 2017, is the data set for which we will be focusing on in this paper. It is a panel data set of 108 variables of more than 108 variables of more than 100,000 individuals covering data from six survey waves carried out between 2004 and 2015. As can be derived from its name, it is designed to be a simplified version of the broader SHARE data set. As a result, the data set includes slightly different observations and does not provide the same level of granularity in variables broader SHARE data set, which we shall show below. Additionally, since we are only concerned with a smaller subset of observations (from the initial wave of data), we will remove observations that are no longer relevant for our study.

The rest of this section describes our process of data transformation, aggregation, and cleaning , not only to match as closely as possible the initial SHARE data set, but also to convert our data sets into a more accessible state for future analyses.

2.2 Implementation

The `read.and.clean()` function is designed to take the raw easySHARE data set file, which is provided in an `.rda` file format, and convert that into a list of data frames. We split the data set into list items because future analyses involve either summary statistics on the entire data set or regressions conducted on a subset of the data set. The function additionally accepts a wave number as an argument. Although we are only concerned with the initial wave for our study, this additional parameter allows us to conduct supplementary panel studies if we choose to do so later. The function only runs specifically with the raw easySHARE data set (`easySHARE_rel6_0_0.rda`), which is the default `dataset` argument. The `dataset` argument may alternatively be specified to point to the directory where that data set is located. Otherwise it is assumed that it is located in the root directory.

We begin by initializing and downloading the packages (if necessary) required for this function. Much of the funciton utilizes the functions in the `tidyverse` family of functions developed by Hadley Wickham, namely `dplyr` and `magrittr`. The `dplyr` package is designed for accessible and uncomplicated data manipulation, and it works specifically with data frames. The package provides a consistent set of verbs that do a small thing very well, such as aggregation, selection, filtering, or creation of new variables. The `magrittr` package expands upon this further with the pipe operator `%>%`, which enables the chaining of smaller functions to create something more complex. Furthermore, the pipe operator dramatically improves readability by reading

chains left to right as opposed to inside-out. This is especially useful when working with nested functions, which can be deconstructed into a series of chained commands.

```
neededPackages = c("dplyr", "magrittr", "infuser", "countrycode")
allPackages    = c(neededPackages %in% installed.packages()[,"Package"])

    if (!all(allPackages)) {
        missingIDX = which(allPackages == FALSE)
        needed     = neededPackages[missingIDX]
        lapply(needed, install.packages)
    }

invisible(lapply(neededPackages, function(x) suppressPackageStartupMessages(
    library(x, character.only = TRUE))))
```

We continue with basic filtering to select only variables that are pertinent to our study. For the purposes of our work, we are only concerned with the initial wave and adults between the ages of 50 and 64. Then we encode missing values. Although the overall response rate in the SHARE are comparably high, the data set still has numerous missing values. The reason for this is due to the fact that the study was carried out on a cross-national scale, with some national survey institutions deciding not to participate in all survey modules. The reason for the missing values are documented well in the "Guide to easySHARE release 6.0.0" and specifically coded. For example, the numbers -13 and -14 refer to "not asked in this wave" and "not asked in this country". Since this coding scheme is not useful for the purpose of our analysis, we recode all of the missing values as "NA".

```
# Encode missing values according to SHARE data set guidelines
a = c(-2, -3, -4, -7, -9, -12, -13, -14, -15, -16)
b = c("tocheck","implausible", "tocheck", "uncoded", "notApplicable",
      "dontKnow", "notAskedWave", "notAskedCountry", "noInformation",
      "noDropOff")
missing.value.codes = data.frame(a,b)

# Find NA locations and declare them as such
df.decl = apply(dat, 2, function(z) {
    na.loc    = which(z %in% a)
    z[na.loc] = NA
   return(z)
})
```

We finish converting the raw data set into a final data frame by adding readable country names and transforming raw variables into useful numeric or logical variables as described in the paper. We use the `countrycode` package to change the country codes into human-friendly country names. There are minor differences between the data set that Kalwij and Vermeulen (2005) use and the easySHARE data set, which means we must make certain assumptions in our data cleaning process. For example, certain questions in the survey are an aggregated total of a series of questions, such as the types of chronic diseases the subjects suffer from. Whereas

the original authors have access to the specific diseases, we are only provided a count of the total number. Therefore we replace certain variables with the closest relative we have available. In other cases, multiple variables are provided in our data set but it is ambiguous which one the original authors use. An interesting example is employment status: our data set contains one variable for type of employment (full-time, part-time, or not working) and one for labor hours (the number of hours worked by the survey respondent). However, some respondents gave answers that ran counter (for example, saying they worked full-time but only working 20 hours a week). The differences in answers can be attributed to varying interpretations of survey responses.

A simplified example of the final script is shown below.

```
df.out       = df %>%
        dplyr::left_join(country_data, by = c("country_mod" = "iso3n")) %>%
        dplyr::filter(iso3c %in% country_list) %>%
        dplyr::mutate(country       = factor(country.name.en),
                      gender        = factor(ifelse(female, "FEMALE", "MALE")),
                      age           = factor(floor(age)),
                      edu_low       = isced1997_r %in% c(0, 1),
                      edu_high      = isced1997_r %in% c(5, 6),
                      h_chronic     = chronic_mod,
                      h_overweight  = bmi2 == 3,
                      labor_ft      = ep013_mod >= 32,
                      labor_hrs     = floor(ep013_mod)) %>%
        dplyr::select(country, gender,
                      starts_with("h_"),
                      starts_with("labor_")) %>%
        na.omit() %>%                              # remove missing values
        set_rownames(NULL)                         # reset row numbering
```

Our last step involves splitting the data sets, standardizing any numeric variables, converting the data frames into a model matrix, then storing the resultant data frames into a list that is the final object returned by the function.

The data frames are split into separate country and gender splits, since we will create separate regressions for each of these splits. Then we apply a custom-built standardization function, which finds standardizes all numeric variables and ignores all other variables. Finally, we use a custom-built function that takes advantage of the base `model.matrix()` function to convert all split data frames into a model matrix. This useful function converts all logical vectors into dummies to replicate the process used in the paper. A corollary benefit is that the resultant data frames can be directly used in R's in-built regression functions.

```
standardize.df = function(df) {
    idx = sapply(df, is.numeric)
    idx = seq(1:length(idx))[idx]

    df.reg = df %>%
        mutate_at(.vars = vars(idx),
                  .funs = standardize) %>%
```

```
        mutate(labor_participation = !labor_np) %>%
        dplyr::select(country, gender, age,
                      h_chronic, h_adla, h_obese, h_maxgrip,
                      edu_second, edu_high, children, couple,
                      labor_participation)

    return(df.reg)
}

splits    = split(df.out, f = list(df.out$country, df.out$gender),
                  drop = TRUE)
df.reg    = standardize.df(df.out)
df.splits = lapply(splits, standardize.df)

dummify = function(df) {
    df = df %>%
        dplyr::select(-country, -gender)
    model      = ~ 0 + .
    new.df     = model.matrix(model, df)
    new.df     = data.frame(new.df)
    return(new.df)
}

df.splits = lapply(df.splits, dummify)
```

2.3 Empirical Results

The validity of our `read.and.clean()` can be verified by comparing the overall results when running the `read.and.clean()` function with the ones presented in the original paper. As the function is called, messages are printed to the console to show the status of the data reading process. Messages additionally show the names of the items in the list so that users can access the specific data set of their choosing.

In the original paper, the authors retain a sample of 12,237 observations, which is slightly smaller than our retained observations of 12,689. However, at the individual country level, some peculiarities in number of observations emerge. Namely, we have almost twice as many observations in France, while having relative sparsity in Austria. This again is likely due to differences in data set and in the data gathering process. We will discuss more details about the summary statistics in future sections.

```
source('Scripts/ReadAndClean.R')
datasets = read.and.clean(dataset = "easySHARE_rel6_0_0.rda", wav = 1)
```

```
## Loading data set...
## Selecting values only from Wave 1 and between ages 50 and 64.
## Removing missing values.
```

```
## Rows removed:   276047
## Rows remaining: 12689
##
## Final output is a list containing 3 data.frames:
## `df.out` is a data.frame containing variables for calculating summary statistics.
## `df.reg` is a data.frame containing standardized variables for regression.
## `df.splits` splits the regression data.frame into individual data.frames for each country/
##         gender split. 22 data.frames are contained in this list.
```

Lastly, when an out-of-bounds value for the wave is selected, an error message occurs. Note that only six waves or cross-sections are included in our data set.

```
source('Scripts/ReadAndClean.R')
datasets = read.and.clean(dataset = "easySHARE_rel6_0_0.rda", wav = 99)
```

```
## Loading data set...
```

```
## Error in read.and.clean(dataset = "easySHARE_rel6_0_0.rda", wav = 99): Out of bounds.
##                                 Select a value between 1 and 6.
```

3 Multidisciplinary and crossnational summary statistics

JULIAN

3.1 Theory and Design

3.2 Implementation

3.3 Empirical Results

4 Crosssectional probit regression

JULIAN

4.1 Theory and Design

4.2 Implementation

4.3 Empirical Results

5 Wald Test

CLAUDIA

5.1 Theory and Design

5.2 Implementation

5.3 Empirical Results

6 Counterfactual exercise

CLAUDIA

6.1 Theory and Design

6.2 Implementation

6.3 Empirical Results

7 Graphical Representation

In this section, we want to introduce novel graphical elements not present in the paper to augment the existing summary statistic tables. We believe that the original paper's lack of any graphical details, either for summarizing valuable data points or for exploratory analysis, hampers a reader's ease of understanding. Tables, although useful for organizing and displaying many discrete data points, lack both detail and decipherability. Furthermore, tables offer no visibility into distributions inherent to the data. We believe that a necessary and important enhancement to the paper would be to include graphs that allow readers to view distributions, analyze trends, and compare data points across groups or geographies.

7.1 Theory and Design

An increasingly commonly used method for representing country-level data graphically is with choropleth maps. Choropleth maps shade or pattern areas within a defined geographic region (such as a country, state, or city) in relation to a data variable. A coloring progression or gradient is used to show value levels within each region. They are useful for identifying values that are associated with a given geographic location and quickly comparing those values across different regions. There are a multitude of great packages in R, along with open source shape files that contain the geographic coordinates to draw these maps, to aid in building choropleth maps.

However, we opt not to use choropleths, instead focusing on a novel approach called tile grid maps. Rather than using actual geographic borders, countries are reduced to a uniform shape and size (in our case, a simple square) in order to ensure equal visual weight. This is done to avoid visual imbalances inherent to choropleth maps: larger geographic regions appear to have more value, influencing a reader's perception on the relative importance of such a region. Additionally, sparsely populated areas may also have a larger visual emphasis. We use tile grid maps since relative population sizes are of little importance to us, but we want to focus rather on comparing country data on variables that are either percentages or have been normalized or fit on the same scale.

Tile grid maps are not without fault, however. Europe does not contain smooth borders like those found in the United States, hence it can be difficult to draw a comparable representation of Europe with grids alone. Nevertheless, the general shape of Europe can be quickly perceived by the reader, which will be shown in the empirical results section.

Additionally, we choose to look at distributions of certain variables, rather than just looking at the mean or median values. By looking at the overall spread of each numeric data variable, we are able to gather a more holistic picture of the nature of the data variable.

7.2 Implementation

We develop two new graphics-oriented functions to generate both graphs listed above. The first is the tile grid maps, defined in the function `health.gridmap()`. Instead of just generating a single tile grid map, we create facets of tile grid maps, where the facetting is determined by the user. Either they can choose to split by gender (in which two grid maps will be displayed side-by-side) or by age group (in which three graphs will be

shown). This facetting method allows readers to distinguish differences between countries as well as within countries. For example, in the same view, we can see how the hours worked per week differ between Germany and Spain, while also seeing how labor hours generally decrease for all countries as groups age over time.

In order to draw the grid map, we must first draw the coordinates. This is done in a simple Excel spreadsheet to align each country with a specific point on a two-dimensional grid. Rather than sourcing this Excel file each time, we load the file once, then use the `dput` function to replicate the structure of the table (this is not shown in the script). This structure is then defined explicitly within the function.

We define a "base" plotting function that draws the initial outline of the tile grid, using the `ggplot2` package. Additional plotting parameters are added later based on the function parameters passed. This is an advantage of using `ggplot2`: we can define the initial plot, then `layer` on additional graphical elements as needed, such as dynamic titles or flexible scales.

```
plot_function = function(df) {

    p = ggplot(data = df, aes(x = X, y = Y)) +
        geom_tile(aes(fill = get(xvar), color = ""),
                  color = "white", size = 0.6) +
        geom_text(aes(label = iso3c), color = "white") +
        geom_text(aes(label = round(get(xvar), 2)), vjust = 2,
                  color = "white", size = 3, na.rm = TRUE) +
        coord_fixed(ratio = 1) +
        theme_minimal() +
        theme(axis.line        = element_blank(),
              axis.text        = element_blank(),
              axis.title       = element_blank(),
              panel.background = element_blank(),
              panel.grid       = element_blank(),
              legend.position  = "bottom") +
        theme(plot.title    = element_text(size=16),
              plot.subtitle = element_text(size=10,
                                           color = "#7F7F7F"))
    return(p)
}
```

We opt for color-blind safe colors, using a sequential gradient color scheme, to enable maximal readability. The range of these color scales is always set to range from the minimum to maximum values between all facets of data. This is done to ensure that there are no values that fall out of bounds, as well as to apply the same color scale for all graphs in view.

```
minval = min(df.f[[xvar]], df.m[[xvar]], na.rm = TRUE)
maxval = max(df.f[[xvar]], df.m[[xvar]], na.rm = TRUE)
```

If a logical variable is selected, we display the percentage of survey respondents for which that value is true. If a numeric variable is selected, we display the average value within survey respondents. The final output is stored as a `grob`, an element from the `gridExtra` package, which allows us to organize mutliple graphs

side-by-side in a single window. We can either draw the final output using the `grid.draw()` function or we can save to disk using the `ggsave()` function.

```
example5 = health.gridmap('labor_hrs', 'age')
grid.draw(example5)
ggsave("Output/gridmap_laborhrs_byage.png", plot=example5, width=12,
       height=8, units="in")
```

The second function creates a series of individual bar charts, split by a collection of selected genders and countries, for a selected numeric variable. A vector of countries and/or genders can be passed as arguments to the function, in the event a user only cares about a subset of the overall data set. In the event no vector of countries and/or genders are supplied, the entire data set is used.

In some cases, outliers in the data set may skew the view of the graphic. Therefore there is an additional default parameter to remove outliers.

```
# Remove outliers (outside 1.5 IQR of 25% and 75% percentiles)
rm.outliers = function(x, na.rm = TRUE) {
    qnt = quantile(df.out[[x]], probs = c(0.25, 0.75), na.rm = na.rm)
    out = 1.5*IQR(df.out[[x]], na.rm = na.rm)

    new.out = df.out
    new.out[(df.out[[x]] < qnt[1] - out), ] = NA
    new.out[(df.out[[x]] > qnt[2] + out), ] = NA

    new.out = new.out[complete.cases(new.out), ]
    return(new.out)
}

if (remove.outliers) df.out = rm.outliers(x = xvar)
```

In addition, bars indicating the average over the entire data set are overlaid on top of each bar chart. This is to enable readers to see how similar or different one distribution is compared to the average. We do this by calculating the distribution of the entire data set, then creating a dummy data frame with all possible combinations of gender and country. We then use this dummy data frame as an additional geometric argument in ggplot.

```
total.dist = table(df.out[[xvar]])/nrow(df.out)

total        = expand.grid(countries, gen, sort(unique(df.out[[xvar]])))
names(total) = c("country", "gender", xvar)
total        = arrange(total, country, gender, get(xvar))
total$value  = rep(total.dist, times = length(countries)*length(gen))
```

Finally, to prevent overcrowding in the view, we decide on the optimal way to arrange the graphs. If only a single gender is selected, we organize the graphs so that only four columns are shown. If all genders are selected, the graphs are organized so that gender is along the y-axis and countries are along the x-axis.

```
if (length(gen) == 1) {
    plot.bar = plot.bar +
        labs(subtitle = paste0("Faceted by country. Gender selected is ", gen,
        ". Distribution of the entire data set shown is overlaid on each graph.")) +
        facet_wrap(~ country, ncol = 4)
} else {
    plot.bar = plot.bar +
        facet_grid(gender ~ country)
}
```

7.3 Empirical Results

We can test the validity of each of the graphics by generating a few sample outputs. For the tile grid maps, we include some stopping conditions to ensure correct facetting and/or to ensure that only a numeric or logical variable is supplied. A similar sequence of error messages is shown for the health distribution maps when the variable is not numeric, the countries supplied are not contained within the data set, or the correct genders are not supplied.

```
source("Scripts/ReadAndClean.R")
source("Scripts/Graphics.R")
datasets = read.and.clean(dataset = "easySHARE_rel6_0_0.rda")

#Only keep relevant data sets
df.out = datasets$df.out

health.gridmap(xvar = 'country', facetting = 'gender')
```

```
## Error in health.gridmap(xvar = "country", facetting = "gender"): 'xvar' must be numeric
##         or logical
```

```
health.gridmap(xvar = 'h_obese', facetting = 'country')
```

```
## Error in health.gridmap(xvar = "h_obese", facetting = "country"): 'facetting' variable
##         must be either 'gender' or 'age'
```

```
health.distribution(xvar = 'age', gen = 'MALEEEEEEE')
```

```
## Error in health.distribution(xvar = "age", gen = "MALEEEEEEE"): 'gender' must be one of
##              'all', 'MALE', or 'FEMALE'
```

```
health.distribution(xvar = 'labor_pt', gen = 'FEMALE', countries = 'all')
```

```
## Error in health.distribution(xvar = "labor_pt", gen = "FEMALE", countries = "all"): 'xvar'
##                                         must be numeric
```

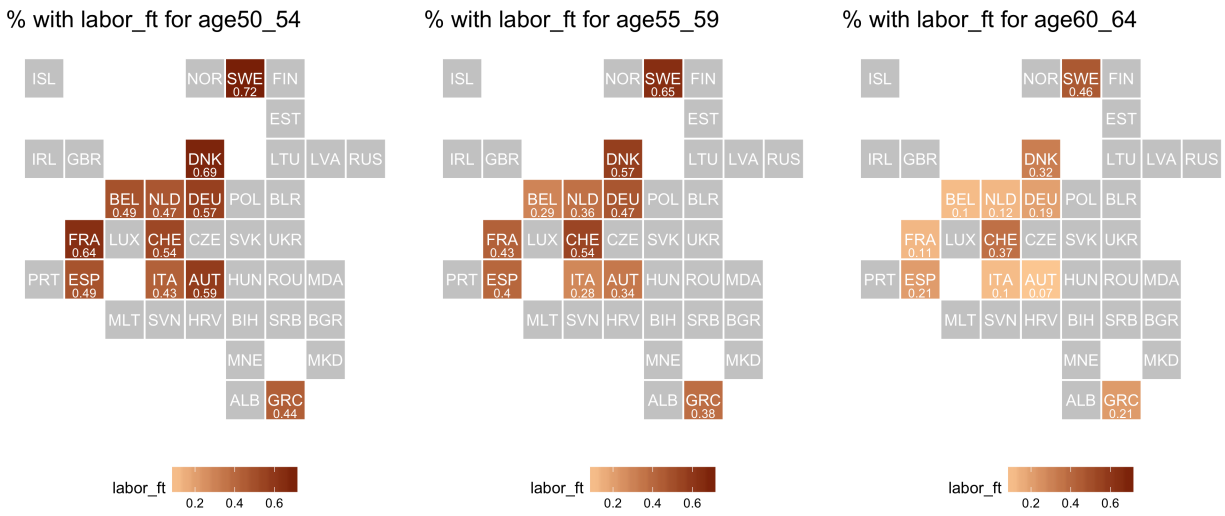
```
health.distribution(xvar = 'age', gen = 'FEMALE', countries = 'China')
```

```
## Error in health.distribution(xvar = "age", gen = "FEMALE", countries = "China"): 'countries'
```

```
##                  must be 'all' or a character vector containing countries
##                  in the `df.out` data frame
```

We show an examples of correct output below. Here, we want to examine average labor hours per week, facetted by age group. As we can see in this example, the percentage of the population that works full time generally decreases for all countries as age increases. However, in countries like Sweden and Switzerland, a higher percentage continue to work even in old age.

```
health.gridmap(xvar = 'h_obese', facetting = 'country')
```

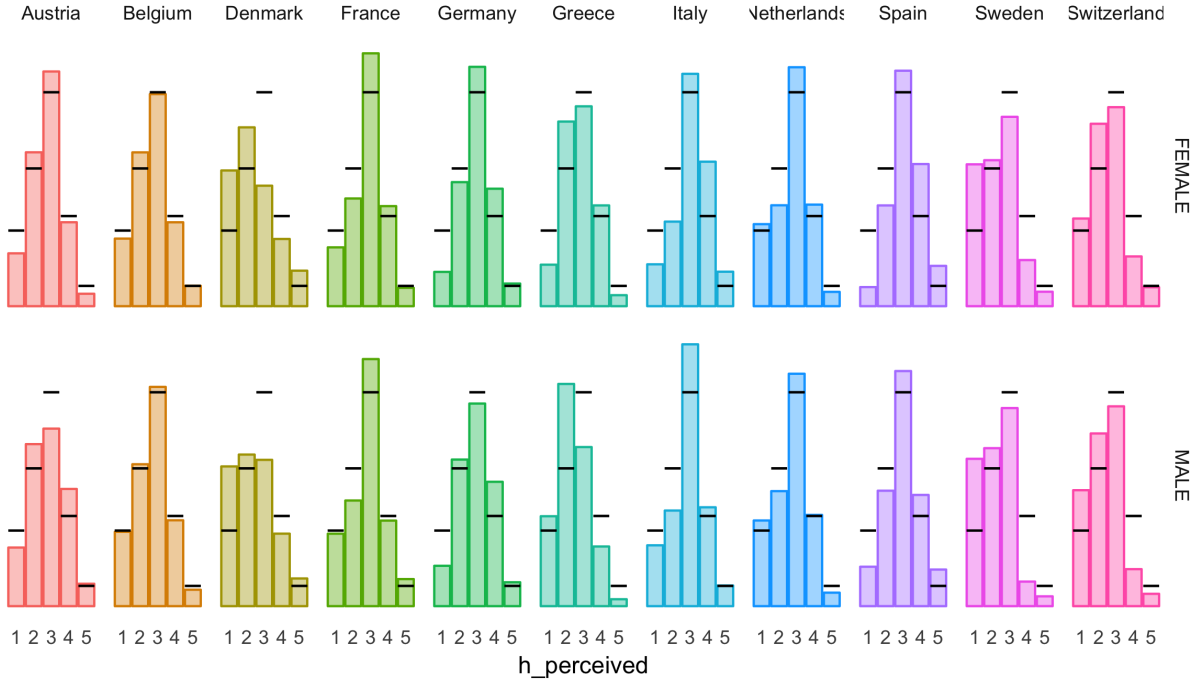


Finally, we display an example of the health distribution map. Here, we seek to evaluate the spread of self-perceived health among survey respondents after removing outliers. The black line indicates the overall distribution for the entire data set. As we can see, there are not too many differences between male and female distributions within countries. However, countries like Spain tend to believe they are healthier than average, whereas the opposite effect is true in Switzerland.

```
health.distribution(xvar = 'h_perceived', remove.outliers = FALSE)
```

Distribution of h_perceived by Country

Faceted by country & gender. Distribution of the entire data set shown is overlaid on each graph.



8 Conclusion

9 References

Declaration of Authorship

We hereby confirm that we have authored this Seminar paper independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such.

Berlin, 2018-03-15

Claudia Günther, Phi Nguyen, Julian Winkel