The Rsocialdata package: Handling survey data in R

Emmanuel Rousseaux[a], Danilo Bolano[a] and Gilbert Ritschard[a]

[a] NCCR LIVES
Institute for Demographic and Life Course Studies
University of Geneva, Switzerland
emmanuel.rousseaux@unige.ch

XXVII IUSSP International Population Conference
26 to 31 August 2013 – Busan, Republic of Korea

Document version: 1.0.1

Abstract. In this article we introduce the Rsocialdata package. This package provides researchers in social sciences with high-level tools for handling survey data in R. As a key feature it allows to store, document and share complex survey data, as panel data and network data. It provides tools for efficiently exploring and recoding data, allowing researchers to proceed more quickly to the task of analysis. Finally, the software puts forward a new methodology for dealing with statistical methods. Adopting a programming syntax especially designed for scientists in social sciences and presenting outputs in PDF files with a "ready-to-publish" formatting, this methodology aims at focusing more quickly on the analysis. Furthermore, printing all parameters used to fine-tune a given statistical method together with the corresponding results facilitates reproducible research. After an introduction to the design of the software, we present a full user guide with various examples based on the Swiss Household Panel data, allowing the reader to become autonomous with the software.

1 Introduction

Population studies strongly rely on survey data. In order to face with the increasing complexity and rigor of the current research studies, the structure of the databases has become in the last decades more and more complex, as longitudinal data, network data, spatial data, etc. The extensiveness volumes of structured data complicate the task of both documenting and manipulating data. This is the reason why there is a need for specific tools to assist the user in handling these complex data. The Rsocialdata software is an effort in this direction. It aims at providing a framework for handling survey data in R, especially network and biographical data. More precisely, the software aims at facilitating the management of survey data by providing researchers in social sciences with high-level tools for storing, documenting, exploring, recoding and sharing survey data in a secure and efficient way. This initiative, conducted within the NCCR LIVES project, targets mainly life course studies and especially data sets collected and used within the NCCR LIVES project. Thus, the current roadmap includes the development of the framework to support (1) cross-sectional data, (2) network data with a specific handling of biographical data of people cited in the network of each respondent, and (3) panel data organized in successive waves.

The software comes in the form of an R package. R (R Core Team, 2012) is a powerful multi-platform and open-source (so free of charge) statistical tool which is increasingly used in social sciences as an alternative to classical commercial software (like SPSS, SAS, Stata). Most of the recent state-of-the-art methods are available in R and many of them exist exclusively in R. This is especially the case for the newest tools for life course analysis (e.g. `TraMineR` for life course sequence analysis (Gabadinho et al., 2011), `ltm` package for latent class model (Rizopoulos, 2006)). Moreover, working in R allows benefiting from the numerous statistical procedures already optimized in R and taking advantages of R's powerful graphic features.

On a general point of view the Rsocialdata software follows three goals:

1. *Providing an efficient framework for storing and documenting complex survey data.* A key feature of the software is its capability to store data together with the survey design. The data and the user manual describing the data are merged together. Among the different features provided we can mention the possibility of assigning short and long labels to variables and variable values, to declare user-defined types of missing values and to account for cross-sectional and longitudinal weights. Relevant metadata can also be stored such as the target population, the sampling design, information about the organization releasing the data, user license type, etc. Since all information is stored within the data object, the software offers the possibility to generate a summary of the whole data base and the information provided. The summary gives for each variable its long label, the percent of valid cases and basic descriptive statistics. This summary can be directly exported as a PDF file and it can be used as an own user-manual of the data base providing particularly useful information to detect errors and to share data with others.

2. *Saving time.* Preparing data for a study is often a very time consuming and tedious task. The Rsocialdata has been designed to to help researchers in this task reducing the time spent on data processing allowing them to proceed more quickly to the actual (task of) analysis.

   For example, the package provides a search function allowing to explore data and retrieve relevant variables, which is especially usefull on huge and multi-dimensional databases. It also provides efficient tools for recoding categorical and quantitative variables and handling missing values. For example the users can easily turn a missing value as a valid case and vice-versa. Furthermore, for some classical (statistical) methods, the software provides front-ends especially designed for social science scholars.These front-ends facilitate researchers daily work within the R environment. As a key feature, the software performs systematic data consistency checks to ensure that data were not altered during data preprocessing operation. When filtering out cases and sample weights are used, the software also process automatic checks to prevent the loss of representativeness with respect to control variables defined by the user.

3. *Facilitating reproductible research.* Demographic and sociologic questions are generally complex and require a lot of work to be understood. Reproductible research, consisting in attaching sufficient information about data analysis performed so that the work can be recreated, is a helpful methodology when studying social dynamics. Having the possibility to recreate an experiment made by other researchers, or by oneself several months ago, gives the possibility to verify, better understand, and continued the work already done. The Rsocialdata software works in this direction by tracing operations made on data, so that the user can find back the operations she performed. Further, for each method provided by the package, results can be printed in a PDF file which also provides the settings used for getting the results. Further, outputs are printed with a "ready-to-publish" formatting, allowing to quickly focus on the intepretation.

Furthermore, the Rsocialdata package is one of the two tools of the Rscocialdata project. The second tools is a website, whose address is http://www.rsocialdata.unige.ch. This platform allows to explore survey databases that are available in the Rsocialdata file format, and to retrieve them easily in a R environment. This platform follows two goals: First, it aims at providing a full database search engine, allowing researchers to easily find databases which turn out relevant for the analyze of their research questions. For each database available, an overview and a detailed summary are provided as well as summary measures (frequency tables, plots, etc.) for each variable the database contains. Secondly, the platform aims at encouraging the dissemination of survey databases by facilitating the process of making them available to the scientific community. Indeed, an important part of survey databases are lost after the end of the research project whithin which they were collected, often because the task of making them available is generally tedious. As a consequence, a such centralized and unified system will minimize the painful work of searching data from different sources, importing them from various file types or working with databases lacking of documentation.
This initiative is free and open to everybody who would like to participate, either helping to carefully document and prepare existing databases, or sharing your own survey data to the scientific community. Databases are securely stored (access restricted to authorized users, data transfers encrypted in SSL) and the authors can manage the access of their data via a license agreement process. This license agreement process allows the database owner to decide who has access to the data and for which purpose.

In Section 2 we discuss the limitations of native R objects for handling survey data and we present some R packages that provide tools to overcome these limitations. In Section 3 we briefly introduce the design of the package. Section 4 introduces some of the main functionalities of the Rsocialdata toolbox in the form of a user guide. It provides all the material needed to the reader to be autonomous with the software and to start working on its own data. Several illustrations of the tools available for handling panel data are provided. The guide uses data from the Swiss Household Panel (SHP) (Voorpostel et al., 2012). It is a yearly panel study following a random sample of households in Switzerland over time interviewing all household members. We conclude in Section 5 and present some future works. We conclude in Section 5

2 Literature review

The R software does not provide a native framework for survey data managment. The native structure to store data, the `data.frame` object, doesn't allow to store metadata of a survey like short/long labels for values and variables. Missing values are limited to only one type, which make impossible to describe the different types of missing values that can occur when we study individuals. The quality of a survey stands on its representativeness of the population studied and often weights are needed to correct for non-responses bias. Although a lot of statistical methods in R are able to deal with weights, the `data.frame` object doesn't provide a specific weigths handling. Socia network analysis is a powerful tool for studying population. It allows to study the social relationship and how the lives of individuals are linked together. Network data are generally difficult to handle, especially when we deal with demographic data not only for each respondant but also for each indivdual within the network of the respondent. The `data.frame` object offers no other choice than storing data in a flat structure, a 2D array, which doesn't take into account the structure of the network. This leads the user to have a lot of data processing steps to perform before having data ready for analysis. Multiplying the number of steps increases the risk of mistake and can lead to data alteration. Finally, for longitudinal studies, the `data.frame` object doesn't provide a specific structure for storing a variable measured several times accross time. The 2D array structure offers no other choice than dealing with each measurement separately instead of manipulating the variable as a whole. Once again, a such framework leads the user to multiply operations for example repeating the same operation on each wave.

Furthermore, no data consistency accross time check are performed, for example to be aware if the same coding is used accross waves.

There are two packages in R providing a framework for the management of survey data. Both packages are released on the CRAN. The `surveydata` package (De Vries, 2012) simply extends the `data.frame` class to allow storing variable labels. The `memisc` package (Elff, 2013) offers a more sophisticated class allowing to store a label for both values and variables, and allows definable missing values. The package also provides several tools for dealing with survey data, as for example production of codebooks, recoding tools, and can display nice tables of regression model estimates. One choice would have been to start from this package to build the Rsocialdata package. However, integrate a native handling of weights, automatic data consistency checks and a specific structure for efficiently storing network data and panel data would lead to rewrite most classes the package provides.

3 Software design

The software is developed as a standard R package. We opted for an object-oriented programming, adopting the S4 framework (R Development Core Team, 2012) provided by R. This framework specifically fits our needs. First of all, it allows to define methods, i.e. functions with the same name but with a different behavior depending on the objects submitted to them. Dispatching is performed according to each of its arguments. This is useful for limiting the number of function names and allows to group functions having the same behavior but which apply for different classes. As a consequense, this considerably simplify the programming syntax for the user. Secondly, the S4 framework allows to perform systematic validy checks on objects. This is particularly useful for implementing our data consistency checks.

3.1 Cross-sectional data management framework

The main data structure defined in the Rsocialdata package is the `Rsocialdata` object. This S4 object contains the following attributes:

- `variables`: a list containing all the variables resulting of the survey.
- `name`: a character which may contain the name of the database, or short label.

- description : a character for storing the long label of the database.
- row.names: a vector of names for the rows.
- weighting: a character, the name of the variable to use to weights data.
- checkvars: a character vector, listing variable names on which the user want to performed representativness checks.
- spatial: a structured list specifying localization variables and how they have to be used.
- infos: a structured list where are specified the metadata of the survey, and serving at storing information about the tracing of operations processed on the object.
- warnings: a structured list where are specified information the user should take into accounts.
- Rsocialdata.version: the version of the structure of the object.

Data coming from respondents are stored in the variable attribute. This attribute is a list of Variable object. With this metodology, each question of the survey has to be coded in one or more Variable object(s). The Variable object is designed as follow:

- codes: a numeric vector, containing the answers of the respondants, numerically coded.
- values: an object of class Values, which is a dictionary matching numeric codes to short and long labels, and specifying if a value has to be considered as a missing value or a valid case.
- name: a character, naming the variable, or short label. This label is mandatory.
- description : a character, the long label of the variable.
- infos: a structured list where other meta-information about the variable are specified, as its translation in different languages for example, and where are stored information about the tracing of operations processed on the object, especially recoding operations. It also stores the initial distribution of the variable which is used for data representativness checks.
- warnings: a structured list where are specified information the user should take into account.
- Variable.version

The Variable object is the basic structure for storing the answers of respondents. However, at this stage, no measurement has been specified. Besides, the Variable class cannot be instanciated: the class is called a virtual class and it is only as a model for creating more specific objects. Specific structures have been designed to handle variable measurement. All of these structures inherits of the Variable class. Figure 1 presents the differents structures implemented and their inheritance links.

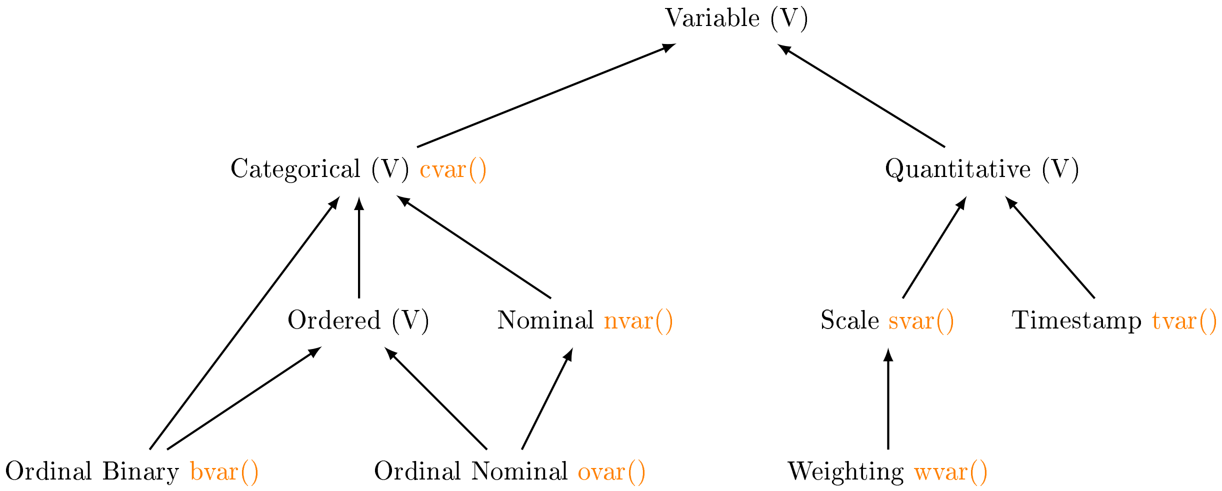


Figure 1: Class diagramme of objects inheriting of the Variable class. Virtual classes are specified by the symbol (V). Commands for creating objects are given in orange.

Only the terminal structures can be instanciated. The list below presents builders for each of these classes:

- svar: builder for creating a variable with a scale measurement.
- cvar: builder for creating a variable with a categorical measurement, either a binary or a nominal measurement.
- ovar: builder for creating a variable with an ordinal measurement.
- tvar: builder for creating a variable with a timestamp measurement

- `wvar`: builder for creating a variable with a weighting measurement. We define a weighting variable as a scale variables where values are positive or null, and no missing values allowed.

For R users used accomodated to native R data structures, here is the corresponding between data structures of the Rsocialdata package and R native data structures:

- `MeasureVariable` objects should be used instead of `numeric` vector objects.
- `CategoricalVariable` objects should be used instead of `factor` objects.
- `OrdinalVariable` objects should be used instead of `ordered factor` objects.
- `Rsocialdata` objects should be used instead of `data.frame` objects.

Note that there is no corresponding for a R "character" vector. Indeed, a character vector doesn't have a sociological meaning, and if a such variable were submitted to a `Variable` builed, the result would be `CategoricalVariable`.

4 Working with the Rsocialdata package: a user guide

In this Section we introduced some functionnalities of the Rsocialdata package. Table 4 summarizes these functionnalies.

4.1 Installing and invoking the package

The toolbox is hosted on the R-Forge platform. The R-Forge is designed for the development of R packages and R-related softwares. The project is accessible at the following URL:

https://r-forge.r-project.org/projects/dataset/

The original name of the package was `Dataset` (Rousseaux et al., 2013). Indeed, we wanted the package able to store any kind of dataset. But last developments focused especially on social and survey data, and new functionnalities, as for example the systematic use of non-response weights, do not make sense for other kind of data. Then, we now prefer the name of `Rsocialdata` for releasing next versions. Currently, for installing the package please use the following command:

```r
install.packages(
  pkgs = 'Dataset',
  repos = 'http://r-forge.r-project.org'
)
```

Once the package is installed, you can load it in your R workspace with the `library` function:

```r
library(Dataset)
```

4.2 Documenting a database

In this Section we presents how a database can be documented. However, Rsocialdata doesn't provide tools for collecting and entering data. The methodologz we advice is to use the GNU PSPP software (Stover, 2010), a freeware multi-plateform, importing the database in Rsocialdata in the same manner as for a SPSS file.

For the following examples we will use data from the Swiss Household Panel (SHP) (Voorpostel et al., 2012).

We first define the paths were data are stored:

```r
datadir.all <- "~/Documents/shp/shp-oct2012/SHP-Data-WA-SPSS/"
datadir.w2011 <- "~/Documents/shp/shp-oct2012/SHP-Data-W1-W13-SPSS/W13_2011/"
```

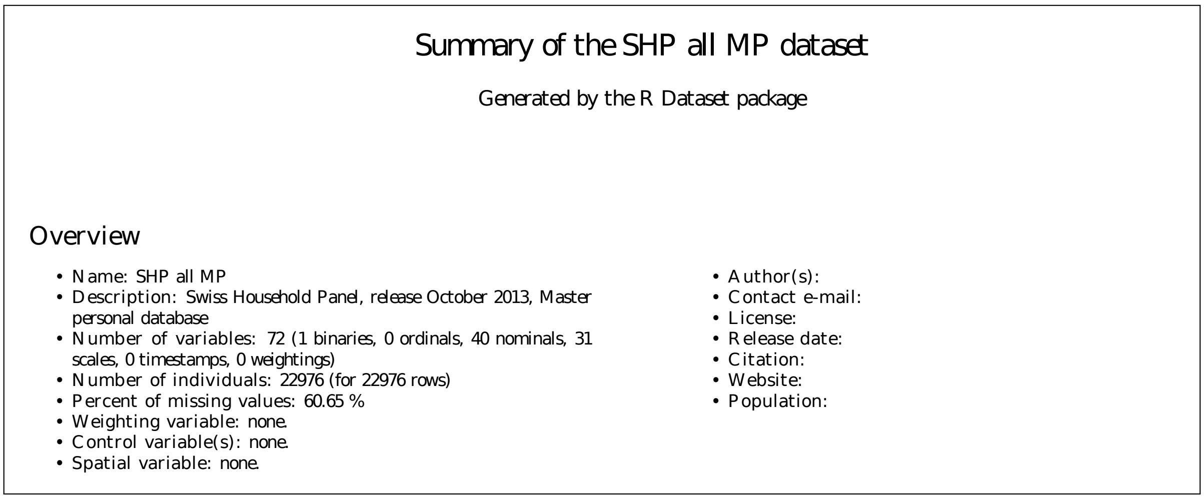
For loading data from an SPSS database, we use the command `get.spss.file` as follow:

```r
shp.all <- get.spss.file(
  file = 'SHP_MP.sav',
  datadir = datadir.all,
  name = 'SHP all MP',
  description = 'Swiss Household Panel, release October 2013, Master personal database'
)
```

In this example we have loaded the Master personal database. We specify a name for the Rsocialdata object and a description. These two last arguments are optional but we strongly recommend to specify them to facilitate database managment when several data bases are loaded in the workspace. The `datadir` argument is also optional, we can actually specify the path by collapsing it with the name of the file in the `file` argument. However we recommend to specify it separately: if the path changes, we will update only one time the path in our script instead of updating it each time we use the `get.spss.file` command. When coding categorical variables a tacit convention is to use positive codes for valid cases and negative codes for missing values. The `get.spss.file` command makes this assumption and will use this criteria to detect missing values. Variable with an ordinal measurement and timestamp variables have to be declared explicitly in the arguments of the `get.spss.file` function. Without these declarations, an ordinal variable will be stored as a nominal variable, and a timestamp variable as a scale variable. On huge databases declaring these measurements is time-consuming. A better choice would be to use a meta-information file where all these declarations already exist for importing them in the software. As Rsocialdata aims at handling longitudinal and network data, we plan to use the DDI specification (Vardigan et al., 2008) to correctly describe data. The structure for handling metadata from a DDI 2.5 XML file is currently under development. Once the database is loaded in the workspace you can generate a codebook with the `exportPDF` method:

```
exportPDF(shp.all)
```

Figure 2 is a screenshot of the first page of the codebook generated. This cover page gives us meta-information about the database. We can recognize the name and the description we have previously defined. The number of variables as well as the number of each type of variable are shown. The number of individuals takes into account the sample weights. If weights are not specified each individual will have a weight of 1 and then the number of individuals is equal to the number of rows. The percentage of missing values is given by the ratio between the total number of missings values in the database and the total number of cells. The weighting variable is the variable we want to use to weigths individuals. Control variables are instead the variables on which we want to assess potential losses of representativness. Weighting and control variables will be discuss in Section 4.3.2. If a localization information has been collected in the survey, it can be explicitly specified. To guarantee correctness of information reported, all numeric information are computed on the fly. On the right side basic information can be specified as authors name, contact information, licence and citation information. It could be particulary useful when we want to share the data with other scholars. The *Population* field allows the administrator of the database to describe the population concerned by the survey.



Summary of the SHP all MP dataset

Generated by the R Dataset package

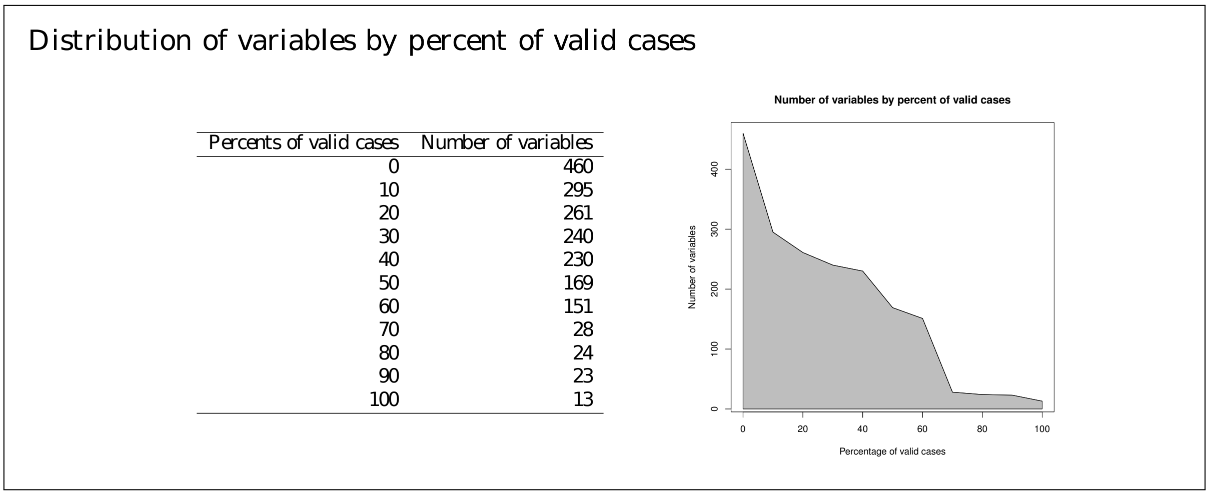
Overview

- Name: SHP all MP
- Description: Swiss Household Panel, release October 2013, Master personal database
- Number of variables: 72 (1 binaries, 0 ordinals, 40 nominals, 31 scales, 0 timestamps, 0 weightings)
- Number of individuals: 22976 (for 22976 rows)
- Percent of missing values: 60.65 %
- Weighting variable: none.
- Control variable(s): none.
- Spatial variable: none.

- Author(s):
- Contact e-mail:
- License:
- Release date:
- Citation:
- Website:
- Population:

Figure 2: Codebook for the shp.all database, page 1.

Figure 3 presents the second page of the codebook. On the left panel, a table shows the number of variables order by the percentage of valid cases. In our example, 3 variables have no one missing values and 7 variables have at least 60% of valid cases. This information is particularly useful when we set up a sub-database for a specific social study. Indeed, many multivariate analyses automatically filter out individuals where a missing value occurs, at least, on one variable. If variables contain a high percent of missing values, we might have a loss of representativness on the sample due to the large number of individuals filtered out. The graphical representation of this table (on the right side) allows to immediatly identify if the variables we are interested in have a sufficient number of valid casesto minimize loss of representativness in such algorithms.



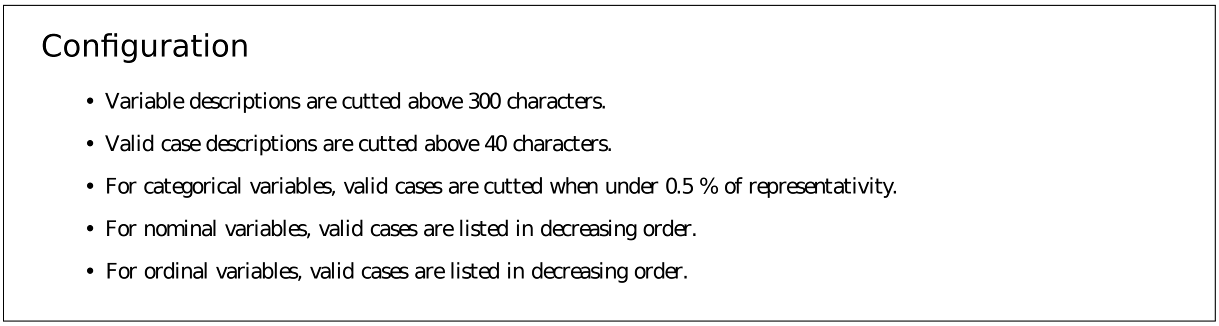
Distribution of variables by percent of valid cases

Percents of valid cases	Number of variables
0	460
10	295
20	261
30	240
40	230
50	169
60	151
70	28
80	24
90	23
100	13

Number of variables by percent of valid cases
Number of variables
0
100
200
300
400
0
20
40
60
80
100
Percentage of valid cases

Figure 3: Codebook for the shp.all database, page 2.

Figure 4 presents the third page of the codebook. This page summarizes the settings used for generating the variable summary for which the first page is presented in Figure 5. As we don't specify these settings when running the command for generating the codebook, default settings are used. Thresholds on the variable and value label lenghts can be specified in order to avoid the codebook gets to heavy and to limit information to the most important one. As some categorical variables might have a high number of levels, e.g. variables storing countries or jobs, it is possible to specify a threshold on the minimum frequency a level has to satisfy for being reported in the codebook. Furthermore, only valid cases are reported, not missing values. Valid cases are listed in decreasing order, increasing order, or without sorting. This order can be specified independently for nominal and ordinal variables, as for ordinal variables it could be chose to preserve the natural order.

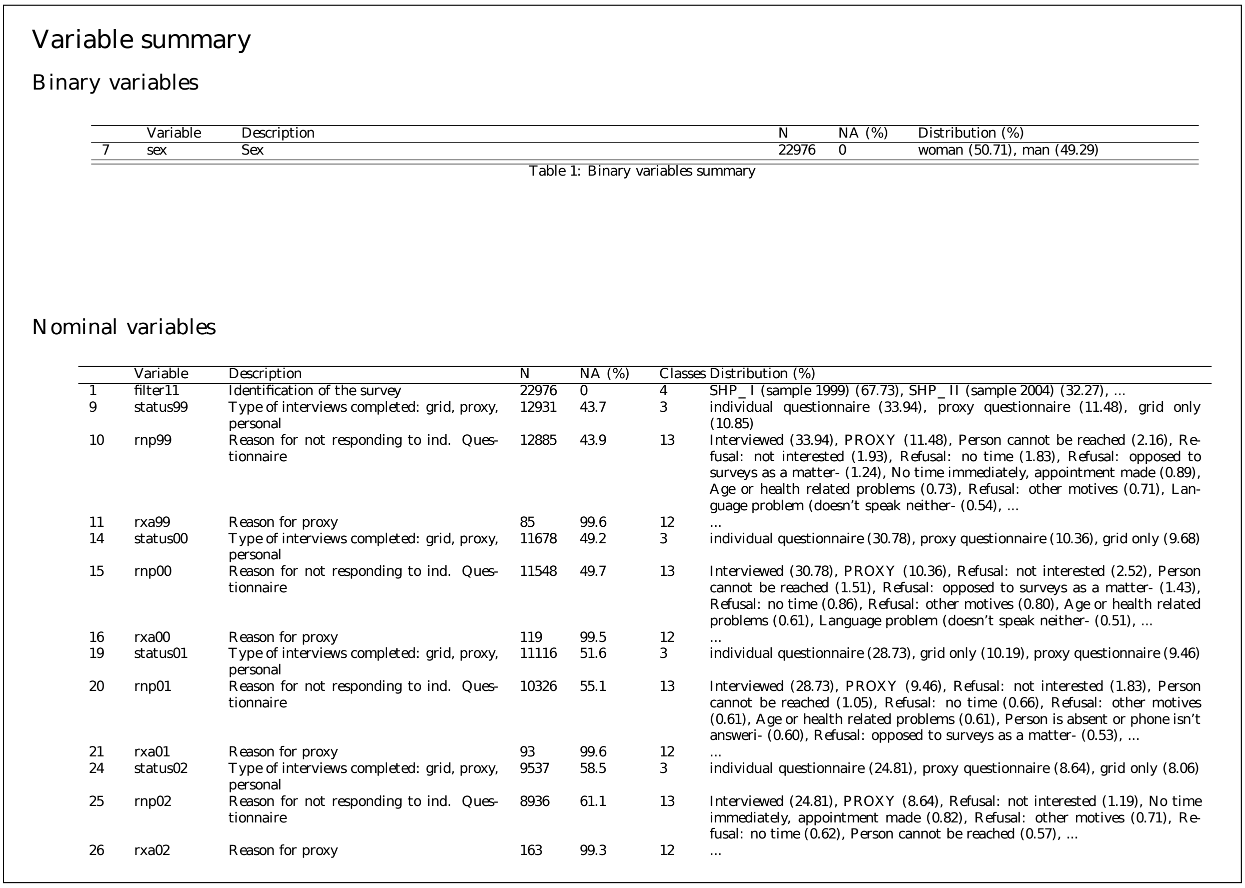


Configuration

- Variable descriptions are cutted above 300 characters.
- Valid case descriptions are cutted above 40 characters.
- For categorical variables, valid cases are cutted when under 0.5 % of representativity.
- For nominal variables, valid cases are listed in decreasing order.
- For ordinal variables, valid cases are listed in decreasing order.

Figure 4: Codebook for the shp.all database, page 3.

Figure 5 presents the fourth page of the codebook which correspond to the first page of the variable summary Section. This Section provides detailed information about each variable stored in the database and it is the core of the codebook. Variables are grouped according to their measurement. For binary variables the codebook displays the column index of the variable in the database, its name and long label, the number of valid cases it contains, the corresponding percent of missing values, and the distribution of its levels. For nominal variables the number of classes is also displayed. In the column 'distributions' a triple-dot mark means that some classes are omitted because they not satisfy the minimal frequency threshold. For instance, for the variable `filter11`, only 2 classes are displayed but 4 classes exist, whereas for the `status99` the 3 classes are displayed. For scales variables basic statistic indicators as min, max, mean and standard deviation are provided.



Variable summary

Binary variables

	Variable	Description	N	NA (%)	Distribution (%)
7	sex	Sex	22976	0	woman (50.71), man (49.29)

Table 1: Binary variables summary

Nominal variables

	Variable	Description	N	NA (%)	Classes	Distribution (%)
1	filter11	Identification of the survey	22976	0	4	SHP_I (sample 1999) (67.73), SHP_II (sample 2004) (32.27), ...
9	status99	Type of interviews completed: grid, proxy, personal	12931	43.7	3	individual questionnaire (33.94), proxy questionnaire (11.48), grid only (10.85)
10	rnp99	Reason for not responding to ind. Questionnaire	12885	43.9	13	Interviewed (33.94), PROXY (11.48), Person cannot be reached (2.16), Refusal: not interested (1.93), Refusal: no time (1.83), Refusal: opposed to surveys as a matter- (1.24), No time immediately, appointment made (0.89), Age or health related problems (0.73), Refusal: other motives (0.71), Language problem (doesn't speak neither- (0.54), ...
11	rxa99	Reason for proxy	85	99.6	12	...
14	status00	Type of interviews completed: grid, proxy, personal	11678	49.2	3	individual questionnaire (30.78), proxy questionnaire (10.36), grid only (9.68)
15	rnp00	Reason for not responding to ind. Questionnaire	11548	49.7	13	Interviewed (30.78), PROXY (10.36), Refusal: not interested (2.52), Person cannot be reached (1.51), Refusal: opposed to surveys as a matter- (1.43), Refusal: no time (0.86), Refusal: other motives (0.80), Age or health related problems (0.61), Language problem (doesn't speak neither- (0.51), ...
16	rxa00	Reason for proxy	119	99.5	12	...
19	status01	Type of interviews completed: grid, proxy, personal	11116	51.6	3	individual questionnaire (28.73), grid only (10.19), proxy questionnaire (9.46)
20	rnp01	Reason for not responding to ind. Questionnaire	10326	55.1	13	Interviewed (28.73), PROXY (9.46), Refusal: not interested (1.83), Person cannot be reached (1.05), Refusal: no time (0.66), Refusal: other motives (0.61), Age or health related problems (0.61), Person is absent or phone isn't answeri- (0.60), Refusal: opposed to surveys as a matter- (0.53), ...
21	rxa01	Reason for proxy	93	99.6	12	...
24	status02	Type of interviews completed: grid, proxy, personal	9537	58.5	3	individual questionnaire (24.81), proxy questionnaire (8.64), grid only (8.06)
25	rnp02	Reason for not responding to ind. Questionnaire	8936	61.1	13	Interviewed (24.81), PROXY (8.64), Refusal: not interested (1.19), No time immediately, appointment made (0.82), Refusal: other motives (0.71), Refusal: no time (0.62), Person cannot be reached (0.57), ...
26	rxa02	Reason for proxy	163	99.3	12	...

Figure 5: Codebook for the shp.all database, page 4.

4.3 Preparing data ready for analysis

We now load the personal and household databases of the Swiss Housel Panel wave 2011:

```
shp.w2011p <- get.spss.file(
  file = 'SHP11_P_USER.sav',
  datadir = datadir.w2011,
  name = 'SHP wave 2011 personal',
  description = 'Swiss Household Panel, release October 2013, wave 2011, personal database'
)
```

```
shp.w2011h <- get.spss.file(
  file = 'SHP11_H_USER.sav',
  datadir = datadir.w2011,
  name = 'SHP wave 2011 household',
  description = 'Swiss Household Panel, release October 2013, wave 2011, household database'
)
```

Our different databases are loaded in the workspace. We often want to merge them in order to have all information on our individuals in the same database. The task is to match all variable values for each individuals. The operation can be performed using the `merge` function which follows the same syntax than the native R `merge` function for `data.frame` objects. We start by merging the Master Personal database with the Household database, then we merge them with the Personal database:

```
shp <- merge(shp.all, shp.w2011h, by = "idhous11")
shp <- merge(shp, shp.w2011p, by = "idpers")
```

After merging we have 11178 rows and 730 variables in our `shp` database.

```r
nrow(shp)

## [1] 11178

ncol(shp)

## [1] 730
```

Let's suppose we are interested in getting some information about the impact on health of being employed and participating to an association. The first stage of our analysis is to select relevant variables for this study. As the database contains a large number of variables the task might be not easy. To facilitate this task the Rsocialdata software provides a function to search across the entire database to retrieve variables containing one of several keywords in their short or long labels. Thus, retrieving variables on the topic of health is easily done with:

```r
health.var <- contains("health", shp)
##                                                         Description
## h11i76a                           Financial help: health insurance
## p11c01                                               Health status
## p11c02                             Satisfaction with health status
## p11c03                       Improvement in health: Last 12 months
## p11c04a               Health problems: Back problems: Last 4 weeks
## p11c05a         Health problems: Weakness, weariness: Last 4 weeks
## p11c06a           Health problems: Sleeping problems: Last 4 weeks
## p11c07a                   Health problems: Headaches: Last 4 weeks
## p11c08         Health impediment in everyday activities: Extension
## p11c19a                Chronic illness or long-term health problem
## p11c11  Number of days affected by health problems: Last 12 months
## p11p54                                      Public expenses: Health
## x11c05                                  Assessment of health status
## x11c06                               Suffering from health problems
## x11c07                                     Cause of health problems
## x11c09                 Days of suffering from health problems: Days
```

Variables related to health are displayed. Further, the output of the command stored in the `health.var` object is itself a `Rsocialdata` object. Thus, we can generated a codebook for this sub-database and easily choose the variable best fitting our problematic, according to its meaning and its characteristics (percentage of valid cases, distribution, etc.).

```r
exportPDF(health.var)
```

In this example we have chosen to work with the health status, which is in the Swiss Household Panel a measure of self-reported health. We now repeat the operation to look at variables related to association membership:

```r
association.var <- contains("association", shp)
##                                                              Description
## p11n40                      Associational membership: Sports or leisure
## p11n41                                Associational membership: Culture
## p11n42                              Associational membership: Syndicate
## p11n43                        Associational membership: Political Party
## p11n44          Associational membership: Protection of the environment
## p11n45                Associational membership: Charitable organisation
## p11n50        Associational membership: Religious organisation or group
## p11n51 Associational membership: Local, parents' or women's association
## p11n52                  Associational membership: Other interest groups
```

```r
exportPDF(association.var)
```

For this study we decide to work with the variable measuring membership to sport or leisure association. The variable has three levels: `Not a member`, `Passive member` and `Active member` and contains about 68% of valid cases. Then we look at a variable related to the working status of respondents. Sometimes only one keyword is not enough for retrieving variables we want, as some words can match to many variables. The `contains` method supports several keywords and can use them in a disjonctive or conjonctive way by specifying the `and` argument.

```
work.var <- contains(c("work", "status"), shp, and = TRUE)
```

```
##            Description
## wstat11 Working status
```

We proceed in the same way for retrieving variables on the sex, age and canton of residence of the respondants. Finally we look for the variable allowing to correctly weight our data in order to make our sample representative of the target population:

```
weights.var <- contains("weight", shp)
```

```
##                                                                                    Description
## wh11t1p            PSMI-PSMII transversal household weight inflating to size of CH-population
## wh11t1s                           PSMI-PSMII transversal household weight keeping sample size
## p11c46                                                                            Weight in kg
## wp11t1p           PSMI-PSMII transversal individual weight inflating to size of CH-population
## wp11t1s                          PSMI-PSMII transversal individual weight keeping sample size
## wp11lp1p       PSMI longitudinal individual weight inflating to size of CH-population in 1999
## wp11lp1s                              PSMI longitudinal individual weight keeping sample size
## wp11l1p  PSMI-PSMII longitudinal individual weight inflating to size of CH-population in 2004
## wp11l1s                         PSMI-PSMII longitudinal individual weight keeping sample size
```

We retrieve the relevant variables for our study. We now summarize them in a vector and then extract our study sample from the whole database:

```
study.variables <- c(
  'wp11t1s',
  'p11c01',
  'age11',
  'sex11',
  'canton11',
  'p11n40',
  'wstat11'
)
study <- shp[, study.variables]
```

We can quickly check the variables existing in our study database with the `alldescriptions` command:

```
alldescriptions(study)
```

```
##                                                            Description
## canton11                                         Canton of residence
## sex11                                                            Sex
## age11                                       Age in year of interview
## p11c01                                                 Health status
## wstat11                                               Working status
## p11n40                   Associational membership: Sports or leisure
## wp11t1s  PSMI-PSMII transversal individual weight keeping sample size
```

4.3.1 Renaming variables and variable values

Variables names in a database are generally coded. When we work on a sub-database for a specific study it is often more confortable to rename variables. This can be done with the `rename` command as follow:

```
study <- rename(study,
  'wp11t1s' = 'weights',
  'p11c01' = 'health',
  'age11' = 'age',
  'sex11' = 'sex',
  'canton11' = 'canton',
  'p11n40' = 'association',
  'wstat11' = 'work.stat'
)
```

We can quickly check that variables have been renamed correctly:

```
alldescriptions(study)
##                                                               Description
## canton                                               Canton of residence
## sex                                                                   Sex
## age                                             Age in year of interview
## health                                                     Health status
## work.stat                                                 Working status
## association                  Associational membership: Sports or leisure
## weights     PSMI-PSMII transversal individual weight keeping sample size
```

The `rename` command can also be used for rename variables values. For example, the variable `health` contains the five following classes:

```
valids(study$health)
##        very well             well so, so (average)    not very well  not well at all
##                1                2                3                4                5
```

One may think that the labels `so, so (average)` and `not well at all` are too long to fit in tables when presenting results of analysis. To solve this problem we can rename them in a shorter form, for example `so, so` and `poor`. This renaming is only for demonstration purpose, in a real study we should make sure that the new labels preserve the meaning of the question. That is, we should be able to justify that with the new labels respondents would have answered in the same manner.

```
study$health <- rename(study$health,
  'so, so (average)' = 'so, so',
  'not well at all' = 'poor'
)
```

```
valids(study$health)
##     very well          well        so, so not very well          poor
##             1             2             3             4             5
```

4.3.2 Setting weights and control variables

The Rsocialdata software offers a native handling of weights. By native we means that the structure for handling weights is integrated *inside* the object storing data. Furthermore, the user has to specify only once the weigthing variable to use. Once weights are specified, they are automatically accounted for by all the analyses provided by the software. Let's start by looking at what returns the software when asking for weights on our study sample:

```
weights(study)
## warning:  no weights defined, equiponderation is used
## number of missings:  0 ( 0 %)
## Description:    !!!empty!!!

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

First of all, the software alerts that we didn't specify weights and informs the user that a weight of 1 will be affected to each individual. Then the software creates the weights with no missing values and an empty description. The output is a replication of the value 1 of the same length as the number of rows. For more lisibility we only display the 15 first values. The `nindividual` function computes the total number of respondent taking into account the sample weights. As currently each individual has a weight of 1, the number of individuals is equal to the number of row:

```
nindividual(study)

## [1] 11178
```

We now set the weights. First, we have to convert our weighting variable, which currently is a `ScaleVariable`, to a `WeightingVariable`. This operation aims at checking that our variable satisfy criteras to be used as a weighting variable: this is a `ScaleVariable`, with all values positive or null, and no missing value.

```
study$weights <- wvar(study$weights)
```

Then we can register the weighting variable in the `Rsocialdata` object.
Now we ask for weights on our study sample:

```
weights(study)
## Description:  PSMI-PSMII transversal individual weight keeping sample size

##  [1] 0.5896 0.4899 1.1283 0.8478 0.5340 0.4377 0.1437 0.3112 0.0000 1.1858
## [11] 0.0000 0.0000 0.5248 0.4899 0.8251
```

In this output we also printed only the first 15 values. Now, The `nindividual` functions returns the number of individuals taking weights into account.

```
nindividual(study)

## [1] 7459
```

After setting weights we can check the control variables, i.e. variables on which we want test of representativness to the initial population were performed. In this example we decide to perform checks on the variables `sex` and `work.stat`.

```
checkvars(study) <- c("sex", "work.stat")
```

Now, each time the `Rsocialdata` object is altered, checks on these variables will be performed. As an example, let's select only people being active member in a sport or leisure association:

```
shp.association <- subset(study, association == "Active member")

## => control on sex:  warning, p-value < 0.05
## man are overrepresented
## woman are underrepresented
## => control on work.stat:  warning, p-value < 0.05
## active occupied are overrepresented
## unemployed, not in labor force are underrepresented
```

From the output we observe that the statistical tests on control variables are not rejected. The tests consist in checking the p-value of the Pearson Chi-squared standardized residual for each class of each control variable. Here, we can conclude that deciding to work only on people who are active members in a sport or leisure association, we will have care of the the significant oversampling of men (and undersampling of women), the oversampling of "active occupied" individuals and undersampling of "not in labor force people". There is no significant loss of representativeness for group "passive member", which is the third class of the "association.assoc" variable, as nothing was reported on this class.

4.3.3 Computing frequency tables

The software is able to display detailed information about frequencies. In the example below we look at frequencies of the health variable:

```
frequencies("health", study)
```

```
##    Coding Missing         Label    N N total Percent Percent (all) Percent total
## 1       1              very well 1428            19.16         19.15
## 2       2                   well 4811            64.52         64.50
## 3       3                 so, so 1037            13.92         13.91
## 4       4          not very well  157             2.11          2.11
## 5       5                   poor   21    7456     0.29          0.29         99.97
## 7      -2       x      no answer    2           100.00          0.03
## 6      -1       x does not know     0             0.00          0.00
## 8      -3       x  inapplicable     0             0.00          0.00
## 9      -7       x  filter error     0             0.00          0.00
## 10     -8       x   other error     0       2     0.00          0.00          0.03
## 11                                         7459                               100
```

The column Coding shows the coding used in the variable. The column missing indicates wether the value has to be interpreted as a missing value or a valid case. The column N counts the number of occurrence of each value. The column N total counts the total number of occurrence respectively for valid cases and missing values, and display the total count. Note that due to the use of weights theses counts are not integers, but they were rounded for a better lisibility. As a consequence, they may be some differences in single counts and aggregated counts. The Percent column displays respectively the percentage of each valid case among all valid cases and the percentage of each missing value among all missing values. The column Percent (all) displays percentages for each value among all values. Finally, the column Percent total indicates respectively the total percentage of valid cases and missing values. The frequencies method can also be used on ScaleVariable objects. For such objects, values are cutted in about 10 ranges with equal width. This allow to appreciate the density of the variable. Here is an example with the variable age.

```
frequencies("age", study)
## number of missings:  6 ( 0.05 %)

##    Coding Missing         Label    N N total Percent Percent (all) Percent total
## 1       1                [0,9.7]    0             0.00          0.00
## 2       2             (9.7,19.4]  592             7.95          7.95
## 3       3            (19.4,29.1] 1066            14.30         14.30
## 4       4            (29.1,38.8]  990            13.27         13.27
## 5       5            (38.8,48.5] 1477            19.81         19.81
## 6       6            (48.5,58.2] 1245            16.70         16.70
## 7       7            (58.2,67.9]  926            12.42         12.42
## 8       8            (67.9,77.6]  716             9.60          9.60
## 9       9            (77.6,87.3]  394             5.29          5.29
## 10     10              (87.3,97]   48    7459     0.65          0.65        100.00
## 11     -1       x does not know     0             0.00          0.00
## 12     -2       x      no answer    0             0.00          0.00
## 13     -3       x  inapplicable     0             0.00          0.00
## 14     -7       x  filter error     0             0.00          0.00
## 15     -8       x   other error     0       0     0.00          0.00          0.00
## 16                                         7459                               100
```

Like most outputs provided by the software, frequency tables can be exported in a PDF file with a nice formatting, or can be included directly in an article as we do below:

Coding	Missing	Label	N	N total	Percent	Percent (all)	Percent total
1		very well	1428		19.16	19.15	
2		well	4811		64.52	64.50	
3		so, so	1037		13.92	13.91	
4		not very well	157		2.11	2.11	
5		poor	21	7456	0.29	0.29	99.97
-2	x	no answer	2		100.00	0.03	
-1	x	does not know	0		0.00	0.00	
-3	x	inapplicable	0		0.00	0.00	
-7	x	filter error	0		0.00	0.00	
-8	x	other error	0	2	0.00	0.00	0.03
				7459			100

4.3.4 Recoding variables

The package provide an easy method for merging levels of a categorical variable. Here an example with the `health` variable. We want to merge levels have codes 1 and 2 together, and 3 to 5 together:

```
study$health.2 <- recode(
  study$health,
  'well' = 1:2,
  'poor' = 3:5
)

## number of missings:  3587 ( 32.09 %)
## Operation completed successfully.
## Here is the allocation of the rows in the different classes.

##
##                  well poor
##   very well     1500    0
##   well          4926    0
##   so, so           0 1015
##   not very well    0  136
##   poor             0   14
```

After performing the recoding operation, the function displays how classes were allocated in the new variable. This lets the possibility to the user to control the result. For having more information, it is also possible to ask for detailed frequencies.

```
frequencies("health.2", study)

##   Coding Missing         Label    N N total Percent Percent (all) Percent total
## 1      0                  well 6239           83.68         83.66
## 2      1                  poor 1216    7456   16.32         16.32         99.97
## 4     -2       x     no answer    2          100.00          0.03
## 3     -1       x does not know    0            0.00          0.00
## 5     -3       x  inapplicable    0            0.00          0.00
## 6     -7       x  filter error    0            0.00          0.00
## 7     -8       x   other error    0       2    0.00          0.00          0.03
## 8                                      7459                                 100
```

The package also provides a method for converting a `ScaleVariable` to a `CategoricalVariable`. It follows the same syntax than the native R `cut` function, but with the difference that there is no need to specify minimal and maximal boundarys. Indeed, if nothing is specified, the method will automatically compute the min and the max. In the following example we decide to create 3 classes by breaking the variable at ages 30 and 65:

```
study$age.3 <- cut(
  study$age,
  breaks = c(30,65)
)

## number of missings:  6 ( 0.05 %)
## Operation completed successfully.
## Here is the allocation of the rows in the different classes.
```

The `cut` method also shows the allocation of values in the new classes, but as the `age` variable contains 98 distinct values the output takes to much place, so we don't display it. However we can have a look at the detailed frequencies of the new variable:

```
frequencies("age.3", study)

##   Coding Missing         Label    N N total Percent Percent (all) Percent total
## 1      1                [0,30] 1775           23.80         23.80
## 2      2               (30,65] 4325           57.99         57.99
## 3      3               (65,97] 1358    7459   18.21         18.21         100.00
## 4     -1       x does not know    0            0.00          0.00
## 5     -2       x     no answer    0            0.00          0.00
## 6     -3       x  inapplicable    0            0.00          0.00
## 7     -7       x  filter error    0            0.00          0.00
## 8     -8       x   other error    0       0    0.00          0.00          0.00
## 9                                      7459                                 100
```

4.4 Statistical Analysis with the Rsocialdata package

As of the main aim of the Rsocialdata software is to provide an efficient framework for increasing researcher's time available to concentrate on their research question when running into analysis, and facilitating reproductible research, we put forward a general methodology for building an analysis tool. The four points a statistical tool provided by the Rsocialdata software has to satisfy are:

1. *The syntax used has to be a direct transposition of what the researchers want to do.* User has to focus on their research question and not on programming-related problems.

2. *The algorithm has to check wether a loss of representativness occured.* In several multivariate analysis methods, individuals with missings values are filtered out before running the analysis. This can be led to a lost of representativness on one or more variable. When this case occurs we think the user has to be alerted of the nature of the data alteration to be able to correctly interpret results of the analysis.

3. *The method has to provide an export option of the outputs of the analysis with the following constraints:* (1) the export generated as a PDF file, (2) exhaustive details about settings and parameters used have to be specified, (3) results are printed with a readable-formatting preferably with a ready-to-publish presentation.

4. *Outputs generated by the analysis can be exported in a TSV (tab-separated values) file.* This allows the user to easily reuse data for publication purpose, for complementary analysis in another software, or share results with other scholars.

Some statistical analysis tools hava been already developed according to this methodology: a bivariate analysis toolbox, some decision tree methods, and a logistic regression method.

4.4.1 Univariate analysis

The package extends classical univariate descriptive statistic methods for taking weights into account. Methods provided are: `min`, `max`, `mode`, `mean`, `standard devition` and `variance`.

4.4.2 Bivariate analysis

The package provides an extensive list of bivariate analysis methods, especially bivariate association measures with their statistical significance. All these methods are integrated within the same function, called `bivan`. The bivan function is able to perform contingency tables (observed and expected) and a series of association measures as the Pearson's Chi-squared, the Cramer's v (Cramer, 1971), the Tschuprow t (Tschuprow, 1919), the Goodmand & Kruskal's τ (Goodman and Kruskal, 1954), the Theil's u (Theil, 1970) and the Somer's d (Somers, 1962). Statistical significance were computed using formulas demonstrated in Ritschard et al. (1996). All results take into account the sample weights. Here an example of bivariate analysis output, the health is the target variable, and 4 explicative variables are tested:

```
bivan(
  health.2 ~ sex + age.3 + association + work.stat,
  study
)
```

Unlike SPSS that provides association measures only in the form of separated tables for each covariate, the package provides a synthetic view of the analysis performed. In addition, the outputs are formatted in a way that it is easy to integrate it in a research publication

	chi2	cramer.v	gk.tau.sqrt	somer.d
sex	23.83 ***	0.06 ***	0.06 ***	0.04 ***
age.3	273.95 ***	0.19 ***	0.19 ***	0.13 ***
association	85.84 ***	0.11 ***	0.11 ***	0.08 ***
work.stat	232.88 ***	0.18 ***	0.18 ***	0.14 ***

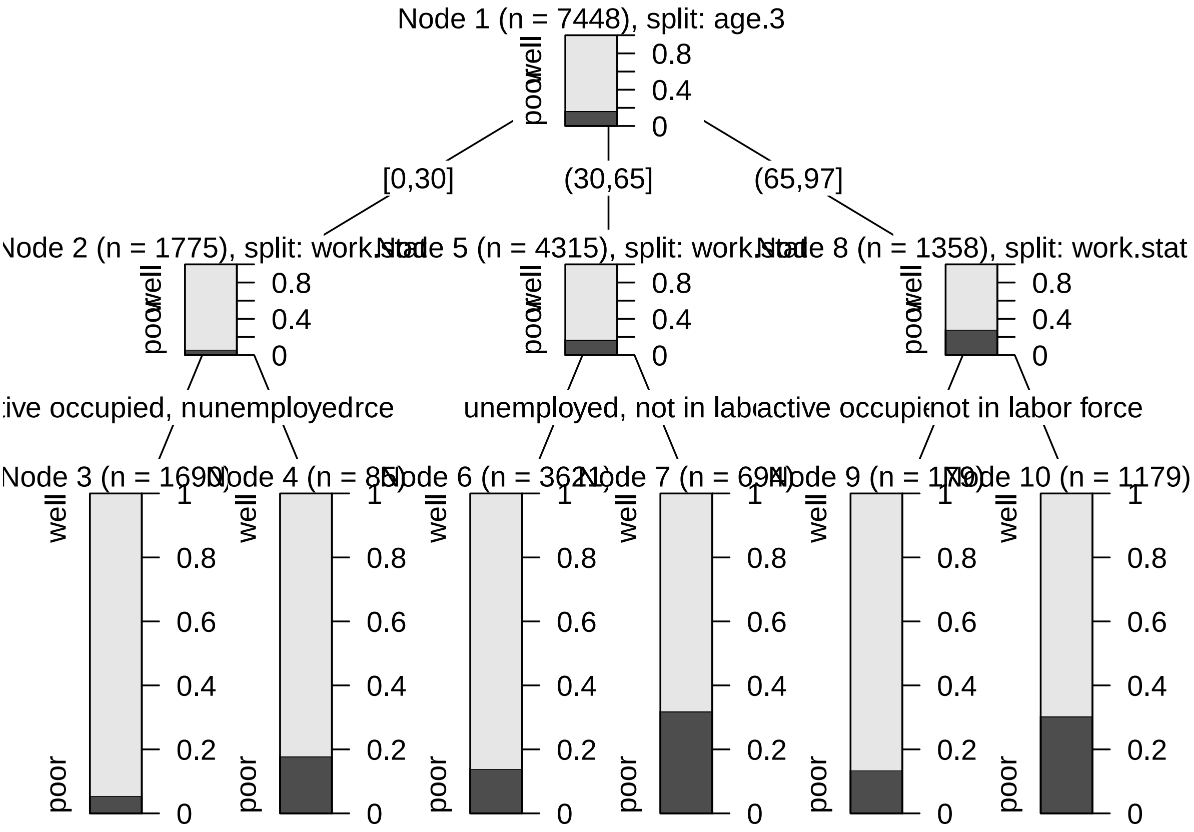
Table 1: Bivariate analysis with the self-reported health as dependend variable. Legend: *** $<$ 0.001, ** $<$ 0.01, * $<$ 0.05, + $<$ 0.1

4.4.3 Tree-based methods

The Rsocialdata package also provides nice front-ends to two classification tree methods: CHAID (Kass, 1980) (package `CHAID`, The FoRt Student Project Team (2009)) and CART (Breiman et al., 1984) (package `rpart`, Therneau et al. (2012)). These methods are very useful for detecting interactions between variables and thus getting more insight about sociological dependences between social determinants. Then, these interactions can be statisticaly testing with a confirmatory analysis, a regression model for example. Here is an example of syntax for computing a tree:

```
tree1 <- tree.learn.chaid(
  formula = health.2 ~ sex + age.3 + association + work.stat,
  data = study,
  control = tree.control(max.height=2)
)
```

```
plot(tree1)
```



4.4.4 Logistic regression

A front-end for a binary logistic regression method is also provided, called `reglog`. The fucntion simply needs the formula of the model and the specification of the target class of the endogenous variable. If the endogenous variable is not a binary variable, a recoding will be performed to create a binary variable with the target supplied having the value 1, and all other classes recoded to 0. This avoid the user to waste time in recoding the variable. Furthermore, the method is able to compute nested models with its argument `nested`. The user can also specify reference classes for each variables. By default, the contrast 'indicator' is used. Here is an example of syntax for computing a logistic regression with the `reglog` function:

```r
reglog(
  formula = health.2 ~ sex + age.3,
  nested = list(
    . ~ association,
    . ~ work.stat
  ),
  target = 'poor',
  reference = list(
    'association' = 'Not a member',
    'age.3' = '[0,30]'
  ),
  data = study
)
```

The method prints results in the R console, and can also be exported in a PDF file, or integrated in an article as shown below:

	Model 1	Model 2	Model 3
sexwoman	1.321 ***	1.254 ***	1.145 *
age.3(30,65]	3.113 ***	2.994 ***	3.468 ***
age.3(65,97]	5.946 ***	5.635 ***	3.598 ***
associationActive member		0.579 ***	0.595 ***
associationPassive member		0.618 ***	0.619 ***
work.statunemployed			2.460 ***
work.statnot in labor force			2.393 ***
(Intercept)	0.056 ***	0.071 ***	0.054 ***

Table 2: Estimated coefficients (odds ratios) , *** $<$ 0.001, ** $<$ 0.01, * $<$ 0.05, + $<$ 0.1, ” $=$ NA

	Model 1	Model 2	Model 3
Deviance	6317.61	6259.56	6147.26
Deviance H0	6626.53	6626.53	6626.53
Model Chi2	308.92 ***	366.97 ***	479.27 ***
Model DF	3.00	5.00	7.00
Block Chi2	308.92 ***	58.05 ***	112.30 ***
Block DF	3.00	2.00	2.00
R2 Cox-Snell	0.04	0.05	0.06
R2 Nagelkerke	0.07	0.08	0.11
N parameters	4.00	6.00	8.00
AIC	6520.87	6468.36	6354.56
BIC	7223.78	7190.42	7071.45
N	7454.00	7454.00	7454.00

Table 3: Quality measures , *** $<$ 0.001, ** $<$ 0.01, * $<$ 0.05, + $<$ 0.1, ” $=$ NA

4.5 Dealing with panel data

Basically, storing panel data means storing repeated cross-sectional data. In the Rsocialdata package, the data structure for handling panel data is the `pRsocialdata` class. This class inherits of the `Rsocialdata` object: to work efficiently on longitudinal data, all waves are merged in only one `Rsocialdata` object. However, the `pRsocialdata` object stores separately metadata of each wave. So, it is always possible to rebuild the original `Rsocialdata` object for a specific wave.

An holistic perspective of the life course has become more and more relevant over the last years. It means to consider the life trajectory as a whole unit of analysis. With the `pRsocialdata` structure the user can directly extract whole trajectories from the panel data without having to bother with extracting the same variable from each wave. The user can extract a whole sequence in a single step simply specifying `'..'` in place of, for example, the two year digits in the variable names. Furthermore, the software automatically checks if each variable has the same missing data pattern across years. Likewise, the user can recode or merge some values, turn a missing value into a valid case with just one command for all the waves.

In our example we will use the waves from 2004 to 2011 of the Swiss Household panel. First of all, we have to create a new object merging the different waves:

```
shp0411 <- get.spss.file(
  waves = c('SHP04_P_USER.sav','SHP05_P_USER.sav','SHP06_P_USER.sav','SHP07_P_USER.sav',
    'SHP08_P_USER.sav','SHP09_P_USER.sav','SHP10_P_USER.sav','SHP11_P_USER.sav'),
  datadir = datadir,
  name = 'SHP waves 2004 to 2011, personal',
  description = 'SHP, release October 2013, waves 2004 to 2011, personal database'
)
```

Then we focus on the self-reported health measured over time.

```
shp0411 <- panelvar(
  'health..' = c('p04c01','p05c01','p06c01','p07c01','p08c01','p09c01',
    'p10c01','p11c01'),
  data = shp0411
)
```

The package automatically check if the variable `health..` has the same numbered codes, range value and value label in all the waves. Also a check on the type of variables (scale, nominal, ordinal, etc.) is performed. These checks are a warranty that your key variable is consistently defined.

Often, variable names change accross waves. For example a variable named `sex` during first waves is named `gender` in the following one. In this case you have to define and add the “panel variable” by hand by calling the `panelvar` function. But if the database follows a strict naming convention, as the Swiss Household Panel does, you can automatically create all the trajectories existing in the database by giving the code identifying each wave to the `get.spss.file` function, as shown in following example :

```
waves = c(
  '04'='SHP04_P_USER.sav','05'='SHP05_P_USER.sav','06'='SHP06_P_USER.sav',
  '07'='SHP07_P_USER.sav','08'='SHP08_P_USER.sav','09'='SHP10_P_USER.sav',
  '10'='SHP10_P_USER.sav','11'='SHP11_P_USER.sav'
  )
```

Once we have define our variables, we use the `wp11l1s` variable to correctly balance our sample for non-response:

```
description(shp0411$wp11l1s)
```

```
##                                                              wp11l1s
## "PSMI-PSMII longitudinal individual weight keeping sample size"
```

```
shp.study$shp0411 <- wvar(shp0411$wp11l1s)
weighting(shp0411) <- "wp11l1s"
```

Now assume that we want to recode, in all waves, the 'health' variable in 3 states instead of its 5 original states. We can perform this operation easily with same `recode` function applied on the new variable:

```
shp0411$health.. <- recode(
  shp0411$health..,
  'well' = 1:2,
  'poor' = 3,
  'very poor' = 4:5
)
```

The package also provides a method for exporting a trajectory as a sequence object ready to be analyzed with the `TraMineR` package (Gabadinho et al., 2011). This package, developed at the Institute for Demographic and Life Course Studies (IDEMO) at the University of Geneva (Switzerland) is intended for mining, describing and visualizing sequential data. We have to distinguish between two types of sequential data: state sequences and event sequences. In state sequences we mainly are interested in the duration and the timing in a life course. In the event sequences, the focus is mainly on the order in which events occur. Indeed, an event is something that occurs at a certain time point and provokes a changing in the state. For instance, "ending a job" is an event, and "being jobless" is the subsequent state. In the Rsocialdata package, with the command `exportSEQ`, the user can easily create either a state sequence or a event sequence simply using the option `type`:

```
shp0411.health.seq <- exportSEQ(
  x = 'health..',
  data = shp0411,
  type = 'state'
)
```

Then, we can use the TraMineR graphic capabilities for rendering our state sequence. Figure 6 renders our health trajectories in a chronogram. A chronogram displays the cross-sectional distributions of the variable at each time, allowing to observe distribution changes over time and to compare the distributions among groups. In Figure 6 we observe that with time the proportion of people in the state `well` decreases.

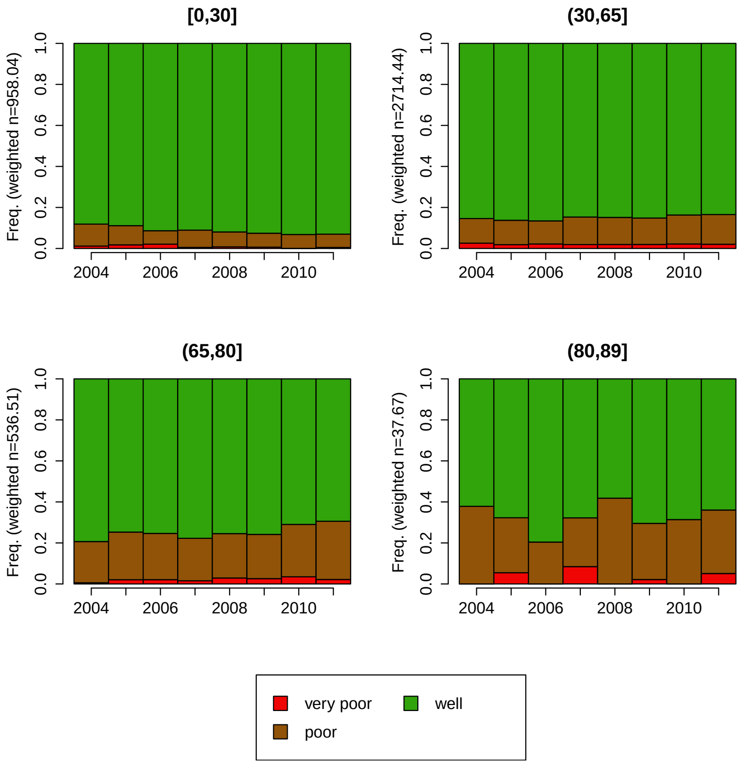


Figure 6: Chronogram of health trajectories by age groups.

As the chronogram plot renders cross-sectional views of the data, it doesn't render individual trajectories. There are several ways to render individual trajectories in TraMineR. To illustrate the `exportSEQ` function, we will use Bürgin's parallel coordinate plot for sequence data (Bürgin et al., 2012), which has been designed to visualize event sequences. We first export the health trajectory as an event sequence:

```
shp0411.health.seq <- exportSEQ(
  x = 'health..',
  data = shp0411,
  type = 'event',
  event.type = 'state'
)
```

The `event.type` parameter specifies how to create the events. One option is to consider existing states, i.e. `very poor`, `poor` and `well`, as start events. An other option is to consider state transitions between the different existing states. This is done by setting the option `event.type = 'transition'` in the previous call. In this case the events would be `well>poor`, `poor>very poor` and so on.

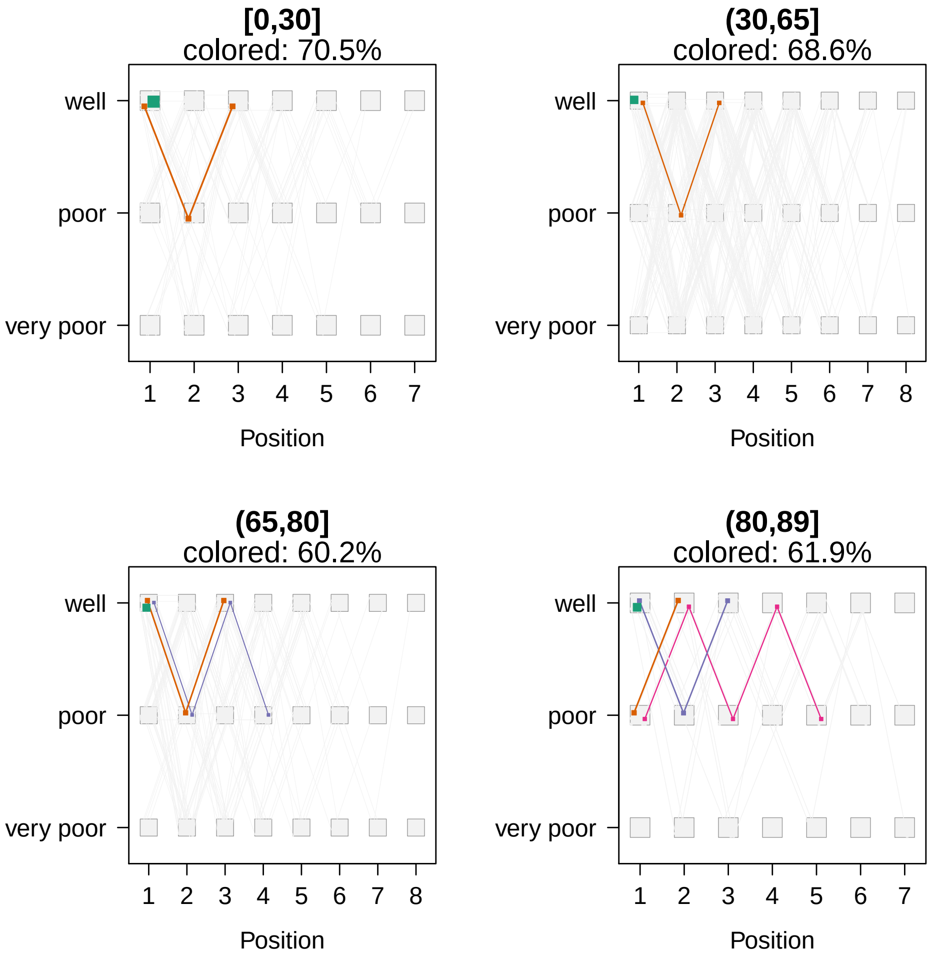


Figure 7: Parallel coordinate plot of health trajectories by age groups, with a highlight of the 60% most frequent sequences.

As you can see in the parallel coordinate plot shown in Figure 7, we are only interested in the sequencing of the events, i.e. in the order in which events occur. The duration of being in a certain state is ignored. In this graphic each unique trajectory is plotted, and the thickness of a line accounts for the frequency of the corresponding trajectory. We also used a filter to highlight only most frequents trajectories. Note that the shape formed made up by the hidden trajectories (in grey) allows to asses the diversity of the remaining trajectories. The plot shows that the youngest cohorts are more likely to stay in good health (green box) or if there is a worsening in the health conditions (from `well` to `poor`), they later move back to the state `well`. On the opposite, health trajectories are less stable for the oldest cohorts.

4.6 Export methods

An important strength of the Rsocialdata package is its ability to export its object in various formats. Indeed, the main goal of the Rsocialdata package is to help social data analysts in their daily activities and writing reports and publications is one of the main tasks. In order to save time, the package comes with a series of command to export most of the outputs with a "ready-to-publish" formatting. The following formats are supported: PDF, LaTeX, HTML, and DOCX (using the Microsoft Office Open XML Format). Note that for DOCX documents the feature is currently in beta testing and not released yet.
Furthermore, we don't want to confine the user in the Rsocialdata framework. R is a modular software which provides a huge number of methods, developed by help of thousands of contributors. Although the package provides a lot of useful tools, we don't want to limit the user to use other R function. For this point the package provides the `exportR` command which export a Rsocialdata object to its nearest equivalent in R. For example a `Rsocialdata` database will be coerce to a `data.frame` object, a `ScaleVariable` object will be coerce to a `numeric`, and a `CategoricalVariable` will be coerce to a `factor`. Further, it can happen that there is no method allowing to do the analysis we need in R, and we need to use another software to do the job. For this purpose the package provide the `exportTSV` function, which export an Rsocialdata object in a Tab-separated-value file which can be read by most of other statistical softwares.

Function	**Description**
Data exploration	
`description`	Return the description database/variable
`alldescriptions`	Provide the name and label for all variables of the database
`contains`	Query the database to retrieve variables whose description matches the keywords
`weighting`	Set/Get the name of the variable used to weight a database
`weights`	Return the weights
`nindividual`	Give the true number of cases taking weights into account
`frequencies`	Display the frequency table or the density of a variable
Recoding	
`recode`	Recode levels of a categorical variable (cross-sectional or panel data)
`cut`	Turn a scale variable to an ordinal variable (cross-sectional or panel data)
`as.valid`	Switch missing values in valid cases
`as.missing`	Switch valid cases in missing values
`checkvars`	Specify variables to use for controlling the representativeness when filtering out cases
Analysis	
`bivan`	Bivariate analysis providing different measures of association
`tree.learn.chaid`	Growth a classification tree using CHAID algorithm
`tree.learn.cart`	Growth a classification tree using CART algorithm
`reglog`	Logistic regression method
Export	
`exportR`	Export an object to a native R object.
`exportTSV`	Export results of an analysis in a Tab-separated values file. Work with statistical methods provided by the package.
`exportPDF`	Generate a summary of an object (database, analysis outputs, etc.) in a PDF file.
`exportTEX`	Generate the summary as a LaTeXdocument.
`exportHTML`	Generate the summary as a HTML file or save it into the clipboard. This export is usefull for importing results in Word.
`exportDOCX`	Generate the object summary in a DOCX document. Currently in development.
`exportSEQ`	Export a `panelvar` to a state sequence or an event sequence.

Table 4: Some key methods provided by the Rsocialdata package

5 Conclusion and future work

The Rsocialdata package provides an efficient and secure framework for handling complex survey data in R. It goes beyond existing solutions by providing an extensive toolbox of methods for documenting data, accounting for sample weights and manipulating panel data. Furthermore, tools for handling network data are forthcoming. In this article we introduce some key functionnalities of the package. We advice the interested reader to go the Rsocialdata website for getting more insight on the package and its functionnalities. The website provides tutorials to learn how to use the package and benefit of its functionnalities.
Future work will mainly concern the upgrade of the tools serving at documenting data as well as developing front-ends for widely used statistical methods. Concerning the tools for documenting data, we work at making the software compatible with the DDI 2.5 specification. We would like to reach out to survey data managment experts for feedback and advices on this implementation, especially to decide which parts of the specification are the most important to implement in the software in the perspective of handling life course data. Concerning front-ends to statistical methods, we firstly plan to add front-ends for other popular methods, as for example linear models (package `stats`, R Core Team (2012)) and clustering (package `cluster`, Maechler et al. (2012)) as well as more specific methods used in life course analysis, as survival analysis (package `survival`, Therneau (2012)) and structural equation modeling (package `lavaan.survey`, Oberski (2012)). We also make efforts to standardize these front-ends and providing documentation about how to set up a new front-end for another method, in order to let other package developers able to follow the methodology we put forward in this project.

References

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and regression trees.* Monterey, CA: Wadsworth & Brooks.

Bürgin, R., G. Ritschard, and E. Rousseaux (2012). Exploration graphique de données séquentielles. In *Atelier Fouille visuelle de données : Méthodologie et évaluation. Conférence EGC2012*, pp. 39–50.

Cramer, H. (1971). *Mathematical Methods of Statistics.* Princeton university press.

De Vries, A. (2012). *surveydata: Tools to manipulate survey data.* R package version 0.1-11.

Elff, M. (2013). *memisc: Tools for Management of Survey Data, Graphics, Programming, Statistics and Simulation.* R package version 0.95-39.

Gabadinho, A., G. Ritschard, N. S. Mueller, and M. Studer (2011). Analyzing and visualizing state sequences in R with TraMineR. *Journal of Statistical Software 40*(4), 1–37.

Goodman, L. and W. Kruskal (1954). Measures of association for cross classifications. *journal of the American Statistical Association 37*, 54–115.

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics 29*(2), 119–127.

Maechler, M., P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik (2012). *cluster: Cluster Analysis Basics and Extensions.* R package version 1.14.3 — For new features, see the 'Changelog' file (in the package source).

Oberski, D. (2012). *lavaan.survey: Complex survey structural equation modeling (SEM).* R package version 0.5.

R Core Team (2012). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0.

R Development Core Team (2012). *Writing R extensions.* R Foundation for Statistical Computing, Vienna, Austria.

Ritschard, G., J. Kellerhals, M. Olszak, and M. Sardi (1996). Path analysis with partial association measures. *Quality and Quantity 30*(1), 37–60.

Rizopoulos, D. (2006). ltm: An r package for latent variable modelling and item response theory analyses. *Journal of Statistical Software 17*(5), 1–25.

Rousseaux, E., D. Bolano, and G. Ritschard (2013). Dataset: An efficient and secure software framework for handling survey data in R. In *Population Days 2013, 10th edition, Brixen, Italy*, pp. 44.

Somers, R. (1962). A new asymmetric measure of association for ordinal variables. *American Sociological Review*, 799–811.

Stover, J. (2010). Gnu pspp: A free clone of spss. In *Joint Statistical Meetings, Vancouver*.

The FoRt Student Project Team (2009). *CHAID: CHi-squared Automated Interaction Detection*. R package version 0.1-1.

Theil, H. (1970). On the estimation of relationships involving qualitative variables. *American Journal of Sociology*, 103–154.

Therneau, T. (2012). *A Package for Survival Analysis in S*. R package version 2.36-14.

Therneau, T., B. Atkinson, and B. Ripley (2012). *rpart: Recursive Partitioning*. R package version 4.0-1.

Tschuprow, A. (1919). On the mathematical expectation of the moments of frequency distributions. *Biometrika 12*, 140–169.

Vardigan, M., P. Heus, and W. Thomas (2008). Data documentation initiative: Toward a standard for the social sciences. *International Journal of Digital Curation 3*(1), 107–113.

Voorpostel, M., R. Tillmann, F. Lebert, U. Kuhn, O. Lipps, V.-A. Ryser, F. Schmid, M. Rothenbuehler, and B. Wernli (2012). *Swiss Household Panel Userguide (1999-2011), Wave 13*. Lausanne: FORS.

Acknowledgement

This publication results from research work is executed within the framework of the Swiss National Centre of Competence in Research LIVES, which is financed by the Swiss National Science Foundation. The authors are grateful to the Swiss National Science Foundation for its financial support.

LIVES
Swiss National Centre of Competence in Research