FACTAGE

STATISTICS AUSTRIA
The Information Manager

Editing EU-SILC UDB Longitudinal Data for Differential Mortality Analyses

R functions and documentation

FACTAGE – WP 4.3

Tobias Göllner, Johannes Gussenbauer and Johannes Klotz

Statistics Austria

June 2018

FACTAGE is a project coordinated in the Joint Programming Initiative: More Years, Better Lives. The Austrian contributions, such as this, are funded by the Federal Ministry of Education, Science and Research

BMBWF
FEDERAL MINISTRY
OF EDUCATION, SCIENCE
AND RESEARCH
www.bmbwf.gv.at

Preface

This R code is a deliverable of the Fairer Active Ageing for Europe (**FACT**AGE) project. **FACT**AGE is exploring emerging inequalities associated with longer working lives. It is a Joint Programming Initiative (JPI) "More Years, Better Lives – The Potential and Challenges of Demographic Change" project. Details on the project can be found on www.factage.eu . Details on the Joint Programming Initiative can be found on www.jp-demographic.eu.

The R code is free to use at your own risk. It is available online at https://github.com/TobiasGold/FACTAGE-method_Mortality.
Suggested citation: Göllner, T., Gussenbauer, J. and Klotz J.: Editing EU-SILC UDB Longitudinal Data for Differential Mortality Analyses. R functions and documentation. FACTAGE project report, June 2019.

Contents

1. Code update: R-package description

1.1. Changes in the UDB

Changes in file and folder structure

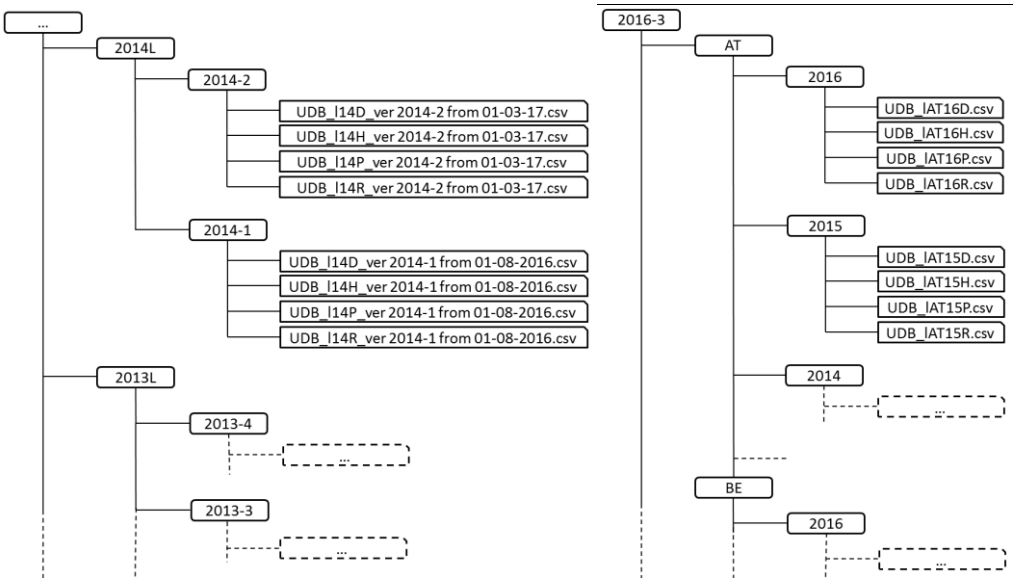
In our first report of this series we used the UDB files from before 2015, since then the files and folder structure has changed. Note that for users coming from academia, who have to apply for the data from Eurostat, on the outside not much has changed, but there is one significant change. Since 2015 personal and household identifier from longitudinal data is linkable to the cross-sectional data. This will open up new analysis possibilities from 2015 onwards, for instance cross-sectional modules, some of which are repeated from time to time, can be enriched with longitudinal information. However, this great opportunity comes with a break in the values of personal and household identifiers, which will be discussed later in this report. Along with this change there was also a change in how data is released.

First, now a release encompasses data of all previous years. Previously, there could have been updates to any release year at any given time. A fictional example would be that in the calendar year 2012 there could have been an update of the release year 2008 from version 3 to 4. A fictional example for the new release logic would be that the new release is 2018 version 2 which includes updates to the files in 2018, 2017 and 2012. So now users do not have to worry about missing any crucial updates of old release years, they can simply rely on the most recent data release.

Second, it used to be that each file contains information of one release year, spanning four calendar years, for all UDB countries. Now files are split between countries. This is also depicted in the new folder structure (see Figure 6). The SAS code which we released along with our first report is still only sensitive to the old structure. The newly released R code is aimed at the new structure.

Figure 6. New and old file structure of EU-SILC UDB.

On the left side the old folder structure where each new release of a longitudinal file was stored in the respective calendar year folder and each file itself contained information on all UDB countries. On the right side the new folder structure, where each release contains data of all countries and each previous year.



Source: Statistics Austria.

Reassignment of personal and household IDs

With this new form of releasing data, there has also been a shift to allow linkage between cross-sectional and longitudinal data. In our first report, the feasibility study, we also found some reassigned personal IDs without this shift. But since this change in data logic and the allowance of linkage, this problem affected basically all countries and observations. Additionally, this also affects the household identifier. Our newly developed R code deals with this in three steps: i.) It defines unique persons through the personal ID (RB030), the country (RB020), sex (RB090) and birthdate (RB070 + RB080); ii.) Then it creates new household identifiers based on consistent person histories (at least one original household member must remain); iii.) And lastly creates new personal IDs. In SAS we use the same logic like in step i.) but we do not account for reassigned household identifier. We advise to use the newly developed method since it produces more accurate results.

Change in death indication

In our first report, around one third of our deceased respondents were recorded in the household register file (D-file). Since the change in file structure, it seems that behind the scenes the way in which this information is recorded was changed. In the data since 2016 we see that the great majority of death indication comes from the personal register file (R-file). This change in recording somehow leads to less cases of deaths in the UDB, even though the

proportions seem to be the same. It is in the nature of living datasets that changes can and will occur. Nevertheless, we hope that these changes will be infrequent. In the previous data we used, in some rare instances, there were cases where the entire household dies but not all persons received the death indication via DB110 = 5 or RB110 = 6. This is not the case anymore and thus our R code doesn't need to correct for this.

1.2. Differences between SAS and R implementation

The R package, just like the SAS code, is designed to use the EU-SILC UDB files as described in earlier reports. The basic functionality of the R package is the same as the SAS code. A minor difference is the naming of certain variables, which should pose no problem, since users probably rename the variables to their own liking anyway. More progress has been made at the issue of the reuse of personal and household IDs (for a description see section 3.1), which is solved in a more satisfactory way in the R package. The SAS code does not deal with reassigned household IDs. Since this is the last report on this project the R package will be released and then only marginally maintained. The code is freely available at GitHub (https://github.com/TobiasGold/FACTAGE-method_Mortality) and we encourage further development on it.

1.3. Quick Start Guide

The R-package "`factage`" is currently being prepared to be put on CRAN (https://cran.rstudio.com/), so that you can install the package and call it via the library command. Until this is process is finished you can download the code from GitHub and use the `source()` command to get the functions into R. We recommend running the example pipeline which is also included in the package as "`pipeline.R`". Each function is basically one step that you need to take in order to get the final output data. You can run the pipeline with minimal modifications. Mainly you have to change the arguments of the first function `readUDB()`. We recommend spending some time to think about what you want to extract and to change those parameters. Afterwards you also have to change the parameters in the last function `castUDB()`, don't forget to include all the variables you specified in the first function as `vars`, as `analyticalVars`. The final dataset can be used for mortality analyses. An exemplary analysis is provided in section 3.4.

```
# load the package
library(factage)
# Specify the path to your data:
path <- "/path/to/my/SILC_UDB_Longitudinal"
# Specify the year range which you want to extract
# One release year inlcudes four calender years.
year_from <- 2008
year_to <- 2016
```

```
# Select your countries. Use any two letter codes. Or specify NULL
for all available countries.
countries <- c("AT", "BE", "FR")
# Choose your analytical variables from the SILC UDB!
vars <- c("PH010","PH030", "HY020")

# read in the data
SILC_UDB <- readUDB(path,year_from,year_to,countries,vars)

# merge data
SILC_UDB_M <- mergeUDB(copy(SILC_UDB))

# reassign IDs
SILC_UDB_MC <- recodeID(copy(SILC_UDB_M))

# flag for eligibilty for follow up
SILC_UDB_E <- setEligibility(copy(SILC_UDB_MC))

# optional:
# save non-eligible respondents
SILC_UDB_nonEligs <- SILC_UDB_E[ELIGIBLEforFU==FALSE]

# calculate entry and exit dates
SILC_UDB_D <- calcDates(copy(SILC_UDB_E))

# get final data set
SILC_UDB_X <- castUDB(copy(SILC_UDB_D),
                      analyticalVars=c("PH010","PH030","HY020"),
                      extractMethod="baseline",
                      DurationUnits="years")
```

1.4. Function Explanation

Here we want to describe the basic functionality of each function. A more detailed explanation is included inside the function documentation of the package. The functions have to be called in the sequence they are described here, which is also given in the "`pipeline.R`" code.

readUDB()

This function reads in the EU-SILC UDB files in their latest folder structure. The user has to specify a folder which contains the csv files of data. The folder structure is: specified folder containing country folders, which contain yearly folders which contain the csv files. See the right hand side of Figure 6 in section 3.1 for a visual representation. Besides defining where the data is, the user can specify which years, countries and variables are to be extracted. The output of the function is a nested list object.

mergeUDB()

This function merges the contents of the nested list object created by `readUDB()` into one data.table. The user only has specify the name of nested list object.

recodeID()

This function deals with the issue of reassigned personal and household IDs as described in section 3.1 and 3.2. Again the user doesn't need so specify anything besides the data which should be used, which is of course the data created by `mergeUDB()`. In short this function does three things: 1. It defines unique persons through certain variables; 2. It creates new household IDs; 3. It renames the personal IDs. The new ID variables follow the nomenclature of EU-SILC, but are not linkable to any other data, and will be different each time you create a new data extraction.

setEligibility()

This function is new and was not represented in the SAS code. When doing differential mortality analysis we can only look at respondents where we can determine a vital status after the initial interview. One can also deal with this by simply removing all respondents with a time at risk in the panel of 0 or less, which should be done regardless as a quality insurance measurement. But, crucially, this function also deals with persons who have a valid first interview and withdrew from the sample at the second interview. These persons are not accounted for in the old SAS code. This function simply requires as data the output from `recodeID()` and creates a new variable called "`ELIGIBLEforFU`" meaning eligible for follow-up. Should you, for whatever reason, not want to call this function but still carry on with the pipeline, you can simply add a variable called "`ELIGIBLEforFU`" to the data.table created by `recodeID()` and set it to "`TRUE`" for every respondent. Similarly, you can also pick out certain groups and set those to the status you need them to. The next function called will only include those lines where "`ELIGIBLEforFU`" is set to "`TRUE`".

calcDates()

This function calculates the entry and exit date of each respondent into the sample. If no valid exit date can be read from the last line, then date of the penultimate line is read and half a year is added. This is the same logic, which we used in the SAS code. The data used is the output of `setEligibilty()` and no other parameters need to be specified.

castUDB()

This function needs some more parameters than just parsing the data from the previous function. Additionally, the user can specify the analytical variables, which usually are just the variables which were also specified for extraction with `readUDB()`. For these variables the user can specify the extraction method, meaning at which point the information of the variable should be extracted, the first or the last observation. Since this function calculates the time at risk of each respondent inside the panel, one can specify in which unit this time should be calculated, e.g. years or days. Lastly, the user can specify additional variables which should

be kept for the final dataset, we recommend the household ID, the sex and the entry and exit date, which are set as the default values, but can be changed.

1.5. Results and Exemplary Analysis Code

After the execution of our envisioned pipeline, you should end up with data as shown in Figure 7. This data then can be used for further differential mortality analyses, such as estimating a Cox regression model.

Figure 7. Exemplary output viewed in R-Studio.

	Country	PID	HHID	FirstSurveyYear	Sex	AgeBaseline	DurationTime	Died	HY020_baseline
2774	AT	24540002	245400	2008	1	20	1.5041096	FALSE	57982.00
2775	AT	24540003	245400	2008	1	47	1.5041096	FALSE	57982.00
2776	AT	24580001	245800	2008	1	57	2.0000000	FALSE	19625.77
2777	AT	24580002	245800	2008	1	53	2.0000000	FALSE	19625.77
2778	AT	24580003	245800	2008	1	45	1.0000000	TRUE	19625.77
2779	AT	24590001	245900	2008	2	58	1.5041096	FALSE	49440.22
2780	AT	24590002	245900	2008	1	39	1.7479452	FALSE	49440.22
2781	AT	24590003	245900	2009	1	41	0.7479452	FALSE	62664.31
2782	AT	24600001	246000	2008	1	58	2.0000000	FALSE	29561.19
2783	AT	24600002	246000	2008	2	56	2.0000000	FALSE	29561.19
2784	AT	24610001	246100	2008	1	52	1.7479452	FALSE	12661.27
2785	AT	24620001	246200	2008	2	64	1.7561644	FALSE	9571.53

Source: Statistics Austria.

Note that here we did not yet restrict the age in any way, something we strongly recommend if you are doing differential mortality analysis; the acceptable age range is 16 to 79. As a difference to the SAS code the variable names slightly changed, but since the labeling is arbitrary anyway, feel free to change it according to your own needs. With this data set you can produce a differential mortality analysis, similar to the one shown in our first report. Assuming that you have specified `HY020`, the annual total disposable household income, as the analytical variable and extracted it from the first observation (baseline) you can do the following analysis:

We estimate a simple Cox proportional hazards regression by comparing the mortality risk of individuals in three household income groups: (1) Less than 10,000 euros; (2) Between 10,000 and 30,000 euros; (3) More than 30,000 euros. Mortality hazard ratios of the two marginal income groups are compared with the reference middle income group. We restrict the sample to individuals aged 30-79 years and with known and positive household income at baseline. We control for differences in age distribution between the income groups.

Disclaimer: This is only a crude model to give you a brief overview of the code. A social scientist would probably specify the model in a more sophisticated way, such as accounting for household size, adjusting for inflation over time and purchasing power differences between countries, including additional control variables, etc.

Exemplary Analysis Code in R

```
# We assume you have already finished the pipeline and
# your final dataset is called SILC_UDB_X

# (Install and) load packages
install.packages(c("haven", "survival"))
library(haven)
library(survival)

# Create an analysis object
analysis <- SILC_UDB_X[which(AgeBaseline >= 30 & AgeBaseline <= 79 &
HY020_baseline > 0),]

# Add a crude income_group variable
analysis$income_group[analysis$HY020_baseline < 10000] <- 1
analysis$income_group[analysis$HY020_baseline >= 10000 &
analysis$HY020_baseline <= 30000] <- 2
analysis$income_group[analysis$HY020_baseline > 30000] <- 3

# Model I: Estimating mortality hazard ratio by household income
category
Model_I <- coxph(Surv(time=DurationTime, event=Died) ~ AgeBaseline +
factor(income_group, levels = c(2,1,3)), data = analysis)
summary(Model_I)

# An alternative model with Sex as stratifier
Model_IIa <- coxph(Surv(time=DurationTime, event=Died) ~ AgeBaseline
+ factor(income_group, levels = c(2,1,3)), subset = Sex=="1", data =
analysis)
summary(Model_IIa)
Model_IIb <- coxph(Surv(time=DurationTime, event=Died) ~ AgeBaseline
+ factor(income_group, levels = c(2,1,3)), subset = Sex=="2", data =
analysis)
summary(Model_IIb)
```

Further Information

Github: @TobiasGold and the repository https://github.com/TobiasGold/FACTAGE-method_Mortality

https://www.factage.eu/ – for further information on FACTAGE

http://ec.europa.eu/research/era/joint-programming-initiatives_en.html – for more information on the various Joint Programming Initiatives

http://www.jp-demographic.eu/ – for information on the JPI – More Years, Better Lives

http://ec.europa.eu/eurostat/web/microdata/european-union-statistics-on-income-and-living-conditions – for information on EU-SILC

http://ec.europa.eu/eurostat/statistics-explained/index.php/Glossary:EU_statistics_on_income_and_living_conditions_(EU-SILC) – for brief information on EU-SILC

https://circabc.europa.eu/ – navigate "Browse categories" → "Eurostat" → "EU-SILC" → "Library". There you find nearly all available information on EU-SILC in general and specifically for the UDB. Under "02. Guidelines" you find the DocSILC065 with all variable information. Under "04. Data and Indicators Dissemination" → "4.1 User Database (UDB)" you find information regarding the UDB.

http://statistik.at/web_en/statistics/index.html – for more information on Statistics Austria and our products