Stacked Data Matrices Workshop

Stata Script Tutorial

Giuseppe Carteny

01.06.2021

Contents

1 Overview

This document consists in walkthrough of the Stata script dedicated to the creation of a stacked data matrix (SDM) of the European Election Study (EES), in particular of the Italian voter study of 2019. After a brief introduction of the workflow employed for creating an exemplary SDM, and after presenting the procedure to download the data for the tutorial, a step by step tutorial for the Stata script provided for the workshop ("`EES2019_Stata_stacking_example.do`") is presented.

1.1 Stata Script Workflow: A Summary

Different workflows can be followed for creating a SDM. These workflows can be summarized, however, in two main strategies:

A. Mutate the relevant variables => Stack the original matrix
B. Stack the original matrix => Mutate the relevant variables

In Stata, both strategies are feasible. However, the script discussed below follows the first strategy (A).
Once defined the general strategy to stack the data, all the routines for generating generic variables (as implemented in the scripts provided for the workshop) are rather similar. Nonetheless, some differences do exist, thus some of the most common routines are briefly presented.
Notice that in each of the following examples, after computing the generic variable desired, the fictional conventional dataset is stacked. In the actual workflow, as already outlined, only once *all* the generic variables are computed the data matrix is finally reshaped.

1.1.1 The basic routine

The basic routine for generating a set of generic variables can be summarised as it follows. Let us say the relevant parties of the selected political context are three, with unique identifiers ranging from `138` to `140`. A fictional dataset is composed by one column referring to survey individual respondents (`respid`), a second column referring to the vote choice of said respondents (`votech`), and a set of columns referring to respondents' propensity to vote for each of the relevant parties (prefix `ptv`). Notice that *each ptv variable name's suffix is related to a relevant party code.*

respid	votech	votech_138	votech_139	votech_140	ptv_138	ptv_139	ptv_140
1	138	1	0	0	9	4	1
2	139	0	1	0	1	8	1
3	140	0	0	1	2	4	9
4	146	.	.	.	1	1	0
5	0	.	.	.	2	1	1
6	.	.	.	.	0	0	0

For computing a *dichotomous generic variable* we mutate the fictional data matrix (1) first creating a variable for each relevant party, and using each relevant party code as suffix of the variable names. Then (2) we assign the "`1`" value when the respondent has voted for the party specified in the suffix of the variable, "`0`"

if the respondent has voted for a different relevant variable, and missing values "`.`" in all the other cases. After these two steps the data matrix should appear like the one presented below:

respid	votech	votech_138	votech_139	votech_140	ptv_138	ptv_139	ptv_140
1	138	1	0	0	9	4	1
2	139	0	1	0	1	8	1
3	140	0	0	1	2	4	9
4	146	.	.	.	1	1	0
5	0	.	.	.	2	1	1
6	.	.	.	.	0	0	0

At this point the fictional data matrix is ready for being stacked. Once specified the variable prefixes (or "stubs") identifying the relevant parties (in our case `votech_` and `ptv`), we stack the fictional matrix above using the `StackMe` package command `genstacks`, resulting in the following SDM:

respid	stackid	votech	votech_stack	ptv
1	138	138	1	9
1	139	138	0	4
1	140	138	0	1
2	138	139	0	1
2	139	139	1	8
2	140	139	0	1
3	138	140	0	2
3	139	140	0	4
3	140	140	1	9
4	138	146	.	1
4	139	146	.	1
4	140	146	.	0
5	138	0	.	2
5	139	0	.	1
5	140	0	.	1
6	138	.	.	0
6	139	.	.	0
6	140	.	.	0

1.1.2 Distance/Proximity variables

The computation of other generic variables follows a similar but somewhat different routine, such in the case of **voter-party distance/proximity** variables.

Let us say that the relevant parties of the selected political context are still those three presented earlier in this section (party codes: `138`,`139` and `140`). Then, let us say that our fictional dataset is now composed by the same individual respondents (`respid`), a second column referring to said respondents' position on the Left-Right *continuum* (`lrself`), and a set of columns referring to respondents' perceptions of the relevant parties' positions on the same *continuum* (`lr_party_138`, `lr_party_139`, and `lr_party_140`).

respid	lrself	lr_party_138	lr_party_139	lr_party_140
1	6	9	4	1
2	5	8	5	1
3	10	9	3	2
4	2	7	5	0
5	0	8	4	1
6	.	.	.	.

With such data, for computing a generic variable measuring *the distance between each respondent and each relevant party on the Left-Right continuum*, we generate three additional variables by computing the absolute difference between each respondent's individual Left-Right self placement and the respondent's perception of each relevant party position on the same dimension, renaming said distance variables using the relevant party codes as suffixes.

respid	lrself	lr_party_138	lr_party_139	lr_party_140	lr_dist_138	lr_dist_139	lr_dist_140
1	6	9	4	1	3	2	5
2	5	8	5	1	3	0	4
3	10	9	3	2	1	7	8
4	2	7	5	0	5	3	2
5	0	8	4	1	8	4	1
6	.	.	.	.	.	.	.

At this point this second fictional data matrix is ready for being stacked. Once specified the variable prefixes (or "stubs") identifying the relevant parties (in our case `lr_party_` and `lr_dist_`), we stack the fictional matrix above using the `StackMe` package command `genstacks`, resulting in a SDM whose first 18 rows are the following:

respid	stackid	lrself	lr_party	lr_dist
1	138	6	9	3
1	139	6	4	2
1	140	6	1	5
2	138	5	8	3
2	139	5	5	0
2	140	5	1	4
3	138	10	9	1
3	139	10	3	7
3	140	10	2	8
4	138	2	7	5
4	139	2	5	3
4	140	2	0	2
5	138	0	8	8
5	139	0	4	4
5	140	0	1	1
6	138	.	.	.
6	139	.	.	.
6	140	.	.	.

The procedure for computing and stacking voter-party distances presented above is one among the many.

Indeed, in the Stata script presented below such distances are computed according to different criteria[1]. Nonetheless, all the routines for calculating such variables follows a very similar logic.

1.1.3 Affinity/Synthetic variables

The procedure for computing **affinity/synthetic** variables is slightly more complicated than those presented earlier in this summary. Since this kind of variables is computed performing (sets of) regression models, we must (1) define the dependent variable of such models, (2) select the proper model to be used (e.g., OLS or logit), (3) fit the model, and (4) compute the predicted scores for each respondent (usually, linear predictions for OLS models and predicted probabilities for logit models).

Let us imagine that the relevant parties of the selected political context are still the three seen earlier (party codes: `138`,`139` and `140`). A fictional dataset is composed by one column referring to survey individual respondents (`respid`), a column referring to respondents' age (`age`), and a set of columns referring to respondents' propensity to vote for each of the relevant parties (prefix `ptv`).

respid	age	ptv_138	ptv_139	ptv_140
1	22	9	4	1
2	50	1	8	1
3	37	2	4	9
4	76	1	1	0
5	0	2	1	1
6	.	0	0	0

Then we regress each `ptv` variable on the `age` variable, and we compute the linear predictions for each respondent. In the `StackMe` package this steop is performed with the `gentyhats` command, as shown in the following pages.

respid	age	ptv_138	ptv_139	ptv_140	age_yhat_138	age_yhat_139	age_yhat_140
1	22	9	4	1	8.2	3.6	1.2
2	50	1	8	1	1.3	8.5	1.4
3	37	2	4	9	2.2	4.6	9.2
4	76	1	1	0	1.4	1.1	0.8
5	0	2	1	1	2.3	1.2	1.2
6	.	0	0	0	.	.	.

Finally we stack this fictional data matrix as done earlier for other variables: we specify the variable prefixes (or "stubs") identifying the relevant parties (in this case `ptv_` and `age_yhat_`), and then we stack the fictional matrix above using the `StackMe` package command `genstacks`.

[1] For instance, party-voter distances are calculated taking the absolute difference between each respondent Left-Right self placement and the *average* of all the respondents' perceptions of each relevant party position on the same dimension

respid	stackid	age	ptv	age_yhat
1	138	22	9	8.2
1	139	22	4	3.6
1	140	22	1	1.2
2	138	50	1	1.3
2	139	50	8	8.5
2	140	50	1	1.4
3	138	37	2	2.2
3	139	37	4	4.6
3	140	37	9	9.2
4	138	76	1	1.4
4	139	76	1	1.1
4	140	76	0	0.8
5	138	0	2	2.3
5	139	0	1	1.2
5	140	0	1	1.2
6	138	.	0	.
6	139	.	0	.
6	140	.	0	.

Although relying on a single predictor for computing the synthetic variable presented above, such variables can be created using *multivariate* regression models, thus *synthetising* the affinity of each respondent with each relevant party on the basis of a set of multiple predictors (e.g., socio-demographic variables).

1.2 Data and Documentation

The workshop data, scripts, and documents can be downloaded from the following link: https://github.com/giucarny/StackMat/archive/refs/heads/master.zip. After clicking or browsing the link, a `.zip` folder named 'StackMat-master' will be automatically downloaded. Unzip it and browse it at your wish. In sum, this folder represents the copy of a public GitHub repository named ' StackMat ' which contains three main subdirectories:

- '**data**': In which are stored the (main and auxiliary) datasets for the stacking procedure (e.g., the dataset of the EES voter study of 2019) as well as the SDM generated by the Stata script explained in the following lines;

- '**scripts**': In which are stored the (main and auxiliary) scripts for stacking country-specific data frames of the EES 2019 voter study , including by said Stata script;

- '**documentation**': A folder containing relevant documents for the stacking process (e.g., the EES 2019 voter study questionnaire and codebook) and the tutorial (e.g., the codebook of the SDM generated by the Stata script presented below).

2 Stata Script

Browse the "`~/data/`" subdirectory and then double-click on the "`EES2019_Stata_stacking_example.do`" file. If "`.do`" files are linked to your current Stata version[2] this will open a new Stata working session. If not, then open Stata, and open the already mentioned "`.do`" file from the software.

2.1 Setup

The first lines of the script are dedicated to the software setup, in particular to the installation/udpate of the `StackMe` package. This package is available on a GitHub repository, but not on the Boston College Statistical Software Components (SSC) archive. Consequently, `StackMe` installation requires the prior installation of another package (`github`) that allows to download and install packages directly from GitHub.
If you already installed both `github` and `StackMe` packages, then you can skip the code lines below. If you did not, you can run the installation of both packages using the following commands. If you already installed both packages and re-run the installation Stata will automatically check the version and consistency of the packages.

```
net install github, from("https://haghish.github.io/github/")
github install ldesio/stackme
```

If you get any error during the installation/update of both packages, or in the script chunks that involve `StackMe` commands, then you might uninstall previous versions of the packages with the following commands, and then rerun the previous steps.

```
ado uninstall github
ado uninstall stackme
```

2.2 Admin

At this point you need to set up the working directory. For compiling properly, the following script requires that you set the working directory in the "`~/data/`" subdirectory. Thus, check the working directory of the current session of Stata and then set the working directory in the "`~/data/`" subfolder.

[2]The script presented in this tutorial has been tested on Stata versions 15 and 16. If you need to purchase, download, or update Stata please browse to the "`~/documentation/`" subdirectory and check the "`StackMat---Software-Setup.pdf`" file.

```
pwd
cd " ~ \StackMat-master\data"
```

2.3 Load (and Merge) Data

After setting up the relevant packages and the working directory, then you can load the data for the stacking procedure.
First, we load on Stata the main data frame (the original EES 2019 voter study dataset).

```
use "ZA7581_v1-0-0.dta", clear
```

Then, we merge this dataset with an auxiliary one, containing party-specific data from the Chapell Hill Experts Survey (CHES) of 2019[3], such as experts average scores of party positions on the Left-Right *continuum* or about the European Union (EU) integration process, that will return later on in the script. Merging the dataset will produce some additional variables, among which the `_merge` one. Without dropping the latter variable the subsequent merging procedure would produce an error. Thus, we drop the `_merge` variable from the current dataset.

```
merge 1:m respid countrycode Q7 using EES_CHES_2019_aux
drop _merge
```

Finally the dataset is merged with a second auxiliary data frame, containing the EES 2019 party identification variable (`Q25`; See the EES2019 Codebook) recoded according to the 2019 European Parliament (EP) Elections vote choice variable values (`Q7`; See the EES2019 Codebook)[4].
The recoding is necessary because the values of these two variables differ in their link with actual political parties. For instance, in the EES 2019 voter study, the value of the 2019 EP elections vote choice variable that refers to the Italian Democratic Party (*Partito Democratico*) is `1501`. The same party is coded `1502`in the party identification variable of the EES 2019 voter study.

[3]These data are derived by the 1999-2019 CHES trend file, available in the "`~/data/`" subdirectory. The dataset derived from the CHES file is created with an R script, available in the "`~/scripts/`" subdirectory, named "`StackMat-aux_data_script.R`". This script can be easily implemented to include further party-specific variables or, in perspective, country-specific variables for comparative analyses.

[4]The dataset containing the recoded variable (`EES_2019_Q25_aux.dta`) is realised with the R script named "`StackMat-aux_data_script.R`", available in the "scripts" subdirectory ("`~/scripts/`").

Since (a) the definition of the relevant parties in a specific party system *is a key passage of the stacking procedure*, and (b) in this tutorial said parties are identified *using the values of the EES 2019 EP elections vote choice variable* (`Q7`), then the values of the party identification variable (`Q25`) have been recoded in accordance with those of the mentioned vote choice variable (`Q7`). Thus, merging the main dataset with the auxiliary one, then dropping the `_merge` and the original party identification variables (`Q25`), and finally renaming the recoded one (`Q25_rec`), will produce a dataset in which the values of the party identification variable (`Q25`) are in accordance with the 2019 European Parliament (EP) Elections vote choice variable values (`Q7`).

```
merge 1:m respid countrycode Q25 using EES_2019_Q25_aux
drop _merge

drop Q25
ren Q25_rec Q25
```

2.4 Select Party System and Relevant Parties

The *selection of a specific (democratic) political context* (1), and the following *choice of a set of relevant political parties constituting the political context under scrutiny* (2), represent **two key steps** of the stacking procedure, in general and in the script presented here. The first step is key because it implies specific constraints on the set of parties available for the stacking process. The second step is critical and needed because (a) most of times theoretical arguments and/or data features might not allow you to keep the whole set of parties considered as constitutive of a specific political context, and because (b) the selection of a specific set of political parties has fundamental implications for all the following steps of the stacking procedure.

In the Stata script, the first step (1) is performed by keeping only the observations referring to the EES 2019 Italian voter study, using the values of the EES 2019 `countrycode` variable (See the EES2019 Codebook).

```
keep if countrycode==1380 // EES2019 Italian voter study
```

The second step (2) is then performed according to the following criteria:

- First, relevant party codes are in accordance with the values of the 2019 EP elections vote choice variable (`Q7`; See the EES2019 Codebook);
- The parties selected are only those for which the EES 2019 provides a propensity to vote (PTV) variable.

Following these (pragmatic) criteria, the selection of relevant parties, and party codes, that guides the exemplary Stata script is the following:

Party Name (it)	Party Name (en)	Party Code (Q7)
Partito Democratico	Democratic Party	1501
Forza Italia	Go Italy	1502
Lega	Northern League	1503
Movimento 5 Stelle	Five Star Movement	1504
Sinistra Italiana	Italian Left	1505
+Europa	More Europe	1506
Fratelli d'Italia	Brothers of Italy	1507

The selection explained above is clearly related to data availability/constraints. Nonetheless, several criteria could guide the selection of the relevant parties in a given context[5].

2.5 Select the relevant variables and mutate the dataset

After selecting the relevant parties, the next step of the Stata script is dedicated to selection of the variables that will be used for computing the *generic* variables constituting the SDM. Since all the selected variables will appear again in the following lines of the script, they will be more thoroughly explained in the following pages of the tutorial. Nonetheless, notice that in addition to the selection of said variables, in this passage the dataset is slightly mutated and a variable measuring EES respondents' age is generated (`age`; See the "`StackMat - EES2019_ita_codebook.pdf`" file in the "`~/documentation/`" subdirectory).

```
keep respid countrycode Q7 q10* Q11 q13* D3 D4_1 Q23 q24* Q25 Q26 EDU lrgen eu_position

gen year = 2019
gen age = 2019 - D4_1
keep if age>17
drop D4_1

ren D3 gndr
ren EDU edu
```

2.6 Dependent Variables (Vote Choice and Propensity to Vote)

The first generic variable generated by the Stata script is a **dichotomous** generic dependent variable. First, the EES 2019 original variable is recoded according to the choice of the relevant parties presented above

[5]For instance, one might select as relevant all the parties that won at least one seat in a legislative election, or those parties that obtained a certain share of valid votes.

(values: from 1501 to 1507), setting as missing all the values referring to (a) unrelevant parties (values: equal to or greater than 1508), (b) EES missing values (values: smaller than 100), and (c) respondents that did not vote for any relevant party (value: 0).

```
replace Q7=. if Q7<100
replace Q7=. if Q7>=1508
```

Then, the script will generate a set of variables following the logic presented in the first part of this document (See Sect. 1.1.1). The loop presented below, using as index the relevant parties' codes (from 1501 to 1507), will produce a set of dichotomous variables based on the vote choice variable (Q7), named using the already mentioned party codes as suffixes (e.g., Q7_stack_1501, Q7_stack_1502,...).

```
forvalues i = 1501/1507 {
    generate Q7_stack_‘i’ = Q7
    replace Q7_stack_‘i’ = 1 if Q7_stack_‘i’==‘i’
    replace Q7_stack_‘i’ = 0 if Q7_stack_‘i’!=‘i’ & Q7_stack_‘i’!=1
    replace Q7_stack_‘i’ = . if missing(Q7)
}
```

As already shown earlier (See Sect. 1.1.1), since the EES 2019 propensity to vote variabes (prefix q10_) are already shaped for being stacked, the following lines of the script are just dedicated to (a) dropping the empty ptv variables, (b) renaming said variables in accordance with the relevant party codes, (c) recoding the variables' missing values, and (d) rescaling the variables' values in order to fit into the [0,1] interval.

```
drop q10_8 q10_9 q10_10

forvalues i = 1/7 {
rename q10_‘i’ q10_150‘i’
}

forvalues i = 1501/1507 {
replace q10_‘i’=. if q10_‘i’>10
replace q10_‘i’= q10_‘i’/10
}
```

These variables will be stacked in the latter part of the script with the genstacks command.

2.7 Party identification

The routine for stacking the EES 2019 party identification variables (`Q25` and `Q26`; se the EES2019 Codebook) is almost the same one adopted for stacking the voting choice variable, thus this part of the code is only lightly commented. What should be noted is that in this script chunk the party identification strenght variable (`Q26`) is recoded in the following manner:

Q26 before	Value labels	Q26 after
0	R does not identify with any party	0
3	R is merely a sympathiser of a specific party	1
2	R is fairly close to a specific party	2
1	R is very close to a specific party	3

Despite this recoding, as already outlined, the routine for stacking said variables is similar to the basic one addressed earlier in this tutorial (see Sect. 1.1.1).

```
forvalues i = 1501/1507 {
    generate Q25_stack_‘i’ = Q25
    replace Q25_stack_‘i’ = 1 if Q25_stack_‘i’==‘i’
    replace Q25_stack_‘i’ = 0 if Q25_stack_‘i’!=‘i’ & Q25_stack_‘i’!=1
    replace Q25_stack_‘i’ = . if missing(Q25)
}

recode Q26 (0=0) (2=2) (3=1) (1=3)

forvalues i = 1501/1507 {
    generate Q26_stack_‘i’ = Q26 if Q25==‘i’
    replace Q26_stack_‘i’=0 if missing(Q26_stack_‘i’)
}

replace Q26=. if missing(Q25)
```

2.8 Distance variables (Left-Right and Positions about European Integration)

This part of the Stata script addresses the creation of voter-party *distance* variables. The routine for computing such variables has been already presented earlier in this tutorial (see Sect. 1.1.2). Nonetheless, the routine is slightly from the one presented earlier.
Focusing on computing party-voter distances on the Left-Right *continuum*, what can said is that the main differences between earlier examples (Sect. 1.1.2) and the actual routine implemented in the Stata script consist in the fact:

- In one case, said differences are computed substracting respondents' Left-Right self placement (`Q11`) to the *average* of individual perceptions of party positions on the same dimension (prefix `q13`);
- In the other case, such differences are computed substracting respondents' Left-Right self placement (`Q11`) to party positions as measured by the 2019 Chapel Hill Experts Survey (CHES) data (`lrgen`; See the 1999-2019 CHES Codebook).

Thus, first (1) we drop the EES 2019 party position variables related to unrelevant parties, then (2) we rename the variables according to the relevant parties codes (from `1501` to `1507`), (3) we remove the missing values from both variables, and finally (4) we rescale the variables to fit into the [0,1] interval.

```
drop q13_8 q13_9

forvalues i = 1/7 {
    rename q13_‘i’ q13_150‘i’
}

replace Q11=. if Q11>10
forvalues i = 1501/1507 {
replace q13_‘i’=. if q13_‘i’>10
}

replace Q11 = Q11/10
forvalues i = 1501/1507 {
    replace q13_‘i’ = q13_‘i’/10
}
```

In the following step we first (5) compute average scores of individuals' perceptions of party positions on the Left-Right *continuum* and then (6) we create a set of columns also for the CHES variable (`lrgen`), that in the end is dropped from the dataset.

```
forvalues i = 1501/1507 {
egen q13_mean_‘i’ = mean(q13_‘i’)
}

forvalues i = 1501/1507 {
gen lrgen2_stack_‘i’ = lrgen if Q7==‘i’
egen lrgen_stack_‘i’ = mean(lrgen2_stack_‘i’)
drop lrgen2_stack_‘i’
}
```

```
drop lrgen
```

Finally (7) we compute the voter-party distances for both EES and CHES based variables by calculating the absolute difference betweee respondents' Left-Right self placement and said party-specific variables.

```
forvalues i = 1501/1507 {
    gen q13_dist_‘i’ = abs(q13_mean_‘i’ - Q11)
}

forvalues i = 1501/1507 {
    gen lrgen_dist_‘i’ = abs(lrgen_stack_‘i’ - Q11)
}
```

The routine for creating voter-party distances for the European integration process is *exactly the same*. Thus it will not be commented in the following pages of the tutorial.

2.9 Synthetic variables

As already explained earlier (see Sect. 1.1.3), for generating the so-called synthetic/affinity we need to rely on (OLS or logit) regression models, thus we need to first (a) define a dependent variable, (b) select the predictors of said models (namely, what this variable should synthesize), and finally (c) fit the regression models for each relevant party and compute some predicted scores for each respondent. In Stata this routine can be implemented in several ways. However, the `StackMe` package provides an easy solution for calculating such scores (a.k.a. "y-hats") that is the `genyhats` command.

In the Stata script said *y-hats* are computed with a sequence of loops including the `genyhats` function. For instance, the loops below will use the respondents' `age` variable to predict two set of scores. First (1) such scores are computed using as dependent variables the set of vote choice dummy variables indicating whether the respondent voted or not for the relevant parties (prefix `Q7_stack`). Then (2) such scores are estimated using as dependent variables those measuring respodents' propensity to vote for each relevant party (prefix `q10`).

Clearly for the first set of dependent variables (1) we use a set of logit models (specifying the `log` option in the `genyhats` function), while for the second set (b) we use a set of OLS models (default option in `genyhats`). Notice that in both cases the choice has been to *do not adjust* in any way the predicted scores (`adjust(no)` option).

Thus, the result of the following two loops will be a set of columns, for each dependent variable, referring to each relevant party, in which are stored the predicted probabilities (1) or the linear predictions (2) of the regression models employed.

```
forvalues i = 1501/1507 {
genyhats age_dich_yhat_‘i’: age, dep(Q7_stack_‘i’) log adjust(no)
}

forvalues i = 1501/1507 {
genyhats age_cont_yhat_‘i’: age, dep(q10_‘i’) adjust(no)
}
```

As already noted earlier (see Sect. 1.1.3), synthetic variables can be computed using more than one predictor. And indeed the script provides examples for estimating “y-hats” using multivariate models, such as the following ones:

```
replace gndr = . if gndr==3

forvalues i = 1/3 {
gen edu‘i’ = edu
}
recode edu1 (1 = 1) (2 3 = 0)
recode edu2 (2 = 1) (1 3 = 0)
recode edu3 (3 = 1) (1 2 = 0)


forvalues i = 1501/1507 {
genyhats age_dich_yhat_‘i’: age, dep(Q7_stack_‘i’) log adjust(no)
}

forvalues i = 1501/1507 {
genyhats age_cont_yhat_‘i’: age, dep(q10_‘i’) adjust(no)
}

drop edu1 edu2 edu3
```

Notice that the respondents’ education variable (`edu`) has been recoded, creating a set of dichotomous variaables, because `genyhats` *cannot handle* categorical variables with more than two values, missing excluded.

2.10 Stacking the observations

After computing all the relevant variables, the final substantive step of the stacking procedure consists in stacking the variables discussed so far. For achieve this goal we rely again on a command from the `StackMe` library, namely `genstacks`.

The syntax of the function si relatively simple. After specifying the command we need to insert the prefixes or "stubs" of the variables that we want to stack. Such stubs have to be specified *leaving out the numerical suffixes* used earlier. This omission is needed because the function will use said suffixes (that, namely, are the relevant parties' codes mentioned in several passages of this tutorial).

For instance, if we want to stack the propensity to vote variables (`q10_1501`, `q10_1502`, `q10_1503`,...) we would insert the `q10_` stub. Then the `genstacks` command would automatically take the suffixes (`1501`, `1502`, `1503`,...) for creating the voter-party dyads. Thus, when specifying the variables to be stacked, it is important that *all the selected variables have the same suffix*, namely the codes of the relevant parties specified at the beginning of the script.

```
genstacks q10_ q13_mean_ q13_dist_ q24_mean_ q24_dist_ ///
lrgen_stack_ lrgen_dist_ eu_position_stack_ eu_position_dist_ ///
Q25_stack_ Q26_stack_ ///
Q7_stack_ ///
age_dich_yhat_ age_cont_yhat_ ///
socdem_dich_yhat_ socdem_cont_yhat_, rep
```

At this point of the script, your dataframe has been finally stacked! The actual Stata script actually continues then re-adjusting the variables names and order, that however will not commented in this tutorial.

If you would like to try to stack the same date using a 'Stata-like' R script, please browse to the "`~/scripts/`" subdirectory and test the "`EES2019_R_stacking_example.R`" file, that runs (almost) parallel to the Stata script presented above.

Thank you again for participating to the workshop!