The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*), *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		**Printed & electronic**	
1-year subscription	$115	1-year subscription	$145
2-year subscription	$210	2-year subscription	$270
3-year subscription	$285	3-year subscription	$375
1-year student subscription	$ 85	1-year student subscription	$115
1-year institutional subscription	$345	1-year institutional subscription	$375
2-year institutional subscription	$625	2-year institutional subscription	$685
3-year institutional subscription	$875	3-year institutional subscription	$965
Electronic only		**Electronic only**	
1-year subscription	$ 85	1-year subscription	$ 85
2-year subscription	$155	2-year subscription	$155
3-year subscription	$215	3-year subscription	$215
1-year student subscription	$ 55	1-year student subscription	$ 55

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata, **Stata**, Stata Press, Mata, **Mata**, and NetCourse are registered trademarks of StataCorp LP.

newspell: Easy management of complex spell data

Hannes Kröger
European University Institute
Florence, Italy
hannes.kroger@eui.eu

Abstract. Biographical data gathered in surveys are often stored in spell format, allowing for overlaps between spell states. On the one hand, these kind of data provide useful information for researchers. On the other hand, the data structure is often complex and not easy to handle. The `newspell` program offers a solution to the problem of spell-data management with three important features. First, it can rank spells and cut off overlaps according to the rank order. Second, `newspell` can combine overlapping parts of spells into new categories of spells, generating entirely new states. Third, it can detect gaps in the spell data that are not yet coded. It also includes subcommands for the management of complex spell data. Spell states can be merged and filled in with information from adjacent spells, and the data can be transformed to `long` or `wide` format. The command can be used to clean data, to combine two spell-data sources that have information on different kinds of states, or to deal with spell data that are complex by survey design. `newspell` is useful for users who are not familiar with complex spell data and have little experience in Stata programming or data management. For experienced users, `newspell` saves a lot of time and coding work.

Keywords: dm0078, newspell, spell data, data management, complex data

1 Complex spell data

Biographical data gathered in surveys are a valuable source of information for researchers from different disciplines. Whether the data are collected retrospectively or prospectively, they are often stored in spell format. Spell format reduces the size of the files that store the data and usually mirrors the data collection process.

However, as soon as overlaps of spells are allowed, complex spell structures emerge. Overlaps occur either if overlaps are possible by design of the data collection process or if two sources of spell data on different kinds of states are combined (for example, spells of marital status could be combined with residency spells to analyze interdependencies of spatial mobility and marital status). Some popular surveys that include complex spell data are the Panel Study of Income Dynamics, the Health and Retirement Study, and the German Socio-economic Panel Study.

Because of the unique nature of spells, several problems and solutions have been discussed for managing this data format in Stata. Cox (2007) details the nature of spells and how to construct and manipulate them in Stata. (Many of the technical

details explained in Cox (2007) are presupposed for this article.) Two commands that help generate spells are `spell` (Cox and Goldstein 1998) and `tsspell` (Cox 2002). They allow the identification of spells or continuous sequences (given a certain condition) from the data. A command to help manage already existing spell data is `spellutil` (Leuven 2003). It allows users to merge adjacent spells, transform spell data to panel data, or join overlapping spells with metric information. `spellutil` is especially useful for spells that carry metric information, like wages.

Survival analysis is a field that often applies spell data. Cleves (1999) describes how data are `stset` for the use of survival analysis. In most cases, however, we want a noncomplex structure of the spell data before running a survival analysis. There might be exceptions, but overlaps of different spell states generally should be resolved. The command I propose here, `newspell`, is ideally suited for such a purpose. In this article, I present the `newspell` command, which simplifies management of complex spell data. In contrast to `spellutil`, `newspell` is most useful for spells defined by categorical spell states.

In the next section, I present an overview of the program along with its core features and its syntax. In the third section, I describe the functionality of the subcommands and provide examples. I conclude the article in the last section.

2 Overview of newspell subcommands

`newspell`[1] is a collection of smaller programs, each of which helps to manage complex spell data. The subcommands for `newspell` are as follows:

`merge`	merges spells of different states
`rank`	solves the problem of overlapping spells by cutting them off according to a given ranking
`fillin`	fills in certain types of spells with information from adjacent spells, according to a priority given by the user
`combine`	creates new states from any combination of overlapping spells
`gaps`	searches for gaps in spell data and fills them with spells of a new state
`towide/tolong`	transforms spell data into wide or long (panel) format, even if overlapping spells still remain

1. For the programming of `newspell`, Stata 12.1 was used.

2.1 Syntax

`newspell` *subcommand* [*if*] [*in*], *subcommand_options* `id(`*varname*`)`
`snumber(`*varname*`)` `stype(`*varname*`)` `begin(`*varname*`)` `end(`*varname*`)`
[`sort(`*varlist*`)` `ncensor(`*newvarname*`)` `missing(`# | *string*`)`
`newsnumber(`*newvarname*`)` `report`]

2.2 Shared options

Several options are shared among many of the subcommands. I describe these shared options only once, in this section, for brevity. See section 3 for details about each individual subcommand and options available only to that subcommand.

`id(`*varname*`)` specifies the variable that uniquely identifies the unit to which the spells refer. If individual survey data are used, this variable will likely refer to an individual or a household. In other contexts, it can be virtually any unit (for example, companies or countries). `id()` takes both numeric and string variables (see section 3.7 on the use of `newspell` with string variables). Together, `id()` and `snumber()` must uniquely identify the observations in the data. Missing values are not allowed. `id()` is required.

`snumber(`*varname*`)` specifies the variable that contains the number of the spell within the units of `id()`. `snumber()` must be a numeric variable. Together, `id()` and `snumber()` must uniquely identify the observations in the data. Missing values are not allowed. `snumber()` is required.

`stype(`*varname*`)` specifies the variable that contains the different spell states (or types of spells). *varname* should indicate what content the spell actually holds. `stype()` takes both string and numeric variables. Missing values are not allowed. `stype()` is required.

`begin(`*varname*`)` specifies the variable defining the beginning of a spell. Missing values are not allowed. `begin()` is required.

`end(`*varname*`)` specifies the variable defining the end of a spell. Missing values are not allowed. `end()` is required.

`sort(`*varlist*`)` indicates how the transformed data should be sorted. The transformed data will always be sorted first by the `id()` variable and then by the variables specified in `sort()`.

`ncensor(`*newvarname*`)` specifies the name of a new censor variable. The new variable contains technical censoring codes. Using the additional option `missing()` yields a more detailed code. If `ncensor()` is not used, no new censoring variable will be created.

`missing(`*#* | *string*`)` indicates whether one of the spell states defined in `stype()` is a missing-information code. This information helps in coding a more precise new censoring variable via the option `ncensor()`. `missing()` can be used only if `ncensor()` is used.

`newsnumber(`*newvarname*`)` specifies the name of the new spell number variable. The default is `newsnumber(snumber_new)`.

`report` requests a brief report about the changes made to the data. The output depends on the subcommand.

3 Subcommands

3.1 merge

`newspell merge` allows the user to merge two or more states to a common state. Adjacent spells of the same state are automatically collapsed into one spell.

Syntax

`newspell merge` [*if*] [*in*]`, merge(`*rule* [*rule* `...`]`)` [`nstype(`*newvarname*`)`]
`id(`*varname*`)` `snumber(`*varname*`)` `stype(`*varname*`)` `begin(`*varname*`)`
`end(`*varname*`)` [`sort(`*varlist*`)` `ncensor(`*newvarname*`)` `missing(`*#* | *string*`)`
`newsnumber(`*newvarname*`)` `report`]

Options

`merge(`*rule* [*rule* `...`]`)` defines the states to be joined. The rules are analogous to those for the `recode` command (see [D] **recode**). Rules should not be separated by commas, and brackets are not needed. For example, `merge(2 3=4 1 5/7 =7 8=9)` recodes the spell type variable so that states 2, 3, and 4 all receive code 4; states 1, 5, 6, and 7 are all recoded to the state with code 7; and states 8 and 9 together form the state with code 9. If the variable specified in `stype()` is a numeric variable, the value labels of the new state variable will contain the different merging rules specified in `merge()`. For the example above, the value 4 would be labeled `2 3=[4]`, the value 7 would be labeled `1 5/7 =[7]`, and the value 9 would be labeled `8=[9]`. `merge()` is required.

`nstype(`*newvarname*`)` specifies the name of the variable that contains the new type of spells. The default is `nstype(spelltype_new)`.

▷ Example

Figure 1 shows a fictional spell structure for one individual, with job history as spells and age in years as the time intervals.

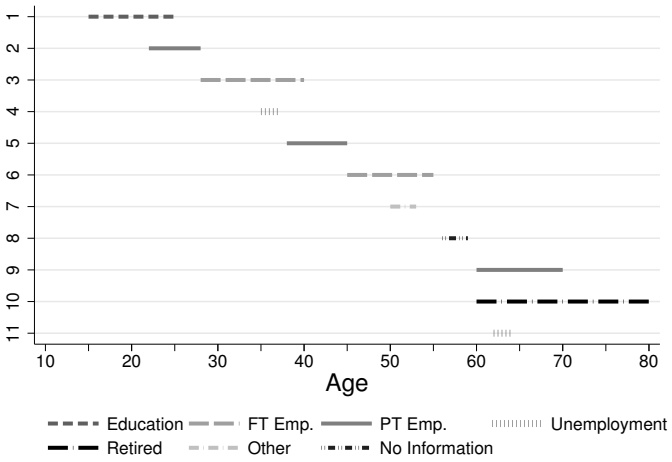

Figure 1. Original spell structure

Figure 2 shows the transformation when `newspell merge` is used to create an employment spell merging full-time and part-time employment.

```
. newspell merge, merge(2 3=2) nstype(stype_m) id(pid) stype(stype)
> snumber(n) begin(begin) end(end) newsnumber(n2) sort(begin)
```

The new variable will be assigned the value label `2 3=[2]` for value 2.

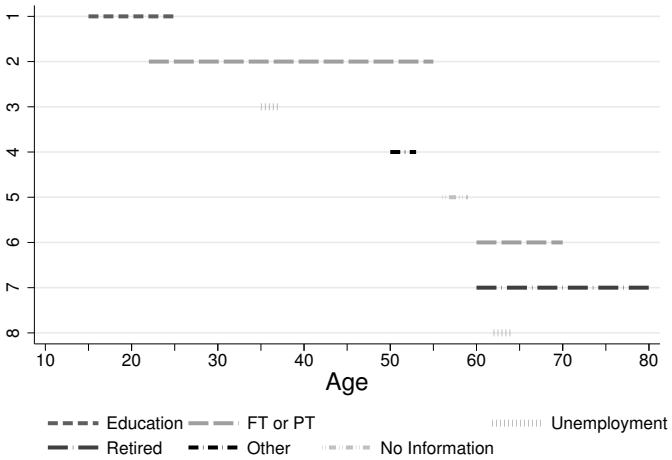

Figure 2. Full-time and part-time employment are merged to one spell

3.2 rank

`newspell rank` solves the problem of overlapping spells. A ranking of the states must be specified. Spells are cut off according to that ranking so that only one state remains per time interval. States are allowed to have equal ranking; in this case, overlaps between states of equal rank are not resolved.

Syntax

`newspell rank` [*if*] [*in*], `rank(`# [#][, # ...]`)` [`split(`*newvarname*`)`]
`id(`*varname*`)` `snumber(`*varname*`)` `stype(`*varname*`)` `begin(`*varname*`)`
`end(`*varname*`)` [`sort(`*varlist*`)` `ncensor(`*newvarname*`)` `missing(`# | *string*`)`
`newsnumber(`*newvarname*`)` `report`]

Options

`rank(`# [#][, # ...]`)` defines the ranking rule that will solve overlaps. The values of the states are inserted and separated by commas to indicate different ranks. The first state in `rank()` is assigned the highest rank; the following states receive an incrementally decreasing rank. Several states can have the same rank by writing them subsequently without a comma. All states in the data that are not specified in `rank()` are treated as being on the lowest rank (jointly). If two spells overlap, only the information of the spell with the higher rank is kept for the overlapping part. Overlaps between spells of equal rank are not resolved. `rank()` is required.

For example, `rank(2, 4 5, 9 8)` means that state 2 is ranked highest, states 4 and 5 are ranked second, states 9 and 8 are ranked third, and all other states are ranked fourth (lowest).

`split(`*newvarname*`)` specifies the name of the new variable that indicates whether and how often a spell was split into two spells during the execution of the `newspell rank` command. The default is `split(split)`.

▷ Example

The data from figure 1 are transformed so that no overlaps remain. Figure 3 shows the result. The following syntax is used:

```
. newspell rank, rank(4,1,2,3,5,6,99) id(pid) stype(stype) snumber(n)
> begin(begin) end(end) newsnumber(n2) sort(begin)
```

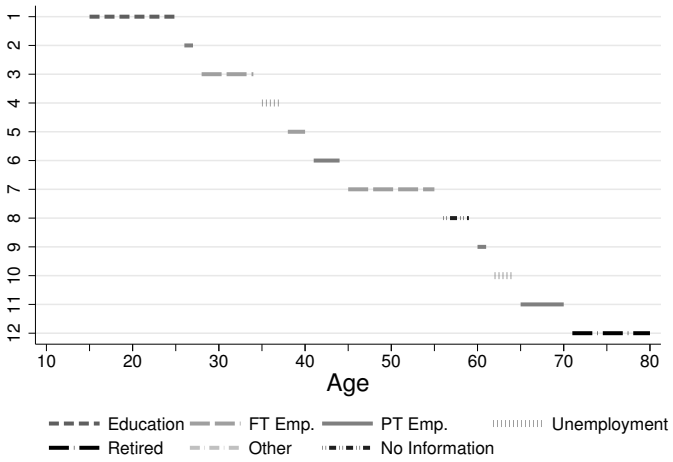

Figure 3. Overlapping spells are cut off

◁

3.3 fillin

`newspell fillin` fills all spells of a type specified by the user with information from adjacent spells. Adjacent spells can begin either before the beginning or after the end of the spell that is filled in. This is useful, for example, for filling in gaps or spells of missing information.

Syntax

`newspell fillin` [*if*] [*in*], `fill(`# | *string*[,
`pre` | `post` | `both` | `bothpost` | `rank` | `prop`]) [`prop(emp` | # #)
`overlap(one` | `whole)` `rank(`#, #, ...)] `id(`*varname*`)` `snumber(`*varname*`)`
`stype(`*varname*`)` `begin(`*varname*`)` `end(`*varname*`)` [`sort(`*varlist*`)`
`ncensor(`*newvarname*`)` `missing(`# | *string*`)` `newsnumber(`*newvarname*`)` `report`]

Options

`fill(`# | *string*[, `pre` | `post` | `both` | `bothpost` | `rank` | `prop`]`)` specifies which state to fill in. The suboptions define how the state should be filled in: `pre` will fill in the spell from the spell ending before it; `post` will fill in the spell from the spell beginning after it; `both` will fill in the spell equally from both sides. If the spell to be filled in has an unequal number of time intervals, the middle is coded from the prespell by default when `both` is used; `bothpost` requests that the postspell be used instead.

`rank` will fill in the spell from the adjacent spell that is ranked highest. `prop` will fill in the spell according to a user-specified ratio from pre- and postspells; the ratio is defined using the `prop()` option. `fill()` is required.

`prop(emp|# #)` specifies a ratio by which to use the adjacent spells to fill in a spell. `prop(emp)` requests that the empirical ratio of the lengths of the prespell and postspell be used; empirical proportions are rounded to 1% precision. `prop(# #)` requests that the specified proportions of prespell and postspell be used to fill in the spell. The two numbers used must be integer numbers that add up to 100. For example, `prop(15 85)` requests that the spell be filled in 15% from the prespell and 85% from the postspell.

`overlap(one|whole)` specifies to fill in the spell from both sides but allow overlaps. `overlap()` may be specified only if `fill(#, both)` is used. `overlap(one)` requests that only the time interval exactly in the middle of the spell overlap. `overlap(whole)` requests that the whole spell overlap with the pre- and postspell.

`rank(#, #, ...)` specifies a ranking for the states. Equal ranks are not allowed.[2] If two or more spells are adjacent to the spell to be filled in, the ranking specifies which information be used for the fill in. If `rank()` is not specified, the ranking is automatically created according to the order of the spell states in the spell state variable (`stype()`).

▷ Example

For this example, the missing-information spell from figure 1 is filled in. Figure 4 shows the result.

```
. newspell fillin, fill(99, both) rank(5,3) id(pid) stype(stype) snumber(n)
> begin(begin) end(end) newsnumber(n2) sort(begin)
```

2. This is in contrast to the `newspell rank` command, where equal ranks are allowed.

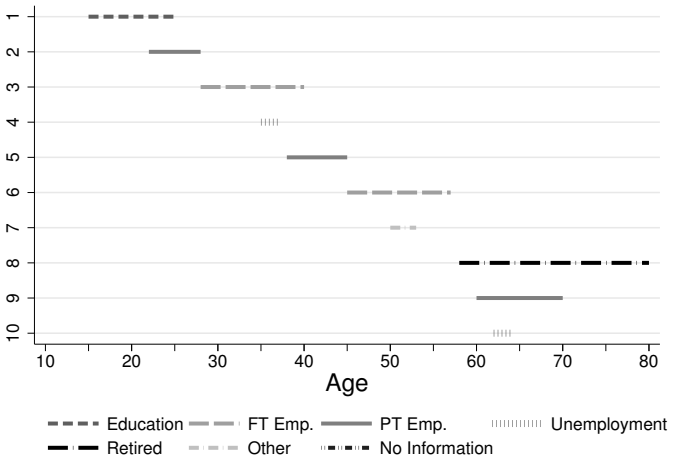

Figure 4. Spell with missing information is filled in from both sides

3.4 combine

The `newspell combine` command will create a new type of spell from overlapping parts of two or more spells. The user specifies the states that are to be combined into a new spell type. For every overlap in the data, `newspell combine` checks whether the overlapping spells (or a subset thereof) can be combined into the new type of spell according to the rules specified by the user. If this is the case, a new spell will be created for the interval of overlap. For example, the states `"married"` and `"full-time employed"` could be combined to `"married and full-time employed"`. Any combination is possible as long as there are corresponding overlaps in the data.

Syntax

`newspell combine` [*if*] [*in*], `combine(`# # [# ...]`)` `ncode(`#`)` [`keepold`]
`id(`*varname*`)` `snumber(`*varname*`)` `stype(`*varname*`)` `begin(`*varname*`)`
`end(`*varname*`)` [`sort(`*varlist*`)` `ncensor(`*newvarname*`)` `missing(`# | *string*`)`
`newsnumber(`*newvarname*`)` `report`]

Options

`combine(`# # [# ...]`)` specifies the original codes of the states to combine. Do not use commas to separate the codes. `combine()` is required.

`ncode(#)` defines the code of the new state that is created. # cannot be a code that already exists. `ncode()` is required.

`keepold` requests that the information from the overlapping parts of the original spells be kept in the data. By default, the overlaps are cut off.

Example

Using the data presented in figure 1, `newspell combine` can create a new state to indicate that a person is still working part-time after retirement. Figure 5 shows the transformation of the data after applying the following syntax.

```
. newspell combine, combine(3 5) id(pid) stype(stype) snumber(n) begin(begin)
> end(end) newsnumber(n2) sort(begin) ncode(7)
```

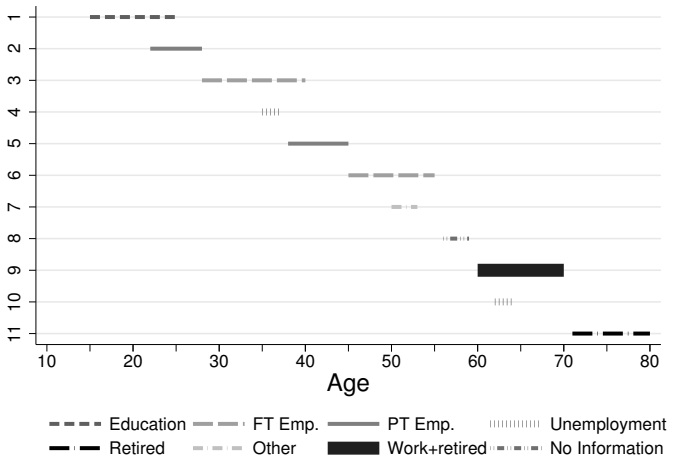

Figure 5. Part-time and retired are combined to a new state

3.5 gaps

A gap exists when a certain period in the data is not covered by a spell. `newspell gaps` detects gaps in the data and then fills them in with a new spell state.

Syntax

`newspell gaps` [*if*] [*in*] `, ncode(#)` [`first(individual|total|#)` `last(individual|total|#)`] `id(`*varname*`)` `snumber(`*varname*`)` `stype(`*varname*`)` `begin(`*varname*`)` `end(`*varname*`)` [`sort(`*varlist*`)` `ncensor(`*newvarname*`)` `missing(#|`*string*`)` `newsnumber(`*newvarname*`)` `report`]

Options

`ncode(#)` defines the code of the new state used to indicate gaps in the data. # cannot be a code that already exists. `ncode()` is required.

`first(individual|total|#)` defines the time interval at which `newspell gaps` should begin filling in gaps. `first(individual)` (the default) specifies to begin filling in gaps at the earliest time point a spell begins within one unit (or individual) defined by `id()`. `first(total)` specifies to begin filling in gaps at the earliest time point a spell begins, across all units in `id()`. `first(#)` specifies to begin filling in gaps at the time point # regardless of whether spells that are already in the data start before that point.

`last(individual|total|#)` defines the time interval at which `newspell gaps` should stop filling in gaps. `last(individual)` (the default) specifies to stop filling in gaps at the latest time point a spell ends within one unit (or individual) defined by `id()`. `last(total)` specifies to stop filling in gaps at the latest time point a spell ends, across all units in `id()`. `last(#)` specifies to stop filling in gaps at the time point # regardless of whether spells that are already in the data end after that point.

▷ Example

Figure 6 shows a fictional spell structure for one individual with job history as spells and age in years as the time intervals. The information on full-time employment shown in figure 1 has been deleted here. This created gaps in the data that `newspell gaps` can detect and fill in. Figure 7 shows the transformation of the data after the `newspell gaps` command was used.

```
. newspell gaps, ncode(15) first(individual) last(individual) id(pid)
> stype(stype) snumber(n) begin(begin) end(end) newsnumber(n2) sort(begin)
```

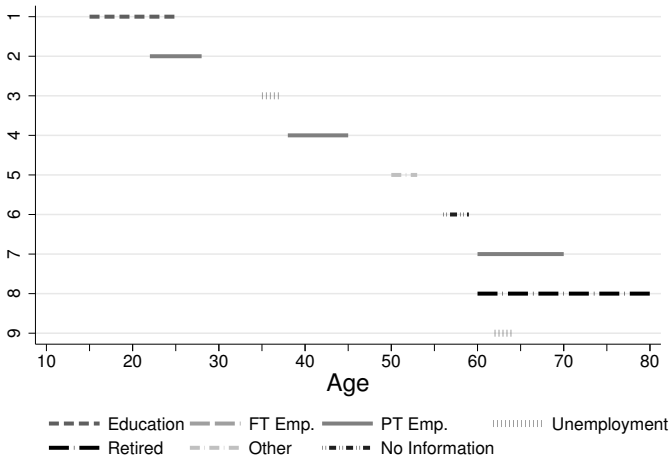

Figure 6. Original spell structure with full-time employment deleted

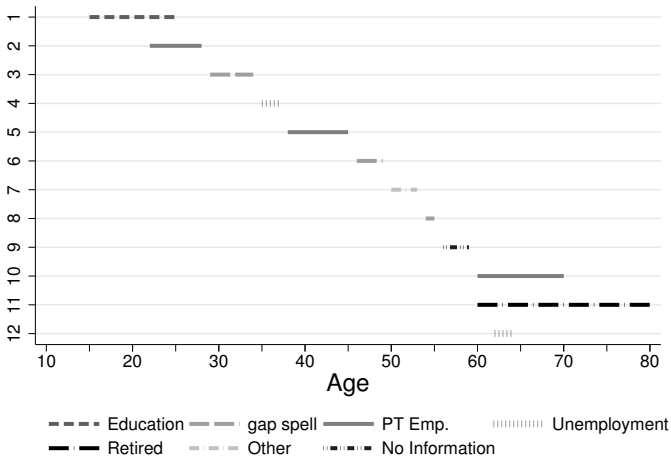

Figure 7. Gaps are detected and filled in with a gap spell

◁

3.6 towide/tolong

`newspell towide` and `newspell tolong` allow the user to transform existing spell data into data structured in the long (panel) or the wide format. For `newspell tolong`, this implies that there will be one entry per time point per unit in `id()`. The time points will range from the lowest value in option `begin()` to the highest value in `end()` for

each unit defined by `id()`. `newspell towide` creates one state variable for each time point in the spell data.[3]

`newspell tolong` is useful if spells are to be analyzed (or managed) as if they were panel data. `newspell towide` could be used before a sequence analysis. For sequence analysis, some programs require the data to be in wide format instead of spell format.

`newspell tolong` [*if*] [*in*], `time(`*newvarname*`)` [`add(`*varlist*`)` `constant(`*varlist*`)` `nooverlaps(#`[`, #, ...`]`)] `id(`*varname*`)` `snumber(`*varname*`)` `stype(`*varname*`)` `begin(`*varname*`)` `end(`*varname*`)`

`newspell towide` [*if*] [*in*], `time(`*newvarname*`)` [`add(`*varlist*`)` `constant(`*varlist*`)` `nooverlaps(#`[`, #, ...`]`)] `id(`*varname*`)` `snumber(`*varname*`)` `stype(`*varname*`)` `begin(`*varname*`)` `end(`*varname*`)`

Options

`time(`*newvarname*`)` specifies the name of the new variable that contains the single time points generated in the `tolong`/`towide` command (for example, year or month). `time()` is required.

`add(`*varlist*`)` allows the user to transform other, additional variables than the state variable from spell format to long/wide format.

`constant(`*varlist*`)` allows the user to transform other, additional variables that are constant over spells from spell format to long/wide format.

`nooverlaps(#`[`, #, ...`]`) requests that overlaps in the spells be cut off. The rank order in which this should be done can be specified according to the same rules as the `rank()` option of `newspell rank`. All existing states must be specified in `nooverlaps()`, without equal ranks, if all overlaps should be resolved. It is possible to resolve only some of the overlaps.

3. The functionality of `newspell tolong` is very similar to the `spell2panel` command of `spellutil.ado` (Leuven 2003). In contrast to `spell2panel`, `newspell tolong` focuses on spell states that are categorical and not metric.

Example

To illustrate the use of `newspell tolong`, the first three spells from figure 1 are transformed from spell format to long format.

```
. use "art.dta"
. keep if n <=3
(8 observations deleted)
. list
```

	pid	begin	end	stype	n
1.	111	15	25	1	1
2.	111	22	28	3	2
3.	111	28	40	2	3

```
. newspell tolong, time(age) id(pid) snumber(n) stype(stype) begin(begin)
> end(end) nooverlaps(1,3,2)
. list
```

	pid	age	stype
1.	111	15	1
2.	111	16	1
3.	111	17	1
4.	111	18	1
5.	111	19	1
6.	111	20	1
7.	111	21	1
8.	111	22	1
9.	111	23	1
10.	111	24	1
11.	111	25	1
12.	111	26	3
13.	111	27	3
14.	111	28	3
15.	111	29	2
16.	111	30	2
17.	111	31	2
18.	111	32	2
19.	111	33	2
20.	111	34	2
21.	111	35	2
22.	111	36	2
23.	111	37	2
24.	111	38	2
25.	111	39	2
26.	111	40	2

◁

Example

To illustrate the use of `newspell towide`, the first three spells from figure 1 are transformed from spell format to wide format.

```
. use "art.dta", clear
. keep if n <=3
(8 observations deleted)
. list
```

	pid	begin	end	stype	n
1.	111	15	25	1	1
2.	111	22	28	3	2
3.	111	28	40	2	3

```
. newspell towide, time(age) id(pid) snumber(n) stype(stype) begin(begin)
> end(end) nooverlaps(1,3,2)
. list, linesize(59)
```

1.

pid	stype_15	stype_16	stype_17	stype_18
111	1	1	1	1

stype_19	stype_20	stype_21	stype_22
1	1	1	1
stype_23	stype_24	stype_25	stype_26
1	1	1	3
stype_27	stype_28	stype_29	stype_30
3	3	2	2
stype_31	stype_32	stype_33	stype_34
2	2	2	2
stype_35	stype_36	stype_37	stype_38
2	2	2	2

stype_39	stype_40
2	2

◁

3.7 Using string variables with newspell

`newspell` supports the use of string variables for both `id()` and `stype()`. If the identifier variable (`id()`) is a string variable, nothing changes for the user. However, if the variable identifying the spell states (`stype()`) is a string variable, some rules must be followed to ensure `newspell` works as intended.

The first rule for a string variable specified in `stype()` is that it cannot contain missing values, as with numeric variables. Missing values for string variables are empty strings, containing no character or space: `""`. The second rule is that the variable cannot

contain values that contain either only spaces (for example, `" "`), only one comma (`","`), or only one equal sign (`"="`). However, spaces, commas, and equal signs are allowed in combination with other characters. For example, the value `"married and living apart"` is allowed. The same goes, for example, for `"part-time,full-time employment"` or `"=20hours per week"`. This means that, not counting the specific exceptions, `newspell` can deal with all kinds of strings stored in the spell state variable, as long as Stata can deal with the strings.

If the spell state variable is a string variable, all values of the variable that are specified in the options of one of the subcommands of `newspell` must be enclosed in quotes, even if the value does not contain spaces. As in all dealings with strings, users must be precise in specifying the string values. If a space is left out, even at the end, `newspell` will recognize it as a different state and might produce an error message.

▷ Example

For the purpose of showing the correct use of `newspell` with a spell state variable that contains strings, I generated data that mirror the example from figure 1 with string codes.

```
. generate pidstring = string(pid)
. generate stypestring = "Education"         if stype ==1
(10 missing values generated)
. replace stypestring = "FT, PT"             if stype ==2
(2 real changes made)
. replace stypestring = "PT Emp."            if stype ==3
(3 real changes made)
. replace stypestring = "Unemployment"       if stype ==4
stypestring was str9 now str12
(2 real changes made)
. replace stypestring = "Retired"            if stype ==5
(1 real change made)
. replace stypestring = "Other"              if stype ==6
(1 real change made)
. replace stypestring = "99=No Information"  if stype ==99
stypestring was str12 now str17
(1 real change made)
```

`newspell rank` is correctly specified in the following way:

```
. newspell rank, rank("Unemployment","Education","FT, PT","PT Emp.","Retired",
> "Other","99=No Information") id(pidstring) stype(stypestring) snumber(n)
> begin(begin) end(end) newsnumber(n2) sort(begin) ncensor(ncens)
> missing("99=No Information")
```

This is how a state can be filled in using a string variable:

```
. newspell fillin, fill("99=No Information", both) rank("Retired","PT Emp.")
> id(pidstring) stype(stypestring) snumber(n) begin(begin) end(end)
> newsnumber(n2) sort(begin)
```

This is how two states, stored as strings, can be merged with `newspell`:

```
. newspell merge, merge("FT, PT" "PT Emp." ="FT, PT") nstype(stypestringm)
> id(pidstring) stype(stypestring) snumber(n) begin(begin) end(end)
> newsnumber(n2) sort(begin)
```

◁

4 Conclusion

In this article, I presented the user-written `newspell` command, which can be used to conduct different important data-management tasks when dealing with spell data. These tasks are merging or cutting off spells, finding gaps in the data, filling those gaps, transforming spell data to wide or long format, and creating completely new states from overlapping spells. Using `newspell`, each of these tasks requires only one line of code. Spell-data management therefore becomes easier for inexperienced users and saves a lot of time for experts.

To the best of my knowledge, there is no comparable tool in any statistics package that has the same range of functions for spell-data management. I am open to suggestions from users of `newspell` with regard to new functions and other improvements so that I can update and improve it on a regular basis.

5 Acknowledgments

I thank Sebastian Beil, Lea Kröger, Jan Goebel, and Oliver Winkler for suggestions and bug-reports on earlier versions of the program. I am especially grateful for the helpful comments of an anonymous reviewer. The work was partly funded by a grant from the European Research Council (ERC: 313532).

6 References

Cleves, M. 1999. FAQ: How do I convert my spell-type data into a survival dataset? http://www.stata.com/support/faqs/statistics/stset-spell-type-data/.

Cox, N. J. 2002. tsspell: Stata module for identification of spells or runs in time series. Statistical Software Components S426901, Department of Economics, Boston College. http://econpapers.repec.org/software/bocbocode/s426901.htm.

———. 2007. Speaking Stata: Identifying spells. *Stata Journal* 7: 249–265.

Cox, N. J., and R. Goldstein. 1998. spell: Stata module for identification of spells or runs of similar values. Statistical Software Components S344901, Department of Economics, Boston College.
http://econpapers.repec.org/software/bocbocode/s344901.htm.

Leuven, E. 2003. spellutil: Stata module of utilities for the manipulation of timespan data. Statistical Software Components S431701, Department of Economics, Boston College. http://econpapers.repec.org/software/bocbocode/s431701.htm.

About the author

Hannes Kröger is a research assistant in the SESandHealth project at the European University Institute in Florence. His research interests include health and labor market sociology, statistical and survey methods, and structural equation modeling. In his dissertation, he investigated the influence of social context on health selection processes on the labor market.