Berlin Social Science Center

Splitting Spells Using Very Large or Daily Data Sets

The stata syntax files splitspells.do and combispells.do

Klaudia Erhardt, German Institute for Economic Research (DIW), departement SOEP

Ralf Künster, Social Science Research Center Berlin (WZB), department NEPS

German Stata Users Group Meeting, June 13th, 2014 in Hamburg

Outline:

- What is spell splitting and why is it needed?
- The functionality of splitspells.do
- combispells.do
  - What is combispells.do used for?
  - The functionality of combispells.do (only if enough time)
- How to use the do files and where to download them

Outline:

- What is spell splitting and why is it needed?
- The functionality of splitspells.do
- combispells.do
  - What is combispells.do used for?
  - The functionality of combispells.do (only if enough time)
- How to use the do files and where to download them

Spell Data:

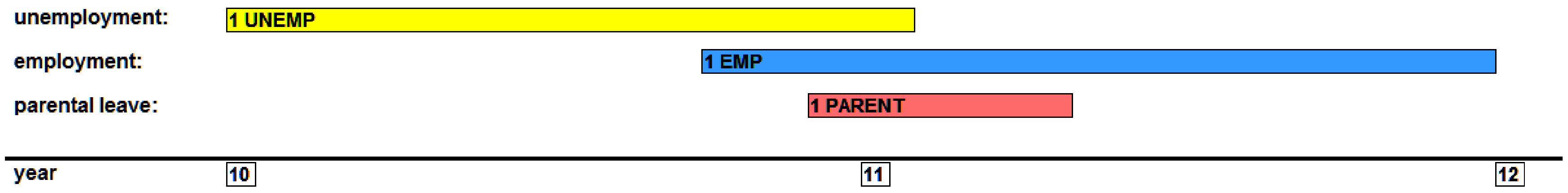
- one or more record(s) per case
- one record represents one episode (spell)
- a spell is characterized by five attributes:
  - ✓ case-ID
  - ✓ spell-ID within the case
  - ✓ start date and end date (start time unit; end time unit) with end date >= begin date
  - ✓ spell type

  No missings in those 5 variables allowed
- Spell data may contain much more additional variables.

Basically, spell data represent time spans of certain qualities. Life course studies or employment histories are typical examples in the social sciences.

Berlin Social Science Center

Example: a case with 3 spells of different spell types



Why split spell data?

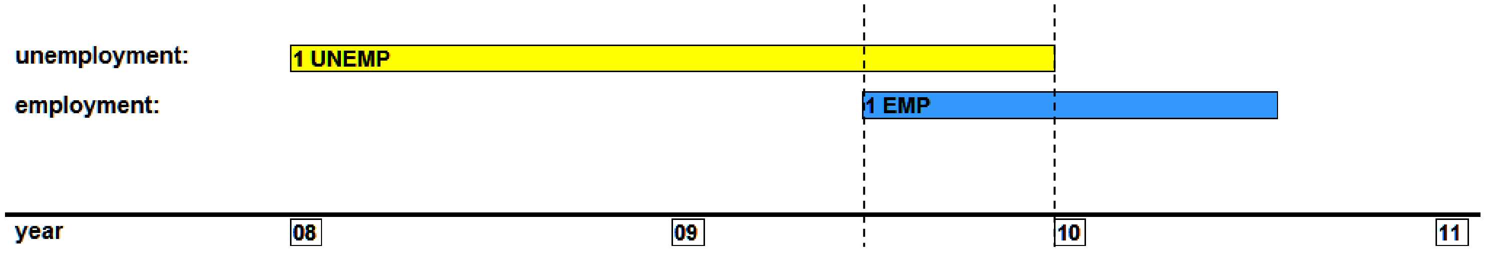
Spell data mostly represent (partly) overlapping episodes: real-life histories do not consist of sequences of unique status

but:

Most methods to analyse episode data assume that the state of a respondent (resp. case) at a given point of time is known and unique:

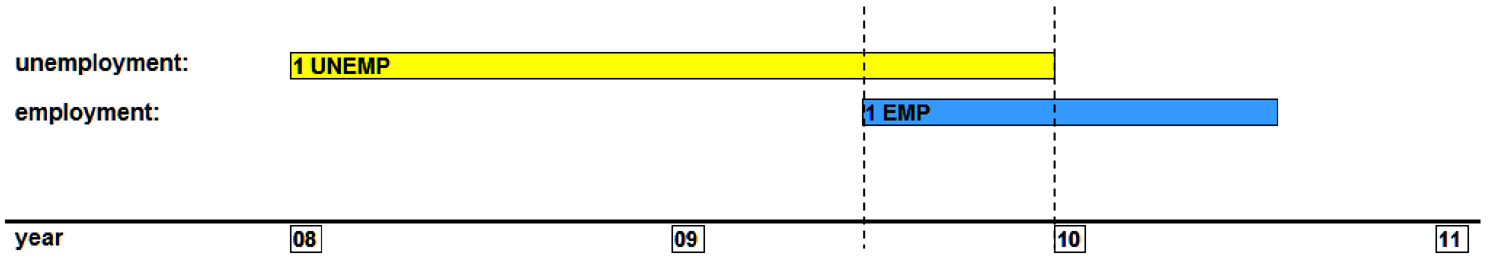
- Exploring life courses by calculating state distributions
- Analysing the duration spent in a certain state / the duration until a transition from one state to a certain other state occurs
- Performing sequence data analysis

life course data with overlapping spells

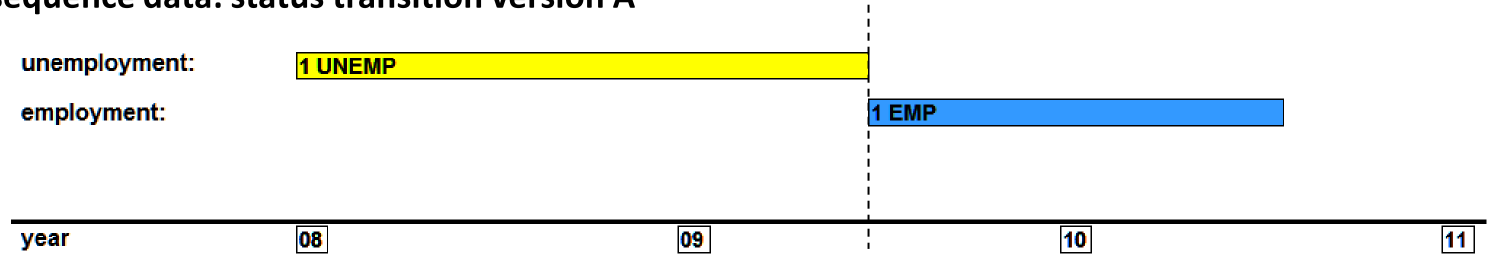


When did the transition from unemployment to employment happen?

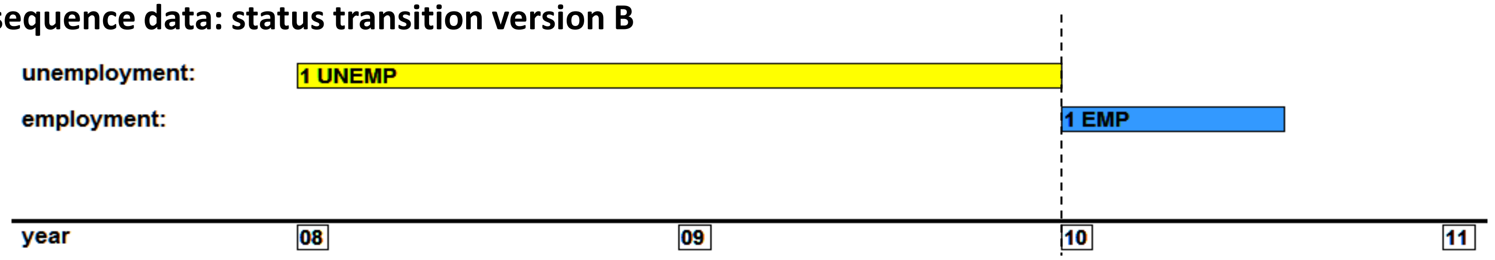
life course data with overlapping spells



sequence data: status transition version A



sequence data: status transition version B



Berlin Social Science Center

Two consecutive jobs to be done in order to produce sequence data:

- Identify overlapping time spans and transform the data structure, i.e. cut the original spells in pieces such that every piece is unambiguously attributable to one ore several spell types (spell splitting).
- Select which of the splitted spells are to be kept in case of isochronicity, usually by means of a priority list of the different spell types and additional criteria for the choice between isochronal spells of the same type.

splitspells.do does the first job

combispells.do helps with the second job: it enables the detailed exploration of the parallel (isochronal) states that occur empirically.

Outline:

✓ What is spell splitting and why is it needed?

- **The functionality of splitspells.do**
- combispells.do
  - What is combispells.do used for?
  - The functionality of combispells.do (only if enough time)
- How to use the do files and where to download them

Berlin Social Science Center

Part One

Splitting spells with splitspells.do

A common and well known way to transform spell data so that no partly overlapping spells remain is to split the data in pieces of one time unit

- Simple and short syntax, easy to apply:
  just calculate the duration (in time units) and expand by duration
- but produces a maximum of additional records!
  therefore not feasible for large datasets or daily measurement

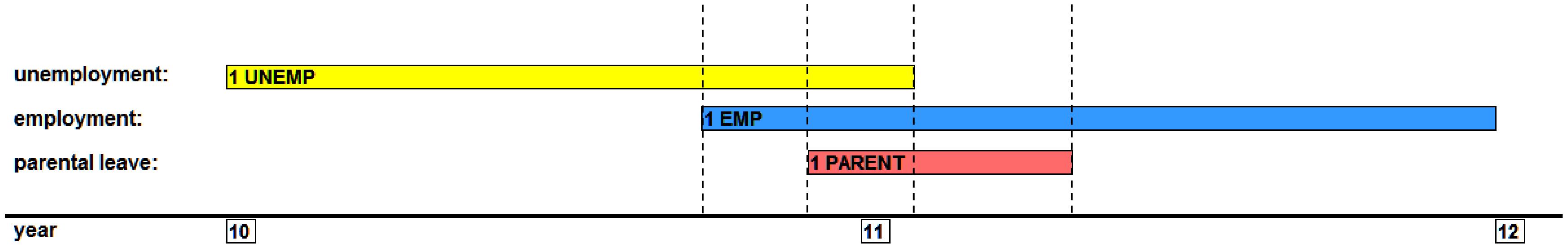
The implicit assumption of spellsplitting by duration:

- State transitions can occur at each point of time of the observed period therefore it has to be looked at each point of time to observe parallelisms of spells

  → All spells have to be split at each point of time

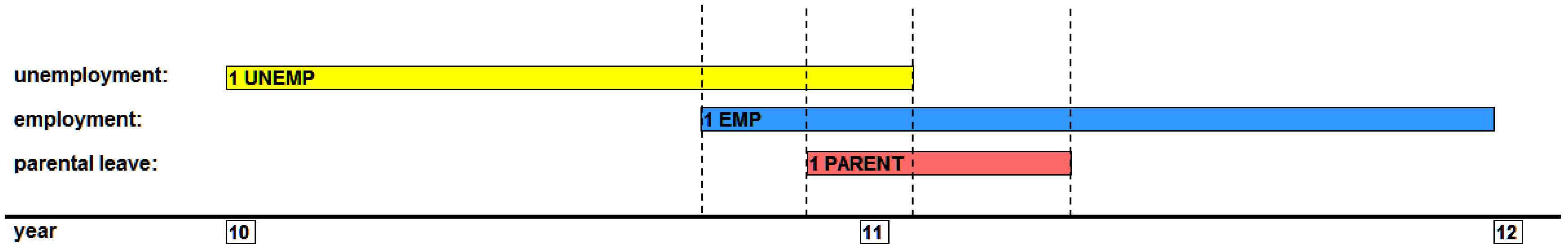
This assumption is much overcautious!

In fact state transitions occur only at the start dates or one time-unit after the end dates of spells within a case

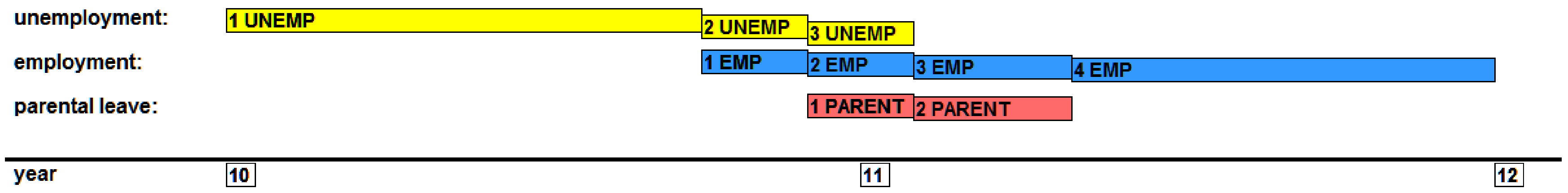


Splitting spells by comparing start and end dates (1)

life course data with overlapping spells



spells split by start and end dates of other spells (of the same case)



Splitting the data according to the start and end dates of other spells within the case:

- Syntax long and complex, easy to apply only when using splitspells.do ☺
- Produces the least possible number of additional records, therefore appropriate even for very large datasets or daily measurement

WZB
Berlin Social Science Center

DIW SOEP

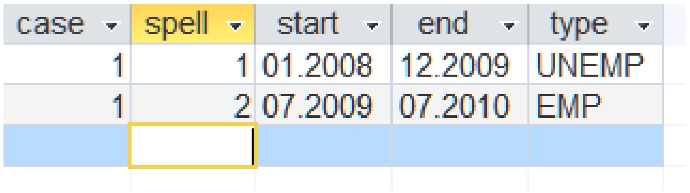
Example: SIAB-r-7508

(SUF integrated employment biographies from 1975 to 2008 of the German Federal Employment Agency)

Number of cases:	~ 1.500.000
Number of spells:	~ 30.000.000

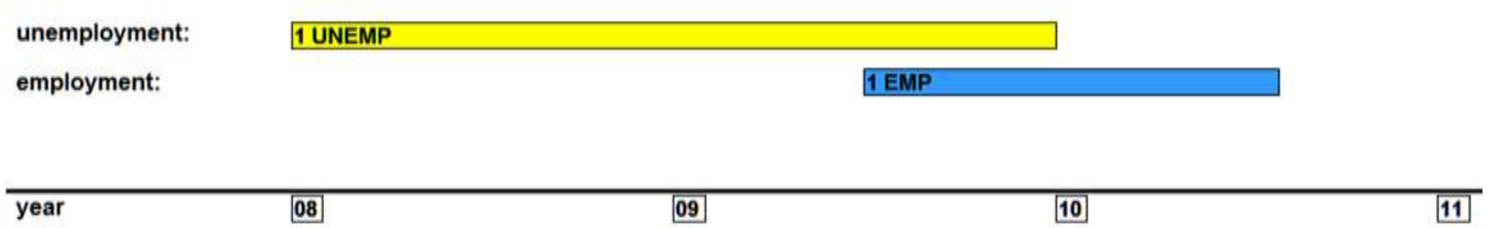
Number of spells after splitting by duration:	~ 12.000.000.000
Number of spells after splitting with splitspells.do:	~ 35.000.000

Splitting spells by comparing start and end dates (1)

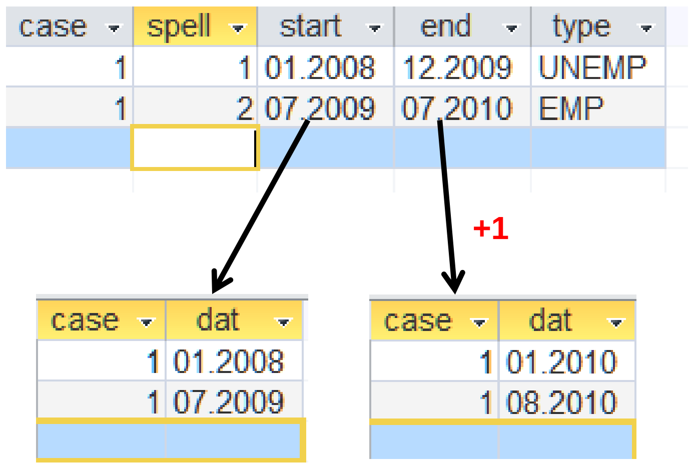


case	spell	start	end	type
1	1	01.2008	12.2009	UNEMP
1	2	07.2009	07.2010	EMP

The data set before splitting



Splitting spells by comparing start and end dates (1)



case	spell	start	end	type
1	1	01.2008	12.2009	UNEMP
1	2	07.2009	07.2010	EMP

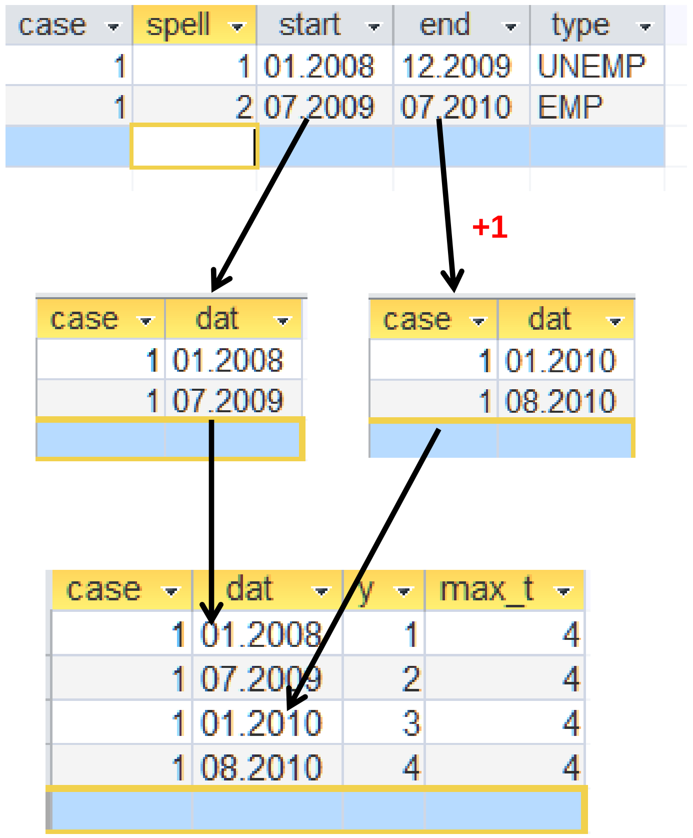
case	dat
1	01.2008
1	07.2009

case	dat
1	01.2010
1	08.2010

The data set before splitting

Extract the start and end (+1) dates into separate data files

Splitting spells by comparing start and end dates (1)



case	spell	start	end	type
1	1	01.2008	12.2009	UNEMP
1	2	07.2009	07.2010	EMP

The data set before splitting

case	dat
1	01.2008
1	07.2009

case	dat
1	01.2010
1	08.2010

Extract the start and end (+1) dates into separate data files

case	dat	y	max_t
1	01.2008	1	4
1	07.2009	2	4
1	01.2010	3	4
1	08.2010	4	4

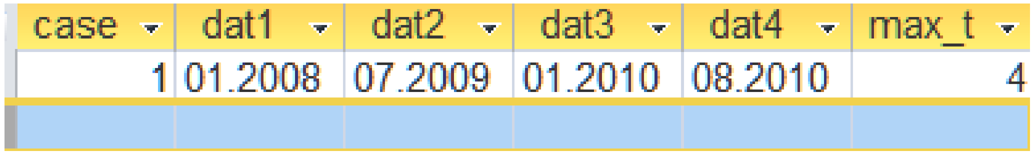
append them, drop duplicates, create a record counter, count the maximal number of date specifications in a case

```
use file_A, clear
append using file_B
sort case dat
duplicates drop case dat, force
by case: gen int y = _n
by case: gen int z = _N
sum z
max_t = r(max)
```

Splitting spells by comparing start and end dates (2)

Reshape the „datum“ file from long to wide

```
reshape wide dat, i(case) j(y)
```

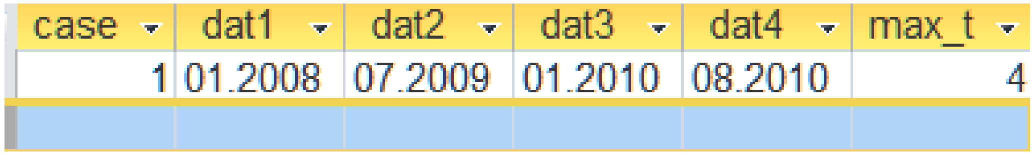


case	dat1	dat2	dat3	dat4	max_t
1	01.2008	07.2009	01.2010	08.2010	4

Splitting spells by comparing start and end dates (2)

Reshape the „datum“ file from long to wide

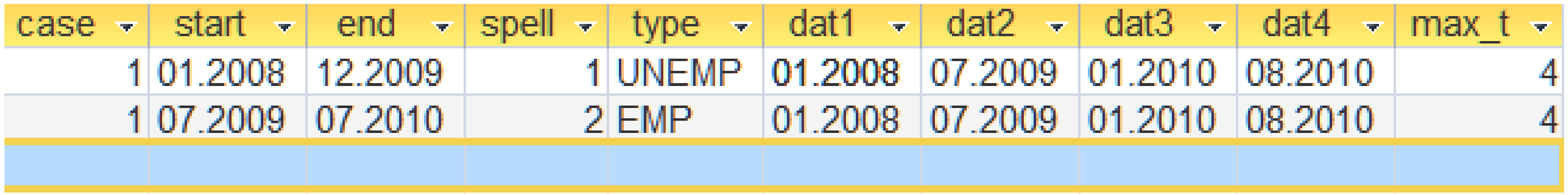
reshape wide dat, i(case) j(y)



case	dat1	dat2	dat3	dat4	max_t
1	01.2008	07.2009	01.2010	08.2010	4

Merge the „datum“ file casewise to each spell of the original file

merge m:1 case using datum_wide'



case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009	01.2010	08.2010	4
1	07.2009	07.2010	2	EMP	01.2008	07.2009	01.2010	08.2010	4

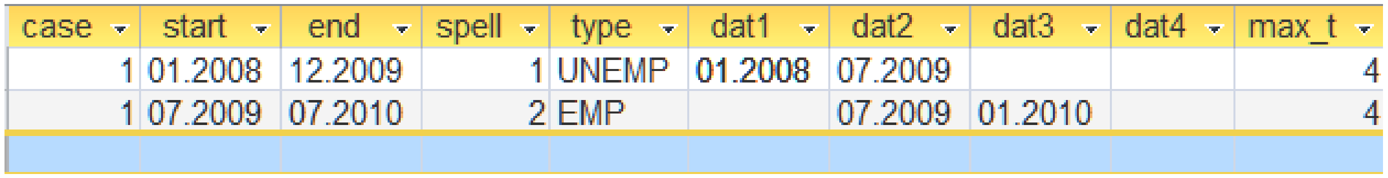
WZB Berlin Social Science Center

DIW SOEP

Splitting spells by comparing start and end dates (3)

Delete all date specifications that do not lie within the time span of the spell

```
gen n_epidat = 0
forvalues i = 1(1) max_t {
                replace dat`i' = . if dat`i' < start | dat`i' > end
                replace n_epidat = n_epidat + 1 if dat`i' != .
}
```



case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4

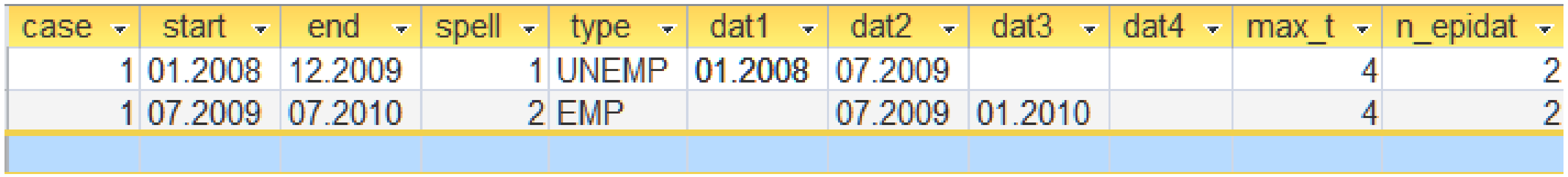
WZB Berlin Social Science Center

DIW SOEP

Splitting spells by comparing start and end dates (3)

Delete all date specifications that do not lie within the time span of the spell and count the date specifications that lie within the time span of the spell

```
gen n_epidat = 0
forvalues i = 1(1) max_t {
            replace dat`i' = . if dat`i' < start | dat`i' > end
            replace n_epidat = n_epidat + 1 if dat`i' != .
}
```



case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t	n_epidat
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	2
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2

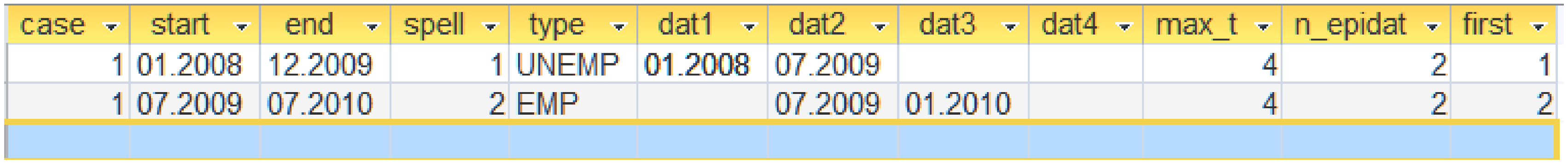
Splitting spells by comparing start and end dates (3)

Delete all date specifications that do not lie within the time span of the spell and count the date specifications that lie within the time span of the spell

```
gen n_epidat = 0
forvalues i = 1(1) max_t {
            replace dat`i' = . if dat`i' < start | dat`i' > end
            replace n_epidat = n_epidat + 1 if dat`i' != .
}
```

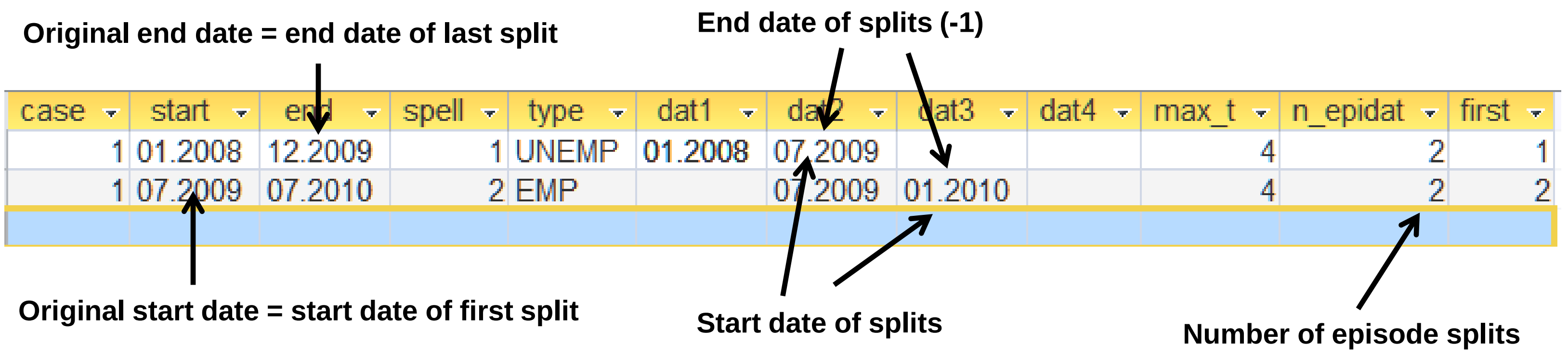
Store the index of the *first* date specification that lies within the time span of the spell.

```
gen first = 0
forvalues i = 1(1) max_t {
            replace first = `i' if dat`i' != . & first == 0
}
```



case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t	n_epidat	first
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	2	1
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	2

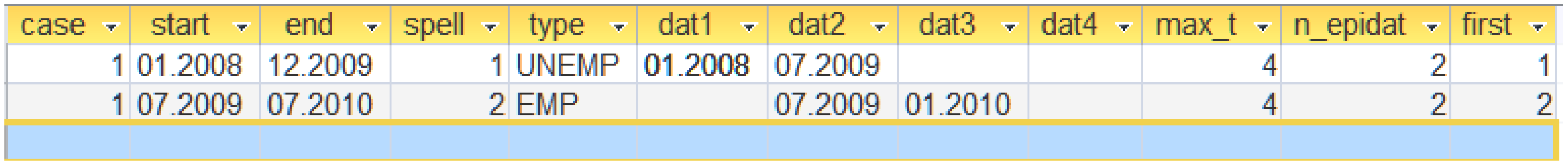
Splitting spells by comparing start and end dates (4)


Original end date = end date of last split
End date of splits (-1)
Original start date = start date of first split
Start date of splits
Number of episode splits

case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t	n_epidat	first
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	2	1
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	2

At this stage we already have all information needed to create the splitted spells !

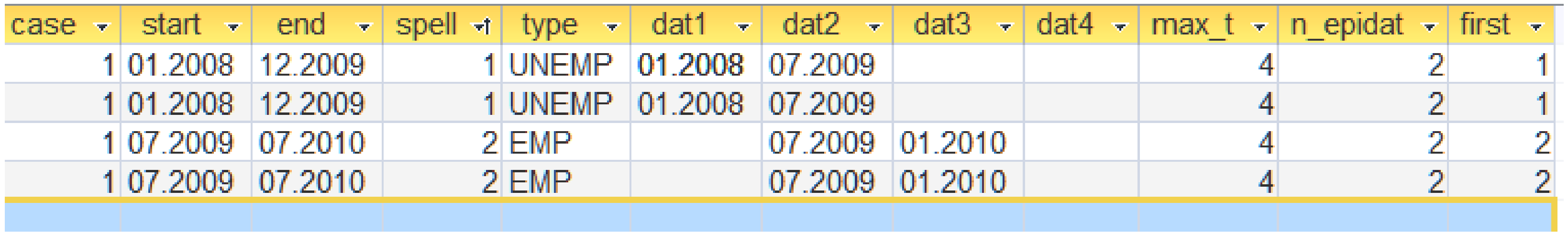
Splitting spells by comparing start and end dates (5)



case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t	n_epidat	first
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	2	1
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	2

Expand by number of episode splits

expand n_epidat

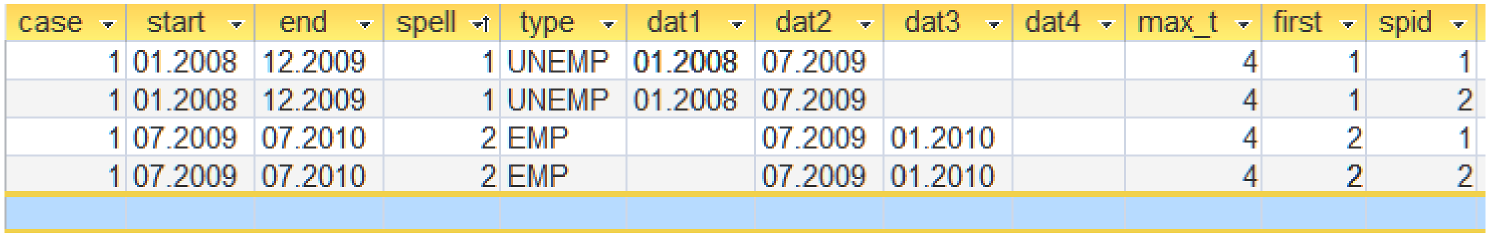


case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t	n_epidat	first
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	2	1
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	2	1
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	2
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	2

Splitting spells by comparing start and end dates (6)

Create an additional spell number for the splitted spells

```
sort case spell
by case spell: gen spid = _n
```



case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t	first	spid
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	1	1
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	1	2
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	1
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	2

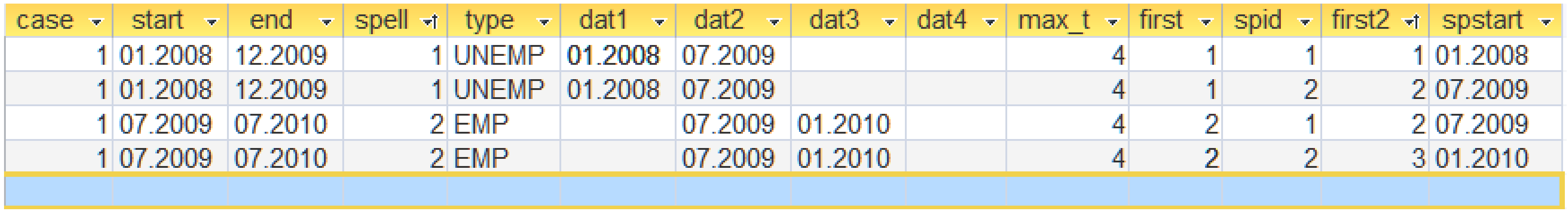
Splitting spells by comparing start and end dates (6)

Create an additional spell number for the splitted spells

```
sort case spell
by case spell: gen spid = _n
```

Identify the index of the „dat“-variable that holds the start value of the split and assign the value to the start variable of the split

```
gen first2 = first + spid – 1
gen spstart = .
forvalues i = 1(1) max_t {
		replace spstart = dat`i' if `i' == first2
}
```



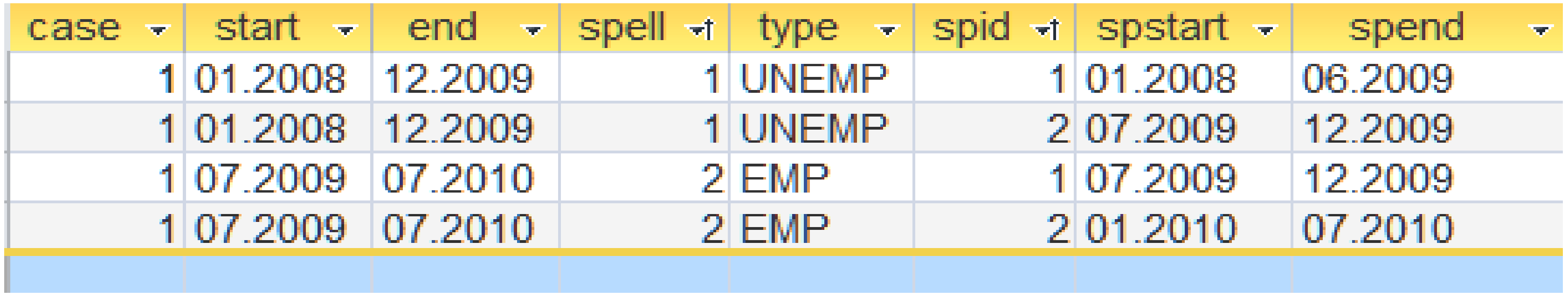
case	start	end	spell	type	dat1	dat2	dat3	dat4	max_t	first	spid	first2	spstart
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	1	1	1	01.2008
1	01.2008	12.2009	1	UNEMP	01.2008	07.2009			4	1	2	2	07.2009
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	1	2	07.2009
1	07.2009	07.2010	2	EMP		07.2009	01.2010		4	2	2	3	01.2010

WZB Berlin Social Science Center | DIW SOEP

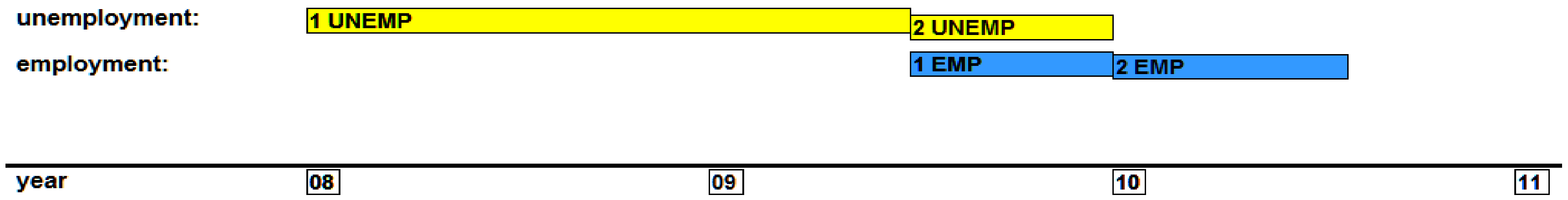
Splitting spells by comparing start and end dates (7)

Create the end date of the splitted spells

```
gen spend = .
by case spell: replace spend = spstart[_n+1] – 1
replace spend = end if spend == .
```



case	start	end	spell	type	spid	spstart	spend
1	01.2008	12.2009	1	UNEMP	1	01.2008	06.2009
1	01.2008	12.2009	1	UNEMP	2	07.2009	12.2009
1	07.2009	07.2010	2	EMP	1	07.2009	12.2009
1	07.2009	07.2010	2	EMP	2	01.2010	07.2010

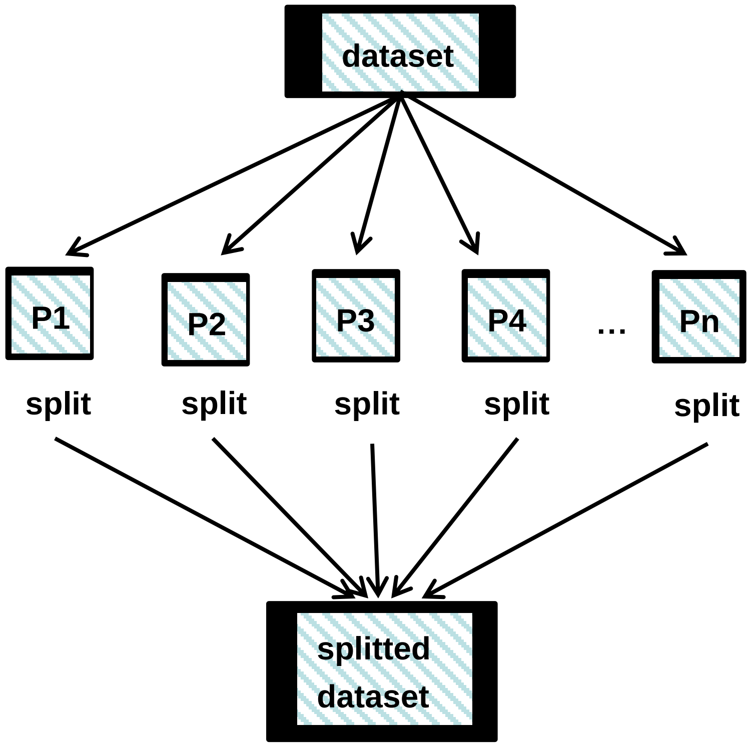


At the end of the program several spell count variables are generated and then:

S P L I T T I N G   C O M P L E T E D !

Berlin Social Science Center

Division into a user-defined number of „portions“: an additional feature for the treatment of large data sets



Outline:

- ✓ What is spell splitting and why is it needed?
- ✓ The functionality of splitspells.do
- **combispells.do**
    - What is combispells.do used for?
    - The functionality of combispells.do (only if enough time)
- How to use the do files and where to download them

WZB
Berlin Social Science Center

DIW SOEP

Part Two

combispells.do - combining information from different spells

Features:

- combispells.do creates two variables – string and labelled numeric – that show the (unique) occurences of the categories of a nominal variable within groups of records, e.g.:
  - ✓ spell types over isochronal spells of a case
  - ✓ spell types over all spells of a case
  - ✓ nationalities over households
  - ✓ ...
- Developped in the context of spell data, but applicable also to other kind of groupable data

Fields of application

- Especially useful if the nominal variable (e.g. spell type variable) has more than four or five different categories which occur in divers combinations within the record groups
- Main field of application: data exploration, e.g.:
  - ✓ explore contemporaneous spell types
  - ✓ find implausible isochronal spell type combinations
  - ✓ identify typical spell type combinations for different sample groups
- Also useful for keeping the information on which spell types have been dropped due to the application of a priority list to produce sequence data

What does combispells.do do? (1)

The user hands two lists over to the program:

- ✓ a list of numeric values of the spell type variable (the nominal variable) as they appear in the data
- ✓ a list of corresponding short labels

Example: spelltypes of the SIAB-SUF with user-invented short labels

```
local typcodes "1 2 3 4"
local shortstrings "BeH LeH ASU LHG‟
```

Take care that short labels are not contained in each other (as would be “EMP” and “UEMP”) to prevent problems with formulating unerring if-conditions

What does combispells.do do? (2)

From these lists, two variables are generated:

- ✓ a new spell type (nominal) variable whose values are the values of the exponential function 2 ^ x (two to the power of x)

  resulting values of the example: 1, 2, 4, 8

- ✓ a string variable whose values are the short labels from the above user defined list

  resulting values of the example: BeH, LeH, ASU, LHG

These two variables are the resource variables of the target combination variables that will be created.

Why these transformations?

With these transformations we can simply add the values of the trans-formed variables over the defined record groups, here: isochronal spells of a case, to get an indicator of the combinations

Strictly speaking: each value is added **only once** to the sum, no matter how often it occurs within the group

In case of the numeric combination variable it is a real addition in the mathematical sense of the word

In case of the string combination variable the programm collects the values in a string, adding an intermittent „+“-sign for better readability

The string combination variable

Example of a value: "BeH+LeH+ASU"

- The string combination variable is mainly useful for human eyes
- Therefore: output is important:
  - ✓ use fre.ado instead of tab1
  - ✓ increase linesize
- To select certain values use string functions such as strpos()

The string combination variable
Example: finding implausible combinations

(Example from SOEP spell file “artkalen”)

```
. fre combistr if length(combistr) > 28, nowrap all

combistr
-----------------------------------------------------------------------------------------------
                                                   |      Freq.    Percent      Valid       Cum.
---------------------------------------------------+-------------------------------------------
Valid   VOZT+BtAB+HAUS+BtEAB+FtBIL+MINIJ           |          6      26.09      26.09      26.09
        VOZT+KURZ+TZ+BtAB+RENTE+ELTZ+WZF+HAUS      |          8      34.78      34.78      60.87
        VOZT+TZ+BtAB+ARBL+ELTZ+SCHUL+WZF+HAUS+SONS |          9      39.13      39.13     100.00
        Total                                      |         23     100.00     100.00
-----------------------------------------------------------------------------------------------
```

The string combination variable
Example: finding certain combinations

(Example from SOEP spell file “artkalen”)

```
fre combistr if strpos(combistr, "VOZT") > 0 & strpos(combistr,
"MINIJ") > 0 & endepi - begepi > 3, nowrap all
```

```
combistr
_________________________________________________________________________
                                  |    Freq.    Percent     Valid     Cum.
__________________________________|______________________________________
Valid   VOZT+HAUS+MINIJ           |      120       9.37      9.37     9.37
        VOZT+MINIJ                |     1094      85.40     85.40    94.77
        VOZT+RENTE+MINIJ          |        9       0.70      0.70    95.47
        VOZT+SCHUL+MINIJ          |       42       3.28      3.28    98.75
        VOZT+TZ+HAUS+MINIJ        |        4       0.31      0.31    99.06
        VOZT+TZ+MINIJ             |       12       0.94      0.94   100.00
        Total                     |     1281     100.00    100.00
__________________________________|______________________________________
```

The numeric combination variable

- each value stands for a unique combination of values of the spell type (nominal) variable: this is why we recoded the original variable with values of an exponential function
- the numeric combination variable gets labelled automatically, so that the user can display the meaning of the numeric values
- additionally a label do-file is generated that can be used as a lookup table

The numeric combination variable
Example: deleting specific combinations

(Example from SOEP spell file “artkalen”)

```
. fre combi if length(combistr) > 28, nowrap all

combi
---------------------------------------------------------------------------------------------------------
                                                        |      Freq.    Percent      Valid        Cum.
---------------------------------------------------------------------------------------------------------
Valid   879   VOZT+KURZ+TZ+BtAB+RENTE+ELTZ+WZF+HAUS      |          8      34.78      34.78       34.78
        3037  VOZT+TZ+BtAB+ARBL+ELTZ+SCHUL+WZF+HAUS+SONS |          9      39.13      39.13       73.91
        29193 VOZT+BtAB+HAUS+BtEAB+FtBIL+MINIJ           |          6      26.09      26.09      100.00
        Total                                            |         23     100.00     100.00
---------------------------------------------------------------------------------------------------------

. drop if combi == 3037
```

drop only after having examined the case(s) !

Outline:

- ✓ What is spell splitting and why is it needed?
- ✓ The functionality of splitspells.do
- ✓ combispells.do
    - What is combispells.do used for?
    - The functionality of combispells.do (only if enough time)
- **How to use the do files and where to download them**

Part Three

How to use the do-files
and
where to get them

User specific parameters must be set in the "Definition Part" at the beginning of the syntax

- The definition part has three sections:
  - ✓ a compulsory section that must be adapted to the user specific environment
  - ✓ an optional section that may be adapted to the wishes of the user
  - ✓ a section for automatic parameter setting that must not be changed.
- Ample comments and filled-in examples help to define the parameters
- Apart from the adaptations in the definition part **no further adjustments of the syntax** are needed

Download

- http://www.diw.de/spelljobs
- http://www.wzb.eu/en/persons/ralf-kuenster
  → Do-File Downloads

Email contact

- kerhardt@diw.de
- ralf.kuenster@wzb.eu

Forthcoming

Erhardt/Künster: Das Splitten von Episodendaten mit Stata, FDZ-Methodenreport, IAB Nürnberg

free download:
http://fdz.iab.de/ → Publikationen → FDZ Methodenreporte

Thanks for your attention.
Any questions?