Erhardt, Klaudia

**Research Report**

# SOEP-Metafile.do - ein Stata-Do-File zur Generierung eines Metafiles zu den SOEP-Daten: Anwendung des Do-Files und Dokumentation des resultierenden Metafiles

SOEP Survey Papers, No. 534

**Provided in Cooperation with:**
German Institute for Economic Research (DIW Berlin)

*Suggested Citation:* Erhardt, Klaudia (2018) : SOEP-Metafile.do - ein Stata-Do-File zur Generierung eines Metafiles zu den SOEP-Daten: Anwendung des Do-Files und Dokumentation des resultierenden Metafiles, SOEP Survey Papers, No. 534, Deutsches Institut für Wirtschaftsforschung (DIW), Berlin

This Version is available at:
https://hdl.handle.net/10419/180391

SOEP The German Socio-Economic Panel study

534

# SOEP Survey Papers

## Series G – General Issues and Teaching Materials

SOEP — The German Socio-Economic Panel study at DIW Berlin 2018

# SOEP-Metafile.do – ein Stata-Do-File zur Generierung eines Metafiles zu den SOEP-Daten

Klaudia Erhardt

DIW SOEP

Running since 1984, the German Socio-Economic Panel study (SOEP) is a wide-ranging representative longitudinal study of private households, located at the German Institute for Economic Research, DIW Berlin.

The aim of the SOEP Survey Papers Series is to thoroughly document the survey's data collection and data processing. The SOEP Survey Papers is comprised of the following series:
**Series A** – Survey Instruments (Erhebungsinstrumente)
**Series B** – Survey Reports (Methodenberichte)
**Series C** – Data Documentation (Datendokumentationen)
**Series D** – Variable Descriptions and Coding
**Series E** – SOEPmonitors
**Series F** – SOEP Newsletters
**Series G** – General Issues and Teaching Materials

The SOEP Survey Papers are available at **http://www.diw.de/soepsurveypapers**



Please cite this paper as follows:

Klaudia Erhardt. 2018. SOEP-Metafile.do – ein Stata-Do-File zur Generierung eines Metafiles zu den SOEP-Daten. SOEP Survey Papers 534: Series G. Berlin: DIW/SOEP


ISSN: 2193-5580 (online)
DIW Berlin
German Socio-Economic Panel (SOEP)
Mohrenstr. 58
10117 Berlin
Germany
soeppapers@diw.de

# SOEP-Metafile.do – ein Stata-Do-File zur Generierung eines Metafiles zu den SOEP-Daten

## Anwendung des Do-Files und Dokumentation des resultierenden Metafiles

Klaudia Erhardt

SOEP-Metafile.do

| | |
|---|---|
| Download: | https://doi.org/10.5281/zenodo.1256001 |
| SOEP-Metafile.do | Do-File zur Generierung eines Metafiles aus SOEP-Daten. |
| Varlabels_Metafile.do | Do-File zum Auswechseln der deutschen und englischen Metafile-Variablenlabels. |

Systemvoraussetzungen:

Der Do-File ist entwickelt mit Stata Version 13, angepasst an die Unicode-Umstellung von Stata Version 14 und ist auch unter Stata Version 15 lauffähig.

Es muss mindestens Stata IC zur Verfügung stehen.

Kontakt zur Autorin für Fragen und Feedback:

erhardtk@gmx.de

## Inhalt

## Vorbemerkung

Die Website des SOEP bietet im Augenblick noch nicht die Möglichkeit, Software mit einer DOI herunterzuladen. Deshalb wird SOEP-Metafile.do vorerst für externe NutzerInnen über das EU-geförderte Open-Science Repository zenodo angeboten.

## 1 Einführung und Überblick über den Inhalt dieses Papiers

In diesem Papier wird der Stata-Do-File SOEP-Metafile.do (Fassung für SOEP-NutzerInnen) vorgestellt. Er dient dazu, einen Überblicksfile (Metafile) zu den SOEP-Daten generieren. AnwenderInnen von SOEP-Metafile.do müssen nur wenige Einstellungen zur Anpassung an die lokale Umgebung vornehmen. Abgesehen davon funktioniert die Generierung eines Metafiles mit SOEP-Metafile.do vollkommen programmgesteuert.

Der mit SOEP-Metafile.do generierte Metafile - der seinerseits auch ein Stata-Datenfile ist - enthält Informationen zu allen Variablen derjenigen SOEP-Stata-Datenfiles, die in die Verarbeitung einbezogen werden. Einen Eindruck von ihn gibt die Abbildung auf der folgenden Seite, die einen Ausschnitt aus dem Metafile im Stata-Dateneditor zeigt.

Im nächsten Abschnitt dieses Papiers wird der Nutzen des mit SOEP-Metafile.do generierten Metafiles für SOEP-AnwenderInnen dargestellt.

Abschnitt 3 erläutert einige Grundbegriffe, die in diesem Papier verwendet werden.

In Abschnitt 4 werden die Variablen des Metafiles im Detail vorgestellt und beschrieben.

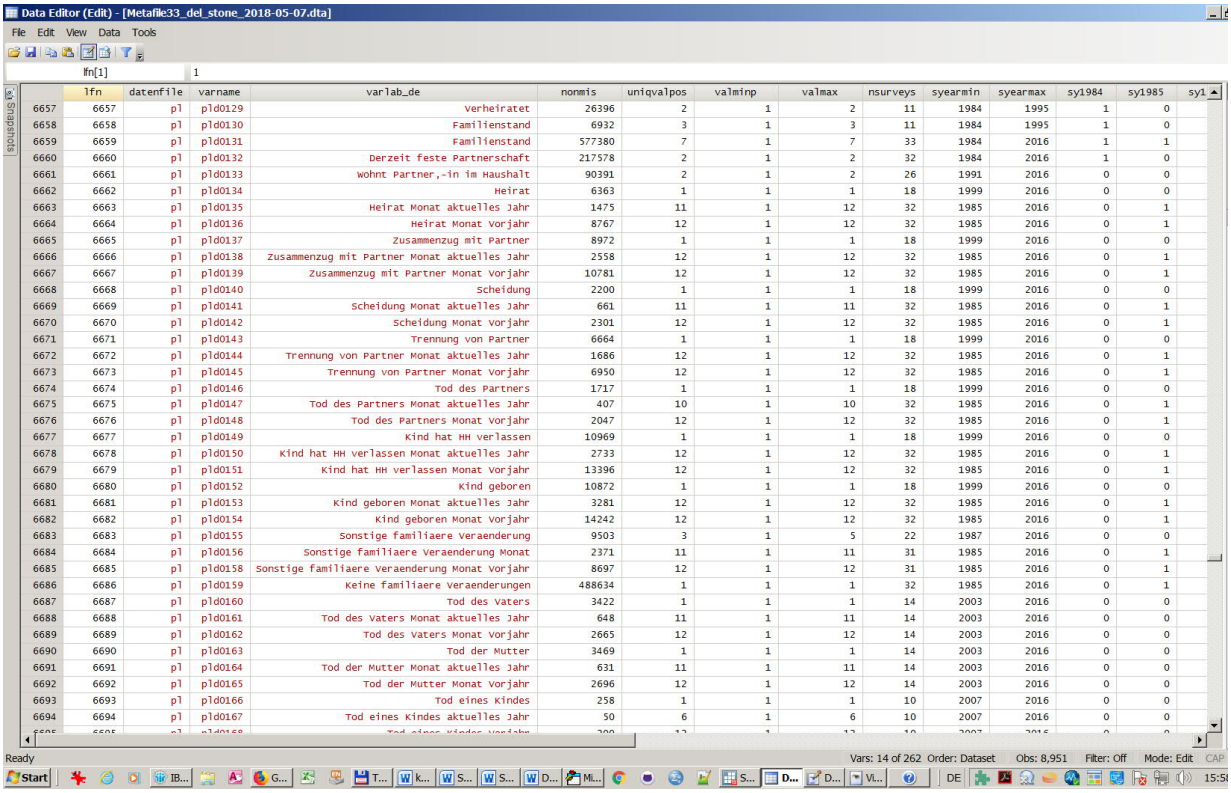
Abschnitt 5 enthält die konkrete Anleitung, was zu tun ist, um mit SOEP-Metafile.do einen Metafile zu den SOEP-Daten in der lokalen Umgebung selbst zu erstellen. Es werden die möglichen Parameter-Einstellungen des Do-Files dargestellt und erklärt, wie sich die verschiedenen Optionen auswirken.

Abschnitt 6 gibt Syntaxbeispiele für die Handhabung des Metafiles.

Für näher an der Syntax Interessierte enthält Abschnitt 7 das Ablaufschema von SOEP-Metafile.do.

Abschnitt 8 schließlich beinhaltet den Abdruck der gesamten Syntax von SOEP-Metafile.do.

Ansicht des Metafiles im Dateneditor




| | lfn | datenfile | varname | varlab_de | nonmis | uniqvalpos | valminp | valmax | nsurveys | syearmin | syearmax | sy1984 | sy1985 | sy1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6657 | 6657 | pl | pld0129 | Verheiratet | 26396 | 2 | 1 | 2 | 11 | 1984 | 1995 | 1 | 0 | |
| 6658 | 6658 | pl | pld0130 | Familienstand | 6932 | 3 | 1 | 3 | 11 | 1984 | 1995 | 1 | 0 | |
| 6659 | 6659 | pl | pld0131 | Familienstand | 577380 | 7 | 1 | 7 | 33 | 1984 | 2016 | 1 | 1 | |
| 6660 | 6660 | pl | pld0132 | Derzeit feste Partnerschaft | 217578 | 2 | 1 | 2 | 32 | 1984 | 2016 | 1 | 0 | |
| 6661 | 6661 | pl | pld0133 | Wohnt Partner,-in im Haushalt | 90391 | 2 | 1 | 2 | 26 | 1991 | 2016 | 0 | 0 | |
| 6662 | 6662 | pl | pld0134 | Heirat | 6363 | 1 | 1 | 1 | 18 | 1999 | 2016 | 0 | 0 | |
| 6663 | 6663 | pl | pld0135 | Heirat Monat aktuelles Jahr | 1475 | 11 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6664 | 6664 | pl | pld0136 | Heirat Monat Vorjahr | 8767 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6665 | 6665 | pl | pld0137 | Zusammenzug mit Partner | 8972 | 1 | 1 | 1 | 18 | 1999 | 2016 | 0 | 0 | |
| 6666 | 6666 | pl | pld0138 | Zusammenzug mit Partner Monat aktuelles Jahr | 2558 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6667 | 6667 | pl | pld0139 | Zusammenzug mit Partner Monat Vorjahr | 10781 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6668 | 6668 | pl | pld0140 | Scheidung | 2200 | 1 | 1 | 1 | 18 | 1999 | 2016 | 0 | 0 | |
| 6669 | 6669 | pl | pld0141 | Scheidung Monat aktuelles Jahr | 661 | 11 | 1 | 11 | 32 | 1985 | 2016 | 0 | 1 | |
| 6670 | 6670 | pl | pld0142 | Scheidung Monat Vorjahr | 2301 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6671 | 6671 | pl | pld0143 | Trennung von Partner | 6664 | 1 | 1 | 1 | 18 | 1999 | 2016 | 0 | 0 | |
| 6672 | 6672 | pl | pld0144 | Trennung von Partner Monat aktuelles Jahr | 1686 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6673 | 6673 | pl | pld0145 | Trennung von Partner Monat Vorjahr | 6950 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6674 | 6674 | pl | pld0146 | Tod des Partners | 1717 | 1 | 1 | 1 | 18 | 1999 | 2016 | 0 | 0 | |
| 6675 | 6675 | pl | pld0147 | Tod des Partners Monat aktuelles Jahr | 407 | 10 | 1 | 10 | 32 | 1985 | 2016 | 0 | 1 | |
| 6676 | 6676 | pl | pld0148 | Tod des Partners Monat Vorjahr | 2047 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6677 | 6677 | pl | pld0149 | Kind hat HH verlassen | 10969 | 1 | 1 | 1 | 18 | 1999 | 2016 | 0 | 0 | |
| 6678 | 6678 | pl | pld0150 | Kind hat HH verlassen Monat aktuelles Jahr | 2733 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6679 | 6679 | pl | pld0151 | Kind hat HH verlassen Monat Vorjahr | 13396 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6680 | 6680 | pl | pld0152 | Kind geboren | 10872 | 1 | 1 | 1 | 18 | 1999 | 2016 | 0 | 0 | |
| 6681 | 6681 | pl | pld0153 | Kind geboren Monat aktuelles Jahr | 3281 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6682 | 6682 | pl | pld0154 | Kind geboren Monat Vorjahr | 14242 | 12 | 1 | 12 | 32 | 1985 | 2016 | 0 | 1 | |
| 6683 | 6683 | pl | pld0155 | Sonstige familiaere Veraenderung | 9503 | 3 | 1 | 5 | 22 | 1987 | 2016 | 0 | 0 | |
| 6684 | 6684 | pl | pld0156 | Sonstige familiaere Veraenderung Monat | 2371 | 11 | 1 | 11 | 31 | 1985 | 2016 | 0 | 1 | |
| 6685 | 6685 | pl | pld0158 | Sonstige familiaere Veraenderung Monat Vorjahr | 8697 | 12 | 1 | 12 | 31 | 1985 | 2016 | 0 | 1 | |
| 6686 | 6686 | pl | pld0159 | Keine familiaere Veraenderungen | 488634 | 1 | 1 | 1 | 32 | 1985 | 2016 | 0 | 1 | |
| 6687 | 6687 | pl | pld0160 | Tod des Vaters | 3422 | 1 | 1 | 1 | 14 | 2003 | 2016 | 0 | 0 | |
| 6688 | 6688 | pl | pld0161 | Tod des Vaters Monat aktuelles Jahr | 648 | 11 | 1 | 11 | 14 | 2003 | 2016 | 0 | 0 | |
| 6689 | 6689 | pl | pld0162 | Tod des Vaters Monat Vorjahr | 2665 | 12 | 1 | 12 | 14 | 2003 | 2016 | 0 | 0 | |
| 6690 | 6690 | pl | pld0163 | Tod der Mutter | 3469 | 1 | 1 | 1 | 14 | 2003 | 2016 | 0 | 0 | |
| 6691 | 6691 | pl | pld0164 | Tod der Mutter Monat aktuelles Jahr | 631 | 11 | 1 | 11 | 14 | 2003 | 2016 | 0 | 0 | |
| 6692 | 6692 | pl | pld0165 | Tod der Mutter Monat Vorjahr | 2696 | 12 | 1 | 12 | 14 | 2003 | 2016 | 0 | 0 | |
| 6693 | 6693 | pl | pld0166 | Tod eines Kindes | 258 | 1 | 1 | 1 | 10 | 2007 | 2016 | 0 | 0 | |
| 6694 | 6694 | pl | pld0167 | Tod eines Kindes aktuelles Jahr | 50 | 6 | 1 | 6 | 10 | 2007 | 2016 | 0 | 0 | |

## 2 Zum Nutzen des mit SOEP-Metafile.do generierten Metafiles

Das SOEP ist eine hochkomplexe Panelstudie, die seit 1983 jährlich durchgeführt wird, und in die über die Jahre hinweg zahlreiche Stichproben integriert wurden. In der 33. Welle (Datenrelease 2017) lag die Zahl der Datenfiles, die an die NutzerInnen weitergegeben werden, bei 440 wellenspezifischen Files mit 72.802 Variablen und 18 Files der SOEPlong-Version mit 8.951 Variablen, wovon der größte File, pl, alleine 3.170 Variablen umfasst.

Der mit SOEP-Metafile.do generierte Metafile ist ein wertvolles Hilfsmittel, um sich in diesem komplexen Datenbestand zu orientieren. Er stellt eine Gesamttabelle aller Variablen der in die Verarbeitung einbezogenen SOEP-Stata-Datenfiles dar. Jede Metafile-Variable (Spalte der Gesamttabelle) bildet ein bestimmtes Merkmal der SOEP-Datenfiles bzw. ihrer Variablen ab. Jede Datenzeile des Metafiles steht für eine Variable eines SOEP-Datenfiles.

Die Metafile-Variablen zeigen z.B., wieviele verwertbare Observationen eine SOEP-Variable hat, wie oft sie erhoben wurde, für welche Surveyjahre sie zur Verfügung steht und mehr (siehe Abschnitt 4, Dokumentation des Metafiles). So kann man beispielsweise anhand des Metafiles in der SOEPlong-Version diejenigen Variablen identifizieren, die in einem bestimmten Zeitraum mindestens für x Wellen mit einer Mindestzahl von y verwertbaren Observationen zur Verfügung stehen. Die entsprechende Variablenliste kann als Ausgangspunkt

für Recherchen im Informationssystem des SOEP, Paneldata, dienen, um weitere Informationen zu den ausgewählten Variablen zu erhalten, oder sie kann in eigenen Do-Files weiter verwendet werden (siehe die Syntaxbeispiele in Abschnitt 6).

Da der Metafile ebenfalls ein Stata-Datenfile ist, stehen alle in Stata vorhandenen Möglichkeiten zur Auswahl und Sortierung von Daten und zur Weiterverwendung ausgewählter Informationen aus dem Metafile zur Verfügung, sei es in der eigenen Stata-Syntax oder in Dritt-Programmen, die eine Schnittstelle zu den Exportformaten von Stata haben.

SOEP-Metafile.do ist auf die Verarbeitung von SOEP-Daten abgestimmt, aber es muss sich dabei nicht um die Originaldaten handeln. Der Do-File kann ebenso dazu benutzt werden, einen Metafile zu den eigenen, modifizierten SOEP-Daten zu generieren.

Nicht zuletzt ist der Metafile ein Hilfsmittel für diejenigen NutzerInnen der SOEP-Daten, denen nur die Stata-Version IC zur Verfügung steht, und die deshalb nicht alle SOEP-Datenfiles unmittelbar öffnen können. Der Metafile dient in diesem Fall als Informationsquelle zur Selektion von Variablen für den use <varlist>, using <SOEP-Datenfile>- Befehl, mit dem Untermengen der Variablen eines Datenfiles in den Arbeitsspeicher geladen werden.

Mit Hilfe der (englischsprachigen) Kommentare in der Syntax in Abschnitt 8 können näher Interessierte Details der Programmierung nachvollziehen, um die dort entwickelten Lösungen (beispielsweise zur Messung der Laufzeit des Programms oder zu einer segmentierten Verarbeitung großer Datenfiles) als Anregung für eigene Programme zu benutzen.

## 3 Erläuterungen zu einigen Grundbegriffen

Einige der in diesem Papier verwendeten Begriffe stellen Eigenschöpfungen dar, andere stehen in Bezug auf SOEP-spezifische Rahmenbedingungen. Deshalb werden zentrale Begriffe hier kurz erläutert.

### Nonmissing Werte

Da im SOEP einstellige negative Integers als Standard-Missing-Codes verwendet werden (d.h. Ziffern zwischen -1 und -9), sind die nonmissing-Werte in der vorliegenden Syntax als Werte >= 0 & < sysmis definiert. Es gibt einige wenige Variablen im SOEP, hauptsächlich generierte Einkommens- und Vermögensvariablen, die negative Werte haben können, welche keinen missing Code darstellen. Diese Variablen werden in der Syntax nicht gesondert behandelt, da es keinen kurzen und bündigen Algorithmus zu ihrer Kennzeichnung gibt. Die Zahl der nonmissing-Observationen werden daher für diese Variablen im Metafile nicht korrekt ausgewiesen. In der Regel sind diese Variablen anhand der Metafile-Variablen valmin (kleinster Wert) identifizierbar (sofern der kleinste Wert kleiner als - 9 und nicht -200 ist):

```
edit datenfile varname varlab valmin if valmin < -9 & valmin != -200
```

Anmerkung: -200 ist ein Metafile-spezifischer Missing Code, s. Abschnitt 4.2.

System missings oder sysmis

Bezeichnung für die Stata missing codes ., .a, .b, (...) .z, zur Unterscheidung von den SOEP-eigenen missing codes. Der standard Stata missing code "." wird im Fließtext leicht übersehen, deshalb wird in diesem Papier häufig stattdessen die Bezeichnung <sysmis> oder sysmis gewählt.

Indikatoren und Typ 2-Indikatoren

Als Indikatoren werden in diesem Papier Kennwerte bezeichnet, die aus den Quell-Datenfiles und ihren Variablen extrahiert und in die Variablen des Metafiles übertragen werden.

Typ 1 Indikatoren haben einen einzigen Wert pro Quell-Variable. Der Kürze halber werden sie in diesem Papier meist nur "Indikatoren" genannt. Typ 2-Indikatoren sind Kennwerte, die pro Quell-Variable eine Liste von Werten haben. Zum Beispiel ist der Maximalwert der Quell-Variablen ein (Typ 1-)Indikator, während die Ausprägungen einer Quell-Variablen einen Typ 2-Indikator darstellen.

Im Metafile wird aus einem Typ 1-Indikator eine einzelne Variable gespeist, aus einem Typ 2-Indikator dagegen eine ganze Variablengruppe.

Makros und Listenmakros

Makros dienen in Stata-Programmen als eine Art Textbausteine. Ihnen können im Programmablauf Zeichenfolgen oder Werte zugewiesen werden, die dann an verschiedenen Stellen des Programms eine Funktion erfüllen, etwa als Schleifenzähler oder um Werte an Variablen zu übertragen.

Stata kennt lokale Makros, die nur innerhalb eines Programmablaufs existieren, und globale Makros, die während der Dauer einer Stata-Sitzung bestehen. In der gesamten Syntax von SOEP-Metafile.do werden ausschließlich lokale Makros verwendet.

Ein in diesem Papier so genanntes "Listenmakro" sammelt einen bestimmten Kennwert für alle Variablen eines Quelldatenfiles, wenn es sich um einen Typ 1-Indikator handelt. Bei Typ 2-Indikatoren wird ein Listenmakro für jede Quellvariable, und somit eine Serie von Listenmakros für alle Variablen eines Files befüllt.

Wellenindikator

Im SOEP wurden über die Jahre hinweg in den verschiedenen Datenfiles unterschiedliche Variablen in unterschiedlichen Formaten zur Kennzeichnung des Erhebungsjahrs benutzt. Die den derzeitigen Standard im SOEP darstellende Variable im vierstelligen Jahresformat syear

<yyyy> ist noch nicht in allen älteren Datenfiles vorhanden. Zudem enthalten einige Datenfiles mehrere verschiedene Survey-Jahr-Variablen.

Deshalb wird die jeweils maßgebliche Surveyjahr-Variable für jeden zu verarbeitenden Datenfile anhand einer Prioritätenliste ermittelt, welche die verschiedenen, im SOEP verwendeten Wellen- und Befragungsjahr-Variablen enthält. Der Wellenindikator (Metafile-Variable wind) bezeichnet die Variable im Quelldatensatz, welche letztlich zur Generierung der Surveyjahr-Indikatoren im Metafile herangezogen wurde.

# 4 Dokumentation des Metafiles

Ein mit SOEP-Metafile.do über die SOEP-Daten der Version V33 (Datenlieferung 2017) generierter Metafile enthält 261 Variablen: 33 Flag-Variablen für die Surveyjahre, in denen die Quell-Variablen mit nonmissing Werten vertreten sind, je 100 Variablen für Ausprägungen und Value-Labels von Quell-Variablen, 18 Variablen mit weiteren Indikatoren zu den Quell-Variablen, 4 Variablen mit Indikatoren zum Quell-Datenfile und 6 Variablen mit "technischen" Angaben.

## 4.1 Übersicht über die Metafile-Variablen

- Filebezogene Indikatoren
  - Zahl der Observationen im File
  - Zahl der Variablen im File
  - Zahl der Variablen im File mit nonmissing Werten (SOEP-spezifisch: mit Werten >= 0 & < .)
  - Wellenindikator: Variable im Quell-File, die das Erhebungsjahr anzeigt
- Variablenbezogene Indikatoren (Typ 1-Indikatoren)
  - Zahl der nonmissing Observationen
  - Zahl der System-missings
  - Zahl der Ausprägungen
  - Zahl der nonmissing Ausprägungen
  - Wert der Variablen, falls sie nur 1 Ausprägung hat
  - Wert der Variablen, falls sie nur 1 nonmissing Ausprägung hat
  - Häufigkeit des Werts 0
  - Kleinster Wert
  - Kleinster nonmissing Wert
  - Höchster Wert (ohne System missings)
  - Name der Value Label Definition

  - Zahl der ungelabelten Observationen (nur bei gelabelten Variablen)
  - Höchster gelabelter Wert
  - Minimale Länge (Stringvariable)
  - Maximale Länge (Stringvariable)
  - Zahl der Erhebungsjahre
  - Frühestes Erhebungsjahr
  - Spätestes Erhebungsjahr

- Variablenbezogene Typ 2-Indikatoren

  - Erhebungsjahre
  - Ausprägungen (für Variablen mit max. 100 Ausprägungen)
  - Value Labels (für Variablen mit max. 100 Ausprägungen)

- "Technische" Angaben

  - Laufende Nummer
  - Name des Quelldatenfiles
  - Name der Quellvariablen
  - Typ der Quellvariablen
  - Speicherformat der Quellvariablen
  - Variablenlabel der Quellvariablen

## 4.2 Detaillierte Dokumentation der Metafile-Variablen

| Variable | Variablenlabel | Beschreibung |
|---|---|---|
| lfn | Laufende Nummer | numVar, Wertebereich: Integers > 0<br>laufende Nummer über alle Observationen. Wird am Ende des Programmlaufs gebildet, kann daher nicht als Identifikator über verschiedene Versionen des Metafiles benutzt werden.<br>(Ein konstanter eindeutiger Identifikator für die Observationen der Metafiles ist die Kombination datenfile+varname) |
| datenfile | Quell-Datenfile | stringVar<br>Name des Quell-Datenfiles |
| varname | Variablenname | stringVar<br>Name der Quell-Variablen |
| type | storage type | stringVar<br>Datentyp der Quellvariablen (generiert durch den Stata-Befehl describe, replace clear) |
| isnumeric | whether numeric or string | numVar, Wertebereich: 1, 0<br>Flag: Quellvariable ist numerisch/nicht numerisch |
| varlab | variable label | stringVar<br>Variablenlabel der Quellvariablen<br>HINWEIS: der Name dieser Metafilevariablen kann im Definitionabschnitt von SOEP-Metafile.do verändert werden. |
| nvrs | Zahl d. Variablen im File | numVar, Wertebereich: Integers > 0 |
| nvrsnm | Zahl d. Variablen im File mit nonmiss. Werten | numVar, Wertebereich: Integers >= 0 |
| nobs | Zahl d. Obs. im File | numVar, Wertebereich: Integers > 0<br>Zahl der Observationen des Quelldatenfiles |
| nonmis | Zahl d. nonmiss. Obs. (variablenbezog.) | numVar, Wertebereich: Integers >= 0<br>Zahl der Observationen in der Quellvariablen mit Werten > = 0 & < . (numvars) oder nicht leer und kein missing code (stringvars)<br>ACHTUNG: nicht valide fuer Variablen, die echte negative Werte haben können. Das sind v.a. generierte Einkommens- und Vermögensvariablen.<br>Für Stringvariablen werden leere oder 2 Zeichen lange Einträge, die ein Minuszeichen enthalten, als missing gewertet. |
| sysmis | Zahl d. system-missing Obs. (variablenbezog.) | numVar, Wertebereich: Integers >= 0<br>Zahl der Observationen, die ein System missing <dot> haben<br>Für Stringvariablen stets 0 |

| Variable | Variablenlabel | Beschreibung |
|---|---|---|
| uniqvals | Zahl d. Auspraegungen | numVar, Wertebereich: Integers > 0<br>Zahl der Ausprägungen (einschließlich Missing Codes und sysmis <dot>)<br>Für Stringvariablen: Zahl der unterschiedlichen Einträge (einschließlich "") |
| uniqvalpos | Zahl d. Auspraeg. >= 0 | numVar, Wertebereich: Integers > 0<br>Zahl der Ausprägungen >=0 & < . (also ohne sysmis und missing codes)<br>Für Stringvariablen: Zahl der unterschiedlichen Einträge, die weder leer sind noch aus einem 2 Zeichen langen Eintrag bestehen, der ein Minuszeichen enthält (= SOEP-missing code) |
| valuniq | Wert falls nur 1 Auspraeg. | numVar, Wertebereich: -900, -200, Werte der Quellvariablen<br>Wert der Quellvariablen if uniqvals == 1<br>-900 if uniqvals > 1<br>-200 für Stringvariablen |
| valuniqpos | Wert falls nur 1 pos. Auspraeg. | numVar, Wertebereich: -900, -200, Werte der Quellvariablen<br>Wert der Quellvariablen if uniqvalpos == 1<br>-900 if uniqvalpos != 1<br>-200 für Stringvariablen |
| nvalzero | Haeufigkeit des Werts 0 | numVar, Wertebereich: Integers >= 0<br>Für Stringvariablen enthält nvalzero die Häufigkeit leerer Strings |
| valmin | Minimalwert der Variablen | numVar, Wertebereich: -200, Werte der Quellvariablen<br>Dezimalwerte auf 2 Nachkommastellen gerundet<br>-200 für Stringvariablen |
| valminp | kleinster pos. Wert der Variablen (incl. 0) | numVar, Wertebereich: -900, -200, Werte der Quellvariablen >= 0<br>Kleinster Wert der Quellvariablen der >=0 ist (= kleinster nonmissing Wert)<br>-900 valmax < 0<br>-200 für Stringvariablen |
| valmax | Maximalwert der Variablen | numVar, Wertebereich: -200, Werte der Quellvariablen<br>Dezimalwerte auf 2 Nachkommastellen gerundet<br>Ohne sysmis <dot>, außer wenn sysmis der einzige Wert ist (dann ist auch valmin sysmis)<br>-200 für Stringvariablen |
| vallabset | Name d. Valuelabel-Sets | stringVar<br>Name des zugeordneten Valuelabel-Sets<br>"-2" wenn kein Valuelabel-Set zugeordnet wurde |

| Variable | Variablenlabel | Beschreibung |
|---|---|---|
| undoc | Anzahl ungelabelter Observ. (gelabelte Vars) | numVar, Wertebereich: -9, -2, Integers >= 0<br>Zahl der Observationen der Quellvariablen, die in eine nicht gelabelte Ausprägung fallen, obwohl die Variable Valuelabels hat<br>-9 für Variablen, denen ein Labelset zugeordnet ist, in dem aber nur die Missingcodes u. evtl. die 0 gelabelt sind<br>-2 für Variablen, denen kein Labelset zugeordnet ist |
| vlabmax | hoechster gelabelter Wert | numVar, Wertebereich: -200, Werte der Quellvariablen<br>-200 wenn kein Valuelabel-Set zugeordnet wurde<br>ACHTUNG: der Indikator bezieht sich auf die definierten Labels, nicht auf die tatsächlich vorhandenen Ausprägungen. |
| strmin | Min. Laenge (Stringvars) | numVar, Wertebereich: -2, Integers >= 0<br>Minimale Länge von Einträgen, die keine leeren Strings sind (= 0, wenn alle Einträge leer sind)<br>-2 für numVars |
| strmax | Max. Laenge (Stringvars) | numVar, Wertebereich: -2, Integers >= 0<br>Maximale Länge von Einträgen, die keine leeren Strings sind (= 0, wenn alle Einträge leer sind)<br>-2 für numVars |
| wind | Wellenindikator | stringVar<br>enthält den Wellenindikator, der den Metafile-Variablen nsurveys, syearmin, syearmax und sy1984 - sy20## zugrundeliegt<br>Mögliche Wellenindikatoren der Quelldatenfiles werden in folgender Rangfolge herangezogen: syear, welle, wave, intyear, erhebj, bioyear<br>-1 wenn der Quelldatenfile keinen der Wellenindikatoren der Prioritätenliste enthält |
| nsurveys | wie oft erhoben? | numVar, Wertebereich: -1, Integers >= 0<br>Anzahl der Wellen, in denen die jeweilige Variable erhoben wurde. Generiert anhand der Variablen: syear, welle, wave, intyear, erhebj, bioyear (in dieser Rangfolge)<br>0 wenn im Quelldatenfile zwar ein Wellenindikator vorhanden ist. die jeweilige Variable aber keine nonmissing Werte enhält<br>-1 wenn der Quelldatenfile keinen der Wellenindikatoren der Prioritätenliste enthält |

| Variable | Variablenlabel | Beschreibung |
|---|---|---|
| syearmin | fruehestes Surveyjahr | numVar, Wertebereich: -9, -1, Integers >= 0 & <= Surveyjahr d. aktuellen Datenlieferung<br>Frühestes Surveyjahr, in dem die jeweilige Variable erhoben wurde bzw. kleinster Wert der herangezogenen Wellenvariablen<br>Die Variable zeigt den Inhalt des zugrunde liegenden Wellenindikators ohne Transformation in das vierstellige Jahresformat yyyy<br>-1 wenn der Quelldatenfile keinen der Wellenindikatoren der Prioritätenliste enthält<br>-9 wenn nsurveys 0 ist (d.h. wenn im Quelldatenfile zwar ein Wellenindikator vorhanden ist. die Quellvariable aber keine nonmissing Werte enhält |
| syearmax | spaetestes Surveyjahr | numVar, Wertebereich: -9, -1, Integers >= 0 & <= Surveyjahr d. aktuellen Datenlieferung<br>Spätestes Surveyjahr, in dem die jeweilige Variable erhoben wurde bzw. größter Wert der herangezogenen Wellenvariablen<br>Die Variable zeigt den Inhalt des zugrunde liegenden Wellenindikators ohne Transformation in das vierstellige Jahresformat yyyy<br>-1 wenn der Quelldatenfile keinen der Wellenindikatoren der Prioritätenliste enthält<br>-9 wenn nsurveys 0 ist (d.h. wenn im Quelldatenfile zwar ein Wellenindikator vorhanden ist, die jeweilige Variable aber keine nonmissing Werte enhält |
| sy1984 - sy20## | -- | numVars, Wertebereich: -1, 0, 1<br>Flagvariablen für ein Vorkommen mit non-missing Werten in den betreffenden Surveyjahren<br>1 wenn die Variable im jeweiligen Erhebungsjahr nonmissing values hat<br>0 wenn die Variable im Erhebungsjahr nicht erhoben wurde oder keine nonmissing values hat<br>-1 wenn der Quelldatenfile keinen der Wellenindikatoren der Prioritätenliste enthält |

| Variable | Variablenlabel | Beschreibung |
|---|---|---|
| val1 - val100 | -- | numVars, Wertebereich: -200, -100, -900, sysmis, Werte der Variablen<br>Die Variablen enthalten die Werte der tatsächlich vorhandenen Ausprägungen (nur für Vars mit maximal 100 nonmissing-Ausprägungen)<br>-100 für numVars, die mehr als 100 nonmissing Ausprägungen haben<br>-900 für numVars, die zwar weniger als 100 Ausprägungen haben, aber alle mit codes < 0 (SOEP-missing Codes)<br>-200 für Stringvars (Die Werte der Stringvars werden in die Variablen lab1-lab100 geschrieben, da val1-val100 numerische Variablen sind)<br>sysmis <dot> in "nicht belegten" val#-Variablen |
| lab1 - lab100 | -- | strVars<br>Die Variablen enthalten die Labels der tatsächlich vorhandenen Ausprägungen (nur für Vars mit maximal 100 nonmissing-Ausprägungen)<br>"###" Ausprägungen ohne value label bei ansonsten mit Value-Labels versehenen Variablen (= undokumentierte Ausprägungen, oder Skalen wo nur die Endpunkte gelabelt sind)<br>"-100" Variablen mit mehr als 100 gelabelten nonmissing-Ausprägungen<br>"-200" Variablen, denen kein Value-Label-Set zugewiesen wurde<br>"-900" Variablen mit Value-Label-Set und maximal 100 Ausprägungen, aber alle mit codes < 0 (SOEP-missing Codes)<br>Für Stringvariablen enthalten die Variablen lab# die unique Werte (Ausprägungen) der Quellvariablen. Bei mehr als 100 Ausprägungen enthalten die Variablen lab# den Wert "-100" |

# 5 Erzeugung eines Metafiles zu den SOEP-Daten mit SOEP-Metafile.do

Um selbst einen Metafile zu den SOEP-Daten zu generieren, müssen AnwenderInnen den Do-File SOEP-Metafile.do nur an ihre lokalen Gegebenheiten anpassen. Was dafür konkret zu tun ist, wird in Abschnitt 5.2 dieses Papiers im Detail erläutert.

Zuvor soll ein kurzer Einblick in das Funktionsprinzip von SOEP-Metafile.do gegeben werden:

SOEP-Metafile.do zieht alle (oder ausgewählte) Stata-Datenfiles heran, die in einem festzulegenden Quelldaten-Verzeichnis stehen. Die Datenfiles werden nacheinander verarbeitet. Dabei werden bestimmte Indikatoren zu den Variablen des jeweiligen Datenfiles extrahiert, in quelldatenfile-spezifische Metafiles zwischengespeichert und am Ende in einem neuen File, dem Gesamt-Metafile, zusammengefügt.

Um die Laufzeit des Programms zu verringern, werden dabei die Files ab einer bestimmten Größe zerlegt und portionsweise verarbeitet (segmentierte Verarbeitung).

SOEP-Metafile.do schreibt sukzessive Fortschrittsmeldungen, eventuelle Datenprobleme, welche die Verarbeitung eines Datenfiles verhindern, Name und Pfad des resultierenden Metafiles sowie die Gesamtlaufzeit in den Output und die anzugebende Log-Datei.

## 5.1 Hinweise und Empfehlungen zur Anwendung von SOEP-Metafile.do

- Die Laufzeit von SOEP-Metafile.do betrug für SOEP v33 in der SOEP-EDV-Umgebung eine knappe Stunde für die wellenspezifischen Files und ca. 3 Stunden für SOEPlong, jeweils bei segmentierter Verarbeitung und einer Portionsgröße von 50 Variablen.

- SOEP-Metafile.do sollte zuerst in einem Testlauf ausprobiert werden, um zu sehen, ob alles funktioniert wie gewünscht. Einen Testlauf kann man durchführen, indem man entweder in dem Makro files ein Kriterium angibt, das nur wenige Datenfiles erfüllen, oder indem man ein Testverzeichnis einrichtet, das nur bestimmte Datenfiles enthält. Es empfiehlt sich zumindest bei Testläufen über SOEPlong-Datenfiles, die Zahl der Beobachtungen in den Testfiles drastisch zu verringern, denn diese beeinflusst die Laufzeit ganz erheblich.

- Es ist besser, getrennte Metafiles für SOEP und SOEPlong zu erzeugen. Denn wenn beide in einem Durchgang generiert werden und somit der Metafile sowohl die SOEP wie die SOEPlong-Variablen enthält, steht kein einfaches Kriterium zur Verfügung, um die Metafiles der Long-Datenfiles von denen der wellenspezifischen Datenfiles zu unterscheiden und ggfls wieder voneinander zu trennen. Wenn die Metadaten für beide SOEP-Versionen in einem einzigen Metafile geführt werden sollen, ist es besser, die getrennt generierten Metafiles mit dem Stata-Befehl append, generate(newvar) zu vereinen. Dadurch wird eine Flag-Variable für die Herkunft der Metadaten aus SOEP oder SOEPlong erzeugt.

## 5.2 Erforderliche Anpassungen im Parameter-Definitionsteil

Die Syntax von SOEP-Metafile.do ist in verschiedene Bereiche gegliedert. Anpassungen durch die AnwenderInnen werden ausschließlich im Teil "B) DEFINITION OF PARAMETERS" vorgenommen, der auf "A) AUTOMATIC DEFINITION OF PARAMETERS" folgt. In die Syntax eingefügte Kommentare und Beispiele helfen bei der manuellen Parameterfestlegung.

Weitere Änderungen sind nicht erforderlich und sollten nur dann gemacht werden, wenn man sicher ist, die Syntax von Grund auf verstanden zu haben.

Im Teil "B) DEFINITION OF PARAMETERS" werden die in der folgenden Tabelle aufgelisteten Makros definiert. Die Angabe "obligatorisch" bedeutet, dass das Makro nicht leer sein darf, sondern einen Wert enthalten muss (d.h. die Definition darf nicht lauten: local xyz "" ). Optionale Makros dürfen leer definiert werden. Die jeweiligen Konsequenzen sind in der Tabelle beschrieben. Die Angabe "optional bzw. obligatorisch falls SYI erstellt werden sollen" bedeutet, dass die Angabe insofern optional ist, als auf die Erstellung von Survey-Jahr-Indikatoren (SYI) verzichtet werden kann. Für die Survey-Jahr-Indikatoren ist die Angabe jedoch obligatorisch.

Viele der aufgelisteten Makros werden entweder im Programm-Ablauf durch einen Default-Wert belegt, wenn sie nicht im Definitionsteil belegt wurden, oder im Definitionsteil ist eine Belegung des Makros vorgegeben, die geändert werden kann, aber nicht geändert werden muss. So schrumpft die Zahl der Parameter, die tatsächlich von den AnwenderInnen eingegeben werden müssen, ganz erheblich.

In der Tabelle sind diese Parameter gelb unterlegt. Daneben gibt es Parameter, die von den AnwenderInnen kontrolliert und gegebenenfalls angepasst werden müssen. Diese sind hellblau unterlegt.

| Makroname | Erläuterung |
|---|---|
| we | Appendix zum Dateinamen des resultierenden Metafiles, sollte zur Kennzeichnung der für den Metafile verarbeiteten SOEP Version genutzt werden.<br>Optional. Wenn das Makro leer bleibt, erkennt man allerdings nicht mehr am Dateinamen des Metafiles, welche Version der SOEP-Daten enthalten sind. Außerdem besteht die Gefahr, dass der Metafile des letzten Durchlaufs ungewollt überschrieben wird, falls der neue Metafile am gleichen Tag in das gleiche Zielverzeichnis geschrieben wird (z.B. die Metafiles für das wellenspezifische SOEP und SOEP long).<br>In der Syntax vorbelegt mit "_SOEPv33_long".<br>Empfehlung: Kontrollieren und ggfls. anpassen. |
| data | Pfad zum Verzeichnis, in dem sich die SOEP-Daten befinden, die verarbeitet werden sollen.<br>Obligatorisch. Hinweis: Die Quell-Datenfiles werden nur in den Arbeitsspeicher gelesen und nicht verändert, es müssen also keine Kopien erzeugt werden, um sich gegen Datenverlust abzusichern. |
| pfad1<br>pfad1d<br>pfad1o | pfad1d: Verzeichnis, in dem der resultierende Metafile gespeichert wird.<br>pfad1o: Verzeichnis, in dem der Logfile gespeichert wird.<br>Beide sind Unterverzeichnisse zum Ziel-Hauptverzeichnis pfad1.<br>Obligatorisch. Die Pfadangaben sind so eingerichtet, dass die verbreitete Struktur getrennter Unterverzeichnisse für Daten, Output (und Do-Files, das hier aber nicht gebraucht wird) unterhalb eines Hauptverzeichnisses abgebildet werden kann. In allen 3 Makros müssen Pfadangaben gemacht werden, es darf aber jeweils das selbe Verzeichnis benannt werden (siehe Beispiele in der Syntax). |
| files | Auswahlkriterium für die zu verarbeitenden Files des Datenverzeichnisses. Das Kriterium darf ein einziges Item enthalten (eine mit oder ohne Platzhalter formulierte Bezeichnung, wie z.B. "bdp.dta" oder "b*.dta" oder "*p.dta").<br>Optional. Wenn das Makro leer bleibt, wird der Default-Wert "*.dta" gesetzt. |
| port | Zahl der Variablen pro Portion bei segmentierter Verarbeitung.<br>Optional. Wenn das Makro leer bleibt oder den Wert 0 hat, wird der Default-Wert 50 gesetzt. Dieser Wert hat sich als guter Kompromiss zwischen der Laufzeitverkürzung durch eine segmentierte Verarbeitung und der Laufzeitverlängerung durch die erhöhte Zahl der Durchläufe über die einzelnen Portionen herausgestellt.<br>Hinweis: Wenn der Wert von port größer ist als die halbe Anzahl von Variablen in einem File, wird dieser File ohne Portionierung verarbeitet. |
| varlab | Name der Variablen im Metafile, welche die Variablenlabels der Quellvariablen enthält.<br>Optional. Wenn das Makro leer bleibt, wird der Default-Wert "varlab" gesetzt.<br>Der Zweck dieses Makros ist es, englische und deutsche Variablenlabels im selben Metafile vorzuhalten. Wenn die Laufzeit keine Rolle spielt, kann man Metafiles für die englischen und deutschen SOEP-Versionen erstellen und dabei jeweils das Makro mit varlab_de bzw. varlab_en belegen. Anschließend ließe sich das Label der anderen Sprache über die eindeutigen Schlüsselvariablen datenfile und varname an den Master-Metafile anmergen. |
| result | Name des resultierenden Metafiles. Ist definiert als: metafile`we'_`datum'.<br>Obligatorisch. Änderungen der Makrodefinition sind erlaubt, aber nicht erforderlich.<br>Empfehlung: Die Vorbelegung des Makros nicht ändern. |

| Makroname | Erläuterung |
|---|---|
| reslab | Label, der dem resultierenden Metafile mit dem Stata-Befehl label data mitgegeben wird.<br>Optional. Wenn das Makro leer bleibt, wird dem Metafile kein label mitgegeben. |
| lg | Name des resultierenden Logfiles. Ist definiert als: metafile`we'_`datum'.log.<br>Obligatorisch. Änderungen der Makrodefinition sind erlaubt, aber nicht erforderlich.<br>Empfehlung: Die Vorbelegung des Makros nicht ändern. |
| syrlist | Prioritätenliste der möglichen Variablen in den verarbeiteten Datenfiles, die Wellenindikatoren sind. Ist definiert als: "syear welle wave intyear erhebj bioyear".<br>Optional bzw. obligatorisch falls SYI erstellt werden sollen. Änderungen der Liste sind erlaubt, aber nicht erforderlich. Änderungen der Reihenfolge der Wellenindikatoren in der Liste sind nicht sinnvoll.<br>Das Makro darf leer sein. Das hat zur Folge, dass der resultierende Metafile keine Informationen zum Vorkommen der Variablen in den Surveyjahren beinhaltet.<br>Empfehlung: Die Vorbelegung des Makros nicht ändern. |
| wave1, wave2 | Frühestes und spätestes Surveyjahr, definiert als wave1 = 1984, wave2 = 2016 (= aktuelle Welle für V33).<br>Optional bzw. obligatorisch falls SYI erstellt werden sollen. Die Makros dürfen leer bleiben, wenn keine Informationen zu den Surveyjahren gewünscht sind. Wenn solche Informationen im Metafile enthalten sein sollen, ist die Anpassung von wave2 an die aktuelle Welle erforderlich, wave1 bleibt unverändert.<br>Hinweis: wave1 und wave2 werden in der Syntax auch zur Abgrenzung von gültigen zu ungültigen Surveyjahren benutzt, deshalb sollten sie mit den tatsächlichen Wellen des SOEP übereinstimmen, sofern sie nicht leer sind.<br>Empfehlung: Die Vorbelegung von wave1 nicht ändern, wave2 kontrollieren und ggfls anpassen. |

# 6 Syntaxbeispiele für den Umgang mit dem Metafile

Hinweis: Im folgenden werden u.a. Stringfunktionen für die Datenselektion besprochen. Ab Stata 14 haben Stringfunktionen in der Regel ein vorangestelltes "u" (für Unicode). Z.B. wird die Funktion strpos() in Stata 14 zu ustrpos(). Für Stata 14 und höher müssen die Beispiele deshalb entsprechend angepasst werden.

Um Syntax zu schreiben, die sowohl für Stata 13 als auch für höhere Stata-Versionen funktioniert, kann man ein Makro definieren, welches den Funktionsnamen vorangestellt wird, und welches ein "u" enthält, wenn Stata ab Version 14 läuft und anderenfalls leer ist.

Die Umsetzung mit einem lokalen Makro zeigt die Syntax in Abschnitt 8.

Die folgenden Beispiele sind, soweit sie sich auf konkrete Daten beziehen, mit dem Metafile zu SOEPlong v33 erstellt worden.

## 6.1 Erstellung von "Sichten" auf den Metafile im Dateneditor von Stata

Im Dateneditor von Stata kann der Metafile wie eine Tabelle gesichtet werden. Im editmodus können Werte manuell geändert und in die Zwischenablage kopiert werden, während der browse-modus nur die Sichtung der Daten erlaubt. Auf jeden Fall ist es für die Betrachtung der Daten bequemer, wenn nur die Informationen angezeigt werden, die gerade gebraucht werden.

### 6.1.1 Auswahl von Metafile-Variablen

Wenn man bestimmte Variablen des Metafiles betrachten will, ist es oft günstiger, vor dem edit- oder browse-Befehl das Anzeigeformat zu verringern, damit mehr Spalten auf den Bildschirm passen:

```
format lfn %6.0g
format datenfile varname %8s
format varlab %46s
format nonmis uniqvalpos valminp valmax nsurveys syearmin syearmax %6.0g
format sy1984-sy2016 %7.0g
edit lfn datenfile varname varlab nonmis uniqvalpos valminp valmax nsurveys
syearmin syearmax sy1984 sy1985 sy1986
```

### 6.1.2 Auswahl von Observationen

```
edit <varliste> if <Bedingungen>
```

oder:

```
browse <varliste> if <Bedingungen>
```

Die Formulierung von if-Bedingungen für Stringvariablen ist weniger bekannt als die für numerische Variablen. Da der Metafile einige Stringvariablen enthält, sollen hier die wichtigsten String-Funktionen für Datenselektion anhand von Beispielen vorgestellt werden.

- … wenn die Zeichenfolge "erwerbs" im Inhalt von varlab vorkommt:

```
…if strpos(varlab, "erwerbs") > 0
```

Die Funktion strpos() gibt 0 zurück, wenn die gesuchte Zeichenfolge im Inhalt von varlab nicht vorkommt.

- …wenn die Zeichenfolge "erwerbs" oder "Erwerbs" in varlab vorkommt:

```
…if strpos(strlower(varlab), "erwerbs") > 0
```

In diesem Beispiel wurden die Stringfunktionen strpos() und strlower() ineinandergeschachtelt. Der Ausdruck besagt: … wenn die Zeichenfolge "(…)" im Inhalt von varlab vorkommt, nachdem er zu Kleinbuchstaben umgewandelt wurde (was gleichbedeutend ist mit: ...egal, ob die Zeichenfolge aus Groß- oder Kleinbuchstaben besteht).

- …wenn der Variableninhalt mit "bgp" beginnt:

```
…if strpos(datenfile, "bgp") == 1
```

oder:

```
…if substr(datenfile, 1,3) == "bgp"
```

Die erste Variante fragt danach, ob die Zeichenfolge "bgp" ab dem ersten Zeichen des Inhalts von datenfile vorkommt. Die zweite Variante fragt, ob die 3 Zeichen ab dem ersten Zeichen des Inhalts von datenfile "bgp" lauten.

Mehr Details und weitere String Funktionen siehe Stata-Befehl help string functions.

## 6.2 Auswahl von Variablen aus den SOEP Long-Files

Für eine geplante, thematisch noch unkonkrete Analyse sollen z.B. diejenigen Variablen aus SOEPlong angezeigt werden, die in mindestens 5 Wellen seit 2008 erhoben wurden, und die mindestens 500 verwertbare Fälle aufweisen. Die aufeinander aufbauenden Beispiele zeigen einen (vereinfachten) Konretisierungsprozess des Analyseplans anhand der verfügbaren SOEP-Variablen.

ACHTUNG Fallstrick: Die Bedingung so zu formulieren:

```
… if syearmin >= 2008 & nsurveys >= 5
```

wäre falsch, denn damit würden Variablen ausgeschlossen, welche die formulierten Anforderungen erfüllen, aber auch bereits vor 2008 erhoben wurden.

Stattdessen wird eine Hilfsvariable gebildet, die anzeigt, wie oft eine Variable seit 2008 erhoben wurde:

```
capture drop h1
egen h1 = anycount(sy2008-sy2016), values(1)
```

Die Hilfsvariable wird dann für die Formulierung des Selektionskriteriums verwendet:

```
ed lfn datenfile varname varlab nonmis nsurveys syearmin syearmax if h1 > 4
& nonmis >= 500
```

→ 1.949 Variablen in SOEP long (V33) erfüllen diese Bedingung.

Die Sichtung zeigt: es sind Zufriedenheitsvariablen darunter, diese sollen näher examiniert werden:

```
ed lfn datenfile varname varlab nonmis h1 if strpos(varlab, "ufriedenh") > 0 &
nonmis >= 500
```

→ es sind 5 Variablen aus dem File jugendl und 12 Variablen aus dem File pl. Wir entscheiden uns dafür, nur mit Daten aus pl zu arbeiten, wollen aber prüfen, ob es dort weitere Zufriedenheits-Variablen gibt, die wir mit unseren Bedingungen ausgeschlossen haben. An dem Kriterium "mindestens 500 verwertbare Fälle" wollen wir festhalten, senken aber die Anzahl der Wellen, in der die Variable in dem uns interessierenden Zeitraum erhoben wurde, auf "mindestens 1 mal":

```
ed lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1 if datenfile
== "pl" & nonmis >= 500 & h1 > 0 & strpos(varlab, "ufriedenh") > 0
```

Eine elegantere Art, mit sehr langen Befehlen umzugehen ist es, für die mehrfach verwendeten Befehlsteile Makros zu definieren:

```
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
local cond "datenfile == "pl" & nonmis >= 500 & h1 > 0"

ed `vars' if `cond' & strpos(varlab, "ufriedenh") > 0
```

→ es sind 37 Variablen, die wir nach der Häufigkeit, mit der sie in dem uns interessierenden Zeitraum erhoben wurden, sortieren. Als zweites Sortierkriterium nehmen wir varname, denn Variablen aus ein und der selben Itembatterie haben im SOEP in der Regel fortlaufende Variablennamen.

```
sort h1 varname
```

Nach Sichtung entscheiden wir uns für die schon vorher angezeigten Zufriedenheits-Variablen, die im interessierenden Zeitraum 9 mal erhoben wurden, plus die 4 Variablen, die nur im Surveyjahr 2016 erhoben wurden und die "Zufriedenheit mit [...] in den letzten 10 Jahren" messen. Diese Variablen markieren wir mit einer Flag-Variablen (siehe nächster Abschnitt).

## 6.3 Verwendung einer Hilfsvariablen zur Kennzeichnung von Auswahlmengen

Manche Auswahlmengen sind nur schwer in ein übersichtliches Auswahlkriterium zu fassen. Z.B. werden durch das Auswahlkriterium mitunter ein paar Variablen mit ausgewählt, die nicht zu der Zielgruppe gehören. Oder das Auswahlkriterium ist bereits sehr komplex, es sollen aber noch weitere Bedingungen hinzukommen. In solchen Fällen kann man eine Flag-Variable generieren, die anzeigt, ob eine Variable zur Auswahlmenge gehört. Der Flag wird dann schrittweise gesetzt oder wieder entfernt.

Obwohl diese Flag-Variable im edit-Modus auch manuell bearbeitet werden könnte, sollte man das nicht tun, sondern alle Schritte im Syntaxfile dokumentieren. Nur so ist die Datenaufbereitung zuverlässig reproduzier- und nachvollziehbar.

Um das obige Beispiel weiterzuführen:

Mit edit prüfen wir zuerst, ob die Auswahlkriterien richtig formuliert sind:

```
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
local cond1 "datenfile=="pl" & nonmis >= 500 & strpos(varlab, "ufriedenh") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, "letzte 10") > 0)"

ed `vars' if `cond1' & `cond2'
```

danach verwenden wir sie, um die Flag-Variable zu generieren:

```
capture drop f1
local cond1 "datenfile=="pl" & nonmis >= 500 & strpos(varlab, "ufriedenh") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, "letzte 10") > 0)"
gen f1 = cond(`cond1' & `cond2', 1, 0)
```

alternativ, falls der cond() Befehl nicht vertraut ist:

```
capture drop f1
local cond1 "datenfile=="pl" & nonmis >= 500 & strpos(varlab, "ufriedenh") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, "letzte 10") > 0)"
gen f1 = 0
replace f1 = 1 if `cond1' & `cond2'
```

(Das Ergebnis stets kontrollieren mit: edit if f1==1)

Es fällt eine merkwürdige Lücke zwischen der lfn 7520 (Variable plh0180) und der lfn 7522 (Variable plh0182) auf. Warum wird die dazwischen liegende Variable mit der lfn 7521 von unseren Auswahlkriterien nicht erfasst? (Anmerkung: wenn nicht Observationen gedroppt wurden, enthält die Variable lfn eine fortlaufende Nummer ohne Lücken).

```
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
local cond1 "datenfile=="pl" & nonmis >= 500 & strpos(varlab, "ufriedenh") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, "letzte 10") > 0)"
ed `vars' if (`cond1' & `cond2') | lfn == 7521
```

→ Der Grund für den Ausschluss: diese Variable wurde in dem interessierenden Zeitraum nur 4 mal erhoben, statt 9 mal, wie die angrenzenden Variablen. Eine unserer Bedingungen lautete: h1 == 9 oder (h1 == 1 und (…)). Wenn wir uns dafür entscheiden, die Variable in unsere Analye einzubeziehen, kennzeichnen wir sie in der Flagvariablen f1:

```
replace f1 = 1 if lfn == 7521
```

Generell muss man bei der Auswahl von Variablen sorgfältig prüfen, ob die erzielte Menge der gewünschten Menge entspricht. Das ist besonders wichtig, wenn das Variablenlabel Teil der Auswahlkriterien ist und es sich nicht um SOEPlong-, sondern um SOEP-Files handelt. Tippfehler und andere Unregelmäßigkeiten in den Labels der wellenspezifischen Files können leicht verhindern, dass eine eigentlich zur Zielgruppe gehörende Variable in die Auswahlmenge gelangt.

Bei der Endauswahl von Variablen für eine Analyse empfiehlt es sich außerdem, stets die umgebenden Variablen zu sichten, die u.U. die Auswahlkriterien nicht erfüllen, aber inhaltlich dazu gehören (z.B. selten genannte Items bei Itembatterien, Gesamt-KA-Variablen). Umgebende Variablen findet man über die angrenzenden lfn-Werte.

Unsere bisherige Variablenauswahl hat die laufenden Nummern 7511 - 7522 und 7676 bis 7679. Diese verwenden wir, um die angrenzenden Variablen zu sichten:

```
sort lfn
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
edit `vars' if (lfn > 7505 & lfn < 7528) | (lfn > 7670 & lfn < 7685)
```

→ Wir sehen: es gibt keine unmittelbar dazu gehörigen Variablen. Uns fallen aber die Variablen zur Häufigkeit bestimmter Gefühlsregungen in den letzten 4 Wochen ins Auge. Diese fügen wir unserer Auswahlmenge noch hinzu:

```
local cond3 "(lfn > 7505 & lfn < 7528) | (lfn > 7670 & lfn < 7685)"
replace f1 = 1 if (`cond3') & strpos(varlab, "letzte 4") > 0
```

Man beachte die zusätzlichen Klammern um `cond3' im replace-Befehl. Ohne diese Klammern würde die Bedingung wie folgt interpretiert: Entweder liegt lfn zwischen 7505 und 7528 ODER sie liegt zwischen 7670 und 7685 und in varlab findet sich die Zeichenfolge "letzte 4" - das wäre nicht das, was wir beabsichtigt haben. Wir möchten, dass die Bestimmung in varlab findet sich die Zeichenfolge "letzte 4" sich auf beide Nummernkreise bezieht.

Deshalb: stets nachkontrollieren, ob die Befehle das gewünschte Ergebnis erbringen!

Nachdem nun die angestrebte Auswahlmenge definiert ist, wird im nächsten Abschnitt gezeigt, wie man diese "abgreifen" und in der eigenen Syntax weiter verwenden kann.

## 6.4 Abgreifen der selektierten Variablennamen zur weiteren Verwendung

Um die Namen der selektierten Variablen abzugreifen, benutzen wir den Stata-Befehl levelsof, mit dem die Ausprägungen von Variablen ermittelt werden. Die Namen der SOEP-Variablen sind ja nichts anderes als Ausprägungen der Metafile-Variablen varname:

```
quietly levelsof varname if f1 == 1, clean local(vars)
global myvars "`vars'"
display "${myvars}"
```

Die erste Zeile dieser Befehlssequenz weist die Variablennamen dem lokalen Makro vars zu. Die Option clean bewirkt, dass keine Anführungszeichen um die Variablennamen ausgegeben werden. Mit levelsof können nur lokale Makros gefüllt werden. Damit die Liste während der gesamten Stata-Sitzung zur Verfügung steht, wird sie in der nächsten Zeile dem globalen Makro myvars zugewiesen.

Man braucht außerdem weitere Variablen, wie IDs und Gewichtungsfaktoren. Diese können entweder wie oben gezeigt, in der Flagvariable markiert werden, oder man fügt sie der Liste hinzu. Auf jeden Fall sollten die Variablen, die eine Observation im Datenfile eindeutig kennzeichnen, in der Liste ebenfalls enthalten sein - bei SOEPlong ist das pid und syear.

```
quietly levelsof varname if f1 == 1, clean local(vars)
global plmyvars "hid cid pid syear `vars'"
```

Hinweis: die Gewichtungsfaktoren (Hochrechnungsfaktoren) sind mit dem Kriterium if strpos(varname, "hrf") > 0 sowohl in den wellenspezifischen SOEP-Files wie in SOEPlong auffindbar. In SOEPlong befinden sie sich in den Datenfiles hpfad und ppfad, sie können also nicht, wie mit der folgenden Syntaxzeile, direkt als Teil der Liste zur Auswahl von Variablen aus einem Quelldatenfile benutzt werden. Das Vorgehen in diesem Fall beschreibt Abschnitt 6.5.

So können wir die zuvor erstellte Variablenliste benutzen, um eine Untermenge von Variablen aus dem Datenfile pl in den Arbeitsspeicher zu laden:

```
use ${plmyvars} using pl, clear
```

## 6.5 Vorgehensweise, wenn die Auswahlmenge aus mehreren Datenfiles stammt

Am einfachsten definiert man für jeden Datenfile eine eigene Liste, entweder manuell oder wie oben beschrieben. Es empfiehlt sich, den Datenfile und eventuell den Inhalt im Namen der Liste zu kennzeichnen:

```
global hrfppfadl "phrf phrfe"
```

Diese Liste der Hochrechnungsfaktoren und die oben schon definierte Liste plmyvars werden dann zum Zusammenspielen der benötigten Variablen in einem Zieldatenfile benutzt:

```
use ${plmyvars} using pl, clear
merge 1:1 pid syear using ppfadl, keepusing ${hrfppfadl}
```

Wenn die interessierenden Variablen aus einer Vielzahl von Datenfiles stammen, z.B. bei wellenspezifischen Files, würde man ebenfalls wie oben beschrieben, im Metafile eine Flagvariable generieren und die benötigten Variablen dort markieren. Mit levelsof datenfile.... wird dann eine Liste der betroffenen Datenfiles erstellt, um in einem zweiten Schritt in

einer Schleife über die Einträge dieser Liste (beispielsweise) filespezifische Variablenlisten zu erstellen. Der Zähler der Schleife wird aus der Zahl der Einträge in der Liste der Datenfiles gewonnen. Der aktive Datenfile ist immer noch der Metafile, f1 war die Flagvariable zur kennzeichnung der benötigten SOEP-Variablen im Metafile.

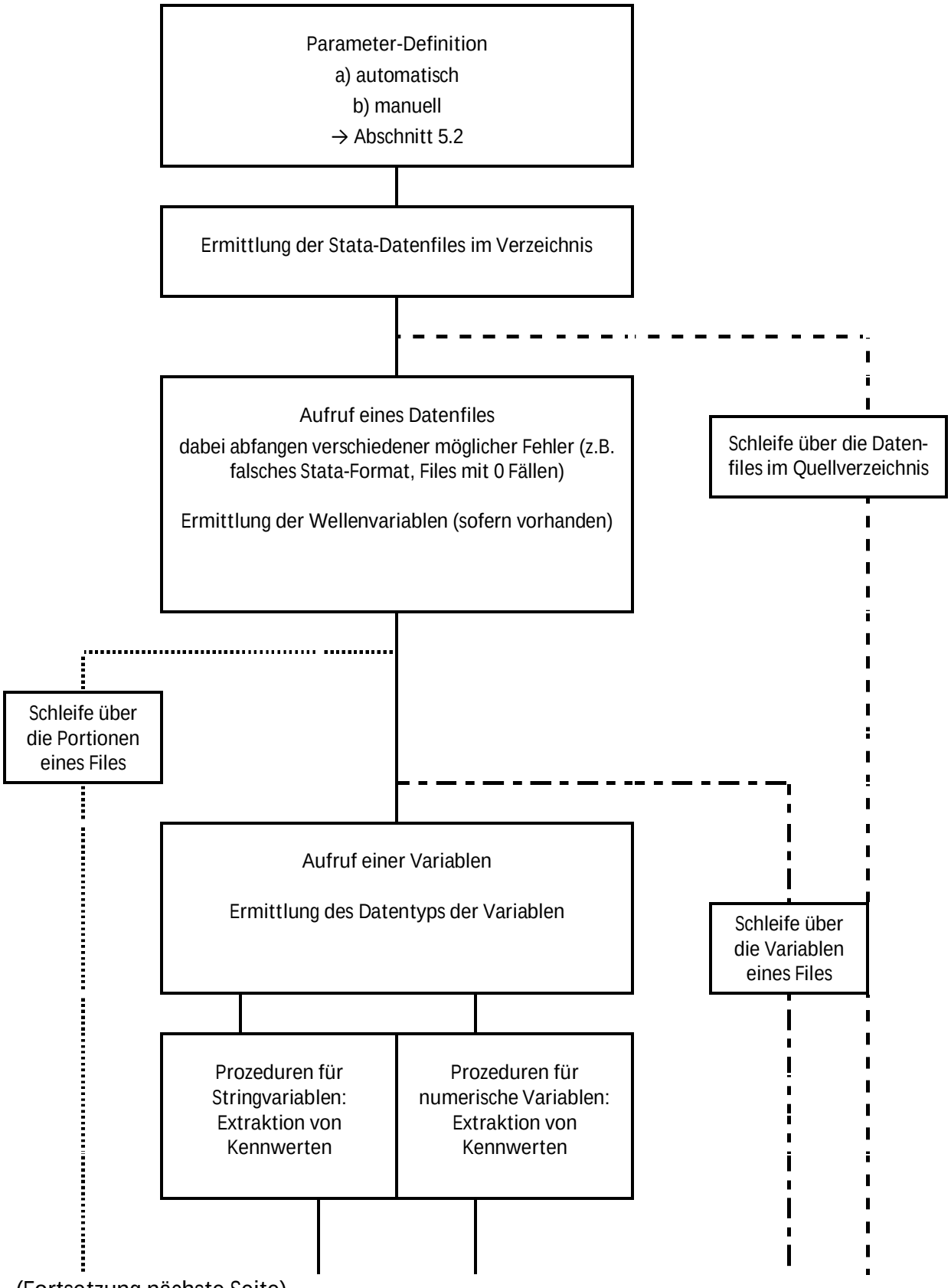
```
quietly levelsof datenfiles if f1 == 1, clean local(files)
global myfiles "`files'"
local nf : word count `files'
forvalues i = 1(1)`nf'   {
    local f : word `i' of `files'
    quietly levelsof varname if f1==1 & datenfile == "`f'", clean local(vars)
    global `f'vars "`vars'"
}
```

Die mit levelsof varname erzeugten Variablenlisten in lokalen Makros werden hier auf globale Makros kopiert, damit sie nach dem Ausführen der Befehlssequenz noch zur Verfügung stehen. Im selben Schritt werden die Variablenlisten datenfile-spezifisch umbenannt.

Für den weiteren Fortgang muss der Metafile nicht mehr im Arbeitsspeicher bleiben, da alle interessierenden Informationen auf Makros übertragen wurden. Deshalb können nun anhand der generierten filespezifischen Variablenlisten (gegebenenfalls in einer erneuten Schleife über die Liste der Datenfiles) die Befehle zusammengesetzt werden, mit denen die eigentlichen Daten aufgerufen werden. Jedoch ist zu beachten, dass die Liste der Datenfiles nur die reinen Dateinamen enthält, der Pfad muss noch hinzugefügt werden.

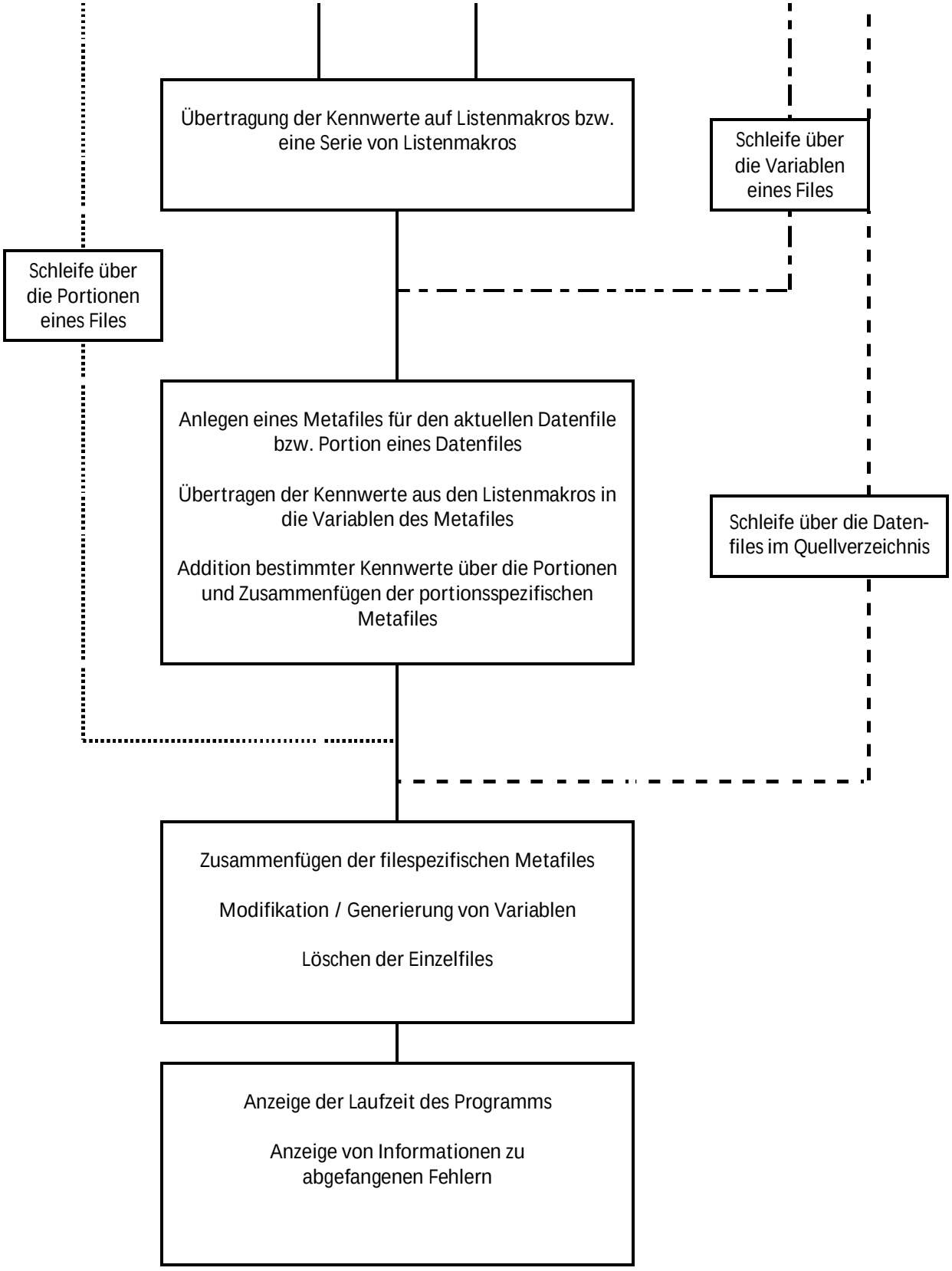
Die Beispiele in diesem Abschnitt sind als Anregung gedacht. Es lassen sich weitere Verwendungen für Variablen- und Dateilisten denken, die aus dem Metafile gespeist werden, z.B. könnte man damit programmgesteuert einen Do-File erzeugen. Im Rahmen dieses Papiers kann allerdings darauf nicht näher eingegangen werden.

# 7 Ablaufschema von SOEP-Metafile.do




(Fortsetzung nächste Seite)

(Fortsetzung von der vorhergehenden Seite)



# 8 Vollständige Syntax von SOEP-Metafile.do

```
/* SOEP-Metafile.do, v3.0.3 Klaudia Erhardt, last updated: 2018-05-26

Version for SOEP-Users

########################

Syntax compiles a Metafile from SOEP data files in a directory.

It is highly advisable to create two separate metafiles for SOEP- and SOEPlong-files

Approximative runtime of this job:
- 1 hrs when applied to all SOEP wave-specific files
- 3 hrs when applied to all SOEPlong files (portion size: 50). Larger portion sizes mean longer runtime

*/

/*#############################################################################################################
##############################                                        ###########################################
##############################  A) AUTOMATIC DEFINITION OF PARAMETERS -  ###########################################
##############################      no changes please!!!                ###########################################*/

clear
set more off
set varabbrev off, permanently
set output error

timer clear
timer on 1       /* Measuring the runtime of this job */
tempname time
scalar `time' = 0

/* prefix for stata14 string functions */
local u ""
if c(stata_version)>=14 {
    local u "u"
}

/* Date specification, is appended to resulting files' names */
local datum : display %td_CY-N-D  date("$S_DATE", "DMY")
local datum = `u'strtrim("`datum'")
```

```
/*##############################################################################################################
##############################                                    ##############################################
##############################  B) DEFINITION OF PARAMETERS       ##############################################
##############################                                    ##############################################*/


/* ########### affix for file names          ######################################################################## */

local we "_SOEPv33_long"  /* Label for wave and version. Will be appended to the resulting files' names. Must
                             comply with the file name conventions of Stata, i.e. no period character.
                             May be defined as empty, but then you have to take care to prevent unintentional
                             overwriting of resulting metafiles. */


/*  #######  Paths and directories    ###########################################################################

NOTE: specification of paths without ending slash ! */

/* Path to source-files directory - mandatory */

local data ""

/* Examples:
local data "//hume/soep-data/DATA/soep33/stata"
local data "c:\Users\myname\soep\soeplong"
*/


/* Paths to resulting files directories - all three are mandatory, but may contain the same path */

local pfad1 ""  /* Main target path */
local pfad1d "`pfad1'/Datenfiles"    /* local data directory for resulting metafile */
local pfad1o "`pfad1'/Output"       /* local output directory - storage location for logfile */


/* Examples:

A) all resulting files are written to the data directory:
local pfad1 "`data'"
local pfad1d "`data'"
local pfad1o "`data'"
```

```
B) resulting files are written to different subdirectories of main directory:
local pfad1 "//smith/users/kerhardt/Zuwanderer/Datenverknuepfung_STATA/SOEP_Variablenuebersicht" /* Main loc path */
local pfad1d "`pfad1'/Datenfiles"    /* local data directory for resulting file */
local pfad1o "`pfad1'/Output"       /* local output directory - storage location for logfile */

*/


/* ########### Source files  ############################################################################## */

local files "*.dta" /* selection criteria within the files of the source directory (`data')
                       you may use wildcards to include only a subset of files.
                       If defined as empty, default "*.dta" is put into effect.  */



/* ###########    Options    ############################################################################## */

local port 50     /* Number of variables per portion for segmented processing of the data files.
                     Larger portions result in a longer runtime of the job.
                     If "port" > than half the number of variables in a data file, the processing of this
                     file will be non-segmented. */

local varlab "varlab_de"  /* Name of the variable in the metafile that contains the variable labels of the data
                             files - if defined as empty, default "varlab" is put into effect.  */


/* ########### Target file names  ######################################################################### */

local result "metafile`we'_`datum'"  /* resulting data set from this syntax */
local reslab ""  /* Label for the resulting data set, may stay empty */
local lg "metafile`we'_`datum'.log"       /* log-file */


/* ########### Specification of wave or survey-year indicators     ################################################ */

/* Priority list of possible wave indicators: highest priority first. */
local syrlist "syear welle wave intyear erhebj bioyear"    /* Usually you don't have to change this, but may stay empty
                                          or can be uncommented if no survey year indicators are wanted. */

local wave1 "1984"   /* earliest wave  - may stay empty if no survey year indicators are wanted. */
local wave2 "2016"  /* newest wave  - may stay empty if no survey year indicators are wanted.   */
```

```
/* ##########################                                    ##############################################
############################  END: DEFINITION OF PARAMETERS          ##############################################
############################                                         ##############################################
################################################################################################################ */

/* ######################################## */

capture log close
log using "`pfad1o'/`lg'", replace

/* no changes in the follwing syntax afforded !!!!!!!!!!!!!!!!
         */

/*################################################################################################################
############################                                         ##############################################
############################  START OF THE PROGRAM                   ##############################################
############################                                         ##############################################*/

/* ##########    Default definition of macros  - no changes please !!! ######################## */


if `u'strlen("`files'") == 0 {
    local files "*.dta"
}

if strlen("`port'") == 0 {
    local port 50
}
if `port' == 0       {
    local port = 50
}

local nw = `wave2'-`wave1'+1 /* ATTENTION: don't change the position of this specification to come after the defi-
                                nition of wave1 and wave2! Otherwise a variable sy0 will be generated if no survey
                                year variables are wanted */
if `u'strlen("`wave1'") == 0 {
    local wave1 "0"
}
if `u'strlen("`wave2'") == 0 {
    local wave2 "0"
}
if `u'strlen("`varlab'") == 0{
    local varlab "varlab"
}
```

```
/* ############################################################ */

local message "    " _n ///
    "The data files to be included are determined. " _n ///
    "After that, a metafile is generated from each data file. Subsequently, the single meta files " _n ///
    "are compiled to a joint metafile which contains data to all variables of all included dta-files. "   _n ///
            "    "   _n ///
            "Processing... Please wait. " _n ///
            "    "
set output proc
display "`message'"
set output error

local infile : dir "`data'" files `"`files'"'
local a : word count `infile'
local infile = `u'subinstr(`"`infile'"',`"""',"",`a'*2)  /* die Anfuehrungszeichen entfernen */
local infile = `u'subinstr(`"`infile'"',".dta","",`a') /* den Zusatz .dta entfernen */

local message "    " _n ///
            " All data files in directory `data' will be included "   _n ///
            "   that comply with criteria: `files'   "

set output proc
display " `message' "
set output error

local nds : word count `infile'
local a = `nds'
local message " `a' data files will be processed"
set output proc
display " `message' "
set output error

local svd "" /* updated list, in case a data file from the infile-list is not found or cannot be opened */
local nvrs = 0  /* local to contain the no of vars of a file (for segmented processing) */
local nvrsnm = 0 /* local to contain the no of nonmissing vars of a file (for segmented processing) */

forvalues i = 1(1)`a'    {       /*  Loop over the files */
    local datei = word(`"`infile'"', `i')
    capture quietly describe using "`data'/`datei'.dta", varlist

    if _rc == 601    {
        local fnexist "`fnexist' `datei'"  /* list: files not found */
    }
```

```
if _rc == 610 {
    local s14file "`s14file' `datei'"  /* list: files could not be opened */
}
if _rc == 0  {  /* if file exists */

    local allvars = r(varlist)
    local nsik = r(k)
    local n = `nsik'
    local message "Processing file `i', file name: `datei'.dta containing `n' Vars at $S_TIME"
    set output proc
    display "`message'"
    set output error


    /* Identification of the wave indicators that are present in the actual date file. Here: without
        consideration of the priority. Only for loading the wave indicators in each portion of the
        segmented file.
    */
    if `u'strlen("`syrlist'") > 0{
        local syrl ""
        local ns : word count `syrlist' /* above defined list of wave indicators */

        /* Generation of the list of wave indicators that in fact are present in the file at hand */
        forvalues j = 1(1)`ns'   {   /* loop over the wave indicators */
            local syr = word(`"`syrlist'"', `j')
            forvalues l = 1(1)`n'    {   /* loop over the variables of the file at hand */
                local av = word(`"`allvars'"', `l')
                if "`av'" == "`syr'"  {
                    local syrl "`syrl' `syr'"
                }
            }
        }
        local nsyrl : word count `syrl'  /* list of the wave indicators that in fact are present */
    }
    /* Segmentation of big files */
    local n = `nsik'
    if `n' > `port'  {
        local list ""
        local p = int(`n'/`port')
        local x = `p' - 1
        local span = `port'
        local beg = -`port' + 1
        forvalues i = 1(1)`x'    {
            local p1 = `beg' + `span'
```

```
            local p2 = `p1' + `span' - 1
            local w1 = word(`"`allvars'"', `p1')
            local w2 = word(`"`allvars'"', `p2')
            local list "`list' `w1'-`w2'"
            local beg = `p1'
        }
        local p1 = `beg' + `span'
        local p2 = `n'
        local w1 = word(`"`allvars'"', `p1')
        local w2 = word(`"`allvars'"', `p2')
        local list "`list' `w1'-`w2'"
    }
    if `n' <= `port' {
        local w1 = word(`"`allvars'"', 1)
        local w2 = word(`"`allvars'"', `n')
        local list "`w1'-`w2'"
    }
    local zn : word count `list'
    local z = `zn'

    forvalues s = 1(1)`z'    {           /* Loop over the portions */
        local nvrs = 0  /* local fuer die nVars eines Files bei portionierten Files */
        local nvrsnm = 0 /* local fuer die nonmissing nVars eines Files bei portionierten Files */

        local p = word(`"`list'"', `s')
        if `z' == 1  {
            capture use `p' using "`data'/`datei'.dta", clear
        }
        if `z' > 1   {
            capture use `p' `syrl' using "`data'/`datei'.dta", clear
        }
        if r(N) == 0 {    /* If the file has no observations */
            if `u'strpos("`nobsfile'", " `datei' ") == 0     {
                local nobsfile "`nobsfile' `datei' " /* List of files with no observations */
            }
        }
        if r(N) > 0 {  /* If the file has observations  */
            if `s' == `z'    {    /* in the last portion */
                local svd = `u'strtrim("`svd' `datei'")
            }
            if `z' > 1   {
                local message "     Processing `datei'.dta, portion `s' of `z': `p' at $S_TIME"
                set output proc
                display "`message'"
```

```
        set output error
    }
    /* compile variable list, determine no of entries in the list, determine loop counter, */
    quietly describe, varlist
    local allvars = r(varlist)
    local nvars : word count `allvars'  /* you could also use r(k) instead  */
    local n = `nvars'

/* ########  Compile lists of the information that is to be transmitted to the metafile:############
   ########  Indicators related to variables are compiled in macro lists (1 entry per   ############
   ########    variable) while looping over the variables of the file                   ############
   ########  Indicators related to the whole file are determined outside the loop over  ############
   ########    the variables.                                                           ######## */

    /* the following lists have to be emptied before a new file is processed */
    local dtyp ""
    local lmin ""
    local lmax ""
    local uval ""
    local uvalp ""
    local valu ""
    local valup ""
    local nzero ""
    local vmax ""
    local vmin ""
    local vminp ""
    local smiss ""
    local nmiss ""
    local nsvys ""
    local symin ""
    local symax ""
    local sylev ""
    local undc ""
    local vlab ""
    local vlabmx ""
    local values ""
    local labels ""

    /* No of vars with nonmissing values. Will be incremented when looping over the variables, therefore
        set to 0 before new file ist processed */
    local nvn = 0

    /* Determine the wave indicator with the highest priority in the file at hand. The procedure runs
        through the priority list from last to first entry. Every time a wave indicator is found to have
```

```
    nonmissing values in the file at hand, syv is updated. Therefore at the end syv contains the
    name of the wave indicator with the highest priority.
*/

if `u'strlen("`syrlist'") > 0{
    local syv ""
    local ns : word count `syrlist' /* the priority list of the wave indicators */

    forvalues j = `ns'(-1)1  {
        local syr = word(`"`syrlist'"', `j')
        capture confirm variable `syr', exact
        if _rc == 0 {       /* `syr' is present in the file at hand */
            quietly summarize `syr', meanonly
            local min = r(min)
            local max = r(max)
            if r(min) < . & r(max) > 0    {   /* `syr' has values between 0 and .  */
                local syv "`syr'"
            }
        }
    }
}


/* ########   Start of variable-related routines: extraction of indicators and           ##########
   ########        transfer to macro lists                                                ####### */


forvalues k = 1(1)`n'    {
    /* Empty all var-related "messenger" macros or count entries in lists */
    local vl ""   /* name of attached value label set */
    local vlmx "" /* highest labeled value  */
    local uv ""      /* no. of unique values */
    local uvp "" /* no. of unique values >= 0 */
    local ln ""      /* minimal length of stringvar */
    local lx ""      /* maximal length of stringvar */
    local sms "" /* no. of system-missings ( >= . ) */
    local nms "" /* no. of non-missing obs. */
    local sys "" /*  no. of surveys (waves) where `var' has nonmissing values  */
    local syn "" /* earliest survey year */
    local syx "" /* latest survey year  */
    local vu ""      /* value, if only 1 unique value */
    local vup "" /* value, if only 1 unique value >= 0 */
    local nz ""      /* frequency of value 0 */
    local ud ""      /* undocumented value (if value labels are attached to variable) */
```

```
local vx ""      /* maximum */
local vn ""      /* minimum */
local vp ""      /* minimal value >=0  */
local sylev ""  /* survey years, in which the variable has nonmissing values */

/* process first (next) variable of the list of all variables */
local var = word("`allvars'", `k')
/* determine data type of `var' */
local dt : type `var'

/* name of attached value label set and highest labeled value */
local vlx : value label `var'

if "`vlx'" == "" {   /* if no label set is attached to `var' */
    local vl "-2"
    local vlmx "-200"
}
if "`vlx'" != "" {   /* if a label set is attached to var... */
    quietly capture label list `vlx'

    if _rc > 0       {   /* ... but the label definition does not exist */
        local vl "-2"
        local vlmx "-200"
    }
    if _rc == 0      {   /* ... and the label definition exists */
        local vl `"`vlx'"'
        local vlmx = r(max)
    }
}

/* Number of unique values of `var' */
sort `var'
capture drop h1
by `var': gen byte h1 = cond(_n==1, 1, 0)
quietly summarize h1, meanonly
local uv = r(sum)


/* ###### data-type depending indicators: routines for stringvars ###### */

if  `u'strpos("`dt'", "str") > 0      {

    /* max und min length */
    capture drop h3
```

```
        gen h3 = `u'strlen(`var')
        quietly summarize h3 if h3 > 0, meanonly
        local ln = r(min)
        local lx = r(max)
        if `lx' == . {
            local ln = 0
            local lx = 0
        }

        /* no. of non-missings (neither empty nor missing-code) and of value 0 (= empty strings) */
        capture drop h2
        gen h2 = cond(`u'strlen(`u'strtrim(`var')) == 2 & `u'strpos(`var', "-") > 0, -1, 1)
        quietly replace h2 = 0 if `u'strtrim(`var') == ""
        quietly inspect h2
        local nms = r(N_pos)
        local nz = r(N_0)


    /* no. of values >= 0  (file is still sorted by `var' )
         uses the previously generated variable h2
         (h2 == 1 if stringvar is neither empty nor missing code)
         The "throwaway-variable" h3 indicates obs. with the first new value
    */
        capture drop h3
        by `var': gen byte h3 = cond(h2 == 1 & _n==1, 1, 0)
        quietly replace h3 = 0 if `var'  == ""  /* empty values are not to be counted */
        quietly summarize h3, meanonly
        local uvp = r(sum)


        /* Survey year indicators - uses the previously generated variable h2  */

        if `u'strlen("`syv'") > 0  {            /* wave indicator is found in file */
            quietly inspect `syv' if `syv' > 0 & `syv' < . & h2 == 1   /* if wave indicator and var
                                                                                                    are both
nonmissing */
            local sys = r(N_unique)
            quietly summarize `syv' if h2 == 1
            local syn = r(min)
            local syx = r(max)
            if `sys' == 0    {
                local syn = -9
                local syx = -9
            }
```

```
            quietly levelsof `syv' if h2 == 1 , missing local(sylev)

        }
        if `u'strlen("`syv'") == 0 {  /* no wave indicator is found in file  */
            local sys = -1
            local syn = -1
            local syx = -1
        }

        /* List for the 100 value label variables */
        quietly levelsof `var' if h2 == 1 & `uvp' < 101 , clean separate(*) local(labels)
        if `u'strlen(`"`labels'"') > 0    {
            local labels = `"`labels'*"'
        }

        /* assign ersatz values for indicators that do not apply to stringvars */

        local vu = -200
        local vp = -200
        local vup = -200
        local ud = -2
        local vx = -200
        local vn = -200
        local vlmx = -200
        local sms = 0


    }       /* end: routines for stringvars */


    /* ###### data-type depending indicators: routines for numvars (= no stringvars) ###### */

    /* ersatz for max-min-str-length, uniqe value (file is still sorted by var) */
    if  `u'strpos("`dt'", "str") == 0     {
        local ln = -2
        local lx = -2
        if `uv' == 1 {
            local vu = `var'[1]
        }
        if `uv' != 1 {
            local vu = -900
        }

        /* no. of unique vlaues >= 0  (file is still sorted by var) */
```

```
capture drop h2
by `var': gen byte h2 = cond(`var' >= 0 & `var' < . & _n==1, 1, 0)
quietly summarize h2, meanonly
local uvp = r(sum)

/* positive value, if `var' has only 1 positive value */
if `uvp' == 1    {
    gsort -h2        /* ATTENTION: re-sorting of file */
    local vup = `var'[1]
}
if `uvp' != 1    {
    local vup = -900
}

/* Freq. of value 0, of nonmiss and of unlabeled values */
quietly inspect `var'
local nz = r(N_0)
local ud = r(N_undoc)
if `ud' == . {
    local ud = 0
}
local nms = r(N_0) + r(N_pos)


/* Freq. of system-missings ( >= .)
ATTENTION pitfall: if there are no system missings, all return codes are . */
quietly misstable summarize `var' if `var' >= 0
local sms = r(N_gt_dot) + r(N_eq_dot)
if `sms' == .    {
    local sms = 0
}

/* maximal and minimal value of `var' */
quietly summarize `var', meanonly
local vx = round(r(max), .001) /* rounded in order to not exceed the max length of macros */
local vn = round(r(min), .001) /* rounded in order to not exceed the max length of macros */

/* minimal positive value of `var' */
quietly summarize `var' if `var' >= 0 & `var' < ., meanonly
local vp = round(r(min), .001) /* rounded in order to not exceed the max length of macros */
if `vp' == . {   /* if `var' has no values >= 0  */
    local vp = -900
}
```

```
                    /*  #### Survey year indicators
#########
                        no. of survey years and earliest and latest survey year                          */

                    if `u'strlen("`syv'") > 0  {      /* wave indicator is found in file */
                        quietly inspect `syv' if `syv' > 0 & `syv' < . & `var' >= 0 & `var' < .
                        local sys = r(N_unique)
                        quietly summarize `syv' if `syv' > 0 & `syv' < . & `var' >= 0 & `var' < ., meanonly
                        local syn = r(min)
                        local syx = r(max)
                        if `sys' == 0    {
                            local syn = -9
                            local syx = -9
                        }
                    }

                    if `u'strlen("`syv'") == 0 {  /* no wave indicator is found in file */
                        local sys -1
                        local syn -1
                        local syx -1
                    }

   /* ################## Generation of the sy####-, val###-,  and lab### vars        ########################
      #################        fill macro lists for each of them                          ################## */


                    /* ##########   Generation of the survey year lists                  ################### */

                    /* create list of survey years where `var' has nonmissing values (not yet standardized).
                        the option missing of levelsof effects that level "." von syv is included   */

                    if `u'strlen("`syv'") > 0  {
                        quietly levelsof `syv' if `var' >= 0 & `var' < . , missing local(sylev)
                    }

                    /* ##########   Generation of the lists for values and value labels         ############# */

                    quietly levelsof `var' if `var' >= 0 & `uvp' < 101, local(values)
                    local c : word count `values'
                    if `c' > 0   {   /* if `var' has nonmissing unique values from 1 to 100 */
                        if "`vlx'" != "-2"     {            /* if a labelset is attached to `var' */
                            local labels ""
                            forvalues j=1(1)`c'       {
                                local wx : word `j' of `values'
```

```
                    local lab : label (`var') `wx', strict
                    local lab `"`lab'*"'  /* asterix used as a delimiter for the entries because
                                             labels in SOEP do not contain any asterixes  */
                    if "`lab'" == "*"    {
                        local lab "###*"
                    }
                    local labels `"`labels'`lab'"'
                }
            }
        }

}   /* end: routines for numvars */


/* Compile lists: each has a value for each variable  */
local uval "`uval' `uv'"        /* no. of unique values */
local uvalp "`uvalp' `uvp'"     /* no. of unique values >= 0 */
local lmin "`lmin' `ln'" /* minimal length of stringvar  */
local lmax "`lmax' `lx'" /* maximal length of stringvar */
local valu "`valu' `vu'"        /* value, if only 1 unique value */
local valup "`valup' `vup'"     /* value, if only 1 unique value >= 0 */
local nzero "`nzero' `nz'"      /* frequency of value 0 */
local vmax "`vmax' `vx'" /* maximum */
local vmin "`vmin' `vn'" /* minimum  */
local vminp "`vminp' `vp'"      /* minimal value >=0 */
local nmiss "`nmiss' `nms'"     /* no. of non-missing obs. */
local smiss "`smiss' `sms'"     /* no. of system-missings   */
local nsvys "`nsvys' `sys'"  /* no. of surveys (waves) where `var' has nonmissing values */
local symin "`symin' `syn'" /* earliest survey year  */
local symax "`symax' `syx'" /* latest survey year */
local undc "`undc' `ud'"        /* undocumented value (if value labels are attached to variable)  */
local vlab "`vlab' `vl'"        /* name of attached value label set */
local vlabmx "`vlabmx' `vlmx'"  /* highest labeled value */



/* compile lists: each list has several values for one variable. The lists are distinguished by
    the index k of the k-th variable */

/* compile list of  standardised survey years for k-th variable */

local sytemp ""      /* list of  standardised survey years for k-th variable */
if "`sylev'" != ""   {   /* list of survey years where `var' has values >= 0, filled above */
    local c : word count `sylev'
```

```
            forvalues j = 1(1)`c'      {
                local sla : word `j' of `sylev'
                local slan = real("`sla'")

                /* treatment of disparate formats of wave indicators in data */
                if `slan' > 0 & `slan' < 84   {
                    if `u'strpos("`datei'", "ost") > 0    {
                        local slan = `slan' + 6
                        local slan = `slan' + 1983
                    }
                    if `u'strpos("`datei'", "ost") == 0    {
                        local slan = `slan' + 1983
                    }
                }
                if `slan' >= 84 & `slan' <= 99      {
                    local slan = `slan' + 1900
                }
                /* more inadmissible values of `slan' are filtered out and compiled in a list
                            further down */

                local sytemp "`sytemp' `slan'" /* compile the list of standardised survey years */
            }
            local sylev`k' "`sytemp'"
        }

        /* k-th list of values if `var' has max. 100 nonmis unique values */
        local c : word count `values'
        if `c' > 0   {
            local values`k' "`values'"
        }

        /* k-th list of labels if `var' has max. 100 nonmis unique values */
        if `u'strlen(`"`labels'"') > 0  {
            local labels`k' `"`labels'"'
        }

        /*  update counting for no. of nonmissing vars  */

        if `nms' > 0 {
            local nvn = `nvn' + 1
        }
    }
```

```
/* #### End: routines for the variables of a file, Start: routines for a file (portion) #### */
capture drop h1
capture drop h2
capture drop h3


/* No. of obs, No. of vars and No. of nonmissing vars of a file  */
local no = c(N)  /* No. obs */
local nv = c(k)  /* No. of vars */
/* (no. of vars in file with nonmissing values is updated before the end of the variable loop) */


/* Generate new file where each variable of the data file is an observation */
describe, replace clear
if c(N) > 0      {  /* if there are variables in the file ... */
    keep name varlab type isnumeric
    rename name varname
    rename varlab `varlab'
    /* Indicator of the data file the variable springs from */
    capture drop datenfile
    gen str datenfile = "`datei'"
    lab var datenfile "source file"
    order datenfile, first
    /* No. of variables (file related) */
    capture drop nvrs
    gen nvrs = `nv'
    lab var nvrs "No. of vars in file"
    /* No. of variables with nonmissing values (file related)*/
    capture drop nvrsnm
    gen nvrsnm = `nvn'
    lab var nvrsnm "No. of vars in file with nonmissing values"
    /* No. of observations (file related)*/
    capture drop nobs
    gen nobs = `no'
    lab var nobs "No. of obs. in file"
    /* No. of nonmissing observations */
    capture drop nonmis
    gen long nonmis = 0
    lab var nonmis "No. of nonmiss. obs. (variable related)"
    /* Zahl der Sysmis-Observationen */
    capture drop sysmis
    gen long sysmis = 0
    lab var sysmis "No. of system-missing obs. (variable related)"
    /* Number of unique values */
```

```
capture drop uniqvals
gen uniqvals = 0
lab var uniqvals "No. of unique values"
/* Number of unique values >= 0 bzw. nonmissing */
capture drop uniqvalpos
gen uniqvalpos = 0
lab var uniqvalpos "No. of unique values >= 0/nonmiss"
/* Value if only 1 unique value */
capture drop valuniq
gen long valuniq = .
lab var valuniq "value if only 1 unique value"
/* pos. Wert, wenn es nur 1 pos. Auspraeg. gibt */
capture drop valuniqpos
gen long valuniqpos = .
lab var valuniqpos "value if only 1 unique value >= 0"
/* frequency of value 0 (numvars) */
capture drop nvalzero
gen long nvalzero = .
lab var nvalzero "freq of value 0"
/* minimal value of variable */
capture drop valmin
gen long valmin = .
lab var valmin "minimal value"
/* minimal value of variable >= 0 */
capture drop valminp
gen long valminp = .
lab var valminp "minimal value >= 0"
/* maximal value of variable */
capture drop valmax
gen long valmax = .
lab var valmax "maximal value"
/* name of attached valuelabel set */
capture drop vallabset
gen str vallabset = ""
lab var vallabset "name of valuelabel set "
/* undocumented Observations (if value labels are attached to variable) */
capture drop undoc
gen long undoc = .
lab var undoc "No. of undocumented obs. (if var is labelled)"
/* highest labeled value */
capture drop vlabmax
gen vlabmax = .
lab var vlabmax "highest labeled value"
/* minimal length (stringvariables) */
```

```
capture drop strmin
gen long strmin = .
lab var strmin "min. length (stringvars)"
/* maximal length (stringvariables) */
capture drop strmax
gen long strmax = .
lab var strmax "max. length (stringvars)"

/* variables for wave indicators */
capture drop wind
gen str wind = "`syv'"
if `u'strlen("`syv'") == 0      {
    quietly replace wind = "-1"
}
lab var wind "wave indicator"
capture drop nsurveys
gen byte nsurveys = 0
lab var nsurveys "No. of waves"
capture drop syearmin
gen syearmin = 0
lab var syearmin "earliest survey year"
capture drop syearmax
gen syearmax = 0
lab var syearmax "latest survey year"

/* generate blank variables for the survey years */
local n = `nw'
forvalues j = 1(1)`n'    {
    local scrpt = `j'-1+`wave1'
    capture drop sy`scrpt'
    gen int sy`scrpt' = 0
}
/* generate blank variables for the values (only vars with 1-100 unique values) */
forvalues j = 1(1)100    {
    local scrpt = `j'
    capture drop val`scrpt'
    gen val`scrpt' = .
}
/* generate blank variables for the value labels */
forvalues j = 1(1)100    {
    local scrpt = `j'
    capture drop lab`scrpt'
    gen str lab`scrpt' = ""
}
```

```
/* Note: the variables undoc, val# and lab# are modified further down, after the single
    files have been assembled */

local n  = `nvars'
forvalues k = 1(1)`n'    {
    local var : word `k' of `allvars'
    local uv : word `k' of `uval'
    local uvp : word `k' of `uvalp'
    local vu : word `k' of `valu'
    local vup : word `k' of `valup'
    local nz : word `k' of `nzero'
    local vx : word `k' of `vmax'
    local vn : word `k' of `vmin'
    local vp : word `k' of `vminp'
    local lx : word `k' of `lmax'
    local ln : word `k' of `lmin'
    local nms : word `k' of `nmiss'
    local sms : word `k' of `smiss'
    local sys : word `k' of `nsvys'
    local syn : word `k' of `symin'
    local syx : word `k' of `symax'
    local ud : word `k' of `undc'
    local vl : word `k' of `vlab'
    local vlmx : word `k' of `vlabmx'

    quietly {
        replace nonmis = `nms' if varname == "`var'"
        replace sysmis = `sms' if varname == "`var'"
        replace uniqvals = `uv' if varname == "`var'"
        replace uniqvalpos = `uvp' if varname == "`var'"
        replace valuniq = `vu' if varname == "`var'"
        replace valuniqpos = `vup' if varname == "`var'"
        replace nvalzero = `nz' if varname == "`var'"
        replace valmax = `vx' if varname == "`var'"
        replace valmin = `vn' if varname == "`var'"
        replace valminp = `vp' if varname == "`var'"
        replace strmax = `lx' if varname == "`var'"
        replace strmin = `ln' if varname == "`var'"
        replace nsurveys = `sys' if varname == "`var'"
        replace syearmin = `syn' if varname == "`var'"
        replace syearmax = `syx' if varname == "`var'"
        replace undoc = `ud' if varname == "`var'"
        replace vallabset = "`vl'" if varname == "`var'"
```

```
replace vlabmax = `vlmx' if varname == "`var'"

/* fill in the sy#-variables for the survey years */

local nok " " /* list: inadmissible values of the wave indicator (blank is important) */

if `u'strlen("`syv'") == 0    {      /* if there was no wave indicator in the file */
    local mark 0
    local c = `nw'
    forvalues j = 1(1)`c'     {
        local scrpt = `j'-1+`wave1'
        replace sy`scrpt' = -1
    }
}

if `u'strlen("`syv'") > 0     {      /* if there was a wave indicator in the file */
    local c : word count `sylev`k''
    forvalues j = 1(1)`c'     {
        local slan = word("`sylev`k'", `j')
        local mark = real("`slan'")
        if `mark' >= `wave1' & `mark' <= `wave2' {
            capture confirm new variable sy`slan', exact
            if _rc != 0       {
                replace sy`slan' = 1 if varname == "`var'"
            }
        }
        if `mark' < `wave1' | `mark' > `wave2' { /* Cond. identical to _rc == 0 */
            if `u'strpos("`nok'", " `mark' ") == 0      {
                local nok "`nok' `mark'"
            }
        }
    }
    local sylev`k' ""  /* emptied, otherwise information migth be transferred to
                                processing of next file */

}

/* fill in the val#-variables */

if `u'strlen("`values`k''") > 0  {
    local c : word count `values`k''
    forvalues j = 1(1)`c'     {
        local val = word("`values`k''", `j')
        local valn = real("`val'")
```

```
                    capture confirm new variable val`j', exact
                    if _rc != 0    {
                        replace val`j' = `valn' if varname == "`var'"
                    }
                }
                local values`k' "" /* emptied, otherwise information migth be transferred to
                                      processing of next file */
            }

            /* fill in the lab#-variables  */
            if `u'strlen(`"`labels`k''"') > 0  {
                local str = `"`labels`k''"'
                local a = `u'strpos(`"`str'"', "*")
                local x = 1
                while `a' > 0      {
                    local lb = `u'substr(`"`str'"', 1,`a'-1)
                    capture confirm new variable lab`x', exact
                    if _rc != 0    {
                        replace lab`x' = `"`lb'"' if varname == "`var'"
                    }
                    local str = `u'substr(`"`str'"',`a'+1,.) /* rest until end of string */
                    local a = `u'strpos(`"`str'"', "*")
                    local x = `x' + 1
                }
                local labels`k' "" /* emptied, otherwise information migth be transferred to
                                      processing of next file */
            }

        }  /* closing bracket for: quietly */
    }   /* closing bracket for loop over variables in allvars */

    if `u'strlen("`nok'") > 0  & "`nok'" != " " {
        local message "   " _n ///
            "The wave indicator `syv' from file `datei' "_n ///
            "   contains the inadmissible value(s) `nok'"
        set output proc
        display "`message'"
        set output error
    }
    save "`pfad1d'/`datei'_xxx`s'.dta", replace

    if `s' == `zn'       {   /* In a last portion */
        if `zn' > 1      {
            local p = `zn'
```

```
                            forvalues t = 1(1)`p'  {
                                use "`pfad1d'/`datei'_xxx`t'.dta", clear
                                local nvrs = `nvrs' + nvrs[1]
                                local nvrsnm = `nvrsnm' + nvrsnm[1]
                            }
                            local p = `zn'
                            use "`pfad1d'/`datei'_xxx1.dta", clear
                            forvalues t = 2(1)`p'  {
                                append using "`pfad1d'/`datei'_xxx`t'.dta"
                            }
                            /* diminuishing by no. of surveyindicator-vars */
                            local abz = (`zn'-1)*`nsyrl'
                            quietly replace nvrs = `nvrs'-`abz'
                            quietly replace nvrsnm = `nvrsnm'-`abz'
                            duplicates drop
                        }
                        save "`pfad1d'/`datei'_meta`we'.dta", replace
                        /* Portionsfiles loeschen */
                        local p = `zn'
                        forvalues t = 1(1)`p'      {
                            capture erase "`pfad1d'/`datei'_xxx`t'.dta"
                        }
                    }   /* closing bracket for: in a last portion */
                }   /* closing bracket for: generate single meta file */
            }   /* closing bracket for: processing of a file / portion */
        }   /* closing bracket for: if there are observations in the file */
    }   /* closing bracket for: if file exists */

}   /* closing bracket for: loop over all data files  */

if `u'strlen("`svd'") > 0{
    /* compile the single meta files  */

    local nds : word count `svd'
    local a = `nds'
    if `a' > 1   {
        local message "    " _n ///
                    "The single meta files are compiled. " _n ///
                    "    "   _n ///
                    "Processing... Please wait. "    _n ///
                    "    "
        set output proc
        display "`message'"
        set output error
```

```
        local datei = word("`svd'", 1)
        use "`pfad1d'/`datei'_meta`we'.dta", clear
        forvalues i = 2(1)`a'    {
            local datei = word("`svd'", `i')
            append using "`pfad1d'/`datei'_meta`we'.dta"
        }
    }


    /* modifikation of the variable undoc, if only missing codes und possibly value 0 are labeled  */
    replace undoc = -9 if undoc == nonmis & undoc != .
    replace undoc = -9 if undoc == (nonmis - nvalzero) & undoc != .

    /* modifikation of the  val#- und lab# variables: ersatz values (missing codes)  */
    forvalues x= 1(1)100 {
        quietly     {
            replace val`x' = -200 if isnumeric == 0
            replace val`x' = -100 if isnumeric == 1 & uniqvalpos > 100
            replace val`x' = -900 if isnumeric == 1 & uniqvalpos <= 100 & nonmis == 0
            replace lab`x' = "-200" if vallabset == "-2" & isnumeric == 1
            replace lab`x' = "-100" if isnumeric == 1 & vallabset != "-2" & uniqvalpos > 100
            replace lab`x' = "-100" if isnumeric == 0 & uniqvalpos > 100
            replace lab`x' = "-900" if isnumeric == 1 & vallabset != "-2" & uniqvalpos <= 100 & nonmis == 0
        }
    }
    /* generate consecutive number */
    capture drop lfn
    gen long lfn = _n
    lab var lfn "consecutive number"
    order lfn, first

    label data "`reslab'"
    save "`pfad1d'/`result'", replace

    /* erase single meta files    */

    local a = `nds'

    forvalues i = 1(1)`a' {
        local datei = word(`"`svd'"', `i')
        capture erase "`pfad1d'/`datei'_meta`we'.dta"
    }
}       /* closing bracket for:  compile single meta files */
```

```
/*##############################################################################################################
##############################                                      ##############################################
##############################      DISPLAY RUNTIME OF THE PROGRAM  ##############################################
##############################                                      ##############################################*/

quietly {
    timer off 1
    quietly timer list
    scalar `time'=(r(t1))
    local hrs = int(`time'/3600)
    scalar `time' = (r(t1)) - (`hrs' * 3600)
    local min = int(`time'/60)
    local sec = round(`time' - (`min' * 60), .01)


    if `u'strlen("`fnexist'")== 0{
        local fnexist "none"
    }

    if `u'strlen("`s14file'")== 0{
        local s14file "none"
    }
    if `u'strlen("`nobsfile'")   == 0{
        local nobsfile "none"
    }

    if `u'strlen("`u'") > 0  {
        local message "  " _n ///
        "Job completed.   " _n ///
        "Runtime of the program: `hrs' hrs `min' min `sec' sec  " _n ///
        "END at $S_TIME on $S_DATE " _n ///
        "  " _n ///
        " Name of the generated meta file: " _n ///
        "   `result' "  _n ///
        "  in directory `pfad1d' " _n ///
        "   "  _n ///
        " files that have not been processed: " _n ///
        "   "  _n ///
        "....- files not found: `fnexist' " _n ///
        "   "  _n ///
        "....- files with 0 observations: `nobsfile' " _n ///
        "  "
    }
```

```
    if `u'strlen("`u'") == 0 {
        local message "  " _n ///
        "Job completed.   " _n ///
        "Runtime of the program: `hrs' hrs `min' min `sec' sec  " _n ///
        "END at $S_TIME on $S_DATE " _n ///
        "  " _n ///
        " Name of the generated meta file: " _n ///
        "   `result' "  _n ///
        "  in directory `pfad1d' " _n ///
        "   "  _n ///
        " files that have not been processed: " _n ///
               "   "  _n ///
        "....- not format of the used Stata-version: `s14file'  " _n ///
                "   "  _n ///
        "....- files not found: `fnexist' " _n ///
        "   "  _n ///
        "....- files with 0 observations: `nobsfile' " _n ///
        "  "
    }
}       /*closing bracket for:  quietly */

set output proc
display " `message' "  _n ///
        " `message1'"
capture log close
```