Universelles Makro zur Zusammenfassung von (lückenlos) dokumentierten Zeitintervallen

Thomas G. Grobe
aQua-Institut
Maschmühlenweg 8-10
37073 Göttingen
Thomas.Grobe@aqua-institut.de

Zusammenfassung

Informationen zu Zeiträumen – beispielsweise zum Versicherungsstatus einer Person mit Von- und Bis-Datum – werden in Datentabellen zumeist in Zeilen untereinander abgelegt. Durch unterschiedliche Umstände können dabei Intervallfolgen ohne zeitliche Lücken und zugleich ohne Veränderungen hinsichtlich der anderweitig erfassten Merkmalsausprägungen resultieren. Entsprechende Zeilen der Datentabelle lassen sich dann ohne Informationsverlust zu einer Zeile zusammenfassen, was u.U. zu einer erheblichen Reduktion der Tabellengröße führt. Im Rahmen des Beitrags wird ein universell einsetzbares SAS-Makro vorgestellt, welches diese Zusammenfassung nach Spezifikation nur weniger Angaben durchführt. Die hierfür erforderliche SAS-Programmsyntax wird im Rahmen des Beitrags komplett erläutert. Genutzt werden dabei neben der Makro-Syntax auch PROC CONTENTS, INTO-Statements in PROC SQL und ein RETAIN-Statement. Trotz einer mit rund 50 Zeilen knapp bemessenen Syntax erlaubt das Makro eine Aufbereitung nahezu beliebiger Datentabellen. Damit lässt sich das Makro einfach nutzen und bei Bedarf auch leicht an weitere Erfordernisse anpassen. Der Autor hofft, mit dem Beitrag auch Anregungen für eigene SAS-Makro-Programmierungen zu liefern.

Schlüsselwörter: Zusammenfassung von Zeitintervallen, SAS Makro, PROC CONTENTS, INTO, RETAIN

1 Hintergrund und Ziele

Hintergrund des Beitrags bilden vielfältige Arbeiten mit Daten zu Versicherten. Diese enthalten häufig Informationen zu eindeutig abgegrenzten Zeiträumen, beispielsweise zum Wohnort einer Person mit Von- und Bis-Datum, welche in Datentabellen zumeist in separaten Zeilen (Beobachtungen) untereinander abgelegt werden. Durch unterschiedliche Umstände können dabei Intervallfolgen ohne zeitliche Lücken und zugleich ohne eine Veränderung hinsichtlich der anderweitig erfassten Merkmalsausprägungen resultieren. Dies ist beispielsweise regelmäßig dann der Fall, wenn Wohnortangaben bei der Datenbereitstellung aus Datenschutzgründen auf wenige Postleitzahlziffern oder Kreisangaben beschränkt werden. Viele Wohnortwechsel finden ausschließlich innerhalb einzelner Kreise statt – es verbleiben nach der reduzierten Bereitstellung entsprechend viele separat dokumentierte und zeitlich direkt aufeinander folgende Intervalle, die nur noch ein und dieselbe (trunkierte) Wohnortangabe beinhalten. Entsprech-

ende Zeilen der Datentabelle lassen sich dann grundsätzlich ohne Informationsverlust zu einer Zeile zusammenfassen, was unter Umständen zu einer erheblichen Reduktion der Tabellengröße führt. Dies kann die Weiterverarbeitung bei sehr großen Datenmengen merklich beschleunigen und führt insbesondere in Situationen, wo aufbereitete Intervallangaben mit Angaben zu anderen Intervallen verknüpft werden sollen, zu deutlich übersichtlicheren Daten.

1.1 Ziele und Gliederung des Beitrags

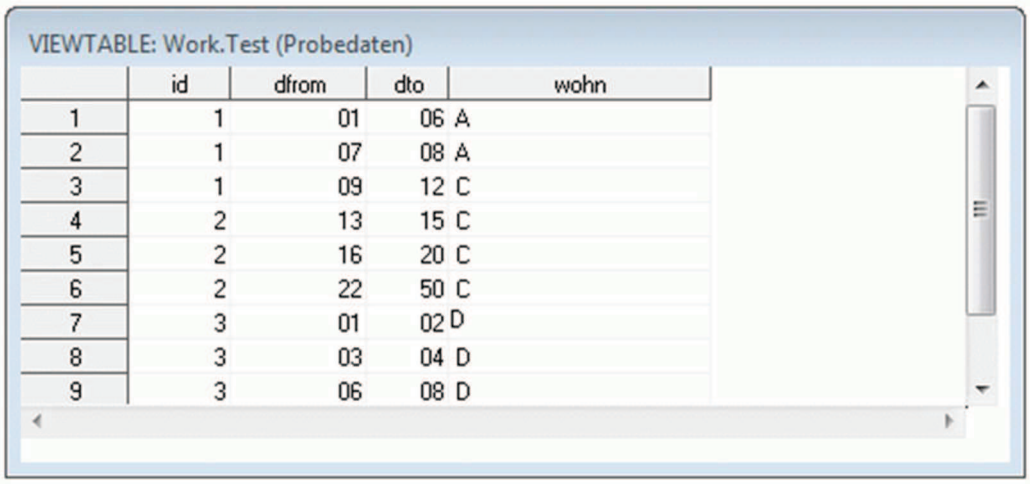
Im Rahmen des Beitrags wird ein universell einsetzbares SAS-Makro vorgestellt, welches die zuvor beschriebene Zusammenfassung nach Spezifikation nur weniger Angaben durchführt. Die hierfür erforderliche SAS-Programmsyntax soll im Rahmen des Beitrags komplett erläutert werden. Dabei folgt die Darstellung im Beitrag einem auch in der Praxis oftmals sinnvollen Vorgehen bei der Programmierung:

- Die Ausgangssituation sowie das erwünschte Resultat der Aufbereitung werden an einem sehr einfachen und leicht überschaubaren Datenbeispiel verdeutlicht.
- Genau für dieses spezielle Datenbeispiel wird ein lauffähiges SAS-Programm ohne Makroelemente entwickelt.
- Erst anschließend werden Anforderungen bei einer Verallgemeinerung erörtert und durch die Ergänzung des Programms um eine geeignete Syntax mit SAS-Makroelementen realisiert bzw. erfüllt.

2 Beispieldaten vor und nach geplanter Zusammenfassung

Die hier verwendeten Beispieldatei `test` umfassen lediglich 9 Beobachtungen bzw. Datenzeilen, die Angaben zu drei durch `ID` 1 bis 3 gekennzeichnete fiktive Personen enthalten. Ansonsten existieren Angaben zu Versicherungszeiten mit Von- und Bis-Angabe, die hier als einfache Zahlenangaben in den Variablen `dfrom` und `dto` abgelegt sind sowie in der Variablen `wohn` eine Angabe zum Wohnort in Form einer einfachen Buchstabenkennung. Tabelle 1 zeigt diese sortierten Beispieldaten vor der Zusammenfassung als Screenshot einer Datentabellenanzeige in SAS. Offensichtlich folgt bei `ID` 1 das zweite Intervall mit Start am Tag 07 lückenlos auf das erste Intervall mit Ende am Tag 06 – die Wohnortangabe bleibt dabei A. Die beiden Intervalle können insofern zu einem Intervall von Tag 01 bis Tag 08 zusammengefasst werden. Das 3. Intervall muss demgegenüber erhalten bleiben, da hier eine veränderte Wohnortangabe vorliegt. Auch bei `ID` 2 und 3 müssen die ersten beiden Intervalle zusammengefasst werden, das jeweils dritte Intervall muss auch hier erhalten bleiben, da zwischen dem zweiten und dritten Intervall bei beiden IDs eine zeitliche Lücke ohne Dokumentation eines Wohnortes besteht. Das gewünschte Endresultat der geplanten Zusammenfassung von Zeitintervallen in der Datei `test_out` ist in Tabelle 2 dargestellt. Ergänzend ist in dieser aufbereiteten Datentabelle eine Variable `Anz_Int` enthalten, welche zu den aufbereiteten Intervallen die Anzahl der ursprünglich in den Ausgangsdaten enthaltenen Intervalle angibt.

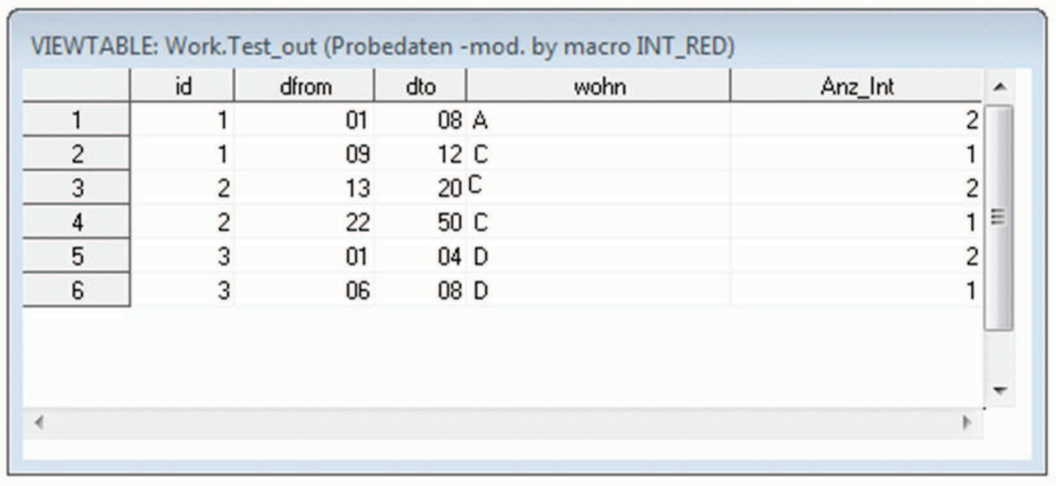
Tabelle 1: Beispieldaten vor Zusammenfassung (work.test)



VIEWTABLE: Work.Test (Probedaten)

	id	dfrom	dto	wohn
1	1	01	06	A
2	1	07	08	A
3	1	09	12	C
4	2	13	15	C
5	2	16	20	C
6	2	22	50	C
7	3	01	02	D
8	3	03	04	D
9	3	06	08	D

Tabelle 2: Beispieldaten nach Zusammenfassung (work.test_out)



VIEWTABLE: Work.Test_out (Probedaten -mod. by macro INT_RED)

	id	dfrom	dto	wohn	Anz_Int
1	1	01	08	A	2
2	1	09	12	C	1
3	2	13	20	C	2
4	2	22	50	C	1
5	3	01	04	D	2
6	3	06	08	D	1

3 SAS-Syntax zur Aufbereitung der Beispieldaten

Die vollständige Syntax zur Aufbereitung der beschriebenen Beispieldaten ist im nachfolgenden Abschnitt unter „Programmsyntax 1“ – hier inklusive einer vorangestellten Zeilennummerierung – wiedergegeben. Alle erforderlichen Bearbeitungen der Daten erfolgen innerhalb eines einzelnen Datenschritts (von Zeile 02 bis Zeile 30). Die nachfolgenden Erläuterungen orientieren sich sinngemäß an der Verarbeitungsreihenfolge in diesem Datenschritt.

Die Verarbeitung der ursprünglichen Datei `test` beginnt im Datenschritt mit dem Einlesen der im **set**-Statement angegebenen Datentabelle (Zeile 05). Durch ein eher selten verwendetes **end=** wird dabei eine nur temporär innerhalb des Datenschritts verfügbare Indikatorvariable mit dem hier frei gewählten Namen `last` spezifiziert, welche zur Identifikation der letzten Zeile der gesamten Datentabelle genutzt werden kann und später noch benötigt wird. Diese Indikatorvariable hat regulär den Wert 0, nur in der letzten Datenzeile der Tabelle ist ihr der Wert 1 zugewiesen.

Um Werte aus unterschiedlichen Zeilen der Datentabelle innerhalb eines Datenschritts vergleichen zu können, lassen sich in SAS Variablen in einem RETAIN-Statement spezifizieren, welche anschließend die einmal in einer bestimmten Zeile zugewiesenen Werte bis auf weiteres auch in den nachfolgenden Zeilen behalten. Das vorliegende Programm nutzt diese Möglichkeit umfänglich – für alle Variablen der Ursprungsdatei werden **retain**-Varianten angelegt (Zeile 06), wobei den Variablennamen aus der Ursprungsdatei hierbei ein `R_` vorangestellt ist. Letztendlich sollen nachfolgend ausschließlich die in diesen neuen Variablen enthaltenen Werte in die finale Datei herausgeschrieben werden.

Beim Einlesen der ersten Zeile der Ursprungsdatei (`if _N_=1`) werden die neuen Variablen erstmals mit den dort vorgefundenen Werten befüllt (Zeile 08) – ohne weitere Modifikationen würden die Werte dann unverändert bis zur letzten Zeile der Datentabelle erhalten bleiben. Bei `_N_` handelt es sich um eine im Datenschritt stets ansprechbare interne Variable, welche die aktuelle Zeilennummer der Datentabelle enthält. Die eigentliche Bearbeitung der Daten erfolgt in der Syntax erst ab Zeile 12, wenn die in retain-Varianten enthaltenen Variablenwerte der ersten Zeile ab der zweiten Datenzeile der Tabelle mit den dann jeweils aktuellen Werten verglichen werden können (`if _N_ gt 1`). Zunächst wird standardmäßig der Wert der gleichfalls im retain-Statement aufgeführten Zählvariable `Anz_Int` um 1 erhöht, welche die Zahl der ursprünglichen Datenzeilen auch bei einer ggf. erfolgten Zusammenfassung in der finalen Datentabelle wiedergeben soll.

Zeigt sich, dass in einer aktuell betrachteten Tabellenzeile identische Werte wie in retain-Varianten der Variablen mit Befüllung aus vorausgehenden Zeilen enthalten sind (`if (R_id=id and R_wohn=wohn …`) und endet zudem das vorausgehende bzw. in der retain-Variablen dokumentierte Intervall exakt am Tag vor Beginn des aktuellen Intervalls (`dfrom-R_dto=1` vgl. Zeile 13) – besteht also eine lückenlose Dokumentation ohne Veränderung von Werten – wird lediglich der Bis-Datumswert in der retain-Variante durch den Wert aus der aktuellen Datenzeile ersetzt (`R_dto=dto;` vgl. Zeile 15).

In allen anderen Fällen (**`else do;`** Zeile 17) beginnt mit den in der aktuellen Datenzeile in den Original-Variablen dokumentierten Werten ein eigenständiges Intervall. Die Daten zu einem demnach abgeschlossenen vorausgehenden Intervall in Variablen der retain-Variante können damit in die finale Datentabelle herausgeschrieben werden (**`output`** `test_out;` Zeile 19). Anschließend werden die Werte der retain-Variablen durch Werte zum neuen, separat zu dokumentierendem Intervall aus der aktuellen Zeile ersetzt (Zeile 21). Zudem wird die Zählvariable auf 0 zurückgesetzt, womit die Überprüfung in der nächsten Tabellenzeile fortgesetzt werden kann. Lediglich die allerletzte Zeile der Datentabelle (identifiziert in Zeile 26 durch **`if`** `last`) bedarf einer gesonderten Behandlung, da hier der sonst regulär durchgeführte Abgleich mit einer noch nachfolgenden Beobachtungen nicht stattfinden kann. Die letzte Datenzeile wird nach Addition von 1 zum vorausgehenden Wert der Zählvariablen grundsätzlich in die finale Da-

tentabelle geschrieben. Durch die Optionen **keep=** und **rename=** im einleitenden data-Statement in Zeile 02 und 03 des Programms wird schließlich gewährleistet, dass die aufbereitete Datei, neben der ergänzten Zählvariablen, lediglich die ursprünglichen Variablen mit ihren ursprünglichen Namen enthält.

Programmsyntax 1: Komplette SAS-Syntax zur Aufbereitung der Beispieldaten

```
01 * final file to be output (based on retained values);
02 data test_out (keep=R_id   R_dfrom   R_dto   R_wohn   Anz_Int   ①
03         rename=(R_id=id R_dfrom=dfrom R_dto=dto R_wohn=wohn) ); ②
04 * set original file, create temp. indicator for last observation;
05 set test end=last;
06 retain   R_id   R_dfrom   R_dto   R_wohn   Anz_Int;
07 if _N_=1 then do;      * first fill of retain variables ;
08     R_id=id;  R_dfrom=dfrom;  R_dto=dto;  R_wohn=wohn;          ③
09 end;
10 * begin with second observation to compare previous values;
11 if _N_ gt 1 then do;
12     Anz_Int=sum(Anz_Int,1);   * counting of intervals ;
13     if (R_id=id and R_wohn=wohn) and (dfrom-R_dto=1) then do;   ④
14 * if nothing changed and interval is continued only update date;
15        R_dto=dto;
16     end;
17     else do;         * otherwise ...........;
18        * write previously retained values to output file;
19        output test_out;
20        * afterwards update retain variables;
21        R_id=id;  R_dfrom=dfrom;  R_dto=dto;  R_wohn=wohn;
22        Anz_Int=0;   * Reset count of intervals;
23     end;
24 end;
25 * last obs. - ret. values cannot be compared with later val.;
26 if last then do;
27     Anz_Int=sum(Anz_Int,1);
28 * write very last updated retain values to output file;
29     output test_out;
30 end; run;
```

4 Verallgemeinerung der Aufbereitung

Die verallgemeinerte Variante der Programmsyntax 1 findet sich im nachfolgenden Abschnitt unter „Programmsyntax 2“. Wie in den Beispieldaten implizit vorgegeben, setzt auch die verallgemeinerte Variante voraus, dass die zu verarbeitende Datei bereits zuvor und abhängig vom Inhalt angemessen sortiert ist (z.B. aufsteigend nach IDs und Zeitintervallen). Die SAS-Makrosprache erlaubt in der Regel grundsätzlich und allgemein ein recht einfaches Vorgehen bei der Verallgemeinerung von SAS-Programmen: Alle variablen Programmteile werden durch Makro-Variablen ersetzt, die dann bei Aufruf eines Makros, oder alternativ beispielsweise zunächst auch durch ein %LET-Statement, mit

nahezu beliebigen Werten oder auch ganzen Syntax-Elementen gefüllt werden können. Sehr simpel lassen sich so die Benennungen der Original-Datentabelle `test` und die gewünschte Benennung der finalen Datentabelle `test_out` sowie auch die speziell darin verwendeten Bezeichnungen für Von- und Bis-Datumsvariablen `dfrom` bzw. `dto` sowie eine gewünschte Bezeichnung der Zählvariable `Anz_Int` durch Makrovariablen mit den hier willkürlich gewählten Benennungen `&DATAIN.`, `&DATAOUT.`, `&DFROMV.`, `&DTOV.` sowie `&COUNTV.` ersetzen. Die Großschreibung bei der Benennung von Makrovariablen ist nicht obligat, führt jedoch zu einer besseren Differenzierbarkeit in der Syntax. Auch die nachgestellten Punkte bei den Makrovariablen sind nur in bestimmten Situationen (bei einer beabsichtigten lückenlosen Verkettung von Werten aus Makrovariablen) erforderlich, schaden jedoch in keinem Fall und können daher vorzugsweise durchgängig verwendet werden. Wird beispielsweise mit der Syntax `%LET DATAIN=test01;` die Makro-Variable mit dem Namen DATAIN gefüllt, wird in der nachfolgenden Syntax im Programmablauf, auch außerhalb von SAS-Makros in einem engeren Sinn, an Stellen mit einer Nennung der Makro-Variablen durch `&DATAIN.` jeweils der String `test01` gelesen und so in diesem Beispiel ggf. eine entsprechend benannte Eingabedatei verarbeitet.

Mit den vier (bzw. inklusive optionaler Zählvariablen insgesamt fünf) zuvor genannten Makrovariablen sind bereits alle obligaten Vorgaben für die Aufbereitung der Datentabelle festgelegt. Die für die Verarbeitung darüber hinaus erforderlichen Informationen zu den – neben den zwei explizit spezifizierten Datumsvariablen – noch in der Ursprungsdatei enthaltenen Variablen lassen sich in SAS mit PROC CONTENTS auslesen und in eine ergänzende Datentabelle schreiben. Anschließend müssen daraus noch die in Programmsyntax 1 grau unterlegten Passagen der SAS-Syntax erzeugt werden. Ein genauerer Blick auf die Programmsyntax 1 zeigt, dass insgesamt vier unterschiedliche Syntax-Elemente benötigt werden, welche in der Programmsyntax 1 mit ① bis ④ gekennzeichnet sind.

- Bei ① wird eine durch Leerzeichen separierte Liste aller Variablen der Ursprungsdatei benötigt, wobei hier alle Variablennamen durch Voranstellung von `R_` im Sinne einer schematischen Benennung von retain-Variablen modifiziert werden sollen. Diese Liste wird identisch in Zeile 02 und Zeile 05 der Syntax verwendet.
- Bei ② wird Syntax zur Umbenennung aller retain-Variablen in ihre jeweils ursprünglichen Bezeichnungen ohne ein vorangestelltes `R_` benötigt.
- Bei ③ werden alle retain-Variablen mit den Werten aus den ursprünglichen Variablen befüllt. Die Syntax wird identisch in Zeile 08 und Zeile 21 benötigt.
- Bei ④ wird die Übereinstimmung der Werte von retain-Variablen mit denen aus ursprünglichen Variablen überprüft, wobei die Überprüfungen mit `and` verknüpft sind (da bei einer Zusammenfassung alle Variablenwerte unverändert sein sollen) und dabei die zwei Datumsvariablen vom Vergleich ausgenommen werden.

Die erforderlichen vier Syntax-Elemente folgen jeweils einem einfachen Schema und lassen sich insofern leicht automatisiert aus einer vorhandenen Variablenliste ableiten, wie nachfolgend noch beschrieben wird. Zugleich wird schon bei Betrachtung der Syntax zu den übersichtlichen Beispieldaten klar, dass die Syntax-Elemente bei Datentabellen mit vielen Variablen eine erhebliche Länge aufweisen können und eine manuelle Erstellung entsprechend recht mühsam wäre, was den Nutzen einer Automatisierung zusätzlich unterstreicht. Im nachfolgenden Abschnitt unter „Programmsyntax 2“ findet sich die komplette Syntax, um auch die vier fehlenden Syntax-Elemente automatisiert zu ergänzen. Unter „Programmsyntax 3“ wird anschließend die verallgemeinerte Syntax zur Ausführung des Datenschritts dargestellt.

Mit **PROC CONTENTS** in Programmsyntax 2 (Zeile 02) wird in einem ersten Schritt eine Datei `x_cont` erstellt, welche umfängliche Informationen zu allen Variablen in der mit `&DATAIN.` spezifizierten Datei (jeweils in einer Zeile) enthält. Die Option `noprint` unterdrückt eine Ausgabe dieser Informationen im SAS-Output. Im nachfolgenden Datenschritt (ab Zeile 04) werden aus der Datei `x_cont` hier lediglich die zwei in **keep=** spezifizierten Variablen `name` und `varnum` weiterverwendet – erstere enthält die Variablennamen, `varnum` eine Nummerierung dieser Variablen entsprechend der Spaltenfolge in der mit PROC CONTENTS beschriebenen Datentabelle. In Zeile 05 werden mit der Funktion **lowcase** Varianten der Variablennamen in konsequenter Kleinschreibung erzeugt, welche in der neuen Variable `name_l` abgelegt werden. Nur durch die zuvor umgesetzte Kleinschreibung aller Variablennamen ist gewährleistet, dass diese auch bei unterschiedlichen Schreibweisen zuverlässig gefunden werden. In Zeile 07 wird zu den ursprünglichen Namen der Variablen in `allvret` eine retain-Variante mit jeweils vorangestelltem R_ abgelegt, in Zeile 08 werden Syntaxelemente nach dem Muster „R_Variablenname=Variablenname“ erzeugt. Die Funktion **compress** stellt in der verwendeten Form sicher, dass in den Texten keine Leerzeichen enthalten sind, die doppelten Hochstriche `||` dienen in SAS allgemein der Verkettung von Textzeichenfolgen.

Die Syntax `call symput("MAKROVARIABLENNAME",Wert);` (vgl. Zeile 10 und 11) eignet sich, um einer bestehenden oder neuen Makrovariablen innerhalb eines Datenschritts einen Wert zuzuweisen, welcher nach Abschluss des Datenschritts abgefragt werden kann. In der hier dargestellten Form wird die Makrovariable mehrfach und dabei final beim Lesen der letzten Zeile der Datentabelle gefüllt. Die Syntax wird hier ausschließlich verwendet, um die Bezeichnungen der Von- und Bis-Datumsvariablen auch in den beiden Makrovariablen konsequent in einer Schreibweise mit Kleinbuchstaben abzulegen. Mit **proc sort** wird die Datei abschließend entsprechend der in `varnum` festgehaltenen ursprünglichen Reihenfolge von Variablen sortiert. Die in der Datei `x_cont` zwischenzeitlich enthaltenen Syntaxelemente zu den einzelnen Variablen werden anschließend unter Rückgriff auf Möglichkeiten in **PROC SQL** verkettet in vier einzelne Makrovariablen eingelesen (ab Zeile 14). Mit der Syntax **`select`** `allvret` **`into`** `:ALLVRET` **`separated by " "`** werden beispielsweise die zu der Variable

`allvret` in unterschiedlichen Tabellenzeilen enthaltenen Variablennamen (jeweils in der retain-Variante) nacheinander und getrennt durch ein Leerzeichen in die Makrovariable `&ALLVRET.` eingelesen (Zeile 15). Die Makrovariable `&ALLVRET.` würde bei den Beispieldaten anschließend die unter ① in der Syntax benötigte Textfolge `R_id R_dfrom R_dto R_wohn` enthalten. Durch Angabe anderer Trennzeichen(-folgen) können auch andersartige Syntaxelemente entstehen (vgl. Zeile 16 bis 18). In der Textfolge zur Variante ④ (für den Vergleich von Variablenwerten) sollen die Datumsvariablen nicht berücksichtigt werden, was hier durch die Bedingung `where name_l not in("&DFROMV.","&DTOV.")` erreicht wird – durch **not in()** werden Datenzeilen mit Angaben zu genau den beiden Datumsvariablen ausgespart. Soll im Programm nachfolgend gewöhnliche SAS-Syntax verwendet werden, muss PROC SQL mit **quit;** beendet werden. Nach dem Füllen der Makrovariablen kann schließlich die primär mit PROC CONTENTS im temporären WORK-Verzeichnis erstellte und jetzt nicht mehr benötigte Datendatei `x_cont` mit PROC DATASETS gelöscht werden (Zeile 21), was auch hier mit `quit;` abgeschlossen werden muss.

Im Abschnitt unter „Programmsyntax 3“ findet sich die für eine allgemeine Aufbereitung von Datentabellen ausschließlich durch eine Verwendung der zuvor beschriebenen Makrovariablen angepasste Syntax des bereits unter Programmsyntax 1 dargestellten Datenschritts. Auf eine umfassendere Kommentierung der nicht grundsätzlich veränderten Syntax wurde an dieser Stelle verzichtet – um Vergleiche der Beispiel-bezogenen sowie der verallgemeinerten Syntax zu erleichtern, wurde die Zeilennummerierung der ursprünglichen Syntax wie in Programmsyntax 1 auch nach einem Wegfall von Programm- bzw. Kommentarzeilen beibehalten. Um aus den beiden Syntaxabschnitten unter Programmsyntax 2 und Programmsyntax 3 ein SAS-Makro im engeren Sinne zu machen, bedarf es lediglich noch einer entsprechenden Klammer durch das Start- und End-Element eines Makros:

```
%MACRO INT_RED(DATAIN,DATAOUT,DFROMV,DTOV,COUNTV);
  … Programmsyntax 2 …
  … Programmsyntax 3 …
%MEND INT_RED;
```

Mit `%MACRO INT_RED` wird dabei angegeben, dass hier die Definition eines Makros mit dem individuell gewählten Namen „INT_RED“ beginnt. In Klammern werden direkt nach dem Namen ggf. verwendete Makrovariablen getrennt durch Kommata gelistet, denen bei Aufruf des Makros ein Wert zugewiesen werden soll. Mit `%MEND INT_RED;` wird angegeben, dass die Spezifikation des Makros an dieser Stelle beendet ist. Um ein Makro innerhalb einer SAS-Session nutzen zu können, muss die entsprechende Syntax als Programmabschnitt mindesten einmalig – z.B. mit Aufruf des Befehls „run“ – gelaufen bzw. kompiliert worden sein. Erst anschließend kann das Makro mit folgender Syntax beliebig oft aufgerufen werden (hier mit Angaben für die Beispieldatei):

```
%INT_RED(test,test_out,dfrom,dto,Anz_Int);
```

Programmsyntax 2: Aufbereitung von Informationen zu Ursprungsdaten

```
01 * reading variable names from input file;
02 proc contents data=&DATAIN. out=x_cont noprint;
03 * preparing syntax based on variable names;
04 data x_cont; set x_cont (keep=varnum name);
05 name_l=lowcase(name);       * names - lowcase only;
06 * pre-prepare syntax to rename and fill retain variables etc.;
07 allvret=compress("R_"||name); * retain-variables;
08 renamevret=compress(allvret||"="||name);
09 * generating lowcase variable names in MACRO-Variables;
10 call symput("DFROMV",lowcase("&DFROMV."));
11 call symput("DTOV",lowcase("&DTOV."));
12 proc sort; by varnum; run;    * sort to keep var. in orig. order;
13           * reading syntax from all rows into 4 MACRO-variables;
14 proc sql noprint;
15 select allvret into :ALLVRET separated by " " from x_cont;       ①
16 select renamevret into :RENAMEVRET separated by " " from x_cont;②
17 select renamevret into :RETUPDATE separated by "; " from x_cont;③
18 select renamevret into :NOCHANGEVAL separated by " and " from
19           x_cont where name_l not in("&DFROMV.","&DTOV.");       ④
20 quit;
21 proc datasets lib=work; delete x_cont; run; quit;
```

Programmsyntax 3: Nachfolgende Aufbereitung beliebiger Datentabellen

```
2 data &DATAOUT. (keep=&ALLVRET. &COUNTV. rename=(&RENAMEVRET.));①②
05 set &DATAIN. end=last;       * original file;
06 retain &ALLVRET. &COUNTV.;                                       ①
07 if _N_=1 then do;             * first fill of retain variables;
08    &RETUPDATE. ;                                                 ③
09 end;
11 if _N_ gt 1 then do;
12    &COUNTV.=sum(&COUNTV.,1);
13    if (&NOCHANGEVAL.) and (&DFROMV. - R_&DTOV. = 1) then do;     ④
15       R_&DTOV. = &DTOV.; * only update to-date variable;
16    end;
17    else do;  * otherwise ... ;
19       output &DATAOUT.; * write retained val. to output file;
21       &RETUPDATE. ;                                             ③
22       &COUNTV.=0;
23    end;
24 end;
26 if last then do; * very last obs.;
27    &COUNTV.=sum(&COUNTV.,1);
29    output &DATAOUT.;
30 end; run;
```

5 Diskussion und Ergänzungen

Das dargestellte Makro wurde bereits häufig erfolgreich auch zu Aufbereitung größerer Datentabellen eingesetzt. Bedingt durch die zwischenzeitliche Ablage von Informationen aus allen Variablen in einer jeweils zweiten Variablen könnte im Rahmen der Aufbereitung temporär mit der Entstehung einer größeren Datei gerechnet werden. Dies scheint jedoch nach Beobachtungen von temporären Dateien bei der Aufbereitung großer Datenmengen keineswegs der Fall zu sein – offensichtlich werden die Zeilen der Originaldaten sequenziell abgearbeitet. Selbst bei einer Zusammenfassung von 100 Millionen lückenlos dokumentierten Intervallen zu einer einzigen Datenzeile entsteht im SAS-Work-Verzeichnis temporär nie eine Datei, die größer als die resultierende Gesamtzusammenfassung der Daten mit nur einer Datenzeile ist.

Bei genauerer Betrachtung des dargestellten Programmcodes und der damit aufbereiteten Daten ergeben sich Hinweise auf eine Limitation sowie einige ergänzende Wünsche. Eine offensichtliche Limitation des Makros resultiert daraus, dass ursprüngliche Variablenbezeichnungen zwischenzeitlich durch Voranstellen der Zeichenfolge `R_` modifiziert werden. Wurde bei der ursprünglichen Bezeichnung der Variablen bereits die in SAS maximal zulässige Länge von 32 Zeichen ausgenutzt, resultiert durch die veränderte Bezeichnung eine unzulässige Zeichenlänge und damit ein Scheitern des Makro-Ablaufs. Die Vermeidung dieses Problems ist jedoch mit einer kleineren Anpassung des Programmcodes möglich, indem eindeutige und stets kurze Bezeichnungen für die nur zwischenzeitlich benötigten retain-Varianten der Variablen aus der Verkettung des Präfix „R_“ und Werten der Variable `varnum` hergeleitet werden, welche alle Variablen mit einer eindeutigen Nummerierung versehen (vgl. Zeile 07 in Programmsyntax 2: `allvret=compress("R_"||varnum);`). Anschließend muss noch eine ergänzende Makrovariable `&DTOVR.` gebildet werden, welche die entsprechend verkürzte Bezeichnung der retain-Variante des Bis-Datums für die spätere Überprüfung der Intervallfolge enthält. Zu diesem Zweck kann die nachfolgende Syntax in Programmsyntax 2 nach Zeile 11 eingefügt werden:

```
if name_l=lowcase("&DTOV.") then
   call symput("DTOVR",compress( "R_"||varnum));.
```

Eher kosmetischer Natur ist der Wunsch, die Zählvariable zur Anzahl zusammengefasster Intervalle nur optional in den aufbereiteten Daten darzustellen. Dies lässt sich dadurch realisieren, dass alle Stellen mit Modifikation der Zählvariablen durch die folgende Syntax eingeschlossen werden: `%IF &COUNTV. ne %THEN %DO; … %END;` Bleibt bei Aufruf des Makros die Angabe zum Namen einer Zählvariablen leer, wird diese dann erst gar nicht angelegt. Hinzuweisen ist im Kontext der Verwendung von bestimmter Makro-Syntax wie `%IF` darauf, dass diese – im Gegensatz zur gesamten übrigen bislang dargestellten Syntax – grundsätzlich nur innerhalb eines Makro-Aufrufs verwendet werden kann. Damit lässt sich dann beispielsweise der einzelne Datenschritt mit der Aufbereitung von Daten auch nach zuvor erfolgter Festlegung der Inhalte von Makrovariablen nicht mehr separat ausführen.

Als weitere Option wurde in einer erweiterten Variante des bislang vorgestellten Makros zusätzlich die Möglichkeit geschaffen, auch Variablen (mehrere ggf. getrennt durch Leerzeichen) als letzte Angabe beim Makro-Aufruf in einer Makrovariable `&IGNORE.` zu benennen, welche nicht bei einer Zusammenfassung von Intervallen auf mögliche Änderungen überprüft werden sollen (womit eine Intervallzusammenfassung dann trotz ggf. vorhandener Unterschiede hinsichtlich dieser Variablen erfolgt). Schließlich wurde durch weitere kleine Erweiterungen dafür gesorgt, dass ein ggf. vorhandenes Label der ursprünglichen Datentabelle sowie alle Format- und Label-Angaben zu den Variablen der Ursprungsdatei auch in der aufbereiteten Datei erhalten bleiben (welche beim Umkopieren von Werten in neue retain-Varianten der Variablen verloren gehen). Die gesamte Syntax des erweiterten Makros inklusive einer Syntax zur Erstellung der hier verwendeten Beispieldaten ist im Anhang A enthalten, dessen Inhalt für eigene Versuche einfach aus der PDF-Version des vorliegenden Beitrags kopiert werden kann. Insbesondere zur eigenen Programmierung möchte der Autor die Leser des vorliegenden und leider teilweise sehr trockenen und wenig spektakulären Beitrags ermuntern!

Anhang A Vollständige Syntax inklusive Beispieldaten

Anmerkung: Die nachfolgende Listung der erweiterten SAS-Syntax soll primär ein einfaches Kopieren aus der PDF-Version des vorliegenden Beitrages erleichtern. Die eingeschränkte Lesbarkeit in der Print-Fassung bittet der Autor zu entschuldigen.

```
/* Macro to reduce intervals if zero days between and otherwise unchanged values ==================
* all intervals must be sorted and should not overlap !!!!!                      ;
* v004 21.01.2018 by Thomas G. Grobe. contact: thomas.grobe@aqua-institut.de    ;

* data - only used to test Macro ;
data test; format id 3.0 dfrom dto z2. wohn $100.;
input id dfrom dto wohn; cards;
1 01 06 A
1 07 08 A
1 09 12 C
2 13 15 C
2 16 20 C
2 22 50 C
3 01 02 D
3 03 04 D
3 06 08 D
; run;
data test (label="Probedaten"); set test; * only to add variable labels ;
label id="ID-Variable - hier mit längerem Label" dfrom="Von-Datum";
run;

* only used to test elements of the macro separately ;
%LET DATAIN=test;       * input data set ;
%LET DATAOUT=test_out;  * output data set ;
%LET DFROMV=dfrom;      * variable containing from-date ;
%LET DTOV=dto;          * variable containing to-date ;
%LET COUNTV=Anz_Int;    * optional: new var. to count orig. intervals - not generated if blank ;
%LET IGNORE= ;          * optional: val. of this var. will not be compared to previous values ;
                        * (val. of var. will only be updated if other variables are updated) ;

* regular call of the macro after compilation - here for test data <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<;

%INT_RED(test,test_out,dfrom,dto,Anz_Int,);

==========================================================================================*/
```

```
%MACRO INT_RED(DATAIN,DATAOUT,DFROMV,DTOV,COUNTV,IGNORE);

proc contents data=&DATAIN. out=x_cont noprint;  * reading variable names from input file ;
data x_cont; set x_cont (keep=varnum name MEMLABEL FORMAT FORMATL FORMATD LABEL);
    name_l=lowcase(name);     * start preparing syntax based on variable names ;
    allvret=compress("R_"||varnum);            * new names for var. used in retain function ;
    renamevret=compress(allvret||"="||name); * syntax to fill retain variables etc. ;
    format ignore $1000.;
    if "&IGNORE." ne " " then do; * generating variable list to be ignored, for in() function ;
        ignore=tranwrd(compbl(lowcase("&IGNORE.")),' ','","') ;     * lowcase ! ;
        call symput("IGNORELIST",compress(',"'||ignore||'"'));
    end;
    else do;  call symput("IGNORELIST"," ");  end;
    call symput("DFROMV",lowcase("&DFROMV.")); * lowcase names also in macro-var. ;
    call symput("DTOV",lowcase("&DTOV."));

    if name_l=lowcase("&DTOV.") then call symput("DTOVR",compress("R_"||varnum)); * retain dat.to;
    call symput("MEMLABEL",trim(MEMLABEL));                              * orig. data set label ;
    if FORMATL gt 0 then formatret=compress(allvret)||" "||compress(FORMAT||FORMATL||"."||FORMATD);
    if length(LABEL) gt 0 then labelret=compress(allvret)||"='"||trim(LABEL)||"'";
proc sort; by varnum; run;                    * sort to keep variables in orig. order ;

proc sql noprint;                             * reading syntax into MACRO-variables ;
  select allvret into :ALLVRET separated by " " from x_cont;    * retain var. list sep. by blanks ;
  select renamevret into :RENAMEVRET separated by " " from x_cont;    * ret.var. to normal syntax ;
  select renamevret into :RETUPDATE separated by "; " from x_cont;    * update ret. var. syntax ;
  select renamevret into :NOCHANGEVAL separated by " and " from x_cont
    where name_l not in("&DFROMV.","&DTOV." &IGNORELIST.);  * list of var. not to compare! ;

  select formatret into :FORMATRET separated by " " from x_cont; * complete ret.var. format list ;
  select labelret into :LABELRET separated by " " from x_cont;   * complete ret.var. label list ;
quit;
proc datasets lib=work; delete x_cont; run; quit; * delete temorary files ;

data &DATAOUT. (keep=&ALLVRET. &COUNTV. rename=(&RENAMEVRET.)
                      label="&MEMLABEL. -mod. by macro INT_RED");
set &DATAIN. end=last;                * set original file ;
retain &ALLVRET. &COUNTV.;
format &FORMATRET. &COUNTV. 8.;
label &LABELRET. &COUNTV.="Anzahl zusammengefasster Beobachtungen";
if _N_=1 then do;                      * first fill of retain variables ;
    &RETUPDATE. ;
end;
if _N_ gt 1 then do;           * starting from second observation (first to compare with previous);
    %IF &COUNTV. ne   %THEN %DO;
        &COUNTV.=sum(&COUNTV.,1);     * counting orig. intervals ;
    %END;
    if (&NOCHANGEVAL.) and (&DFROMV.- &DTOVR.=1) then do;  * if no val. changed and int. continued ;
        &DTOVR. = &DTOV.;                                  * then only update to-date variable ;
    end;
    else do;                         * otherwise ... ;
        output &DATAOUT.;            * write retained values to output file ;
        &RETUPDATE. ;                * update retained values ;
        %IF &COUNTV. ne   %THEN %DO;    * reset count ;
            &COUNTV.=0;
        %END;
    end;
end;
if last then do;             * very last obs.-retained val. cannot be compared with later observation ;
    %IF &COUNTV. ne   %THEN %DO;
        &COUNTV.=sum(&COUNTV.,1);      * counting orig. intervals ;
    %END;
    output &DATAOUT.; * write last observation to output file ;
end; run;

%MEND INT_RED;
```

Viel Spaß beim Programmieren und Dank für das Interesse am Beitrag!