Tipps & Tricks

Sabine Erbslöh, Christina Gelhorn
Accovion GmbH
Helfmann-Park 10
65760 Eschborn
sabine.erbsloeh@accovion.com
christina.gelhorn@accovion.com

Zusammenfassung

Oft sind es ganz kleine Dinge, über die man im Programmieralltag stolpert, für deren ausführliche Bearbeitung man aber meist keine Zeit hat. Hier werden einige Tipps und Tricks zur effektiven Arbeit mit SAS vorgestellt.

Schlüsselwörter: Konfidenzintervalle, PROC REPORT, REPORT-Prozedur, Datumsfelder, LAG-Funktion, MERGE, IN Operator, Windows Editor, Klonen, DO-Schleifen, SELECT, CALL MISSING, SMALLEST, LARGEST, STRIP, COMPBL, SUBSTRN

1 Problem aus der Statistik: Berechnung von Konfidenzintervallen für 0% mit PROC FREQ

Es gibt manchmal die Situation, in der die Bestimmung eines Konfidenzintervalls zu einer Rate von 0% gewünscht ist. PROC FREQ gibt im Normalfall nur Konfidenzintervalle für beobachtete Ereignisse aus. Nachfolgend ist ein Weg beschrieben wie man vorgehen kann, falls in einer Stichprobe das eigentlich interessierende Ereignis nicht auftritt.

Beispiel: Man interessiert sich für die Wahrscheinlichkeit des Auftretens einer allergischen Reaktion auf das verabreichte Medikament. Diese Reaktion ist bei keinem der Patienten beobachtet worden. Der Inputdatensatz sieht folgendermaßen aus:

Obs	Patient	1=allergische Reaktion, 2=keine allergische Reaktion
1	1	2
2	2	2
3	3	2
4	4	2
5	5	2
6	6	2
7	7	2

Obs	Patient	1=allergische Reaktion, 2=keine allergische Reaktion
8	8	2
9	9	2
10	10	2

PROC FREQ liefert im Normalfall nur Schätzer für die Eintrittswahrscheinlichkeit des Ereignisses „keine allergische Reaktion“. Ist man aber an einem Konfidenzintervall für die Eintrittswahrscheinlichkeit des Ereignisses „allergische Reaktion“ interessiert, so lässt sich dies mit PROC FREQ nur über Umwege berechnen:

Dazu wird zuerst ein Format definiert, das alle möglichen Ausprägungen der Variable enthält:

```
proc format;
   value statusf
      1 = "allergische Reaktion"
      2 = "keine allergische Reaktion";
run;
```

Die absoluten Häufigkeiten aller so festgelegten Kategorien werden mit PROC TABULATE unter Verwendung der Optionen PRELOADFMT und PRINTMISS ausgezählt.

```
proc tabulate data = status out = anz;
   class  ereignis     / preloadfmt;
   table  ereignis * n / printmiss;
   format ereignis statusf.;
run;
```

1=allergische Reaktion, 2=keine allergische Reaktion	
allergische Reaktion	keine allergische Reaktion
N	N
	10

Anschließend werden die ‚Missings’ durch Nullen ersetzt:

```
data anz;
   set anz;
   if n eq . then n = 0;
run;
```

Das so erhaltene Ergebnis wird der PROC FREQ unter Verwendung der WEIGHT Anweisung übergeben. Als Gewichte werden dabei die Anzahlen der Beobachtungen in

den jeweiligen Kategorien verwendet. Damit die Null für die Gewichtung berücksichtigt wird, muss zusätzlich die Option ZEROS angegeben werden.

```
proc freq data = anz order = data;
   weight n       / zeros   ;
   table ereignis / binomial;
run;
```

So wird das Konfidenzintervall für die Eintrittswahrscheinlichkeit des Ereignisses „allergische Reaktion“ berechnet.

Binomial Proportion for EREIGNIS = allergische Reaktion	
Proportion	0.0000
ASE	0.0000
95% Lower Conf Limit	0.0000
95% Upper Conf Limit	0.0000
Exact Conf Limits	
95% Lower Conf Limit	0.0000
95% Upper Conf Limit	0.3085

2 Feinheiten zu PROC REPORT

2.1 Erstellen von horizontalen Linien in RTF-Tabellen

Bei RTF-Outputs werden vertikale Linien ohne weiteren Aufwand erzeugt. Etwas komplexer ist die Darstellung von horizontalen Linien. Anhand von Beispielen wird die Verwendung von STYLES, BRDRT, BRDRS und BRDW erklärt.

2.1.1 Erzeugen des Beispieldatensatzes

```
DATA test;
 ATTRIB
 subjid    length=5      label="Pat.ID"
 soc       length=$30    label="System Organ Class"
 event     length=$20    label="Adverse Event Term"
 ;
 subjid=123; soc="Eye disorders";    event="Glaucoma";       output;
                                     event="Conjunctivitis"; output;
             soc="Gastrointestinal"; event="Severe Nausea";  output;
 subjid=456; soc="Eye disorders";    event="Blind";          output;
```

```
                                        event="Conjunctivitis"; output;
                                        event="Glaucoma";       output;
             soc="Gastrointestinal"; event="Severe Nausea";  output;
 subjid=789; soc="Gastrointestinal"; event="Severe Nausea";  output;
RUN;
```

2.1.2 Regulärer Lst-Output

```
PROC REPORT data=test nowindows spacing=3 headline;
  column soc event subjid ;
  define soc    / order width=30 flow;
  define event  / order width=20 flow;
  define subjid /       width=10;
run;
```

System Organ Class	Adverse Event Term	Pat.ID
Eye disorders	Blind	456
	Conjunctivitis	123
		456
	Glaucoma	123
		456
Gastrointestinal	Severe Nausea	123
		456
		789

2.1.3 RTF Output ohne weitere Angaben:

```
ods rtf file = "&out_path./testpad.rtf" ;

PROC REPORT data=test nowindows
  style={rules=cols cellspacing=0 cellpadding=5};
  column soc event subjid ;
  define soc    / order width=30 flow;
  define event  / order width=20 flow;
  define subjid /       width=10;
run;

ods rtf close;
```

Die erzeugte Tabelle enthält vertikale Linien und einen äußeren Rahmen.

System Organ Class	Adverse Event Term	Pat.ID
Eye disorders	Blind	456
	Conjunctivitis	123
		456
	Glaucoma	123
		456
Gastrointestinal	Severe Nausea	123
		456
		789

2.1.4 RTF Output mit horizontalen Linien

Die Definition von horizontalen Linien basiert auf RTF Spezifikationen. Diese werden innerhalb eines COMPUTE Statements als call define definiert. Die nachfolgende Tabelle aus [1] bietet einen kurzen Überblick der verwendeten Einstellungen für die Absatzränder.

Paragraph borders:

\brdrt	Oberer Rand (Border top)
\brdrs	Einfache Rahmenstärke (Single-thickness border)
\brdrw*N*	Breite der Linie, angegeben mit Zahl als N ***(N*** is the width in twips of the pen used to draw the paragraph border line. ***N*** cannot be greater than 75. To obtain a larger border width, the ***\brdth*** control word can be used to obtain a width double that of ***N.)***

```
ods rtf file = "&out_path./testhorizontal.rtf" ;

PROC REPORT data=test nowindows spacing=3 headline headskip
  style={rules=cols cellspacing=0 cellpadding=0}
  style(header column)=[protectspecialchars=off ];
  column soc event subjid ;
  define soc    / order width=30 flow;
  define event  / order width=20 flow;
  define subjid /       width=10;
compute soc;
if soc ne ' ' then call define
   ('_c1_','style','style=[pretext="\brdrt\brdrs\brdrw11 "]');
endcomp;
compute event;
```

```
if event ne ' ' then call define
   ('_c2_','style','style=[pretext="\brdrt\brdrs\brdrw11 "]');
endcomp;
compute subjid;
if subjid ne . then call define
   ('_c3_','style','style=[pretext="\brdrt\brdrs\brdrw11 "]');
endcomp;
RUN;

ods rtf close;
```

Achtung: Die Tabelle muss im Print Preview oder Reading Layout angezeigt werden, um die horizontalen Linien zu sehen!

System Organ Class	Adverse Event Term	Pat.ID
Eye disorders	Blind	456
	Conjunctivitis	123
		456
	Glaucoma	123
		456
Gastrointestinal disorders	Severe Nausea	123
		456
		789

2.2 Seitenumbruch nach einer Spalte

Um Reports übersichtlich zu gestalten, wird häufig vor Beginn einer neuen Ausprägung der Gruppierungsvariablen ein Seitenumbruch eingefügt. Manchmal ist es aber auch wünschenswert, nach einer bestimmten Spalte einen Seitenumbruch zu erzwingen. PROC REPORT bietet auch die Möglichkeit, im DEFINE Statement mit /PAGE zu arbeiten.

Der Beispieldatensatz wurde um 3 Variablen erweitert: pt (preferred term), serious und related. Die Variable subjid wird als Identifier verwendet.

2.2.1 PROC REPORT mit automatischem Spaltenumbruch

```
PROC REPORT data=test3 nowindows spacing=3 headline;
  column subjid soc pt event serious related ;
  define subjid  / id order width=10;
  define soc     / order width=20 flow;
  define pt      / order width=20 flow;
  define event   /       width=20 flow;
  define serious /       width=7;
  define related /       width=7;
RUN;
```

Der erzeugte Output benötigt 2 Seiten, der automatische Umbruch erfolgt NACH der Variablen Event (Label Adverse Event Term).

```
Pat.ID   System Organ Class   Preferred Term      Adverse Event Term
_____________________________________________________________________
   123   Eye disorders        Conjunctivitis      Conjunctivitis
                              Glaucoma            Glaucoma
         Gastrointestinal     Nausea              Severe Nausea
         disorders
   456   Eye disorders        Blindness           Blind
                              Conjunctivitis      Conjunctivitis
                              Glaucoma            Glaucoma
         Gastrointestinal     Nausea              Severe Nausea
         disorders
   789   Gastrointestinal     Nausea              Severe Nausea
         Disorders
-------------------------  neue Seite  --------------------------

            serious   related
Pat.ID         flag      flag
_____________________________
   123             1         1
                   0         1
                   0         1
                   0         0
                   1         0
   456             0         1
                   1         0
                   0         0
   789             0         0
```

2.2.2 PROC REPORT mit definiertem Spaltenumbruch

Möchte man den Umbruch aber bereits vor der Spalte Event haben, kann man das als zusätzliche Option innerhalb des Define angeben.

```
PROC REPORT data=test3 nowindows spacing=3 headline;
  column subjid soc pt event serious related ;
  define subjid  / id order width=10;
  define soc     / order width=20 flow;
  define pt      / order width=20 flow;
  define event   /       width=20 flow page;
  define serious /       width=7;
  define related /       width=7;
RUN;
```

Der erzeugte Output benötigt ebenfalls 2 Seiten, der definierte Umbruch erfolgt VOR der Variablen Event

Pat.ID	System Organ Class	Preferred Term
123	Eye disorders	Conjunctivitis
		Glaucoma
	Gastrointestinal disorders	Nausea
456	Eye disorders	Blindness
		Conjunctivitis
		Glaucoma
	Gastrointestinal disorders	Nausea
789	Gastrointestinal disorders	Nausea

neue Seite

Pat.ID	Adverse Event Term	serious flag	related flag
123	Conjunctivitis	0	1
	Glaucoma	0	0
	Severe Nausea	0	1
456	Blind	1	0
	Conjunctivitis	0	1
	Glaucoma	1	0
	Severe Nausea	0	1
789	Severe Nausea	0	1

3 Arbeiten mit Datumsfeldern

3.1 Auffüllen von unvollständigen Datumsangaben mit dem letzten Tag des Monats

Um unvollständige Daten in die Analyse mit einzubeziehen muss man manchmal unvollständige Datumsangaben auffüllen, z.B. bei einem fehlenden Tag mit dem letzten Tag des Monats.

Folgende Ersetzungen sollen vorgenommen werden:

Wert	**Monat**
28	Februar ohne Schaltjahr
29	Februar mit Schaltjahr
30	April, Juni, September, November
31	Januar, März, Mai, Juli, August, Oktober, Dezember

Umständliche Algorithmen mit Zuweisung von 28, 29, 30 oder 31 Tagen kann man umgehen, in dem man vom ersten Tag des Folgemonats einen Tag abzieht. Dazu wird ein Makro vorgestellt.

Voraussetzungen zum Makro-Lauf: Vorhandene numerische Monatsvariable und Jahres-Variable, die beim Aufruf übergeben werden.

```
%macro get_lmdt(month=,year=,outdt=);

*** if month is December ---> fill with 31 ***;
if &month eq 12 then
  &outdt=input(compress("3112"||put(&year,z4.)),ddmmyy8.);

*** else ---> 01 of next month - 1 day ***;
else
   &outdt=input(compress("01"||put(&month+1,z2.)
                         ||put(&year,z4.)),ddmmyy8.)-1;

format &outdt date9.;
%mend get_lmdt;
```

3.2 Umwandeln von einzelnen Datums- und Zeitvariablen in eine Datetime Variable

Um aus mehreren Datums- und Zeitvariablen eine einzige Variable im Datetime Format zu erzeugen, wird diese oft mühselig durch Input und Konkatenierung zusammengesetzt, wobei der Zeitanteil oft noch in Sekunden umgerechnet wird. Die Funktion DHMS bietet eine kurze und elegante Methode, um ans Ziel zu kommen.

3.2.1 Umständliche Varianten:

```
datetm= input(put(dat,date7.) || "," || put(tim,time5.),datetime.);
```

oder

```
datetm= (dat*24*60*60)+ tim;
```

3.2.2 Elegantere Methode:

```
datetm= dhms(dat,0,0,tim);
```

Syntax der Funktion: DHMS(date,hour,minute,second)

Da eine SAS Zeitvariable die Zeit bereits in Sekunden enthält, werden Stunde und Minute auf 0 gesetzt.

Die DHMS Funktion eignet sich auch hervorragend um Zeitfenster abzufragen, z.B. 5 Stunden und 30 Minuten vor und nach Ende eines Event.

```
timelow = dhms(end,-5,-30,endtim);
timeupp = dhms(end,5,30,endtim);
```

4 ACHTUNG – Was geht NICHT in SAS

4.1 LAG-Funktion in einer IF/ELSE-Anweisung

Die LAG Funktion kann unerwartete Ergebnisse liefern, falls sie innerhalb einer IF/THEN-Anweisung aufgerufen wird.
Beispiel: In einer Studie mit mehreren Studienphasen wurde nur das Startdatum der Phase dokumentiert:

Obs	Patient ID	Phase	Start Datum
1	1	1	06MAY2009
2	1	2	27MAY2009
3	2	1	08MAY2009
4	2	2	29MAY2009
5	2	3	19JUN2009

Jede Phase endet mit dem Beginn der darauffolgenden Phase. Man möchte für alle Phasen – bis auf die letzte - das Enddatum berechnen. Für die letzte Phase soll das Enddatum fehlen.

Der Berechnungsalgorithmus wird wie folgt programmiert:

```
data phase_end;
   set phase;
   attrib enddt format=date9. label = "End Datum";

   * Den Datensatz auslesen (sortiert nach Patienten und
     innerhalb des Patienten absteigend nach Phase);
   by patient descending phase startdt;

   * für den ersten Eintrag innerhalb des Patienten (die letzte
     phase) -> Enddatum leer lassen;
   if first.patient then enddt = .;
   * ansonsten ist das Enddatum = Startdatum der nächsten Phase,
     d.h. der vorherigen Zeile;
   else enddt = lag(startdt);
run;
```

Das Programm liefert folgendes Ergebnis:

Obs	Patient ID	Phase	Start Datum	End Datum
1	1	1	06MAY2009	
2	1	2	27MAY2009	
3	2	1	08MAY2009	29MAY2009
4	2	2	29MAY2009	06MAY2009
5	2	3	19JUN2009	

Man erkennt, dass das Enddatum entweder gar nicht (beim Patienten 1) oder nicht korrekt (Patient 2, Phase 2) berechnet wurde. Offensichtlich liefert die LAG Funktion nicht den Wert des vorherigen Eintrags.
Es kommt nämlich darauf an, ob die Funktion innerhalb einer Bedingung aufgerufen wurde oder nicht. In der folgenden Tabelle sehen wir die Ergebnisse der Funktion, die mit dem gleichen Funktionsaufruf LAG(startdt) berechnet wurden, und zwar einmal außerhalb der IF/ELSE Klausel, dann innerhalb der IF und schließlich innerhalb der ELSE Anweisung.

Obs	Patient ID	Phase	Start Datum	Lag (normal)	Lag (if)	Lag (else)
1	1	2	27MAY2009			
2	1	1	06MAY2009	27MAY2009		
3	2	3	19JUN2009	06MAY2009	27MAY2009	
4	2	2	29MAY2009	19JUN2009		06MAY2009
5	2	1	08MAY2009	29MAY2009		29MAY2009

Falls die LAG Funktion innerhalb einer Bedingung aufgerufen wird, wird die Menge der für die Berechnung verwendeten Beobachtungen beschränkt und zwar auf die Beobachtungen, die diese Bedingung erfüllen. Jeder Aufruf der LAG Funktion stellt den aktuellen Wert in eine Queue und liefert den zuvor gespeicherten Wert zurück. Bei obigem Beispiel befindet sich der Funktionsaufruf innerhalb einer ELSE Anweisung, somit „kennt“ die LAG Funktion nur die Beobachtungen 2, 4 und 5. Der vorherige Wert für die Beobachtung 4 ist also der aus der Beobachtung 2 und nicht – wie gewünscht – aus der Beobachtung 3.
Den Wert der Variablen aus der vorherigen Beobachtung sollte man also außerhalb der Bedingung in eine Variable schreiben, und diese dann in der Bedingung verwenden:

```
data phase_end;
   set phase;
   by patient descending phase startdt;
   attrib enddt format=date9. label = "End Datum";

   lag_startdt = lag(startdt);

   if first.patient then enddt = .;
   else enddt = lag_startdt;
run;
```

Das Programm liefert nun das gewünschte Ergebnis:

Obs	Patient ID	Phase	Start Datum	End Datum
1	1	1	06MAY2009	27MAY2009
2	1	2	27MAY2009	
3	2	1	08MAY2009	29MAY2009
4	2	2	29MAY2009	19JUN2009
5	2	3	19JUN2009	

4.2 Vorsicht mit weiteren Anweisungen nach Merge

Der Tipp wurde von Hr. Elmar Dunkl, RPS Research Germany GmbH Nürnberg übernommen.

Manchmal führen Variablenmodifikationen in Kombination mit dem MERGE Statement zu unerwarteten Ergebnissen.

Beispiel: Patientendaten per Zentrum wurden erhoben:

Zentrum	Zentrum Nummer
Berlin	29
Heidelberg	99

Die Zentreninformationen sind in einem separaten Datensatz gespeichert:

Zentrum	Patient ID
Berlin	1
Berlin	2
Berlin	3
Heidelberg	4
Heidelberg	5

Die beiden Datensätze werden verknüpft:

```
data zentpat;
   merge zentinfo
         patinfo;
   by zent;
run;
```

und das Ergebnis sieht wie folgt aus:

Zentrum	Zentrum Nummer	Patient ID
Berlin	29	1
Berlin	29	2
Berlin	29	3
Heidelberg	99	4
Heidelberg	99	5

Nachträglich stellt sich heraus, dass sich Zentrennummern verändert haben (wurden um eins erhöht). Das Programm wird kurzerhand angepasst:

```
data zentpat;
   merge patinfo
         zentinfo;
   by zent;
   zentnum = zentnum+1;
run;
```

mit dem folgenden Ergebnis:

Zentrum	Patient ID	Zentrum Nummer
Berlin	1	30
Berlin	2	31
Berlin	3	32
Heidelberg	4	100
Heidelberg	5	101

Die Nummerierung für das Zentrum erhöht sich in jeder Beobachtung, und nicht – wie beabsichtigt – nur einmal für jedes Zentrum. Was ist passiert?

Zuerst ein kleiner Exkurs in die SAS Datenschrittverarbeitung: Hier geht SAS sequenziell vor und arbeitet eine Beobachtung nach der anderen ab. Die Variablen werden vom Inputdatensatz ausgelesen, evtl. abgeleitet und die Zwischenergebnisse in einen Programmdatenvektor (PDV) geschrieben. Der Programmdatenvektor ist ein logischer Speicherbereich, der während der Datensatzerzeugung als Zwischenspeicher verwendet wird, um die Werte der aktuell verarbeiteten Beobachtung zu behalten. Sind alle Anweisungen abgearbeitet, so wird der Inhalt des PDV in den Zieldatensatz geschrieben.

In unserem Fall (1:N Verknüpfung + Modifikation) werden zuerst die Variablen FIRST. und LAST. erzeugt. Sie markieren jeweils die erste und die letzte Beobachtung innerhalb einer BY Gruppe.
Danach liest SAS die erste Beobachtung der ersten BY Gruppe aus den beiden Datensätzen aus und schreibt diese in den Programmdatenvektor. Nach der Bearbeitung der ersten Beobachtung aus dem letzten Datensatz und Ausführung anderer Anweisungen, schreibt SAS den Inhalt des Programmdatenvektors in den Zieldatensatz.
SAS behält die Werte aller Variablen im Programmdatenvektor mit Ausnahme der neu erzeugten Variablen und setzt das Verknüpfen der Datensätze fort bis alle Beobachtungen der ersten BY Gruppe abgearbeitet sind. Danach werden alle Variablen in dem Programmdatenvektor auf „Missing“ gesetzt und die Schritte für die nächste BY Gruppe wiederholt.
Das Behalten der Werte im Programmdatenvektor führt dazu, dass die Modifizierung mehrfach durchgeführt wird. Um dies zu vermeiden, sollen solche Ableitungen in einem separaten Datenschritt bzw. nur einmal innerhalb einer BY Gruppe (z.B. unter Verwendung der FIRST Anweisung) durchgeführt werden.

4.3 Ab SAS 9.2 Abfrage mit IN auch in der Makro-Sprache möglich

Im DATA Step ist es schon lange üblich, statt länglicher Abfragen mit OR den Operator IN zu verwenden. Der Code wird so übersichtlicher und beinhaltet weniger Wiederholungen. Außerdem ist es effektiver, weil keine Mehrfach-Abfragen nötig sind.

```
DATA subset;
   SET sashelp.class;
   *** ausgeschrieben ***;
   IF name='Alice' or name='James' or name='John' or name='Judy';
   *** abgekürzt mit IN ***;
   IF name in('Alice', 'James', 'John', 'Judy');
RUN;
```

Innerhalb der Makrosprache war es vor SAS 9.2 nicht möglich, mit %IN zu arbeiten.

```
*MACRO mit IN als operator;
%macro usein(proc,ds);
  %if %upcase(&proc) in (MEANS,PRINT) %then %do;
    proc &proc data=&ds;
    run;
  %end;
%MEND usein;

%usein(print,sashelp.class);
```

Die Ausführung des Makros ergab eine Fehlermeldung, die den vorzeitigen Abbruch zur Folge hatte:

```
ERROR: Required operator not found in expression:
       %upcase(&proc) in (MEANS,PRINT)
ERROR: The macro USEIN will stop executing.
```

Diese Fehlermeldung erscheint ab Version 9.2 bei Verwendung des IN Operators innerhalb von Makros nach wie vor - zur gewünschten Ausführung müssen zwei Optionen gesetzt werden:

- MINOPERATOR: Erlaubt es, den IN Operator zu benutzen.
- MINDELIMITER=: Ein Zeichen wird als Trennzeichen der einzelnen Werte in der Liste definiert. Standard: Leerzeichen

```
options minoperator mindelimiter=',';
```

In diesem Beispiel wurde der MINDELIMITER als Komma definiert, die Werte in der Abfrage werden durch Komma getrennt, wie auch in der DATA Step Version. Daraufhin funktioniert die Anwendung des IN Operators im Makro.
Der folgende Makroaufruf führt nun zum Ausdruck des Datensatzes sashelp.class.

```
%usein(print,sashelp.class);
```

5 Einfach bequem

5.1 Anzeigen von Variablennamen UND Variablenlabels (proc sql)

Leider zeigt PROC PRINT nur entweder die Variablennamen oder die -label an. Um beides zu sehen, kann folgendes Script verwendet werden.
Als Beispiel wird der Datensatz Class aus Sashelp verwendet. Der Datensatz wird ins Work Directory eingelesen und es werden Label vergeben.

```
DATA class;
  SET sashelp.class;
  label name= "Vorname" sex= "Geschlecht" age= "Alter in Jahren"
        height= "Größe in Inch" weight= "Gewicht in Pound";
RUN;
```

Übergabe der Library und des Memname in Makrovariablen:

```
%let lib= work;
%let memname= class;
```

Die folgenden Druckbefehle führen zur Anzeige entweder des Name oder des Label:

```
title3 "&lib..&memname.: name only";
PROC PRINT width= minimum data= &lib..&memname;
run;

title3 "&lib..&memname.: label only";
PROC PRINT width= minimum data= &lib..&memname label;
run;
```

Mit Hilfe von PROC SQL wird eine Makrovariable generiert, die beide Informationen enthält. Dazu benötigt man Angaben aus dem Dictionary-Datensatz Columns.

Obs	name	label
1	Name	Vorname
2	Sex	Geschlecht
3	Age	Alter in Jahren
4	Height	Größe in Inch
5	Weight	Gewicht in Pound

Alle ‘name‘ und ‘label‘ des ausgewählten Datensatzes werden in eine Makrovariable geschrieben, die später zum Ausdrucken verwendet wird

```
PROC SQL noprint;
  select trim(left(name)) !! "= '" !! trim(left(label))
         !! " (" !! trim(left(name)) !! ")'"
         as newlabel into :newlabel separated by " "
  from dictionary.columns
  where libname eq upcase("&lib") and memname eq upcase("&memname.")
;
quit;
```

Der Ausdruck wird dann mit der Option LABEL und der Angabe der Makrovariablen im Befehl Label gestartet:

```
title3 "&lib..&memname.: label (name)";
PROC PRINT width= minimum data= &lib..&memname label;
   label &newlabel.;
run;
```

Der Inhalt der Makrovariablen &newlabel ist der folgende:

```
Name= 'Vorname (Name)' Sex= 'Geschlecht (Sex)' Age= 'Alter in Jahren
(Age)' Height= 'Größe in Inch (Height)' Weight= 'Gewicht in Pound
(Weight)'
```

PROC PRINT erzeugt folgenden Output:

Obs	Vorname (Name)	Geschlecht (Sex)	Alter in Jahren (Age)	Größe in Inch (Height)	Gewicht in Pound (Weight)
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
...					

Vorsicht: Wenn im Variablenlabel einfache Hochkommata vorkommen, dann gibt es ein Problem. Man kann es aber mit einem zwischengeschalteten TRANSLATE überbrücken, falls notwendig.

5.2 Angriff der SAS Kloneditoren unter Windows

Speziell bei langen Programmen muss der Programmierer oft vor und zurück scrollen, bis alle Informationen zusammengetragen sind. SAS unter Windows bietet die Möglichkeit, den Editor zu klonen. Es wird ein zweites Fenster mit dem Inhalt des aktuellen Fensters geöffnet.

Dazu öffnet man im Editor mit dem aktuellen Programm ein neues Fenster:
Menü ***Window*** (Alt W) und dann ***New Window***.

Änderungen im Code werden in beiden Fenstern parallel durchgeführt, wobei man aber in den beiden Fenstern an unterschiedlichen Stellen im Programm sein kann. Das zweite Fenster kann ohne Datenverlust einfach wieder geschlossen werden. Erst beim Schließen des ersten Fensters erfolgt die Nachfrage, ob man speichern möchte.

6 Kurz und elegant

Oft gibt es bei der Programmierung von Datensätzen und Tabellen statt länglicher Definitionen zur Darstellung aller verschiedenen Möglichkeiten von Variablen-Zuweisun-

gen die Möglichkeit, durch geschicktes Nutzen von Befehls-Optionen Programmteile wesentlich abzukürzen. Einige Beispiele zu verschiedenen DO-Schleifen, Möglichkeiten mit SELECT und CALL MISSING:

6.1 Verschiedene DO Schleifen

Die klassische Anwendung einer Schleife ist das Hochzählen eines Index mit Start- und Endwert:

```
DO i=1 to n
```

Es sind aber auch Sprünge möglich, man kann Text in der Schleife verwenden und die Schleifenvariable kann auch als Variable im Datensatz weiterverwendet werden.

```
DO iday= 5, 7, 10, 14;
   output;
END;

DO proc= "Print", "Freq", "Report";
   output;
END;
```

Anwendungsbeispiel: Erzeugen eines Dummy-Datensatzes zur späteren Tabellierung aller möglichen Ausprägungen einer Variablen

6.1.1 Version mit einzelnen Zuweisungen

```
DATA dummy;
  catord=1; cattxt='Number of discontinuations';       OUTPUT;
  catord=2; cattxt='Protocol violation';               OUTPUT;
  catord=3; cattxt='Protocol entry criterion not met'; OUTPUT;
  catord=4; cattxt='Adverse Event';                    OUTPUT;
  catord=5; cattxt='Lost to follow-up';                OUTPUT;
RUN;
```

6.1.2 Version mit Variablennamen in DO-Schleife und Format

Hier wird zunächst ein Format für die Textvariable definiert. Das hat den Vorteil, dass Änderungen einfacher im Format durchgeführt werden können. Außerdem wird das Format noch weiterverwendet werden.

```
PROC FORMAT;
   value txtf
      1 = 'Number of discontinuations'
      2 = 'Protocol violation'
      3 = 'Protocol entry criterion not met'
      4 = 'Adverse Event'
      5 = 'Lost to follow-up'
      ;
RUN;
```

Die Schleifenvariable catord wird als Ordnungsvariable weiterverwendet, die Textvariable wird mit dem Format erzeugt.

```
DATA dummy;
  DO catord=1 TO 5;
     cattxt=put(catord, txtf.);
     OUTPUT;
  END;
RUN;
```

6.2 Möglichkeiten des SELECT Statements

6.2.1 Mehrere Elemente in der WHEN Klammer

Statt einzelner Variablenausprägungen in der WHEN Klammer können auch mehrere Variablenausprägungen zusammengefasst werden.

```
SELECT(catord);
  when (3,4,5) put ">= 3";
  when (1,2)   put "<= 2";
  otherwise;
END;
```

6.2.2 Komplette Bedingung direkt in WHEN Klammer

Man kann auch die Klammer hinter SELECT weglassen und die komplette Bedingung direkt in die WHEN Klammer einfügen.

```
SELECT;
  when (catord IN (3,4,5)) put ">= 3";
  when (catord IN (1,2))   put "<= 2";
  otherwise;
END;
```

6.3 CALL MISSING

Diese schöne Funktion zum Initialisieren gibt es neu ab Version 9. Sie weist einer Liste von Variablen fehlende Werte zu, die numerisch oder alphanumerisch sein können.

Syntax: CALL MISSING (n1, c1, n2, c2, ...)

Im folgenden Beispiel werden die Variablen num1, num2 und char1-char3 initialisiert. Dass es sich um numerische bzw. alphanumerische Variablen handelt, wird über das LENGTH-Statement festgelegt. Beim Initialisieren muss man sich dann keine Gedanken darüber machen, ob die Variable numerisch oder alphanumerisch ist.

```
DATA dummy;
  LENGTH num1 num2 4 char1-char3 $10 ;
  CALL MISSING (num1, num2, char1, char2, char3);
RUN;
```

Das entspricht den Zuweisungen:

```
Num1 = .;
Num2 = .;
Char1 = "";
Char2 = "";
Char3 = "";
```

Im Vergleich dazu wird bei der Funktion MISSING nur geprüft, ob das Argument einen fehlenden Wert hat. Der Wert selbst wird nicht verändert.

```
IF missing(var1) THEN...
```

Hinweis: In der Variablenliste müssen die Variablen bei CALL MISSING durch Komma getrennt werden, eine Aufzählung wie char1-char3 ist nicht möglich.
Fehlermeldung:

```
CALL MISSING (num1, num2, char1-char3);

WARNING 134-185: Argument #3 is an expression, which cannot be up-
dated by the MISSING subroutine call.
```

7 Nützliche Funktionen

7.1 SMALLEST und LARGEST (ab SAS 9)

Mit den Funktionen ist es möglich, den x-kleinsten / x-größten Wert einer Zahlenreihe zu ermitteln.

Syntax: SMALLEST (k, value-1<,value-2 ...>)
LARGEST (k, value-1<,value-2 ...>)

k=numerische Konstante, Variable oder Ausdruck
value=Liste von Variablen, die bearbeitet werden

Beispiel: SMALLEST (k, 456, 789, ., 123);

```
DATA largsmall;
   label k="x-" xklein='SMALLEST' xgross='LARGEST';
   do k = 1 to 4;
      xklein = smallest (k, 456, 789, ., 123);
      xgross = largest  (k, 456, 789, ., 123);
      output;
   end;
RUN;
```

x-	SMALLEST	LARGEST
1	123	789
2	456	456
3	789	123
4		

Wenn die Angabe von k fehlt, erfolgt eine Fehlermeldung.

Missings werden ans Ende sortiert und zunächst ignoriert. Es werden immer erst die vorhandenen Werte genommen und erst wenn die Anzahl von k größer ist als die Anzahl der vorhandenen Werte wird missing ausgegeben. Es erfolgt keine Fehlermeldung, aber eine NOTE über fehlende Werte wird angezeigt.

```
NOTE: Missing values were generated as a result of performing an
operation on missing values.
```

Wenn k größer als die Anzahl der Gesamtwerte ist (inklusive Missings), erfolgt eine Fehlermeldung.

7.2 STRIP (ab SAS 9)

In einem Schritt werden alle führenden und abschließenden Leerzeichen von einem Textargument entfernt. Dies ist eine Verkürzung gegenüber der V8: left(trim(string)).

Syntax: STRIP(text)

Beispiel:

```
svar="*"||strip("   Text1    Text2   ")||"*";
```

➔ *Text1 Text2*

7.3 COMPBL (ab SAS 8)

In einem Schritt werden mehrfache Leerzeichen innerhalb des Arguments durch nur ein Leerzeichen ersetzt.

Syntax: COMPBL(text)

Beispiel:

```
cvar="*"||compbl("   Text1    Text2   ")||"*";
```

➔ * Text1 Text2 *

7.4 SUBSTRN (ab SAS 9)

Ist verwandt mit der SUBSTR Funktion, verhält sich anders bei ‘ungültigen’ Argumenten (z.B. negative Startposition, negative Textlänge). SUBSTRN ist robuster.

Syntax: SUBSTRN(Text, Position <, Länge>)

Text:	Konstante, Variable oder Ausdruck. Wenn Text numerisch ist, wird zunächst in Zeichen umgewandelt mit BEST32, anschliessend werden Leerzeichen am Anfang und Ende entfernt.
Position:	Startposition für die zu extrahierenden Zeichen in Text.
Länge:	Integer Länge der Zeichenkette (ohne Angabe bis zum Ende des String).

Die Funktion ist sehr robust:

- Wird die Länge der Zielvariablen im DATA Step nicht definiert, erhält sie die Länge des ersten Argumentes.
- Wenn Position oder Länge fehlen, erhält man einen String der Länge 0.
- Wenn die Positionsangabe < 0 ist, wird das Ergebnis am Anfang abgeschnitten. Das erste Zeichen des Ergebnisses ist das erste Zeichen des String. Die Länge wird entsprechend angepasst.
- Wenn "length" größer als "string" ist, wird am Ende des "strings" abgeschnitten. Das letzte Zeichen des Ergebnisses ist das letzte Zeichen des Strings.

Hier ein Beispiel für die Umwandlung von numerischen Angaben in Zeichen.

```
data substrn;
  v_substr  = "*" || SUBSTR(1234.5678,2,6) || "*";
  v_substrn = "*" || SUBSTRN(1234.5678,2,6) || "*";
run;
```

Ergebnisse der Variablen nach Anwendung der Funktion SUBSTR bzw. SUBSTRN:

```
v_substr  =*  1234*
v_substrn =*234.56*
```

Mit einer Stellenübersicht wird der Unterschied zwischen beiden Funktionen deutlich:

```
123456789012 - Stelle 1-12
   1234.5678 - Umwandlung in Zeichen durch substr
1234.5678    - Umwandlung in Zeichen durch substrn
```

SUBSTR wandelt auf 12 Stellen rechtsbündig um, SUBSTRN wandelt zunächst auf BEST32 um, entfernt dann aber die führenden Leerzeichen.

In der SAS Language Reference findet sich eine Übersicht der Unterschiede zwischen SUBSTR und SUBSTRN.

Literatur

[1] Rich Text Format (RTF) Specification, version 1.6
http://msdn.microsoft.com/en-us/library/aa140283(v=office.10).aspx#rtfspec_paraborders

Funktionen und Befehle allgemein wurden auf folgenden Internetseiten nachgeschlagen:

[2] SAS(R) 9.2 Language Reference: Dictionary, Fourth Edition
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#titlepage.htm

[3] SAS(R) 9.2 Language Reference: Concepts, Second Edition
http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#titlepage.htm