Tipps und Tricks
für den leichteren Umgang mit der SAS Software

Hans-Peter Altenburg, Heinrich Stürzl
Siemens Healthcare Diagnostics
Emil-von-Behring-Str. 76
35041 Marburg

Almut Hahn
MSource Medical Development GmbH
Lütticher Straße 281
52064 Aachen

Carina Ortseifen
Universitätsrechenzentrum Heidelberg
Im Neuenheimer Feld 293
69120 Heidelberg
carina.ortseifen@urz.uni-heidelberg.de

Grischa Pfister
iCASUS GmbH
Vangerowstraße
69117 Heidelberg
g.pfister@icasus.de

Zusammenfassung

In Form von kurzen Beiträgen werden nützliche Lösungen zu Problemen und Fragestellungen vorgestellt, die bei der täglichen Arbeit mit der SAS Software auftreten können. Es werden dabei nicht unbedingt neue Prozeduren, Optionen oder Module vorgestellt. Stattdessen soll die effektive Anwendung vorhandener Anweisungen und Prozeduren an Beispielen aufgezeigt werden. Unter anderem sind folgende Themen geplant:

1. Gegenstück zur Lag-Funktion - Wie kann man auf die nachfolgende(n) Beobachtung(en) im Data Step zugreifen?
2. Formate automatisieren
3. Visualisierung von zweifaktoriellen Designs mit der Prozedur Boxplot
4. Beschriftung von Balkendiagrammen – Intervallgrenzen statt Intervallmittelpunkte
5. Makros verstecken
6. Eigene Makrofunktionen
7. SAS/ODS – Makro zum Routing der SAS-Ausgabe in ein RTF- oder PDF-Dokument
8. National Language Support (NLS)

Neben den Tipps wird die Publikumsfrage von der 11. KSFE 2007 in Ulm aufgelöst.

Schlüsselwörter: Balkendiagramm, BOXPLOT-Prozedur, Data Step, Firstobs, FMTSEARCH-Option, Formate, FORMAT-Prozedur, GCHART-Prozedur, genestetes Design, hierarchisches Design, kompilierte Makros, LAG-Funktion, Makrobibliothek, Makrofunktion (eigene), MERGE-Anweisung, MSTORED-Option, Nachfolger, National Language Support (NLS), ODS, one-to-one merging, Optionen speichern, OPTLOAD-Prozedur, OPTSAVE-Prozedur, Option FMTSEARCH, Option MSTORED, Option SASMSTORE, Option STORE, PDF, PROC BOXPLOT, PROC FORMAT, PROC GCHART, PROC OPTLOAD, PROC OPTSAVE, RTF, SAS-Makro, SASMSTORE-Option, STORE-Option, Zweifaktorielles Design.

1 Gegenstück zur Lag-Funktion - Wie kann man auf die nachfolgende(n) Beobachtung(en) im Data Step zugreifen?

Von: Heinrich Stürzl

Die Funktion LAG*n*() mit *n*=1, 2, … liefert im Data Step den n-ten Vorgänger einer Beobachtung. Sie erlaubt dadurch Vergleiche mit vorangegangenen Beobachtungen. In manchen Situationen wird jedoch die Ausprägung einer nachfolgenden Beobachtung benötigt, um davon die Behandlung der aktuellen Beobachtung abhängig zu machen. Dafür gibt es bisher keine entsprechende Funktion in SAS. Nachfolgend wird gezeigt, wie man auf einen beliebigen Nachfolger einer Beobachtung zugreifen kann.

1.1 Anwendungsbeispiel: Blöcke ohne By-Variable

Gegeben:
Unterschiedlich lange Blöcke jeweils aufsteigender Zahlenreihen, wobei eine By-Variable zur Identifizierung der Blöcke nicht zur Verfügung steht.

```
10
20
40
80
5
10
20
1
2
4
8
16
```

Gesucht:
Die jeweils letzte Beobachtung eines Blockes aufsteigender Zahlen d.h. diejenige Beobachtung, bei der die Kette der aufsteigenden Zahlen endet und nach der eine kleinere Zahl oder keine weitere Zahl folgt. Im Beispiel die 4., 7. und 12. Beobachtung.
→ Bedingung für Blockende: Nachfolger ist kleiner als die aktuelle Zahl.

Die besondere Schwierigkeit dieser Aufgabe liegt darin, dass keine By-Variable zur Unterscheidung der Blöcke genutzt werden kann. Andernfalls wäre das Blockende direkt über die entsprechende LAST.Variable zu identifizieren.

1.2 Lösung: Versetzter Merge der Tabelle mit sich selbst

Um auf den direkten Nachfolger jeder Beobachtung zugreifen zu können, genügt es, die Tabelle (data set) mit einer um eine Beobachtung versetzten Kopie zu mergen. Die

Kopie beginnt mit der 2. Beobachtung des Originals (Option FIRSTOBS=2) und wird zur 1. Beobachtung in Form einer weiteren Variable hinzugefügt.

Angenommen die zu untersuchenden Zahlenreihen sind in der Variable Z in der Tabelle WERTE enthalten.

```
DATA neu;
  MERGE werte
        werte (FIRSTOBS=2 KEEP=z RENAME=(z=Next_z))
        ;
  IF Next_z < z THEN Blockende=1;
RUN;
```

→

z	Next_z	Blockende
10	20	.
20	40	.
40	80	.
80	5	1
5	10	.
10	20	.
20	1	1
1	2	.
2	4	.
4	8	.
8	16	.
16	.	1

1.3 Verallgemeinerung

Durch die Einstellung der Option FIRSTOBS= lässt sich der Versatz beider Tabellen steuern d.h. auf den wievielten Nachfolger zugegriffen werden soll.
Für den n-ten Nachfolger gilt FIRSTOBS=1+n.

Diese Lösung nutzt die Eigenschaft des „Merge ohne By“ (one-to-one merging), dass beim Mischen von Tabellen mit unterschiedlich vielen Beobachtungen die Ergebnistabelle genau so viele Beobachtungen hat wie die größte Einzeltabelle. Dadurch bleiben alle originalen Beobachtungen erhalten. Außerdem werden dadurch in der Variable mit den Nachfolger Werten (Next_z) am Ende der Tabelle entsprechende Missing Values erzeugt.

2 Formate automatisieren

Von: Grischa Pfister

Formate spielen in SAS/BASE eine herausragende Rolle. Sie werden für eine Vielzahl von Aufgaben verwendet, z.B. um kodierte Werte in der Anzeige durch Klartext zu ersetzen oder zur Klassifizierung von Gruppierungsvariablen .
Typische Aufgaben aus dem Bereich Datenmanagement sind das Umschlüsseln von Werten und die Validierung von Daten.
An dieser Stelle geht es um die Frage, wie Formate permanent gespeichert und dann vom SAS System auch wiedergefunden werden, und wie vorhandene Formate z.B. zu Dokumentationszwecken in SAS-Tabellen geschrieben werden können. Der umgekehrte Weg, nämlich Formate direkt aus SAS-Tabelle zu erzeugen, erlaubt die vollständige Automatisierung von Formaten und erleichtert die Aktualisierung wesentlich.

2.1 Formate permanent speichern

Formate werden permanent angelegt, indem beim Aufruf der Prozedur FORMAT noch der Name einer Bibliothek plus eventuell der Name eines Kataloges angegeben wird.

```
Proc Format lib=libref<.catalog>;
…
```

Wird der Katalog weggelassen, speichert SAS die Formate automatisch in einem Katalog namens FORMATS. Ist ein Format permanent gespeichert, bedeutet das aber mitnichten, dass es auch automatisch in SAS zur Verfügung steht, denn es gibt hierfür vordefinierte Suchpfade und hier muss der eigene Katalog zunächst registriert werden.
Die erste von zwei Möglichkeiten ist, den bereits vorhandenen Suchpfad zu nutzen. SAS sucht Formate zunächst in WORK.FORMATS, dann in LIBRARY.FORMATS um anschließend auf die vordefinierten Formate zuzugreifen. Die Bibliothek LIBRARY ist SAS-seitig nicht allokiert, sondern dient als Möglichkeit, eigene Formate ohne viel Aufwand verwenden zu können.

```
* GP Library fuer Formate anlegen **;
Libname gpFmt "d:/temp";

* GP Format in Katalog gpFmt.Formats erstellen *;
Proc Format lib=gpFmt;
  Value note
    1 = "toll"
    2 = "nicht so toll"
    other = "inakzeptabel"
  ;
Run;

* GP Test des Formates
     -> wird nicht gefunden *;
Data _null_;
  x = 1;
```

```
  Put x= x=note.;
Run;
```

LOG:

```
12   Data _null_;
13     x = 1;
14     Put x= x=note.;
                -----
                48
ERROR 48-59: The format NOTE was not found or could not be loaded.

15   Run;
NOTE: The SAS System stopped processing this step because of errors.
```

```
* GP Bibliothek LIBRARY setzen *;
Libname library (gpFmt);

* GP Zweiter Test des Formates
     -> jetzt wird es gefunden *;

Data _null_;
  x = 1;
  Put x= x=note.;
Run;
```

Im Beispiel wird zunächst eine Bibliothek allokiert und dann ein Format erstellt. Da kein Katalogname angegeben ist, wird automatisch FORMATS verwendet. Das ist auch gewollt, da später der automatische Suchpfad LIBRARY.FORMATS benutzt werden soll.
Ein erster Test zeigt, dass das Format nicht zur Verfügung steht, deshalb wird anschließend in einem weiteren Libname-Statement die Bibliothek LIBRARY so angelegt, dass sie direkt auf die bereits vorhandene Bibliothek GPFMT verweist. Der anschließende Test zeigt, dass das Format jetzt gefunden wird.

Soll das Format in einer anderen Bibliothek und/oder in einem anderen Katalog gespeichert werden, oder gibt es mehrere Format-Kataloge, muss auf die Option FMTSEARCH zurückgegriffen werden. Sie steuert den Suchpfad für die benutzerspezifischen Formate. I.d.R. werden zunächst WORK.FORMATS und LIBRARY.FORMATS durchsucht, anschließend folgen die in FMTSEARCH angegebenen Kataloge, auch diese Reihenfolge ist aber beeinflussbar (vergl. [1] ab Seite 1648).

```
* GP Format neu anlegen in gpFmt.gpFormate *;
Proc Format lib=gpFmt.gpFormate;
  Value note
    1 = "toll"
```

```
    2 = "nicht so toll"
    other = "inakzeptabel"
  ;
Run;

* GP Suchpfad fuer Formate setzen *;
Options
  fmtsearch=(gpFmt.gpFormate)
;

* GP Format testen *;
Data _null_;
  x = 1;
  Put x= x=note.;
Run;
```

Das Format wird diesmal in einen eigenen Katalog gespeichert, der mit Hilfe von FMTSEARCH in den Suchpfad aufgenommen wird. Der anschließende Test zeigt, dass das Format gefunden wird. Dabei wird – abhängig von der Suchreihenfolge – das erste passende Format verwendet. Diese Konkatenierung von Katalogen kann übrigens auch anders erfolgen, seit Version 8 gibt es das CATNAME-Statement, das ebenfalls eine solche Funktionalität zur Verfügung stellt.

```
Proc Format lib=gpfmt.cat01;
  value test other = "cat01 test";
Run;

Proc Format lib=gpfmt.cat02;
  value test other = "cat02 test";
  value cat other = "cat02";
Run;

Catname gpfmt.formats (gpfmt.cat01 gpfmt.cat02);

Options
  fmtsearch=(gpfmt)
;

Data _Null_;
  x = 1;
  put x = test.;
  Put x = cat.;
Run;
LOG:
```

```
x=cat01 test
x=cat02
NOTE: DATA statement used (Total process time):
```

Zunächst wird in Katalog GPFMT.CAT01 das Format TEST angelegt, in Katalog GPFMT.CAT02 sind es ebenfalls das Format TEST sowie ein Format CAT. Anschließend werden beide Kataloge zusammengefasst zum Katalog GPFMT.FORMATS, der dann via FMTSEARCH in den Suchpfad aufgenommen wird. Die Ausgabe im LOG zeigt, dass das Format TEST aus Katalog CAT01 verwendet wird, während Format CAT aus Katalog CAT02 genommen wird. Die Such-Reihenfolge wird in diesem Falle also durch die Angabe im CATNAME-Statement festgelegt, da in CAT01 bereits ein Format namens TEST gefunden wird, bleibt das gleichnamige Format in CAT02 unbeachtet.

2.2 Formate in eine SAS-Tabelle speichern

Dies ist zum einen hilfreich, wenn Formate von Dritten übernommen werden und der eigentliche Code nicht zur Verfügung steht. Aus der resultierenden Tabelle kann genau abgelesen werden, welche Klassifizierungen vorgenommen werden. Zum anderen kann dies auch ein sinnvolles Werkzeug für die Dokumentation von Formaten sein, da die reguläre Ausgabe von PROC FORMAT nicht als ODS-Outputobjekt angeboten wird.

```
* GP Format-Definition in Tabelle ausgeben *;
Options
  ls = 256
  nocenter
;
Proc Format lib=gpFmt.gpFormate cntlout = Work.Fmt_Note;
  Select note;
Run;

Proc Print Data = Work.Fmt_Note;
Run;
```

Durch Angabe der Option CNTLOUT=*Libref.Table* wird das Format in eine sog. Kontroll-Tabelle ausgegeben, die dann weiterverarbeitet werden kann. Eine genaue Beschreibung der Tabelle und der enthaltenen Tabelle bietet [2] ab Seite 466.

2.3 Formate aus Tabellen erstellen

Diese Kontroll-Tabellen können natürlich auch verwendet werden, um ein Format neu zu erstellen, wobei der eigentlich interessante Ansatz ist, bereits vorhandene Tabellen so zu erweitern, dass sie als Input-Kontroll-Tabelle verwendet werden können. Dieser Ansatz wird in einer Vielzahl von Projekten verwendet, um automatisiert Anpassungen an Formaten vorzunehmen, die einer starken Dynamik unterliegen.

```
* GP Input Control Data Set fuer Format erzeugen *;
Proc Sql;
  Create View Work.Tmp As
    Select "name2sex" As fmtName, "C" As type,
      name As start, name As end, sex As label
    From Sashelp.Class
  ;
Quit;

Proc Format cntlin = Work.Tmp;
Run;

Data _Null_;
  Set Sashelp.Class(keep = name);
  Put name = @20 name $name2sex.;
Run;
```

Im Beispiel wird die Tabelle SASHELP.CLASS so umgewandelt, dass sich eine Input-Kontroll-Tabelle für PROC FORMAT ergibt. Die Konstante „name2sex“ wird als Format-Name (FMTNAME) angelegt, Beginn (START) und Ende (END) des jeweiligen Format-Bereiches werden durch die Variable NAME festgelegt. Der Format-Wert (LABEL) wird aus der Variable SEX ausgelesen. Ergebnis ist ein Format, das den Namen in das zugehörige Geschlecht übersetzt.

Literatur:

[1] SAS 9.1.3 - Language Reference: Dictionary, fifth Edition.

[2] Base SAS .1.3 Procedures Guide, second Edition.

3 Visualisierung von zweifaktoriellen Designs mit der Prozedur Boxplot

Von: Carina Ortseifen

3.1 Problemstellung

Boxplots dienen der grafischen Darstellung von Verteilungen von metrischen Variablen. Typische Anwendungen sind dabei Untersuchungen mit zwei oder mehr Gruppierungsvariablen, wie in unserem Beispiel: Hierbei werden die Veränderungen des Blutdrucks bei drei verschiedenen Krankheiten und vier Therapien notiert.

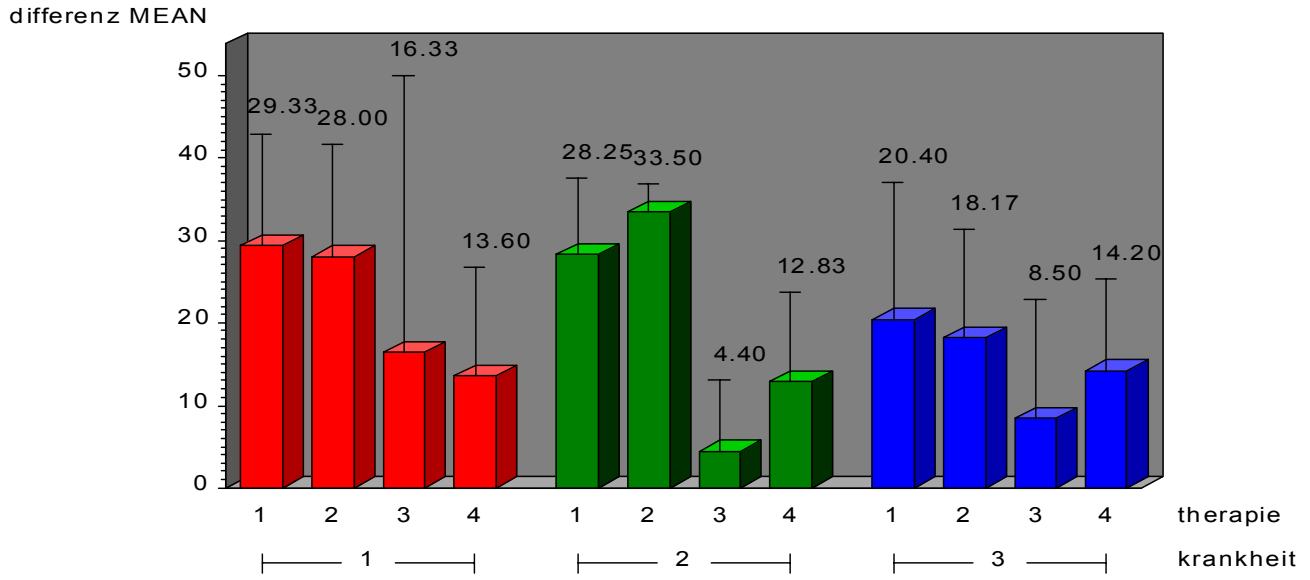
Beispieldatei:

Obs	therapie	krankheit	differenz
1	1	1	42

```
  2          1          1          44
  3          1          1          36
  .          .          .           .
 16          1          3          25
 17          1          3          25
 18          1          3          24
  .          .          .           .
  .          .          .           .
 71          4          3          12
 72          4          3           .
```

Der folgende Aufruf der Prozedur Gchart erzeugt einen ersten Eindruck von den zugrunde liegenden Beobachtungen:

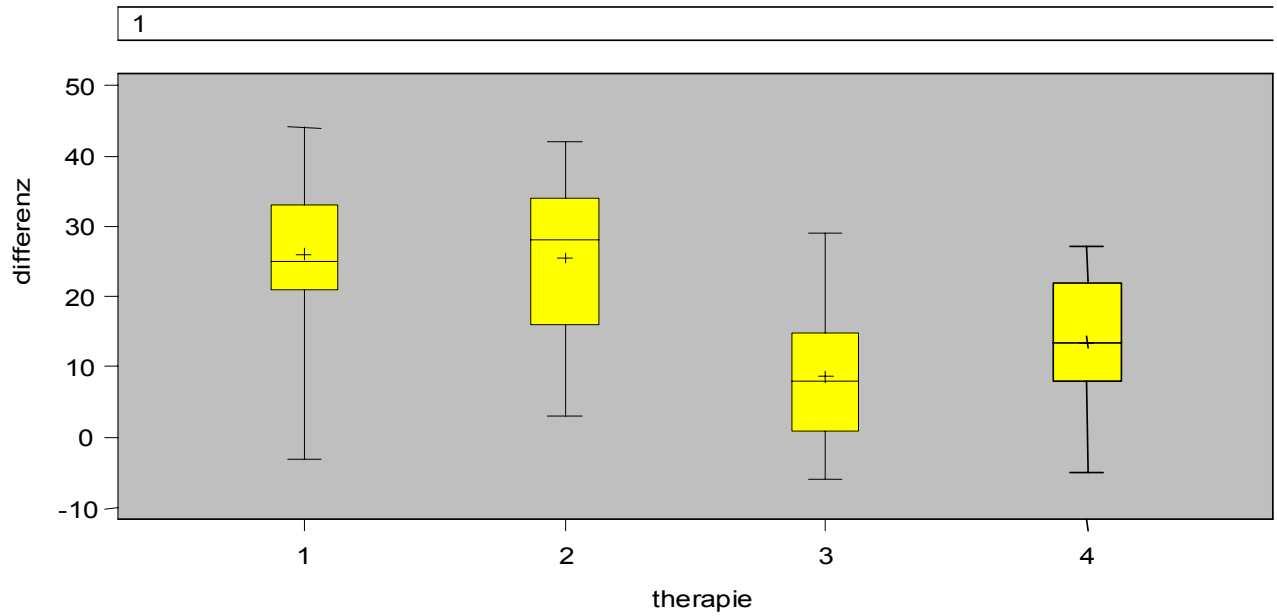
```
Axis1 Order=(0 To 50 By 10);
Proc Gchart Data=a;
   Vbar3d therapie /
   Type=Mean Mean
     Sumvar=differenz Discrete Group=krankheit Errorbar=Top
       Clm=95 Shape=Block Patternid=Group Raxis=axis1;
Run;Quit;
```



Da mit der Prozedur Gchart nur die Mittelwerte und die zugehörigen Standardfehler angezeigt werden können, soll im nächsten Schritt ein Boxplot erzeugt werden, das neben den Mittelwerten auch Mediane, 25- und 75%-Quartile, Minima und Maxima sowie Ausreißer darstellen kann. Der Aufruf mit:

```
Proc Boxplot Data=a;
   Plot differenz*therapie(krankheit)
```

```
   / Cboxfill=Yellow
        Cboxes=Black
        Cframe=Liggr;
Run;Quit;
```



führt allerdings nicht zu dem erwarteten Ergebnis sondern hinterläßt eher ein Fragezeichen beim Anwender.

Auch der Austausch der Rollen von therapie und krankheit führt zu keinem zufrieden stellendem Ergebnis. Daher wird als nächstes die Online-Hilfe näher studiert:

„*Syntax*: PLOT *(analysis-variables)*group-variable <(block-variables)>;*
group-variable specifies the variable that identifies groups in the data. The group variable is required.
block-variables specify optional variables that group the data into blocks of consecutive groups. These blocks are labeled in a legend, and each block variable provides one level of labels in the legend."

Und weiter:
"*Displaying Blocks of Data:* The values of a block variable must be the same for all observations with the same value of the group variable. In other words, groups must be nested within blocks determined by block variables. "

Ein anderer Begriff für genestete Daten ist hierarchisch angeordnet oder geschachtelt. In unserem Beispiel aber liegen zwei Faktoren vor, die gleichberechtigt sind: Krankheit und Therapie. Jeder Wert von Krankheit wird mit jedem Wert von Therapie kombiniert.

3.2 Lösung

Um die Beispieldaten dennoch in einem Boxplot darstellen zu können, sind daher folgende Schritte notwendig:

1. Generierung einer neuen Variablen, welche die Werte von Therapie und Krankheit kombiniert
2. Sortierung nach der neuen Variablen
3. Erzeugung eines Formats für diese neue Variable
4. Aufruf der Prozedur Boxplot

Schritt 1: Generierung der neuen Variablen kt

```
Data b; Set a;
   Label kt="Therapie";

   If therapie=1 and krankheit=1 then kt=1;
   If therapie=1 and krankheit=2 then kt=6;
   If therapie=1 and krankheit=3 then kt=11;

   If therapie=2 and krankheit=1 then kt=2;
   If therapie=2 and krankheit=2 then kt=7;
   ...
   If therapie=4 and krankheit=2 then kt=9;
   If therapie=4 and krankheit=3 then kt=14;
Run;
```

Schritt 2: Sortierung nach kt

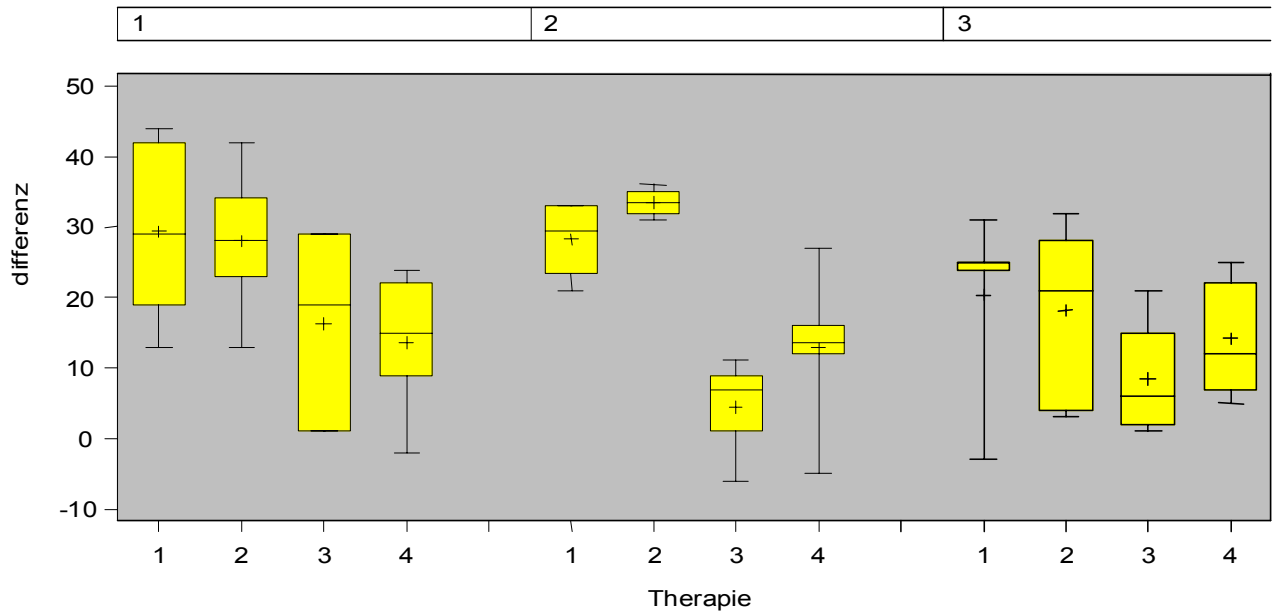
```
Proc Sort …;
   By kt;
Run;
```

Schritt 3: Erzeugung eines geeigneten Formats für kt und dessen Zuweisung

```
Proc Format;
   Value box
     1="1" 2="2" 3="3" 4="4" 5=" "
     /* Leerraum zwischen Blöcken */
     6="1" 7="2" 8="3" 9="4" 10=" "
    11="1" 12="2" 13="3" 14="4";
Run;
Data b;
   Set a;
   Format kt box.;
Run;
```

Schritt 4: Aufruf der Prozedur Boxplot

```
Proc Boxplot Data=b;
   Plot differenz*kt(krankheit)
   / Cboxfill=Yellow Cboxes=Black Cframe=Liggr Continuous ;
Run;Quit;
```



Damit erhalten wir ein zufrieden stellendes Bild mit den von uns gewünschten Werten.

Literatur

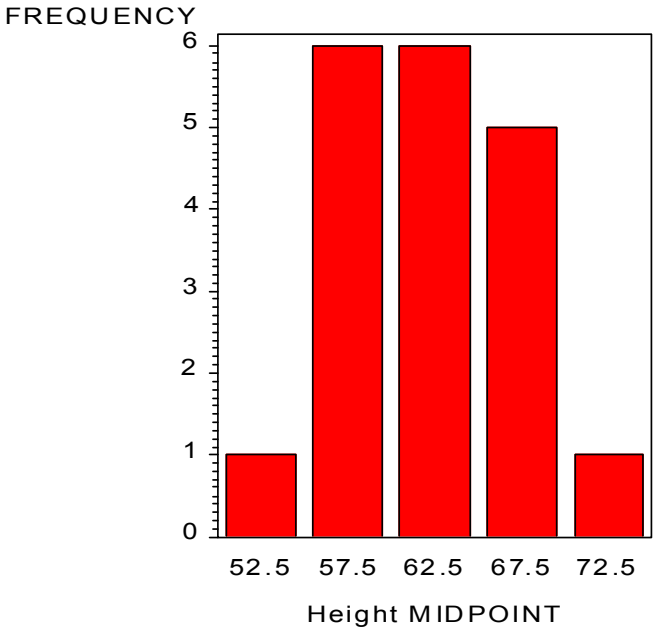
[1] Schneider, B.: PROC FORMAT Supports PROC BOXPLOT to Handle Twofold Grouped Data. SUGI 28, http://www2.sas.com/proceedings/sugi28/228-28.pdf [04.05.2008]

4 Intervallgrenzen statt Intervallmittelpunkte – Beschriftung von Balkendiagrammen

Von: Carina Ortseifen

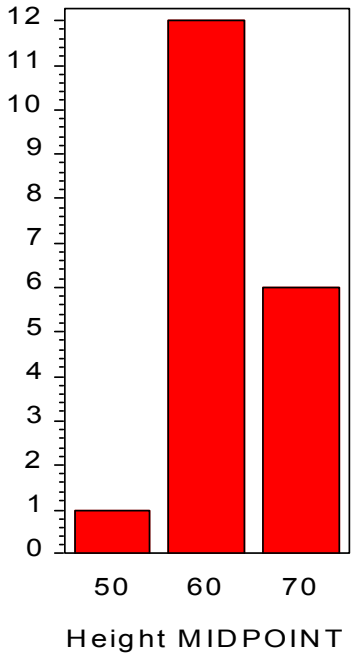
Im folgenden Tipp sollen verschiedene Möglichkeiten aufgezeigt werden, wie die Balkenachse bei mit der Prozedur Gchart im SAS System erzeugten Balkendiagramme beschriftet werden können. Dabei spielt es keine Rolle, ob die Anweisung Vbar oder Hbar verwendet wird.
Als Beispieldatei dient die Variable Height aus der SAS-Tabelle sashelp.class.

Mit dem **Standardaufruf der Prozedur Gchart** erhält man folgendes Diagramm:




```
Proc Gchart
   Data=sashelp.class;
   Vbar height;
Run;Quit;
```

Auf der vertikalen Achse links werden die Häufigkeiten (Frequency) ausgegeben, die horizontale Achse trägt die Balken. Dafür ermittelt das SAS System nach einem intern implementierten Algorithmus die Intervallgrenzen und –mittelpunkte und gibt diese "Midpoints" auf der Achse aus (Height Midpoint).



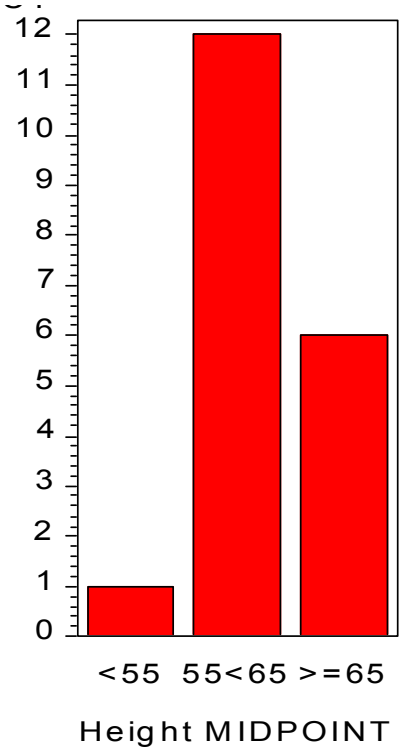
Mit der **Option Levels=** kann die Zahl der Balken vom Anwender bestimmt werden. In unserem Beispiel sollen drei Balken angezeigt werden.

```
Proc Gchart
   Data=sashelp.class;
   Vbar height /
     Levels=3;
Run;Quit;
```

Angezeigt werden die vom System errechneten Mittelpunkte 50, 60 und 70.

Ergänzt man die Syntax zusätzlich mit der Option **Range**, werden statt der Mittelpunkte die Intervallgrenzen der Balkendiagramme angezeigt:

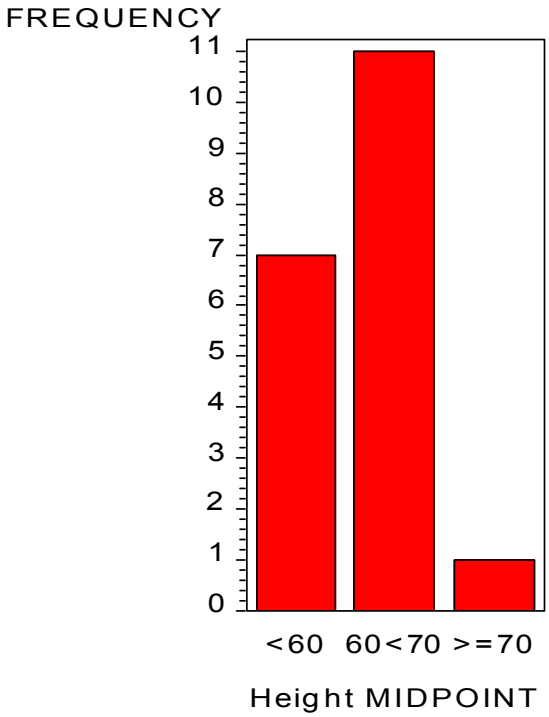
```
Proc Gchart
   Data=sashelp.class;
   Vbar height /
     Levels=3
     Range;
Run;Quit;
```



Die Balken umfassen dabei die halboffenen Intervalle, z.B. 55<65. Bei den beiden Balken am Rand wird als Grenze <55 bzw. >=65 angegeben.

Die Option Range kann auch mit **Midpoints=** kombiniert werden, wenn man feste Grenzen vorgeben möchte:

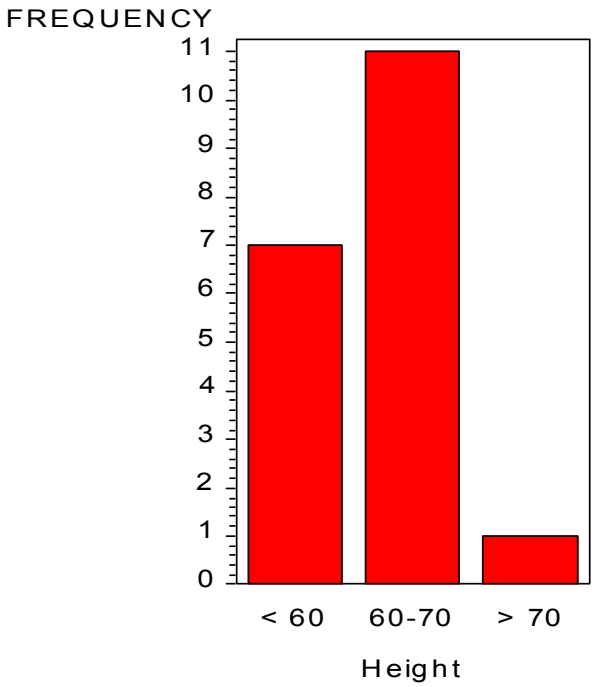
```
Proc Gchart
   Data=sashelp.class;
   Vbar height /
     Midpoints=55 65 75
     Range;
Run;Quit;
```



Um die Mittelpunkte 55, 65 und 75 werden nun die Intervallgrenzen angelegt, die Balken entsprechend eingezeichnet und die Intervallgrenzen angezeigt.

Jetzt bleibt nur noch ein Wunsch übrig: Nach unten halboffene Intervallgrenzen. Dafür muss zunächst ein **Format** definiert werden, das dann im Prozedur Gchart-Schritt der Variable height zugewiesen wird:

```
Proc Format;
   Value hoehe Low-60 = "< 60„
               60<-70 = "60-70„
             70<-High = "> 70";
Run;
Proc Gchart Data=sashelp.class;
   Vbar height / Discrete;
   Format height hoehe.;
Run;Quit;
```



Zusammenfassung

Per Default werden die Balkenmittelpunkte bei Anwendung der Prozedur Gchart mit Anweisung Vbar/Hbar angezeigt.
Einfluss auf Anzahl und Lage der Balken bieten die Optionen Midpoints= und Levels=.
Mit der Option Range können alternativ die Balkengrenzen dargestellt werden.
Der Einsatz von Formaten bietet darüber hinaus weitere Möglichkeiten.

Literatur

[1] Altenburg et al.: Tipps und Tricks für den leichteren Umgang mit der SAS Software
Proceedingsband der 11. KSFE 2007 Ulm. Shaker Verlag, 2007.

5 Makros verstecken

Von: Almut Hahn

Man möchte mühsam erarbeitete Makros einem anderen zur Verfügung stellen (Kollegen oder Kunden), möchte aber evtl. verhindern, dass derjenige den zugehörigen Code ändern oder auch nur einsehen kann (Stichwort: validierte Makros, Urheberrecht).

SAS bietet dazu die Möglichkeit, fertig kompilierte Makros abzuspeichern.

5.1 Kompilierte Makros abspeichern

Um kompilierte Makros abzuspeichern muss man

1. eine Bibliothek (library) einrichten
2. zwei Systemoptionen einstellen
3. das Makro unter einer bestimmtem Option kompilieren

Als Bibliothek könnte man beispielsweise einrichten:

```
libname mymacros „C:\...\Eigene Dateien\MeineMakros";
```

Als Systemoptionen müssen dann eingestellt werden:

```
options mstored sasmstore=mymacros;
```

Wobei `mstored` der SAS Session sagt, dass vorkompilierte Makros verwendet oder erzeugt werden können und `sasmstore=` erklärt, wo die Makros gesucht bzw. abgelegt werden sollen.

Beim Kompilieren des Makros muss dann die Makro-Option `/STORE` eingeschaltet sein

```
%macro rangecheck(mdat) /STORE;
proc means data=&mdat min max;
run;
%mend;
```

Durch diesen Vorgang wird unter `mymacros` eine Datei sasmacr.sas7bcat erzeugt, die das/die kompilierte(n) Makro(s) enthält und die dann an den User weitergegeben werden kann.

Man sollte unbedingt auch eine Klartextversion des Makros abspeichern, da sich der Code nicht aus der kompilierten Version rekonstruieren lässt.

5.2 Kompilierte Makros verwenden

Beim User müssen in dem Programm, in dem die kompilierten Makros verwendet werden sollen, eine entsprechende Bibliothek und die Systemoptionen wie oben gesetzt sein, also:

```
libname othmacros „C:\...\Eigene Dateien\FremdMakros";
options mstored sasmstore=othmacros;
```

Die kompilierten Makros können dann wie gewohnt aufgerufen werden. Da der User den Originalcode nicht mehr einsehen kann, sollte man daran denken, eine Anleitung mitzuliefern.

Durch die Verwendung bereits vorkompilierter Makros kommt es übrigens auch zu einer geringfügigen Einsparung in der Rechenzeit.

Bei möglicher Namensgleichheit verschiedener Makros ist folgende Reihenfolge zu berücksichtigen:

1. Zuerst werden die in der aktuellen SAS-Session erzeugten Makros durchsucht, (work.sasmacr)
2. anschließend die ggf. Unter `sasmstore=` hinterlegten
3. und danach ggf. mit der Option `sasautos=` eingebundene Makros.

5.3 Makros verstecken

Werden kompilierte Makros unter Systemoptionen wie z.B. MPRINT aufgerufen, so kann man im Log den Code weitgehend rekonstruieren. Man kann auch diese Stelle noch „verblinden", indem man am Anfang des Makros alle diese Optionen abstellt. Um die Funktion des restlichen Programms nicht zu stören, sollte man allerdings die bestehenden Optionen vorher abspeichern und hinterher wieder herstellen. Dies funktioniert beispielsweise mit `proc optsave` **`und`** `proc optload` (weitere Methoden siehe [1]). Es empfiehlt sich, zum Speichern der Optionen einen Makro-spezifischen Dateinamen zu wählen, damit es nicht zu Kollisionen kommt, falls innerhalb eines Makros ein anderes Makro aufgerufen wird.

```
%macro rangecheck (mdat) /STORE;

proc optsave out=rangecheck_opt;
run;

options NoMPrint NoMPrintnest NoMlogic NoMlogicNest
        NoMexecNote NoMFile NoMacrogen NoSymbolgen
        NoNotes NoSource NoSource2;

proc means data=&mdat min max;
run;
```

```
proc optload data=rangecheck_opt;
run;

proc datasets library=work nolist;
  delete rangecheck_opt;
quit;

%mend;
```

Die Liste der Optionen in diesem Beispiel erhebt weder einen Anspruch auf Vollständigkeit noch auf Notwendigkeit. Es ist natürlich jedem selbst überlassen, wie viel er seinem User sichtbar machen möchte.

Literatur

[1] Ortseifen et al.: Tipps und Tricks – Nützliche Lösungen zu Problemen und Fragestellungen der SAS Versionen 8, 8.1 und 8.2, Systemoptionen auslesen und speichern, Proceedings der 6. KSFE 2002 Dortmund. Shaker Verlag, 2003.

6 Eigene Makrofunktionen

Von: Grischa Pfister

Eigene Makrofunktionen erlauben es, Makro-Code wesentlich effizienter zu schreiben. Zum einen können oft verwendete SAS/BASE-Funktionen auf diese Weise gekapselt und in der Makrosprache verfügbar gemacht werden, zum anderen können typische Abläufe oder Aufgaben zu einer Funktion zusammengefasst werden.
Eine Makrofunktion ist zunächst ein ganz normales Makro, allerdings unterliegt es bestimmten Voraussetzungen. Dafür besitzt eine Makrofunktion aber einen Rückgabewert, der dann in Zuweisungen oder bedingten Verzweigungen abgefragt werden kann:

```
%Let wert = %Funktion();
%If ( %Funktion() ) %Then …
```

Ein Teil der SAS-eigenen Makrofunktionen ist so implementiert, unter !sasroot/core/sasmacro finden sich z.B. die Programme für %Trim und %Lowcase.

Das Grundprinzip der Makrosprache ist die Textersetzung, und dieses Prinzip steht auch hinter den Makrofunktionen. Der Rückgabewert ist in Wirklichkeit ein String, der übrig bleibt, wenn das Makro vom Makroprozessor verarbeitet worden ist. Dieser String wird dann an den Wordscanner zurückgegeben und an die Stelle des Makroaufrufes gesetzt. Aus diesem Grunde darf eine Makrofunktion keinen Text außer dem Rückgabewert erzeugen – anders formuliert bedeutet das, dass das Makro weder einen Data Step noch Prozeduren verwenden darf, denn das wäre „normaler“ Base-Code, der vom Makroprozessor zurück an den Wordscanner geht (zur Arbeitsweise von Wordscanner

und Makroprozessor siehe [1], ab Seite 36). Dafür stehen aber (fast) alle SAS/BASE Funktionen zur Verfügung und damit lässt sich allerhand anstellen.
Ein einfaches Beispiel sieht so aus:

```
%Macro bsp01;
  Textersetzung
%Mend;
%Put Das Grundprinzip ist %Bsp01 !;
```

Log:

```
Das Grundprinzip ist Textersetzung !
```

Das Makro enthält lediglich eine Textkonstante. Wenn der Code abgeschickt wird, wird zunächst das Makro kompiliert und dann das %PUT-Statement verarbeitet. Der Word-Scanner erkennt dabei den Makroaufruf %BSP01 und ruft den Makroprozessor auf. Der führt das Makro aus und setzt den Ergebnis-String „Textersetzung“ an die Stelle des Makroaufrufes.
Wichtig ist, dass die Zeichenkette „Textersetzung“ **nicht** mit einem Semikolon beendet wird. Es handelt sich hier nicht um eine Code-Zeile, sondern um den Rückgabewert, das Semikolon würde kein Statement abschließen, sondern wäre Teil des Wertes!! Das Semikolon würde also im Wordscanner landen und dort das eigentliche Programm durcheinanderbringen.

Das Prinzip von Makrofunktionen ist also sehr einfach, die Einsatzmöglichkeiten dafür umso größer, deshalb noch ein paar Beispiele.

```
* GP Pfad von Library/Fileref abfragen *;
%Macro GetPath(ref);
  %Local path;
  %Let path = %Sysfunc(pathname(&ref));
  &path
%Mend;

%Put WORK wird gespeichert unter %GetPath(work);
```

Oft sollen in Anwendungen temporäre Dateien abgelegt werden, wofür sich das Verzeichnis der WORK-Bibliothek anbietet. Um an diesen Pfad heranzukommen gibt es auch eine SAS/BASE Funktion namens pathname(). Die Makrofunktion %GetPath() kapselt den Zugriff auf die Funktion und kann z.B. auch verwendet werden um zu prüfen, ob eine Bibliothek überhaupt vorhanden ist.
Es ist üblich, den Rückgabewert zunächst in eine lokale Makrovariable zu speichern. Das macht es zum einen einfacher, Werte in mehreren Schritten zusammenzufassen, zum anderen stellt die lokale Makrovariable sicher, dass nicht unbeabsichtigt im übergeordneten Makro eine Makrovariable überschrieben wird (vergl. [1] ab Seite 42).

```
%* GP Erzeugen eines Timestamp *;
%Macro timestamp;
  %Local ts;
  %Let ts = %Sysfunc(date());
  %Let ts = %Sysfunc(putn(&ts,ddmmyyd10.));
  %Let ts = &ts._%Sysfunc(hour(%Sysfunc(time())))_   [Umbruch
            %Sysfunc(minute(%Sysfunc(time())));       layoutbedingt!]
  &ts
%Mend;

%Put Der aktuelle Zeitstempel ist: %Timestamp;
```

Ein weiteres typisches Beispiel sind Zeitstempel, die z.B. für die Benennung von Dateien gebraucht werden. Durch Hinzufügen von Parametern kann das Makro z.B. so erweitert werden, dass es nur Datum oder nur Uhrzeit zurückgibt. Genauso gut könnte das Makro verschieden formatierte Versionen des Zeitstempels liefern, hier sind viele Varianten denkbar.

```
%* GP Anzahl der Obs in Tabelle abfragen *;
%Macro Nobs(table);
  %Local dsid nobs;
  %Let dsid = %Sysfunc(open(&table));
  %If ( &dsId ) %Then %Do;
    %Let nobs = %Sysfunc(attrn(&dsid,nlobs));
    %Let dsid = %Sysfunc(close(&dsid));
  %End;
  %Else %Let nobs = -1;
  &nobs
%Mend;

%Put Sätze in Tabelle Sashelp.Class: %Nobs(Sashelp.Class);
%Put Sätze in Tabelle Sashelp.Shoes: %Nobs(Sashelp.Shoes);
```

Das letzte Beispiel verwendet SAS/BASE Funktionen, die ursprünglich aus der SCL (SAS Component Language) stammen. Mit deren Hilfe kann eine SAS-Tabelle geöffnet und Information aus dem Header ausgelesen werden (und einiges mehr, eine Übersicht der Funktionen für den Zugriff auf SAS-Tabellen gibt es in [2] ab Seite 304).
Da eine Makrofunktion weder einen Data Step noch eine Prozedur ausführen kann, ist es auf diesen Wegen nicht möglich, die Anzahl der Beobachtungen zu ermitteln. Mit Hilfe der SCL-Funktionen geht es aber doch. So kann z.B. ganz einfach geprüft werden, ob ein vorangegangener Schritt auch wirklich Beobachtungen in eine Ergebnistabelle geschrieben hat oder nicht.

Literatur

[1] SAS 9.1 Macro Language – Reference.

[2] SAS 9.1.3 Language Reference: Dictionary, fifth Edition.

7 SAS/ODS – Makro zum Routing der Ausgabe in ein RTF oder PDF Dokument

Von: Hans-Peter Altenburg

Die SAS-Ausgabe oder Teile einer SAS-Ausgabe sollen mit Hilfe zweier kleiner Makros in entsprechende PDF- bzw. RTF-Dokumente gespeichert und in zugehörige Unterverzeichnisse abgelegt werden.

Wir gehen der Einfachheit von folgender Verzeichnisstruktur aus: Neben einem Projekt-Hauptverzeichnis („MAINPROJEKT_XXX“) gibt es dort verschiedene Unterverzeichnisse, wie z.B. Data (für die Daten), P (für Programme), PDF (für PDF-Dokumente), RTF (für RTF-Dokumente), M (für SAS-Makros) usw.:

```
D:\....\MainProjekt_XXX
                \Data
                \M
                \P
                \PDF
                \RTF
                . . .
```

Zum Routing der Ausgabe benötigt man zwei SAS-Makros, welche etwa folgendermaßen aussehen können:

```
/* ---------------------------- ODS Start ------------- */
%MACRO StartODS(tout, stammpfad,Out_Dset) ;
%LET outpfad=&stammpfad\&tout ;
GOPTIONS ROTATE=Landscape ;
OPTIONS ORIENTATION=Landscape   PAPERSIZE="ISO A4"    ;
ODS LISTING CLOSE ;
ODS &tout FILE="&outpfad\&Out_DSet..&tout" STARTPAGE=NO
%IF %UPCASE(&tout)=PDF %THEN %STR(NOTOC ; ) ;
                       %ELSE %STR( ; ) ;
%MEND ;
/* ---------------------------- ODS Start ------------- */

/* ---------------------------- ODS End --------------- */
%MACRO EndODS(tout) ;
ODS &tout CLOSE ;
ODS LISTING ;
```

```
%LET tout=  ;
%MEND ;
/* --------------------------- ODS End -------------- */
```

Die Makrovariable "tout" bezeichnet den Dokument-Typ, also etwa RTF oder PDF.
Die Makrovariable "stammpfad" bezeichnet den Projektpfad (z.B. MAINPROJEKT_XXX).
Im angegebenen "stammpfad" muss ein Unterverzeichnis mit dem Namen RTF oder PDF existieren. Ein solches Verzeichnis könnte auch neu angelegt werden, dann muss das Makro entsprechend modifiziert werden.
Die Makrovariable "Out_Dset" bezeichnet den Namen der Datei (ohne die Erweiterung RTF bzw. PDF!) in welche die SAS-Ausgabe geschrieben werden soll.

Während die Ausführung des Makros STARTODS(...) das Routing in eine entsprechende Datei startet, führt das Makro EndODS(...) die Ausgabe wieder zurück ins normale LISTING. Mehrere Start/End "Klammern" hintereinander mit unterschiedlichen Dateinamen erzeugen entsprechend verschiedene Ausgabe-Dateien in den Verzeichnissen (PDF oder RTF).

Falls die Pfade / Verzeichnisse zuerst mit Ausführung des StartODS-Makros im Projektverzeichnis "stammpfad" angelegt werden sollen, so kann ein entsprechender DATA-Step ins Makro eingebaut werden, z.B.:

```
...
DATA _NULL_ ;        * Anlegen der Projektunterverzeichnisse im Verzeichnis stammpfad ;
ReturnCode=DCREATE("PDF","&stammpfad") ;
PUT /// 'PFad für das Projektunterverzeichnis angelegt: ' Returncode
/// ;
ReturnCode=DCREATE("RTF","&stammpfad") ;
PUT /// 'PFad für das Projektunterverzeichnis angelegt: ' Returncode
/// ;
RUN ;
...
```

Siehe hierzu auch Tipps und Tricks vom letzten Jahr!

Die Ausgabe erfolgt im Beispiel im Quer-Format, wegen der langen SAS Outputs und damit Grafiken besser zu interpretieren sind.

Beispiel:

```
%LET stammpfad=D:\... MAINPROJEKT_XXX  ;
%LET PDF_DSet =Bsp_NVData ;
%INCLUDE "&stammpfad\M\StartEndODS.sas" ;          * Einlesen der Macros ;

TITLE1 H=2 C=BLUE 'Standard-Normal verteilte Zufallsvariablen' ;
```

```
/* Erzeugen einer Test Datei */
DATA Bsp1 ;
     DO Gruppe=1 TO 3 ;
          DO IDVar=1 TO 20 BY 1 ;
          xx=NORMAL(0) ;
          OUTPUT ;
          END ;
     END ;
     RUN ;


/* ---------------------------- ODS Start ---------------------- */
%LET DocType=PDF ;            * zB   PDF RTF    ;
%StartODS(&DocType,&stammpfad,&PDF_DSet)
/* ---------------------------- ODS Start ---------------------- */

PROC PRINT DATA=Bsp1 ;
BY Gruppe ;
ID IDVar ;
RUN ;

ODS &Doctype STARTPAGE=NOW ;   * neue Seite im Output ;

PROC MEANS DATA=Bsp1  N Mean STD VAR ;
BY Gruppe ;
VAR xx ;
RUN ;

/* ---------------------------- ODS End ------------------------ */
%EndODS(&DocType)
/* ---------------------------- ODS End ------------------------ */
```

Individuelle Änderungen und Erweiterungen sind natürlich jederzeit möglich.
Das Beispiel lässt sich auch noch etwas modifizieren und die Ausgabe in zwei verschiedene Dateien mit Namen „`LISTING`“ bzw. „`MeanStd`“:

```
%LET DocType=PDF ;            * zB   PDF RTF    ;
/* - ODS Start - */ %StartODS(&DocType,&stammpfad,Listing)
/* ---------------------------- ODS Start ----------------- */
PROC PRINT DATA=Bsp1 ;
BY Gruppe ;
ID IDVar ;
RUN ;
/* - ODS End --- */ %EndODS(&DocType) /* -- ODS End -------- */
/* - ODS Start - */ %StartODS(&DocType,&stammpfad,MeanStd)
/* ---------------------------- ODS Start ----------------- */
PROC MEANS DATA=Bsp1  N Mean STD VAR ;
BY Gruppe ;
VAR xx ;
```

```
RUN ;
/* - ODS End --- */ %EndODS(&DocType) /* -- ODS End ------- */
```

8 National Language Support (NLS)

Von: Grischa Pfister

An dieser Stelle erfolgt keine vollständige Beschreibung des NLS, sondern es werden nur einige für den täglichen Umgang nützliche Eigenschaften vorgestellt. Eine ausführliche Darstellung findet sich unter [1].

Die Option **LOCALE** steuert verschiedene sprachabhängige Einstellungen des SAS-Systems. Dazu gehören Seitengrößen, Randeinstellungen und Standardformate. Das ist insbesondere für Datumsformate interessant, da es hier bekanntermaßen international sehr unterschiedliche Formen gibt. Durch Verwendung des NLS und der entsprechenden Formate ist es auf einfache Weise möglich, die eigenen Programme international verfügbar zu machen. Dabei ist zu beachten, dass der Wert für die LOCALE aus zwei Teilen besteht, der Angabe der Sprache plus dem Land - Arabisch wird eben in mehr als einem Land gesprochen. Eine Übersicht der möglichen Lokalen findet sich in [1] ab Seite 397.
Typische Zuweisungen sind:

```
Locale = German_Germany bzw. Locale = de_DE
Locale = French_Canada bzw. Locale = fr_CA
```

Die Namen der speziell für das NLS entwickelten Formate beginnen mit „nl“, es gibt eine ganze Reihe davon, die vollständige Liste findet sich in [1] ab Seite 63.
Das Format „nldate.“ beispielsweise gibt ein formatiertes Datum zurück:

```
Options locale = de_de;

Data _Null_;
  datum = date();
  Put datum= datum=nldate.;
Run;
Log:
datum=17661 datum=09. Mai 2008
```

Wird die LOCALE auf „it_IT“ umgestellt, zeigt das LOG

```
datum=17661 datum=09 maggio 2008
```

Für das Einlesen von Datumswerten steuert LOCALE die Einstellung der Option **DATESTYLE**. Hier wird für das Informat angegeben, in welcher Reihenfolge Tag, Monat und Jahr gelesen werden sollen (DATESTYLE = LOCALE). Die Option kann aber auch unabhängig von der Lokalen verwendet werden.

```
Options
  locale = de_de
  datestyle = locale
;

Data _Null_;
  string = "02/12/31";
  datum = input(string,anydtdte.);
  Put datum= datum=nldate.;
Run;
```

Das Beispiel liefert als Ergebnis „`datum=-10257 datum=02. Dezember 1931`“, wird die LOCALE auf „en_US“ gestellt aber „`datum=-10550 datum=February 12, 1931`“.
Über die Option DATESTYLE lässt sich auch innerhalb der gleichen LOCALE die Reihenfolge steuern, um Datumswerte mittels Informat anydtdte. einzulesen.

Dateststyle	**Ergebnis**
MDY	datum=-10550 datum=12. Februar 1931
DMY	datum=-10257 datum=02. Dezember 1931
YMD	datum=15705 datum=31. Dezember 2002

(MYD, YDM und DYM ergeben in diesem Beispiel missing values, da das Resultat kein gültiges Datum ist)

Die Funktionen NLDATE(), NLDATM(), und NLTIME() erzeugen eine Zeichenkette, die Datum, Zeit oder beides enthält. Der Trick dabei ist, dass dabei auf eine Vielzahl von Platzhaltern zurückgegriffen werden kann, die für den Monatsnamen, den Tag, die Stunde etc. stehen (und grundsätzlich schon aus PROC FORMAT bekannt sind).
Die Sprache, in der Monate, Tage etc. ausgeben werden, wird über die LOCALE erzeugt.

```
Options locale=de_DE;

Data _Null_;
  string = nldate(date(),'%A, %d. %B %Y');
  Put string=;
Run;
```

Das Beispiel liefert als Ergebnis zunächst „`string=Freitag, 09. Mai 2008`“, wird die LOCALE auf „fr_FR“ umgestellt „`string=vendredi, 09. mai 2008`“.

Diese Funktion ist natürlich prädestiniert, Zeitstempel jeder Art zu erstellen, z.B. lässt sich das weiter oben gezeigte Makro %Timestamp so wesentlich eleganter kodieren.

Literatur

[1] SAS 9.1 National Language Support (NLS) – User's Guide.

9 Auflösung der Publikumsfrage der 11. KSFE 2007 Ulm

Von: Carina Ortseifen

Aufgabenstellung

In einem Programmschritt (oder auch Makro) soll die SAS-Datei lib.tab erzeugt werden, etwa mit

DATA lib.tab;

Damit dies ohne Komplikationen abläuft, soll vorneweg geprüft werden, ob

1. der Bibliotheksname lib existiert und auf den richtigen Ordner zeigt und
2. die SAS-Datei tab in dieser Bibliothek noch nicht existiert.

Ergänzungen

Der richtige Ordnername für die Bibliothek steht in der Makrovariable libpfad.
Sollte die SAS-Datei lib.tab bereits existieren, wird diese vor Neuanlegen umbenannt in __tab.

Lösung

Von den eingereichten zwei Lösungen wird hier diejenige vorgestellt, die der Autorin am besten gefallen hat.

```
%macro ksfe_07_frage;
  %put Start Macro &sysmacroname;

  %if %sysfunc(libref(lib)) %then %put Libname lib nicht gesetzt;
  %else

    %if %sysfunc(pathname(lib)) ne &libpfad %then
      %put Libname lib zeigt nicht auf &libpfad;
    %else %do;

      %if %sysfunc(exist(lib.tab)) %then %do;
        %if %sysfunc(exist(lib.__tab)) %then %do;
          %put Dataset lib.__tab wird gelöscht;
```

```
          proc datasets library=lib nolist;
             delete __tab;
          quit;
        %end;

        %put Dataset lib.tab existiert und wird umbenannt in __tab;

        proc datasets library=lib nolist;
           change tab=__tab;
        quit;
      %end;

      %put Dataset tab wird in Bibliothek   lib erzeugt;

      data lib.tab;
         einEintrag="Hier steht was";
      run;
    %end;

  %put Ende Macro &sysmacroname;
  %put;

%mend ksfe_07_frage;
```

Ein herzliches Dankeschön abschließend für die aktive Teilnahme.