Tipps und Tricks für den leichteren Umgang mit der SAS Software

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

Grischa Pfister
iCASUS GmbH
Vangerowstraße 2
69115 Heidelberg
g.pfister@icasus.de

Heribert Ramroth
Universität Heidelberg
Institut für Public Health
Im Neuenheimer Feld 324
69120 Heidelberg
heribert.ramroth@uni-heidelberg.de

Marianne Weires
Deutsches Krebsforschungszentrum
Im Neuenheimer Feld 580
69120 Heidelberg
m.weires@dkfz.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.

1. Dateioptionen in PROC SQL
2. Verschachtelte SQL-Abfragen
3. Austausch von Variablenlisten zwischen PROC SQL und SAS Prozeduren/Datenschritten
4. PROC PRINT – Einfügen einer Leerzeile nach jeder x.ten Beobachtung
5. Vergleich von SAS Tabellen – Suchen nach gemeinsamen Tabellen
6. Orientierung in der SAS-Umgebung - Pfadangaben zu Bibliotheken, Formatkatalogen und Programmen
7. Farbangaben für SAS/Graph – leichtgemacht
8. Praxis Beispiel Annotate – ein Balkendiagramm erweitern

Schlüsselwörter: Prozedur SQL, SQL Prozedur, Dateioption, DROP, KEEP, RENAME, WHERE, SORTEDBY, Prozedur PRINT, PRINT-Prozedur, BLANKLINE=, Tabellenvergleich, Variablen gemeinsame, Prozedur COMPARE, COMPARE-Prozedur, Prozedur DATASETS, DATASETS-Prozedur, Dictionary Tables, Dictionary.Columns, Funktion PATHNAME(), PATHNAME() Funktion, Dictionary.Libnames, Dictionary.Extfiles, Funktion GETOPTION(), GETOPTION() Funktion, FMTSEARCH Option, Option FMTSEARCH, Windows-Umgebungsvariablen, Funktion %SYSGET(), %SYSGET() Funktion, SAS/GRAPH, Farben, Farbangabe, Annotate Facility, Annotate, SYSFUNC() Funktion

1 Dateioptionen innerhalb von PROC SQL

(Marianne Weires)

1.1 Einsatz von Dateioptionen

Im Folgenden wird die grobe Struktur von PROC SQL und der Einsatz von verschiedenen Dateioptionen erläutert. Dateioptionen innerhalb von PROC SQL sind SAS-spezifische Erweiterungen von SQL, sind optional und können nach CREATE TABLE oder FROM folgen.

```
PROC SQL;
  CREATE TABLE ErgebnisTabelle <(Dateioptionen)> AS
    SELECT ...
      FROM Tabelle <(Dateioptionen)>...;
QUIT;
```

Tabelle 1: Beispieldaten für die PROC SQL Abfragen

name	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	B1	B2	B3	B4	B5
Louise	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
John	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Alice	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45

1.2 DROP und KEEP

In SQL werden mit dem Asterix „*" als Abkürzung sämtliche Spalten/Variablen einer Tabelle (oder auch mehrerer Tabellen) ausgegeben. Es können aber auch nur ein paar Variablen ausgewählt werden. Im Gegensatz zum SAS Datenschritt werden dann die einzelnen Spalten mit Kommata aufgezählt und dann auch in dieser Reihenfolge ausgegeben. Durch die Verwendung der Dateioptionen KEEP und DROP kann gezielt spezifiziert werden welche Variablen verarbeitet werden sollen und welche nicht. Zusätzlich kann mit weiteren Kürzeln aus dem Datenschritt gearbeitet werden, wie zum Beispiel „-" oder „--" zur Spezifikation von Variablenlisten.

Beispiel 1: Einzelne Variablen nicht selektieren
Werden alle Variablen bis auf wenige benötigt (hier: Variable name nicht benötigt), so kann statt die gewünschten Variablen alle einzeln aufzulisten mit „*" und der Dateioption DROP gearbeitet werden.

```
PROC SQL;
  SELECT *
    FROM tab(DROP=name);
QUIT;
```

Beispiel 2: Mehrere Variablen selektieren

Möchte man eine Liste an Variablen auswählen und haben diese einen gemeinsamen Präfix und fortlaufenden numerischen Suffix (z.B. A1, A2 etc.), so kann mit KEEP und „-“ eine Liste spezifiziert werden. Ohne die Dateioption KEEP müssten alle Variablen einzeln aufgelistet werden.

```
PROC SQL;
  SELECT *
    FROM tab(KEEP=A1-A10);
QUIT;
```

Mit „--“ werden nur Variablen ausgewählt, die zwischen A6 und B4 im Datensatz tab liegen.

```
PROC SQL;
  SELECT *
    FROM tab(KEEP= A6--B4);
QUIT;
```

Mit „:“ können Variabeln mit einem bestimmten Präfix angegeben werden.

```
PROC SQL;
  SELECT *
    FROM tab(KEEP= A:);
QUIT;
```

Beispiel 3: Beobachtungen selektieren
Sollen basierend auf einer Variablen A1 Beobachtungen ausgewählt werden, die Variable aber selbst nicht mit in der neuen Tabelle kommen, so kann DROP erst innerhalb von CREATE TABLE spezifiziert werden, denn sonst ist die Spalte nicht für mehr die WHERE Anweisung vorhanden.

```
PROC SQL;
CREATE TABLE tab2(DROP=A1) AS
  SELECT *
    FROM tab WHERE A1=11;

/*funktioniert nicht*/
CREATE TABLE tab2 AS
  SELECT *
    FROM tab(DROP=A1)
    WHERE A1=11;
QUIT;
```

Hinweise:

- Das Selektieren von Variablenlisten funktioniert nur als Dateioption: SELECT A1-A5 FROM... würde eine neue Variable erzeugen mit der Differenz der beiden Variablen A1 und A5.

- WHERE wird nach FROM aber vor CREATE TABLE ausgewertet.

1.3 RENAME

Beispiel 4: Variablen umbenennen
Im Fall von Variablen mit gemeinsamem Präfix und numerischem Suffix lassen sich mit der RENAME Dateioption ganze Variablenlisten einfach umbenennen.

```
PROC SQL;
  SELECT *
    FROM tab (RENAME=(A1-A5 = Neu_A1-Neu_A5) KEEP=A1-A5);

/*statt*/
  SELECT A1 AS Neu_A1,A2 AS Neu_A2, ... ,A5 AS Neu_A5
    FROM tab;
QUIT;
```

Hinweis:
KEEP wird vor RENAME ausgeführt

1.4 SORTEDBY

Beispiel 5: Unnötiges Sortieren vermeiden
Mit dieser Dateioption kann angegeben werden, dass ein Datensatz bereits nach der angegebenen Variable sortiert ist und somit z.B. unnötiges Sortieren vermieden werden. Manchmal sind Datensätze bereits sortiert, wenn diese z.B. sortiert vorlagen als sie eingelesen wurden, ohne dass diese Information in den Metadaten der Tabelle vermerkt ist. Wird nun ein JOIN durchgeführt, so kann SAS einen SORT MERGE JOIN als Verknüpfungsalgorithmus durchführen (dann nämlich wenn die Datensätze zu groß sind für einen HASH JOIN oder es liegt kein passender Index vor für einen INDEX JOIN). Bei SORT MERGE JOIN werden zunächst die beteiligten Tabellen nach der Verknüpfungsvariable sortiert. Liegen eine oder mehrere Tabellen bereits sortiert vor, so kann dies mit SORTEDBY angegeben werden und damit unnötiges Sortieren vermieden werden. Mit der undokumentierten Anweisungsoption _METHOD wird der gewählte Algorithmus angezeigt und auch welche Datensätze sortiert wurden.

```
PROC SQL _METHOD;
  SELECT tab.A1,tab2.B1
    FROM tab (SORTEDBY=A1) INNER JOIN tab2
    ON tab.A1 = tab2.B1;
QUIT;
```

Im Log-Fenster wird folgender Hinweis angezeigt:

```
NOTE: SQL execution methods chosen are:
     sqxslct
          sqxjm
               sqxsort
```

```
                    sqxsrc( WORK.TAB2 )
                sqxsrc( WORK.TAB )
```

sqxjm gibt an, dass ein SORT MERGE JOIN durchgeführt wurde und es wurde nur die Tabelle tab2 sortiert (sqxsort sqxsrc(WORK.TAB2)).

Hinweis:
Mit PROC CONTENTS kann man sich anzeigen lassen, ob eine Tabelle sortiert ist und nach welcher Variablen.

1.5 WHERE

Beispiel 6: Statt Inline View Dateioption WHERE
Statt einer Inline View kann auch WHERE als Dateioption benutzt werden.

```
PROC SQL;
  SELECT tab.A1
    FROM tab INNER JOIN (SELECT * FROM tab WHERE A1=11) AS tab2
    ON tab.A1 = tab2.B1;
QUIT;

PROC SQL;
  SELECT tab.A1
    FROM tab INNER JOIN tab(WHERE=(A1=11)) AS tab2
    ON tab.A1 = tab2.B1;
QUIT;
```

Hinweise:

- Weitere Optionen sind z.B. FIRSTOBS,OBS, IDXWHERE, IDXNAME etc.
- Nicht alle Dateioptionen können mit SQL generierten Views innerhalb von PROC SQL (im FROM oder CREATE VIEW Teil) verwendet werden.

Literatur

[1] Kenneth W. Borowiak. Using data set options in proc sql. In SUGI31, 2006.

2 Verschachtelte SQL-Abfragen

(Heribert Ramroth)

Durch die Verwendung verschachtelter SQL Abfragen können Datensätze in bestimmten Situationen elegant aus einer Tabelle herausgefiltert werden.

2.1 Ausgangsituation

Gegeben seien Daten im sogenannten Phasenkonzept, d.h. für jede Person, z.B. mit der Variablen ID identifiziert, seien Datenzeilen aus mehreren Phasen vorhanden. Diese

Situation tritt zum Beispiel bei der Erhebung von Krankenhausaufenthalten von Patienten auf, beim Erheben von Rauchphasen oder bei wiederholten Einkäufen eines Kunden. Das Beispiel in Tabelle 1 zeigt Daten von Rauchphasen mehrerer Studienteilnehmer.
Angenommen, das Interesse gilt all den Studienteilnehmern, die mindestens 3 Rauchphasen haben, hier also den Teilnehmern mit den IDs 3 und 5. Ziel ist es, die gesamte Rauchhistorie der Teilnehmer darzustellen, nicht nur die Informationen der Phasen größer 3. Letzteres wäre im Datenschritt mit der Bedingung (if Phase>3) leicht möglich. Abbildung der gesamten Rauchhistorie (= im Beispiel auch der Phasen 1 und 2 dieser beiden Teilnehmer) erfordert bei alleiniger Verwendung von Daten- und Prozedurschritten mehrere Durchgänge, z.B. Kombination <Mergen> mit den Originaldaten. Dies lässt sich leichter mit Hilfe von PROC SQL lösen.

Tabelle 1: Rauchphasen in einer Studie

	ID	Phase	Beginn	Ende	Zig
	1	1	54	57	20
	1	2	57	98	60
	2	1	48	91	20
*	3	1	58	68	20
*	3	2	68	94	40
*	3	3	94	98	20
*	3	4	68	75	25
	4	1	76	82	25
*	5	1	95	98	6
*	5	2	42	52	5
*	5	3	59	68	15

Die Datensätze mit * sollen ausgewählt werden.

2.2 Lösung mit PROC SQL

Verwendung zweier verschachtelter `Select`-Anweisungen. Die innere `Select`-Anweisung selektiert die IDs aller Datensätze, deren Phasennummer größer oder gleich 3 ist. Diese Liste der IDs ist Auswahlkriterium für die äußere `Select`-Anweisung. Somit werden auch doppelte IDs aufgelistet.

```
PROC SQL;
     SELECT *
     FROM Rauchdaten
     WHERE ID in
          (SELECT ID
           FROM Rauchdaten
```

```
        WHERE Phase >= 3)
    ORDER BY ID, Beginn; QUIT;
```

3 Austausch von Variablenlisten zwischen PROC SQL und SAS-Prozeduren/Datasteps

(Heribert Ramroth)

Werden Variablenlisten im Makro eingegeben, so mag die Situation auftreten, dass ein Ergebnis leichter mit einer SQL-Anweisung erreicht werden kann. Allgemeiner ausgedrückt, die gleichen Variablenlisten sollen sowohl in Prozeduren als auch in PROC SQL verwendet werden. Die Syntax von PROC SQL erfordert jedoch die Trennung der Variablen durch Kommata, im Gegensatz zum allgemeinen Prozedurschritt.

```
proc print data=sashelp.class ;
     var Age Height Weight ;

proc corr data=sashelp.class ;
     var Age Height Weight ;

proc sql;
     select Age, Height, Weight from sashelp.class;
quit;
```

Die Liste der Variablen, die hier zum Beispiel über `%LET` übergeben wird, lässt sich in PROC SQL also nicht verwenden.

```
%LET VarList = Age Height Weight ;
```

3.1 Lösung 1

Durch Anwendung der Funktion %sysfunc lassen sich jedoch normale SAS Prozeduren auch im Makro-Zusammenhang anwenden. Die Funktion translate ersetzt im folgenden Schritt die Leerzeichen zwischen den Variablen durch Kommata, %sysfunc ermöglicht die Anwendung ausserhalb des Datenschrittes.

```
%LET VarList = Age Height Weight ;
%LET SqlVarList=%sysfunc(translate(&VarList.,", ", " "));

proc print data=sashelp.class ;
     var  &VarList.;
proc corr data=sashelp.class ;
     var  &VarList.;
proc sql;
     select &SqlVarList. from sashelp.class;
```

```
quit;
```

3.2 Lösung 2

Durch Anwendung eines Makros aus der SAS Hilfe lässt sich ein Text-String schnell in seine Einzelteile zerlegen.

```
...
%LET Nr = 0;
%DO %WHILE(%SCAN(&VarList,&Nr.+1,%STR( ))^=);
    %LET Nr = %EVAL (&Nr.+1);
    %LET VarNr&Nr.=%SCAN(&VarList,&Nr.,%STR( ));
%END;
...
```

Dies kann zum Beispiel im Falle der Berechnung roher und adjustierter Odds Ratios nötig sein. Der obige Code erzeugt die Variablen:

VarNr1 = Age / VarNr2 = Height / VarNr3 = Weight

Wenn man die Variablenliste nun sowieso schon zerlegt vorliegen hat, kann man diese auch auf einfache Art wieder mit neuem Trennzeichen zusammensetzen:

```
%MACRO VarListKomma;
     &VarNr1. %DO I=2 %TO &Nr.;, &&VarNr&I. %END;
%MEND VarListKomma;
%PUT SqlListe = %VarListKomma;
```

Somit sieht die Makro-Variable SqlListe wie folgt aus: Age, Height, Weight

4 PROC PRINT – Einfügen einer Leerzeile nach jeder x.ten Beobachtung

(Carina Ortseifen)

Die Ergebnisse der Prozedur PRINT sind bei großen Datenmengen nicht allzu übersichtlich. Etwas lesbarer wird die Ausgabe, wenn nach jeder 5., 10. oder allgemein: x-ten Beobachtung eine Leerzeile eingefügt wird. Damit wird die Tabelle aufgebrochen und für das Auge werden Strukturen sichtbar. Am Beispiel der SAS-Tabelle SASHELP.CLASS könnte das wie folgt aussehen:

```
SAS-Tabelle SASHELP.CLASS
  Obs     Name       Sex     Age     Height     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
    4     Carol        F       14       62.8       102.5
    5     Henry        M       14       63.5       102.5
```

```
 6    James       M      12     57.3       83.0
 7    Jane        F      12     59.8       84.5
 8    Janet       F      15     62.5      112.5
 9    Jeffrey     M      13     62.5       84.0
10    John        M      12     59.0       99.5

11    Joyce       F      11     51.3       50.5
12    Judy        F      14     64.3       90.0
..    ....
```

4.1 Der „steinige“ Weg

Dieses Ziel kann mit dem im Folgenden beschriebenen Trick erreicht werden. Zunächst werden in die Tabelle künstliche Beobachtungen, komplett mit fehlenden Werten ausgefüllt, dort eingefügt, wo später Leerzeilen erscheinen sollen.

```
* Der steinige Weg;
Data class_blanks;
     Set sashelp.class;
     Output;  /* Output real observation */
     If Mod(_n_,5)=0;
     Array Allnums {*} _Numeric_ ;
     Array Allchar {*} _Character_ ;
     Drop i;
     Do i=1 To Dim(allnums); allnums{i}=.; End;
     Do i=1 To Dim(allchar); allchar{i}=' '; End;
     Output;  /* Output blank observation */
Run;
```

Die erste OUTPUT-Anweisung überträgt die vorhandenen Beobachtungen in die neue Tabelle. Zusätzlich wird nach jeder 5. Beobachtung, berechnet mittels MOD(_N_,5)=0, mit einer zweiten OUTPUT Anweisung eine zusätzliche Beobachtung eingefügt.
Die beiden Arrays und die DO-Schleife sorgen dafür, dass die numerischen und die Textvariablen mit fehlende Werte besetzt werden.

Damit die fehlenden numerischen Werte bei der anschließenden Ausgabe mit der Prozedur PRINT nicht auffallen, wird die Systemoption MISSING= verwendet.

```
Options Missing=' '; /* Display numeric missing as blank */
Title 'SAS-Tabelle SASHELP.CLASS - Der steinige Weg';
Proc Print Data=class_blanks Noobs;
Run;
```

Da zusätzliche Beobachtungen eingefügt wurden, stimmt die Spalte OBS nicht mehr mit den ursprünglichen Beobachtungszahlen überein und muss daher mittels der Option NOOBS auf jeden Fall unterdrückt werden.
Wird dies alles beachtet, erzielt man das gewünschte Ergebnis. Ein wahrlich „steiniger Weg“ zum Ziel, der ab SAS 9.2 viel eleganter beschritten werden kann.

4.2 Der „elegante“ Weg mit SAS 9.2

Seit SAS 9.2 gibt es für dieses Problem eine sehr viel elegantere Lösung, die über eine einzige Option der PROC PRINT-Anweisung realisiert werden kann: BLANKLINE=.

```
* Die elegante Lösung;
Options Nodate Nonumber;
Title "SAS-Tabelle SASHELP.CLASS - Der elegantere Weg";
Proc Print Data=sashelp.class Blankline=5;
Run;
```

4.3 Unregelmäßige Leerzeilen

Die beiden vorgestellten Lösungen gehen davon aus, dass die Leerzeilen immer nach der gleichen Anzahl von Beobachtungen eingefügt werden sollen. Wie geht man aber vor, wenn die Anzahl nicht gleich ist, sondern unregelmäßige Abstände notwendig sind?

Die Option BLANKLINE= hilft da leider nicht weiter. Hier braucht man den (steinigen) Weg mit einem Datenschritt. Entweder mit den Steuervariablen FIRST.var/LAST.var und BY-Verarbeitung oder einem Array wie im folgenden Beispiel.

```
Options Missing = " ";
Data _null_;
  Attrib n length=8 format=4. label="Nobs";
  Set Sashelp.Class;
  n = _n_;
  Array breaks (3) _temporary_ (3,5,12);
  File Print Ods;
  Put _ods_;
  If ( n In breaks ) Then Put;
Run;
```

Das Array breaks enthält die Stellen für die Leerzeilen, also nach der 3., 5. und 12. Beobachtung. Dank ODS sieht das ganze dann so aus, als wäre es mit der Prozedur PRINT erzeugt:

Nobs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90
13	Louise	F	12	56.3	77
14	Mary	F	15	66.5	112
15	Philip	M	16	72	150
16	Robert	M	12	64.8	128
17	Ronald	M	15	67	133
18	Thomas	M	11	57.5	85
19	William	M	15	66.5	112

Literatur

[1] Support-Website von SAS Institute: http://support.sas.com/kb/31/366.html

5 Vergleich von SAS Tabellen – Suchen nach gemeinsamen Tabellen

(Carina Ortseifen)

Wer häufig mit mehreren Tabellen oder Generationen von einer Tabelle arbeitet, steht oft vor der Aufgabe, diese Tabellen vergleichen zu müssen. Insbesondere interessierte uns beim folgenden Tipp, welche gemeinsamen Variablen in den Tabellen vorhanden sind. Zur Bewältigung der Fragestellung kommen zum Einsatz:

- PROC COMPARE
- PROC DATASETS (bzw. PROC CONTENTS) und Datenschritt-Logik
- Dictionary Tables (mit PROC SQL)

Gezeigt wird das ganze anhand der folgenden drei Beispieltabellen:

```
Data eins;
   Input x y z;
   Datalines;
1 2 3
;
Data zwei;
   Input x a y;
   Datalines;
1 1 1
;

* Der Name der Variable X ist groß geschrieben! *;
Data drei;
   X=1;
Run;
```

5.1 Die Prozedur COMPARE

Mit Hilfe der Prozedur COMPARE können zwei SAS-Tabellen – und nur genau zwei - miteinander verglichen werden. Für unseren Zweck genügt folgender Prozedurschritt:

```
Proc Compare Base=eins Compare=zwei Novalues;
Run;
```

Im Abschnitt „Variables Summary“ gibt die Prozedur u. a. die Anzahl der gemeinsamen Variablen ausgibt:

```
Number of Variables in Common: 2.
```

Die Prozedur liefert mir die Anzahlen, jedoch nicht die Namen der gemeinsamen Variablen.

5.2 Die Prozedur DATASETS mit Datenschritt-Logik

Mit der Prozedur DATASETS und der CONTENTS-Anweisung können die gemeinsamen Variablen extrahiert werden und in einem anschließenden Datenschritt mit MERGE, IN= und OUTPUT ausgelesen werden.

Damit das große X der Tabelle drei nicht „stört“ (Der Anwender kann Groß- und Kleinbuchstaben verwenden, aber x und X bleibt das gleiche, nur nicht in unseren Vergleichen.), werden mit der Systemoption VALIDVARNAME alle Variablennamen in Großbuchstaben ausgegeben:

```
Options Validvarname = Upcase;
```

Die Prozedur DATASETS greift auf die Bibliothek WORK zu. Die Anweisung CONTENTS kann mehrfach gesetzt werden und trägt mit Hilfe der Option OUT= die Variablennamen in neue Tabellen ein:

```
Proc Datasets Lib=work;
   Contents Data=eins Noprint Out=out1(Keep=name);
   Contents Data=zwei Noprint Out=out2(Keep=name);
   Contents Data=drei Noprint Out=out3(Keep=name);
Quit;
```

Die Option NOPRINT unterdrückt die Prozedurausgabe im Output-Fenster.

Die drei neuen Tabellen Out1, Out2 und Out3 werden im anschließenden Datenschritt bezüglich der Variablennamen mit MERGE verknüpft. Mittels der Dateioptionen IN= gelingt es, dann die Beobachtungen, also Variablennamen, herauszufiltern, die in allen drei Tabellen auftreten.

```
Data gemeinsam;
   Merge out1(in=a) out2(in=b) out3(in=c);
   By name;
   If a and b and c Then Output;
Run;

Title "Gemeinsame Variablen der Tabellen eins, zwei und drei";
Proc Print;
Run;
```

Und wir erhalten die gewünschte Ausgabe:

```
Gemeinsame Variablen der Tabellen eins, zwei und drei

       Obs    NAME
        1      X
```

5.3 Dictionary Tables

Der Vollständigkeit wegen sollen hier auch noch die Dictionary Tables erwähnt werden, mit deren Hilfe man die gesuchte Information natürlich auch erhält.

Mit einem Prozedur SQL-Schritt greift man auf die Dictionary.Columns zu, die die Variableninformation enthält, beschränkt sich auf die drei Ausgangstabellen und gruppiert und addiert:

```
Proc SQL;
 * Describe Table Dictionary.Columns;
   Create Table gemeinsamSQL (Drop=anz) As
      Select name,count(*) As anz
      From (
         Select libname, memname, name
         From Dictionary.Columns
         Where libname='WORK' And memname In ('EINS','ZWEI','DREI'))
      Group By name
      Having anz = 3
   ;
Quit;
```

Ein anschließender PROC PRINT-Schritt liefert das identische Ergebnis zur Prozedur DATASETS-Variante.

Literatur

[1] Support-Website von SAS Institute: http://support.sas.com/kb/38/059.html

6 Orientierung in der SAS-Umgebung – Pfadangaben zu Bibliotheken, Formatkatalogen und Programmen

(Carina Ortseifen)

Der Programmierer steht häufiger mal vor dem Problem, Pfadangaben dynamisch bestimmen zu müssen. Oder eine Ausgabe soll mit dem Namen des Programms versehen werden, von dem es erzeugt wurde. Beim interaktiven Arbeiten ist dies kein Problem, nicht so im Batchmodus. Im nun folgenden Tipp werden einige Möglichkeiten dazu vorgestellt.

6.1 Bibliotheken und Datei-Referenzen

Die Funktion PATHNAME() liefert den vollständigen Pfad von Bibliotheken und Datei-Referenzen. Im offenen Programmcode kann diese Funktion mit der Makrofunktion %SYSFUNC genutzt werden.

Im folgenden Beispiel wird zunächst die Datei-Referenz saspgm definiert und dann anschließend mit PATHNAME() aufgelöst und der Makrovariablen pgmpfad zugewiesen:

```
Filename saspgm "E:\ksfe-14\TuT\TuT-CO3.sas";

%Let pgmpfad=%Sysfunc(Pathname(saspgm));
%Put &pgmpfad;
```

Den physikalischen Ordner der Work-Bibliothek liefern folgende Zeilen:

```
* Bibliotheken;
%Let workpfad=%Sysfunc(Pathname(Work));
%Put &workpfad;
```

Im Log-Fenster erscheint daraufhin:

```
C:\DOKUME~1\x16.AD\LOKALE~1\Temp\SAS Temporary Files\_TD152
```

Das ganze funktioniert auch bei verbundenen Bibliotheken wie etwa SASAUTOS:

```
* Verbundene Bibliotheken;
%Let sasautospfad=%Sysfunc(Pathname(sasautos));
%Put &sasautospfad;

('C:\Programme\SAS\SASFoundation\9.2\core\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\accelmva\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\inttech\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\assist\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\eis\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\ets\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\genetics\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\graph\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\iml\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\or\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\qc\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\share\sasmacro'
'C:\Programme\SAS\SASFoundation\9.2\stat\sasmacro')
```

Alternativ können zu diesem Zweck auch die beiden Dictionary Tables Dictionary.Libnames und Dictionary.Extfiles verwendet werden. Neben den Pfadangaben werden dort weitere Informationen, z.B. die Engines gespeichert, wie das folgende Beispiel zeigt:

```
Proc SQL Noprint;
 * Describe Table Dictionary.Libnames;
   Select path, engine into :workpfadsql,:workengine
      From Dictionary.Libnames
          Where libname="WORK";
Quit;

%Put &workpfadsql;
```

```
%Put &workengine;
```

Im Log-Fenster erscheint:

```
12   %Put &workpfadsql;
C:\DOKUME~1\x16.AD\LOKALE~1\Temp\SAS Temporary Files\_TD152
13   %Put &workengine;
V9
```

Und auch Dateireferenzen können abgefragt werden:

```
Proc SQL Noprint;
 * Describe Table Dictionary.Extfiles;
   Select xpath into :pgmpfadsql
      From Dictionary.Extfiles
          Where Lowcase(fileref)="saspgm";
Quit;

%Put &pgmpfadsql;
```

Im Log erscheint:

```
E:\ksfe-14\TuT\TuT-CO3.sas
```

6.2 Formatkataloge

Formate werden vom SAS System in Katalogen namens Formats.Sas7bdat. Das System kann mehrere Kataloge in verschiedenen Ordnern (oder Bibliotheken) verwalten. Wird ein Format im Daten- oder Prozedurschritt verwendet, durchsucht das System in einer festgelegten Reihenfolge die vorhandenen Formatkataloge nach dem gewünschten Format.
Die Reihenfolge, in der die Kataloge durchsucht werden, legt die Systemoption FMTSEARCH fest. Die Einstellung dieser Option kann mit der Funktion GETOPTION() abgefragt werden.
Die folgenden Zeilen

```
%Let formatpfad=%Sysfunc(getoption(fmtsearch));
%Put &formatpfad;
```

liefern die zu durchsuchenden Bibliotheken:

```
(WORK LIBRARY)
```

Kombiniert man die Funktion PATHNAME() mit den Ergebnissen der Option FMTSEARCH erhält man die kompletten Pfade.

6.3 Name des ausführenden Programms

Ergebnisse von Prozedurschritten, Tabellen und/oder Grafiken, müssen häufig mit dem Namen des Programms, das sie generiert hat, beschriftet werden, um bei notwendigen Änderungen nicht lange suchen zu müssen oder um eine sorgfältige Dokumentation zu erhalten.
Wenn man im Batchmodus arbeitet, steht mit der Systemoption SYSIN in Kombination mit der Funktion GETOPTION() eine einfache Methode zur Verfügung, dies zu erreichen.
Arbeitet man dagegen interaktiv, z.B. mit SAS unter Windows, liefert SYSIN keine Informationen. Stattdessen greift man auf Umgebungsvariablen des Betriebssystems zu. Unter Windows heißen diese SAS_ExecFileName und SAS_ExecFilePath. Diese Variablen können mit der Funktion %SYSGET() abgefragt werden, wie das folgende Beispiel zeigt:

```
%Let saspgm=%Sysget(sas_execfilepath);
Title SAS-Tabelle sashelp.class;
Footnote "Erzeugt mit &saspgm";
Proc Print Data=sashelp.class;
Run;
```

Die Fußzeile der Ausgabe hat folgende Form:

```
Erzeugt mit \\netfile1...\home\x\x16\KSFE 2010\TuT-CO3.sas
```

Literatur

[1] Arthur L. Carpenter: The Path, The Whole Path, And Nothing But the Path, So Help Me Windows. SAS Global Forum 2008.
http://www2.sas.com/proceedings/forum2008/023-2008.pdf

7 Farbangaben für SAS/Graph – leichtgemacht

(Grischa Pfister)

Farbangaben in SAS/GRAPH sind sehr einfach vorzunehmen, wenn bekannt ist, wie die wichtigsten Farb-Mischsysteme in SAS umgesetzt sind. Da dank des ODS auch hier immer mehr Farbe ins Spiel kommt, ist dieser Beitrag nicht nur für eingefleischte SAS/GRAPH-Fans interessant.

Farbangaben werden in SAS an verschiedenen Stellen benötigt, klassischerweise natürlich in SAS/GRAPH, seit Einführung des ODS wird aber auch die normale SAS-Ausgabe zunehmend bunter.

Im Falle der Grafiken werden Farbangaben unter anderen benötigt für Text, Linien, Symbole und Füllfarben. Dabei gilt es oft, Vorgaben von Kunden oder firmeninterne Stylesheets umzusetzen. Die Frage ist, wie die vorhandenen Angaben in SAS-Farbsyntax übersetzt werden können.
Zunächst einmal bietet SAS eine Reihe vordefinierter Farben an, z.B. „red“, „green“, „blue“, die allerdings vor Version 9.2 nur in der Dokumentation zu finden war. Seit 9.2 wird in der SAS-Registry eine Liste mit den vordefinierten Farben gepflegt. Folgendes Programm zeigt die Liste an:

```
* GP Registrierte Farben *;
Proc Registry
  startat="COLORNAMES\HTML"
  list
 ;
Run;
```

Welche Farbe tatsächlich hinter so klingenden Namen wie „oldLace“, „PaleTurquoise“ oder „SaddleBrown“ steckt, lässt sich immer noch am besten durch Ausprobieren herausfinden.

Eine weitere Möglichkeit der Farbdefinition ist das SAS Color Naming Scheme (CNS), das eine Grundfarbe durch zusätzliche Angabe von Sättigung, Licht und eventuell einer zweiten Farbkomponente modifiziert. Die so entstehenden Farb-Namen werden entweder einfach zusammengeschrieben „darkGreyischGreen“, mit „_“ getrennt „light_vivid_purple“ oder durch Leerzeichen getrennt und in Anführungszeichen gesetzt: „medium strong red“.

In der Praxis findet sich allerdings vorrangig die Farbzusammenstellung mit Hilfe des RGB-Modells. Hier wird die Farbe durch Angabe von Rot-, Grün- und Blau-Komponenten gemischt. Dieses an sich einfache Modell wird in der Syntax aus zwei Gründen etwas erschwert. Zunächst erfolgen die Angaben im Wertebereich von 0 bis 255 statt in Prozent, dann muss die Angabe der Werte in Hexadezimal-Darstellung erfolgen. SAS erlaubt nämlich für den Namen einer RGB-Farbangabe genau acht Zeichen, die sich aus dem Marker „CX“, gefolgt von den Rot-, Grün- und Blauwerten zusammensetzen, also „CXrrggbb“, für Rot, Grün und Blau bleiben also jeweils zwei Stellen – deshalb die Darstellung in Hexadezimal. Die benötigten Werte können mit Hilfe des Wissenschaftlichen Taschenrechners von Windows genauso ermittelt werden, wie mit den üblichen Office-Programmen (z.B. in Word: Schriftfarbe → Weitere Farben → Benutzerdefiniert) und natürlich im Kopf gerechnet werden. ;-)

SAS selbst bietet eine Reihe von Hilfsmakros an, die die Nutzung der verschiedenen Farb-Systeme erleichtern soll (neben den hier vorgestellten gibt es auch noch das HLS-System). Diese Makros müssen vor der ersten Verwendung kompiliert werden, was durch Aufruf des Makros %Colormac erfolgt. Anschließend steht z.B. das Makro %RGB zur Verfügung, das als Parameter die Werte für Rot, Grün und Blau in Prozent erhält und dann eine entsprechende Farbangabe zurückliefert. Der Aufruf von:

```
%Colormac;
%Put farbe ist: %rgb(80,20,0), %rgb(0,80,20), %rgb(20,0,80);
```

zeigt im LOG das folgende Ergebnis:

```
…
*** Color Utility macros are now available ***

 For further information on the Color Utility macros
 enter:
    %HELPCLR(macroname) - Specific macro information
    %HELPCLR(ALL)       - All macro information
    %HELPCLR            - List of macro names

2    %Put farbe ist: %rgb(80,20,0), %rgb(0,80,20), %rgb(20,0,80);
farbe ist: CXCC3300, CX00CC33, CX3300C
```

Eine typische Verwendung im Pattern-Statement sieht dann so aus:

```
Pattern1 v=s c = %rgb(80,20,0);
Pattern2 v=s c = %rgb(0,80,20);
Pattern3 v=s c = %rgb(20,0,80);
```

Da Angaben in Prozent aber eher unüblich sind, sondern oft in den Vorgaben die „echten“ Werte von 0 bis 255 enthalten sind, lohnt es sich, ein eigenes Hilfsmakro zu schreiben, das diese Werte als Eingabe akzeptiert und dann wieder die passende Farbangabe zurückgibt. Ein solches Makro ist auch erstaunlich einfach zu schreiben, die numerischen Werte für Rot, Grün, Blau müssen lediglich mit dem Format HEX2. versehen werden und dann zu einer Zeichenkette zusammengesetzt werden. Da in der Makrosprache sowieso nur Text existiert, sieht das Hilfsmakro dann z.B. so aus:

```
%Macro gpRgb(r,g,b);
  %Local color;
  %Let color = cx%Sysfunc(putn(&r,hex2.))_
               %Sysfunc(putn(&g,hex2.))_
               %Sysfunc(putn(&b,hex2.));
  &color
%Mend;
```

(Achtung: das “_” zeigt Zeilenumbrüche an, die hier nur aus Lesbarkeitsgründen eingefügt sind, der Code gehört eigentlich in eine Zeile ohne Leerzeichen.)

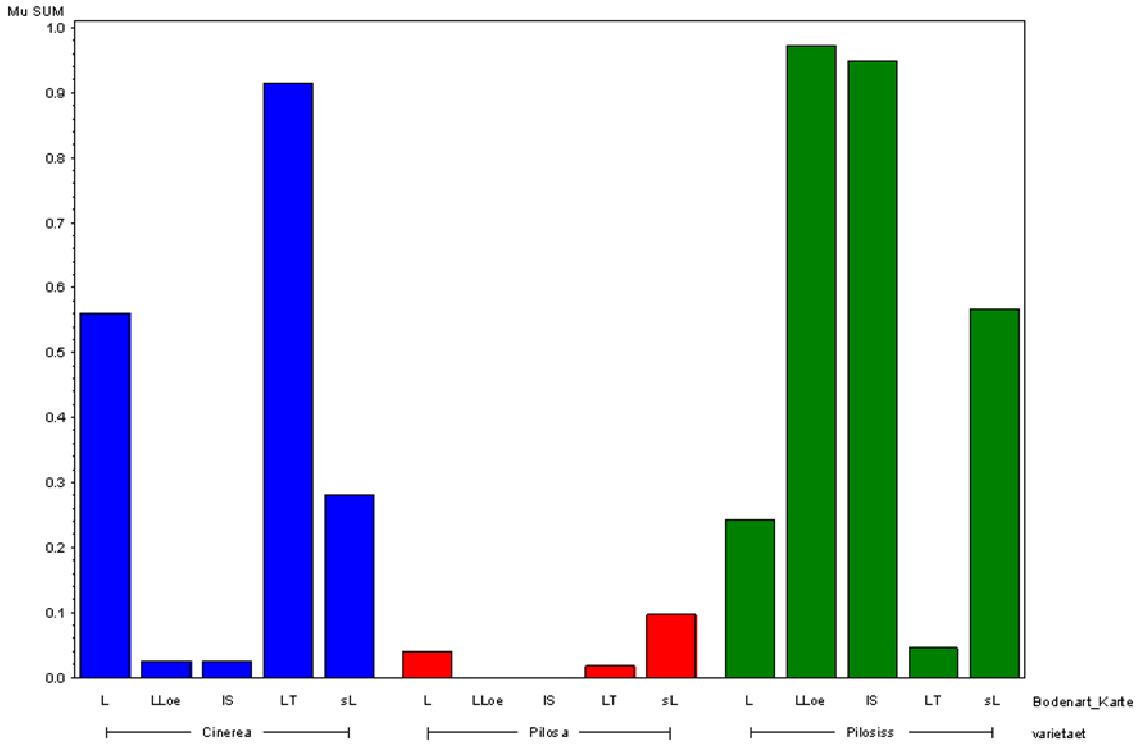
Wenn das Makro kompiliert wurde, lässt es sich genauso wie die SAS-Hilfsmakros verwenden:

```
Pattern1 v=s c = %gpRgb(165,210,170);
Pattern2 v=s c = %gpRgb(80,175,90);
Pattern3 v=s c = %gpRgb(55,115,60);
```

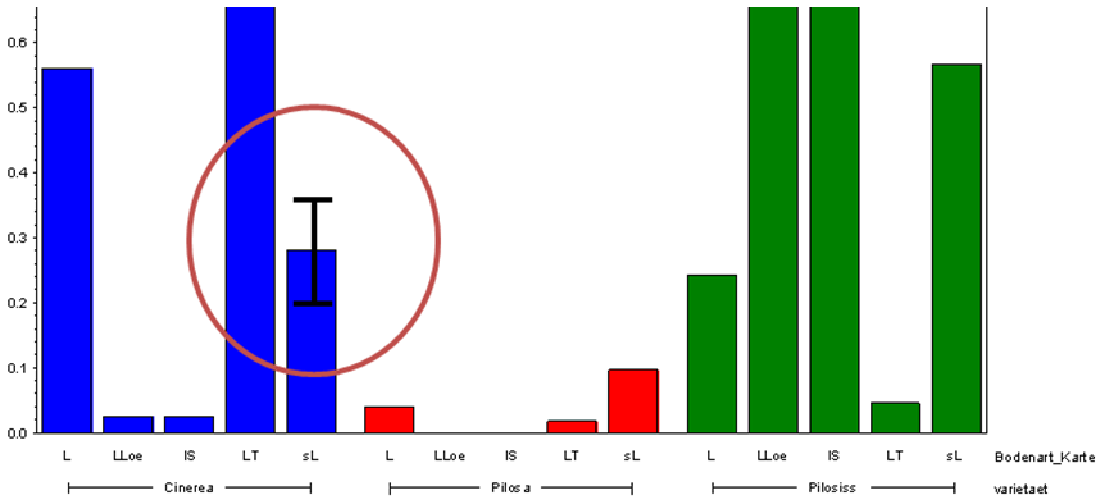
8 Praxis Beispiel Annotate – ein Balkendiagramm erweitern

(Grischa Pfister)

Ausgangspunkt ist ein Balkendiagramm, das einen statistischen Kennwert für verschiedene Pflanzen-Varietäten aufzeigt, die auf unterschiedlichen Bodentypen gezogen wurden.

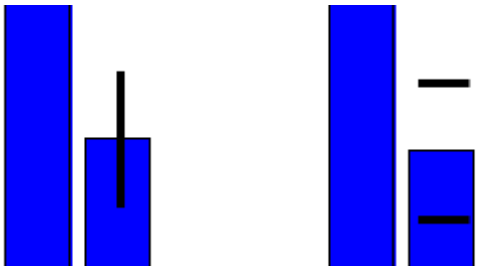


Zusätzlich zu den angezeigten Mittelwerten sollen für jeden Balken noch die Extremwerte dargestellt werden.



Das ist ein typisches Beispiel für den Einsatz der SAS/GRAPH Annotate Facility. Dieses Werkzeugt erlaubt die freie Grafikprogrammierung um entweder die Ausgabe der Standard-Prozeduren zu ergänzen, oder vollständig eigene Grafiken zu erstellen.
Die Funktionsweise ist dabei, dass eine SAS-Tabelle pro Zeile jeweils eine Grafik-Steueranweisung enthält, die einen gedachten Cursor innerhalb der Grafik bewegt und an gegebenen Koordinaten Grafikelemente erzeugt. Die Tabelle steuert, welche Aktion (was) an welcher Stelle (wo) gezeichnet wird, und wie das jeweilige graphische Element aussieht (wie). Dazu muss die Tabelle vordefiniert Variablen (Name, Typ und eventuell Länge) enthalten, die was, wo und wie genau beschreiben.
Eine Annotate-Tabelle kann von Hand geschrieben werden, meistens wird sie jedoch aus den gleichen Daten gebildet, die auch für die eigentliche Grafik verwendet werden. Diese Ausgangstabelle wird eingelesen und mit Hilfe der typischen Data-Step Syntax - end=eof; If (_n_ = 1); By-Verarbeitung; Output-Anweisung – werden dann die benötigten Sätze in die Annotate-Tabelle geschrieben.

In dem konkreten Fall soll mittig über jedem Balken eine vertikale Linie den Minimalwert mit dem Maximum verbinden. Außerdem soll an Minimum und Maximum für die bessere Lesbarkeit noch eine horizontale Linie (Whisker) ergänzt werden.



Zunächst müssen die Koordinaten für die einzelnen Elemente ermittelt werden. Die Y-Werte sind schnell geklärt, denn die Ausgangstabelle enthält bereits die zwei Variablen LowerMu und UpperMu, die Minimum und Maximum für jeden Balken bereithalten.
Die X-Werte sind etwas spannender, denn erstens ist die X-Variable alphanumerisch und zweitens werden die Balken noch über eine GROUP-Option unterteilt. Für beides hält die Annotate Facility aber Lösungen bereit, denn für Diagramme mit alphanumerischen Koordinaten gibt es spezielle Variablen in der Annotate-Tabelle. Mit Hilfe einer GROUP-Variable wird die Gruppe angesteuert, XC enthält dann den alphanumerischen Wert, der den individuellen Balken innerhalb der Gruppe identifiziert.
Die vertikale Linie wird folgendermaßen gezeichnet: Mittels Annotate-Anweisungen (= Sätzen in der Annotate-Tabelle) wird der gedachte Cursor zunächst in einen Balken platziert (X-Koordinate) und zwar an die Y-Koordinate LowerMu. Dann wird eine Linie von LowerMu nach UpperMu gezeichnet.
Und die Whisker? Sie sollen als horizontale Linie an LowerMu/UpperMu erzeugt werden, die Frage ist aber, wie genau das passieren soll, denn es muss die X-Koordinate verändert werden. Die ist aber alphanumerisch – und da kann man schlecht um die Hälfte nach links oder rechts variieren. An dieser Stelle kommen die verschiedenen Bezugssysteme ins Spiel, die die Annotate Facility bietet.

X- und Y-Koordinaten müssen sich auf ein konkretes System beziehen, damit SAS/GRAPH die Koordinatenangaben korrekt interpretieren kann. Grundsätzlich werden absolute Bezugssysteme (jede Koordinatenangabe bezieht sich auf den Ursprung) und relative Bezugssysteme (jede Koordinatenangabe bezieht sich auf die Angabe zuvor und enthält nur die Änderung in Bezug auf diese Koordinate) unterschieden. Dann gibt es verschiedene Räume, auf die sich die Koordinaten beziehen können, den Gesamtraum der Grafik, inklusive dem Platz für Titel und Fußnoten, den sog. Prozedurbereich, das ist der Platz, den die Grafik ohne Titel und Fußnoten einnimmt und als drittes der Datenbereich, das ist der Teil, der z.B. durch die Achsen eines Balken- oder Liniendiagramms umfasst wird. In dem letztgenannten Bereich ist die Angabe von Koordinaten sehr einfach, da die Werte bereits in der Tabelle stecken, die für die Erstellung der eigentlichen Grafik verwendet wird. Alternativ können Werte aber auch immer in Prozent angegeben werden, sodass die Fragestellung wie folgt beantwortet werden kann.
Die Whisker werden gezeichnet, indem zunächst der (gedachte) Cursor an die X-Koordinate Group/XC bewegt wird, d.h. der Cursor steht jetzt in der Mitte des Balkens – entweder an LowerMu oder UpperMu. Dann wird für die X-Koordinate das Bezugssystem auf Prozent/relativ geändert und der Cursor um 1% nach links bewegt. Anschließend wird die X-Koordinate um 2% nach rechts bewegt und eine Linie gezeichnet.

Das vollständige Programm sieht dann so aus:

```
Data Work.Test_grafik;
Input  varietaet $ Bodenart_Karte $  Mu  LowerMu   UpperMu;
Cards;
   Cinerea      L               0.5611      0.4156      0.6967
   …
;
Run;

Pattern1 v=s c=blue;
Pattern2 v=s c=red;
Pattern3 v=s c=green;

* GP Aktivieren der Hilfs-Makros von Annotate *;
%Annomac;

%* GP Erzeugen der Annotate-Tabelle durch Einlesen
      der Daten-Tabelle *;
Data Work.Anno;
  * GP Deklarieren der Annotate-Variablen ueber Hilfs-Makro *;
  %Dclanno;
  * GP Einstellen des Bezugs-Systems auf Daten fuer X und Y auf
       Datenraum *;
  %System(2,2);
  Set Work.Test_grafik;
  * GP Setzen von Standard-Werten *;
  * GP Annotate-Anweisungen werden nach der Prozedurausgabe
       gezeichnet (after) *;
```

```
when = "A";
* GP Linientyp ist 1 (= durchgehende Linie) *;
line = 1;
* GP Linienstaerke ist 1 *;
size = 1;
* GP Linienfarbe ist Schwarz *;
color = "black";

* GP Angabe der X-Koordinaten mit Hilfe der Annotate-Variablen
     GROUP und XC *;
group = varietaet;
xc = bodenart_karte;

* GP Unterer Whisker *;
* GP Der Cursor wird auf die Y-Koordinate LowerMu gesetzt und die
     erste Beobachtung in die Annotate-Tabelle geschrieben *;
y = LowerMu;
function="move";
Output;

* GP Das Bezugssystem fuer X-Koordinaten wird auf Prozent/relative
     geaendert und der Cursor um ein Prozent nach links bewegt *;
xsys = "7";
x = -1;
Output;
* GP Jetzt wird von der vorherigen Koordinate ausgehend der Cursor
     um zwei Prozent nach rechts bewegt, dabei wird eine Linie
     gezeichnet *;
x = +2;
function="draw";
Output;

* GP Vertikale Linie LowerMu-UpperMu *;
* GP Das Bezugssytem fuer X-Koordinaten wird wieder auf Datenraum
     umgestellt und der Cursor wieder an die Position GROUP/XC
     platziert *;
xsys = "2";
function="move";
Output;
* GP die Linie von LowerMu nach UpperMu wird gezeichnet *;
y = UpperMu;
function="draw";
Output;

* GP Oberer Whisker analog zu unterem Whisker *;
function="move";
Output;
xsys = "7";
x = -1;
Output;
x = +2;
function="draw";
```

```
  Output;
Run;


* GP Erzeugen der Grafik inclusive Annotate *;
Proc Gchart data = Work.Test_grafik;
  Vbar  bodenart_karte/
    sumvar    = mu
    coutline  = black
    group     = varietaet
    patternid = group
    anno      = Work.Anno
  ;
  Run;
Quit;
```

Und hier ist das dazugehörige Ergebnis: