Tipps, Tricks und Stolperfallen zum Umwandeln von Variablen

Sabine Erbslöh
Accovion GmbH,
Abteilung Statistical Programming
Helfmann-Park 10
65760 Eschborn
sabine.erbsloeh@accovion.com

Zusammenfassung

Bei der Arbeit mit Daten aus klinischen Studien müssen die Rohdaten für eine Auswertung erst entsprechend aufbereitet werden. Dazu sind oft Umwandlungen nötig, z.B. von numerisch in alphanumerisch oder von mehreren einzelnen Variablen in eine Datumsvariable. Bei der Weiterbearbeitung erlebt man dann manche Überraschung, z.B. wenn man nicht daran gedacht hat, die Länge einer alphanumerischen Variablen vorher zu definieren, wenn die Rohdaten nicht immer so aussehen wie erwartet, oder wenn es durch Standardeinstellungen beim Umwandeln zu unerwarteten Leerzeichen oder gar zum Abschneiden von Zeichen kommt.
Oft gibt es auch ganz spezielle Vorgaben, die für einen Datensatz erfüllt sein sollen. Dann ist es hilfreich zu wissen, wie man beispielsweise alle Formate oder Label entfernt oder Variablen initialisiert, ohne sich Gedanken machen zu müssen, ob sie numerisch oder alphanumerisch sind. Auch die Reihenfolge der Variablen in einem Datensatz kann wichtig sein, sei es mit führenden Identifizierungsmerkmalen oder in alphabetischer Reihenfolge.
In diesem Beitrag werden Beispiele zur Umwandlung von Variablen unter Verwendung von Funktionen, Formaten und Prozeduren vorgestellt. Dabei wird speziell auf Stolperfallen eingegangen.

Schlüsselwörter: Format, Label, PROC DATASETS, ATTRIB, RETAIN, SELECT, Reihenfolge, PICTURE, Umwandeln, numerisch, alphanumerisch, Datum, Tag, Monat, Jahr, SCAN, NOTDIGIT, INDEX, INDEXC, REVERSE, TRANWRD

1 Darstellung der Variablen im Datensatz

Bei der Arbeit mit Daten aus klinischen Studien sind Attribute und Reihenfolge der Variablen im Datensatz meist genau definiert. Deshalb ist es hilfreich zu wissen, wie man Formate und Label entfernen kann und wie die Reihenfolge der Variablen im Datensatz beeinflusst werden kann.

1.1 Entfernen von Labels und Formaten

Das Entfernen von Labels und Formaten kann beim Erzeugen von Rohdatenlistings gewünscht sein, oder den Vergleich von Datensätzen (PROC COMPARE bei Doppelpro-

grammierung) erleichtern. Auch wenn beim Arbeiten mit Standards (z.B. CDISC) Variablen nicht mit einem Format belegt sein sollen, sondern in 2 Variablen als Code und Decode dargestellt werden, ist das Entfernen aller Formate hilfreich.

Beispiel: PROC CONTENTS für Datensatz cars aus sashelp

```
 #  Variable    Type  Len  Format   Label
 9  Cylinders   Num   8
 5  DriveTrain  Char  5
 8  EngineSize  Num   8             Engine Size (L)
10  Horsepower  Num   8
 7  Invoice     Num   8    DOLLAR8.
15  Length      Num   8             Length (IN)
...
```

Formate und Label sollen nun entfernt werden. Dafür gibt es verschiedene Möglichkeiten. Zwei davon werden hier vorgestellt.

1.1.1 PROC DATASETS mit MODIFY

Mit ATTRIB _all_ können sowohl Label als auch Formate entfernt werden.
Hier am Beispiel des Datensatzes cars, der zuvor in der library WORK gespeichert wurde:

```
PROC DATASETS lib=work memtype=data;
  MODIFY cars;
  ATTRIB _all_ label=' ';
  ATTRIB _all_ format=;
RUN;
```

Die Formate und Label stehen nun nicht mehr zur Verfügung und werden entsprechend im PROC CONTENTS nicht mehr angezeigt.

```
#   Variable    Type  Len
 9  Cylinders   Num   8
 5  DriveTrain  Char  5
 8  EngineSize  Num   8
10  Horsepower  Num   8
 7  Invoice     Num   8
15  Length      Num   8
```

1.1.2 Im Datenschritt mit FORMAT und ATTRIB für Label

Auch im Datenschritt können Formate und Label entfernt werden:

- FORMAT _ALL_ entfernt alle Formate
- INFORMAT _ALL_ entfernt alle Informate
- ATTRIB _ALL_ label='' entfernt alle Label

Am Beispiel des Datensatzes sashelp.cars sieht der SAS-Code so aus:

```
DATA cars;
   SET sashelp.cars;
   FORMAT _ALL_;
```

```
   INFORMAT _ALL_;
   ATTRIB _ALL_ label=' ';
RUN;
```

Bei der ATTRIB Anweisung ist zu beachten, dass innerhalb des Datenschritts nur für Label mit _ALL_ gearbeitet werden kann. Für Formate innerhalb des ATTRIB ist das nicht möglich, d.h. ATTRIB _ALL_ format= funktioniert nicht.

1.2 Reihenfolge und Initialisierung

Die Reihenfolge der Variablen im Datensatz kann wichtig sein, wenn es bestimmte Vorgaben in den Spezifikationen für die Erstellung von Analysedatensätzen gibt. Die gleiche Reihenfolge in Datensätzen zu haben hilft auch bei studienübergreifenden Vergleichen oder einfach beim Abgleich der Variablen im Datensatz gegen die Spezifikationen.

1.2.1 Reihenfolge von Variablen im Datensatz

In klinischen Studien ist es oft gewünscht, dass die Variablen, die Identifikationsmerkmale enthalten, ganz vorne stehen. Auch hierfür gibt es wieder verschiedene Möglichkeiten mit Vor- und Nachteilen.

Zunächst ein kleiner Ausflug in die Prozedur PROC CONTENTS. Die Reihenfolge der Variablen wird ohne weitere Optionen alphabetisch ausgegeben. Mit der Option POSITION wird jedoch die Reihenfolge im Datensatz angezeigt:

```
PROC contents DATA=cars POSITION; RUN;
```

ergibt folgenden output:

```
  #    Variable       Type    Len
  1    Make           Char     13
  2    Model          Char     40
  3    Type           Char      8
  4    Origin         Char      6
  5    DriveTrain     Char      5
  …
 15    Length         Num       8
```

1.2.1.1 Sortierung der Variablen über RETAIN

Die in RETAIN angegebenen Variablen werden nach vorne sortiert.

```
DATA cars_retain;
   RETAIN origin type make model horsepower
          drivetrain cylinders length;
   SET cars;
RUN;
```

```
   #    Variable       Type    Len
   1    origin         Char      6
   2    type           Char      8
```

```
   3    make           Char     13
   4    model          Char     40
   5    horsepower     Num       8
  …
  15    Wheelbase      Num       8
```

Der Vorteil dieser Methode ist, dass nur die Variablen im RETAIN angegeben werden müssen, die nach vorne sortiert werden sollen. Alle anderen Variablen bleiben erhalten. Nachteil dieser Methode können allerdings ungewollte Ersetzungen aus dem Vorgänger-Record sein.

1.2.1.2 Sortierung der Variablen über ATTRIB

Wenn aufgrund vorgegebener Spezifikationen sowieso mit ATTRIB gearbeitet wird, ist es sinnvoll die Variablen gleich in der gewünschten Reihenfolge zu definieren. Die im ATTRIB gewählte Reihenfolge ist dann auch die im Datensatz.

```
DATA cars_attrib;
  ATTRIB
    origin     length=$6   label='Herkunft'
    type       length=$8   label='Klasse'
    make       length=$13  label='Hersteller'
    model      length=$30  label='Modell'
    horsepower length=8    label='PS'         format=4.0
    ... ;
  SET cars;
RUN;
```

Der Vorteil ist eine übersichtliche Darstellung aller Attribute. Wie beim RETAIN müssen nicht alle Variablen aufgelistet werden, sondern nur die, für die eine Reihenfolge oder andere Attribut-Angabe gewünscht ist.

Nachteil ist hier, dass es zu WARNINGs kommen kann, wenn im eingelesenen Datensatz eine andere Variablenlänge definiert wurde, als in der ATTRIB Anweisung.

1.2.1.3 Sortierung der Variablen über PROC SQL

Die Variablen werden in der Reihenfolge des SELECT erzeugt.

```
PROC SQL;
  CREATE TABLE cars_sql AS
  SELECT origin, type, make, model, horsepower,
         drivetrain, cylinders, length,
         MSRP, invoice, enginesize, mpg_city,
         mpg_highway, weight, wheelbase
  FROM cars;
RUN;
```

Vorteil: schnell und einfach
Nachteil: alle Variablen müssen im SELECT angegeben werden

Der Vorteil von PROC SQL wird beim Mischen zweier Datensätze besonders deutlich. Hier ein Beispiel zum Mischen nach Studie und Patient aus zwei Datensätzen patident

und labor, wobei die Identifikationskriterien vorne stehen sollen und danach Variablen aus beiden Datensätzen in vorgegebener Reihenfolge kommen.

```
PROC SQL;
  CREATE TABLE patlab AS
  SELECT a.studie, a.patient, b.visite,
         a.mw, a.alter, a.groesse,
         b.gewicht, b.SBP, b.DBP, b.HR
  FROM patident a, labor b
  WHERE a.studie=b.studie and
        a.patient=b.patient;
RUN;
```

Im Datenschritt wären statt dieser einfachen Lösung mehrere Schritte sowie Sortierungen notwendig.

1.2.2 Initialisieren von Variablen im Datensatz

Wenn z.B. mit ATTRIB Variablenattribute definiert wurden, die Variableninhalte selbst aber erst in einem nachfolgenden Schritt definiert werden, kommt es zu störenden "uninitialized"-Meldungen im Log-Fenster. Um das zu umgehen, sollten die Variablen initialisiert, also mit leeren Werten vorbelegt werden. Hierfür werden zwei Methoden vorgestellt.

1.2.2.1 Vorbelegen von Variablen mit CALL MISSING

Mit CALL MISSING kann man in einem Aufruf numerische und alphanumerische Variablen initialisieren, ohne wissen zu müssen, welche numerisch oder alphanumerisch sind. Der Aufruf erfolgt im Datenschritt.

Im folgenden Beispiel werden die numerischen Variablen neunum1, neunum2 und neunum3 sowie die alphanumerische Variable neutxt initialisiert.

```
CALL MISSING (neunum1, neunum2, neunum3, neutxt);
```

Der Nachteil ist, dass alle Variablen einzeln aufgelistet werden müssen. Die Zusammenfassung zu neunum1-neunum3 ist nicht möglich.

1.2.2.2 Vorbelegen von Variablen mit ARRAY

Mit einer ARRAY-Zuweisung kann man einfach alle Variablen initialisieren und verhindert so eine "uninitialized" Meldung im Log-Fenster (oder der Log-Datei).

```
DATA labini;
  SET labor;
  ARRAY aanum  {*} _numeric_ ;
  ARRAY aachar {*} _character_ ;
RUN;
```

2 Formatieren mit PICTURE

Mit PICTURE werden Werte sozusagen in ein Bild gepresst. Zusätzlich zur Formatierung kann ein Präfix angegeben werden. Ein PICTURE wird wie ein FORMAT in PROC FORMAT definiert.

Syntax:

```
PROC FORMAT;
  PICTURE name <(format-options)>
  <value-range-set1>
  … <value-range-setn> ;
RUN;
```

Ein Beispiel zur Darstellung eines Gehaltes in US Dollar mit führendem $-Zeichen:

```
PICTURE pay low-high='000,009.99' (prefix='$');
```

Hierbei bedeutet eine 0, dass die Zahl nur dargestellt wird, wenn die entsprechende Stelle vorhanden ist. Bei Verwendung der 9 wird die Zahl immer dargestellt, also auch als führende 0.

Mit dem Beispiel-Picture würden die Gehälter 12345,67 und 456,7 folgendermaßen dargestellt:

```
12345,67 -> $12,345.67
456,7    ->    $456.70
```

In Auswertungstabellen zu klinischen Prüfungen werden häufig Prozent-Angaben gemacht, also der Anteil an der Gesamtpopulation dargestellt.

In diesem Beispiel soll die prozentuale Angabe in Klammern mit einer Nachkommastelle erfolgen. Das erwartete Ergebnis für 10.78 wäre also (10.8). Um die Umsetzung des Picture-Formates für verschiedene Werte zu zeigen, wurde ein Beispieldatensatz mit negativen und positiven Werten und unterschiedlicher Dezimalstellenzahl in der Variablen "Val" erzeugt.

Obs	Val
1	-100.876
2	-10.789
3	-1.56
4	0
5	0.678
6	10.78
7	99.987
8	100
9	100.188
10	1000

2.1 Einfaches PICTURE vom niedrigstem zum höchsten Wert

Der Variablen val1 wird der Wert von val zugewiesen und dann mit dem folgenden Picture belegt:

```
PICTURE val1_ low-high =' 99.0)'(prefix='(')
```

Hier das Ergebnis:

```
Obs       val    val1
  1  -100.876  (00.8)
  2   -10.789  (10.7)
  3     -1.56  (01.5)
  4         0  (00.0)
  5    0.6789  (00.6)
  6     10.78  (10.7)
  7    99.987  (99.9)
  8       100  (00.0)
  9   100.188  (00.1)
 10      1000  (00.0)
```

Man sieht, dass nur die Zehner- und Einerstellen sowie die erste Nachkommastelle der Variablen val extrahiert wird. Alles andere, auch das negative Vorzeichen, wird abgeschnitten und es wird auch nicht gerundet.

2.2 Einfaches PICTURE für Werte von 0-100

Im nächsten Schritt wurde das PICTURE nur für die Werte von 0-100 zugewiesen, was zu der Verbesserung führt, dass nur die vorgesehenen Werte mit dem PICTURE belegt werden. Die anderen bleiben unformatiert.

```
PICTURE val2_ 0-100 =' 99.9)'(prefix='(')
```

Ergebnis für val2, belegt mit PICTURE val2_ als Vergleich zu val1:

```
Obs       val    val1    val2
  1  -100.876  (00.8)  -100.9
  2   -10.789  (10.7)  -10.79
  3     -1.56  (01.5)   -1.56
  4         0  (00.0)  (00.0)
  5    0.6789  (00.6)  (00.6)
  6     10.78  (10.7)  (10.7)
  7    99.987  (99.9)  (99.9)
  8       100  (00.0)  (00.0)
  9   100.188  (00.1)  100.19
 10      1000  (00.0)    1000
```

2.3 Gerundetes PICTURE für Werte von 0-100

Nun wird versucht, die fehlende Rundung mit ins PICTURE zu bringen.

```
PICTURE val3_ (round) 0-100 =' 99.9)'(prefix='(')
```

Ergebnis für val3, belegt mit PICTURE val3_ im Vergleich zu den vorherigen PICTURE-Angaben in val1 und val2.

```
Obs       val    val1    val2    val3
  1  -100.876  (00.8)  -100.9  -100.9
  2   -10.789  (10.7)  -10.79  -10.79
  3     -1.56  (01.5)   -1.56   -1.56
  4         0  (00.0)  (00.0)  (00.0)
  5    0.6789  (00.6)  (00.6)  (00.7)
  6     10.78  (10.7)  (10.7)  (10.8)
  7    99.987  (99.9)  (99.9)  (00.0)
  8       100  (00.0)  (00.0)  (00.0)
  9   100.188  (00.1)  100.19  100.19
 10      1000  (00.0)    1000    1000
```

Die Werte sind nun gerundet, bei den Observations 7 und 8 gibt es aber noch Probleme, weil das PICTURE nur 2 Vorkommastellen beinhaltet. Der Wert wird aber erst auf 100 gerundet und dann das Picture darauf gelegt.

2.4 Gerundetes PICTURE für Kategorien von Werten

Im PICTURE wird neben der Rundung auch die Stellenzahl berücksichtigt.

```
PICTURE val4_ (round)
              0-<10= '   9.9)' (prefix='(')
             10-<100=' 99.9)' (prefix='(')
                 100=' 100.0)' (prefix='(')
```

Ergebnis für val4, belegt mit PICTURE val4_ im Vergleich:

```
Obs       val    val1    val2    val3     val4
  1  -100.876  (00.8)  -100.9  -100.9  -100.88
  2   -10.789  (10.7)  -10.79  -10.79  -10.789
  3     -1.56  (01.5)   -1.56   -1.56    -1.56
  4         0  (00.0)  (00.0)  (00.0)    (0.0)
  5    0.6789  (00.6)  (00.6)  (00.7)    (0.7)
  6     10.78  (10.7)  (10.7)  (10.8)   (10.8)
  7    99.987  (99.9)  (99.9)  (00.0)   (00.0)
  8       100  (00.0)  (00.0)  (00.0)  (100.0)
  9   100.188  (00.1)  100.19  100.19  100.188
 10      1000  (00.0)    1000    1000     1000
```

Auch das Einführen von Kategorien führt bei Observation 7 noch nicht zum gewünschten Ergebnis. Für die Darstellung wird zuerst festgestellt, dass der Wert im Bereich 10-<100 liegt, er wird also mit dem PICTURE (99.9) ausgegeben. Gerundet lautet der Wert nun 100, so dass durch die formatierte Ausgabe (00.0) als Ergebnis erscheint. Die Rundung muss also nicht im PICTURE, sondern bereits vorher erfolgen.

2.5 Anwendung von PICTURE für vorher gerundete Werte

Die Werte in der Variablen val werden zunächst auf eine Nachkommastelle gerundet und dann mit dem PICTURE aus 2.1.4 ohne die Rundung belegt:

```
val9 = round(val,0.1)

PICTURE val9_ (round)
              0-<10= '   9.9)' (prefix='(')
             10-<100='  99.9)' (prefix='(')
                 100=' 100.0)' (prefix='(')
```

Ergebnis für val9, belegt mit PICTURE val9_ im Vergleich

```
Obs       val    val1    val2    val3     val4      val9
  1  -100.876  (00.8)  -100.9  -100.9  -100.88    -100.9
  2   -10.789  (10.7)  -10.79  -10.79  -10.789     -10.8
  3     -1.56  (01.5)   -1.56   -1.56    -1.56      -1.6
  4         0  (00.0)  (00.0)  (00.0)    (0.0)     (0.0)
  5    0.6789  (00.6)  (00.6)  (00.7)    (0.7)     (0.7)
  6     10.78  (10.7)  (10.7)  (10.8)   (10.8)    (10.8)
  7    99.987  (99.9)  (99.9)  (00.0)   (00.0)   (100.0)
  8       100  (00.0)  (00.0)  (00.0)  (100.0)   (100.0)
  9   100.188  (00.1)  100.19  100.19  100.188     100.2
 10      1000  (00.0)    1000    1000     1000      1000
```

Man sieht, dass man sich oftmals an das gewünschte Ergebnis herantasten muss und dass möglichst viele unterschiedliche Variablenausprägungen bei der Programmierung berücksichtigt werden müssen.

3 Der Schein trügt

Ein vorhandener Datensatz labor_ca mit den numerischen Variablen Patient, Visite und Calcium soll für die Berechnung von Änderungen gegenüber dem Baseline-Wert aufbereitet werden. Dazu müssen zunächst die Baseline-Werte (Visite 1) selektiert werden.

Der Datensatz sieht folgendermaßen aus:

Obs	Patient	Visite	Calcium (mg/dl)
1	100	1	8.7
2	100	2	9.1
3	101	1	9.6
4	101	2	9.2
5	102	1	10.5
6	102	2	10.5

Mit folgendem Programmcode soll ein Datensatz mit den Baseline-Werten erzeugt werden:

```
DATA baselab;
   SET labor_ca;
   WHERE visite=1;
```

```
RUN;
```

Überraschenderweise hat der Datensatz baselab 0 Records. Woran liegt das? Ein Blick in das Log-Fenster ergibt keinen Hinweis auf einen Programmierfehler, einen inkompatiblen Vergleich oder Ähnliches. Auch beim Auswählen des Suchkriteriums *visit=1* direkt im SAS Editor werden keine Records gefunden.

Bei Abfrage *auf 0 LT visite LT 2* werden allerdings drei Records selektiert:

Obs	Patient	Visite	Calcium (mg/dl)
1	100	1	8.7
2	101	1	9.6
3	102	1	10.5

Ein zweiter Blick in den Datensatz enthüllt dann das Geheimnis. Wenn die Variable Visite mit dem Format 32.30 dargestellt wird, also mit 30 Nachkommastellen, sieht man, dass der Wert nicht wie es scheint 1, sondern tatsächlich 1.000000000000000000009999999999 ist. Das kann durch einen Datentransfer aus einem anderen System passieren oder durch Rundungsfehler bei Berechnungen.

Wie kann man sich helfen, wenn man nicht auf die vielen Nachkommastellen abfragen möchte? Eine einfache Lösung ist das Erzeugen einer neuen Variable, die auf eine begrenzte Zahl von Nachkommastellen gerundet ist, je nachdem ob auch Werte von z.B. 1.01 oder 1.10 für die Visite erwarten würde.

```
visneu = ROUND(visite,0.001)
```

Dann kann man für Abfragen auf die Visite mit der Variablen visneu anstelle der Variablen visite arbeiten.

Je nach Anwendungsbereich können auch die Funktionen INT, FLOOR oder CEIL helfen. Bei INT und FLOOR wird der Nachkommawert abgeschnitten, bei CEIL wird die nächsthöhere ganze Zahl verwendet.

4 Umwandeln von Variablen

4.1 Umwandlung von numerisch in alphanumerisch

Die Umwandlung von numerisch in alphanumerisch erfolgt mit der bekannten Funktion PUT.

Syntax: PUT (<Zahl/Variable>, format)
Beispiel: PUT (numvar,8.)

Im folgenden Beispiel soll ein numerisch vorliegender Medikamentencode in eine Text-Variable umgewandelt werden.

Obs	dcode
1	12345678
2	87654321

Der Code in dcode soll nun in Text in der Variablen dtext umgewandelt werden. Da man einen 8-stelligen Code erwartet wurde die Variable dtext mit der Länge 8 vorbelegt.

```
DATA drug_text;
   LENGTH dtext $8;
   SET drugcode;
   dtext = PUT(dcode,best.);
RUN;
```

Wenn man sich den Datensatz dann ansieht, erlebt man eine böse Überraschung:

Obs	dcode	dtext
1	12345678	1234
2	87654321	8765

Die Textvariable dtext beinhaltet nur die ersten 4 Stellen aus der Variablen dcode. Was ist passiert?
Die Erklärung hierfür ist ganz einfach: Das Format best, mit dem der 8-stellige Code eingelesen wurde, verwendet beim Umwandeln standardmäßig 12 Stellen. Das bedeutet, dass die Variablen mit führenden Leerstellen aufgefüllt werden. Beim Schreiben in die

Variable der Länge 8 wird dann nach dem 8. Zeichen abgeschnitten:

12345678 → "....12345678" → "....1234"

Die Leerzeichen wurden hier zur Lesbarkeit als Punkte dargestellt.

Die Lösung ist ebenso einfach: Um böse Überraschungen durch Standard-Einstellungen zu vermeiden sollte man ein festes Format verwenden, also

```
dtext = PUT(dcode,best8.);  oder
dtext = PUT(dcode,best8.);
```

4.2 Umwandlung von alphanumerisch in numerisch

Die Umwandlung von alphanumerisch in numerisch erfolgt mit der Funktion INPUT.

Syntax: INPUT (<Text/Variable>, format)
Beispiel: INPUT(“12345678“,best.);

Auch hier muss man auf Leerzeichen achten und die Länge des Textes und des Einleseformates beachten.

Im folgenden Beispieldatensatz gibt es für die alphanumerische Variable dtext je einen Record mit und ohne führende Leerzeichen (als Punkte dargestellt). Die Variable dtext soll nun mit unterschiedlichen Formaten in drei numerische Variable umgewandelt werden:

```
num_b  = INPUT(dtext,best.);
num_8  = INPUT(dtext,8.);
num_12 = INPUT(dtext,12.);
```

Obs	dtext	num_b	num_8	num_12
1	12345678	12345678	1234	12345678
2	12345678	12345678	12345678	12345678

Die Verwendung des Formates best (Variable num_b) war in diesem Fall gut und richtig. Bei Verwendung des Formates 8 (Variable num_8) wird nach der 8. Stelle abgeschnitten. Hier sollte man mit Funktionen wie TRIM oder LEFT vorher sicherstellen, dass es keine ungewollten führenden Leerzeichen gibt.

4.3 Umwandeln von Text in Datum

Das Umwandeln von Text in Datum ist im Grunde nichts anderes als das Umwandeln von alphanumerisch in numerisch, da ein Datum als numerischer Wert dargestellt wird.

Das Referenzdatum ist der 1. Januar 1960, das entspricht der Zahl 0. Um das Datum als solches besser lesbar darstellen zu können, wird der numerische Wert mit einem Datumsformat belegt wie z.B. date9.

Zum Einlesen von Datumsangaben mit der Funktion INPUT können folgende Formate verwendet werden:

- ddmmyy, mmddyy und yymmdd verschiedener Länge,
  z.B. ddmmyy10
- date verschiedener Länge, z.B. date7 oder date9

Zur Veranschaulichung des Einlesens mit verschiedenen Formaten wurde ein Beispieldatensatz erzeugt, der in der alphanumerischen Variable mydate immer den 3. Januar 2012 in unterschiedlichen Darstellungsweisen beinhaltet. Die Reihenfolge ist aber vorgegeben, nämlich immer Tag (dd), Monat (mm), Jahr (yy).

Obs	mydate
1	03JAN2012
2	03/JAN/2012
3	03JAN12
4	03012012
5	03/01/2012
6	3-1-2012
7	03/01/12

4.4 Einleseformat DDMMYYw.

Zunächst wird die Umwandlung Text in Datum mit dem Format DDMMYYw. durchgeführt. Für drei verschiedene Längen werden drei Variablen erzeugt.

```
d7=INPUT(mydate,ddmmyy7.)
 d8=INPUT(mydate,ddmmyy8.)
 d9=INPUT(mydate,ddmmyy9.)
```

Die erzeugten Variablen werden dann mit dem Format date9. angezeigt.

Obs	mydate (Text)	d7 (date9.)	d8 (date9.)	d9 (date9.)
1	03JAN2012			
2	03/JAN/2012			
3	03JAN12			
4	03012012	***30DEC2001***	03JAN2012	03JAN2012
5	03/01/2012	03JAN***2002***	03JAN***1920***	
6	3-1-2012		03JAN2012	03JAN2012
7	03/01/12	03JAN***2001***	03JAN2012	03JAN2012

Auf den ersten Blick fällt auf, dass die Tabelle im oberen Bereich nicht gefüllt ist. Das liegt daran, dass das Format DDMMYw. einen numerischen Monat benötigt.

Leere Felder springen beim Validieren sofort ins Auge und man kann im Log-Fenster nachsehen, ob es einen Hinweis auf Probleme beim Umwandeln gibt.
Problematischer sind ausgefüllte Felder, die aber falsch sind bzw. nicht das erwartete Datum enthalten (hier hellgrau unterlegt).

Es wurde bereits darauf hingewiesen, dass nur die im Format vorgegebene Länge beim Umwandeln der Textvariablen berücksichtigt wird. Das führt für das verwendete Format ddmmyy7 in Variable d7 in Obs 4 dazu, dass nur die ersten 7 Stellen des Textes, also 0301201 eingelesen werden. Das sieht zunächst nicht nach einem gültigen Datum aus. SAS versucht aber umzuwandeln und lässt die führende Null weg. Damit wird 301201 umgewandelt in den numerischen Wert, der dem 30. Dezember 2001 entspricht.

4.5 Einleseformat DATEw.

Derselbe Datensatz wird nun verwendet, um mit dem Format DATEw. einzulesen. Für 3 verschiedene Längen werden drei Variablen erzeugt.

```
 dat7=INPUT(mydate,date7.)
 dat9=INPUT(mydate,date9.)
dat11=INPUT(mydate,date11.)
```

Die erzeugten Variablen werden wieder mit dem Format DATE9. angezeigt.

Obs	Mydate (Text)	dat7 (date9.)	dat9 (date9.)	dat11 (date9.)
1	03JAN2012	03JAN***1920***	03JAN2012	03JAN2012

2	03/JAN/2012		03JAN***1920***	03JAN2012
3	03JAN12	03JAN2012	03JAN2012	03JAN2012
4	03012012			
5	03/01/2012			
6	3-1-2012			
7	03/01/12			

Diesmal fällt auf, dass die Tabelle im unteren Bereich nicht gefüllt ist. Das liegt daran, dass das Format DATEw. den Monatsanteil in Textform benötigt. Außerdem muss bei DATEw. der Tag vornestehen.

Weiterhin fällt auf, dass manchmal in das Jahr 1920 umgewandelt wird, manchmal aber auch in 2012. Auch das hängt mit der eingelesenen Länge zusammen. Bei Obs 1 werden für die Spalte dat7 nur die ersten 7 Stellen eingelesen, d.h. die Jahreszahl lautet 20. Bei Obs 3 ist die Jahreszahl sowieso nur 2-stellig als 12 angegeben. Warum wird in 1920 und 2012 umgewandelt?

Wenn eine Jahreszahl nur 2-stellig angegeben ist, wird sie je nach Yearcutoff in eine 4-stellige Jahreszahl umgewandelt. Der Standard des Yearcutoff ist 1920, das bedeutet eine 100-Jahr Spanne von 1920 bis 2019. Dabei werden die Zahlen von 20-99 zu 1900 zugeordnet, die Zahlen von 0-19 zu 2000.

Der Yearcutoff ist einstellbar mit OPTIONS YEARCUTOFF=nnnn.
Für nnnn können Jahreszahlen von 1582-19900 eingegeben werden.

4.6 Zusammenfassung zu den Einleseformaten

Um zu verdeutlichen, wie wichtig es ist sich die Daten und Ergebnisse genau anzusehen, wird der Beispieldatensatz, der in der Reihenfolge Tag-Monat-Jahr definiert wurde, mit unterschiedlichen Formaten eingelesen, auch mit Formaten, die eine andere Reihenfolge voraussetzen.

Damit soll darauf hingewiesen werden, dass nicht das Programm erkennt, wie ein Datum vorliegt, sondern der Programmierer das richtige Format auswählen muss. Dabei muss sowohl auf die Reihenfolge als auch auf die Länge geachtet werden. Wichtig ist, dass berechnete Werte nicht unbedingt richtig sein müssen. Sehr hilfreich sind Plausibilitätschecks.

Die Einleseformate stehen jeweils in der Spaltenüberschrift. Falsch umgesetzte Datumsanteile sind fett und kursiv dargestellt.

Mydate	**date7.**	**date8.**	**date9.**	**date11.**
03JAN2012	03JAN***1920***		03JAN2012	03JAN2012
03/JAN/2012		03JAN***2002***	03JAN***1920***	03JAN2012
03JAN12	03JAN2012	03JAN2012	03JAN2012	03JAN2012

Mydate	ddmmyy7.	mmddyy7.	yymmdd7.	yymmdd11.
03012012	***30DEC2001***		***01DEC1930***	
03/01/2012	03JAN***2002***	***01MAR2002***	***02***JAN***2003***	
3-1-2012				
03/01/12	03JAN***2001***	***01MAR2001***	***01***JAN***2003***	***12***JAN***2003***

5 Arbeiten mit Datumsangaben als Freitext mit Hilfe verschiedener Funktionen

In den vorhergehenden Beispielen wurde bereits deutlich, dass beim Umwandeln von Text in eine Datumsangabe viele Vorüberlegungen notwendig sind. Vor allem muss man sich überlegen, wie das Datum in Textform vorliegt:

- Tag einstellig / zweistellig
- Monat als Zahl oder Text, deutsch oder englisch
- Jahr zweistellig oder vierstellig
- Reihenfolge von Tag/Monat/Jahr
- mit Trennzeichen (leer, Komma, Punkt, Striche)
- Englisch: mit 1st, 2nd, 3rd, 4th – klein oder groß
- Fehlende Datumsanteile (leer, --, UNK, NUL,??)

In den bisherigen Beispielen waren immer alle Datumsanteile vorhanden. Nun wird gezeigt wie man vorgehen kann, wenn ein Datum wirklich als Freitext vorliegt.

5.1 Beispiel mit etwas vorstrukturierten Daten

Hier zunächst ein Beispiel mit etwas vorstrukturierten Daten, wobei die Reihenfolge mit Monat-Tag-Jahr festliegt und die Datumsanteile durch einen Schrägstrich getrennt sind. Allerdings sind nicht immer alle Datumsanteile vorhanden und die unvollständigen Angaben wurden durch unterschiedliche Zeichen aufgefüllt.

obs	sdate
1	01/15/2000
2	/15/2000
3	/ /2000
4	--/--/--
5	UNK/UNK/2000
6	01/UNK/2000
7	//

Man kann nun aus dem Gesamttext die Datumsanteile Tag, Monat und Jahr heraussuchen. Wenn ein Trennzeichen vorhanden ist, ist das relativ einfach mit der Funktion SCAN möglich.

```
month =SCAN(sdate,1,"/");
day   =SCAN(sdate,2,"/");
year  =SCAN(sdate,3,"/");
```

SCAN sucht „Wörter“ zwischen den Trennzeichen.
Im 2. Argument steht die Nummer des Wortes.
Im 3. Argument kann das Trennzeichen angegeben werden.

Die einzelnen Bestandteile müssen dann auf Plausibilität geprüft werden und sollen anschließend in eine Form gebracht werden, die in ein Datum umgewandelt werden kann.

In Plausibilitätsprüfungen kann z.B. auf unvollständige Angaben oder zulässige Wertebereiche abgefragt werden:

```
IF _month_ in ("NUL","UNK"," ","--", "??")
THEN _month_="--";

IF NOT (1900 le _year_ le heute) THEN put
"Check: ungültig für " _year_=;
```

Zur Besseren Übersicht sollen die Bestandteile dann mit fester Länge und festem Trennzeichen zusammengefügt werden. Eine Umwandlung in numerisch ist nur für vollständige Datumsangaben möglich.

Obs	sdate	fxdate	fdate
1	01/15/2000	**2000-01-15**	**15JAN2000**
2	/15/2000	**2000----15**	
3	/ /2000	**2000------**	
4	--/--/--	**----------**	
5	UNK/UNK/2000	**2000------**	
6	01/UNK/2000	**2000-01---**	
7	//	**----------**	

5.2 Datum eingebunden in ein Kommentarfeld

Manchmal liegen Datumsangaben aber auch nicht in vorstrukturierter Form vor, sondern müssen z.B. aus einem Kommentarfeld herausgesucht werden.

Beispieldatensatz:

Obs	textdate
1	**performed on Jan 11th, 2001**
2	**performed on Jan 11, 2001**
3	**performed on DEC 11TH, 1999**
4	**performed on Feb 3rd, 1999**
5	**performed on Mar 2nd, 1998**
6	**performed on Okt 11th 1998**

```
         12345678901234567890123456
```

Zur besseren Übersicht ist am Ende des Datensatzes eine Stellenübersicht eingefügt. Bei der Suche nach den Datumsanteilen führen sicher viele Wege zum Ziel. Man muss sich die vorhandenen Einträge ansehen und nach Gemeinsamkeiten suchen. Dann kann man sich schrittweise an die Lösung des Problems herantasten.

Als erstes kann der Einleitungstext "performed on" entfernt werden:

```
exdate=substr(textdate,14);
```

Anschliessend müssen aus exdate wieder die Bestandteile Monat, Tag und Jahr herausgesucht werden. Auf Basis des ersten Record würde das folgendermaßen aussehen:

Obs	exdate
1	**Jan 11th, 2001**

```
       123456789012345
```

```
month= upcase(substr(exdate,1,3))
day  = substr(exdate,5,2)
year = substr(exdate,11,4)
```

Die Datumsanteile werden dann zusammengesetzt in ein "formatiertes" Textfeld datefx geschrieben und anschließend in eine Datumsvariable date umgewandelt. Zur Anzeige wird das Format date9. verwendet.

```
datefx = day||month||year
date   = input(datefx,date9.)
```

Das Ergebnis ist noch nicht ganz so wie gewünscht:

obs	exdate	datefx	date (date9.)
1	**Jan 11th, 2001**	11JAN2001	11JAN2001
2	**Jan 11, 2001**	11JAN01	11JAN2001
3	**DEC 11TH, 1999**	11DEC1999	11DEC1999
4	**Feb 3rd, 1999**	3rFEB999	
5	**Mar 2nd, 1998**	2nMAR998	
6	**Okt 11th 1998**	11OKT998	

```
       123456789012345
```

Wenn kein "th" nach dem Tag steht, verschiebt sich das Jahr entsprechend nach vorne. Außerdem ist der Tag manchmal nur einstellig angegeben.

Bei den folgenden schrittweisen Überlegungen zum Heraussuchen der Datumsbestandteile wird jeweils ein Auszug aus dem Beispieldatensatz mit Stellenangabe vor die beschriebene SAS-Funktion gestellt.

5.3 Berücksichtigung von "th" usw. nach dem Tag

Mit der Funktion INDEX kann man die Position einer Zeichenkette in einer Variablen ermitteln. Das kann mit der Abfrage auf gleich oder größer Null auch als Indikator für das Vorhandensein einer Zeichenkette benutzt werden.

```
123456789012345
Jan 11, 2001
Jan 11th, 2001
```

```
IF INDEX(exdate,'th') = 0 THEN year =substr(exdate,9,4)
IF INDEX(exdate,'th') > 0 THEN year =substr(exdate,11,4)
```

INDEX gibt die erste Stelle aus, an der die Zeichenkette des 2. Arguments im 1. Argument vorkommt.

Die Position des Jahrs ändert sich natürlich auch bei Verwendung von "1st", "2nd" und "3rd".

5.4 Anpassung von day für einen 1-stelligen Tag

Wenn der Tag nur eine Stelle hat, muss der Tagesanteil mit einer führenden Null aufgefüllt werden. Um herauszufinden, ob der Tag zweistellig vorliegt, wird geprüft, ob die 2. Stelle des Tages numerisch ist.

```
123456789012345
Jan 11th, 2001
Mar 2nd, 98

day = substr(exdate,5,2)
IF INDEXC(substr(day,2),'0123456789') = 0
THEN day = "0"||substr(exdate,5,1);
```

INDEXC gibt die erste Stelle aus, an der irgendein Zeichen der Zeichenkette des 2. Arguments im 1. Argument vorkommt. Im Beispiel wurden die Zahlen von 0-9 angegeben. Das kann man auch noch eleganter mit folgender Funktion erreichen:

```
day = substr(exdate,5,2)
IF NOTDIGIT(day) > 0
THEN day = "0"||substr(exdate,5,1);
```

NOTDIGIT gibt die erste Stelle aus, an der das Argument nicht numerisch ist. Wenn das Ergebnis also größer 0 ist, besteht day nicht nur aus Zahlen.

Wenn der Tag nur eine Stelle hat oder wenn zusätzliche Zeichen wie "th" oder Kommas vorkommen, muss die Stelle, ab der das Jahr extrahiert wird entsprechend angepasst werden. Dazu kann man sich weitere Abfragen überlegen oder versuchen, die Stellen zwischen Tag und Jahr irgendwie zu eliminieren.

5.5 Ignorieren der Stellen zwischen Tag und Jahr

Ein Ansatz dazu ist, den Text einfach rückwärts einzulesen, weil das Jahr im Beispieldatensatz ganz hinten steht.

```
123456789012345
Jan 11th, 2001
Jan 11, 2001
Mar 2nd 98

revtxt = LEFT(REVERSE(exdate));
posleer = INDEX(revtxt," ");
year = reverse(substr(revtxt,1,posleer-1));
```

REVERSE gibt eine Zeichenkette in umgekehrter Reihenfolge aus. INDEX sucht dann nach dem ersten Leerzeichen vom Variablenende aus.

Eine andere Möglichkeit ist die Anwendung von COMPRESS mit dem dritten Argument "k":

```
123456789012345
Jan 11th, 2001
Jan 11, 2001
Mar 2nd 98
```

```
intdate = COMPRESS(exdate,"1234567890","k");
```

Normalerweise löscht die Funktion COMPRESS die im zweiten Argument angegebenen Zeichen. Hier wurde aber „k" = KEEP als drittes Argument angegeben und damit behält COMPRESS die angegebenen Zeichen

- Jan 11th, 2001 → 112001
- Jan 11, 2001 → 112001
- Mar 2nd 98 → 298

Man muss natürlich trotzdem noch herausfinden, welche Zahlen nun zu Tag oder Jahr gehören. Wenn man aber den Text erst ab Stelle 7 verwendet, erhält man direkt das Jahr.

- th, 2001 → 2001
- , 2001 → 2001
- d 98 → 98

5.2.4 Berücksichtigung deutscher Monatskürzel

Deutsche Monatskürzel wie OKT oder MÄR können nicht mit dem Format date9 eingelesen werden. Deshalb müssen sie vorher in englische Kürzel umgewandelt werden.

```
123456789012345
Okt 11th, 1998
```

```
smonth = upcase(substr(exdate,1,3))
month = TRANWRD(smonth,"OKT","OCT");
```

TRANWRD setzt Zeichenketten um: Im 1. Argument wird die Zeichenkette im 2. Argument durch die im 3. Argument ersetzt.

Die deutschen Kürzel MÄR, MAI und DEZ müssten dann ebenso in die entsprechenden englischen Monatskürzel umgesetzt werden.

```
month = TRANWRD(smonth,"MÄR","MAR");
month = TRANWRD(smonth,"MAI","MAY");
month = TRANWRD(smonth,"DEZ","DEC");
```

Nach der Anwendung der eben genannten Funktionen sieht der Beispieldatensatz dann so aus wie gewünscht:

obs	textdate	datefx	date
1	Jan 11th, 2001	11JAN2001	11JAN2001
2	Jan 11, 2001	11JAN2001	11JAN2001
3	DEC 11TH, 1999	11DEC1999	11DEC1999
4	Feb 3rd, 1999	03FEB1999	03FEB1999
5	Mar 2nd, 98	02MAR98	02MAR1998
6	Okt 11th 1998	11OCT1998	11OCT1998

Für die momentan vorhandenen Daten wurde nun alles berücksichtigt und richtig umgesetzt. Man muss aber immer damit rechnen, dass beim nächsten Datentransfer neue Konstellationen vorkommen können!

Literatur

Zum Nachschlagen der SAS Funktionen wurde die Internet-Seite http://support.sas.com verwendet.