# Proc Transpose oder Do-it-yourself

Heribert Ramroth
Institute of Public Health
INF 324
69120 Heidelberg
Heribert.Ramroth@uni-heidelberg.de

**Zusammenfassung**

Bei der Datenanalyse taucht regelmäßig die Situation auf, Daten in eine andere Struktur überführen zu müssen, als diese aktuell vorhanden sind. Hier bietet sich je nach Datensituation PROC TRANSPOSE an. Der folgende Beitrag vergleicht PROC TRANSPOSE und deren Alternativen (einfacher Datastep, Datastep mit Arrays, PROC IML, PROC SQL) anhand eines konkreten Beispiels epidemiologischer Daten. Das Ziel soll möglichst innerhalb eines SAS Moduls erreicht werden, ohne zwischen verschiedenen Programmieransätzen zu wechseln. Ein zusätzliches Ziel besteht darin die Ergebnisdaten publikationsfertig, wenn möglich mit Label und Format der Variablen auszugeben, so dass der potentielle Aufwand einer Nachbearbeitung der Ergebnistabellen minimiert wird.

**Schlüsselwörter:** Erstellung publikationsfertiger epidemiologischer Ergebnisse, Datastep, Array, PROC IML, PROC TRANSPOSE, PROC SQL

# 1 Einleitung

Die Erstellung publikationsfertiger Ergebnisse klinisch-epidemiologischer Studien sind ein wichtiger Aspekt wissenschaftlicher Auswertungen, bei der die manuelle Übertragung bzw. Nachbearbeitung von Ergebnistabellen möglichst vermieden werden soll [1, 2]. Ergebnisse von Standardprozeduren in SAS können dabei zur Nachbearbeitung in einer per ODS erzeugten Tabelle abgelegt werden, deren Form jedoch zur Erlangung des letztlich gewünschten Ergebnisses noch geändert werden muss. Beispielhaft werden hier Daten einer epidemiologischen Fall-Kontrollstudie behandelt, in denen die Häufigkeitsverteilungen von Kontrollpersonen und Patienten bzgl. der interessierenden Risikofaktoren nebeneinander gestellt werden. Als Möglichkeit zur Lösung eines Teil-Aspektes bietet sich PROC TRANSPOSE zum Transponieren der Daten der ODS-Ergebnistabelle aus PROC FREQ an [1]. Wünschenswerte Nebenbedingungen sind zusätzlich die Übertragung von Formaten und Labeln der Risikofaktoren, zur besseren Lesbarkeit der Ergebnisse und zum Minimieren des Nachbearbeitungsaufwandes.

In vorangegangenen Beiträgen [1,2] wurde dabei mehr auf das Erreichen des Ziels geachtet, nicht darauf, das Ziel möglichst unter Verwendung nur *eines* Programmierkonzeptes zu erreichen. Der jetzige Beitrag zeigt verschiedene Konzepte zur Problemlösung auf:

(1) einfacher Datastep

(2) Datastep unter Verwendung von Arrays

(3) PROC IML

(4) PROC TRANSPOSE

(5) PROC SQL

# 2 Methoden

**Der Beispiel-Datensatz**

Als Anwendungsbeispiel dienen Daten der SAS Hilfe, Version 9.2 (SAS-Example 65.4: Conditional Logistic Regression for m:n Matching [3]). Hierbei handelt es sich um eine Fall-Kontrollstudie zu Risikofaktoren bei untergewichtigen Babys. Die hier verwendeten Variablen Rauchstatus der Mutter (Smoke) und Bluthochdruck (HT) dienen der Beschreibung der Methodik und stellen nur einen Auszug der im Beispiel vorhandenen Variablen (=Risikofaktoren) dar. Die hier gezeigten Programme lassen sich jedoch problemlos auf alle Risikofaktoren anwenden.

| Variablenname | Label | Kodierung | Format |
|---|---|---|---|
| Low | Fall-/Kontroll-status | 0, 1 | 0=‘Kontrolle‘<br>1=‘Fall‘ |
| Smoke | Rauchstatus der Mutter | 0, 1 | 0=‘Nichtraucherin‘<br>1=‘Raucherin‘<br>. =‘ ‘ |
| HT | Bluthochdruck | 0, 1 | 0=‘kein Bluthochdruck‘<br>1=‘Bluthochdruck‘<br>. =‘ ‘ |

Als Ausgangsdatensatz zur Illustration der verschiedenen Konzepte dient der per Output Delivery System (ODS) aus PROC FREQ erzeugte Datensatz *Start*. Die gleichzeitige Angabe von absoluten sowie prozentualen Verteilungen ist sinnvoll – nicht nur in epidemiologischen Studien. Somit werden im Folgenden beide Angaben kombiniert in einer *NPct* genannten Variablen dargestellt.

```
ods output CrossTabFreqs=Start;
proc freq data=LBW;
     tables Low * (Smoke  HT) /nopercent norow;
run;
```

Im folgenden Ergebnis sind nur die inneren Werte der Kreuztabelle dargestellt (_TYPE_="11"). Details hierzu finden sich bei Ramroth 2008 [1].

| *Table* | *Low* | *Smoke* | *Frequency* | *RowPercent* | *Missing* | *HT* |
|---|---|---|---|---|---|---|
| Table Low * Smoke | 0 | 0 | 78 | 65.0000 | . | . |
| Table Low * Smoke | 0 | 1 | 42 | 35.0000 | . | . |
| Table Low * Smoke | 1 | 0 | 26 | 48.1481 | . | . |
| Table Low * Smoke | 1 | 1 | 28 | 51.8519 | . | . |
| Table Low * HT | 0 | . | 115 | 95.8333 | . | 0 |
| Table Low * HT | 0 | . | 5 | 4.1667 | . | 1 |
| Table Low * HT | 1 | . | 48 | 88.8889 | . | 0 |
| Table Low * HT | 1 | . | 6 | 11.1111 | . | 1 |

Vier Schritte sind zur angestrebten Darstellung (Datei: *Ziel*) notwendig [1]:

(i) Extraktion der im Argument von PROC FREQ aufgeführten Risikofaktoren (Zielvariable bezeichnet als *OrigVar*).

(ii) Zusammenfassen der Variablen Frequency und RowPercent in einer Zielvariablen *NPct*, die gleichzeitig die absoluten und relativen Häufigkeiten der Risikovariablen für Fälle bzw. Kontrollpersonen enthält.

(iii) Zusammenfassen der Werte der Variablen Smoke und HT in einer Zielvariablen (*VarValue*). Sinnvoll ist es hier, dies gleichzeitig auch für die Formate der Variablen durchzuführen (Zielvariable: *VarFormat*).

(iv) Transponieren der neu erstellten Variablen *NPct* für Fälle und Kontrollen nebeneinander in 2 Spalten. Dabei werden die redundanten Informationen zu *Low, OrigVar* und *VarValue* weggelassen.

Das Ergebnis der Schritte (i) bis (iii) wird zur späteren Verwendung im Zwischendatensatz *Zwischen* festgehalten. Der Datensatz *Ziel* sieht nach Schritt (iv) wie folgt aus:

| *OrigVar* | *VarValue* | *VarFormat* | *Kontrollen* | *Fälle* |
|---|---|---|---|---|
| Smoke | 0 | Nichtraucherin | 78 (65.0) | 26 (48.1) |
| Smoke | 1 | Raucherin | 42 (35.0) | 28 (51.9) |
| HT | 0 | Bluthochdruck nein | 115 (95.8) | 48 (88.9) |
| HT | 1 | Bluthochdruck ja | 5 ( 4.2) | 6 (11.1) |

## 2.1 Der einfache Datastep (Konzept 1)

Die Zeilen 1 bis 10 des nachfolgenden Codes sind nummeriert, da sie aufgrund der übersichtlichen Lesbarkeit im Datastep leicht referenziert werden können.

```
 1 DATA Contr Cases;
 2   SET Start;
 3   OrigVar=SCAN(Table, 3);
 4   NPct=Frequency||" ("||PUT(RowPercent,4.1)||")";
 5   IF _TYPE_="11";   * nur innere Werte, keine Randverteilung;

 6   VarValue=Sum(of Smoke HT);

 7   FSmoke=PUT(Smoke, Smoke.);
 8   FHT=PUT(HT, HT.);
 9   VarFormat=CATS(FSmoke, FHT);

10  IF Low=0 THEN OUTPUT Contr;
11  IF Low=1 THEN OUTPUT Cases;

PROC SORT DATA=Contr; BY OrigVar VarValue;
PROC SORT DATA=Cases; BY OrigVar VarValue;
DATA Ziel;
  MERGE Cases (keep=OrigVar VarValue VarFormat NPct
               rename=(NPct=Cases))
        Contr (keep=OrigVar VarValue VarFormat NPct
               rename=(NPct=Contr));
  BY OrigVar VarValue;
RUN;
```

Die Ziele (i) und (ii) lassen sich unter Verwendung der Funktionen SCAN und PUT leicht erreichen (Zeilen 3-4). Für Ziel (iii) kann in dieser Datensituation die Funktion SUM angewandt werden, da von den Variablen Smoke und HT immer nur eine mit einem numerischen Wert belegt ist (Zeile 7). Für das Pendant, d.h. das Zusammenfassen der Formate der Variablen Smoke und HT in einer Zielvariablen *VarFormat* werden mit Hilfe der Funktion PUT erst zwei neue Variablen *FSmoke* und *FHT* erzeugt, um sie schließlich mit der Funktion CATS in der Zielvariablen *VarFormat* zusammenzufassen. Alternative Funktionen zu SUM und CATS sind auch COALESCE und COALESCEC. Wichtig: Hierbei muss vorab berücksichtig werden, dass für die Variablen Smoke und HT auch Formate für die aus der ODS Anweisung mittels PROC FREQ entstandenen fehlenden Werte angelegt werden müssen (siehe Datensatz *Start*).
Für das Zusammenfügen der beiden Teildateien Cases und Contr (*Merge by*) müssen diese vorab sortiert werden. Eine möglicherweise beabsichtigte hierarchische Darstellung der Risikofaktoren (z.B. Rauchen vor Bluthochdruck) würde durch die Sortierung verändert werden. Um diese Reihenfolge beizubehalten, könnte z.B. im Ausgangsdatensatz die Variable _N_ eingefügt werden.

## 2.2 Datastep mit Arrays (Konzept 2)

Die Möglichkeiten von *Arrays* sind vielfältig und können hier nicht ausreichend dargestellt werden. Der implizite Array (Do-Over) bietet sich hier an, Variablen deren Werte auf verschiedenen Positionen stehen, zusammenzufassen.

Die Zeilen 2 bis 5 aus Konzept 1 bleiben gleich (d.h. der Code für die Ziele (i) und (ii) bleibt unverändert) und werden daher hier nicht aufgeführt

```
1 DATA Zwischen;

2-5 (Definition von OrigVar, NPct)

  ARRAY ArrVar Smoke HT;
  DO OVER ArrVar;
    IF ArrVar ne . THEN VarValue=PUT(ArrVar,1.);
  END;

7-8 (Definition von FSmoke, FHT)

  ARRAY FArrVar $ 20. FSmoke FHT;
  DO OVER FArrVar;
     IF FArrVar ne " " THEN VarForm=FArrVar;
  END;
```

Für Ziel (iii) fasst die Do-Over-Array-Anweisung die beiden Variablen Smoke und HT in der Zielvariablen *VarValue* zusammen. Analog ist der Arraycode für die Formatvariablen FSmoke und FHT zu lesen (Zielvariable *VarFormat*). Der Datensatz *Zwischen* wird hier separat gespeichert, da er den beiden Konzepten mit PROC IML und PROC TRANSPOSE als Ausgangsdatensatz dient.
Der folgende Abschnitt bezieht sich nur auf Ziel (iv), d.h. das Transponieren der Daten:

```
PROC SORT DATA=Zwischen;
  BY Origvar VarValue;

DATA Ziel;
  ARRAY x[2] $10. Contr Cases;
  DO UNTIL (LAST.VarValue);
    SET Zwischen;
    BY OrigVar VarValue;
    x[Low+1]=left(NPct);
  END;
RUN;
```

Elegant ist die hier verwendete Lösung zum Transponieren der Daten, da sie die SET-Anweisung in ungewohnter Weise benutzt (mehrfaches Einlesen des *Start* Datensatzes). Wichtig: die Indizierung des expliziten Arrays x kann hier nur aufgrund der konkreten Werte der Variablen Low auf diese Weise erfolgen (Low+1). Ziel ist es, die Positionen

1 und 2 des Arrays anzusteuern. Bei einer anderen Variablenkodierung mag die konkrete Lösung auf andere Art zu realisieren sein.

## 2.3 PROC IML (Konzept 3)

Die hier verwendete Variante zur Anwendung von PROC IML verwendet den Datensatz *Zwischen*. Es ist jedoch auch denkbar den Ausgangsdatensatz *Start* zu verwenden, dessen Teilinformationen aufwendiger einzulesen sind. Letztlich ist jedoch PROC IML nicht das Standardpaket für diese Datensituation, so dass auf diesen Aufwand hier verzichtet wird.

```
PROC IML;
  USE Zwischen;
  READ all VAR {OrigVar VarValue NPct} INTO contr where(Low=0);
  READ all VAR {OrigVar VarValue NPct} INTO cases where(Low=1);;

  IF (contr[,1]=cases[,1] & contr[,2]=cases[,2])
                                    THEN CaCo=contr || cases[,3];

  varname={OrigVar Value Contr Cases};
  CREATE ziel FROM CaCo [colname=varname];
  APPEND FROM CaCo;
QUIT;
```

## 2.4 PROC TRANSPOSE (Konzept 4)

Ausgangspunkt ist der Datensatz *Zwischen*. Verschiedene Optionen und Anweisungen werden ausprobiert, um Ziel (iv) zu erreichen:

Wichtig im ersten Beispielcode ist die VAR Anweisung: PROC TRANSPOSE würde ohne diese nur numerische Variablen transponieren. Mittels VAR Anweisung wird aber auch die Textvariable *NPct* transponiert.

```
PROC TRANSPOSE DATA= Zwischen OUT=Out1;
  VAR NPct;
RUN;
```

| *Obs* | *_NAME_* | *COL1* | *COL2* | *COL3* | *COL4* | *...* | *COL7* | *COL8* |
|---|---|---|---|---|---|---|---|---|
| *1* | NPct | 78 (65.0) | 42 (35.0) | 26 (48.1) | 28 (51.9) | ... | 48 (88.9) | 6 (11.1) |

Diese Syntax transponiert natürlich die eine Spalte NPct in eine Zeile. Abhilfe schafft hier die BY Anweisung.

Dazu ist es nötig, die Daten vorab zu sortieren:

```
PROC SORT DATA=Zwischen;
  BY OrigVar VarValue;
PROC TRANSPOSE DATA= Zwischen OUT=Out2;
  VAR NPct;
  BY OrigVar VarValue;
RUN;
```

| *OrigVar* | *VarValue* | *_NAME_* | *COL1* | *COL2* |
|---|---|---|---|---|
| HT | 0 | NPct | 115 (95.8) | 48 (88.9) |
| HT | 1 | NPct | 5 ( 4.2) | 6 (11.1) |
| Smoke | 0 | NPct | 78 (65.0) | 26 (48.1) |
| Smoke | 1 | NPct | 42 (35.0) | 28 (51.9) |

Die äußere Erscheinung des Ergebnisses passt, es fehlt nur noch die Beschriftung. Ein Versuch mit der Option NAME bringt jedoch nicht das gewünschte Ergebnis, da sie nur die Spalte _NAME_ umbenennt, nicht aber die Spalten COL1 und COL2:

```
PROC TRANSPOSE DATA=Zwischen OUT=Out3 NAME=Transp_Spalte;
  VAR NPct;
  BY OrigVar VarValue;
RUN;
```

| *OrigVar* | *VarValue* | *Transp_Spalte* | *COL1* | *COL2* |
|---|---|---|---|---|

Eine Umbenennung gelingt mit der PREFIX Option. Natürlich könnte hier statt PREFIX=CaCo auch PREFIX=NPct stehen, was inhaltlich stimmt. Andererseits führt dies vielleicht bei der Belegung der Anweisungen nur noch zu mehr Verwirrung. Die hier nicht verwendete Variable _NAME_ lässt sich leicht mit DROP entfernen.

```
PROC TRANSPOSE DATA=Zwischen OUT=Out4 (drop=_NAME_) PREFIX=CaCo;
  VAR NPct;
  BY OrigVar VarValue;
RUN;
```

| *OrigVar* | *VarValue* | *CaCo1* | *CaCo2* |
|---|---|---|---|

Welche Werte verbergen sich jedoch hinter CaCo1, die Fälle oder die Kontrollen? Um dies sagen zu können müsste man die Werte kennen.

Zum Ziel führt letztlich die ID Anweisung. Da 0 der Wert der Kontrollen der Variablen Low ist, enthält nun die Variable CaCo0 die Häufigkeitsverteilung der Kontrollen, CaCo1 die der Fälle.

```
PROC TRANSPOSE DATA=Zwischen OUT=Ziel (drop=_NAME_) PREFIX=CaCo;
  VAR NPct;
  BY OrigVar VarValue;
  ID Low;
RUN;
```

| *OrigVar* | *VarValue* | *CaCo0* | *CaCo1* |
|---|---|---|---|
| HT | 0 | 115 (95.8) | 48 (88.9) |
| HT | 1 | 5 ( 4.2) | 6 (11.1) |
| Smoke | 0 | 78 (65.0) | 26 (48.1) |
| Smoke | 1 | 42 (35.0) | 28 (51.9) |

Anwendbar sind mit der Prozedur TRANSPOSE auch Formate. Für den Zweck die Daten später noch mit Risikoschätzern aus logistischen Modellen zu ergänzen [1, 2] ist dies alleine jedoch noch nicht ausreichend, da ja gleichzeitig die nötigen Variablen-Werte (*VarValue*) verloren gehen.

## 2.5 PROC SQL (Konzept 5)

Der folgende Code ist zum einfacheren Referenzieren wieder durchnummeriert.

Ausgangsdatensatz ist der per ODS erzeugte Datensatz *Start*.

Zur besseren Lesbarkeit ist der erste Teil, der die Ziele (i) bis (iii) erreicht in einem View realisiert (genannt: *ZwischenView*), der nur eine Anweisung darstellt, aber keine Tabelle erzeugt.

Die Zeilen 5 und 6 sind selbsterklärend. Sie entsprechen dem Code im Datastep (Zeilen 4-5). Interessanterweise bedarf es hier bei der Konstruktion der Variablen *NPct* der Funktion PUT auch bei Anwendung auf die Variable Frequency. Der Datatep hat das Fehlen der Funktion PUT ignoriert, weil er dies zu interpretieren wusste.

Die Funktion COALESCE fasst die numerischen Variablen Smoke und HT zur Variablen *VarValue* zusammen. Das Pendant für Text-Variablen ist die Funktion COALESCEC.

```
 1 PROC SQL;
 2  CREATE VIEW ZwischenView AS
 3  SELECT
 4    Low,
 5    SCAN(Table, 3) AS OrigVar FORMAT $10.,
 6    PUT(Frequency,3.0)||" ("||PUT(RowPercent,4.1)||")" AS NPct
                                                    FORMAT $10.,
 7    COALESCE(Smoke, HT) AS VarValue FORMAT 3.,
 8    COALESCEC(PUT(Smoke, Smoke.), PUT(HT, HT.)) AS VarFormat
                                                     FORMAT $20.
 9  FROM Start
10  WHERE _TYPE_="11";
```

Elegant lässt sich das Transponieren der Daten lösen. Hierbei lässt sich der *view* mit sich selbst verknüpfen. Er muss dazu nur doppelt referenziert werden. Ausgewählt werden die Variablen *OrigVar*, *VarValue* und *VarFormat* natürlich nur einmal. Da sie aber in beiden Referenzen (one und two) vorhanden sind, muss genau angegeben werden, auf welche Referenz der Code sich bezieht. Die Verknüpfung mit sich selbst wird dadurch gelöst, dass die Verknüpfungsvariablen natürlich aus dem Paar *OrigVar* und *VarValue* bestehen, zur Unterscheidung der Werte von Fällen und Kontrollen die Variable *Low* referenziert wird: *one.Low>two.Low*. Damit bezieht sich one.*NPct* auf die Fälle, was man bei der Bezeichnung der Variablen *Cases* direkt berücksichtigen kann.

```
11  CREATE TABLE Ziel AS
12  SELECT
13    one.OrigVar,
14    one.VarValue,
15    one.VarFormat,
16    one.NPct AS Cases,
17    two.NPct AS Controls
18  FROM ZwischenView one, ZwischenView two
19  WHERE (one.OrigVar=two.OrigVar AND one.VarValue=two.VarValue
20                                 AND one.Low>two.Low);
21 QUIT;
```

Mit wenig Aufwand lassen sich nun sogar noch die Label der Variablen Smoke und HT in die Ausgabetabelle einfügen. Dazu wird aus Gründen der Übersichtlichkeit ein weiterer View definiert, der die Label aus den sogenannten Dictionary Columns bezieht:

```
CREATE VIEW label_view AS
SELECT name, label
  FROM DICTIONARY.COLUMNS
  WHERE UPCASE(LIBNAME)='WORK' AND
        UPCASE(MEMNAME)='LBW' AND
        UPCASE(NAME) in ('SMOKE' 'HT');
```

Der obige Code muss dann nur noch wie folgt ergänzt werden:

Zwischen Zeile 12 und 13:

```
        three.label,
```

in Zeile 18 wird der neue View aufgenommen

```
        label_view three,
```

und in die Klammer der WHERE Bedingung in Zeile 19:

```
    WHERE (three.name=one.OrigVar   AND
```

# 3 Schlussfolgerung

PROC SQL erfüllt alle vom Autor angeführten Bedingungen:
Es ist möglich, in wenigen übersichtlichen Schritten Variablen zusammenzufassen, die Daten zu transponieren und zusätzlich auch Label und Formate anzugeben. So vorbereitet, lässt sich der Datensatz später noch durch Hinzufügen von Odds Ratios und Kon-

fidenzintervallen der dazugehörigen Risikofaktoren aus der Tabelle der Risikoschätzer ergänzen [1,2].

Übersichtlich und leicht geht dies auch im Datastep, sowohl mit der einfachen Variante, als auch im Array-Konzept. Eine Einbeziehung von Variablenlabeln sowie eine angestrebte Endsortierung sind jedoch mit weiterem Aufwand verbunden.

PROC TRANSPOSE ist für Teilaspekte der Aufgabenstellung geeignet.

## Literatur

[1] Ramroth H (2008): Publikationsfertige Kombination von Häufigkeiten und Risiko-Kennwerten aus Ergebnissen von klinisch-epidemiologischen Studien; Proceedings der 12. KSFE; Aachen

[2] Ramroth H (2009) Makros zur Kombination von deskriptiven und analytischen Ergebnissen aus klinisch-/epidemiologischen Studien; Proceedings der 13. KSFE; Halle/Saale

[3] SAS-Example 65.4: Conditional Logistic Regression for m:n Matching

[4] http://www.urz.uni-heidelberg.de/statistik/sas-ah/01.03/transform.html (H. Stürzl)

[5] Zirbel D (2009): Lern the Basics of Proc Transpose; SAS Global Forum; Paper 060-2009