INTRODUKTION TIL SAS

MOGENS RING PETERSEN

August 2010

INDHOLDSFORTEGNELSE

```
 4      INFILE SUPDATA;
 5      INPUT NAVN $ FOEDDAG PERSNR HOEJDE VAEGT SYSTOLE DIASTOLE;
 6    RUN;
 7
 8    DATA MYSAS.DAT85ALL;
 9      SET MYSAS.DATA85 TEMP;
10    RUN;
11
12    PROC SORT;
13      BY NAVN PERSNR;
14    PROC SORT DATA=MYSAS.DATA86;
15      BY NAVN PERSNR;
16    RUN;
17
18    DATA MYSAS.TOTAL;
19      MERGE MYSAS.DAT85ALL(RENAME=(VAEGT=VAEGT85))
20            MYSAS.DATA86(RENAME=(VAEGT=VAEGT86))
21      ;
22      BY NAVN PERSNR;
23    RUN;
```

På lignende måde kan du bruge `DROP` og `KEEP` i parentes ud for de enkelte datasæt.. Eksempel:
`'DATA SUPDATA (DROP = HOJDE VAEGT)';`

PROC SQL

Nu skal vi se de samme eksempler på sammensætning af datasæt, som vi har lavet med datastep på side 35-38, men nu ved anvendelse af PROC SQL. Først skal vi dog lige vende tilbage til linie 18-23 i programmet på forrige side og understrege, at …

God programmerings-skik med indrykninger osv. hjælper dig til at undgå unødvendige fejl og sparer i sidste ende tid. Kommentarer i dit program, som fortæller kort, hvad der sker, kan være en god hjælp, når du vender tilbage til det igen efter flere måneder/ år.

Konkatenering af datasæt

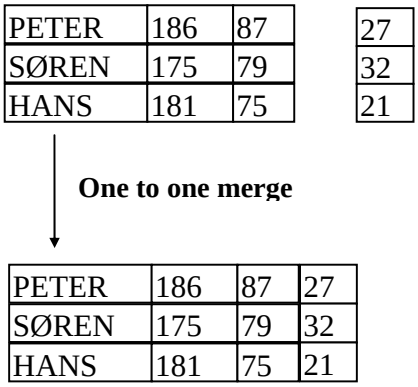
To eller flere datasæt *med samme variabelstruktur* kan forenes (konkateneres). De enkelte datasæt indlæses i den rækkefølge, de nævnes. Eksempel (sammenlign med eksemplet på side 35):

```
PROC SQL;
  CREATE TABLE MYSAS.SAMDATA AS
  SELECT *
    FROM MYSAS.DATA85
  OUTER UNION CORRESPONDING
  SELECT *
    FROM MYSAS.SUPDATA
  ;
  QUIT;
```

Her indlæses først samtlige observationer fra MYSAS.DATA85, derefter observationerne fra MYSAS.SUPDATA. Stjernen betyder alle variable (kolonner) fra datasættet.
Ordet 'CORRESPONDING', som kan forkortes til 'CORR' gør, at kolonnerne i de to datasæt MYSAS.DATA85 og MYSAS.SUPDATA bliver koblet sammen efter navnene på variablene i datasættet, så der ikke forekommer kolonner med det samme navn mere end én gang.
Hvis man prøver at lade være med at skrive ordet 'CORRESPONDING' vil PROC SQL i stedet lave dubletter af kolonnerne - så vi f.eks. får to kolonner, som hedder navn - men da der ikke kan være flere variable med det samme navn i et SASDATA-sæt, vil der gå kage i den, når SAS forsøger at lave et datasæt ud af det.
I PROC SQL følger programteksten standarden i SQL-sproget. Der er derfor oftest først ';' til allersidst i proceduren - og så er der komma imellem, hvis to datasæt eller variable nævnes lige efter hinanden (i modsætning til normalt i SAS). En hurtig og god gennemgang af SQL-sproget for begyndere finder du på http://www.w3schools.com/sql/default.asp .

'One-to-one merging' af datasæt, MERGE

Datasættene MYSAS.EXPDATA1 og MYSAS.EXPDATA2, som antages at have *forskellige variable*, kan blandes som vist nedenfor ('one-to-one merge'). Der vil normalt være det samme antal observationer i de to datasæt, men dette er intet krav.
For at få datasættene blandet sammen linie for linie laver vi lige en ekstra variabel, linie, som vi kun bruger som hjælp til at blande de to datasæt. Vi benytter os af, at når man læser et datasæt ind (både med 'input' og 'set'), så findes der en automatisk variabel '_N_', som angiver observationsnummeret. _N_ er 1 for første observation, 2 for næste osv. . Denne automatiske variabel bliver ikke skrevet ud i de færdige datasæt. Derfor må vi gemme _N_ i en ny variabel, som vi her kunne kalde LINIE. Denne nye variabel derimod bliver skrevet ud til datasættene, og vi kan derfor bruge den, når vi senere vil blande de to datasæt sammen.



I skemaet er vist, hvordan data bliver sat sammen. Variablen LINIE er ikke vist, selv om den er med i både EXPDATA1 og EXPDATA2. I datasættet EXPDATA1 har variablen LINIE værdien '1' for observationen med 'PETER 186 87' og værdien '2' for observationen med 'SØREN 175 79'. Tilsvarende har variablen LINIE værdien '1' for observationen med '27' (ALDER) i EXPDATA2 og '2' for observationen med '32'. Variablen LINIE er en numerisk variabel (idet _N_ er numerisk).

```
DATA EXDATA1;
  SET MYSAS.EXPDATA1;
```

```
  LINIE=_N_;
RUN;

DATA EXDATA2;
  SET MYSAS.EXPDATA2;
  LINIE=_N_;
RUN;

PROC SQL;
 CREATE TABLE MYSAS.COMBINE AS
 SELECT A.NAVN
       ,A.HOEJDE
       ,A.VAEGT
       ,B.ALDER
 FROM EXDATA1 A, EXDATA2 B
 WHERE A.LINIE=B.LINIE
 ;
 QUIT;

PROC PRINT;
  TITLE3 'Combined SAS dataset';
  TITLE5 'One-to-one merging of EXPDATA1 and EXPDATA2';
RUN;
```

I det resulterende datasæt MYSAS.COMBINE kombineres den første observation fra MYSAS.EXPDATA1 med den første observation fra MYSAS.EXPDATA2, og så fremdeles. Hvis et de oprindelige datasæt løber tør for observationer, vil dets variable få værdien 'missing'. Processen stopper først, når der ikke er flere observationer tilbage i nogen af de to datasæt. De tre afsluttende sætninger i programmet giver blot en udskrift til printfilen af observationerne i det resulterende datasæt.
Vi har brugt et synonym for datasætnavnene. Ved at skrive '`FROM EXDATA1 A`' kan vi nu referere til datasættet ved bare at skrive '`A`'. Ordren '`SELECT A.NAVN`' betyder derfor "Hent variablen '`NAVN`' fra datasættet `EXDATA1`". Du kunne også her have skrevet '`FROM EXDATA1 AS A`' – det havde betydet nøjagtigt det samme. På denne måde sparer du lidt skrivearbejde, især ved lange datasætnavne.

Match-merging af datasæt

Det er også muligt at blande datasæt ved 'match-merging'. Datasættene behøver i modsætning til datastep (jfr. side 36 og 38) ikke forinden at være *sorteret* efter den pågældende 'matching'-variable.

32	SØREN
27	PETER

87	186	PETER
79	175	SØREN
75	181	HANS

Match merging

27	87	186	PETER
32	79	175	SØREN
	75	181	HANS

Bemærk, at der er i forhold til før er kommet en variabel mere med i EXPDATA2, nemlig NAVN, som vi bruger til at koble de to datasæt sammen.

```
PROC SQL;
  CREATE TABLE MYSAS.MATCHED AS
  SELECT A.*
        ,B.ALDER
  FROM EXPDATA1 A

  FULL JOIN EXPDATA2 B
    ON
  A.NAVN=B.NAVN
;
QUIT;
```

Her benytter vi en en 'FULL JOIN' i vores SQL. Er der eksempelvis en værdi af NAVN i det første datasæt på 'PP', og der ikke er nogen observationer af NAVN i det andet datasæt med værdien på 'PP', vil der i det nye datasæt MYSAS.MATCHED alligevel forekommer en observation med et NAVN på 'PP'. De variable, der kommer fra det andet datasæt - hvor der altså ikke var en observation med NAVN på 'PP' - vil blot blive missing. Sammenlign med SQL-eksemplet nedenfor.

Match-merging af datasæt
– kun observationer, som forekommer i begge datasæt, er med.

```
PROC SQL;
  CREATE TABLE MYSAS.MATCHED AS
  SELECT *
    FROM MYSAS.EXPDATA1 A, MYSAS.EXPDATA2 B
  WHERE A.NAVN = B.NAVN
;
QUIT;
```

Her "matches" observationer, hvor variablen NAVN har samme værdi i de to datasæt med sætningen 'WHERE A.NAVN = B.NAVN'. Yderligere begrænses datamænden så samtidigt automatisk til kun at omfatte observationer, hvor en værdi af variablen NAVN ikke kun kan forekomme i det ene datasæt. Er der således en værdi af NAVN i det første datasæt på 'PP', og der ikke er nogen observationer af NAVN i det andet datasæt med værdien på 'PP', vil der i det nye datasæt MYSAS.MATCHED slet ikke forekommer en observation med et NAVN på 'PP'. Sammenlign med SQL-eksemplet fra før. I eksemplet ovenfor forekommer variablen NAVN både i det første og andet datasæt. Derfor kan vi ikke bare bruge 'SELECT *' (tag alle variable) , men må dele det mere op, som vist herunder:

```
PROC SQL;
  CREATE TABLE MYSAS.MATCHED AS
  SELECT A.*
        ,B.ALDER
    FROM MYSAS.EXPDATA1 A, MYSAS.EXPDATA2 B
  WHERE A.NAVN = B.NAVN
;
QUIT;
```

Nu tager vi nu alle variable fra A, men kun ALDER fra B, idet der også findes en variabel NAVN, som vi ikke ønsker, at PROC SQL også skal forsøge at hente fra B – og derved lave problemer – da vi jo allerede har hentet den fra A.

PROC SQL, Left Join:

Tit står man i en situation, hvor man næsten har et færdigt datasæt A, parat til et bestemt formål, men så mangler man bare lige en enkelt variabel. Denne variabel findes i et andet datasæt, B. I dette andet datasæt er der en hel masse andre variable. Dels er der en del variable, vi ikke har brug for til det formål, vi skal anvende vores færdige datasæt A og dels er der flere variable, vi allerede har i forvejen i datasæt A.

Findes der mon ikke en måde, vi bare lige kan klippe denne variabel ud af det nye datasæt B og så koble denne variabel på det rigtige sted i vores datasæt A?
Jo, her kan vi bruge 'LEFT JOIN'. (Vi kunne også have brugt 'RIGHT JOIN', men så skulle vi bare have byttet rundt på datasættene).

```
PROC SQL;
  CREATE TABLE LSTDATA AS

  SELECT A.*
        ,B.DOSELEVEL
  FROM PK-DATA A
  LEFT JOIN DERIVED.DOSEADMIN B
    ON
  A.NUMPT=B.NUMPT AND
  A.VISITNUM=B.VISITNUM
;
QUIT;
```

Her blev variablen DOSELEVEL hentet ud af datasættet DERIVED.DOSEADMIN og derefter koblet på datasættet PK-DATA, så vi får det færdige datasæt LSTDATA..

Har man yderligere en variabel, man skal have på fra et tredje datasæt, kan dette gøres i ét hug ved at forsætte med at koble på med kommandoen 'LEFT JOIN', som vist i eksemplet herunder:

```
PROC SQL;
  CREATE TABLE LSTDATA AS
  SELECT A.*
        ,B.DOSELEVEL
        ,C.TREAT
  FROM PK-DATA A

  LEFT JOIN DERIVED.DOSEADMIN B
    ON
  A.NUMPT=B.NUMPT AND
  A.VISITNUM=B.VISITNUM

  LEFT JOIN DERIVED.SUBJINFO C
    ON
  A.NUMPT=C.NUMPT AND
  A.VISITNUM=C.VISITNUM
;
QUIT;
```

Denne gang hentede vi variablen TREAT fra datasættet DERIVED.SUBJINFO.

Bemærk i øvrigt, at det ikke er nødvendigt at skrive 'RUN;' for at få 'PROC SQL' til at køre. Efter PROC SQL er slut, skriver vi 'QUIT;' for at få den til at holde op igen. Hvis man ikke skriver 'QUIT;' kan man blive ved med at sende "queries" (forespørgsler) til proceduren, uden først at skulle skrive 'PROC SQL;' – man kan med andre ord arbejde interaktivt med proceduren.

En smart ting ved PROC SQL i forhold til det tilsvarende i Datastep, er, at man med det samme kan se resultatet på skærmen, hvis man udlader 'CREATE TABLE LSTDATA AS' og nøjes med at "Selecte". Du behøver altså ikke åbne datasættet, udskrive eller lignende. PROC SQL sender resultatet til output-vinduet med det samme.

Altså i det første eksempel, kunne vi bare have skrevet:

```
PROC SQL;
  SELECT *
    FROM MYSAS.EXPDATA1 A, MYSAS.EXPDATA2 B
  WHERE A.NAVN = B.NAVN
;
QUIT;
```

Eller - for at gøre programmet næsten klart - kunne vi midlertidigt have udkommenteret program-trinnet, der laver datasættet:

```
PROC SQL;
/* CREATE TABLE MYSAS.MATCHED AS */
 SELECT *
    FROM MYSAS.EXPDATA1 A, MYSAS.EXPDATA2 B
```

```
  WHERE A.NAVN = B.NAVN
;
QUIT;
```

PROC SQL, andre nyttige tips.

Dubletter:

Har man et datasæt, hvor der er dubletter i, og det kunne f.eks. være det samme patientnummer, der optræder flere gange efter hinanden, kan man få en liste af patienter, hvor patientnummeret kun optræder én gang ved at anvende ordet '`DISTINCT`'.

```
DATA TEST;
INPUT PT;

CARDS;
 1001
 1001
 1001
 1002
 1002
 1003
 1004
 1004
 1005
 ;
RUN;

PROC SQL;
  CREATE TABLE PATIENTER AS
  SELECT DISTINCT *
  FROM TEST
;
QUIT;
```

Datasættet `PATIENTER` indeholder kun variablen `PT` og kommer til at se sådant ud:

```
  1001
  1002
  1003
  1004
  1005
```

Man kan også bruge `PROC SORT` med optionen `NODUBKEY` for at undgå dubletter (se hjælpe-system).

Beregninger med resultatet lagt i nye variable:

PROC SQL kan også være meget nyttig og simpel til at lave beregninger af forskellige ting og derpå gemme disse beregninger i en ny variabel i datasættet. Eksempelvis vil vi nedenfor tage datasættet SASHELP.CLASS, som vi jo altid kan hente uden først at skulle angive LIBNAME, og indlæse dette i et nyt temporært datasæt, som vi kalder TEMP.
Herefter bruger vi PROC SQL til at beregne middelværdien af vægt, højde og alder og desuden beregne minimum, maximum, forskellen mellem minimum og maximum – det der kaldes range – og standardafvigelsen for alder. Desuden vil vi gerne have angivet antallet af observationer (personer) og have delt materialet op på køn (variablen SEX). Alt dette kan nemt gøres med PROC SQL på denne måde:

```
DATA TEST;
  SET SASHELP.CLASS;
RUN;

PROC SQL;
  CREATE TABLE CLASS02 AS
  SELECT *
    ,MEAN(WEIGHT) AS MEAN_WEIGHT
    ,MEAN(HEIGHT) AS MEAN_HEIGHT
    ,MEAN(AGE)AS MEAN_AGE
    ,MIN(AGE) AS MIN_AGE
    ,MAX(AGE) AS MAX_AGE
    ,RANGE(AGE) AS RANGE_AGE
    ,STD(AGE) AS STD_AGE
    ,N(DISTINCT NAME) AS COUNT
  FROM TEST
  GROUP BY
    SEX
;
QUIT;
```

Med funktionen MEAN har vi her taget gennemsnit af WEIGHT, HEIGHT og AGE. Desuden har vi beregnet den mindste værdi med funktionen MIN() og den største værdi med funktionen MAX(). RANGE_AGE er den afstand, der er mellem MIN_AGE og MAX_AGE (altså minimum trukket fra maximum). STD() beregner standardafvigelsen og N() tæller antal af personer op. Var den samme person gået igen to gange, sikrer vi os med DISTINCT, at vedkommende kun bliver talt med én gang.

Hele matrialet deles op i mænd og kvinder ved statementet 'GROUP BY SEX', således at de forskellige statistiske beregninger (MEAN,MIN,MAX,RANGE,STD,N) bliver lavet for henholdsvis kvinder ('F') og mænd ('M').
Tallene fra beregningerne bliver som sagt lagt over i nye variable. Når vi således skriver følgende sætning i SQL: 'MEAN(WEIGHT) AS MEAN_WEIGHT', så betyder det "udregn gennemsnit af WEIGHT og læg derefter resultatet over i en ny variabel, som du skal kalde 'MEAN_WEIGHT'. Gør dette for henholdvis mænd og kvinder ('GROUP BY SEX')".

Sortering af variable:

Endelig kunne vil have sorteret materialet. Vi ville for eksempel gerne have sorteret efter køn med kvinder først og mænd sidst. Indenfor de to køn vil vi desuden gerne have de tungeste til at komme først og så ellers den næst tungeste osv.. Det kan gøres således:

```
DATA TEST;
  SET SASHELP.CLASS;
RUN;

PROC SQL;
  CREATE TABLE CLASS02 AS
  SELECT *
  FROM TEST
  ORDER BY
    SEX, WEIGHT DESCENDING
;
QUIT;
```

I SQL bruges kommandoen "ORDER BY". Herefter fortæller vi PROC SQL hvilken variabel, der først skal sorters efter, og det er i dette tilfælde 'SEX'. Dernæst er det WEIGHT, der skal sorteres efter. Uden ordet 'DESCENDING' ville de letteste komme ørst, og derpå den næst letteste osv.. Ordet 'DESCENDING', som vel nærmest kan oversættes med faldende, sørger for, at sorteringen foregår i omvendt rækkefølge. Havde vi brugt den på SEX også, altså skrevet 'ORDER BY SEX DESCENDING, ..' var 'M' kommet før 'F' i datasættet TEST.

Hvad er bedst at bruge - datastep eller PROC SQL?

Nu har vi set de samme ting lavet med datatrin og med PROC SQL. Hvad er så bedst at bruge? Svaret må være, at det er et smagsspørgsmål. Har man nemmere ved at bruge datastep, så bruger man bare det. Er man derimod øvet i at bruge SQL, så er det jo fint, at SAS giver en mulighed for at kunne fortsætte med at bruge SQL. De allerfleste SQL-ting, man kender i forvejen, virker også i PROC SQL.

Nogle ting er dog indlysende nemmere i et datatrin fremfor at bruge PROC SQL. Skal man således konkatenere flere datasæt, er det meget nemt bare at skrive 'SET' og bagefter anføre de datasæt, der skal konkateneres. Ved matchmerging kan det derimod virke som en befrielse, at man ikke skal sortere datasættene først, og når man først har lært sig, hvad man skal skrive i PROC SQL – og det kan der godt gå lidt tid med – så er det måske den metode, man vil foretrække.

ANDRE DATATRIN OG PROCEDURER

MACRO-SPROG

Når man skal gøre det samme flere gange i et program, er en mulighed at kopiere programstumpen ved at mærke den af som en blok (klikke med venstre musetast, holde den nede og trække en blok), kopiere den (Ctrl-C) og derpå ”paste” den flere gange (Ctrl-V).

Vi har f.eks. en programstump, der skriver et datasæt ud:

```
PROC PRINT DATA=DATA1;
  TITLE "Udskrift af DATA1";
  TITLE2 "Liste over mænd, der ryger";
RUN;
```

Vi skal nu også have skrevet 3 andre lister, og vi kopierer derfor programstumpen 3 gange, således her:

```
PROC PRINT DATA=DATA1;
  TITLE "Udskrift af DATA1";
  TITLE2 "Liste over mænd, der ryger";
RUN;

PROC PRINT DATA=DATA1;
  TITLE "Udskrift af DATA1";
  TITLE2 "Liste over mænd, der ryger";
RUN;

PROC PRINT DATA=DATA1;
  TITLE "Udskrift af DATA1";
  TITLE2 "Liste over mænd, der ryger";
RUN;

PROC PRINT DATA=DATA1;
  TITLE "Udskrift af DATA1";
  TITLE2 "Liste over mænd, der ryger";
RUN;
```

Vi har nu et stykke kode, hvor det samme står 4 gange. Derpå retter vi den til, så programmet passer til de ting, vi skal have skrevet ud. Vi sørger for at rette datasæt-navnene, både ved ’DATA=’ og inde i ’TITLE’, så alle de datasæt, vi gerne vil have skrevet ud, kommer med. Desuden er ’TITLE2’ forskellig ved hver udskrift, så der skal vi også har rettet for hvert eneste datasæt. Når vi er færdig, skulle programmet f.eks. gerne se sådan ud:

```
PROC PRINT DATA=DATA1;
  TITLE "Udskrift af DATA1";
  TITLE2 "Liste over mænd, der ryger";
RUN;
```

```
PROC PRINT DATA=DATA2;
  TITLE "Udskrift af DATA2";
  TITLE2 "Liste over mænd, der har vægtproblemer";
RUN;

PROC PRINT DATA=DATA3;
  TITLE "Udskrift af DATA3";
  TITLE2 "Liste over kvinder, der ryger";
RUN;

PROC PRINT DATA=DATA4;
  TITLE "Udskrift af DATA4";
  TITLE2 "Liste over kvinder, der har vægtproblemer";
RUN;
```

Vi har sparet tid ved at kopiere programstumpen, men der findes en bedre og mere effektiv måde til at gøre dette på - og det er ved at anvende macroer.

Ovenstående kunne i stedet være skrevet på denne måde:

```
%MACRO UDSKRIFT(DATANAVN,SEX,PROBLEM);
  PROC PRINT DATA=&DATANAVN.;
    TITLE "Udskrift af &DATANAVN.";
    TITLE2 "Liste over &SEX., der &PROBLEM.";
  RUN;
%MEND;

%UDSKRIFT(DATA1,mænd,ryger);
%UDSKRIFT(DATA2,mænd,har vægtproblemer);
%UDSKRIFT(DATA3,kvinder,ryger);
%UDSKRIFT(DATA4,kvinder,har vægtproblemer);

RUN;
```

Man opnår to ting ved at lave en macro. Dels er den noget hurtigere at skrive, men frem for alt er den hurtigere at rette i. Man får nemmere rettet det hele. Hvis vi f.eks. senere får at vide, at vi ikke skal bruge datasættet DATA4, men derimod et datasæt, som hedder DATA6, så skal man kun lige ind at rette ét sted:

```
%UDSKRIFT(DATA4,kvinder,har vægtproblemer);
  til
%UDSKRIFT(DATA6,kvinder,har vægtproblemer);
```

Uden macroer skulle man have rettet to steder. Dette her er bare et simpelt eksempel. Ved mere komplicerede programmer opdager man hurtigt, at macroer hjælper en til at holde langt bedre styr på det hele. Man glemmer meget nemt at rette et eller andet sted rundt i koden.

Sådan laves en macro med en macrovariabel

Man starter med at give macroen et navn og så sætte et procenttegn foran. I ovenstående tilfælde har vi kaldt vores macro for '%UDSKRIFT'. Macroen afsluttes igen med en '%MEND'; Efter '%MEND' kan man godt skrive navnet på macroen, altså '%MEND UDSKRIFT;', men det er ikke nødvendigt – kan dog bruges til at holde mere styr på det, hvis man har flere macroer samtidigt.

Til macroen er tilknyttet en række macrovariable. Disse defineres i parentes efter macronavnet. Man kan gøre det på forskellig måde, men den viste her er en god måde at gøre det på, hvor macrovariabelnavnene bliver skrevet efter hinanden i parantesen adskilt af komma. Når man så kalder macroen, og det gøres ved blot at skive selve navnet på macroen med et procenttegn foran, skal værdierne for de enkelte macrovariablene stå i den samme rækkefølge som ved definitionen. Altså i det første tilfælde, '%UDSKRIFT(DATA1,mænd,ryger); ', får første macrovariabel DATANAVN værdien 'DATA1', anden macrovariabel SEX får værdien 'mænd' og tredje macrovariabel PROBLEM får værdien 'ryger'.

I macroen kalder man så de enkelte macrovariable ved at skrive navnet på macrovariablen med en '&', foran, og macrovariablen afsluttes med et punktum. Hvis man glemmer punktummet, går det dog for det meste godt, men havde vi nu haft fire datasæt, som havde heddet 'MALE.1', 'MALE.2', 'FEMALE.1' og 'FEMALE.2' – altså hvor der er et punktum i selve datanavnet – og hvor vi så havde skrevet således her:

```
%MACRO UDSKRIFT(DATANAVN,NUMMER,SEX,PROBLEM);
  PROC PRINT DATA=&DATANAVN.&NUMMER.;
    TITLE "Udskrift af &DATANAVN.";
    TITLE2 "Liste over &SEX., der &PROBLEM.";
  RUN;
%MEND;

%UDSKRIFT(MALE,1,DATA,mænd,ryger);
%UDSKRIFT(MALE,2,DATA,mænd,har vægtproblemer);
%UDSKRIFT(FEMALE,1,DATA,kvinder,ryger);
%UDSKRIFT(FEMALE,2,DATA,kvinder,har vægtproblemer);
```

så var det ikke blevet rigtigt. 'DATA=&DATANAVN.&NUMMER.' var af macroprocessoren blevet oversat til 'DATA=MALE1' ved den første kald af '%UDSKRIFT'og 'DATA=MALE2', 'DATA=FEMALE1' og 'DATA=FEMALE2' ved de tre næste. Punktummet i datanavnene er ikke kommet med, da SAS havde opfattet punktummet som en markering for afslutning af macrovariabel-navnet.

Vi skulle i stedet have skrevet to punktummer, 'DATA=&DATANAVN..&NUMMER.', så var det blevet rigtigt, så datanavnene var blevet 'MALE.1', 'MALE.2', 'FEMALE.1' og 'FEMALE.2'. Husk derfor altid punktum efter macrovariabelnavnet.

Derimod ville det alligevel være gået godt, hvis vi havde undladt punktum de andre steder. Herunder en tilsvarende programstump som ovenfor, der vil blive tolket korrekt:

```
%MACRO UDSKRIFT(DATANAVN,NUMMER,SEX,PROBLEM);
  PROC PRINT DATA=&DATANAVN..&NUMMER.;
    TITLE "Udskrift af &DATANAVN";
    TITLE2 "Liste over &SEX, der &PROBLEM";
  RUN;
%MEND;

%UDSKRIFT(MALE,1,DATA,mænd,ryger);
%UDSKRIFT(MALE,2,DATA,mænd,har vægtproblemer);
%UDSKRIFT(FEMALE,1,DATA,kvinder,ryger);
%UDSKRIFT(FEMALE,2,DATA,kvinder,har vægtproblemer);
```

Sådan kører et program med macroer i, kort fortalt

Når man kører et program med macroer i, sker der det, at SAS først undersøger programmet for, om der forekommer macroer i det. Gør der det, bliver macroerne af macroprocessoren først oversat til almindelig kode. Det vil sige, at alle de steder, hvor der står en macrovariabel, vil værdien af macrovariablen blive sat ind.
I virkeligheden er processen en del mere kompliceret – interesserede kan se en mere udførlig forklaring i manualen eller i hjælpesystemet.

Denne programstump:

```
%MACRO UDSKRIFT(DATANAVN,SEX,PROBLEM);
  PROC PRINT DATA=&DATANAVN.;
    TITLE "Udskrift af &DATANAVN.";
    TITLE2 "Liste over &SEX., der &PROBLEM.";
  RUN;
%MEND;

%UDSKRIFT(DATA1,mænd,ryger);
%UDSKRIFT(DATA2,mænd,har vægtproblemer);
%UDSKRIFT(DATA3,kvinder,ryger);
%UDSKRIFT(DATA4,kvinder,har vægtproblemer);

RUN;
```

vil således af macroprocessoren - før den egentlige SAS-kørsel - blive oversat til:

```
PROC PRINT DATA=DATA1;
  TITLE "Udskrift af DATA1";
  TITLE2 "Liste over mænd, der ryger";
RUN;

PROC PRINT DATA=DATA2;
  TITLE "Udskrift af DATA2";
  TITLE2 "Liste over mænd, der har vægtproblemer";
RUN;

PROC PRINT DATA=DATA3;
```

```
  TITLE "Udskrift af DATA3";
  TITLE2 "Liste over kvinder, der ryger";
RUN;

PROC PRINT DATA=DATA4;
  TITLE "Udskrift af DATA4";
  TITLE2 "Liste over kvinder, der har vægtproblemer";
RUN;
```

Man kan også tildele værdier til macrovariable defineret på ovenstående måde ved at anvende lighedstegn. På denne måde kan man lave værditilskrivningen i vilkårlig rækkefølge. Eksempel:

```
%UDSKRIFT(DATA1, PROBLEM=ryger, SEX=mænd);
```

Her har vi først tildelt 'DATA1' til DATA og herefter tildeler vi værdier med metoden med lighedstegn. Man kan ikke gøre det omvendt. Altså når vi først går i gang med lighedstegn skal vi fortsætte med det resten af parantesen.

MPRINT

Hvis der er noget, man ikke rigtigt kan få til at virke, og som man er i tvivl om, at macro-processoren oversætter rigtigt, kan man få en oversættelse ved i programmet at skrive 'OPTIONS MPRINT;' – så kan man i logvinduet se, hvad macrokoden bliver oversat til. Det kan dog hurtigt blive for meget med store programmer, hvor der er anvendt en del andre macroer. Her bliver logvinduet hurtigt helt uoverskueligt med macro-oversættelser, så det kan være svært at finde eventuelle fejl i andre dele af koden.

Så er det rart at kunne slå det fra igen med ved at ændre i den 'OPTIONS MPRINT;', vi skrev før, så der nu står 'OPTIONS NOMPRINT;' Så kommer der ikke oversættelse af macrokode i loggen. Der findes også andre options, hvor man kan få oplysninger om macrokoden – se evt. i hjælpesystemet.

Macrosproget rummer mange detaljer. Her vil blot blive fremhævet nogle få ting.

Sådan laves en macrovariabel, som samtidig tildeles en værdi

Vi bruger her '%LET' og herefter kommer navnet på macrovariablen, 'VAR1' efterfulgt af '='. Teksten, som man tildeler macrovariablen, skal IKKE omsluttes af anførselstegn, som man ellers altid skal anvende ved tekstvariable, men det gælder altså ikke macrovariable. Bagefter har vi brugt macrovariablen i titlen.

```
%LET VAR1=Adverse Events;
  PROC PRINT DATA=AE;
    TITLE "Patiens with &VAR1.;";
RUN;
```

Sådan laves betingede spring, så selv programteksten er forskellig ved de enkelte kald af macroen

Nedenfor har udvidet programmet fra før, så værdien af macrovariablen `VAR1` automatisk bliver ændret, alt efter hvilke datasæt, vi kører. Når først macrovariablen er dannet, kan den nemlig få et andet indhold ved at tildele den dette på samme måde som før:

```
%MACRO DEMO(DATANAVN);
%IF %UPCASE(&DATANAVN.)=AE %THEN %DO;
  %LET VAR1=Adverse Events;
%END;
%IF %UPCASE(&DATANAVN.)=SAE %THEN %DO;
  %LET VAR1=Serious Adverse Events;
%END;

 PROC PRINT DATA=&DATANAVN.;
    TITLE "Patients with &VAR1.;";
  RUN;
%MEND;

%DEMO(AE);
%DEMO(SAE);
RUN;
```

Vi bruger her `%IF..%THEN..%DO..%END..`, som opfører sig på samme måde som `IF..THEN..DO..END`, men med den forskel, at her bliver disse programlinjer kørt før resten af programmet.

Et andet eksempel nedenunder, hvor der er flere betingede spring, der bliver oversat af macrofortolkeren, inden selve programmet kører. Her bruger vi også `%ELSE..%IF og %ELSE..`

```
%MACRO UDSKRIFT(DATANAVN,SEX,PROBLEM);
  PROC PRINT DATA=&DATANAVN.;
    WHERE UPCASE(SEX)="%UPCASE(&SEX.)";
      TITLE "Udskrift af &DATANAVN.";
      TITLE2 "Liste over personer, der &PROBLEM.";
    %IF %UPCASE(&SEX)=F %THEN %DO;
      TITLE3 "I denne undersøgelse indgår kun kvinder.";
    %END;
    %ELSE %IF %UPCASE(&SEX)=M %THEN %DO;
      TITLE3 "I denne undersøgelse indgår kun mænd.";
    %END;
    %ELSE %DO;
      TITLE3 "I denne undersøgelse er ikke angivet køn.";
    %END;
  RUN;
%MEND;

%UDSKRIFT(DATA1,M,ryger);
%UDSKRIFT(DATA2,m,har vægtproblemer);
%UDSKRIFT(DATA1,F,ryger);
```

```
%UDSKRIFT(DATA2, ,har vægtproblemer);

RUN;
```

Bemærk, at vi har sikret os, at der ikke er en, – og der er der altså tit ved store datamængder – der har skrevet F eller M med småt. I programmet her er der også taget hensyn til, at programmøren har skrevet med lille (som det er sket i det andet kald af %UDSKRIFT, hvor der står 'm' for SEX. Med sidste kald af udskrift kommer de data ud, hvor der ikke er angivet køn.

Ovenstående program bliver af macrofortolkeren oversat til:

```
PROC PRINT DATA=DATA1;
  WHERE UPCASE(SEX)="M";
    TITLE "Udskrift af DATA1";
    TITLE2 "Liste over personer, der ryger";
    TITLE3 "I denne undersøgelse indgår kun mænd.";
RUN;

PROC PRINT DATA=DATA2;
  WHERE UPCASE(SEX)="M";
    TITLE "Udskrift af DATA2";
    TITLE2 "Liste over personer, der har vægtproblemer";
    TITLE3 "I denne undersøgelse indgår kun mænd.";
RUN;

PROC PRINT DATA=DATA1;
  WHERE UPCASE(SEX)="F";
    TITLE "Udskrift af DATA1";
    TITLE2 "Liste over personer, der ryger";
    TITLE3 "I denne undersøgelse indgår kun kvinder.";
RUN;

PROC PRINT DATA=DATA2;
  WHERE UPCASE(SEX)="";
    TITLE "Udskrift af DATA2";
    TITLE2 "Liste over personer, der har vægtproblemer";
    TITLE3 "I denne undersøgelse er ikke angivet køn.";
RUN;
```

I stedet for at skrive 'WHERE UPCASE(SEX)="%UPCASE(&SEX.)";' kunne vi have brugt 'WHERE UPCASE(SEX)=UPCASE("&SEX."); '.

Det ville have givet det samme resultat, men bemærk, at '%UPCASE(&SEX.)' fortolkes og oversættes før selve programmet kører. Nedenfor programoversættelsen, hvor der er anvendt 'WHERE UPCASE(SEX)=UPCASE("&SEX.");' til sammenligning. Her er macrovariablen '&SEX' bare blevet fortolket og har fået indsat en værdi. Der er ikke, fordi det ene er bedre end det andet, men bare nævnt her for at uddybe, hvad der sker ved de forskellige program-udtryk.

```
PROC PRINT DATA=DATA1;
  WHERE UPCASE(SEX)=UPCASE("M");
    TITLE "Udskrift af DATA1";
    TITLE2 "Liste over personer, der ryger";
    TITLE3 "I denne undersøgelse indgår kun mænd.";
RUN;


PROC PRINT DATA=DATA2;
  WHERE UPCASE(SEX)=UPCASE("m");
    TITLE "Udskrift af DATA2";
    TITLE2 "Liste over personer, der har vægtproblemer";
    TITLE3 "I denne undersøgelse indgår kun mænd.";
RUN;

PROC PRINT DATA=DATA1;
  WHERE UPCASE(SEX)=UPCASE("F");
    TITLE "Udskrift af DATA1";
    TITLE2 "Liste over personer, der ryger";
    TITLE3 "I denne undersøgelse indgår kun kvinder.";
RUN;

PROC PRINT DATA=DATA2;
  WHERE UPCASE(SEX)=UPCASE("");
    TITLE "Udskrift af DATA2";
    TITLE2 "Liste over personer, der har vægtproblemer";
    TITLE3 "I denne undersøgelse er ikke angivet køn.";
RUN;
```

Macro, andet

Macrosproget er meget omfattende, så dette var kun en lille introduktion til det. Nogle få ting mere skal dog lige nævnes.

%STR()

Hvis man skal skrive en tekst i en macrovariabel, og macroprocessoren begynder at ændre på den, kan man bruge '`%STR()`' for at forhindre ændringer. I andre tilfælde er brugen af '`%STR()`' ikke nok, og da må man bruge andre macro-funktioner, som "beskytter" teksten endnu mere. Der er flere forskellige funktioner, som vi ikke vil komme ind på her. Normalt er det tilstrækkeligt at bruge '`%STR()`'.

Et lille eksempel vil illustrere, hvordan funktionen virker. Vi vil således f.eks. gerne lave en macrovariabel, der ser således ud (' er kun taget med for at kunne vise mellemrum og er ikke med i selve variablen):

```
'   NOW'
```

I programmet nedenunder prøver vi at lave sådan en makrovariabel 'TIME1' ved hjælp af '%LET', hvor der er de nødvendige mellemrum før 'NOW'. Herefter anvender vi den bagefter ved hjælp af %PUT:

```
%LET TIME1=   NOW;
%PUT TIME IS:&TIME1.;
RUN;
```

Med '%PUT' får vi skevet udtrykket bag ved '%PUT' ud - fortolket - til loggen. Først kommer noget tekst 'TIME IS:', som bare bliver skrevet ud, som det står, og derefter kommer udskriften af makrovariablen. Resultatet af ovenstående program er, at der kommer til at stå:

```
TIME IS:NOW
```

Vi ser her, at mellemrummene er blevet "ædt" undervejs, men det var ikke meningen. Vi ville netop gerne have haft disse mellemrum med. For at få det rigtigt kunne vi have brugt '%STR'.

```
%LET TIME1=%STR(   NOW);
%PUT TIME IS:&TIME1.;
RUN;
```

Med udtrykket indsat i macro-funktionen %STR() bliver mellemrummene ikke fjernet, og vi får resultatet:

```
TIME IS:   NOW
```

%SYSFUNC()

En anden ting, du vil støde på i de færdige programmer er funktionen %SYSFUNC(). Denne funktion, som først er kommet med i de senere SAS-versioner, kan bruges til at lave mange forskellige ting, bla. kan man finde pathname til en fil og lignende (pathname på denne Wordfil er således for øjeblikket 'X:\Documents\Course\SAS\Intro'). En anden nyttig anvendelse er, at man kan bruge denne macrofunktion til at få de almindelige SAS-funktioner, som vi kender, til at køre på en macrovariabel ved at "pakke dem ind i " %SYSFUNC()-funktionen. Et simpelt eksempel nedenunder.

```
%LET TIT=HELBREDSDATA FRA 2006;
%LET TIT=%SYSFUNC(TRANWRD(&TIT.,2006,2007));
RUN;
%PUT &TIT.;
```

Her har vi først lavet en makrovariabel 'TIT' indeholdende teksten 'HELBREDSDATA FRA 2006'. Nu vil vi så gerne skifte '2006' ud med '2007'. Dette kan vi bruge funktionen TRANWRD til. For at kunne bruge denne funktion, skal vi omgive funktionskaldet med en %SYSFUNC(), så kan man også få funktionen til at virke på en macrovariabel.
På denne måde kan vi få ændret '2006' til '2007'. Når vi kører overstående program, får vi nedenstående resultatet. Bemærk ingen anførselstegn (' eller "), da det er en macrovariabel - og her bruges anførselstegn ikke til angivelse af en tekststreng (jfr. %LET TIT=HELBREDSDATA FRA 2006;).

Med '%PUT &TIT.;' får vi skrevet indholdet af macrovariabel &TIT. ud til log-vinduet, så vi kan se resultatet:

```
HELBREDSDATA FRA 2007
```

PROC REPORT

Som omtalt tidligere, kan man med fordel anvende PROC REPORT i stedet for PROC PRINT , hvis man skal skrive store tabeller ud, der fylder mere end bredden af en side. Med denne procedure kan man nemlig definere udskriftbredden for de enkelte variable og få teksten til at ombrydes i de enkelte variabelkolonner, så den fylder flere linjer. Her er nogle flere detaljer:

```
 1  PROC REPORT DATA=DATA85 NOWINDOWS SPLIT="^" HEADSKIP;
 2    COLUMN NAVN FOEDDAG PERSNR HOEJDE VAEGT VOVERH;
 3    DEFINE NAVN    / "NAVN" WIDTH=8 ORDER=INTERNAL SPACING=0;
 4    DEFINE FOEDDAG / "FØDSELSDAG" WIDTH=11 FORMAT=Z6.0 LEFT;
 5    DEFINE PERSNR  / "NUMMER" WIDTH=8 FORMAT=Z4.0 RIGHT;
 6    DEFINE HOEJDE  / "HØJDE^(CM)" WIDTH=7 FORMAT=3.0 CENTER;
 7    DEFINE VAEGT   / "VÆGT^(HG)" WIDTH=6 FORMAT=3.0 CENTER;
 8    DEFINE VOVERH  / "VÆGT/HØJDE^(HG/CM)" WIDTH=10 FORMAT=5.3;
 9    DEFINE NOTE    / "KOMMENTAR" WIDTH=24 FLOW;
10  RUN;
```

I linie 1 står først, hvilket datasæt, vi gerne vil have skrevet ud. Hvis vi ikke skriver 'DATA=' og så et datasæt-navn, vil det sidst dannede datasæt blive skrevet ud – men det er altid en god skik at angive et specifikt datasæt, der skal benyttes, for hvis der er en fejl i programmet, der hvor DATA85 bliver dannet, så der ikke kommer noget datasæt ud, så tager SAS bare det forrige datasæt og bruger det i stedet for - og det kunne f.eks. have været et fra en undersøgelse af udbredelsen af kattedyr, som slet ikke havde noget med vores helbredsundersøgelse at gøre.
Det næste, der står er 'NOWINDOWS '. Undlades dette ord, kommer man ind i et menusystem, men det vil vi gerne undgå. 'SPLIT=' kender vi fra eksemplerne med PROC PRINT tidligere. Vi bruger her karakteren "^" til at lave linieskift i titlerne med (herom om lidt). 'HEADSKIP' gør, at der kommer en blank linie efter variabel-overskrifterne. Vi kunne også have skrevet 'HEADLINE', så havde vi fået en linie under overskrifterne - eller/og vi kunne have skrevet 'HEADSKIP', så var der kommet en blank linie under overskriften.
I linie 2 defineres de kolonner (variable), der skal være i rapporten, og de kommer på papiret i samme rækkefølge, som de angives efter ordet 'COLUMN'.

Fra linie 3 med DEFINE defineres for de enkelte variable, hvordan de skal skrives ud. Man kan godt undlade at skrive disse linier med DEFINE, men man finder hurtigt ud af, at så bliver der problemer med udskrivningen, når f.eks. den længde, som variablene har i datasættet er for lang. Er den således 200 karakterer lang – og man støder tit på en variabel, der er defineret med sådan en længde, selvom det længste, der aktuelt står i den måske kun fylder 24-36 karakterer eller mindre – vil PROC REPORT ikke lave output.
I linie 3 har vi defineret, at som overskrift skal der stå 'NAVN' – det er det, der står inde i " ". Herefter skal kolonne være 8 karakterer bred, WIDTH=8. Da vi gerne vil have navnene sorteret alfabetisk skiver vi 'ORDER=INTERNAL'. Skulle variablene derimod være i den samme orden, som de står i datasættet, skulle der have stået 'ORDER=DATA'.
I linie 3 har vi også skrevet 'SPACING=0', det gør, at kolonnen bliver rykket helt til venstre – ellers vil den være en lille smule indrykket.
I næste linie, 4, har vi med 'FORMAT=Z6.0' bedt om at få tallet skrevet ud i en bredde på 6, uden decimaler og med et foranstillet nul, hvis fødselsdagen er under '10 '. Eksempel: En foeddag på '71046 ' vil blive udskrevet som '071046'. Der står også 'LEFT', og det betyder, at overskrift og tallene i kolonnen vil blive venstrestillet.
I de næste linier er vist, at man også kan skrive 'RIGHT' og 'CENTER'.
I linie 6 bruger vi så '^' første gang. Da vi i første linie i PROC REPORT har defineret den som en "SPLIT-karakter", vil der ske det, at overskriften bliver delt her. Det er en meget nyttig ting at vide, når man skal lave sit output pænere.
Endelig er der i linie 9 vist, at man kan skrive 'FLOW'. Vi forestiller os, at der også i datasættet DATA85 er en variabel, som hedder NOTE, som er en kommentar. Sådan en kan pludselig godt være meget lang, for en bestemt person. Der kunne f.eks. have stået "Måling af blodtrykket gentaget efter 5 minutters liggende hvile". Havde vi så kun lavet en bredde på 24 karakterer, altså skrevet 'WIDTH=24' – og vel at mærke IKKE skrevet flow, havde der i kolonnen med variablen NOTE kun have stået:

```
KOMMENTAR

Måling af blodtrykket ge
```

Havde vi derimod tilføjet 'FLOW' i linien, var der blevet skiftet linie, når der ikke var plads til det næste ord. Med 'FLOW' havde udskriften for NOTE været:

```
KOMMENTAR

Måling af blodtrykket
gentaget efter 5
minutters liggende hvile
```

PROC TRANSPOSE

Rimelig tit oplever man, at data står omvendt af, hvad man gerne ville have haft dem til at stå. Altså i stedet for, at f.eks. de enkelte visits står ud af som kolonner, så kommer oplysningerne i stedet som observationer, altså nedad i stedet for udad. I dette tilfælde kan man benytte sig af PROC TRANSPOSE til at få vendt sine data.

Eksempel: Data bliver indlæst med INPUT-sætningen og arrangeret ligesom de står under CARDS. For at gøre det nemt, er der kun én patient, nr. 1, med i dette studie.

```
DATA VISITS;
  INPUT PT VISIT VISITDATE : DATE7.;

CARDS;
1 1 27JUL07
1 2 03AUG07
1 3 10AUG07
1 4 14AUG07
1 5 20AUG07
;
RUN;
```

Hvis du nu gerne vil finde ud af, hvor mange dage, der er mellem de enkelte visits, så står data ikke særligt hensigtsmæssigt. Bedre havde det været, hvis der kun havde været en linie (observation) for hver patient med datoerne for de enkelte visits ud af på samme linie.
Dette kan vi som sagt opnå ved at bruge `PROC TRANSPOSE`.

```
PROC TRANSPOSE DATA=VISITS OUT=TEMP1;
  BY PT;
RUN;
```

Ved at skrive '`BY`' angiver vi den variabel, vi gerne vil have "blive stående", de andre variable bliver så flyttet fra at stå lodret til nu at være vandret. Resultatet bliver:

```
Obs    PT    _NAME_       COL1    COL2    COL3    COL4    COL5

 1      1    VISIT           1       2       3       4       5
 2      1    VISITDATE   17374   17381   17388   17392   17398
```

Det hele står rigtigt og vi skal bare pynte lidt på resultatet med noget kode.

```
1  DATA TEMP2 (RENAME=(COL1=VISIT1 COL2=VISIT2 COL3=VISIT3
                       COL4=VISIT4 COL5=VISIT5));
2    SET TEMP1;
3    IF _NAME_ = 'VISIT' THEN DELETE;
4    DROP _NAME_;

5  PROC PRINT NOOBS;
6    FORMAT VISIT1--VISIT5 DATE7.;
7  RUN;
```

I linie `1` omdøbes de variable, som SAS har lavet, da datasættet blev vendt, `COL1-COL5`, så de har nogle mere sigende navne (`VISIT1-VISIT5`). Desuden har vi ikke brug for linien, hvor

NAME (en variabel dannet af SAS under PROC TRANSPOSE) er lig med 'VISIT'. Vi ved jo godt, at COL1 er visit1, COL2 er visit2 osv. – derfor sletter vi den i linie 3. Vi kunne også have skrevet 'IF UPCASE(_NAME_) = 'VISIT' THEN DELETE', men her er det jo et variabelnavn, hvor der ikke lige pludseligt kunne have været store bogstaver i. Bemærk i øvrigt, at SAS automatiske navne ofte begynder og ender med '_' - jfr. den tidligere omtalte _N_ ved merging under PROC SQL.
I linie 4 smider vi variablen _NAME_ væk – nu har vi ikke mere at bruge den til. I linie 5 har vi skrevet 'NOOBS'. Det er ikke nødvendigt, men så slipper vi for observationsnummeret og endeligt er der i linie 6 vist, hvordan man formatterer en række variable VISIT1, VISIT2, VISIT3, VISIT4 og VISIT5 på en nem måde med '--'.

Output ser nu sådan ud – og vi har opnået, hvad vi ville:

```
PT     VISIT1     VISIT2     VISIT3     VISIT4     VISIT5

 1    27JUL07    03AUG07    10AUG07    14AUG07    20AUG07
```

Nu er det en smal sag at finde ud af, hvor lang tid, der er mellem de enkelte visits:

```
 1  DATA TEMP2 (RENAME=(COL1=VISIT1 COL2=VISIT2 COL3=VISIT3
                        COL4=VISIT4 COL5=VISIT5));
 2    SET TEMP1;
 3    IF _NAME_ = 'VISIT' THEN DELETE;
 4    DROP _NAME_;
 5    DAGE_1-2 = VISIT2-VISIT1;
 6    DAGE_2-3 = VISIT3-VISIT2;
 7    DAGE_3-4 = VISIT4-VISIT3;
 8    DAGE_4-5 = VISIT5-VISIT4;

 9  PROC PRINT NOOBS;
10    FORMAT VISIT1--VISIT5 DATE7.;
11  RUN;
```

Og endelig – hvis vi kun ville skrive de patienter ud, hvor der var gået mere en 7 dage mellem to visits, kunne vi have skrevet følgende WHERE-statement under PROC PRINT (vi kunne også have lavet noget tilsvarende i datasteppet).

```
 9  PROC PRINT NOOBS;
10    WHERE ((DAGE_1-2 GT 7) OR
           (DAGE_2-3 GT 7) OR
           (DAGE_3-4 GT 7) OR
           (DAGE_4-5 GT 7))
      ;
11    FORMAT VISIT1--VISIT5 DATE7.;
12  RUN;
```

RETAIN

Nu har vi netop set, hvordan man kan vende værdierne i en variabel, så de hver især bliver til variable i stedet for med `PROC TRANSPOSE`. Her skal vi se et andet eksempel på, hvad man kan komme ud for og en anden metode til at løse det på.

I dette eksempel drejer det sig om patient nr. 1, som kommer på 5 visits i alt, hvor vedkommende får målt sin puls. Vi vil gerne undersøge ændringer fra besøg nr. 3, som vi kalder "baseline". Umiddelbart lyder det rimelig simpelt. Når vi kommer til visit nr. 3, så laver vi en ny variabel, som vi kalder baseline og sætter den lig med pulsen. Nu har vi så denne variabel, og ved næste besøg kan vi bare trække den aktuelle puls fra denne baseline. Programmet har vi lavet sådan her:

```
DATA TEST;
  INPUT PT VISIT PULS;

  IF VISIT EQ 3 THEN BASELINE = PULS;
  CHANGE_FROM_BASELINE = PULS - BASELINE;

CARDS;
1 1  95
1 2 105
1 3 100
1 4 110
1 5 120
;
RUN;
```

Det ser jo umiddelbart rigtigt ud, men hvis vi skriver datasættet TEST ud, ser det sådant ud:

```
Obs    PT    VISIT    PULS    BASELINE    CHANGE_FROM_BASELINE

1      1      1        95         .           .
2      1      2       105         .           .
3      1      3       100       100           0
4      1      4       110         .           .
5      1      5       120         .           .
```

Problemet er, at lige så snart SAS kommer til linie 4 i indlæsningen, altså obs 4 med værdierne '`1 4 110`', så bliver `BASELINE` sat lig med 'missing', altså '.' – og det var jo ikke meningen, men sådan virker SAS, og derfor bliver `CHANGE_FROM_BASELINE` også lig med missing for obs 4 og 5.

Dette kan man afhjælpe med at skrive '`RETAIN`' og så variablens navn. Når man således skriver '`RETAIN BASELINE`', så betyder det "bliv ved med at huske den værdi, der er blevet tildelt BASELINE". Programmet ser nu sådant ud:

```
DATA TEST;
  INPUT PT VISIT PULS;
  RETAIN BASELINE;
```

```
  IF VISIT EQ 3 THEN BASELINE = PULS;
  CHANGE_FROM_BASELINE = PULS - BASELINE;

CARDS;
1 1  95
1 2 105
1 3 100
1 4 110
1 5 120
;
RUN;
```

Og resultatet, hvis vi skriver datasættet TEST ud med en PROC PRINT vil denne gang være rigtigt:

```
Obs    PT    VISIT    PULS    BASELINE    CHANGE_FROM_BASELINE

1      1      1        95         .           .
2      1      2       105         .           .
3      1      3       100       100           0
4      1      4       110       100          10
5      1      5       120       100          20
```

Nu skal man så bare lige huske at nulstille BASELINE, når der kommer en ny patient. Dette kunne gøres ved først at sortere datasættet efter PT og "SET'te" det sorterede datasæt med en 'BY'. Herefter kan man direkte spørge, om det drejer sig om en ny patient ved at bruge SAS-udtrykket 'IF FIRST.PT'. (Tilsvarende kan man samtidig bruge 'IF LAST.PT', hvis det drejede sig om den sidste observation for en patient, når man i forvejen har "SET'et" datasættet med en 'BY'). Programmet kunne se sådant ud:

```
DATA TEST;
  INPUT PT VISIT PULS;

CARDS;
1 1  95
1 2 105
1 3 100
1 4 110
1 5 120
2 1 100
2 2 100
2 3 105
2 4 110
2 5 115
;
RUN;

DATA TEST2;
  SET TEST;
```

```
  BY PT;
  RETAIN BASELINE;

  IF FIRST.PT THEN BASELINE= . ;
  IF VISIT EQ 3 THEN BASELINE = PULS;
  CHANGE_FROM_BASELINE = PULS - BASELINE;
RUN;
```

Det har i dette tilfælde ikke været nødvendigt at sortere, da PT i forvejen stod i den rigtige rækkefølge. Hvis vi skrev datasættet TEST2 ud med en PROC PRINT ville vi få følgende resultat:

Obs	PT	VISIT	PULS	BASELINE	CHANGE_FROM_BASELINE
1	1	1	95	.	.
2	1	2	105	.	.
3	1	3	100	100	0
4	1	4	110	100	10
5	1	5	120	100	20
6	2	1	100	.	.
7	2	2	100	.	.
8	2	3	105	105	0
9	2	4	110	105	5
10	2	5	115	105	10

Tekststrenge

Fra tekst til tal og omvendt

Tit får man brug for at omsætte en tekst til en numerisk variabel og omvendt. Det gøres for data, der allerede findes i et SAS-datasæt, ved at anvende funktionen '`input()`', når man vil omsætte tekst til tal og funktionen '`put()`' til det modsatte, altså at omdanne en numerisk værdi til en character-værdi.

I nedenstående eksempel læser vi først to tekstværdier ind. Hvorfor er det tekstværdier? Jo, vi har sat en '$' for at tilkendegive, at vi vil have, at det skal være en tekstværdi. Det er altså ikke tallet 123 – hvis binære tal ville have været '`01111011`', hvis nu SAS havde haft en heltals-variabeltype – men nu er det i stedet derimod tegnet '`1`' efterfulgt af tegnet '`2`' og derpå tegnet '`3`', der kommer til at stå i variablen `MINTEKST1`, altså ASCII-tegn 49, 50 og 51 – med de binære værdier '`00110001`', '`00110010`' og '`00110011`'. Se evt. http://www.asciitable.com/. Herefter omsættes `MINTEKST1` til det numeriske tal 123, linie 3. Med informatet '`BEST.`' finder SAS selv det bedste informat for tallet. I linie 4 omsættes teksten `MINTEKST2` også til et tal, men dette tal er ikke 261007, men derimod tallet 17465, som er SAS's interne tæller, der angiver det antal dage, der er gået siden dag 0, som man på SAS Institute har besluttet skal være den 1. januar 1960.

```
1  DATA TILTAL;
2    INPUT MINTEKST1 $ MINTEKST2 $;
3      MITTAL = INPUT(MINTEKST1, BEST.);
```

```
4      MINDATO = INPUT(MINTEKST2, DDMMYY6.);
5  CARDS;
6  123 261007
7  ;
8  RUN;
```

Vi vil nu konvertere disse to nye tal, `MITTAL` og `MINDATO` tilbage i to nye tekstvariable, som vi kalder et lidt andet navn end før, nemlig `MINTXT1` og `MINTXT2`. Umiddelbart kunne man måske tro, at vi den anden vej skulle bruge character-informater (dem med $ foran), men begge veje bruger vi faktisk numeriske informater. Det kan godt virke forvirrende.

```
1  DATA FRATAL;
2    SET TILTAL;
3    MINTXT1 = PUT(MITTAL, BEST.);
4    MINTXT2 = PUT(MINDATO, DDMMYY6.);
5  RUN;
```

I stedet for at læse tallene ind igen, har vi bare hentet det forrige datasæt, `TILTAL`, ind i et nyt, som vi kalder `FRATAL`. Vi bruger funktionen put() til at lave tallet om til tekst.

Behandling af tekststrenge

Der findes flere praktiske funktioner til at behandle tekstrenge med. Vi skal se på nogle simle eksempler. For at gøre det mere simpelt bruger vi en tekststreng i funktionen. Denne tekststreng kunne f.eks. være ”AABB”, og den skal angives med ” ”, men her kunne i stedet i funktionen også have være indsat et variabelnavn, og altså uden ” ”, hvor så funktionen vil blive brugt på værdien af variablen i de enkelte observationer.

I nedenstående eksempler på forskellige funktioner til behandling af tekst fås resultatet ved udover de datatrin, der står i rammen også tilføjes proceduretrinnet:

```
PROC PRINT;
RUN;
```

SCAN()

Med denne funktion kan vi klippe den tekst ud, som er adskilt af en skillekarakter. Det kunne være mellemrum eller komma. Tallet til sidst angiver, om ordet er nr. 2, 3, 4 eller andet i tekststrengen:

```
DATA TEST;
 ORD2=SCAN("En række ord,adskilt af mellemrum eller komma",2);
 ORD3=SCAN("En række ord,adskilt af mellemrum eller komma",3);
 ORD4=SCAN("En række ord,adskilt af mellemrum eller komma",4);
RUN;
```

Resultatet her vil være:

```
Obs    ORD2     ORD3     ORD4

1     række    ord     adskilt
```

INDEX()

Med denne funktion kan vi se, om en bestemt tekststreng findes i den tekst, vi undersøger. Gør den det, får vi kolonnenummeret på, hvor den starter.

```
DATA TEST;
 FINDES1=INDEX("En række ord adskilt af mellemrum","En");
 FINDES2=INDEX("En række ord adskilt af mellemrum","EN");
 FINDES3=INDEX("En række ord adskilt af mellemrum","ord");
 FINDES4=INDEX("En række ord adskilt af mellemrum","r");
RUN;
```

Resultatet her vil være:

```
Obs    FINDES1    FINDES2    FINDES3    FINDES4

1        1          0          10         4
```

Bemærk, at det er ikke ligegyldigt, om der står ”En” eller ”EN” og desuden, at det er den første forekomst fra venstre i tekststrengen, der angives, jfr. FINDES4. Findes strengen ikke, jfr. ’FINDES2’, returneres værdien ’0’

SUBSTR()

Med denne funktion kan vi klippe et stykke ud af en bestemt tekststreng. Første tal angiver startposition, og andet tal angiver længden, der skal klippes ud.

```
DATA TEST;
 UDKLIP1=SUBSTR("En række ord adskilt af mellemrum",1,2);
 UDKLIP2=SUBSTR("En række ord adskilt af mellemrum",10,3);
 UDKLIP3=SUBSTR("En række ord adskilt af mellemrum",16,5);
RUN;
```

Resultatet her vil være:

```
Obs    UDKLIP1    UDKLIP2    UDKLIP3

1       En         ord       skilt
```

Bemærk, at det er ikke ligegyldigt, om der står ”En” eller ”EN” og desuden, at det er den første forekomst fra venstre i tekststrengen, der angives, jfr. FINDES4.

Der findes flere praktiske funktioner til at behandle tekstrenge med. Vi skal se på nogle simle eksempler.

LENGTH()

Med denne funktion kan vi finde længden af et tekststykke.

```
DATA TEST;
 LENGTH1=LENGTH("En række ord adskilt af mellemrum");
 LENGTH2=LENGTH("En række");
 LENGTH3=LENGTH("En");
RUN;
```

Resultatet her vil være:

```
Obs    LENGTH1    LENGTH2    LENGTH3

1        33         8          2
```

TRANWRD()

Med denne funktion kan vi søge og erstatte en tekststreng:

```
DATA TEST;
  OVERSAT=TRANWRD("HELBREDSDATA FRA 2006","2006","2007");
RUN;
```

Resultatet her vil være:

```
Obs           OVERSAT

1     HELBREDSDATA FRA 2007
```

Den tekststreng, der skal søges efter i tekststrengen (argument 1), står som funktionens 2. argument, mens den tekststreng, der så skal indsættes i stedet for som det 3. argument i funktionen.

TRANSLATE()

Med denne funktion kan vi oversætte enkelte karakterer.

```
DATA TEST;
  OVERSAT=TRANSLATE("HELBREDSDATA FRA 2006","197","206");
RUN;
```

Resultatet her vil være:

```
Obs           OVERSAT

1     HELBREDSDATA FRA 1999
```

I dette eksempel vil vi gerne have oversat ’2’ til ’1’, ’0’ til ’9’ og ’6’ til ’7’. Bemærk at det er 1. tegn funktionens 3. argument, der søges efter i tekststrengen (argument 1), og som så erstattes af 1. tegn i funktionens 2. argument. Derpå er det 2. tegn i funktionens 3. argument, der søges efter i argument1 og derpå erstattes af 2. tegn i funktionens 2. argument osv.
Det virker lidt forvirrende, at det er tegnene i funktionens 3. argument, der søges efter i tekststrengen, og tegnene fra 2. argument, der sættes ind i stedet for. Man ville umiddelbart forvente - ud fra hvordan SAS andre funktioner er opbygget - at det var lige omvendt?

!!

Med denne funktion (operator) kan vi sætte to tekststykker sammen til ét tekststykke.

```
DATA TEST;
  SAMTEKST1="HELBREDSDATA FRA" !! " 2007" !! ".";
  SAMTEKST2="HELBREDSDATA FRA" !! "   2007  " !! ".";
RUN;
```

Resultatet her vil være:

```
Obs           SAMTEKST1                    SAMTEKST2

1     HELBREDSDATA FRA 2007.    HELBREDSDATA FRA   2007  .
```

SAMTEKST2 illustrerer meget godt det resultat, vi ofte kommer frem til i virkeligheden, da de tekststykker, vi skal bruge, enten ikke er venstrestillet eller har mellemrum efter den egentlige tekst. Dette kan vi afhjælpe med de to næste funktioner.

LEFT()

Med denne funktion kan vi venstrestille en tekststreng. Vi benytter funktionen i eksemplet fra før og ser, hvad der sker:

```
DATA TEST;
  SAMTEKST1="HELBREDSDATA FRA" !! " 2007" !! ".";
  SAMTEKST2= "HELBREDSDATA FRA" !! " " !! LEFT("   2007  ") !! ".";
RUN;
```

Resultatet her vil være:

```
Obs           SAMTEKST1                    SAMTEKST2

1     HELBREDSDATA FRA 2007.    HELBREDSDATA FRA 2007     .
```

Nu fik vi bragt ’2007’ hen til ’HELBREDSDATA FRA’. Da teksten bliver helt venstrestillet, bliver vi nødt til lige at tilføje et ekstra mellemrum efter ’HELBREDSDATA FRA’. Nu mangler vi bare at fjerne de efterfølgende mellemrum efter ’2007’, og det kan vi gøre med funktionen TRIM().

TRIM()

Med denne funktion kan vi fjerne blanktegn til sidst. Vi har nedenunder brugt funktionen udenom det, vi har laver i forvejen med LEFT(). Det, der så sker, er, at først bliver teksten venstrestillet og derefter fjernes blanktegnene.

```
DATA TEST;
 SAMTEKST1="HELBREDSDATA FRA" !! " 2007" !! ".";
 SAMTEKST2=
    "HELBREDSDATA FRA" !! " " !! TRIM(LEFT("   2007  ")) !! ".";
RUN;;
```

Resultatet her vil være:

```
Obs          SAMTEKST1                  SAMTEKST2

1     HELBREDSDATA FRA 2007.    HELBREDSDATA FRA 2007.
```

Nu har vi opnået, hvad vi ville. Vi kunne ikke have gjort det i den modsatte rækkefølge, altså ”trimmet” først og så venstrestillet bagefter – idet vi ganske vist i første omgang ville få blanktegnene bag ’2007’ væk med TRIM(), men i anden omgang ville de blanktegn, der står foran ’2007’ ved hjælp af LEFT() blive skubbet bagved 2007.

COMPRESS()

Vi kunne også have brugt COMPRESS(). Med denne funktion kan vi fjerne blanktegnene på begge sider af ’2007’ samtidigt. Evt. blanktegn inde i selve teksten bliver også fjernet!

```
DATA TEST;
 SAMTEKST1="HELBREDSDATA FRA" !! " 2007" !! ".";
 SAMTEKST2="HELBREDSDATA FRA" !! " " !! COMPRESS("   2007  ")!! ".";
RUN;;
```

Resultatet her vil igen være:

```
Obs          SAMTEKST1                  SAMTEKST2

1     HELBREDSDATA FRA 2007.    HELBREDSDATA FRA 2007.
```

- SLUT -

REGISTER