Statistics Norway
Statistisk sentralbyrå

Håndbok i SAS

Del 1: Innføring

Forord

SAS er et av de viktigste edb-verktøy i Statistisk sentralbyrå. Det benyttes i samtlige avdelinger, i en rekke ulike sammenhenger. SAS kan brukes til alt fra utvalgstrekking, dataregistrering, validitetskontroll, kobling, omkoding og all annen databearbeiding, via tabeller, til fremskrivninger, modellsimulering og avansert statistisk analyse. SAS finnes på en mengde forskjellige edb-plattformer og kan benytte data lagret i andre databaseverktøy, som for eksempel Oracle. Dette gir store muligheter til fleksible løsninger.

Alle mulighetene innebærer at dokumentasjonen av SAS har antatt kolossale dimensjoner. Brukerveiledningene er svært mange og tunge, også bokstavelig talt. Det kan også være vanskelig å finne fram i det interaktive hjelpesystemet under Byrånettet. Det har derfor lenge vært et ønske om en mer håndterlig håndbok spesielt tilpasset våre behov.

SAS-håndboken kommer nå i to deler: **Håndbok i SAS Del 1: Innføring** og **Del 2: Oppslag**. Innføringen kan leses fra perm til perm for den som vil lære seg SAS på egen hånd. Den kan også være nyttig for de som kan SAS fra før, de mange eksemplene kan gi nyttige tips om hvordan ting kan gjøres. Oppslagsboken er en grundig gjennomgang av detaljene, den inneholder også mange lærerike eksempler.

Når en skal beskrive engelskspråklige edb-verktøy, vil det alltid være et dilemma hvorvidt begrep og termer skal oversettes eller ikke. Vi har tilstrebet å bruke norske ord og uttrykk, men enkelte ord (*options*, *view*, *batch* og noen få andre) er det ennå ikke gode, innarbeidete oversettelser for. Disse ordene har vi derfor beholdt. Siden SAS manualene (sic!) er på engelsk, vil dette ikke være noen stor ulempe, det vil snarere gjøre det lettere å forstå det som står i manualene.

Denne håndboken er opprinnelig skrevet av Liv Daasvatn. Den er nå oppdatert til SAS versjon 8.1.

Oslo, 5. januar 2001

Kristian Lønø

Innhold

1. Noen sentrale begrep

1.1 SAS program

SAS program bygges opp av instruksjoner som kalles SAS setninger. Setningene er de minste byggestenene i et SAS program. Flere setninger sammen utgjør det som er de største byggestenene, kalt *steg*. Det finnes to forskjellige typer steg:

- DATA-steg
- PROC-steg

Sterkt forenklet kan en si at DATA-steg oftest benyttes til forbehandling av data (kobling, reorganisering, etc) mens PROC-steg oftest benyttes til å lage tabeller og analyser.

Et program kan bestå av ett eller mange steg

SAS program skal lagres på filer med filhale **.sas** (Dette skjer automatisk hvis vi skriver programmet i SAS' program editor vindu og lagrer programmet derfra.)

1.2 DATA-steg

Den generelle regelen er at DATA-steget benyttes til forbehandling av data mens PROC-steget benyttes til tabeller og analyse. Uttrykket *forbehandling* er i denne sammenheng et vidt begrep! DATA-steg kan for eksempel benyttes til å trekke tilfeldige utvalg, til å gjøre data lagret på 'uleselige' format leselig, til å finne hvor mange dager det er mellom 9.mars 1954 og 30.november 1962 osv osv.

Det er altså oppgaven, hva vi ønsker å gjøre, som avgjør om vi skal bruke DATA-steg eller PROC-steg. (Noen få oppgaver, for eksempel aggregering, kan utføres både i DATA-steg og i PROC-steg. I slike tilfelle vil andre momenter være avgjørende for om vi bruker det ene eller det andre.) Følgende er huskeregler for når vi *må* bruke DATA-steg:

- Lage SAS datasett av rådata (rådata er alle data som ikke er SAS datasett)
- Lage nye (avledete) variable
- Koble datasett
- Reorganisere datasett

Det er ofte forbehandlingen av data som er mest krevende, både med hensyn på tid og omtanke. Det er dermed DATA-steget som innebærer mest programmering. Mens PROC-steg ofte består av kun få linjer, kan DATA-steg gjerne være lange. (Men skal du gjøre noe veldig stort og komplisert i et DATA-steg, kan det være lurt å dele programmet i flere mindre biter!)

Et DATA-steg starter *alltid* med en DATA-setning.

1.3 PROC-steg

Mens DATA-steget benyttes til forbehandling av data, benyttes PROC-steget til tabeller, analyse og diverse annet. PROC står for procedure, prosedyre. SAS har hundrevis av slike ferdigskrevne prosedyrer (proc'er) til de forskjelligste formål. Vil du skrive ut observasjoner fra et SAS datasett bruker du PROC PRINT. Vil du lage frekvenstabeller, bruker du PROC FREQ. Vil du sortere, bruker du PROC SORT. PROC MEANS beregner gjennomsnitt og annen statistikk, mens PROC TABULATE lager tabeller. PRINT, SORT, FREQ, MEANS og TABULATE er de aller vanligste proc'ene. Men det finnes i tillegg utallige proc'er til spesielle formål: PROC REG til regresjonsanalyse, PROC ANOVA til variansanalyse, PROC NLIN til ikke-lineær regresjon, PROC IML til matrisehåndtering osv osv. Dersom vi vil gjøre kompliserte analyser, kan en av utfordringene være er å finne den 'rette' proc. (Til hjelp finnes det en veiledning i SAS onlinedoc, som ligger under IT-info, SAS på Byrånettet. Velg Base SAS Software og så Procedures Guide og Concepts).

Et PROC-steg starter *alltid* med en PROC-setning.

1.4 SAS setninger

SAS er et programmeringsspråk der vi kaller instruksjonene *setninger* (engelsk: *statement*). Et SAS program består altså av setninger. Det er ved hjelp av setningene vi gir beskjed til SAS hva som skal gjøres i programmet. Viktige huskeregler:

- *Nesten alle* setninger starter med et SAS *nøkkelord*
- *Alle* setninger avsluttes med et semikolon (;)

SAS nøkkelord er reserverte ord som SAS gjenkjenner som ord med helt spesiell betydning. I det følgende vil SAS nøkkelord bli skrevet med store bokstaver, hvilket også er gjennomført i SAS-manualene. (Dette er *kun* av pedagogiske årsaker! I programmene kan nøkkelordene gjerne stå med små bokstaver.) Setningene har navn etter nøkkelordet de starter med: INPUT-setningen, LABEL-setningen osv. Det er forskjellige typer setninger:

- Setninger som *kun* kan brukes i DATA-steg
- Setninger som *kun* kan brukes i PROC-steg
- Setninger som kan brukes både i DATA-steg og i PROC-steg
- Setninger som brukes *utenfor* steg, kan kalles globale setninger

1.5 SAS datasett

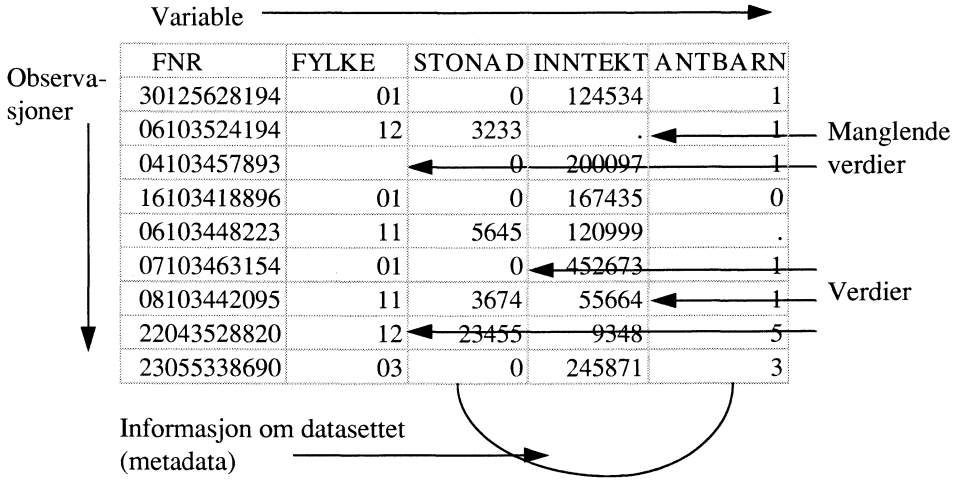
Data som er lagret i SAS' eget lagringsformat kalles SAS datasett. Et SAS datasett kan tenkes på som en tabell med et uformelig vedheng. Tabellen er selve dataene, mens vedhenget er informasjon *om* dataene (*metadata*). Linjene (recordene) i tabellen kalles *observasjoner*, mens kolonnene (feltene) kalles *variable*. Innholdet i en tabellcelle (selve dataene) kalles *verdi*. SAS datasett inneholder altså både dataverdier og beskrivende informasjon om dataene.

Observasjoner *nedover* og variable *bortover*

Det er ikke alltid det er noe innhold i alle tabellcellene, det kan være celler som mangler verdi. Dette kalles i SAS *missing values*. Det er viktig å merke seg hvordan SAS representerer slike manglende

verdier: Dersom variabelen er numerisk (omsetning, mengde, alder o.l.) representeres manglende verdi med et punktum, mens hvis variabelen er av type karakter (kommunenr, fødselsnr, varekode o.l.) er manglende verdi rett og slett blank.

Slik kan vi tenke oss et SAS datasett:



FNR	FYLKE	STONAD	INNTEKT	ANTBARN
30125628194	01	0	124534	1
06103524194	12	3233	.	1
04103457893		0	200097	1
16103418896	01	0	167435	0
06103448223	11	5645	120999	.
07103463154	01	0	452673	1
08103442095	11	3674	55664	1
22043528820	12	23455	9348	5
23055338690	03	0	245871	3

Når SAS datasett lagres på disk får de filnavn med hale **.sas7bcat** eller **.ssd04** (avhengig av SAS versjon). Dette skjer helt automatisk så det er ikke noe vi behøver tenke på, men det er nyttig å vite for å gjenkjenne SAS datasett i en liste over filer.
SAS datasett kan ha lang eller kort levetid, dvs de kan være permanente eller temporære. Temporære SAS datasett varer bare så lenge vi er inne i SAS, når vi avslutter SAS, slettes de. Vi avgjør selv om SAS datasett vi lager skal være permanente eller temporære.

- Temporære datasett slettes når SAS avsluttes 1-leddet navn
- Permanente datasett varer (og varer...) 2-leddet navn

Hver gang vi skal lage et SAS datasett, sier vi i SAS programmet hva datasettet skal hete. Dersom navnet kun har ett ledd, blir datasettet temporært. Oppgir vi et 2-leddet navn (leddene adskilt med punktum), blir datasettet permanent.

Ikke lagre datasett *permanent* uten at det virkelig trengs!

Vanlige sekvensielle ('flate') filer, omtales i SAS som *rådata*. Dersom dataene som skal brukes er rådata og ikke SAS datasett, blir den første oppgaven å gjøre dataene tilgjengelige for SAS. Dette betyr at det må lages SAS datasett (eller SAS view) av dem. Straks dataene foreligger som SAS datasett eller view, kan vi bruke SAS til å analysere, lage tabeller osv. (SAS view er en slags SAS datasett som nesten ikke tar plass, forklares i eget punkt.)

1.6 Eksempel på et SAS program

Selv om vi i dette heftet skriver alle nøkkelord og alt som er bestemt av SAS med store bokstaver, mens vi bruker små bokstaver på det vi selv har bestemt, er dette ingen regel. For SAS er det helt likegyldig om setningene skrives med små eller store bokstaver. Det er heller ingen andre regler for hvordan setningene skal skrives, rent utseendemessig. Imidlertid kan det være lurt å innarbeide gode vaner helt fra begynnelsen av. Ved å følge noen få retningslinjer, vil programmet både bli lettere å forstå og lettere å feilsøke. Slik kan et SAS program kan se ut:

```
***------------------------------------------------------------------**;
*  HOTELLSTATISTIKKEN                                                  ;
*                                                                      ;
*  Programfil: $REISELIV/hotell/prog/p01les.sas                        ;
*  Skrevet:    7. august 1995, LDa                                     ;
*  Endret:                                                             ;
*                                                                      ;
*  Lager SAS datasett av rådata, lager noen nye variable.              ;
*                                                                      ;
*  Inndata: $REISELIV/hotell/wk12/juni95.dta                           ;
*  Utdata:  $REISELIV/hotell/wk12/juni95.sas7bcat                      ;
*                                                                      ;
***------------------------------------------------------------------**;

DATA perm.juni95 ;
   INFILE '$REISELIV/hotell/wk12/juni95.dta' TRUNCOVER LRECL=200 ;
   INPUT @1   lopenr     $CHAR6.
         @7   hotellgr   $CHAR1.
         @13  kommune    $CHAR4.
         @34  ank_nord        5.
         @39  ank_utl         5.
         @192 omsetn          9.
         ;
   LABEL  lopenr    = 'Løpenr'
          hotellgr  = 'Hotellgruppe'
          kommune   = 'Kommune'
          ank_nord  = 'Norske gjester'
          ank_utl   = 'Utenlandske gjester'
          omsetn    = 'Omsetning'
          fylke     = 'Fylke'
          gjester   = 'Totalt antall gjester'
         ;
   LENGTH  fylke $ 2 ;
   fylke = SUBSTR(kommune,1,2) ;
   gjester  = SUM(ank_nord,ank_utl) ;

PROC FREQ DATA=perm.juni95 ;
   TABLES hotellgr  fylke  fylke*hotellgr ;
   TITLE  'HOTELLSTATISTIKKEN           Juni 1995 ';
   TITLE2 'Antall hotell pr hotellgruppe og fylke. ';

PROC MEANS DATA= perm.juni95  MAXDEC=0 FW=9 ;
   TITLE2 'Statistikk på alle numeriske variable.';

RUN;
```

Program-eksempelet ovenfor følger visse regler når det gjelder utseende. Dette gjør programmet lettere å lese. Gode leveregler for hvordan program bør se ut kan oppsummeres slik:

- Skriv info i en kommentarblokk først i programmet.
- Ikke flere setninger på samme linje.
- Kun DATA, PROC, RUN og kommentar-setninger skrives helt ut til venstre, alle andre setninger rykkes inn med et par blanke tegn.
- Hvis flere steg: ha minst én blank linje mellom hvert steg.
- Hvis DO-løkke: DO og END på egne linjer, alt inni løkken innrykket.
- Hvis IF-THEN/ELSE: IF og ELSE IF, ELSE IF, ELSE på egne linjer.
- Heller flere små program enn ett stort.

2. SAS datasett og view

2.1 Hvordan lage et SAS datasett?

SAS datasett lages vanligvis i DATA-steget. Skal vi for eksempel lage SAS datasett av rådata, *må* vi gjøre dette i et DATA-steg. Eksempelet i forrige punkt inneholder et DATA-steg som viser hvordan vi kan lage SAS datasett av rådata. Skal vi lage et koblet SAS datasett, *må* vi gjøre dette i et DATA-steg. Vi kan sikkert si at 9 av 10 SAS datasett lages i DATA-steg. Men vær klar over at det også er mulig å lagre resultatet av et PROC-steg i et SAS datasett. Det gjør vi ved å skrive `OUT=datasettnavn` et eller annet sted i proc'en (avhengig av hvilken proc det dreier seg om).

2.2 Navn på datasett, navn på variable

Det er i DATA-setningen vi sier hva SAS datasettet (eller SAS datasettene) vi vil lage skal hete. Når vi skal gi navn til SAS datasett og variable må vi holde oss innenfor visse regler, regler for **SAS navn**. Et SAS navn kan være på maksimum 32 tegn, kun bokstaver, tall og understrek er lov (bindestrek er altså *ikke* lov), navnet kan ikke starte med et tall, æøå er ikke lov. Følgende regler gjelder:

- Maks 32 tegn
- Kun bokstaver, tall og understrek (_)
- Må starte med bokstav eller understrek (_)
- Ikke æøå eller ÆØÅ
- Unngå SAS nøkkelord

Det er ikke bare navn på SAS datasett og variable som følger disse reglene, de gjelder også for *libref'er*, *fileref'er, arrayer* og *formatnavn*, begrep vi vil forklare senere. Librefs, filerefs og formatnavn kan ha maksimalt 8 tegn.

2.3 Permanente eller temporære datasett?

Et SAS datasett kan være temporært eller det kan være permanent. Et permanent SAS datasett er lagret permanent på disk, slik at det kan brukes dag etter dag. Det vil eksistere (og okkupere diskplass) inntil du selv aktivt sletter det(*). Temporære SAS datasett varer bare så lenge SAS 'er aktiv'. Det vil si at temporære datasett varer til du avslutter SAS. Hvis du kjører SAS i batch, varer temporære datasett så lenge SAS-jobben går.

Det som avgjør om et SAS datasett skal være temporært eller lagres permanent, er om vi i SAS programmet gir datasettet navn som består av ett eller to ledd. Har navnet bare ett ledd blir datasettet temporært, består navnet av to ledd (med punktum mellom leddene) blir datasettet lagret permanent. Det første leddet kalles *libref*. Det angir hvor i filsystemet datasettet skal lagres, det vil si hvilken katalog datasettet skal lagres på, mens det andre leddet angir selve navnet til SAS datasettet. En libref opprettes med LIBNAME-setningen, der vi først oppgir navnet på libref'en. Navnet på libref'en kan vi selv velge, men det må holde seg innenfor reglene om max 8 tegn osv. Etter libref'en kommer navnet på katalogen der datasettene er lagret eller skal lagres. Navnet på katalogen *må* stå i fnutter (slike: '). Eksempel på en LIBNAME-setning: `LIBNAME hotell '$REISELIV/hotell/wk12' ;`

SAS har sitt eget vindu for å liste opp SAS datasett ('filemanageren', lib-vinduet). I dette vinduet ser vi alle libref'ene vi har satt opp. (Vi husker at en libref er første leddet i navnet på et permanent SAS

datasett.) Hvis vi så velger en libref, får vi opp en liste over alle SAS datasett som ligger lagret på den libref'en. I denne listen er det selve navnet på SAS datasettene som vises (altså andre ledd i det to-leddete navnet). Bemerk at inne fra SAS ser vi ikke noe til endelsen .sas7bdat, filhaler på SAS datasett ser vi bare i filoversikter utenfor SAS (LIST, Windows filemanager, ls-kommandoen etc).

(*) Vær obs på reglene for lagring av data på unix. Ved å følge navnestandarden kan vi sørge for at arbeidsdata lagres en viss tid og deretter slettes automatisk.

2.4 Variable

Kolonnene, feltene, i SAS datasett heter variable. Det er 3 forskjellige typer variable: de numeriske, karakter- og dato-variable. Vi sier en variabel kan være i numerisk format, karakter-format eller dato-format.

- numerisk
- karakter
- dato

Bemerk at forskjellen mellom numeriske og karaktervariable *ikke* ligger i om variabel*verdiene* inneholder tall eller ikke! Selv om en variabel er i karakter-format, inneholder den svært ofte bare tall. Det som er avgjørende for om en variabel skal være numerisk eller karakter, er om vi skal benytte variabelen til *beregninger* eller ikke. Variable som lønn, skatt, timeverk, moms, toll, priser og liknende er størrelser vi helt klart kan tenke oss å skulle regne på. Slike variable skal være numeriske, i motsetning til variable som fødselsnummer, kommunenummer, varegruppe, bedriftsnummer, næringskode og liknende som selv om de kun inneholder tall, ikke skal inngå i beregninger.

***Kun* de variable som skal inngå i *beregninger* skal være numeriske!**
***Alle* andre variable skal være av type karakter.**

2.5 SAS view

Et SAS view er et slags luftig bilde av data. Et view inneholder kun *beskrivelser* av dataene, referanser til hvor dataene befinner seg, men ingen data. Et view kan referere til data lagret som SAS datasett, men det kan like gjerne referere til data lagret på sekvensielle filer eller i Oracle-tabeller. Siden view bare inneholder beskrivelser av data tar et view mindre plass enn et datasett. Å bruke view kan derfor være et nyttig alternativ for å spare lagringsplass. Selv om et view ikke inneholder data kan det likevel brukes på nøyaktig samme måte som et vanlig SAS datasett: I det øyeblikk programmet skal bruke et view, hentes dataene.

3. Options, funksjoner og format

3.1 Options

Options betyr frivillig (optional) valg. I SAS finnes det uendeligheter av slike frivillige valg. De finnes på 3 forskjellige nivåer:

- options som gjelder hvert enkelt setning
- datasett options
- system options

Det er altså options på setnings-nivå, datasett-nivå og system-nivå.

3.1.1 Options på setnings-nivå

Options som gjelder for hvert enkelt setning står beskrevet under den setning der de hører hjemme. Noen options er bare et enkelt ord, for eksempel option'en `LABEL` i PROC PRINT. Andre options har med et likhets-tegn og krever at vi angir en verdi, for eksempel option'en for å angi antall desimaler i utskriften fra PROC MEANS, `MAXDEC=` Vil vi ha tallene med én desimal, skriver vi der `MAXDEC=1` I andre options skal vi oppgi et spesielt tegn, for eksempel hva som er benyttet som skilletegn mellom feltene i en rådatafil. Da må vi omgi tegnet med fnutter ('). I INFILE-setningen der vi angir navnet til en rådatafil vi skal lese inn, kan vi oppgi at skilletegnet i rådatafilen er komma. Dette gjør vi med option'en `DELIMITER=` for eksempel slik: `INFILE '$SOSHJELP/wk9/k0101.dta' DELIMITER=';' ;`

3.1.2 Datasett options

I motsetning til options på setnings-nivå, er datasett options lette å få oversikt over, det er kun noen få av dem. De er til gjengjeld meget viktige og brukes svært ofte. Datasett options benyttes hvis vi ønsker å arbeide på *deler* av et datasett (kun enkelte variable, bare noen få observasjoner). Datasett-options kan også benyttes for å endre navn på variable. Overalt der vi har skrevet navnet på et SAS datasett, kan vi skrive datasett options. Datasett options må alltid stå i (vanlige) parenteser. Huske-regler for datasett options:

- Skrives rett etter navnet på SAS datasettet
- Omgis av parenteser (gjerne flere options i samme parentes)
- Brukes når vi vil arbeide på deler av et SAS datasett
- Brukes til å endre navn på variable
- Gjør programmet sikrere og mer effektivt
- De aller viktigste:

```
KEEP=variabelliste
DROP=variabelliste
RENAME=(gammeltnavn1=nyttnavn1 gammeltnavn2=nyttnavn2 osv.)
OBS=antall
```

Eksempel:

```
PROC PRINT DATA=hotell.mars95 (OBS=70 KEEP=lopenr omsetn gjester ant_seng) NOOBS LABEL;
   TITLE 'Mars 1995: Sjekk  om omsetning, gjester antall senger ser rimelig ut !' ;
RUN;
```

Dette programmet skriver ut de første 70 observasjoner av datasettet (OBS=70) og det er bare utvalgte variable som skrives ut (KEEP=lopenr omsetn gjester ant_seng) Legg merke til NOOBS og LABEL Disse står *utenfor* parentesen, de er altså *ikke* datasett options. De er derimot options til PROC PRINT-setningen (NOOBS innebærer at observasjonsnummeret ikke kommer med på utskriften, mens LABEL innebærer at vi vil ha variablenes forklarende tekst (lablene) som kolonneoverskrifter istedenfor variabelnavnene).

Noen få datasett options brukes bare i spesielle situasjoner, for eksempel når datasett skal kobles eller når vi har å gjøre med svært store eller ømtålige datasett:

- Datasett options til kobling, krymping og kryptering:

```
IN=Boolsk variabel
COMPRESS=yes|no
ENCRYPT=yes|no
```

3.1.3 System options

System options kan brukes for å angi hvor mange linjer vi vil ha på hver side i utskriften, hvor mange tegn det skal være pr linje, om tabeller skal være sentrert osv. System options kan vi angi på alternative måter:

- i OPTIONS-setningen
- i options-vinduet

OPTIONS-setningen er en global setning, altså en setning som står utenfor DATA- og PROC-steget. Vi skriver den ofte tidlig i programmet, etter de innledende kommentar-setningene. Hvis det er noen system options vi *alltid* vil ha med, skriver vi gjerne OPTIONS-setningen i en egen oppstart-fil for SAS (autoexec.sas). Med en OPTIONS-setning som dette, vil vi få 66 linjer pr side, tabellene blir ikke sentrert og vi unngår dato på tabellene: `OPTIONS PS=66 NOCENTER NODATE ;`
System options kan alternativt settes i options-vinduet. Dette får vi opp ved å skrive ordet options i et kommando-felt (eller velge Globals, Options, Global options fra SAS menyen).

3.2 SAS funksjoner

Det finnes en lang rekke ferdige funksjoner vi kan benytte oss av, for å spare programmering. De kan brukes til for eksempel å avrunde, summere, finne den største, den minste, gjennomsnittet, standard avviket, venstrejustere, trekke deler av en tekststreng, etc etc. Vi sier at funksjonene *tar argumenter* og *returnerer en verdi.* Argumentene til funksjonen står i parentes. I eksemplene under lager vi nye variable ved å bruke tilordnings-setning. Det som står til venstre for likhetstegnet er navnet på den nye variabelen, mens det som står til høyre er verdien. I eksemplene gis de nye variablene verdi ved hjelp av funksjoner:

```
f_aar = SUBSTR(f_nr,5,2);
```

SUBSTR-funksjonen brukes til å trekke *en delstreng* ut av en karaktervariabel. I eksempelet trekkes fødselsåret ut av fødselsnummeret. Fødselsåret er en del av fødselsnummeret, det starter i posisjon 5 i fødselsnummeret og er 2 tegn langt.

```
tot_lonn = SUM(lonn,tillegg,bonus);
```

SUM-funksjonen brukes til å summere verdier. Bemerk at SUM-funksjonen behandler manglende verdier som 0 (i motsetning til om vi hadde summert ved å bruke tegnet +).

```
mnd_lonn = ROUND(aarslonn/12);
```

ROUND-funksjonen avrunder et desimaltall. Legg merke til at det er fullt lovlig å dividere (evt multiplisere) elementet som er argument til funksjonen.

```
skatt = MAX(beregnet, 0 ) ;
```

MAX-funksjonen returnerer den største verdien blant argumentene. I eksempelet er argument nr 2 konstanten 0. Dersom variabelen `beregnet` har negativ verdi, vil MAX-funksjonen returnere verdien 0, mens hvis `beregnet` er større enn 0, er det verdien til `beregnet` som blir returnert.

NB! Det er fullt mulig å bruke flere funksjoner utenpå hverandre! Vil vi både summere og avrunde, kan dette gjøres slik: `mndlonn = ROUND( SUM(fast,tillegg,aarbonus/12) );`

Funksjonene er delt inn i forskjellige kategorier:

- karakter-
- statistikk-
- dato og tids-
- aritmetiske
- trunkerings-
- spesial
- tilfeldige talls-
- sannsynlighets-
- kvantile
- matematiske
- trigonometriske og hyperboliske
- finansiell
- logiske binærtalls-

Det finnes i tillegg andre spesielle funksjoner. Mange av funksjonene er nærmere beskrevet i **Håndbok i SAS Del 2: Oppslag.**

3.3 Format

Formater er et omfattende begrep i SAS. Vi skiller mellom SAS-definerte format og egendefinerte format. SAS-definerte format brukes for å gi variable det format vi ønsker, både internt på datasettet og ved utskrift. Egendefinerte format kan også brukes til å formatere utskrift, men i tillegg kan de benyttes blant annet til *gruppering*. SAS-definerte format og egendefinerte format er svært viktige begrep, de inngår i enhver sammenheng der vi benytter SAS.

For å gi variable på datasettet format bruker vi SAS innlesningsformater, mens for å formatere variable som skrives ut brukes SAS utskriftsformater. SAS innlesningsformat brukes i INPUT-setningen der vi knytter innlesningsformater til variable. SAS utskriftsformater knyttes til variable i FORMAT-setningen. Egendefinerte format *defineres* i PROC FORMAT og *brukes* med en FORMAT-setning i en annen proc.

- SAS innlesningsformater
- SAS utskriftsformater
- egendefinerte formater

Når vi vil knytte en variabel til et format, skriver vi navnet på variabelen etterfulgt av formatet. Når vi *bruker* et format, er det alltid et punktum involvert i format-navnet (som regel aller bakerst)! Dette gjelder både SAS format og egendefinerte format. (Legg merke til at når egendefinerte format *defineres*, skal det ikke være noe punktum i format-navnet.)

Når format *brukes*: Alltid et punktum involvert!

3.3.1 SAS innlesningsformat

Innlesningsformatene er lette å forholde seg til, de brukes i INPUT-setningen når vi skal lage SAS datasett av en rådatafil. I INPUT-setningen angir vi hvilke posisjoner på recorden variablene skal leses inn fra, hva variablene på datasettet skal hete og hvilke format de skal ha. Vi vil sterkt oppfordre til *alltid* å bruke innlesningsformat i INPUT-setningen, selv om det i noen tilfelle vil gå bra uten. I mange tilfelle *må* vi lese inn formatert og da er det greit å være vant til å bruke slike formater.

Bruk *alltid* innlesningsformater i INPUT-setningen!

I innlesningsformatet angir vi om variabelen som skal leses inn skal være i numerisk format, i karakter-format eller i dato-format. Innlesningsformatet brukes også til å angi bredden på feltet (antall posisjoner) som skal leses inn, samt for numeriske variable eventuelt antall desimaler. I eksemplene (og manualene) betegnes feltbredde med bokstaven `w` (for *width*). I praksis erstattes `w` av et tall.
Det finnes svært mange forskjellige innlesningsformater, men det er bare et fåtall av dem vi bruker til daglig. Her kommer de aller vanligste:

Karakterformat:

`$CHARw.`

`$CHAR.`-formatet brukes for å lese inn karakter-variable (altså alle variable som vi ikke skal utføre beregninger på). `w` angir hvor mange posisjoner som skal leses inn, `w` er et tall mellom 1 og 32767. Alle format som starter med tegnet `$` er karakter-format. `$CHAR.`-formatet beholder verdien som leses inn *eksakt* slik den står på recorden hvilket blant annet innebærer at ledende blanke beholdes (verdien blir med andre ord ikke venstrejustert). Dette i motsetning til karakterformatet `$w.` som fjerner ledende blanke.

Numeriske format:

`w.`

`w.`-formatet brukes for å lese inn numeriske variable. Også her angir `w` hvor mange posisjoner som skal leses inn. `w.`-formatet er egentlig en spesialvariant av `w.d`-formatet (se nedenfor) der `d` er utelatt.

`w.d`

`w.d`-formatet brukes for å lese inn numeriske variable i de tilfelle vi ønsker at de siste posisjonene skal leses inn som desimaler. `d` brukes da til å angi antall desimaler. Legg merke til at `w` angir *totalt* antall posisjoner som skal leses inn, mens `d` angir hvor mange *av disse* som skal være desimaler. `w.d`-formatet kan for eksempel brukes hvis dataene vi skal lese inn er oppgitt i øre, mens vi vil operere med kroner som enhet.

NB! Om vi leser inn en numerisk variabel uten desimaler, betyr *ikke* dette at tallet lagres 'unøyaktig', uten desimaler internt på datasettet! Tvert imot lagres alle numeriske variable med meget stor presisjon (*floating-point*).

Dato format:

`YYMMDDw.` `YYMMDDw.`-formatet er ett av SAS mange dato-format. `YYMMDDw.`-formatet brukes for å lese inn datoer som er på form år-måned-dag. `w` angir antall posisjoner som skal leses inn. Legg merke til at `w` ikke er begrenset til å være 6 posisjoner! Hvis det er brukt skilletegn mellom år, måned og dag, kan vi bruke formatet `YYMMDD8.` uansett hvilket tegn som er brukt som skille. Dersom året er angitt med 4 siffer (1999) kan vi lese det inn med `YYMMDDN8.` .

`DDMMYYw.` `DDMMYYw.`-formatet er et annet av SAS dato-formater. Det brukes for å lese inn datoer der rekkefølgen er dag-måned-år. Ellers er reglene med hensyn på bredden `w` og skilletegn akkurat som for formatet `YYMMDDw.` og de andre dato-formatene.

Ikke glem at dette bare var en liten smakebit på alle de innlesningsformat som finnes! Det er en av SAS' sterke sider at det tilbyr svært mange innlesningsformater. Det finnes knapt noe dataformat SAS ikke kan lese.

Vi anbefaler sterkt å *alltid* bruke innlesningsformater i INPUT-setningen, det gir oss full kontroll over dataene. I visse situasjoner er dessuten innlesningsformat helt påkrevet: Hvis dataene vi skal lese inn er lagret på spesielle måter, kommer vi ikke utenom å bruke innlesningsformater. Dataene kan for eksempel være *pakket* eller de kan ha fortegnet lagret sammen med siste siffer (også kalt *IBM overpunch*, et format som har vært mye benyttet i Byrået). I slike tilfelle *må* vi bruke format ved innlesningen. (Pakkete felt kan leses med formatet `PDw.` mens *overpunch*-felt kan leses med formatet `ZDw.`)

3.3.2 SAS utskriftsformat

SAS utskriftsformat brukes til å formatere variables verdier ved *utskrift*. Utskrift i denne sammenheng betyr enten at variabelen skrives ut til en sekvensiell fil (også kalt 'flat' fil eller rådata-fil) eller at variabelen skrives ut i en tabell. Formålet med å formatere variable som skrives ut til fil, er å ha full kontroll over recordbeskrivelsen til filen vi skal lage, mens formålet med å formatere tall i tabeller er å gjøre tabellene lettere å lese. Utskriftsformatene brukes på litt forskjellig måte i de to anvendelsene. Utskriftsformat brukes når vi vil

- lage *sekvensiell fil* av SAS datasett, formatene skrives i en PUT-setning i et DATA-steg
- formatere tall i *tabell*, formatene skrives i en FORMAT-setning i et PROC-steg

3.3.2.1 Formatere utskrift til sekvensiell fil

Hvis vi vil lage sekvensielle filer av SAS datasett, gjør vi dette i et DATA-steg med blant annet **FILE**- og **PUT**-setningene. PUT-setningen virker på helt tilsvarende måte som INPUT-setningen,

men mens INPUT-setningen brukes til å *lese inn* rådata til SAS datasett, brukes PUT-setningen til å *skrive ut* SAS datasett til rådata. På tilsvarende måte som vi bruker innlesningsformater i INPUT-setningen, kan vi bruke utskriftsformater i PUT-setningen for å få recorden på filen vi lager til å bli nøyaktig slik vi vil. (Se eksempel i avsnitt 5.3)
De aller fleste SAS formatene kan brukes både til innlesning og til utskrift, men ikke absolutt alle. Derfor er det i manualene skilt mellom innlesnings- og utskriftsformater.

3.3.2.2 Formatere utskrift til tabell

Mens sekvensielle filer lages i DATA-steg, lages tabeller i PROC-steg. For å formatere variable som skal skrives ut i en proc, kan vi bruke **FORMAT**-setningen der vi knytter variablene til utskriftsformater. Med utskriftsformater styrer vi 'utseendet' til tallene. Derfor blir en tabell lettere å lese når tallene i den er formatert. (Det er ikke helt riktig å si *tallene*, for det som formateres er variablenes *verdier*. Som oftest er dette tall, men det kan også være bokstaver, tekst.)

Et eksempel: Vi vil skrive ut observasjoner fra et datasett for å sjekke om en beregnet variabel ser riktig ut i forhold til andre variable. Vi skriver ut datasettet med PROC PRINT. Vi vil ha med så mange variable som mulig på utskriften, men uten at én observasjon går over flere linjer. Ved å formatere variablene, får vi 'presset' dem sammen slik at vi får plass til flere på én linje. En ny variabel som er fremkommet ved beregning, kan ofte inneholde desimaler. På kontroll-utskriften er det ikke så viktig for oss å se desimalene, tvert imot, dette dreier seg om store tall. For at kolonnene ikke skal 'forstyrres' av desimaler, bruker vi derfor utskriftsformat slik:

```
PROC PRINT DATA=hotell.april96 (KEEP=omsetn nyomsetn ant_rom ant_seng gjester
                                     ank_nord ank_utl hotellgr OBS=100) ;
   FORMAT nyomsetn 8. ;
   TITLE 'Sjekk ny (beregnet) omsetning mot oppgitt. OK? ';

RUN;
```

I FORMAT-setningen oppgir vi først navnet på den eller de variable som skal formateres, deretter utskriftsformatet. I eksempelet skal den nye variabelen `nyomsetn` skrives ut med formatet `8.` som betyr at det totalt avsettes 8 plasser til tallet og verdien skrives ut uten desimaler.
(Enkelte proc'er har i tillegg til FORMAT-setningen sine egne måter å angi utskriftsformat på, for eksempel PROC TABULATE.)

3.3.3 Egendefinerte format: PROC FORMAT pluss FORMAT-setning

Vi kan definere våre egne format, disse virker på en litt annen måte enn SAS formatene. Egendefinerte format brukes til to forskjellige formål:

- erstatte variabelverdier med forståelig tekst i tabeller (for eksempel at verdien '1' erstattes med teksten 'Menn' og verdien '2' med teksten 'Kvinner')
- gruppere variabelverdier

Egendefinerte format *defineres* med PROC FORMAT, og *brukes* med FORMAT-setningen i PROC- eller DATA-steg. Hver statistikk vi arbeider med har gjerne sine egne formater. Det er lurt å samle alle formatdefinisjonene i et eget SAS format-program og la dette programmet hete **format.sas** Da har vi alle definisjonene samlet på ett sted og vi slipper å lure på hvordan de enkelte formatene er definert. (Noen formater er standard for hele SSB. Dette gjelder blant annet format som erstatter fylkeskoder med fylkesnavn og kommunnummer med kommunenavn. Disse felles-formatene ligger ferdig definert på unix, klare til bruk, tilgjengelig for alle. Det er bare å bruke dem, direkte i

FORMAT-setningen. Hvilke format som er definert og hva de heter, er oppgitt i vedlegg til dette heftet.)

Ikke lag egendefinerte format for kommune, fylke etc. De finnes allerede tilgjengelig!

3.3.3.1 Erstatte verdier med forståelig tekst i en tabell

Det er ikke innlysende at verdien '1' for en variabel som angir kjønn betyr 'mann' og at verdien '2' betyr 'kvinne'! Derfor må vi sørge for at det ikke står slike verdier i tabellene, men forståelig tekst. Dette gjør vi ved hjelp av egendefinerte formater. Først *definerer* vi formatene i PROC FORMAT (og lagrer gjerne formatene permanent), deretter *bruker* vi formatene med FORMAT-setningen i f.eks PROC TABULATE. (Legg merke til at det *ikke* skal stå punktum etter formatene når de defineres, mens det *må* stå punktum etter når de senere brukes i FORMAT-setningen.) Slik definerer vi to permanente formater `$kjonn` og `$svar` :

```
***------------------------------------------------------------------**;
*  HELSEUNDERSØKELSEN                                                  ;
*                                                                      ;
*  Programfil: $HELSUND/prog/format.sas                                ;
*  Skrevet:    7. august 1996, LDa                                     ;
*                                                                      ;
*  Definerer permanente format til bruk i Helseundersøkelsestabellene;
*                                                                      ;
***------------------------------------------------------------------**;

LIBNAME library '../kat' ;

PROC FORMAT LIBRARY = library;
   VALUE  $kjonn
         '1' = 'Menn'
         '2' = 'Kvinner'
       other = 'Uoppgitt'
          ;
   VALUE  $svar
         '1' = 'Ja'
         '2' = 'Nei'
         '3' = 'Vet ikke'
         '9' = 'Uoppgitt'
          ;
RUN;
```

Når PROC FORMAT er kjørt, er formatene *kun definert*. Det er først når vi knytter formatene sammen med variable i FORMAT-setningen i en annen proc, f.eks PROC FREQ vi *bruker* dem:

```
***------------------------------------------------------------------**;
*  HELSEUNDERSØKELSEN                                                  ;
*                                                                      ;
*  Programfil: $HELSUND/prog/tab01.sas                                 ;
*  Skrevet:    7. august 1996, LDa                                     ;
*                                                                      ;
*  Frekvenstabeller, noen få variable (kjønn, spm 25 27 30 og 34)      ;
***------------------------------------------------------------------**;

PROC FREQ DATA = omnibus.kv2_96 ;
   TABLES  kjonn v25 v27 v30 v34 ;
   FORMAT  kjonn $kjonn.
           v25
           v27
           v30
           v34   $svar.
           ;
RUN;
```

Legg merke til følgende:

- når vi *bruker* formatene, setter vi *et punktum* rett etter formatnavnet (dette gjør vi altså *bare* når vi bruker formatene, *ikke* når vi *definerer* dem)

- punktumet må stå *rett etter* siste tegn i formatnavnet, det må ikke være blank mellom
- samme format kan brukes på *flere* variable (I eksempelet har variablene `v25`, `v27`, `v30`, og `v34` samme verdier: '1' for 'Ja', '2' for 'Nei' osv. Både `v25`, `v27`, `v30`, og `v34` skal derfor benytte det samme formatet, nemlig `$svar.`)

3.3.3.2 Gruppere variabelverdier

Vi opererer ofte med alders*grupper*, inntekts*grupper*, nærings*grupper og* andre grupper. I slike tilfelle kan det (tilsynelatende) være fristende å sette igang med store IF-THEN-ELSE-konstruksjoner for å lage nye, grupperte variable. **Dette frarådes på det mest innstendige!** Det er mye, mye bedre å gruppere ved hjelp av egendefinerte formater. Ved å bruke formater beholder vi dataene i sin opprinnelige form, vi står friere til å operere med flere alternative grupperinger på samme variabel, grupperingene er mer fleksible. Det er tusen gode grunner for å gruppere ved hjelp av formater.

```
PROC FORMAT LIBRARY = library;
   VALUE  inntgrp
         low     -< 100000 = 'Mindre enn 100 000 kr'
         100000 -< 200000 = ' 100 000 - 199 999 kr'
         200000 -< 250000 = ' 200 000 - 249 999 kr'
         250000 -< 300000 = ' 250 000 - 299 999 kr'
         300000 -  high   = ' 300 000 kr og mer   '
         ;
   VALUE  $naring
         '61000' - '61999' = 'Detaljhandel'
         '62000' - '62999' = 'Engroshandel'
                   other   = 'Andre næringer'
          ;
   VALUE  $sektor
         '231000',
         '231110',
         '231200',
         '234010' - '234040' = 'Energisektorene'
                     other   = 'Alle andre sektorer'
          ;
```

Legg også merke til følgende:

- format som skal brukes på *karakter*variable (som f.eks kjønn og næring) *må* ha navn som starter med `$`
- navn på format som skal brukes på *numeriske* variable (som f.eks inntekt) *må* starte med en bokstav
- formatnavn kan *ikke* slutte med et tall
- `low` og `high` i definisjonen av numeriske format brukes til å lage grupper med uspesifiserte grenser oppad og nedad (manglende verdier vil *ikke* komme innunder kategorien `low`)
- `-< 100000` betyr at `100000` *ikke* skal være med i intervallet
- enkeltverdier kan ramses opp, adskilt med komma
- en gruppe kan inneholde både enkeltverdier og intervaller

4. SAS i praksis

4.1 SAS interaktivt eller SAS i batch

SAS kan kjøres *interaktivt* eller *i batch*. Hvis vi dobbeltklikker på SAS-ikonet i Windows eller skriver kommandoen **sasx** på unix, får vi opp SAS *interaktivt*. Vi får da opp flere vinduer, blant andre program-(editor-)vinduet, SASlog-vinduet og output-vinduet (sistnevnte kalles også SASlist-vinduet eller resultat-vinduet). I program-vinduet skriver vi SAS programmet. Når programmet er klart, kjører vi det (vi *submitter* det). Feilmeldinger og annen informasjon om kjøringen kommer i SASlog-vinduet, mens tabeller og andre resultat kommer i output-vinduet.

En annen måte å kjøre SAS på, er i batch. Det forutsetter at vi har et ferdigskrevet SAS program som vi bare skal kjøre, uten noe om og men. Kjøres SAS i batch får vi ikke opp vinduene, resultatet av kjøringen (SASlisten) og kjørerapporten (SASlogen) legges da automatisk på egne filer med spesielle filhaler. SASlog-filen får filhale **.log** mens SASlist-filen får filhale **.lst** Når vi er i utprøvingsfasen kjører vi som regel SAS interaktivt, mens når utprøvingen er over og program skal kjøres rutinemessig, kjøres de i batch. For å kjøre et program i batch på unix, skriver vi kommandoen **sas** etterfulgt av navnet på SAS programmet.

- **SAS interaktivt**:
  - på **unix** (med **XVision**), skriv: **sasx**
  - i **Windows**: **dobbeltklikk på SAS-ikonet**
- **SAS i batch**:
  - på **unix**, skriv: **sas SAS-programfilnavn**

Har du ikke noe SAS-ikon under mappen Mine mest brukte i Windows, kontakt Kundestøtte for å få mulighet til å installere SAS og en oppskrift på hvordan du gjør det.

4.2 SAS vinduer, kommandoer og funksjonstaster

Når vi tar opp SAS interaktivt møter vi SAS' vindussystem. Det består av en rekke vinduer:

- program-vinduet
- SASlog-vinduet
- output- (SASlist-)vinduet
- keys-vinduet
- options-vinduet
- lib-vinduet
- explorer-vinduet
- osv osv...

For å komme fra ett vindu til et annet, brukes *kommandoer*. Disse kan aktiveres fra rullgardinmenyene eller skrives i et kommandofelt. For å komme til options-vinduet bruker vi kommandoen *options*, for å komme til keys-vinduet bruker vi kommandoen *keys* osv osv. Men kommandoer brukes til mye mer enn navigering, for eksempel brukes kommandoen *sub* til å utføre et program, *clear* brukes til å blanke innholdet i et vindu, *recall* brukes til å hente tilbake programmet du nettopp kjørte, etc. Legg merke til forskjellen mellom *kommandoene* og *setningene*! Det er setningene som utgjør SAS programmet, men kommandoen *sub* som får programmet til å bli utført.

Kommandoer vi ofte bruker, legger vi gjerne på funksjonstaster (functionkeys). I keys-vinduet ser vi hva som ligger på funksjonstastene. Vil vi endre en funksjonstast, er det bare å skrive over. Men det er en fordel om funksjonstastene følger et visst felles oppsett. Dette er et standardoppsettet for SSB:

- **F1 clear** blanker vinduet
- **F2 keys** tar opp keys-vinduet
- **F3 log** tar opp SASlog-vinduet
- **F4 output** tar opp output- (SASlist-)vinduet
- **F5 end** lukker et vindu
- **F6 pgm** går til program-vinduet
- **F7 zoom** blåser opp vinduet (av/på annenhver gang)
- **F8 rfind** gjentar søk i teksten
- **F9 recall** henter tilbake til programvinduet program som er utført
- **F10 submit** utfører et program
- **F11 pmenu** bytter mellom kommandolinje og rullgardinmenyer for alle vinduer
- **F12 rsubmit** sender SAS-setninger til eksekvering på tjenermaskin

4.3 Kommandolinje eller rullegardin-menyer

Når vi kjører SAS interaktivt, kan vi skrive kommandoene på en kommandolinje eller velge kommando fra rullegardinmenyer. Kommandolinjen ser omtrent slik ut: ===> og befinner seg i sitt eget vindu eller øverst til venstre i de andre vinduene. Kommandolinjen krever at du kjenner kommandoene (men de er ikke så mange), til gjengjeld slipper du å bruke mus. (I Windowsversjonen av SAS står det at det er mer ressurskrevende for maskinen å bruke menyer.) Har du fått en kommandolinje og vil bli kvitt den igjen, skriver du kommandoen **com** og trykker enter. Har du meny og vil ha kommandolinje, velger du **Tools**, **Options**, **Preferences** og arkfanen **View** for der å fjerne valget **Command Line.**

4.4 SASlog

Når et SAS program kjøres, lages det automatisk en kjørerapport, en SASlog. SASlogen inneholder svært nyttige opplysninger som kan brukes som dokumentasjon. Hvis kjøringen har gått galt, er det i SASlogen vi finner feilmeldingene. SASlogen dukker naturlig nok opp i SASlog-vinduet. Det lønner seg alltid å studere SASlogen grundig! Vi kan lagre SASlogen på fil, den får da automatisk filnavn med endelse **.log**. (Hvis vi kjører SAS i batch, blir SASlogen automatisk lagret på en fil med samme navn som programmet, men med filhale .log)

4.5 SASlist

Hvis vi kjører en PROC som skal gi resultat i form av tabeller, lister eller andre 'utskrifter' (for eksempel proc'ene PRINT, FREQ, MEANS, TABULATE, ANOVA, REG, GLM etc etc) kommer tabellen i resultatvinduet, output-vinduet. Resultatet kalles også SASlisten. Lagrer vi SASlisten, får den automatisk filhale **.lst**. Hvis vi kjører SAS i batch, blir SASlisten automatisk lagt på en fil med samme navn som programmet, men med **.lst** som filhale.

4.6 autoexec.sas

Dersom det er noen faste setninger vi ønsker å utføre hver eneste gang vi starter SAS, kan det lønne seg å legge disse setningene i en oppstartfil, et eget lite oppstart SAS program. Dersom det ligger en fil som heter **autoexec.sas** på den katalogen som vi starter SAS fra, vil denne filen (det vil si dette SAS programmet) bli utført idét SAS starter. Vi trenger altså ikke hente inn programmet og kjøre det, det blir kjørt automatisk hvis programfilen heter autoexec.sas og ligger på den katalogen vi starter SAS fra.

Advarsel! Vær forsiktig med å lage en fil som heter autoexec.sas før du er *helt* fortrolig med hvordan en slik fil virker og hva som skjer! Hvis du i begynnelsen kaller oppstartfilen start.sas kan du heller endre navn på den til autoexec.sas når tiden er moden. (Heter oppstartfilen start.sas må den hentes inn og kjøres 'manuelt' med en gang vi har tatt opp SAS.)

Setninger det er naturlig å ha på en slik autoexec.sas-fil, er OPTIONS-setningen der du velger dine egen preferanser med hensyn på sidenummerering, sentrering osv, og LIBNAME-setninger som knytter forbindelse mellom SAS og kataloger der du har permanente datasett liggende (eller der du ønsker å lagre SAS datasett) eventuelt katalogen der du har permanente, egendefinerte format.
En autoexec.sas-fil kan se slik ut:

```
***------------------------------------------------------------------------***;
*  TILBAKEFALLSPROSJEKTET   1987                                             ;
*                                                                            ;
*  Programfil: $KRIMINAL/tilbfall/prog/autoexec.sas                          ;
*  Skrevet:    22 juni 1995,  KrL                                            ;
*                                                                            ;
*  Oppstartfil for SAS.                                                      ;
*                                                                            ;
*  Vil ikke ha dato eller sidenummer, 66 linjer pr side.                     ;
*  Knytter forbindelse til katalogen der SAS datasettene er lagret.          ;
*  Knytter også forbindelse til katalogen med egendefinerte format.          ;
***------------------------------------------------------------------------***;

OPTIONS NODATE NONUMBER PS=66 ;
LIBNAME tilbfall '$KRIMINAL/tilbfall/wk12' ;
LIBNAME library  '$KRIMINAL/tilbfall/kat' ;

RUN;
```

4.7 Hvordan SAS utfører et program

Dersom et SAS program består av flere steg, utfører SAS ett og ett steg ad gangen. Alle setningene i ett steg utføres før SAS begynner å utføre setningene i neste steg. DATA-, PROC- og RUN-setninger avgrenser steg. Et DATA-steg begynner med en DATA-setning og varer helt til det kommer en ny DATA-setning, en PROC-setning eller en RUN-setning. Tilsvarende starter et PROC-steg med en PROC-setning og varer helt til det kommer en ny PROC-setning, en DATA-setning eller en RUN-setning.

Et SAS program utføres steg for steg

Når et DATA-steg utføres, henter SAS inn én observasjon, utfører alle setningene på den ene observasjonen og legger så observasjonen ut på datasettet som lages. Så hentes neste observasjon inn, alle setningene utføres for den observasjonen, den legges ut på det nye datasettet, neste observasjon hentes inn, osv osv.

Et DATA-steg utføres observasjon for observasjon

Det er viktig å merke seg at i det øyeblikk en observasjon er lagt ut på det nye datasettet og før neste observasjon lese inn, 'blankes' alle verdier (mer presist: variablene gis manglende verdi, de settes til *missing*). Dersom vi i spesielle tilfelle ønsker at en variabel ikke skal blankes men beholde verdien sin fra observasjon til observasjon, kan vi bruke RETAIN-setningen.

4.8 Spørsmål som må besvares før du setter igang....

Når du skal sette igang med en databehandlingsoppgave, er det viktig å vite så mye som mulig om datamaterialet på forhånd. Hva er filnavnet? Hvor er filen lagret? Er det en flat, sekvensiell fil? Et SAS datasett? En Oracle-tabell? Hvor mange recorder er det? Hvor mange felt? Trenger jeg alle feltene? Skal jeg bruke alle recordene? Er det spesielle ting jeg bør vite om dataene? Blant annet er det viktig å merke seg hvordan manglende verdier (*missing*) er representert i dataene. (Hvis for eksempel manglende fødselsår er satt til verdien 90, kan det oppstå kjedelige situasjoner dersom en ikke er klar over dette!) Når det gjelder å skaffe seg informasjon om et datamateriale, er filbeskrivelsen et meget viktig dokument. Alle filer som benyttes i vår statistikkproduksjon *skal* være dokumentert. Før vi starter, må følgende spørsmål besvares:

- Hva er filnavnet?
- Hvor er filen lagret?
- Er det en flat fil, et SAS datasett, en Oracle-tabell eller hva?
- Trenger jeg *hele* filen eller kan jeg greie meg med enkelte variable/observasjoner?
- Er det spesielle ting å ta hensyn til? (Sjekk spesielt manglende verdier!)

Når disse spørsmål er besvart, kan vi gå videre til det som har med SAS å gjøre. Dersom datamaterialet allerede foreligger som SAS datasett eller view, er saken grei. Da kan vi gå rett på analyser, tabeller eller videre bearbeiding. Hvis dataene derimot *ikke* foreligger slik, må vi først lage SAS datasett(*) av dem.

Er data allerede SAS datasett eller view?

Ja: Gå rett på tabeller, analyse eller videre databearbeiding: **PROC**- eller **DATA-steg**

Nei: Lag SAS datasett først: **DATA-steg**

(*) Det er ikke en absolutt betingelse at vi lager SAS datasett av datamaterialet for å kunne bruke SAS til å lage tabeller, analyser osv. Vi kan greie oss med å lage *view* av dataene. Det tar da litt lenger tid for SAS å aksessere, hente dataene, men vi sparer til gjengjeld lagringsplass.

5. DATA-steget

Det er hovedsakelig i to forskjellige situasjoner vi *må* bruke DATA-steget: Det er når vi skal lage SAS datasett av rådata, og det er når vi skal bearbeide, omstrukturere eksisterende SAS datasett. I begge tilfelle bruker vi DATA-steg, men programmene vil likevel se nokså forskjellige ut.

5.1 Lage SAS datasett av rådata

Når vi skal lage SAS datasett av rådata, må vi som første punkt gi SAS informasjon om SAS datasettet vi skal lage: Hva skal det hete? Hvor skal det lagres? Deretter må vi gi opplysninger om rådataene: Hvilken fil skal SAS lese rådata fra? Vi må også angi nøyaktig hvilke felt på rådatafilen vi vil skal være med på SAS datasettet.

På neste side følger eksempel på et program som lager SAS datasett av rådata. (Det er for å få eksempel og forklaring på motstående sider vi hopper over resten av denne siden.)

Eksempel på program som lager SAS datasett av rådata:

```
/*************************************************************************
Prosjekt ..............: TILBAKEFALLSPROSJEKTET   1987
Programnavn ...........: $KRIMINAL/tilbfall/prog/les87.sas
Skrevet av ............: Kristian Lønø, 303
Dato ..................: 9.3.1994
Versjon ...............: 1.0
Programmets funksjon ..: Lager SAS datasett av sekvensiell fil med reaksjonsdata
                         (reaksjon = dom)
Anmerkning.............:
Programmet kaller .....:
Filer inn..............:
Filer ut...............:
Endret når ............:
Endret av .............:
Grunn til endring .....:
*************************************************************************/

DATA tilbfall.reak87;
   INFILE '$KRIMINAL/reaksjon/aargang/dommer.g87' TRUNCOVER LRECL=217;
   INPUT  @2   fnr        $CHAR11.
          @2   fdato      DDMMYY6.
          @17  domsdato   YYMMDD6.
          @23  bostedp     $CHAR4.
          @29  sivst       $CHAR1.
          @30  statsbp     $CHAR3.
          @40  gjtid      YYMMDD6.
          @66  bot              7.
          @73  erstatn          7.
          @89  provetid         4.
          @99  sikraar          2.
          @144 dagubet          5.
          @149 dagbet           5.
          @157 bosted      $CHAR4.
          @211 lovbkode    $CHAR3.
            ;
   LABEL fnr      = 'Fødselsnr'
         fdato    = 'Fødselsdato'
         domsdato = 'Domsdato'
         bostedp  = 'Bosted (fra politiet)'
         sivst    = 'Sivilstand'
         statsbp  = 'Statsborgerskap (fra politiet)'
         gjtid    = 'Gjerningstidspkt'
         bot      = 'Bot'
         erstatn  = 'Erstatning'
         provetid = 'Prøvetid'
         sikraar  = 'Sikring, antall år'
         dagubet  = 'Antall dager ubetinget fengsel'
         dagbet   = 'Antall dager betinget fengsel'
         bosted   = 'Bosted (fra Personreg)'
         lovbkode = 'Lovbruddskode'
         alder    = 'Alder på gjerningstidspkt, år'
         dagtdom  = 'Antall dager fra gjerning til dom'
         soning   = 'Beregnet soningstid, dager'
         ;

   * Skal beregne alder (år) på gjerningstidspunktet.                    ;
   * Siden gjerningstidspkt og fødselsdato er lest inn med datoformat,   ;
   * kan variablene trekkes fra hverandre.                               ;
   * Differansen blir antall dager mellom de to datoene.                 ;
   * Finner først alder i antall dager. 1 år er 365,25 dager.            ;
   * Finner dermed alder i år ved å dividere med 365,25.                 ;

   alder   = ROUND( (gjtid - fdato) / 365.25 ) ;

   * Finner antall dager mellom ugjerning og dom.                        ;
   dagtdom = domsdato - gjtid ;

   * Hvis straffen er ubet. fengsel 180 dager eller mer, beregnes        ;
   * soningstiden til 2/3 av idømt straff.                               ;
   * Ellers sones hele straffen.                                         ;

   IF    dagubet >= 180 THEN soning = ROUND(dagubet * 2 / 3) ;
   ELSE                      soning = dagubet ;

RUN;
```

Forklaring til eksempelet:

Først et lite forbehold: Eksempelet over er et komplett program med mange kommentarer og lange INPUT- og LABEL-setninger. For å gjøre forklaringen mer oversiktlig, har vi derfor kortet litt ned på programmet. Det betyr at vi ikke omtaler innlesning av *alle* variablene, men bare forklarer de som tilfører noe nytt. Likeledes forklarer vi ikke *alle* lablene, kommentarene etc.
Øverst i programmet er en kommentarblokk som viser hvilket prosjekt programmet tilhører, hvor programfilen er lagret, når og hvem som har skrevet programmet og litt om hva som utføres. Kommentarene skilles fra selve programmet med en blank linje. Kommentarer skrives med **kommentar**-setningen, ***-setningen. Kommentar-setningen starter med en stjerne (*) og avsluttes med et semikolon (;). Alt som står mellom stjerne og semikolon ignoreres av SAS. I kommentaren kan vi skrive allslags vanlig tekst, men husk at semikolon avslutter kommentaren! (Med andre ord: Ha ikke semikolon som en del av teksten i kommentaren.)

```
/*************************************************************************
Prosjekt ..............: TILBAKEFALLSPROSJEKTET   1987
Programnavn ...........: $KRIMINAL/tilbfall/prog/les87.sas
Skrevet av ............: Kristian Lønø, 303
Dato ..................: 9.3.1994
Versjon ...............: 1.0
Programmets funksjon ..: Lager SAS datasett av sekvensiell fil med reaksjonsdata
                         (reaksjon = dom)
Anmerkning.............:
Programmet kaller .....:
Filer inn..............:
Filer ut...............:
Endret når ............:
Endret av .............:
Grunn til endring .....:
*************************************************************************/
```

Kommentarer kan også skrives mellom disse tegnene: /* og */. Alt mellom disse tegnene blir regnet som en kommentar.

```
/* Beregne alder (år) på gjerningstidspunktet.  */
```

Siden vi skal lage et SAS datasett av rådata, må programmet være et DATA-steg. DATA-steget starter alltid med en **DATA**-setning. I DATA-setningen oppgir vi hva SAS datasettet vi vil lage skal hete og om det skal være permanent eller temporært (ett eller to ledd i navnet). I dette tilfelle skal datasettet lagres permanent, det ser vi av at det er to ledd i navnet. Det første leddet sier noe om hvilken katalog datasettet skal lagres på, det andre leddet sier hva datasettet skal hete. Datasettet i eksempelet skal hete `reak87` Dette navnet holder seg innenfor 8 tegn og består av kun bokstaver og tall. Det første leddet, `tilbfall` innebærer at datasettet skal lagres permanent. Det skal lagres på en katalog i filsystemet som på forhånd er gitt kortnavnet (*libref*'en) `tilbfall` Det er i LIBNAME-setningen vi oppretter libref'er, det vil si knytter forbindelse mellom SAS og den katalogen der vi vil at SAS datasett skal lagres. LIBNAME-setninger ligger gjerne i SAS' oppstartfil, autoexec.sas, slik at forbindelsen til lagringskatalogen(e) opprettes idét SAS starter. Derfor er ikke LIBNAME-setningen med i programeksempelet.) I eksempelet over er libref'en `tilbfall` . I virkeligheten betyr `tilbfall` unix-katalogen `/ssb/ursus/a1/kriminal/tilbfall/work/` Når programmet er kjørt, vil det hele og fulle filnavnet til SAS datasettet være `/ssb/ursus/a1/kriminal/tilbfall/wk12/reak87.sas7bcat` Alle SAS datasett får automatisk filhalen `.sas7bcat` Dette er det SAS selv som håndterer. I programmet skriver vi kun `reak87` som navn.

```
DATA tilbfall.reak87 ;
```

Etter at vi har sagt hva datasettet vi vil lage skal hete, må vi angi filnavnet til rådatafilen det skal leses fra. Det gjør vi i **INFILE**-setningen. Der oppgir vi det fulle filnavnet, i enkle fnutter (`'`). `TRUNCOVER` og `LRECL=` er options til INFILE-setning. `TRUNCOVER` tar vi med for å unngå problemer hvis det mangler

verdier på slutten av recorden, `LRECL=` bør være med for sikkerhets skyld (den *må* være med hvis recordlengden er lang, derfor er det best å ha den med alltid).

```
INFILE '$KRIMINAL/reaksjon/aargang/dommer.g87' TRUNCOVER LRECL=217;
```

I neste setning, **INPUT**-setningen, oppgir vi alle detaljer om hva vi vil lese fra recorden. Vi oppgir startposisjon for hver variabel, hva variablene på SAS datasettet skal hete og hvilken type og format hver variabel skal ha. INPUT-setningen er en veldig viktig setning! Det vi skriver der, vil være bestemmende for den videre programmeringen.

```
INPUT     @2   fnr         $CHAR11.
          @2   fdato       DDMMYY6.
          @17  domsdato    YYMMDD6.
          @29  sivst        $CHAR1.
          @66  bot               7.
          @99  sikraar           2.
          ;
```

For hver variabel vi vil lese inn, posisjonerer vi oss til der feltet starter. Det gjør vi med `@`-tegnet etterfulgt av posisjonen. `@2` betyr derfor at vi vil starte i posisjon 2 på recorden. Fra posisjon 2 vil vi lese inn en variabel som skal ha navn `fnr` Variabelen skal leses inn med SAS innlesningsformatet `$CHAR11.` Formatet `$CHAR11.` innebærer at variabelen blir en karaktervariabel (`$`), at verdien ikke venstrejusteres (`CHAR`) og at 11 posisjoner leses inn. Det er innlesningsformatet som bestemmer om variabelen skal være av type numerisk, karakter eller dato. Det finnes en rekke slike innlesningsformater, men det er bare noen ganske få vi vanligvis benytter. (De fleste er med i eksempelet over!)
Neste variabel som skal leses inn skal hete `fdato` Den starter også i posisjon 2 (`@2`). Vi kan altså gjerne lese samme posisjon på recorden flere ganger. Fødselsdato er en del av fødselsnummeret, men det vil ofte være hensiktsmessig å ha fødselsdato som en egen variabel. I vårt eksempel leser vi inn variabelen med innlesningsformatet `DDMMYY6.` Dette innebærer at variabelen blir av type dato. Formatet `DDMMYY6.` forutsetter at datofeltet går over 6 kolonner og at det er på formen ddmmåå, slik det er i fødselsnumre.
Neste variabel som skal leses inn starter i kolonne 17 (`@17`). Den skal hete `domsdato`, den er av type dato og skal leses inn i 6 kolonners bredde (`YYMMDD6.`). Bemerk at datofeltet her er på formen ååmmdd, i motsetning til datofeltet i fødselsnummeret. (Det finnes en rekke forskjellige SAS datoformat å velge mellom!) Variable som er av type dato, gjør det enkelt å finne varigheter. Siden både domsdato og fødselsdato er lest inn som variable av type dato, kan vi finne antall dager fra fødselsdato til domsdato ved ganske enkelt å trekke den ene variabelen fra den andre.
Neste variabel starter i kolonne 29 (`@29`) skal hete `sivst` og skal leses inn med formatet `$CHAR1.` Formatet `$CHAR1.` innebærer at variabelen blir av type karakter og at det bare er én posisjon som leses inn.
Hittil i eksempelet har vi lest inn variable av type karakter og av type dato. Nå skal vi lese inn et par numeriske variable. Den første skal starte i posisjon 66 (`@66`), den skal hete `bot` den skal være av type numerisk (starter ikke med `$`) og den skal leses inn i 7 posisjoners bredde (`7.`), altså fra og med posisjon 66 til og med posisjon 72. Den andre skal starte i posisjon 99 (`@99`), den skal hete `sikraar` være numerisk og leses inn over 2 posisjoner (`2.`).

```
          @66  bot               7.
          @99  sikraar           2.
```

Bemerk at det alltid er et punktum involvert når vi bruker et SAS format! I eksemplene ovenfor er punktumet plassert bakerst i formatet. For karakter- og dato-format er det alltid slik, men for numeriske format står punktumet sist bare i de tilfelle der feltet som leses inn ikke inneholder desimaler. Dersom bøtene i eksempelet hadde vært oppgitt i øre, kunne vi lest feltet inn som kroner ved å bruke formatet `7.2` Formatet `7.2` betyr at feltet skal leses inn 7 posisjoner bredt og med 2 desimaler. Bemerk at tallet som står til venstre for punktumet er *total* bredde på feltet. `7.2` betyr altså 7 posisjoner totalt, hvorav 2 posisjoner er desimaler.

Et variabelnavn kan være på maksimalt 8 tegn. Det kan være vanskelig å lage gode, forståelige navn så korte. For å bøte litt på dette, har vi **LABEL**-setningen. I LABEL-setningen gir vi variablene mer utførlige forklaringer. En label kan være på opptil 40 tegn og i labelen er alle tegn lov, inklusive æøå. Men siden hver label omgis av fnutter ('), kan ikke labelteksten uten videre inneholde fnutt, det ville oppfattes som slutt på labelen. (Vil du absolutt ha med en fnutt i labelteksten, må du skrive *to* fnutter rett etter hverandre. Det blir til én fnutt i labelen.)

```
LABEL    fnr      = 'Fødselsnr'
         fdato    = 'Fødselsdato'
         domsdato = 'Domsdato'
         bostedp  = 'Bosted (fra politiet)'
         provetid = 'Prøvetid'
         sikraar  = 'Sikring, antall år'
         bosted   = 'Bosted (fra Personreg)'
         alder    = 'Alder på gjerningstidspkt, år'
         dagtdom  = 'Antall dager fra gjerning til dom'
         ;
```

LABEL-setningen er nyttig på flere måter: Det tar seg penere ut i tabeller om det står `Prøvetid` enn om det hadde stått `PROVETID`. Ikke bare får vi med eventuelle æ'er, ø'er og å'er, men vi kan også bruke små bokstaver. Hvis en variabel ikke har noen label, brukes variabelnavnet med store bokstaver i tabeller.
I tillegg til det rent estetiske, har lablene stor verdi som forklaring på hva den enkelte variabel egentlig rommer. En utenforstående vil ikke uten videre skjønne at for eksempel `bostedp` betyr bosted oppgitt fra politiet, mens `bosted` betyr bosted hentet fra Personregisteret.
Legg merke til at LABEL-setningen i eksempelet ikke bare gir labler til de variable som leses inn i INPUT-setningen, men også til nye variable som *lages* senere i DATA-steget. Ha som huskeregel at samtlige variable, også nye som lages, skal ha labler.

Gi *alle* variable labler!

Etter LABEL-setningen kommer noen kommentar-setninger som sier hva som skal skje videre i programmet. Deretter kommer et eksempel på en av de meget få setninger som *ikke* starter med et nøkkelord, nemlig **assignment-** eller **tilordnings**-setningen. Det er tilordnings-setningen vi bruker for å lage nye variable. Vi skriver navnet på variabelen vi vil lage på venstre side av likhetstegnet, på høyre side står det hvordan denne nye variabelen skal beregnes. Setningen avsluttes på vanlig måte med semikolon.
I eksempelet skal det beregnes hvor gammel personen var da ugjerningen ble utført, alder ved gjerningstidspunktet. Vi har variable som angir gjerningstidspunktet (`gjtid`) og fødselsdato (`fdato`). Siden disse variablene er lest inn med datoformat, kan vi finne antall dager mellom dem ved ganske enkelt å trekke den ene fra den andre. Datoformatene tar hensyn til skuddår, ulike antall dager i de forskjellige måneder osv. Uttrykket `gjtid - fdato` gir oss antall dager fra det ene tidspunkt til det andre. Siden vi vil ha alderen i antall år, må vi dele på 365.25 som er gjennomsnittlig antall dager i et år. Divisjon medfører *desimaler*. Vi vil ha alder i hele år, derfor må vi avrunde. Dette gjør vi med SAS funksjonen ROUND som avrunder korrekt. Det finnes en mengde SAS funksjoner til forskjellige formål, ROUND-funksjonen er blant de som brukes ofte. Felles for SAS funksjonene er at *argumentet*, det som funksjonen skal virke på, står i parenteser. I eksempelet er det to sett parenteser. Dette er for å ha styring med hva som skjer når: Først skal differansen mellom fødsel og ugjerning beregnes: `(gjtid-fdato)`, deretter skal dette tallet divideres med 365.25: `(gjtid-fdato)/365.25` Til sist avrundes tallet: `ROUND((gjtid-fdato)/365.25)`

```
* Skal beregne alder (år) på gjerningstidspunktet.                ;
* Finner først alder i antall dager. 1 år er 365,25 dager.        ;
* Finner dermed alder i år ved å dividere med 365,25.             ;

alder   = ROUND( (gjtid - fdato) / 365.25 ) ;
```

Videre i eksempelprogrammet kommer noen flere kommentarlinjer og enda en tilordnings-setning. Så kommer noe nytt: En **IF-THEN-ELSE**-konstruksjon. IF-THEN-ELSE brukes hvis noe skal skje *bare* hvis de eller de betingelser er oppfylt. I eksempelet vårt skal vi beregne antall dager soningstid. Det er slik at hvis straffen er ubetinget fengsel i 180 dager eller mer, så skal soningstiden settes til to tredeler av straffen. Hvis ikke denne betingelsen er oppfylt, skal soningstiden settes lik hele straffen. Vi skal altså lage en ny variabel som inneholder soningen. Til dét bruker vi en tilordnings-setning. Siden soningen beregnes på forskjellig måte alt ettersom idømt straff er sånn eller sånn, brukes IF-THEN-ELSE-konstruksjon sammen med tilordnings-setningen. Vi ønsker soningen i hele antall dager. Siden det inngår multiplikasjon med en brøk, må vi avrunde med ROUND-funksjonen:

```
IF    dagubet >= 180 THEN soning = ROUND(dagubet * 2 / 3) ;
ELSE                      soning = dagubet ;
```

Den siste setningen i eksempelprogrammet, er en RUN-setning. Dette avslutter DATA-steget. For at et program skal være komplett, må det være med et RUN-setning helt til slutt. Når programmet skal utføres, submitter vi det. Hvis vi har submittet et program, men det ikke skjer noe, er det høyst sannsynlig at vi har glemt RUN-setningen. Da er det tilstrekkelig å bare skrive dét, og så submitte RUN-setningen alene. `RUN;`

5.2 Bearbeide eksisterende SAS datasett

DATA-steget brukes til langt mer enn bare å lage SAS datasett av rådata. Det er mange situasjoner der vi ønsker å benytte *eksisterende* SAS datasett i et DATA-steg. Det kan for eksempel være at vi ønsker å lage et arbeidsdatasett som bare er et *utdrag* av et eksisterende datasett, vi kan ønske å legge til *flere variable* på datasettet, eller vi kan ønske å *omstrukturere* datasettet. Omstrukturering kan for eksempel innebære at vi ønsker å få datasettet over fra å inneholde én observasjon pr person til å inneholde én observasjon pr husholdning (der opplysninger om alle husholdningens personer ligger som variable bortover). For slike formål benytter vi **SET**-setningen til å hente det eksisterende SAS datasett inn i DATA-steget.
Andre situasjoner der vi benytter DATA-steg er når vi ønsker å koble sammen flere datasett, eller vi vil oppdatere et datasett. Også i disse situasjonene må eksisterende SAS datasett gjøres tilgjengelige i DATA-steget. For å koble, matche datasett brukes **MERGE**-setningen, for å oppdatere datasett brukes **UPDATE**-setningen. Både MERGE- og UPDATE-setningene må brukes sammen med en BY-setning.

5.2.1 Lage utdrag av datasett, tilføye nye variable

Noen statistikkområder har store SAS datasett som inneholder *'alt'*. Datasettet inneholder mye, mye mer enn det du skal arbeide med. I slike tilfelle vil det være fornuftig å lage et arbeidsdatasett som kun inneholder den informasjon du trenger. Vi tar «Tilbakefallsprosjektet 1987» som eksempel: Der er en av problemstillingene følgende: «Hvem av de kriminelle fra 1987 er senere døde? Og isåfall hvor lang tid gikk det fra de ble dømt til de døde?» For å finne ut dette, trenger vi *litt* informasjon fra Personregisteret som er et *kjempestort* register. Vi trenger dødsdato og fødselsnummer for personene som er døde i 1987 eller senere. Disse opplysningene vil vi så koble sammen med datasettet med de straffedømte fra 1987. (Vi *kunne* gjort koblingen mellom Personregisteret og vårt reaksjons-datasett i én jafs, men siden kobling mot store registre er svært ressurskrevende, vil det lønne seg bedre å lage et utrekk fra det store registeret *før* vi kobler.) Vi lager derfor først et temporært datasett som er et uttrekk fra Personregisteret.
For å lage et nytt SAS datasett utfra et eksisterende, må det eksisterende datasettet hentes inn i DATA-steget. Det gjøres ved hjelp av **SET**-setningen.

Eksempel på program som lager SAS datasett av et eksisterende datasett:

```
***------------------------------------------------------------------------***;
*  TILBAKEFALLSPROSJEKTET   1987                                              ;
*                                                                             ;
*  Programfil: $KRIMINAL/tilbfall/prog/preg.sas                               ;
*  Skrevet:    21 juli 1995,  KrL                                             ;
*                                                                             ;
*  Henter dødsdato fra Personregisteret                                       ;
*  Lager et temporært datasett med de døde.                                   ;
***------------------------------------------------------------------------***;

DATA dode87 ;
   SET befolk.persreg (KEEP=fodselnr regstat dodsdato);
   * Vil bare ha med de som er døde => registreringsstatus = 1 ;
   * De skal være døde etter 1987                              ;
   IF regstat = '1' AND dodsdato >= '01JAN1987'D ;
   DROP regstat ;
RUN;
```

I SAS sier vi *først* hva vi skal *lage*. Det gjør vi med `DATA dode87;` Der sier vi at vi vil lage et temporært datasett som skal ha navn `dode87` Datasettet blir temporært fordi det er ett og ikke to ledd i navnet. Deretter sier vi hva vi skal lage det nye datasettet utfra. Det gjør vi med neste setning `SET befolk.persreg` Etter navnet på datasettet vi henter inn, står det noe i parentes: Dette er *datasett options* som i dette tilfelle benyttes for å si ifra at vi bare vil ha med *noen* av variablene på datasettet: `(KEEP=fodselnr regstat dodsdato)` Hvis vi samtidig for eksempel hadde villet *endre navn* på noen av variablene, kunne vi også gjort dette med datasett options, alt plassert i de samme parentesene `(KEEP=fodselnr regstat dodsdato RENAME=(fodselsnr=fnr dodsdato=ddato))`
I eksempelet vil vi lage et datasett som kun består av personer som døde etter 1. januar 1987. (De som er døde har verdien 1 i variabelen `regstat`.) En slik utvelgelse gjør vi med den såkalte **subsetting IF**-setningen(*): `IF regstat = '1' AND dodsdato >= '01JAN1987'D;` Det er kun de observasjonene som tilfredsstiller betingelsen som blir med på datasettet. Variabelen `dodsdato` har verdi lagret med dato-format. Ved å angi 1. januar 1987 i fnutter med en `D` bakerst, slik: `'01JAN1987'D` vil SAS forstå at dette dreier seg om en verdi lagret med dato-format.
(*) Setningen subsetting IF brukes til å velge ut hvilke observasjoner som skal være med på det nye datasettet. Hvis det som står mellom IF og semikolonet altså *betingelsen* er sann, skal observasjonen være med, ellers ikke.

5.2.2 Koble SAS datasett

Vi holder oss til eksempelet med «Tilbakefallsprosjektet 1987» der vi nå ønsker å vite hvem av 'våre' kriminelle som senere er døde. For å finne ut dette, trenger vi en ny variabel på kriminelle-datasettet, nemlig dødsdato. I programeksempelet ovenfor laget vi et temporært datasett som består av alle personer som er døde etter 1. januar 1987. Ved å koble dette datasettet med kriminelle-datasettet etter fødselsnummer, vil vi få laget et nytt datasett der variabelen for dødsdato er hektet på hver enkelt observasjon fra vårt opprinnelige kriminelle-datasett. De observasjonene (egentlig *fødselnummer*) som finnes på begge datasett, altså de som matcher, vil få en gyldig dato som verdi i variabelen for dødsdato, mens de observasjoner som *kun* finnes på kriminelle-datasettet vil få manglende verdi (missing) for dødsdato. De observasjoner som finnes blant de døde men som *ikke* finnes blant våre kriminelle er vi ikke interessert i, disse er jo døde ikke-kriminelle.
Når vi skal koble SAS datasett, brukes **MERGE**-setningen og **BY**-setningen. I MERGE-setningen sier vi hvilke datasett som skal kobles, og i BY-setningen sier vi hvilken variabel som skal være koblingsnøkkel. Når datasett skal kobles, må vi *alltid* bruke en (eller flere) felles variabel som koblingsnøkkel. Koblingsvariabelen kalles i SAS *BY-variabelen.*
Før vi skal koble datasett må vi tenke nøye gjennom følgende:

- ***Hvilken variabel skal være nøkkel til koblingen?***

I vårt eksempel er koblingsbegrepet fødselsnummer. På datasettet `tilbfall.pers87` heter variabelen `fnr` mens på datasettet `dode87` er variabelnavnet `fodselnr`

- ***Hvilke variable skal være med på det koblete datasettet?***

Det nye datasettet skal ha alle variable fra vårt opprinnelige datasett `tilbfall.pers87` mens vi bare vil hente dødsdato (og selvsagt koblingsnøkkelen fødselnummer) fra `dode87`

- ***Hvilke observasjoner skal være med ut på det koblete datasettet?***

I vårt tilfelle vil vi koble variabelen dødsdato fra datasettet dode87 på alle observasjoner i datasettet tilbfall.pers87 . Vi vil altså ha med alle observasjonene fra tilbfall.pers87 men kun disse, ut på det nye datasettet.

Eksempel på program som blant annet kobler SAS datasett:

```
***------------------------------------------------------------------------***;
*  TILBAKEFALLSPROSJEKTET   1987                                              ;
*                                                                             ;
*  Programfil: $KRIMINAL/tilbfall/prog/kobledod.sas                           ;
*  Skrevet:    12 juli 1995,  KrL                                             ;
*                                                                             ;
*  Er noen av de straffede fra 1987 senere døde?                              ;
*  Kobler datasettet med reaksjoner mot                                       ;
*         datasettet med døde (etter 1/1-1987, hentet fra Person-registeret)  ;
***------------------------------------------------------------------------***;

* Kobler eventuell dødsdato på den enkelte straffedømte:  ;

DATA tilbfall.koblet ;
   MERGE tilbfall.pers87 (IN=frapers)
                 dode87 (IN=fradode KEEP=fodselnr dodsdato
                                   RENAME=(fodselnr=fnr) )
        ;
   BY fnr;
   IF frapers;
   * Vil lage en variabel som angir kjønn: ;
   * Posisjon nr 9 i fødselsnr avgjør om mann eller kvinne ;
   * Hvis posisjon nr 9 oddetall =>  mann   => verdi for kjønn skal være 1 ;
   * Hvis posisjon nr 9 partall  =>  kvinne => verdi for kjønn skal være 2 ;
   IF      SUBSTR(fnr,9,1) IN ('1','3','5','7','9') THEN kjonn = '1' ;
   ELSE IF SUBSTR(fnr,9,1) IN ('0','2','4','6','8') THEN kjonn = '2' ;
   LABEL kjonn = 'Kjønn' ;

* Sjekker om antall døde ser rimelig ut ;

PROC FREQ DATA=tilbfall.koblet ;
   TABLES dodsdato /MISSING;
   FORMAT dodsdato MONYY5.;
   TITLE 'Antall straffedømte i 1987 som senere er døde. Ser det rimelig ut?';
RUN;
```

Forklaring til koblingen:

Aller først i programmet er det noen kommentar-setninger. Det er en god vane å starte alle program med en slik kommentarblokk. Etter kommentarene starter selve programmet. Siden kobling er blant de ting som *må* gjøres i DATA-steget, kommer en DATA-setning først. Der sier vi ifra at vi vil lage et permanent datasett: `tilbfall.koblet`

Selve koblingen skjer i MERGE-setningen. Der lister vi opp de to datasettene som skal kobles, det permanente med de kriminelle fra 1987 `tilbfall.pers87` og det temporære med alle som er døde etter 1987 `dode87` Det lønner seg å skrive de to datasettene på hver sin linje, det blir mest oversiktlig. Den naturlige rekkefølgen er å skrive navnet på det 'opprinnelige' datasettet først `tilbfall.pers87` mens datasettet som vi henter noe *fra* nevnes sist `dode87` .

Når vi skal koble datasett bruker vi som regel flere datasett options, både for å si eksplisitt hvilke variable vi vil ha med `(KEEP=....)`, for å endre navn på variable `RENAME=(gmlt1=nytt1 gmlt2=nytt2 ..)` og for å kunne holde rede på hvilket datasett observasjonene kommer fra `(IN=....)`.

I vårt eksempel skal *alle* variable fra `tilbfall.pers87` være med, mens *kun* variablene for fødselsnr og dødsdato skal være med fra `dode87` Derfor skriver vi datasett option `(KEEP=fodselnr dodsdato)` etter datasett navnet `dode87`

Legg merke til at *koblingsvariabelen*, BY-variabelen, har forskjellig navn på de to datasettene, `fnr` og `fodselnr` Idet vi kobler, *må* BY-variabelen fra de to datasettene ha samme navn, vi må derfor endre variabelnavnet idet vi henter koblingsvariabelen fra ett av datasettene. Det gjør vi med datasett option `(RENAME=(fodselnr=fnr))` Vi velger selv om vi endrer navn på BY-variabelen fra det ene eller det andre datasettet. Bemerk at variabelnavnet *ikke* endres på det originale datasettet!

Etter hvert av datasettene bruker vi datasett option `(IN= )` Etter navnet på kriminal-datasettet skriver vi `(IN=frapers)` mens etter navnet på datasettet med de døde skriver vi `(IN=fradode)` Dette innebærer at det lages to nye, flyktige hjelpevariable, `frapers` og `fradode`. Hjelpevariablene er Boolske, logiske, det vil si at de har verdi sann/usann (1/0). Hjelpevariablene kommer *ikke* med på det nye datasettet, men kan brukes i IF-setninger for å velge ut de observasjonene som skal være med på det nye, koblete datasettet. Dersom observasjonen som behandles i øyeblikket kommer fra datasettet `tilbfall.pers87` vil `frapers` være sann, ellers usann. Dersom observasjonen kommer fra datasettet `dode87` er `fradode` sann, ellers usann.

I vårt eksempel er det setningen subsetting IF som brukes til å velge ut hvilke observasjoner som skal være med på det nye datasettet. Hvis det som står mellom IF og semikolonet (altså *betingelsen*) er sant, skal observasjonen være med, ellers ikke. I setningen `IF frapers;` er betingelsen sann hvis `frapers` er sann, med andre ord hvis observasjonen kommer fra kriminal-datasettet. Vi får altså med alle observasjonene fra `tilbfall.pers87` og *kun* disse.

```
DATA tilbfall.koblet ;
   MERGE tilbfall.pers87 (IN=frapers)
                 dode87 (IN=fradode KEEP=fodselnr dodsdato
                                    RENAME=(fodselnr=fnr) )
         ;
   BY fnr;
   IF frapers;
```

☞ **Ikke koble flere enn 2 datasett om gangen!**

☞ **Bruk alltid KEEP= slik at bare de variable som virkelig trengs blir med!**

☞ **BY-variabelen må ha samme navn, bruk om nødvendig RENAME=**

☞ **Alle datasett options til et datasett i én og samme parentes!**

☞ **Tenk nøye igjennom: Hvilke observasjoner skal beholdes?**

Når vi kobler datasett, må vi altså alltid reflektere over følgende: *Hvilke observasjoner vil vi ha med på det nye datasettet?* Bare de som finnes på begge datasett, altså de som virkelig matcher? Alle observasjoner fra begge datasettene? Alle fra ett av datasettene?
I vårt eksempel ville vi koble variabelen dødsdato fra døde-datasettet på *alle* observasjoner i kriminelle-datasettet. Alle de kriminelle ville få en variabel for dødsdato (variabelen ville mangle verdi for de kriminelle som ikke er døde). Vi ville altså lage et nytt datasett med *alle* observasjonene fra kriminelle-datasettet. For å få til denne utvelgelsen av observasjoner, brukte vi subsetting IF slik:
`IF frapers;`

Vi kunne også tenkt oss at vi ville lage et datasett med *bare* de observasjonene som matchet, altså med de observasjoner med likt fødselsnummer som finnes på begge datasett, de døde kriminelle. Da ville vi også brukt subsetting IF, men med en *sammensatt* betingelse: `IF frapers AND fradode;` I dette tilfelle vil betingelsen være sann bare hvis *både* `frapers` og `fradode` er sanne, med andre ord når vi har en 'ekte' match.
Et tredje alternativ er å ville styre observasjonene ut til forskjellige datasett. Dette kan vi også bruke de Boolske hjelpevariablene til, men da i IF-THEN-ELSE-setninger kombinert med OUTPUT-setninger.

Forklaring til resten av eksempelet:

Samtidig som vi kobler på dødsdato vil vi gjerne lage en ny variabel på datasettet, *kjønn*. Det midterste tegnet i personnummeret (altså tegn nr 9 i fødselsnummeret) sier om personen er mann eller kvinne. Hvis dette tegnet er et partall er personen kvinne, hvis tegnet er et oddetall er personen mann. Til en slik hvis-så situasjon bruker vi **IF-THEN-ELSE**-setninger kombinert med **tilordnings**-(assignment-)setninger. I IF-THEN-ELSE er det alltid med en betingelse. Hvis betingelsen som står etter IF er sann, blir det som står etter THEN utført. Hvis betingelsen *ikke* er sann, går programmet videre til en eventuell ELSE-gren. I ELSE-grenen kan vi ha med en ny betingelse med IF-THEN som i eksempelet. Vi kan ha så mange ELSE IF-grener vi vil. Hvis ønskelig, kan vi også ha med en ELSE-gren som utføres hvis *ingen* av de andre betingelsene er oppfylt.
For å trekke ut det 9. tegnet i fødselsnummeret, bruker vi SUBSTR-funksjonen: `SUBSTR(fnr,9,1)` For å finne ut om dette tegnet er et oddetall, bruker vi IN-operatoren: `IN ('1','3','5','7','9')` Dersom denne betingelsen er sann, tilordnes variabelen for kjønn verdien 1: `kjonn = '1';` Hvis betingelsen derimot ikke er sann, går programmet videre til ELSE IF-grenen. Der testes det på om tegn nr 9 er et partall. Hvis dette er tilfellet, utføres det som står etter THEN, nemlig at variabelen for kjønn får verdien 2: `kjonn = '2';`

```
* Vil lage en variabel som angir kjønn: ;
* Posisjon nr 9 i fødselsnr avgjør om mann eller kvinne ;
* Hvis posisjon nr 9 oddetall =>  mann   => verdi for kjønn skal være 1 ;
* Hvis posisjon nr 9 partall  =>  kvinne => verdi for kjønn skal være 2 ;
IF      SUBSTR(fnr,9,1) IN ('1','3','5','7','9') THEN kjonn = '1' ;
ELSE IF SUBSTR(fnr,9,1) IN ('0','2','4','6','8') THEN kjonn = '2' ;
LABEL kjonn = 'Kjønn' ;
```

I eksempelet er det bare én ELSE IF-gren. Hvis situasjonen tilsier det, kan vi ha så mange ELSE IF-grener vi vil! Hvis vi også ønsker en 'sekkepost' til slutt, kan vi ha med en ELSE-setning til slutt. ELSE-setningen er uten betingelse og uten ordet THEN, den utføres hvis ingen av testene over har slått til. (I eksempelet var det ingen ELSE-gren. Hva hvis *ingen* av de to testene hadde slått til? Da hadde variabelen for kjønn fått missing verdi. Imidlertid *vet* vi i eksempelet at vi kun har gyldige fødselsnumre på datasettet, dette har vi sjekket på forhånd.)
LABEL-setningen tilslutt sørger for at den nye variabelen for kjønn også får en label.

***Alle* variable, også nye, bør ha labler!**

Uansett hva vi gjør i et program, lønner det seg alltid å sjekke resultatet! I program-eksempelet skulle vi nå ha fått koblet dødsdato på de kriminelle. De som er døde har fått en gyldig dato i variabelen for dødsdato, mens de som ikke er døde har missing verdi for dødsdato. For å se at resultatet ser rimelig ut, lager vi en frekvenstabell med PROC FREQ: `PROC FREQ DATA=tilbfall.koblet ;` I `DATA=` sier vi hvilket datasett proc'en skal utføres på. (Ikke glem =-tegnet i `DATA=` da tror SAS at dette er starten på et DATA-steg!) PROC FREQ teller opp hvor mange observasjoner som har *samme verdi* for en variabel og lager en tabell som viser dette. Hver verdi får en linje i tabellen. På den linjen står det antall observasjoner med denne verdi, hvor mange prosent disse utgjør osv. Hvilken variabel vi vil ha i tabellen, oppgir vi i TABLES-setningen (som er en setning som hører til PROC FREQ, det kan ikke brukes andre steder). Hvis vi vil at manglende verdi (missing)også skal være en linje i tabellen og

inngå i beregningen av prosenter, har vi med MISSING som options til TABLES-setningen: `TABLES dodsdato /MISSING;` I vårt eksempel er det de som ikke er døde som mangler verdi for dødsdato, disse vil vi jo ha med i tabellen. Men vil det ikke bli en forferdelig lang og uoversiktlig tabell hvis hver eneste dødsdato skal ha sin egen linje i tabellen? Det vil i praksis si én linje for hver dato en kriminell er død. Dette problemet løser vi med FORMAT-setningen: `FORMAT dodsdato MONYY5.;` I denne setningen *formaterer* vi dødsdatoen med et av SAS datoformatene for måned og år. Dette innebærer at vi ikke får én linje for hver eneste dato i variabelen, vi får isteden samlet opp datoene månedsvis. Dette gjør tabellen mye mer håndterlig. (Det finnes mange andre datoformat vi kunne valgt!)

```
* Sjekker om antall døde ser rimelig ut ;

PROC FREQ DATA=tilbfall.koblet ;
   TABLES dodsdato /MISSING;
   FORMAT dodsdato MONYY5.;
   TITLE 'Antall straffedømte i 1987 som senere er døde. Ser det rimelig ut?';
```

Utdrag av tabellen:

```
Antall straffede i 1987 som senere er døde. Ser det rimelig ut?

Dødsdato
```

DODSDATO	Frequency	Percent	Cumulative Frequency	Cumulative Percent
.	10428	94.5	10428	94.5
MAY87	2	0.0	10430	94.5
JUL87	2	0.0	10432	94.6
AUG87	3	0.0	10435	94.6
SEP87	3	0.0	10438	94.6
OCT87	3	0.0	10441	94.6
NOV87	1	0.0	10442	94.7
DEC87	11	0.1	10453	94.8
JAN88	3	0.0	10456	94.8
FEB88	6	0.1	10462	94.8
MAR88	5	0.0	10467	94.9
APR88	5	0.0	10472	94.9
MAY88	4	0.0	10476	95.0
JUN88	6	0.1	10482	95.0
JUL88	11	0.1	10493	95.1
AUG88	3	0.0	10496	95.1
SEP88	7	0.1	10503	95.2
OCT88	9	0.1	10512	95.3
NOV88	11	0.1	10523	95.4
DEC88	10	0.1	10533	95.5
JAN89	4	0.0	10537	95.5
FEB89	5	0.0	10542	95.6
MAR89	5	0.0	10547	95.6
APR89	3	0.0	10550	95.6
MAY89	10	0.1	10560	95.7
...				

5.2.3 Omstrukturere datasett

En situasjon vi ofte støter på, er at vi har tilgjengelig et datasett som ikke er organisert på den måten som det passer best for oss. Det kan for eksempel være det finnes et datasett der hver observasjon representerer én person slik at en husholdnings personer ligger som hver sin observasjon under hverandre, mens det vi ønsker er et datasett med én observasjon pr husholdning og opplysningene om alle husholdningens personer som variable bortover. Det motsatte kan også være tilfellet, vi har et datasett der en persons sykehistorie ligger som variable bortover, mens det vi ønsker er at hvert eneste sykdomstilfelle skal være en egen observasjon. I slike tilfeller må datasettet *omstruktureres.* Vi sier gjerne at vi '*vrenger*' datasettet nedover (gjør om variable til observasjoner) eller bortover (gjør om observasjoner til variable).

5.2.3.1 Gjøre om observasjoner til variable, 'vrenge' bortover

I eksempelet under har vi et datasett der hver observasjon representerer én person i en husholdning. Datasettet inneholder blant annet personens fødselsnr (fnr), husholdningsnr (hushnr) og vedkommendes inntekt (inntekt). Vi ønsker å lage et datasett der hver observasjon representerer en husholdning. Opplysninger om alle husholdningens medlemmer (fnr og inntekt) skal ligge som variable bortover på husholdningsobservasjonen. Vi vil også beregne husholdningens samlede inntekt (hushinnt). For å vite hvor mange variable vi trenger på det nye datasettet, må vi vite hvor mange personer den største husholdningen består av. I vårt eksempel er det ingen husholdning som har flere enn 20 medlemmer. Dette har vi på forhånd funnet ut med en frekvenstabell som viser antall observasjoner for hvert enkelt husholdningsnummer. (Frekvenstabeller lages med PROC FREQ.)

Eksempel på program som gjør om observasjoner til variable:

```
***------------------------------------------------------------------------***;
*  HUSHOLDNINGERS INNTEKT                                                    ;
*                                                                            ;
*  Programfil: $SKATT/prog/pers2hh.sas                                       ;
*  Skrevet:    29 januar 1996,  KrL & LDa                                    ;
*                                                                            ;
*  Har data om personers inntekt, ønsker data på husholdningsnivå.           ;
*  Vil bl.a ha en variabel med husholdningens samlede inntekt.               ;
*  Vet (av PROC FREQ) at ingen husholdninger har fler enn 20 medlemmer.      ;
***------------------------------------------------------------------------***;

DATA skatt.hushold ;
   SET skatt.personer ;
   BY hushnr ;
   ARRAY fnummer (20) $ 11 fnr1-fnr20 ;
   ARRAY innt    (20)      innt1-innt20 ;
   RETAIN fnummer innt ;
   antpers + 1 ;
   fnummer(antpers) = fnr ;
   innt(antpers) = inntekt ;
   IF LAST.hushnr THEN
      DO;
         hushinnt = SUM(of innt(*) ) ;
         LABEL hushinnt = 'Husholdningens samlede inntekt'
               antpers  = 'Antall personer i husholdningen'
               ;
         OUTPUT;
         DO i=1 TO antpers ;
            fnummer(i) = '' ;
            innt(i) = . ;
         END;
         antpers = 0;
      END;
   DROP i inntekt fnr ;
RUN;
```

Forklaring til eksempelet:

Omstrukturering av datasett må gjøres i DATA-steget, derfor starter vi med en DATA-setning: `DATA skatt.hushold;` Her sier vi at vi vil lage et permanent datasett `skatt.hushold` Siden vi skal ta utgangspunkt i det eksisterende datasettet `skatt.personer` hentes dette inn med `SET skatt.personer;` Det er viktig for oss å ha kontroll med hvilken husholdning en person tilhører, vi vil behandle datasettet husholdning for husholdning. Derfor skriver vi setningen `BY hushnr;` der BY er nøkkelordet og husholdningsnummeret kalles BY-variabelen. (For å kunne bruke BY-setningen, *må* datasettet være sortert på BY-variabelen.) BY-setningen brukt etter SET-setningen gir oss mulighet til å vite om observasjonen som behandles i øyeblikket er den første innen en husholdning (eventuelt om den er den siste eller hverken først eller sist). Når vi bruker SET med BY lages det nemlig to logiske (Boolske) hjelpevariable som vi kan bruke til å teste om vi er først eller sist innen en verdi av BY-variabelen. Hjelpevariablene heter FIRST.byvariabel og LAST.byvariabel (uttales gjerne first *sin* og last *sin* byvariabel). Når vi skriver `IF FIRST.hushnr THEN ...` betyr det at vi vil noe skal skje dersom

observasjonen er den første med dette husholdningsnr. Tilsvarende betyr `IF LAST.hushnr THEN ...` at noe skal skje dersom observasjonen er den siste innen husholdningen.

Målet for vår omstrukturering av datasettet er å gjøre opplysninger om *personene* i en husholdning om til opplysninger om *husholdningen*. Istedenfor å ha *person* som enhet, skal vi ha *husholdning* som enhet. Med andre ord: Variablene til *mange* observasjoner i person-datasettet skal gjøres om til variable på *én* observasjon på husholdnings-datasettet. Husholdnings-datasettet må derfor ha mange flere variable enn person-datasettet. Når vi skal omstrukturere datasettet slik, benytter vi *arrayer*. Et array er ikke noe annet enn en samling variable. Variablene kan allerede eksistere på datasettet, eller de kan være nye. Det eneste kravet er at variable som er samlet i et array er av samme *type*. Et array kan altså bestå av enten numeriske variable eller karaktervariable.

Et array må defineres i en **ARRAY**-setning: `ARRAY fnummer (20) $ 11 fnr1-fnr20;` Etter nøkkelordet `ARRAY` gir vi arrayet et navn: `fnummer` Deretter angir vi hvor mange variable som skal inngå i arrayet, det vil si antall elementer i arrayet. Dette kalles også arrayets *dimensjon*. `(20)` betyr at arrayet skal bestå av 20 elementer, altså 20 variable. Et array som inneholder karaktervariable må ha en `$` etter dimensjonsangivelsen. Vårt array skal bestå av 20 nye variable `fnr1-fnr20` der hver variabel skal inneholde fødselsnummeret til en person i husholdningen. (Vi vet at det ikke er noen husholdning med fler enn 20 personer.) Et fødselsnummer er en karaktervariabel og består av 11 tegn, derfor både `$` og `11` før variabelnavnene. I eksempelet definerer vi nok et array. Dette skal bestå av 20 numeriske variable som skal inneholde inntekten til hvert husholdningsmedlem: `ARRAY innt (20) innt1-innt20;`

Prinsippet for å gjøre *mange person-observasjoners* variable om til variable på *én husholdnings-observasjon*, er å legge person-variablene inn i arrayer: Hver person-variabel har sin egen array. Variablene til første person i en husholdning blir første element i arrayene, variablene til andre person i husholdningen blir andre element i arrayene, osv osv. Første persons fødselsnr legges inn i første element i fødselsnummer-arrayet, mens første persons inntekt legges inn i første element i inntekt-arrayet. Andre persons fødselsnr legges inn i andre element i fødselsnummer-arrayet, andre persons inntekt legges inn i andre element i inntekt-arrayet, osv osv.

For å holde styr på hvilken person i rekken en person er, har vi med **summe**-setningen laget en tellevariabel `antpers`. Tellevariabelen økes med 1 for hver observasjon som leses inn: `antpers + 1;` og nullstilles mellom hver ny husholdning. Bemerk at en variabel som lages med summe-setningen beholder sin verdi fra observasjon til observasjon, i motsetning til hva som skjer ellers. Variabelen `antpers` inneholder altså husholdningsmedlemmets nummer i rekken. Den kan dermed brukes som *indeks* til arrayene, dvs den kan brukes til å angi elementnummer i arrayene: `fnummer(antpers) = fnr;` betyr at element nr `antpers` i arrayet `fnummer` skal gis verdien til variabelen `fnr` . Når `antpers` f.eks har verdien 4 skal element nr 4 i arrayet `fnummer` (altså variabelen `fnr4`) settes lik verdien til variabelen `fnr` og element nr 4 i arrayet `innt` (altså variabelen `innt4`) settes til verdien av `inntekt` .

Det er veldig viktig at variablene i arrayene beholder sin verdi når ny observasjon leses inn(*). Dette gjør vi med **RETAIN**-setningen: `RETAIN fnummer innt;` Her sier vi at variablene som ligger i arrayene *ikke* skal nullstilles mellom hver observasjon, men beholde sin verdi. Vanligvis ramser vi opp de enkelte variabelnavn i en RETAIN-setning, men siden vi her har lagt variablene i arrayer, er det tilstrekkelig å skrive arraynavnene i RETAIN-setningen.
Vi fyller opp arrayene med variabelverdier så lenge vi holder oss til personer innen en og samme husholdning. Først når vi er kommet til siste person i en husholdning, er husholdnings-observasjonen klar til å bli lagt ut på husholdnings-datasettet. Vi vet når vi er kommet til siste person innen en husholdning, det er når `LAST.hushnr` er sann. Når `LAST.hushnr` er sann, vil vi gjøre flere ting: Vi vil lage en variabel som inneholder husholdningens samlede inntekt, vi vil legge husholdnings-observasjonen ut på husholdnings-datasettet og vi vil nullstille alle fødselsnummer- og inntekts-variablene i arrayene.

Vi ønsker altså å gjøre noe, hvis vi er kommet til *siste* person i en husholdning. Da skriver vi `IF LAST.hushnr THEN` Siden det er *flere* setninger vi ønsker å utføre hvis betingelsen er sann, må setningene legges i en **DO-END**-gruppe. Først `DO;` så setningene, og så `END;` tilslutt.
Det første vi vil gjøre når vi er kommet til siste person innen en husholdning, er å summere husholdningsmedlemmenes inntekt. Disse ligger nå i variablene `innt1-innt20` som igjen ligger i arrayet `innt` For å summere alle inntektsvariablene, kunne vi skrevet `SUM(of innt1-innt20)` men den enkleste måten å gjøre det på, er å bruke *arrayet* som argument i SUM-funksjonen: `SUM(of innt(*) ) .` `innt(*)` betyr *alle* elementene i arrayet. Hele setningen blir derfor slik: `hushinnt = SUM(of innt(*));` Variablene `hushinnt` og `antpers` gis forklarende tekst: `LABEL hushinnt = 'Husholdningens samlede inntekt' antpers = 'Antall personer i husholdningen' ;` før hele observasjonen legges ut på det nye datasettet: `OUTPUT;` Siden både inntektsvariablene og fødselsnummer-variablene er *retainet* er det viktig å huske på å nullstille dem før vi starter på en ny husholdning. Dette gjør vi i en løkke: `DO i=1 TO antpers;` I løkken blanker vi fødselsnr-arrayet: `fnummer(i) = '';` og setter inntekts-arrayet til missing: `innt(i) =. ;` Vi avslutter løkken med: `END;` Så nullstiller vi tellevariabelen: `antpers = 0;` og avslutter DO-END-gruppen. For at ikke uønskete variable skal være med på det nye datasettet, droppes disse: `DROP i inntekt fnr ;`

(*)Dette i motsetning til hva som vanligvis skjer: Vanligvis 'nullstilles' alle variable i det øyeblikk én observasjon er skrevet ut og før en ny skal leses inn. Alle variablene som ikke er nevnt i en RETAIN-setning (bortsett fra variable laget med summe-setningen) settes da til missing.

5.2.3.2 Gjøre om variable til observasjoner, 'vrenge' nedover

I eksempelet foran, gjorde vi observasjoner om til variable. Her følger et eksempel på hvordan vi kan gjøre det motsatte, nemlig gjøre variable om til observasjoner. Vi har et datasett `krim.personer` der enheten er *personer* med kriminelt rulleblad. Hver person er én observasjon på datasettet. De lovbrudd personen er anmeldt for, ligger som 15 variable `lovb1-lovb15` bortover. Hvert lovbrudd har en tilhørende dato for når gjerningen fant sted, `gdato1-gdato15` Vi ønsker nå å lage et datasett `krim.lovbrudd` der enheten er *lovbrudd* med én variabel `lovbrudd` for lovbruddskoden og én variabel `gjerdato` for datoen gjerningen fant sted.
Som i forrige eksempel lønner det seg også nå å legge lovbruddene og tilhørende datoer i arrayer: `ARRAY lovb (15) $ lovb1-lovb15;` og `ARRAY gdato (15) gdato1-gdato15;` Variablene i lovbrudds-arrayet er naturligvis av type karakter, derfor `$` i arraydefinisjonen. Lovbrudds-datoene er i dato-format, de er derfor numeriske av type. Datasettet vi vil lage skal bare ha én observasjon for hvert lovbrudd, de 30 variablene for lovbrudd og datoer skal derfor ikke være med på det nye datasettet: `DROP i lovb1-lovb15 gdato1-gdato15;` Legg merke til at **DROP**-setningen *utføres* helt tilslutt i DATA-steget, uansett hvor setningen er plassert i programmet! Vi kan derfor gjerne droppe variable som ennå ikke finnes, som løkketelleren `i` eksempelet.
For å lage mange observasjoner av én, må vi bruke OUTPUT-setningen inni en løkke. Løkken opprettes med: `DO i=1 TO 15 WHILE (lovb(i) ^= '');` Den skal gjenomløpes maksimalt 15 ganger, men bare så lenge det er en verdi i lovbruddsvariabelen. Vi har lagt lovbruddene med tilhørende datoer i arrayer, og går igjennom arrayene element for element inni løkken. Hvert lovbrudd legges i en ny variabel: `lovbrudd = lovb(i);` det samme skjer med den tilhørende dato: `gjerdato = gdato(i);` Så legges observasjonen ut på det nye datasettet: `OUTPUT;` og løkken avsluttes: `END;`

```
DATA krim.lovbrudd ;
   SET krim.personer (KEEP=id_nr lovb1-lovb15 gdato1-gdato15);
   ARRAY lovb   (15) $ lovb1-lovb15 ;
   ARRAY gdato (15)   gdato1-gdato15 ;
   DROP i lovb1-lovb15 gdato1-gdato15 ;
   DO i=1 TO 15 WHILE (lovb(i) ^= '') ;
      lovbrudd = lovb(i) ;
      gjerdato = gdato(i) ;
      OUTPUT ;
   END;

RUN;
```

Arrayers mulighet til fleksibilitet:

Det er mulig å gjøre program som inneholder arayer mer fleksible enn det som er vist i eksemplene foran. I en arraydefinisjon er det nemlig lov å utelate indeksen som sier hvor mange elementer arrayet består av, vi kan skrive `ARRAY lovb (*) $ lovb1-lovb15;` I løkken der arrayet blir brukt, trenger vi heller ikke eksplisitt skrive hvor mange ganger løkken skal gå rundt, det finnes nemlig en funksjon, **DIM**-funksjonen, som returnerer et arrays dimensjon, altså hvor mange elementer det består av. Vi kan derfor skrive: `DO i=1 TO DIM(lovb) ;` I praksis vil dette si at det kun er ett eneste sted vi trenger å oppgi arrayets dimensjon, og det er sist i arraydefinisjonen der vi ramser opp variablene som skal inngå.
Det er åpenbart at dét å bruke `lovb (*)` og `DIM(lovb)` reduserer muligheten for feil når programmet skal endres. Den dagen maksimum antall lovbrudd er endret fra 15 til 21, holder det å endre ett eneste sted, nemlig bakerst i arraydefinisjonen: `ARRAY lovb (*) $ lovb1-lovb21;`

5.2.4 Oppdatere et SAS datasett med et annet

Når et datasett skal oppdateres med et annet, gjør vi dette i et DATA-steg. En enkel måte å forklare oppdateringer på, er ved et eksempel med endringer i en telefonkatalog. Endringer kan for eksempel være adresseendringer og nye personer som får telefon. Telefonkatalogen er ett SAS datasett, alle endringene ligger på et annet datasett. Datasettet med endringene kalles gjerne et transaksjons-datasett. Begge datasettene inneholder en nøkkelvariabel, for eksempel fødselsnr. Ved oppdatering skal ny adresse erstatte gammel, og det kan gjerne være flere adresseendringer for en og samme observasjon. Hvis det er *flere* endringer for én og samme ident, er det viktig å merke seg at den endring som ligger *sist* i transaksjonsdatasettet er den som blir gjeldende. Setningen som brukes til oppdatering er **UPDATE**-setningen. Det må alltid følges av **BY**-setningen der vi oppgir nøkkel-variabelen. Bruk av BY-setningen betinger at datasettene er sortert på denne variabelen, den kalles BY-variabelen. I UPDATE-setningen nevnes først hoved-datasettet, deretter transaksjons-datasettet.
For å være på den sikre siden når vi skal endre et viktig datasett ved å oppdatere, lagrer vi endringene først på et nytt datasett, et hjelpedatasett. Når vi er sikre på at hjelpedatasettet er i orden, at alle endringene har blitt riktige, kan vi endre navnet på hjelpedatasettet.

```
***------------------------------------------------------------------------***;
*  OPPDATERE TLF-KATALOGEN                                                    ;
*                                                                             ;
*  Programfil: ........................                                       ;
*  Skrevet:    ........................                                       ;
*                                                                             ;
*  Katalog-datasett:   perm.tlfkat                                            ;
*  Endrings-datasett:  perm.endring                                           ;
*                                                                             ;
*  Har datasett med adresse-endringer, navne-endringer etc.                   ;
*  Vil oppdatere tlf-katalogen.                                               ;
*  Lager først et midlertidig datasett, for sikkerhets skyld.                 ;
***------------------------------------------------------------------------***;

DATA mellom ;
   UPDATE perm.tlfkat
          perm.endring
           ;
   BY ident;

RUN;
```

Det er viktig å merke seg at med UPDATE-setningen endres *kun* de verdiene som faktisk står i transaksjon-datasettet. Hvis variable mangler verdi i transaksjons-datasettet, blir likevel *ikke* hoved-datasettet oppdatert med missing (hvilket ville vært tilfellet dersom vi hadde brukt MERGE).

5.3 Lage sekvensiell fil av SAS datasett

SSB har strenge regler for hvordan vi skal lagre data som skal tas vare på for ettertiden. Slike data skal *ikke* lagres som SAS datasett eller i andre 'leverandør-spesifikke' format, men som 'flate', sekvensielle filer.
Når vi skal lage en sekvensiell fil ('rådata', 'flat' fil) av et SAS datasett, lages programmet over samme lest som når vi skal lage SAS datasett fra rådata. Forskjellen er at setningene som brukes, er **FILE**-setningen og **PUT**-setningen istedenfor INFILE og INPUT. FILE- og PUT-setningene virker på helt tilsvarende måte som INFILE- og INPUT, bare med motsatt 'fortegn'.

Eksempel på program som lager en sekvensiell fil av et SAS datasett (programmet er forkortet):

```
***------------------------------------------------------------------**;
*  OVERNATTINGSSTATISTIKK                                             ;
*                                                                     ;
*  Lager en sekvensiell, flat årsfil for langtidslagring.             ;
*                                                                     ;
*  Program:     $REISELIV/hotell/prog/p08arkiv.sas                    ;
*  Lagringsfil:          -     /arkiv/g1995                           ;
*                                                                     ;
*  Skrevet:  29.mars 1996, LDa                                        ;
*  Endret:                                                            ;
***------------------------------------------------------------------**;

DATA _null_ ;
   SET hotell.hele95;
   FILE '$REISELIV/hotell/arkiv/g1995' LRECL=516 ;
   PUT     @1    LOPENR        $CHAR6.
           @7    KOMMUNE       $CHAR4.
           @11   HOTELLGR      $CHAR1.
           ...
           @52   NORDMENN           7.
           @59   SVENSKER           7.
           @66   DANSKER            7.
           @73   FINNER             7.
           ...
           @357  IMP_OMS       $CHAR1.
           @358  IMP_OVER      $CHAR1.
           ...
           @507  ANT_SENG           5.
           @512  ANT_ROM            5.
           ;
   LABEL   LOPENR    = 'Løpenr'
           KOMMUNE   = 'Kommune'
           HOTELLGR  = 'Hotellgruppe'
           ...
           NORDMENN  = 'Nordmenn'
           SVENSKER  = 'Svensker'
           DANSKER   = 'Dansker'
           FINNER    = 'Finner'
           ...
           IMP_OMS   = 'Imputert omsetning?'
           IMP_OVER  = 'Imputert overnattinger?'
           ...
           ANT_SENG  = 'Antall senger'
           ANT_ROM   = 'Antall rom'
               ;
RUN;
```

Legg merke til den første setningen i DATA-steget `DATA _null_ ;` DATA-setningen skrevet slik (altså `_null_`) betyr at det *ikke* skal lages noe datasett i DATA-steget! (Dette i motsetning til hva som ellers er vanlig.) Vi er jo ikke interessert i å lage noe SAS datasett, tvert imot *har* vi et SAS datasett som vi vil lage en flat fil av. Vi henter inn det eksisterende datasettet med `SET hotell.hele95;`
I neste setning, **FILE**-setningen, oppgir vi filnavn og recordlengde for den sekvensielle filen vi vil lage: `FILE '$REISELIV/hotell/arkiv/g1995' LRECL=516 ;` Legg merke til at vi oppgir vi fullstendig navn (med fil-hale) på den nye filen. Vi kunne godt benyttet relativ path for å angi lagrings*katalog*, men selve filnavnet *må* skrives fullt ut. FILE-setningen gir mulighet for svært detaljerte spesifikasjoner av den nye filen, se options til FILE-setningen i **Håndbok i SAS Del 2: Oppslag**.

I PUT-setningen spesifiseres *recorden*. For hvert felt angis først startposisjon, deretter navnet på variabelen, tilslutt utskriftsformatet. For eksempel: `@357 IMP_OMS $CHAR1.` Dette betyr at variabelen `IMP_OMS` skal starte i posisjon 357 og skrives ut som en karakter-variabel i ett tegns bredde.
Program som lager en sekvensiell arkivfil, en fil som skal lagres for ettertiden, er svært viktig som dokumentasjon. Programmet inneholder jo *filbeskrivelsen* til arkivfilen. Derfor er det svært viktig at vi i et slikt program også har med LABEL-setningen, som sier hva hvert enkelt variabelnavn betyr, for eksempel `IMP_OMS = 'Imputert omsetning?'`

Tips! PUT- og LABEL-setninger kan lages ved hjelp av PROC CONTENTS

Alle SAS datasett inneholder *metadata*, informasjon om datasettet selv. Denne informasjon får vi ut ved hjelp av PROC CONTENTS. Ved å lagre resultatet av PROC CONTENTS (SASlisten) på en fil og deretter manipulere denne filen f.eks. i editoren WinEdit, kan vi på en enkel måte lage både PUT- og LABEL-setningene. Ved å bruke denne metoden sparer vi mye skrivearbeid, spesielt når det gjelder filer med mange variable. Metoden er også sikrere, siden SAS selv har skrevet variabelnavnene unngår vi feilkilder forbundet med dette.

```
OPTIONS PS=1000 ;

PROC CONTENTS DATA=hotell.hele95  POSITION ;

RUN;
```

Setningen `OPTIONS PS=1000;` betyr at vi vil ha 1000 linjer pr side i utskriften. Dette for å unngå uønsket sideskift med tilhørende overskrifter som likevel bare skal redigeres bort. (Tallet 1000 er tilfeldig valgt, det er bare å sette et stort nok tall.) Resultatfilen (SASlisten) etter denne kjøringen vil se omtrent slik ut:

```
CONTENTS PROCEDURE

            -----Variables Ordered by Position-----

 # Variable Type Len Pos     Label
-----------------------------------------------------------
 1 LOPENR   Char   6   0     Løpenr
 2 KOMMUNE  Char   4   6     Kommune
 3 HOTELLGR Char   1  10     Hotellgruppe
   ...
11 NORDMENN Num    8  67     Nordmenn
12 SVENSKER Num    8  75     Svensker
13 DANSKER  Num    8  83     Dansker
14 FINNER   Num    8  91     Finner
   ...
52 IMP_OMS  Char   1 395     Imputert omsetning?
53 IMP_OVER Char   1 396     Imputert overnattinger?
   ...
90 ANT_SENG Num    8 634     Antall senger
91 ANT_ROM  Num    8 642     Antall rom
```

De viktige kolonnene er `Variable`, `Type`, `Len` og `Label`, de øvrige kolonnene kan fjernes. `Variable` og `Label` er tilstrekkelig til å lage LABEL-setningen, mens `Variable`, `Type` og `Len` inneholder informasjon som kan brukes i PUT-setningen. For karaktervariablene, de med `Type Char` stemmer det som står i `Len` overens med bredden av variabelen. Dette i motsetning til de numeriske variablene, der det som regel vil stå `Len 8` for alle variablene. Dette vil være tilfellet uansett om variabelen inneholder små tall som alder eller store tall som omsetning. For numeriske variable må vi derfor vite hvor mange siffer det største tallet vil kunne ha, dette finner vi ut med PROC MEANS. (Kolonnen `Pos` kan altså ikke brukes for å posisjonere.)
Hvis datasettet vårt stammer fra en Oracle-tabell, vil resultatet av PROC CONTENTS inneholde flere kolonner, `Informat` og `Format` . Ved hjelp av kolonnen `Informat` kan vi lage et lite SAS program som generer startposisjon for variablene.

5.4 Trekke utvalg

Vi har ofte behov for å trekke utvalg av data: Vi kan ønske å teste et program på en liten datamengde, eller vi skal rett og slett trekke et utvalg av et større datamateriale. Når vi skal lage utvalg kan vi enten ønske å lage et utvalg av omtrentlig størrelse, eller vi kan ønske å lage et utvalg av eksakt størrelse. Et utvalg med omtrentlig størrelse kan for eksempel være et 10 prosent utvalg. Hvis datasettet som er utgangspunkt har 2517 observasjoner, vil utvalget ha ca 250 observasjoner. Skal vi lage et utvalg med eksakt størrelse, vet vi nøyaktig hvor mange observasjoner vi vil ha i utvalget.

5.4.1 Utvalg med omtrentlig størrelse

For å lage et utvalg med omtrentlig størrelse, kan vi bruke RANUNI-funksjonen. Programmet under lager et utvalg på ca 1 prosent:

```
* Lager et ca 1 % utvalg av datasettet perm.allepers  ;

DATA utvalg ;
   SET perm.allepers;
   IF RANUNI(0) <= .01 ;
```

I eksempelet er tallet null argumentet til RANUNI-funksjonen: `RANUNI(0)` Dette betyr at SAS benytter *klokken* internt i datamaskinen som *startverdi* for trekkefunksjonen. Siden klokken stadig endrer seg, vil et forskjellig utvalg av observasjoner bli trukket ut hver gang programmet kjøres. Hver gang vi kjører programmet vil utvalget altså bli *litt* forskjellig fra forrige gang, det vil bestå av andre observasjoner og det vil inneholde et *litt* annet antall observasjoner.

Hvis vi ønsker at samme observasjoner skal velges hver gang vi kjører programmet, må vi skrive et 9-sifret oddetall som argument til funksjonen, for eksempel `RANUNI(123456789)` . Da vil utvalget bli helt likt hver gang programmet kjøres.

5.4.2 Utvalg med nøyaktig størrelse

For å lage et utvalg som består av et nøyaktig antall observasjoner kan vi igjen bruke RANUNI-funksjonen, men programmet blir litt større enn i forrige eksempel. Programmet under lager et utvalg på med nøyaktig 257 observasjoner:

```
* Lager et utvalg på nøyaktig 257 observasjoner ;

DATA utvalg ;
   SET perm.allepers NOBS=totids ;
   RETAIN restiutv 257 restids ;
   IF _N_ = 1 THEN restids=totids ;
   IF RANUNI(123456789) <= restiutv/restids THEN
      DO;
         OUTPUT;
         restiutv = restiutv - 1 ;
      END;
   restids = restids - 1 ;
   IF restiutv = 0 THEN STOP ;
RUN;
```

Antall observasjoner som skal være med i utvalget spesifiseres i RETAIN-setningen, som initialverdi for variabelen som sier hvor mange observasjoner som gjenstår å trekke.

5.5 DATA-steget til å finne dubletter

Hvis vi vil undersøke om vi har dubletter på et datasett, kan vi gjøre dette i et DATA-steg. Det er en forutsetning at datasettet er sortert på nøkkelvariabelen, den vi vil undersøke om vi har dubletter av. I eksempelet under, har vi et datasett der observasjonene representerer personer. Det skal helst ikke være flere observasjoner med samme fødselsnummer (altså fødselsnummerdubletter), men av erfaring vet vi at sånt likevel kan forekomme. Vi ønsker å skille eventuelle dubletter ut på et eget datasett for videre gransking:

```
* Legger fødselsnummer-dubletter ut på et eget datasett  ;

DATA dubletter ;
   SET perm.allepers ;
   BY fnr ;
   IF NOT (FIRST.fnr AND LAST.fnr) ;
RUN;
```

I eksempelet bruker vi en BY-setning i tillegg til SET-setningen. Dette krever at datasettet er sortert på fødselsnummer (BY-variabelen), men det gir oss til gjengjeld mulighet til å vite om vi er først eller sist innen en eventuell gruppe observasjoner med samme fødselsnummer. Uttrykket `FIRST.fnr` er sant hvis og bare hvis vi er ved den *første* observasjonen med et bestemt fødselsnummer mens `LAST.fnr` er sant hvis og bare hvis vi er ved den *siste* observasjonen med et bestemt fødselsnummer. Hvis det kun er én observasjon med dette fødselsnummeret, vil *både* `FIRST.fnr` og `LAST.fnr` være sanne samtidig. Vi kan av dette vite at hvis så *ikke* er tilfellet, nemlig hvis `NOT (FIRST.fnr AND LAST.fnr)` da har vi flere observasjoner med samme fødselsnummer, altså har vi en dublett. I så fall legges denne observasjon ut på det nye datasett med subsetting IF-setningen: `IF NOT (FIRST.fnr AND LAST.fnr) ;`

På små datasett kan vi alternativt bruke PROC FREQ til å finne dubletter, se under PROC FREQ.

6. PROC-steget

I PROC-steget *bruker* vi SAS datasettene vi har laget i DATA-steget(*). PROC-steget brukes til å lage tabeller og analyser av alle slag. PROC-steget brukes dessuten til diverse nyttig, som å sortere datasett, dokumentere innholdet i datasett m.m.m.

(*) Dette er ikke den hele og fulle sannhet: Datasettene vi bruker i PROC-steget trenger ikke nødvendigvis være laget i et DATA-steg, de kan godt være laget i et tidligere PROC-steg, mange proc'er kan lagre resultatene sine i datasett. Det er dessuten ikke alltid det er et SAS datasett som behandles i et PROC-steg, det kan like gjerne være et SAS-*view*. I noen få proc'er er det hverken datasett eller view involvert, for eksempel PROC FORMAT.

Her følger en kort gjennomgang av noen få av de grunnleggende og mest benyttede proc'ene. Husk at det finnes veldig, veldig mange andre proc'er! Noen andre proc'er er beskrevet i **Håndbok i SAS Del 2: Oppslag**. For å finne frem til andre proc'er igjen er vi henvist til SAS OnlineDoc på Byrånettet.

6.1 PROC PRINT

PROC PRINT bruker vi for å skrive ut observasjoner fra et SAS datasett. PROC PRINT er en typisk 'sjekke-og-kontrollere'-proc. Vi bruker den for å se hvordan variablene på datasettet ser ut, gjerne straks vi har laget et datasett. Ved å granske variablene til 10-15 observasjoner, vil vi raskt se om variablene er lest riktig inn, om variabelverdiene ser rimelige ut. Hvis for eksempel variabelen for kjønn har andre verdier enn 1 og 2, bør alarmen lyse.
Hvis vi bruker PROC PRINT uten å gjøre noen restriksjoner på hvilke observasjoner som skal skrives ut, blir *hele* datasettet skrevet ut til SASlisten. For å begrense antallet observasjoner som skrives ut kan vi bruke datasett-option OBS= og spesifisere et passende antall.
PROC PRINT sammen med WHERE-setningen er spesielt nyttig når vi vil granske *bestemte* observasjoner på datasettet. I eksempelet under, vil vi skrive ut de observasjoner som har negativ omsetning:

```
PROC PRINT DATA=hotell.hele95 N NOOBS ;
   WHERE . < omsetn < 0 ;
   TITLE 'Disse har negativ omsetning. Hva har skjedd?' ;
RUN;
```

Legg merke til betingelsen i WHERE-setningen: `WHERE . < omsetn < 0 ;` Dersom vi hadde skrevet `WHERE omsetn < 0 ;` ville vi fått skrevet ut alle observasjoner som *manglet* omsetning i tillegg til de med negativ omsetning. Dette fordi manglende verdi (missing) regnes som det aller, aller minste, mindre enn noe annet og ihvertfall mindre enn 0. Ved å skrive betingelsen som et intervall med manglende verdi på venstre siden av mindre-enn-tegnet, slik: `. < omsetn < 0` får vi bare ut de observasjoner vi vil studere nærmere.
Legg også merke til option N i PROC PRINT-setningen: `PROC PRINT DATA=hotell.hele95 N NOOBS ;` Med option N får vi vite hvor mange observasjoner som oppfyller betingelsen og dermed blir skrevet ut. Option NOOBS gjør at vi ikke får med observasjonsnummeret på utskriften.

6.2 PROC MEANS

PROC MEANS kan benyttes til to helt forskjellige ting: Det ene er å *kvalitetssjekke* og få *oversikt over verdiene* på et datasett ved hjelp av enkel statistikk. Statistikken kan være gjennomsnitt, maksimums- og minimumsverdier, standard avvik, antall som mangler verdi osv. Resultatene, statistikken, får vi da ut i en *tabell* som vi eventuelt kan skrive ut på papir og granske.

Det andre PROC MEANS kan brukes til, er å beregne statistikk på et *høyere aggregeringsnivå* av dataene. Hvis datasettet for eksempel er på kommunenivå, kan statistikken beregnes på fylkesnivå. Er datasettet på personnivå, kan statistikken beregnes pr menn og pr kvinner. Statistikken kan være sum, gjennomsnitt, maksimums- og minimumsverdier osv. osv. Når vi bruker PROC MEANS til aggregering, får vi resultatene, statistikken, ut på et *datasett*. Vi kaller gjerne et slikt datasett et aggregert datasett.
PROC MEANS kan altså brukes på to helt forskjellige måter:

- Kvalitetssjekke, få oversikt over et datamateriale ved hjelp av beskrivende statistikk
- Beregne statistikk på *aggregerte* data

Det er viktig å være klar over disse to forskjellige anvendelsene, for skrivemåten (syntaksen) til PROC MEANS er helt forskjellig i de to.

PROC MEANS beregner statistikk kun på *numeriske* variable!

Det gir ikke mening å beregne gjennomsnittlig kommunenummer eller fødselsnummer, så PROC MEANS beregner statistikk kun på numeriske variable.

6.2.1 PROC MEANS brukt til å kvalitetssjekke, få oversikt over et datamateriale

Det første vi gjør når vi har laget et nytt datasett, er å kjøre PROC MEANS på det. PROC MEANS gir oss nyttig informasjon om alle *numeriske* variable på datasettet og egner seg ypperlig til å finne eventuelt 'snusk' i datamaterialet. Informasjonen er i form av beskrivende statistikk som gjennomsnitts-, maksimums- og minimums-verdier, standard avvik, antall observasjoner som mangler verdi, m.m. Ved å studere resultatet fra PROC MEANS får vi et godt inntrykk av datamaterialet. Spesielt kan største og minste verdier for en variabel kan gi en pekepinn om eventuelle feil.

```
PROC MEANS DATA=perm.vareh92 MAXDEC=2 ;
   TITLE  'Ansatte i varehandelen 1992';
   TITLE2 'Enkel statistikk på numeriske variable. SJEKK!';
RUN;
```

I PROC MEANS kan vi spesifisere antall desimaler i utskriften med option MAXDEC= Dette gjør det lettere å lese tabellen. MAXDEC= er en option som hører til PROC MEANS, den gjelder ikke i andre proc'er. Slik blir resultatet av programeksempelet ovenfor:

```
Ansatte i varehandelen 1992
Enkel statistikk på numeriske variable. SJEKK!

Variable  Label            N          Mean       Std Dev       Minimum       Maximum
-------------------------------------------------------------------------------------
FAAR      Fødselsår       4901         53.69         12.56          8.00         90.00
MNDLONN   Mnd lønn        4901      13394.51       7095.71          0.00      62487.00
TILLEGG   Faste tillegg   4901         13.97        137.62          0.00       3135.00
BONUS     Bonus           4901       3632.79      22589.68          0.00     898900.00
-------------------------------------------------------------------------------------
```

Datamaterialet består av ansatte i varehandelen i 1992. Burde vi ikke da stusse over at maksimum fødselsår er 90? Dét betyr jo at det i datamaterialet er ansatte som enten er 2 år eller 102 år! Ved å granske filbeskrivelsen til dette datamaterialet, viste det seg at manglende fødselsår var kodet 90. Denslag er ikke helt uvanlig, så hvis vi finner 9999999 eller liknende som maksimumsverdi, må alarmklokkene ringe. Slik omkoding av manglende verdier er spesielt skummel, verdiene vil jo telle med i gjennomsnitt osv. osv. Vanligvis vil vi jo få hint om missing verdier ved at tallet i kolonnen for N varierer, men hvis missing er kodet om, vil kolonnen med N 'lyve'.

Hvis vi ikke spesifiserer hva slags statistikk vi vil ha beregnet, får vi antall, gjennomsnitt, minimum og maksimum som vist i eksempelet over. Ønsker vi mer, eller annen statistikk, må vi spesifisere dette i PROC MEANS-setningen. Legg merke til at vi i slike tilfelle må spesifisere *alt* vi vil ha, ikke bare det som kommer i tillegg til det som er standard:

```
PROC MEANS  DATA=perm.lonn92 MAXDEC=0 FW=8 N MEAN MAX MIN SUM ;
   TITLE 'Ansatte i varehandelen 1992 ' ;
RUN;
```

Med dette programmet vil resultatet bli slik:

```
Ansatte i varehandelen 1992

Variable  Label             N       Mean    Maximum    Minimum        Sum
-------------------------------------------------------------------------
FAAR      Fødselsår      4901         54         90          8     263138
MNDLONN   Mnd lønn       4901      13395      62487          0   65646489
TILLEGG   Faste tillegg  4901         14       3135          0      68444
BONUS     Bonus          4901       3633     898900          0   17804317
-------------------------------------------------------------------------
```

Her har vi ved hjelp av options i PROC MEANS-setningen fjernet alle desimaler med `MAXDEC=0` og avsatt 8 plasser til hver statistikk med `FW=8`

6.2.2 PROC MEANS brukt til å beregne statistikk på aggregerte data

Når vi bruker PROC MEANS til å få oversikt over et datamateriale slik som beskrevet ovenfor, blir resultatet en tabell. Bruker vi PROC MEANS til statistikk på *aggregerte* data blir resultatet ikke en tabell, men derimot et *datasett*. Det er viktig å merke seg disse to forskjellige anvendelsene av PROC MEANS, for skrivemåten i proc'en er helt forskjellig i de to tilfellene!

Når vi vil aggregere, spesifiserer vi *ikke* ønsket statistikk i selve PROC MEANS-setningen, men derimot i en egen OUTPUT-setning. I tillegg til OUTPUT-setningen, må vi også ha med en CLASS-setning der vi angir hvilken eller hvilke variable som skal være grupperings- eller klassevariable. Typiske klasssevariable er fylke, kommune, kjønn o.l. Vi må også oppgi hvilken eller hvilke variable vi vil ha beregnet statistikk på, disse lister vi opp i EN VAR-setning.

- Ønsket statistikk skrives *ikke* i PROC MEANS-setningen, men i en OUTPUT-setning.
- Grupperingvariable listes i en CLASS-setning.
- Variable som skal analyseres listes i en VAR-setning.
- I OUTPUT-setningen angis navn på aggregert datasett samt ønsket statistikk

OUTPUT-setningen kan bli omfangsrik: I denne angir vi hva det aggregerte resultat-datasettet skal hete, vi angir hva slags statistikk som skal beregnes og vi gir navn til de resultat-variablene som skal inneholde statistikken.

6.2.2.1 Aggregering med én klassevariabel

I eksempelet under har vi et datasett der hver observasjon representerer data fra en kommune. Variable angir kommunens utgifter til sosialhjelp samt antall personer som mottar sosialhjelp. På datasettet er det også en variabel som sier hvilket fylke kommunen ligger i. Vi ønsker å aggregere datasettet fra å være på kommunenivå, opp til fylkesnivå. Vi ønsker å få beregnet totalt antall sosialhjelpmottakere i hvert fylke, samlet utgift til sosialhjelp i hvert fylke, samt gjennomsnittlig sosialhjelpsutgift for kommunene innen fylket.

```
PROC MEANS DATA=soshjelp.kommuner NOPRINT ;
   CLASS fylke ;
   VAR antpers sosutg ;
   OUTPUT OUT=fylker
          SUM=totpers totutg
          MEAN(sosutg)=komsnitt
          ;
RUN;
```

Vi tar med NOPRINT-option i PROC MEANS-setningen, slik at vi bare får laget et datasett og ikke en utlisting i tillegg. I setningen `CLASS fylke ;` sier vi at vi vil aggregere dataene opp til fylkesnivå. Vi vil at det skal beregnes statistikk på variablene `antpers` og `sosutg` Dette sier vi med setningen `VAR antpers sosutg ;` Det skal lages et datasett med statistikk på fylkesnivå. Hva dette datasett skal hete og hva slags statistikk som skal beregnes, angir vi i OUTPUT-setningen: `OUTPUT OUT=fylker` betyr at det skal lages et temporært datasett som skal hete `fylker` Med `SUM=totpers totutg` sier vi at det skal beregnes SUM-statistikk på variablene `antpers` og `sosutg` som er variablene vi har listet opp i VAR-setningen. SUM-statistikken blir lagt ut som variable på det nye datasettet og får der navnene `totpers` og `totutg` Disse nye variabelnavnene velger vi fritt selv. Legg merke til at rekkefølgen på statistikk-variablene følger rekkefølgen på variablene i VAR-setningen! Fylkessummen av variabelen `antpers` er altså `totpers` mens fylkessummen av variabelen `sosutg` er `totutg` For å være på den sikre siden, kunne vi gjerne skrevet `SUM(antpers sosutg)=totpers totutg`
Siden vi ikke vil beregne gjennomsnitt av begge variablene, men bare av `sosutg` skriver vi `MEAN(sosutg)=komsnitt` Variabelen som inneholder gjennomsnittlig kommunal utgift til sosialhjelp innen fylket, blir hetende `komsnitt` Det er lurt å sette semikolonet på en egen linje, da er det mindre sjanse for feil hvis vi senere vil føye til flere linjer med statistikk.

Når programmet over er kjørt, kommer det ikke noe resultat i output-vinduet. Vil vi studere det aggregerte datasettet, kan vi skrive det ut:

```
PROC PRINT DATA=fylker NOOBS ;
   TITLE  'Sosialhjelp aggregert opp fra kommune- til fylkesnivå';
RUN;
```

Resultatet av denne proc'en kan se slik ut:

```
Sosialhjelp aggregert opp fra kommune- til fylkesnivå

FYLKE       _TYPE_   _FREQ_   TOTPERS     TOTUTG       KOMSNITT

             0       435      177657     3964214        9113.14
01           1        18       11510      251114       13950.78
02           1        22       14102      360295       16377.05
03           1         1       29914      956645      956645.00
04           1        22        7892      164153        7461.50
05           1        26        7701      162802        6261.62
06           1        21        8463      174036        8287.43
07           1        15        8374      182428       12161.87
08           1        18        7506      154944        8608.00
09           1        15        5143       84873        5658.20
10           1        15        5747      115474        7698.27
11           1        26       11146      258368        9937.23
12           1        34       16090      358609       10547.32
14           1        26        2520       40942        1574.69
15           1        38        7664      136747        3598.61
16           1        25       10102      196908        7876.32
17           1        24        4292       69432        2893.00
18           1        45        9919      146231        3249.58
19           1        25        5964      100474        4018.96
20           1        19        3608       49741        2617.95
```

Her er det flere ting å legge merke til: I vår PROC MEANS ba vi om at det skulle lages et datasett der statistikk-variablene skulle hete `totpers totutg` og `komsnitt` I utskriften ser vi at SAS i tillegg til variablene vi ba om, har laget noen flere, nemlig `_TYPE_` og `_FREQ_` Disse variablene er av stor nytte for oss: `_FREQ_` sier hvor mange observasjoner som ligger bak de statistiske beregningene, i vårt tilfelle antall kommuner som ligger bak beregningene (altså antall kommuner innen fylket). `_TYPE_` sier noe

om aggregeringsnivået. Vi ser at den første observasjonen har verdi 0 for _TYPE_ mens alle de andre observasjonene har verdi 1 Dette betyr at den første observasjonen er på et annet aggregeringsnivå enn de andre. Vi ser også at den første observasjonen ikke har noen verdi for fylke, den første observasjonen representerer nemlig hele landet! Statistikkene er altså her beregnet på landsbasis. Verdien for _FREQ_ angir totalt antall kommuner i landet, verdien for TOTPERS angir totalt antall personer som mottok økonomisk sosialhjelp, TOTUTG angir landets totale utgift til sosialhjelp, mens KOMSNITT angir gjennomsnittlig utgift til sosialhjelp pr kommune.

Dersom vi *ikke* vil ha med den første observasjonen med _TYPE_ lik 0 på datasettet, må vi ha med options NWAY i PROC MEANS-setningen. Option NWAY innebærer at vi *kun* får beregninger på laveste aggregeringsnivå.

6.2.2.2 Aggregering med flere klassevariable

I neste eksempel skal vi se hvordan vi kan ha med *flere* klassevariable. Vi ønsker å aggregere på fylke *og* urbanitet. Kommunenes urbanitet er gitt i en variabel på datasettet, den sier om kommunen er en bykommune eller en landkommune. Forøvrig er statistikkønsker og variabelnavn helt tilsvarende som i eksempelet ovenfor. (Dette eksempelet er *helt* konstruert, så ikke se på tallene. Det er prinsippene for programmeringen og hvordan resultatet blir, som er viktig.)

```
PROC MEANS DATA=soshjelp.kommuner NOPRINT ;
   CLASS fylke urbanhet ;
   VAR antpers sosutg ;
   OUTPUT OUT=fylker
          SUM=totpers totutg
          MEAN(sosutg)=komsnitt
          ;
RUN;
```

Hvis vi nå skriver ut det aggregerte datasettet, vil resultatet (i utdrag) kunne se slik ut:

```
Sosialhjelp aggregert opp fra kommune- til fylke*urbanitet (urbanitet: by/land)

FYLKE URBANHET _TYPE_   _FREQ_   TOTANT      TOTUTG    SNITTUTG

                  0       435     177657     3964214       9113.14
        0         1        47     105455     2703342      57517.91
        1         1       388      72202     1260872       3249.67
01                2        18      11510      251114      13950.78
02                2        22      14102      360295      16377.05
...               2       ...        ...         ...           ...
...               2       ...        ...         ...           ...
...               2       ...        ...         ...           ...
19                2        25       5964      100474       4018.96
20                2        19       3608       49741       2617.95
01        0       3         4       1489       50506      12626.50
01        1       3        14      10021      200608      14329.14
02        1       3        22      14102      360295      16377.05
03        0       3         1      29914      956645     956645.00
...               3       ...        ...         ...           ...
...               3       ...        ...         ...           ...
...               3       ...        ...         ...           ...
...               3       ...        ...         ...           ...
...               3       ...        ...         ...           ...
...               3       ...        ...         ...           ...
19        0       3         2       1004       10272       5136.60
19        1       3        23       4960       90202       3921.83
20        0       3         3        601        9701       3233.67
20        1       3        16       3007       40040       2502.50
```

Det første vi legger merke til, er at resultat-datasettet nå har mange flere observasjoner enn i forrige eksempel. Den første observasjonen, den med _TYPE_ lik 0 er lik i begge eksemplene. Den er beregnet på basis av *hele* inn-datasettet, altså aggregert over *alle* kommunene. I eksempelet med én klassevariabel, hadde vi kun observasjoner av _TYPE_ 0 og _TYPE_ 1 Når vi har *to* klassevariable, får

vi observasjoner med _TYPE_ 0 1 2 og 3 Vi ser at observasjonene med _TYPE_ 1 nå er observasjoner som er aggregert opp på urbanhet de har dermed ingen verdi for fylke. Observasjonene aggregert på fylke har _TYPE_ 2 Det er rekkefølgen på variablene i CLASS-setningen som bestemmer rekkefølgen i det aggregerte datasettet. Hadde vi derfor skrevet CLASS urbanhet fylke ; hadde fylkes-aggregatene hatt _TYPE_ 1 og urbanitets-aggregatene _TYPE_ 2
Observasjonene med _TYPE_ 3 er aggregert over alle kombinasjoner av klassevariablene. Det er én observasjon for Østfolds bykommuner, én observasjon for Østfolds landkommuner, én observasjon for Akershus landkommuner, (Akershus har ingen bykommuner, derfor ingen observasjon med denne kombinasjon), én observasjon for Oslo bykommune, osv osv.
Ved å benytte oss av variabelen _TYPE_ kan vi i senere steg velge ut nettopp dét aggregeringsnivået vi ønsker. Husk at ved å bruke option NWAY i PROC MEANS-setningen, får vi *kun* lagt observasjoner av laveste aggregeringsnivå på resultat-datasettet. Hvis vi hadde brukt NWAY i eksempelet over, ville vi kun fått ut observasjonene aggregert på kombinasjonene fylke-by fylke-land.

6.3 PROC FREQ

PROC FREQ *teller opp* antall observasjoner som har samme verdi for en gitt variabel. Dette kaller SAS frekvenstabeller, derav navnet PROC FREQ. Vi kan lage *enkle* frekvenstabeller, det er tabeller som viser frekvenser for én og én variabel, eller vi kan lage *krysstabeller* som viser frekvenser for to variable samtidig. Til PROC FREQ hører TABLES-setningen, der vi ramser opp den eller de variable det skal lages tabeller av.

6.3.1 PROC FREQ til enkle frekvenstabeller

La oss si at vi ønsker å finne ut hvor mange observasjoner som har den og den verdi for ekteskapelig status. Programmet kan da se slik ut:

```
PROC FREQ DATA=aku.aku_9206 ;
   TABLES ektstat;
   TITLE 'Frekvensfordeling på ekteskapelig status';
RUN;
```

Resultatet av dette programmet blir slik:

```
Frekvensfordeling på ekteskapelig status

                                 Cumulative  Cumulative
EKTSTAT   Frequency   Percent    Frequency    Percent
-----------------------------------------------------
1             593      29.8          593       29.8
2            1051      52.7         1644       82.5
3             153       7.7         1797       90.2
4             196       9.8         1993      100.0
```

Tabellen over er lite informativ for de som ikke kjenner kodene for ekteskapelig status. For å erstatte slike koder med forståelig tekst må vi bruke egen-definerte SAS-formater, slik som her:

```
PROC FREQ DATA=aku.aku_9206 ;
   TABLES ektstat;
   FORMAT ektstat $estat. ;
TITLE 'Frekvensfordeling på ekteskapelig status';
RUN;
```

Tabellen blir nå mer forståelig:

```
Frekvensfordeling på ekteskapelig status

                                 Cumulative  Cumulative
EKTSTAT   Frequency   Percent    Frequency    Percent
-----------------------------------------------------
Ugift         593      29.8          593       29.8
Gift         1051      52.7         1644       82.5
Samboer       153       7.7         1797       90.2
Skilt         196       9.8         1993      100.0
```

6.3.2 PROC FREQ til krysstabeller

Krysstabeller er tabeller som viser en variabel i tabellens forspalte og en annen variabel i tabell-hodet. Ønsker vi for eksempel en tabell som viser ekteskapelig status krysset med kjønn, kan programmet se slik ut:

```
PROC FREQ DATA=aku.aku_9206 ;
   TABLES ektstat * kjonn ;
   TITLE 'Frekvensfordeling på ekteskapelig status krysset med kjønn';
RUN;
```

Tabellen får nå både forspalte og hode, den blir slik:

```
Frekvensfordeling på ekteskapelig status krysset med kjønn

EKTSTAT
          KJONN
Frequency|
Percent  |
Row Pct  |
Col Pct  |1       |2       |  Total
---------+--------+--------+
1        |    329 |    264 |    593
         |  16.51 |  13.25 |  29.75
         |  55.48 |  44.52 |
         |  33.27 |  26.29 |
---------+--------+--------+
2        |    526 |    525 |   1051
         |  26.39 |  26.34 |  52.73
         |  50.05 |  49.95 |
         |  53.19 |  52.29 |
---------+--------+--------+
3        |     74 |     79 |    153
         |   3.71 |   3.96 |   7.68
         |  48.37 |  51.63 |
         |   7.48 |   7.87 |
---------+--------+--------+
4        |     60 |    136 |    196
         |   3.01 |   6.82 |   9.83
         |  30.61 |  69.39 |
         |   6.07 |  13.55 |
---------+--------+--------+
Total         989     1004     1993
             49.62    50.38   100.00
```

Hver celle i krysstabellen inneholder fire tall, de er forklart i øverste venstre hjørne av tabellen. Det øverste tallet er antallet observasjoner med denne kombinasjonen verdier, *frekvensen*. Det nest øverste tallet er hvor mange prosent av *totalprosenten* denne cellen utgjør. Det nest nederste tallet er *linjeprosenten*, disse summerer seg til 100 prosent bortover langs linjen, '*row*', mens det nederste tallet er *kolonneprosenten* som summerer seg til 100 prosent nedover langs kolonnen, '*column*'. Vi kan selv si ifra hvis vi ikke ønsker å ha med alle de fire tallene i tabellen. Options til TABLES-setningen i PROC FREQ gir oss mulighet til å ta vekk frekvensen (NOFREQ), totalprosenten (NOPERCENT), linjeprosenten (NOROW) og kolonneprosenten (NOCOL). Vi kan bruke én eller flere av disse optionene.

Tabellen i eksemplene over er lite informativ for utenforstående. Er det opplagt at koden '1' betyr 'Menn' og koden '2' betyr 'Kvinner'? For å gjøre tabellen mer forståelig, må vi bruke SAS-formater. I det neste eksempelet benytter vi formater vi tidligere har definert. Vi velger også bort linje- og kolonne-prosentene:

```
PROC FREQ DATA=aku.aku_9206 ;
   TABLES ektstat * kjonn / NOCOL NOROW ;
   FORMAT ektstat $estat.
          kjonn   $kjonn.
          ;
   TITLE 'Frekvensfordeling på ekteskapelig status krysset med kjønn';
RUN;
```

Tabellen blir nå mer forståelig:

```
Frekvensfordeling på ekteskapelig status krysset med kjønn

EKTSTAT
          KJONN
Frequency|
Percent  |Menn     |Kvinner |  Total
---------+--------+--------+
Ugift    |    329 |    264 |    593
         |  16.51 |  13.25 |  29.75
---------+--------+--------+
Gift     |    526 |    525 |   1051
         |  26.39 |  26.34 |  52.73
---------+--------+--------+
Samboer  |     74 |     79 |    153
         |   3.71 |   3.96 |   7.68
---------+--------+--------+
Skilt    |     60 |    136 |    196
         |   3.01 |   6.82 |   9.83
---------+--------+--------+
Total         989     1004     1993
            49.62    50.38   100.00
```

6.3.3 Fallgruver i PROC FREQ

Det er to ting vi må huske på når vi bruker PROC FREQ: Det ene er å ha med TABLES-setningen, det andre er å tenke nøye igjennom hva hvilke variable vi vil bruke PROC FREQ på. Hvis vi glemmer TABLES-setningen, lager PROC FREQ tabeller av *absolutt alle* variablene på datasettet! Hvis vi ber PROC FREQ lage tabell av en variabel som for eksempel omsetning eller inntekt, får vi en linje for *hver eneste verdi* den variabelen har.

- Ikke glem å ha med TABLES-setningen
- Unngå variable med veldig mange verdier (omsetning, inntekt, fødselsnr o.l.)

PROC FREQ teller altså opp hvor mange observasjoner som har samme verdi for en bestemt variabel. Det blir en linje i tabellen for hver eneste verdi variabelen har. Variable som er typiske å bruke i PROC FREQ er fylke, kommune, kjønn, ekteskapelig status, stillingskode og andre variable som har noen få, distinkte verdier. Det er altså ikke særlig lurt å ta på PROC FREQ på variable som for eksempel omsetning, inntekt, fødselsnummer eller andre variable som kan ha forferdelig mange verdier. Tabellen kan da fort få en dramatisk lengde! (Men husk at det går an å *gruppere* variabelverdiene ved hjelp av PROC FORMAT og FORMAT-setningen. På den måten kan vi 'formattere' variabelverdiene til noen få grupper.)

6.3.4 PROC FREQ til å finne dubletter

Det kan ofte være nyttig og noen ganger helt nødvendig, å vite om et datasett inneholder *dubletter*, observasjoner som har samme identifikasjonsnøkkel. Identifikasjonsnøkkel er en variabel (eller en kombinasjon av flere variable) med entydig verdi, som fødselsnummer e.l. Dersom vi har et datasett som ikke er altfor stort (maksimum ca 30 000 observasjoner) kan vi bruke PROC FREQ til å finne ut om vi har dubletter på datasettet.

I eksempelet under, ønsker vi å finne ut om datasettet inneholder flere observasjoner med samme fødselsnummer. Men kan vi bruke PROC FREQ til dette? Har vi ikke nettopp sagt at vi må unngå variable som fødselnummer i PROC FREQ? Det som er poenget når vi bruker PROC FREQ til å finne dubletter, er at vi ber PROC FREQ om *ikke* å lage en tabell, men derimot et *datasett* som inneholder frekvenstabellen. Å skrive en tabell på 30 000 linjer ut til output-vinduet er ikke så lurt, men å lage et datasett på 30 000 observasjoner er helt kurant. Datasettet som er resultatet fra PROC FREQ inneholder en variabel som heter COUNT Det er denne vi benytter oss av, for å finne frem til de eventuelle dublettene:

```
PROC FREQ DATA=fy88.personer ;
   TABLES fnr / NOPRINT OUT=frekv_ds ;

PROC PRINT DATA=frekv_ds NOOBS;
   WHERE COUNT > 1 ;
   TITLE 'Disse er DUBLETTER ! Må sjekkes!';

RUN;
```

Det er option'ene `/NOPRINT OUT=` i TABLES-setningen i PROC FREQ som fører til at ikke blir laget noen tabell i output-vinduet, men istedet et datasett. Navnet på datasettet bestemmer vi selv, når vi skriver `OUT=frekv_ds` Det nye datasettet inneholder variablene COUNT og PERCENT i tillegg til fødselsnummeret.

6.4 PROC FORMAT

PROC FORMAT er en proc som ikke bearbeider noe SAS datasett, dens funksjon er å *definere* formater. Formatene som er definert med en PROC FORMAT, *brukes* så i en senere proc, ved hjelp av en FORMAT-setning.
Det kan defineres forskjellige typer format: De som brukes til å erstatte variabelverdier i form av *koder* med *forståelig tekst* (som i eksempelet vist med PROC FREQ, der kodene 1 og 2 erstattes av tekstene Menn og Kvinner) og de som *grupperer* verdier (som av en variabel med mange verdier lager verdi*grupper*). Det finnes også en type format som brukes til å skrive tall ut på en bestemt måte, for eksempel vil vi i SSB gjerne ha tall skrevet ut med *tusenskille* og desimal*komma.*
Anvendelsen av PROC FORMAT er beskrevet i kapittel 3.3.3

6.5 PROC CONTENTS

Et SAS datasett inneholder data*verdier* såvel som *informasjon om* dataene (også kalt *meta*data). Informasjonen om dataene kan vi se på som en slags filbeskrivelse til SAS datasettet. Den inneholder informasjon om hva variablene heter, hva slags type de er, hva variabel-lablene er, om datasettet er sortert, hvor i filstrukturen datasettet er lagret, m.m. Denne informasjonen får vi skrevet ut ved hjelp av PROC CONTENTS, se eksempelet sist i forrige hovedkapittel. (Tar vi PROC CONTENTS på et SAS-view av en Oracle-tabell, vil utskriften inneholde noen flere opplysninger enn det som vises i dette eksempelet.)

6.6 PROC SORT

Når vi skal sortere et SAS datasett, bruker vi PROC SORT. PROC SORT brukes *alltid* sammen med en BY-setning. I BY-setningen ramser vi opp den eller de variablene vi vil at datasett skal sorteres etter. SAS sorterer stigende (med de laveste verdier først) med mindre vi sier at vi vil ha avtagende rekkefølge. Isåfall bruker vi option'en DESCENDING foran den aktuelle variabelen i BY-setningen.

```
PROC SORT DATA=hotell.oktober ;
   BY kommune DESCENDING omsetn ;

RUN;
```

I eksempelet over, vil datasettet `hotell.oktober` når programmet er kjørt, være sortert etter kommune, med kommune 0101 først. Av alle observasjonene fra kommune 0101, vil den med høyest omsetning ligge først.
I visse situasjoner kan det være ønskelig å lage et *nytt* datasett som er sortert, mens det originale datasettet forblir uendret. Til dette formål har vi option'en OUT= som brukes i selve PROC SORT-setningen:

```
PROC SORT DATA=aku.original  OUT=privat ;
   BY astatus tkode kjonn DESCENDING alder ;

RUN;
```

I eksempelet over, lager vi et *temporært* datasett som er sortert, mens original-datasettet forblir uendret.

Husk at når vi bruker BY-setningen i DATA-steg eller andre proc'er, *krever* SAS at datasettet er sortert.

6.7 PROC TABULATE

PROC TABULATE er den proc som er best egnet til å lage tabeller. Den er utfyllende beskrevet i «SAS som tabellverktøy» (Interne dokumenter) og **Håndbok i SAS Del 2: Oppslag** (SSH 71).

6.8 PROC UNIVARIATE

PROC UNIVARIATE likner endel på PROC MEANS, den beregner statistikk på numeriske variable. Men UNIVARIATE produserer *mye* mer papir enn MEANS. Derfor er det viktig å ha med en VAR-setning når vi bruker UNIVARIATE. I VAR-setningen ramser vi opp den eller de variablene vi vil ha beregnet statistikk på.
Noe av det som er fint med PROC UNIVARIATE, er at den skriver ut de *fem* observasjonene som har høyest og lavest verdi for en variabel. Den kan dessuten brukes til å få ut *percentiler*.

7. SAS macro-språk

SAS har et eget programmeringsspråk som ligger 'utenpå' eller 'rundt' det vanlige SAS språket. Dette heter **SAS macro-språk**. Macro-språket kan forenkle programmeringen og gjøre program mer fleksible. Macro-språket kan også brukes til å 'styre' utførelsen av DATA- og PROC-steg. Macro-språket er svært anvendelig, ikke desto mindre er det et minefelt! Brukt på en enkel måte er SAS macro-språk gull verdt, mens kompliserte macroer kan gjøre program direkte ubrukelige.

Unngå kompliserte macroer!

SAS macro-språk består av egne **macro-setninger** som kjennetegnes ved at de starter med et **%**-tegn. De fleste macro-setninger har sin makker i de vanlige SAS-setningene, som %IF - %THEN og %DO %END . Ved hjelp av macro-setninger og/eller vanlige SAS setninger kan vi bygge opp **SAS macroer**. En SAS macro har et navn som den refereres ved. Når en macro refereres, settes en % foran navnet.

Macro språket innbefatter også **macro-variable**. Når en macro-variabel refereres, starter navnet med et **&**-tegn. En macro-variabel har én verdi, i motsetning til en datasett-variabel som har et sett verdier, én for hver observasjon på datasettet.

- Macroer består av macro-setninger og/eller vanlige SAS setninger
- En macro har et navn som den refereres ved
- Macroer refereres med en % foran navnet, men slutter *ikke* med ;
- Macro-setninger starter alltid med **%** og slutter med **;**
- Macro-variable refereres alltid med en **&** foran navnet

Macro-språket kan benyttes på to forskjellige måter: Vi kan bruke macro-variable alene, eller vi kan bygge SAS macroer som vi først deklarerer (definerer) og deretter kaller opp. Det enkleste er å bruke macro-variable alene. Selv om det er enkelt, kan det være svært nyttig og arbeidsbesparende (se eksempelet under). Hvis vi vil bygge SAS macroer, kan vi bygge dem enkle eller med parametre.

- Enten: *Kun* bruke macro-variable
- Eller: Bygge macroer:
  - enkle
  - parameterstyrt

Hvis vi vil lage en parameterstyrt macro, kan det ofte være lurt å lage den enkel først, og heller utvide med parametre etterpå.

7.1 Macro-setninger

Macro-setningene kjennetegnes ved at de starter med et %-tegn. Macro-setninger kan stå *utenfor* stegene, i motsetning til de fleste vanlige SAS setningene som står innenfor et DATA- eller PROC-steg. I likhet med de vanlige SAS setningene avsluttes macro-setningene med semikolon. Her følger de vanligste macro-setningene:

- **%MACRO** starte deklareringen av en macro, gi den navn
- **%MEND** avslutte deklareringen av en macro
- **%LET** opprette en macro-variabel og gi den verdi
- **%DO %WHILE** starte løkke, tester betingelsen *før* løkken
- **%DO %UNTIL** starte løkke, tester betingelsen *etter* løkken
- **%DO ... %TO** starte løkke
- **%END** avslutte løkke
- **%IF ... %THEN ... %ELSE** betinget utførelse
- ***%navn*** kalle opp en macro

Når en macro skal *defineres*, starter vi med en %MACRO-setning: `%MACRO imputere;` og avslutter med en %MEND-setning: `%MEND;` Mellom disse to setningene kommer så alle de andre macro- og vanlige setningene som utgjør macroen. I dette tilfellet er macroens navn `imputere` Når macroen senere skal utføres, kalles den med en % foran navnet, slik: `%imputere` Legg merke til at det *ikke* skal stå semikolon etter macrokallet! I noen situasjoner vil det ikke spille noen rolle om det står et semikolon der, mens i andre situasjoner vil det bli direkte feil. (Hvis macroen skal utgjøre *en del* av en setning, vil et semikolon etter kallet *avslutte* setningen, utilsiktet.)

7.2 Bruk av macro-variable alene

7.2.1 Automatisk årstall i overskrifter

Mange statistikker kommer ut en gang i året. Dersom rutinen består av mange tabeller, kan det være en kjedelig jobb å endre årstallet i overskriften manuelt i alle tabellprogrammene. Ved å legge årstallet i en macro-variabel, holder det å endre ett eneste sted.

```
***-------------------------------------------------------------------***;
* JAKTSTATISTIKKEN, TABELLER                                           *;
*                                                                      *;
*  Program: tabeller.sas                                               *;
*  Skrevet: 8. august 1996  LDa                                        *;
*  Endret:                                                             *;
*                                                                      *;
***-------------------------------------------------------------------***;

%LET aargang = 1995 ;

PROC TABULATE DATA=jakt.rein FC=' ----------' MISSING NOSEPS F=ssb7_0v. ;
   CLASS omrade kommune ;
   VAR   .......... ;
   FORMAT ...
          ;
   TABLE .....   ,
         .....
          ;
   TITLE "Villreinjakt &aargang.. Tillat felt og felte dyr pr område og kommune";

PROC TABULATE DATA=jakt.hjort FC=' ----------' MISSING NOSEPS F=ssb7_0v. ;
   CLASS omrade kommune ;
   VAR   .......... ;
   FORMAT ...
          ;
   TABLE .....   ,
         .....
          ;
   TITLE "Hjortejakt &aargang.. Tillat felt og felte dyr pr område og kommune";

PROC TABULATE DATA=jakt.elg FC=' ----------' MISSING NOSEPS F=ssb7_0v. ;
   CLASS omrade kommune ;
   VAR   .......... ;
   FORMAT ...
          ;
   TABLE .....   ,
         .....
          ;
   TITLE "Elgjakt &aargang.. Tillat felt og felte dyr pr område og kommune";

PROC TABULATE DATA=jakt.rein FC=' ----------' MISSING NOSEPS F=ssb7_0v.;
   CLASS fylke kommune ;
   VAR   ............;
   FORMAT ...
          ;
   TABLE .....   ,
         .....
          ;
   TITLE "Villreinjakt &aargang.. Tillat felt og felte dyr pr fylke og kommune";

PROC TABULATE DATA=jakt.hjort FC=' ----------' MISSING NOSEPS F=ssb7_0v.;
   CLASS fylke kommune ;
   VAR   ............;
   FORMAT ...
          ;
   TABLE .....   ,
         .....
          ;
   TITLE "Hjortejakt &aargang.. Tillat felt og felte dyr pr fylke og kommune";

PROC TABULATE DATA=jakt.elg FC=' ----------' MISSING NOSEPS F=ssb7_0v.;
   CLASS fylke kommune ;
   VAR   ............;
   FORMAT ...
          ;
   TABLE .....   ,
         .....
          ;
   TITLE "Elgjakt &aargang.. Tillat felt og felte dyr pr fylke og kommune";
RUN;
```

I programeksempelet over er det en lang rekke tabeller, hver med sin overskrift i en TITLE-setning. I alle overskriftene skal det stå årstall. Med macro-setningen `%LET aargang = 1995;` helt først i programmet oppretter vi en macro-variabel `aargang` som vi gir verdien `1995` Vi ser at `%LET` er en macro-setning ved at den starter med `%` Macro-setninger kan stå utenfor de vanlige program-stegene, derfor kan `%LET` stå aller først i programmet, før proc'ene. Forskjellen mellom en datasett-variabel og en macro-variabel er at en datasett-variabel har *et sett verdier*, én verdi for hver observasjon på datasettet, mens en macro-variabel har *én verdi*, som den beholder inntil den eksplisitt gis en ny verdi.

- En **datasett-variabel** har ***et sett verdier***
- En **macro-variabel** har ***én verdi***

Istedenfor å skrive årstallet eksplisitt i alle TITLE-setningene, refererer vi isteden til macro-variabelen `TITLE "Elgjakt &aargang.. Tillat felt og felte dyr pr fylke og kommune";` Når vi refererer til en macro-variabel, må vi ha med `&` foran navnet og `.` etter, slik: `&aargang.` Årsaken til at vi skriver `..` altså `&aargang..` er at det skal stå `Elgjakt 1995.` dvs et punktum etter årstallet i overskriften. Legg merke til at TITLE-setningen må stå i ekte gåseøyne (doble fnutter), altså slike: `"` Dette er helt nødvendig for at macro-variabelen skal oppløses til årstallet.

7.2.2 Automatisk seleksjon av data

Nå skal vi se på hvordan vi kan utvide bruken av macro-variabelen. Hvis vi skal bruke én årgang av et datasett der mange årganger ligger samlet, kan vi benytte macro-variabelen til å selektere ut passende årgang:

```
%LET aargang = 1995 ;

PROC TABULATE DATA=jakt.rein FC=' -----------' MISSING NOSEPS F=ssb7_0v. ;
   WHERE aar = "&aargang." ;
   ... ;
   ... ;
   TITLE "Villreinjakt &aargang.. Tillat felt og felte dyr pr område og kommune";
```

Ved å bruke WHERE-setningen der vi sammenlikner verdien av datasett-variabelen `aar` med verdien av macro-variabelen `&aargang.` er det tilstrekkelig å endre årstallet ett eneste sted i programmet for hvert nytt år.

I eksempelet er datasett-variabelen `aar` av type karakter. Dette er årsaken til at vi skriver `"&aargang."` altså doble fnutter rundt macro-variabelen i WHERE-setningen. Dersom datasett-variabelen hadde vært numerisk, skulle WHERE-setningen sett slik ut: `WHERE aar = &aargang. ;` altså uten fnutter.

7.3 SAS macroer

Vi kan bruke SAS macro for å spare skriving og for å få oversikt over store program, når samme setning eller setninger skal gjentas flere ganger. Programmet som skal gjentas, legges i en macro. Hver gang denne programbiten skal gjentas, kaller vi isteden opp macroen.
En SAS macro består som oftest av vanlige SAS setninger, eventuelt ispedd macro-setninger. Macroen kan være et komplett SAS-program, eller bare en del av et program. En macro kan være noe så lite som *en del* av en enkelt setning!
En SAS macro må først defineres og gis et navn, deretter kalles den ved å skrive macroens navn med en % foran. Når en macro skal defineres, starter vi med setningen **%MACRO macronavn ;** og avslutter definisjonen med **%MEND ;**

Mens setningene i et vanlig SAS program utgjør DATA- eller PROC-steg, kan macro-setningene stå *utenfor* stegene (som %LET-setningen i eksempelet over).

7.3.1 Macro uten parametre

SAS macroer kan brukes for å spare skriving. Hvis det samme (lange) programmet skal utføres mange ganger, men for eksempel på forskjellige datasett, kan vi legge programmet i en macro.

Et eksempel:

Vi har 10 SAS datasett, måneds-datasett, som vi vil skrive ut deler av. Datasettene har nøyaktig samme variable, de inneholder samme slags data, men fra forskjellige måneder. Vi vil bare skrive ut visse observasjoner fra datasettene og bruker derfor en WHERE-setning for å angi betingelsene. WHERE-setningen er lang og innviklet. Det er veldig mange variable på datasettet. Ikke alle variablene skal være med på utskriften, så vi bruker en VAR-setning (også den veldig lang) for å spesifisere hvilke variable som skal være med.
Ved hjelp av en SAS macro kan vi slippe å skrive WHERE- og VAR-setningene om igjen i alle 10 proc'ene. Vi legger istedet disse setningene i en macro som vi så kaller opp 10 ganger:

```
%MACRO stmts ;
    WHERE .....................................................
          .....................................................
          .....................................................
          .....................................................
          ;
    VAR   .....................................................
          .....................................................
          .....................................................
          .....................................................
          ;
%MEND;

PROC PRINT DATA=hotell.januar;
   %stmts
   TITLE "Utskrift av datasettet for januar. " ;

PROC PRINT DATA=hotell.februar;
   %stmts
   TITLE "Utskrift av datasettet for februar. " ;

PROC PRINT DATA=hotell.mars;
   %stmts
   TITLE "Utskrift av datasettet for mars." ;
.....
.....
.....
.....

PROC PRINT DATA=hotell.oktober;
   %stmts
   TITLE "Utskrift av datasettet for oktober." ;
RUN;
```

Foruten at vi sparer skriving, vil det her være mye enklere å *endre* både betingelsene for utskrift og hvilke variable som skal skrives ut. Endringene gjøres bare *ett* sted, en klar fordel. (Hvem har ikke opplevd at betingelsene i WHERE-setningen måtte endres, eller at utskriften inneholdt variable en likevel ikke ville ha med?)

7.3.2 Macro med parametre

Oppgaver som den i eksempelet over kan forenkles ytterligere hvis vi benytter oss av en macro med parametre. Parametrene er macro-variable som skrives i parentes i %MACRO-setningen.

Hvis vi skal benytte macro med parameter i eksempelet ovenfor, vil det der være naturlig å la navnet på datasettet være en macro-variabel, *parameteren*. Isåfall legger vi hele proc'en inn i macroen, og ved hjelp av macro-variabelen (parameteren) styrer vi hvilket datasett som skal skrives ut:

```
%MACRO print(maaned) ;
   PROC PRINT DATA=hotell.&maaned. ;
      WHERE ..........................................
            ..........................................
            ..........................................
            ..........................................
          ;
      VAR   ..........................................
            ..........................................
            ..........................................
            ..........................................
            ;
      TITLE "Utskrift av datasettet for &maaned. " ;
   RUN;
%MEND;

%print(januar)

%print(februar)

%print(mars)
.....
.....
.....

%print(oktober)
```

Hadde datasettene våre hatt navn som var nummerert, altså hotell.mnd1, hotell.mnd2 osv, kunne vi gjort det hele ved hjelp av en macro-løkke:

```
%MACRO loop ;
   %DO nr=1 %TO 10 ;
      PROC PRINT DATA=hotell.mnd&nr. ;
         WHERE ..........................................
               ..........................................
               ..........................................
               ..........................................
             ;
         VAR   ..........................................
               ..........................................
               ..........................................
               ..........................................
               ;
         TITLE "Utskrift av datasettet for måned: &nr. " ;
      RUN;
   %END;
%MEND;

%loop
```

En ulempe her, er at TITLE-setningen ikke gir så forståelig overskrift når en bruker månedsnummer som når en bruker månedsnavn i klartekst (vi kan uansett ikke ha datasett-navn som *september*, så...).

TITLE-setningen i eksempelet over, gir ikke pen overskrift. Heldigvis kan vi gjøre noe med dette! Riktignok må vi involvere et DATA-steg og litt avansert programmering, men dette er nyttige håndgrep også i andre sammenheng, så det er vel verd å studere litt nærmere:

For å få månedsnavnene pent ut i overskriftene, kan vi benytte oss av egendefinerte formater, PUT-funksjonen og CALL SYMPUT (grenseflaten mellom DATA-steget og SAS macro, forklares i neste avsnitt). Hvis vi har definert et format for månedsnavnene, kan vi i et DATA-steg lage en

hjelpevariabel som vi ved hjelp av PUT-funksjonen 'putter' ut i dette formatet. (SAS funksjoner kan kun brukes i DATA-steg, det er derfor vi må gjøre dette i et eget DATA-steg foran proc'en.) Ved deretter å benytte CALL SYMPUT som legger verdien av en datasett-variabel over i en macro-variabel, får vi laget en macrovariabel som inneholder månedsnavn. Denne macrovariabelen benyttes så i TITLE -setningen:

```
PROC FORMAT ;
   VALUE $mndfmt
       '1'='januar'
       '2'='februar'
       '3'='mars'
       ...
        ;

%MACRO loop ;
   %DO nr=1 %TO 5 ;
       DATA hjelp;
          mndnr   = "&nr." ;
          mndtext = put(mndnr,$mndfmt. ) ;
          call symput('maaned',mndtext) ;

       PROC PRINT DATA=mnd&nr. ;
          WHERE ................ ;
          VAR   ................  ;
       TITLE "Utskrift av datasettet for &maaned. " ;
       RUN;
   %END ;
%MEND;

%loop
```

7.4 Fra datasett-variabel til macro-variabel og omvendt

Det vil ofte være behov for å legge verdien av en datasett-variabel over i en macrovariabel eller omvendt. Til disse formål brukes macro-rutinen **CALL SYMPUT** og macro-funksjonen **SYMGET**.

- **CALL SYMPUT(**'macro-var',datasett-var) datasett-variabel til macro-variabel
- **SYMGET(**'macro-var'**)** macro-variabel til datasett-variabel

For å legge verdien av en datasett-variabel over i en macrovariabel brukes CALL SYMPUT-rutinen. Bemerk at navnet på macro-variabelen skal stå i fnutter, slik:

```
CALL SYMPUT ('macr_var' , dats_var) ;
```

For å legge verdien av en macro-variabel over i en datasett-variabel brukes SYMGET-funksjonen, som oftest i en tilordnings-setning slik:

```
dats_var = SYMGET ('macr_var') ;
```

Bemerk at navnet på macro-variabelen skal stå i fnutter.

7.5 Macro system-variable

Det finnes spesielle macro-variable som SAS selv har definert, disse kalles også **macro system-variable**. Blant de nyttigste er variabelen SYSPARM. Denne kan brukes til å bringe en *parameter* (for eksempel et passord) med inn i et program. SYSPARM kan med fordel brukes når vi skal opprette en forbindelse til Oracle, for å slippe å skrive Oracle-passordet i SAS-programmet:

```
***------------------------------------------------------------------***;
* DØDSÅRSAKSSTATISTIKKEN                                              *;
*                                                                     *;
*  Skal selektere noen variable fra Europeisk kortliste               *;
*  Oracle tabellen heter europeisk_kortliste og ligger på server opr3 *;
*                                                                     *;
*  Program: doduttak.sas                                              *;
*                                    4. januar 2001 POL               *;
***------------------------------------------------------------------***;

LIBNAME dodora ORACLE USER=pol PATH="@OPR3" PASSWORD=&sysparm SCHEMA=dodreg;

 PROC SQL;
  SELECT icd_kode, euro_gruppe, gyldig_fra_aar, gyldig_til_aar, euro_kode
  FROM dodora.europeisk_kortliste;
 QUIT;
```

LIBNAME-setningen brukes for å opprette en forbindelse mellom SAS og Oracle. For å bestemme hvilken Oracle-tabell SAS skal lese, bruker vi PROC SQL med en SELECT-setning. SAS-programmet må inneholde Oracle-brukernavn og -passord. Det er for å unngå å skrive passordet i programmet SYSPARM kommer inn i bildet. Ved å skrive `PASSWORD=&sysparm` som i eksempelet, kan vi kjøre programmet i batch og oppgi passordet som parameter slik:

```
/ssb/ursus/a1/dodarsak/prog> sas  doduttak.sas  -sysparm  Oracle-passordet
```

Når vi kjører et SAS-program i batch, har vi ikke SAS-vinduene oppe. Vi står på unix-promptet, skriver kommandoen `sas` etterfulgt av navnet på programmet som skal utføres, her `doduttak.sas` Etter programnavnet skrives eventuelle parametre, som her `-sysparm` etterfulgt av Oracle-passordet.

8. Stikkordregister

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

De sist utgitte publikasjonene i serien Statistisk sentralbyrås håndbøker

45 Håndbok i datasikkerhet og fysisk sikring. Revidert utgave, november 1998. 1998. 83s.

46 Telefonkatalog. 1998. 89s.

47 EØS-avtalen. Det statistiske samarbeid og konsekvenser for Statistisk sentralbyrås statistikkproduksjon. 1994. 55s.

48 Håndbok i tilsettingssaker. 1994. 32s.

49 Oppgaveplikt og tvangsmulkt. 1995. 55s.

50 Emneinndeling 1995. 1995. 43s.

51 Intervju: EDB-arbeidsbok. 1995.

52 Intervju: EDB-oppslagsbok. 1995.

53 Intervju: Opplæring og administrasjon. 1995.

54 Internkontroll: Revidert utgave 1997. 25s.

55 Nordisk statistikk på CD-ROM: Veiledning. 20s.

56 PC-Axis versjon 2.2: Brukerhåndbok. 69s.

57 Produktregister versjon 4.0: Brukerveiledning. 49s.

58 Håndbok i prosjektstyring. 20s.

59 Personalreglement for Statistisk sentralbyrå. 22s.

60 Produktnummerkatalog pr. 28.02.1996. 55s.

61 Innkjøpshåndbok. 1996.

62 Timeplan versjon 3.0: Brukerveiledning. 16s.

63 Håndbok i EDB-metode. 52s.

64 Publiseringshåndbok: Regler og retningslinjer for publisering i Statistisk sentralbyrå. 93s.

65 Håndbok i utvikling av statistikksystemer: Med vekt på IT-metode. 52s.

66 Håndbok i datarevisjon. 48s.

67 Arkivnøkkel for Statistisk sentralbyrå. 76s.

68 Rapporteringshåndbok for KOSTRA-regnskap 1999: Oppslagshefte til hjelp ved filuttrekk for KOSTRA-rapportering. 52s.

69 Yrkeskatalog for innrapportering av yrke til arbeidstakerregisteret. 86s.

70 Håndbok for KOSTRA-rapportering 2000. Oppslagshefte til hjelp ved filuttrekk for KOSTRA-rapportering. 74s.

B *Returadresse:*
Statistisk sentralbyrå
N-2225 Kongsvinger

Statistisk sentralbyrå

Oslo:
Postboks 8131 Dep.
0033 Oslo

Telefon: 22 86 45 00
Telefaks: 22 86 49 73

Kongsvinger:
2225 Kongsvinger

Telefon: 62 88 50 00
Telefaks: 62 88 50 30

Statistisk sentralbyrå
Statistics Norway