



The name "a" or "b" is arbitrary, but can be used later in your SAS program to identify that particular data set.

Data values should have spaces between them. If data are missing, a period (.) should be used as a place holder.

DATA TRANSFORMATIONS are usually done in a separate data step.

```
data logs;          /* create data set "logs" */
  set a;           /* using existing set "a" */
  logy = log(y);   /* creates new variable "logy" as the natural log of "y" */
  sqy = sqrt(y);  /* square root */
  z = x*y;        /* multiplication */
  y2 = y**2;      /* exponent: "y squared" or "y to the 2nd power" */
```

Data sets may also be manipulated by restricting the observations they contain or by recoding numbers.

```
data b;            /* create "other" from "big" */
  set a;
  if x > 10;       /* only use the obs. where x is greater than 10 */
  if trt = 'control' then delete; /* delete control group */
  if z < 10 then g = 1; /* g=1 for small z */
  if y = 99 then y = .; /* recode 99 as missing data */
```

#### \*\*\*\*\* PROCEDURES \*\*\*\*\*

Once you have created a SAS data set with a DATA step, you can analyze and process it with SAS procedures. SAS procedures are programs that read SAS data sets, compute statistics, print results, and create other SAS data sets. A procedure or PROC step begins with a PROC statement and may include other statements or options which are particular to the procedure invoked. A PROC step always processes the most recently created SAS data set unless otherwise specified.

```
proc print;       /* prints the last data set created */
proc print data=a; /* explicitly prints the data set "a" */
```

Procedures usually produce printed output but do not create a new data set unless they are explicitly told to do so. The "output" statement is used by several SAS procedures to create new data sets.

#### NUMERICAL SUMMARIES:

```
proc univariate; /* detailed univariate summaries */
  var x y;       /* for variables x and y */
```

```
output out=b m=mx std=sx; /* create set b with mean and SD for x only */

proc means noprint;      /* suppress printout of proc means */
  var x y;               /* for variables x and y */
  output out=b m=mx my std=sx sy; /* output means and SDs for x and y */
proc print;              /* prints data set "b" */
```

#### SORTING AND RUNNING PROCS BY SUBGROUPS:

Sometimes it is very helpful to run each of several subgroups through some summary or analysis procedure. This must be preceded with the "sort" procedure and use of the "by" phrase.

```
proc sort;          by trt sex;
proc means;         by trt sex;
```

Sorting is cheap. It is a good idea to always "sort" before using "by", even if you think you did it earlier in your program.

#### GRAPHICAL SUMMARIES:

```
proc univariate plot normal; /* histogram type summaries */
  var x;
```

The "plot" option to "proc univariate" produces a stem-and-leaf plot, a boxplot, and a normal probability plot. The "normal" option tests for normal distribution.

```
proc plot;          /* scatter plot */
  plot y*x;         /* plot y vertical and x horizontal */
  plot y*x='*';    /* use "*" as plotting symbol */
  plot y*z=trt;    /* use value of trt as plot symbol */
  plot y*x='*' y*z / overlay; /* overlay two plots on same page */
```

#### \*\*\*\*\* REGRESSION and ANOVA \*\*\*\*\*

The main workhorse for regression is "proc reg", and for analysis of variance, "proc anova". The general linear model "proc glm" combines features of both and handles problems with aspects of regression and of analysis of variance. Further, one must use "proc glm" for analysis of variance when the design is not balanced.

```
proc reg;
  model y = x; /* simple linear regression */
  model y = x1 x2 x3; /* multiple regression */
  model y1 y2 y3 = x; /* several different responses at once */
```

The "model" phrase indicates which variables are response (y) and which are predictors (x, or x1,x2,x3). Here are some print options for the model phrase:

```

model y = x / noint;          /* regression with no intercept */
model y = x / p;             /* print predicted values and residuals */

```

You can also create new data sets with proc reg.

```

output out=b p=py r=ry      /* predicted & residual values in "py" & "ry" */

```

PROC ANOVA: This procedure is used for analysis of variance of balanced data. It will yield results with unbalanced data but the results are WRONG! Unbalanced designs should use "proc glm".

```

proc anova;    class trt;          /* 1-way with multiple comparisons */
  model y = trt;

```

```

proc anova;    class fert var;    /* two-way anova */
  model y = fert var;
  means fert var / lsd lines;     /* means of fert and var with LSD test*/

```

```

proc anova;    class fert var;    /* two-way anova with interaction */
  model y = fert var fert*var;    /* interaction signified by asterisk */
  means fert var / lsd lines;

```

```

  model y = fert|var;            /* same as above; the | tells SAS to do all
                                possible interactions between those variables. */

```

The "class" phrase is required and identifies the categorical variables. The "model" phrase has only a few options, and these are not often used. The "means" phrase is quite handy to do multiple comparisons. Options include:

```

  means trt / bon;              /* Bonferroni */
  means trt / lsd lines alpha=.10; /* LSD at level 10% (5% is default) */
  means trt / tukey;           /* Tukey's test */

```

If you want to save predicted values or residuals, or to evaluate contrasts, you must use "proc glm" instead of "proc anova".

PROC GLM: This works much like "proc reg" and "proc anova" except that we can combine regression variables with categorical (class) variables. The organization of the printout is slightly different from "reg" and "anova", and some model and output options are different. Further, if you want model parameter estimates, it is best to explicitly request the "solution" option in the "model" phrase.

```

proc glm;          /* simple linear regression */
  model y = x / solution;

```

```

proc glm;    class fert var;    /* two-way anova */

```

```

  model y = fert var;

```

```

proc glm;    class trt;          /* analysis of covariance */
  model y = x trt;

```

The "class" phrase works like "proc anova". However, here we can have both categorical (identified in "class") and continuous variables in the model.

The "model" phrase indicates which variables are response (y) and which are predictors (x, or x1,x2,x3). You won't get parameter estimates (solution) if there is a "class" phrase unless you ask for them. Here are some options:

```

  model y = trt x / solution;    /* print parameter estimates and SEs */
  model y = x / noint;          /* no intercept (as in proc reg) */

```

The "means" phrase works much the same in "glm" as in "anova".

Contrasts can be set up if means aren't enough. The "contrast" phrase contains a quoted title, variable name and the contrast coefficient values.

```

  contrast 'A vs. rest' trt 2 -1 -1;
  contrast 'BD vs. CE' trt 0 1 -1 1 -1;

```

The "output" phrase can have several keywords which can be used together.

```

  output out=b p=py r=ry      /* predicted & residual values in "py" and "ry" */

```

SPLIT PLOT DESIGN:

The split plot design below uses the "test" phrase, which allows one to test the whole plot factor (a) against the first error (a\*rep), while testing the split plot (or nested) factor (b) against the second error (the error part from fit).

```

proc anova; class a rep b;      /* split plot anova */
  model y = a a*rep b a*b;
  test h = a e = a*rep;

```

EXAMPLE PROGRAMS

```

data a;                          dat a;
  infile ' test.dat' missover;    infile 'test2.dat' missover;
  input block trt $ y;           input x1 x2 x3 y;
proc sort; by trt;              data b; set a;
proc means noprint; by trt;     logy=log(y);
  var y;                         proc plot;
  output out=b n=ny mean=my stderr=sey; plot (y logy)*x1;
proc print;                     proc reg;
proc plot; plot my*trt;         model y logy=x1 x2 x3 / solution;
proc anova data=a;             output out=c p=py ploxy r=ry rlogy;

```

```
class block trt;  
model y=block trt;  
means trt / lsd lines;
```

```
proc plot;  
plot ry*py;  
plot rlogy*plogy;
```



University of Wisconsin - Madison  
College of Agricultural & Life Sciences

**CALS Computer Lab**

1675 Observatory Drive  
Madison, WI 53706  
(608) 263-2817