SAS Procedures

Class 4

TITLE STATEMENT

- enclose in quotes
- prints at the top of each page in OUTPUT window
- limit of 10 (Title, Title2, ..Title10)
- limit of 132 characters each
- in effect for session until new title statement

Syntax:

TITLEn ‘text string’;

Example:

Title1 ‘Example of title’;
Title2 ‘Second title’;

SAS Formats

- It is sometimes useful to store data in one way and display it in another. For example, dates can be stored as integers but displayed in human readable format.
- A SAS format changes the way the data stored in a variable is displayed. There are two types of format:
  - Internal formats (SAS already knows about these)
  - User defined formats (you define these yourself).

Internal SAS formats: Class4_1.SAS

A format statement tells SAS to use that format with one or more variables

Permanent formats: Class4_2.SAS

A format statement added to a dataset permanently connects the format to a variable. The format information is stored in the dataset header.
**User defined formats: Class4_3.SAS**

- Define the format using `proc format`.
- Tell SAS to use the format with a specific variable by using the `format statement` as before.

```sas
proc freq data=mylib.nmes_tot;
table male;
format male genlb.
run;
```

<table>
<thead>
<tr>
<th>gender</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>3097</td>
<td>56.31</td>
<td>3097</td>
<td>56.31</td>
</tr>
<tr>
<td>male</td>
<td>2403</td>
<td>43.69</td>
<td>5500</td>
<td>100.00</td>
</tr>
</tbody>
</table>

**User defined formats**

- `proc format` defines the format `genlb`.
- The format statement applies the format to the variable `gender`.

```sas
proc format;
value genlb 1='male'
0='female';
proc freq
data=mylib.nmes_tot;
table male;
format male genlb .;
run;
```

<table>
<thead>
<tr>
<th>gender</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>3097</td>
<td>56.31</td>
<td>3097</td>
<td>56.31</td>
</tr>
<tr>
<td>male</td>
<td>2403</td>
<td>43.69</td>
<td>5500</td>
<td>100.00</td>
</tr>
</tbody>
</table>

**User defined formats**

**Syntax:**

```sas
proc format <options>;
value FormatName
    range1 = 'formatted value1'
    ....
    rangen = 'formatted valuen';
run;
```

Used to define a format.

**Format names:**

- Must be 8 or fewer characters long.
- Cannot end with a number.
- Character formats begin with a `$`.
- Can not use a SAS internal format name.
- Refer to format in format statement by using the name followed by a period.

**Format ranges:**

- You can specify a range of values to be formatted in a given way.

```sas
proc format;
value age
    10-29 = '10 - 29'
    30-39 = '30 - 39'
    40-49 = '40 - 49'
    50-75 = '50 - 75';
value $dpt
    'A' = 'Dept A.'
    'B' = 'Dept B.';
run;
```

Defines three formats, gen, age and dpt.

Format dpt is a character format suitable for character variables.
specifying format ranges

- **low**: lowest value (excludes missing)
- **high**: highest value
- **other**: all other values not listed (including missing values)

- value1 - value2 means \([value1, value2]\]
- value1 -< value2 means \([value1, value2)\]
- value1 <- value2 means \((value1, value2]\)

Example: Class4_4.sas

```sas
proc format;
  value expf
    low=<170='<=170'
    170-<629='[170,629)'
    629-<1896='[629-1896)'
    1896-high='>=1896';
proc print data=a; *1st 10 records;
  var totalexp_copy totalexp;
  format totalexp expf.;
run;
```

format names:

- must be 8 or fewer characters long
- cannot end with a number
- character formats begin with a $ sign
- can not use a SAS internal format name
- refer to format in format statement by using the name followed by a period

Descriptive statistics

- exploratory data analysis is very important from many perspectives
- in SAS there are three procedures used routinely

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>tables for categorical data</td>
</tr>
<tr>
<td>univariate</td>
<td>Descriptive statistics for numeric data</td>
</tr>
<tr>
<td>means</td>
<td>Descriptive statistics for numeric data</td>
</tr>
</tbody>
</table>

proc freq

- produces frequency counts and cross-tabulation tables
- computes tests and measures of association

Syntax:

```sas
proc freq <options>;
  tables requests / <options>;
run;
```

Example:

```sas
proc freq data=mydata;
  tables gender race chd ;
  tables gender * chd / chisq relrisk;
run;
```

Example: data=mydata is an option

chisq and relrisk are requests for statistics
**Example data: NMES_TOT**


Libname mylib 'd:\temp\sasclass';
proc format;
  value smoke
    0 = 'never'
    1 = 'current'
    2 = 'former';
  value gen
    0 = 'female'
    1 = 'male';
runk;

**Example data: Class4_5.sas**

mylib is a libname (folder), nmes_tot is the data

proc freq data=mylib.nmes_tot;
  tables male*smoke / chisq;
  format male gen. smoke smoke. ;
runk;

**Check Output**

**proc univariate**

- produces simple descriptive statistics
- use PLOT options on PROC statement
- stem-and-leaf plot
- box plot
- normal probability plot (QQ plot)
- side by side box plots for by variable groups

**Syntax:**

proc univariate <options>;
  var variables / <options>;

**Check Output**

**Example: proc univariate : Class4_6.sas**

Example:

proc univariate data=mylib.nmes_tot plot;
  title "Univariate Output for Age";
  var lastage;
runk;
**proc means**
- similar to univariate – no plots
- nicer output, particularly for more than one variable

**Syntax:**
```sas
proc means <options>;
  class varlist;
  var variables / <options>;
  by varlist;
  output out=outdata <options>;
run;
```

**proc means options**
- data=dataset
- statistic
  - default is: n mean std min max
  - Others are: nmiss range median clm
- noprint – suppress printing of output

**stmts**
- class statistics produced for each combination of class variable
- by statistics produced by each combination of by variables
- output produce an output dataset which contains the statistics

**proc means: Class4_7.sas**

Example:
```sas
proc means data=mylib.nmes_tot noprint
  n mean std stderr range nmiss;
  class male;
  var lastage;
  output out=results n=nage mean=mage std=sage;
  format male gen.;
run;
proc print data=results;
run;
```

**Check Output**

**Modeling with SAS**
- examine relationships between variables
- estimate parameters and their standard errors
- calculate predicted values
- evaluate the fit or lack of fit of a model
- test hypotheses
- design
- outcome

**Exercise I**
### The linear model

\[
y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \varepsilon
\]

\[
\varepsilon \sim N(0, \sigma^2)
\]

**Example:**

\[
Weight = \beta_0 + \beta_1 Height + \beta_2 Age + \varepsilon
\]

**Note:** outcome variable must be continuous and normal given independent variables

### The linear model with proc reg

- estimates parameters by least squares
- produces diagnostics to test model fit (e.g. scatter plots)
- tests hypotheses

**Example:**

```
proc reg data=mydata;
  model weight = height age;
run;
```

### proc reg

**Syntax:**

```
proc reg <options>;
  model response = effects </options>;
  plot yvariable*xvariable = 'symbol';
  by variable-list;
  output <OUT=SAS data set>
    <output statistic list>;
run;
```

### proc reg statement syntax:

- `data = SAS data set name` input data set
- `outest = SAS data set name` creates data set with parameter estimates
- `simple` prints simple statistics

### proc reg

**The model statement**

```
model response=<effects</options>;
```

- required
- variables must be numeric
- many options
- can specify more than one model statement

**Example:**

```
model weight = height age;
model weight = height age / p clm cli;
```

### proc reg

**The plot statement**

```
plot yvariable*xvariable <=symbol> </options>;
```

- produces scatter plots - yvariable on the vertical axis and xvariable on the horizontal axis
- can specify several plots
- optional `symbol` to mark points
- yvariable and xvariable can be variables specified in model statements or statistics available in output statement

**Example:**

```
plot weight * age / pred;
plot r. * p. / vref = 0;
```
**Check Output**

**Proc reg**

Some statistics available for plotting:
- P. predicted values
- R. residuals
- L95. lower 95% CI bound for individual prediction
- U95. upper 95% CI bound for individual prediction
- L95M. lower 95% CI bound for mean of dependent variable
- U95M. upper 95% CI bound for mean of dependent variable

Example:
```
plot weight * age / pred;
plot r. * p. / vref = 0;
plot (weight p. l95. u95.) * age / overlay;
```

**Proc reg example: Class4_8.sas**

Variables of interest:
- totalexp – total medical expenditure ($)
- chd5 – indicator of CHD
- lastage – age at last interview
- male – sex of participant

```
proc reg data=mylib.nmes_tot;
model totalexp = chd5 lastage male;
run;
```

The run statement

Many people assume that the run statement ends a procedure such as proc reg. This is because when SAS encounters a run statement it executes any outstanding instructions in the program buffer. But it may or may not end the procedure.

```
proc reg data=mylib.nmes_tot;
model totalexp = chd5 lastage male;
run;
```

/ * ends the procedure */
**proc glm (the general linear model)**

- uses least-squares with generalized inverses
- performs linear regression, analysis of variance, analysis of covariance
- accepts classification variables (discrete) and continuous variables
- estimates and performs tests for general linear effects
- proc anova is suitable for "balanced" designs; proc glm can be used for either balanced or unbalanced designs
- suitable for random effects models

**proc glm**

**Syntax:**

```sas
proc glm data=name <options>;
class classification variables;
model response=effects /options;
means effects / options;
random effects / options;
estimate 'label' effect value / options;
contrast 'label' effect value / options;
run;
```

**proc glm**

**response (dependent) variable is continuous** – same normality assumption as in proc reg

**independent variables are discrete or continuous; discrete must listed on class statement**

**interaction terms can be with an asterisk a*b, e.g.**

```sas
model bmi= a b a*b;
```

**proc glm example Class4_9.sas**

```sas
proc glm data=mylib.nmes_tot;
where totalexp>0;
class chd5 smoke;
model totalexp=chd5 smoke /solution;
means smoke;
run;quit;
```

**Check Output**

- 1. **solution** show estimated parameters
- 2. **means** show means for smoke variable
- 3. **class** treat smoke as discrete
**proc glm example**

```plaintext
proc glm data=mylib.mnes_tot;
  where totalexp>0;
  class chd5 smoke;
  model totalexp=chd5 smoke /solution;
  means smoke;
  format smoke smoke. ;
run; quit;
```

**Check Output**

1. format changes reference group

---

**reg and glm**

- `proc reg` and `proc glm` procedures are suitable only when the outcome variable is normally distributed.
- `proc reg` has many regression diagnostic features, while `proc glm` allows you to fit more sophisticated linear models such as random effects models, models for unbalanced designs etc.

---

**non-normal outcomes**

- In many situations we cannot assume our response variable is normally distributed.
- `proc reg` and `proc glm` are not suitable for modeling such outcomes.

**Example:**
Suppose you are interested in estimating the prevalence of disease in a population. You have an indicator of disease (1 = Yes, 0 = No)

---

**proc logistic**

- fits linear logistic regression models for binary or ordinal response data by the method of maximum likelihood
- enables you to specify categorical variables (also known as CLASS variables) as explanatory variables. It also enables you to specify interaction terms in the same way as in the GLM procedure.

**Example:**
You are interested in estimating how the incidence of infant mortality has changed as a function of time.
**proc logistic**

**Syntax:**

```
proc logistic <options>;
  by variables;
  class variables </options>;
  model response = effects </options>;
run;
```

**Example:** `Class4_10.sas`

---

**proc logistic:**

- **class** statement says which variables are classification (categorical) variables
- **by** statement produces a separate analysis for each level of the by variables (data must be sorted in the order of the by variables)
- **response** variable is the response (dependent) variable in the regression model.
- `<effects>` are a list of variables. These are the independent variables in the regression model. Any independent variables that are categorical must be listed in the Class statement.

---

**proc logistic:**

- By default, PROC LOGISTIC assigns Ordered Value 1 to response level 0, causing the probability of the nonevent to be modeled.
- Specify the DESCENDING option in the PROC LOGISTIC statement, which reverses the default ordering of Y from (0,1) to (1,0), making 1 (the event) the level with Ordered Value 1:

```
proc logistic data=mylib.nmes_tot descending;
model chd5 = lastage male ;
```

---

**proc logistic:**

- `Check Online Doc` for a further description of the options and statements available for the logistic procedure

---

**example: logistic regression**

- Perform a logistic regression analysis to determine how the odds of CHD are associated with age and gender in the 1987 NMES
**Example: Class4_10.sas**

```sas
proc logistic data=mylib.nmes_tot descending;
class male / param=reference; *add descending option to get 1 vs 0;
model chd5 = lastage male ;
*format gen.;
run;
```

- The `descending` option on the `PROC` statement means that we are modeling the probability that `chd5=1` and not the probability that `chd5=0`.

**survival analysis**

- Survival analysis is a class of statistical methods for studying the occurrence and timing of events.

- The survivor curve $S(t)$ is the probability of surviving to time $t$:
  \[ S(t) = P(T > t) = 1 - F(t) \]

- Censored observations usually arise in such analyses.

**proc lifetest**

- Produces estimates of survivor functions using Kaplan-Meier or actuarial method.
- Tests the null hypothesis that the survivor functions are identical for two or more groups.

**Syntax:**

```
proc lifetest <options>;
  time variable <*censor(list)>;
  by variables;
  freq variables;
  id variables;
  strata variable <(list)> < ... variable <(list)>>;
  test variable;
run;
```
**proc lifetest options**

data = data set
method = KM default
notable suppresses printing of survival function estimates
plots = (S,LS,LLS,H,P)

**Example:**

```
proc lifetest
   data=mylib.mydata
   plots=(survival,hazard);
```

**the time statement**

time variable*censor(list);
where:
variable - time to event variable
censor - censoring indicator
   (right censoring)
list - list of censored values

**Example:**

time t * flag(1,2);

**the strata statement**

strata variables;
where:
variables - variables that determine groups

**Example:**

time t * flag(1,2);
strata gender;

**Example data: CHOLEX**

The CHOLEX dataset is a subset of the data from the Johns Hopkins Precursors Study.
Want to compare time to cerebrovascular disease (cvd) for persons with different cholesterol levels (cholest).

**Strategy:**
- create format for cholest variable
- Kaplan-Meier plots
- Use cvdage (age at CVD) as the time variable
- cvd - censoring indicator;
  value of 0 indicates a censored individual
- cholest - strata variable categorized into 4 groups by format

**Example: Class4_11.sas**

```
proc format ;
value chollb low-173='<=173'
  174-193='174-193'
  194-214='194-214'
  215-high='215+';
proc lifetest data=mylib.cholex
   plots=(survival,ls,loglogs) notable;
   time cvdage*cvd(0);
   strata cholest;
   format cholest chollb.;
RUN;
```

**proc phreg**

- uses Cox’s partial likelihood method to estimate regression models with censored data
- handles both continuous-time and discrete-time data
- performs stratified analyses
- allows time-dependent covariates
- allows counting process formulation input and thus recurring outcomes
- perform conditional logistic regression analysis for matched case-control studies
**proc phreg**

Syntax:

```
proc phreg <options>;
  model response*censor(list) = variables / <options>;
  strata variable <list>;
  output out=name <keywords=names>;
run;
```

Note: no class statement, create your own dummy variables for categorical explanatory variables.

**some proc phreg options**

```
data  = name /* input data name */;
  outtest = name /* output data name */;
no祺拳  /* suppress printing */;
  covsandwich /* request robust sandwich estimator for variance */;
```

Example:

```
proc phreg data=mylib.mydata;
  model survtime * status(0) = sex trntment ;
  strata agegroup;
run;
```

**Example: Class4_12.sas**

```
/* create dummies for levels of cholest */
if cholest NE . then do;
  group1=0; group2=0 ; group3=0;
  if (cholest >= 174 and cholest <= 193) then group1=1;
  else if (cholest >= 194 and cholest <= 214) then group2=1;
  else if cholest>214 then group3=1;
end;
```

**time-varying covariates**

- The Stanford Heart transplant data
- 103 cardiac patients enrolled in transplantation program
- After enrollment patients waited varying lengths of time for a suitable donor
- 30 patients died before receiving transplant

Variables:

- `dob` date of birth
- `doa` date of acceptance
- `dot` date of transplant
- `dls` date last seen (dead or censored)
- `dead` coded 1 if dead at dls; 0 otherwise
- `surg` 1 if patient had open-heart surgery prior to doa

**Example: Class4_12.sas**

```
proc phreg data=temp;
  model cvdage*cvd(0) = group1 group2 group3 /risklimits;
run;
```

```data scan```
```
  set lecture5_stem;
  surv1 = dls - doa;
  surv2 = dot - doa;
  wait = dot - doa;
  age_trn = |dot - dob|/365.25;
  age_accpt = |doa - dob|/365.25;
  age_dls = |dls - dob|/365.25;
  if dot = . then tran = 0;
  else tran = 1;
run;
```
```
proc phreg data=stan;
  model surv1 * dead(0) = tran surg age_accpt;
run;
```
```
Do the results make sense?

**time-varying covariates**
- the surgery effect might be what you would expect
- the age effect might be real
- the transplant effect looks strange
- One reason why people did not get a transplant is because they died before a suitable donor could be found
- Maybe we need a time-varying covariate?
- SAS will let you place “data step” like code inside a `proc phreg` to handle time-varying covariates

---

```
proc phreg
    data=trans;
    model survi * dead(0) = plant surg ageasopt / time=exact;
    if wait > survi or wait = . then plant = 0; else plant = 1;
run;
```

---

More Statistical Procedures:
- Random effects models can be fit using Proc GLM and Proc Mixed
- Longitudinal/Repeated Measures/Panel data can be modeled using Proc GLM, Proc Genmod and Proc Mixed
- Conditional logistic regression can be fit using proc phreg