meat.sas: Explanation of code

Goals of code:

- Reading data included with the SAS code
- Fitting a regression line
- Estimating mean Y at a specified X
- Confidence intervals for regression parameters or estimates
- Calculating predictions for many X values
- Calculating confidence and prediction intervals
- Printing a few observations in a data set
- Overlaying observed and predicted values
- Analyzing a subset of the data
- ANOVA lack of fit test

Reading data saved with the SAS code: `cards;`
The data can be stored in the data step. This can be very handy when you want to read a small data set. I discourage it when the data set is large (It is no fun to help debug a file of SAS commands that includes 200+ lines of data).

To save the data with the SAS program, write a data step to read that data. Omit the infile statement. **After** the input statement and any data manipulation commands, includes a `datalines;` statement followed by lines of data. The `cards;` command is equivalent to `datalines;`. (who remembers punched cards any more?) Notice that `datalines;` ends with a semi-colon. Subsequent lines are considered to be data lines **until the next ;**. My practice is to put a semicolon on a line by itself, so the end of the data is clearly marked. The run; tells SAS to execute the code and create the data set.

In this problem, the desired regression equation uses X = log(time), so the variable logtime is created with the log transformed time. Remember that the definition of the logtime variable is placed after the input statement but before the datalines; statement.

Fitting a regression line: `proc glm; model ph = logtime;`
Many SAS procs will fit a regression line. We will consider two, `glm` and `reg`. Each provides slightly different functionality. proc glm makes it easy to print out estimates of the predicted mean for any X value.

To fit a regression with proc glm, the desired regression model goes on the model statement. As with fitting an ANOVA model, the response variable (Y) goes on the left of the = and the predictor variable (X) goes on the right. You do not X in a class statement. If you include `class ph`, each unique ph value is used to define a group and proc glm will fit a model with a separate mean for each ph. Here, we want to fit a regression line. No class statement.

The output from proc glm includes a block of results with names: Parameter Estimate .... These are the fitted regression coefficients. The intercept is β_0. The logtime row is the regression slope, β_1. It is labelled by the name of the X variable, which simplifies understanding the output when there are more than one X variable.

The values in the table are the estimate, its standard error, the T statistic testing H0: parameter = 0, and the p-value for that test. The test of the intercept is usually not very interesting (ph 0 is seriously bad), but the slope test is almost always very interesting.

Estimating mean Y at a specified X: `estimate 'label' intercept 1 logtime 1.6094;`
The predicted pH at 5 hours is the predicted value from the regression line when logtime = log(5) = 1.6094. That can be calculated using an estimate statement. That prediction is:

$$\begin{aligned}\widehat{\mathrm{pH}} &= \hat{\beta}_0 + 1.6094\hat{\beta}_1 \\ &= (1)\hat{\beta}_0 + (1.6094)\hat{\beta}_1\end{aligned}$$

The estimate statement estimates this. We need the intercept multiplied by 1 plus the logtime slope multiplied by 1.6094. That is written as: `intercept 1 logtime 1.6094`. As with earlier uses of estimate, the estimate statement has a label in quotes followed by the description of the quantity to estimate.

Confidence intervals for parameters or estimates: `/clparm`
Adding the `\clparm` option to the model statement provides confidence intervals for all model parameters (i.e., intercept and slope) and all linear combinations of parameters (i.e., the stuff in estimate statements). The default is a 95% interval; adding `alpha=` after the `/clparm` will give you 1-alpha intervals.

Setup to calculate regression predictions at many X values: `data meat2;`
The estimate statement just demonstrated prints predicted values for specified X values. Each new X requires a new estimate statement. If you want predicted values for more than a few observations, it is easier to add the desired Xs to the data set.

The key idea is that if you provide an observation with an X value and a missing Y value, that observation wont be used to estimate parameters (because Y is missing) but you can calculate predicted values and related information (because X is specified). You just cant calculate a residual (again because Y is missing).

The two `data meat2;` steps show two ways to do similar things. You only need one of those data steps. The first data step (with the datalines statement) reads a set of new X values at which you want to make predictions. The code to indicate a missing numeric value in SAS is . (not in quotes). The string . in quotes is the character value .. The lonely . without quotes is the numeric missing value. This can be used in any data set. In particular, if a value for one variable is missing, use a lonely . to indicate that.

The second data meat2; step uses a loop to sequentially generate 1.0, 1.25, 1.5, ... 8. ph is set to a missing value. The do loop has the obvious syntax: do variable = start to end by step. If by step is omitted, the loop steps by 1, so do i = 1 to 5 generates i=1, 2, 3, 4, 5. The do loop ends with the end; statement. All statements between do and end; are executed each time through the loop. Note: If you write do time = 1, 5; , SAS will execute the loop twice, once with time = 1 and then with time = 5. The loop sets the value of the time variable. In addition, we want to calculate the log transformed time, with is done the usual way. The output; writes the observation to the data set. An explicit output; statement is needed here because we are generating a sequence of observations, and we need to write each one to the data set.

The last piece of setup is to create a data set (to be called all) with both the real data (meat) and the prediction points (meat2). We have used the set command before to read observations from a pre-existing SAS data set. When there are multiple data set names, set will read all observations from the first data set, then all from the second, and then the third, and fourth, ... for all data sets. So, set meat meat2; will concatenate the meat and meat2 data sets. We use that data set to fit the regression.

Calculating confidence and prediction intervals: `output;` statement
The previous code demonstrates fitting a regression using proc glm. The new things here are an output statement with keywords to request specific items. The output statement creates a new data set with all the variables in the original data set and additional pieces of information. Each desired piece is specified by keyword = name . The keyword specifies what piece of information you want; the name is the variable name in which that piece is stored. You choose the variable name. There are lots and lots of different keywords (and proc reg, the other "fit a regression line" proc, has even more). I indicate some of the most useful keywords in meat.sas. Comments in the file describe what each keyword gives you. A full list of keywords can be found in the SAS documentation for proc glm.

Only printing a few observations: `data = resid (obs=5);` Sometimes, I want to print a few observations in a data set to get variable names (e.g., after a proc import) or check for gross errors. Anytime you explicitly indicate the name of a data set (by data=), you can add a data set option in (). `obs=5` says to only process the indicated number of observations, so `data=resid (obs=5)` prints the first 5 observations in the resid dataset. `data=resid (obs=25)` prints the first 25 observations. Note: You can use obs= in any proc step, but you probably dont want to. If you used it in a proc step that does a statistical analysis, that analysis would only use part of the data.

Overlaying observations and predicted values: `proc sgplot;`
We have previously used proc sgplot; scatter ; to plot data. You can provide multiple plot commands which will overlay all the requested pieces. The series command draws lines (by connecting the observations). Since we want the lines to connect adjacent time values, I sort the data set by time

first (proc sort; by time).

The first `proc sgplot` overlays the data and fitted line. The second adds the lower and upper prediction limits. The `/lineattrs` option (line attributes) modifies how the line is drawn. The suboptions go in (), exactly as indicated here. Besides color and pattern, there is also a width suboption. There are many other options besides lineattrs and many other drawing commands besides scatter and series. The SAS documentation for proc sgplot gives you all the details.

Analyzing a subset of the data: `where` or `data set; if`
SAS provides a couple of ways to restrict an analysis to a subset of the data. If you want to do many analyses or plots with a subset, I find it easier to create a second data set with the desired subset of data. If you want to do one analysis, it can be easier to specify that subset in the analysis. The first proc glm illustrates specifying a subset in the analysis. This is done with a `where` statement. where is followed by a logical expression, e.g., `time <= 6`. The analysis will use only the rows of data where this is true. So here, only times 1, 2, 4, and 6. Not 8.

To create a new data set with a subset of values:
This is done in a data step, `data meat16;`, followed by `set` to read observations from the meat data set. The crucial statement is `if` followed by a location expression. The new data set will only include observations where that logical statement is true.

Logical operators: SAS provides an extensive list of logical operators. These include

Symbol	meaning
=	equals
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
<>	not equal
in (1,3,6)	will be true if first argument matches any of the values in the ()

ANOVA lack of fit test: `data meat3;` and subsequent `proc glm`.
To construct the ANOVA lack of fit test, we need to fit the regression (with a continuous X variable) followed by the ANOVA model (with a categorical X variable). We can do this by create a copy of the X variable so that we can make one be continuous and the other be categorical. The meat3 data step does that. The copy is called ctime, but the name is immaterial. It just has to be something different from the original name. The actual work is done by the proc glm with the model ph = time ctime / ss1; statement. One copy of the X variable (ctime) goes in a class statement (to fit the ANOVA model with a separate mean for each unique X value). The model statement fits the regression (time) followed by the ANOVA model (ctime). The change in SS is the SS for lack of fit. The /ss1 option requests only sequential SS (SAS type 1). That's because some of the type III SS don't make sense here (e.g. for ctime followed by time).
Note: If the first variable in the model is the one specified in the class statement) the SS and df for the second term are both 0. Fix by reversing the order.