dietall.sas: Explanation of code

Goals of code:

- Log transform data values, also other transformations
- Fitting ANOVA models
  - plotting residuals against predicted values
  - Comparing models
- “After the ANOVA”
  - estimating group means and standard errors
  - estimating linear combinations of means
  - all pairwise differences
  - common multiple comparisons adjustments

Log transform data values: Two ways to do this:

1. As part of reading in the data: \( \text{loglife} = \log(\text{lifetime}) \);
   You can add statements to create new variables to a data step. These need to go after the
   input statement and before the run; This makes sense when you realize that SAS executes
   the statements in order. The input statement reads the value for the lifetime variable. The
   log(lifetime) code computes a new value from the lifetime value.

2. As part of a new data step: \( \text{set temp;} \);
   If you use proc import or the import wizard to read a data file, you just get the contents of the
   data file. Any additional variables have to be computed in a new data step. The data temp;
   step reads the data file into the temp SAS data set. The next data diet; creates the diet data
   set (this will overwrite the previously created one, but that isn’t important). This data step
   reads observations from the temp data set (the set temp;) command, then uses the value of
   lifetime to create new variables: the log transformation, and also a square-root transformation \( \sqrt{} \) and reciprocal transformation \( 1/\).

Calculating group-specific summaries: proc means n mean std stderr;
We’ve used proc means before to compute summary statistics for each group. The keywords after
proc means request specific summary statistics. In order, these are: the sample size, the average,
the standard deviation, and the standard error.

Note: the sd and se are estimated from each group separately. There is no pooling across groups.
Hence, these estimates (in particular, the se) are not the same as the values from proc glm.

Fitting a separate means ANOVA model: proc glm;
We used proc glm last week to get residuals. It is a very flexible proc that fits many different sorts
of models. The code for two groups (last week) is identical to the code for many groups (ANOVA, this week). You can’t use proc ttest; for more than 2 groups; you can use proc glm.

The model to be fit is specified by the class and model statements:

```plaintext
class trt;  // identifies variables that specify groups  
model lifetime = trt;  // identifies the response variable (left hand side of =) and the desired model. Here, the model is a function of the trt value. Because of the class statement, trt identifies groups. Hence, glm will fit a model with a separate mean for each trt group.
```

Important pieces of the output are:

- Error SS, error df, and MSE: In the first box of output, line labelled Error
- Pooled sd: In the second box of output, box labelled Root MSE
- F test of H0: no difference in means: In the last box of output, with the column labeled Type III SS, line labeled trt.

There is also a box with a Type I SS. For this model, the type I and type III SS and tests are exactly the same. Lecture will discuss the differences between these two. Except in special circumstances, US practice is to use type III SS.

**Storing residuals and predicted values:**

```plaintext
output out=resids r=resid p=yhat;
```

We used the output command to store residuals last week. Now, we want to plot residuals (Y axis) against predicted values (X axis). That means we need to store both residuals and predicted values for each observation. `r=` names the variable to store the residual value for each observation. `p=` names the variable to store the predicted value.

These values are then plotted using `proc sgplot; scatter x=yhat y=resid;`

The scatter statement draws a scatter plot. `x=` identifies the variable for the X axis; `y=` identifies the variable to go on the Y axis.

**Fitting the equal means model:**

```plaintext
second proc glm;
```

This proc glm has only a model statement. That model has the name of the response variable and `an =`, but nothing more. Hence SAS fits a model with a single mean for all groups (SAS calls it Intercept).

The important piece of information is the Error SS, which you can use to hand calculate an F statistic.

This is demonstrate what SAS does for you in the previous proc glm and show how to compute numbers used in the text. In practical data analysis, I almost never have to explicitly fit the equal means model.

**After the ANOVA:** The last `proc glm;`

I will argue in lecture that fitting an ANOVA model and calculating the F statistic is really just the
start of a data analysis. All of what I call “After the ANOVA” is specified by additional statements in the proc glm. All come after the model statement. The example includes a lot of different things. None depends on any other statement, so each can be used in isolation or combination.

**group means and se’s:** `lsmeans trt /stderr;`
These se’s use the pooled sd.

**estimating linear combinations of means:** `estimate 'Diff: R/R50 - N/R50' trt 0 0 -1 0 1 0;`
The code includes many different estimate statements, because many different specific combinations are of interest. My code illustrates how to do some things that are not part of the book’s analysis of the diet data. The purpose of these results will be explained in lecture.

Each estimate statement has the same structure: the word estimate followed by:
a title in quotes (single or double): This can be anything, for your use to identify this result in the output
a variable name: the variable identifying the groups to be compared
a string of numbers separated by spaces: This specifies coefficients for your desired linear combination.

The coefficients are specified **in the same order** that SAS uses for the group names. You see this order in two places:

- The output from the class statement. Look for the Values list.
- The output from any lsmeans statement.

The order is alphanumeric (ASCII sort sequence, if you recognize/care about that), with lower case letters following upper case. I find it easiest to run a proc glm to figure out the order, then write the estimate statements and rerun the proc glm.

Fractions can be written as decimal numbers (e.g., 0.5) if the fraction is a nice number. They can also be written as integers followed by a `/divisor = ` option. The two lines:
`estimate 'N/N85 - ave(N/R50, N/R40)' trt 1 -0.5 -0.5 0 0 0;`, and
`estimate 'same thing, diff. way' trt 2 -1 -1 0 0 0 /divisor = 2;` are identical. You need to use `/divisor = ` when the fraction is not “nice”, e.g. 1/3 or 1/7. That is because the linear combination only makes sense when the coefficients sum to 0. Approximating 1/3 doesn’t always work unless you use enough digits to trigger roundoff. For example `-0.333 -0.333 -0.333 1` sums to 0.001, which is not 0. If you have various not nice fractions, you have to find the least common multiple, so all fractions have the same divisor. That’s what’s being done in the equal rates estimate.

**all pairwise differences, with multiple comparisons adjustments:**
`lsmeans trt /stderr pdiff adjust = tukey cl;`
The cl option requests 95% confidence intervals for each lsmean. The pdiff options requests all pairwise differences. The default for pdiff is just the p-value for the test. pdiff and cl give you the...
estimated difference and 95% confidence intervals for each pair. Adding alpha = gives you other coverages. E.g., alpha = 0.10 gives you 90% intervals. adjust= specifies which multiple comparisons adjustment to use. There are many choices. We will discuss tukey, t, and bonferroni. The book also discusses scheffe.