patty.sas: Explanation of code

Goals of code:

- Averaging observations to one value per experimental unit

Averaging observations: description of the data in patty.txt
The hamburger study used last week was run a second time with a slight modification. In the second experiment, hamburger pattys were randomly assigned to the treatment (active or control) but the bacterial concentration in each patty was measured three times per patty. So there are 12 experimental units (pattys) and 36 observations. The experimental unit is the patty; the observational unit (in the original data) is plate count. This is a violation of independence. A more appropriate analysis is based on the average count per patty. After averaging, the observational unit is the patty (because one row of data per patty), which is the same as the experimental unit (good).

One appropriate analysis is to calculate the averages for each patty and save them in a new data set. Then, we will analyze that new data set. More advanced courses discuss other approaches.

Read the patty.txt file. You should have a data table with three columns: trt, rep and cfu. There are three cfu values for each replicate. We want to average each set of three to get a single mean for each combination of trt and rep.

Averaging observations for subsets of observations: `proc sort; proc means;`
Two steps are necessary to average the three observations for each experimental unit:

- Sort the data so all observations for each group are together: `proc sort`
- Calculate and save the averages for each group `proc means`

We have previously used `proc means` with a class statement to compute summary statistics for subgroups of observations. We could do this with two grouping variables (trt and rep) but if we did, SAS gives us more than what we want. The most straightforward way to get averages only for each combination of trt and rep is to use a by statement.

By statements can be used in **any** SAS procedure. They tell SAS to do separate analyses on each group of observations, where a group is defined as having the same values of the variables in the by statement. A by statement can also be used in data step where it has a slightly different use (to indicate start and end of groups of observations). SAS requires that each block of observations is contiguous. The easiest way to ensure this is to first sort the data set by the necessary variables.

Sorting a data set: `proc sort; by ;`
The by statement in a proc sort step specifies the order in which the data set will be sorted. When the by statement has more than one variable, the first variable is the "major" sort order, the second is the "minor". So `by treatment rep;` will put all of treatment A first, then all of treatment B

(major variable). Within treatment A, all of rep 1 are first, then rep 2, ... You can add as many variables as appropriate; each additional one is nested inside the previous sort.

Character variables are sorted in alphabetic order, so 'a' is before 'b', 10' is before '100', and '100' is before '20'. Capital letters are before lower case letters, so 'B' is before 'a'. Numeric variables are sorted in numeric order, so 10 is before 20 is before 100. Missing values are before any actual number. The details of the sort order are not relevant (for our current purpose). We only need all observations with the same treatment and rep values to be put together.

The sorted data set replaces the original data set. You can add proc sort out=name; to save the sorted version in a new data set.

Computing means for subgroups: `proc means noprint; by ;`
Proc means with a by statement will compute results for each separate group of observations. Because the by statement specifies both trt and rep, a group is defined as a unique combination of those two variables. As before the var statement specifies the variable to use in the computation.

Saving the results in a new data set: `output out= mean= ;`
By default, proc means will compute the specified summary statistics and print them out. We need to store the averages in a new data set that we can pass into proc ttest, some other analysis proc, or a graphics proc.

The output statement creates a new data set containing specified summary statistics. `out=` names the new data set. `mean=` requests that means be saved (mean=) and provides the name of the variable in which to save them (the word after the =). You can have multiple keyword=name requests if you want to save different summary statistics. The documentation for the output statement lists all the keywords. You can choose any variable name you want.

When you print out the means data set, you see that means contains the by variables (trt and rep)and the desired average (meancfu). You could have called the result cfu (output out=means mean=cfu;) instead of meancfu.

The means data set contains two additional variables `_TYPE_` and `_FREQ_`. Variables bracketed by underscores are variables that SAS automatically provides to you. `_TYPE_` can be ignored; it has no useful information unless you have a class statement. `_FREQ_` is the number of observations in that group, i.e. the sample size for the group.

Suppressing printed output: `proc means noprint;`
Since the purpose of the proc means is to produce a data set, we don't really need the printed output. noprint suppresses the printed output. If there were 100's or 1000's of groups, you really don't want to see everything printed. There are other ways to suppress printed output, using ods commands, which we will see later in the semester.

You now have created the means data set. This can be used for your desired analysis. SAS proc steps don't care how a data set was created, they just require that the data are in a data set. Because

means was created after the patty data set, means will be used whenever you don't explicitly name a data set to use (with data= in a proc statement).