Class Exercise 10

This exercise is based upon Chapter 7 of Delwiche and Slaughter's "The Little SAS book. We will study two different macros (`strata` and `regall`) here found on the webpage in the two SAS source code files `SafetyMacro.sas` and `RegMacro.sas`, respectively. Take your time and be sure you understand every step; be sure to look at intermediate data sheets in the `WORK` library so that you understand the macros' actions.

The first macro (`strata`) was introduced in class. When running this macro, you may need to include `..` in between `&strata` and `csv` in the first `INFILE` statement; on some computers, this may not be necessary–you may need only one period. I've included the version with two periods here. You should be able to tell from the `LOG` file if the correct data set name (`upurban.csv`) was assigned by the macro; if not, adjust accordingly.

The macro computes the probability that sampling units are included in a sample of size 2, when sampling without replacement with unequal sampling probabilities. In class, we had shown that if unit i has sampling weight π_i , then the probability it is selected would be:

$$P(i) = P(i \text{ selected on first draw}) + P(i \text{ selected on second draw}) =$$

$$\pi_i + \sum_{j \neq i} \pi_j \frac{\pi_i}{1 - \pi_j} = \pi_i \left[1 + \sum_{j \neq i} \frac{\pi_j}{1 - \pi_j} \right]$$

The `strata` macro includes a `DATA` step with a nested `DO` loop. It looks complicated, but the actual macro features are really not that sophisticated–`&n` is used mostly to modify the `DO` loop upper limit, and `&strata` is used to modify file names, and that's about it!

Download and save `upurban.csv`–it's referred to as the "traffic intensity" data on the web page. You can also go ahead and run the entire macro; several datasets will be created. Be sure to step through the code and take a look at them:

- Data set `c` drops the county name.
- Data set `d` transposes the data so that traffic intensities are more easily converted to proportions (remember that SAS handles row operations easier than column operations).
- Data set `e` creates proportions from the traffic intensities.
- Data set `one` cleans up some variables.
- Data set `two` computes our sampling probabilities.
- Data set `three` cleans up after our calculations.
- Data set `four` recaptures the county names.

Notice that we're using `DO` and not `%DO` here; that's because were looping *within* a `DATA` step, and hence don't need the added flexibility. Just because we're writing a macro doesn't mean we need to use macro statements indiscriminately.

The rest of the macro is relatively routine. Data `four` looks a little odd and is frankly redundant–we use `firstobs` and `obs` here because when this program was first written, the Stat Lab used to only compute sampling probabilities for counties that had actually been sampled, and so we wouldn't necessarily read the entire list of county names. We don't use a match merge in `final` to add the county names back into the data set since we never changed the order of the variables.

Add up the sampling probabilities for the three counties–what do you think is the significance of the sum?

In our second macro (`regall`), we want to perform a series of simple linear regressions. Our macro should be able to accept a single dependent variable and a list of independent variables, and then construct regressions on each independent variable in turn. Let's start with a straightforward example, and then add a couple important features on our way to our fin- ished macro.

Our data set consists of assessment tests for a group of students. Our dependent (response) variable will be `read`. For our first attempt, we'll simply use some macro variables. Go ahead and run all the commands up through the following statements:

```
...
proc reg data=hsb2;
model read=&indvar;
run;
```

We actually just read in the macro `regall`, but I don't want to use it just yet. The inclusion of `&SYSDAY` and `&SYSDATE` is a little gratuitous here. Otherwise, you see a typical use of `%LET` commands to perform a multiple linear regression. The `%scan` function is new; `%scan(string,i)` can save element i from `string`, where the elements are separated by spaces (as we did here) or by delimiters. Delimiters can be specified as a third argument to `%scan`.

To conduct 4 simple linear regressions on each of those 4 variables, we could construct a macro that accepted a dependent variable, and a set of independent variables. As a first attempt, we would likely find it easiest to also include the number of independent variables, since it would simplify looping. Thats a bit of a cop-out though, since we really should be able to read in the string, and then figure out the number of independent variables ourselves. And that's exactly what we do in the macro `regall`.

SAS's approach to counting all the elements in the string isnt exactly elegant. We can loop through the elements in the `&indep` string using `%scan` until no more elements are found; that is what the `%do %while` statement does. Note that we have to treat the string

`&indep` as a character by enclosing it in quotes, and look for missing character data (“”), not missing numeric data. SAS increments the loop using the `%eval` statement–a form of macro math.

Why do we use `%do %while` here? Because we are executing a `proc reg` within the DO loop, which couldn't be done with the usual `do while` statement. Highlight and run the last statement in the program. Observe the output. You can run regall again with a different choice of dependent variable and independent variables if you would like.