**BIOS 511 Lecture Notes – Fall 2018**

# Wednesday, August 22, 2018 – Lecture 1

-All programs you write in SAS must have a header (at the top) 🡺 documentation that lists the author, the date created, the purpose, the revision history (you won’t really make use of this, but it’s good practice)

-Important option for your SAS programs for this class (will learn about what all this means later):

Option mergenoby=nowarn;

-When you turn in assignments, make sure you export and save your SAS logs, which will be graded

-The only difference (that matters for us) between SAS Studio and SAS University Edition is that SAS Studio can see your entire computer and all your file paths

-SAS Studio and University Edition will only display the latest stuff in your SAS log when you run new code, but SAS Display Manager will append the stuff to the log so you can see everything in that session

-**ON SAS STUDIO edition, you can click and drag the Log tab (or any other tab) all the way to the right until a new window (gray) and green checkmark pops up**

**-Then, you can have your code tab and your log tab on the same page (you can also drag the separated tab back onto the bar to restore the view back to the default, so that each tab has to be viewed in its own page)**

# Monday, August 27, 2018 – Lecture 2

-When you use title and footnote statements in SAS, you can also use title(n) **h=x** “”, where h=x adjusts the height of the title (h=1 is the default)

# Wednesday, August 29, 2018 – Lecture 3

Efficiently Producing Output on SAS

**Q: How can I produce multiple output files in a single location and avoid**

**having to type the file path to that location many times in the program?**

**A: Create and use a SAS macro variable (**in SAS, macro variables are used for TEXT SUBSTITUTIONS, and not just as a shorthand for file paths**)**

**\*\* cumbersome option;**

ods pdf file="C:/Users/psioda/Documents/GitHub/BIOS-511-FALL-2018/programs/2018-08-29-lecture-03/output/file1.pdf";

proc print data = orion.charities(obs=10); run;

ods pdf close;

ods pdf file="C:/Users/psioda/Documents/GitHub/BIOS-511-FALL-2018/programs/2018-08-29-lecture-03/output/file2.pdf";

proc print data = orion.customer(obs=10); run;

ods pdf close;

ods pdf file="C:/Users/psioda/Documents/GitHub/BIOS-511-FALL-2018/programs/2018-08-29-lecture-03/output/file3.pdf";

proc print data = orion.country(obs=10); run;

ods pdf close;

**\*\* more elegant option;**

\*\* define a SAS macro variable named root and give it the value of the "root" directory where your files will go;

**%let** root = C:/Users/psioda/Documents/GitHub/BIOS-511-FALL-2018/programs/2018-08-29-lecture-03;

ods pdf file="&root/output/file4.pdf";

proc print data = orion.charities(obs=10); run;

ods pdf close;

ods pdf file="&root/output/file5.pdf";

proc print data = orion.customer(obs=10); run;

ods pdf close;

ods pdf file="&root/output/file6.pdf";

proc print data = orion.country(obs=10); run;

ods pdf close;

\*\* one a SAS macro variable is defined (i.e., the %LET statement is submitted), anywhere that the SAS code

compiler finds an ampersand (&) followed by the macro variable name (e.g., root), SAS will replace the

text &root with the value of the macro variable. This is called "resolving" the macro variable.

\*\* generally macro variable definitions that are to be used throughout a program are defined near the top of the program.

**A “%let” statement** is used to define a **MACRO VARIABLE** 🡺 not part of any dataset and can be used anywhere in your program after defining/creating 🡺 to utilize/reference a macro variable, you must put an “&” (without quotation marks) in front of the macro variable name (in this case, the macro variable name “root”)

-When SAS sees &*word*, SAS will check to see if the *word* is defined as a macro variable, and if it is, SAS will replace the macro variable with the defined text before running the code

-**YOU HAVE TO ENCLOSE THE USE OF A MACRO VARIABLE AND THE AMPERSAND (&) WITH “”, OTHERWISE, SAS WILL NOT ASSUME YOU’RE TRYING TO REFERENCE A MACROVARIABLE**

**-**In SAS, variable names (including macro variable names) are not case-sensitive (so, in this example, “&ROOT”, “&Root”, “&root”, and “&rOOt” will all call the same macro variable reference.)

-Macro variables must follow the same variable naming rules as normal (for example, 32 characters length limit, first character must be an English letter or underscore, other characters must be English letters/numbers/underscores)

-You can reassign macro variable references using an **%let** statement (or you could physically delete the reference from memory…more on that later)

SAS Options Statements

option mergenoby=nowarn ;\*nodate nonumber nocenter orientation=landscape;

-This options statement modifies the default output files that SAS produces

-**nodate** removes the date stamp printed on the files (that you typically see in the top right)

-**nonumber** removes the page number

-**nocenter** prevents the default behavior of centering the output (now, the default is left-justified)

-**orientation=landscape** makes the document landscape-oriented

-**mergenoby=nowarn** prevents a warning from appearing in your log when you have a particular **merge** statement in your code without a **by** statement (**more on what this means later**)

-THERE ARE MANY MORE OPTIONS, BUT THESE ARE COMMON ONES

-To undo these options, you have to submit a new options statement that resets the default (there isn’t a default statement)

-**date** restores the date

-**number** restores the page number

-**center** recenters the output

-**orientation=portrait** restores the portrait-orientation

-Typically, for options statements, **x** does something, and **nox** (no-x) undoes it

Proc Sort Duplicates

**\*\* simple code to generate a dataset;**

data clinic\_visits;

ID = 1; visit\_number = 1; visit\_date = '01Jan2018'd; output;

ID = 1; visit\_number = 2; visit\_date = '05Jan2018'd; output;

ID = 1; visit\_number = 3; visit\_date = '10Jan2018'd; output;

ID = 1; visit\_number = 4; visit\_date = '21Jan2018'd; output;

ID = 3; visit\_number = 1; visit\_date = '01Aug2018'd; output;

ID = 3; visit\_number = 2; visit\_date = '03Aug2018'd; output;

ID = 3; visit\_number = 2; visit\_date = '04Aug2018'd; output;

ID = 3; visit\_number = 3; visit\_date = '08Aug2018'd; output;

ID = 2; visit\_number = 1; visit\_date = '01Feb2018'd; output;

ID = 2; visit\_number = 2; visit\_date = '02Mar2018'd; output;

ID = 2; visit\_number = 3; visit\_date = '03Apr2018'd; output;

ID = 2; visit\_number = 4; visit\_date = '06May2018'd; output;

format visit\_date date9.;

run;

proc print data = clinic\_visits; run;

**\*\* sort the data by ID and then visit\_number;**

proc sort data = clinic\_visits out = clinic\_visits\_sorted;

by ID visit\_number;

run;

proc print data = clinic\_visits\_sorted; run;

**\*\* check for duplicate ID;**

proc sort data = clinic\_visits out = clinic\_visits\_sorted\_dups nodupkey;

by ID visit\_number;

run;

proc print data = clinic\_visits\_sorted\_dups; run;

-**nodupkey** removes duplicate values, categorized using the “by” statement (for example, in this case, it will remove entries/values that share the same ID AND visit number, and not if they only share one of those)

-So, it will excise duplicate values for each combination of variables in the **by** statement

**\*\* check for duplicate IDs and output duplicates;**

proc sort data = clinic\_visits out = clinic\_visits\_sorted\_dups dupout=dup\_recs nodupkey;

by ID visit\_number;

run;

proc print data = clinic\_visits\_sorted\_dups; run;

proc print data = dup\_recs; run;

-dup\_recs could be named anything you want that is valid by SAS rules

# Wednesday, September 5, 2018 – Lecture 4

Formatting is a mapping from one set of values to another (doesn’t just have to be for character values; works for numeric values too)

-Formats are character-specific 🡺 numeric formats can’t be applied to character variables and character formats can’t be applied to numeric variables

-Formats you can use are either ones that SAS gives you or ones you create

**-Q: Where can I find a list of SAS supplied formats?**

**-A: You can google "SAS 9.4 Formats and Informats: Reference"**

**Q: Can I permanently attach a format to a variable?**

**A: If you attach a format to a variable in a DATA step, the newly created**

**dataset retain the format association**

-If you place the format statement in the PROC step, the label is only temporarily created/assigned and will only last for that PROC step

-Labels/label statements operate under the same principle

-For dates, people tend to prefer the yymmdd10 format (YYYY-MM-DD) because it lets you sort dates in chronological order

\*\* this DATA step simply creates a dataset named "dates" that contains

one numeric variable named "date" with three observations;

data work.dates;

date = 0; output;

date = "28Jan1982"d; output;

date = today(); output;

run;

title "Print out of unformatted numeric date variables";

proc print data = work.dates noobs;

run;

title "Print out of formatted (date9. format) numeric date variables";

proc print data = work.dates noobs;

format date date9.;

run;

title "Print out of formatted (yymmdd10. format) numeric date variables";

proc print data = work.dates noobs label;

format date yymmdd10.;

label date = "Special Dates in American History"; \*\* temporarily attach a label;

run;

data work.dates2; \*\* dataset to create;

set work.dates; \*\* dataset to use as input;

format date yymmdd10.;

label date = "Special Dates in American History"; \*\* permanently attach a label;

run;

**Q: Can I create my own formats and use them in a PROC step?**

**A: You can create your own formats using the FORMAT procedure and make use**

**of them in any PROC step by temporarily attaching them to a variable using**

**a FORMAT statement;**

proc format;

\*\* format for a numeric variable;

value ageCatA *#(ageCatA is the name of the format 🡺 can name anything except you can’t start it with a number or space, and you can’t end it with a space)#*

0 - 65 = '<=65' *#(Dashes between numbers means inclusive)#*

65< - 150 = '>65'; *#(Values that fall outside of these ranges for the format will each be given its own category if its value is unique 🡺 for example, if you have two values like 152 and 162, you will get four categories: 0-65, 65<-150, 152, and 162; if you have two values of 153, you will get three categories: 0-65, 65<-150, 153)#*

*#(One benefit of setting defined, delimited categories like this is that it could be useful for detecting messy/bad data 🡺 if you’re sure of the range of values that that data could take)#*

\*\* mostly equivalent to the ageCatA. format -- when is it not;

value ageCatB

LOW - 65 = '<=65' *#(LOW essentially refers to negative infinity and HIGH essentially refers to positive infinity)#*

65< - HIGH = '>65';

*#(Note that the two categories together only have 1 semi-colon 🡺 all the categories/intervals are part of one statement)#*

\*\* format for a character variable;

value $ gender *#(The $ has to come before the name “gender” because you are formatting a character variable)#*

"M" = "Male"

"F" = "Female";

run;

\*\* creating a “catch-all” category for data that falls outside your main/intended categories;

value $ gender

"M" = "Male"

"F" = "Female"

other = “bad data”;

run;

proc freq data = echo.dm order=freq;

table age sex; *#(This will do two one-way frequency tables. If you wanted a two-way frequency table, you include an asterisk between the two variables.)#*

\*\* appropriate format statement for a numeric variable;

format age ageCatA.; *#(The period after the format name is necessary when formatting a variable.)#*

\*\* appropriate format statement for a character variable;

format sex $gender.;

run;

\*\* can use a single format statement for multiple assignments;

format age ageCatA. sex $gender.;

run;

*#(format [variable] [format name]. [variable 2] [format name 2]. …)#*

\*\* notes:

[1] Permanently attaching user-defined formats to temporary datasets (i.e., datasets in

the work library) is a good practice if the format is to be used multiple times.

[2] Permanently attaching user-defined formats to permanent datasets (i.e., datasets that

may be shared with collaborators) is not a good practice because one would need to

provide the format catalog with the dataset for users to be able to use the format

and even view the data!;

data echo.DM\_FORMAT\_ATTACHED\_DELETE;

set echo.DM;

format age ageCatB. sex $gender.;

run;

# Monday, September 10, 2018 – Lecture 5

This program is designed to help students understand: how to create datasets storing summary results from PROC MEANS, PROC UNIVARIATE, and PROC FREQ (without using ODS)

\*\* Notes:

[1] Any ODS object can be turned into a SAS dataset by delivering it to the ODS OUTPUT

destination

[2] Most SAS procedures also allow certain results to be written to a more user-friendly

SAS dataset using PROC-specific options/statements

[3] Most of the time, if you want basic summary results in a data set,

option [2] is better than [1]

*\*\* write the one-way frequency table ODS object into a SAS dataset;*

ods output OneWayFreqs = work.A; */\*The name of the object (ODS object) you want to make into a dataset (on the left) =*

*the name of the library you want to store your SAS dataset in and your SAS dataset name (on the right)\*/*

proc freq data = echo.DM;

table armcd ;

run;

proc print data = work.A noobs ; run;

**Key takeaway**: ODS output statement takes a procedure’s default output and turns/writes it into a SAS dataset (use ODS trace or Google to find the name of the ODS object because you need the specific name)

-ALSO, options that affect the ODS output will affect the dataset created from it (**so, you can add options to modify the ODS output to also modify your dataset**)

*\*\* write the one-way frequency data into a SAS dataset with the (PROC FREQ specific) OUT= option on the TABLE statement;*

*\*\* note that the NOPRINT option is used to suppress \*ALL\* ODS output that would be produced by the procedure;*

proc freq data = echo.DM noprint;

table armcd / out = work.B;

run;

proc print data = work.B noobs; run;

**Key takeaway:** Most procedures have a built-in method for writing/turning the output into a SAS dataset, other than using the ODS output statement (and in general, if you are able to use this built-in method, do so, because the resulting dataset is more user-friendly; also, the built-in method tends to produce a more concise dataset; **the datasets produced by the two methods are not equivalent**)

*\*\* write the five number summary ODS object from PROC MEANS to an output dataset;*

ods output summary=work.E;

proc means data = echo.DM n nMiss mean std min max fw=2;

class armcd;

var age;

run;

proc print data = work.E noobs ; run;

*\*\* write the five number summary from PROC MEANS to an output dataset using the OUTPUT statement;*

*\*\* this code works verbatim for PROC UNIVARIATE;*

proc means data = echo.DM noprint;

class armcd;

var age;

output out = work.F n=sampleSize nMiss=numMiss mean=meanVal std=stdDev min=minimum max=maximum ;

run;

proc print data = work.F noobs ; run;

**#After the output statement, you have to put out = (name of the SAS dataset you want to create). Then, you can rename the column names in the new data set (so, n in the five-number summary becomes “sampleSize” for example)**

#**When you have a class statement in PROC MEANS or PROC UNIVARIATE, SAS will by default give another row that summarizes/accumulates the numbers/values for each member of that class/those classes (so, SAS will by default give a summary values row for the entire group)**

*\*\* using the AUTO\_NAME option on the OUTPUT statement to name variables;*

*\*\* PROC UNIVARIATE does not support AUTO\_NAME;*

proc means data = echo.DM noprint;

class armcd;

var age;

output out = work.G n= nMiss= mean= std= min= max= / autoname;

run;

proc print data = work.G noobs; run;

#The /autoname option lets SAS name your variables using a “reasonable” option/metric

# Wednesday, September 19, 2018 – Lecture 6

Subsetting Observations

*Code Snippet:*

*\*\* to begin this lab, we will read in a subset of the observations from the ECHO*

*trail DM dataset;*

*data work.DM;*

*set echo.DM(obs=10);*

*drop studyid ageu vis: arm rf: dmdtc;*

*run;*

*title "Print out of work.DM";*

*proc print data = work.DM; run;*

*\*\* Note: using vis: in the drop statement drops all variables whose name starts with the "vis" prefix;* ***a similar principle applies with the rf: in the drop statement***

*Code Snippet 2:*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Task 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*[1] Create a temporary dataset WORK.DM2x.*

*[2] Remove all observations where age <= 50 and race is not equal to white.*

*[3] Compare approaches based on subsetting IF and WHERE statements;*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*\*\* attempt 1;*

*data work.DM2a;*

*set work.DM;*

*where age <= 50 and* ***upcase(RACE) ^= 'WHITE'****;*

*\*\* The WHERE statement can only use variables that exist in all*

*input data sets listed on the SET statement;*

*run;*

*title "Print out of work.DM2a";*

*proc print data = work.DM2a; run;*

**BOLD is an example of defensive programming 🡺 in the dataset, all the races are in all-caps anyway, so the upcase(RACE) is not necessary, but it might be important if you were working with more free-text data**

**Numeric values that are missing are treated by SAS as smaller than any non-missing number 🡺 this statement will (age <= 50) will allow missing values in**

*\*\* attempt 2 - more defensive programming;*

*data work.DM2b;*

*set work.DM;*

***where . < age <= 50*** *and upcase(RACE) ^= 'WHITE';*

*\*\* using a compound inequality to exclude missing age values;*

*run;*

*title "Print out of work.DM2b";*

*proc print data = work.DM2b; run;*

**This new statement for age in the WHERE statement will allow you to exclude missing values for age 🡺 KEEP THIS IN MIND**

**The RACE ^= statement will also keep missing values for RACE (as a character variable, the missing variable would be a blank space). If you want to be more defensive, you could add *and RACE ^= “ ”* *or you could do and upcase(RACE) not in (‘WHITE’ “ ”)***

**WHERE statements are used to determine which data observations are read in the first place**

**-This is done in your computer’s RAM**

**-WHERE statements are more efficient and take less time, so they are advantageous with large datasets**

*\*\* attempt 3 - subsetting IF approach (for pedagogical purposes);*

*data work.DM2c;*

*set work.DM;*

*if (. < age <= 50 and upcase(RACE) ^= 'WHITE');*

*run;*

*title "Print out of work.DM2c";*

*proc print data = work.DM2c; run;*

**Subsetting IF statements will read all data observations, but will only keep the ones that evaluate TRUE for the IF statement**

**-The removal of data with subsetting IF statements is done on your hard drive**

**-Subsetting IF statements are advantageous because you can use variables that don’t exist in all the input datasets 🡺 because it is executed as the data is processed**

*\*\* attempt 4 - subsetting IF approach (for pedagogical purposes);*

*data work.DM2d;*

*set work.DM;*

*ageCat = ( . < age <= 50);* **Binary; evaluated as Boolean expression 🡺 ageCat = 1 if the corresponding age fits in this range, = 0 if it doesn’t**

*include\_obs = (* ***ageCat*** *and upcase(RACE) ^= 'WHITE');*

*\*\* does this look odd?;*

**Here, just saying ageCat here is equivalent to saying ageCat=1, but it is better practice to say ageCat=1**

**-This is partly because SAS will treat anything bigger than 0 as TRUE, and 0 is FALSE**

*if include\_obs;*

**Same deal as above with ageCat; best practice is “if include\_obs=1;”**

*\*\* QUESTION: How do I get rid of the ageCat varible and in the include\_obs variable*

*which were only created to facilitate subsetting the observations for the new work.DM2d dataset?;*

**Drop statement 🡺 drop ageCat include\_obs**

*run;*

*title "Print out of work.DM2d";*

*proc print data = work.DM2d; run;*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Task 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*[1] Analyze the echo.DM dataset to compute the mean age for whites.*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*title1 "Summary of Age for Echo Trial Subjects";*

*title2 "Race = White";*

*proc means data = echo.DM;*

*where RACE = 'WHITE';*

*var age;*

*run;*

**WHERE statements can be used for any PROC statements (or just about anywhere)**

*title1 "Summary of Age for Echo Trial Subjects";*

*title2 "Race = White";*

*proc means data = echo.DM;*

*if RACE = 'WHITE'; \*\* note the color -- this code does not work;*

*var age;*

*run;*

**Subsetting IF statements can only be used with data steps**

**-You will get a “Error: Statement is not valid or it is used out of proper order.”**

Conditional Logic

*Code Snippet:*

*\*\* to begin this lab, we will read in a subset of the observations from the ECHO*

*trail DM dataset;*

*data work.DM;*

*set echo.DM( where=(age<45) ); \*\* example of using a WHERE dataset option;*

*drop studyid ageu vis: arm rficdtc dmdtc;*

*run;*

*title "Print out of work.DM";*

*proc print data = work.DM; run;*

*\*\* Note: using vis: in the drop statement drops all variables whose name starts with the "vis" prefix;*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Task 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*[1] Create a temporary dataset WORK.DM2a.*

*[2] Create a variable aCAT that has values "White:age<40","White:age>=40","Non-white:age<40","Non-white:age>=40"*

*[3] Create a variable aCATN that has values 1 ,2 ,3 ,4*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*\*\* attempt 1;*

*data work.DM2a;*

*set work.DM;*

*label aCat = 'Race/Age Category'*

*aCatn = 'Race/Age Category (Numeric)';*

*\*\* code not perfect, need to correct and/or clean up.;*

*if age < 40 and upcase(RACE) = 'WHITE' then aCat = "White:age<40";*

*else if upcase(RACE) = 'WHITE' then aCat = "White:age>=40";*

*else if age < 40 and upcase(RACE) ^= 'WHITE' then aCat = "Non-White:age<40";*

*else if upcase(RACE) ^= 'WHITE' then aCat = "Non-White:age>=40";*

*if age < 40 and upcase(RACE) = 'WHITE' then aCatn = 1;*

*else if upcase(RACE) = 'WHITE' then aCatn = 2;*

*else if age < 40 and upcase(RACE) ^= 'WHITE' then aCatn = 3;*

*else if upcase(RACE) ^= 'WHITE' then aCatn = 4;*

*run;*

*\*\* Note: the symbol ^= in SAS means not equal to. Other languages use !=.;*

aCat **variables are truncated 🡺 need a length statement to assign variable lengths yourself in this case 🡺 the default SAS will give is the first value SAS sees for that variable (in this case, the length of White:age<40)**

**Also, there are missing age values (. < age < 40)**

*title "Print out of work.DM2a";*

*proc print data = work.DM2a; run;*

*\*\* Question: How do I make the labels for aCat and aCatn (and other variables*

*show up?;*

# Monday, September 24, 2018 – Lecture 7

# Wednesday, September 26, 2018 – Lecture 8

# Monday, October 1, 2018 – Lecture 9 and Wednesday, October 3, 2018 – Lecture 10

*option mergenoby=****error*** *nodate nonumber nobyline;*

*ods noproctitle;*

*title;*

*footnote;*

*%let root = C:\Users\linra\Documents\UNC Masters Degree\Fall 2018\BIOS 511\BIOS-511-FALL-2018-master;*

*%let dataPath = &root.\programs\2018-10-01-lecture-09;*

*libname lecture "&dataPath";*

*%let path=C:\Users\linra\Documents\UNC Masters Degree\Fall 2018\BIOS 511\Lab Data\echo;*

*libname echo "&path";*

**Truncated dataset containing vital signs (in this case, heart rate) for just two patients at the same type (but multiple visits per patient)**

*\*\* Task #1: Merge on treatment group information (ARMCD) from the DM dataset;*

*\*\* [1] The new dataset (WORK.VS2) should only have observations matching those in the WORK.VS dataset.*

*That is, we should not keep observations from DM that have no match in the WORK.VS dataset.;*

*/\* Question: Conceptually... what do we need to do to achieve this goal?*

*[1]* **Identify variables that allow matching observations (USUBJID)**

*[2]* **Make sure both datasets are sorted by those variables (could be more than one)**

*[3]* **Write a DATA step that merges the two sources and keeps the observations you want in the newly created dataset**

*\*/*

**Don’t take for granted that your data is sorted. To be safe, write a PROC SORT step.**

*proc sort data = lecture.VS out = work.VS;*

*by usubjid;*

*run;*

**The WORK.VS statement is in long format (multiple measurements of a particular variable for each person; each measurement of that variable is an observation) 🡺 in general, few variables; more observations 🡺 generally, data storage in the long format is preferred**

**-Wide format is the opposite (one observation per person; each measurement of that variable is its own variable)**

*proc sort data = echo.dm(keep=usubjid armcd) out = dm;*

*by usubjid;*

*run;*

*data work.VS2;*

*merge work.VS****(in=A)*** *work.dm****(in=B)****;*

*by usubjid;*

***putlog “ BEFORE ” \_all\_;***

***if (A=1 and B=1);***

***putlog “ AFTER ” \_all\_;***

*run;*

**If you don’t include a BY statement in merging, SAS will merge the two (or more datasets) line by line 🡺 if one/some of the datasets is longer (so, the datasets do not have equal numbers of rows), the last row/observation of the shortest dataset will be merged with each of the remaining observations of the other datasets 🡺 this almost always produces a nonsensical result**

**-This leads us to the “option mergenoby” 🡺 the default is “nowarn”, meaning SAS will not warn you if you merge without a BY statement; from now on, change “nowarn” to “error” 🡺 SAS will terminate the process if you merge without a BY statement**

**-This also means that option mergenoby=nowarn doesn’t actually do anything, since it is SAS’ default behavior**

**11 observations read from the dataset WORK.VS**

**602 observations read from the dataset WORK.DM**

**WORK.VS2 has 611 observations and 8 variables**

**-Why 611 observations and not 602+11? Because in the WORK.DM dataset, there are 2 observations that represent the two people in WORK.VS 🡺 so 602-2+11=611**

**-(in=A) and (in=B) 🡺 variables don’t really matter, since they are only used for this DATA step; they will not be put into the dataset you create 🡺 they are temporary variables and you can, for example, use subsetting IF statements with them**

**-Note that you cannot use subsetting WHERE statements, since the variables A and B don’t exist in the input datasets before processing**

**-IF statements cannot be used in a PROC step (because you can’t create variables in those steps) 🡺 if you can use WHERE, you should because it is more efficient/faster**

**-A has a value of 1 if the resulting observation is in WORK.VS and has a value of 0 otherwise**

**-B is the same, but with WORK.DM**

**-if (A=1 and B=1) will only include observations in the output merged dataset if they exist/come from both input datasets (i.e.: in this case, that means both datasets have that USUBJID)**

**-The “putlog” code is just there to let you see the values of A and B for each observation as SAS does the merging (don’t include this sort of thing in your actual code)**

*\*\* Task #2: Construct a dataset (WORK.HR[x]) that has all HR values on a single observation for each subject.*

*\*\* [1] The variables in this dataset should be named: usubjid armcd SCR WK00 WK08 WK16 WK24 WK32 (or something similar);*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\* Strategy #1 -> Make a dataset of results for each visit and merge them all together. \*/*

**Need tools to convert between long and wide data formats**

*data SCR WK00 WK08 WK16 WK24 WK32;*

*set work.VS2;*

*if upcase(VISIT) = 'SCREENING' then output SCR;*

*else if upcase(VISIT) = 'WEEK 0' then output WK00;*

*else if upcase(VISIT) = 'WEEK 8' then output WK08;*

*else if upcase(VISIT) = 'WEEK 16' then output WK16;*

*else if upcase(VISIT) = 'WEEK 24' then output WK24;*

*else if upcase(VISIT) = 'WEEK 32' then output WK32;*

*keep usubjid armcd vsstresn;*

*run;*

**Split my one dataset into six datasets (one-to-one 🡺 take six variables, split into six datasets 🡺 each resulting dataset doesn’t take on more than one of those six original variables)**

*\*\* code to print out all the datasets;*

*proc print data = work.SCR; run;*

*proc print data = work.WK00; run;*

*proc print data = work.WK08; run;*

*proc print data = work.WK16; run;*

*proc print data = work.WK24; run;*

*proc print data = work.WK32; run;*

*data work.HRa;*

*merge SCR(rename=(VSSTRESN=SCR))*

*WK00(rename=(VSSTRESN=WK00))*

*WK08(rename=(VSSTRESN=WK08))*

*WK16(rename=(VSSTRESN=WK16))*

*WK24(rename=(VSSTRESN=WK24))*

*WK32(rename=(VSSTRESN=WK32));*

*by usubjid;*

*run;*

***/\*This DATA step simply changes the order in which the variables are stored in the dataset. The FORMAT statement we use must come before the SET statement to change the way/order SAS will encounter the data. This FORMAT statement will also eliminate any formats on the variables, but we don't have any right now. In addition, we can also create and format new variables that don't exist in the initial dataset (see the addition of the variable test, which has been commented out). We can also do the same thing using a LENGTH statement that keeps the length of each variable the same as it currently is (basically, we can use any statement that defines an attribute for a variable and orders the variables, but FORMAT and LENGTH are the easiest.)\*/***

*data HRa;*

*format usubjid /\*test\*/ armcd scr wk:;*

*set HRa;*

*\*test="CHECK";*

*run;*

*proc print data = work.HRa;*

*var usubjid armcd scr wk:;*

*run;*

**MISTAKE: Basically, when merging, the values of the dataset to the right overwrite the values of the dataset to the left 🡺 the result just gives you the data for WK32**

**-You need to rename the datasets as they are being merged (so that SAS doesn’t overwrite them)**

**-When printing out the dataset, here we use the VAR statement to change the order in which the variables are printed out (but this doesn’t change the order in which the variables are actually stored)**

**-wk: selects for all variables that begin with “wk” – Not case-sensitive**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\* Strategy #2 -> Use an ARRAY /w RETAIN and conditional OUTPUTs statements in a single data step \*/*

*data work.HRb;*

*set work.VS2;*

*by usubjid; \*Needed to get first.usubjid and last.usubjid;*

*retain SCR WK00 WK08 WK16 WK24 WK32;*

***/\*As data are processed by SAS, you can only access one observation at a time, more-or-less.***

***SAS processes row-by-row. It is nearly impossible to pull a later observation and put it on an earlier row.***

***Instead, we pull observations down (from earlier rows to later rows.)***

***The RETAIN statement is critical because we need SAS to keep observations from earlier rows***

***(to pull down to later rows) until we tell SAS to change them otherwise.\*/***

*array hr[6] SCR WK00 WK08 WK16 WK24 WK32; \*\* note this array CREATES variables that do not exist;*

*if first.usubjid=1 then do i = 1 to 6;*

*hr[i]=.;*

*end;*

***/\*Defensive programming to prevent data from one subject from being pulled down to another subject (if there was missing data, for example).\*/***

*\*\* what does this code do?;*

*if visitnum = -1 then arrayID = 1;*

*else arrayID = visitnum + 1;*

*hr[arrayID] = vsstresn;*

*\*\* why is this conditional output statement necessary;*

*if last.usubjid then output; \*Could also do if last.usubjid=1;*

***/\*We can identify the last time we see a subject id (which, in this case, should be the row that contains all the blood pressure measurements for each meeting) and tell SAS to drop the other rows (which contain partially-filled, redundant data).\*/***

*keep usubjid armcd SCR WK00 WK08 WK16 WK24 WK32;*

*run;/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\* Strategy #3 -> Use PROC TRANSPOSE (always nice when it can be used) \*/*

*proc transpose data = work.VS2 out = work.HRc;*

*by usubjid; \*\* defines the observations in the new data set;*

*id visit; \*\* defines the columns in the new data set;*

*var vsstresn; \*\* defines the values of the variables (columns) in the new data set;*

*run;*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*data work.VS "C:\Users\psioda\Documents\GitHub\BIOS-511-FALL-2018\programs\2018-10-01-lecture-09\data\vs";\*/*

*/\* set echo.VS;\*/*

*/\* where usubjid in ('ECHO-011-001' 'ECHO-011-002');\*/*

*/\* where also vstestcd in ('HR');\*/*

*/\*\*/*

*/\* if usubjid = 'ECHO-011-002' and visitnum = 4 then delete;\*/*

*/\*\*/*

*/\* keep usubjid vstest: vsstresn vsstresu visit:; \*/*

*/\*run; \*/*

*/\*proc sort; by visitnum; run;\*/*

*/\*proc print noobs ; run;\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

# **Monday, October 8, 2018 – Lecture 11**

See SAS Program for Lecture 11

# **Monday, October 15, 2018 – Lecture 12**

Data formats that we will commonly work with:

List format 🡺 Each data value of an observation is followed by a delimiter (which could be anything, including a space) 🡺 probably the most common data format you will receive 🡺 when setting a delimiter, try not to choose a delimiter that appears in your data values of the observations

-Space delimiters

-Tab delimiters (often looks like column format, at least somewhat, but it’s not column format)

-Comma delimiters 🡺 most common type of delimiter now (.csv)

Column format 🡺 Data structured in columns; there is a column index (that is not part of the data, but can be used to select columns) 🡺 A single data value can occupy multiple columns

Excel format 🡺 Techniques we will need to use are different compared to text files, since Excel files aren’t text files

.dat files are typically just named so to indicate that they are data files; they can just as easily be opened with a basic text editor like Notepad; you can change the extension to .txt if you so wanted

-Most of the time, when you get data files, you can open with a text editor (except for when you get Excel files); the extension usually doesn’t matter for what you’re doing (it’s just there to provide extra information at a glance)

PROC import is a black box method to import data from other formats

A DATA step lets you have more control

-To import data from other formats in a DATA step, you need at least two lines:

-An “infile” line, which says where the data file is

-An “input” line, which says how the data is to be read in

*title1 "Print out of CLASS\_COLUMN\_FORMAT.DAT";*

*\*\* reading in data in COLUMN format;*

*\*\* using $ sign ensures variables are created as character;*

*data classA;*

*infile "&rawPath./class\_column\_format.dat";*

*input Name $ 1-10 Age 12-14 Sex $ 16 Height 18-22 Weight 23-29;*

*run;*

**For this input line, SAS knows that all the data for the Name variable will be in columns 1-10 ($ indicates a character variable)**

**-The ranges specified in this input line cannot overlap and have to be disjoint**

**-The ranges also don’t have to be exact (for example, Name goes from columns 1-10, even though none of the names are that long) 🡺 as long as the column ranges contain all the data you want and don’t overlap with other variables, you’re good**

**-Column format doesn’t have delimiters, so you pretty much have to count columns to determine the ranges 🡺 so, this gets annoying for large datasets, so those are typically not in column format**

*proc print data = classA; run;*

*ods select Variables;*

*proc contents data = classA; run;*

*\*\* reading in data in COLUMN format assuming it is in LIST format;*

*\*\* using length statement;*

*data classB;*

*infile "&rawPath./class\_column\_format.dat" dlm=" ";*

*length Name $20 Sex $5;*

*input Name 1-10 Age 12-14 Sex 16 Height 18-22 Weight 23-29;*

*run;*

**Another way to read in data in Column format using a LENGTH statement. The lengths can really be anything and do not have to agree with the column indices (though it’s logical to). Since the length statement already states that Name and Sex are character variables (the $), the input statement does not need to do so.**

**The dlm=“ ” bit doesn’t do anything here, since our data isn’t in List format. The dlm = “ ” is also the default, so even if you don’t include it, it will be there.**

*\*\* example to show how not using COLUMN format will break with missing data;*

*data classC;*

*infile "&rawPath./class\_column\_format\_missing.dat" dlm=" ";*

*input Name $ Age Sex $ Height Weight;*

*run;*

*title1 "Print out of CLASS\_COLUMN\_FORMAT\_MISSING.DAT";*

*proc print data = classC; run;*

**The dlm = “ ” also makes it so that we won’t have to specify column ranges, since SAS will read along until it finds the delimiter.**

**For Alice, the data read in is missed up because of missing values (for AGE). When read in using this code, SAS reads a bunch of spaces (which we set as the delimiter) after “Alice”, and SAS will collapse the consecutive delimiters into one. The next non-delimiter value it sees is “F”, which SAS takes as the second data value. However, the second variable we have is AGE, which is a numeric variable (no $.) SAS will attempt to put a character into a numeric variable, which will print out a note that this cannot be done (however, this won’t print a WARNING or ERROR in the log. Check the MSGLEV = option to see if this can be changed). Thus, the value of AGE for Alice is missing.**

**-The point: When reading in Column format data, read it in as if it was Column format. Don’t try to assume it is in List format. That will only will work if there aren’t missing data values.**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LIST FORMAT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*\*\* space delimiter;*

*data classD;*

*infile "&rawPath./CLASS\_LIST\_FORMAT\_SPACE.TXT" dlm=" " firstobs=2;*

*length Name $10 Sex $5;*

*input Name Age Sex Height Weight;*

*run;*

*title1 "Print out of CLASS\_LIST\_FORMAT\_SPACE.TXT";*

*proc print data = classD; run;*

**The firstobs =2 option tells SAS to start reading from the second row (which in this case, is a header). For now, we can’t transfer the names of the variables from the header into the names of the variables in the dataset we are creating.**

*\*\* TAB delimiter;*

*data classE;*

*infile "&rawPath./CLASS\_LIST\_FORMAT\_TAB.DAT" dlm="09"x firstobs=2;*

*length Name $10 Sex $5;*

*input Name Age Sex Height Weight;*

*run;*

*title1 "Print out of CLASS\_LIST\_FORMAT\_TAB.DAT";*

*proc print data = classE; run;*

**In UNICODE, Tab is represented by “09”x or ‘09’x. This is the hexadecimal representation of Tab. You’ll just have to memorize this. Tab is not the same as whitespace.**

*\*\* COMMA delimiter;*

*data classF;*

*infile "&rawPath./CLASS.CSV" dlm="," firstobs=2;*

*length Name $10 Sex $5;*

*input Name Age Sex Height Weight;*

*run;*

*title1 "Print out of CLASS.CSV";*

*proc print data = classF; run;*

*\*\* Using PROC IMPORT;*

*proc import datafile = "&rawPath./CLASS\_LIST\_FORMAT\_TAB.DAT" out = classG dbms=dlm replace;*

*delimiter="09"x;*

*run;*

*title1 "Print out of CLASS\_LIST\_FORMAT\_TAB.DAT";*

*proc print data = classG; run;*

# **Wednesday, October 17, 2018 – Lecture 13**

Continuation of program from Monday, Lecture 12 => finished in like 20 minutes

See also the SGPLOT eLearning.sas program in the programs folder for 2018-10-17-Lecture 13

# **Monday, October 22, 2018 – Lecture 14**

Covered the default ODS graphics 🡺 See programs folder for 01-default-ODS-graphics.sas (2018-10-22-Lecture 14 folder)

# **Wednesday, October 24, 2018 – Lecture 15**

Three programs in today’s lecture folder 🡺 further customizing graphs

Style Templates

In SAS Studio, you (probably) cannot change the style of the output in the Results viewer/window; you can only change the style of the physical file you create

The style called “default” for the pdf **destination** in SAS is actually not the default style for pdf outputs

The “journal” style is greyscale, while the “sasweb” style is what the SAS Results viewer shows you (I think)

You can google a list of style templates (there is also code you can run to list all the available style templates – should look it up)

Styleattrs statement

Different types of graphs have different style attributes, and those style attributes can have different keywords describing/controlling them 🡺 look in the documentation

-Example syntax:

styleattrs dataColors=(blue black orange)

Here, **dataColors** is a keyword (used for a bar chart here), then you use “=” and the values of the keyword in parentheses 🡺 for colours, you can also use RGB hexadecimal representations

-Colours here are assigned in the order of the levels of the variable(s) that you use for your group (for example, if you have a Country variable, with values “Canada”, “Mexico”, and “USA”, the order used is alphabetical order of the values of Country 🡺 blue would go with “Canada”, black would go with “Mexico”, and orange would go with “USA”)

For funsies: If you create a new variable using the CATX function, and you don’t specify the length, the default length is 200 characters

You’ll probably want to use ods graphics / noborder if you’re ever making a graphic that you intend to paste into another type of document

**dataContrastColors** controls the boxing/outline of objects in the graph (so you could, for example, use this option to change the outline of objects to match their fill) 🡺 I think this is what it does, but not 100% percent sure

When you have a discrete axis, you can/have to adjust the order of the values of that variable plotted along that axis by using PROC sort on the data set and sorting by that variable

*ods graphics / attrpriority=none;*

*title "Example 3: Age Distribution by Country and Treatment Group";*

*proc sgplot data = dm;*

*styleattrs dataContrastColors=(red purple) datasymbols=(diamondFilled squareFilled);*

*scatter x=cnt\_arm y=age / group=armcd jitter markerattrs=(size=5);*

*yaxis label = 'Age (years)';*

*xaxis label = 'Country / Treatment Group' splitchar='-' fitpolicy=splitalways;*

*run;*

*ods graphics / reset=all;*

Without the “attrpriority = none” statement, all your objects would be diamonds and not squares. **The default behavior of SAS is to cycle through colors first.** Since there are only two groups (ECHOMAX and PLACEBO), the first group would get a red diamond, then the second group would get a purple diamond. Then, since it goes back to the first group, it would get a red diamond again, then a purple diamond. With “attrpriority=none” turned on, SAS will cycle through all of the attributes as you go through the groups. **The only options for “attrpriority” are “color” or “none.”** If you had more than two groups, then after going through the first two groups (and cycling through the two values of color), then SAS would cycle through the two values of “datasymbol”.

# **Monday, October 29, 2018 – Lecture 16**

sgPanel 🡺 see Monday, October 29, 2018 programs folder

# **Wednesday, October 31, 2018 – Lecture 17**

See Wednesday, October 31, 2018 programs folder (01-SGSCATTER.sas)

*proc contents data=sashelp.iris; run;*

*\* http://support.sas.com/documentation/cdl/en/grstatproc/62603/HTML/default/viewer.htm#sgscatter-stmt.htm;*

*\*[1] The MATRIX statement;*

*\*Creates a scatter plot matrix; \*Diagonal option adds more useful details;*

*\*Creates a n x n matrix, where n is the number of variables;*

*proc sgscatter data=sashelp.iris;*

*matrix petallength petalwidth sepallength sepalwidth / diagonal=(histogram kernel);*

*run;*

*\*[2] The PLOT statement;*

*\*Creates a paneled graph that contains multiple independent scatter plots (y vs x ==> y\*x);*

*\*Could do other methods of fitting, like pbspline (and could put multiple at once);*

*\*Here we can do 4 different scatter plots at once (with sepallength and sepalwidth as y and petallength and petalwidth as x);*

*proc sgscatter data=sashelp.iris;*

*plot (sepallength sepalwidth) \* (petallength petalwidth) / reg pbspline;*

*run;*

*\*[3] The COMPARE statement;*

*\* Creates a comparative panel of scatter plots with shared axes;*

*\*Linear regression here done for each group;*

*proc sgscatter data=sashelp.iris;*

*compare x=(sepallength petallength) y=(sepalwidth petalwidth)/ group=species reg spacing=14;*

*run;*

*%let dataPath = /folders/myfolders/;*

*libname echo "&dataPath." access=read;*

*proc sort data = echo.LB out = LB;*

*by usubjid;*

*run;*

*proc sort data = echo.DM out = DM;*

*by usubjid;*

*run;*

*data merged;*

*merge LB(in=a) DM(in=b);*

*by usubjid;*

*if a and b;*

*run;*

*proc contents data=merged; run;*

*proc sgscatter data=merged;*

*matrix age LBSTRESN;*

*run;*

# **Wednesday, November 7, 2018 – Lecture 18**

Class cancelled Monday, November 5, 2018 (b/c water main break in Chapel Hill)

Macros

SAS macros are about text substitution 🡺 make a template and can use macros to substitute in the appropriate text so that the program actually runs (without you having to copy-paste the template code so many times)

01-Simple-Macro-Program

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Basic Use of Macro Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*%let root = C:\Users\linra\Documents\UNC Masters Degree\Fall 2018\BIOS 511; \*\* define the ROOT macro variable;*

*%let dataPath = &root.\Lab Data\echo; \*\* use TEXT SUBSTITUTION to define the DATAPATH macro variable;*

* **%let statement lets you manually define a macro variable (we’ve been doing this all along to shorten path names) 🡺 here the macro variable “root” does not have a type 🡺 this step does not process any data or anything like that**
* **Referencing a macro variable requires &<macro\_variable\_name> (sometimes, a period after the macro variable name is required) 🡺 here, we use &root.**
* **Macro variable substitutions occur first before code is processed**

*%put NOTE: dataPath = &dataPath.; \*\* write value of DATAPATH to SAS log;*

*%put NOTE: &=datapath.; \*\* short hand;*

* **%put statements writes macro variable values to the SAS log. Here SAS would replace &dataPath. with the actual value of the macro value dataPath. You can write anything in a %put statement (“NOTE: dataPath =” is just something Dr. Psioda wrote to make it easier to identify the statement in the log.) “NOTE:” tells SAS to colour the text blue in the SAS log, like how notes normally are. You could use “ERROR:” to colour the text red, or “WARNING:” to colour the text yellow (or green?).**

*libname echo "&dataPath"; \*\* use TEXT SUBSTITUION to define the ECHO libref;*

**Macro variable references can be used anywhere. Here the period (.) is not used after the macro variable reference**

**“WARNING: Apparent symbolic reference xxxxxx not resolved.” is the warning you will receive on the SAS log if you try to use a macro variable reference that wasn’t previously defined. Often, this means that you mistyped the name of your macro variable.**

**To try to resolve a macro variable reference, you MUST place the macro variable name in double quotes and NOT single quotes. This is because if sometimes you want to use & as an actual character (and not a macro variable reference indicator), you have to put & in single quotes.**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Brute Force Code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*\*\* Analyze Country Variable \*\*\*/*

*proc freq data = echo.DM noprint order=freq;*

*table country / out = countryDist;*

*run;*

*title1 "Number and Percent of ECHO Trial subjects by Country";*

*proc print data = countryDist label noobs;*

*label count = "n";*

*label percent = "%";*

*format percent 6.2;*

*var country count percent;*

*run;*

*/\*\*\* Analyze Sex Variable \*\*\*/*

*proc freq data = echo.DM noprint order=freq;*

*table sex / out = sexDist;*

*run;*

*title1 "Number and Percent of ECHO Trial subjects by Sex";*

*proc print data = sexDist label noobs;*

*label count = "n";*

*label percent = "%";*

*format percent 6.2;*

*var sex count percent;*

*run;*

*/\*\*\* Analyze Race Variable \*\*\*/*

*proc freq data = echo.DM noprint order=freq;*

*table race / out = raceDist;*

*run;*

*title1 "Number and Percent of ECHO Trial subjects by Race";*

*proc print data = raceDist label noobs;*

*label count = "n";*

*label percent = "%";*

*format percent 6.2;*

*var race count percent;*

*run;*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Simple Macro w/o Parameters \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*%macro freqTab1;*

*proc freq data = echo.DM noprint order=freq;*

*table &var1. / out = &var1.Dist;*

*run;*

*title1 "Number and Percent of ECHO Trial subjects by &var1.";*

*proc print data = &var1.Dist label noobs ;*

*label count = "n";*

*label percent = "%";*

*format percent 6.2;*

*var &var1. count percent;*

*run;*

*%mend freqTab1;*

* **Every SAS macro definition starts with %macro <macro\_name> and ends with %mend <macro\_name>**
  + **A macro definition is simply a template that will have values substituted in**
  + **When you have SAS run this code, SAS only defines this template and associates it with a name (freqTab1) 🡺 this is called compiling the macro**
  + **A SAS macro can be thought of/is a type of macro function**
* **Here, &var1.Dist must use a “.” because otherwise, SAS will think the macro variable is &var1Dist. A “.” is not a character that can be part of a macro variable name, so this “.” tells SAS, when it encounters “.” that the macro variable ends right before it.**
  + **So, the “.” is used to indicate to SAS where a macro variable name ends**

*%let var1 = Country;*

*%freqTab1;*

*%let var1 = Sex;*

*%freqTab1;*

*%let var1 = Race;*

*%freqTab1;*

* **This is how you execute a macro 🡺 “%let var1 = ” defines the macro variable that you included in your macro template; “%freqTabl1” calls the macro template and executes it 🡺 when used together, SAS will execute this macro and use your defined macro variable for text substitution in the template**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Simple Macro w Positional Parameter\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*%macro freqTab2(var2);*

*proc freq data = echo.DM noprint order=freq;*

*table &var2. / out = &var2.Dist;*

*run;*

*title1 "Number and Percent of ECHO Trial subjects by &var2.";*

*proc print data = &var2.Dist label noobs ;*

*label count = "n";*

*label percent = "%";*

*format percent 6.2;*

*var &var2. count percent;*

*run;*

*%mend freqTab2;*

* **Here, var2 is a parameter (a positional parameter, because it has no default value 🡺 the user must provide the default value when executing the macro)**

*%freqTab2(Country);*

*%freqTab2(Sex);*

*%freqTab2(Race);*

* **When the macro is executed here, the text “Country” is substituted into the template everywhere SAS sees &var2 (the name of the macro variable)**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Simple Macro w Non-positional Parameter\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*%macro freqTab3(var3=armcd);*

*proc freq data = echo.DM noprint order=freq;*

*table &var3. / out = &var3.Dist;*

*run;*

*title1 "Number and Percent of ECHO Trial subjects by &var3.";*

*proc print data = &var3.Dist label noobs ;*

*label count = "n";*

*label percent = "%";*

*format percent 6.2;*

*var &var3. count percent;*

*run;*

*%mend freqTab3;*

* **A non-positional parameter is given a default value, which is what the macro will execute with unless the user specifies another value 🡺 You can mix positional and non-positional parameters. 🡺 To be safe, you should honestly just use non-positional parameters (since the default values can help you remember what that macro variable is supposed to take; in addition, you can then execute the macro with the parameters in any order)**

*%freqTab3;*

*%freqTab3(var3=Country);*

*%freqTab3(var3=Sex);*

*%freqTab3(var3=Race);*

# **Monday, November 12, 2018 – Lecture 19**

Missed class 🡺 continued going over programs from 2018-11-05-Lecture 18 folder

# **Wednesday, November 14, 2018 - Lecture 20**

Missed class 🡺 continued going over programs from 2018-11-05-Lecture 18 folder

# **Monday, November 19, 2018 – Lecture 21**

Finished going over programs from 2018-11-05-Lecture 18 folder

Generally, 4 options to help you debug your macros

mlogic (SAS log provides a lot of information as a macro is run, such as when it is beginning to execute, which statements inside the macro are being executed, **whether conditional statements resolve to TRUE or FALSE**, what iteration a loop is on, whether the loop will iterate again, and more), symbolgen (SAS log will tell you what macro variables resolve to any time a macro variable is resolved), mprint (SAS log shows you the code that SAS actually generates with the macro 🡺 the end product), mfile (less used than the other three; basically writes the stuff from mprint to a file)

-These options don’t affect any output

-Turn these on with an option statement (these are global options statements 🡺 will remain in effect until you start a new SAS session)

-Put “no” in front of them in an options statement to turn them off (ex: nomlogic)

-You must direct the output of mfile to a permanent file (for example, using a FILENAME statement, which is constructed identically to a LIBNAME statement 🡺 gives us an alias to reference a particular file) 🡺 this doesn’t have to be an already existing file as SAS will create it (by default a .txt file, but you could make a .sas file or a lot of other things)

# **Wednesday, November 21, 2018 – Lecture 22**

Missed lecture 🡺 get notes

Probably on macros (maybe call symput) or SQL?

# **Monday, November 26, 2018 – Lecture 23**

Missed lecture 🡺 get notes

Probably on SQL and joins?

# **Wednesday, November 28, 2018 – Lecture 24**

-Lecture 20 program folder on GitHub

Basic Joins

In PROC SQL, you don’t need to have selected a variable in the SELECT clause to use it elsewhere in the SQL statement. The SELECT clause is more like a KEEP statement, and it determines which variables we want to have in our final output. However, in PROC SQL, the SELECT and FROM clauses are necessary.

Inner Joins

Keeps only the observations that exist in both/all datasets (so, here for example, it would only keep observations where the value of USUBJID exists in both datasets)

*proc SQL noprint;*

*create table work.ae1 as*

*select dm.usubjid,dm.age,dm.armcd,dm.sex,dm.country,dm.race,ae.\**

*from echo.dm, echo.ae*

*where dm.usubjid=ae.usubjid*

*order by usubjid,aestdtc,aeendtc;*

*quit;*

This produces a warning (not an error 🡺 the code will still run)

WARNING: Column named usubjid is duplicated in a select expression (or a view). Explicit references to it will be to the first one.

WARNING: Variable USUBJID already exists on file WORK.AE1.

This is because both the DM and AE datasets both contain the USUBJID variable/column. Here it doesn’t affect the final result (because the usubjiid column is the same from both datasets), but having WARNINGS in your log is bad practice.

*proc SQL noprint;*

*create table work.ae2 as*

*select dm.usubjid,dm.age,dm.armcd,dm.sex,dm.country,dm.race,ae.aeterm,ae.aedecod,ae.aesoc,ae.aestdtc,ae.aeendtc*

*from echo.dm, echo.ae*

*where dm.usubjid=ae.usubjid*

*order by usubjid,aestdtc,aeendtc;*

*quit;*

*proc print data = work.ae2(obs=20); run;*

You could list out all the variables you want in your SELECT clause, but that is incredibly tedious. Note that the SELECT clause does not contain AE.USUBJID, but we can still use AE.USUBJID in the WHERE clause. The SELECT clause determines what variables we want to keep in our final output.

*proc SQL noprint;*

*create table work.ae3(drop=u) as*

*select dm.usubjid,dm.age,dm.armcd,dm.sex,dm.country,dm.race,ae.\**

*from echo.dm,*

*echo.ae(rename=(usubjid=u))*

*where dm.usubjid=ae.u*

*order by usubjid,aestdtc,aeendtc;*

*quit;*

*proc print data = work.ae3(obs=20); run;*

You could also include some SAS code. Use the RENAME option in the FROM clause to rename USUBJID from AE to U, and then drop it from Work.AE3 (in the CREATE clause). The WHERE clause is like a key for joins in the tidyverse in R.

The SELECT clause selects variables from datasets, while the FROM clause selects datasets. Even though the datasets we are using are not from the WORK library (and we need a libname in the FROM clause), we do not need the libname reference in the SELECT clause. However, if the FROM clause included both echo.dm and work.dm, for example, the SELECT clause would probably need to use echo.dm and work.dm.

Left Joins

Keeps all observations from the dataset listed on the left and only the observations that match from the dataset listed on the right

In SAS code, for example, that would be something like

merge x(in=a) y(in=b);

by …..;

if a=1;

*\*\* a simple left join to produce a dataset ;*

*proc SQL noprint;*

*create table work.ae4 as*

*select dm.usubjid,dm.age,dm.armcd,dm.sex,dm.country,dm.race,ae.aeterm,ae.aedecod,ae.aesoc,ae.aestdtc,ae.aeendtc*

*from echo.dm left join echo.ae on dm.usubjid=ae.usubjid*

*order by usubjid,aestdtc,aeendtc;*

*quit;*

*proc print data = work.ae4(obs=20); run;*

Other Types of Joins

-Full Joins (keeps all observations from both/all datasets)

-Outer Joins (includes what an inner join would give but also includes other rows for which no corresponding match is found in the other dataset)

Creating Macro Variables in SQL

*\*\* creating macro variable using SQL;*

*proc SQL noprint;*

*select strip(put(max(age),best.)) into :maxAge from echo.dm;*

*quit;*

*%put &=maxAge;*

General creating a macro variable using SQL. SQL is handy here because SQL makes it easy to do operations down columns.

*\*\* creating macro variable using SQL;*

*proc SQL noprint;*

*select distinct scan(usubjid,2,'-') into :siteList separated by '|' from echo.dm;*

*select count( distinct scan(usubjid,2,'-') ) into :siteNum from echo.dm;*

*quit;*

*%put &=siteNum.;*

*%put &=siteList.;*

siteList is a single macro variable that contains the list of all the site numbers. siteNum is the number of unique sites.

*\*\* a simple print macro;*

*%macro prt(obs=5);*

*%do j = 1 %to &siteNum.;*

*data temp;*

*set echo.dm;*

*where scan(usubjid,2,'-') = "%scan(&siteList.,&j,|)";*

*run;*

*title "ECHO.DM data for Site %scan(&siteList.,&j,|) -- First &obs. Subjects";*

*proc print data = temp(obs=&obs.) noobs; run;*

*%end;*

*%mend;*

*%prt;*

# **Monday, December 3, 2018 – Lecture 25**

ODS Layout 🡺 take a page, divide it into sections (can make sections any shape that you want), and put different things into different sections

Exam will not require you to use PROC SQL, but you can use it if you want

PROC REPORT allows you to do everything PROC PRINT does, but it also allows you to add some analysis and formatting of output

ODS Layout makes it easy to put specific, custom titles over specific objects