Paper P10-2009

Dietary Glycemic Load and Risk of Colon Cancer: A Guide to Data Management

Svetlana Zelenskiy[1], Lauren Byrne[1], Cheryl L. Thompson[1,2,3], Thomas C. Tucker[4], David Bruckman[1], Li Li[1,2,3]

[1]Departments of Epidemiology & Biostatistics and [2]Family Medicine, Case Western Reserve University, Cleveland, OH

[3]Transdisciplinary Research in Energetics and Cancer, Case Comprehensive Cancer Center, Cleveland, OH

[4]Kentucky Cancer Registry, University of Kentucky, Lexington, KY

ABSTRACT

Data come in different file formats and before we can summarize and explore patterns in the data and draw any inferences, it is essential to 1) import data from an outside source and assign variable names; 2) merge several sections into a single data set; and 3) clean variables and create new variables. After these initial steps, we can proceed with 4) exploratory analyses and model building.

This paper is geared toward graduate students and novice coders. The goal of this paper is to discuss effective methods to creating a data set ready to be used for exploratory analyses and model building. The techniques include 1) %MACRO IMPORT to import the data into SAS® and an example for standardizing variable names in each section and among sections; 2) %MACRO SORT to simplify sorting for multiple sections, ARRAY to simplify coding, and SET and MERGE statements to combine data sets; 3) a macro to check for duplicate IDs, a code to check for missing sections and values in a data set, a macro to recode race variable, and a code for creating a BMI variable. Tools to perform an initial exploratory analysis and a logistic regression will be briefly discussed at the end of this paper. We import the data from multiple sections of a risk factor questionnaire as well as coded responses to a comprehensive dietary food frequency questionnaire from a colon cancer case-control study into SAS and create a single ready-to-use data set. Examples evaluate the association of dietary glycemic load with colon cancer risk.

INTRODUCTION

With the computer age it has become easier for epidemiologists, biostatisticians, physicians and other health-related professionals to collect information. Data collection methods include conducting of interviews or telephone surveys, mailing questionnaire surveys, and collection of biological samples. A vast amount of data is also available from hospital records, police records, death certificates and census records. In addition to being obtained from multiple sources, data come in different file formats and before we can summarize and explore patterns in the data and draw any inferences, it is essential to 1) import data from an outside source and assign variable names; 2) merge several sections into a single data set; and 3) clean variables and create new variables. After these initial steps, we can proceed with 4) exploratory analyses and model building.

This paper is geared toward graduate students and novice coders. The goal of this paper is to discuss effective methods to creating a data set ready to be used for exploratory analyses and model building. The techniques include 1) %MACRO IMPORT to import the data into SAS and an example for standardizing variable names in each section and among sections; 2) %MACRO SORT to simplify sorting for multiple sections, ARRAY to simplify coding, and SET and MERGE statements to combine data sets; 3) a macro to check for duplicate IDs, a code to check for missing sections and values in a data set, a macro to recode race variable, and a code for creating a BMI variable. Tools to perform an initial exploratory analysis and a logistic regression will be briefly discussed at the end of this paper.

In this paper we import the data from multiple sections of a risk factor questionnaire as well as the data from a comprehensive dietary food frequency questionnaire into SAS and create a single ready-to-use data set. This data has been gathered over the past several years through a case-control study in the state of Kentucky in efforts to investigate the role of environmental and genetic risk factors on the development of colon cancer. As part of the study each participant (a “control” – a person who was never diagnosed with cancer or a “case” – a person who was diagnosed with colon cancer) mails in a filled-out risk factor questionnaire which is then entered into database software called FileMaker Pro 6.0. FileMaker Pro database is comprised of two categories (Cases and Controls), and each category has 11 sections or 11 data sets: 1) personal medical history, medications, and screening, 2) reproductive history (women only), 3) family history, 4) diet, 5) physical activity, 6) alcohol consumption, 7) smoking, 8) height and weight, 9) demographics and background information, 10) contact information, and 11) hip and waist measurements. Because control’s section 1) is broken into 2 parts, we ended up importing a total of 23 separate data sets. Additionally, we imported a 24th data set comprised of the dietary responses. This 24th data set was downloaded from a server at Arizona University. Our example will mainly illustrate the code for the control category, and wherever the code encompasses data from both categories, it will be noted with an asterisk (*).

OUTLINE

A. Import data sections with %MACRO and IMPORT statements
B. Combine data sets with %MACRO, SET AND MERGE statements; simplify the code with ARRAY statement
C. Clean the data set with %MACRO and other basic statements, and create new variables
D. Perform basic exploratory analysis and build a logistic regression model

A. IMPORT ALL DATA SECTIONS

Our code will accomplish four goals: 1) importing of all 24 data sets; 2) combining 24 data sets into a single data set; 3) cleaning the data set and creating new variables; and 4) exploratory analysis and model building of the final data set. First, we will use a macro to import all 23 data sets from Excel into SAS. A separate IMPORT procedure will be used to import the 24th data set. Another macro will then be used to sort each of the 24 data sets, while SET and MERGE statements will create a single data set. An example of ARRAY will illustrate how to simplify your code. A macro to check for duplicate IDs, a code to check for missing sections and values in a data set, a macro to recode race variable, and a code for creating a BMI variable will produce a ready-to-use data set. Finally, to accomplish the fourth goal, we will perform an exploratory analysis along with building a logistic regression model.

First, raw data for each section was exported from the FileMaker Pro in a tab delimited format and then each section was manually converted to an Excel format (with an exception to the dietary questionnaire data). Before we talk about %MACRO IMPORT, we should talk about the reasons why we would use macros and how they work. One of the ways that macros can help is that they can be used to automate procedures that need to be run over and over again (Delwiche & Slaughter, 1998). Upon submission of a standard SAS program, SAS would compile and then immediately execute the program (Delwiche & Slaughter, 1998). With macro, however, the macro statements are "resolved" via a macro processor in order to create standard SAS statements (Delwiche & Slaughter, 1998). A macro represents a section of a code that can have complete DATA and PROC statements as well as macro statements like %IF-%THEN/%ELSE and %DO, %END (Delwiche & Slaughter, 1998). Furthermore, macros usually, but not always, include macro variables. The %MACRO statement informs SAS that this is where the macro starts and the %MEND indicates the end of the macro (Delwiche & Slaughter, 1998). Macro's name should be consistent with standard SAS naming conventions (start with a letter or underscore, contain only letters, numerals or underscores, and can be no longer than 32 characters in length) (Delwiche & Slaughter, 1998). The macro's name in the %MEND statement is optional, but useful.

Example

In our example, we would like to import all 23 data sets from Excel into SAS. The following program (Code 1) creates a macro named IMPORT that has PROC IMPORT statements to import the data sets. Note that "&file" and "&output" are macro variables since they start with an ampersand (&) (Delwiche & Slaughter, 1998). In this particular example "&file" and "&output" have a local scope because they are defined inside a macro and can only be used inside their own macro (Delwiche & Slaughter, 1998). In contrast, a macro with a global scope is defined outside a macro and can be used at any place in the program (Delwiche & Slaughter, 1998). Furthermore, %MACRO IMPORT contains parameters for the file name to be exported from Excel and a file name to be created upon importing into SAS called "file" and "output".

Code 1. %MACRO IMPORT

```
%MACRO IMPORT (file, output);
PROC IMPORT DATAFILE= "C:\UKY_Data\Working Excel Files\&file..xls"
   OUT=&output
   DBMS=excel97
   REPLACE;
   GETNAMES = no;
RUN;
%MEND IMPORT;
```

After your macro is defined, you need to call it whenever you want to use it by adding a percent sign in front of the macro's name:

*Code 2. Calling macro**

```
%IMPORT (case_section1, case_section1);
%IMPORT (case_section2, case_section2);
%IMPORT (case_section3, case_section3);
%IMPORT (case_section4, case_section4);
%IMPORT (case_section5, case_section5);
%IMPORT (case_section6, case_section6);
%IMPORT (case_section7, case_section7);
%IMPORT (case_section8, case_section8);
%IMPORT (case_section9, case_section9);
```

```
%IMPORT (case_section10, case_section10);
%IMPORT (case_waist_hip, case_waist_hip);
%IMPORT (control_section1A, control_section1A);
%IMPORT (control_section1B, control_section1B);
%IMPORT (control_section2, control_section2);
%IMPORT (control_section3, control_section3);
%IMPORT (control_section4, control_section4);
%IMPORT (control_section5, control_section5);
%IMPORT (control_section6, control_section6);
%IMPORT (control_section7, control_section7);
%IMPORT (control_section8, control_section8);
%IMPORT (control_section9, control_section9);
%IMPORT (control_section10, control_section10);
%IMPORT (control_waist_hip, control_waist_hip);
```

The first call to the macro gives &file macro variable the value case_section1 &output macro variable the value case_section1, so that the statement generated by this call will export case_section1.xls file and create a file case_section1.sas7bdat. The standard SAS statements produced by the macro processor are listed below:

*Code 3. Standard SAS statement: PROC IMPORT**

```
PROC IMPORT DATAFILE= "C:\UKY_Data\Working Excel Files\case_section1.xls"
   OUT=case_section1
   DBMS=excel97
   REPLACE;
   GETNAMES = no;
RUN;
```

Because the data section containing dietary information is formatted differently from the other 23 data sections (it combines cases and controls together and has variable names listed for each column), it is imported via a separate PROC IMPORT statement (Code 4). The main difference between Code 3 and Code 4 is the GETNAMES statement. In Code 3, GETNAMES is set to “no”, while in Code 4 it is set to “yes”, thus allowing us to get variable names from the first line in the food frequency questionnaire:

*Code 4. Importing Arizona Food Frequency Questionnaire**

```
PROC IMPORT OUT= Work.FFQ
   DATAFILE= "C:\UKY_Data\Working Excel Files\ffq.xls"
   DBMS=excel2000 REPLACE;
   SHEET="Sheet1$";
   GETNAMES=yes;
RUN;
```

Standardizing variable names

When giving names for each data set, it is important to keep things consistent: that is the first part of the name indicated the category (a control or a case) and second part of the name specified the section. The same consistency is applied when creating variables within the data set. For example, the question asking whether or not a participant ever had a hemoccult test contains multiple parts to it and each part is coded as a separate variable. In order to keep things consistent, each part (variable) of the question carries a “hemoccult” tag as seen in Code 5 below:

Code 5: Assigning variable names

```
DATA control_section1a_new;
   LENGTH origin $8.;
   SET control_section1a;
   origin = 'Working';
   seq_id = F1;
   id = F2;
   hemoccult_test = F4;
   age_first_hemoccult = F100;
   year_first_hemoccult = F101;
   yearsago_first_hemoccult = F102;
   hemoccult_test_first_dnk = F5;
   hemoccult_reason1 = F7;
   hemoccult_reason2 = F9;
   hemoccult_reason3 = F10;
   hemoccult_reason4 = F11;
   hemoccult_resaon_dnk = F6;
   hemoccult_reason_othr = F8;
   number_sep_hemoccult = F103;
   num_sep_hemoccult_dnk = F12;
```

```
age_last_hemoccult = F104;
year_last_hemoccult = F105;
yearsago_last_hemoccult = F106;
hemoccult_test_last_dnk = F13;
```

The same convention of assigning variable names is applied throughout the questionnaire.

B. COMBINE DATA SETS AND SIMPLIFY CODE

Once all 24 data sets are imported into SAS and before they can be merged, we need to sort each data set by the unique BY variable which occurs in every data set. In our example, this unique BY variable is the ID variable. Having a common variable which uniquely identifies each observation in a data set ensures accurate matching (Delwiche & Slaughter, 1998). And again instead of running 24 separate PROC SORT procedures, we will use a macro called SORT to condense the recurring code and sort all 24 data sets at once:

Code 6. %MACRO SORT with macro call

```
%MACRO SORT (data);
PROC SORT DATA = &data;
   BY id;
RUN;
%MEND SORT;

**call macro to sort;
%SORT (control_section1a_new);
%SORT (control_section1b_new);
%SORT (control_section2_new);
%SORT (control_section3_new);
%SORT (control_section4_new);
%SORT (control_section5_new);
%SORT (control_section6_new);
%SORT (control_section7_new);
%SORT (control_section8_new);
%SORT (control_section9_new);
%SORT (control_section10_new);
%SORT (control_waist_hip_new);
```

One-to-one match merge

In our example, we need to merge control_section1a_new data set (Figure 1) containing the first half of the data for personal medical history, medications, and screening with control_section1b_new data set (Figure 2) containing the second half of the data. Figure 1 represents a sample of the data from the control_section1a_new data set with variables related to hemoccult testing, while Figure 2 represents a sample of the data from control_section1b_new data set with variables related to aspirin intake. In order for us to have a complete control_section1, these two data sets need to be merged together using the ID variable as the common variable.

Obs	id	hemoccult_test	age_first_hemoccult	year_first_hemoccult	yearsago_first_hemoccult
22	20055	Yes	18	1993	10.0
23	20056	Yes	.	.	6.5
24	20058	Yes	40	.	.
25	20060	Yes	.	.	.
26	20066	Yes	62	.	.
27	20069	No	.	.	.

Figure 1. *Data set control_section1a_new*

Obs	id	aspirin	aspirin_day	aspirin_week	aspirin_dnk
8	20055	Yes	2	5	
9	20058	No	.	.	
10	20060	No	.	.	
11	20066	Yes	.	.	
12	20069		.	.	

Figure 2. *Data set control_section1b_new*

Both data sets have already been sorted (Code 6). If you try to merge unsorted data, SAS will produce an error message and will not complete the merge (Delwiche & Slaughter, 1998). Code 7 below produces a data set named control_section1 (Figure 3) by merging the control_section1a_new data set together with the control_section1b_new data set using the ID variable as the common variable in the BY statement:

Code 7. One-to-One Match merge

```
DATA control_section1;
   MERGE control_section1a_new control_section1b_new;
   BY id;
RUN;
```

Obs	id	hemoccult_test	age_first_hemoccult	year_first_hemoccult	yearsago_first_hemoccult	aspirin	aspirin_day	aspirin_week	aspirin_dnk
22	20055	Yes	18	1993	10.0	Yes	2	5	
23	20056	Yes	.	.	6.5		.	.	
24	20058	Yes	40	.	.	No	.	.	
25	20060	Yes	.	.	.	No	.	.	
26	20066	Yes	62	.	.	Yes	.	.	
27	20069	No	.	.	.		.	.	

Figure 3. *Merged data set control_section1*

Notice that in the final data set, ID 20056 has missing values for the variables aspirin, asprin_day, aspirin_week, and aspirin_dnk because that ID was not present in the data set control_section1b_new. Therefore, we see that all observations from both data sets were merged into the final data set regardless whether or not these observations had matched.

Stacking data sets with the SET statement

When we want to combine two or more data sets which contain different observations but all or almost all of the variables being the same, the SET statement can be very useful (Delwiche & Slaughter, 1998). The SET statement stacks one data set on top of the other. In our example, we need to combine control_section1 data set (Figure 4) containing the data on personal medical history, medications, and screening for controls with case_section1_new data set (Figure 5) containing the same variables but for cases. Figures 4 and 5 represent a sample of the data with variables related to aspirin intake for the controls and cases, respectively. In order for us to have a complete section1, these two data sets need to be stacked together.

Obs	id	aspirin	aspirin_day	aspirin_week	aspirin_dnk
18	20004	Yes	1	4	Don't Know
19	20038	No	.	.	
20	20039	No	.	.	
21	20042	Yes	1	.	
22	20055	Yes	2	5	

Figure 4. *Data set control_section1*

Obs	id	aspirin	aspirin_day	aspirin_week	aspirin_dnk
728	11422	Yes	2	.	
729	11426	Yes	1	.	
730	11428	Yes	4	.	
731	11429	Yes	1	.	
732	11431	No	.	.	

Figure 5. *Data set case_section1_new*

The new data set section1 will contain the number of observations equal to the sum of the number of observations in the control_section1 and case_section1_new data sets (Delwiche & Slaughter, 1998). If, for instance, one of the data sets contains the variable which is not present in the other data set, then the missing values will be assigned to those observations in the other data set (Delwiche & Slaughter, 1998). Code 8 below results in a new data set named section1 (Figure 6) by combining control_section1 data set with the case_section1_new data set:

*Code 8. Stacking data sets**

```
DATA Working.section1;
   SET control_section1 case_section1_new;
RUN;
```

Obs	id	aspirin	aspirin_day	aspirin_week	aspirin_dnk
718	11422	Yes	2	.	
719	11426	Yes	1	.	
720	11428	Yes	4	.	
721	11429	Yes	1	.	
722	11431	No	.	.	
723	20004	Yes	1	4	Don't Know
724	20038	No	.	.	
725	20039	No	.	.	
726	20042	Yes	1	.	
727	20055	Yes	2	5	

Figure 6. *Stacked data set section1*

Merging data sets with the IN=option

Recall that when doing a one-to-one match merge, all observations from both data sets were merged into the final data set regardless whether or not these observations had matched. With the IN=option, you can select matching or non-matching observations during a merge of data sets (Delwiche & Slaughter, 1998). Although the IN=option can be used with SET, MERGE, or UPDATE, it is very common to be used with MERGE statement (Delwiche & Slaughter, 1998). In the Code 9 below, we merged the Final.masterfile3 data set which represents the assembled data from the risk factor questionnaire with the Work.FFQ data set which represents the assembled data from the food frequency questionnaire. The IN=option creates two variables called A and B. Therefore, the code below produces a data set named Final.masterfile4 by merging the data set A and the data set B. The subsetting IF B statement keeps only the observations with dietary data present (i.e. only observations with IDs in the food frequency data set) (Delwiche & Slaughter, 1998):

*Code 9. Merging with the IN=option**

```
DATA Final.masterfile4;
   MERGE Final.masterfile3 (IN = A) Work.FFQ (IN = B);
   BY id;
   IF B;
RUN;
```

Arrays

In our example, we wanted to condense some of the repetitive code that pertained to the family history section in our data set of 654 variables. For illustration purposes only, we want to count how many first-degree relatives have been diagnosed with colon cancer. First, we want to turn every occurrence of response “mother” to “1” for a question asking about relatives affected with colon cancer. The question is set up so that it provides a space to list 13 relatives that were affected with colon cancer, thus creating a variable for each of the 13 relatives. Instead of writing a series of assignment statements or IF statements, particularly if you have to recode a lot of variables, we can use arrays to simplify our code (Delwiche & Slaughter, 1998). An array represents a group of variables which are either all character or all numeric (Delwiche & Slaughter, 1998).

The following code (Code 10) changes every occurrence of response “mother” in these 13 variables to a value of “1”. An array, relation_cc, comprises a group of 13 variables, representing the 13 relatives affected with colon cancer. If you want to reference a variable using the array name, type that array name with the subscript for that variable (Delwiche & Slaughter, 1998). For instance, our first variable F7 has subscript 1, the second variable F8 has subscript 2, etc. To reference the first variable, we write relation_cc(1); to reference the second variable, we write relation_cc(2) and so on.

In the Code 10, the number of variables in the array list should be equal to the number listed in parentheses. That is, we have 13 variables listed (F7 – F19) and the number in parentheses right next to an array name relation_cc is also 13 (Delwiche & Slaughter, 1998). Additionally, we place the $ before listing the variables, because they are character variables. As you can see, all statements between the DO and the END statement are executed once for each

variable in the array, that is 13 times (Delwiche & Slaughter, 1998). That is, the response 'mother' has been changed to '1' in each of the 13 variables, whenever it occurred. The variable 'i' serves as an index variable, which is incremented by 1 each time the DO loop is executed (Delwiche & Slaughter, 1998). During the first DO loop, the variable 'i' gets a value of 1 and the corresponding IF statement will be IF relation_cc(1) = 'mother' THEN relation_cc(1) = '1'; and so forth, going through the DO loop in such fashion 13 times (Delwiche & Slaughter, 1998).

Code 10. Array statement

```
DATA control_section3_new;
   SET control_section3;
   id = F1;
   adopted = F3;
   med_history_blood_relatives = F4;
   relative_diag_colon_cancer = F5;
   ARRAY relation_cc(13) $ F7 - F19;
   drop F1-F5;
   DO i = 1 to 13;
   IF relation_cc(i) = 'mother' THEN relation_cc(i) = '1';
   END;
RUN;
```

Figure 7 below represents a sample of the output, where every 'mother' response for variable F7 was changed to '1'. Please note that while variables relation_cc(1) to relation_cc(13) in the array are not part of the data set, the index variable 'i' is (Delwiche & Slaughter, 1998). Similar array statement can be run to turn every occurrence of response "father" to "1" for a question asking about relatives affected with colon cancer. And, then after all of the responses for first-degree relatives are changed into '1', we can count a number of first-degree relatives diagnosed with colon cancer. Furthermore, we can run similar array statements with the rest of the variables in the family history section, thus creating arrays for colon_cancer variables (recoding the responses for presence or absence of colon cancer for each of the relatives), for relative_age_cc (recoding the responses for the age when colon cancer was diagnosed) and so on.

Obs	F7	id	i
12	1	20056	14
13	Father	20058	14
14		20060	14
15		20072	14
16		20080	14
17		20081	14
18		20096	14
19	grandmother	20097	14
20		20102	14
21	father	20104	14
22		20123	14
23		20136	14
24		20137	14
25	father	20139	14
26	1	20242	14

Figure 7. *Array code*

C. CLEANING THE DATA SET AND CREATING NEW VARIABLES

Checking for missing sections

Now that we have a single data set to work with, it is important to ensure the completeness of our data set. First we want to ensure that we have all 10 sections of the risk factor questionnaire completed for each of our observations. In the Code 11, we excluded section 10 (contact information) since that data cannot be used for analysis. Furthermore, data from the food frequency questionnaire also is not part of this code below since that data set comes as a single complete data file without separate sections. Final.masterfile contains all IDs. Final.complete contains IDs that have responses in section1 and section3 through waist_hip (since section2 is completed by women only and we assume that if an observation is missing section2, then that observation represents a male). Final.anymissing is all IDs that have at least 1 section missing. Final.anymissing_wo2 leaves out the IDs that just have section2 missing. In the

DATA step, we use the KEEP = option to specify the variables to keep, with 'have1', 'have2' and so on representing each section of the risk factor questionnaire. Following the DATA step, we proceed with the MERGE statement with the IN = option, thus creating a tag for each section. Variables have1 through havewh are set to zero. Then, we proceed with series of IF statements. For instance, the first IF statement indicates that if section1 is present for a particular ID, variable have1 will be given a value of 1; otherwise the variable have1 will have a value of zero.

Code 11. Checking for missing sections (Hawes, 2008)*

```
DATA Final.masterfile Final.complete Final.anymissing (keep = id have1 have2 have3
have4 have5 have6 have7 have8 have9 havewh)
Final.anymissing_wo2 (keep = id have1 have2 have3 have4 have5 have6 have7 have8
have9 havewh);
   MERGE Final.section1 (IN=a) Final.section2 (IN=b) Final.section3 (IN=c)
   Final.section4 (IN=d) Final.section5 (IN=e) Final.section6 (IN = f)
   Final.section7 (IN=g) Final.section8 (IN=h) Final.section9 (IN=i) Final.waist_hip
   (IN=j);
   BY id;
   have1 = 0; have2 = 0; have3 = 0; have4 = 0; have5 = 0; have6 = 0; have7 = 0;
   have8 = 0; have9 = 0; havewh = 0;
   IF a THEN have1 = 1;
   IF b THEN have2 = 1;
   IF c THEN have3 = 1;
   IF d THEN have4 = 1;
   IF e THEN have5 = 1;
   IF f THEN have6 = 1;
   IF g THEN have7 = 1;
   IF h THEN have8 = 1;
   IF i THEN have9 = 1;
   IF j THEN havewh = 1;
   OUTPUT Final.masterfile;
   IF a AND c AND d AND e AND f AND g AND h AND i AND j THEN OUTPUT Final.complete;
   IF a = 0 OR b = 0 OR c = 0 OR d = 0 OR e = 0 OR f = 0 OR g = 0 OR h = 0 OR i = 0
   OR j = 0 THEN OUTPUT Final.anymissing;
   IF a = 0 OR c = 0 OR d = 0 OR e = 0 OR f = 0 OR g = 0 OR h = 0 OR i = 0 OR j = 0
   THEN OUTPUT Final.anymissing_wo2;
RUN;
```

TMP1.anymissing

	id	have1	have2	have3	have4	have5	have6	have7	have8	have9	havewh
1	10007	1	1	1	1	1	1	1	1	1	0
2	11083	1	0	1	1	1	1	1	1	1	1
3	11095	1	1	1	1	1	1	1	0	1	1
4	11129	1	1	1	1	1	1	1	0	1	1

Figure 8. *Checking for missing sections*

From Figure 8 above, we can see, for instance, that ID 11129 is missing section 8.

Checking for duplicates

Now that we have checked for missing sections, it is equally important to check for duplicate observations. The following program (Code 12) creates a macro named CHECKID. In the DATA step, the SET statement reads the &old macro variable data set and creates the &new macro variable data set. The data set is ordered by ID variable. Next, the FIRST.ID automatic variable named FirstID is created and has a value of 1 upon processing an observation with the first occurrence of a new value for the variable ID and a value of zero for the consecutive occurrences (Delwiche & Slaughter, 1998). This DATA step ends with the KEEP statement, where variables ID and FirstID are kept in the data set (Delwiche & Slaughter, 1998). Furthermore, %MACRO CHECKID contains parameters for the file name to be created and a file name to be read from called "new" and "old".

The next portion of this macro includes the PROC FREQ procedure of the &new macro variable data set which produces table where FirstID = 0, that is the table of duplicate IDs (or consecutive occurrences of the same ID). The macro ends with the %MEND CHECKID statement. The first call to the macro gives &new macro variable the value check_section1 &old macro variable the value final.section1, so that the statement generated by this call will read the final.section1 file and create a file check_section1 containing the list of duplicate IDs.

*Code 12. %MACRO CHECKID with macro call**

```
%MACRO CHECKID (new, old);
TITLE1 "Duplicate IDS in &old";

DATA &new;
   SET &old;
```

```
   BY id;
   FirstID = FIRST.ID;
   KEEP id FirstID;
RUN;

PROC FREQ DATA = &new;
   TABLES id;
   WHERE FirstID = 0;
RUN;
%MEND CHECKID;

**call macro to CHECKID;
%CHECKID (check_section1, final.section1);
%CHECKID (check_section2, final.section2);
%CHECKID (check_section3, final.section3);
%CHECKID (check_section4, final.section4);
%CHECKID (check_section5, final.section5);
%CHECKID (check_section6, final.section6);
%CHECKID (check_section7, final.section7);
%CHECKID (check_section8, final.section8);
%CHECKID (check_section9, final.section9);
%CHECKID (check_section10, final.section10);
%CHECKID (check_hip_waist, final.waist_hip);
```

Checking for missing values

The final step of cleaning data involves identifying missing values in your variables of interest. The missing values are typically represented as a dot (.) for a numeric variable and a blank space (‘ ‘) for a character variable. Code 13 below utilizes the PROC FREQ statement to produce a list of IDs WHERE the variable date_of_birth is missing. The missing value for this particular variable is represented as a dot since it’s a numeric variable. Similar statements can be run for other variables of interest.

Code 13. Identifying missing values

```
PROC FREQ DATA = Final.masterfile;
   WHERE date_of_birth = .;
   TABLES id;
RUN;
```

%MACRO RACE

Now that we have cleaned our data set, we need to recode some of the variables. As an example, we will show how to recode race and create a new variable called bmi_2yrs. Section 9 (demographics and background information) included information on race of a participant (variable F19), participant’s mother (variable F20), father (variable F21), maternal grandmother (variable F22), maternal grandfather (variable F23), paternal grandmother (variable F24) and paternal grandfather (variable F25). However, entries for race were not uniform, that is they were entered into the FileMaker Pro database exactly the way they were filled out on the questionnaire: “w”, “american”, “irish”, “white/italian”, “black”, “afr”, and so forth. This created a lot of confusion and we wanted to categorize race into 3 categories only: Caucasian, African American, and Other by creating macro named RACE.

First, using PROC FREQ statement, we ran frequencies of race variables in case_section9 data set. Then, using the SET statement, a new data set named case_section9_recode was created. %MACRO RACE has 3 parameters: section (section name for where the data is coming from), input (column input), and newvar (name of the new variable). In recoding some of the values, we needed to use ‘=:’ which means ‘begins with’ in standard SAS language. Calling %MACRO RACE using parameters for section=control_section9_recode, input=F19, and newvar=race produces a new variable called race that has only 3 categories: Caucasian, African American, and Other. When running the rest of the %MACRO RACE, 6 more new variables containing only 3 race categories are created: race_mother-race_ffather.

*Code 14. %MACRO RACE with macro call**

```
**Preliminary frequencies of race variables;
PROC FREQ DATA = case_section9;
   TABLES F19 - F25;
RUN;

**Create a new data set called case_section9_recode where race variables will be
recoded;
DATA case_section9_recode;
   SET case_section9;
RUN;
```

```
**Recoding race variables;
**RACE MACRO;
**Note that =: means 'begins with' in SAS code;

%MACRO RACE (section, input, newvar);
DATA &section;
   LENGTH &newvar $20.;
   SET &section;
/* First, recode any value with a "/" that is not "other" */
   IF &input = 'caucasian / european' THEN &newvar = 'caucasian';
   ELSE IF &input =: 'lebanese' THEN &newvar = 'caucasian';
   ELSE IF &input = 'w / american' THEN &newvar = 'caucasian';
   ELSE IF &input = 'white / caucasian' THEN &newvar = 'caucasian';
   ELSE IF &input = 'w / irish descent' THEN &newvar = 'caucasian';
   ELSE IF &input = 'scottish / english' THEN &newvar = 'caucasian';
   ELSE IF &input = 'w / scot irish descent' THEN &newvar = 'caucasian';
   ELSE IF &input = 'white/italian' THEN &newvar = 'caucasian';
   ELSE IF &input = 'white/irish' THEN &newvar = 'caucasian';
   ELSE IF &input = 'white / black' THEN &newvar = 'african american';
   ELSE IF &input = 'black / white' THEN &newvar = 'african american';
   ELSE IF &input = '1/2black,1/2white' THEN &newvar = 'african american';
   ELSE IF &input = 'african american / caucasian' THEN &newvar = 'african
   american';
   ELSE IF &input = 'jewish / black' THEN &newvar = 'african american';
   ELSE IF &input =: 'german' THEN &newvar = 'caucasian';
/* THEN recode all other values with a "/" to "other" */
   ELSE IF index (&input,'/') THEN &newvar = 'other';

/* Second, recode any value that begins with "w" but is not "caucasian" */
   ELSE IF &input = 'white-native american' THEN &newvar = 'other';
   ELSE IF &input =: 'west ind' THEN &newvar = 'other';
   ELSE IF &input = 'white - american indian' THEN &newvar = 'other';
/* THEN recode all other values that begin with "w" as "caucasian" */
   ELSE IF &input =: 'w' THEN &newvar = 'caucasian';

/* Third, recode any value that begins with "c" but is not "caucasian" */
   ELSE IF &input = 'caucasian (part native american)' THEN &newvar = 'other';
   ELSE IF &input =: 'cherokee' THEN &newvar = 'other';
   ELSE IF &input = 'chinese' THEN &newvar = 'other';
   ELSE IF &input = 'caucasian - indian' THEN &newvar = 'other';
   ELSE IF &input = 'caucasian - native american' THEN &newvar = 'other';
   ELSE IF &input = 'caucasian and native american' THEN &newvar = 'other';
/* THEN recode all other values that begin with "c" as "caucasian" */
   ELSE IF &input =: 'c' THEN &newvar = 'caucasian';

/* Fourth, recode any value that begins with "a" but is not "caucasian" */
   ELSE IF &input =: 'afr' THEN &newvar = 'african american';
   ELSE IF &input = 'arican american' THEN &newvar = 'african american';
   ELSE IF &input =: 'asian' THEN &newvar = 'other';
   ELSE IF &input = 'american indian' THEN &newvar = 'other';
   ELSE IF &input = 'am. indian' THEN &newvar = 'other';
/* THEN recode all other values that begin with "a" as "caucasian" */
   ELSE IF &input =: 'a' THEN &newvar = 'caucasian';

/* Last, recode all other values that are not 'other' into proper categories */
/* caucasian */
   ELSE IF &input =: 'english' THEN &newvar = 'caucasian';
   ELSE IF &input =: ' c' THEN &newvar = 'caucasian';
   ELSE IF &input = 'american' THEN &newvar = 'caucasian';
   ELSE IF &input =: 'french' THEN &newvar = 'caucasian';
   ELSE IF &input =: 'german' THEN &newvar = 'caucasian';
   ELSE IF &input =: 'irish' THEN &newvar = 'caucasian';
   ELSE IF &input = 'italian' THEN &newvar = 'caucasian';
   ELSE IF &input = 'portugese' THEN &newvar = 'caucasian';
   ELSE IF &input =: 'scot' THEN &newvar = 'caucasian';
   ELSE IF &input = 'spanish' THEN &newvar = 'caucasian';
   ELSE IF &input = 'swiss' THEN &newvar = 'caucasian';

/* african american */
   ELSE IF &input =: 'black' THEN &newvar = 'african american';
```

```
   ELSE IF &input = 'negro' THEN &newvar = 'african american';

/* missing */
   ELSE IF &input = ' ' THEN &newvar = ' ';
   ELSE IF &input = 'unknown' THEN &newvar = ' ';
   ELSE IF &input = '?' THEN &newvar = ' ';
   ELSE IF &input = 'x' THEN &newvar = ' ';

/*THEN recode all other values into "other" category */
   ELSE &newvar = 'other';
RUN;
%MEND RACE;

**call macro to recode race;
%RACE (control_section9_recode, F19, race);
%RACE (control_section9_recode, F20, race_mother);
%RACE (control_section9_recode, F21, race_father);
%RACE (control_section9_recode, F22, race_mmother);
%RACE (control_section9_recode, F23, race_mfather);
%RACE (control_section9_recode, F24, race_fmother);
%RACE (control_section9_recode, F25, race_ffather);
```

BMI variable

We also wanted to create a new variable named bmi_2yrs which calculates a participant's Body Mass Index 2 years prior to a participant's participation (for control) or Body Mass Index 2 years prior to a colon cancer diagnosis (for case). This calculation below is based on the weight measurements in pounds and height measurements in inches:

$$\text{BMI (kg/m}^2\text{)} = \frac{(weight\ in\ pounds * 703)}{height\ in\ inches^2}$$

Code 15. Creating bmi_2yrs variable

```
DATA Final.masterfile2;
   SET Final.masterfile2;
   bmi_2yrs = weight_2yrs/(height*height)*703;
RUN;
```

D. EXPLORATORY ANALYSES AND UNCONDITIONAL LOGISTIC REGRESSION MODEL

Now that our data set is ready for statistical analysis, we will briefly talk about how to perform exploratory analysis and fit multivariate logistic regression to our data. High dietary glycemic load (GL) has been inconsistently associated with the risk of colon cancer in epidemiologic studies. We seek to further clarify this relationship in a population-based incident case-control study.

Our data set MWSUG.masterfile included the following 11 variables: ID (participant's ID), ener (energy in kcal), gl (glycemic load in g), tfib (total dietary fiber in g), case (case or control), gender (participant's gender), age, bmi_2yrs (BMI in kg/m^2), relative_diag_colon_cancer (family history of colon cancer), race, NSAID (whether or not Nonsteroidal Anti-inflammatory Drugs were taken for more than 6 months) and contained 1627 observations. Our binary response variable measures the presence or absence of the colon cancer represented by the variable case, while our explanatory variable is the glycemic load represented by the variable gl. The other 8 variables represent potential covariates. Variables ener, gl, tfib, age, and bmi_2yrs are continuous, while case, gender, relative_diag_colon_cancer, race, and NSAID are categorical.

The data set was again checked for missing values among all variables using PROC FREQ statement. Variables case, gender, and NSAID were each missing 1 observation. Variable age was missing 30 observations, bmi_2yrs had 66 missing observations, relative_diag_colon_cancer had 7 missing observations, and race variable had 3 missing observations. Overall descriptive statistics and broken down BY case descriptive statistics (mean, median, standard deviation, minimum, and maximum, as well as test of normality, a stem-and-leaf plot, and a box plot) were obtained for each continuous explanatory variable using PROC UNIVARIATE procedure. Variables ener, gl, tfib, and bmi_2yrs each had a skewed distribution, while variable age had a normal distribution. Additionally, Mann-Whitney U-test (using PROC NPAR1WAY procedure) and t-test (using PROC TTEST procedure) were run on all continuous variables. Mann-Whitney U-test is a nonparametric test that is used when the assumptions for using a t-test are not met. That is, the assumptions of having a large sample size and data with normal distribution are violated. Both tests showed that there was a statistically significant difference among cases and controls in each of the explanatory variable except tfib.

For variables gl and bmi_2yrs, we created categorical variables gl_cat and bmi2yrs_cat, respectively. Additionally, in order to use categorical variables in building a multivariate logistic regression, dummy variables were created for all categorical variables using a series of IF/THEN/ELSE statements. Frequencies of the newly created categorical and dummy variables were checked with PROC FREQ statements. Finally, the association of potential categorical covariates with colon cancer was assessed with the PROC FREQ statement and an EXPECTED CHISQ MEASURES option to get measures of association.

*Code 16. Exploratory analysis**

```
**Missing values for ener;
PROC FREQ DATA = MWSUG.masterfile;
   WHERE ener = .;
   TABLES id;
RUN;

**Overall descriptive statistics for continuous variables;
PROC UNIVARIATE DATA = MWSUG.masterfile NORMAL PLOT;
   TITLE 'Descriptives Statistics on ener gl tfib age bmi_2yrs';
   VAR ener gl tfib age bmi_2yrs;
RUN;

**Descriptive statistics for continuous variables broken down BY case;
PROC SORT DATA = MWSUG.masterfile;
   BY case;
RUN;

PROC UNIVARIATE DATA = MWSUG.masterfile NORMAL PLOT;
   TITLE 'Descriptives Statistics on ener gl tfib age bmi_2yrs BY case';
   VAR ener gl tfib age bmi_2yrs;
   BY case;
RUN;

**Running t-tests on continuous variables;
PROC SORT DATA = MWSUG.masterfile;
   BY case;
RUN;

**T-tests are statistically significant for all variables except tfib;
PROC TTEST DATA = MWSUG.masterfile;
   CLASS case;
   VAR ener gl tfib age bmi_2yrs;
RUN;

**Mann-Whitney U-tests are statistically significant for all variables except tfib;
PROC NPAR1WAY DATA = MWSUG.masterfile WILCOXON;
   CLASS case;
   VAR ener gl tfib age bmi_2yrs;
RUN;

**Creating categorical and dummy variables;
DATA MWSUG.masterfile;
   SET MWSUG.masterfile;

**Create categorical variable for gl;
   IF gl <= 74.29 THEN gl_cat = '1stq';
   ELSE IF gl > 74.29 AND gl <= 102.74 THEN gl_cat = '2ndq';
   ELSE IF gl > 102.74 AND gl <= 134.66 THEN gl_cat = '3rdq';
   ELSE IF gl > 134.66 AND gl <= 182.35 THEN gl_cat = '4thq';
   ELSE IF gl > 182.35 THEN gl_cat = '5thq';
   ELSE gl_cat = ' ';

**Create dummy variables for gl_cat: Reference = 1stq;
   IF gl_cat = '2ndq' THEN Q2 = 1;
   ELSE IF gl_cat NE ' ' THEN Q2 = 0;
   IF gl_cat = '3rdq' THEN Q3 = 1;
   ELSE IF gl_cat NE ' ' THEN Q3 = 0;
   IF gl_cat = '4thq' THEN Q4 = 1;
   ELSE IF gl_cat NE ' ' THEN Q4 = 0;
   IF gl_cat = '5thq' THEN Q5 = 1;
   ELSE IF gl_cat NE ' ' THEN Q5 = 0;
RUN;
```

```
**Frequency of glycemic load variable to double check counts;
TITLE1 'Frequency of glycemic load';
PROC FREQ DATA = MWSUG.masterfile;
  TABLES gl gl_cat Q2 Q3 Q4 Q5;
RUN;

**Association of potential covariates with colon cancer;
PROC FREQ DATA = MWSUG.masterfile;
  TABLES gl_cat*case / EXPECTED CHISQ MEASURES;
RUN;
```

Finally, we ran a number of variable selection models (i.e. multivariate unconditional logistic regression models) using PROC LOGISTIC procedure and going through forward, backward, stepwise, and manual variable selection process in order to come up with the best model. PROC LOGISTIC statements are pretty straightforward. The DESCENDING option specifies that the resulting equation predicts the log odds of having colon cancer given a certain set of explanatory values. The logistic regression example in Code 17 depicts several MODEL options: LACKFIT (the Hosmer and Lemeshow Goodness-of-Fit test), CTABLE (a classification table), and RISKLIMITS (the Odds Ratios for each variable along with 95% confidence limits). To specify the forward selection process, for instance, include SELECTION=FORWARD option in the MODEL statement.

*Code 17. Multivariate unconditional logistic regression modeling**

```
TITLE1 'Logistic Regression : gl categorical';
PROC LOGISTIC DESCENDING DATA = MWSUG.masterfile OUTEST =BETAS COVOUT;
  CLASS gl_cat (REF='1stq') /PARAM = ref;
  MODEL case = gl_cat ener tfib gender age bmi_2yrs famhist NSAID
  gl_cat*gender /
  LACKFIT
  CTABLE
  RISKLIMITS;
  OUTPUT OUT=pred P=phat LOWER=lcl UPPER=ucl
  PREDPROB=(individual crossvalidate);
RUN;
```

Model Fit Statistics

Criterion	Intercept Only	Intercept and Covariates
AIC	2022.713	1925.119
SC	2028.045	2010.427
-2 Log L	2020.713	1893.119

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	127.5933	15	<.0001
Score	123.0643	15	<.0001
Wald	113.5003	15	<.0001

Figure 9a. *Final multivariate unconditional logistic regression model: Model fit statistics*

Type 3 Analysis of Effects

Effect	DF	Wald Chi-Square	Pr > ChiSq
gl_cat	4	3.6699	0.4525
ener	1	3.6110	0.0574
tfib	1	7.8129	0.0052
gender	1	0.2276	0.6333
age	1	51.6721	<.0001
bmi_2yrs	1	20.3749	<.0001
famhist	1	16.1174	<.0001
NSAID	1	8.7977	0.0030
gender*gl_cat	4	9.8560	0.0429

Analysis of Maximum Likelihood Estimates

Parameter		DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept		1	-4.4209	0.4874	82.2616	<.0001
gl_cat	2ndq	1	0.0592	0.2085	0.0805	0.7766
gl_cat	3rdq	1	0.1542	0.2225	0.4804	0.4882
gl_cat	4thq	1	0.4235	0.2432	3.0312	0.0817
gl_cat	5thq	1	0.4100	0.3349	1.4990	0.2208
ener		1	0.000198	0.000104	3.6110	0.0574
tfib		1	-0.0218	0.00779	7.8129	0.0052
gender		1	-0.1541	0.3231	0.2276	0.6333
age		1	0.0403	0.00560	51.6721	<.0001
bmi_2yrs		1	0.0405	0.00896	20.3749	<.0001
famhist		1	0.4804	0.1197	16.1174	<.0001
NSAID		1	-0.3353	0.1130	8.7977	0.0030
gender*gl_cat	2ndq	1	0.8480	0.4246	3.9889	0.0458
gender*gl_cat	3rdq	1	0.5374	0.4092	1.7244	0.1891
gender*gl_cat	4thq	1	-0.0501	0.4052	0.0153	0.9015
gender*gl_cat	5thq	1	0.7809	0.4172	3.5036	0.0612

Odds Ratio Estimates

Effect	Point Estimate	95% Wald Confidence Limits	
ener	1.000	1.000	1.000
tfib	0.978	0.964	0.994
age	1.041	1.030	1.053
bmi_2yrs	1.041	1.023	1.060
famhist	1.617	1.279	2.044
NSAID	0.715	0.573	0.893

Association of Predicted Probabilities and Observed Responses

Percent Concordant	66.9	Somers' D	0.342
Percent Discordant	32.7	Gamma	0.343
Percent Tied	0.4	Tau-a	0.160
Pairs	546832	c	0.671

Wald Confidence Interval for Adjusted Odds Ratios

Effect	Unit	Estimate	95% Confidence Limits	
ener	1.0000	1.000	1.000	1.000
tfib	1.0000	0.978	0.964	0.994
age	1.0000	1.041	1.030	1.053
bmi_2yrs	1.0000	1.041	1.023	1.060
famhist	1.0000	1.617	1.279	2.044
NSAID	1.0000	0.715	0.573	0.893

Figure 9b. *Final multivariate unconditional logistic regression model: Maximum Likelihood and Odds Ratio estimates*

```
Hosmer and Lemeshow Goodness-of-Fit Test

Chi-Square       DF     Pr > ChiSq
    6.4616        8         0.5957
```

Figure 9c. *Final multivariate unconditional logistic regression model: Goodness-of-Fit test*

The study sample consisted of 572 incident colon cancer cases and 956 population controls. Cases were recruited through the Kentucky Cancer Registry, and controls were recruited via random digit dialing. Glycemic load was assessed based on a self-administered food frequency questionnaire.

On average, the cases had a significantly higher GL (mean = 147.4g, SD = 93.4g) than the controls (mean = 130.1g, SD = 78.2g) (p = 0.0001). In multivariate unconditional logistic regression model adjusted for age, gender, body mass index (BMI), family history of colorectal cancer, NSAID use, total dietary fiber and total caloric intake, the odds ratio (OR) for the 2nd through the upper quintiles of GL were: 1.06 (95% CI: 0.71, 1.60), 1.17 (95% CI: 0.75, 1.80), 1.53 (95% CI: 0.95, 2.46), 1.51 (95% CI: 0.78, 2.90), respectively (p for trend = 0.1527), as compared to those at the bottom quintile of GL intake. There is a suggested evidence for a positive association between glycemic load and the risk of colon cancer. The results also indicated an effect modification by gender that needs to be further evaluated through stratification analysis by gender. P-value of Hosmer and Lemeshow Goodness-of-Fit test was 0.60, indicating that the above model was a good fit.

Missing values in PROC LOGISTIC

Reviewing the log file after running PROC LOGISTIC statement revealed that there were 99 observations that were excluded from the analysis due to missing values in the variables specified previously. Since PROC LOGISTIC does not accommodate missing data, we need to correct for missing values in these variables by accessing the original filled out questionnaires. If the information in the original questionnaires is not present, we could perhaps impute missing values using the PROC MI procedure.

CONCLUSION

Data come in different file formats and before we can summarize and explore patterns in the data and draw any inferences, it is essential to 1) import data from an outside source and assign variable names; 2) merge several sections into a single data set; and 3) clean variables and create new variables. After these initial steps, we can proceed with 4) exploratory analyses and model building.

This paper provided effective methods to creating a data set ready to be used for exploratory analyses and model building. The techniques included 1) %MACRO IMPORT to import the data into SAS and an example for standardizing variable names in each section and among sections; 2) %MACRO SORT to simplify sorting for multiple sections, ARRAY to simplify coding, and SET and MERGE statements to combine data sets; 3) a macro to check for duplicate IDs, a code to check for missing sections and values in a data set, a macro to recode race variable, and a code for creating a BMI variable; and 4) tools to perform an initial exploratory analysis and a logistic regression model.

Having a single ready-to-use data set allows for a greater flexibility and efficiency in handling and data manipulation compared to working with multiple separate data sets. Furthermore, the techniques covered in this paper can be easily expanded and applied in other areas of research.

REFERENCES

1. Delwiche, Lora D., and Susan J. Slaughter. 1998. *The Little SAS® Book: A Primer, Second Edition.* Cary, NC: SAS Institute Inc.

2. Hawes, Stephen E. 2008. *Advanced Topics in SAS Programming or Everything I Can Think of to Tell You About in SAS.* [Electronic Document] Chicago, IL: University of Washington School of Public Health and Community Medicine.

ACKNOWLEDGMENTS

This research was supported by a Damon Runyon Cancer Research Foundation Clinical Investigator Award (CI-8), the Case Center for Transdisciplinary Research on Energetics and Cancer (1U54 CA-116867-01), and a National Cancer Institute K22 Award (1K22 CA120545-01).

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Svetlana Zelenskiy
Case Western Reserve University
Department of Epidemiology & Biostatistics
10900 Euclid Avenue, Cleveland, OH 44106-4945
svetlana.zelenskiy@gmail.com