Paper 11521-2016

Building Macros for Quick Survey Scoring

Vincent Chan & Lorena Ortiz, IMPAQ International LLC

ABSTRACT

Surveys are a critical research component when gathering information about a population of people. Researchers often implement surveys after a population has participated in a class or educational program with the aim to capture knowledge, attitudes, and experiences of participants. Survey developers create surveys using psychometric principles to ensure the reliability and validity of each question, also known as an 'item.' A developer can purposefully create sets of items, which when analyzed as a group, can measure a *construct* that describes the underlying behavior, attribute, or characteristic of a study participant. The programmer calculates the *construct* score by combining (e.g. summing or averaging) all component items of the *construct*.

The programmer is tasked with creating such scores quickly. With an unlimited amount of time to spend, a programmer could operationalize the creation of these constructs by manually entering predefined formulas in the DATA step. More often, time and resources are limited. Therefore, the programmer is more efficient by automating this process using macro programs. In this paper, we present a technique that uses an externally created specification Excel file (*spec*) that contains the construct specifications and corresponding items of each construct. By looping through macro variables created from this *spec* file, we can create construct scores for an unlimited number of items. This technique can be generalized to other processing tasks that involve any number of mathematical combinations of variables to create one single score.

INTRODUCTION

Our example is taken from a project evaluating a teacher professional development program aimed at improving student literacy. Teachers were asked to fill out a survey after participating in the professional development and applying learned techniques in their classrooms. The items in this survey are purposefully created to fit into broad classifications, which are further classified into smaller categories. Survey items can be comprised of Likert scales (ranging 1-5), yes/no (1 or 0) responses, items that ask the respondent to select all that apply (yes=1/not selected=blank), and/or items that ask for a numeric response (e.g. number of minutes spent on a task, number of days spent in a training). Every item in our example belongs to 'check all that apply' questions where a respondent selected the checkbox if they applied a particular technique in their classroom.[1]

STEP 1: CREATING A SPEC FILE

The macro code in this paper relies on the creation of an Excel file that saves in each column the specifications of the survey items. **Table 1** provides an example of how the spec file is created. We will call this spec file *simple_spec*. The spec file includes the following variables[2]:

1. **Variable**: The variable name should be entered exactly as it is saved in the data set. We assume this survey dataset has been cleaned, and original raw survey questions have been changed to names that are more intuitive. In this example in the survey, we are asking whether a teacher applied a certain teaching technique in the classroom via different modes of instruction (e.g. through personally instructing the class, group discussion, etc.). A teacher checks which techniques were used under each mode of instruction.

[1] Constructs can be created using different response types (Likert, yes/no, check all that apply, number of minutes). A construct can contain one or more of response type depending on how the survey developer intended to define the construct.

[2] The column headers in *simple_spec* should be adjusted and named according to your particular dataset. Depending on the categories in your data set you should add or omit columns.

2. **Domain**: Indicates the primary category for the survey items. The domain is usually a broad concept, e.g. collaboration, comprehension, identity. For this example, we are focusing on one domain.
3. **Classification**: Within each domain the classification is a category that addresses an underlying concept. For example, within the domain of 'collaboration', classifications could include: student collaboration, student engagement. For this example, we are focusing on only one classification.
4. **Subclassification**: Within each classification, the subclassification narrows in on a specific teaching technique. For example, within student engagement classification, subclassifications could include: participation in classroom discussion, respectful collaboration with peers.
5. **Number**: This variable numbers the survey questions that are part of each subclassification, and is used to accumulate totals. The main purpose of this column is to allow us to uniquely identify each item option or question and to loop through each subclassification. However, the numbers can be useful in referring back to the original survey.

In our example we focus on a framework that categorizes concepts around literacy. Specifically we are looking at three constructs comprised of the following items:

- Construct 1: Mode1_technique1, Mode2_technique1, Mode3_technique1
- Construct 2: Mode1_technique2, Mode2_technique2, Mode3_technique2
- Construct 3: Mode1_technique3, Mode2_technique3, Mode3_technique3

What we want to end up with is totals or mean for each construct. For example, for classification 1, we want to create a total summing up Mode1_technique1, Mode2_technique1, Mode3_technique1, and Mode4_technique1.

Table 1. Specification file (*simple_spec.xlsx*)

Variable	Domain	Class	Subclass	Num
Mode1_technique1	1	3	1	1
Mode1_technique2	1	3	2	1
Mode1_technique3	1	3	3	1
Mode2_technique1	1	3	1	2
Mode2_technique2	1	3	2	2
Mode2_technique3	1	3	3	2
Mode3_technique1	1	3	1	3
Mode3_technique2	1	3	2	3
Mode3_technique3	1	3	3	3
Mode4_technique1	1	3	1	4
Mode4_technique2	1	3	2	4
Mode4_technique3	1	3	3	4

STEP 2: SETTING UP THE FILES

IMPORTING THE SPEC FILE

To begin, we will import *simple_spec* (see **Table 1**), and store the *variable, domain, class, subclass,* and *num* columns in separate macro variables using PROC SQL with the INTO clause. The code below creates the macro variables that will be used in scoring each construct:

```
proc sql print;
      select variable, domain, class, subclass, num
            into  :varlist    separated by " ",
                  :domainlist separated by " ",
                  :classlist  separated by " ",
                  :sublist    separated by " ",
                  :numlist    separated by " ",
      from simple_spec;
quit;
```

Each macro (*varlist*, *domainlist*, *classlist*, *sublist*, *numlist*) now contains a list of each of the values from the respective column with each value separated by a space.

STEP 3: CLEANING EACH ITEM

Next, we introduce our data set *survey* that contains the survey responses we want to score. We use the macro *clean_em* to loop through each variable, and create a clean recoded score variable that will be used later to produce construct totals. As we iterate through each variable, we recode missing values and codes to 0, since those values do not contribute any value to the score. Additional code could be added here to perform recoding, cleaning of outlies or other codes, or even creating alternative categorizations and handling of values. The following code shows how we program these score variables. Essentially, this code creates a data set *score* that contains the score variables for each item held in the macro variable *outvar*:

```
/* Count number of variables and store in a macro */
%let Nwithin=%sysfunc(countw(&varlist.));
/* Iterate through each domain/class/subclass item    **
** and create score variable                          **/
%macro clean_em;
data score;
set surveys;
      %do item=1 %to &Nwithin.;
            %let domain = %scan(&domainlist., &item., " ");
            %let class  = %scan(&classlist.,  &item., " ");
            %let sub    = %scan(&sublist.,    &item., " ");
            %let num    = %scan(&numlist.,    &item., " ");
            %let invar  = %scan(&varlist.,    &item., " ");
            %let outvar = Score_&domain._&class._&sub._&num.;

            /* ********** CREATE SCORE VARIABLE ********** */
            /* Recode missing or missing codes; treat as 0s in sum */
            if &invar. in (.,-999) then &outvar.=0;
                  else &outvar.=&invar.;
      %end;
run;
%mend clean_em;
```

After running the *clean_em* macro on the *survey* dataset, you will end up with 14 variables, one for each item, and each item will be named according to the convention "Score_[*domain*]_[*class*]_[*subclass*]_[*num*]". For example, Score_1_3_1_1 will correspond to the first item under domain 1 class 1 subclass 1. The resulting dataset will include the raw variables and the recoded, clean variables all saved under the naming convention for the score variables.

The SAS log below shows what happens in the 12th and last iteration of the *clean_em* macro. It creates the variable Score_1_3_3_1 based on Mode4_technique3. We've activated the MLOGIC, SYMBOLGEN, and MPRINT system options to easily check what is stored in each macro we are creating.

```
MLOGIC(SCORE_EM):  %DO loop index variable ITEM is now 12; loop will iterate again.
MLOGIC(SCORE_EM):  %LET (variable name is DOMAIN)
SYMBOLGEN:  Macro variable DOMAINLIST resolves to 1 1 1 1 1 1 1 1 1 1 1 1
SYMBOLGEN:  Macro variable ITEM resolves to 12
MLOGIC(SCORE_EM):  %LET (variable name is CLASS)
SYMBOLGEN:  Macro variable CLASSLIST resolves to 3 3 3 3 3 3 3 3 3 3 3 3
SYMBOLGEN:  Macro variable ITEM resolves to 12
MLOGIC(SCORE_EM):  %LET (variable name is SUB)
SYMBOLGEN:  Macro variable SUBLIST resolves to 1 2 3 1 2 3 1 2 3 1 2 3
SYMBOLGEN:  Macro variable ITEM resolves to 12
MLOGIC(SCORE_EM):  %LET (variable name is NUM)
SYMBOLGEN:  Macro variable NUMLIST resolves to 1 1 1 2 2 2 3 3 3 4 4 4
SYMBOLGEN:  Macro variable ITEM resolves to 12
MLOGIC(SCORE_EM):  %LET (variable name is INVAR)
SYMBOLGEN:  Macro variable VARLIST resolves to Mode1_technique1 Mode1_technique2 Mode1_technique3
            Mode2_technique1 Mode2_technique2 Mode2_technique3 Mode3_technique1 Mode3_technique2
            Mode3_technique3 Mode4_technique1 Mode4_technique2 Mode4_technique3
SYMBOLGEN:  Macro variable ITEM resolves to 12
MLOGIC(SCORE_EM):  %LET (variable name is OUTVAR)
SYMBOLGEN:  Macro variable DOMAIN resolves to 1
SYMBOLGEN:  Macro variable CLASS resolves to 3
SYMBOLGEN:  Macro variable SUB resolves to 3
SYMBOLGEN:  Macro variable NUM resolves to 4
SYMBOLGEN:  Macro variable INVAR resolves to Mode4_technique3
SYMBOLGEN:  Macro variable OUTVAR resolves to Score_1_3_3_4
MPRINT(SCORE_EM):   if Mode4_technique3 in (.,-999) then Score_1_3_3_4=0;
SYMBOLGEN:  Macro variable OUTVAR resolves to Score_1_3_3_4
SYMBOLGEN:  Macro variable INVAR resolves to Mode4_technique3
MPRINT(SCORE_EM):   else Score_1_3_3_4=Mode4_technique3;
MLOGIC(SCORE_EM):  %DO loop index variable ITEM is now 13; loop will not iterate again.
MPRINT(SCORE_EM):   run;

NOTE: There were 1188 observations read from the data set WORK.SURVEYS.
NOTE: The data set WORK.SCORE has 1188 observations and 26 variables.
NOTE: DATA statement used (Total process time):
      real time           0.20 seconds
      cpu time            0.20 seconds
```

Note that since we want to generalize to other types of analyses, we use a macro statement to automate counting the number of variables we need to loop through. This number is then saved to the *Nwithin* macro and used in the %do loop under the *clean_em* macro.

STEP 4: CREATING A RULE DATASET

Next, we need to program one more step so each score variable is associated with the corresponding construct. It is possible for a construct to have varying numbers of items, and we need our code to be flexible when creating construct totals. Therefore, we create an additional dataset from our spec file using the *reference_rule* macro:

```
%macro reference_rule;
data rule;
set simple_spec;
      /**** Iterate through each domain/class/subclass item **
      ***** and create score variable                       **/
      %do item=1 %to &Nwithin.;
            %let domain = %scan(&domainlist., &item., " ");
            %let class  = %scan(&classlist.,  &item., " ");
            %let sub    = %scan(&sublist.,    &item., " ");
            %let num    = %scan(&numlist.,    &item., " ");
            %let outvar = Score_&domain._&class._&sub._&num.;
            /**** Create an "outvar2" variable that associates the **
            ***** score with the total it is part of               **/
            %let outvar2= Total_&domain.&class.&sub.;

            if Domain=&domain. AND Class=&class. AND
               Subclass=&sub. AND Num=&num.
                  then do;
                        Score="&outvar";
                        ClassName="&outvar2";
                  end;
      %end;
run;
%mend reference_rule;
```

We iterate through each row of the *simple_spec* and will store the same construct name for items with identical domain, class, and subclass. This way, we can accumulate totals of all the component items within the same construct total variable when we process the dataset. **Table 2** shows the resulting *rule* dataset, with the name of the variable attached as a reference.

Table 2. The *rule* dataset

Variable	Score	ClassName
Mode1_technique1	Score_1_3_1_1	Total_131
Mode1_technique2	Score_1_3_2_1	Total_132
Mode1_technique3	Score_1_3_3_1	Total_133
Mode2_technique1	Score_1_3_1_2	Total_131
Mode2_technique2	Score_1_3_2_2	Total_132
Mode2_technique3	Score_1_3_3_2	Total_133
Mode3_technique1	Score_1_3_1_3	Total_131
Mode3_technique2	Score_1_3_2_3	Total_132
Mode3_technique3	Score_1_3_3_3	Total_133
Mode4_technique1	Score_1_3_1_4	Total_131
Mode4_technique2	Score_1_3_2_4	Total_132
Mode4_technique3	Score_1_3_3_4	Total_133

STEP 5: TOTALING `EM UP

With our macros in place, we can now write code to create the construct totals. Essentially, we need SAS to iterate through each construct and sum the relevant items. We first create a list of totals from the rule dataset created in Step 3. Next, we program a macro loop to iterate through each total. Then, for each total using PROC SQL with the INTO clause, save variables associated with the construct into a separate macro. Lastly, we accumulate a sum of all items by looping through each construct item. In this last piece, we also program a construct mean. Because the mean of an average is the same as the average of all underlying items, we accumulate a mean of means by iterating through each construct item. We end up with a dataset that now contains one variable per construct that contains the construct totals, and a separate set of variables that contains the construct means. The data is now ready for descriptive and outcome analyses. The following code totals 'em scores up!

```
%macro total_em_up;
/* Store list of TOTAL variables we will create in macro &namelist. */
proc sql;
      select distinct ClassName
            into :namelist separated by " "
            from rule;
quit;

/** Loop through each TOTAL variable in macro &namelist.           **/
%do cond=1 %to %sysfunc(countw(&namelist.));

      /* Store construct total in macro &name. */
      %let name=%scan(&namelist.,&cond.," ");

      /* Lookup associated score variables based on &name. */
      proc sql;
            select Score
                  into :varlist separated by " " from rule
                  where ClassName="&name.";
      quit;

      /* Use the same data file and add the totals and mean totals. */
      data score;
      set score;
      /** Now loop through each item, and add each item to **
       ** the total and means variable                     **/
      %do i=1 %to %sysfunc(countw(&varlist.));
            &name.=sum(of &varlist.);
            M_&name.=mean(of &varlist.);
      %end;

      run;
%end;
%mend;
%total_em_up
```

CONCLUSION

With this very simple example, we can create the construct total by summing the “YES” responses of the component items. Often times, construct totals could include items that are Likert scales or other types of responses. In those cases, a construct could be constructed by standardizing items to a similar scale before combining into one total. This code can be modified to accommodate such scenarios.

In addition, we can generalize this technique to other similar tasks and projects where we have to create multiple construct totals quickly. Based on previously done psychometric analyses on survey items, perhaps certain construct totals require a mathematical combination of the items or have different numbers of items. We can add these additional rules into the macros in this paper to accommodate these situations. In the end, our aim is to increase efficiency and maintain accuracy when creating these sums. By using macros, this helps us approach the task efficiently rather than manually typing out each formula by hand.

ACKNOWLEDGMENTS

We would like to thank our mentor Seungyoung Hwang for his helpful advice and guidance in our development of this paper.

CONTACT INFORMATION

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

Vincent Chan
IMPAQ International, LLC
vchan@impaqint.com

Lorena Ortiz
IMPAQ International, LLC
lorenaortiz@impaqint.com