PhUSE 2007

Paper DM03

Preparing Real World Data in Excel Sheets for Statistical Analysis

Volker Harm, Bayer Schering Pharma AG, Berlin, Germany

ABSTRACT

This paper collects a set of techniques of importing Excel sheets and creating keys to prepare the data for statistical analysis using proc import and macro variables. The data stem from a real world problem that arose due to a possible production or transport problem. Main topics that came up were: How to name lists of variables in a set of sheets? How to preserve contents of labels in a readable form? How to import character columns with some numerical values? How to eliminate superfluous blanks and line feeds? How to rename variables in automated way?

INTRODUCTION

To assess factors that affect the quality of our products data were given to us in a manually collected Excel sheet. Goal of one of the projects was to collect and merge all available related logistic and production data. The data stem from various systems, which all have an Excel interface. As SAS® is our system for statistical analysis, my task was to import the Excel sheets and to merge the data sets to get a consistent set of data, so that a multivariate approach as Principal Component Analysis or Logistic Regression could be applied.

WHAT DOES "REAL WORLD" MEAN?

In all we had ten Excel workbooks containing 14 Excel sheets with up to 120 columns and up to 900 lines.
Some workbooks contained manually collected data. This means you have to be aware of typos, comments in numerical columns, line feeds and other special characters in cells. Especially the individual layout of header lines in Excel sheets require special attention. Sheets that were delivered from Excel interfaces of the various operational systems give further challenges: Some variables that mean the same have different names in different systems or may have different formats. "Real World" also means that there are lots of variables, where you do not exactly understand the meaning of the content. So you have to be careful to preserve the information, which is given in column headers.

THE NAMING SCHEME

The first step in preserving this information was to develop a naming scheme for the variables of the resulting merged data set. The variable names in this data set must reflect the workbook they come from, the sheet and the variable name in a way that our statistician could easily identify it while doing the analysis.
The workbooks were delivered from the operational function and had typical long names as '#Status_4 mit Produktions- und Versanddaten 20070413.xls'. So at first we translated the ten workbook names into a list tp01 to tp10 and added another digit for the number of the sheet in the workbook. So tp041 is a typical name for a data set of an imported Excel sheet. Due to the same reason we did not use column headers as variable names, but used the automatic numbering provided by proc import while importing Excel sheets resulting in variable names F1-Fn.

IMPORTING THE EXCEL SHEETS

When we started importing the data, it was clear to choose a method that is quite repeatable and makes almost no assumptions about the structure of the data. Therefore we decided to use proc import after also considering the Import Wizard or the External File Interface (EFI).

IMPORTING THE DATA

To get a consistent structure of code little macros were used.

For each Excel sheet we started importing the data part:

```
%macro ImportData (fnm =, sheet =, drange =);
    * creates data set work.tp_D that contains the data of the specified Excel;
    * sheet;
    * Parameters:
    *   fnm - filename of Excel worbook;
    *   sheet - name of Excel sheet (in quotes);
```

```
    *   drange - range of data in Excel sheet (in quotes);
    * Global macro variables:
    *   pjd - contains the name of directory of the Excel workbooks;

    proc import dbms = EXCEL2000 replace
        datafile = "&pjd.\&fnm." out = work.tp_D (label = "Data from &fnm.");
        sheet = &sheet.;
        range = &drange.;
        getnames = no;
        mixed = yes;
    run;
%mend;
```

By specifying the range in the Excel sheet appropriately only the data not the header information was imported. The header information was handled separately as the content of the header in most cases is not suited to form reasonable variable names. Instead we set the option getnames to no to get an automatic numbering of the variables (F1-Fn).

Importing columns that contain both character and numeric data is a bit tricky. If SAS decides the column to be of type numeric character entry entries are read in as missing, which seems reasonable. You have to take care of. But the other way round, if SAS decides a column to be of type character, strings, which can be interpreted as numeric, are also read in as missing.

The second case caused a real problem. Main key variables in our data are the batch numbers. There are two types of batches, filling batches and sales batches. Filling batch numbers consist of five digits (e.g. 55000). Sales batches, which are produced from filling batches normally have numbers consisting of five digits followed by a character (e.g. 55000A). But in some cases the whole filling batch becomes a sales batch and the sales batch number is the same as the filling batch number. SAS sees a numerical value, sets it to missing, and the batch is lost.

The remedy is setting the option mixed to yes. Then SAS sets the variable to be of character type, if it finds character values, and converts the numerical values to character.

IMPORTING THE HEADER

The next step is to import the column headers. We put them in a separate data set. The idea is to use them later on as labels for the variable in the resulting data set.

```
%macro ImportHeader (fnm =, sheet =, hrange =);
    * creates data set work.tp_H that contains the column headers of the;
    * specified Excel sheet;
    * Parameters:
    *   fnm - filename of Excel workbook;
    *   sheet - name of Excel sheet (in quotes);
    *   hrange - range of header in Excel sheet (in quotes);
    * Global macro variables:
    *   pjd - contains the name of directory of the Excel workbooks;

    proc import dbms = EXCEL2000 replace
        datafile = "&pjd.\&fnm."
        out = work.tp_H (label = "Column headers from &fnm.");
        sheet = &sheet.;
        range = &hrange.;
        getnames = no;
    run;
%mend;
```

The specified header range hrange should consist of only one line to get a data set with only one observation.

SETTING THE LABELS

The following macro combines the two temporary data sets and creates the raw import data set for each Excel sheet.

```
%macro SetLabels (dsn =);
    * creates data set work.&dsn.;
    * line feeds and multiple are removed from the column headers;
    * contents of tp_H is used to create a attrib statement for tp_D;
    * Parameters:
    *   dsn - name of the resulting data set;
    data _null_;
        set work.tp_H end = last;
        array vars _all_;
```

```
        attrib labels length = $20000;
        retain labels " ";
        do i = 1 to dim(vars);
            vars[i] = compbl(translate(vars[i], " ", byte(10)));
            labels = trim(left(labels))||' '||vname(vars[i])||
                ' LABEL="'||vars[i]||'"';
        end;
        if last then call symput("labels", trim(left(labels)));
    run;
    data work.&dsn.;
        set work.tp_D;
        attrib &labels.;
    run;
%mend;
```

The resulting data set now has variable names F1 to Fn, which is quite convenient for printing a compact list, and a description of the variable in the variable label, if you want to know. For convenience line feeds, which can occur in Excel cells and really destroy your output, are removed as well as multiple blanks.

PREPARING THE DATA FOR STATISTICAL ANALYSIS

So far we have a set of robust methods to get the Excel data into SAS. In the following some steps to make the data more appropriate for statistical analysis will be described.

CONVERTING CHARACTER VARIABLES TO NUMERICAL VARIABLES

The use of option mixed leads to a number of character variables, which are actually numerical variables. If you use the following little macro utilizing the input function, the conversion is quite straightforward.

```
%macro Char2Num (dsn =, var =, format = best8.);
    * converts a character variable to a numerical with same name and label;
    * Parameters: ;
    *   dsn - data set name;
    *   var - name of variable to be converted;
    *   format - format of the new variable;
    data tmp;
        set &dsn.;
        call symput ('Label', vlabel (&var.));
        NumValue = input (&var., &format.);
        drop &var.;
    run;
    data &dsn.;
        set tmp;
        label &var. = &label.;
        &var. = NumValue;
        drop NumValue;
    run;
    proc datasets nolist;
          delete tmp;
    run;
    quit;
%mend;
```

This macro can easily be extended to convert more than one variable in a data set. If you have a lot of variables to be converted, this may be more efficient.

Converting character variables to numerical variables can be done analogously by using the put function.

COMPRESSING CHARACTER VARIABLES

Later during the statistical analysis results are mainly presented by printing lists. As mentioned above, it happens often that line feeds and multiple blanks appear in Excel cells. The technique we used to compress character values when setting the labels can be applied to all character values of the data set.

And we went even a step further. If you reduce the length of the values, the length of the variables remains the same. So if it happens that there are really masses of blanks in your character variables, you can reduce the length of the variables by constructing a length statement evaluating the actual length of the character values. Here is the code we used:

```
%macro CompressCharVars (dsn =);
    * replace line feeds by blank;
    * reduce length of variables;
```

```
    * Parameters:;
    *   dsn - name of data set;
    data ds;
        set &dsn. end = last;
        array ac _character_;
        do i = 1 to dim(ac);
            ac[i] = compbl(translate(ac[i],' ',byte(10)));
            ac[i] = left(trim(ac[i]));
        end;
        if last then call symput ('NoOfCharVars', dim(ac));
    run;
    * get names of character variables;
    proc contents data = work.tp011 out = work.cont noprint;
    run;
    data work.ccont;
        set work.cont;
        where type = 2;
        n = _n_;
        keep name type n;
    run;
    proc sort data = ccont;
        by descending n;
    run;
    * build attrib statement;
    %let e = ;
    %do i = 1 %to &NoOfCharVars.;
        data work.ccont;
            set work.ccont;
            call symput ('var', name);
            if n = &i. then delete;
        run;
        * get length of variable;
        * create ATTRIB statement in e;
        data _null_;
            set ds (keep = _character_);
            array ac _all_;
            retain l1-l&NoOfCharVars.;
            array an l1-l&NoOfCharVars.;
            do i = 1 to &NoOfCharVars.;
                an[i]=max(an[i],length(ac[i]));
            end;
            attrib d length = $2000;
            do i = 1 to &NoOfCharVars.;
                d = compress( "&var.")||" length = $"||compress(l&i.)||
                    " format = $"||compress(l&i.)||". ";
            end;
            call symput ('d', d);
        run;
        %let e = &e. &d.;
    %end;
    data &dsn.;
        attrib &e.;
        set ds;
    run;
    proc datasets nolist;
        delete ds cont ccont;
    run;
    quit;
%mend;
```

RENAMING VARIABLES

Until now our naming scheme was quite handy. But if we want to merge the different data sets, you see there are a lot of duplicate variable names. To make the variable names unique they need an identifier of the data set they come

from. In this way we had a complex renaming task, which we accomplished by the following macro adding the data set name as a prefix to the variable name:

```
%macro RenameVars (lib = WORK, dsn =);
    * adds the data set name as prefix to variable names;
    * retrieves the number of variables in the data set and the variable names;
    * from dictionay tables;
    * Parameters:
    *   lib - library where the data set resides;
    *   dsn - data set name;
    proc sql noprint;
        select nvar into :num_vars
            from dictionary.tables
            where libname = upcase ("&lib") and memname = upcase ("&dsn");
        select distinct (NAME) into :var1-:var%trim(%left (&num_vars))
            from dictionary.columns
            where libname = upcase ("&lib") and memname = upcase ("&dsn");
    quit;
    run;
    proc datasets library = &lib;
        modify &dsn.;
            rename
                %do i = 1 %to &num_vars;
                    &&var&i = &dsn._&&var&i.
                %end;
            ;
    quit;
    run;
%mend;
```

In this way variable F1 in data set tp011 is renamed to tp011_F1.

IDENTIFYING THE KEYS

The final step before delivering to the statisticians is to merge all the data sets by unique keys. In finding the keys SAS cannot help you. If you have them, find short mnemonic name for them and rename the appropriate variables. Then the merge can start.

RESULTS

All the techniques we used are not new and were mainly collected from papers of various SAS user groups. The challenge was to apply them consistently to the given complex set of data.

CONCLUSION

The really difficult questions about the quality of the data, the exact representation of keys, abundant information and more could not be addressed in this paper and will be specific for any new task of preparing Excel data, but after this experience I think we have very valuable tool box to hold the course in the next project.

REFERENCES

Ravi, Prasad (2003). “Renaming All Variables in a SAS Data Set using the Information from PROC SQL´s Dictionary Tables”, Paper 118-28, SUGI 28

SAS Institute, Inc. (2006). Base SAS 9.1.3 Procedures Guide, Second Edition: The Import Procedure. cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Volker Harm
Bayer Schering Pharma AG
Müllerstraße 178
13342 Berlin
Work Phone: +49-30-468-11208
Fax: +49-30-468-91208
Email: volker.harm@bayerhealthcare.com
Web: http://www.bayerscheringpharma.de