Paper 015-2008

3 weeks of work in 5 minutes: A Designed Solution for Automatic Reporting and Distribution

Danni Bayn, Capella University
Richard Koopmann Jr., Capella University

ABSTRACT

This paper describes a program designed to automatically generate reports on End of Course evaluations and then email them to faculty. In addition to saving many hours of tedious labor by implementing an efficient 5-minute program, the delay of feedback was shortened considerably. This allows for the reports to be used more effectively. After describing the program, highlights and 'hiccups' to development are discussed.

INTRODUCTION

In every aspect of human endeavor, feedback is crucial to performance improvement. While some skills have immediate feedback built into their structure (e.g. riding a bike), most skills are reliant upon delayed feedback: feedback that is riddled with noise or absent all together. As an example, take End of Course evaluations for teachers. Evaluations are filled out by students at the very end of the course and then passed off to the school. From here, they may or may not be disseminated to the rest of the faculty, and they may come as raw data or generic reports. At Capella University, all End of Course Evaluations are online, so we are lucky enough to already have the information in electronic format. Even with that advantage, the number of hours it took to process this data combined with the length of time before feedback was delivered to faculty, was extensive. This paper will briefly describe the old process that was used, before delving deeper into a SAS® solution that reduced the quarterly workload for the reports from three weeks (or longer) to five minutes.

Old Process

After acquiring the raw data from the End of Course evaluations for over 400 courses, data for individual instructors were cut and pasted into an Excel workbook with pre-existing formulas and formatting requirements. Each file was then checked, saved, and stored in a server side folder. This boring, tedious process took one person a little over a week to complete and was very prone to human error.

Next, the administrator for each school (total of five), was responsible for going into the report folder, gathering the appropriate file/set of files for the faculty, chair, and dean, and then emailing them out by hand. The workload estimate was about two weeks from the schools that actually went through this entire process. Most schools never distributed the reports at all.

New Process

The morning after the evaluation window has closed, the SAS program is updated for the current term and then run. Within five minutes (ten minutes on a day where the server is being sluggish), all reports for all faculty have been automatically generated, placed in the server side folder, and each faculty member has their individual report waiting in their inbox.

REPORT CREATION

To generate reports from the evaluations, the program uses three main macros. The first, *Agg_Eval*, does most of the work. This macro calculates mean and standard deviations for the data at different levels of aggregation. For example, it currently aggregates and reports on data at the course section level, then for each course, then for each faculty, before finally reporting the final scores.

```
%macro Agg_Eval(data=, by=, out=);
proc sort data=&data out=_&data;
by &by;
run;

proc datasets nolist; delete &out; run; quit;

%let by_n = %sysfunc(countw(&by));
%do i = &by_n %to 1 %by -1;
%let by_&i = ;
%do j = 1 %to &i;
%let by_&i = &&by_&i %scan(&by, &j);
%end;

proc means data=_&data noprint;
by &&by_&i;
output out=_&data._&i( rename=(_freq_=N _stat_=Statistic)
drop=_type_
where=(statistic in('MEAN','STD'))
);
var Course_Well_Organized -- Instructor_Quality
    Achieved_Learning_Outcomes -- Course_Quality;
run;
proc append base=&out data=_&data._&i; run; quit;
proc datasets nolist; delete _&data._&i; run; quit;
%end;
proc datasets nolist; delete _&data; run; quit;

proc sort data=&out;
by &by statistic;
run;

data &out(drop=i);
set &out;
array by_(*) &by;
do i = 1 to dim(by_);
if by_[i] = ' ' then by_[i] = '[ROLL-UP]';
end;
run;
%mend;
```

The second macro is *ce_file*. It takes the path where the files are to be stored (defined at the top of the program) and the Instructor's name (from the dataset) to create and save the final excel report file. It creates two worksheets in each, one containing the raw data and the other containing the aggregate data from the *Agg_Eval* macro.

```
40 %macro ce_file(type=,unit=);
41 %if %upcase(&type.) ne INSTRUCTOR
42 %then libname xls excel "&thePath.\2007-11 _&unit..xls" ver=2002;
43 %else libname xls excel "&thePath.\2007-11 &unit..xls" ver=2002;
44 ;
45
46 data xls.Submissions (dblabel=yes);
47 set ce_results(drop=Course_Start_Year Course_Start_Qtr);
48 where upcase(&type.) = upcase("&unit.");
49 run;
50
51 data xls.&type (dblabel=yes);
52 set &type;
53 where upcase(&type.) = upcase("&unit.");
54 run;
55
56 libname xls clear;
57 %mend;
```

The last macro, *Process_List*, is the one that controls the flow of the other two macros. It gets called with statements like:

```
%Process_List(type=New_School);
```

or,

```
%Process_List(type=Instructor);
```

to indicate the desired level of aggregation. For example, the first call will produce reports that are aggregated on a school by school basis; in other words, it creates a report file for each school. The second call, on the other hand, creates reports aggregated by instructor and ends up producing individualized reports for each.

Process_List calls *agg_eval* (Line 59) to get the summary data and then saves it to files by calling *ce_file* (Line 71).

```
58 %macro Process_List(type=, by=Course_Start_Month &type Course Section
    Instructor);
59 %agg_eval(data=ce_results
60 , by=&by
61 , out=&type);
62 proc sql;
63 create table &type._List as
64 select distinct
65 &type.
66 from ce_results
67 ;
68 quit;
69 data &type._list;
70 set &type._list;
71 calltext=cats('%CE_File(type=&type.,unit=%str(', &type., '));');
72 call execute(calltext);
73 run;
74 %mend;
```

EMAIL DISTRIBUTION

Once all the files have been created, the following data step loops through each of the instructors, finds their reports, and then emails them out. In our case, we saved the excel files based on the instructor's name and then attached the file to the email in the same fashion. This is one of the many places where it is very important to go through and clean up your data, as any wayward character could stop the file from being saved and/or attached and relayed through your mail server.

The body of the email is specified in lines 82-90 and the headers and attachments are defined below it in lines 91-98. Any files that are not able to be sent get thrown into an error checking table.

```
75 filename outbox email 'CourseFeedback@capella.edu';
76 data instructor_list instructor_Fails; *SEND FILES;
77 set rwork.instructor_list;
78 file outbox;

79 thefile = cat('2007-11 ', trim(Instructor), '.xls');
80 attach = catt("'&thePath.\", thefile ,"'");

81 if index(Instructor_Email,'@') then do;
82 put 'Hello ' Instructor_First_Name +(-1) ','
83 // 'Attached please find your End of Course Evaluation results. Please note
that there are two tabs in your results file: '
84 / ' - The Submissions tab contains all submitted End of Course Evaluations
for the course(s). Each row in this tab contains results from one learner. The
first few columns contain section and course information, while columns O to AP
contain the responses.'
85 / ' - The Instructor tab contains a summary of these data for each course
section, including means and standard deviations by section, course, and
overall.'
86 // 'For the quickest response, please direct inquiries to the
CourseFeedback@Capella.edu mailbox.'

87 // 'Thank you,'
88 / 'Course Feedback team'
89 / 'Office of Assessment & Institutional Research'

90 // 'ATTACHMENT: ' thefile /;

91 put '!EM_TO! ' Instructor_Email
92 / '!EM_BCC! CourseFeedback@capella.edu'
93 / '!EM_FROM! CourseFeedback@Capella.edu'
94 / '!EM_SUBJECT! Course Evaluation Results for ' Instructor_First_Name
Instructor_Last_Name
95 / '!EM_ATTACH!' attach
96 / '!EM_SEND!'
97 / '!EM_NEWMSG!'
98 / '!EM_ABORT!';

99 end;
100 else do;
101 put 'WARNING: ' Instructor_Email ' does not appear to be a valid email
address.'
102 / 'NOTE: Nothing was sent to this recipient.';
103 output instructor_Fails;
104 end;
105 output instructor_list;
106 run;
107 filename outbox clear;
108 option obs=max;
```

Highlights and Hiccups

When sending out the emails, any email address can be used in the “from” field (Line 93). However, the recipient will see that it was sent by you (or your computer’s network address) “on behalf of” the email listed. People looking to modify the program for automated emails that get distributed externally should be aware of this and may want to find a dummy account from which to run the program.

Another option would be to modify the relay settings on your company’s mail server. This is something that should be checked beforehand, regardless of whether or not the email will be internal or external. When first using this program, external email addresses were automatically rejected because our SASV9.cfg[1] file was pointing to an internal mail server, rather than to our general smtp server.

[1]For us, this file was located at in C:\Program Files\SAS\SAS 9.1\nls\en. Check the –EMAILHOST option to make sure it is right.

As is usually the case, the biggest ‘hiccup’ to full implementation of the program was unclean data. Our final data set was pulled from many other data sets and, initially, a lot of data got dropped because the ‘join’s in the SQL code were not being exactly matched (e.g. ‘BUS3040’ didn’t match with ‘bus3040’) or because special characters in names were affecting the calls to excel and the mail server.

Much of the code for creating the excel files and generating the emails is dedicated to making sure the instructor’s name was coming through in a standard fashion. Despite this, dirty data from the source (e.g. an apostrophe in someone’s name) caused problems much further down the line (e.g. went through file creation wonderfully, but would not email out).

CONCLUSION

As with most SAS programs, the above program decreased workload and improved efficiency while removing human error from the process. More importantly to us, the program drastically shortened the time between when a course ended and when the instructor got feedback. (Plus, we can ensure that they are receiving the feedback at all). This sort of immediate and consistent feedback is necessary for improving the caliber of our courses and it would not have been done without SAS ☺.