

Paper 4169-2020

Automate Patient Safety Survey PDF Report Production

Lan Luong, Yao Zhang, and Michelle Simkins, BJC HealthCare.

ABSTRACT

The Agency for Healthcare Research and Quality (AHRQ) Surveys on Patient Safety Culture (SOPS™) **assess staff's views about their organizational culture for patient safety every two years.** The results of these surveys are used by BJC HealthCare leadership to determine **areas of improvement and monitor the organization's safety culture. At BJC, the survey results are summarized for 15+ provider organizations, at many human resources (HR) hierarchical levels, and distributed to leadership throughout the system. Therefore, the reports require complex formatting, customization, and data validation. During 2016, about 500 PDF reports were developed using a manual process that was error prone and time consuming. In 2018, based on lessons learned from 2016, our team created SAS Enterprise Guide® programs to automatically produce 855 PDF reports. This paper presents techniques used for the report production process including macros, ODS PDF, ODS LAYOUT, PROC REPORT, DO loop, and more. Additionally, ODS formatting techniques that have been discussed in existing publications will be summarized and referenced. The focus will be the strategies for designing program structure, creating PDF templates, and building macros. Techniques that improve efficiencies, such as the DO loop and the automated PDF output validation process, are discussed extensively. With the improved and automated report production process used in 2018, the resources spent for this project were reduced significantly compared to 2016.**

INTRODUCTION

Every two years, the Agency for Healthcare Research and Quality (AHRQ) sends out the **Surveys on Patient Safety Culture (SOPS™) to assess an organizational culture's support for patient safety.** There are separate surveys for hospital and medical office staff, and each survey includes 42-47 questions from 10-12 patient safety domains. Results **include each employee's responses to the questions as well as other pertinent information** such as their years of service, type of unit they work on, and type of position. The results of these surveys are used by leadership to determine areas that need improvement and **monitor the organization's patient safety culture over time.**

During 2016, around 10,000 BJC HealthCare employees completed the survey. The results of these surveys were distributed to leadership with various levels of customization including but not limited to survey type (medical vs. hospital survey), staff type (all employees vs. clinical employees), location (specific hospital or other service location), and leader-level (e.g. reports showing results for an individual manager, director, etc.). Furthermore, the results included a variety of metrics, such as the percent positive scores of survey composites and items, the difference of the score to the AHRQ benchmark, the difference of the score to past survey score, and the categorical percentile of score compared to AHRQ data. The metrics provided in the reports were also customized depending on the report level, survey type, facility, etc. Overall, about 500 PDF reports were developed using a manual process that was prone to error and time-consuming.

In 2018, based on lessons learned from the 2016 reporting project, our team developed an automated report production process to create 855 PDF reports for leaders across our organization using SAS Enterprise Guide® software. This paper presents techniques used to create this automated report production process including macros, *ODS PDF*, *ODS LAYOUT*, *PROC REPORT*.

MAIN TASKS IN SURVEY REPORTING PROJECT

DATA PREPARATION AND ANALYSIS

The first step in automating the report production was data management and analysis of about 10,000 individual survey responses. We converted the raw dataset, where each row represented an individual survey response, to aggregate datasets for various levels of our reporting needs. We calculated all the results needed to be presented on the PDF report such as percent positive score for 2018, differences between **hospitals' 2018 scores** from national benchmarks, and differences **between 2018 scores and previous years' scores**. These calculations and data management were done using array and PROC TRANSPOSE. Through these steps, we ensured that the data structure and variable names are consistent across various aggregation datasets to simplify the SAS PDF code.

Example of array code used in data management:

```
*Build array variable to keep track of positive response and missing;
data AllEmployeeV01;
  set Raw;
  *Build array variable to keep track of positive response and missing;
  array A (24) A1-A4 A6 A9 A11 A13 A15 A18 B1 B2 C1-C5 D1-D3 F1 F4 F8 F10;
  array AO (24) AO1-AO4 AO6 AO9 AO11 AO13 AO15 AO18 BO1 BO2 CO1-CO5 DO1-DO3 FO
    FO4 FO8 FO10;
  array AM (24) AM1-AM4 AM6 AM9 AM11 AM13 AM15 AM18 BM1 BM2 CM1-CM5 DM1-DM3 FM1
    FM4 FM8 FM10;
  do i= 1 to 24;
    if (A{i} = 4 or A{i} = 5 or A{i}=6) then AO{i} = 1;
    if A{i}=. then AO{i} = 0;
    if A{i}=. then AM{i} = 1;
  end;
run;
```

Example of transpose code used in data management (convert wide data to long data):

```
proc transpose data=AllEmpV04
  out= AllEmpV05 (rename=(col1=PercentPos _name_=Question));
  var A1-A4 A6 A9 A11 A13 A15 A18 B1 B2 C1-C5 D1-D3 E1 F1 F4 F8 F10 A5R A7R A8R
    A10R A12R A14R A16R A17R B3R B4R C6R F2R F3R F5R F6R F7R F9R F11R comp_1-
    comp_12 ;
  by HSO;
run;
```

AUTOMATING PDF REPORTS

The next step in this reporting project was creating the PDF reports of the survey results at the various levels of customization. In order to do this, our team prepared a total of 11 aggregated datasets for 11 different report levels. These report levels varied greatly in terms of formatting and content. Given many reports being requested, we decided to take a more automated approach to improve accuracy and efficiency

Our automation process combined various common SAS techniques together to produce a PDF report. The bulk of our codes were ODS Region codes combined with PROC REPORT codes to produce the desired display for each page. We also used a *DO* loop code to shorten

the code for pages with a similar display. Finally, we wrapped these codes with a macro to allow us to quickly produce hundreds of reports in one run.

AUTOMATING VALIDATION OF THE PDF REPORTS

The last step in this reporting process was ensuring the accuracy of the information in the PDF reports, specifically, ensuring the numerical results in the aggregated datasets matched what was in the report. This was one of the most time-consuming tasks during the 2016 reporting project, which involved two team members manually validating reports for several days. For the 2018 reporting project, we fully utilize SAS techniques to automate this step, which significantly reduced the validation time and decreased the chance of human error. We extracted relevant measure data from each PDF report, compiled these data into one dataset, merged this new dataset back to the original aggregate dataset that was used to produce these PDF reports, and identified if there are any differences between the base dataset and the dataset produced from the PDF report.

DETAILS ON THE METHODS AND TECHNIQUES USED IN OUR SURVEY REPORTING PROJECT

STARTING SIMPLE

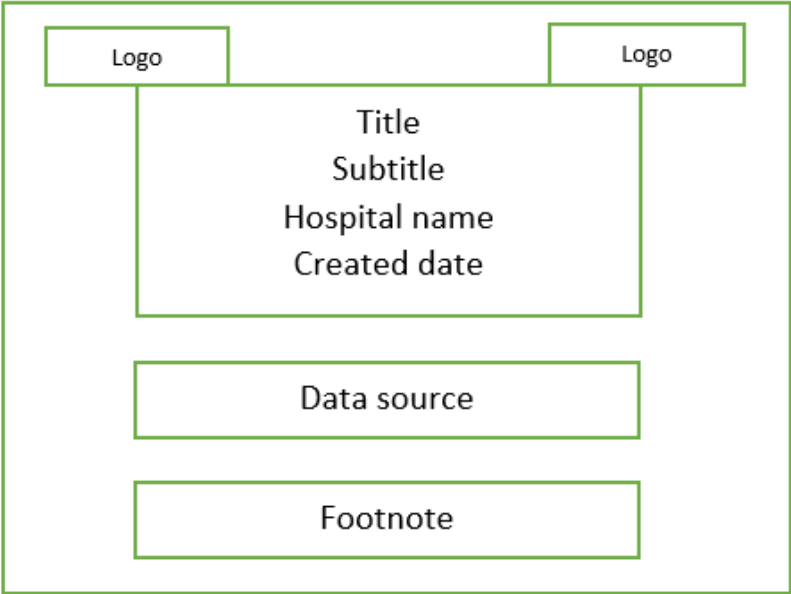
It might be daunting to start especially when building a report package from scratch using code only. But with a few preparation steps, programming can be broken down into simpler steps. The first step is to plan the PDF report template. Making a planning table and creating a few visual mock-ups like the ones shown in display 1 and 2 would be helpful. Creating a report mockup helps breakdown the unique report templates required for the PDF report package. Once the code populating the unique templates are built, it can be easily replicated, repeated, and further automated using macros.

For the AHRQ Patient Safety Culture Survey Hospital level reports, we identified 6 unique report templates. This means that at least 6 blocks of code, 1 block per report templates, are required to produce this PDF report. Once the templates are built, macro input variables can be used to control the actual contents displayed. For example, a hospital variable is controlling which logo and hospital name displayed on a report page. Therefore, thinking about the different input variables to be used would also be helpful when planning.

Template Name	Key elements (see sample below)	Variables (macro input parameters)	Data Tables/Charts
Cover	Title, subtitle, logos, footnotes	Hospital, created date	n/a
Background	Title, logos, text (2 paragraphs)	Hospital	n/a
Survey information	Title, logos, a table containing text	Hospital	Static text table
Demographics	Title, subtitle, logos, tables with headers	Hospital, report type	A few demographics tables
Composite summary report	Title, subtitle, logos, footnote	Hospital, report type	Data table

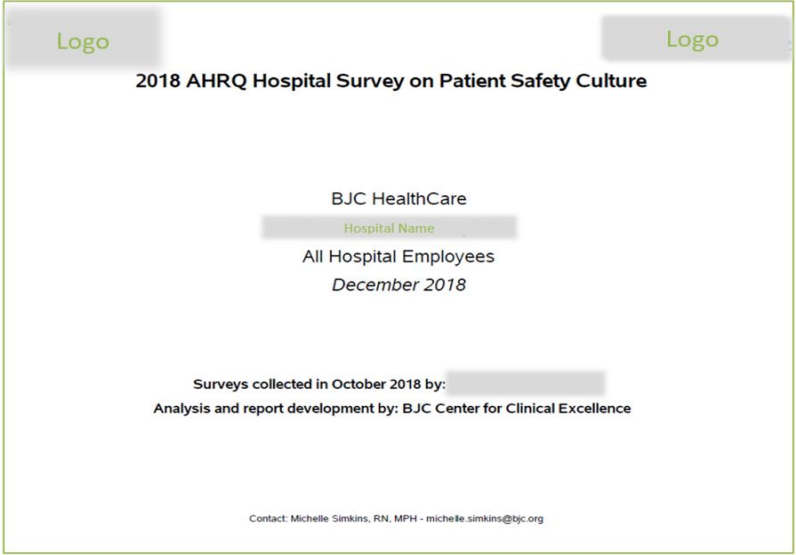
	with hospital bed size, data table		
Composite specific report	Title, subtitle, logos, footnote with hospital bed size, data table	Hospital, report type	Data table

Display 1. Sample report template planning table



Display 2. Cover page mockup

In display 2, the different elements shown in display 1 are represented in a visual mockup. By planning the position of each element, a developer can then “draw” it out in SAS Enterprise Guide® software.



Display 3. Actual cover page

The actual cover page was created using the blueprint shown in display 2.

Logo		Logo			
Title					
Question	Percentile	N	Score	Diff. from benchmark	Diff. from baseline
Footnote					

Display 4. Comprehensive page mockup

Display 4 is another visual mockup example. In this Comprehensive report page, not only the elements and position of each element are visualized, but also the columns/variables to be included in the data table are identified.

Logo						Logo	
2018 AHRQ Safety Survey Report Alton Memorial Hospital Hospital Composite Scores: All Employees							
Composite/Question Text	Percentile	N	2018 Score	Difference from Bed Size Benchmark	Difference from 2016 Score		
Communication Openness	At or below █ Percentile	█	█	█	█	6.5%	
Feedback & Communication About Error	At or below █ Percentile	█	█	█	█	3.1%	
Frequency of Events Reported	At or below █ Percentile	█	█	█	█	-2.1%	
Handoffs & Transitions	At or below █ Percentile	█	█	█	█	2.0%	
Management Support for Patient Safety	At or below █ Percentile	█	█	█	█	3.3%	
Nonpunitive Response to Error	Between █ percentile and overall benchmark	█	█	█	█	8.1%	
Organizational Learning – Continuous Improvement	At or below █ Percentile	█	█	█	█	3.4%	
Overall Perceptions of Patient Safety	At or below █ Percentile	█	█	█	█	6.2%	
Staffing	At or below █ Percentile	█	█	█	█	5.1%	
Supv/Mgr Expectations & Actions Promoting Patient Safety	At or below █ Percentile	█	█	█	█	5.1%	
Teamwork Across Units	At or below █ Percentile	█	█	█	█	1.9%	
Teamwork Within Units	At or below █ Percentile	█	█	█	█	0.7%	
Overall Patient Safety Grade	At or below █ Percentile	█	█	█	█	3.8%	

Percentiles are based on results for all facilities reporting. Bed Size Benchmark group is based on licensed bed size (n: 200-299).

5

Display 5. Actual comprehensive report

In display 5, the actual report built based on the mockup shown in display 4. In this report a few elements require special formatting – the report type is highlighted in red in subtitle; and the columns, Difference from Bed Size Benchmark and Difference from 2016 Score, have conditional formatting – positive differences are in green and negative differences are in red.



Display 6. Actual comprehensive report page shown in Display 5.

CODING FOR EACH REPORT PAGE

The next step is creating code to populate each element on a report page. We will take the comprehensive report shown in display 6 as an example to walk through the techniques and code used for each element in this report.

Element 1. Logo

Using the title statement to place logos (image file) at the top of the report page. Note that **an escape character is defined using the ods escapechar= <...> statement.** The escape character separates actual text and the formatting parameters in the title statement.

This technique is called inline formatting. A previous SAS proceeding provides a comprehensive guide for the technique (Gebhart, 2009). It can be used to format partial text in a title, footnote, or insert logos. We used it to insert 2 logos on the top of the PDF page.

Example code:

```
title height=0.5in j=left '^S={ preimage="&indir.\Image\&fac..png"}'
j=right '^S={ postimage="&indir.\Image\BJClogo.png"}';
```

Element 2. Title

Report titles are printed using the ODSTEXT procedure. The advantage of using the ODSTEXT approach over multiple TITLE statements is to have more control over the format. Here, escape characters are used again to allow inline text formatting – the report type, All Employees, is highlighted in red. The report year (year), hospital (hso), and report type (reptype) are macro input variables.

Example code:

```
proc odstext;

p "&year. AHRQ Safety Survey Report" / style=[color=black fontweight=bold
textdecoration=underline fontsize=14pt just=c];

p "&hso." / style=[color=black fontweight=bold fontsize=12pt just=c];

p "Hospital Composite Scores: ^{style[color=red fontweight=bold
fontsize=12pt]&reptype.}" / style=[fontweight=bold fontsize=12pt just=c];

run;
```

Element 3. Data table

PROC REPORT is used to populate the composite summary report table shown in Display 6. In this table, there are 6 columns, displaying data for 6 different variables. *PROC REPORT* can calculate and format the variables in the tabular report. In this example, we conditionally formatted the Difference from Bed Size Benchmark and Difference from 2016 Score columns – green font indicates improvement, and red, the opposite.

So how is this done? The columns showing comparison results are created using *COMPUTE* blocks in *PROC REPORT*. A condition logic in the *COMPUTE* statement controls the font color of the 2 survey score comparison columns based on positive/negative data values. A previous SAS conference proceeding is a great resource for this technique (Eslinger, 2015).

Another report style, alternate row formatting, also known as, banded rows, is also created using a simple loop within the *COMPUTE* block. This technique has been described in detail by SAS Support (SAS, n.d.). It is achieved by using a simple *DO* loop within one of the *COMPUTE* blocks.

Example code:

```
proc report data=composite(where=(company="&fac.)) SPLIT='~' style(header)={ rules=groups
frame=below}
out=composite_T;

*Define columns and format of the columns in the report;
column Composite_Question_Text percentile N_18 _2018_Percent_Positive_Score diff_18
diffAMC diff_18_16;
define Composite_Question_Text / 'Composite/Question Text';
define percentile / 'Percentile';
define N_18 / 'N' center;
define _2018_Percent_Positive_Score / '2018 Score' center format=percentn8.1;
define diff_18 / 'Difference from Bed Size Benchmark' center format=percentn8.1;
```



```

define diff_18_16 / 'Difference from 2016 Score' center format=percentn8.1;
define diffAMC / 'Difference from the AMC/Teaching Benchmark' center format=percentn8.1;

*Conditionally highlight the Difference from the Bed Size Benchmark column;
compute diff_18;
  if diff_18.sum lt 0 then call define(_col_, 'style', 'style=[foreground=red]');
  else call define(_col_, 'style', 'style=[foreground=green]');
endcomp;

*Conditionally highlight the Difference from the 2016 Scores column;
compute diff_18_16;
  if diff_18_16.sum lt 0 then call define(_col_, 'style', 'style=[foreground=red]');
  else call define(_col_, 'style', 'style=[foreground=green]');
endcomp;

compute diffAMC;
  if diffAMC.sum lt 0 then call define(_col_, 'style', 'style=[foreground=red]');
  else call define(_col_, 'style', 'style=[foreground=green]');
endcomp;

*Alternating row formatting using a loop in COMPUTE block;
compute Composite_Question_Text;
  tmpvar+1;
if mod(tmpvar,2) ne 0 and Composite_Question_Text ne 'Overall Patient Safety Grade' then
  call define(_row_, "style", "style={background=bwh}");
if Composite_Question_Text='Overall Patient Safety Grade' then call define(_row_, 'style',
  'style=[font_weight=bold background=#AEADD9 font_style=italic]');
endcomp;

run;

```

Element 4. Footnote with variable

A footnote displaying text and variables can also be created using *PROC REPORT*. Here, the footnote contains hospital bed size data stored in a table called bench-footnote. In this table, the bed size for each hospital is stored as an observation. The code below prints the bed size column for the specific hospital.

Example code:

```

ods region x=0in y=7.25in height=0.5in width=11in;

proc report data=bench_footnote nowd noheader SPLIT='~' style(report)={rules=none frame=void
just=left};
  column &fac.;
  define &fac. / style(column)={just=left font_weight=light font_size=9pt}
  style(header)={just=left font_weight=light};
run;

```

Arranging individual elements

Now that some building blocks have been created, the next step is to put them together on a report page. We used *ODS LAYOUT* for building the individual pages for the AHRQ Safety Culture Survey report packages.

To demonstrate, **let's look** at the layout of the Comprehensive Report example again (Display 6). The page can be divided into a few regions. The size of the PDF page is 11 in. by 8.5 in., a standard Letter size page size. Imagine that there is an x-y coordinate on the

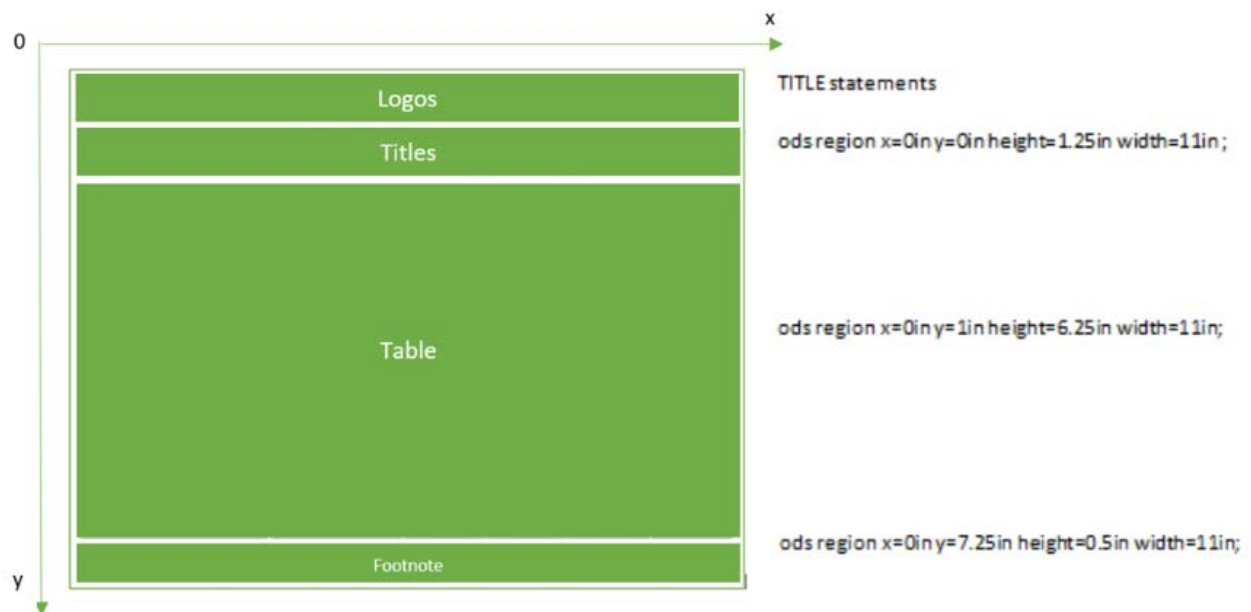
page, with the (0,0) point at the upper left corner. The region of the Logos band can be specified as:

```
ods layout start;
```

```
ods region x=0in y=0in height=1.25in width=11in;
```

```
ods layout end;
```

The position and size of the 4 regions are controlled using an *ODS REGION* statement shown below. The x and y values of a region are the coordinate of the upper left corner of each region. The width and height of a region are also specified in the *ODS REGION* statement. It takes some trials and errors to get the region position and size right so each section of content can fit and position well on the page. The code blocks mentioned in the previous chapter are wrapped by the *ODS REGION*; and *ODS REGION END*; statements.



GETTING MORE ADVANCED

After you can produce a test PDF report, it is time to design the macros structure to mass-produce these reports. The first step you need in designing a macros structure is to identify the number of macro variables you will need for the PDF report template you are producing. These macro variables are inputs that will be varying between reports, such as hospital name, hospital abbreviation, survey type, etc. In our project, the report macro was created with different macro variables to control report output.

For example:

```
% macro HSORPT(fac, hso, type, title);
```

In this example, we have four different macros variables representing: facility abbreviation, facility full name, report type, and running title in the report. In the following codes, we will show how we used macros variables to perform various tasks.

Control which hospital data is used in the report:

```
proc report data=&&data&i(where=(company="&fac.") sortedby=sort_order) SPLIT='~'  
    out=&&data&i._T;
```

Control the appearance of certain text in the report:

```

proc odstext;
  p "&year. AHRQ Safety Survey Report" / style=[color=black fontweight=bold
    textdecoration=underline fontsize=14pt just=c];
  p "&hso." / style=[color=black fontweight=bold fontsize=12pt just=c];
  p "Demographics: ^{style[color=red fontweight=bold
    fontsize=12pt]&reptype.}"/style=[fontweight=bold fontsize=12pt just=c];
run;

```

Once you have decided on all the macros variables needed to accomplish your goal, you can start building the macros structure by inserting these macros variables where you need them to control the exact outputs. This process involved using basic macros procedures that are easy to learn and perform. We will go over two of these procedures that we used in our report.

The first of which is *%IF* statement. In our report, certain facilities required slightly different formatting compared to the other. We used this *%IF* statement to tell the macros process to use a specific chunk of codes for producing the PDF report producing if the facility in the load process is one of these facilities. Due to different requirements from different facilities, it was common for us to use this statement multiple times in one template.

%IF example:

```
%if %upcase(&fac.)=HSO1 or %upcase(&fac.)=HSO2 %then %do;
```

The second common macros procedure we used in our code was *%LET* and *%DO*. We used a combination of these two statements to produce a Do Loop for report pages containing the same type of tables on multiple pages. In our case, we have 10 or 12 (depending on the report) composites scores that needed to be produced for each report. Each page of the report contains 2 of these composites' results. Our approach was using the *%LET* statement to declare these composites as macro variables and used *%DO* to loop the PDF report codes over these composites. This process reduced our coding time significantly since we do not have to produce multiple *PROC REPORT* codes for different composites.

%Let and *%Do* example:

```

%let data1      =      CommOpen      ;
%let data2      =      ErroComm      ;
%let data3      =      FreqRepo      ;
%let data4      =      HandTran      ;
%let data5      =      MgmtSupp      ;
%let data6      =      ErroResp      ;
%let data7      =      OrgLearn      ;
%let data8      =      OverPerc      ;
%let data9      =      Staffing;
%let data10     =      MgmtExpe      ;
%let data11     =      TeamAcrs      ;
%let data12     =      TeamWith      ;

```

```
%let N=12;
```

```
%do i=1 %to &N;
```

Once you have the macros structure completed, you can mend the macros and test a single load process with it to see if the automation process gives you the correct PDF report. The steps after you have checked that the macros process produced the correct PDF report would be to mass-produce all the PDF reports for all your reporting needs. In our case, this automation process allowed us to mass produce hundreds of reports of several report types at the same time quickly and efficiently. This process reduced the times needed to produce these reports and free up time for doing more meaningful analyses on the data.

VALIDATION OF THE PDF REPORTS

One of the great challenges in producing many reports is the validation step. How do we make sure the numbers produced in our PDF reports were correct? We think this is a problem every team will run into if you go through a rigorous process of report production. This process can be exceedingly time-consuming if you do the validation manually. During this round of reporting, we created a process to automate the validation of the results printed in the PDF report.

The first step in our validation strategy was to extract the data from each Proc Report table using the *OUT=* option. Previously, when we designed the macros structure, we used the *%LET* statement to declare a macros variable for each composite name and we used these macros variables in the Do Loop to reproduce a report page for each composite without having to write multiple *PROC REPORT* code. We leveraged these macros variables and used them in the *OUT=* option of the Proc Report code in the form of *&&data&i* to extract the data we needed.

Example of *OUT=* option:

```
proc report data=&&data&i(where=(company="&fac.") sortedby=sort_order) SPLIT='~'
  style(header)={ rules=groups frame=below cellheight=0.75in fontsize=9pt}
  style(report)={ rules=groups frame=below outputwidth=100% outputheight=100%}
  out=&&data&i._T;
```

The next step in our validation process was to compile all the extracted datasets into a complete dataset for validation purposes. This was done at the end of the macros structure. Here, we build a data step to combine all the extracted datasets into one dataset represent that one specific report. It is important to note here that this data step compiles data for one PDF report only.

Example of data step to compile extracted datasets:

```
data Hosp_&fac.;
  retain composite Pt_safety_comp Composite_Question_Text HSO Name PERCENTILE n_18
  pctpos_2018 Diff2016 Diff2018 DiffAMC diff_18_14;
  set composite_T CommOpen_T      ErroComm_T      FreqRepo_T      HandTran_T
  MgmtSupp_T  ErroResp_T  OrgLearn_T  OverPerc_T  Staffing_T  MgmtExpe_T
  TeamAcrs_T  TeamWith_T past12Mo_T;
  length HSO $10;
  HSO="&fac";
run;
```

A problem you may run into at this step is the naming of the dataset due to SAS naming convention. For some pdf productions, we used the specific name of the leadership to name our extracted datafile. This was done to keep track of the number of extractions we produced and to ensure we were consistent with our naming convention of the PDF report. However, some of the leadership names contained special characters that are not accepted by SAS naming convention. Thus, we used several macros function such as *%SYSPFUNC()*, *%QUOTE()*, and *%STR()* to resolve these issues.

Example of *%SYSPFUNC()*, *%QUOTE()*, and *%STR()* usage:

```
%let newname=%sysfunc(compbl(%sysfunc(tranwrd(%quote(&group),%str(,),%str( )))));
%let newname2=%sysfunc(compbl(%sysfunc(tranwrd(%quote(&newname),%str(,),%str(_)))));
```

After all the extracted datasets for a specific type of PDF report were generated, we combined these extracted datasets into one dataset to use for validating this specific type of PDF report. This was done using a data step and a *PROC DATASET* statement after the macros structure.

Example of the data step used and a *PROC DATASET* statement:

```
*Compile all the extracted dataset for one specific report group from the working library;
data Valid.HSO_1;
```

```

        set work.T_;;
run;
*Removed all the extracted dataset for one specific report group from the working library after we
finished compiling;
proc datasets lib=work;
        delete T_;;
run;

```

For our validation step, we used *PROC SQL* (left outer join) to merge the validation dataset and the original dataset. From this combined dataset, we performed calculations to make sure the numbers generated in the PDF report are the same from the original dataset. It is important to note here that this process will only validate the numerical output on the PDF reports. Formatting problems still needed to be checked manually. The manual validation of each report for formatting issues is unavoidable; however, our work to automate the validation of the numeric output on the PDF report reduced the time we needed to spend on manual validation down significantly.

Example code for *PROC SQL* for joining and data step for data validation:

```

proc sql;
        create table ValidationV02 as
        select *
        from Original as a left outer join validation as b
        on a.company =B.HSO
        and a.orgLevelName = b.name
        and a.composite_question=b.composite_questiontext;
quit;

data validationfinal;
        set work.validationv02;
        diff=pct_pos_18-pctpos18;
        if diff>0 or diff<0 then pctposval=1;
        difdif=diff_18-Diff18;
        if difdif>0or difdif<0 then diffval=1;
        Ndif=N_18-N18;
        if Ndif>0 or Ndif<0 then Ndiffval=1;
        if results_2016_comparison ne "Missing" then do;
        pct2016 = diff_18_16-diff18_16;
        if pct2016>0 or pct2016<-0 then Diff16=1;
        end;
run;

```

CONCLUSION

At the beginning of this project, our team was tasked with producing 855 PDF reports **showing the result of the AHRQ SOPS™ for different levels of leadership in our organization.** We accomplished this task promptly on time by utilizing our knowledge of SAS to automate the process of producing these different PDF reports and validating them. With the improved and automated report production process used in 2018, the resources spent on this project were reduced significantly compared to 2016. This provided our team with more time to dedicate to a deeper dive analysis on the results of the survey that we were not able to do in 2016. We think the processes we developed can be used to automate other reporting projects, within and outside of healthcare.

REFERENCES

SAS Support. "Usage Note 23647: How can I color every other row in my ODS HTML, PDF, and RTF output?". Accessed February 25, 2020. <http://support.sas.com/kb/23/647.html>.

Eslinger, Jane. 2015. "Paper SAS1642-2015 The REPORT Procedure: A Primer for the Compute Block". Available at <https://support.sas.com/resources/papers/proceedings15/SAS1642-2015.pdf>.

Gebhart, Eric. 2009. "Paper 222-2009 Inline Formatting with ODS Markup". Available at <https://support.sas.com/resources/papers/proceedings09/222-2009.pdf>.

ACKNOWLEDGMENTS

We wished to thank all the members of our team who supported us through this whole project.

RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS® For Dummies®*
- *SAS® Macro Language: Instructor-based training*

CONTACT INFORMATION

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

Lan Luong
BJC HealthCare
Lan.Luong@bjc.org

Yao Zhang
BJC HealthCare
Yao.Zhang@bjc.org

Michelle Simkins
BJC HealthCare
Michelle.Simkins@bjc.org