PharmaSUG 2021 - Paper AP-048

Using Stata for Handling CDISC Compliant Data Sets and Outputs

Tong Zhao and Jin Shi, LLX Solutions, LLC

ABSTRACT

In cases where SAS software is not capable or financially beneficial to many researchers, Stata may be a good substitute for generating CDISC compliant data and results for company's use. In this paper, we shall explain how Stata can simplify many of the routine tasks encountered in handling CDISC data sets / TFLs, and the great efficiencies that can result from using Stata.

INTRODUCTION

Stata is among the popular and powerful statistical tools to provide accurate analysis results. It is used in analyzing clinical data through all the phases. The U.S. Food and Drug Administration (FDA) accepts new drug applications performed in Stata and uses it to do the reanalysis of the submissions. Its capabilities and accuracy make it an attractive choice for many people.

This paper highlights Stata's advantages in generating CDISC compliant data sets and TFLs as well as some useful tips for accomplishing such tasks.

USING STATA HAS THE FOLLOWING ADVANTAGES

THE OPERATIONAL INTERFACE IS USER FRIENDLY

Stata is user-friendly for both beginners and experienced users. The interface as well as commands are intuitive and easy to learn. Beginners can pick it up quickly and experienced users can utilize its optimized interface to perform tasks efficiently. Its diversity in features makes it applicable for a broad range of functions.

- Stata allows users to perform analysis using either code or menus. One can run all the commands through menus and paste the code that shows up in the "Review" box into a do-file for storage, modification, or re-run.
- The help guide and manuals can be easily found by typing "help" in the command box or from the "Help" menu tab. The "Viewer" window will pop-up, showing various syntax examples of the command and the documents containing solutions to the problems.
- The do-file editor can automatically provide a list of options for the users to customize their code. Stata keeps memories of all the variable names and file names in the do-file. It also provides command options that may be useful in the completion of the current task.

COMMANDS ARE SHORTER AND EASIER TO UNDERSTAND

Stata's commands are shorter, more concise, and easier to understand compared to other softwares. Mostly, an action only takes a single line to complete. This reduces the overall complexity required to perform operations on data.

Unlike SAS, the data step and procedure (proc) step can be omitted in Stata's command. The command's syntax is simple and straightforward. Some commonly used functions can also be combined into one function.

For example, to create a by group sum variable, SAS usually takes several steps, while Stata only takes one line.

SAS code - using data step:

```
proc sort data = data; by the group; run;

data data;
set data;
retain sum;
by group;
if first.group then sum = aval;
else sum = sum + aval;
run;
```

SAS code - using procedure “SQL”:

```
proc sql;
create table datagroup as select *, sum(aval) as sum
from data
group by group;
quit;
```

Stata code:

```
bysort group: generate sum = sum(aval)
```

Stata’s unrestrained programming regulations help to curb redundancies in code.

- Variable type can be switched freely in Stata between character and numeric, without creating new variables.
- The user is not forced to create a new data set within any steps, unlike using the “SQL” procedure in SAS.
- The user always works on the data set on hand, so one does not need to specify the file name when using the most recent file.

THE STATA COMMUNITY PROVIDES STRONG SUPPORTS

Stata Corp receives questions and requests and sends out updated files every two months with new features and fixes to reported problems. In a professional Stata community, users can ask questions and get their problems solved quickly. Besides, one can import the commands or external packages written by a third party into the software, saving time used to develop a complex code from scratch.

GENERATING ADAM USING STATA AND SOME USEFUL NOTES

Unlike generating SDTM data sets, producing ADaM data sets implies more data manipulation work and involves more complex variable derivations. It may take the following steps to create an ADaM data set successfully: set up directory and folder path, import data, conduct data manipulations, and finally export the data set. The following tips can make this process much smoother.

SET UP FOLDER PATHWAY IN A SEPARATED PROGRAM

At the beginning of a program, one sets up the directory and folder pathway to access the source files and ensure the output files go to the right place. It is more convenient to set up folder pathways in a separated program and include the initial program in every ADaM program as illustrated below. This approach allows the change of pathways quickly from initial do-file for all the programs.

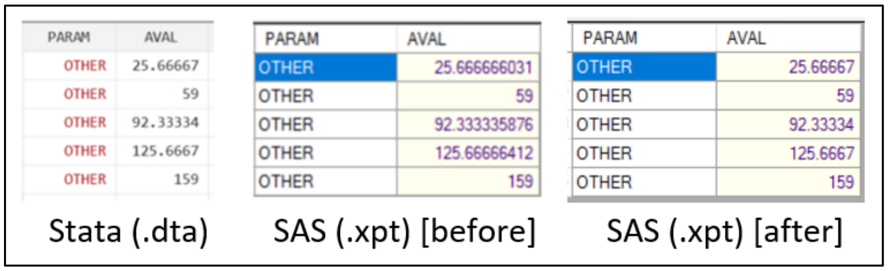
```
do initial.do
```

STORE VALUES IN STRING FORMAT

Stata has the default format for float as “%9.0g”, limiting the width of the value to 9 in the display. When the data from other software has other default width, values may change due to the difference in length. It is safer to store the original value in string format and convert it to number in the downstream analysis as needed.

For example, as shown in Figure 1, random numbers will be added to numeric variable “AVAL” if we convert .dta data set to .xpt file. To solve this problem, use the commands below by specifying the format and length while switching the variable type.

```
tostring AVAL, replace format(%9.0g) force
destring AVAL, replace force
```



PARAM	AVAL
OTHER	25.66667
OTHER	59
OTHER	92.33334
OTHER	125.6667
OTHER	159

PARAM	AVAL
OTHER	25.666666031
OTHER	59
OTHER	92.333335876
OTHER	125.66666412
OTHER	159

PARAM	AVAL
OTHER	25.66667
OTHER	59
OTHER	92.33334
OTHER	125.6667
OTHER	159

Figure 1. Store Values in String Format

USE VALUE LABEL TO CONVERT NUMBERS TO STRING CONTENTS AND VICE VERSA

Stata can convert a series of numbers to corresponding string contents simultaneously and vice versa.

The example below shows how to assign multiple PARAMCDs using parameter order number, and vice versa. The code first defines a label named “paramcd,” and it converts “ord” to get the character variable “PARAMCD” according to values defined in label “paramcd.” The “PARAMCD” variable is then switched back to order number as a new variable “ordnew.”

```
label define paramcd ///
1   "PARAMCD1" ///
2   "PARAMCD2" ///
3   "PARAMCD3" ///
4   "PARAMCD4" ///
5   "PARAMCD5"

label values ord paramcd
decode ord, generate(PARAMCD)

encode PARAMCD, generate(ordnew)
label values ordnew
```

FORMATING VARIABLES

To generate CDISC compliant data, we always want the variables to be in the standard format.

For date variables, we prefer them in ISO 8601 format. In Stata, we can use the format of “%td_CY-N-D” for compatible “YYMMDD10” date format in SAS. For character variables, we may want to set the length of them, which can be done using the command “recast” in Stata.

The code below changes numeric variable “ADT” to ISO 8601 date format, generates its character version as “ADTC”, and assigns the length of character variable “AGEU” to 5.

```
format ADT %td_CY-N-D
tostring ADT, generate(ADTC) format(%td_CY-N-D) force

recast str5 AGEU
```

When there are many variables, the format can be changed quickly using character operators for pattern matching or create macro variables that contain a list of variables. The first two lines of the code below illustrate two ways that can directly change the length of variables “AVALCAT1”, “AVALCAT2”, and “AVALCAT3” without specifying each of them. The rest of the code presents two ways to create macro variables that store a list of variable names.

```
recast str100 AVALCAT1 - AVALCAT3
recast str100 AVALCAT*

ds AVALCAT*, has(type string)
//variable names stored in local macro variable `r(varlist)'

unab AVALCAT: AVALCAT*
//variable names stored in local macro variable `AVALCAT'
```

USE MACRO VARIABLES AND LOOPS TO ASSIGN VARIABLE LABEL

Stata has two types of macro variables, including local and global. The local macros exist only when the file is running. The global macros are saved until Stata is shut down. They can be called using `name’ or $name. Stata offers two main types of loops --- “forvalues” and “foreach.” Both of them create and use local macros.

These tools and functions can be handy to repeat the code when there is a consistent rule to follow; For example, the code below shows how to assign the label of three variables using a loop. It first creates local macros “allvar”, “nvar”, and “alllabel” that store the variable names, the number of variables, and the labels. It uses a loop to take each of the content in “allvar” and “alllabel” and assigns the label to each variable.

```
unab allvar: STUDYID USUBJID SUBJID
local nvar: word count `allvar'

local alllabel ""Study Identifier""Unique Subject Identifier""Subject
Identifier for the Study""

forvalues i = 1/`nvar' {
    local var: word `i' of `allvar'
    local label: word `i' of `alllabel'
    label var `var' "`label'"
}
```

GENERATING OUTPUTS USING STATA AND SOME USEFUL NOTES

Generating outputs may include the following steps: conducting analysis using ADaM data sets, generating data set for output and validation, and exporting results to a word document. The tips mentioned for generating ADaM data sets are also useful when generating outputs. Meanwhile, some other notes can help.

CREATE AND USE TEMPORARY FILES

When data analysis becomes more complicated, we need to create some sub data sets. In Stata, we can use temporary files (so-called “tempfile”) to work with multiple data sets in the same session and avoid saving unnecessary files on the hard drive. The “tempfile” can be created when you are ready to save the data set after some commands.

In the code below, tempfile "data1" is created and saved, and after some commands, tempfile "data2" is then saved. After that, the tempfile "data1" is used again. Since the names of temporary files are local macro variables, they will disappear if the program stops running. We need to run the code from the creation of "data1" to its use to make sure Stata still recognizes the temporary file "data1" and uses it accordingly.

```
tempfile data1
save `data1', replace

[code]

tempfile data2
save `data2', replace

use `data1', clear
```

CHOOSE THE BEST COMMAND TO USE FOR ANALYSIS

Stata can perform a wide range of statistical analysis. There is usually more than one way to do so.

Table 1 shows the commands providing summary statistics and how they store the results. Each of them has its benefits and restrictions. You may choose the most suitable command to use under each of the different situations.

Command	Statistics	Sample Code	Result Stored in
tabulate	n	`tab AVALC, matcell(x)` `svmat x`	matrix x column in original data set
summarize	n, mean, sd, p50, min, max, etc.	`sum AVAL, detail`	variables r()
tabstat	n, mean, sd, p50, min, max, etc.	`tabstat AVAL, stat(n mean sd min max) save`	variables r()
collaspe	n, mean, sd, p50, min, max, etc.	`collapse (count) n=AVAL (mean) mean=AVAL`	columns in a new data set
egen	n, mean, sd, p50, min, max, etc.	`egen col1 = rowtotal(x y z)` `egen col2 = sum(AVAL)`	columns in original data set

Table 1. Commonly Used Commands for Summary Statistics

DEFINE YOUR OWN PROGRAM IN ADO-FILE

Like macro programs in SAS, programmers can write their Stata programs and use them in the future. To include a defined function in all programs, one can write it in the initial program that sets the pathways. It is also encouraged to write a system help (.sthlp file) to explain the command.

As the sample code shown below, defining a program requires the command "program" with the program name specified and an "end" at the end of the code. All the real commands can be included in between, as well as other previously defined programs. To execute the program, type the program name "hello," and "hi." will be displayed in the window.

```
program hello
version 16
display "hi."
end
```

PUTDOCX IS A POWERFUL PACKAGE TO OUTPUT WORD DOCUMENTS

Once we have the final data set for the output, we can choose a package to generate the output in a word document. There are diverse options for each function.

Package	Features
`dyndoc`	For markdown file.
`esttab`	Powerful for regression analysis.
`tabout`	Powerful for frequency analysis.
`asdoc`	More suitable to analyze within asdoc command instead of using an existing data set.
`putdocx`	Accept several output types, including the data in memory, matrices, and estimation results.

Table 2. Packages to Generate Word Document

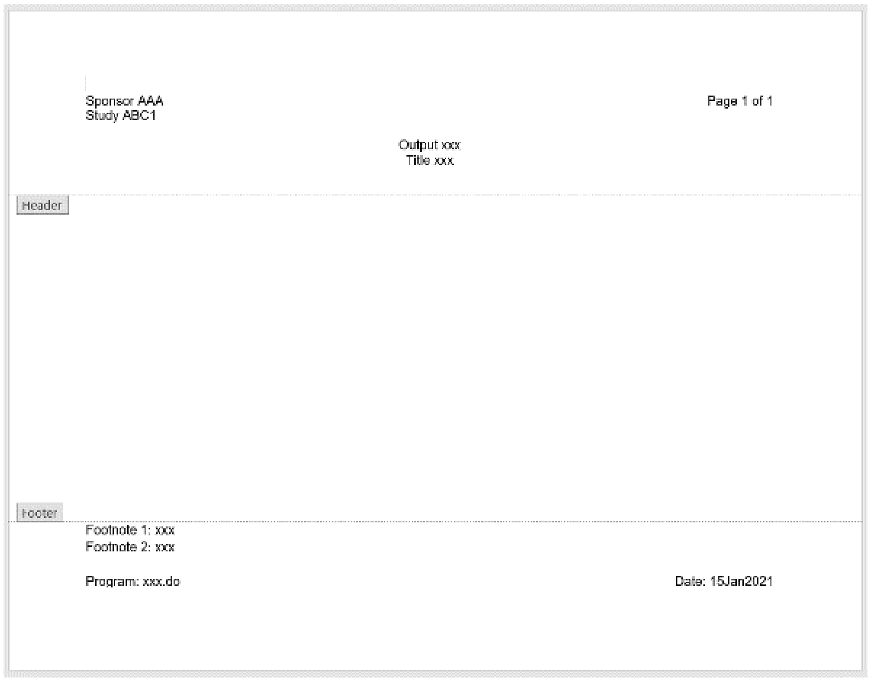
Among the packages shown in Table 2, "putdocx" is a comprehensive and functional one. It works similarly to the procedure "report" in SAS. The main commands of "putdocx" are:

- "putdocx begin": create a .docx file for export and complete the settings of the document
- "putdocx table": create a new table in the .docx file containing data in the final data set
- "putdocx save": save and close the .docx file.

CREATE A SEPERATED DO-FILE FOR THE SETTINGS

In real work, there are often dozens of outputs to generate in a single study. Therefore, it saves us much time to set up the document's general settings, header, and footer in a separate do-file. We can run it first before we complete the code for the central part of the output.

As Figure 2 shows, we can create a separated do-file that generates plain output with desired settings and basic information. We can then add the code for the main context in each of the programs.



Sponsor AAA
Study ABC1

Page 1 of 1

Output xxx
Title xxx

Footnote 1: xxx
Footnote 2: xxx

Program: xxx.do

Date: 15Jan2021

Figure 2. Output with General Settings and Basic Information

To get an output like this, we first use "putdocx begin" to create a file and set up the layout, such as page size, landscape, and margins. We also want to include header and footnotes, and they can be added using "header()" and "footer()." For header and footer, they may include tables and paragraphs just like the main content. In this case, the header consists of a 3*3 table, and the footer is made of a paragraph and a 1*2 table. We just need to use “toheader()” or “tofooter()” and match the names of them as we define them in “header()” or “footer()” in “putdocx begin” to show them in header or footer.

```
putdocx begin, pagesize(letter) landscape font(Arial, 12) margin(top, 1in)
header(head) footer(foot)

putdocx table header = (3, 3), border(all, nil) cellmargin(top, 0in)
cellmargin(bottom, 0in) toheader(head)
[code]    //contents can be added into the table using “putdocx table”

putdocx paragraph, tofooter(foot) spacing(after, 0in)
putdocx text ("Footnote 1: xxx"), linebreak

putdocx table footnote = (1, 2), border(all, nil) cellmargin(top, 0in)
cellmargin(bottom, 0.2in) tofooter(foot)
[code]    //contents can be added into the table using “putdocx table”
```

OUTPUT FINAL DATA SET AND FIGURE DIRECTLY

Using “putdocx,” the final data set for output can be added directly to the word document. To use data in memory for output, we can create a table in the main content and specify the included columns. The “putdocx table” command offers options to adjust the display of the table, columns, rows, and single cells. The options look like the settings in Microsoft Word. Take table layout as an example, there are three kinds of options in “putdocx” command, autofit contents, autofit windows, and fixed column widths, and they match exactly the autofit options provided by Microsoft Word. The “putdocx” also allows us to add columns or rows into the existing table and append other tables. Therefore, there is nearly no limit to generating a table output that follows the shells.

For a figure, we can first save it as a .png or .jpg file and then use either “putdocx table” or “putdocx paragraph” to include it in a table or output it directly to a word doc.

CONCLUSION

Although pharmaceutical companies and regulatory agencies generally welcome SAS software, Stata can supersede SAS on specific tasks depending on budget and expectations. It is always applaudable to first understand the statistical requirement and then pick the best software to fulfill them.

CONTACT INFORMATION

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

Tong Zhao
LLX Solutions, LLC
tong.zhao@llxsolutions.com

Any brand and product names are trademarks of their respective companies.